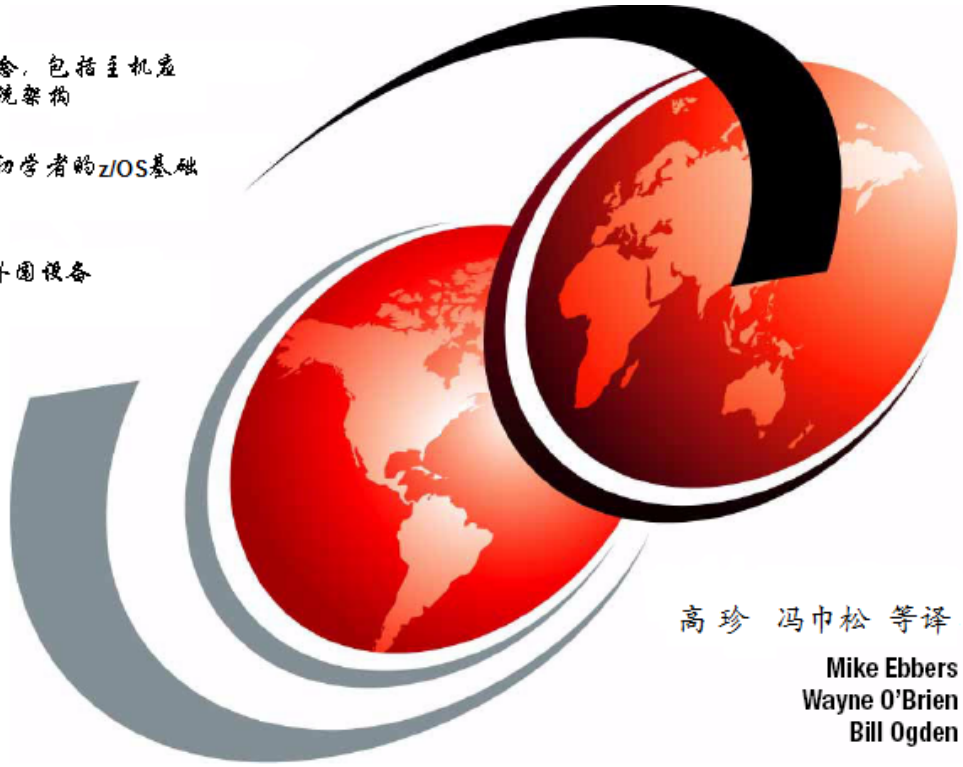
新型主机介绍：z/OS基础

主机基础概念，包括主机应用和主机系统架构

适合学生和初学者的z/OS基础知识

主机硬件和外围设备



高 珍 冯巾松 等译

Mike Ebbers
Wayne O'Brien
Bill Ogden

ibm.com/redbooks

Redbooks

国际技术支持组织

新型主机介绍：z/OS基础

2006 年 7 月

说明：

- 该译文边注中的页码是对应的原文页码
- 该译文正文中的页码参考均为原文页码

SG24-6366-00

注意：在使用本信息及其所支持的产品前，请先阅读635页的”注意事项”。

第一版(2006年7月)

目录

前言

本IBM®红皮书使学生了解信息系统技术，提供相应的背景知识以及一些用来使用主机基本工具的必备技能。这是向学生介绍主机概念的系列丛书中的第一本，以帮助他们为在大型系统计算领域从事相关职业做好准备。

为了取得最佳的教学效果，学生应该已经学习过计算机系统概念的导论性课程，如计算机组成原理和体系结构，操作系统，数据管理或者数据通信。他们也需要学习过一种或者多种编程语言，并熟悉PC。

本教材也可以用作高级课程的前修课程，或者为实习和专题研究打下基础。它并不期望覆盖主机操作的方方面面，也不是讨论主机的每个特点和选项的参考书籍。

还有其他人会从阅读本书中获益：非主机平台上经验丰富的数据处理专家，或者熟悉主机的某些方面而想对主机环境的其他方面有更全面了解的人员。

当我们浏览该课程时，我们建议老师在教学过程中穿插课本讲解，演讲，讨论，和练习指导。很多练习是累积递进的，它们的设计向学生展示了怎样设计和实施介绍过的专题。以老师为主导的讨论和练习是完整课程资料的一部分，并可以包括本书所没有覆盖的主题。

在本课程中，我们使用简化的示例并主要聚焦于基础系统功能的介绍。贯穿于本课程的实验有助于学生对主机计算模式的研究探索。

在课程结束后，您会学习到：

- 主机的基本概念，包括主机用途和体系结构
- z/OS(一种广泛使用的主机操作系统)的基础知识
- 理解主机工作负载和当前主机上使用的主要中间件应用程序
- 后续课程的基础，这些后续课程包括更高级的z/OS专用领域，如系统管理或者应用程序编程

本书是怎样组织的

本书被分为如下4个部分，如下：

- **第一部分"z/OS和主机环境的介绍"**概括介绍了主机上常用的工作负载，如批处理作业和联机交易。本部分内容帮助学生探索z/OS(一种常用的主机操作系统)的用户接口。讨论的主题包括TSO/E和ISPF，UNIX®接口，作业控制语言(JCL)，文件结构和作业输入子系统 。也着重讨论了主机的用户以及主机在当今商务世界中所扮演的不断演变的角色。

- **第二部分"z/OS的应用程序编程"**介绍了开发一个在z/OS上运行的简单程序所用到的工具和实用程序。该部分将引导学生进行应用程序设计，选择编程语言和使用一个运行时环境。

- **第三部分"z/OS联机工作负载"**考察z/OS的交互式工作负载的主要类别，例如交易处理，数据库管理和Web服务。本部分还讨论了几个常用的中间件产品，包括DB2®，CICS®和WebSphere®应用程序服务器。

- **第四部分"z/OS的系统编程"**提供多个主题帮助学生熟悉z/OS系统程序员的角色。本部分内容讨论了系统库，开启和停止系统，安全，网络通信和多系统的集群。也概览了主机的硬件系统，包括处理器和I/O设备。

在本课程中，我们使用简化的例子并主要关注基础系统功能。贯穿于本课程的实验有助于学生对主机计算模式的探索。这些练习包括在系统中输入任务，检查任务状态和察看已提交作业的输出。

每章是怎样组织的

每章遵循一种共同的组织格式：

- 学习目标
- 有关主机计算的一系列中心主题
- 本章主要内容的总结
- 本章关键术语列表
- 复习题，用于帮助学生自我检验对材料理解的程度
- 思考题，用于鼓励学生拓展研究本章内容之外的问题
- 实验，帮助学生巩固对材料的理解

关于作者

本教材由在国际技术支持组织Poughkeepsie中心工作的技术专家们编写。

Mike Ebbers 已经在IBM从事了32年的主机系统相关工作。有一段时间里，他为刚从大学里毕业的新雇员开设主机课程。Mike目前在编写IBM红皮书，这一系列产品文档可以在如下网站上找到：

http://www.ibm.com/redbooks

Wayne O'Brien是IBM Poughkeepsie的顾问软件工程师。此从1988年加盟IBM以来，他已经为多个软件产品开发了用户手册和在线帮助。Wayne获得纽约特洛伊瑞斯勒综合理工学院(Rensselaer Polytechnic Institute，RPI)的通信技术专业的理学硕士学位。

Bill Ogden是一名退休的IBM高级技术组的成员。他持有BSEE和MS(计算机科学)学位，并且从1962年就开始主机相关工作，远在z/OS操作系统还是OS/360

Release 1/2时，Bill就开始接触主机操作系统了。自从Bill 1978年参加了ITSO，他就致力于引导新的用户入门主机操作系统和相关硬件知识。

关于译者

高珍　博士，同济大学软件学院主机方向骨干教师。IBM z900主机系统管理员，主讲‘大型机操作系统(z/OS)’、‘大型机数据库系统(DB2 for z/OS)’、‘大型机系统管理’等课程。

冯巾松　同济大学软件学院主机方向骨干教师。主讲‘大型机程序设计语言(COBOL)’等课程。

吕晴　同济大学学生，将获计算机应用技术工学硕士学位。曾在IBM中国全球技术支持中心、中国软件开发中心和美国硅谷实验室实习，主要方向是主机系统管理。

顾晟　同济大学学生，现是ATA研究部引擎程序员。本科期间热爱CICS技术和COBOL语言。现在专注于 C++ / 多线程 / 计算机视觉和模式识别，依然从CICS思想中获益。

刘恒　同济大学学生，将获计算机应用技术工学硕士学位。曾在IBM中国软件开发中心的CICS小组实习一年，并担任过同济大学软件学院“大型机程序设计语言”助教工作。

张夏宁　同济大学计算机科学专业研究生，主要从事分布式计算领域的研究。曾在IBM中国软件研发中心与IBM美国加州硅谷实验室实习，主要从事主机中间件CICS和DB2方向的研究。

林敏　同济大学学生，爱好主机技术。

王丽丽　同济大学研究生，曾在IBM GBS商务智能部门实习从事数据仓库开发10个月，在IBM DB2z Level 2技术支持部门实习一年，并与2008年赴美国硅谷实验室学习工作半年。对IBM的各类操作系统及DB2等数据管理系统有着浓厚的兴趣。

王轶　同济大学学生，获硕士学位，对主机系统管理有浓厚兴趣。

特别感谢　同济大学软件学院客座教授 **黄小平** 老师的悉心指导以及对本书中主机术语的辛勤校对工作。

翻译日期　2008 年 2 月

致谢

诚挚感谢以下人员对本书所作的贡献：

Dan Andrascik 宾西法尼亚州立大学的大四学生，主攻专业为信息科学与技术。Dan精通多种计算机语言(C++，Visual Basic®，HTML，XML，SQL)，组成原理，数据库原理与设计，以及项目规划与管理。当他在IBM Poughkeepsie的ITSO组织实习时，主要致力于z系列平台的相关工作。

Rama Ayyar 澳大利亚悉尼IBM支持中心的资深IT专家。他拥有20年的MVS™操作系统经验，并在IT领域工作了30多年。他擅长的领域包括TCP/IP，安全，存储管理，配置管理问题定位。Rama持有堪萨斯州印第安技术学院的计算机科学硕士学位。

Emil T. Cipolla 是一名在美国有超过40年经验的信息系统顾问。他持有康奈尔大学的机械工程和商务管理的硕士学位。现在Emil是一个大学的兼任讲师。

Mark Daubman 是圣-波拿巴大学的大四学生。主修商务信息系统，辅修计算机科学。 在其IBM实习期间，Mark接触过本书中所讨论的许多z/OS接口。毕业之后，Mark打算找一份主机相关职业。

Myriam Duhamel 是一个比利时的IT专家。她有20年的应用程序开发经验并已在IBM工作过12年。她精通z/OS上的各种开发(如COBOL，PL/I，CICS，DB2和WebSphere MQ)。Myriam目前教授DB2和WebSphere MQ的课程。

Per Fremstad 是一位来自IBM挪威的IBM系统和技术组的IBM认证的IT专家。他从1982年开始为IBM工作并对主机和z/OS有丰富的经验。他精通z/OS上的Web，WebSphere和z/OS上的Web使能技术。他经常教授z/OS，zSeries和WebSphere for z/OS课程。Per持有挪威奥斯陆大学的理学士学位。

Luis Martinez Fuentes 是一位来自IBM西班牙的系统与技术组的IBM认证的顾问IT专家。他有20年的IBM主机经验，主要集中在CICS和DB2领域。他现在在做主机新产品的销售技术支持。Luis是伊比利亚半岛技术专家组的成员，该组织附属于IBM技术研究院。Luis在马德里的两所大学教授主机课程。

Miriam Gelinski 是巴西Maffei顾问组的成员，她在其中负责客户计划支持和主机软件安装。她有5年的主机经验。她持有圣保罗São Marcos大学的信息系统学士学位。她通晓z/OS操作系统及其子系统，以及TSO和ISPF。

Michael Grossmann 是德国的一名IT教育专家，并有9年的z/OS系统程序员和相关执教经验。他擅长教授z/OS的入门课程，精通z/OS操作，自动化，主机硬件和并行系统综合体技术。

Olegario Hernandez 曾是智利的一名IBM顾问系统工程师。他有超过35年的主机系统应用程序设计和开发的经验。他编写过CICS应用接口，系统管理和网格运算。Olegario持有智利大学的化学工程学位。

Roberto Yuiti Hiratzuka 是一名巴西的MVS系统程序员。他有15年的主机系统程序员的经验。Roberto持有”Faculdade de Tecnologia Sao Paulo”(FATEC-SP)的信息系统学位。

John Kettner 是一个zSeries高级架构组的软件架构顾问。他拥有30年的主机经验并持有L.I.U的计算机科学学士学位。他通晓zSeries的内部构造，WebSphere产品整合和容量规划。John编写了数本红皮书，并对多个IBM教育项目作出过贡献。

Georg Müller 是德国莱比锡大学的学生。他有三年的z/OS和主机硬件的经验。他准备于明年完成他的计算机科学硕士学位。对于本教材，Georg撰写了关于WebSphere MQ和HTTP服务器的章节，编写了示例程序，并帮忙确认了学习模块的最终序列。

Rod Neufeld 是加拿大的一名资深的技术服务专家。他有25年的MVS和z/OS的系统编程经验。他擅长的领域包括z/OS系统软件和支持，并行系统综合体，业务连续和恢复。Rod持有曼尼托巴大学的荣誉理学学士学位。

Paul Newton 是一名来自美国德克萨斯州达拉斯IBM开发者事务组技术支持中心的资深软件工程师。他有25年的IBM主机操作系统，子系统和数据网络经验。Paul持有亚利桑那大学的商务管理学位。

Bill Seubert 是一名来自美国的zSeries软件架构师。他对主机和分布式计算有超过20年的经验。他持有密苏里大学的计算机科学的学士学位。他精通z/OS，WebSphere整合软件和软件架构。Bill经常向IBM的客户们阐述整合架构和企业建模技术。

Henrik Thorsen 是一名来自IBM丹麦资深IT顾问。他拥有25年的主机经验，持有哥本哈根大学的工程硕士学位以及哥本哈根商学院的经济学士学位。他专长于z/OS，并行系统综合体，高可用性，性能和和容量规划。Henrik编写了数本IBM红皮书和其他文档，也对IBM及zSeries技术社区的不同教育项目作出过贡献。

Andy R. Wilkinson 是来自英国的一名IT专家。他有25年的预订系统经验和z/OS系统编程的经验，并已经在IBM工作了6年。他精通硬件配置和SMP/E。Andy持有英国设菲尔德大学材料科学与技术的学位与英国开放大学的计算机学位。

最后，特别感谢纽约州Poughkeepsie ITSO中心的编辑们：

- **Terry Barthel**
- **Ella Buslovich (图片)**
- **Alfred Schwab**

欢迎提出您的意见

您的意见对我们非常重要！

我们希望红皮书尽可能地有用。您可以采用以下方式对这本或其他红皮书提出您的意见：

- 使用在线”contact us”评论表，参见以下网址：

  ibm.com/readbooks

- 用e-mail来发表意见：

  redbook@us.ibm.com

- 寄来您的意见：
  IBM Corporation, International Technical Support Organization
  Dept. HYJ Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Part 1

第 1 部分 z/OS 和主机环境的介绍

欢迎来到主机计算世界！本书的开始，我们对主机及其在当今IT组织中的地位进行了概要的介绍。我们探索了为什么全世界的公有企业和私有企业依赖主机作为大规模计算的基础。我们讨论了常和主机关联在一起的工作负载的类型，例如批处理作业和在线或交互式交易系统，并介绍了一种广泛使用的主机操作系统——z/OS处理这些工作的独特方式。

在该部分，我们尤为关注使用主机的人们以及新型主机在当今商务世界中扮演的角色。

1

2

1

第 1 章 新型主机介绍

目标：作为一名主机计算领域的技术专家，您需要知道主机是如何支持您公司的IT基础设施和实现商务目标的，同时您也需要了解您公司主机支持团队各位成员的工作职责。

学完本章，您将具备以下技能：

- 列出当前主机挑战关于‘集中式计算对分布式计算’传统思维的几种方式
- 解释企业如何利用主机的处理能力，主机的典型用途，以及主机计算如何区别于其他类型的计算
- 列出最适合主机的主要工作负载类型
- 说出和主机计算相关的5种工作或职责
- 指出4种主机操作系统

3

1.1 新型主机

今天，主机在绝大多数世界最大型企业中的日常运作中扮演了核心角色，这些企业包括许多财富1000强的公司。尽管其他类型的计算被大量应用在不同级别的商务活动中，主机仍然在当今的电子商务环境中占据着令人羡慕的地位。在银行，金融，医疗，保险，公用事业，政府机关，和大量的其他公有及私有企业中，主机继续构成现代商务的基础。

电子商务
在诸如因特网的电子媒介上进行的交易。

主机长期的成功在IT领域是没有先例的。周期性的剧变动摇着世界经济，信息时代中持续地、常常是剧烈地改变使许多曾经璀璨夺目的发明在科技无情的进步中成为殉葬品。新科技不断跃入公众眼帘，但是其中的很多都马上被更加先进的科技所打败，成为废弃。然而今天，和20世纪60年代后的每一个10年一样，主机和主机计算始终掌控着大规模商务计算领域。

为什么主机计算如此强烈地吸引着众多的世界各地的公司？本章我们来看一看为什么主机依然是多数大规模商务运算的选择？

1.2 S/360：主机历史上的转折点

主机是什么时候诞生的？主机的起源要追溯到20世纪50年代，可能还会更早。那时候，主机不仅是最大的计算机，也是唯一的计算机，很少有企业有能力购买它。

System/360
第一台通用计算机，于1964年引入。

从20世纪50年代起，主机历经了几代的发展。比如第一代系统，1954年发布的IBM 705和1959年发布的IBM 1401，虽然和后来推出的功能强大的机器相差甚远，但是他们明显具备了主机的特征。这些主机被作为商务机器出售并且和现在一样在公司的数据处理中心充当着中心数据库[1]。

在20世纪60年代，主机制造商开始标准化他们提供给客户的硬件和软件，与此同时，计算历史的进程发生巨大的改变。1964年诞生的IBM System/360™(或者S/360™)标志着第三代的开始：第一台通用计算机。早期诸如1401之类的系统要么专用于商业计算要么专用于科学计算。只要客户，软件公司或咨询师提供相应程序，革命性的S/360系统就可以完成这两种类型的计算。事实上，S/360这个名字意味着架构范围广泛：360度地适用各方面的用户，具有全方位的特点。

S/360也是第一台使用微码来实现很多机器指令的计算机，而不是将所有机器指令都硬写入电路中。微码(有时称为固件)包含存储的微指令，对用户并不可用，但它在硬件和软件之间提供了一个功能层。使用微码的优势在于灵活，任何更正或新功能都可以通过只改变现存的微码来实现，无需更换计算机。

客户通过使用标准化的主机来运行工作负载，可以编写无需特定硬件或软件的商业应用程序。更进一步，客户在不必担心现有程序兼容性的前提下，可以随意升级到更新更强大的处理器。第一波客户商业应用程序大多用汇编，COBOL，FORTRAN或PL/1完成，这些古老的程序中很多今天仍在使用。

20世纪60年代以后，主机已经逐渐拥有巨大的处理能力。新型主机能力无敌，它能够服务数以万计的终端用户，管理千兆兆数据，允许重新配置硬件和软件资源以适应工作量的变化，所有的这些都可以通过单点控制做到。

1.3 主机演化而来的架构(演变着的主机架构)

架构
描述系统的组织结构。

‘架构’是用一套定义的术语和规则，用于指导构建产品。在计算机科学中，架构描述一个系统的组织结构。一个架构可以被递归分解为：通过接口交互的很多部分，关联这些部分的关系和组合这些部分的约束条件。部分之间通过接口交互，这些部分包括类，组件和子系统。

从20世纪60年代出现的第一代大型机器开始，这些大型机器被称为‘大铁块’(相比较小的部门系统)，每一代新的主机在架构上都在以下一个或多个方面有所改进：[1]

- 更多且更快的处理器
- 更多物理内存(也称为实体记忆体)和更大的内存寻址能力
- 动态升级硬件和软件的能力
- 加强的硬件错误自动检查和自动恢复功能
- 为输入输出配备更强设备，在I/O设备和处理器之间使用更多更快的通道
- 更完善的I/O附加装置，比如拥有强大的内部处理能力的LAN适配器。
- 更强的功能可以将一个机器的资源划分为多个逻辑相互独立、分离的系统，每个系统上运行自己的操作系统
- 高级集群技术，比如并行系统综合体，和在不同系统中共享数据的能力

5

尽管计算平台在持续不断的变化，主机仍旧是最稳定，安全和兼容的计算平台。最新的主机型号可以处理最先进和最苛刻的客户工作，也可以继续运行20世纪70年代甚至更早时候写的应用程序。

“我预计在1996年3月15日，最后一台主机将停用。”
Stewart Alsop，
信息世界，
1991年3月

一项科技怎么可能变化如此之大，却又能保持稳定呢？原因是不断演化以迎接新挑战。在20世纪90年代早期，客户端/服务器计算模型，该模型的分布式节点上采用相对不太强大的计算机，开始挑战主机的统治地位。业界权威人士就此预言主机将会迅速灭亡，他闷还称主机为‘恐龙’。作为回应，面对时代的改变和用户需求的增长时，主机设计人员一如既往：设计可以满足需求的新型主机。为了表示对”反恐论者”的支持和尊敬，IBM作为主机的领先制造商，为那时最新的机器起代号为T-Rex。

主机拥有更多的扩展功能，增加了数据处理能力层，比如网络服务，自治，灾难恢复和网格计算，这些都让主机乘上了IT产业下一波增长潮。主机生产厂商，例如IBM，又一次在报告主机的年销售增长达到了两位数百分比。

6

变革还在继续，主机在IT机构中保持着它传统、中心的角色，现今主机也充当着最大的分布式网络中主要网络集线器的角色。事实上，因特网本身主要也是基于

[1] 自从 1964 年 IBM 引入 S/360，大概每十年 IBM 就大规模地扩展平台：1970 年的 System/370™，1983 年的 System/370 扩展架构(370-XA)，1990 年的企业系统架构/390(Enterprise Systems Architecture/390®，ESA/390)和 2000 年的 z/Architecture。更多关于早期的主机硬件系统的信息，参考 565 页的附录 A：“IBM 主机历史简要回顾”。

数不胜数的互相关联的主机建立起来的，这些主机被用作主要的网络集线器和路由器。

随着主机的形象不断地演化，您可能会问：主机是一个独立的的计算环境，还是一个分布式计算中的一部分呢？答案是两者都是。因为主机自身很强大，在一个安全的地点足够处理最大量和最多元化的工作；同时当主机作为一个主要服务器应用于公司的分布式服务器场时，它也体现出有效性。从效果看来，主机在客户端/服务器的计算模式中是一个成熟的服务器。

1.4 主机在我们中间

尽管主机在商业世界里占有优势地位，但它对公众，学术界，甚至很多有经验的IT专家很大程度上是不可见的。相反，其他形式的计算吸引了更多的注意力，至少在可见度和公众认知度上是这样的。这并不让人惊讶。毕竟，我们之中有谁需要直接访问主机呢？就算我们需要直接访问，我们去哪里找一台主机呢？事实是，无论我们意识到主机的存在是否，我们实实在在都是主机用户。

我们大多数拥有PC知识和足够资金的人可以买一台笔记本电脑然后很快使用它，在上面运行程序，浏览网站，可能还会写论文交给大学教授来评分。只要稍加努力和借助科技，我们可以更深入地钻研典型的基于INTEL的工作站的各种设备，也可以通过直接的访问和动手实验来学习它的性能——无论是否借助网络上或书本上的大量容易获得的相关信息资源。

但是主机却从公众眼前藏了起来。它们独立工作，基本完全可靠，对PC病毒(比如通过EMAIL传播的病毒和木马病毒)高度防御。主机表现稳定，几乎无宕机时间，是评判所有其他电脑的典范。但同时，缺乏关注也让主机淡入后台。

而且，在典型的客户机器中，主机和很多其他的硬件设备共享空间：简单罗列几个如外部存储设备，硬件网络路由器，通道控制器和自动化磁带库‘机器人’。相对于以上很多设备，新型主机在体积上并不比他们大，它和外围设备放在一起一点也不显眼。

那我们要如何才能知道主机在现实世界中的性能如何呢？我们怎样才能学习如何和主机交互，了解它的性能，理解它在商业世界中的重要性呢？大公司渴望雇佣新型主机的专业人才，但是没有那么简单，之前的经验也会有用。

假如主机能在21世纪的IT机构内繁荣发展，那么在看到一台主机时我们能了解它么？我们需要的是一名有经验的导游指引我们去‘恐龙探险’，这正是这本书的由来。

1.5 什么是主机？

首先，我们来解决术语问题。当前，计算机制造商并不总是使用术语”主机”来指代主机。取而代之的是，大部分计算机制造商把商用计算机——无论大小——都称作服务器，主机当前仅仅是作为最为强大的服务器来使用。举例而言，IBM把它

最新推出的主机称为IBM System z9™服务器。在本文中，我们使用术语主机来指代那些能够支持成千上万的应用程序和输入/输出设备以并发地为成千上万的用户提供服务的计算机。

服务器是可扩充的。一个公司可能有大量的服务器集合，包括交易服务器，数据库服务器，电子邮件服务器和网络服务器。超大规模的服务器集合有时被称为服务器场(事实上，一些数据中心占用的空间是用英亩来衡量的)。从仅仅几台台式PC机组成的集群到功能最为强大的新型主机均可以提供服务器的功能。

服务器场
服务器的超大规模集合。

在企业的数据处理中心，主机是中心数据储存器，也是网络集线器；它通过诸如工作站和终端等能力稍逊的设备连接到用户。主机的存在通常意味着集中式计算形式，与之相对的是分布式计算形式。将数据集中在一台主机存储器中避免了用户面对管理商业数据的多个备份同时更新的烦恼，提高了数据是最新的可能性。

然而，集中式计算和分布式计算的区别不断模糊，这是因为小型计算机不断增强处理能力，而主机也比以往任何时候都更灵活多变，功能多样。市场压力使得当今的商业公司不断重新评估它们的IT策略以寻求更好的支持多变市场的途径。结果，主机现在按照不同配置频繁地和小型服务器组成的网络一起使用。在不干扰应用程序运行的同时，动态重新配置主机软硬件资源(例如处理器，内存和连接装置)的能力进一步展现了现代主机灵活，扩展性强的特性。

当主机的硬件变得难于归类时，同样，运行在主机上的操作系统也难以归类。事实上，多年以来，这两个术语定义了二者的关系：主机便是所有运行IBM操作系统[2]的硬件系统。这一定义在近年来已经变的不清晰了，因为这些操作系统现在也可以运行在小型系统中了。

平台
计算机架构(硬件和软件)。

计算机制造商和IT专业人士常常使用术语"平台"来指代那些根据某种特殊体系结构关联在一起的硬件和软件。例如，一台主机和它的操作系统(以及它们的前代系统[3])被称作一个平台；精简指令集计算机(RISC)系统上的UNIX也被认为是一种平台，它一定程度上独立于具体的RISC机器；个人电脑根据使用了何种操作系统可以被归为几种不同的平台。

主机
大型计算机系统，用于运行数据库，交易服务器和需要高安全性和可用性的应用程序。

因此，现在让我们回到我们的问题："什么是主机？"。如今，主机这个术语能够最恰当的描述一种别具风格的操作，应用程序和操作系统设施。我们从工作定义开始：主机是商业中用于储存商业数据库，交易服务和应用程序的机器，相比于规模较小的机器上常见的这些服务，他们要求更高的安全性和可用性。

早期的主机系统是存放在巨大的的房间大小的金属箱子或框架(Frame)中，这大概就是主机(Mainframe)这一术语的由来。早期的主机需要大量的电力供应和空调系统，装主机的房间主要被I/O设备所占用。同样，一个典型客户站通常装有几台主机，大部

[2] 传统上，该名字也被应用于其他厂商生产的大型计算系统。

[3] IBM System/390®(S/390®)指的是一特定系列的主机，出现在 IBM zSeries 机器之前。然而，许多 S/390 系统现今仍在使用中。因此，记住虽然我们本书中讨论 zSeries 系统，但所有讨论内容都可用于 S/390 机器。唯一的特例是 64 位寻址只存在于 zSeries。

分的I/O设备和所有的主机都相连。在主机最为庞大的阶段，按照物理大小来算，通常主机占据的空间为2000到10000平方英尺大小(200到1000平方米)。某些主机装置甚至比上述还要巨大。

从1990年左右开始，主机处理器和它的大部分I/O设备都在功能和容量持续增长的同时拥有了更小的物理体积。如今的主机相比早期的主机系统已经小了很多——大约跟一个大冰箱一般大。

在某些情况下，在个人计算机上运行主机操作系统来仿效主机已经成为了可能。使用这些模拟机可以开发和测试商业应用程序，然后再将它们移植到主机生产系统上。

很明显，主机这个术语含义已经不仅仅是描述一个系统的物理特性，而是通常适用于下列特征的一些组合：

- 兼容主机操作系统，应用程序和数据。
- 资源的集中式管理。
- 硬件和操作系统可以和其他系统互相共享访问磁盘驱动器，并采用自动锁和保护机制来避免破坏性的同时使用磁盘数据。
- 一种别具风格的操作，通常涉及专业操作人员使用详细的操作流程手册和高度组织化的流程来进行备份，恢复，培训和在备选地点的灾难恢复操作。
- 在例行工作中需要成百上千次并行I/O操作的硬件和操作系统。
- 允许客户将操作系统的若干副本当作一个系统来运行业务的集群技术。这一被称作并行系统综合体的架构，在概念上类似于UNIX簇，但却能够在应用程序持续运行的同时，根据需要来对系统进行添加或删除。这一功能使得主机客户可以依据商业行为的变化而引入新的应用程序或者终止已经存在的应用程序的使用。
- 额外的数据和资源共享能力。例如，在一个并行系统综合体中，用户可以跨多个系统平台并发访问同一个数据库，其数据访问控制可达到记录层。

随着诸如中央处理器(CPU)和外部存储媒体等硬件资源性能和成本的提高，以及CPU附属设备数量和种类的增加，操作系统软件能够从硬件的升级中获益更多。同时，软件功能上持续不断的进步也驱动着每一代新硬件系统的发展。

10

1.6 谁使用主机?

那么，谁使用主机呢？几乎所有人都在某一点或者其他方面使用着主机。如果您曾使用过自动提款机(ATM)与您的银行账户交互，您就使用过主机。

今天，主机在绝大多数世界最大型企业中的日常运作中扮演了核心角色，这些企业包括许多财富1000强的公司。尽管其他类型的计算被大量应用在不同级别的商务活动中，主机仍然在当今的电子商务环境里占据着令人羡慕的地位。在银行，金融，医疗，保险，公用事业，政府机关，和大量的其他公有及私有企业中，主机继续构成现代商务的基础。

直到20世纪90年代中期，主机为大型业务的数据处理操作提供了唯一的可行方

案。这些需求在那时候是(现在也常常是)基于庞大而复杂的批处理任务，例如工资和总账处理。

自从1964年System/360诞生以来，随着扎实而稳定的技术进步，主机与生俱来的可靠性和稳定性为主机的流行和经久不衰贡献良多。没有其他计算机体系能够在维持前代兼容性的基础上拥有这么多持续而渐进的提升。

因为这些设计中的长处，主机常常被IT机构用于运行最为重要、关键的应用程序。其中典型的有客户订单处理，金融交易，生产和库存控制，工资发放以及一些处理其他类型工作的程序。人们对主机用户界面的一个共同印象是80×24字符的"绿色屏幕"的终端，这得名于多年前主机使用的老式阴极射线管发出的是绿光。实际上，如今的主机界面已经和个人电脑或者UNIX系统毫无二致。当人们通过网络浏览器访问商务应用程序时，主机常常在"后台"执行最为关键的任务。

如今许多最为繁忙的网络站点将他们的生产数据库储存在主机上。由于主机的设计允许大量用户和应用程序快速并发查询相同数据而不会互相干涉，主机对于网络交易无疑是很理想的选择。主机的安全性，可扩展性和可靠性对于有效而安全地实施现代信息处理业务非常重要。

企业使用主机运行一些极为依赖可扩展性和可靠性的应用程序。例如，银行机构用主机管理客户账户存放的数据库，交易可以在全世界范围成千上万ATM站点中任何一台上提交给该数据库。

如今的企业依靠主机来实现：

- 执行大规模的交易处理(每秒钟数千次交易[4])
- 支持成千上万的用户和应用程序并行地访问大量资源
- 在数据库中管理千兆字节的信息
- 处理大带宽的通信

信息高速公路的大道常常是通往主机的。

1.7 决定主机如此有用的要素

使用主机的理由很多，但是大部分通常都可以归结为以下一类或几类原因：

- 可靠性，可用性和可服务性
- 安全性
- 可扩展性
- 持续兼容性
- 不断进化的体系结构

让我们深入的看看这些原因

[4]IBM 最新的主机为 IBM System z9 109(也叫 z9-109)，它一天可以处理的十亿笔交易，数目之大令人惊愕。

1.7.1 可靠性, 可用性和可服务性

在数据处理中，计算机系统的可靠性，可用性和可服务性(或者说"RAS")一直是非常重要的因素。当我们说某一计算机系统"展示了RAS特性"的时候,我们的意思是指它的设计高度优先保证在任何情况下系统都可用。在理性的情况下，RAS是包括应用程序在内的计算机系统所有方面最为核心的设计特征。

RAS作为描述许多被主机用户重视的软硬件特征的集合术语，已经被大众接受。这些术语定义如下：

可靠性 系统的硬件组成部分拥有强大的自我检查和自我恢复的能力。对软件进广泛的测试并对检测出的问题进行迅速更正，这些保证了系统的软件可靠性。

12

可用性 系统能够恢复失效组件，而不影响系统中正在运行的其他部分。它包括硬件恢复(自动使用备份替换失效组件)和软件恢复(操作系统提供的故障恢复层)。

可用性
从组件失败中恢复的同时，而不影响运行系统的其他部分的能力。

可服务性 系统能够确定故障发生的原因，它允许在尽可能小地影响业务系统的前提下对软硬件进行替换。这一术语也暗含了软件或硬件都具有明确的替换单元。

当计算机系统的应用程序是可用的，这个计算机系统就是可用的。可用的系统也是可靠的系统；这意味着，它很少因为需要升级或者修理而停工。同样，当系统发生故障的时候，它必须是可服务的；这意味着，它在相对短的一段时间内能比较容易地修好。

故障间的平均时间间隔(MTBF)标识了计算机系统的可用性。新型的主机和它的相关软件发展十分迅速，现在客户数月乃至数年才会遇到一次宕机。此外，当系统因为某一计划外的故障或者预定的升级而不可用时，不可用时间通常是非常短的。在当今24小时都在运作的全球经济体系下，负责处理公司关键任务的应用程序的高度可用性是至关重要的。和主机硬件一样，主机操作系统具备诸如存储保护和可控维护过程的特征，同样展现了RAS特性。

除了RAS，最新的主机系统提供了高可用性和高容错能力。在系统重要通道上放置冗余的硬件组件、进一步加强的存储保护、可控的维护过程以及为无限可用性而设计的系统软件，在系统发生组件故障时，均有助于为商务应用程序提供一个一致的、高可靠性的运行环境。这种措施使得系统设计者能够将单点故障对计算机系统整体RAS进行破坏的风险降到最低。

1.7.2 安全性

一个公司最为重要的资源是它的数据：客户名单，账户数据，雇员信息等等。这些重要的数据需要安全的管理和控制，同时，只有那些拥有权限的用户才能访问它们。主机允许多用户同时共享公司数据，并提供对这些数据的保护。

13

在IT环境中，数据安全被定义为保护数据免受未经授权的无意或故意地访问，移动，修改，或破坏操作。为了保护数据和维护必需达到某些安全目标的系统资源，客户通常为主机操作系统添加一个先进的安全管理工具。用户的安全管理员常常要负责使用有效技术将公司的安全策略转化成可行的计划。

一个安全的计算机系统会阻止使用者访问或改变系统的任何对象，包括用户数据，除非通过系统提供的加载了授权规则的界面进行访问。新一代主机系统为访问重要数据的大量的、各种各样的应用程序提供了非常安全的系统环境。在本书中，我们在第18章，529页的"z/OS上的安全"中讨论了主机安全系统的一个例子。

1.7.3 可扩展性

俗话说唯一不变的便是"改变"。没有一个地方比IT业更加适用这句话了。商业的飞速发展常常导致IT基础设施为了应对日益增加的需求而增长。计算平台的可扩展性很大程度决定了IT机构在不破坏正常的业务流程，或带来额外开销(属于非生产性过程)的前提下进行功能增加的程度。

可扩展性
当系统加入处理器，内存和存储时保持系统性能级别的能力。

对于可扩展性，我们对其的定义是硬件、软件或者分布式系统在大小或者容量发生改变的时候仍然能够正常运转；例如，当在增加处理器、内存和存储器时系统仍然能够保持以往性能的能力即可称为可扩展性。不管网络规模的大小，执行任务的复杂程度的高低，一个可扩展的系统可以有效地适应工作环境的变化。

随着公司雇员，客户和商业伙伴数量的增长，公司常常需要增加运算资源以支持业务的增长。其中一个解决办法就是增加更多同样规格的处理器，但是管理这一更加复杂设备需要额外的管理费用。作为另一种选择，公司可以将很多较小的处理器统一为少数却较大的系统。应用主机，许多公司显著的降低了它们资产管理的总花费(TCO),这些花费并不仅仅是机器的开销(包括它的软件和硬件)，还包括运行的开销。

主机可以将操作系统的多个拷贝运行为一个单一实体，这个实体被称为系统综合体 ，它展示了主机在软件和硬件上的可扩展特性。我们将在54页的2.8节"什么是并行系统综合体?"中进一步探究主机的集群技术和它的应用。

14

1.7.4 持续兼容性

主机客户往往会对他们的应用程序和数据进行很大的金融投资。一些应用程序的开发和完善时间往往长达数十年。一些应用程序可能是许多年以前编写的，而其他应用程序也许是"昨天"才编写完成的。应用程序在系统中的工作能力或者它们与其他设备或程序协同工作的能力被称为兼容性。

兼容性
系统同时运行需要新老硬件指令的所有软件的能力。

由于需要支持不同年代的应用程序，人们对主机的软硬件提出严格的兼容性要求，这些软硬件自从第一台System/360主机产品在1964年推出以来已经升级了许多次了。应用程序必须继续正常工作。因此，对于新硬件和系统软件的许多设计工作都是围绕着这些兼容性需求开展的。

对兼容性需求的最重要的原因也是为什么系统的许多方面这样工作的最主要原因，以作业控制语言(JCL)的语法限制的为例，JCL是用来控制批处理作业的。任何对JCL的新增设计必须保证对老作业的兼容性以便这些作业不经过更改仍能继续运行。对于持续兼容性的期望和需求是主机的一大内在特征。

实现跨越十数年变迁和升级的绝对兼容性当然是不可能的，但是主机软硬件的设计者将这作为第一要务。当某一非兼容性不可避免的时候，设计者们通常会至少提前一年提醒用户软件可能需要改变。

1.8 典型的主机负载

大部分主机负载都可以分为这2个部分：批处理和在线交易处理，后者包括基于网络的应用程序(图1-1)。

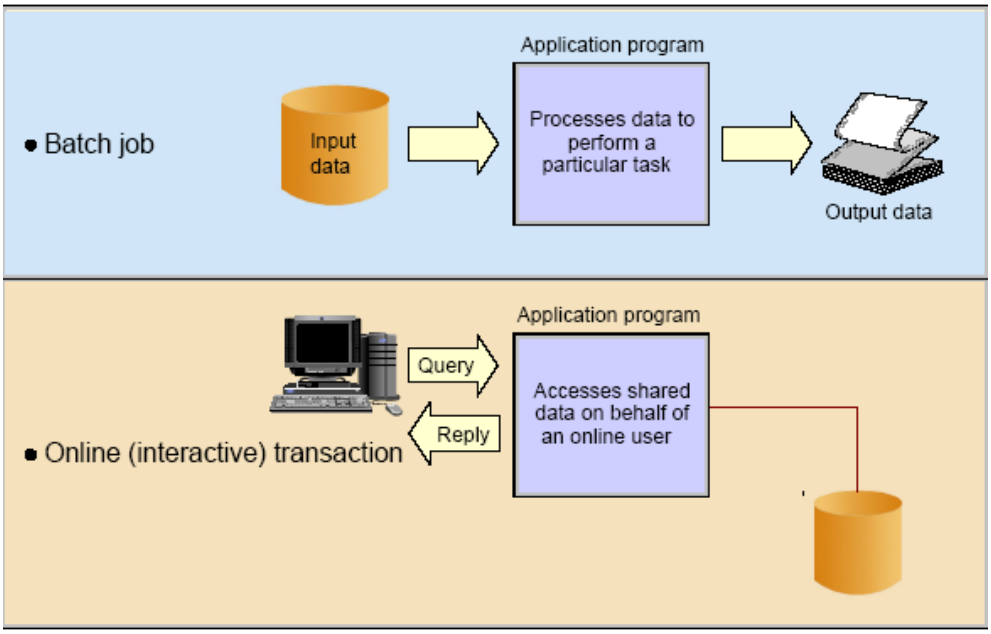


图1-1 典型的主机负载

这些负载在本书的几个章节中都会有讨论，以下部分提供其概要。

1.8.1 批处理

批处理
主机上无需用户交互而运行的作业。

主机一个关键的优势在于它能处理存储在高速存储设备上的海量数据，输出有价值的结果。比如，银行和其他金融机构可以利用主机系统完成季度结算，生成客户需要的报告(如季度股票或养老金报表)或政府需要的报告(如金融业绩表)。有了主机，零售商店可以每晚生成和整合销售报告，供给地区销售经理检阅。

生成这些报表的是批处理程序，确切地说，它们在主机上运行的时候不需要与用

户交互。一个批处理程序提交给计算机，而后读入并且处理大量数据——可能是海量数据——并且输出结果，比如客户账单表。一个等价的概念是UNIX脚本文件或Windows命令文件，但是z/OS批处理作业可以处理百万条记录。

分布式系统也支持批处理处理，但是并没有主机上常见，因为分布式系统通常缺乏：

16

- 足够的数据存储
- 优越的处理器处理能力
- 在系统综合体范围内进行资源和作业调度管理

主机操作系统通常配备复杂的作业调度软件，它允许数据中心的员工提交和管理批处理作业，并跟踪其执行情况及其输出结果[5]。

批处理通常具有以下特性：

- 要处理和保存大量输入数据(可能是兆兆位或者更多)，访问大量记录，输出大量信息。
- 通常来说，很短的响应时间是不必要的。然而，批处理作业通常必须在'批处理窗口'时间内完成，在这段时间内在线活动不紧密，服务等级协议(SLA)对此有规定。
- 生成的信息通常和大量用户和数据实体有关。(比如，客户订单或零售商手头的存货)
- 一个调度好的批处理包含成百上千个按照事先预定好的顺序执行的作业。

在批处理的过程中，可以运行多种类型的工作。整合信息比如投资基金的收益率，预定的数据库备份，日常订单的处理和存货更新都是常见例子。图1-2显示了在一个典型的主机环境中运行的若干批处理作业。

在图1-2中，在调度好的批处理过程中考虑以下因素：

1. 晚上，不计其数的执行了函数和实用程序的批处理作业被处理。这些作业整合白天在线交易的结果。
2. 批处理作业生成商业统计报告。
3. 在'批处理窗口'前后均需要备份重要文件和数据库。
4. 第二天商业统计报告被送往指定地点进行分析。

17

5. 有例外情况的报告被送往分支办事处。
6. 生成账户月收支平衡结算报告，发送给每个银行客户。
7. 处理总结报告送往合作的信用卡公司。

[5] 早期，主机通常使用穿孔卡片将作业输入系统中执行。打孔机打孔的操作员使用卡片打孔机输入数据，由此一叠卡片(或一批)就产生了。这些卡片进入读卡机，将作业和数据读入到系统中。您可以想象，过程非常繁琐且容易出错。现今，可以通过一个 PC 文本文件传送相同的打孔卡片数据。我们将在 231 页的第 7 章“批处理和 JES”中介绍多种将作业输入到主机的方法。

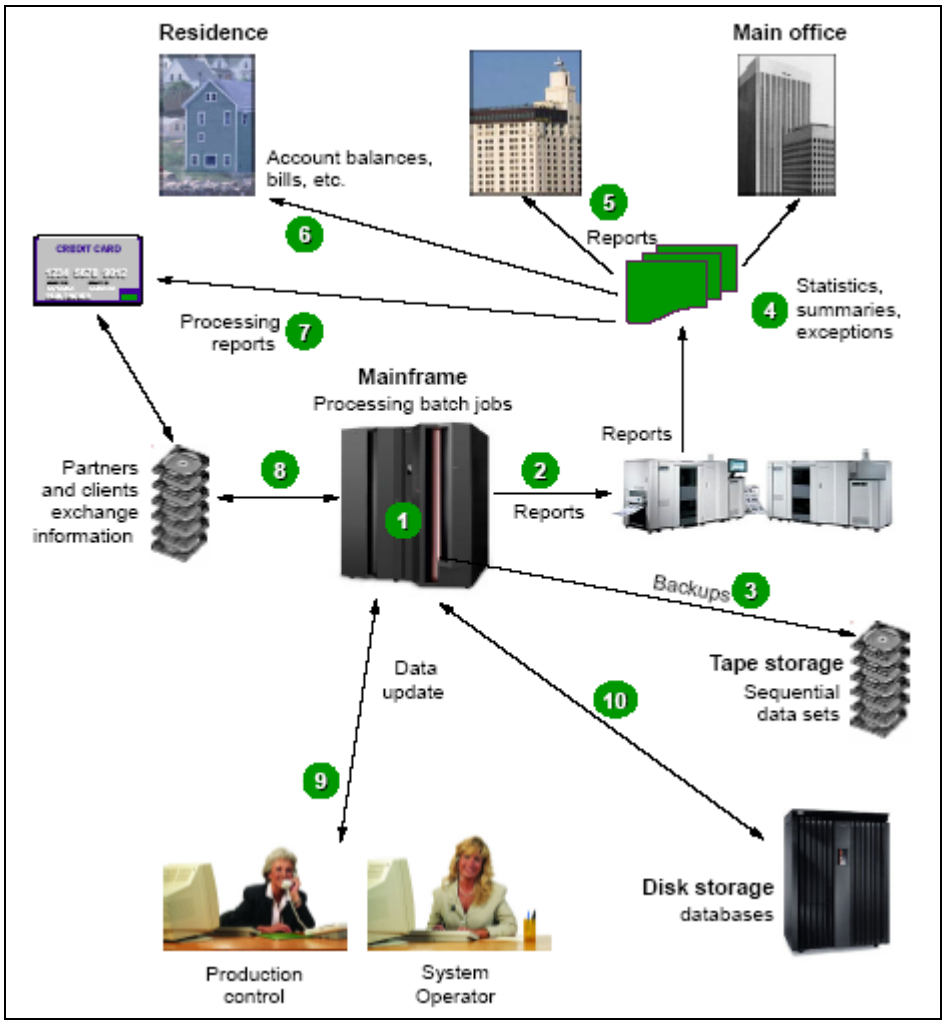


图1-2 典型批处理使用

8. 从合作伙伴公司处接收信用卡交易报告。
9. 在生产控制部门，操作员们监控在系统控制台上出现的信息和众多作业的执行。
10. 作业和交易读取或更新数据库(同在线交易使用同一个数据库)，同时将很多文件保存到磁带上。

1.8.2 在线交易处理

和终端用户有交互行为的交易处理被称为在线交易处理或OLTP。一般来说，主机为大量的交易系统提供服务。这些交易系统往往非常重要，公司依靠它们实现自身的核心功能。交易系统必须能供数量无法估计的用户同时使用，还要支持各种各样的交易类型。大多数交易在很短时间内完成——有时候一秒都不到。

交易系统有一个主要特征：用户和系统间的交互时间很短。通过几次简短交互，

用户就可以完成一个商务交易，其每次交互的相应时间都很短。这些系统中通常运行着非常关键的应用程序，因此它要有持续的可用性，卓越的性能，同时要在数据保护和维护数据完整性方面有出色的表现。

大多数人对在线交易都很熟悉，举几个例子：

在线交易处理(OLTP)
与终端用户交互的交易处理。

- ATM机器交易，例如存钱，取钱，查询和转账
- 使用借记卡和信用卡在超市付账
- 网上购物

举例来说，在某银行支行内或者通过网上银行，当客户查询账户余额或进行基金操作的时候，他们就是在使用在线交易。

实际上，在线系统的功能和操作系统有很多相似之处，比如：

- 管理和分发任务
- 控制用户对系统资源的访问权限
- 管理内存使用
- 管理和控制数据文件的同步访问
- 提供设备无关性

一些产业使用基于主机的在线系统，其中包括：

- 银行业——提供客户服务的ATM机，柜员系统
- 保险业——为策略管理和索赔处理设计的代理系统
- 运输业——航空订票系统
- 制造业——库存控制，生产调度
- 政府部门——税收处理，许可证的发行和管理

在以上这些产业中，终端用户是如何和主机交互的呢？公司交易处理系统的设计受许多因素的影响，这些因素包括：

- 某个时间点与系统交互的用户数量
- 每秒交易数(TPS)
- 应用程序的可用性要求情况。例如，该应用程序必须一天24小时，一周7天可用吗？抑或可以在每周的某个晚上短暂停机？

在PC和智能工作站流行之前，和在线主机应用程序交互的最常见方法就是使用3270终端。这些设备有时被称为‘哑’终端，不过它们已经有足够的能力去收集和显示整屏的数据而不是每次按键都与主机交互，以此节省处理器周期。主机程序在黑色屏幕上显示绿色字符，所以在当时有着‘绿屏’程序的昵称。

基于这些因素，不同主机系统间的用户交互不尽相同。结合当前正在设计的应用程序，主机系统正在改写其上已经存在的应用程序，使用户可以通过网络浏览器和主机应用程序进行交互。这个工作有时需要开发新程序，不过却常通过购买第三方软件来实现为应用程序”整容”的目的。这样一来，终端用户通常意识不到在后台有主机的存在了。

本书里，我们并没有必要去描述用户通过网络浏览器和主机交互的过程，这和任何通过网络交互的过程基本一样。唯一的差别就在于交互另一端的机器不同。

在线交易通常有以下特点：

- 少量的输入数据，少量的存储记录被访问与处理，输出的数据也很少
  很短的响应时间，通常小于1秒
- 用户数量大，交易数量多
- 用户的交易终端需要全天候的可用
- 交易安全和用户数据安全须得到保障

举一个例子：在一个银行支行中，客户使用在线交易服务去查看账户余额或做一笔投资。

20

图1－3 显示了使用主机的一系列常见在线交易。

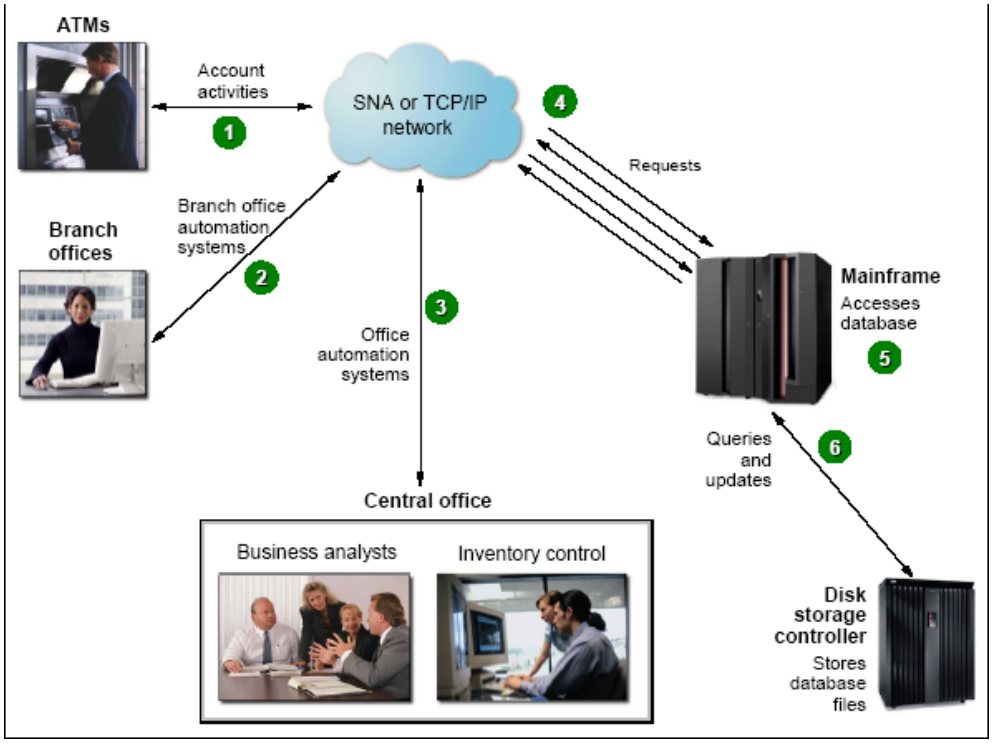


图1-3 典型的在线应用

1. 一个用户使用ATM提供的友好的用户界面来完成多种功能：取钱，查询账户，存钱，转账或从信用卡预支现金。
2. 在同一个内网的别处，银行支行的一名员工完成一些操作，诸如咨询，基金申请和邮汇业务。
3. 在银行的总部，商务分析师为了提升性能而调整交易。其他职员使用专门的在线系统进行办公自动化操作，完成客户关系管理，预算规划和库存控制等工作。
4. 所有的请求都交给主机处理。
5. 在主机上的程序对数据库管理系统(比如DB2)进行查询和更新操作。
6. 将数据库文件存储在专门的磁盘存储系统中。

21

1.9 主机世界中的角色

主机系统被设计成能被大量人使用。大多数与主机交互的人是终端用户----他们使用在主机上运行的应用程序。但是由于终端用户多，系统中运行的应用程序也很多，用于支持他们的系统软件具有混合性和复杂性，故操作和支持主机系统需要多种不同角色的人员。

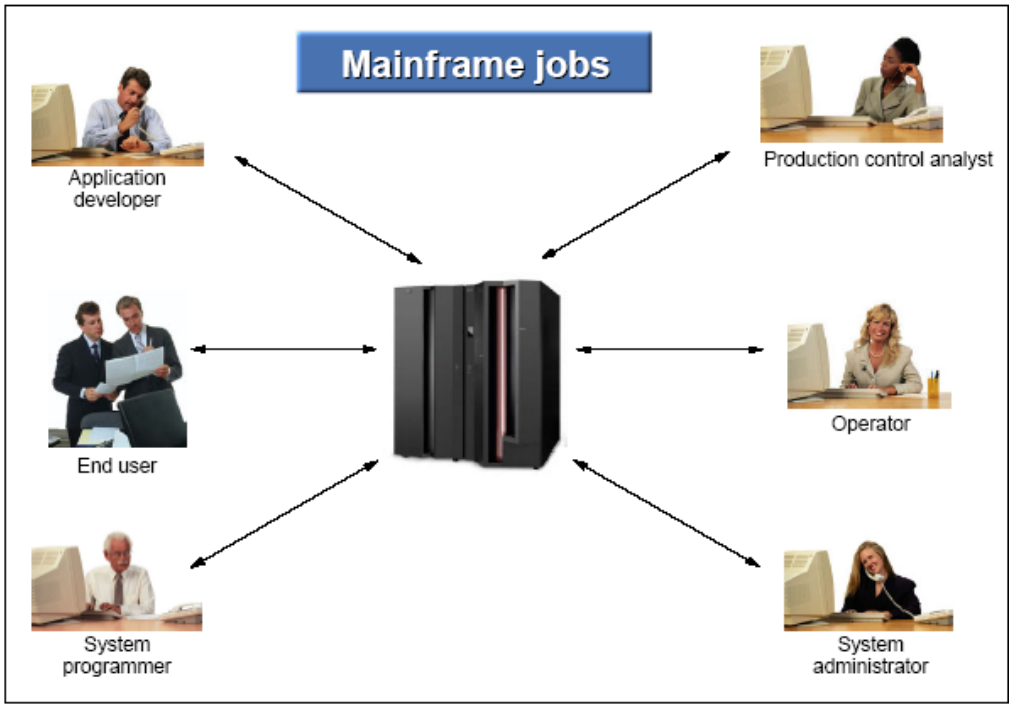


图1-4 主机世界中的角色

在IT界中，这些角色被冠有不同的名称，本书采用的名称如下：

- 系统程序员
- 系统管理员
- 应用程序设计人员和开发人员
- 系统操作员
- 生产控制分析员

在分布式系统环境中也需要很多类似的角色。然而各角色工作职责的划分却没有主机那么精确。从20世纪60年代开始，主机的角色在不断的演化扩大，提供了一个能让系统软件和应用程序平稳有效运行的环境，并对成千上万的用户提供有效的服务。虽然主机技术支持人员的数量看起来很大且难于管理，但考虑到主机支持的用户数量，在主机上运行的交易数量和在主机上执行的工作具备的巨大商业价值，这个数量相比下就很小了。

22

这本书主要介绍主机环境中的两个角色：系统程序员和应用程序开发人员。但还有一些其他重要角色来支持主机运行，我们会略微谈到这些角色以让您对这些"幕

后英雄"们有更深的了解。

- 主机相关活动经常需要各个角色之间的合作，举几个例子：
- 安装和配置系统软件
- 设计和编码在主机上运行的新的应用程序
- 引入和管理系统中新的负载，比如批处理作业和在线交易处理
- 主机软件和硬件的操作和维护

在以下部分，我们会详细描述每个角色。

1.9.1 谁是系统程序员？

系统程序员
安装，客户化和维护操作系统的人。

在主机IT机构中，系统程序员扮演着核心角色。系统程序员安装操作系统，并对其进行客户化和维护工作，同时他们也安装和升级运行在操作系统上的主机产品。系统程序员可能被给予了最新的操作系统来替代现有系统。有时，他只需要升级单个简单程序，例如排序程序。

系统程序员要做以下工作：

- 规划硬件和软件系统的升级和配置更改工作。
- 令操作自动化
- 容量规划
- 运行系统作业和脚本
- 执行系统指定的客户化任务
- 培训系统操作员和应用程序开发人员
- 结合现有的应用程序和用户过程对新产品进行集成测试
- 进行系统级别的性能调优来满足服务品质需求

系统程序员必须具有调试系统软件问题的技能。这些问题常常可以在计算机内存区域的一个拷贝中捕获，该拷贝称为转储(_DUMP)，是系统在软件产品，用户作业或交易失败的情况下产生的。根据这份转储和专业的调试工具，系统程序员可以确定组件出错的地方。当错误在软件产品中发生时，系统程序员直接和软件厂商的支持人员一起来探究问题根源是否已知，并且是否存在可用的补丁程序。

系统程序员需要对主机上的中间件进行安装和维护，比如数据库管理系统，在线交易处理系统和网络服务器。中间件是在操作系统和终端用户或终端用户应用程序之间的一个软件层。它提供操作系统无法提供的功能。主要的中间件产品比如DB2，CICS和IMS可以和操作系统一样复杂，甚至更复杂。

1.9.2 谁是系统管理员？

系统管理员
维护存放在主机上的重要商业数据的人。

系统程序员和系统管理员的区别在各个主机系统之间差别甚大。在规模小一些的主机公司中，一个人可能同时担任很多角色，所以角色的名称就会交替使用。

在较大的IT机构中有很多部门，因此工作职责也会更清晰地划分。系统管理员更多地完成与维护主机上重要商业数据相关的一些日常工作；而系统程序员则关注

维护系统本身。这样区别职责的一个原因是要和审计过程相符。因为审计要求IT企业中不能有人对敏感数据和资源拥有不受限制的权限。举例来说，系统管理员包括数据库管理员(DBA)和安全管理员。

系统程序员主要专攻主机硬件和软件领域，系统管理员更多关注应用程序领域。他们经常和程序员和终端用户直接接触以确保满足应用程序管理方面的需求。在
24 主机环境中这些角色未必是非常特别的，但却是稳定操作的关键所在。

在大型IT企业中，系统管理员为实现商业目的维护系统软件环境，包括为保持系统平稳运行而进行的每日维护工作。比如，数据库管理员必须确保在数据库中存储的数据的完整性和访问有效性。

其他一些常见的系统管理员任务包括：

- 安装软件
- 添加和删除用户，维护用户档案
- 维护安全资源访问列表
- 管理存储设备和打印机
- 管理网络连接
- 监控系统性能

问题诊断方面，在系统程序员不参与的时候，系统管理员通常靠软件生产商技术支持中心的人员来诊断问题，读取转储并做出改正。

1.9.3 谁是应用程序设计人员和开发人员？

应用程序设计人员和开发人员为公司终端用户和客户设计，构建，测试和发布主机程序。基于从商业分析员和终端用户处收集的需求分析，设计人员写出详细的设计方案，供开发人员进行程序架构。在这个过程中，代码修改和编译，程序构建以及单元测试会反复进行。

在程序开发过程中，设计人员和开发人员必须与企业中其他角色进行交互。比如，一名程序员经常和开发相关模块的程序员组一起工作。完成之时，每个模块都会通过一个测试流程，包括功能测试，集成测试以及系统级别测试。测试之后，程序还要经过客户验收测试来决定代码是否真正满足最初的用户需求。

除了创建新的程序代码，程序员也负责维护和升级公司现有的主机程序。事实上，这才是现今很多主机程序员的主要工作。尽管主机仍旧可以用COBOL语言和PL/1语言编写新的程序，但对于构建新的主机应用程序，Java语言已经成为流行的选
25 择，正如在分布式平台上那样。

与传言相反，使用诸如COBOL和PL/1之类的高级语言的主机程序仍旧发展迅速。不计其数的程序运行在全世界的主机生产系统上，这些程序对于使用它们的公司的日常业务来说至关重要。ＣＯＢＯＬ和其他高级语言的程序员需要维护现有的代码，修改和更新现有的程序。也有很多公司继续使用ＣＯＢＯＬ和其他传统语言来实现新的应用程序逻辑，并且IBM也在持续不断地增强这些语言的高级语言编译器，通过增加新功能和新特性，以让这些语言能够继续利用新科技和新的数

据格式。

我们将会在本书的第二部分中继续深入讨论应用程序设计人员和开发人员的职责。

1.9.4 谁是系统操作员？

系统操作员监控主机硬件和软件的操作。操作员可以启动和停止系统任务，监控系统控制台获取异常情况，和系统程序员及生产控制工作人员一起确保系统的正常完好运行。

系统操作员
监控并控制主机硬件和软件操作的人。

当应用程序部署到主机上时，系统操作员负责保证它们平稳运行。新的应用程序将会从应用程序开发部门交付给操作员，同时给操作员提供"运行手册"作为参考。运行手册中标明了操作员需要了解的作业执行过程中应用程序特定的操作需求。运行手册可能包含一些内容，比如应用程序特定的控制台信息，这些信息需要操作员的干涉；对特定的系统事件操作员如何应答，如何修改作业流程以使它满足新的商业需求[6]。

操作员也负责启动和关闭主要的子系统，比如交易处理系统，数据库系统和操作系统本身。这些重启操作不像以前那么常用，因为历经数年主机的可用性已经大幅度提高。但是，操作员仍旧要在需要时依顺序关闭和启动系统及其工作负载。

万一发生故障或者遇到不正常的情况，操作员要和系统程序员沟通，由系统程序员帮助操作员制定适当的行为方案，同时要和生产控制分析员沟通，一起确保生产系统的工作负载正确完成。

1.9.5 谁是生产控制分析员？

生产控制分析员
确保批处理负载准时无误按时完成运行的人。

生产控制分析员负责保证批处理工作负载无错误无延迟地完成。一些主机系统运行交互式工作负载，为在线用户提供服务，当在线系统不运行时再切换去执行批处理更新操作。尽管这个运行模式目前仍旧很常见，很多公司的全球业务都是通过互联网实时地去访问生产数据，因此这种'白天在线处理，晚上批处理'的模式已经趋于陈旧。然而批处理工作负载仍是信息处理的一部分，技术卓越的生产控制分析员也依然扮演着重要的角色。

一些人对主机颇有微词：它们不具灵活性，很难操作，在有变化发生时尤其如此。生产控制分析员经常听到这样的抱怨，但是他们知道使用架构得当的规则和过程来控制变化(主机环境的强项之一)能够有利于防止系统崩溃。事实上，主机一直有高可用性和卓越性能的原因就是它们对变化有控制，如果没有使用正确的过程，您将很难引入改变。

[6] 控制台消息的数量曾经非常庞大，以至于操作员通常很难决定某个情形是否真的有问题存在。近年来，一些工具的出现让操作员可以专注于需要人为干预的异常事件中，这些工具减少信息量，自动对常规情形做出消息回应。

1.9.6 厂商扮演什么角色？

在主机中，厂商很常见。由于大多数主机是由IBM卖出，操作系统和主要的在线系统也由IBM提供，所以大多数厂商联系人都是IBM员工。但是，其他独立软件供应商的产品在IBM主机上也是可用的，客户也可以使用原始设备生产商(ＯＥＭ)硬件，比如磁盘或磁带存储设备。

典型的厂商角色如下：

- *硬件支持人员或客户工程师*

  硬件厂商经常为硬件设备提供现场支持。IBM硬件维护人员经常被称作客户工程师(ＣＥ)，他们提供主机硬件和外围设备的安装维修服务。当硬件故障或安装新硬件时，CE通常直接与操作小组一起工作。

27

- *软件支持*

  一些厂商的职责是为主机上的软件产品提供支持[7]。IBM有一个集中的"支持中心"，针对软件缺陷或用法协助，提供授权的额外付费支持。依据企业的大小及客户的特定情况，IT专家和架构师也会为软件产品提供附加的售前及售后支持。

- *现场技术销售支持，系统工程师或客户代表*

  对于大的主机客户，IBM和其他厂商提供面对面的销售支持。厂商代表精通各种软硬件产品线，负责和客户公司中影响产品采购的部门打交道。在IBM，技术销售专家被称为现场技术销售支持(FTSS)，早期的称呼为系统工程师(SE)。

  对于大的主机客户，IBM经常派驻一位熟悉某个领域业务的客户代表，专职和少量客户人员一起工作。客户代表的职能是在IBM各部门和客户之间成为单一的"联络点"。

1.10 z/OS 和其他主机操作系统

本书的很多部分都是教您z/OS的基本知识，这也是IBM最高端的主机操作系统。我们从第75页第3章的"z/OS概述"开始讨论z/OS的概念。不过，对于学习主机的学生来说，有其他主机操作系统的工作经验还是很有用的。原因之一在于一台特定主机上可能会运行多个操作系统。例如，在同一台主机上运行z/OS，z/VM和Linux是很常见的。

主机操作系统是很复杂的产品，各自的特性和用途大不相同，每一个产品都可以用一本书来详细介绍。除了z/OS，另外四个操作系统主导了主机的使用：z/VM, z/VSE™, Linux for zSeries, 和z/TPF。

28

[7] 本文并无意图检视主机软件的市场和价格。然而，中间件和其他特许程序的可用性和价格是影响主机使用和发展的一个重要因素。

1.10.1 z/VM

z/虚拟机(z/VM)有两个基本组件：一个控制程序(CP)；和一个单用户操作系统，CMS。作为一个控制程序，z/VM允许在它创建的虚拟机上运行其他操作系统，因此它是管理者。任何一个IBM主机操作系统，如z/OS, Linux for zSeries, z/VSE, 和z/TPF，可以在它们各自的虚拟机上作为客户系统运行，并且z/VM上可以运行任意组合的客户系统。

控制程序人工地从真实硬件资源中创建多个虚拟机。对于终端用户而言，好像他们独享了实际被共享的真实硬件资源。共享的真实资源包括打印机，磁盘存储设备和CPU。控制程序保证在客户系统中数据和应用程序的安全。真实硬件可以被多个客户共享，也可以出于性能原因为单个客户独享。系统程序员在多个客户之间分配真实的物理设备。对于大多数客户来说，客户系统的使用避免了更大硬件配置的需要。

z/VM的另一个主要部件是会话监视系统(CMS)，它运行在一台虚拟机上，既提供交互式终端用户接口，也提供通用的z/VM应用程序编程接口。

1.10.2 z/VSE

z/扩展虚拟存储(z/VSE)受到小型主机用户的欢迎。这些用户中的一部分在z/VSE的性能不能满足要求时，最终会迁移至z/OS。

与z/OS相比，z/VSE操作系统为在线处理和批处理提供了一个相对较小，不那么复杂的基础平台。z/VSE的设计和管理架构非常适合运行例行的生产负载，包括多个批处理作业(并行运行)以及广泛的传统交易处理。实际上，绝大多数z/VSE用户也有z/VM操作系统，并把它用作z/VSE应用开发和系统管理的通用终端接口。

z/VSE原先叫磁盘操作系统(DOS)，是第一个被引入到System/360主机系统中的基于磁盘的操作系统。DOS曾被认为是OS/360就绪前的暂时举措。然而，一些主机用户喜欢它的简单(以及短小)，决定在OS/360发布后继续使用它。DOS后来被称为DOS/VS(当它开始使用虚拟存储之后)，接着又被称为VSE/SP以及后来的VSE/ESA，最近的名字是z/VSE。VSE这个名字经常被用作任何最新版本的统称。

1.10.3 zSeries 中的 Linux

若干(非IBM)Linux版本可以用在主机上。这些Linux版本有两个通用的名字：

- S/390中的Linux (使用31位寻址和32位寄存器)
- zSeries中的Linux(使用64位寻址和寄存器)

‘Linux on zSeries ’这个短语在不必明确指出是31位版本或64位版本时，指的是运行在S/390或zSeries系统上的Linux。我们假设学生了解Linux，因此我们只提和主机使用相关的一些特性，包含如下：

Linux使用传统计数关键数据(CKD[8])磁盘设备以及和SAN连接的SCSI类型设备。其他主机操作系统可以认出这些驱动是Linux驱动，不过不能使用这些驱动上的数据格式。这就意味着， Linux和其他主机操作系统之间不能进行数据共享。

- Linux不使用3270显示终端，而其他主机操作系统使用3270作为它们的基本终端架构[9]。Linux使用基于X Window系统的终端或PC上的X-Window系统模拟器。它也支持典型的ASCII终端，通常是通过telnet协议连接的。X-Window系统是Linux图形接口的标准，它是硬件和窗口管理程序之间的中间层。
- 若设置正确，z/VM下的Linux系统可以被快速克隆为另一个独立的Linux镜像。z/VM模拟的LAN可用来连接多个Linux镜像，并为它们提供一条外部LAN的路由。只读文件系统，比如典型的/usr文件系统，可以被多个Linux镜像共享。
- 主机上的Linux通过ASCII字符集进行操作，而不是主机上存储数据常用的EBCDIC[10]形式。这里，EBCDIC只有在写入字符敏感设备，如打印机和显示器时才被使用。这些设备的Linux驱动程序处理字符转换。

1.10.4 z/TPF

z/交易处理工具(z/TPF)操作系统是有着特殊目的的操作系统，用于那些有着非常高交易量的公司，如信用卡公司和航空公司预订系统。z/TPF本来叫做航空控制程序(ACP)。目前它仍为航空公司所使用，并被扩展到其他有着高速、高交易量处理要求的超大型系统中。

z/TPF可在松散耦合的环境里使用多台主机，来例行处理每秒上万次的交易，同时提供多年不间断的高可用性。规模非常庞大的终端网络，包括在部分"预定"产业中使用的具有特殊协议的网络，是很常见的。

1.11 总结

今天，主机在绝大多数世界最大型企业中的日常运作中扮演了核心角色，这些企业包括许多财富1000强的公司。尽管其他类型的计算被大量应用在不同级别的商务活动中，主机仍然在当今的电子商务环境里占据着令人羡慕的地位。在银行，金融，医疗，保险，公用事业，政府机关，和大量的其他公有及私有企业中，主机继续构成现代商务的基础。

主机如此流行并经久不衰，要归功于它一直以来的可靠性和稳定性，这是从1964年IBM推出System/360之后不断科技进取的结果。现存的计算机架构中，没有哪个像主机一样在保持现有程序兼容性的同时不断变革创新。

主机这个术语从对IBM大型计算机的物理描述慢慢发展到计算风格的分类。一个

[8] CKD 设备被格式化，这样磁盘的读磁头可以直接访问个体数据块。

[9] 在一个限制性模式下，存在一个最小化 3270 操作的 Linux 驱动器，但人们却不常使用。3270 终端是一个满屏缓冲的，非智能的终端，用控制单元和数据流来最大化数据传输的效率。

[10] EBCDIC 代表广义二进制编码的十进制交换码(extended binary coded decimal interchange code)，是一个由 256 个 8 位字符组成的编码字符集，为表示文本数据而开发。EBCDIC 并不和 ASCII 码兼容。593 页的附录 D"EBCDIC ASCII 表"就是一个便捷的换算表。

决定性特性就是主机在几十年中体现的持续兼容性。

在一个主机IT机构中，角色和职责范围宽泛且互不不同。技术卓越的工作人员保持主机平稳可靠运行。看起来或许一台主机环境中需要的资源远比小规模的分布式系统要多。但是，如果分布式系统中角色被全部确定，也会存在很多相同的角色。

目前，若干操作系统可以在主机上运行。本书主要介绍其中的一个：z/OS。但是学生应该知道其他操作系统的存在，并理解它们相对z/OS的地位。

31

本章关键术语				
架构 (architecture)	可用性 (availability)	批处理(batch processing)	兼容性 (compatibility)	电子商务 (e-business)
主机 (mainframe)	在线交易处理 (online transaction processing，OLTP)	平台(platform)	生产控制分析员 (production control analyst)	操作手册(run book)
可扩展性 (scalability)	可扩展性 (scalability)	系统操作员 (system operator)	系统程序员 (system programmer)	360系统 (System/360)

1.12 复习题

为了帮助测试您对本章的理解，完成以下问题：

1. 列出主机挑战关于‘集中式计算对分布式计算’这个传统观点的几个方面。
2. 解释企业如何利用主机的处理能力，以及主机计算与其他形式的计算的不同之处。
3. 列出主机计算的3个优势，简述最适合主机的主要工作负载类型。
4. 说出5种和主机计算相关的工作或职责。
5. 本章提到了至少5个主机使用的操作系统。选择其中的三个，描述它们每一个的主要特性。

1.13 思考题

1. 今天，主机是指什么？这个术语如何产生的？它还合适么？
2. 为什么对老的应用程序维持系统兼容性非常重要？为什么在有改进的接口出现时，不去简单改变现有的应用程序编程接口？

32

3. 鉴于运行一个主机系统需要众多角色，描述怎样运行一台主机才能符合成本效益？
4. 鉴于主机世界中存在的各种角色，在主机处理环境中会有什么样的好的或者坏的特性呢？(效率？可靠性？可扩展性？)

5. 大多主机业务执行严格的系统管理，安全和操作过程，这些是否也在分布式系统环境中执行呢？请说出原因。

6. 您可以找出主机应用在您日常生活中的例子么？对他们展开描述并阐述主机对终端用户透明的程度。例子可以包括如下：
   - 常见的网站，用主机作为后台服务器来支持在线交易和数据库。
   - 当地使用主机的情况。可以包括银行和金融中心，主要零售商，交通枢纽和医疗保健行业。

7. 您能找到日常使用的分布式系统的例子么？通过增加一台主机，这些系统能够加强吗？具体怎样实施呢？

2

第 2 章 主机硬件系统和高可用性

目标：作为一个新的主机系统程序员，您将需要全面了解运行z/OS操作系统的硬件。z/OS被设计成能充分利用主机硬件及其复杂的外围设备。您也应该知道硬件和软件是如何通过并行系统综合体和无单点故障等理念，达到近乎不间断的可用性。

阅读本章后，您应该可以：

- 讨论S/360和zSeries的硬件设计
- 解释处理单元和磁盘硬件
- 解释主机和PC机在数据编码方面的差别
- 列出一些典型的硬件配置
- 解释并行系统综合体如何达到不间断的可用性
- 解释动态工作负载平衡
- 解释单系统镜像

35

2.1 主机硬件系统介绍

本章提供了一个主机硬件系统的综述，重点介绍处理器 ‘机盒’。

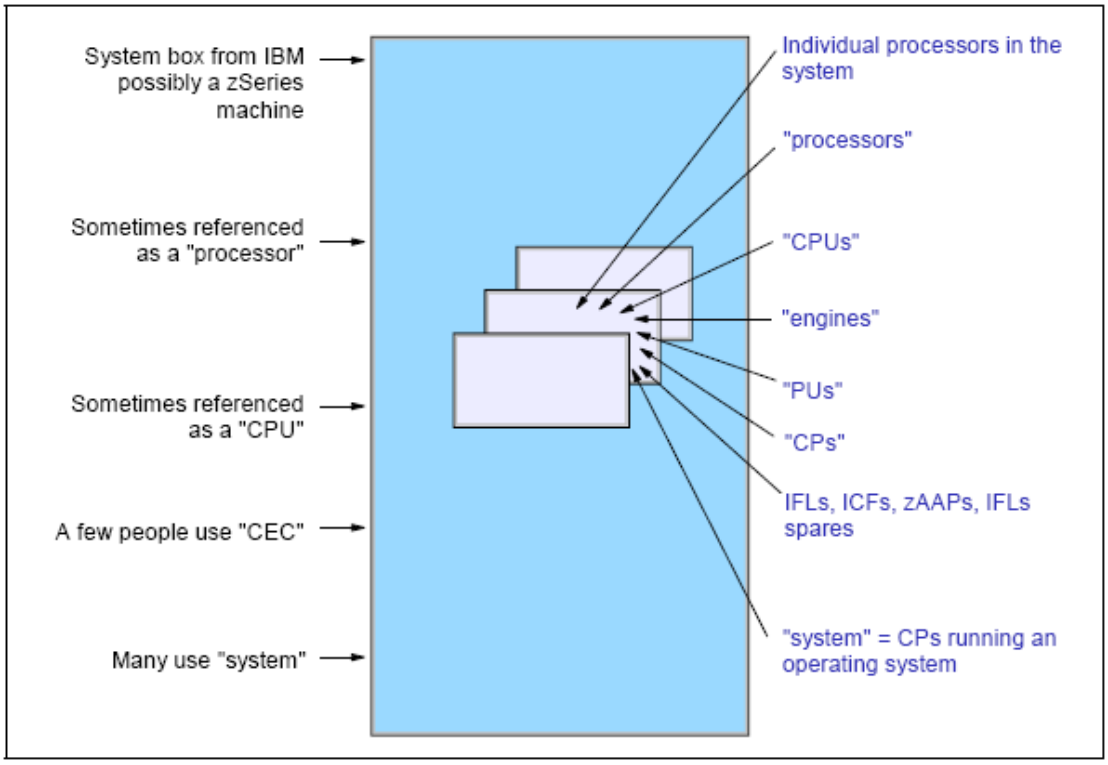
相关阅读：要知道更多z架构主要工具的详细信息，请阅读‘z/3 Principles of Operation’。您可以在z/OS互联网知识库网站中找到它和其他的IBM出版物：

http://www.ibm.com/servers/eserver/zseries/zos/bkserv/

本章我们首先了解一下主机硬件相关的术语。知道这些术语(系统，处理器，中心处理器等)的多种意思，对您了解主机很重要。

CPU
与处理器同义。

在早期的S/360时代，一个系统只有一个处理器，也被称为中央处理单元(CPU)。这些诸如系统，处理器，中心处理器等的术语可以交替使用。但是当系统有多个处理器的时候，这些术语就变得容易混淆了。如图2.1所示。



36 *图2.1 术语重叠*

处理器和CPU可以指一个完整的系统机盒或者系统机盒中的一个处理器。从讨论的上下文来看语义可能会很清晰，但甚至主机专家在用到这些术语时也必须阐明处理器或CPU的具体含义。系统程序员使用IBM的术语‘中央处理复合系统’(CPC)来表明主机的机盒。本书中，我们使用CPC来指定包含主存，一个或多个中央处理器，计时器和通道的多个硬件物理集合。

CPC
硬件的物理集合，包括主存，一个或多个中央处理器，计时器和通道。

本章将稍后讨论图2.1中的分区和一些术语。简单来说，所有在CPC中的S/390或z/架构处理器都叫处理单元(PU)。当IBM发布CPC概念时，CP(用于正常工作)，Linux集成工具(IFL)，以及并行系统综合体配置中所用的集成耦合设施(ICF)等均表现出PU的特征。

本书中，我们希望‘系统’和‘处理器’的词义在上下文中是清晰的。一般”系统”按照上下文的不同可以指硬件机盒，一个完整的硬件环境(有I/O设备)，也可以指一个操作环境(软件)。我们通常使用”处理器”指示CPC中的单个处理器。

2.2 早期系统设计

中央处理器机盒包括处理器，内存[1]，控制线路和通道的接口。通道在I/O设备和内存之间提供了一条独立的数据及控制路径。早期的系统最多拥有16条通道；在撰写该文时最大的主机可以拥有超过1000条通道。

通道和控制单元相连接。控制单元包含了连接某种类型I/O设备的工作逻辑。例如，一个打印机的控制单元和磁带机的控制单元在内部电路和逻辑上都有很多不同。一些控制单元可以有多个通道连接，为控制单元及其连接的设备提供了多条路径。

控制单元连接着设备，比如磁盘驱动器，磁带驱动器，通信接口等等。控制单元与设备之间的电路和逻辑并没有明确分开，但是把大部分电路放到控制单元中通常会更经济。

图2-2展示了一个S/360系统的概念图。图2-2中没有连接现有系统。但是，该图帮助解释了贯穿主机讨论的背景术语。

[1]一些 S/360 有单独的盒子存放内存。然而，本书是概念性的讨论，我们忽略这些细节。

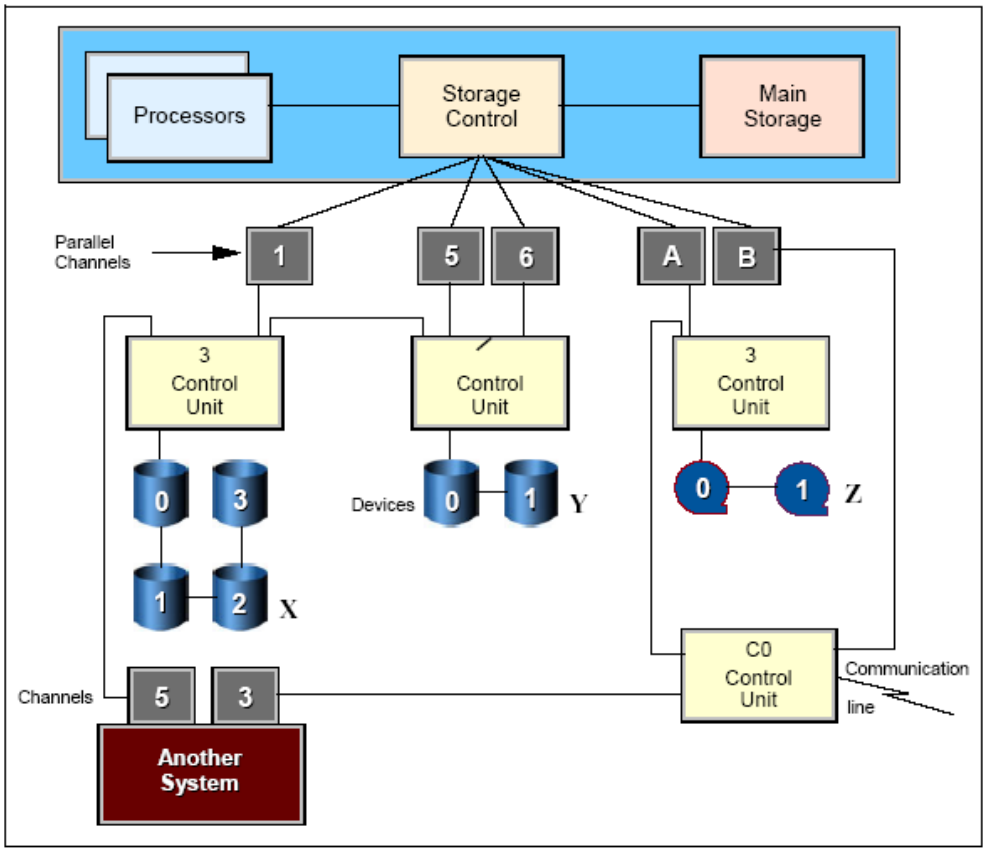
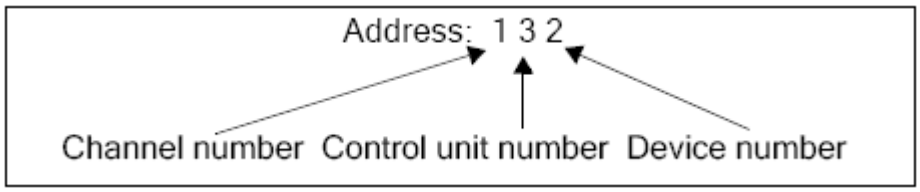


图2-2 S/360概念图

图2-2中的通道是并行通道(也称为总线和标签通道，名字来源于它使用2根重铜缆)。一个并行通道可以最多连接8个控制单元。大多数控制单元可以连接到多个设备，最大数由特定的控制单元决定，不过通常来说最大数为16。

每个通道，控制单元和设备都有一个地址，由16进制数表示。图2-2中有X标志的磁盘驱动器地址为132，地址位见图2-3：



38

图2-3：设备地址

在图中有Y标记的磁盘驱动器的地址为171，571或671，因为它连接到3个通道。按照常规，设备采用最低地址访问(171)，但是所有的三个地址都可以被操作系统使用来访问磁盘驱动器。设备的多条访问路径对提高性能和可用性很有帮助。当一个应用程序要访问磁盘171，操作系统先尝试通道1.如果通道1忙或不可用，它会尝试通道5等等。

图2-2包含另外一个S/360系统，它的2个通道连接到了第一个系统使用的控制单

元。这样共享I/O设备在主机系统中很常见。磁带驱动器Z地址对第一个系统来说为A31，不过对第二个系统来说地址为331。共享设备，特别是磁盘驱动器的共享，不是一个简单的问题，操作系统采用了软硬件技术来控制数据共享，比如两个独立的系统中同时更新同一磁盘数据。

ESCON
企业系统连接

- 如我们所说，现今主机的使用不是完全如38页的图2-2所示。不同之处在于：
- 并行通道在最新的主机上不可用，在旧系统上也在慢慢被取代。
- 并行通道已经被替换成ESCON(企业系统连接)和FICON(光纤连接)通道。这些通道均是光纤，他们仅连接一个控制单元，或者更多时候，它们被连接到一个交换器。
- 现今主机的通道超过16条，固使用2个16进制位来表示一个地址的通道部分。
- 虽然术语通道(Channel)依然准确，在之后的系统中通道一般叫做CHPID(通道路径指示符)，或者PCHID(物理通道指示符)。这些通道全部集成在主处理器机盒中。

软件使用的设备地址更准备地说是设备号(虽然术语”地址”一词仍旧被广泛使用)。设备号并不直接和控制单元及设备地址相关。

想要了解更多的IBM主机在1964年之后的发展，请看附录A，565页的”IBM主机历史的简要回顾”。

2.3 当前设计

现今的CPC设计比早期的S/360复杂了许多，复杂性体现在以下多个方面：

- I/O连通性和配置
- I/O操作
- 系统分区

2.3.1 I/O 连通性

41页上的图2-4展示了一个较新的配置。一个真实的系统将会有更多的通道和I/O设备，不过该图已经阐释了关键概念。分区，ESCON通道和FICON通道以后再讨论。

简单来说，在CPC中，分区创建了逻辑分离的机器。ESCON和FICON通道逻辑上类似并行通道，不过它们使用光纤，速度更快。如今，一个系统可能有100到200条通道或CHPID[2]。图中展示的要点概念包含以下几点：

CHPID
通道路径标识符

- ESCON和FICON通道只连接一台设备或者交换机上的一个端口。
- 大多数现代主机在通道和控制单元之间使用交换机。交换机可能和多个系统相连接，在多个系统间共享控制单元及与控制单元相连的一部分或全部I/O设备。

[2] 最新的主机机器可以用超过 256 条通道，但是需要额外的装置。通道分配方式为只用两个十六进制位指定 CHPID 地址。

- 一些或所有系统上它的I/O设备。
- CHPID地址是两个16进制数字。
- 多个分区之间有时可以共享CHPID。是否可以共享取决于CHPID中使用的控制单元的类型。总体来说，磁盘使用的CHPID可以被共享。

I/O子系统层位于分区(如果无分区则是基本机器)的操作系统和CHPID之间。

ESCON或FICON交换器是非常复杂的设备，它们可以支持很多连接之间的高速数据传输。(例如，一个大型控制器可能有200个连接，所有这些连接可以同时传输数据。)交换器必须追踪哪个CHPID(和分区)发起了哪次I/O操作，这样数据和状态信息就能返回至正确的地方。从多个系统上的分区连接的CHPID发出的很多I/O请求，可以通过一个单独的控制单元中进行。

I/O控制层使用一个控制文件，叫IOCDS(I/O控制数据集)，该控制文件可以解析物理I/O地址(由CHPID编号，交换器端口号，控制单元地址和设备地址组成)至设备号，操作系统软件通过这些设备号访问设备。IOCDS在开机时就装入硬件保存区(Hardware Save Area ，HSA)，我们也可以的动态修改它。设备号看起来像我们之前描述过的早期S/360机器的地址，区别在于设备号可以包含3或4位十六进制位。

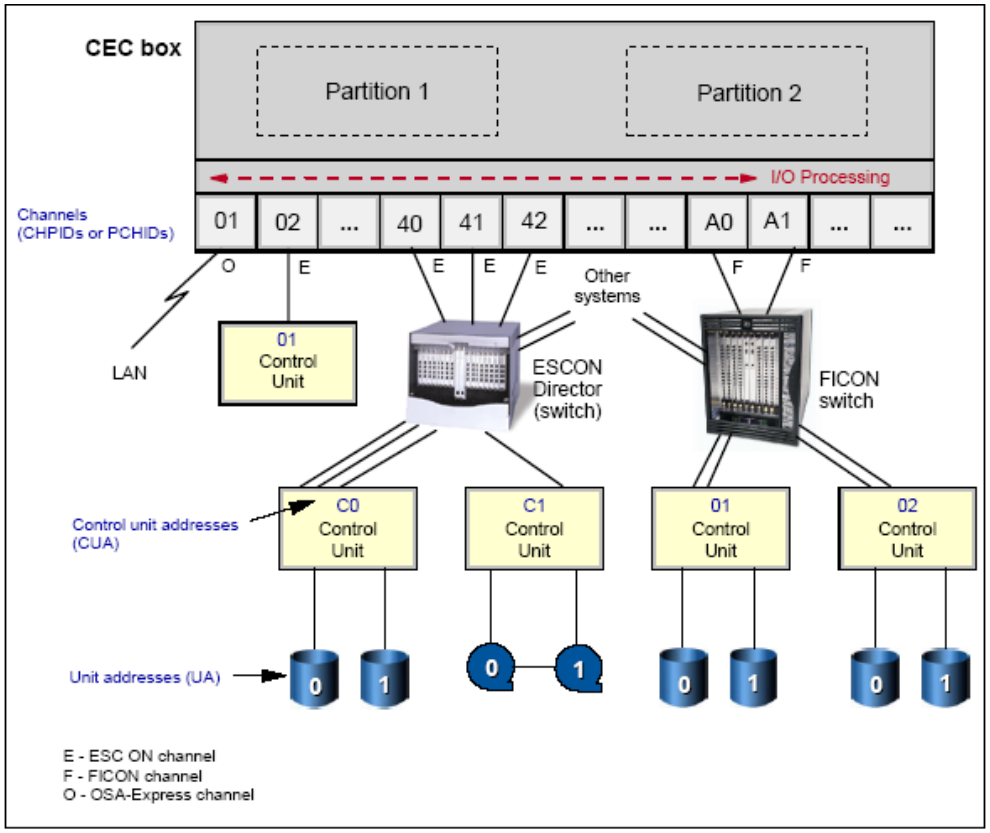


图2-4 较新的系统配置

虽然设备号是x'0000'到x'FFFF'之间的任意数值，但很多用户仍旧将它们称为'地址'。撰写本书时的，最新型主机在真实I/O设备和操作系统软件之间有2层I/O地址解析层。添加第二层是为了更容易地迁移到更新的系统。

现代控制单元，特别是磁盘的控制单元，通常连接了很多通道(或交换机)和很多设备。它们可以在不同的通道上同时处理多个数据传输。每个设备在z/OS镜像上将有一个单元控制块(UCB)。

41

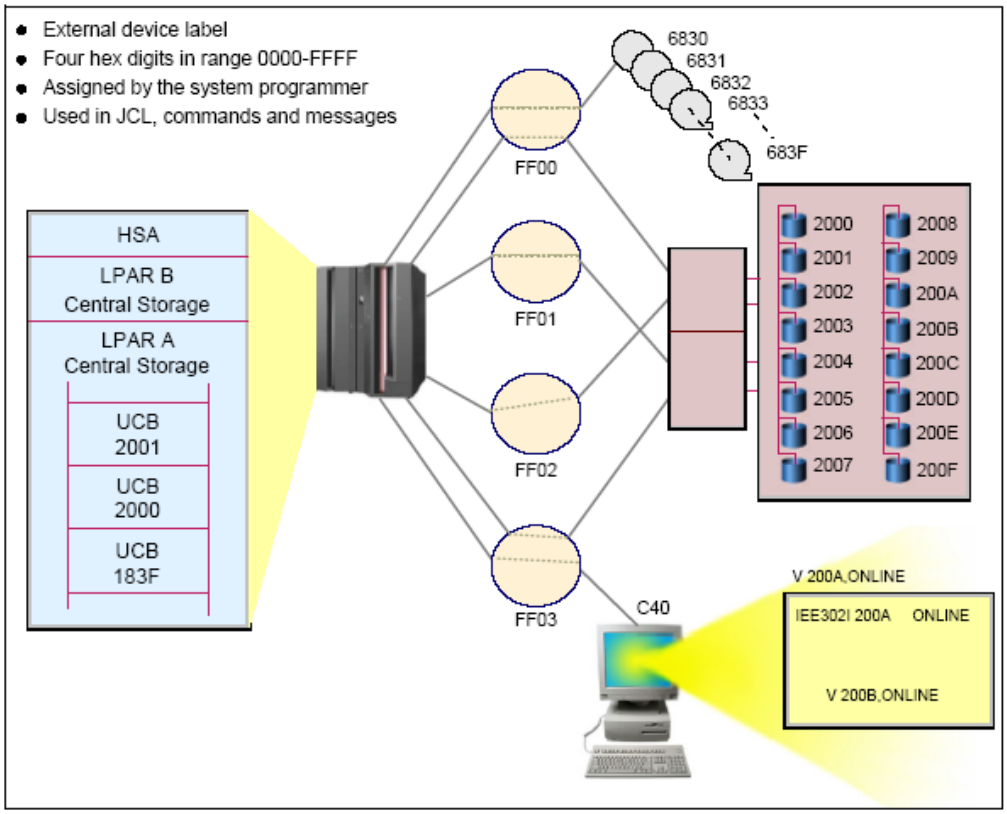


图2-5 设备寻址

2.3.2 系统控制和分区

42

根据我们强调的重点不同，有很多不同的方式来阐明主机内部结构。43页的图2-6高度概念化地表现了现有主机内部的系统控制的几项功能。内部控制器也是微处理器，不过在结构和指令组上比zSeries处理器都要简单。为了不与zSeries处理器混淆，我们称之为控制器。

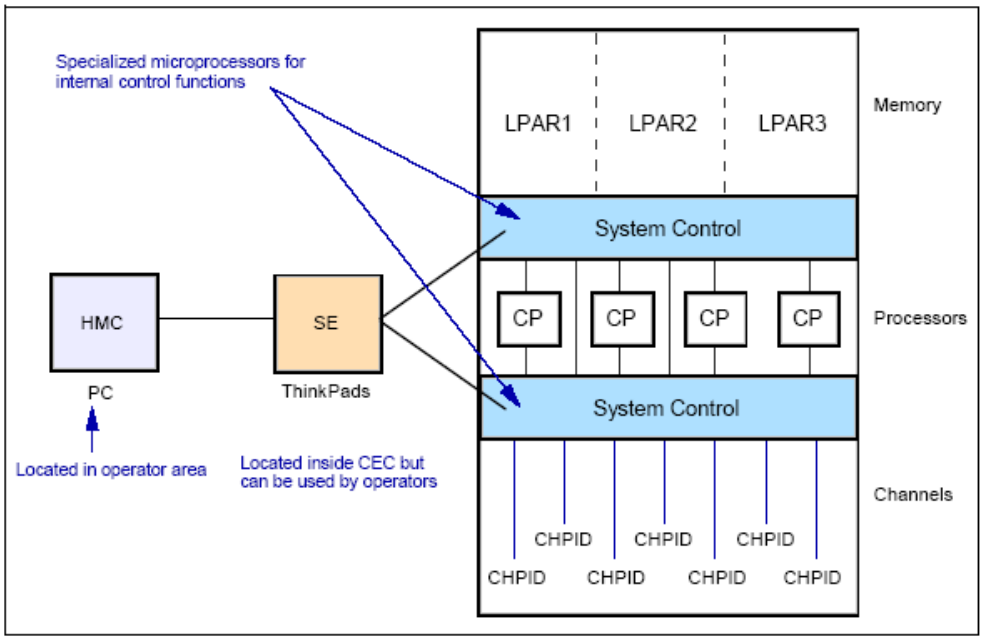


图2-6 系统控制和分区

逻辑分区
处理器硬件的一个子集，用来支持一个操作系统。

在系统控制功能中有个功能是将系统分区为多个逻辑分区(logical partitions ，LPARs)。一个LPAR是处理器硬件的一个子集，用来支持一个操作系统。一个LPAR包含诸多资源(处理器，内存，输入输出设备)并像一个独立系统一般操作。多个逻辑分区可以在一个主机硬件系统上共存。

很多年来，一台主机限制只能有15个LPAR；最近，机器允许有30个LPAR(可能会更多)。实际情况中，内存大小，I/O可用性和处理器强度的有限都会限制LPAR的数量，通常不会超过最大值。

注意：提供分区的硬件和固件叫做PR/SM™(处理器资源/系统管理器 Processor Resource/System Manager)。PR/SM负责创建和运行LPAR。PR/SM(内嵌设备)和LPAR(使用PR/SM的结果)的区别常被忽略，术语LPAR被拿来指代PR/SM设备和它的结果。

系统管理员给每个LPAR分配一部分内存；各个LPAR之间无法共享内存。管理员可以将处理器(即图2-6中的CP)分配给特定的LPAR，或者使用一个内部负载平衡算法分配一个或全部处理器给所有的LPAR。根据每条通道上的设备分类，通道(CHPID)可以被分配给某些特定的LPAR或可以被多个LPAR共享。

一个仅有一个处理器(CP处理器)的系统可以有多个LPAR。PR/SM有一个内部分配器可以给每个LPAR分配单个处理器的一部分，就好象一个操作系统分配器分配处理器时间片给系统中的每个进程，线程或任务。

HMC
监视和控制诸如主机微处理器之类硬件的控制台。

分区控制规格一部分包含于IOCDS中，一部分存在系统描述文件(profile)中。IOCDS和描述文件都存放于支撑元件(SE，Support Element)上，SE仅仅是系统里面的一台笔记本。SE可以连接一个或多个硬件管理控制台(HMC)，HMC是用来监视和控制诸如主机微处理器之类硬件的台式个人计算机。HMC比SE使用起来更

方便，且HMC可以控制多台主机。

操作员通过HMC工作(或在某些特殊情况下通过SE工作)，需要选择和载入一个描述文件和一个IOCDS来准备启动主机。这些文件用来创建LPAR，提供设备号，LPAR分配情况和多路径信息等等来配置通道。我们称这个过程为上电复位(Power-on Reset (POR))。通过载入不同的描述文件和IOCDS，操作员就可以完全改变LPAR的数量和特性以及I/O配置。然而，这样做通常会破坏正在运行的操作系统和应用程序，因此，如果没有事先规划，是很少会这样操作的。

2.3.3 LPAR 特性

实际上，LPAR等同于单独的主机。每个LPAR运行它自己的操作系统。它可以是任意的主机操作系统；没有必要在每个LPAR上运行z/OS。系统安装计划人员可以选择在多个LPAR之间共享I/O设备，不过这是一个本地共享。

系统管理员可以让一个LPAR专用一个或多个系统处理器。交替地，管理员也可以将所有的处理器都分配给一些或所有的LPAR。这里，系统控制功能(通常叫做微码和固件)提供一个分配器来在指定的LPAR之间共享处理器。管理员可以指定在每个LPAR上可用的最大并发处理器数目；也可以指定对于不同LPAR处理器使用的权重。比如，指定LPAR1占用处理器的时间是LPAR2的两倍。

44

每个LPAR上的操作系统是分开启动(IPL)的，他们拥有自己的操作系统拷贝[3]和操作控制台(如果需要)等等。如果一个LPAR上的系统崩溃了，它不会影响到其他的LPAR。

举例来说，在43页的图2-6中我们可能在LPAR1上安装生产系统z/OS，在LPAR2上安装测试版本z/OS，在LPAR3上安装Linux for S/390。如果我们系统一共有8GB内存，那我们可以分配给LPAR1 4GB内存，分配给LPAR2 1GB内存，分配给LPAR3 1GB内存，剩下的2GB内存留作他用。两个安装了z/OS操作系统的LPAR的控制台可能在完全不同的地方[4]。

对于大部分的实用目的而言，3台都运行z/OS的分开的主机(他们之间共享大多数的I/O配置)和一台主机上的3个LPAR，当在处理相同事件时，两者没有区别。多数情况下，z/OS，操作员和程序都不会察觉到这种不同。

他们之间细微的不同在于z/OS(在LPAR定义时如果允许)在整个主机系统中可以获取性能和资源利用率信息，并能动态转移资源(处理器和通道)来提升系统性能。

2.3.4 主机整合

今天正在使用的主机数量比15或20年前少了。某些情况下，所有的应用程序都移植到其他类型的系统上了。然而，大多数情况下，主机数量的减少是因为系统整

[3] 大多数的，并非全部，z/OS 系统库可以被共享。
[4] Linux 没有 z/OS 控制台意义上的操作控制台。

合。那就是说，一些小规模的主机被数量较少的大规模的主机所替代。

对于整合来说，有个很有力的理由。主机软件(从很多厂商提供)价格昂贵，常常超过主机硬件的价格。如果将多张软件许可证(在小型机器上的)换成1张或2张软件许可证(在大型机器上的)，那势必将有所节约(有时会便宜很多)。软件许可证售价通常和系统的处理能力相关，不过价格曲线还是显示：小数量的大型机器价格更优惠。

主机上软件许可证的价格已经成为主机产业增长和走向的一个决定因素。有几个因素令主机软件定价特别困难。我们必须记住主机软件并不像PC软件一样有很大的市场。近年来主机处理能力的增长已经趋于指数级，而非线性。

45

相对来说，运行传统的主机程序(如COBOL写的批处理作业)比运行新的程序(用C或Java编写的，带有GUI接口)需要的处理能力要弱很多。整合已经使得主机变得很强劲。只需要使用1%的处理能力就能运行老的应用程序，不过应用程序厂商总是按照机器的总体性能定价，甚至对于老的应用程序。

这就造成了一个奇怪的情形，客户想要最新型的主机(来获得新功能；相较于型号老的机器，还能降低维护成本)，不过同时他们也想要最慢的主机来运行应用程序(降低基于整体系统处理器性能的软件开销)。

2.4 处理单元

z/Architecture 主机和外围设备的一种 IBM 体系架构。被 zSeries 家族服务器使用。

36页的图2-1列出了系统的几种不同类型的处理器。这些都是z/Architecture处理器，它们的用途稍有不同[5]。一些用途与软件成本控制有关，另外一些则更加基础。

所有的处理器都是从处理器单元[6](PU)或引擎开始的。PU是一个没有特殊使用特性的处理器。每个处理器都是由PU开始的，在安装或之后的时间里，IBM赋予它一些特性。

潜在的特性有：

- 中央处理器(CP)

  这是一个常规的操作系统和应用程序软件可用的处理器。

- 系统协处理器(SAP)

  每个现代主机都至少有一个SAP；较大的系统可能有几个SAP。SAP执行内部代码[7]来提供I/O子系统。例如，SAP解析设备号和CHPID的真实地址，控制单元地址和设备号。SAP管理通往控制单元的多条路径，遇到临时错误时完成错误恢复。操作系统和应用程序无法察觉SAP，并且SAP不使用任何‘常规’内存。

46

- IFL(Integrated Facility for Linux)

[5] 不要将这些与控制微处理器混淆。本节讨论的处理器是完全的、标准的主机处理器。
[6] 本讨论对写书时的 zSeries 机器都适用。较期的系统没有这么多处理器特性，更早的系统并不使用这些技术。
[7] IBM 指的是许可内部代码(LIC)。这经常叫做微码(从技术上严格说不准确)或固件。这肯定不是用户代码。

这是一个常规处理器，只不过将1至2条仅供z/OS使用的指令禁用了。Linux并不使用这些指令，IFL和CP都可以运行Linux。不同之处就在于IFL不计入系统的模型数目[8]中。软件成本将因此而大有不同。

- zAAP

  这是一个禁用了若干指令(中断处理等一些指令)的处理器，所以不能运行一个完整的操作系统。然而，z/OS可以检测到zAAP处理器的存在并使用它执行Java代码(未来还可能执行其他类似代码)。同样的Java代码可以在标准CP上执行。此外，zAAP引擎并不计入系统的模型数目。与IFL一样，zAAP仅仅为了控制软件成本而存在。

- zIIP

  z9系统集成信息处理器(zIIP)是一个处理符合一定条件的数据库工作负载的专用引擎。zIIP选择运行主机上的工作负载，比如商业智能(BI)，企业资源计划(ERP)，客户关系管理(CRM)，来降低软件成本。zIIP直接地、更具成本效益地访问DB2，降低对多个数据拷贝的需性，以此来加固主机作为公司数据中心的地位。

- 集成耦合设施(ICF)

  这些处理器只运行得到许可的内部代码。它们对常规操作系统或应用程序透明。集成耦合设施其实是一个巨大的内存便笺本，多系统使用它来协调工作。ICF必须分配给LPAR，该LPAR之后会成为耦合设施(Coupling Facility)。

- 备用处理器

  没有被个性化的PU用于‘备用’。如果系统控制器检测到损坏的CP或SAP，它就会用备用PU替换。在大多数情况下，替换不需要任何系统中断，即便是原先运行在损坏处理器上的应用程序也不受影响。

- 存在各种形式的按需扩容和类似的机制，是因为客户在某些时候(比如发生无法预测的峰值负荷)可以开启一些额外的CP。

除了这些处理器的特性之外，一些主机的型号或版本被配置成低速运转，速度低于他们CP的潜在速度。这是众所周知的低速运转，IBM称为性能设定(capacity setting)。低速运转是通过使用微码向处理器指令流插入空周期来完成。它的目的同样是控制软件成本，使用主机的最小型号或版本，来达到应用程序的要求。由于IFL，SAP，zAAP和ICF这些处理器不计入软件价格计算，它们总是在处理器的全速运转下发挥作用[9]。

47

2.5 多处理器

所有先前的讨论和例子基于一个假设：一个系统(或者在一个LPAR上)上有多个处理器(CP)。购买一个只有一个处理器(CP)的当前的主机也是可能的，但是这样的主机[10]并不常见。‘多处理器’这个术语是指几个处理器(CP处理器)并且意味着多个

[8] 一些系统没有不同的模型；此种情况下，使用容量模型数目。

[9] 对于 IBM 软件来说是真的，但对于所有软件厂商来说未必如此。

[10] 所有现今的 IBM 主机都需要至少一个 SAP，所以最小的系统也有两个处理器：一个 CP 和一个 SAP。然而，文字上使用的“处理器”通常指的是一个处理应用程序的 CP 处理器。当我们使用处理器一词，但是不是指 CP

多处理器
一个 CPC 可以物理上划分为两个操作处理器集群。

处理器(CP)被一个z/OS拷贝使用。

现今所有的操作系统，从PC到主机，都可以在多处理器环境下工作。但是多个处理器之间的整合程度却相差甚多。比如，任一系统上(或LPAR)的处理器都能接受这个系统中(或LPAR)的未决的(pending)中断。任一处理器可以启动并管理任一通道或设备的I/O操作，只要这些通道或设备对处理器所在的系统或LPAR是可用的。通道，I/O设备，中断和内存都是属于整个系统的(或LPAR)而不是属于某个特定的处理器。

表面上多处理器整合看似简单，但是实现起来却相当复杂。最大化性能很重要；任一处理器都能接受发送给系统(或LPAR)的任何中断显得尤为重要。

系统(或LPAR)上的每个处理器都有一块小的私有内存空间(从物理地址0开始的8KB，总是映射到虚拟地址0)，这对于每个处理器来说都是唯一的。其中是用来处理中断和错误的前缀存储区(PSA)。处理器可以通过特殊的编程访问其他处理器的PSA，也可以使用特别指令(SIGP，代表信号处理器)来中断其他处理器，这些都只有在错误恢复时才被使用。

48

2.6 磁盘设备

IBM 3390磁盘驱动器在现今主机上使用广泛。概念上说，这是一个简单排列，如图2-7所示。

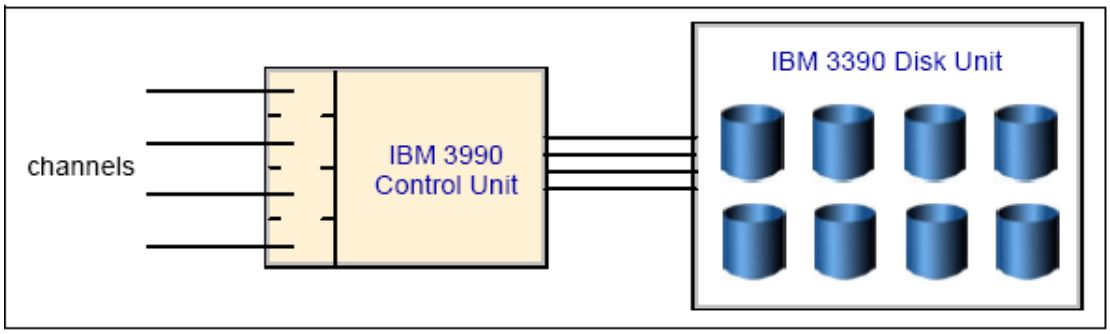


图2-7 最初的IBM 3390磁盘实现

连接的控制单元(3990)通常有4个通道连接到一个或多个处理器上(可能通过一个交换机)，3390单元一般连接超过8个磁盘驱动器。每个磁盘驱动器都具有之前提到的特性。该图图示了3990和3390单元，也表明了现今设备的架构概念。

现今与之等同的设备是一台IBM 2105企业存储服务器，简单用图2-8表示。

49

时，我们会清楚说明这点。

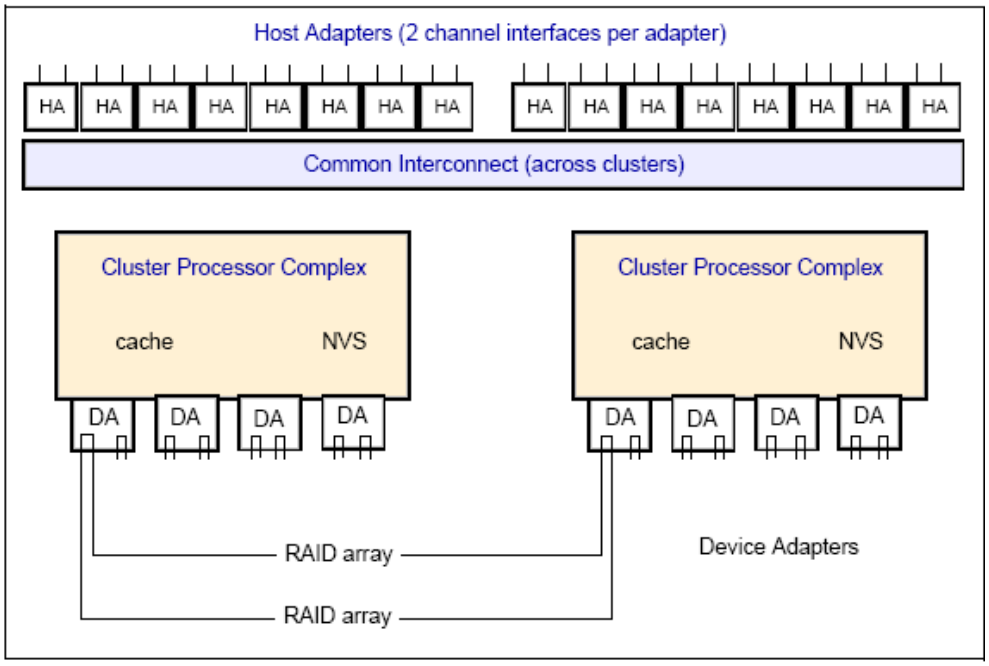


图2-8 现有3390实现

2105单元是一个非常复杂的设备。它仿效大量控制单元和3390磁盘驱动器。它包含高达11TB的磁盘空间，有多达32个通道接口，16GB高速缓冲处理器和284MB非易失性内存(用于写队列)。主机适配器可以看作控制单元接口，它可以连接多达32条通道(ESCON或FICON)。

物理磁盘驱动器是常见的SCSI类型单元(虽然串口SSA用来提供对磁盘更快和冗余的访问)。若干内部排列都是可能的，但最常见的是带有热备件(hot spare)的RAID 5矩阵。实际上，单元中的每样东西都有一个备用或低效运行的单元。内部处理(仿效3990控制单元和3390磁盘)是由在2个处理器复合体中的4个高端RISC处理器提供的，每个复合体都可以操作整个系统。内部电池用于在短暂断电时保存临时数据。单独的控制台用于配置和管理该单元。

2105提供很多3390单元不提供的功能，包括快闪拷贝(FlashCopy)，扩展远程拷贝，并发拷贝，并行访问卷(PAV)，多应用(Multiple Allegiance)，超大缓冲等等。

50

一个简单的3390磁盘驱动器(和控制单元)与刚提到的2105不同，运用了不同的技术。但是从软件看来，它们基本架构是相同的。这就允许为3390磁盘驱动器设计的应用程序和系统软件无需修正[11]就能使用更新的技术。

实现3390磁盘驱动器的新技术分为几个阶段；2105是最新的。使用新的不同技术实施一个架构标准(该处指3390磁盘驱动器和相关控制单元)的时候，保持软件的兼容性，这是主机发展的一个特性。这点已经提及过多次，在长期科技更新的过程中保持应用程序的兼容性是主机的一个重要特性。

[11] 我们需要软件增强版本来使用一些新功能，但是这些是操作系统级别的兼容扩展，不会影响应用程序。

2.7 集群

虽然‘集群’这个术语很少使用，但早在S/360时代，集群就已经开始在主机上运用。集群技术就和共享DASD配置一样简单，需要人为控制或人为规划来防止不必要的数据重合。

这些年来新的集群技术不断被引入。以下章节，我们将讨论3种级别的集群：基本DASD共享，CTC环和并行系统综合体(Parallel Sysplex)。大多数今天使用的z/OS系统使用这些集群级别的一个或多个，单一的z/OS系统相对比较少见。

在讨论中我们使用术语‘镜像’。一个z/OS系统(有一个或多个处理器)是一个z/OS镜像。一个z/OS镜像可以存在于一个S/390或zSeries服务器(有多个LPAR)上，或存在于一个LPAR上，或在z/VM(28页的1.10节”z/OS和其他主机操作系统”中提及到的虚拟化管理程序操作系统)下运行。一个有6个LPAR的系统——每个LPAR上运行自己的z/OS系统——就有6个z/OS镜像。我们使用术语镜像来表明我们不在乎z/OS系统运行在哪里(基础系统，LPAR，z/VM)。

2.7.1 基本 DASD 共享

基本DASD共享环境由图2-9表示。图显示了z/OS镜像，但是它们也可以是操作系统的更早期的版本。这可能是在同一个系统中的2个LPAR或者2个单独的系统；这2种情况在概念上或操作上绝对没有任何差异。

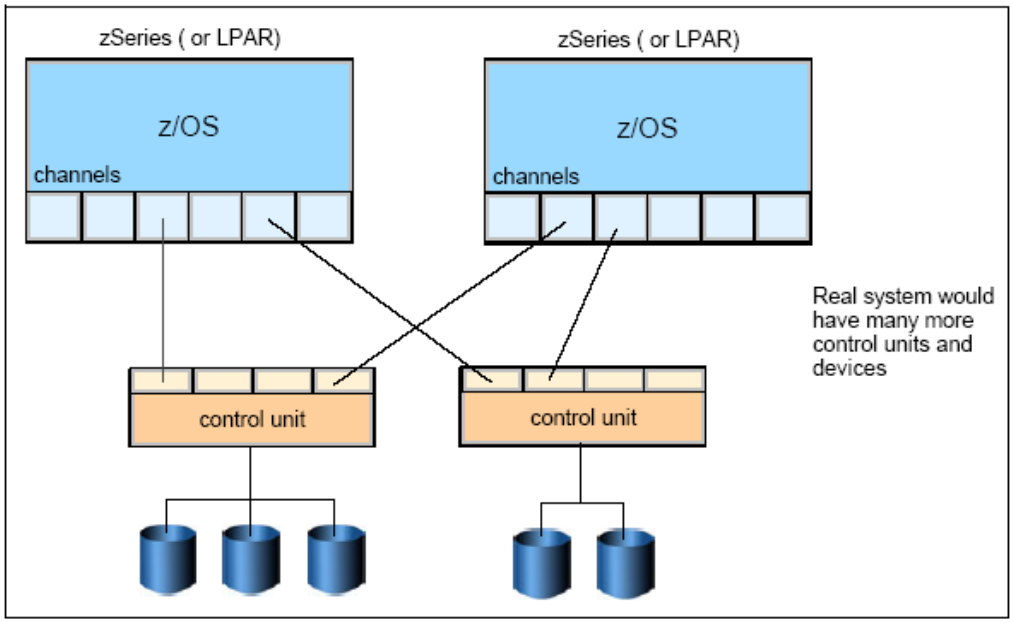


图2-9 基本DASD共享

基本DASD共享系统的能力是有限的。在更新卷内容表(VTOC)或目录(Catalog)前操作系统自动对DASD发出RESERVE和RELEASE命令。(在165页第5章”数据集

操作”，VTOC和目录都包含DASD的元数据，指示数据集存放位置。)RESERVE命令限制对整个DASD的访问，对整个系统发出一条RESERVE命令将一直有效直到发出RELEASE命令为止。这些命令在有限时段有效(比如更新元数据)。应用程序也可以发出RESERVE/RELEASE命令，在程序执行期间保护它们的数据集。系统并不会自动做这样的操作，事实上人为也很少如此操作因为这可能在很长时间内锁住其他系统，禁止它们访问DASD。

基本DASD共享系统主要运用在：操作人员控制哪个作业要运行在哪个系统上，以此来避免诸如两个系统在同一时间更新同一数据这样的冲突。尽管存在局限性，基本DASD共享环境对于测试，恢复，和负载平衡都是非常有用的。

其他类型的设备或控制单元可以连接到两个系统。比如，一个磁带控制单元，带有多个磁带驱动器，可以连接到两个系统。在这个配置中，操作员可以按照需求给系统分配各自的磁带驱动器。

2.7.2 CTC 环

图2-10表明了集群的下一个级别。这个不仅有之前讨论的DASD共享，在系统之间还有2个通道对通道(channel-to-channel CTC)的连接。我们称这种为CTC环。(当超过2个系统时，环的面貌会愈发明显)。

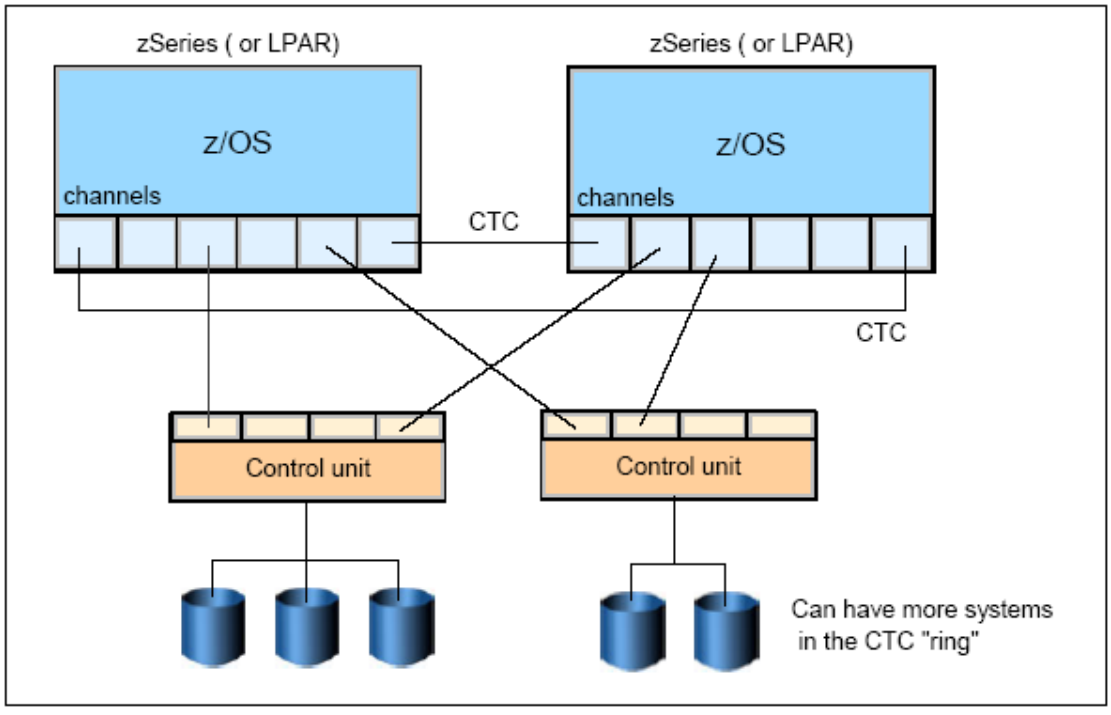


图2-10 基本sysplex

z/OS可以使用CTC环在环中的所有系统之间传递控制信息。可传递的信息包括：

- 磁盘数据集的使用和锁信息。这允许系统自动防止对数据集不需要的重复访

问。锁是在发送给系统的JCL作业中指定的，这点将在第6章201页"使用JCL和SDSF"中说明。

- 作业队列信息，例如环中的所有系统可以从某一单独输入队列中接收作业。同样地，所有系统可以向某一单独的输出队列发送打印输出。
- 安全控制，它允许所有系统都有同样的安全决策。
- 磁盘元数据控制。这样，RESERVE和RELEASE磁盘命令就不必要了。

很大程度上说，在这个配置中，任何一个系统上都可以运行批处理作业和交互用户，因为磁盘数据集可以从任何一个z/OS镜像中访问。并且，作业(和交互用户)可以分配给当时负载最轻的系统运行。

53

当CTC配置被最初使用时，共享的基本控制信息是锁信息。我们会在115页的"串行化使用资源"中讨论这一点，完成该任务的z/OS组件叫全局资源串行化功能，这种配置叫GRS环。GRS环的主要限制是在环中发送信息时的反应时间。

环技术被使用之前，主机使用了一个不同的CTC配置。在这个配置中，需要每两个系统中之间都有2个CTC连接。当多于2个或3个系统加入时，这就变得非常复杂，需要很多通道。

CTC 连接
直接或通过交换器连接同一处理器或不同处理器上的两个 CHPID。

早期的CTC配置(每个系统对每个系统配置，或环配置)之后发展为基本的系统综合体配置。这包括在共享DASD上的控制数据集。这些可以用来保证所有系统的一致操作规范，并在系统重启时保留信息。

由共享的DASD，CTC连接，共享作业队列组成的配置被称为松散耦合系统(loosely coupled systems)。(多处理器，即多个处理器被同一个操作系统所使用，有时相比之下称为紧密耦合系统(tightly coupled systems)，但是这个术语很少用，我们也称它为对称多处理器(SMP)，SMP这个术语在RISC系统中很常见，但在主机上并不使用。

2.8 什么是并行系统综合体(Parallel Sysplex)？

并行系统综合体
使用一个或多个耦合设施的系统综合体。

系统综合体(Sysplex)是多个z/OS系统一起协作，使用一些硬件和软件产品来处理工作。这是一种能提供近乎连续可用性的集群技术。

传统的大型计算机系统也用硬件和软件产品来协调共同处理工作。它和系统综合体主要的区别就在于系统综合体具备的改进的增长潜力和提高的可用性级别。Sysplex增加了处理单元和可协作的z/OS操作系统的数量，这样就增加了可以处理的工作量。为了推动这种协作，新产品得以开发，老产品进行了提升。

并行系统综合体是一个使用多系统数据共享技术的系统综合体。它允许配置中的所有处理节点(或服务器)直接，并发地读/写共享数据，而不影响性能或数据完整性。

54

通过硬件辅助集群范围的串行化和一致性控制，每个节点都可以并发地将共享数据缓存在本地处理器内存中。这样的一个结果就是，单一工作负载的处理请求，诸如商务交易或数据库查询，可以基于每个节点可用的处理器能力，动态被分发到系统综合体集群上的节点上并发执行。

图2-11显示一个并行系统综合体的可见部分，即其硬件。这些是并行系统综合体的关键部分，在硬件架构上实现。

耦合设施
一个特殊的逻辑分区，在系统综合体中提供高速缓冲，列表处理和锁功能。

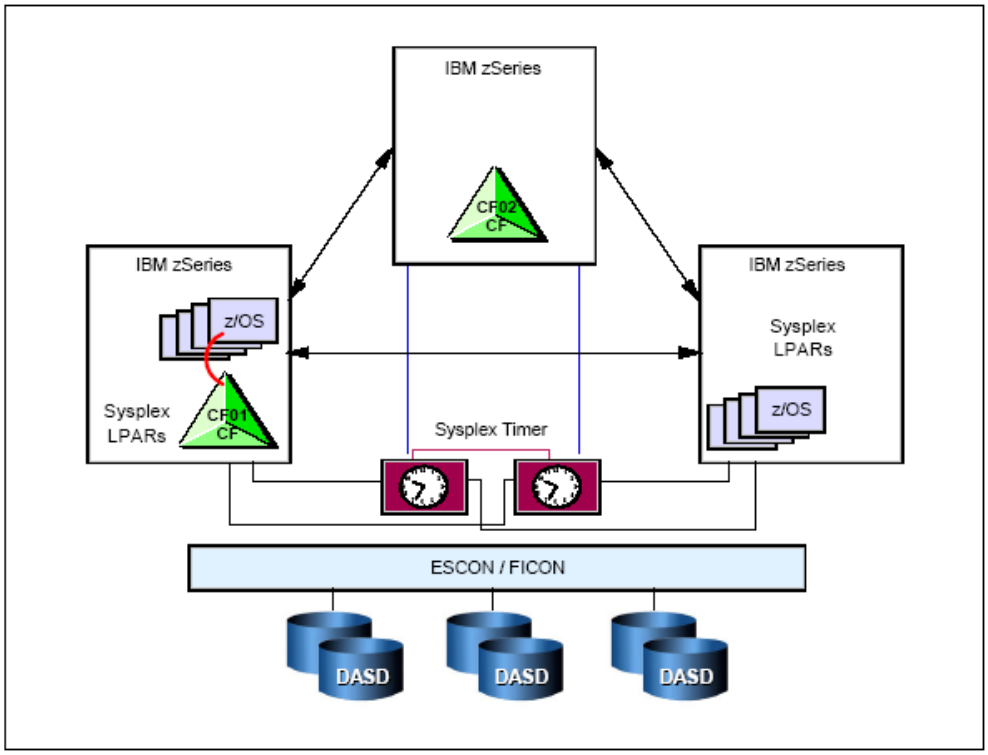


图2-11 系统综合体硬件总览

2.8.1 什么是耦合设施？

并行系统综合体依靠一个或多个耦合设施(Coupling Facilitie，CF)。一个耦合设施是一个主机处理器，带有内存、特殊通道和一个内嵌的操作系统。它有特殊通道，却没有I/O设备，操作系统也很小[12]。

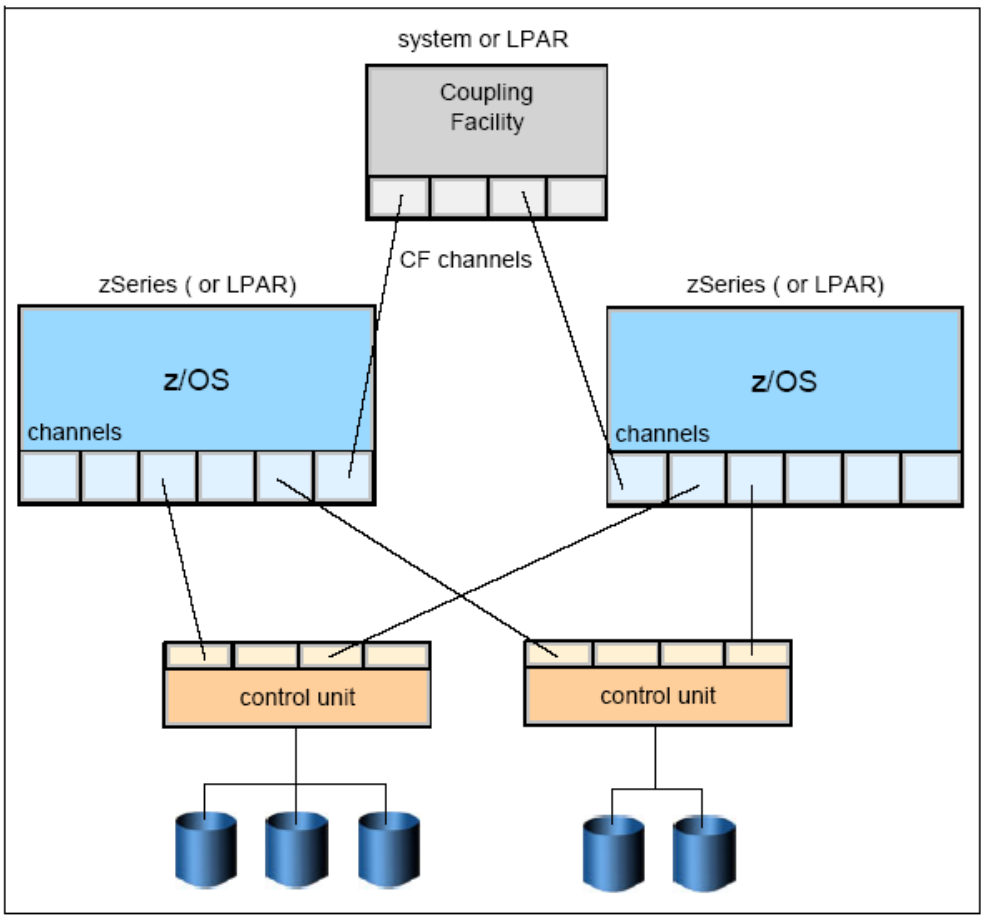
CF的功能和一个快速的便笺本很相似，它的用途有三：

- 对所有连接系统之间共享的信息进行加锁
- 对所有连接系统之间共享的信息(比如一个数据库)进行缓存
- 所有连接系统之间共享的数据列表信息

CF的信息存放在内存中，CF通常有很大的内存。CF可以是一个独立的系统，也可以是一个LPAR。

图2-12 展示了一个小型的拥有2个z/OS镜像的并行系统综合体。此外，这整个配置可以部署在一个系统的3个LPAR上，也可以在3个独立系统上，或者是以上两者的组合。

[12] CF 操作系统一点也不像 z/OS，它没有直接的用户接口。



56 *图2-12 并行系统综合体*

在许多方面，并行系统综合体看上去就像一个单独的大型系统。它有一个单独的操作员界面(控制所有系统)。如果规划合适，操作得当(两者都很重要)，那并行系统综合体中的任何一个或所有系统都可以共享处理复杂的工作负载，很多工作负载的恢复操作(由并行系统综合体中的另外一个系统执行)也能自动完成。

2.8.2 主机集群技术

并行系统综合体技术帮助确保现今大型系统环境的持续可用性。并行系统综合体可以连接多达32台服务器，具有近乎线性扩展性，创造出了强大的商业处理集群系统。并行系统综合体中的每一个服务器经过配置后，都可以共享数据资源，一个应用程序的复制实例也可以在每台服务器上运行。

即便遇到巨大变化，并行系统综合体的设计特点仍能帮助公司持续运转。系统综合体站点可以动态地增加和改变系统综合体中的系统，并可以将系统配置成为没有单点故障。

通过这种最新的集群技术，多个z/OS系统可以一起和谐工作，更有效地处理大规模商业负载。

共享数据集群

并行系统综合体将IBM主机的优势继续发，它可以连接多达32台服务器，具有近乎线性扩展性，创造出强大的商业处理集群系统。在并行系统综合体中每一个服务器都可以访问所有的数据资源，每个"复制"的应用程序都可以在每台服务器上运行。使用主机耦合技术，并行系统综合体提供一种'共享数据'的集群技术，它允许多系统数据共享的同时，保证高性能和读/写完整性。

这里的'共享数据'(和'毫无共享'相反)方法使得工作负载在并行系统综合体集群的服务器中得到动态均衡分配。这也让一些关键的商务应用程序可以利用多服务器的综合能力来帮助保证系统在峰值处理阶段具有最大的系统吞吐量和最好的性能。如果发生硬件或软件中止工作，无论是否是计划中的，工作负载都可以被动态重定向到可用的服务器上，这样也就提供了近乎不间断的应用程序可用性。

57

有序维护

并行系统综合体技术的另一个独特的优势在于它可以有序维护和安装硬软件。

通过共享数据和动态管理工作负载，服务器可以动态地从集群中删除或添加到集群中，这就可以在其余系统继续处理工作的同时对服务器进行维护和安装工作。而且，遵循IBM软硬件共存策略，每次只有一个系统进行软硬件升级。这允许客户可以选择一个适合于他们自己的商业交易的步调来'滚动式'改变系统。

这种有序地滚动式维护软硬件的功能允许公司既可以执行关键商务功能，也可以应对快速增长, 而无需担心影响客户可用性。

2.9 典型的主机系统

本节我们略述主机配置3种不同级别的常规配置。这些不是详细描述，而只是简单的概述而已。

2.9.1 小型系统

58

59页的图2-13有2个例子，表明主机不仅仅是一种独特的硬件，更是一种计算形式。图中展示了2个不同的系统，在普遍接受的意义上来说，他们都没有使用主机硬件。

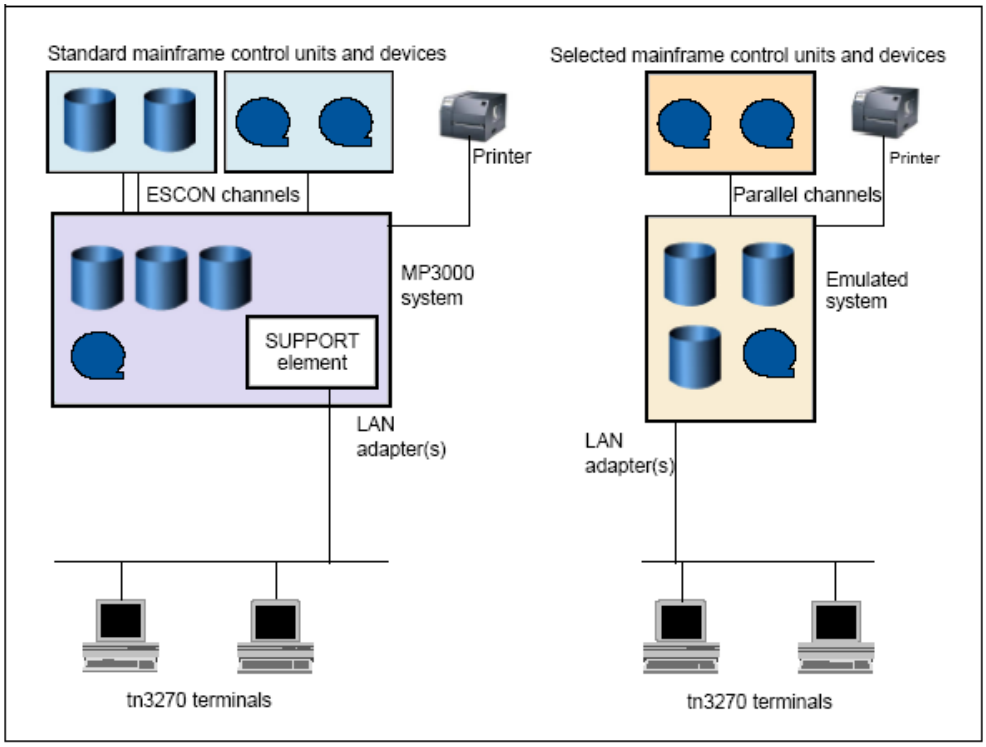


图2-13 小型主机配置

图中的第一个系统是IBM Multiprise® 3000系统(MP3000)，在撰写本书时已经退出市场。这是近年来生产的最小的S/390系统。MP3000有1个或2个S/390处理器和1个SAP处理器。它也有内部磁盘驱动器，配置后可以和一般IBM 3390磁盘驱动器一样操作。它一般使用一个微型内部磁带驱动器来安装软件。MP3000可以具备大量的ESCON或并行通道来连接传统外部I/O设备。

MP3000和S/390主机完全兼容，但是缺少后来的zSeries特性。它可以运行z/OS的早期版本和操作系统的先前版本。一般它使用z/VM或z/VSE操作系统。(在28页1.10节"z/OS和其他主机操作系统"中简单介绍过)

所展示的第二个系统模拟zSeries系统，没有主机硬件。它基于一台个人计算机(运行Linux或UNIX)并使用软件来模拟z/OS。使用特殊的PCI通道适配器可以连接选中的主机I/O设备。运行模拟z/OS的个人计算机可以有很大的内部磁盘(一般是RAID阵列)，用它来模拟IBM 3390磁盘驱动器。

这两个系统都缺乏真实主机的一些特性。但是，它们都有能力确保质量地完成工作。一般的应用程序软件无法辨别出这些系统和真正的主机的区别。事实上，人们认为这些系统就是主机，因为它们的操作系统，中间件，应用程序和使用风格都和主机一样。MP3000可以配置为多个LPAR，同时运行测试和生产系统。模拟的系统不提供LPAR，但是可以通过运行多份模拟软件的拷贝来达到差不多的效果。

这些系统的一个关键吸引人之处就在于它们是"在盒子里的主机"。在很多情况下不需要外部的传统的I/O设备。这很大程度上降低了主机系统的入门价位。

2.9.2 中型单系统

61页上的图2-14显示一个中型的主机系统和它通常需要的外部元素。图中显示的特定系统是IBM z890系统，它有2个新近的外部磁盘控制器，若干磁带驱动器，打印机，LAN附加装置和控制台。

这是一定程度上理性化的配置，没有涉及到陈旧的设备。这里简略描述的系统可能有若干活动的LPAR，比如：

- 一个z/OS生产系统，运行交互式应用程序
- 第二个z/OS生产系统，主要用于批处理应用程序(这些也可以在第一个LPAR上运行，不过一些系统为了管理方面更倾向于使用另一个独立的LPAR。)
- 一个z/OS测试版本，测试新的软件发布版本，新的应用程序等等。
- 一个或多个Linux分区，可能运行网络相关的应用程序。

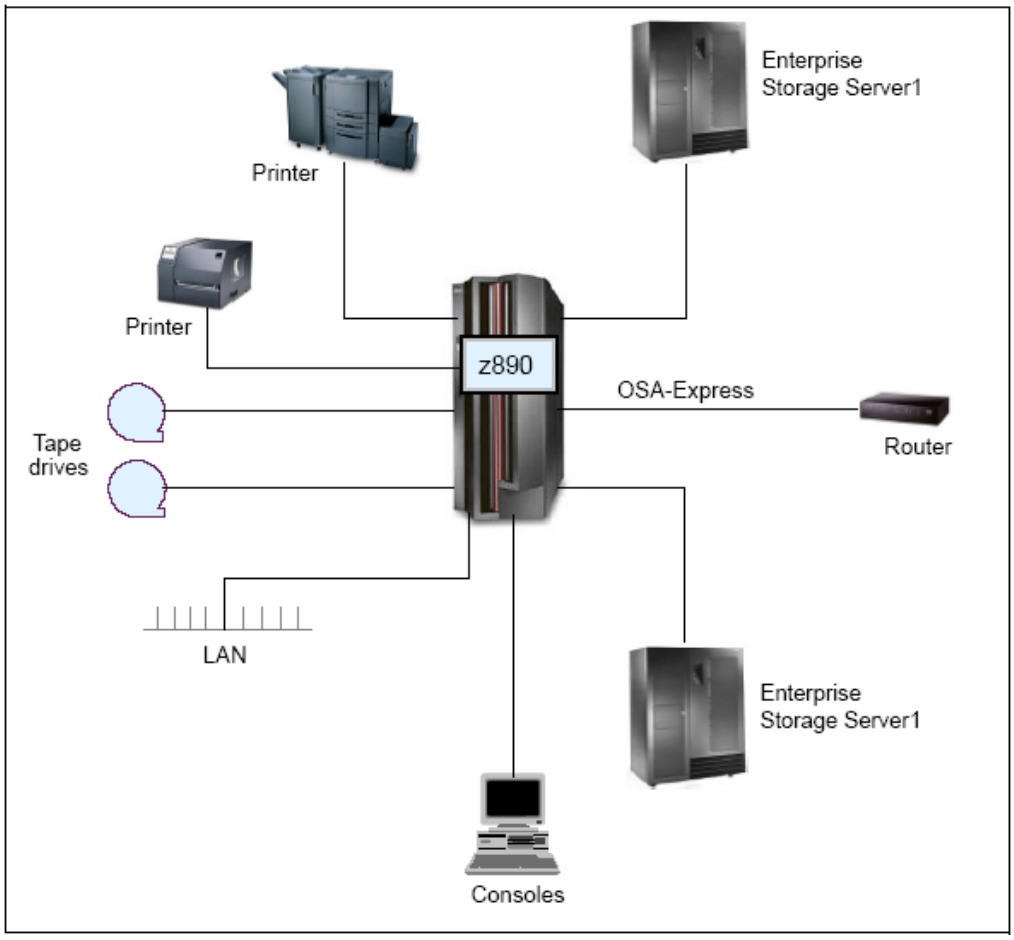


图2-14 中型主机配置

图2-14中的磁盘控制器包含大量的常用驱动器，这些驱动器运行在多种RAID配置下。控制单元将它们的接口转换成和标准IBM 3390磁盘驱动器一样，3390是主机最常见的磁盘。这些磁盘控制单元有多个通道接口，可以全部并行处理。

2.9.3 大型系统

图2-15显示了一台主机，虽然与主机系统相比起来，这仍旧是一个中型配置。该例的典型之处在于本例中，新旧主机均有，并带有通道交换器，允许所有系统访问大多数I/O设备。同样，在本例中也同时存在新旧磁盘控制器(和设备)和磁带控制器(和设备)。整个系统属于一个中型并行系统综合体配置。

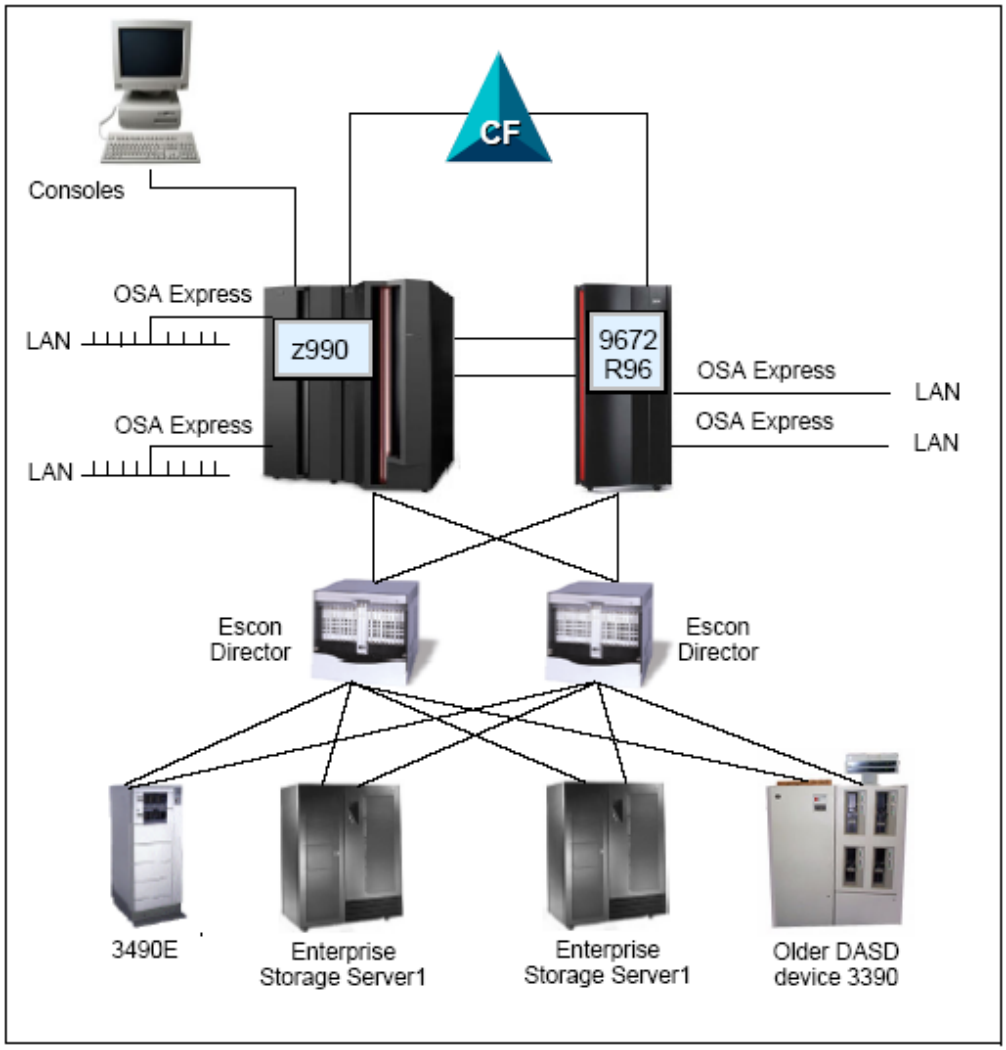


图2-15 较主机配置

简单来说，图2-15中的设备包括：

- 一台IBM 3745，它是一个优化的连接远程终端和控制器以及LAN的通信控制器。一台3745看似主机的控制单元。
- 一台3490E磁带驱动器，虽然有些过时，但是可以处理大多数主机兼容的广泛使用的盘式磁带。
- 一个第6代主机的设计(G6)

- 一台新型z990主机
- 企业存储服务器(ESS)
- ESCON交换器
- OSA快速连接，连接到一些LAN
- CF(图中作为一个独立的盒子显示，但它也可能是主机中的一个LPAR)

62

2.10 主机持续可用性

并行系统综合体技术是一种使能技术，它使高度可靠，冗余，稳健的主机技术可以获取不间断可用性。一个配置适当的并行系统综合体集群可以对用户和应用程序保持可用性，获取最短的宕机时间。比如：

- 为实现并发提供的软硬件组件有利于主机的有序维护，以"容量按需升级"为例，允许通过一次增加一个工作引擎提升处理能力和耦合能力，而不会打乱正在运行的工作负载。
- DASD子系统部署磁盘镜像或RAID技术来帮助保护数据不丢失，开拓技术来实现瞬间备份，而不需要关闭应用程序。
- 网络技术提供以下功能来实现容错网络连接：VTAM®普通资源(VTAM® Generic Resources)， 多节点永久会话(Multi-Node Persistent Sessions)，虚拟IP寻址(Virtual IP Addressing)和系统综合体调度器(Sysplex Distributor)。
- I/O子系统支持多条I/O路径和动态交换，来防止数据访问丢失，提升吞吐量。
- z/OS软件组件允许新的软件版本与这些软件组件的低版本共存，这样有利于进行'滚动式'维护。
- 商务应用程序是"可以数据共享"的，而且在每台服务器上都有复制副本，这样可以在损坏发生时，允许均衡分配工作负载，防止失去应用程序可用性。
- 操作和恢复过程是全自动的，对用户透明的，减小或排除了人工干预的需要。

并行系统综合体是管理这种多系统环境的一种方式，提供以下帮助：

- 64页的"没有单点故障"
- 64页的"容量和扩展性"
- 65页的"动态均衡工作负载"
- 65页的"使用方便"
- 68页的"单系统镜像"
- 69页的"兼容地改变和有序增长"
- 69页的"应用程序兼容"
- 70页的"灾难恢复"

63

在这章的下面章节中我们将继续讨论这些帮助。

2.10.1 没有单点故障

在并行系统综合体集群中，构建一个没有单点故障的并行处理环境是可能的。因为在并行系统综合体中的所有系统都可以对关键应用程序和数据进行并发访问，因此由硬件或软件故障造成的系统失败并不一定会导致丢失应用程序的可用性。

失败子系统的对等实例运行在其他正常系统节点上，它们可以恢复失败实例所持有的资源。或者，利用自动重启功能，失败的子系统可以在仍然正常工作的系统中自动重启，来恢复在子系统失败时正在处理的工作。当失败的子系统实例不可用时，新的工作需求会被重定向至其他集群节点上的可共享数据的子系统实例，在应用程序失败和恢复期间提供持续的应用程序可用性。这就对终端用户屏蔽了计划内外的系统运行中断。

配置的冗余让单点失败次数大大减少。没有并行系统综合体，一台服务器的失效可能严重影响到一个应用程序的性能，在错误被修正之前，也将带来一些系统管理困难，包括重新分配工作负载或重新分配资源。在并行系统综合体环境里，一台服务器的失败对应用程序来说可能是透明的，并且服务器工作负载会在并行系统综合体范围内自动被重新分配，性能降低得很少。因此，在并行系统综合体环境下，那些本可能会严重影响应用程序可用性的事件，诸如中央处理复合系统(CPC)硬件元素损坏或关键操作系统组件的损坏，都会减小影响。

尽管并行系统综合体中的节点一起工作，对外提供一个单独镜像，但它们仍旧为个体系统，这使得系统安装，操作和维护都互不影响。系统程序员可以逐一引入变化，比如软件升级，一次只升级一个系统，这样其他系统就可以继续处理工作。如此以来，主机IT人员就可以按照一个对方便商务的时间表逐一'滚动式'改变系统。

2.10.2 容量和扩展性

并行系统综合体环境可以接近线性地从2台扩展至32台系统。它可以是任何支持并行系统综合体环境的服务器的混合体。该配置的综合能力可以满足现今所知的每个处理需求。

64

2.10.3 动态平衡工作负载

整个并行系统综合体在终端用户和商务应用程序看来就是一个单独的逻辑资源。正如一个单独的SMP服务器中工作可以在各处理器中动态分配，在并行系统综合体中，工作也可以被定向到任何一个有能力的节点上。这就避免了在集群中个体节点间划分数据和应用程序，也避免了在多个服务器上复制数据库。

工作负载平衡允许一个交易在并行系统综合体里运行多种应用程序，且保持对交易重要的响应级别。主机IT主管选择每项工作负载需要的服务水平协议，z/OS的工作负载管理组件(WLM)和诸如CP/SM或IMS的子系统一起，在并行系统综合体的所有资源范围内自动平衡任务，来满足商务需求。这些工作来源可以多种多样，比如批处理，SNA，TCP/IP, DRDA®,或WebSphere MQ。

系统恢复时需要考虑几个因素。第一，当失败发生时，自动跳过它并重新分发工作负载以利用剩余可用资源是非常重要。第二，恢复失败时正在进行的工作单元是很必要。最后，恢复失败单元后，应该尽快尽可能透明地把它重新加入到集群中来，再次开始处理工作。并行系统综合体技术让这一切都成为可能。

工作负载分发

在失败单元被隔离后，有必要有序地将工作负载重定向到并行系统综合体的剩余可用资源。如果并行系统综合体环境中发生失败，在线交易工作负载无须操作员干预会自动被重分配。

通用资源管理

通用资源管理能够指定一个通用网络接口给VTAM。它可以被CICS终端拥有区域(terminal owning regions，TOR)，IMS交易管理器，TSO或DB2 DDF工作使用。例如， 如果一个CICS TOR失败了，只有一部分网络会受影响。受影响的终端能立即重新登录，在连接到另外一个TOR后继续工作。

2.10.4 使用方便

并行系统综合体解决方案满足了大部分客户希望有一天24小时，一周7天不间断可用性的需求，也提供和这种需求一致的简单系统管理技术。并行系统综合体解决方案的一些特征，对增加可用性，减少系统管理任务大有裨益。比如：

- 66页的”工作负载管理(WLM)组件”
- 66页的”系统综合体失败管理器(SFM)”
- 66页的”自动重启管理器(ARM)”
- 67页的”复制和符号参数”
- 67页的”zSeries资源共享”

工作负载管理(WLM)组件

z/OS的WLM组件提供系统综合体范围内的工作负载管理，它的依据是工作负载安装时指定的性能目标及其商务重要程度。WLM试图通过动态资源分配达到性能目标。WLM提供给并行系统综合体何种工作应该以何种优先级去何处执行的智能。优先级基于用户的商务目标而定，并由系统综合体技术管理。

系统综合体失败管理器(SFM)

SFM策略允许系统指定检查失败的时间间隔，和当系统综合体中的一个系统失败时启动的恢复动作。

ARM
系统恢复功能，用来提高批处理作业和开始任务的可用性。

如果没有SFM，当一个并行系统综合体中的系统失败时，会告知操作员，提示他采取一些恢复行为。操作员可以选择将无应答的系统从并行系统综合体中分离出来，或采取某种行动来恢复系统。操作员干涉的期间，可能占用剩余活动系统要求的重要系统资源。SFM允许系统编写一条策略来 定义当检测到特定类型的问题时需启动的恢复动作，比如隔离失败的镜像来防止它访问共享资源，关闭LPAR或获取中央存储或扩展存储，这些动作将会在检测到系统综合体失败时自动执行。

自动重启管理器(ARM)

ARM使得那些在失败时占有重要资源的子系统能够快速恢复。如果在并行系统综合体中的其他子系统实例需要这些重要资源，快速恢复可以让这些资源更快可用。虽然如今也使用自动操作包来重启子系统，解决这类死锁问题，但在失败时ARM能以更快速度激活。

66

ARM减少操作员干预活动，涉及的领域如下：

- 检测重要作业或开始任务的失败
- 在开始任务或作业失败后自动重启

在作业或开始任务异常终止后，作业或开始任务可以在特定情况下重启，比如覆盖之前的JCL或指定作业依赖条件，而无需依赖程序员。

- 在系统失败后，自动将工作重新分配给给其他适当的系统。

这就不再需要人为费时地评估最适合的目标系统来执行重启工作了。

复制和符号参数

复制指的是在并行系统综合体中不同的物理服务器上复制硬件和软件配置。就是说，一个应用程序要利用并行处理，可能要在并行系统综合体的所有镜像上都运行完全相同的实例。在并行系统综合体的所有系统中，支持这些应用程序的软硬件也可以被设定为完全一样，以此来减少定义和支持该环境所需的工作。

‘对称’概念意味着新系统可以被引入；也意味着在失败出现的时候或一个系统计划维护时，可以对工作负载进行自动分发。这也会减少系统程序员建立环境所需的工作量。注意对称并不排除有些系统需要有独特的配置要求，比如不对称的打印机附件和通信控制器，或者不对称的工作负载，这些工作负载并不适合并行环境。

系统符号参数有利于管理”复制”。在启动参数，JCL，系统命令和开始任务中z/OS提供了对替代值的支持。这些值可以在参数或过程中指定，允许使用一个唯一的替代值来动态组成一个资源名。

zSeries资源共享

若干基础z/OS组件发现：IBM耦合设施共享存储提供了一种方法共享组件信息，来实现多系统资源管理。这个发现，称为IBM zSeries资源共享，可以以更低的开销，更快的性能和简单的系统管理来共享诸如文件，磁带驱动器，控制台和目录等物理资源。不要把它和由数据库子系统实现的并行系统综合体数据共享弄混淆了。通过基本z/OS软件栈提供的本地系统发现，zSeries资源共享甚至可以为没有使用数据共享的客户提供立即值。

67

并行系统综合体解决方案的目标之一是通过减少管理，操作和维护一个并行系统综合体的复杂度来提供简化的系统管理，无需增加系统支持人员的数量，无需降低可用性。

2.10.5 单系统镜像

尽管并行系统综合体中可能有多台服务器和多个z/OS镜像以及不同技术的混合利用，但并行系统综合体中多个系统的集合，在操作员，终端用户和数据库管理员等看来，应该仍旧是一个单独实体。从操作和定义的观点看，一个单独系统镜像可以降低复杂性。

无论系统镜像数量和底层硬件的复杂度如何，并行系统综合体解决方案从以下的多个方面提供单个系统镜像：

- 数据访问，允许工作负载动态平衡功能和更高的可用性
- 动态交易路由，提供工作负载动态平衡功能和更高的可用性
- 终端用户接口，允许登录到一个逻辑网络实体
- 操作接口，允许更简单的系统管理

单点控制
一个系统综合体特性；您可以在单个工作站上完成给定的任务集合。

单点控制

我们需要逻辑单点控制并行系统综合体里的所有系统的管理。术语"单点控制"指能访问任意一个任务要求的接口而不需要依靠物理硬件。比如，在一个有很多系统的并行系统综合体里，有必要能够将命令或操作发给其中的任何一个系统，而不需要控台或控制点物理连接到所有的系统。

失败时的稳固单系统镜像

即使并行系统综合体的个体硬件单元或整个系统失效，单系统镜像必须被保持。这就是说，和单点控制概念一样，单系统镜像的存在不依赖于配置中的特定物理单元。从终端用户的角度来看，并行系统综合体环境中应用程序的并行属性必须是透明的。无论哪个物理z/OS镜像支持该应用程序，它都应该是可访问的。

2.10.6 兼容地改变和有序增长

并行系统综合体的主要目标就是保持系统持续可用性。因此，需求之一就是诸如新应用程序，软件或硬件的变化可以有序地被引入，它们可以与现有系统共存。并行系统综合体解决方案的软硬件组件支持兼容变化，允许共存2个级别，分别是级别N和级别N+1。这就表示，比如，任何IBM软件产品都会在确保和先前发布版本相容的情况下做一些变化。

2.10.7 应用程序兼容

并行系统综合体的设计目标之一是：避免因技术变革而产生的应用程序变化。虽然我们需要为得到配置的最大优势而研究(affinities)，但在极大程度上这还是很正确的。

从应用程序的架构角度来看，3点可以决定这个应用程序应该运行在并行系统综合体上：

- 技术优势

  在应用程序在客户商务流程中扮演关键角色的情况下，借助扩展性(甚至有序升级)，可用性和动态工作负载管理，一个架构可以满足客户的需要。有了多系统数据共享技术，并行系统综合体中所有的处理节点都可以完全并发读写访问共享数据，而不会影响数据完整性和系统性能。

- 联合优势

  由于历史的原因，很多应用程序都是基于S/390或z/OS的，因此，z/OS上的新应用程序是有性能和维护优势的，尤其当它们和现有的应用程序有关联时。

- 底层架构优势

  如果已经存在一个并行系统综合体，仅需要极少量的底层架构工作就可以集成一个新的应用系统。很多情况下，系统并不需要集成新的服务器。相反，它可以支持现有底层架构，充分利用现存系统综合体的优势。地理分散的并行系统综合体(GDPS®)连接多个位于不同位置的系统综合体，主机IT人员可以用它创建一个配置用于灾难恢复。

2.10.8 灾难恢复

地理分散的并行系统综合体(GDPS)是主要的保证灾难恢复和持续可用性的解决方案，它主要用于基于主机的多站点公司。GDPS可以自动镜像重要数据，并有效平衡站点间的工作负载。

GDPS
一个在并行系统综合体中增强应用程序可用性和灾难恢复的应用程序。

GDPS也使用自动操作和并行系统综合体技术帮助管理多站点数据库，处理器，网络资源和存储子系统镜像。这项技术提供持续可用性，站点间有效转移工作负载，资源管理和关键业务主机应用程序的快速数据恢复工作。现在GDPS 2个站点间最大距离为100公里(大约62英里)光纤距离，当然还有其他的一些限制。这就提供了一个保证无数据丢失的同步的解决方案。

还存在GDPS/XRC，它可以用在更远的距离间，并且应该提供低于2分钟的恢复点目标。(即恢复或丢失的数据最多不能超过2分钟)

2.11 总结

zAAP
专用处理辅助单元，配置用来在选中的zSeries 机器上运行 Java 程序。

得知一些术语的意义对您理解主机是很重要的，这些术语包括系统，处理器，CP等等。初始的S/360架构基于CPU，内存，通道，控制单元，设备和它们的寻址方式，是理解主机硬件的基础——尽管初始设计的每一个细节都已经发生了各种各样的改变。初始设计中的概念和术语仍然渗透着主机的描述和设计。

实际应用中，将一个大系统分区成多个小系统(LPAR)是现在所有主机系统的核心需求。硬件设计具有灵活性，允许任一处理器(CP)访问和接受来自于连接在该

LPAR上任何通道、控制单元和设备的中断，这也对整个系统的灵活性，可靠性和性能有巨大贡献。一组处理器可以被IBM配置成客户处理器(CP)、I/O处理器(SAP)、Linux专用处理器(IFL)，Java类专用处理器(zAAP)和备用处理器，这些是主机特有的，这也提供了满足客户需求的高度灵活性。一些主机软件的成本结构决定了这其中的某些需求。

70

除了刚刚提到的主要处理器(PU和它的所有特性)，主机还有一个控制器(特殊微处理器)网络，将系统作为一个整体来控制。这些控制器对操作系统和应用程序来说是透明的。

从20世纪70年代早期开始，虽然当时仅安装了一个处理器，但主机已经被设计为多处理器系统。所有的操作系统软件都被设计成多处理器适用的，只有一个处理器的系统被认为是多处理器设计的一个特例。几乎最小的主机系统也在使用集群技术，尽管它们一般不使用"集群"(Cluster)这个术语。

正如之前所述，集群技术可以和DASD共享配置一样简单，需要人为控制或计划来防止不需要的数据重叠。今天常见的配置允许所有系统共享锁和队列控制，这样做的一个好处就是，自动管理对数据集访问，避免不必要的并发使用。

最复杂的集群技术就是并行系统综合体。这项技术允许以近线性扩展性的方式连接32台服务器，创建一个强大的商务处理集群系统。

在并行系统综合体中每一个服务器都可以访问所有的数据资源,每个"复制"的应用程序都可以在每台服务器上运行。当使用了耦合技术时，并行系统综合体可以提供一项"数据共享"集群技术，在确保高性能和读写完整性情况下，允许多系统数据共享。系统综合体的设计特性帮助商务连续运行，即使在发生巨变期间也不例外。系统综合体站点可以动态地增加和改变系统综合体中的系统，并可以将系统配置成为没有单点故障。

通过最新的集群技术，多个z/OS系统可以协同工作，更有效地处理海量商业工作负载。

本章关键术语		
自动重启管理器(Automatic Restart Manager，ARM	中央处理复合系统(central processing complex，CPC)	中央处理器(central processing unit，CPU)
通道路径指示符(channel path identifier，CHPID)	通道对通道连接(channel-to-channel connection，CTC)	耦合设施(coupling facility，CF)
ESCON通道(ESCON channel)	地理分散的并行系统综合体(Geographically Dispersed Parallel Sysplex，GDPS)	硬件管理控制台(hardware management console，HMC)
逻辑分区(logical partition，LPAR)	多处理器(multiprocessor)	并行系统综合体(Parallel Sysplex)
单点控制(single point of control)	z架构(z/Architecture)	zSeries应用程序辅助处理器(zSeries Application Assist Processor，zAAP)

71

2.12 复习题

为了测试您对本章内容的理解情况，完成以下问题：

1. 为什么主机软件定价看起来那么复杂？
2. 为什么IBM的新型主机上有那么多型号(或‘容量设置’)？
3. 为什么处理传统COBOL应用程序与处理新型Java应用程序所需的处理能力不是线性关系？
4. 多处理器(Multiprocessor)是指多个处理器(这些处理器被操作系统和应用程序所使用)。那什么是多道程序设计(multiprogramming)呢？
5. 松散耦合系统和紧密耦合系统有什么区别？
6. 为了在一个LPAR中运行，z/OS应用程序需要改变什么？

2.13 思考题

如果能安排就请访问一台主机系统。在典型系统上寻找新的，旧的和更旧的系统和设备会非常有趣，也能帮助您直观了解持续性对主机用户的重要性。

1. 并行系统综合体从外部来看是一个单镜像，这有什么优势和劣势？
2. 现今的市场为何需要持续可用性？
3. 建立并行系统综合体时，人们怎么去证明冗余硬件成本和软件许可证成本是值得的？

2.14 练习

1. 显示CPU配置
   a. 从ISPF主选项菜单访问SDSF
   b. 在命令输入行区域输入/D M=CPU，然后按回车
   c. 使用SDSF中的ULOG选项查看命令显示结果
2. 显示页数据集的使用情况：
   a. 在命令输入行区域，输入/D ASM，然后按回车
   b. 按PF3键回到之前的屏幕

3

第 3 章 z/OS 概述

目标：作为贵公司主机IT团队的最新成员，您需要了解主机操作系统的基本功能特点。本课程中讲授的操作系统为z/OS，这是一种被广泛使用的主机操作系统。众所周知，它能以一个安全，可靠，便捷的途径，同时为数千名用户提供服务，处理大量工作。

在完成这章后，您将能够：

- 列出几个z/OS操作系统的标志性特征。
- 举例说明z/OS与一个单用户操作系统的区别。
- 列出z/OS上使用的几种主要存储类型。
- 解释虚存的概念以及它在z/OS中的使用。
- 说明页面，页框，页槽之间的关系。
- 列出一些与z/OS一起使用以提供一个完整系统的软件产品。
- 描述z/OS和UNIX操作系统的一些相同点和不同点。

75

3.1 操作系统是什么？

最简单地说，操作系统是那些管理计算机系统内部运作的程序集合。设计操作系统是为了充分利用计算的各种资源，并确保尽可能有效地处理最大数量的工作。虽然操作系统不能提高计算机的速度，但它可以最大限度地使用计算机，通过允许在给定的时间里完成更多的工作，从而使计算机看起来更快了。

计算机的体系结构包括计算机系统提供的各种功能。体系结构有别于物理设计，事实上，不同的机器设计也可能符合同一种计算机体系结构。某种意义上来说，体系结构就是被用户(比如一个系统程序员)看到的计算机。举例来说，体系结构的其中一部份就是计算机能够识别并执行的机器指令的集合。在主机环境中，系统软件和硬件构成一个高度发达的计算机体系结构，这是几十年技术创新的结果。

3.2 z/OS 是什么？

本课程中讨论的操作系统是z/OS[1]，一种广泛使用的主机操作系统。设计z/OS是为了给在主机上运行的应用程序提供一个稳定，安全，可持续可用的环境。

今天的z/OS是几十年技术进步的成果。它从一个曾经只能一次只能处理一个程序的操作系统，演变成能够并行处理数以千计的程序和为数以千计的用户提供交互服务的操作系统。想要知道z/OS如何以及为什么能这样运行，那么就很有必要来了解一些z/OS的基本概念以及它运行的环境。本章介绍一些概念，来帮助您理解z/OS操作系统。

在大多数早期的操作系统中，一次只能向系统发送一项任务请求。操作系统把每个请求或作业作为一个单元，在当前正在处理的单元结束前不会启动下一个作业。当一个作业能够连贯地从头到尾执行的时候，这种分配方式可以工作得很好。但是通常的情况是，一个作业必须等待从诸如磁带驱动或者打印机的设备上读入信息或输出信息到这些设备上。和处理器的速度相比，输入输出(I/O)要花很长的时间。当作业在等待I/O的时候，处理器就闲置了。

当作业在等待的时候，找到一种使处理器持续工作的方法可以在不增加额外的硬件的条件下提高处理器完成的工作总量。z/OS通过把任务分片并将每片分配给独立运行的不同的系统组件和子系统来完成整个任务。在任何时间点，只有一个组件能获得处理器的控制权，使其工作，然后将控制权传递给一个用户程序或者其他组件。

[1] z/OS 是为了利用 IBM z 系列架构的优势而设计，该架构在 2000 年提出。

3.2.1 z/OS 使用的硬件资源

z/OS操作系统在处理器内执行并且在执行过程中驻留在处理器存储器中。z/OS通常被认为是系统软件。

主机硬件有处理器和大量的外围设备如硬盘驱动器(称为直接访问存储设备或DASD)，磁带机和各种类型的用户控制台；如图3-1。磁带和DASD用于系统功能，并供z/OS执行的用户程序使用。

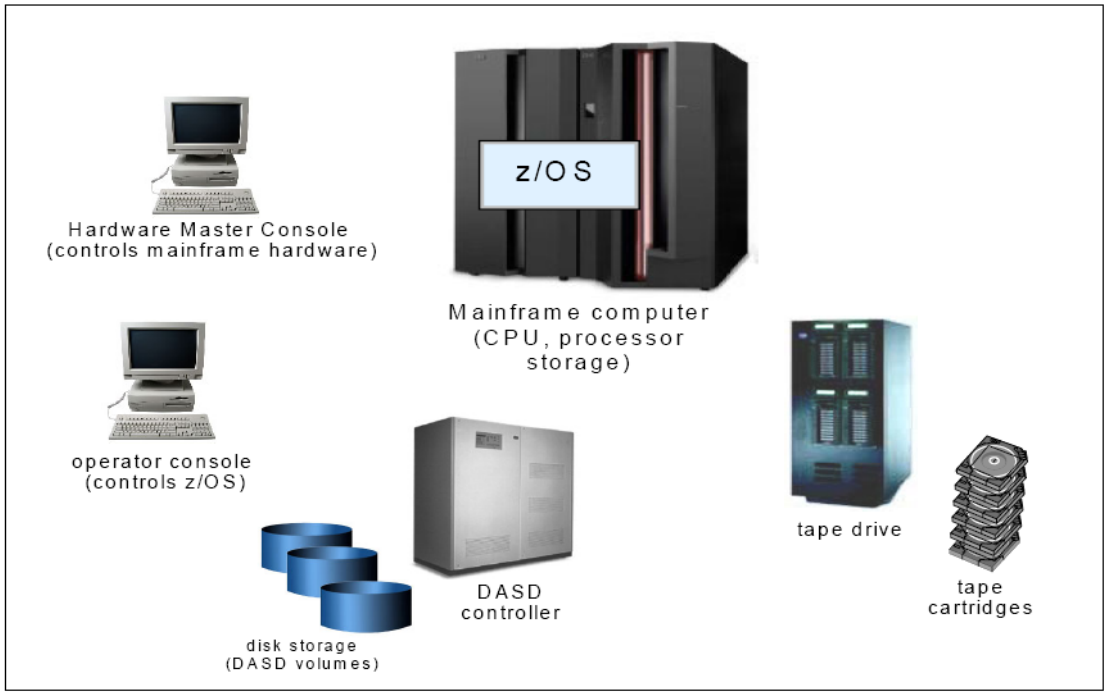


图3-1 z/OS使用的硬件资源

为了完成一份z/OS系统的新订单，IBM通过因特网或者物理磁带(根据用户需要)把系统代码发送给客户。在客户端，一名人员如z/OS系统程序员接收该订单并将新系统拷贝到DASD盘卷上。在系统完成客户化并准备投入运行后，需要使用系统控制台启动和操作z/OS系统。

z/OS操作系统的设计是为了充分利用最新的IBM主机硬件和它许多复杂的外围设备。77页上的图3-1呈现了主机概念的一个简单视图，我们在本课程中通篇采用的正是这种视图：

- 软件——z/OS操作系统由装载模块或者可执行代码构成。在安装过程中，系统程序员把这些装载模块拷贝到DASD上的装载函数库中(load libraries)。
- 硬件——系统硬件由构成主机环境的所有设备，控制器和处理器组成。
- 外围设备——这些外围设备包括了磁带机，DASD设备和控制台，还有很多其他类型的设备，其中一些在第2章，第35页上的"主机硬件系统和高可用性"中讨论。

- 处理器存储器——通常被称为实存或中央存储器(或内存)，这是z/OS操作系统执行的地方。同样，所有用户程序和操作系统共享处理器存储器。

作为一个典型的主机硬件配置的全局，图3-1 给出的绝不是全部。譬如，还有连接主机到其他磁带机，DASD设备和控制台的硬件控制单元等就没有出现在图上。

相关阅读：关于z架构的主要设备描述的标准参考书是IBM出版的《z/Architecture Principles of Operation》。在下面的z/OS互联网知识库网站中，能找到这本书及相关出版物。

http://www.ibm.com/servers/eserver/zseries/zos/bkserv/

3.2.2 多道程序设计和多重处理

多道程序
同时执行多个程序

78

最早的操作系统被设计用来控制单用户计算机系统。那时，操作系统读入一个任务，找到这个任务需要的数据和设备，让任务运行直到完成，然后读入另一个任务。相反地，由z/OS管理的计算机系统能够实现多道程序设计，或者同时执行多个程序。有了多道程序设计，当一个任务无法使用处理器的时候，系统可以挂起或者说中断当前任务，释放处理器让它处理另外一个任务。

在另一个程序执行之前，z/OS获取并保存中断程序的所有相关信息，以此使多道程序设计成为可能。当中断程序准备再次投入执行时，它可以在中断的地方恢复执行。多道程序设计允许z/OS为多个用户同时运行数以千计的程序，这些用户可能在世界各地从事不同的业务。

z/OS还可以执行多重处理，即两个或更多的处理器同时运行，它们共享各种的硬件资源，如内存和外部磁盘存储设备。多道程序设计和多重处理技术使得z/OS非常适合处理需要多I/O操作的工作负载。典型的大型机工作负载包括长时间运行的应用程序，它可以更新数据库中的上百万条记录，以及在任何给定时刻，成千上万交互式用户使用的在线应用程序。作为对比，考虑可能在单用户计算机系统上使用的操作系统。这种操作系统通常只需要为一个用户执行程序。例如，在个人计算机(PC)中，一个用户可以独占整个计算机的所有资源。

多重处理
两个或更多的处理器同时运行，它们共享各种的硬件资源

多个用户运行许多各自的程序意味着，除了需要大量复杂的硬件以外，z/OS还需要大量的内存来确保相应的系统性能。大型企业需要运行复杂的商业应用软件来访问大型数据库和企业级中间件产品。这些应用软件需要操作系统在保护用户间隐私的前提下共享数据库和软件服务。

因此，多道程序设计，多重处理和大量内存的需求意味着z/OS必须提供除了简单的单用户应用程序以外的更多功能。接下来的几小节概括地介绍了能够使z/OS管理复杂计算机配置的特性。本章的后面部分更详细地探讨了这些特性。

3.2.3 模块和宏

z/OS是由控制计算机系统运行的一系列程序指令组成的。这些指令确保有效地使

用计算机硬件并允许应用程序运行。z/OS包括一些指令集，例如，接受作业，把作业转化成计算机能识别的形式，跟踪作业，为作业分配资源，执行作业，监控作业和处理输出。一组相关的指令称为一个程序或功能模块。能完成某项特殊系统功能的相关功能模块集成为系统组件。例如常见的有z/OS工作负载管理组件(WLM)负责控制系统资源；而恢复终止管理器(RTM)处理系统恢复。

79

执行常用的系统功能的一系列指令能通过调用可执行的宏指令实现。z/OS宏可以完成诸如打开和关闭数据文件，加载或者删除程序，发送消息给计算机操作员等功能。

3.2.4 控制块

当程序执行z/OS系统的任务时，它们在名为控制块的存储区域监控该任务的执行情况。一般说来，在z/OS中有四种类型的控制块：

- 与系统相关的控制块
- 与资源相关的控制块
- 与作业相关的控制块
- 与任务相关的控制块

每个与系统相关的控制块就代表一个z/OS系统并包含整个系统范围内的信息，譬如有多少处理器在工作。每个与资源相关的控制块就代表某种资源，譬如一个处理器或存储设备。每个与作业相关的控制块就代表一个在系统中执行的作业。每个与任务相关的控制块就代表一个工作单元。

控制块
一种作为传输媒介为整个z/OS通信提供服务的数据结构

控制块作为传输媒介为整个z/OS中的通信提供服务。这种通信之所以可以实现是因为使用控制块的程序了解控制块的结构，所以这些程序可以找到与任务单元或资源相关的必需信息。代表许多同类型单元的控制块可以串接成队列，每个控制块指向队列中的下一个控制块。操作系统可以查询队列以找到某个特定任务单元或资源的信息，这些信息可能包含：

- 一个控制块或必需程序(Required Routine)的地址
- 实际数据，譬如一个数值，一个数量，一个参数或一个名称
- 状态标志(通常是一个字节中的某个位，字节中每个位都有特定的含义)

z/OS使用很多不同类型的控制块，许多有专门的目的。这一节讨论了三种最常使用的控制块：

- 任务控制块(TCB)，用于表示一个任务单元
- 服务请求块(SRB)，用于表示一个系统服务的请求
- 地址空间控制块(ASCB)，用于表示一个地址空间

80

3.2.5 z/OS 使用的物理存储

从概念上来说，主机和所有其他类型计算机都有两类物理存储[2]。

中央存储器
处理器上的物理存储

- 主机处理器自身的物理存储。也称为处理器存储器，实存或中央存储器，它是主机的内存。
- 主机外部的物理存储，包括直接访问的存储设备，譬如磁盘驱动器和磁带驱动器。该物理存储被称为页面存储或辅助存储。

这两种类型的存储的主要区别与访问它们的方式有关，如下：

辅存
主机外部的物理存储，包括直接访问的存储设备，譬如磁盘驱动器和磁带驱动器。

- 实存的访问与处理器是同步的。就是说，在数据从实存[3]中取出之前处理器必须等待。
- 辅助存储是异步访问方式。处理器通过输入/输出(I/O)请求访问辅助存储，该I/O请求和其他工作请求一起在系统中调度运行。在一个I/O请求过程中，处理器可以自由执行其他不相关的任务。

和个人计算机的内存一样，主机中央存储器和处理器本身紧密地耦合在一起，相反，主机辅助存储位于速度较慢的(相对实存而言)外部硬盘和磁带上。因为实存与处理器耦合得更加紧密，所以相对比辅助存储，处理器只需花费少得多的时间访问实存中的数据。然而，辅存比实存便宜。大多数z/OS系统都采用了大量的实存和辅存。

3.3 z/OS 设备概述

扩展系统工具集和独特的属性使z/OS特别适合处理大型的复杂的工作负载，譬如那些要求许多I/O操作，海量数据存取或者广泛的安全性的工作。

典型的大型机工作负载包括长时间运行的应用程序，用于更新数据库中的上百万条记录，以及那些同时为数以千计的用户提供服务的在线应用程序。

图3-2 显示了z/OS操作环境的一个”快照”视图。

[2] 许多计算机都有一个快速存储，局部于处理器，称为处理器缓冲区。该缓冲区对程序员或应用程序甚至操作系统都不是直接可见的。

[3] 某些处理器的实现是通过使用指令或数据预取或“流水线”技术来提高性能。这些技术对应用程序甚至操作系统都是不可见的，但一个复杂的编译器可以组织生成的代码来利用这些技术的优势。

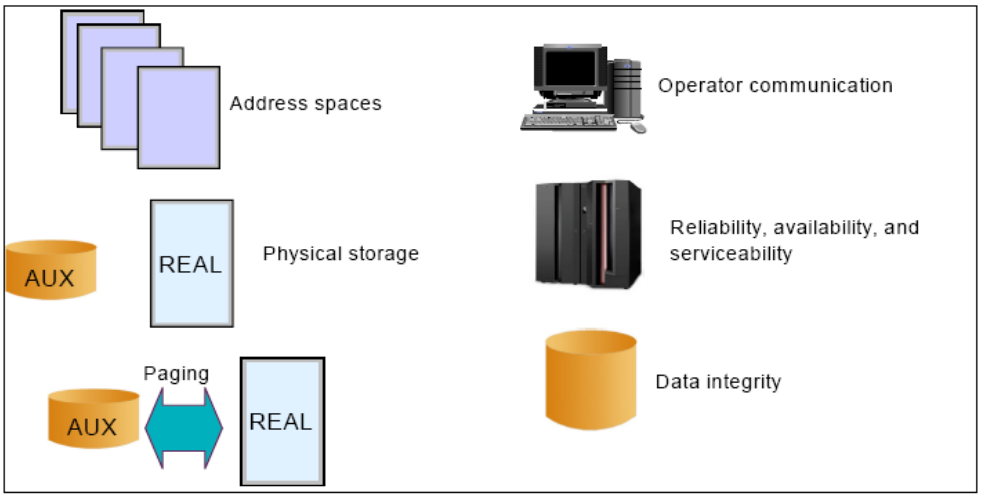


图3-2 z/OS操作环境

本文接下来的部分会更加深入地探讨这些设备，这里先概括地介绍一下：

- 地址空间，描述在线用户或者运行程序能够使用的虚存寻址范围。
- 两种可用的物理存储：中央存储器和辅助存储(AUX)。中央存储器也称为实存或者内存。
- z/OS通过页面调度和交换过程实现程序和数据在中央存储器和辅存之间的移动。
- z/OS分配要执行的任务(图中没有显示)。即，它根据优先权和执行能力选择要运行的程序，然后把程序和数据载入中央存储器。执行时所有程序指令和数据都必须在中央存储器中。
- 扩展工具集管理直接访问存储设备(DASDs)或磁带机上存放的文件。
- 操作员使用控制台来启动和停止z/OS，输入命令和管理操作系统。

z/OS还有其他一些操作特性，譬如安全性，可恢复性，数据完整性和工作负载管理。

3.4 虚拟存储和其他主机概念

z/OS使用两种类型的物理存储(中央存储器和辅存)来实现另外一种存储——虚拟存储。在z/OS中，每个用户都访问虚拟存储，而不是物理存储。虚拟存储的使用对实现z/OS独特的能力是非常重要的，该能力包括在处理大量工作负载的同时并发地和大量用户交互。

3.4.1 虚拟存储是什么？

虚拟存储意味着每个运行的程序可以认为它访问的是由体系结构寻址模式定义的所有存储。唯一的限制是存储器地址位的个数。这种使用大量存储空间的能力是很重要的，因为程序可能很长很复杂，并且程序的代码和所需数据都必须在中央

存储器中以供处理器访问。

z/OS支持64位的地址，这允许一个程序寻址高达18,446,744,073,709,600,000字节(16EX)的存储空间。在实际中，主机一般会装配比这个小得多的中央存储器。具体小多少取决于计算机的型号和系统配置。

为了使每个用户执行时感觉在计算机系统中好像真有这样大的存储空间，z/OS只把每个程序的活动部分存放于中央存储器中，而把其余的代码和数据保存在辅助存储器的称为页数据集(page data sets)的文件中，这种辅助存储通常由一些高速的直接访问存储设备(DASDs)组成。

虚拟存储器实际就是主存和辅存的结合，z/OS用一系列表和索引来关联辅存空间和中央存储器空间上的位置。它也使用特殊的设置(位设置)来跟踪每个用户或程序的身份和授权。z/OS使用多种存储管理组件来管理虚拟存储。本节简单描述该处理过程中的关键点。

该处理过程在86页上的3.4.4节的"虚拟存储概述"中有详细描述。

术语：主机工作人员交替使用中央存储，实际内存，实际存储及主要存储等术语。类似地，他们用虚拟内存和虚拟存储表达同一个意思。

3.4.2 地址空间是什么？

操作系统分配给用户或独立运行程序的虚拟地址范围称为一个地址空间(address space)。地址空间是一组相邻的虚拟存储空间，用来执行指令和存储数据。地址空间的虚拟地址范围从0开始，并可扩展到操作系统的体系结构允许的最高地址。

地址空间
操作系统分配给用户或程序的虚拟地址的范围。

z/OS为每个用户提供独立的地址空间，并维护属于各个地址空间的程序和数据之间的差异。在每个地址空间中，用户可以启动多个任务，使用任务控制块或TCBs来支持多道程序设计。

从某种程度上讲，一个z/OS地址空间就好像一个UNIX系统的进程，而地址空间标识符(ASID)则好比进程ID(PID)。进一步说，TCBs正像UNIX线程，操作系统用它们来支持任务的多个实例并发处理。

然而，z/OS中地址空间的这种使用方式，有很多独特的优点。虚拟寻址允许一个超过系统中央存储器容量的寻址范围。通过给每个作业分配自己独立虚拟地址空间，多重虚拟地址空间的使用为系统中的每个用户提供了这种虚拟寻址能力。潜在的大量地址空间给系统提供了一种极大的虚拟寻址能力。

有了多重虚拟地址空间，除了那些在通常的可寻址存储中(commonly addressable storage)的错误，一般的错误能被限制在一个地址空间内。这样就能提高系统可靠性，并且使错误恢复更加容易。在分离的地址空间内的程序互相独立，达到保护的功效。将数据隔离在各自地址空间中也可以起到保护数据的作用。

z/OS使用许多地址空间。运行中的每个作业都至少有一个地址空间，每个通过

TSO，telnet, rlogin 或FTP登录的用户至少有一个地址空间(用户通过一个主要的子系统，像CICS或者IMS，来登录z/OS,他们使用该子系统的地址空间，而不是他们自己的地址空间)。还有许多地址空间是为实现操作系统的功能开设的，这些功能包括操作员通信(operator communication)，自动控制，网络，安全等等。

地址空间隔离

地址空间使z/OS能够维持每个地址空间的程序和数据的区别。一个用户地址空间内的私有区域和其他地址空间的私有区域互相隔离，这对操作系统的安全性大有裨益。

然而，每个地址空间也包括了一个其他任何地址空间都能访问的公有区域。因为它映射了所有可用的地址，所以地址空间包括系统代码和数据以及用户代码和数据。因此，不是所有的映射地址对用户代码和数据来说都是可用的。

很多用户可以共享同一资源，这意味着需要保护用户，使它们互相独立，并且需要保护操作系统本身。除了使用"关键字"来保护中央存储器，使用码字来保护数据文件和程序，还要使用隔离的地址空间以确保多个用户的程序和数据不会在存储区域中重叠。

地址空间通信

在多重虚拟地址空间环境中，应用程序需要实现地址空间之间的通信。z/OS为地址空间之间的通信提供了两种方法：

- 调度一个服务请求块(SRB)，一个异步过程
- 使用交叉内存服务和访问寄存器，一个同步过程

程序使用SRB在另一个或同一个地址空间内开始一个进程。SRB本质上是异步的，独立于启动它的程序而运行。因此提高了在一个多处理环境中的资源可用性。在112页上的"SRB是什么？"中有进一步的讨论。

一个程序使用交叉内存服务可以直接访问另一个用户的地址空间。您可能把z/OS的交叉内存服务和UNIX共享内存功能相比，后者没有特殊授权也能在UNIX上使用。然而，不像UNIX，z/OS交叉内存服务要求发起的程序要拥有特殊的授权，这是由授权程序设备(APF)控制的。这种方法允许高效、安全地访问数据，这其中包括他人持有的数据，或者为了方便存储在另一个地址空间的用户自己的数据，除此以外，还能和一些服务进行快速安全的通信，这些服务包括交易管理器和数据库管理器等。

相关阅读：在IBM出版物《z/OS MVS Programming: Extended Addressability Guide》中详细描述了交叉内存服务。在下面的z/OS 互联网知识库网站上，能找到这本书及相关出版物。

http://www.ibm.com/servers/eserver/zseries/zos/bkserv/

3.4.3 动态地址转换是什么？

动态地址转换(DAT)，是指在存储定位过程中将虚拟地址转换成所对应的实存地址的过程。如果虚拟地址已经存在于中央存储器中，DAT就通过使用转换后备缓冲区(TLB)而加速处理过程，如果虚拟地址不在实存中，则产生缺页中断，通知z/OS，然后z/OS从辅助存储器中调入该页。

85

仔细分析以上过程发现，机器能呈现多个不同类型错误中的任何一个。到底是一个类型错误，区域错误，段错误还是一个页错误，取决于在DAT结构上的哪一点上发现的无效条目。错误会一直在DAT结构中延续直到页错误最终被发现，然后虚页被第一次导入到中央存储器(辅存上没有拷贝)或是从辅存中导入。

DAT是通过页表，段表，区域表和转换后备缓冲区等硬件和软件共同实现。DAT允许不同地址空间共享相同的程序和其他的只读数据。这是因为不同地址空间中的虚拟地址在中央存储器中可以被转换成相同的页框。否则，中央存储器中将会有很多份程序和数据的备份，每个地址空间都需要一份。

3.4.4 虚拟存储概述

回顾一下，处理器执行一条程序指令时，指令和它所涉及到的数据两者都要存放在中央存储器中。当指令执行时，早期操作系统一般是将整个程序读入实存中。但是当执行一条指令时，整个程序实际上不需要全部处于实存中。而实际上，当处理器就绪时将多个程序块读入中央存储器，而将不需要的程序块移动到辅助存储器中，如此一来，操作系统就可以并发执行更多更大的程序。

操作系统是如何跟踪每个程序块的？它如何知道程序块是在中央存储器中还是在辅助存储器中？具体在什么位置？对于z/OS专业人员来说，理解操作系统如何实现这些是很重要的。

物理存储器被分成很多区域，每个区域大小相同并且拥有唯一的访问地址。在中央存储器中这些区域被称为页框(Frame)；在辅存中这些区域被称为页槽(Slot)。类似地，操作系统可以把程序分成若干大小与页框和页槽相同的块，并给每个块分配唯一的地址。这种安排允许操作系统保持对这些块的追踪。在z/OS中，程序块被称为页。在90页上的"页框，页和页槽"一节中有更详细的讨论。

86

页通过虚拟地址被引用，而不是通过实际地址。从程序进入系统到完成，页的虚拟地址保持不变，不管该页是在中央存储器还是辅存中。页由多个字节构成，每个字节都有唯一的虚拟地址。

虚拟地址的格式

如上文所述，我们可以将虚拟存储看作是体系结构创建的一个假象，使系统看起来比实际拥有更多的内存。每个用户或程序使用一个地址空间，每个地址空间包含同样范围的存储地址。地址空间中，只有那些需要运行的部分才真正被装载入中央存储器中，z/OS将地址空间不活动的部分保存在辅助存储器中。z/OS以不同

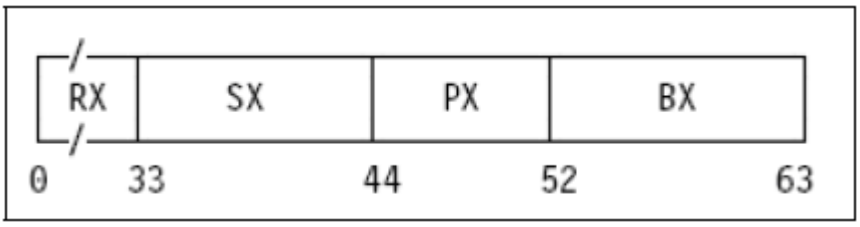
大小的单元管理地址空间，如下：

页： 地址空间被划分成4KB大小的虚拟存储单元，称为页。

段： 地址空间被划分成1MB大小的单元的，称为段。一个段是连续的跨越兆字节的虚拟地址块，它从1MB的边界处开始。譬如，一个2GB的地址空间由2048个段组成。

区域： 地址空间被划分成2-8GB大小的单元，称为区域。一个区域由跨越2-8GB的连续虚拟地址块组成，它从2GB的边界开始。譬如，一个2TB的地址空间由2048个区域组成。因此，虚拟地址被划分为四个主要的区域，0-32位称为区域索引(RX)，33-43位称为段索引(SX)，44-51位称为页索引(PX)，52-63位称为字节索引(BX)。

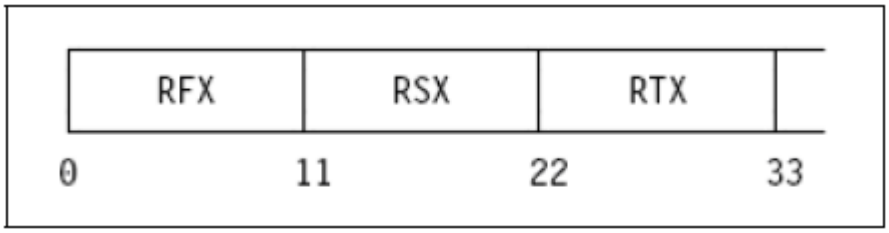
虚拟地址的格式如下：



由地址空间控制元素决定，虚拟地址空间可以是由一个区域组成的2GB大小的空间，也可以是16-exabyte大小的空间。在2GB大小的地址空间的应用中，虚拟地址的RX部分必须全为零。否则就会产生异常。

87

虚拟地址的RX部分本身被划分成3个区域，0-10位称为区域第一索引(RFX)，11-21位称为区域第二索引(RSX)，22-32位称为区域第三索引(RTX)。虚拟地址的0-32位格式如下：



以RTX为左边最高有效部分的虚拟地址(42位地址)能编址4T字节(4096个区域)，以RSX为左边最高有效部分的虚拟地址(53位地址)能编址8PB(四百万个区域)，而以RFX为左边最高有效部分的虚拟地址(64位地址)能编址16 EB(八十亿个区域)。

z/OS中虚拟存储寻址如何工作

正如前面提到的，使用z/OS中虚拟存储意味着只有当前活动的程序块在执行时需要处于中央存储器中。不活动的程序指令则存放在辅助存储器中。图3-3显示了z/OS中虚拟地址的工作原理。

88

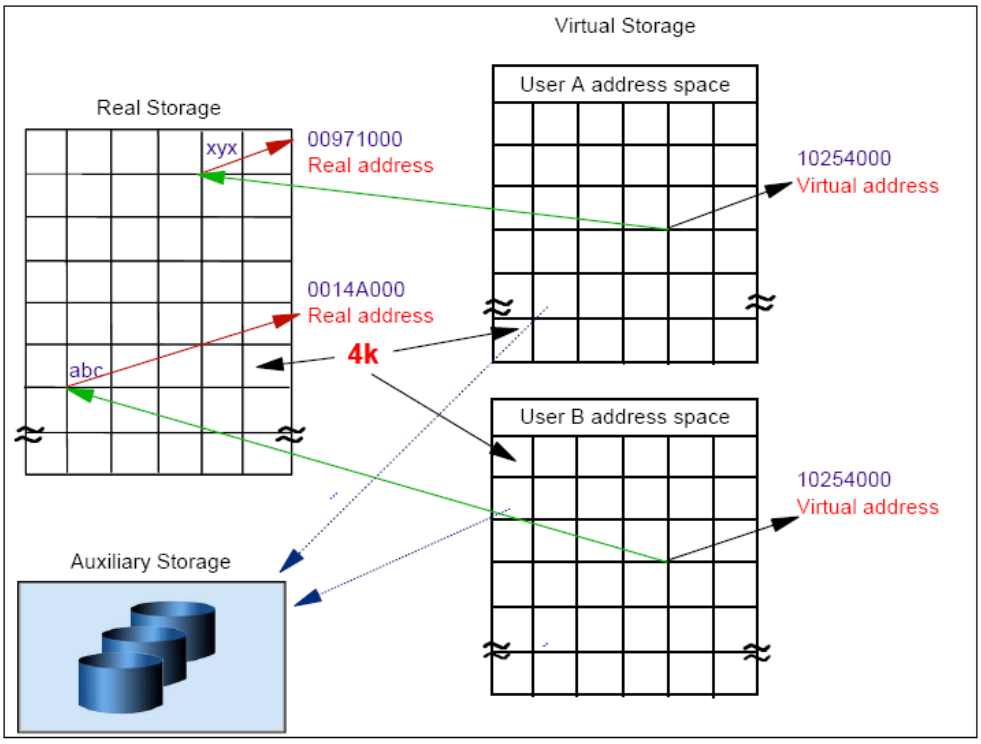


图3-3 实存和辅存结合起来构造成虚存

在图3-3中，注意以下几点：

- 地址就是一个所需的信息块的标识符，但不是信息块在中央存储器中位置的说明。这就允许地址空间(即一个程序可用的所有地址)超过可用的中央存储器的大小。
- 对多数用户程序，所有的中央存储器定位是根据虚拟存储地址[4]实现的。
- 在存储定位过程中，动态地址转换(DAT)用来将一个虚拟地址转换成中央存储器中的一个物理位置。如图3-3所示，虚拟地址10254000可以多次使用，因为每个虚拟地址可以映射到中央存储器中不同的地址。
- 当所请求的地址不在中央存储器中时，一个硬件中断产生并发给z/OS，操作系统将所需的指令和数据页调入中央存储器中。

帧
位于中央存储中相同大小的区域，可以由一个唯一的地址访问。

槽
位于辅助存储中相同大小的区域，可以由一个唯一的地址访问。

页框，页面和页槽

当程序被选中执行时，系统将其调入虚拟存储中，分成4KB大小的页，将页面调入中央存储器中执行。对于程序员来讲，感觉整个程序一直占用了连续的存储空间。

实际上，并非程序的所有页面都有必要放在中央存储器中，并且在中央存储器中的页面也不必要占据连续的存储空间。

在虚拟存储中执行的程序块必须在实存和辅存之间进行移动。为了实现这个移动，z/OS按4K字节的单元或块来管理存储器。存储器中定义了以下的块：

[4] 一些指令，主要是操作系统所使用，要求实际地址。

- 在中央存储器中，这个4K大小的块被称为页框。
- 在虚存中，这个4K大小的块被称为页。
- 在辅存中，这个4K大小的块被称为页槽。

页框，页和页槽的大小均为4KB。一个活动的虚拟存储页面驻留在中央存储器页框中，而不活动的虚拟存储页面则驻留在辅存的页槽中(在页面调度数据集中)。图3-4说明了系统中页，页框和页槽的关系。

在图3-4中，z/OS为虚拟存储器中运行的程序进行页面调度。假设每个印有字母的方块代表程序的一部分。在这个简化的视图中，程序的A，E，F和H部分处于活动状态，运行在中央存储器页框中，而程序的B，C，D和G部分是不活动的，被移动到辅存的页槽中。然而，所有的程序部分都驻留在虚拟存储中并拥有虚拟存储地址。

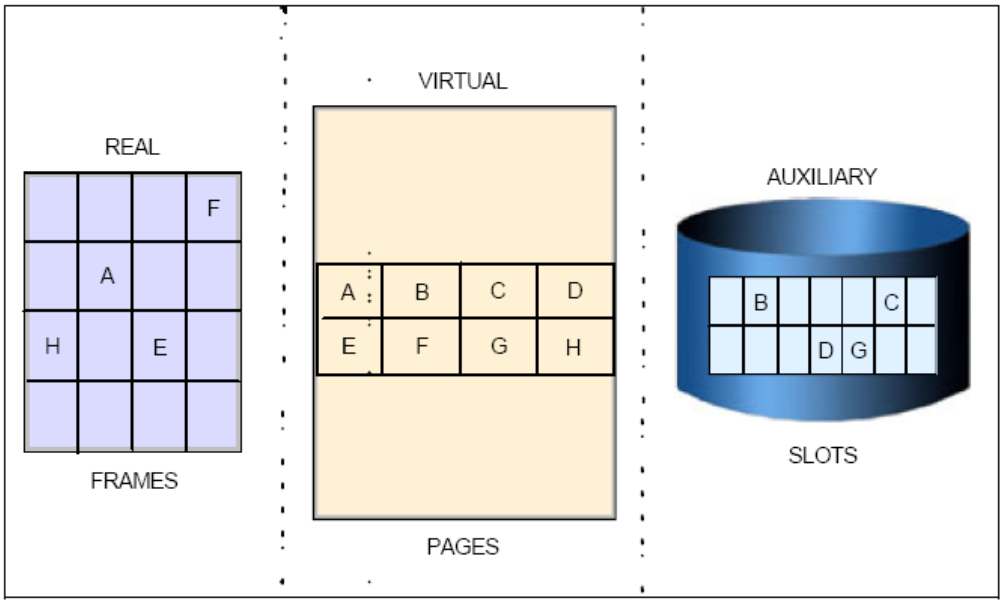


图3-4 页框，页和页槽

3.4.5 页面调度是什么？

正如前文提到的，z/OS使用一系列的表来确定一个页是在实存中还是在辅存中，并确定其在存储器中的位置。在查找程序的一个页时，z/OS通过查表的方法来得到该页的虚拟地址，而不是搜索所有的物理存储来查找该页。然后z/OS根据需要将该页调入到中央存储器或者调出到辅存中。在辅存的页槽和实存的页框之间的这种页面的移动称为页面调度。页面调度是理解z/OS中虚拟存储的关键。

z/OS中的页面调度对于用户来说是透明的。在执行作业时，只有那些必需的应用程序块才被调入中央存储器中。当页面不再需要，或者同一应用程序或具有更高优先级的应用程序要求调入新页面，而又没有空闲的中央存储器空间时，页面将被调出。z/OS采用近期最少使用算法来选择调出到辅存的页。这个算法认为在一段时间内没有使用的页面在将来的一段时间内也不会再使用，所以选择这个页面

作为替换页面。

z/OS中页面调度如何工作

除了DAT硬件和地址转换要求的段表页表外，页面调度还包括了使用一些系统组件来移动页面，以及几个附加表来跟踪每个页面的最新版本。

91

为了理解页面调度是如何工作的，假定DAT在地址转换过程中遇到一条无效页表条目，显示一个需要的页面当前不在中央存储器页框中。为了解决这个页错误，系统必须从辅存中把该页调入。然而，首先它必须分配中央存储器里的一个可用页框。如果没有一个可用的，请求必须被保存并释放一个指定页框。为了释放一个页框，系统就把页框的内容拷贝到辅存中，然后把该页框对应的页表条目标记为无效的。这个操作被称为页调出。

为需要的页找到了一个可用页框后，页的内容就从辅存拷贝到中央存储器中，页表的无效位就可以关掉。这个操作称为页调入。

页面调度也可能发生在z/OS把整个程序载入虚存的情况。z/OS为用户程序获取虚拟存储并为每个页分配一个中央存储器页框。每个页随后激活，可以进行正常的页面调度活动，即最活跃的页面会驻留在中央存储器中，而当前不活跃的页就有可能被调出到辅存中。

页面窃取

z/OS总是尽力保持足够多的可用的中央存储器页框。当程序涉及到的页面不在中央存储器中的时候，z/OS从可用的页框中选取一个用于存储页。

当可用页框减少时，z/OS用页面窃取的方法来补充。也就是说，它获得一个分配给活动用户的页框，并使它对别的任务可用。窃取哪个特定页面取决于当前驻留在中央存储器页框中的每个页面活动的历史记录。相当长一段时间未活动的页面比较适合用于页面窃取。

未被引用的时间间隔数

z/OS使用复杂的页面调度算法判断哪些页面最近被使用，来高效管理虚拟存储。未被引用的时间间隔计数显示了自从程序引用页以来持续了多久时间。在定期的时间间隔，系统会检查每个页页框的引用位。如果引用位未被使用，那就是说页框没有被使用，系统就增加页框的未被引用的时间间隔数。它加上地址空间上次检查引用计数后经过的秒数。如果使用了引用位，说明页框被使用了，系统就把引用位关闭，把未被引用的时间间隔数置为0。未被引用的时间间隔数最高的页框最有可能被窃取。

92

z/OS还使用各种存储管理器来监控系统中所有的页，页框和页槽。这部分将在94页3.4.8 节中的”存储管理器的角色”中阐述。

3.4.6 交换和工作集

交换
在中央存储器和辅存之间交换一个地址空间所有页的过程。

交换(Swapping)是指在中央存储器和辅存之间交换一个地址空间所有页的过程。被换入的地址空间是活动的，在中央存储器的页框中和辅存的页槽中都有页面。被换出的地址空间是不活动的；它驻留在辅存中，直到被换入中央存储器之后才能执行。

只有地址空间页的一个子集(称为它的工作集)才有可能一直都存在中央存储器中，交换用于有效地移动整个地址空间。它是z/OS采用的平衡系统工作负载、确保有足够中央存储器页框的几个方法之一。

交换由系统资源管理器(SRM)组件执行，是对来自工作负载管理器(WLM)组件建议的响应。WLM将在104页，3.5节的"什么是工作负载管理？"中描述。

3.4.7 存储保护是什么？

目前为止，我们讨论的虚拟存储都是建立在单用户或单个程序的基础上。而在实际中，当然有很多程序和用户在竞争着使用系统。z/OS使用如下技术来保持每个用户任务的完整性：

- 为每个用户提供一个私有的地址空间
- 页保护
- 低地址保护
- 本节将要描述的多重存储保护键

存储保护键是如何使用的

在z/OS中，中央存储器中的信息通过使用多重存储保护键而避免未经授权的使用。存储中的控制域称为键，存储器中一个被称为键的控制区域和中央存储器中的每个4K大小的页框相关。

当发生修改中央存储器某个位置的内容的请求时，和该要求相关的键就被拿来和存储保护键相比较。如果两者相符或者程序在键等于0的情况下执行，则请求被满足。如果两者不相符，系统就会拒绝该请求，产成一个程序异常中断。

当发生读取中央存储器中某个位置的内容的请求时，请求会自动被满足，除非读取保护位是打开的，预示着该页框是读保护的。当发生读取受到读保护的中央存储器内容时，存储键就被拿来和请求键做比较。如果两者相符，或请求键等于0，请求则会得到满足。如果两者不相符，而且请求键不等于0，系统就会拒绝该请求，并产成一个程序异常中断。

存储保护键是如何分配的

z/OS使用16个存储保护键。根据正在执行的任务的种类会分配一个特定的键。如图3-5中所示，键被存储在程序状态字(PSW)的8到11位中。系统中的每个作业都

被分配了一个PSW。

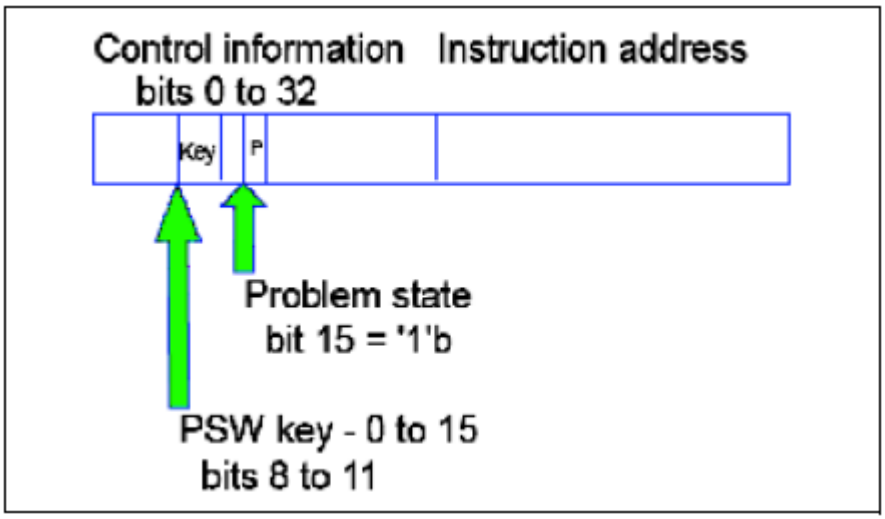


图3-5 存储保护键的位置

存储保护键0-7由z/OS的基础控制程序(BCP)、不同的子系统和中间件产品使用。存储保护键0是主键。它仅限于BCP的一部分程序使用，这些程序要求几乎不受限的存储和读取能力。几乎在任何情况下，和保护键0相关的请求，要求访问或者修改实存中某处内容都能得到满足。

存储保护键8-15分配给用户。因为所有用户都隔离在自己的私有地址空间中，多数用户——那些在虚存里运行程序的用户——可以使用相同的存储保护键。这些用户被称为V=V(虚拟=虚拟)用户，被分配了保护键8。然而，也有一些用户必须在中央存储中运行。这些用户被称为V=R(虚拟=实际)用户，他们需要独立的存储保护键因为他们的地址不受DAT过程保护，DAT能保证每个地址空间不同。如果没有独立的键，V=R用户可能会引用到互相之间的代码和数据。这些键的范围是从9-15。

3.4.8 存储管理器的角色

94

中央存储器页框和辅存页槽，以及它们支持的虚存页，是由z/OS单独的组件管理的。这些组件被称为实存管理器(抱歉，不是中央存储管理器)，辅存管理器以及虚存管理器。这里我们简单描述以上每一个角色。

实存管理器

实存管理器或RSM™跟踪中央存储的内容。它管理我们早先讲述过的页面调度活动，譬如页换入，页换出以及页面窃取，并且，它还用在地址空间的换入换出。RSM还能执行页面锁定(使页面无法被窃取)。

辅存管理器

辅存管理器或ASM使用系统的页数据集来跟踪辅存页槽。特别用于跟踪：

- 那些不在中央存储器页框中的虚存页使用的页槽
- 那些不占据页框的页使用的页槽，但由于页框的内容没有改变，所以页槽还是有效的。

当请求页换入或页换出时，ASM和RSM一起工作来确定合适的中央存储器页框和辅存页槽。

虚存管理器

虚存管理器或VSM™响应要求得到和释放虚存的请求。VSM还管理那些必须在实存而不是虚存中运行的程序的存储分配工作。当代码和数据被载入虚存时，将对它们分配实存。在运行中，它们可以通过系统服务比如GETMAIN宏来请求更多的存储。程序可以通过FREEMAIN宏来释放存储。

VSM跟踪每个地址空间的虚存映射。这样做，它把每个地址空间看做256个子池的集合，是一些由数字0-255标识的逻辑相关的虚拟存储区域。逻辑相关意味着同一个子池中的存储区共享这些特性：

- 存储保护
- 它们是否有读取保护，是否可页调度或页交换
- 它们必须驻留在虚存中的位置(在16M以上或以下)
- 它们是否可以被多个任务共享

部分子池(从128到255号)已经被系统程序预定义使用了。例如，子池252是给来自授权库的程序使用的。其他子池(0-127号)可以由用户程序定义。

3.4.9 虚拟存储的历史和 64 位寻址

1970年，IBM开发了System/370，这是第一个使用虚拟存储和地址空间的体系结构。从那以后，操作系统在许多方面都发生了变化。其中的一个关键内容就是寻址能力的变化。

一个在地址空间内运行的程序能引用所有和该地址空间相关的存储。在本文中，一个程序引用和一个地址空间相关的所有存储的能力被称作寻址能力。

寻址能力
一个程序引用和一个地址空间相关的所有存储的能力。

System/370定义的存储地址长度为24位，这意味着最大的访问地址为16,777,215字节(或2^{24}-1字节)[5]。24位寻址能力使得当时的操作系统MVS/370给每个用户分配一个16MB的地址空间。多年过后，当MVS/370获取了更多的功能并试图处理更复杂的应用程序时，甚至16M的虚存访问能力也满足不了用户的需求了。

1983年，随着System/370-XA体系结构的发布，IBM将体系结构的寻址能力扩展到31位。有了31位寻址能力后，操作系统(现在称为MVS扩展体系结构或MVS/XA™)虚拟存储的寻址能力从16MB增加到了2GB字节。换句话说，MVS/XA

[5] 地址从 0 开始，因此最后一个地址通常比地址字节的总数少 1。

为用户提供的地址空间是MVS/370的128倍。16MB地址成为两种体系结构的分界点，一般称为”线(line)”(见图3-6)。

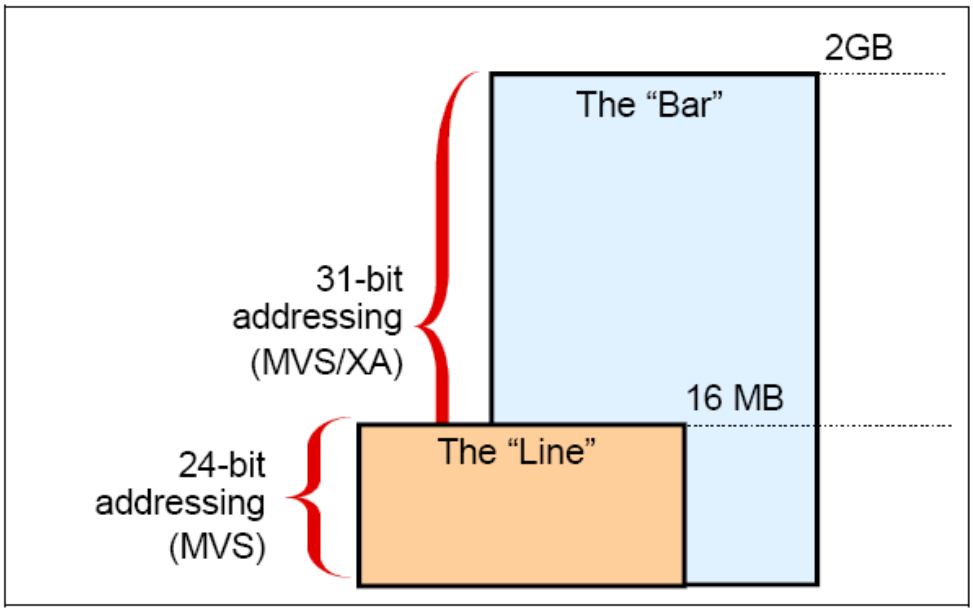


图3-6 MVS/XA6 31位寻址允许2G地址空间

96 这种新的体系结构不需要客户改变现有的应用程序。为了保留对现有程序的兼容性，MVS/XA支持运行在MVS/370上以24位地址寻址的程序，同时允许应用程序开发者编写基于31位寻址技术的新程序。

为了保持不同寻址模式之间的兼容性，MVS/XA不使用地址的最高位(位0)寻址。而是用这个位指示寻址时所用的地址位数：31位寻址(位0开)或者24位寻址(位0关)。

2000年，随着zSeries主机的发布，IBM进一步将体系结构的寻址能力扩展到64位。使用64位地址后，z/OS的地址空间的潜在能力扩大到需要用新的术语来描述。每个地址空间的大小为16千兆兆字节(EB)，称为一个64位地址空间；一个千兆兆字节比10亿GB要稍小些。新的地址空间有2^{64}个逻辑地址。是以前2GB地址空间的80亿倍，或者说18,446,744,073,709,600,000字节(图3-7)。 97

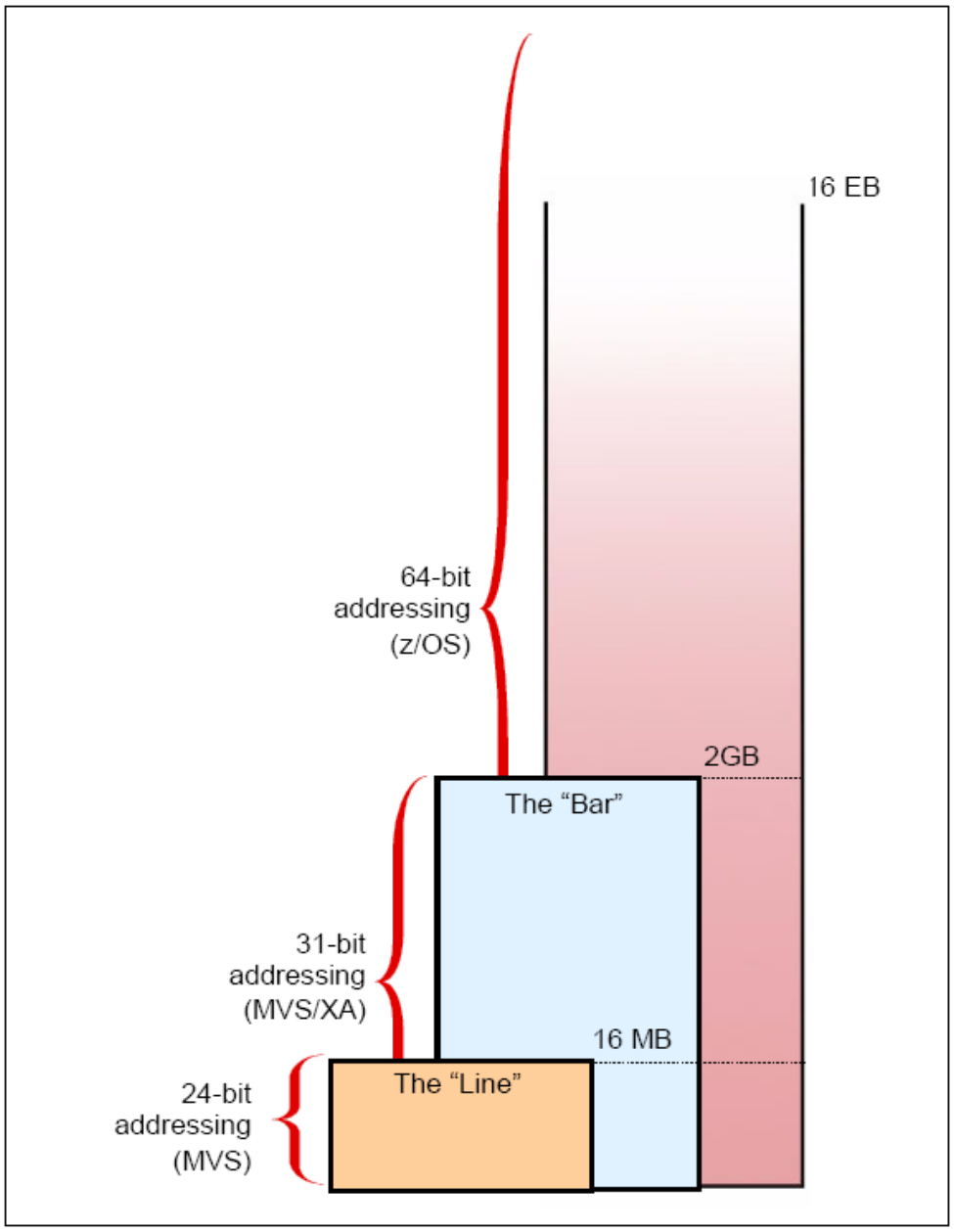


图3-7 64位寻址允许16EB可寻址存储

我们之所以说16EB为潜在大小，是因为在缺省情况下，z/OS仍然创建2GB大小的地址空间。当且仅当分配给一个程序运行的虚拟存储空间超过2GB时才突破这个限制。如此，z/OS操作系统把分配给用户的可用存储空间从2GB增加到16EB。

在z/OS和zSeries主机上运行的程序可以使用24位、31位或64位寻址(如果需要的话还可以在三者之间转换)。为了寻址64位体系结构中的可用的高位虚拟存储，程序使用64位专用指令。尽管体系结构引入了独立的64位开发指令，程序可以根据需要使用31位和64位指令。

为了保持兼容性，地址空间在2GB以下的存储区域的布局是相同的，提供了既能

支持24位也可以支持31位寻址的环境。将2GB以下的虚拟存储区域和用户私有区域分隔开的区域称为"条(bar)"，见图3-8中所示。用户私有领域是为应用程序代码而不是操作系统代码分配的。

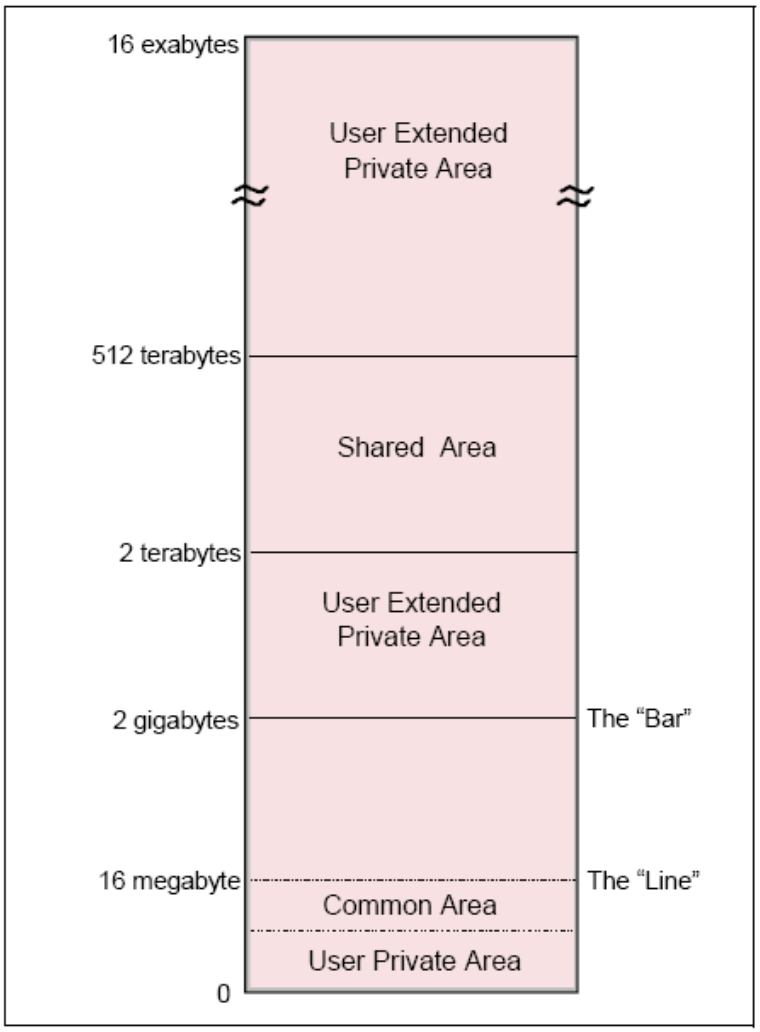


图3-8 64位地址空间的存储映射

0 – 2^{31} 布局相同；见图3-8。

2^{31} – 2^{32} 从2GB到4GB的区域称为条。在条以下，可以用31位寻址，而在条以上要用64位寻址。

2^{32} – 2^{41} 低位非共享区域(用户私有区域)从4GB开始扩展到2^{41}。

2^{41} – 2^{50} 共享区域(用于存储共享)从2^{41}开始扩展到2^{50}，如果有要求的话也可以扩展到更高。

2^{50} – 2^{64} 高位非共享区域(用户私有区域)从2^{50}或共享区域结束的任何位置开始直到2^{64}。

在以64位虚拟地址寻址的16EB地址空间中，转换表还有三个附加层，称为区域表：区域第三表(R3T)，区域第二表(R2T)和区域第一表(R1T)。区域表长为16KB，每个表有2048个条目。每个区域为2GB。

在条以下的虚拟地址中，段表和页表的格式不变。当转换一个64位虚拟地址时，一旦系统识别出指向段表的相对应的2GB区域条目，处理过程就和前面描述的一样。

3.4.10 “线下存储”意味着什么？

z/OS程序和数据在虚存中驻留，必要的时候，由中央存储支持。大多数程序和数据不依赖于实际地址。然而，有些z/OS程序依赖于实际地址，并且有些还要求实际地址低于16M。z/OS程序员把这种存储称为在”16M线以下”。

在z/OS中，一个程序的属性之一是驻留模式(RMODE)，它确定了程序是不是驻留(或被载入)在16M以下。RMODE(24)的程序必须驻留在低于16M以下的区域，RMODE(31)的程序可以驻留在虚存里的任何区域。

要求线下存储的程序例子包括了要求分配一个数据控制块(DCB)的任何程序。然而，这些程序通常可以是31位驻留模式或RMODE(31)，因为它们能运行在31位寻址模式或AMODE(31)。z/OS保留尽量多16M以下的中央存储，提供给那些程序，并且大多数情况下，在不要求程序做出任何变化的情况下，处理它们的中央存储独立性。

如今数以千计的正在运行的程序使用的是AMODE(24)及因此而来的RMODE(24)。每个在MVS/XA诞生之前编写的，并在之后没有改变的程序都有上述特性。几乎没有什么理由来解释为什么一个新程序需要是AMODE(24)，因此一个新的应用程序差不多也不会使用RMODE(24)。

3.4.11 地址空间中有什么？

100

另外一种理解地址空间的方式是把它看作程序员的代码和数据的可用虚拟存储的映射。地址空间让每个程序员可以访问整个计算机体系结构中所有可用地址(先前我们把这个定义为寻址能力)。

z/OS为每个用户提供一个唯一的地址空间并区别属于每个地址空间的程序和数据。由于地址空间映射了所有的可用地址，而一个地址空间中既包含了系统代码和数据，也包含了用户代码和数据。因此，并不是所有被映射的地址都可供用户代码和数据使用。

通过图表更容易理解地址空间中的存储区域的划分。图3-9所示的图表比本课程的这部分所要求的内容更为详细，在这里给出是为了说明地址空间区分了属于用户的程序和数据，也区分了属于操作系统的程序和数据。

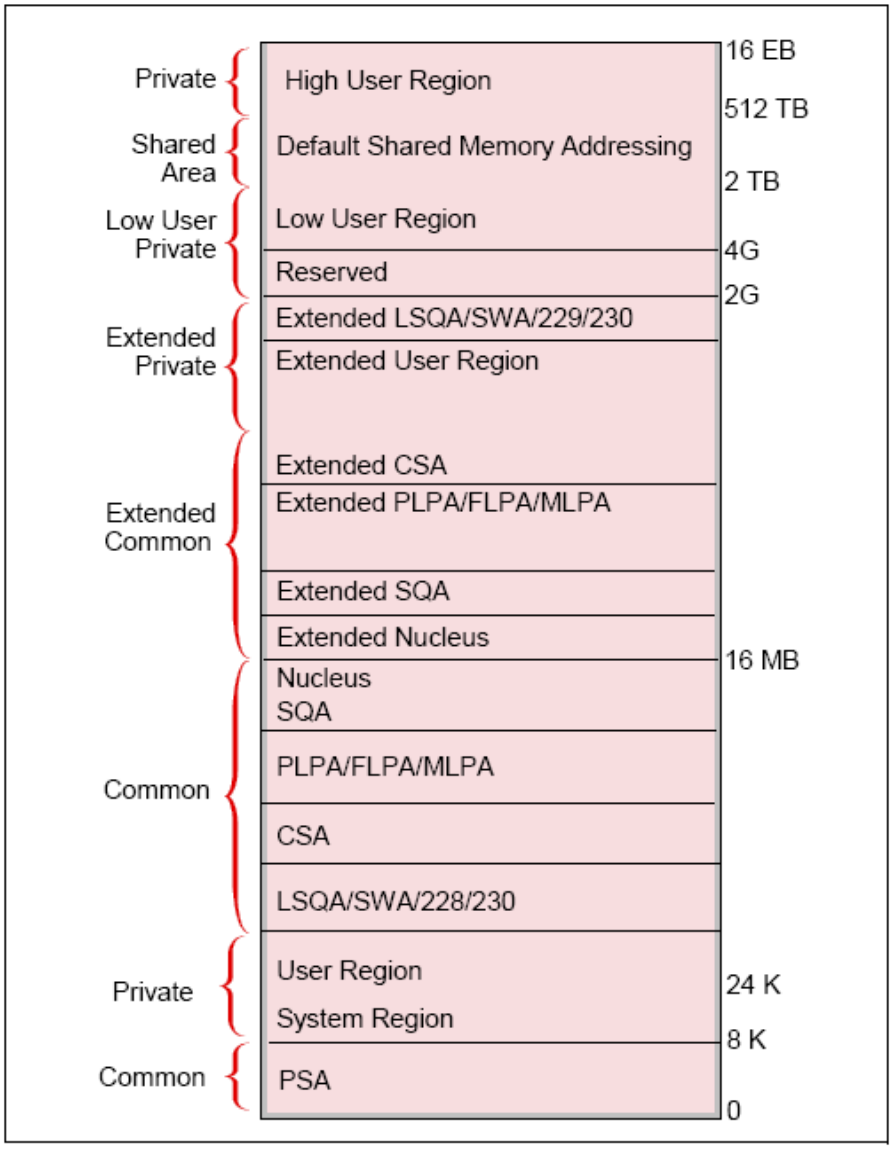


图3-9 地址空间的存储区域

图3-9显示了每个地址空间中的主要区域。下面是简要描述：

- 2GB以上的所有存储

  这个区域被称为高位虚拟区域，只能由运行在64位模式下的程序寻址。它被高位虚拟共享区域分割，该区域在安装时定义大小，可以用它来建立跨地址空间可视连接来获取该区域的某些区。

- 16MB以上的扩展区域

  这个范围内的区域，它们位于线(16MB)以上，但是在条(2GB)以下，有点像16MB以下公共领域的"镜像"。它们和线以下的相同区域，它们有相同属性，但由于加上了线以上的存储，所以它的空间更大。

- 内核

这是一个键值为0的只读公共存储区域，包含操作系统控制程序。

- SQA

  这个区域包含了系统层(键值为0)的数据，该数据可以被多个地址空间访问。SQA区域是不允许进行页调度的(固定的)这意味着它会一直驻留在中央存储中直到被要求使用它的程序释放。SQA区域的大小是在安装时预定义的，在操作系统活动时不能被改变。然而，它的独到之处是能"溢出"到CSA区域，前提是还有没有被使用的可以转化成SQA的CSA存储。

- PLPA/FLPA/MLPA

  这个区域包括了连接装配区(可调页连接装配区，固定连接装配区，修正连接装配区)，它包含了经常在多个地址空间运行的系统层次的程序。因为这个原因，连接装配区驻留在每个地址空间都可以寻址的公共区域。因此排除了让每个地址空间都拥有自己的程序副本的需要。这个存储区域在线下方，因此可以由运行在24位模式下的程序寻址。

- CSA

  公共区域的这个部分(所有地址空间都可寻址)对所有应用程序都是可访问的。CSA通常用来包含经常被多地址空间访问的数据。CSA区域的大小是由系统初始时间(IPL)建立的，在操作系统活动的时候不能改变。

- LSQA/SWA/子池228/子池230

  这是多个不同类别子池的集合，每一个都有自己专门的属性，它们主要被系统功能使用，当函数需要地址空间层次存储独立的时候。因为位于线下方，因此可以由运行在24位模式下的程序寻址。

- 用户区

  这个区域是任何运行在用户地址空间内的程序(包括用户关键程序)都能得到的。它驻留在线(Line)下方因此可以由运行在24位模式下的程序寻址。

- 系统区

  这个小区域(通常只有4页)是为每个地址空间的区域控制任务使用。

- 前置保护区(PSA)

  这个区域通常被称为"低位核心"。PSA是地址空间里地址为0到8191的虚存的公共区域。系统中每个处理器都有一个唯一的PSA。PSA映射从体系结构上来说，确定了处理器的硬件和软件存储位置。因为每个处理器都有一个独特的PSA，从运行在z/OS上的程序来说，PSA的内容可以在程序派发到一个不同的处理器上时发生变化。这个特性是PSA独有的，通过一个独特的称为前置(prefixing)的DAT操作技术来完成。

考虑到地址空间中的大范围的可寻址存储。图3-9所示的图表与实际不成比例。

系统中的每个地址空间都是由一个地址空间控制块(ASCB)代表的。为了表示一个地址空间，系统在公共区域(系统队列区域，SQA)创建了一个ASCB，使该地址空间能被其他地址空间访问。

3.4.12 系统地址空间和主调度程序

许多z/OS系统功能都在它们自己的地址空间运行。例如，主调度子系统，在一个叫”*MASTER*”的地址空间里运行，它在z/OS和自己的地址空间之间建立通信。

当启动z/OS时，主初始化程序初始化系统服务，如系统日志和通信任务，并创建主调度地址空间。然后，主调度器会启动作业输入子系统 (JES2或JES3)。JES是主要的作业输入子系统 。在许多生产系统中，JES不是立即启动的，相反，自动化包会依照一个受控顺序启动所有任务。然后其他子系统将被启动。

子系统都是在一个关于系统设置的特殊文件中记录，这个文件称为参数库或PARMLIB。这些子系统是二级子系统。

每个被创建的地址空间有一个关联的数字，称为地址空间ID(或ASID)。因为主调度地址空间是系统创建的第一个地址空间，所以它就是第一地址空间(ASID=1)。然后其他的系统地址空间在z/OS的初始化过程中逐一创建。

这里，您只需要理解z/OS和它的相关子系统都要有自己的地址空间，以实现操作系统的功能。下面简短地描述一下每种类型的地址空间：

- 系统

  z/OS的系统地址空间在主调度器初始化后创建。这些地址空间为z/OS上创建的所有其他类型的地址空间执行某些功能。

- 子系统

  z/OS要使用各种各样的子系统，如作业输入子系统 (JES，在231页第7章”批处理和JES”中讲述)。另外，还有为中间件产品如DB2，CICS和IMS的地址空间。

除了系统地址空间，当然还有典型的为用户和独立运行程序提供的地址空间。如：

- TSO/E地址空间是为每个登录到z/OS的用户创建的(在127页上的第4章”TSO/E，ISPF和UNIX：z/OS的交互工具”中描述)。
- 系统会为在z/OS运行的每个批处理作业创建一个地址空间。批处理作业地址空间是由JES启动的。

3.5 什么是工作负载管理？

对z/OS来说，系统资源的管理是负载管理(WLM)组件的职责。WLM根据公司的商业目标(比如响应时间)管理系统中工作负载的处理。为了完成这些目标，WLM也管理系统资源(比如处理器和存储)的使用。

3.5.1 WLM 做什么？

104

简单地说，WLM有三个目的：

- 完成安装时定义的商业目标，根据工作负载的重要程度和目标，给它们自动分配系统综合体资源。本目标称为目标达成(goal achievement)。
- 从系统角度看，完成系统资源的最佳利用。这个目标称为吞吐量(throughput)。
- 从单个地址空间角度看，完成系统资源的最佳利用。这个目标被称为响应和周转时间(response and turnaround time)。

目标达成是WLM首要并且是最重要的任务。随后是优化吞吐量和把周转时间控制到最小。通常后两者是互相矛盾的。优化吞吐量意味着要一直使用资源。而优化响应和周转时间则要求当需要资源时资源应该是可用的。完成一个比较重要的地址空间的目标可能会导致另一个次重要的地址空间的周转时间变得糟糕。因此，WLM必须做出保持冲突目标间平衡的决定。

工作负载管理
根据确定的商业目标来管理系统资源的z/OS组件。

为了平衡吞吐量和响应及周转时间之间的平衡，WLM完成以下工作：

- 监控各个地址空间使用的资源。
- 监控系统范围内使用的资源来决定它们是否被完全利用。
- 决定哪些地址空间被交换出去(以及交换的时间)。
- 当缺乏中央存储存时，阻止新地址空间的创建或进行页面窃取。
- 改变地址空间的分配优先权，从而控制了能够使用系统资源的地址空间的情况。
- 如果可以选择则选择设备进行分配，来平衡I/O设备的使用。

其他的z/OS组件，交易管理器和数据库管理器可以就某地址空间(或者整个系统)状态的变化，和WLM进行通信，或者调用WLM的决策制定能力。

譬如，当以下情况发生时，WLM会得到通知：

- 在系统中加入或者移出中央存储。
- 将要创建一个地址空间。
- 删除了一个地址空间。
- 开始或者完成一次换出。
- 分配程序可以为一个请求选择设备进行分配。

105

讲到这里，我们只是在单个z/OS系统的语境下讨论WLM。在实际中，为了处理复杂的工作负载，客户安装通常采用多个z/OS系统的集群。曾记得我们早些时候关于集群的z/OS系统(系统综合体)的讨论。

WLM特别适合系统综合体环境。它能够跟踪在系统集群和数据共享环境中的所有系统使用情况和工作负载目标完成情况。譬如，WLM可以在z/OS中根据进程相关的资源可用性来快速决定运行哪个批处理作业。

3.5.2 如何使用 WLM?

主机安装时通过建立一组策略把系统性能和商业需求紧密联系起来，并以此影响WLM做出的几乎所有决策。工作负载被赋予目标(如目标平均响应时间)和重要性(即工作负载满足目标对业务的重要程度)。

服务等级协议(SLA)
一种书面协议，关于提供给用户关于计算机安装的服务。

在引入WLM之前，通知z/OS公司的商业目标的唯一方法是由程序员把高层目标翻译成系统能识别的极为技术化的术语。这种翻译工作要求高技能的职员，而且可能会延时，易错，并最终和原始的商业目标冲突。

更进一步，通常很难预计一项系统设置改变后的影响，改变可能是必须的，譬如系统容量升级带来的影响。这可能会导致不平衡的资源分配，使工作失去关键的系统资源。这种操作方法，被称为兼容性模式，在新的工作负载被引入后及多个系统一起管理的情况下变得难于管理。

在目标模式系统操作中，WLM提供更少，更简单，更一致的系统外部组件，它反映了用商业目的中通用的术语来表述的工作目标，WLM和系统资源管理器(SRM)通过持续监控和调整系统，分配资源来达成那些目标。工作负载管理器为管理工作负载分布，工作负载平衡以及为竞争的工作负载分发资源提供了一种途径。

WLM策略通常是基于一个服务等级协议(SLA)上的，这是一个为计算机安装用户提供信息系统服务的书面协议。WLM尝试通过恰当分配，而不是过度使用资源，以努力达到SLA里描述的工作负载需求(响应时间)。同样重要的，WLM最大化系统使用(吞吐量)来获得安装的硬件和软件平台的最大收益。

106

3.6 I/O 和数据管理

系统中几乎所有工作都涉及数据的输入和输出。在主机上，通道子系统管理I/O设备的使用，如硬盘，磁带和打印机。操作系统必须把一个特定任务的数据和某个设备联系起来，并管理文件分配，放置，监视、迁移，备份，回调，恢复和删除。

这些数据管理的工作可以人工处理也可以通过使用自动化过程。当数据管理自动处理的时候，系统决定目标的放置，并自动管理目标的备份，移动，空间和安全。一个典型的z/OS生产系统既有人工也有自动化过程来管理数据。

根据z/OS系统和它的存储设备配置方式，一个用户或者程序可以直接控制数据管理的许多方面，而在早期的操作系统中，却要求用户来操作。然而，渐渐地z/OS系统依赖于特定安装设置来进行数据和资源管理，以及附加的存储管理产品用于自动化使用存储。z/OS中管理存储主要是通过DFSMS组件，这将在165页上的第5章"数据集操作"中讨论。

3.7 监督系统中的作业执行

为了实现多道程序设计，z/OS要求使用一些监督控制，如下：

- 中断处理

  多道程序设计需要一些技术来实现从一个程序到另一个程序转移控制权，譬如当程序A必须等待一个I/O请求时，这时程序B就可以执行。在z/OS中，该交换是通过中断实现的，中断改变处理器执行指令的顺序。当一个中断发生的时候，系统保存被中断程序的执行状态，分析并处理该中断。

- 创建作业调度单元

  为了识别和跟踪作业，z/OS操作系统用一个控制块表示每个作业单元。有两种类型的控制块表示作业调度单元：任务控制块(task control block，TCB)表示在一个地址空间内执行的任务；服务请求块(service request block，SRB)表示优先级更高的系统服务。

- 调度任务

  在中断执行后，操作系统决定哪一个工作单元(在系统所有的工作单元范围内)已经准备好运行并在其中选出优先级最高的，然后将控制权传递给它。

- 串行化系统资源的使用

  在多道程序设计系统中，几乎任何指令都能被中断，稍后再继续。如果那些指令集合操作或者修改某个资源(例如一个控制块或者一个数据文件)，操作系统必须防止其他程序使用这个资源直到中断程序完成它的资源处理。

  串行化资源通过一些技术实现；队列法和加锁法是最常用的两种(第三种技术被称为闭锁法)。所有用户都可以使用队列法，但是只有授权的程序可以使用加锁法来串行化资源使用。

3.7.1 什么是中断处理?

中断是一个可以改变处理器执行指令顺序的事件。中断可能是计划的(由当前正在执行的程序专门请求)，也可能是非计划的(由事件引发，该事件可能和当前正在运行的程序相关，也可能不相关)。z/OS使用六种类型的中断，如下：

- 访管中断或SVC中断

  此调用在程序发出一个SVC来请求特殊的系统服务时发生。SVC中断当前执行的程序，把控制传递给管理程序，以让它可以执行服务。程序通过宏实现这些服务的请求，比如OPEN宏(打开一个文件)，GETMAIN宏(获取存储)，或者WTO宏(把一条信息发送给系统操作员)。

- I/O中断

  当信号子系统发出状态改变的信号时，该类中断发生。譬如完成了一个I/O操作，发生错误，或者一个I/O设备如打印机准备好工作。

- 外部中断

  几种事件的发生会引起这种中断，例如一个时间间隔到期，操作员按下了控制台上的中断键，或者处理器接收到了来自另一个处理器的信号。

108

- 重启中断

  当操作员在控制台选择了重启功能或者一个处理器接收到了来自另一个处理器的一个重启SIGP(信号处理器)时，该类中断发生。

- 程序中断

  这类中断是由程序错误(例如，程序试图执行一个非法操作)，页错误(程序引用了一个不在实存中的页)，或由监控事件的请求所引起的。

- 机器检查中断

  这是由机器故障引起的。

当中断发生的时候，硬件保存被中断程序的相关信息，如果可能的话则使处理器无法再处理更多同类的中断。硬件然后把控制传递给适合的中断处理程序。在这个过程中，程序状态字(PSW)是一个关键的资源。

如何使用程序状态字？

程序状态字(program status word，PSW)是处理器中一个128位的数据区，和许多其他类型的寄存器(控制寄存器，计时寄存器和前缀寄存器)一起向硬件和软件提供至关重要的细节信息。当前的PSW包括了下一个程序指令的地址和正在执行的程序的控制信息。每个处理器都只有一个当前的PSW。因此，一个处理器上某一时刻只能有一个任务在执行。

PSW控制了提供给处理器的指令的顺序，显示和当前运行程序相关的系统状态。尽管每个处理器都只有一个PSW，为了理解中断处理，来看看三种类型的PSW还是很有用的。

- 当前PSW
- 新PSW
- 原PSW

当前PSW显示即将要执行的指令。它还显示了处理器是否能处理I/O中断，外部中断，机器检查中断及特定的程序中断。当处理器能处理时，这些中断能发生。否则，系统将忽视或者挂起这些中断。

这6类中断中的每类都有一个相关的新PSW和原PSW。新PSW包括了能够处理相关中断的程序的地址。当中断发生的时候，如果处理器能够实现中断，系统就通过以下方法切换PSW：

109

1. 把当前的PSW存储在和发生中断类型相关的原PSW中
2. 把和发生中断类型相关的新PSW中的内容载入到当前PSW中

当前PSW用于指定将要执行的下条指令，现在它包含了合适程序的地址来处理中断。如此以来，此切换就能够把控制传递到合适的中断处理程序了。

寄存器和PSW

主机体系结构提供了寄存器，用来跟踪事件的发生。例如PSW是用来包含当前活动程序执行时需要的信息。主机还提供了其他一些寄存器，如下：

- 访问寄存器用来指定数据被找到的地址空间。
- 通用寄存器用来寻址内存中的数据，也被用于存放用户数据。
- 浮点寄存器用来存放浮点形式的数值型数据。
- 控制寄存器由操作系统自己使用，例如用来指向转换表。

相关阅读：IBM出版物《z/Architecture Principles of Operation》描述了用于切换系统状态的硬件设备，包括了CPU状态，控制模式，PSW以及控制寄存器。可以在z/OS互联网知识库网站上找到本书和其他相关的出版物：

110 http://www.ibm.com/servers/eserver/zseries/zos/bkserv/

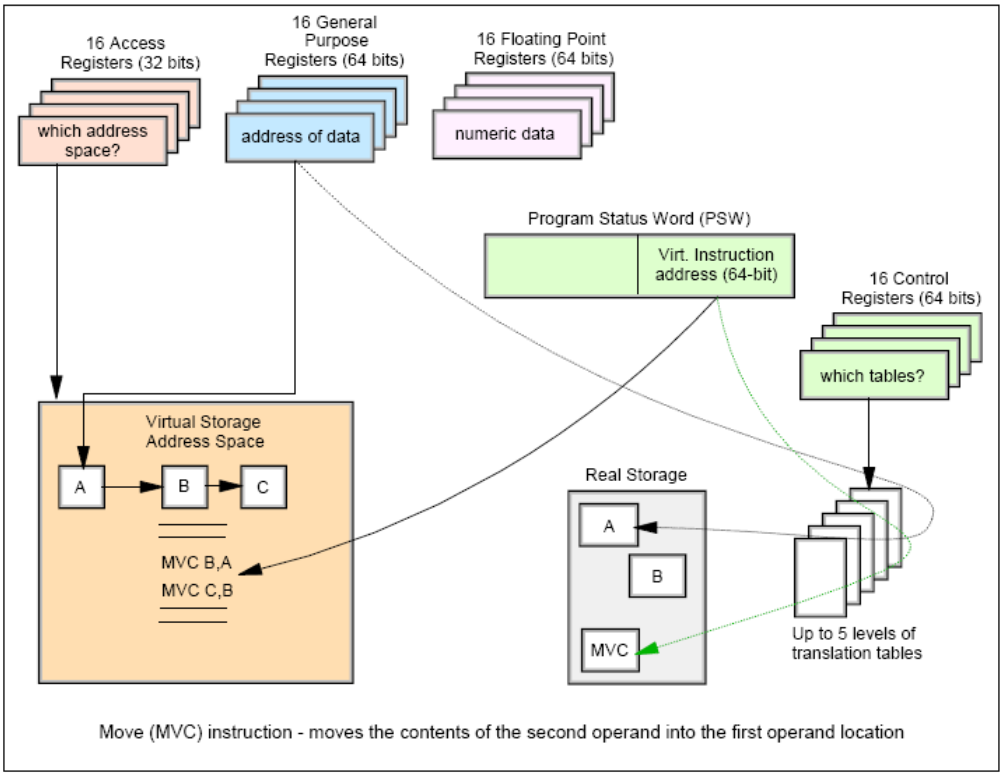


图3-10 寄存器和PSW

3.7.2 创建任务调度单元

在z/OS中，任务调度单元由两种控制块表示：

- 任务控制块(TCB)

  TCB代表在一个地址空间内执行的任务，譬如用户程序和支持用户程序的系

统程序。

- 服务请求块(SRB)

  SRB表示执行系统程序的请求。当某地址空间中侦测到某事件影响到另一个地址空间时，系统创建SRB。它为不同地址空间之间通信提供了一种机制。

TCB是什么？

TCB是控制块，用来表示一个任务，如运行在地址空间里的程序。TCB包含了运行任务的信息，譬如它创建的存储区域的地址。不要把z/OS术语TCB和UNIX数据结构进程控制块(process control block，PCB)搞混淆了。

111

TCB为响应一个ATTACH宏而创建。通过执行ATTACH宏，用户程序或系统程序(以下称为调用任务)开始执行ATTACH宏中指定的程序，该程序将作为调用任务的子任务执行。作为一个子任务，该程序可以竞争处理器时间，使用已经分配给调用任务的某些资源。

区域控制任务(region control task，RCT)负责为换入换出作业准备地址空间，是地址空间中优先级最高的任务。地址空间中的所有任务都是RCT的子任务。

SRB是什么？

SRB是一个控制块，它代表在特定地址空间内执行一个特定功能或服务的程序。一般地，SRB在一个地址空间正在执行并且发生了影响另一个地址空间的事件时创建。

完成以上所述功能或服务的程序称为SRB程序；初始化此过程称为调度一个SRB；SRB运行的操作模式称为SRB模式。

SRB定义了系统中的一个任务单元，这一点和TCB很像。和TCB不同，SRB不能”拥有”存储区域。SRB程序可以获取，引用和释放存储区域，但该存储区域必须为TCB所拥有。在一个多处理器环境中，SRB程序被调度后，可以被另一个处理器处理并且可以和调度程序同时运行。调度程序可以和SRB程序并行执行，继续完成它的其他处理。正如前面提到的，SRB为运行在z/OS上的程序提供了一种异步的地址空间之间的通信方法。

只有运行在具有高优先级的模式下才能创建SRB，该模式称为超级用户状态。这些授权程序获取存储空间，利用目标地址空间和指向处理请求的代码块的指针来初始化控制块。创建SRB的程序接着发布SCHEDULE宏，指定SRB是否有全局(系统范围内的)或本地(一个地址空间范围内)优先权。系统把SRB放在合适的调度队列里，在那里它会保持不变直到它成为队列里具有最高优先权的任务。

112

不管SRB实际运行的地址空间是哪个，具有全局优先权的SRB有比任何地址空间有更高的优先权。而有本地优先权的SRB和它们运行的地址空间具有相同的优先权，但却比该地址空间内任何TCB都有更高的优先权。全局或本地优先权的分配取决于请求的”重要性”。譬如处理I/O中断的SRB被赋予全局优先权，这样能最小化I/O延迟。

相关阅读：在IBM出版物《z/OS MVS Authorized Assembler Services Guide》里描述了SRB的使用。可以在z/OS互联网知识库网站上找到本书和其他相关出版物：

http://www.ibm.com/servers/eserver/zseries/zos/bkserv/

3.7.3 可抢占对比不可抢占

中断发生后哪个程序接收控制权取决于被中断的任务单元是否是可抢占的。如果是可抢占的，操作系统决定接下来执行哪个任务单元。即系统决定系统中所有任务单元中哪一个具有最高的优先权，然后把控制权传递给它。

不可抢占任务单元也可以被中断，但是在中断处理完成后必须重新接收控制权。例如，SRB通常是不可抢占的[6]。因此，如果由不可抢占SRB表示的程序被中断后，它会在中断处理完毕时接收到控制权。相反地，由TCB表示的程序(譬如用户程序)经常是可抢占的[7]。如果它被中断，中断处理完成后控制权将返回给操作系统。z/OS接着决定在所有就绪任务中优先执行哪一个。

3.7.4 调度器怎么工作？

在一些情况下新的任务将被选定，这些情况包括：一个任务被中断或者成为非调度的，或者一个SRB完成或被挂起(即SRB因为请求的资源得不到满足而延迟执行)。

在z/OS中，调度器组件负责把控制权传递给就绪的任务单元中优先级最高的那一个。调度器按照下面的顺序调度任务：

1. 特殊的出口程序

   这是一些由于系统的特定情况而具有较高优先级的特殊程序的出口程序。例如，多处理器环境中的某个处理器失败了，那么系统将调用特殊出口程序来完成备用CPU恢复失败处理器上正在执行的工作。

2. 拥有全局优先权的SRB

3. 按优先级排序的就绪的地址空间

   如果地址空间被换入并且不等待某些事件的发生，那么它就是执行就绪的。地址空间的优先级是由用户或者系统指定的调度优先级决定的。

   在选择了最高优先级的地址空间后，z/OS(通过调度器)首先调用具有本地优先权的分配给该地址空间的SRB，然后再调用该地址空间中的TCB。

[6] 调用程序可以指定 SRB 为可抢占的，允许具有相同或更高优先级的任务可以使用处理器。同时，客户端 SRB 和 enclave SRB 也是可抢占的。这些话题不在本书的讨论范围之内。

[7] TCB 在执行一个 SVC 时是不可抢占的。

如果系统中没有就绪的任务，z/OS就呈现出一种状态，称为使能等待(enabled wait)，直到新的任务进入系统。

z系列硬件的不同模式可以有1到54个不等的中央处理器(CP)[8]。每一个CP可以同时执行指令。调度优先级决定了就绪的地址空间什么时候被调度。

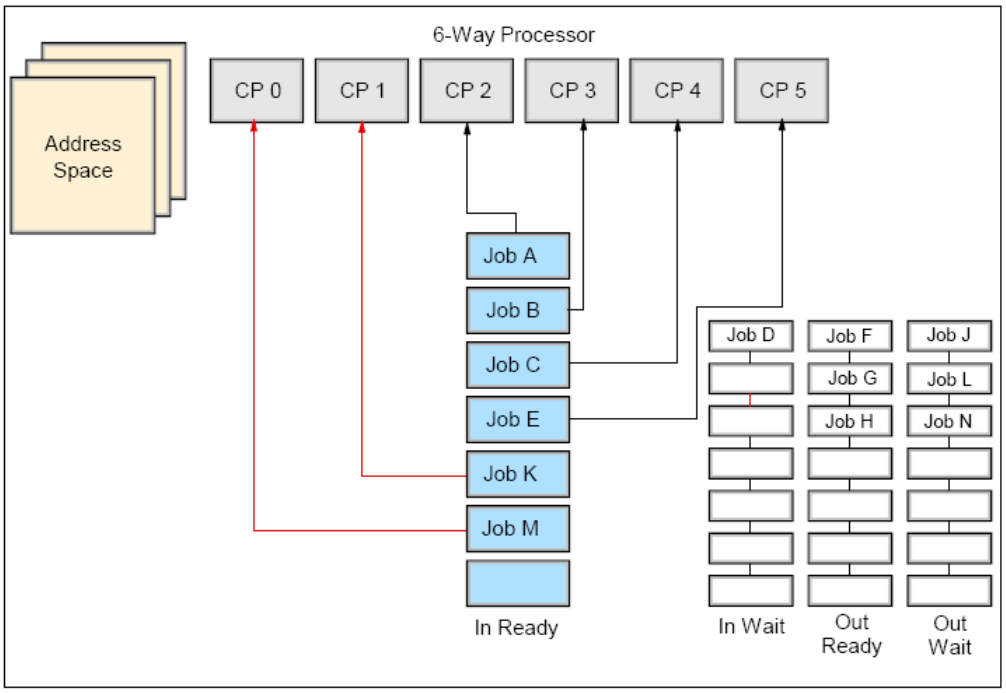


图3-11 调度工作

地址空间可以存在于以下4个队列中的任何一个队列：

- 内就绪队列(IN-READY)– 在中央存储中，等待调度
- 内等待队列(IN-WAIT)– 在中央存储中，但是在等待一些事件完成
- 外就绪队列(OUT-READY)– 处于执行就绪状态，但却已经被换出
- 外等待队列(OUT-WAIT)– 被换出并且等待一些事件完成

只有内就绪队列中的任务可以被选中进行调度。

3.7.5 串行化使用资源

在多任务，多处理环境中，资源串行化是用来，在资源被多于一个的应用程序使用时，协调资源访问的技巧。改变数据的程序需要对数据进行独占式访问。否则，如果几个程序同时对相同的数据进行更新，数据就会被破坏(也称为数据完整性的损失)。从另一方面来说，那些只需要读取数据的程序可以同时共享对同一数据的访问。

[8] IBM z9-109 模式 S54 可以定制高达 54 个 CP(模型的数字和服务器能定制的处理器的最大数目一致的)。

串行化使用资源最常用的技术是排队和加锁。这些技术允许在多程序或多处理环境中对被多个用户需要的系统资源进行有序访问。在z/OS中，队列法是由全局资源串行化组件管理的，而加锁法是由监控组件里的多个锁管理程序管理的。

什么是全局资源串行化？

全局资源串行化组件处理运行在z/OS中的程序对资源的请求。全局资源串行化对资源的访问串行化以保持他们的完整性。通过安装可以用通道—通道(CTC)适配器把两个或多个z/OS系统联系起来，形成一个GRS联合体，从而实现系统中共享资源访问的串行化。

当系统请求访问一个可再度使用的资源时，访问要求也许是唯一或者共享的。当全局资源串行化程序把共享访问赋给一个资源时，希望独占资源的用户就不能访问该资源了。相似的，当全局资源串行化程序把唯一访问赋给资源时，所有其他的对该资源的请求都必须等待直到当前的唯一请求者释放该资源。

什么是队列化？

队列化是指运行在z/OS上的程序请求访问一系列可再次使用的资源。队列化是通过ENQ(enqueue)和DEQ(dequeue)宏实现的，这两个宏对所有运行在系统上的程序都可用。对被多个z/OS系统共享的设备来说，队列化是通过RESERVE和DEQ宏完成的。

通过ENQ和RESERVE，程序定义了一个或多个资源的名字，请求对资源的唯一或共享控制。如果要修改资源，程序必须请求唯一控制；如果不修改资源，程序可以请求共享控制，允许资源被其他不要求唯一控制的程序共享。如果资源不可用，系统挂起请求程序直到资源可用。如果程序不再要求对某个资源的控制，就使用DEQ宏来释放资源。

加锁法是什么？

通过加锁法，系统通过授权程序或在系统集群中通过处理器使资源串行化。锁是存储区里一块简单的名字域，显示了资源是否正在被使用，如果是，还显示谁在使用。在z/OS中，有两种锁：全局锁和局部锁，前者用在和多个地址空间相关的资源上。后者用于分配给一个特定的地址空间。全局锁被不可再用或不可共享的程序和大量资源使用。

要使用被锁保护的资源，程序必须首先请求对那个资源的锁。如果锁当前不可用(即已经被其他程序或另一个处理器使用)，请求锁的程序或处理器采取的行为取决于锁是旋转锁还是挂起锁：

- 如果旋转锁不可用，正在请求该锁的处理器会继续测试直到其他处理器释放了该锁。锁一被释放，正在请求的处理器就立即得到该锁，因此也就得到了被保护的资源的控制权。大多数全局锁是旋转锁。旋转锁的持有者碰到大多数中断都会失去对锁的控制(当持有者即将要被中断，它可能永远也得不到放弃锁的控制)。
- 如果挂起锁不可用，请求锁的任务单元被延迟直到锁可用。在请处理器上的其他任务被调度。所有的局部锁都是挂起锁。

您也许会问如果两个用户互相请求被对方持有的锁，会发生什么情况。它们会互相等待对方先释放锁，从而陷入僵局？在z/OS中，这种情况称为死锁。幸运的是，z/OS锁机制能预防死锁。

为了预防死锁，锁以层次结构安排，处理器或程序只能无限制地请求比它当前持有锁所在层更高层的锁。例如，如果处理器1持有锁A，请求锁B；处理器2持有锁B请求锁A，这个时候死锁就发生了。

116

如果请求锁必须以层次顺序进行，这种情形就不会发生。在这里例子中，假定在层次关系中，锁A在锁B之上。那么处理器2就不能无限制地申请锁A因为它持有锁B。它必须首先释放锁B，请求锁A，然后再请求锁B。正因为这种层次关系，死锁不会发生。

相关阅读：IBM出版物《z/OS Diagnosis Reference》半包含了一张表，列出了z/OS锁的层次关系，书中还有一些关于它们的描述和特性。在下面的z/OS 因特网库网页上，能找到这本书及相关出版物。

http://www.ibm.com/servers/eserver/zseries/zos/bkserv

3.8 z/OS 的标志性特征

z/OS的标志性特征总结如下：

- z/OS中地址空间的使用带来了很多优点：不同地址空间中私有空间的隔离提供了系统安全性，然而每个地址空间也提供所有地址空间都能访问的公共区域。
- 系统设计成能够保持数据完整性，无论系统中的用户数量有多大。z/OS阻止用户随意访问或改变任何系统的对象，包括用户数据，除非使用系统提供的强制遵守授权规定的接口。
- 系统设计成能完成大量并发的批处理作业，而不需要客户从外部管理数据负载均衡或完整性问题，这些问题可能是由对给定数据集的同时冲突性的使用引起的。
- 安全性设计覆盖了从系统功能到一般简单文件的各个层面。安全性可以整合到应用程序、资源和用户描述文件。
- 系统允许同时存在多个通信子系统，这为同时运行多个完全不同的面向通信的应用程序(包括每个应用程序的测试版，生产版和后备版)提供了非同寻常的灵活性。譬如，多个TCP/IP栈可以同时操作，每个栈都有不同的IP地址并为不同的应用程序提供服务。
- 系统提供了广泛的软件恢复级别，使生产环境中的系统几乎不必进行非计划的系统重启。系统接口允许应用程序提供自己的恢复级别。简单的应用程序很少使用这些接口，通常复杂的应用程序才会使用到。

117

- 系统设计成日常管理多个完全不同的任务，自动平衡资源以满足系统管理员确定的生产要求。
- 系统设计成日常管理大量I/O配置，这些配置可能扩展成数千个硬盘，多个自动磁带库，多台大型打印机，大量网络终端等等。
- 系统可以由一个或多个操作员终端控制，也可以从应用程序编程接口(API)

控制，API允许自动化例行的操作员功能。

- 操作员接口是z/OS的关键功能。它提供状态信息，异常情况消息，工作流程控制，硬件设备控制，并允许操作员管理特殊的恢复情形。

3.9 z/OS 的附加软件产品

z/OS系统通常包含附加的收费的软件产品，它们是构建实际工作系统时所必需的。例如，一个z/OS生产系统通常包含一个安全管理器产品和一个数据库管理器产品。我们谈到z/OS时，人们通常假定包括了这些附加产品。这些从讨论的上下文来看，一般会很明显，但有时很必要问一问某个特定的功能是"基本z/OS"的一部分还是附加产品。IBM把它自己的附加产品称为IBM特许程序(*IBM licensed program*)。

特许程序(licensed program) 一种附加的，标价出售的软件产品，而不是基本z/OS的一部分。

大量独立软件供应商(ISV)提供了大量产品，这些产品功能有所变动但是大多相似，如安全管理器和数据库管理器，可以从大量特许程序中进行选择来完成一项任务，这种能力提高了z/OS操作系统的灵活性，并允许主机IT组根据他们公司的需要裁剪运行的产品。

本书中我们不打算列举所有的z/OS特许程序(存在数百种)；只介绍一些常用的：

- 安全系统

  z/OS为客户提供了一个架构，用户可以通过添加安全管理产品以增强安全性(IBM的特许程序是资源访问控制工具RACF®)。当然也可以使用非IBM的安全系统特许程序。

- 编译器

  z/OS包含一个汇编器和一个C编译器。其他的编译器，如COBOL编译器，和PL/1编译器是作为单独的产品提供的。

- 关系数据库

  一个例子是DB2。其他类型的数据库产品，如层次型数据库也可以使用。

- 交易处理工具

  IBM提供如下几种：

  - 客户信息控制系统(CICS)
  - 信息管理系统(IMS)
  - z/OS上的WebSphere应用服务器

- 排序程序

  在批处理中要求对海量数据进行快速、高效的排序。IBM和其他软件厂家都提供成熟的排序产品。

- 大量不同的实用程序

  例如，系统显示和查询工具(SDSF)程序，它是本书中我们广泛使用的用来观察批处理作业输出的一个许可产品。并不是每个安装系统都购买SDSF；还有其他可选的产品。

另外，从各种独立的软件供应商(行业中一般称为ISV)那里可以获得大量其他产品。

3.10 z/OS 的中间件

中间件
提供操作系统具有的一些重要功能的软件。

中间件通常指介于操作系统和终端用户或终端用户应用程序之间的软件。它提供操作系统不具有的一些重要功能。这个术语通常指主要的软件产品，如数据库管理器，交易监控器，网络服务器等。"子系统"是经常用于这类软件的另外一个术语。这些通常是特许程序，但也有值得注意的例外，如HTTP服务器。

z/OS是使用中间件产品和功能的基础。运行各种不同的中间件功能是很常见的，其中一些功能可以有多个产品实例实现。对广泛的工作负载(混合了批处理任务，交易处理，网络服务，数据库查询更新等等)的日常使用是z/OS的一大特征。

典型的z/OS中间件包括：

- 数据库系统
- 网络服务器
- 消息队列和路由功能
- 交易处理器
- Java虚拟机
- XML处理功能

中间件产品通常包括一个应用程序编程接口(API)。在一些情况下，应用程序被编写成完全在该中间件API的控制下运行，而在其他一些情况下，它仅用于一些特定用途。主机上的中间件API一些例子有：

- WebSphere产品套件，它提供一个完整的API，能够方便地跨越多个操作系统。在这些产品中，WebSphere MQ提供跨平台的API和平台间的消息通信。
- DB2数据库管理产品，它提供一个API(用SQL语言表示)，可以和许多不同的语言和应用程序一起使用。

网络服务器可以被视为中间件，网络编程(网页，CGI等等)大量使用网络服务器提供的接口和标准，而不是操作系统的接口。Java是另一个例子，应用程序被编写运行在一个Java虚拟机(JVM™)[9]上，而很大程度上独立于所使用的操作系统

3.11 z/OS 和 UNIX 的一个简单比较

如果比较一下z/OS和UNIX，我们会发现什么？在很多情况下，我们发现两个操作系统之间很多概念可以被任意一方的用户理解，虽然它们的术语不同。

对于有经验的UNIX用户，表格3-1提供了其熟悉的计算术语和概念的少量样例。作为z/OS的新用户，z/OS的许多术语可能听起来会很陌生。然而， 当您在学习该课程的过程中，z/OS含义将得到解释，您将发现很多UNIX的元素在z/OS中都

[9] JVM 和 z/VM 建立的虚拟机不相关。

有相似体。

用户从UNIX转移到z/OS一个最大的不同就是您只是很多用户中的一个。在一个从UNIX系统到z/OS环境的迁移过程中，用户通常会问一些问题，比如"能告诉我根密码吗因为我要做…… "或者"您能不能改变这个或那个，然后重启系统？"对新z/OS用户来说，很重要的一点是记住在同一个系统上潜在地有数以千计的用户在上面活动，因此用户行为的范围以及z/OS和z/OS UNIX的重启都得到很仔细的控制，以避免对其他用户和应用程序造成负面影响。

120

在z/OS中，不存在单个的根密码或根用户。用户ID在z/OS UNIX系统服务之外。在z/OS系统中，用户ID维护在UNIX及非UNIX功能共享的安全数据库中，甚至可能和其他z/OS系统共享。典型情况下，一些用户ID拥有根权限，但每个独立的用户ID拥有独立的密码。同样，另外一些用户ID并没有根权限，但当环境需要的时候，它们可以切换到"根"。

为了内存中的数据能被进程间共享，z/OS和UNIX都提供了API。在z/OS中，一个用户可以直接通过跨内存服务访问另一个用户的地址空间。相似地，UNIX有共享内存功能的概念，在UNIX上没有特殊权限就能使用这些。

然而，z/OS跨内存服务要求发起的程序有特殊权限，这由授权程序工具(APF)控制。这种方法允许高效安全地访问其他用户的数据，和访问自己的数据，这些数据是为了便利和为了快速安全地和诸如交易管理器以及数据库管理器等服务通信，而存储在其他地址空间中的。

121

表3-1　UNIX和z/OS术语和概念的对照

术语或概念	**UNIX**	**z/OS**
启动操作系统	导入系统	IPL(初始化程序装载)系统
给系统的每个用户分配的虚拟存储	用户在硬件和操作系统限定内，得到所需引用的任何虚拟存储	每个用户获得一个地址空间其地址可以扩展到2GB(甚至16EB)的虚拟存储，尽管 其存储的一部分包含了对所有用户公用的系统代码
数据存储	文件	数据集(有时称为文件)
数据格式	面向字节；数据组织由应用程序提供	面向记录；通常是 80 字节的记录，反映了传统的穿孔卡片图像
系统配置数据	/etc 文件系统控制特性	PARMLIB 中的参数控制系统是如何启动的以及地址空间的行为
脚本语言	shell 脚本，Perl， awk 和其他语言	CLISTS(命令列表)和 REXX execs
执行任务的最小单元	线程。系统核心支持多线程	任务或服务请求块(SRB)。z/OS 基础控制程序(BCP)支

		持多任务和多个 SRB
长时间运行的任务单元	守护进程(daemon)	一个已经开始的任务或一个长时间运行的任务；常是 z/OS 的一个子系统
查找顺序，系统用它来查找程序运行	依据用户的 PATH 环境变量(包含一个用于查询的目录列表)从文件系统装载程序	系统查询如下的库以装载程序：TASKLIB，STEBLIB， JOBLIB，LPALST 和 linklist
操作系统提供的交互工具(不包括可以在后来加入的交互应用程序)	用户登录系统并在 shell 环境下执行 shell 会话。他们可以发送 rlogin 命令或者 telnet 命令连接系统。每个用户可以同时打开多个登录会话	用户通过 TSO/E 和它的面板驱动接口 ISPF 登录系统。一个用户 ID 一次只能有一个活动的登录会话。 用户也使用 telnet, rlogin 或 ssh 登录 z/OS UNIX shell 环境。
编辑数据或代码	有很多编辑器，例如 vi, ed, sed 和 emacs	ISPF编辑器[a]
输入/输出数据源和目的	stdin 和 stdout	SYSIN 和 SYSOUT SYSUT1 和 SYSUT2 用于实用程序 SYSTSIN 和 SYSTSPRT 供 TSO/E 用户使用
管理程序	ps shell 命令允许用户查看进程和线程，并通过 kill 命令结束作业	SDSF 允许用户查看和终止他们的任务

a.系统也有一个TSO编辑器，尽管很少使用。例如，当通过TSO发送电子邮件时，SENDNOTE exec打开一个TSO EDIT会话，允许用户写一封电子邮件。

3.12 总结

操作系统是管理计算机系统内部工作的程序的集合。本课程中讲授的操作系统是z/OS，这是一种广泛使用的主机操作系统。z/OS操作系统对多道程序设计和多处理技术的使用，以及它能够访问和管理大量的存储和I/O操作的能力，使它自身特别适合运行主机工作负载。

虚拟存储的概念是z/OS的核心。虚拟存储是由体系结构创建的假象，这样系统看起来比实际拥有更大的存储。虚拟存储通过使用表把虚存页映射到中央存储器页框或辅存页槽上而实现。实际上，只有需要的程序部分被载入到内存中运行。z/OS

把地址空间中不活动的片保留在辅存中。

z/OS在地址空间上构建，地址空间是虚拟存储的地址的范围。每个z/OS用户都得到一个地址空间，里面包含了相同范围的存储地址。z/OS中地址空间的使用，允许隔离不同地址空间的私有区域，以获取系统安全，同时也允许通过每个地址空间都可以访问的公共区域来实现程序和数据在地址空间中的共享。

普遍来说，术语中央存储器，实际内存，实际存储及主要存储被交替使用。同样，虚拟内存和虚拟存储表达了同一个意思。

需要用来支持一个地址空间虚存的中央存储的数量，取决于运行的应用程序工作集的大小，而该大小是随着时间不断变化的。用户不能自动访问地址空间中的所有虚拟存储。如果请求使用一定范围的虚拟存储，系统会首先检查空间大小的限制，然后创建需要的页表条目，最终得以创建请求的虚拟存储。

在z/OS和zSeries主机上运行的程序可以使用24位、31位或64位寻址模式(如果需要的话还可以在三者之间转换)。程序可以使用具有16位，32位，或64位操作数的指令，如果有需要，也可以在三者之间切换。

主机操作系统很少提供完整的操作环境。它们依赖特许程序来实现中间件和其他功能。许多供应商包括IBM，提供了中间件和各种各样的实用程序产品。

中间件是一个相对比较新的术语，它同时包含了几个概念。中间件的一个共同特征是能够提供编程接口，应用程序可以全部(或部分)使用接口编写。

本章关键术语			
地址空间 (address space)	寻址能力 (addressability)	辅助存储(auxiliary storage)	中央存储(central storage)
控制块(control block)	动态地址转换 (dynamic address translation，DAT)	页框(frame)	输入/输出 (input/output，I/O)
特许程序 (licensed program)	中间件(middleware)	多道程序设计 (multiprogramming)	多处理技术 (multiprocessing)
页/页交换 (page/paging)	页面窃取(page stealing)	服务等级协议(service level agreement，SLA)	页槽(slot)
交换(swapping)	虚拟存储(virtual storage)	工作负载管理 (workload management，WLM)	z/OS

3.13 复习题

为了测试您对本章中材料的理解，完成下列问题：

1. z/OS和单用户操作系统有什么区别？给出两个例子。
2. z/OS的设计利用了主机体系结构的哪些方面？它是在哪年发布的？
3. 列出z/OS使用的三种主要的存储类型。

4. 虚拟存储中的”虚拟”指什么？

5. 匹配下列术语：

   a.页 ___辅助存储
   b.页框 ___虚拟存储
   c.页槽 ___中央存储

6.在z/OS系统中WLM扮演了什么样的角色？

7.列举z/OS操作系统的几点标志性特征。

8.列举三种软件产品，它们和z/OS一起能提供一个完整的系统。

9.列举z/OS和UNIX操作系统的一些相同点和不同点。

10.下面的内容哪个/些不是z/OS系统的中间件产品？

   a.网络服务器
   b.交易管理器
   c.数据库管理器
   d.辅助存储管理器

3.14 思考题

为了深入探索z/OS概念，可以讨论以下一些领域问题：

125

1. z/OS提供了64位寻址。假设您要使用这个能力来使用一个很大的虚拟存储区域。您将采用合适的编程接口来获取一个30G字节大小的虚拟存储，您也许会写一个循环来为应用程序初始化这块区域。这些操作可能会产生什么副作用呢？该设计什么时候是可行的？需要考虑什么外部环境？在另一个平台上，如UNIX上会有什么不同？

2. 对应用程序所有者来说，为什么把程序和数据块从线以下迁移到线上时这么复杂？在不破坏和已经存在程序的兼容性的前提下，该怎样完成该操作？

3. 一个应用程序可以采用24位，31位或64位的寻址模式编写。程序员如何选择呢？如果是一个高级语言呢？如果是汇编语言呢？您已经开始使用了ISPF。它使用了什么寻址模式？

4. 更多的实存会让系统运行得更快吗？什么情况下表示系统需要更多实存？什么时候不再需要更多实存？什么可能会改变这个状况？

5. 如果当前的z/OS只运行在z/Architecture模式，为什么我们还要提及24位，31位和64位操作？为什么提到32位的操作数？

6. 为什么把分配虚拟内存设计的那么复杂？为什么不在第一次创建地址空间的时候，为所有的虚拟存储创建所有必需的页表？

126

7. 为什么我们需要特许程序？为什么不简单地把所有软件产品都包括到操作系统中？

4

第 4 章 TSO/E，ISPF 和 UNIX：z/OS 的交互工具

目标：当您使用z/OS操作系统工作时，您需要知道它的终端用户接口。这里面主要的接口有TSO和它的菜单驱动式接口ISPF。您可以通过这些接口登录系统，运行程序，以及操作数据文件。您还需要了解z/OS实现与UNIX接口的交互工具，我们称之为z/OS UNIX系统服务，或简称为z/OS UNIX。

学习完本章后，您将可以：

- 登录到z/OS。
- 在TSO READY提示符下运行程序。
- ISPF的菜单选项导航。
- 使用ISPF编辑器修改数据集。
- 使用z/OS UNIX接口，包括z/OS UNIX shell命令。

127

4.1 我们如何与 z/OS 交互？

我们已经提到过，z/OS擅长处理批处理作业——在后台运行的工作负载，几乎不需要与人交互。但是，z/OS不仅仅是一个批处理系统，它也是一个交互式系统。这里所说的‘交互’指的是终端用户(有时会有几十万用户并发使用)可以通过直接交互方式使用系统，比如通过命令或菜单形式的用户接口来访问系统。

z/OS提供若干允许用户直接和操作系统交互的工具。本章将总体介绍各个工具：

- 第128页的‘TSO概述’介绍了如何登录z/OS，描述了一部分基本的TSO命令，这些命令是核心操作系统的一部分。与z/OS的这种交互方式被称为在本机模式下使用TSO。
- 第133页的‘ISPF概述’介绍了ISPF菜单系统，有很多用户完全依靠使用ISPF菜单系统在z/OS上执行工作。ISPF菜单列出了在线用户最常使用的功能。
- 第151页的‘z/OS UNIX交互接口’探究了一些z/OS UNIX shell和实用程序。这个工具可以让用户编写或调用shell脚本和实用程序，以及使用shell编程语言。

本章最后的实际操作练习可以帮助学生更好地了解这些重要的工具。

4.2 TSO 概述

登录
用户用于开始终端会话的程序。

分时选项/扩展(TSO/E)允许用户创建一个和z/OS系统交互的会话。TSO[1]提供一个单用户登录功能和z/OS的基本命令提示接口。

很多用户都通过菜单驱动的接口使用TSO工作，这叫做交互式系统生产工具(ISPF)。这些菜单和面板的集合提供了大量功能来协助用户在系统上处理数据文件。ISPF用户包括系统程序员，应用程序开发人员，管理员和其他访问z/OS的人员。总体来说，TSO和ISPF让经验水平不同的用户更简单地与z/OS系统交互。

128

3270 模拟器
该软件的使用使一个客户端模拟一个 IBM 3270 显示器站或打印机，并通过它使用一个主机系统的功能。

在z/OS系统中，每个用户都有一个经授权的用户名和密码用来登录TSO。登录到TSO需要3270显示设备或更常见的PC上运行的TN3270模拟器。

在TSO登录期间，系统在用户3270显示设备或TN3270模拟器上显示TSO登录屏幕。登录屏幕和Windows登录面板的用途一致。

z/OS系统程序员经常修改特定文本的布局和TSO登录面板信息来更好地适应系统用户的需求。因此，本书中的截屏可能会和您在真正的生产系统上看到的不一样。

图4-1是TSO登录屏幕的典型例子。

[1] 大多数 z/OS 用户将 TSO/E 视为“TSO”，本书中也这样叫。同样，“用户”一词与“终端用户”同义。

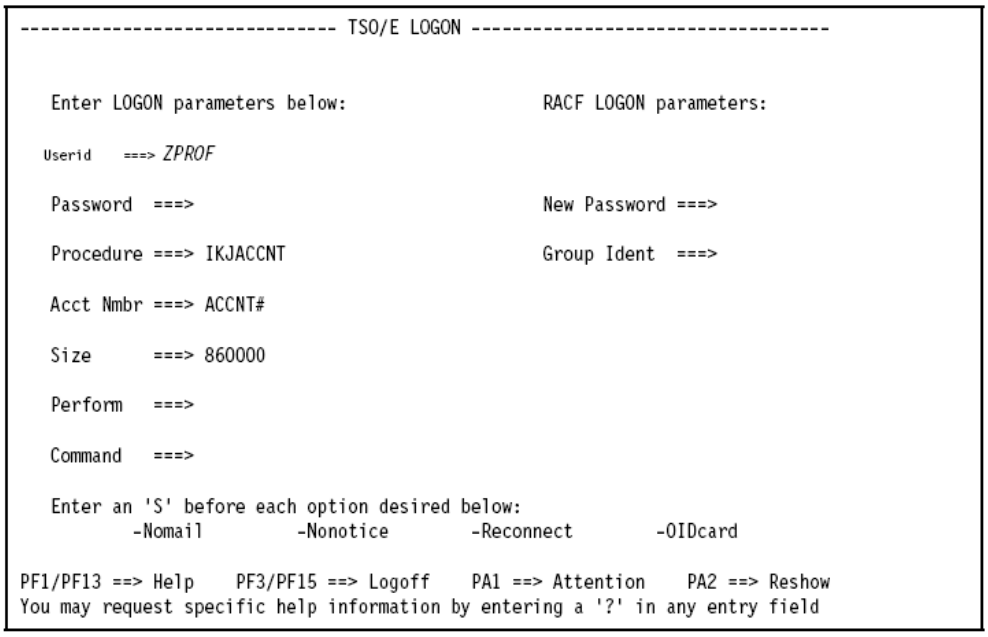


图4-1 典型TSO/E登录屏幕

本书使用的很多截屏示例显示了程序功能(PF)键设置。因为在z/OS中客户化PF键来满足用户需求很是常见的，本书中展示的键分配可能和您系统中的键配置不尽相同。

在本书的139页的4.3.1章节"本课程使用的键盘映射"中提供了本书使用的PF键分配列表。

4.2.1 数据文件术语

z/OS文件称为数据集。在您可以向数据集写入数据之前，必须在磁盘上保留出数据集的空间。用户需要指定数据集空间大小及其格式。

记录
作为一个单元的一组相关数据，单词或者字段。

在主机上新建一个文件的操作比在PC上新建文件稍微有点复杂。这并不是陈旧的科技，两者区别的存在有很多原因。一个不同之处就在于z/OS传统地使用面向记录的文件系统。相反，PC操作系统(微软Windows，Linux，Mac OS等)使用字节流文件系统。

那不同之处在哪里呢？在字节流文件系统中，文件只是二进制位顺序流的集合，会有一些特殊符号告诉计算机在哪里一行(或一条记录)结束，和在哪里新的一行开始。在一个面向记录的文件系统里，文件在磁盘上是按单独的记录组织起来的。在面向记录的文件里，您明确定义记录的属性和大小，所以不需要特殊的终止行符号，这样可以帮助节省系统资源。顺便提一下，z/OS也支持特殊的字节流文件系统，叫HFS或zFS，我们将在5.13节189页的'z/OS UNIX文件系统'中讨论它们。

这里有一些分配数据集使用的术语。

卷序列号 包含6个字符，是磁盘或磁带的卷名，比如TEST01。

设备类型 一个磁盘设备的型号或类型，比如3390。

组织 处理数据集的方式，比如顺序。

记录格式 在块中存储的数据叫做记录，格式为定长或变长。

记录长度 每条记录的长度(字符数目)。

块大小 为了节省空间，记录连接在一起形成块，块大小是以字符为单位的块长度。

分区 用于存储数据的空间的分配。当首次分配量(primary extent)满了，系统自动分配更多分区，叫二次分配区。

空间 磁盘空间以块，磁道或柱面为单位分配。

4.2.2 在本机模式下使用 TSO 命令

很多z/OS系统倾向于在TSO登录后，将TSO用户会话自动切换到ISPF接口。然而，在这个部分中我们将简单讨论一些基本的TSO命令，这些命令独立于其他补充程序，比如ISPF。用这种方法使用TSO叫做在本机模式下使用TSO。

本机模式
在没有诸如ISPF的补充程序情况下使用TSO。

当一个用户登录到TSO，z/OS系统显示READY提示符来响应用户，并等待输入，如图4-2所示。

```
ICH70001I ZPROF  LAST ACCESS AT 17:12:12 ON THURSDAY, OCTOBER 7, 2004
ZPROF LOGON IN PROGRESS AT 17:12:45 ON OCTOBER 7, 2004
You have no messages or data sets to receive.
READY
```

图4-2 TSO登录READY提示符

READY提示符接受简单的行命令，诸如HELP，RENAME，ALLOCATE和CALL。图4-3展示了一个使用ALLOCATE命令在磁盘上创建一个新数据集的例子。

```
READY
 alloc dataset(zschol.test.cntl) volume(test01) unit(3390) tracks space(2,1)
recfm(f) lrecl(80) dsorg(ps)
READY
listds
 ENTER DATA SET NAME -
zschol.test.cntl
 ZSCHOL.TEST.CNTL
  --RECFM-LRECL-BLKSIZE-DSORG
    F     80    80      PS
 --VOLUMES--
   TEST01
 READY
```

图4-3 通过TSO命令行分配数据集

本机TSO和本机DOS提示符提供的接口很相似。TSO还包含一个很基本的行模式编辑器，这与ISPF提供的全屏编辑器形成对比。

在132页的图4-4是用户可能在READY提示符下输入的行命令的另外一个例子。这里，用户正在输入命令来给数据排序。

131

```
 READY
ALLOCATE DATASET(AREA.CODES) FILE(SORTIN)    SHR
 READY
ALLOCATE DATASET(*)          FILE(SORTOUT)   SHR
 READY
ALLOCATE DATASET(*)          FILE(SYSOUT)    SHR
 READY
ALLOCATE DATASET(*)          FILE(SYSPRINT)  SHR
 READY
ALLOCATE DATASET(SORT.CNTL)  FILE(SYSIN)     SHR
 READY
CALL 'SYS1.SICELINK(SORT)'

 ICE143I 0 BLOCKSET     SORT  TECHNIQUE SELECTED
 ICE000I 1 - CONTROL STATEMENTS FOR Z/OS DFSORT V1R5
             SORT FIELDS=(1,3,CH,A)
 201  NJ
 202  DC
 203  CT
 204  Manitoba
 205  AL
 206  WA
 207  ME
 208  ID
 ***
```

图4-4 使用本机TSO命令来给数据排序

本例中，用户键入一些TSO ALLOCATE命令来给排序程序的工作站分派输入输出。然后用户键入一个单独的CALL命令来运行排序程序：DFSORT，这是IBM可选的软件产品。

每个ALLOCATE命令要求指定以下内容(在DATASET操作数中指定)：

- SORTIN － 本例中是AREA.CODES
- SORTOUT － 本例中是*,表示终端屏幕
- SYSOUT
- SYSPRINT
- SYSIN

在输入输出分配和用户输入CALL命令后，排序程序在用户屏幕上显示结果。如图4-4所示，由SORT FIELDS控制语句决定了结果要按照区域代码排序。比如，NJ(New Jersey)的电话区域代码最小，为201。

132

本机TSO屏幕控制是非常基本的。例如，当一个屏幕充满数据时，三个星号(***)就会显示以指示满屏。这里，您就要按回车键来清屏，让屏幕继续显示剩下的数

据。

4.2.3 TSO 下使用 CLIST 和 REXX

CLIST
将一组命令当成一个命令执行。

有了本地TSO，我们就可以把命令列到一个文件中，叫做命令列表或CLIST(读成‘see list’)，然后就像执行一条命令一样来执行这个命令列表。当您调用一个CLIST，它将依序执行TSO/E命令。CLIST可以用来完成日常任务；使用户可以更有效地使用TSO工作。

例如，132页例4-4中的命令可以组合在一个文件中，叫AREA.COMMND。用户可以只用一句命令来执行CLIST(如下)，来获得相同的结果。

```
EXEC ‘CLIST AREA.COMMND’
```

REXX
TSO 使用的一种解释命令语言。

TSO用户使用CLIST命令语言创建CLIST。和TSO一起工作的另外一种命令语言叫做重构扩展执行器或REXX。CLIST和REXX都提供Shell脚本类型处理。它们都是解释型语言，与编译型语言相对照(虽然REXX也可以被编译)。在本书第277页第9章”在z/OS上使用编程语言”中将更详细的讨论CLIST和REXX。

一些TSO用户写一些CLIST和REXX程序实现某种功能，不过这些更多地是由ISPF功能或各种软件产品来实现。CLIST程序是z/OS所特有的，而REXX语言在很多平台上都可以使用。

4.3 ISPF 概述

ISPF
z/OS 提供给用户对许多最常需要的功能进行访问的工具。

登录到TSO后，用户通常会访问ISPF菜单。实际上，很多用户完全依靠使用ISPF完成z/OS上的工作。ISPF是一个使用键盘控制的全面板式的应用程序。ISPF包含一个文本编辑器和浏览器，以及定位和列出文件，执行其他实用程序的功能。ISPF菜单列出在线用户最常用的一些功能。

图4-5展示了用ISPF创建一个数据集的分配过程。

```
 Menu  RefList  Utilities  Help
 ------------------------------------------------------------------------
Allocate New Data Set
 Command ===>
Data Set Name  . . . : ZCHOL.TEST.CNTL
Management class . . .               (Blank for default management class)
 Storage class  . . . .               (Blank for default storage class)
  Volume serial . . . . TEST01        (Blank for system default volume) **
  Device type . . . . .               (Generic unit or device address) **
 Data class . . . . . .               (Blank for default data class)
  Space units . . . . . TRACK         (BLKS, TRKS, CYLS, KB, MB, BYTES
                                       or RECORDS)
  Average record unit                 (M, K, or U)
  Primary quantity  . . 2             (In above units)
  Secondary quantity    1             (In above units)
  Directory blocks  . . 0             (Zero for sequential data set) *
  Record format . . . . F
  Record length . . . . 80
  Block size  . . . . .
  Data set name type  :               (LIBRARY, HFS, PDS, or blank)  *
                                      (YY/MM/DD, YYYY/MM/DD
  Expiration date . . .                YY.DDD, YYYY.DDD in Julian form
 Enter "/" to select option            DDDD for retention period in days
    Allocate Multiple Volumes          or blank)

 ( * Specifying LIBRARY may override zero directory block)

 ( ** Only one of these fields may be specified)
  F1=Help F2=Split F3=Exit F7=Backward F8=Forward F9=Swap F10=Actions  F12=Cancel
```

134

图4-5 使用ISPF面板分配一个数据集

图4-6展示了使用ISPF分配数据集的结果。

```
 Data Set Information
 Command ===>

 Data Set Name  . . . : ZCHOL.TEST.CNTL

 General Data                        Current Allocation
  Volume serial . . . : TEST01        Allocated tracks  . : 2
  Device type . . . . : 3390          Allocated extents . : 1
  Organization  . . . : PS
  Record format . . . : F
  Record length . . . : 80
  Block size  . . . . : 80           Current Utilization
  1st extent tracks . : 2             Used tracks . . . . : 0
  Secondary tracks  . : 1             Used extents  . . . : 0

  Creation date . . . : 2005/01/31
  Referenced date . . : 2005/01/31
  Expiration date . . : ***None***

  F1=Help F2=Split F3=Exit F7=Backward  F8=Forward   F9=Swap F12=Cancel
```

图4-6 使用ISPF分配数据集的结果

135

图4-7展示了ISPF菜单结构。

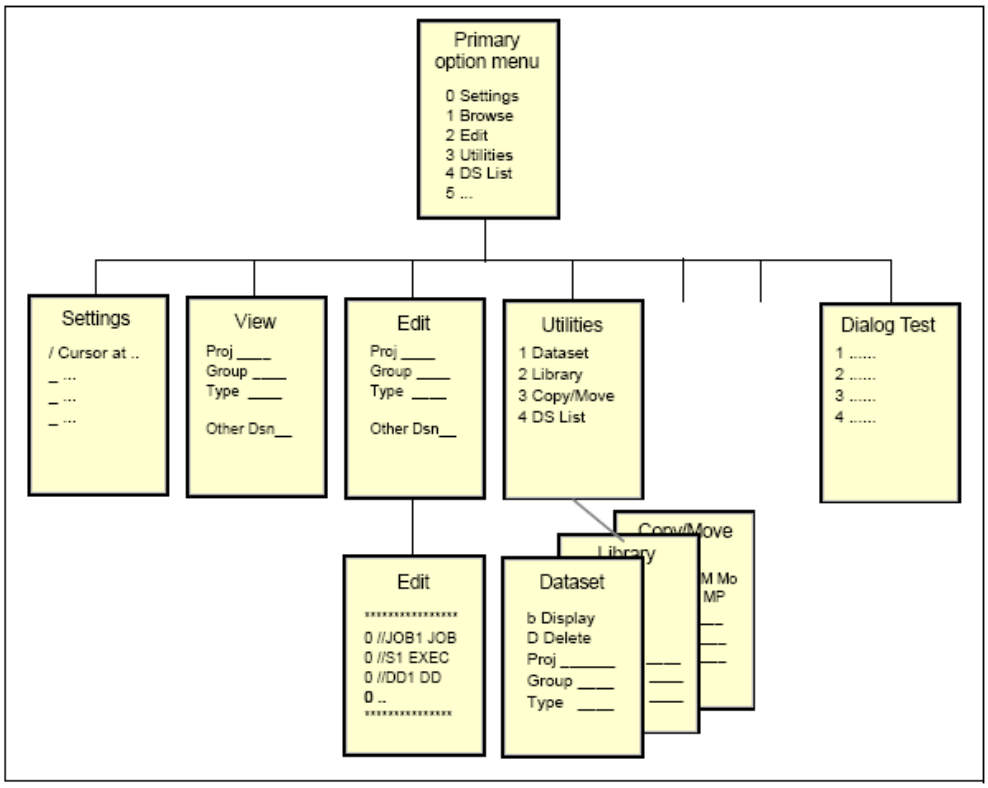


图4-7 ISPF菜单结构

136

在TSO下访问ISPF，用户要在READY提示符下输入一个命令，比如ISPPDF，来显示ISPF主选项菜单。

图4-8展示了一个ISPF主菜单的例子。

```
Menu  Utilities  Compilers  Options  Status  Help
-------------------------------------------------------------------------------
                          ISPF Primary Option Menu
Option ===>

0  Settings      Terminal and user parameters          User ID . : ZPROF
1  View          Display source data or listings       Time. . . : 17:29
2  Edit          Create or change source data          Terminal. : 3278
3  Utilities     Perform utility functions             Screen. . : 1
4  Foreground    Interactive language processing       Language. : ENGLISH
5  Batch         Submit job for language processing    Appl ID . : PDF
6  Command       Enter TSO or Workstation commands     TSO logon : IKJACCT
7  Dialog Test   Perform dialog testing                TSO prefix: ZPROF
8  LM Facility   Library administrator functions       System ID : SC04
9  IBM Products  IBM program development products      MVS acct. : ACCNT#
10 SCLM          SW Configuration Library Manager      Release . : ISPF 5.2
11 Workplace     ISPF Object/Action Workplace
M  More          Additional IBM Products

Enter X to Terminate using log/list defaults

 F1=Help F2=Split F3=Exit F7=Backward F8=Forward F9=Swap F10=Actions  F12=Cancel
```

图4-8 ISPF主选项菜单

ISPF面板可以由本地系统程序员客户化，增加其他选项。所以在每个系统上它的特性和内容可能都不一样。

137

要到达图4-9显示的ISPF菜单选项，您可以在选项行输入M。

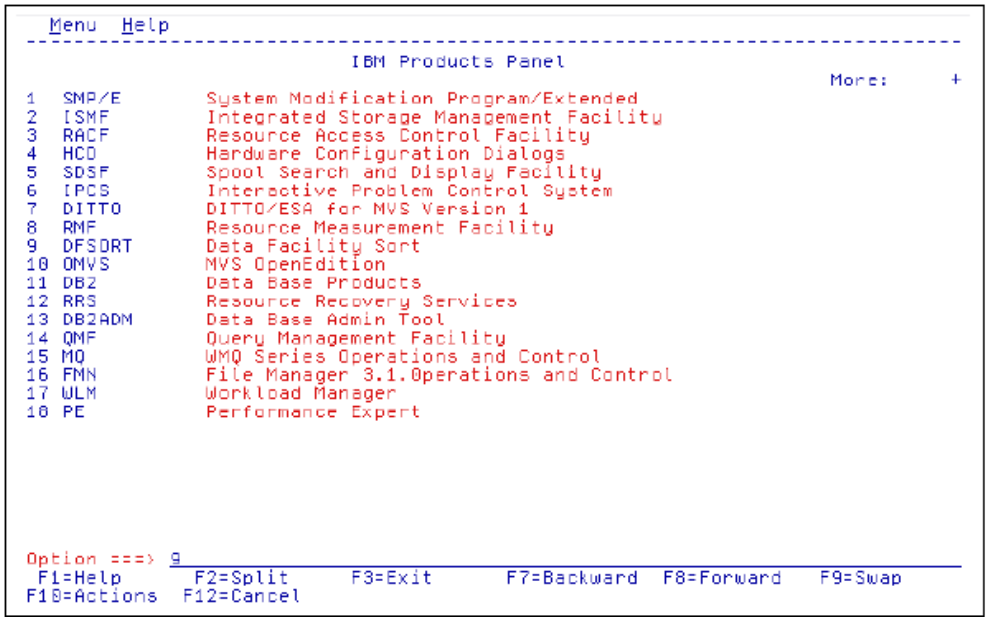


图4-9 更多ISPF选项显示

在图4-9中的面板中的选项9是SORT。我们现在选择它，这是ISPF面板驱动的应用程序的一个很好的例子。

138

图4-10展示了选中ISPF选项9之后的显示面板。

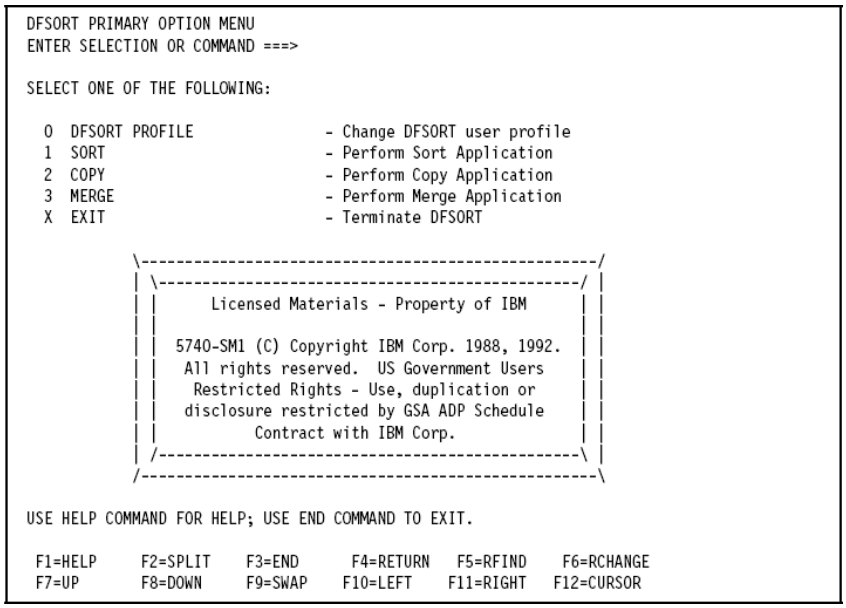


图4-10 SORT面板

我们来回忆一下131页，4.2.2章节”在本机模式下使用TSO命令”，其中有一个例子是关于TSO用户在本机模式下如何通过TSO命令完成简单排序操作的。在这里，ISPF的一个菜单选项可以实现同样的排序功能。通过SORT选项，用户可以让ISPF去处理TSO分配，创建SORT控制语句，调用SORT程序来得出排序结果。

注意在每个面板底部的键盘程序功能键(PF键)选项，使用PF3(END)将用户返回到之前的面板。

4.3.1 本课程使用的键盘映射

本书使用的很多截屏例子都在显示面板底部显示了ISPF程序功能(PF)键设置。就像我们之前提到的一样，因为z/OS用户客户化PF键分配来适应他们的需求是很普遍的，所以本书显示的键分配或许和您的系统上使用的PF键配置不相符。实际上每个客户的功能键设置都不尽相同。

139

表4-1 列出了一些最常用的PF键和其他的键盘功能及其对应的键。

表4-1 键盘映射

Function	Key
Enter	Ctrl (right side)
Exit, end, or return	PF3
Help	PF1
PA1 or Attention	Alt-Ins or Esc
PA2	Alt-Home
Cursor movement	Tab or Enter
Clear	Pause
Page up	PF7
Page down	PF8
Scroll left	PF10
Scroll right	PF11
Reset locked keyboard	Ctrl (left side)

本书中的例子使用了这些键盘设定。例如，按下‘回车’键的指示意味着您应该按下键盘右下的控制键(Ctrl)。如果键盘锁住了，那就按左下的控制键。

4.3.2 使用 PF1 帮助和 ISPF 指南

在ISPF主菜单里按下PF1帮助键可以显示ISPF指南。ISPF新用户应该让自己熟悉这个指南(图4-11)和大量的ISPF在线帮助工具。

140

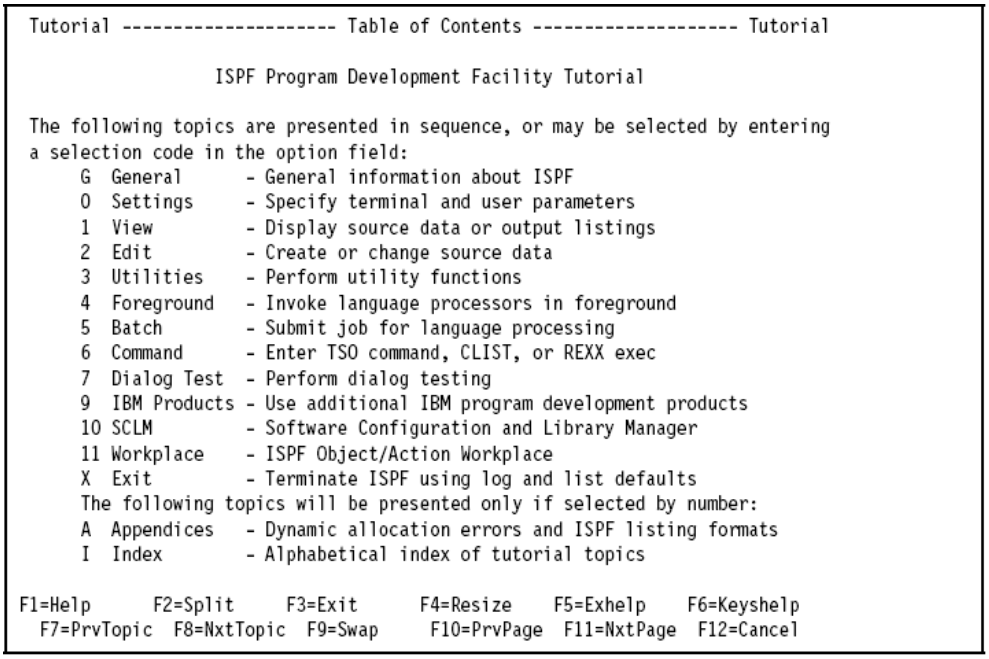


图4-11 ISPF指南主菜单

大多数情况下您只会使用整个ISPF指南中的一小部分。

除了指南，您可以从任意一个ISPF面板中访问在线帮助。当您调用帮助时，您可以滚动浏览信息。按下PF1帮助键可以查询常见的ISPF入口错误解释，和正确入口示例。ISPF帮助也包括主选项菜单中各种功能的帮助。

4.3.3 使用 PA1 键

我们现在要打断您阅读本书的乐趣，为PA1键做一个简短的广告。这是一个对TSO用户非常重要的键，每个用户都应当学会如何在键盘上找到它。

回到以前，真正的3270终端都有标明的PA1、PA2和PA3键。它们被叫做程序行为键或PA键。实际上只有PA1键仍旧被广泛使用，它的功能相当于TSO的中断键。用TSO术语来说，这是一个注意中断。也就是说按下PA1键将会终止当前任务。

在像TN3270模拟器这样的3270终端模拟器的键盘上找到PA1键是非常具有挑战性。一个3270模拟器可以被客户化出不同的键组合。在一个未修改的x3270会话中，PA1键是左Alt-1。

让我们来试试使用PA1键(以后您会发现它非常有用)。如果您现在已经打开了一个TSO会话，尝试以下操作：

1. 到ISPF的选项6中。这个面板可以接受TSO命令。

2. 在命令行输入LISTC LEVEL(SYS1) ALL命令，按回车键。这样会产生一个屏幕的输出，并且屏幕最后一行有3个星号(***)。在TSO中，***表示还有更多的

输出，您可以按回车键看到更多地内容(这一点在几乎所有TSO使用中都一致)。

3. 按回车键查看下一屏，再按回车键查看再下一屏，可以一直继续到输出完毕。

4. 利用您所用的TN3270模拟器定义的键组合来按下PA1键，就可以终止输出。

4.3.4 ISPF 菜单导航

ISPF包含一个文本编辑器和浏览器，以及定位和列出数据集、执行其他实用程序的功能。本书还没介绍过数据集，不过您需要初步了解数据集以完成本章的实验练习题。

目前为止，我们可以把数据集看成z/OS上储存数据和可执行代码的文件。一个数据集的名字最多可以有44个字符，比如ZSCHOLAR.TEST.DATA。在165页的第5章"数据集操作"中将有对数据集更为详细的描述。

数据集名字通常是分段的，由一个或多个独立的数据集限定词组成，每个限定词可以有1-8个字符。第一个数据集限定词是高级限定词或者叫做HLQ。在上面一个例子里，数据集名中ZSCHOLAR这一部分就是HLQ。

z/OS用户通常使用ISPF数据集列表实用程序来处理数据集。要从ISPF主选项菜单中访问这个实用程序，先选择**Utilities**，然后选择**Dslist**来显示实用程序选择面板，如图4-12所示。

142

```
Menu  RefList  RefMode  Utilities  Help
------------------------------------------------------------------------------
                           Data Set List Utility
Option ===> ____________________________________________________

   blank Display data set list              P Print data set list
       V Display VTOC information           PV Print VTOC information

Enter one or both of the parameters below:
  Dsname Level . . . ZPROF_____________________
   Volume serial  . .  ______
Data set list options
   Initial View . . . 1  1. Volume       Enter "/" to select option
                         2. Space        /  Confirm Data Set Delete
                         3. Attrib       /  Confirm Member Delete
                         4. Total        /  Include Additional Qualifiers

When the data set list is displayed, enter either:
  "/" on the data set list command field for the command prompt pop-up,
  an ISPF line command, the name of a TSO command, CLIST, or REXX exec, or
  "=" to execute the previous command.

 F1=Help F2=Split F3=Exit F7=Backward  F8=Forward   F9=Swap F10=Actions  F12=Cancel
```

图4-12 使用数据集列表实用程序

该面板中，您可以使用'Dsname Level'数据输入区来定位和列出数据集。要搜索一个特定的数据集，可以输入完整的(或完全限定的)数据集名。要搜索某个范围内

的数据集，比如共享一个HLQ的所有数据集，就在‘Dsname Level’ 区域中只填写HLQ。

限定词可以完全指定，部分指定或缺省指定。至少要指定一个限定词。如果要用数据集名的一部分进行搜索，可以在这一部分名字前面或者后面使用星号(*)做通配符，这样程序会返回所有符合搜索要求的数据集。避免只使用*搜索，因为TSO要搜索z/OS的很多地方，会比较耗时。

在大多数ISPF面板中，完全限定的数据集名要用单引号括起来。如果没有使用单引号，那么系统会默认加上TSO PROFILE中指定的HLQ前缀。这个默认值可以通过PROFILE PREFIX命令来修改。除此之外有个特例，就是在ISPF选项3.4 DSLIST的面板‘Dsname Level’中不需使用单引号。

例如，如果您在‘Dsname’ 区域输入ZPROF，实用程序会列出以ZPROF作为HLQ的所有数据集。数据集名称结果列表(如图4-13)允许用户编辑或浏览列表中任意数据集的内容。

143

```
 Menu  Options  View  Utilities  Compilers  Help
 ------------------------------------------------------------------------------
DSLIST - Data Sets Matching ZPROF                          Row 1 of 4
Command ===>                                             Scroll ===> PAGE

Command - Enter "/" to select action                Message          Volume
------------------------------------------------------------------------------
         ZPROF                                                       *ALIAS
         ZPROF.JCL.CNTL                                              EBBER1
         ZPROF.LIB.SOURCE                                            EBBER1
         ZPROF.PROGRAM.CNTL                                          EBBER1
         ZPROF.PROGRAM.LOAD                                          EBBER1
         ZPROF.PROGRAM.SRC                                           EBBER1
***************************** End of Data Set list ****************************


F1=Help F2=Split F3=Exit F5=Rfind F7=Up F8=Down F9=Swap F10=Left F11=Right F12=Cancel
```

图4-13 Dsname为ZPROF的数据集列表结果

144

要想知道对一个数据集的所有可能操作，您可以在数据集名字左边的命令列键入斜杠(/)。ISPF就会显示一个所有可能动作的列表，如图4-14所示。

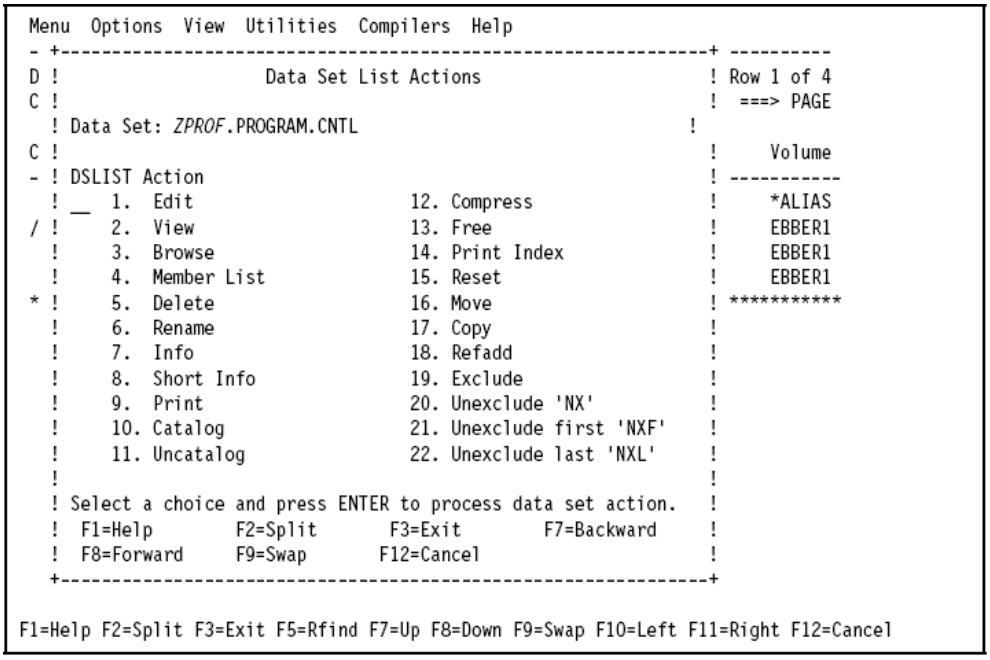


图4-14 显示数据集动作列表

4.3.5 使用 ISPF 编辑器

要想编辑数据集内容，可以在数据集名字左边键入一个e(edit)。在数据集中，每行文本都是一个记录。

您可以执行以下任务：

- 查看数据集内容，要在命令列键入一个v(view)行命令。
- 编辑数据集内容，要在命令列键入一个e(edit)行命令。
- 编辑数据集内容，要把光标移到要修改的记录区域，在已存的文本上进行输入。
- 查找和修改文本，可以在编辑器命令行键入命令。
- 插入，复制，删除，移动文本，要在需要修改行的行号处键入命令。

145

提交修改时用PF3键或save命令。退出数据集而不保存修改，要在编辑命令行键入Cancel命令。

图4-15显示了一个在编辑模式下打开的数据集ZPROF.PROGRAM.CNTL(SORTCNTL)的内容。

```
File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
-------------------------------------------------------------------------------
EDIT       ZPROF.PROGRAM.CNTL(SORTCNTL) - 01.00       Columns 00001 00072
Command ===>                                          Scroll ===> CSR
****** ***************************** Top of Data ******************************
000010 SORT FIELDS=(1,3,CH,A)
****** **************************** Bottom of Data ****************************
```

图4-15 编辑一个数据集

我们来看一下行号、文本区和编辑器命令行。主命令行，在行号处输入的行命令，以及文本覆盖是您可以使用的3种方式来完成对数据集内容的修改。TSO编辑器中的行号以10为步长递增，所以程序员可以在每两行之间插入9行而无需给程序重编行号。

4.3.6 使用在线帮助

当编辑数据集时，记住您的私人指导， F1=帮助。在编辑模式下按下PF1键将显示全部的编辑器指南(图4-16)。

```
TUTORIAL ------------------------- EDIT ---------------------------- TUTORIAL
OPTION  ===>

                    ----------------------------------
                    |             EDIT               |
                    ----------------------------------

  Edit allows you to create or change source data.

The following topics are presented in sequence, or may be selected by number:
  0 - General introduction              8 - Display modes (CAPS/HEX/NULLS)
  1 - Types of data sets                9 - Tabbing (hardware/software/logical)
  2 - Edit entry panel                 10 - Automatic recovery
  3 - SCLM edit entry panel            11 - Edit profiles
  4 - Member selection list            12 - Edit line commands
  5 - Display screen format            13 - Edit primary commands
  6 - Scrolling data                   14 - Labels and line ranges
  7 - Sequence numbering               15 - Ending an edit session

The following topics will be presented only if selected by number:
 16 - Edit models
 17 - Miscellaneous notes about edit


 F1=Help      F2=Split     F3=Exit      F4=Resize    F5=Exhelp    F6=Keyshelp
 F7=PrvTopic  F8=NxtTopic  F9=Swap     F10=PrvPage  F11=NxtPage  F12=Cancel
```

图4-16 编辑帮助面板和指南

在实验中，您将会编辑数据集，并使用F1帮助来探索编辑行命令和编辑主命令功能。在帮助功能下，选择和回顾FIND，CHANGE和EXCLUDE命令。这个实验对进一步提升本课程的技能很重要。

一些行命令包括：

i	插入1行
回车键	不输入任何字符直接按下回车可以退出输入模式
i5	插入5行
d	删除1行
d5	删除5行
dd/dd	删除多行
r	重复1行
rr/rr	重复多行
c,后面跟着a或b	之后或之前复制1行
c5,后面跟着a或b	在之后或之前复制5行
cc/cc,后面跟着a或b	在之后或之前复制多行
m, m5, mm/mm	移动行
x	隐去1行

147

4.3.7 客户化您的 ISPF 设置

您的ISPF会话命令行可能在面板底部显示，而您老师的可能会在顶部显示。这是个人偏好，不过传统用法是在面板顶部显示。

如果您想让您的命令行显示在面板顶部，可以按照如下操作：

1. 进入ISPF主选项菜单。
2. 选择选项0显示设置菜单，如149页的图4-17所示。
3. 在这些选项中，除去”Command line at bottom.”前面的’/’。使用Tab或New Line键来移动光标。

148

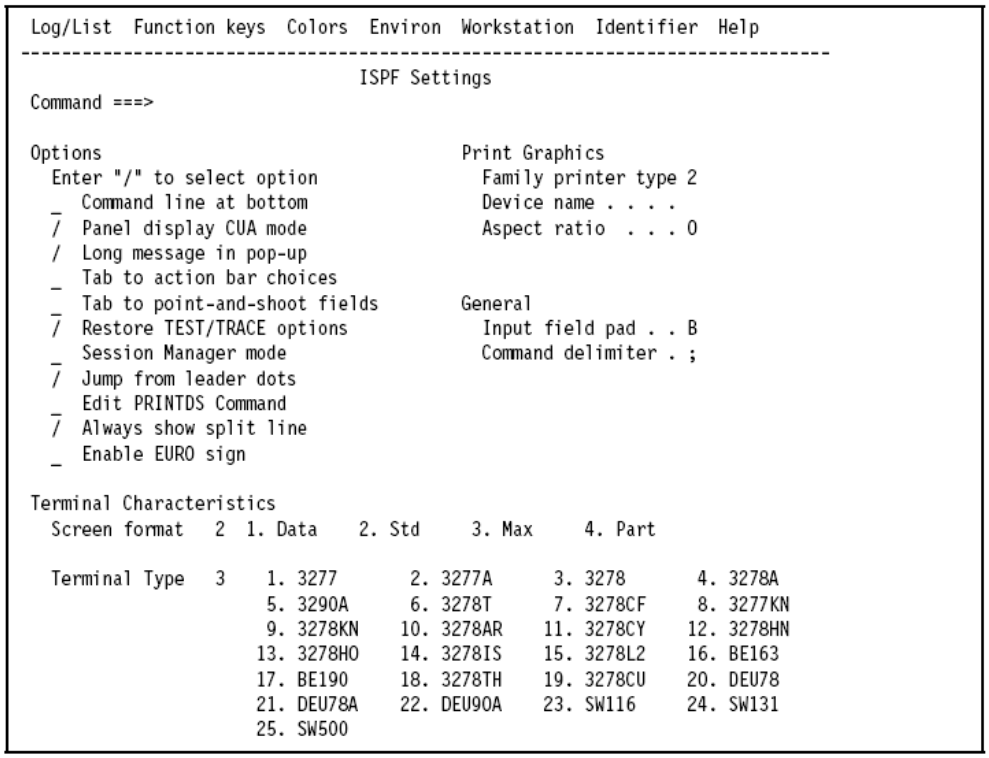


图4-17 ISPF设置

在这个菜单中，您还可以更改一些其他的参数以备今后需要：

- 移除‘Panel display CUA mode’前面的‘/’
- 将‘Terminal Type’改成4，这可以使3270支持C语言的符号。
- 将光标移到顶部的‘Log/List’选项，按下回车。
  - 选1(日志数据集默认值)。
  - 将Process Option 设为2(删除数据集，无需打印)。
  - 按PF3退出
- 将光标再次移到顶部的‘Log/List’选项。
  - 选择2(列表数据集默认值)。
  - 将Process Option设为 2，删除数据集，无需打印。
  - 按PF3键退出。
- 再次按PF3键退出到主菜单。

每个系统顶部命令条的作用通常都不一样。

另外一个客户化ISPF面板的方法是使用hilite命令，如图4-18所示。该命令允许您自行根据环境需求裁剪不同的ISPF选项。

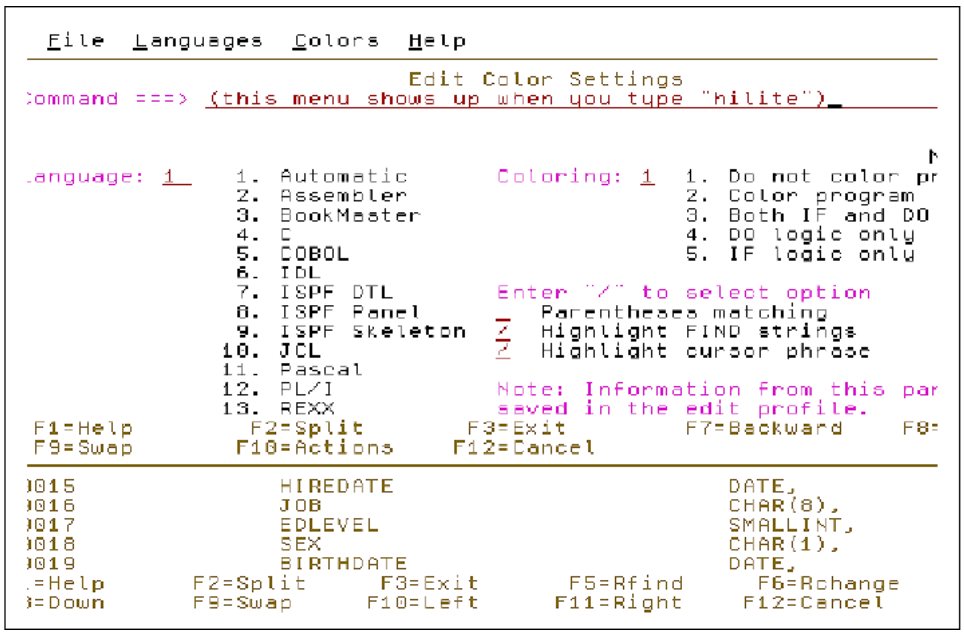


图4-18 使用HILITE命令

4.3.8 给 ISPF 添加图形化用户界面(GUI)

ISPF是一个由键盘操作的全面板应用程序。然而您可以在z/OS系统中下载安装一些ISPF图形化用户接口(GUI)客户端。安装ISPF GUI客户端以后就可以使用鼠标了。

150 图4-19 展示了一个ISPF 图形化用户界面的例子。

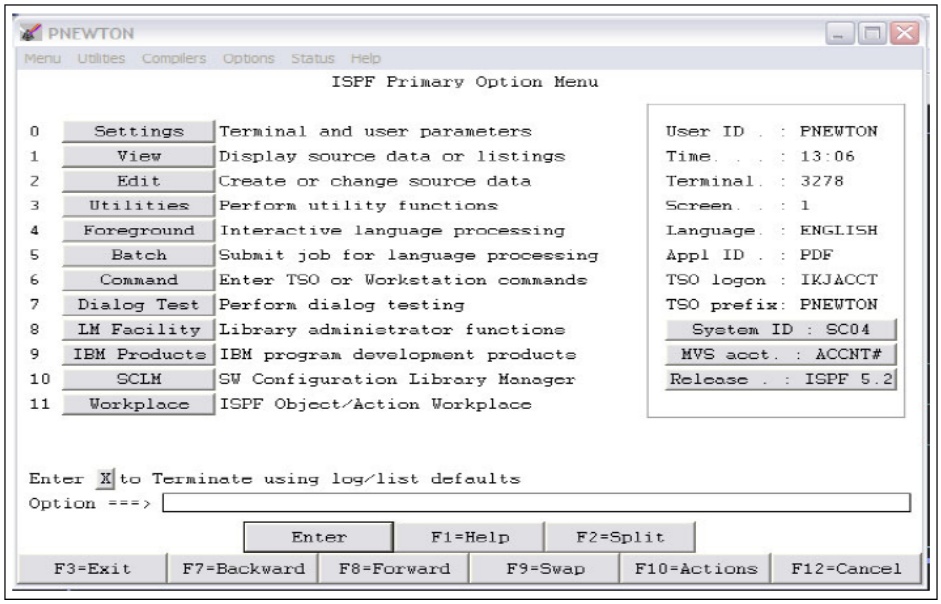


图4-19 ISPF图形化用户界面(GUI)

您可以先将光标移动到ISPF面板顶部的下拉式菜单的选项上，然后按回车。在下拉式选项中移动ISPF GUI客户端鼠标指针就可以显示各自相应的子选项。回车键和PF键在图形化用户界面中也可以使用。

4.4 z/OS UNIX 交互接口

z/OS UNIX shell和实用程序提供了一个与z/OS交互的接口。Shell和实用程序可以和z/OS中的TSO功能相比较。

Shell
UNIX 命令和 shell 命令语句的命令解释器。

为了完成一些命令请求，shell调用其他程序，这些程序叫做实用程序，shell可以用来：

- 调用shell脚本和实用程序。
- 编写shell脚本(一个shell命令的命名列表，使用shell编程语言)。
- 在TSO后台或以批处理的形式，交互的运行shell脚本和C语言程序。

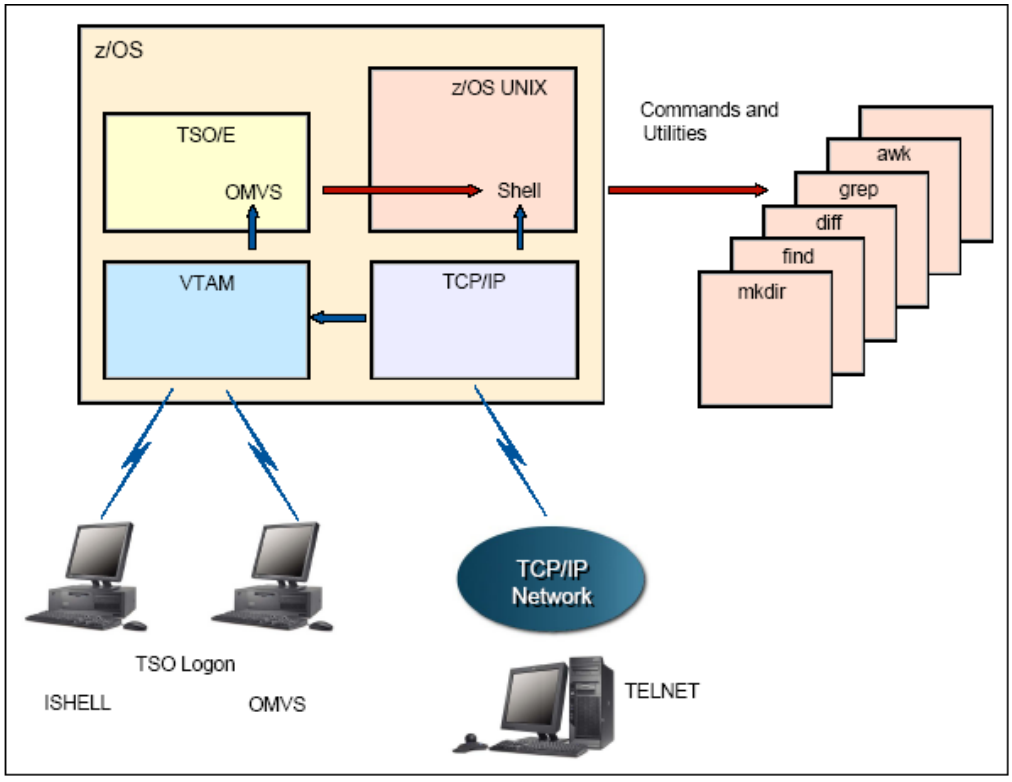


图4-20 shell和实用程序

ISHELL
TSO 命令，用于调用 ISPF 面板接口完成许多 z/OS UNIX 操作。

用户可以用以下方式调用z/OS UNIX shell：

- 从3270显示器或运行3270模拟器的工作站调用。
- 使用rlogin和telnet命令从TCP/IP连接的终端调用。
- 使用OMVS命令从TSO会话调用。

直接调用shell的另外一种方法是用户在TSO中键入ISHELL命令来使用ISHELL。

ISHELL提供了一个可以执行很多z/OS UNIX操作的ISPF面板。

图4-21展示了z/OS UNIX shell和ISHELL这些交互接口的概览。还有一些TSO/E
152 命令也支持z/OS UNIX，不过也仅限于复制文件和创建目录这些功能。

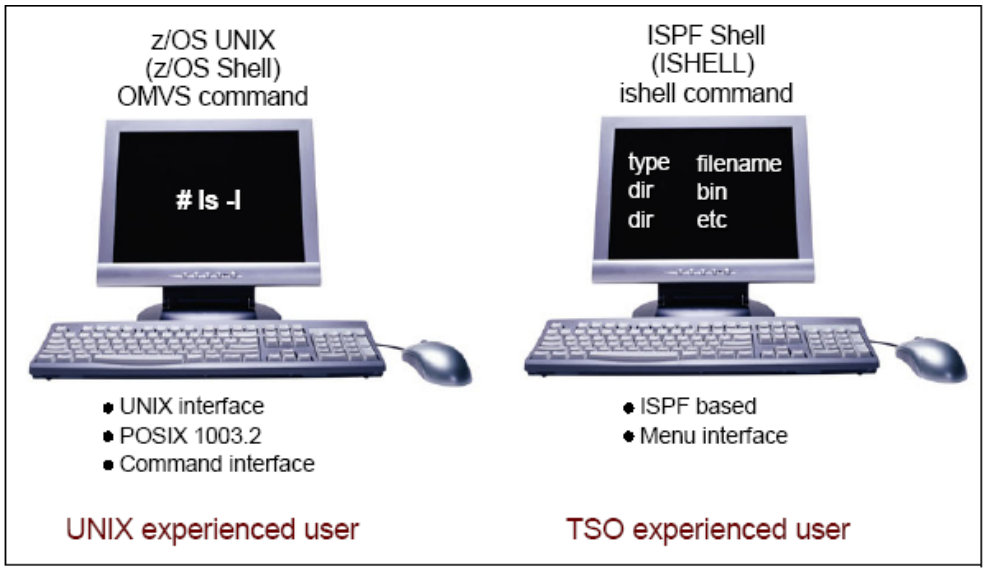


图4-21 z/OS UNIX交互接口

z/OS UNIX shell基于UNIX系统V shell，并且有一些UNIX Korn shell的特性。POSIX标准这样区别命令和实用程序：命令是指示shell完成一个特定任务的，实用程序是shell中可以通过名字调用的程序的名字。对用户来说，命令和实用程序并无区别。

z/OS UNIX shell提供的环境可以实现大部分的功能和能力。它支持正规编程语言的许多特性。

您可以将一系列的shell命令按顺序存储在一个可执行的在文本文件中。这就叫做shell脚本。

z/OS UNIX使用的TSO命令有：

ISHELL ISHELL命令调用z/OS UNIX系统服务的ISPF面板接口。对于熟悉TSO和ISPF，而又想使用z/OS UNIX的用户来说，ISHELL是-一个很好的开端。这些用户可以使用ISHELL完成很多工作，ISHELL提供了很多z/OS UNIX文件系统操作面板，包括加载和卸载文件系统的面板和进行一些z/OS UNIX管理操作的面板。

对熟悉z/OS，并且要为用户建立起UNIX资源的系统程序员来说，ISHELL通常是很有用的。

153

OMVS OMVS命令是用来调用z/OS UNIX shell的。

如果一个用户主要使用的交互计算环境是UNIX系统，那么他将对z/OS

UNIX shell环境非常熟悉。

4.4.1 ISHELL 命令(ish)

图4-22 展示了ISHELL或ISPF Shell面板，这是在ISPF选项6中输入ISHELL或ISH命令的结果。

```
 File  Directory  Special_file  Tools  File_systems  Options  Setup  Help
-------------------------------------------------------------------------
                     UNIX System Services ISPF Shell

Enter a pathname and do one of these:

    - Press Enter.
    - Select an action bar choice.
    - Specify an action code or command on the command line.

Return to this panel to work with a different pathname.
                                                              More:     +
   /u/rogers_____________________________________________________________
   ______________________________________________________________________
   ______________________________________________________________________
   ______________________________________________________________________
```

图4-22 执行ISH命令后显示的面板

4.4.2 ISHELL-用户文件和目录

要搜索某个用户的文件和目录，输入以下内容然后按回车：

/u/userid

154 例如，图4-23显示了用户rogers的文件和目录。

```
                              Directory List

 Select one or more files with / or action codes.  If / is used also select an
 action from the action bar otherwise your default action will be used.  Select
 with S to use your default action.  Cursor select can also be used for quick
 navigation.  See help for details.
 EUID=0   /u/rogers/
   Type  Perm  Changed-EST5EDT   Owner      ------Size  Filename    Row 1 of 9
 _ Dir    700  2002-08-01 10:51  ADMIN            8192  .
 _ Dir    555  2003-02-13 11:14  AAAAAAA             0  ..
 _ File   755  1996-02-29 18:02  ADMIN              979  .profile
 _ File   600  1996-03-01 10:29  ADMIN               29  .sh_history
 _ Dir    755  2001-06-25 17:43  AAAAAAA          8192  data
 _ File   644  2001-06-26 11:27  AAAAAAA         47848  inventory.export
 _ File   700  2002-08-01 10:51  AAAAAAA            16  myfile
 _ File   644  2001-06-22 17:53  AAAAAAA         43387  print.export
 _ File   644  2001-02-22 18:03  AAAAAAA         84543  Sc.pdf
```

图4-23 一个用户的文件和目录的显示

这里，您使用动作代码来完成以下任何操作：

b 浏览一个文件或目录

e 编辑一个文件或目录

d 删除一个文件或目录

r 重命名一个文件或目录

a 显示一个文件或目录的属性

c 复制一个文件或目录

4.4.3 OMVS 命令 shell 会话

您可以使用OMVS命令来调用z/OS UNIX shell。

Shell是一个命令处理器，可以用来：

- 调用shell命令或实用程序来向系统请求服务。
- 使用shell程序语言编写shell脚本。
- 交互的(在前台)或在后台或以批处理的方式，运行shell脚本和C语言程序。

您常常可以指定Shell命令的选项(也叫标记)，Shell命令还经常带有参数，比如一个文件或目录的名字。命令的格式为命令名，然后是一个或多个选项，如果有参数最后就是参数。

比如，156页的图4-24显示了如下的命令：

```
ls -al /u/rogers
```


这里ls 是命令名，-al是选项。

```
ROGERS @ SC43:/>ls -al /u/rogers
total 408
drwx------   3 ADMIN    SYS1        8192 Aug  1  2005 .
dr-xr-xr-x  93 AAAAAAA  TTY            0 Feb 13 11:14 ..
-rwxr-xr-x   1 ADMIN    SYS1         979 Feb 29  1996 .profile
-rw-------   1 ADMIN    SYS1          29 Mar  1  1996 .sh_history
-rw-r--r--   1 AAAAAAA  SYS1       84543 Feb 22  2001 Sc.pdf
drwxr-xr-x   2 AAAAAAA  SYS1        8192 Jun 25  2001 data
-rw-r--r--   1 AAAAAAA  SYS1       47848 Jun 26  2001 inventory.export
-rwx------   1 AAAAAAA  SYS1          16 Aug  1  2005 myfile
-rw-r--r--   1 AAAAAAA  SYS1       43387 Jun 22  2001 print.export
```

图4-24 执行OMVS命令后显示的OMVS shell会话

Path / Pathname
通过一个文件系统到一个指定文件的路径。

该命令列出了用户的文件和目录。如果这个路径是文件，ls就会按照请求的选项显示该文件信息。如果是一个目录，ls显示目录中的文件和子目录。您可以使用-d选项得到目录本身的信息。

如果您没有指定任何选项，ls就只显示文件名。当ls把输出发送到管道或文件时，它每行写一个名字，当它发送输出给终端时，则会使用-c(多列)格式。

术语注释：一般z/OS用户会认为数据集和文件是同义的，但是在z/OS UNIX系统服务中则不是这样。在UNIX支持的z/OS中，文件系统是一个包含了目录和文件的数据集。所以文件有特殊的定义。z/OS UNIX文件和其他的z/OS数据集不同，它们是面向字节而非面向记录的。

4.4.4 直接登录 shell

您可以从任何一台通过TCP/IP连接到z/OS的系统直接登录z/OS UNIX shell。有如下方法：

rlogin 您可以从一台有rlogin客户支持的系统上rlogin(远程登录)到shell。使用您的系统上支持的rlogin命令语法。

telnet 您可以telnet进入shell。从您的工作站或其他有telnet客户支持的系统上使用telnet命令登录。

如157页图4-25所示，上面所述的每个方法都需要在z/OS上安装并运行internet端口监听程序(inetd daemon)。

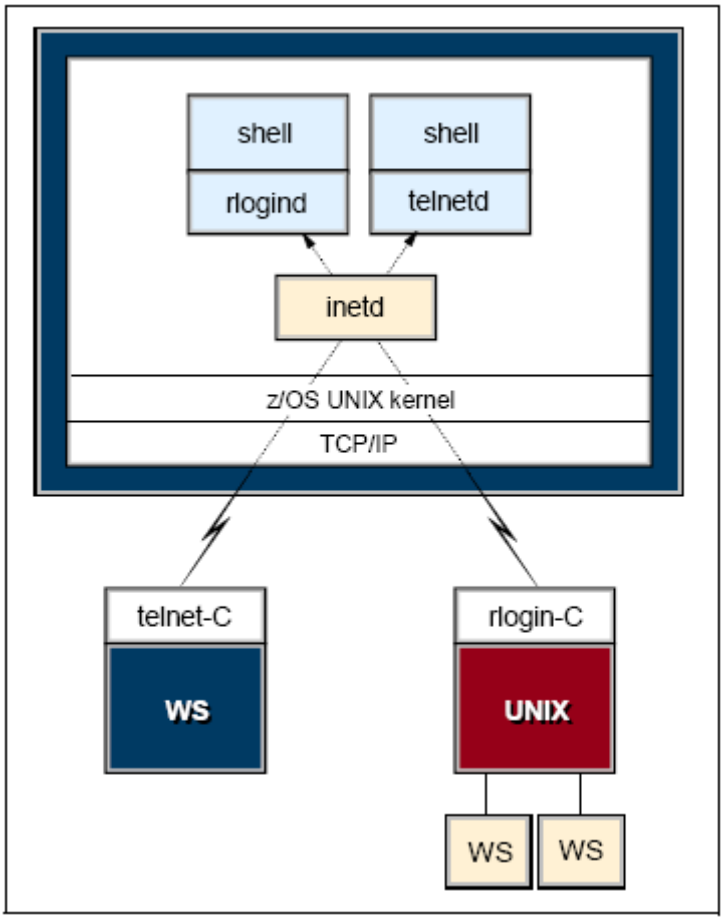


图4-25 从终端登录shell的图

图4-26显示了通过telnet登录后的z/OS shell。

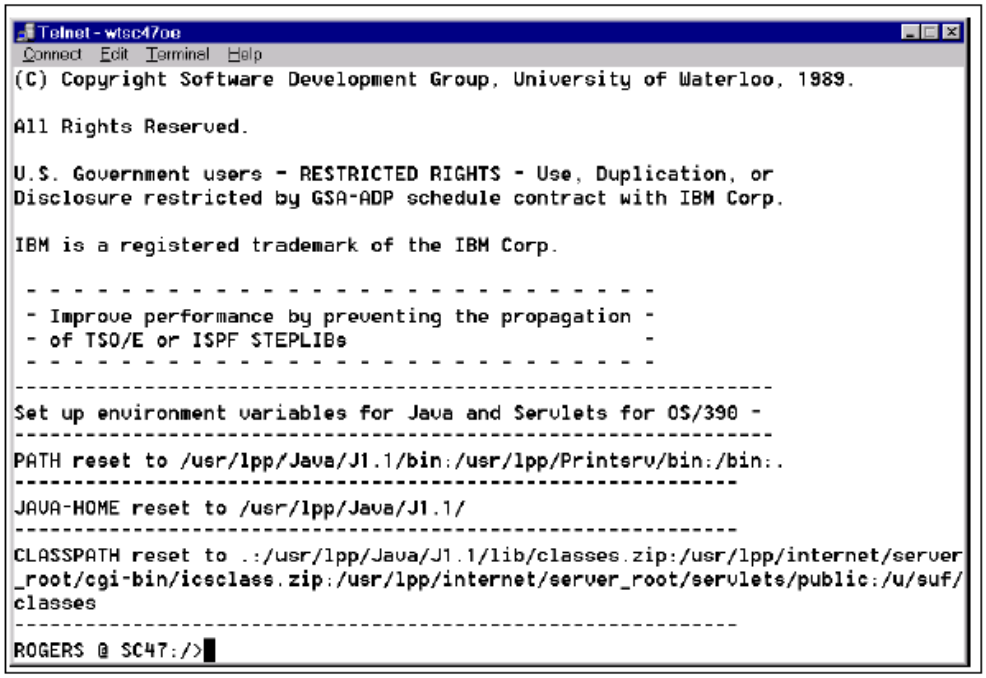


图4-26 Telnet登录shell的屏幕

异步终端支持(直接shell登录)和3270终端支持(OMVS命令)有很多不同之处，如下：

- 您不可以切换到TSO/E。然而，您可以使用TSO SHELL命令在您的shell会话中运行TSO/E命令。
- 您不能使用ISPF编辑器(包括调用ISPF编辑的oedit命令)。
- 您可以使用UNIX vi编辑器以及其他接收按键输入而不需要敲回车的交互实用程序。
- 您可以使用UNIX风格的命令行编辑。

4.5 总结

TSO允许用户登录到z/OS，使用一部分基本命令。有时候，这叫做在本机模式下使用TSO。

158

ISPF是一个用户和z/OS系统交互的菜单接口。ISPF环境从本机TSO下执行。

ISPF向用户提供实用程序，一个编辑器和ISPF应用程序。在各种安全控制允许的范围内，ISPF用户可以完全访问大部分的z/OS系统功能。

TSO/ISPF应该被看成是传统z/OS编程的系统管理接口和开发接口。

z/OS UNIX shell和实用程序提供了访问z/OS UNIX环境的命令接口。您可以通过登录TSO/E或使用TCP/IP远程登录工具(rlogin)访问shell。

如果使用TSO/E， OMVS命令可为您创建一个shell。您可以在shell环境中工作，直到退出或暂时切换到TSO/E环境中。

本章关键术语		
3270模拟器(3270 emulation)	CLIST	ISHELL
ISPF	登录(logon)	自然模式(native mode)
OMVS命令(OMVS command)	路径/路径名字(path / pathname)	记录(record)
重构扩展执行程序 (Restructured Extended Executor，REXX)	shell	分时共享选项/扩展(Time Sharing Option/Extensions，TSO/E)

4.6 复习题

为了帮助您检测对本章内容的理解，请完成以下问题：

1. 如果您需要更多的关于某个特定的ISPF面板的信息，或用户错误的帮助，您的第一步操作应该是什么？

2. 是什么使ISPF命令PFSHOW OFF命令变得有用？

3. ISPF是一个全屏接口，带有一个全屏编辑器。TSO是一个命令行接口，只有一个行编辑器。TSO行编辑器很少使用。您能想出一个需要使用TSO行编辑器的场景么？

4. IBM提供的ISPF面板可以被客户化么？

5. 说出2个z/OS UNIX交互接口的名字，解释它们之间的不同。

159

4.7 练习

本章的实验练习将会帮助您提高TSO，ISPF和z/OS UNIX命令shell的使用技能。这些技能在本书其余部分的实验练习中也需要用到。为了完成实验题，每个学生或小组需要一个TSO用户名和密码(需教师协助)。

练习涉及以下方面的技能：

- 第160页"登录z/OS，输入TSO命令"
- 第161页"通过ISPF菜单选项导航"
- 第162页"使用ISPF编辑器"
- 第163页"使用SDSF"
- 第164页"开启z/OS UNIX shell并输入命令"
- 第164页"使用OEDIT和OBROWSE命令"

140页的表4-1显示了最常用的功能映射到键盘使用的情况。

4.7.1 登录 z/OS，输入 TSO 命令

使用工作站3270模拟器和z/OS之间建立一个3270连接，用您的用户名(后面我们使用yourid代替)登录。在TSO READY提示符下(在您键入'=x'从ISPF退出到本机模式下的TSO以后)，键入以下命令：

1. PROFILE

   什么是前缀值？记录下来；这是您在系统中的用户名。

2. PROFILE NOPREFIX

   这改变了您的profile，使得TSO将不会在您命令之前加上前缀了。指定PROFILE PREFIX(加上一个指定值)或NOPREFIX是告诉系统是否使用一个值(比如您的用户名)来查找系统中的文件。NOPREFIX是告诉系统不用把结果局限于以您的用户名开头的文件，而如果不指定的话系统就会默认这样做。

3. LISTC

   LISTCAT命令(或简写成LISTC)列出特定目录中的数据集(我们在下一章讨论目录)。您的3270模拟器有一个PA1(注意)键。您可以使用PA1键来终止命令输出。

   注意：当您看到三个星号(***)时，它表示您的屏幕满了，按回车键或PA继续。

160

4 PROFILE PREFIX(userid)

   该命令指定了在所有不完全限定的数据集名前加上您的用户名作为前缀。这将会过滤下一条命令的结果。

5 LISTC

   显示什么？

6 ISPF(或ISPPDF)

   进入TSO的ISPF菜单驱动接口。

   注意：在一些系统中，您可能需要选择选项P来访问ISPF主屏幕。

4.7.2 通过 ISPF 菜单选项导航

在ISPF主选项菜单中，做以下操作：

1. 选择Utilities，然后在Utility Selection面板中选择Dslist。

2. 在‘Dsname Level’输入区填入SYS1，按回车。显示了什么？

3. 使用F8键来向下翻页，F7向上翻页，F10向左移动，F11向右移动，F3退出。

4. 在‘Dsname Level’输入区填入SYS1.PROCLIB，按回车。显示了什么？

5. 在SYS1.PROCLIB左边命令栏中输入v，这是一个分区数据集，有多个成员。在任一成员左边键入s来选择这个成员查看。按F1，出现怎样的帮助信息？

6. 在ISPF命令或选项行输入‘ =0’。在‘ISPF Settings’面板中列出的第一个选项是什么？修改您的设置将命令行置于面板底部，退出设置面板后设置就会生效。

7. 键入PFSHOW OFF，然后键入PFSHOW ON。区别在哪里呢？这有什么作用呢？

8. 退回到ISPF主选项菜单。哪个值可以用来选择Utilities？

9. 选择**Utilities。**

10. 在Utility Selection面板，哪个值对应选择Dslist？退回到ISPF主选项菜单。在选项行，键入Utilities的选项值，后面加一个句点，然后再键入Dslist选项值。出现的是哪个面板呢？

11. 退回到ISPF主选项菜单。将光标置于在面板最顶部的‘Status’条目上，然后按回车。选择‘Calendar’值按回车，然后选择‘Session’值。改变了什么？

12. 现在将您的屏幕设置回原先的配置，使用‘Status’下拉菜单，然后选择**‘Session’。**

4.7.3 使用 ISPF 编辑器

在ISPF主选项菜单中，做以下操作：

1. 进入DSLIST实用程序面板，在‘Dsname Level’处输入yourid.JCL，按回车。

2. 在yourid.JCL左边输入e(edit)，在成员EDITTEST左边输入s(select)。在编辑命令行输入PROFILE，观察以‘profile’和‘message’为前缀的数据行。阅读profile设置和‘message’，然后在命令行输入RESET。结果是什么？

3. 在第一行数据结尾输入任意字符串，然后按回车。在命令行，输入CAN(cancel)。然后按回车确认取消请求。再次编辑数据集中的EDITTEST，您的修改保存了么？

   提示：当您对ISPF熟悉起来时，您将记住常用选项使用的字母和数字。在选项之前键入‘=’键将可以跳过中间的菜单，直接到达该选项面板。

您可以使用‘=’和‘．’符号直接跳转到嵌套选项。如=3.4可以使您直接跳转到常用的数据集实用程序菜单。

4. 移动光标到显示器顶部的行中，按F2。结果出现了第二个ISPF面板。如果反复按F9会发生什么？

5. 使用F9，切换到ISPF主选项菜单，然后按F1显示ISPF指南面板。

6. 从ISPF指南面板中选择**Edit**，然后选择**Edit Line Commands**，再选择**Basic Commands**。按回车来滚动浏览这些基本命令指南。当您做这些时，经常切换(F9)到编辑会话中，在EDITTEST中练习使用这些命令。重复同样的场景练习来使用Move/Copy命令和移位命令。

7. 在ISPF指南面板中选择**Edit**，再选择**Edit Primary Commands**，然后选择**FIND/CHANGE/EXCLUDE commands**。按回车滚动浏览FIND/CHANGE/EXCLUDE命令指南。当您做这些时，经常切换(F9)到编辑会话，在EDITTEST中练习使用这些命令。

8. 在ISPF帮助面板输入‘=x’来终止第二个ISPF面板会话。保存并退出编辑面板(F3)回到ISPF主选项菜单。

4.7.4 使用 SDSF

从ISPF主选项菜单中，定位并选择‘System Display and Search Facility’(SDSF)，这是一个让您查看输出数据集的实用程序。选择More找到SDSF选项(5)，或直接输入‘=M.5’。ISPF主选项菜单一般包括很多选项，而不只有第一个面板列出的那些，第一个面板中有如何显示更多选项的指示。

1. 输入LOG，然后向左平移(F10)，向右平移(F11)，向上翻页(F7)和向下翻页(F8)。在命令输入行输入TOP，然后再输入BOTTOM。在命令输入行输入DOWN 500和UP 500。后面您将会学到如何阅读系统日志。

2. 观察命令输入行最右面的SCROLL值。

   Scroll ===> PAGE

用TAB键移到SCROLL值，SCROLL值可以是：

C或CSR 滚动至当前光标位置
P或PAGE 整页或整屏滚动
H或HALF 半页或半屏滚动

3. 您可以在很多ISPF面板里找到SCROLL值，包括编辑器面板。您可以在当前值的第一个字母处输入滚动模式的第一个字母来修改滚动模式。将值修改成CSR，将光标移动到系统日志中的另一行，然后按F7。光标所在的那行是否显示在顶部了呢？

4. 在SDSF命令输入行输入ST(status)，然后输入SET DISPLAY ON。观察Prefix,

Dest, Owner和 Sysname的值。为了显示它们的当前值，输入*作为过滤符，比如：

PREFIX *
OWNER *
DEST
结果应该是：
PREFIX=* DEST=(ALL) OWNER=*

5. 输入DA显示所有活动的作业。输入ST得到在输入，活动和输出队列中所有作业的状态。再次按F7(上一页)，F8(下一页)，F10(向左平移)，F11(向右平移)。

4.7.5 开启 z/OS UNIX shell 并输入命令

在ISPF主选项菜单中，选择选项6，然后输入OMVS命令。在您的主目录下，输入以下shell命令：

id	显示您的当前id。
date	显示时间和日期。
man date date	命令的指南。您可以按回车浏览面板，输入quit退出面板。
man man	指南的帮助。
env	该会话的环境变量。
type read	识别read究竟是命令，实用程序还是别名等等。
ls	列出一个目录。
ls -l	列出当前目录。
ls -l /etc.	列出目录/etc。
cal	显示当前月份的日历。
cal	**2005** 显示2005年日历。
cal 1752	显示1752年日历。九月是否少了13天呢？【答案：是的。所有UNIX日历在1752年9月都少了13天。】要知道原因么？问一下历史系学生吧！

Exit 终止OMVS会话。

4.7.6 使用 OEDIT 和 OBROWSE 命令

另外一个进入OMVS shell的方法是在任意ISPF面板输入TSO OMVS命令。在您的主目录下，输入以下shell命令：

cd /tmp	这是一个您有权限更新的目录。

oedit myfile	打开ISPF编辑面板，在当前路径下创建一个新的文本文件。在编辑器中写入一些文本，保存退出(F3)。
ls	简单显示当前目录信息。
ls -l	详细显示当前目录信息。
myfile	myfile可以是您选择创建的任何文件。
obrowse myfile	浏览您刚刚创建的文件
exit	终止OMVS会话。

164

5

第 5 章 数据集操作

目标：在z/OS操作系统的环境下工作，您必须理解数据集，这是包含了程序和数据的一些文件。传统z/OS数据集的特点和UNIX及PC系统上的文件系统有不少区别。更有意思的是，您还可以在z/OS上创建具有UNIX系统共同特点的UNIX文件。

完成本章后，您将能：

- 解释数据集是什么
- 描述数据集的命名规则及记录格式
- 列举管理数据和程序的一些访问方法
- 解释目录和VTOC的作用
- 创建，删除和修改数据集
- 解释UNIX文件系统和z/OS数据集之间的区别
- 描述z/OS UNIX文件系统是如何使用数据集的

5.1 数据集是什么？

系统中几乎所有作业都涉及数据的输入输出。在主机系统上，通道子系统管理I/O设备，如磁盘，磁带和打印机等的使用，而z/OS将给定任务的数据和设备连接起来。

z/OS通过数据集管理数据。术语数据集指包含了一个或多个记录的文件。任何记录的集合命名为数据集。数据集可以保存信息，比如医疗记录或保险记录，可以被运行在系统上的程序使用。数据集还可以用来存储应用程序或操作系统本身需要的信息，比如源程序，宏库或系统变量或者是参数。那些包含了可读文本的数据集可以打印出来或者在一个控制台上显示(许多数据集包含了装载模块或二进制数据，这些不能真正地打印出来)。数据集可以被编目，通过目录，数据集可以通过名称引用而不需要指定它的存储位置。

数据集
一组逻辑上相关的数据记录的集合，比如一些宏或源程序的库。

简单来说，记录就是包含数据的固定数目的字节。通常，记录包含一些相关的信息，这些信息可以看作一个单元，例如数据库中的一个条目或者一个部门某成员的个人信息。术语'域'指一条记录中的一个特定部分，用于对数据进行更细的分类，如职员的名字或其所在部门。

记录是运行在z/OS[1]上的程序是使用的基本信息单元。数据集中的数据可以有很多种组织方式，这是由我们打算如何来访问这些信息决定的。例如，如果您想要写一个用来处理个人信息之类的应用程序，您在程序里就可以为每个人的数据定义一条记录格式。

在z/OS中有很多类型的数据集，有多种不同的访问方法。本章讨论三种数据集：顺序，分区和VSAM数据集。

在顺序数据集中，记录是连续存放的数据条目。例如，为了得到数据集中的第10项，系统必须首先通过前面的9项。必须按顺序使用的数据，像班级花名册里按字母表排序的姓名列表就最好存储在顺序数据集中。

分区数据集(PDS)由一个目录和多个成员组成。目录中有每个成员的地址，因此程序或操作系统能够直接访问每个成员。然而，每个成员又是由顺序存储的纪录组成。分区数据集通常被称为库。程序作为分区数据集的成员存储。一般来说，操作系统按顺序把PDS的多个成员载入存储区，但是当选择一个程序来运行时，它可以直接访问该成员。

在虚拟存储访问方法(VSAM)键值顺序数据集(KSDS)中，记录是带着控制信息(键值)的数据项，这样在数据集中，系统就能在不需要遍历之前所有条目的情况下，直接获取一个条目。对于使用频繁且访问顺序不可预知的数据条目，VSAM KSDS数据集是理想的选择。这章的后面部分我们将讨论这些不同的数据集以及目录的用处。

相关阅读：关于数据集的一些信息，标准参考详见IBM出版物《z/OS DFSMS Using

[1] z/OS UNIX 文件和典型的 z/OS 数据集有很大不同，因为它们是面向字节，而不是面向记录的。

Data Sets》。可以在z/OS的因特网书库网站上找到这本书以及相关出版物：

http://www.ibm.com/servers/eserver/zseries/zos/bkserv/

5.2 数据集存放在哪里？

z/OS支持很多不同的设备进行数据存储。磁盘或磁带是最常用的长期存储数据集的设备。磁盘驱动器被称为直接访问存储设备(DASD)，因为尽管其上有些数据集是顺序存储的，但这类设备能进行直接访问。磁带驱动器被称为顺序访问设备因为磁带上的数据集必须顺序访问。

术语DASD应用于磁盘或磁盘的模拟等价物。DASD可以存储所有类型的数据集(磁带只能存储数序数据集)。DASD卷用来存储数据和可执行程序，包括操作系统本身，也用于临时工作存储。一个DASD卷可以用来存放很多不同的数据集，上面的空间也可以被重复分配和使用。

为了使系统能够快速定位一个数据集，z/OS包括了一个称为主目录的数据集，它允许访问计算机系统中的所有数据集或其他数据集的目录。z/OS要求主目录驻留在DASD上，该DASD总装载在一个在线的驱动器上。我们将在183页上的5.11节”目录和VTOC”中进一步讨论目录。

5.3 什么是访问方法？

访问方法定义了用来存储和获取数据的技术。访问方法有它们自己的数据结构来组织数据，有系统提供的程序(宏)来定义数据集，有实用程序来处理数据集。

访问方法主要通过数据集的组织方式来识别。例如，z/OS用户使用基本顺序访问方法(BSAM)或队列顺序访问方法(QSAM)来访问顺序数据集。

根据一个组织方式识别的访问方法有时能用来访问用另一种不同方式定义的数据集。例如，用BSAM创建的一个顺序数据集(不是扩展格式的数据集)可以被基本直接访问方法(BDAM)访问，反之亦然。另一个例子是UNIX文件，可以用BSAM，QSAM，基本分区访问方法(BPAM)或虚拟存储访问方法(VSAM)访问。

本文并没有描述z/OS所有可用的访问方法。通常使用的访问方法包括：

QSAM 队列顺序访问方法(经常使用)
BSAM 基本顺序访问方法(用在特殊情况下)
BDAM 基本直接访问方法(逐渐陈旧)
BPAM 基本分区访问方法(供库使用)
VSAM 虚拟存储访问方法(用在比较复杂的应用程序中)

5.4 如何使用 DASD 卷？

DASD卷用来存储数据和可执行程序(包括操作系统本身)，以及用于临时工作存储。一个DASD卷上可以存储许多不同的数据集，上面的空间也可以被再次分配和使用。

在一个卷上，数据集的名字必须是唯一的。定位数据集需要设备类型，卷序列号和数据集名字。这不像UNIX系统的文件树。基本的z/OS文件结构不是层此形的。z/OS数据集没有和路径名等价的东西。

尽管DASD卷在物理性质，容量和速度上不同，但它们在数据记录，数据检索，数据格式以及编程方面是相似的。每个卷的记录表面都被分成许多同心的磁道。

磁道数和容量根据设备类型的不同而不同。每个设备有一种访问机制，使用读/写头在记录表面旋转通过它们时传输数据。

5.4.1 UNIX 和 PC 用户理解 DASD 术语

主机软硬件的磁盘和数据集特性和UNIX及PC系统有很大的区别，它们有自己的专用术语。在本文中，下面的术语用来描述z/OS存储管理中的各个方面：

- 直接访问存储设备(DASD)是磁盘驱动器的别名
- 磁盘驱动器也被称为磁盘卷，磁盘包，或磁头集合(HDA)。本文中除了讨论设备的物理性质时，我们采用术语‘卷’来表示磁盘驱动器。
- 一个磁盘驱动器包含了许多柱面。
- 柱面包含了许多磁道。
- 磁道包含了许多数据记录，它们以计数关键数据(CKD)[2]格式存在。
- 数据块是磁盘上的记录单元。

5.4.2 DASD 标签是什么？

操作系统使用标签组来识别DASD卷和它们包含的数据集。客户应用程序通常并不直接使用这些标签。DASD卷必须使用标准标签。标准标签包括了卷标签，每个数据集的数据集标签以及选择性的用户标签。卷标签，存储在0号柱面的0号磁道上，用于确定每个DASD卷。

在每个DASD卷在系统中使用前，z/OS系统程序员或存储管理员使用ICKDSF实用程序来初始化它。ICKDSF产生卷标签，建立卷目录表(volume table of contents，VTOC)，VTOC是包含了数据集标签的一种结构(我们将在183页的"什么是VTOC？"中讨论)。系统程序员还可以使用ICKDSF来扫描一个卷以保证它是可用的并且可以重新格式化所有磁道。

[2] 当前的设备实际上使用扩展计数关键数据(ECKD™)协议，但本书中我们仍采用 CKD 作为集合名。

5.5 分配数据集

为了使用数据集，必须首先分配它(建立一个它的链接)，然后用您选择的访问方法的宏来访问数据。

169

分配数据集意味着以下两件事情或其中之一：

- 在磁盘上为一个新的数据集创建空间。
- 在一个作业步和任何数据集之间建立一个逻辑链接。

在本章结束部分，我们用ISPF面板上的选项3.2分配一个数据集。其他几种分配数据集的方法如下：

访问方法服务 您可以通过一个称为访问方法服务的多功能服务来分配数据集。访问方法服务包括了通常使用的处理数据集的命令，像ALLOCATE，ALTER，DELETE和PRINT。

ALLOCATE 您可以使用TSO ALLOCATE命令来创建数据集。这个命令实际上引导您指定必须指定的分配值。

ISPF菜单 您可以使用称为交互式系统生产工具(ISPF)的一系列TSO菜单。其中一个菜单引导用户分配数据集。

使用JCL 您可以使用一些称为作业控制语言的命令来分配数据集。

5.6 数据集是如何命名的

每当用户分配一个新的数据集时，必须给数据集一个唯一的名字。

一个数据集名字可能是一个名字段，或一系列联合的名字段。每个名字段代表了一级限定。例如，数据集名VERA.LUZ.DATA是由三个名字段组成。左边的第一个名字段被称为高级限定词(HLQ)，右边的最后一个名字段是最低级的限定词(LLQ)。

HLQ
具有多个段的名字的第一个段

每个名字段的长度可以是一到八个字符，名字段的第一个字符必须是字母(A到Z)或特殊符号(# @ $)。剩下的七个字符可以是任一字母，数字(0到9)，特殊符号或连接符(-)。名字段之间用句点(.)隔开。

包括所有的名字段和句点，数据集名的长度不能超过44个字符。因此，一个数据集名最多可以由22个名字段组成。

170

例如，下面的名字就是无效的数据集名：

- 限定词的字符数多于8的名字(HLQ.ABCDEFGHI.XYZ)
- 包含了两个连续句点的名字(HLQ..ABC)

- 以句点结束的名字(HLQ.ABC.)
- 包含了不是以字母或特殊字符打头的限定词的名字(HLQ.123.XYZ)

一个数据集的HLQ通常由安全系统控制。其余的名字段也有许多命名约定，这些是约定而不是规定，但是它们被广泛使用，包括以下各项：

- 名字中的字母LIB表示数据集是一个库，字母PDS也可以表示一个库，但它较少使用。
- 名字中的字母CNTL，JCL或JOB通常表示数据集中包含JCL(但是不一定专用于JCL)
- 名字中的字母LOAD，LOADLIB或LINKLIB表示数据集中包含可运行的模块(一个具有z/OS可执行模块的库只能包含可执行模块)。
- 名字中的字母PROC，PRC或PROCLIB表示是一个JCL的过程库。
- 不同的组合用来表示一种特定语言的源代码，例如COBOL，Assembler，FORTRAN，PL/1，JAVA，C或C++。
- 数据集名的一部分可以表示一个特定的项目，例如PAYROLL。
- 尽量不要使用太多限定词。例如P390A.A.B.C.D.E.F.G.H.I.J.K.L.M.N.O.P.Q.R.S是一个有效的数据集名(大写，不超过44个字节，没有特殊符号)，但它并不能够很好地表达数据集的意义。通常一个好的数据集名只包含三或四个限定词，而且尽量地表达数据集的意思。
- 再说一次，所有段的字符长度不能超过44个字节。

5.7 通过 JCL 在 DASD 卷上分配空间

这个部分描述了用作业控制语言(JCL)来分配一个数据集。在这本书中，我们稍候会讨论JCL的使用；这个部分先预览一些本书会面会用到的数据集空间分配设置。除了JCL，其他分配数据集的常用方法包括IDCAMPS实用程序或使用DFSMS来自动化分配数据集。

171

在JCL中，您可以以块，记录，磁道或柱面为单位来指定请求的空间。创建一个DASD数据集的时候，您可以使用SPACE参数来明确指定所需的空间数量，或者使用数据类(data class)[3]中可用的信息隐含地指定。如果用&&来开始数据集名，JCL处理器就会把它分配成临时数据集，在作业处理完后会将它删除。

如果SMS处于激活状态，即使数据集不是SMS管理的，系统也可以使用数据类。对系统管理的数据集，系统来选择卷，在分配数据集的时候，您就不必指定卷了。

如果您根据平均记录长度来指定请求的空间，空间分配就独立于设备类型。对系统管理的存储来说，设备独立非常重要。

[3]通过DFSMS或IDCAMPS实用程序来分配一个新数据集的时候，可以以千字节(kilobytes)或百万字节(megabytes)来指定空间分配，而不是块(blocks),记录(records),磁道(tracks),或是柱面(cylinders)。

5.7.1 逻辑记录和块

逻辑记录长度(LRECL)是关于处理单元(例如一个客户，一个账户，一个在册职员等等)的信息单位。它是处理数据的最小单位，由能被处理应用程序识别的信息组成。

位于DASD，磁带或其他光学设备上的逻辑记录以组为单位放在称为块的物理记录中。BLKSIZE为这些块的大小。DASD卷上的每个数据块都有一个清楚的位置和唯一的地址，因此不必广泛搜索就能查找到任何块。逻辑记录可以被直接或顺序存储和检索。

LRECL
最大记录长度—
数据集 DCB 属性之一。

一个逻辑记录的最大长度(LRECL)受所使用介质的物理大小限制。

当请求的空间数量以块为单位表示，必须指定数据集内块的数量和平均长度。

我们来看下面这个请求磁盘存储的例子：

- 平均块长度以字节为单位=300
- 初次块分配量=5000
- 二次块分配量，当初次分配的块被数据填满时使用二次分配的块=100

操作系统从这些信息来估计和分配请求的磁盘空间。

5.7.2 数据集分区

磁盘数据集的空间按照分区进行分配。分区是连续数量的磁盘驱动器磁道，柱面或块。数据集在分区内增长。较老类型的数据集在每个卷最多有16个分区。较新类型的数据集在每个卷最多有128个分区或在多卷情况下最多可以有255个分区。

当您没有在使用PDSE而且必须自己管理地址空间而不是通过DFSMS的时候，分区是相应的。为了最大化磁盘性能，您希望数据集可以放置在一个单一的分区中。读写连续磁道要比读写分散在磁盘上的磁道更快，后者可能是动态分配磁道造成的。但如果没有足够的相连空间，数据集就得使用多个分区。

5.8 数据集记录格式

传统的z/OS数据集是面向记录的。正常使用下，它没有PC系统和UNIX系统中的字节流文件。(z/OS UNIX有字节流文件，并且字节流函数也存在于其他特定区域中。这些不是传统的数据集。)

在z/OS中，没有新行(NL)或回车换行(CR+LF)字符来指示记录的结束。在一个给定的数据集里，记录可以是定长或变长的。例如，当用ISPF编辑数据集时，每一行都是一个记录。

传统的数据集可以有如下五种记录格式：

F-Fixed 定长不组块记录格式是指磁盘上的一个物理块就是一个逻辑记录，并且所有的块/记录大小相同。该格式很少用。

FB-Fixed Blocked 定长组块记录格式是指一些逻辑记录组合成一个物理块。这能提供有效的空间利用和操作。该格式通常适用于定长记录。

V-Variable 变长不组块记录格式是指一个逻辑记录就作为一个物理块，变长逻辑记录包含一个记录描述字(RDW)，之后是数据。记录描述字是一个4字节的用来描述记录的域。前2位包含了逻辑记录的长度(包括这4字节的RDW)。长度可以从4到32760字节。第3和第4字节必须是0，因为其他值是用来表示跨范围记录的。该格式很少被使用。

173

VB-Variable Blocked 变长组块记录格式，是指一个物理块由若干条变长逻辑记录组成，每条记录都有RDW描述。软件必须在块的开始处放一个附加的块描述字(BDW)，里面包含了整个块的长度。

U-Undefined 无定义格式， 该格式是由没有预定义结构的变长的物理记录/块组成的。虽然这种格式可能对许多特殊的应用程序来说很有吸引力，但是它通常只用于可执行模块。

我们必须强调块和记录之间的区别。块是写在磁盘上的而记录是一个逻辑实体。

这里提到的术语在z/OS环境中都是普遍使用的。关键的术语有：

- 块大小(BLKSIZE)是针对F和FB记录的写在磁盘上的物理块大小。对V，VB和U记录格式，是数据集可以使用的最大物理块大小。
- 逻辑记录大小(LRECL)是数据集的逻辑记录的大小(F，FB)或允许的最大逻辑记录大小(V，VB)。U格式记录没有LRECL。
- 记录格式(RECFM)有F，FB，V，VB或U，正如前面介绍的。

块大小
是针对 F 和 FB 记录的写在磁盘上的物理块大小。

这些术语都是数据控制块(DCB)特性，这是根据它们在汇编语言程序中定义的控制块来命名的。创建一个新数据集时，用户应该给这些参数指定参数值。数据集的类型和长度是根据记录格式(RECFM)和逻辑记录长度(LRECL)来决定的。固定长度的数据集的RECFM可能是F，FB，FBS等。变长数据集的RECFM可能是V，VB，VBS等。

RECFM
记录格式；是数据控制块特征之一。

RECFM=FB及LRECL=25是一个固定长度(FB)的数据集，记录长度为25个字节(B表示blocked)。对一个FB数据集来说，LRECL告诉您数据集中每个记录的长度；几乎所有记录都是一样长的。FB记录中第一个数据字节在位置1。LRECL值为25的FB数据集中一条记录可能是这样的：

位置1-3： Country Code = ‘USA’
位置4-5： State Code = ‘CA’

174

位置6-25： City = ‘San Jose’ 右边用12个空格填满。

RECFM=VB，LRECL=25的数据集是一个变长的数据集，最大记录长度为25个字节。在一个VB数据集中，记录长度可以不同。每个记录的头四个字节包含了RDW，RDW的头两个字节包含了那条记录的长度(以字节为单位)。一条VB记录的第一个数据字节位于位置5，在4个字节的RDW之后。LRECL值为25的VB数据集的一条记录可能是这样的：

位置1-2： Length in RDW = hex 0011 = decimal 17
位置3-4： Zeros in RDW = hex 0000 = decimal 17
位置5-7： Country Code = ‘USA’
位置8-9： State Code = ‘CA’
位置10-17：City = ‘San Jose’

175页上的图5-1显示了五种数据格式记录和块之间的关系。

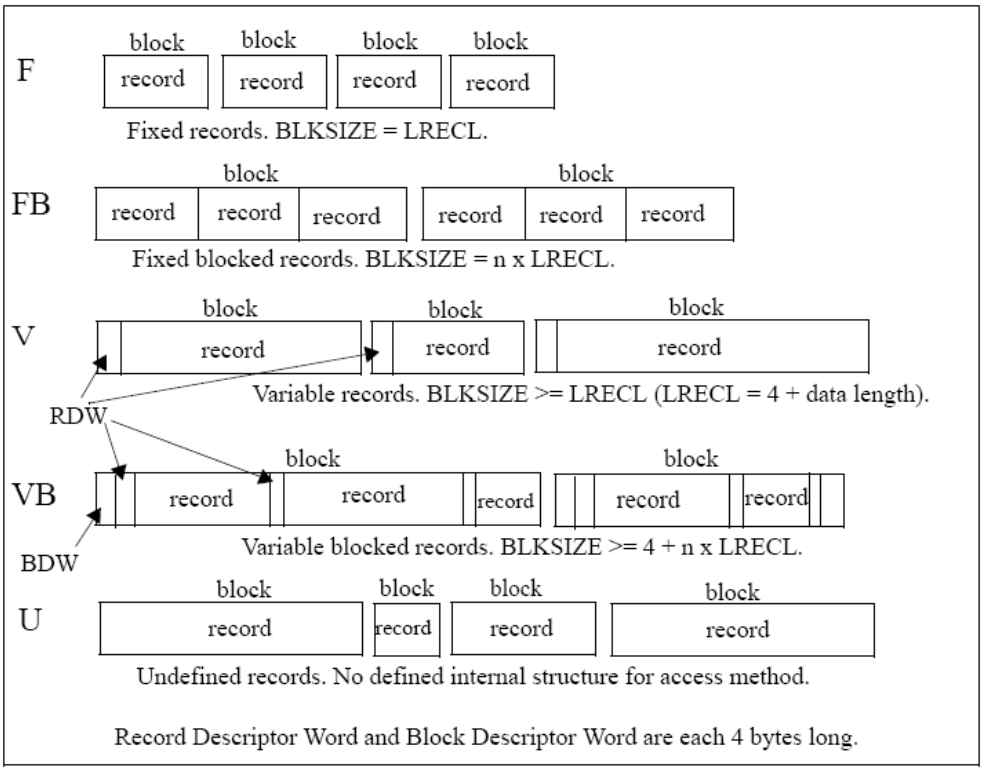


图5-1 基本记录格式

5.9 数据集的类型

z/OS有很多类型的数据集，并用不同的方法来管理它们。本章讨论了三种：顺序，分区和VSAM。这些都是用在磁盘存储上的；我们也会简单提及磁带数据集。

5.9.1 顺序数据集是什么？

z/OS上最简单的数据结构就是顺序数据集。它包含了一条或多条记录，这些记录以物理顺序存储并顺序处理。新记录被插入到数据集末尾。

举个例子，顺序数据集可能是一个行式打印机的输出数据集或者一个日至文件。

z/OS用户通过作业控制语言(JCL)定义一个组织方式为PS(DSORG=PS)的顺序数据集。PS代表物理顺序(physically sequential)。换句话说，数据集中的记录物理上是一个接一个存放的。

本章主要覆盖了磁盘数据集，但主机上的应用程序很多情况下也会用到磁带数据集。磁带上存储顺序数据集。主机磁带驱动器有变长的记录(块)。日常事务处理编程方法中的最大块大小是65K字节。特殊编程可以使用更长的块。很多磁带驱动器产品具有不同的特性。

5.9.2 PDS 是什么？

成员
一个分区数据集(PDS)或扩展分区数据集(PDSE)的一部分。

分区数据集在顺序数据集的简单结构上加了一层组织。分区数据集是顺序数据集的集合，这些顺序数据称为成员。每个成员就像一个顺序数据集，有一个简单的名字，长度最多是8个字符。

PDS包含了一个目录。PDS每个成员在目录中都有一个条目，通过引用(指针)指向成员。成员名按照字母顺序存于目录中，但在库中成员可以按照任何顺序排列。目录使系统可以查找到数据集中某个特定成员。

分区数据集通常被称为库。在z/OS中，库用来存放源程序，系统和应用程序控制参数，JCL和可执行模块。系统数据集基本上都是库。

一旦某个成员更新或增加，PDS的空间就会减少。因此，z/OS用户经常需要压缩一个PDS来恢复减少的空间。

z/OS用户通过数据集组织为PO(DSORG=PO)的JCL定义PDS，PO代表分区组织(partitioned organization)。

库
一个分区数据集，用来存储源程序，参数和可执行模块。

为什么PDS的结构是那样的？

PDS结构这样设计是为了提供对库的有效访问，库中是相关的成员，这些成员可以是装载模块，程序源代码，JCL或者许多其他类型的内容。

许多系统数据集也保存在PDS数据集中，尤其是如果它们是由许多小的，相关的文件组成。例如，关于ISPF面板定义就是存放在PDS数据集中。

ISPF的一个主要用途就是创建和操作PDS数据集。这些数据集典型地由程序源代码，手册文本或帮助屏幕，或用以分配数据集和运行程序的JCL组成。

PDS的优点

PDS数据集提供了一种简单有效的方法来组织相关的顺序文件的集合。对z/OS用户来说，PDS有如下优点：

- 将相关数据集组和放在一个单一名字下使得z/OS数据管理更容易一些。作为PDS成员存储的文件可以单独处理，或者所有成员也可以作为一个单元处理。
- 因为给z/OS分配的地址空间通常从磁盘上的磁道边界开始，所以使用PDS是在一个磁道上存储多个小数据集的好方法。如果您有很多小于一个磁道的数据集，这将为您省下很多空间。在3390磁盘设备上，一个磁道有56,664字节。
- PDS的成员可以做为顺序数据集使用，也可以附加到(或并置到)别的顺序数据集之后。
- 多个PDS数据集可以通过并置组成大的库。
- 用JCL或ISPF很容易创建PDS数据集；用ISPF实用程序或TSO命令很容易操作它们。

PDS的缺点

PDS数据集简单灵活，广泛使用。然而，PDS某些方面的设计影响了磁盘存储的性能和有效使用，如下：

177

- 浪费空间

  当PDS中的一个成员被替代的时候，新数据区域会被写到分配给PDS的存储区内的新区域。当成员被删除的时候，它的指针也被删除，因此就没办法再次使用它的空间。浪费的空间通常称为gas，必须通过重新组织PDS来定期移除，例如通过使用实用程序IEBCOPY来压缩它。

- 受限制的目录大小

  PDS目录大小是在分配时就设置好的。随着数据集增长，可以根据数据集分配的时候定义的二次空间分配的单元数来申请更多的空间。这些额外的单元称为二次分配区(secondary extent)。

  然而，在PDS目录中只能存储固定数量的成员条目因为数据集分配的时候它的大小就已经确定了。如果您需要存储比实际容量更多的成员条目，您必须分配一个新的PDS，包含更多目录块，然后把成员从原PDS中复制过来。这意味着每当您分配一个PDS，您必须计算出需要的目录空间数量。

- 过长的目录搜索

  正如早先提到过的，PDS目录的条目包括了成员名和指向成员位置的指针。条目是以字母顺序存储的。在一个大目录里靠前的地方插入一个条目会导致大量的I/O活动，因为新条目之后所有条目都要往后移动以便给它留出空间。

  条目也是按照字母顺序搜索的。如果目录很大而成员又很小，搜索目录的时间开销，可能比当定位到成员对成员内容检索的时间还要长。

5.9.3 PDSE 是什么？

PDSE是扩展分区数据集。它包括了目录和零个或多个成员，正像PDS一样。它可以由JCL，TSO/E和ISPF创建，正如PDS，它还可以用相同的访问方法处理。PDSE数据集只存储在DASD上，而不能在磁带上。

PDS/PDSE
包含了成员的z/OS 库，比如源程序库。

目录可以根据需要自动扩展，直到522,236个成员的寻址限制。它还包括一个索引，提供对成员名的快速搜索。删除或者移走的成员的空间可以被新成员自动重用，因此您不用压缩一个PDSE以移除浪费的空间。每个PDSE的成员可以有多达15,728,639条记录。PDSE最大可以有123个分区，不过不能扩展到一个卷外。当PDSE的目录在使用中时，它存储在处理器存储中以便更快地访问。

178

PDSE数据集几乎可以代替用来存储数据的所有PDS。但PDSE格式不是为了作为PDS替代品设计的。当PDSE用来存储装载模块时，它把它们存储在称为程序对象的结构中。

PDS和PDSE比较

在很多方面，PDSE和PDS很像。每个成员名可以是8字节长。访问PDS目录或成员时，大多数PDSE接口和PDS接口没有明显区别。PDS和PDSE数据集用相同的访问方法(BSAM，QSAM，BPAM)处理。如果您还不清楚，在一个给定的PDS或PDSE中，成员必须使用相同的访问方法。

然而PDSE有不同的内部格式，提高了它们的可用性。您可以用PDSE来代替PDS存储数据或程序。在PDS中，程序被存储为装载模块。在PDSE中，把程序存储为程序对象。如果您想在PDSE中存储一个装载模块，您必须首先把它转化成程序对象(用IEBCOPY实用程序)。

PDSE数据集有一些特性能够提高用户生产力和系统性能。使用PDSE比PDS优越的主要地方是PDSE能够在数据集内自动重用空间而不需要任何人来定期运行实用程序来重新组织空间。

另外，PDS的目录大小是确定的，不管其中有多少成员，而PDSE的目录大小是可变的，可以根据其中存储的数据集成员数来扩展。

进一步地，当某个成员被删除或代替的时候，系统会自动回收空间，将它返还给可以供同一个PDSE下其他成员分配使用的空间池。返回的空间可以被重用而不需要进行IEBCOPY压缩，。

PDSE的其他优点还有:

- PDSE成员可以共享。这使得当同时修改PDSE的多个单独成员的时候，更容易维护PDSE的完整性。
- 减少目录搜索时间。PDSE目录是索引的，并使用索引来搜索。PDS目录是根据字母顺序组织的，按顺序搜索。系统会将常用的PDSE的目录缓存在内存中。
- 同时创建多个成员。例如，可以为一个PDSE打开两个DCB，同时写两个成

179

员。

- PDSE数据集包含了多达123个分区。分区是DASD存储卷上的一块连续区域，为某个特定的数据集所占用或保留。
- 当写到DASD卷上的时候，逻辑记录被从用户块中抽取出来并重新组块。被读取的时候，PDSE上的记录被重新组合成定义在DCB中的块大小。用于重组的块大小可以和原块大小不同。

5.9.4 数据集何时用完空间

正如前面提到的，分配数据集的时候，您在磁盘上保留了一定数量的以块，磁道或柱面为单位的空间。如果分配的空间用完了，系统会显示SYSTEM ABEND ‘0D37’,或者也有可能是B37或E37。

我们在本文中还没有讨论异常结束或ABEND，但遇到的时候，这是您不得不处理的问题。如果您正处于编辑状态，除非解决了问题，否则无法退出该会话。

解决空间存储异常中止问题您可以如下操作：

- 如果数据集是PDS，您可以通过以下这些方法来压缩数据集：
  a. 分屏(PF2)，选择UTILITIES(选项3)。
  b. 在Utility Selection 菜单面板中，选择LIBRARIES(选项1)。
  c. 指定数据集的名字，在选择栏里输入C。
  d. 当数据集压缩完的时候，您会看到以下信息：COMPRESS SUCCESSFUL
  e. 可以按PF9切到编辑会话，保存新内容。

- 分配一个更大的数据集，通过以下方法将内容拷贝进去：
  a. 分屏(PF2)，选择UTILITIES(选项3)，然后在分屏的另一端选择DATASET(选项2)。
  b. 指定收到异常中止信息的数据集的名字，回车查看其信息
  c. 分配一个有更大空间的数据集。
  d. 在Utility Selection菜单面板上选择MOVE/COPY(选项3)来把成员从原来的数据集复制到这个新的更大的数据集。
  e. 浏览(选项1)新数据集，确保所有成员都正确复制。
  f. 切换(PF9)到异常中止的编辑会话屏，在输入内容的最顶行和最底行输入CC，在命令栏里输入CREATE，回车。

  g. 输入该新的更大的数据集名字以及一个成员名来接受复制信息。
  h. 您将再次看到异常中止的编辑会话。在命令行里输入CAN。按RETURN键(PF4)来退出编辑会话。
  i. 在Utility Selection菜单面板上选择DATASET(选项2)来删除原来的数据集。
  j. 将新数据集更名为原数据集。

- 通过在命令行里输入CAN来取消在编辑会话里输入的新内容，。这样就能在没有异常中止的情况下退出了；然而前面所有没保存的信息就丢失了。

5.10 VSAM 是什么？

术语虚拟存储访问方法(VSAM)既是一种数据集类型，也是一种访问方法，用来管理各种用户数据类型。作为一种访问方法，VSAM比其他磁盘访问方法提供了更多更复杂的功能。VSAM以一种唯一的，其他访问方法不理解的格式来保存磁盘记录。

VSAM
一种访问方法，对定长和变长记录进行直接或顺序处理。

VSAM主要用在应用程序上。它不用来存放源程序，JCL或可执行模块。VSAM文件通常不能用ISPF显示和编辑。

您可以使用VSAM组织记录，放到四种类型的数据集中：键顺序数据集，进入顺序数据集，线性数据集，或相关记录数据集。这四类数据集的不同主要是它们的记录存储和访问的方式不同。

VSAM数据集可以简要描述如下：

- 键顺序数据集(KSDS)

  这是VSAM最常见的格式。每个记录有一个或多个键域，记录可以通过键读取(或插入)。这提供了对数据的随机访问。记录是变长的。

- 进入顺序数据集(ESDS)

  这种形式的VSAM顺序存放记录。记录可以顺序访问。在IMS，DB2和z/OS UNIX上使用。

- 相对记录数据集(RRDS)

  这种VSAM格式允许通过编号来读取记录；记录1，记录2等等。这提供了随机访问，并假设应用程序有得到所需要记录编号的方法。

- 线性数据集(LDS)

  实际上，LDS是字节流数据集并且在传统的z/OS文件(相对于z/OS UNIX文件)中它是唯一的字节流数据集。许多z/OS系统功能大量使用这种格式，但在应用程序中很少使用。

在VSAM中还有几种附加的访问数据方法，这里没有列出。大部分应用程序将VSAM用于键控数据。

VSAM具有逻辑数据区域，称为控制区间(CI)， 如图5-2所示。CI的缺省大小是4K字节，但最大可以到32K字节。CI包含了数据记录，未使用的空间，记录描述符域(RDF)和一个CI描述符域。

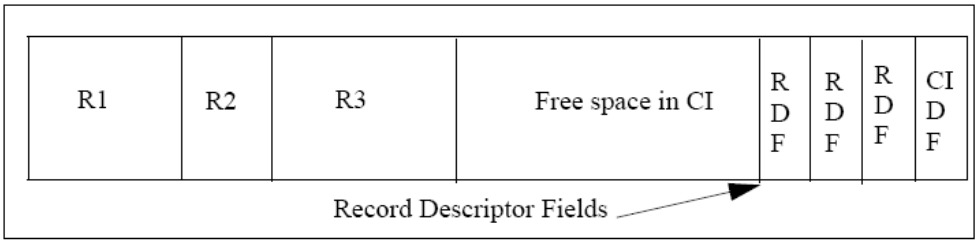


图5-2 简单的VSAM控制区间

多个CI位于一个控制区域(CA)中。VSAM数据集包括了控制区域和索引记录。索引记录的一种形式是顺序集合，是指向一个控制区间的最低级别的索引。

VSAM数据总是变长的，并且记录在控制区间中自动组块。RECFM属性(F，FB，V，VB，U)不适用于VSAM，BLKSIZE属性也是。您可以使用访问方法服务(AMS)实用程序来定义和删除VSAM结构，比如文件和索引。例5-1显示了一个例子。

182

例5-1 用AMS定义一个VSAM KSDS

```
DEFINE CLUSTER -
(NAME(VWX.MYDATA) -
VOLUMES(VSER02) -
RECORDS(1000 500)) -
 DATA -
(NAME(VWX.KSDATA) -
 KEYS(15 0) -
RECORDSIZE(250 250) -
BUFFERSPACE(25000) ) -
INDEX -
(NAME(VWX.KSINDEX) -
CATALOG (UCAT1)
```

在这里的简单描述中，许多VSAM处理的细节都没有包括。许多处理是通过VSAM透明处理的；应用程序只需要基于键值读取，更新，删除或者添加记录。

5.11 目录和 VTOC

z/OS使用目录和每个DASD上的卷目录表(VTOC)来管理数据集的存储和布置；分别在以下篇目中描述：

- 183页上的"什么是VTOC"
- 184页上的"什么是目录？"

z/OS还可以把基于历史相关数据的数据集组成组，这在187页上的"什么是世代数据组"中描述。

5.11.1 什么是 VTOC?

VTOC
包含了数据集标签的结构。

z/OS要求一种特定的磁盘格式，如图5-3所示。在第一个柱面的第一个磁道的第一个记录上提供了卷的标签。它包括一个6字符长的卷序列号(volser)和一个指向卷目录表(VTOC)的指针，VTOC可以位于磁盘的任何地方。

VTOC列出了该卷上所有的数据集，以及每个数据集的位置信息和大小还有一些其他的数据集属性。一个标准的z/OS实用程序ICKDSF可以用来创建标签和VTOC。

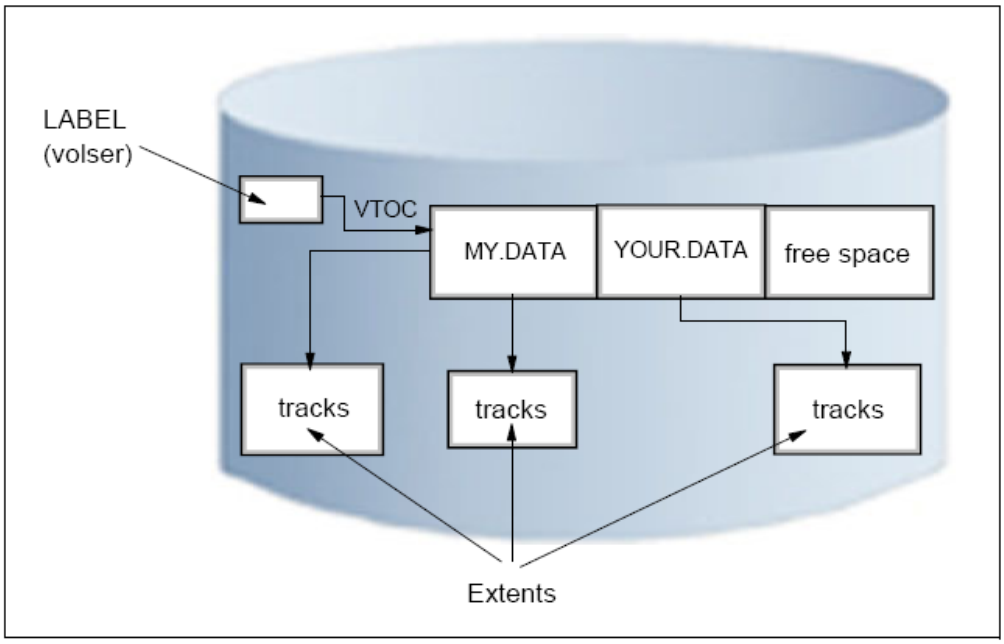


图5-3 磁盘标签，VTOC和分区

当一个磁盘卷用ICKDSF来初始化时，可以指定VTOC的位置和大小。大小是可变的，可以从几个磁道到100个磁道，这取决卷的预计使用情况。磁盘卷上的数据集越多，要求VTOC的空间越大。

VTOC上也有卷上所有自由空间的入口。为数据集分配空间会导致系统程序检查并更新自由空间记录，并且创建一个新的VTOC条目。数据集通常是整数个磁道(或柱面)，并从一个磁道(或柱面)开始处开始分配。

您还可以创建带索引的VTOC。VTOC索引事实上是一个叫SYS1.VTOCIX.volser的数据集，它包含的条目按照数据集名字排序，并且每个条目中有一个指针指向相应的VTOC条目。它还包含了卷上自由空间的位图。VTOC索引使用户更快地找到数据集。

5.11.2 什么是目录?

目录描述了数据集属性并显示了数据集所在的卷。当一个数据集被编目的时候，它可以通过名字被引用而不需要用户指定它存储的位置。数据集可以被编目，取消编目或重新编目。所有系统管理的DASD数据集都自动编目在一个目录中。磁带上的数据集不要求编目，但它可以简化用户的工作。

184

在z/OS中，主目录和用户目录存储了数据集的位置。磁盘和磁带数据集都可以编目。

为了定位您请求的数据集，z/OS必须知道三条信息：

目录
描述数据集属性，包括数据集的位置。

- 数据集名
- 卷名
- 设备单元(卷设备的类型，如3390磁盘或3590磁带)

您可以通过ISPF面板或JCL语句指定这些参数。然而设备单元和卷通常和一个终端用户或应用程序无关。一个系统目录用来存储和读取一个数据集的设备单元(UNIT)和卷(VOLUME)位置。以它最基本的形式，目录可以为已经编目的任何数据集提供单元设备类型和卷名。系统目录提供了一个简单的查找功能。有了这个工具，用户只需要提供数据集名。

主目录和用户目录

z/OS系统中，通常至少有一个主目录。如果z/OS系统只有一个目录，那么这个目录就是主目录，并且所有数据集的入口位置都保存在这里。但是，单一的一个目录既不够高效，又不够灵活，所以一个典型的z/OS系统都会使用一个主目录和许多与之相连的用户目录，如图5-4所示。

用户目录用来存储数据集的名字和位置(dsn/volume/unit)。主目录通常只存储数据集的HLQ及其对应的用户目录的名字，该用户目录包含所有以HLQ开头的数据集名。HLQ称为别名。

在图5-4中，主目录的数据集名是SYSTEM.MASTER.CATALOG。该主目录存储了所有以SYS1开头的数据集的全名和位置，如SYS.A1。主目录中还定义了两个HLQ(别名)的入口，IBMUSER和USER。定义IBMUSER的语句包括用户目录的数据集名，该用户目录包含具有全部限定词的IBMUSER数据集和它们各自的位置。USER HLQ(别名)也是这样。

当请求SYS.A1时，主目录向请求者返回位置信息，卷名(WRK001)和设备单元(3390)。当请求IBMUSER.A1时，主目录将请求传递给USERCAT.IBM，然后USERCAT.IBM向请求者返回数据集IBMUSER.A1的位置信息。

185

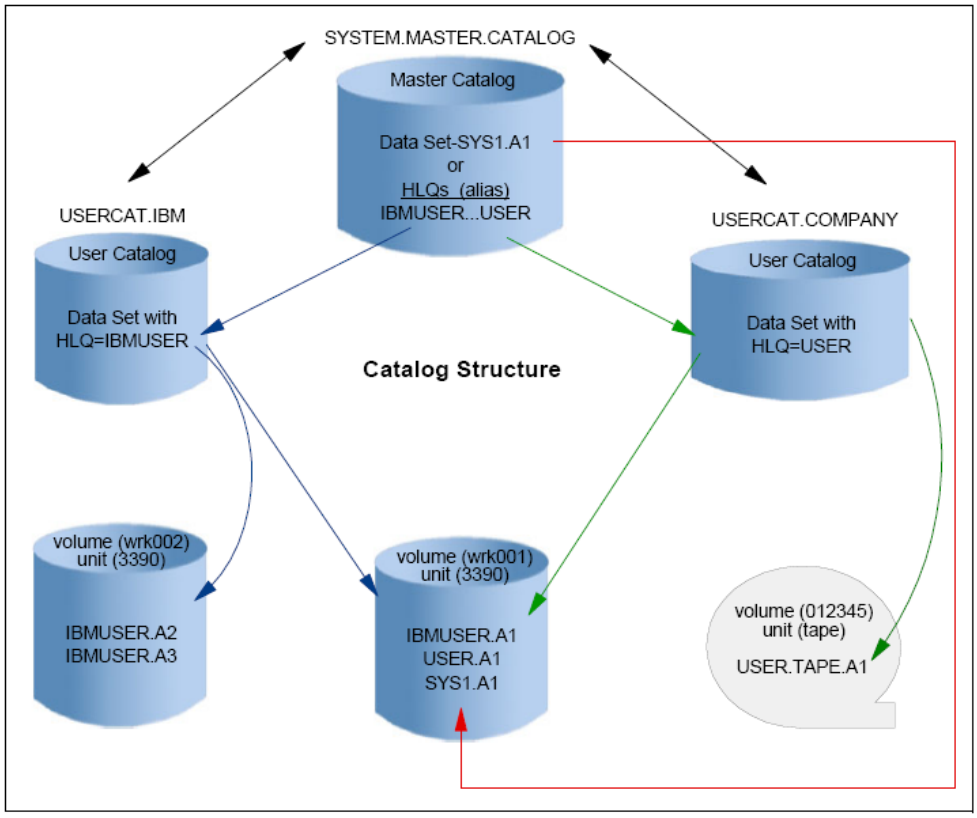


图5-4 目录概念

更进一步举个例子，看下面的DEFINE语句：

```
DEFINE ALIAS(NAME(IBMUSER) RELATE(USERCAT.IBM))
DEFINE ALIAS(NAME(USER) RELATE(USERCAT.COMPANY))
```

这用来将IBMUSER和USER别名及其用户目录名放在主目录中，用户目录中存储数据集的全名和位置信息。如果IBMUSER.A1被编目了，一个把它分配给作业的JCL语句可以是：

```
//INPUT DD DSN=IBMUSER.A1,DISP=SHR
```

如果IBMUSER.A1没有被编目，把它分配给作业的JCL语句可以是：

```
//INPUT DD DSN=IBMUSER.A1,DISP=SHR,VOL=SER=WRK001,UNIT=3390
```

186

作为一般规则，z/OS中所有数据集都是被编目的。系统中很少需要没有编目的数据集，没有编目的数据集一般只用于与恢复问题或安装新软件相关的方面。通过ISPF创建的数据集都是自动编目的。

使用备用主目录

那么，如果一个系统丢失了它的主目录或者主目录某种程度遭到了破坏了怎么办？如果发生这种事，可能会引起严重的问题，需要迅速的恢复操作。

为了解决这个潜在的问题，大多数系统程序员会给主目录定义一个备份。系统程序员在系统启动的时候就指定这个备份主目录。在这种情况下，建议系统程序员把备份主目录放在和主目录不同的卷上(为了防止卷不可用)。

5.11.3 什么是世代数据组？

在z/OS中，编目连续的更新或是几代相关的数据是可能的。它们被称为"世代数据组(GDG)"。

每个GDG内的数据集被称为一代或世代数据集(GDS)。世代数据组(GDG)是按时间顺序排列的一些历史相关的非VSAM数据集的集合。即，在这个组中，每个数据集和其他数据集都是历史相关的。

在一个GDG中，这些代可以有类似或不类似的DCB属性以及数据集组织方式。如果一个组中所有代的属性和组织结构都是一样的，那么这些代就可以作为一个单独的数据集被读取。

把相关数据集组成组有很多优点，比如：

- 这个组中所有的数据集可以用一个共同名字引用
- 操作系统可以按时间顺序把这些代排列
- 过时或者废旧的代会由系统自动删除

世代数据集按顺序排序代表了它们年龄的绝对名和相对名。操作系统的目录管理程序使用绝对世代名。较老的数据集使用较小的绝对数字。相对名字是有符号的整数，用来指向最新的代(0)，次新的代(-1)等等。

举个例子，数据集名字为LAB.PAYROLL(0)指向组中最新的数据集；LAB.PAYROLL(-1)指向次新的数据集等等。相对数字还可以用来编目新的一个代(+1)。世代数据组(GDG)的基需要在世代数据集编目之前分配在一个目录中。每个GDG由GDG的基入口表示。

对新的非系统管理数据集来说，如果您不指定卷而且数据集没有打开，系统就不编目该数据集。新的系统管理数据集在分配的时候总是被编目的，从存储组(storage group)中获取卷信息。

5.12 DFSMS 在空间管理中的角色

在z/OS系统中，空间管理涉及了数据集的分配，布置，监控，迁移，备份，回调，恢复和删除。这些动作可以人工完成也可以通过自动进程完成。如果数据管理是自动完成的，操作系统决定目标的布置，并自动管理数据集的备份，移动，空间和安全。典型的z/OS生产系统既包括了人工，也包括了自动进程来管理数据集。

根据z/OS系统和它的存储设备的配置，用户或者程序可以直接控制数据集使用的许多方面，在早期的操作系统中，要求用户来做这些工作。然而，z/OS客户渐渐

地依赖于安装指定的配置(为数据和资源管理以及空间管理产品)，如DFSMS，来自动管理数据集的存储。

数据管理主要包括这些任务：

- 在DASD卷上分配空间
- 通过名字自动读取编目数据集
- 在驱动器中安装磁带
- 在应用程序和媒介之间建立逻辑连接
- 控制对数据的访问
- 在应用程序和媒介之间传输数据

z/OS中管理空间的主要方法是通过操作系统的DFSMS部件。DFSMS执行系统中一些基础的数据，存储，程序和设备管理功能。DFSMS是一些产品的集合，其中一个产品叫DFSMSdfp是z/OS运行所必需的。DFSMS，和硬件产品以及安装时指
188 定的数据和资源管理设置一起，在z/OS环境下提供系统管理存储功能。

DFSMS的核心是存储管理子系统(SMS)。使用SMS，系统程序员或存储管理员定义自动化管理存储和硬件设备的策略。这些策略描述了系统的数据分配属性，性能和可用性目标，备份和保持需求，以及存储需求。SMS管理系统的这些策略，并且交互式存储管理工具(ISMF)提供了用户界面来定义和维护这些策略。

通过SMS分配的数据集称为系统管理数据集或SMS管理数据集。当您分配或定义一个数据集使用SMS时，您通过数据类，存储类和管理类来指定数据集需求。通常您不必指定这些参数，因为存储管理员已经建立了自动类选择(ACS)程序，用它为给定的数据集决定应该使用哪些类。

SMS
存储管理子系统。

DFSMS提供了一套Construct，用户接口和程序的集合(使用DFSMS产品)来帮助存储管理员。DFSMS的核心逻辑，如ACS程序，ISMF代码，和Construct，它们驻留在DFSMSdfp™中。而DFSMShsm™和DFSMSdss™包含在管理类的Construct中。用DFSMS，z/OS系统程序员或存储管理员可以定义性能目标和数据可用性需求，为典型的数据集创建模型数据定义，自动化数据备份。基于系统策略，DFSMS可以在数据集创建时自动为它们分配这些服务和数据定义属性。IBM存储管理相关产品决定数据布置，管理数据备份，控制空间使用，以及提供数据安全。

5.13 z/OS UNIX 文件系统

把UNIX文件系统想象成一个容器，里面装着全部UNIX目录树的一部分。不象传统的z/OS库，UNIX文件系统是层次的，面向字节的。在UNIX文件系统里查找文件是通过查找一个或多个目录来完成的(见图5-5)。它没有用于直接指向文件的z/OS目录的类似概念。

189

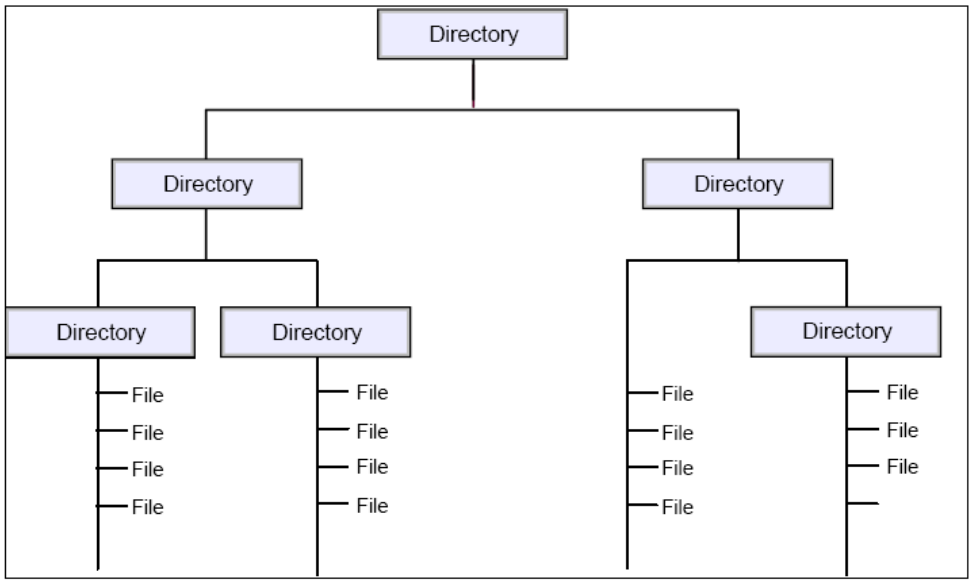


图5-5 分层文件系统结构

z/OS UNIX系统服务(z/OS UNIX)允许z/OS用户在z/OS上创建UNIX文件系统和文件系统目录树，并允许访问z/OS和其他系统上的UNIX文件。在z/OS中，UNIX文件系统由系统程序员(或者是一个有安装权限的用户)安装在一个空目录下。

在z/OS UNIX中可以使用如下几种文件系统类型：

- z系列文件系统(zSeries File System，zFS)，是一种在VSAM线性数据集中存储文件的文件系统
- 分层文件系统(Hierarchical file system，HFS)，是一种可装载的文件系统，正在被zFS逐渐淘汰。
- z/OS网络文件系统(z/OS Network File System，z/OS NFS)，允许z/OS系统通过TCP/IP访问远程UNIX(z/OS或者非z/OS)文件系统，好像它是本地z/OS目录树的一部分一样。
- 临时文件系统(Temporary file system，TFS)，是一个临时的，内存中的物理文件系统，它支持存储内可装载的文件系统。

和其他UNIX文件系统一样，一个路径名可以确定一个文件，它包含目录名和一个文件名。一个有完全限定的文件名可以有1023个字节长，它由该文件路径中的每个目录名和文件名自身组成的。

190

路径名是由独立的目录名和文件名组成，它们之间用'/'符号分隔开，例如：

```
/dir1/dir2/dir3/MyFile
```

正如UNIX，z/OS UNIX对文件名和目录名大小写敏感。例如，在同样的目录下，文件MYFILE和MyFile是不一样的。

分层文件系统中的文件是顺序文件，并且作为字节流访问。除了应用程序定义的结构之外，这些文件中并不存在记录的概念。

包含了UNIX文件系统的zFS数据集是一个z/OS数据集类型(一种VSAM线性数据集)。zFS数据集和z/OS数据集可以驻留在相同的DASD卷上。z/OS提供命令来管理zFS空间使用。

zFS和现存的z/OS文件系统管理服务的集成提供了一种自动化文件系统管理的能力，这种能力可能在其他UNIX平台上不可用。这种集成使得文件拥有者能花费更少的时间在一些工作上，如备份和重新恢复整个文件系统。

5.13.1 z/OS 数据集和文件系统中的文件对比

UNIX的很多元素在z/OS操作系统中都有相似体。例如，在文件系统中，一个用户目录的组织结构就与文件系统中的用户目录(/u/ibmuser)是相似的。

在z/OS系统中，分配给z/OS数据集的用户前缀指向了一个用户目录。通常情况下，用户拥有所有的名字以其用户前缀开头的数据集。例如，属于TSO/E用户ID为IBMUSER的数据集都是以高级限定符(前缀)IBMUSER开头的。这些不同的数据集名可以是IBMUSER.C, IBMUSER.C.OTHER以及IBMUSER.TEST。

在UNIX文件系统中，ibmuser可以有一个指定的用户目录/u/ibmuser。在该目录下，可以有个指定的子目录/u/ibmuser/c，并且/u/ibmuser/c/pgma将指向文件pgma(如图5-6所示)。

在z/OS数据集的所有类型中，分区数据集(PDS)和文件系统中的用户目录相似。在一个分区数据集中，如IBMUSER.C，可能有PGMA，PGMB等成员(文件)。例如，您可能有IBMUSER.C(PGMA)和IBMUSER.C(PGMB)。相似的，一个诸如/u/ibmuser/c的子目录可以存放许多文件，如pgma，pgmb等等。

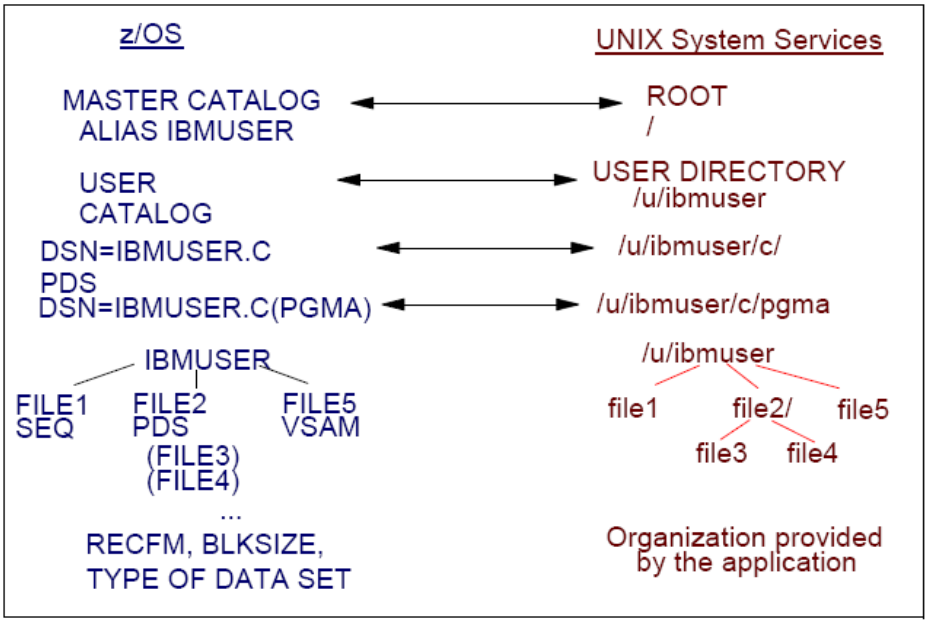


图5-6 z/OS数据集和文件系统文件的比较

所有数据一旦被写入分层文件系统，都可以马上被程序读取。当程序调用fsync()时，数据就被写入了磁盘。

5.14 zFS 文件系统的使用

除了分层文件系统(HFS)，z/OS分布式文件服务(DFS™)z系列文件系统(zFS)也是一个可以使用的z/OS UNIX系统服务(z/OS UNIX)文件系统。zFS文件系统包括了可以由z/OS UNIX应用程序接口(API)访问的文件和目录。这些文件系统可以支持访问控制列表(ACL)。zFS文件系统可以和其他一些本地的(或远程的)文件系统类型(比如HFS，TFS，AUTOMNT和NFS)一起安装到z/OS UNIX层次上。

分布式文件服务消息块(SMB)提供了一个服务器端使得UNIX文件和数据集对SMB客户端是可用的。支持的数据集包括顺序数据集(在DASD卷上)，PDS和PDSE，以及VSAM数据集。数据集支持通常被称为记录文件系统(RFS)支持。SMB协议通过使用z/OS上的TCP/IP得到支持。这个通讯协议允许客户端访问共享目录路径和共享打印机。个人电脑(PC)客户端可以使用他们操作系统里包含的文件和打印共享功能。

支持的SMB客户端包括Windows XP Professional，Windows 2000服务器之上的Windows终端服务器，Windows 2003上的Windows终端服务器和LINUX。同时，这些文件可以被本地的z/OS UNIX应用程序和DCE DFS客户端共享。

相关阅读：在IBM出版物《z/OS DFS Administration》有关于使用DFS使用的描述。您可以在z/OS因特网书库网站上找到这本书及相关出版物。

http://www.ibm.com/servers/eserver/zseries/zos/bkserv/

5.15 总结

数据集是逻辑上相关的数据的集合；它可以是一个源程序，一个程序库，或由一个处理程序使用的数据记录组成的文件。数据集记录是处理程序使用的基本信息单元。

在使用数据集之前，用户必须定义一定的空间来分配数据集，或者通过使用DFSMS自动分配空间。用DFSMS，z/OS系统程序员或存储管理员可以定义性能目标和数据可用性要求，为典型的数据集创建模型数据定义以及自动建立数据备份。根据系统策略，当创建数据集时，DFSMS可以自动分配这些服务和数据定义属性给数据集。其他和存储管理相关的产品可以用来决定数据位置，管理数据备份，控制空间使用，以及提供数据安全。

几乎所有的z/OS数据处理都是面向记录的。虽然字节流文件是z/OS UNIX的一个标准部分，但是在传统的处理中是不存在字节流文件的。z/OS记录和物理块采用定义好的几个格式之一。大多数的数据集有DCB属性，这一属性包含记录格式(RECFM-F，FB，V，VB，U)，最大逻辑记录长度(LRECL)和最大块大小(BLKSIZE)。

z/OS库就是分区数据集(PDS或PDSE)，它包含成员。源程序，系统和应用程序控制参数，JCL和可执行模块几乎都存在库中。

虚拟存储访问方法(VSAM)是一种访问方法，它比其他磁盘访问方法提供了复杂的多的功能。VSAM主要是为应用程序服务的，不能通过ISPF编辑。

z/OS数据集的名字由最多不超过44个大写字符组成，它们被句点分成若干个限定词，每个限定词的名字最多有8个字符。高级限定词(HLQ)可能由系统安全控制来确定，但数据集名的其余限定词就由用户来分配。这些名字的命名存在着一些命名约定。

当知道数据集的名字，卷名和设备类型时，就可以定位一个存在的数据集。如果数据集已经被编目，那么只需知道该数据集名即可。尽管系统目录的数据可能分布在主目录和许多用户目录中，系统目录也只是一个简单的逻辑功能。事实上，几乎所有的磁盘数据集都被编目。这样做的副作用是所有(已编目的)数据集必须拥有唯一的名字。

UNIX文件系统中的文件可以是文本文件或二进制文件。在文本文件中，文本的每一行都通过一个新行分隔符独立开来。二进制文件包括二进制字(字节流)序列，除了由应用程序定义的结构外，不存在记录概念。读取文件的应用程序负责解释数据格式。z/OS把整个UNIX文件层次作为数据集的集合。每个数据集是一个可装载的文件系统。

本章关键术语		
块大小(block size)	目录(catalog)	数据集(data set)
高级限定词(high-level qualifier，HLQ)	库(library)	逻辑记录长度(logical record length，LRECL)
成员(member)	PDS/PDSE	记录格式(record format，RECFM)
系统管理存储(system-managed storage，SMS)	虚拟存储访问方法(virtual storage access method，VSAM)	VTOC

5.16 复习题

为了帮您更好地理解本章的内容，完成下面的问题：

1. 数据集是什么？z/OS上使用了哪些类型的数据集？
2. z/OS为什么需要数据集具有唯一的名字？
3. 为什么使用PDS？
4. 应用程序使用库吗？为什么使用或为什么不是用？
5. 传统的UNIX系统可以使用的最大文件是由什么决定的？在z/OS中有相同的限制吗？

194

6. 您看到过任何形式的用临时数据名的数据集吗？

7. 在JCL流中，什么特殊字符用来指定一个临时数据集？

8. 通过ISPF3.4给出的数据集信息很有用。为什么不在基本的数据集列表面板中显示所有的信息？

9. 我们在练习中创建了一个源库，指定了80字节固定长度的记录。为什么？

10. 班级练习可以使用的磁盘卷是WORK02。您能在其他卷里分配一个数据集吗？在任何卷上呢？

11. 数据集的哪些信息是存储在目录中的？如果数据集不在目录中，需要指定什么DD操作数？

12. 主目录和用户目录之间有什么区别？

5.17 练习题

本章的实验练习将帮助您锻炼使用ISPF操作数据集的技能。这些技能对于完成本书后面章节的练习是很有帮助的。

为了完成实验练习，用户需要一个TSO用户ID和密码(可以找老师帮忙)。

练习主要内容如下：

- “探索ISPF选项3.4”，在第196页
- “用ISPF 3.2分配数据集”，在第197页
- “复制源库”，在第198页
- “使用数据集成员”，在第198页
- “列出数据集和其他ISPF 3.4选项”，在第199页
- “执行目录搜索”，在第200页

提示：3270回车键和PC回车键通常会混在一起。大多数3270模拟器允许用户把这些功能分配到键盘的任何键上，我们假设3270回车功能分配给了右CTRL键。然而，一些z/OS用户更喜欢用大的PC回车键来执行3270回车功能，并把Shift-Enter(或数字回车键)来执行3270的新行功能。

195

5.17.1 探索 ISPF 选项 3.4

ISPF面板中最有用的就是选项3.4。这个术语是指，从ISPF的主选项菜单开始，选择选项3(Utilities)，接着选择选项4(Dslist，代表数据集列表)。这个可以简化为在主菜单中输入”3.4”，或是从任何面板上输入”=3.4”。

许多ISPF用户都几乎只在选项3.4面板中操作。这里我们只介绍”选项3.4”的部分功能，其余的将在本文后面章节的练习中介绍。”选项3.4”操作要十分注意，操作的

结果可能会对个人甚至在系统范围内产生影响。

z/OS用户通常使用ISPF选项3.4来检查DASD卷上的数据集，或查看特定数据集的属性。用户可能需要知道：

- 哪些数据集存在这个卷上？
- 卷上有多少个不同的数据集类型？
- 某特定数据集的DCB属性是什么？

下面让我们用卷WORK02，或者用您的老师指定的另外一个卷，作为一个样例来回答这些问题：

1. 在3.4面板中，在‘Volume Serial’域输入WORK02，不要在‘Option➔’行或是在‘Dsname Level’域中输入任何字符。

2. 用PF8和PF7在产生的数据集列表之间上下翻页

3. 用PF11和PF10左右翻动显示更多的信息。这种情况并不是真正的滚动；只有当使用PF11 或PF10 时才能获得额外的信息。

第一个PF11显示的内容包括磁道，使用百分比，分区(XT)和设备类型。XT值是用于获得显示的磁道总数的分区数量。ISPF实用程序功能可以看到一些数据集实际使用的空间大小，并且如果可能的话，它还可以作为一个百分比来显示。

第二个PF11显示的内容包括DCB特性：数据集组织形式(DSORG)，记录格式(RECFM)，逻辑记录长度(LRECL)和块的长度(BLKSIZE)。

PS	顺序数据集(QSAM，BSAM)
PO	分区数据集
VS	VSAM数据集
blank(空格)	未知组织方式(或者内容为空)

RECFM，LRECL和BLKSIZE应该是大家很熟悉的。在某些情况下，若没有使用一个标准的访问方法或没有写入数据时，这些参数是不能确定的。VSAM数据集没有这些参数的直接等价物，通常是以问号来表示。

196

来看另一个卷，可以观察到更大范围的属性。老师可以提供卷号。另一种查看这种卷的方法是使用选项3.2来查看SYS1.PARMLIB在哪里，然后查看那个卷。

5.17.2 用 ISPF3.2 分配数据集

ISPF提供了一种方便的方式来创建新数据集。在该练习中，您创建一个新的库，您可以稍后使用它来存储程序源数据。新的数据集应该放在WORK02卷上，并且命名为yourid.LIB.SOURCE(这里yourid就是您的学生用户ID) 。

对于该练习，假设对于数据集10个磁道的主空间和5个磁道的二次分配区是足够的，并且10个目录块空间也是足够的。更进一步，我们知道我们像在库中存储80字节固定长度的记录。我们可以如下操作：

1. 打开ISPF的主菜单。

2. 选择选项3.2或先选择选项3(Utility)，再选择选项2(Data Set)。

3. 在选项区域(Option➔)输入字母A但是不要按ENTER键。

4. 同时在‘Data Set Name’区域中输入新数据集的名字，但是同样也不要按ENTER键。名字可以用单引号引起来(例如，’yourid.LIB.SOURCE’)，或不用引号引起来(LIB.SOURCE)以便TSO/ISPF自动地把当前的TSO用户ID作为HLQ使用。

5. 在‘Volume Serial’区域输入WORK02，然后按回车键。

6. 完成指定的参数设置并且按回车。

   - Space Units = TRKS
   - Primary quantity = 10
   - Secondary blocks = 5
   - Directory blocks = 10
   - Record length = 80
   - Block size = 0(告诉z/OS选择一个最合适的值)
   - Data set type = PDS

这样应该在WORK02上分配了一个新的数据集。检查屏幕右上角，出现如下的信息：

```
Menu RefList Utilities Help
------------------------------------------------------------------------
-
Data Set Utility Data set allocated


Option ===>
A Allocate new data set C Catalog data set
.....
```

197

5.17.3 复制源库

练习里需要一些源程序，这些程序在WORK02卷上的ZPROF.ZSCHOLAR.LIB.SOURCE里。复制数据集(包括库)有几种方法。我们使用下面的方法：

1. 选择ISPF选项3.3 (Utilities，Move/Copy)

2. 在第一个面板上：

   a.在选项区输入C。
   b.在数据集名字区域输入”ZPROF.ZSCHOLAR.LIB.SOURCE”。这种情况下

需要单引号。

c.不需要卷序列号因为数据集已经被编目了。

d.按回车键。

3. 在第二个面板上：

a.在数据集名字区域中输入”yourid.LIB.SOURCE”然后按回车键。如果这个PDS不存在，输入1以便继承源库的属性。这样就会产生一个列出了输入库中所有成员的面板：

b.在每个成员名字前输入S，并且按回车键。

这样就把所有指定的成员从源库复制到目标库里了。我们可以把输入数据集指定为’ZPROF.ZSCHOLAR.LIB.SOURCE(*)’；这样就会自动复制所有成员。这是将通配符用在z/OS数据集名字中的不多见的案例中的一个。

4. 创建另一个库，移动LIB.SOURCE中的几个成员到新库下。可以取名”yourid.MOVE.SOURCE”。确保复制过来的成员位于新库中，而不再存在于原来的那个库中。然后再把这些成员复制回LIB库。确认他们存在于两个库中。

5.重命名MOVE库中的一个成员名。把MOVE库重命名为”yourid.TEST.SOURCE”。

5.17.4 使用数据集成员

系统提供了几种方法可以向数据集中添加一个新成员。假定我们这里想将TEST2的成员加入到我们先前创建的数据集中：

198

1.在ISPF主菜单中使用选项2。

2.输入数据集的名字，但是不指定成员名字，例如yourid.JCL。这样会列出数据集中已有的成员名列表。

3.确认成员EDITTEST和您之前使用的内容相同：

a.有必要的话，可以滚动，这样就可以看到成员名EDITTEST。

b.把鼠标移到这一行的左边。

c.输入S并按回车。

d.查看您先前的作业，确保它没有被改变。

e.按PF3退出成员EDITTEST。您会再次看到库成员名列表。

4.在屏幕顶部的命令行输入S TEST2，并按回车键。(S TEST2可以被解读成”Select TEST2”。)这样就创建了成员TEST2，并把屏幕设置为输入模式。

5.用我们先前讨论过的命令或功能，任意输入几行内容。

6.按PF3保存TEST2然后退出。

7.再按PF3从ISPF的编辑功能中退出。

以后在编辑内容或使用其他ISPF功能时，我们会简单地说”输入xxx”。意思是(1)键入xxx(2)按回车。新行键(上面印着Enter)只用来定位光标在屏幕上的位置。

5.18 列出数据集和其他 ISPF 3.4 选项

在ISPF的3.4面板上的‘Dsname Level’区域输入yourid，并按回车键。这个操作将列出系统中以yourid为HLQ的所有已编目的数据集。另一种选择是把‘Dsname Level’区域空着，什么都不填，在卷序列号区域输入WORK02；这将会列出指定卷上的所有数据集。(如果这两个区域都使用了，那么列表将只包括出现在指定卷上的，已编目的具有匹配的HLQ的数据集。)

通过在一个数据集名字前面输入合适的字母可以调用许多功能。例如，将光标放在一个数据集名字前并且按PF1(帮助)。帮助面板将列出所有在3.4面板的数据集列表中可用的行命令。如果没有理解这些功能就不要使用它们。这些功能并不都与本课程有关。有关的命令是：

E 编辑数据集

B 浏览数据集

D 删除数据集

R 重命名数据集

Z 压缩PDS库来恢复丢失的空间

C 编目数据集

U 取消编目数据集

当显示一个成员列表时(当一个库被编辑或者浏览时)，如下几个行命令可用：

S 选择该成员编辑或浏览

R 重命名一个成员

D 删除一个成员

5.18.1 执行目录搜索

我们可以使用ISPF 3.4选项在部分名字上进行目录搜索。要了解这个重要功能的更多细节可以使用PF1帮助，如下：

1. 选择选项3.4。

2. 按PF1帮助，选择‘Display a data set list’。按回车键将在信息面板上滚动。

3. 然后选择‘Specifying the DSNAME LEVEL’。按回车键在信息面板上滚动。

4. 按PF3从帮助功能里退出来。

注意3.4‘DSNAME LEVEL’区域不使用引号，在这个区域中当前的TSO/E用户ID不会自动用作名字的前缀。对于TSO中指定数据集名字的一般规则，这是很少几个例外之一。

200

6

第 6 章 使用 JCL 和 SDSF

目标：作为一名大型机计算领域的专业技术人员，您将需要懂得JCL，该语言告诉z/OS处理一个批作业或者启动一个系统任务时需要什么资源。

结束这一章的学习之后，您将能够：

- -解释JCL是怎样在系统中工作的，总体了解JCL编程技术和一些比较重要的语句和关键词。
- -创建一个简单的作业，并且提交执行。
- -通过SDSF查看作业的输出。

201

6.1 什么是 JCL？

作业控制语言JCL(Job Control Language)用来告知操作系统执行什么程序，并且包含关于程序输入输出的描述。可以提交JCL进行批处理，或者启动一个JCL过程(PROC)，称为启动任务。JCL的细节十分复杂，但是基本的概念却非常简单。而且，实际应用中至少90%的使用集中在JCL的一个小的子集中。本章挑选出一些JCL选项进行讨论。

JCL
告知系统执行什么样的程序并且定义程序的输入和输出

应用程序员需要一些JCL的知识，而产品控制分析师必须要高度精通JCL，要能够创建，监控，修改，重运行公司日常的批处理作业。

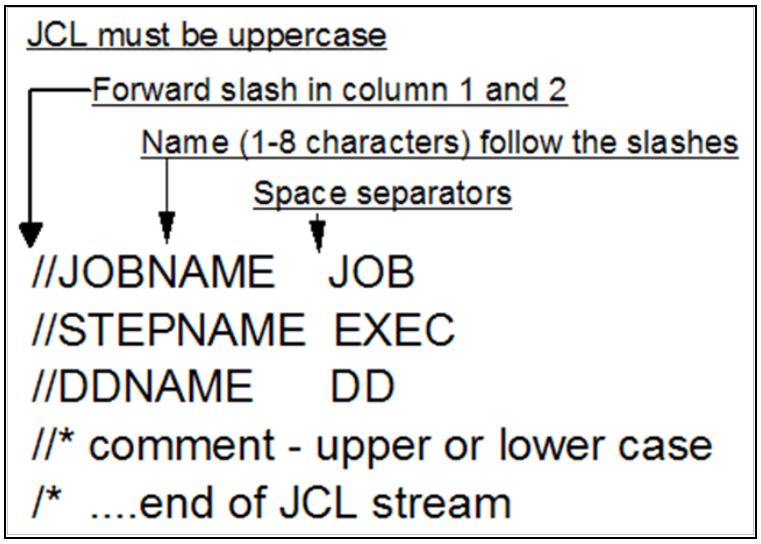
有三种基本JCL语句：

作业语句(**JOB**)：　向系统提供该批处理作业的名字(作业名)，它还可以包含记账信息和一些作业范围内的参数。

执行语句(**EXEC**)：　提供要执行的程序的名字。一个作业中可以包含多个EXEC语句。一个作业中的每个EXEC语句叫做一个作业步。

数据定义语句(**DD**)：　数据定义语句提供执行语句中的程序运行所需要的输入和输出文件。这一语句将程序中的一个ddname链接到一个数据集或者其他的I/O设备或函数上。DD语句与某一个特定的作业步相关联。

图6-1是基本的JCL编程语法



202　*图6-1 基本JCL编程语法*

例6-1中是一些简单的JCL。

例6-1 JCL示例

```
//MYJOB     JOB 1
//MYSORT    EXEC PGM=SORT
//SORTIN    DD DISP=SHR,DSN=ZPROF.AREA.CODES
//SORTOUT   DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
//SYSIN     DD *
  SORT FIELDS=(1,3,CH,A)
/*
```

在第127页，第4章"TSO/E，ISPF和UNIX：z/OS的交互工具"中，在TSO READY提示符下我们运行了相同的程序。每一个JCL DD语句都等价于TSO ALLOCATE命令。它们都用来把z/OS数据集关联到一个被作为程序输入或输出的ddname上。它们在执行方法上的不同是TSO在前台执行排序，而JCL在后台执行排序。

当提交执行时：

MYJOB 是使系统能够与这个作业相关联的作业名。

MYSORT 是指示系统执行SORT程序的作业步名。

SORTIN 在DD语句中，这是ddname。在SORT程序中，这个SORTIN ddname是程序的输入。这个DD语句中的数据集名(DSN)是ZPROF.AREA.CODES。这个数据集是可以被其他系统程序共享的(DISP=SHR)。ZPROF.AREA.CODES的数据内容就是SORT程序的输入。

SORTOUT 这里的ddname是SORT程序的输出。

SYSOUT SYSOUT=*指定了将系统输出信息发送到作业输入子系统 (JES)输出打印区域。还可以将输出发送到数据集中。

SYSIN DD *是另外一种输入语句。它指定了下面的内容是数据或者控制语句。在这个例子中，下面是排序指令，告诉SORT程序需要将SORTIN数据中的哪一区域的数据进行排序。

在文章中我们使用术语"JCL语句"；有一些z/OS用户使用旧的术语"JCL卡片"，即使现在JCL是位于存储器中而非打孔卡上了。

203

6.2 JOB，EXEC 和 DD 参数

JOB，EXEC和DD语句有很多参数，可以允许用户用来指定指令和信息。要想详细地说明这些参数，恐怕要写一整本书了(例如IBM出版物，z/OS JCL Reference)。

这一部分只就一些JOB，EXEC和DD语句中的常用的参数进行简要的说明。

6.2.1 JOB 参数

作业语句//MYJOB JOB 1 定义了作业名是MYJOB。其中1是记账区。它受系统出口程序(system exit)支配，可以用来管理对系统用户的收费。

JOB语句的一般常用参数包括：

JOB 语句
JCL 用来定义作业和提交作业的用户

REGION= 用于请求作业所需分配的特定的内存资源。

NOTIFY= 将作业完成信息发送给指定的用户，例如发送给作业提交者。

USER= 用于指定作业具有该用户ID所拥有的权限。

TYPRUN= 延迟或者暂停搁置运行的作业，稍后再释放。

CLASS= 指定JCL语句在一个特定的输入队列中执行。

MSGCLASS= 指定作业输出到一个特定的输出队列。

MSGLEVEL= 控制可接受的系统信息的数量。

例：

```
//MYJOB JOB 1,NOTIFY=&SYSUID,REGION=6M
```

6.2.2 EXEC 参数

EXEC JCL语句//MYSTEP EXEC定义了作业步名是MYSTEP。EXEC后面的是PGM=(可执行程序名)或者是一个JCL过程名。如果后面是JCL PROC，那么参数就是JCL PROC需要的变量替换。EXEC PGM=语句中的参数一般有：

EXEC 语句
JCL 用来提供要执行的程序名

PARM= 传递到程序中的程序能够识别的参数。

COND= 控制作业中其他作业步执行的布尔值。使用IF，THEN，ELSE这些JCL语句要优于使用COND； 然而，一 些旧的产品环境中的JCL仍然在使用这些语句。

TIME= 规定时间限制。

例：

```
//MYSTEP EXEC PGM=SORT
```

6.2.3 DD 参数

DD JCL语句//MYDATA DD 定义了一个ddname:MYDATA。DD语句的参数比JOB语句和EXEC语句多得多。DD JCL语句可以包含定义和描述程序输入和输出特性的很多方面。一些常见的DD语句参数有：

DD 语句
为 EXEC 语句中的程序指定输入和输出

DSN= 数据集名；可以是临时数据集或新数据集，或是数据集名字的一个向后参考。

DISP= 数据集部署，例如数据集是需要新创建还是已经存在的，以及数据集是否可以在多个作业中共享。 事实上DISP= 非常的重要，我们将在下一个部分：第206页的6.3，"数据集部署，DISP参数"中详细说明。

SPACE= 一个新数据集所需要的磁盘存储空间。

SYSOUT= 定义打印位置(以及输出队列或数据集)。

VOL=SER= 卷名，磁盘名或磁带名。

UNIT= 系统磁盘，磁带，特殊设备类型，或者本地名称。

DEST= 将输出发送到一个远程目的地。

DCB= 数据集控制块，有许多子参数。最常用的子参数有：

- LRECL= 逻辑记录长度。每个记录中包含的字节/字符数。
- RECFM= 记录格式，定长，组块，变长等。
- BLOCKSIZE= 存储记录的记录块的大小，通常是LRECL的倍数。0值是默认使用系统选择的最优值。
- DSORG= 数据集组织结构——顺序数据集，分区数据集等。

LABEL= 预期的磁带标签(数据集位置后是没有标签NL或是标准标签SL)。一个磁带可以存储许多数据集；磁带上的每个数据集都有一个文件位置。磁带上的第一个数据集是文件1。

DUMMY 导致一个空的输入或者舍弃写在这个ddname上的数据。

* 后面跟随输入数据或者控制语句——这是一种将数据从JCL流传递到程序的方法。

*,DLM= 后面跟的一切都是输入数据(甚至//)，一直到指定的两个特殊的数字或字符出现在第一列为止。

6.3 数据集部署，DISP 参数

所有的JCL的参数都很重要，但是对DD语句DISP功能应该是最重要的。作为用途之一，DISP参数向系统建议一个作业所需的数据集的队列，以便防止其他的作业对i这些数据集的冲突性使用。

完整的参数有这些字段：

```
DISP=(status，normal end，abnormal end)
DISP=(status，normal end)
DISP=status
```

其中的status可以是NEW，OLD，SHR或者MOD：

NEW 是指要创建一个新的数据集。作业运行时独占该数据集的访问。这个数据集作为新数据集不能已经存在于同一个卷中，也不能存在于系统或用户目录中。

OLD 是指数据集已经存在，作业运行时独占该数据集的访问。

SHR 是指数据集已经存在，几个并行运行的作业可以共享访问该数据集。这里所有并行的作业都必须指定为SHR。

MOD 是指数据集已经存在，当前运行的作业独占该数据集的访问。如果当前运行的作业打开该数据集用于输出，输出将会追加到数据集当前的尾部。

作业步
请求或者控制一个程序的执行并且指定程序运行所需资源的 JCL 语句

参数normal end指的是当运行的作业正常结束时对数据集(部署)的动作。同样的，参数abnormal end指的是当运行的作业非正常结束时对数据集的动作。

这两个参数的选项都是一样的：

DELETE 作业步结束时删除(并取消编目)数据集。

KEEP 作业步结束时保留(但并不编目)数据集。

CATLG 作业步结束时保留并编目数据集。

UNCATLG 作业步结束时保留数据集，但是将其取消编目。

PASS 允许后面的作业步指定其最终的部署。

206

默认的部署参数(对于正常和非正常结束)是保持数据集在作业步开始前的状态。(目录在第184页5.11.2章节"什么是目录？"中讨论过。)

您也许想知道，如果对一个已经存在数据集指定DISP=NEW会发生什么？事实上，这是几乎不存在的！为了防止因为疏忽而删除或替换了文件，z/OS拒绝对一个已经存在的数据集指定DISP=NEW。您不会创建一个新的数据集而是会得到一个JCL错误。

6.3.1 创建新数据集

如果一个数据集的DISP参数是NEW，您必须提供更多的信息，包括：

- 一个数据集名。
- 数据集的设备类型。
- 如果是磁盘或者标签磁带，需要提供卷标。
- 如果用到一个磁盘，需要指定分配给数据集首次分配量的空间大小。
- 如果创建的是分区数据集，需要指定目录的大小。
- 也可以可选地指定设备控制块参数，或者由将要写这个数据集的程序提供这些参数。

我们已经介绍了DISP和数据集名。其他的参数简单介绍如下：

卷标(**Volser**) 它在DD语句里面的格式是VOL=SER=xxxxxx，这里xxxxxx代表卷标。由于格式的原因，VOL参数也可以

指定其他的细节。

设备类型(**Device type**) 有许多种方法可以定义，但UNIT=xxxx是最常用的。xxxx可以是一种IBM设备类型(例如3390)，或是指定的设备地址(例如300)，或者安装时定义的名称(例如SYSDA)。一般的，使用SYSDA就是告诉系统可以在一系列可用的设备中选择任何可用的盘卷。

成员名(**Member name**) 一个库(或者分区数据集，PDS)成员可以被许多应用程序或实用程序当作一个数据集处理。格式DSNAME=ZPROF.LIB.CNTL(TEST)被用来指定一个特定的成员。如果应用程序或实用程序需要一个顺序数据集，那么必须指定一个顺序数据集或者一个库成员。只有在程序或实用程序需要一个库名时，才会使用整个库名(不包括特定的成员名)。

空间(***Space***)：

在DASD上分配数据集时需要指定SPACE DD参数。它定义了您的数据集所需要的空间。在一个数据集在磁盘上创建之前，系统必须知道这个数据集需要多少空间，以及空间的度量。空间参数有许多不同的格式和变化。常见的例子有：

SPACE=(TRK,10)	10个磁道，没有二次分配区信息。
SPACE=(TRK,(10,5))	初始分配10个磁道，此后每次扩展分配5个磁道。
SPACE=(CYL,5)	用柱面代替磁道。
SPACE=(TRK,(10,5,8))	有8个目录块的分区数据集。
SPACE=(1000,(50000,10000))	初始有50000个记录，每个记录1000字节(bytes)。

在基本的情况下，SPACE有两个参数。它们是度量单位和空间的单位数量。度量单位可以是磁道，柱面或者块的平均大小。[1]

空间的大小一般有三个子参数定义：

- 第一个参数是首次分配量大小，即初始分配的空间大小，采用度量单位表示。对于这些空间系统会尽量分配一个单一的分区(连续的空间)。如果系统不能在作业开始之前，在不超过5个区域分配出这些空间，作业就要宣告失败。
- 如果使用第二个参数则表示每次扩展追加的空间大小。在作业开始之前，系统不用获取这些空间并且不保证这些空间是可用的。在作业运行时，系统会动态的分配扩展空间。在这里展示的基本例子里，扩展空间和初始空间在同一个卷上。
- 如果使用第三个参数，则表示创建一个分区数据集(库)。这是创建一个分区数据集而不是其他类型的数据集的唯一的指示方法。这个数字的值是指分区数据集的目录块(每个255字节)的数量。(如果要创建另外一种数据集PDSE则需要另外一个JCL参数。)

[1]度量单位也可以是 KB 和 MB，但通常并不使用。

如果空间参数包含不只一个子参数，则所有的空间参数要放在一个圆括号里。

6.4 续行和并置

208

在早期的系统里，因为80列的打孔卡片所能容纳的字符数的限制，z/OS引入了续行和并置(concatenation)的概念。因此，为了最小化与早先系统应用程序和操作的冲突，z/OS保持了这些惯例。

JCL续行的语法要求前一行的参数写完整，然后在后面加以逗号。下面一行的JCL的开头是//，后面要有至少一个的空格。JCL参数语法要求在续行里内容必须在第16列或第16列之前开始，并且不能超过第72列。[2]

```
//JOBCARD JOB 1,REGION=8M,NOTIFY=ZPROF
```

上面的JCL语句与下面有续行的JCL语句的效果是相同的：

```
//JOBCARD JOB 1,
//            REGION=8M,
//            NOTIFY=ZPROF
```

并置
一个 ddname 可以对应多个 DD 语句(输入数据集。)

DD语句的一个重要特征就是一个ddname可以有多个DD语句。这就叫做并置。

下面的JCL 就表示数据集是并置的：

```
//DATAIN DD DISP=OLD,DSN=MY.INPUT1
//          DD DISP=OLD,DSN=MY.INPUT2
//          DD DISP=SHR,DSN=YOUR.DATA
```

并置只能应用于输入数据集。这些数据集自动的按顺序被处理。在上面的例子中，当应用程序读到MY.INPUT1的结尾时，系统自动打开MY.INPUT2并开始读取。应用程序并不知道他在读取第二个数据集。这样一直持续到并置中的最后一个数据读取完毕，这时应用程序会接受到一个文件结束的指示。

6.5 z/OS 为何使用符号文件名

通常z/OS使用符号文件名[3]，并且这是这个操作系统的另一个标志性特征。它应用于程序中使用的与数据集有关的名字和执行程序时用到的实际的数据集之间的命名重定向。如第210页的图6-2所示。

209

[2]第 73 列到第 80 列留作卡片序列号专用。
[3]这应用于传统正规处理过程。一些语言，例如 C 定义了绕过这一功能的接口。

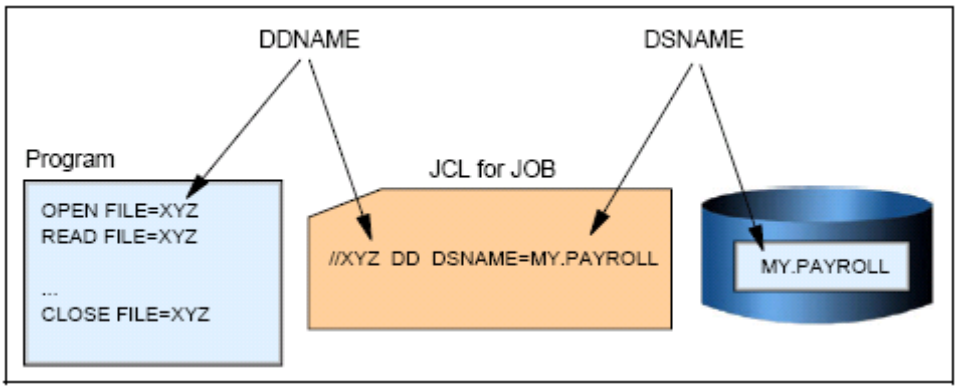


图6-2 DDNAME和DSNAME

在这个图示里有一个以任意语言写的程序需要打开并读取一个数据集[4]。当编写这个程序时任意选择了XYZ这个名字来引用数据集。这个程序可以编译并保存为可执行程序。当有人要运行这个可执行程序时，需要一个JCL语句来把XYZ这个名字关联到一个实际的数据集名。这个JCL语句就是DD语句。程序中使用的符号名是DDNAME，而数据集真实的名字是DSNAME。

符号文件名
程序中与数据集有关的名字和执行程序时用到的实际的数据集之间的命名重定向

通过简单的更改JCL中的DSNAME，这个程序可以用来处理不同的输入数据集。这对一些执行单一程序时需要使用许多数据集的大型商业应用程序来说意义非常重大。一个大公司的薪资程序就是一个很好的例子。这会是一个非常复杂的应用程序，可能要用到成百上千的数据集。同样的程序通过配合不同的JCL运行可以用于同一个公司的不同部门，同样的，还可以通过使用一套不同的JCL使用特殊的测试数据集对程序进行测试。

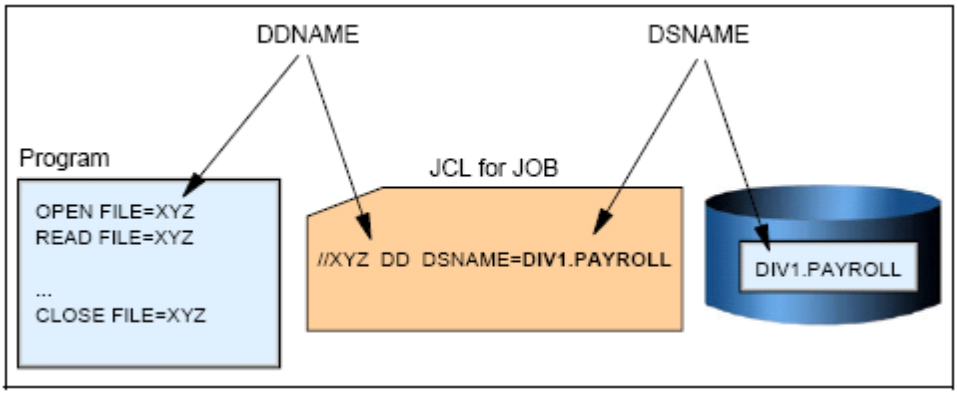


图6-3 符号文件名-同样的程序，不一样的数据集

仅仅通过修改JCL卡片中的一个参数(DD DSN=DIV1.PAYROLL),这个公司就可以在各个部门使用同一个适用全公司范围的薪水册应用程序。参数值DIV1.PAYROLL使得程序访问部门1的数据集。这个例子证明了JCL和符号文件名强大的能力及其提供的灵活性。

这种DDNAME—JCL—DSNAME处理过程应用于所有传统的z/OS工作，虽然有时候并不是很明显。例如，当使用ISPF编辑一个数据集时，ISPF建立DD语句的内

[4]像在大多数计算机语言中一样，伪程序使用文件(file)这个术语。

部等价物，然后打开DD语句请求的数据集。ISPF的用户看不到这一处理—它以对用户透明的方式进行工作。[5]

6.6 保留 DDNAME

程序员几乎可以选择任意的名字来作为DD名，但是，我们推荐使用有意义的名字(最长不超过8个字符)。

有一些保留的DD名是程序员不能使用的(所有这些都是可选择的DD语句)：

```
//JOBLIB DD …
//STEPLIB DD …
//JOBCAT DD…
//STEPCAT DD …
//SYSABEND DD …
//SYSUDUMP DD …
//SYSMDUMP DD…
//CEEDUMP DD …
```


JOBLIB DD语句就放在JOB语句之后，指定该作业查找程序时最先搜索的库。STEPLIB DD语句放在EXEC语句之后，指定EXEC语句执行程序时最先搜索的程序库。如果两个都用到了，STEPLIB覆盖JOBLIB。

JOBCAT和STEPCAT用来指定私有目录，但通常很少用(最新版本的z/OS已经不再支持私有目录)。然而这两个DD名也应该被当作保留名。

SYSABEND, SYSUDUMP, SYSMDUMP, 和 CEEDUMP DD 语句用来指定程序异常中止(或ABEND)时产生的不同类型的信息转储。

6.7 JCL 过程(PROC)

PROC
一个过程库成员，包含给定任务JCL的一部分(通常是不变的部分)

有一些程序和任务需要大量的JCL，用户不能够很容易的输入这些JCL。这些功能的JCL可以保存在一个过程库中。一个过程库成员包含给定任务的部分JCL—通常是JCL中固定的，不会改变的部分。过程的用户为特定的作业提供变化的那部分JCL。换句话说一个JCL过程就像是一个宏。

这样的一个过程有时候被叫做编目过程。一个编目过程与系统目录无关；它的名字是从其他操作系统带过来的。

例6-2展示了一个JCL过程(PROC)的实例。

[5]这里，我们暂时忽略 z/OS 的 z/OS UNIX 接口的一些操作特征；讨论适用于传统 z/OS 的使用。

例6-2 JCL过程实例

```
//MYPROC    PROC
//MYSORT    EXEC PGM=SORT
//SORTIN    DD DISP=SHR,DSN=&SORTDSN
//SORTOUT   DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
//          PEND
```

现在，这里的大部分JCL都应当能看懂了。这里出现的JCL功能包括：

212

- PROC和PEND语句在过程中是唯一的。它们用来标志JCL过程的起始和结束。
- PROC的前面会加上一个标签或名字。在例6-2中定义的名字是MYPROC。
- JCL的变量替换是使用JCL PROC的原因。&SORTDSN是例6-2中唯一的变量。

在例6-3中，我们在作业流中包含了例6-2中的内嵌过程。

例6-3 内嵌过程示例

```
//MYJOB     JOB 1
//*--------------------------------*
//MYPROC    PROC
//MYSORT    EXEC PGM=SORT
//SORTIN    DD DISP=SHR,DSN=&SORTDSN
//SORTOUT   DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
//          PEND
//*--------------------------------*
//STEP1     EXEC MYPROC,SORTDSN=ZPROF.AREA.CODES
//SYSIN     DD *
  SORT FIELDS=(1,3,CH,A)
```

当提交MYJOB时,第212页例6-2中的JCL会有效的替代EXEC MYPROC。&SORTDSN的值必须提供。

SORTDSN和它的值被放置在一个单独的EXEC的扩展行里。注意MYPROC后面的逗号。

//SYSIN DD *后面跟着SORT控制语句，它将会附加到替代JCL的后面。

6.7.1 JCL PROC 语句覆写

当一个完整的JCL PROC 语句需要被代替时，可以使用一个JCL PROC覆写语句。一个覆写语句有以下的形式：

```
//stepname.ddname DD …
```

例6-4显示了MYPROC中的SORTOUT DD语句的覆写。这里SORTOUT指向一个新创建的顺序数据集。

例6-4 有覆写语句的过程示例

```
//MYJOB     JOB 1
//*--------------------------------*
//MYPROC    PROC
//MYSORT    EXEC PGM=SORT
//SORTIN    DD DISP=SHR,DSN=&SORTDSN

213

//SORTOUT   DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
//          PEND
//*--------------------------------*
//STEP1     EXEC MYPROC,SORTDSN=ZPROF.AREA.CODES
//MYSORT.SORTOUT DD DSN=ZPROF.MYSORT.OUTPUT,
//          DISP=(NEW,CATLG),SPACE=(CYL,(1,1)),
//          UNIT=SYSDA,VOL=SER=SHARED,
//          DCB=(LRECL=20,BLKSIZE=0,RECFM=FB,DSORG=PS)
//SYSIN     DD *
  SORT FIELDS=(1,3,CH,A)
```

6.7.2 如何以批处理方式提交作业？

我们用UNIX and AIX®作为类比，对于一个UNIX进程，可以在命令或者脚本后加一个&符号使它在后台处理。按下回车键就可以将作业提交为后台进程。

在z/OS的术语中，工作(作业)以批处理方式提交。批处理大致等同于UNIX的后台处理。作业独立于交互式会话运行。使用'批'这个术语是因为可以有一大批的作业排成队列，在所需资源可用的情况下等待执行。提交作业的命令可以使用下面任何一种形式：

ISPF编辑命令行 SUBmit并按下回车

ISPF命令解析器 SUBmit 'USER.JCL'

这里的数据集是顺序数据集。

ISPF命令行 TSO SUBmit 'USER.JCL'

这里的数据集是顺序数据集。

ISPF命令行 TSO SUBmit 'USER.JCL(MYJOB)'

这里的数据集是一个包含成员MYJOB的库或分区数据集。

214 **TSO命令行** SUBmit 'USER.JCL'

例6-5展示了您可以输入SUBMIT命令的三种情况。

```
 EDIT ---- userid.SORT.JCL ------------------------ LINE 00000000 COL 00
 COMMAND ===> SUBMIT                                          SCROLL ===>
****************************** TOP OF DATA  ***************************
//userid   JOB 'accounting data',
                      .
                      .
                      .
```

TSO/E command line:

```
------------------------  TSO COMMAND PROCESSOR  ------------------------
ENTER TSO COMMAND OR CLIST BELOW:


===> SUBMIT 'userid.SORT.JCL'



ENTER SESSION MANAGER MODE ===> NO     (YES or NO)
```

After READY mode message:

```
    .
    .
    .
  READY
 SUBMIT 'userid.SORT.JCL'
```

例6-5 提交JCL流进行处理的几种方法

6.8 理解 SDSF

提交完一个作业后，通常可以使用系统显示和查找工具(SDSF)来查看成功执行的输出或者查看并更正JCL错误。SDSF允许您显示在JES spool区中打印的输出。许多由批处理作业(和其他作业)发送到JES中的打印输出实际上并没有被真正的打印。而是在通过SDSF查看后被删除了，或者根据需要使用。

SDSF提供了许多附加的功能，包括：

SDSF
显示 JES spool 区域中供查看的打印输出

- 查看系统日志，查找任意字符串
- 输入系统命令(在早期的操作系统版本中，只有操作员可以输入命令)
- 控制作业处理(保持，释放，取消，以及清除作业)
- 在作业处理的过程中监控作业
- 在决定打印之前显示作业输出
- 控制作业处理的顺序
- 控制输出打印的顺序

► 控制打印机和启动程序

图6-4展示了SDSF的主选项菜单

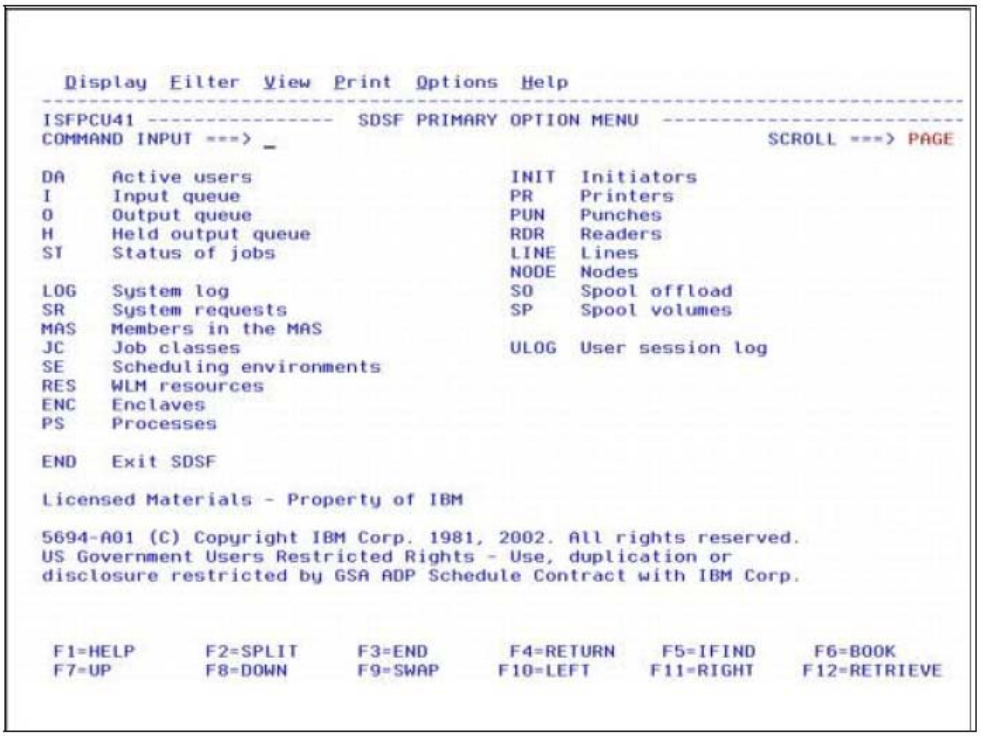


图6-4 SDSF主选项菜单

如图6-5所示，SDSF使用一个层级在线面板来指导用户了解它的功能。

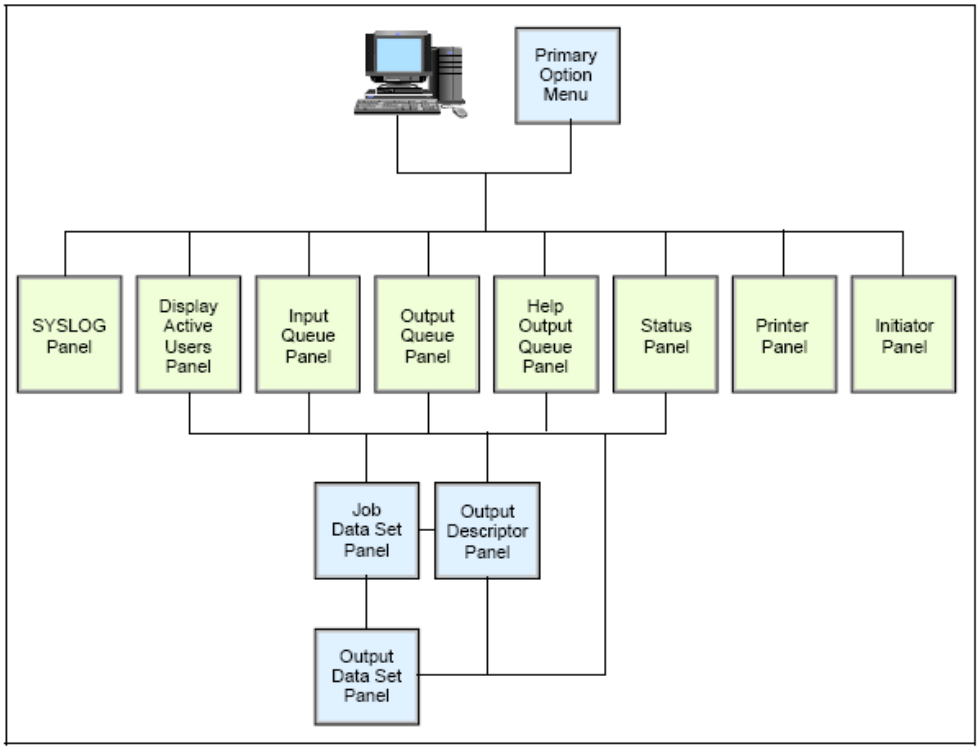


图6-5 SDSF面板层次

您可以看到JES输出数据集在批处理作业执行的过程中创建。它们保存在JES spool数据集中。

您可以看到在以下任意一种队列中的JES数据集：

I 输入

DA 执行队列

O 输出队列

H 保持队列

ST 状态队列

对于输出和保持队列，您可以将MSGCLASS设置成定义为不保存输出的sysout类，这样就可以让输出自动被清理掉，您就看不到那些JES数据集了。同样的，根据您在JOB卡片上选择的MSGCLASS，sysout可以在输出队列中，也可以在保持队列中。

```
Screen 1
  Display  Filter  View  Print  Options  Help
--------------------------------------------------------------------------------
SDSF HELD OUTPUT DISPLAY ALL CLASSES  LINES 44          LINE 1-1 (1)
COMMAND INPUT ===>                                       SCROLL ===> PAGE
PREFIX=*  DEST=(ALL)  OWNER=*  SYSNAME=
NP    JOBNAME  JobID    Owner    Prty C ODisp Dest              Tot-Rec  Tot-
?_    MIRIAM2  JOB26044 MIRIAM    144 T HOLD  LOCAL                   44


Screen 2
  Display  Filter  View  Print  Options  Help
--------------------------------------------------------------------------------
SDSF JOB DATA SET DISPLAY - JOB MIRIAM2  (JOB26044)    LINE 1-3 (3)
COMMAND INPUT ===>                                       SCROLL ===> PAGE
PREFIX=*  DEST=(ALL)  OWNER=*  SYSNAME=
NP    DDNAME   StepName ProcStep DSID Owner    C Dest              Rec-Cnt Page
      JESMSGLG JES2                 2 MIRIAM   T LOCAL                  20
      JESJCL   JES2                 3 MIRIAM   T LOCAL                  12
      JESYSMSG JES2                 4 MIRIAM   T LOCAL                  12
```

图6-6 通过SDSF查看JES2输出文件

在图6-6中的Screen 1中显示了我们提交的作业列表，对于这些作业的输出，我们通过在作业卡片上定义参数MSGCLASS=T将其指向到保持队列(Class T)。在我们的例子中，只有一个作业提交执行。因此在保持队列中我们只有一个作业。在NP这一列输入一个?命令显示job 7395生成的输出文件。

作业名
系统(JCL语句)可以识别的作业的名字

图6-6中的Screen 2中显示了三个ddname:JES2消息日志文件，JES2 JCL文件，以及JES2系统消息文件。当您查看的作业有很多文件定向到SYSOUT，而您只想显示与一个特定的作业步相关的文件时，这一选项就非常有用。您可以在NP这一栏输入一个S来选择您想要看的文件。

要想看到所有的文件，NP这一栏输入一个S而不再是?；JES2作业日志与例6-6中显示的相似。

```
J E S 2  J O B  L O G  --  S Y S T E M  S C 6 4  --  N O D E

13.19.24 JOB26044 ---- WEDNESDAY, 27 AUG 2003 ----
13.19.24 JOB26044  IRR010I  USERID MIRIAM   IS ASSIGNED TO THIS JOB.
13.19.24 JOB26044  ICH70001I MIRIAM   LAST ACCESS AT 13:18:53 ON WEDNESDAY,
AUGU
13.19.24 JOB26044  $HASP373 MIRIAM2  STARTED - INIT 1    - CLASS A - SYS SC64
13.19.24 JOB26044  IEF403I MIRIAM2 - STARTED - ASID=0027 - SC64
```

```
13.19.24 JOB26044  -                                          --TIMINGS
(MINS.)--
13.19.24 JOB26044  -JOBNAME  STEPNAME PROCSTEP    RC   EXCP    CPU    SRB
CLOCK
13.19.24 JOB26044  -MIRIAM2           STEP1       00      9    .00    .00
.00
13.19.24 JOB26044  IEF404I MIRIAM2 - ENDED - ASID=0027 - SC64
13.19.24 JOB26044  -MIRIAM2  ENDED.  NAME-MIRIAM              TOTAL CPU TIME=
13.19.24 JOB26044  $HASP395 MIRIAM2  ENDED
------ JES2 JOB STATISTICS ------
  27 AUG 2003 JOB EXECUTION DATE
           11 CARDS READ
           44 SYSOUT PRINT RECORDS
            0 SYSOUT PUNCH RECORDS
            3 SYSOUT SPOOL KBYTES
         0.00 MINUTES EXECUTION TIME
        1 //MIRIAM2 JOB 19,MIRIAM,NOTIFY=&SYSUID,MSGCLASS=T,
          // MSGLEVEL=(1,1),CLASS=A
          IEFC653I SUBSTITUTION JCL -
19,MIRIAM,NOTIFY=MIRIAM,MSGCLASS=T,MSGLEVE
        2 //STEP1 EXEC PGM=IEFBR14
          //*-------------------------------------------------*
//* THIS IS AN EXAMPLE OF A NEW DATA SET ALLOCATION
          //*-------------------------------------------------*
        3 //NEWDD DD   DSN=MIRIAM.IEFBR14.TEST1.NEWDD,
          //           DISP=(NEW,CATLG,DELETE),UNIT=SYSDA,
          //           SPACE=(CYL,(10,10,45)),LRECL=80,BLKSIZE=3120
        4 //SYSPRINT  DD  SYSOUT=T
          /*
ICH70001I MIRIAM   LAST ACCESS AT 13:18:53 ON WEDNESDAY, AUGUST 27, 2003
IEF236I ALLOC. FOR MIRIAM2 STEP1
IGD100I 390D ALLOCATED TO DDNAME NEWDD    DATACLAS (        )
IEF237I JES2 ALLOCATED TO SYSPRINT
IEF142I MIRIAM2 STEP1 - STEP WAS EXECUTED - COND CODE 0000
IEF285I   MIRIAM.IEFBR14.TEST1.NEWDD                  CATALOGED
IEF285I   VOL SER NOS= SBOX38.
IEF285I   MIRIAM.MIRIAM2.JOB26044.D0000101.?           SYSOUT
IEF373I STEP/STEP1   /START 2003239.1319
IEF374I STEP/STEP1   /STOP  2003239.1319 CPU    0MIN 00.00SEC SRB    0MIN
00.00S
IEF375I  JOB/MIRIAM2 /START 2003239.1319
IEF376I  JOB/MIRIAM2 /STOP  2003239.1319 CPU    0MIN 00.00SEC SRB    0MIN
00.00S
```

例6-6 JES2 作业日志

6.9 实用程序

z/OS中有一些在批处理中有用的程序，叫做实用程序。这些程序提供很多较小的，

实用程序
提供许多有用批处理功能的程序。

容易理解的并且有用的功能。在第581页，附件C"实用程序"中描述了系统提供的实用程序的基本集合。

在客户终端通常还会加入他们客户自己编写的实用程序(尽管大部分用户不把它们称作实用程序)，并且很多这些实用程序在用户社区广泛共享。独立的软件厂商也会提供很多类似的产品(收费)。

6.10 系统库

系统库
系统盘卷上的 PDS 数据集，包含 z/OS 的控制参数，JCL 过程，基本执行模块等等。

z/OS有很多标准系统库。我们在这里适当简明介绍一些库。传统的库包括：

- SYS1.PROCLIB 这个库包括了z/OS发布的JCL过程。实际上，它并置着许多其他的JCL过程库(由不同的程序产品提供)。
- SYS1.PARMLIB 这个库包括了z/OS和一些程序产品的控制参数。实际上也许有其他的库与它相并置。
- SYS1.LINKLIB 这个库包括了许多系统的基本执行模块。实际上，它是大量并置在一起的执行库中的一个。
- SYS1.LPALIB 这个库包括了在系统初始化时加载到连接装配区的系统执行模块。也许有其他的一些库与之并置。存储在这里的程序对其他的地址空间也是可用的。
- SYS1.NUCLEUS 这个库包含了z/OS的基本管理模块(内核)。
- SYS1.SVCLIB 这个库包含了操作系统例程，也叫做访管程序(SVCs)。

220

这些库都是建立在系统盘卷上的标准格式的分区数据集(PDS)。在第466页16.3.1章节"z/OS系统库"这一节将有更为详细的介绍。

6.11 总结

基本JCL包括三种类型的语句：JOB，EXEC和DD。一个作业可以包含多个EXEC语句(作业步)，每一个作业步可以有多个DD语句。JCL提供了形形色色的参数和控制；这里只介绍了一小部分基本的。

一个批处理作业通过ddname来访问数据集。在程序的一次执行中，JCL DD语句把ddname和指定的数据集(数据集名)连接起来。通过修改每个作业的JCL，一个程序可以访问不同群组的数据集(在不同的作业中)。

DD语句中的DISP参数可以帮助阻止不必要的数据集同时访问。这在一般的系统操作中非常重要。DISP参数并不是安全控制，但它可以帮助管理数据集的完整性。在JCL中使用参数DISP=NEW，并且指定想要的空间大小和想要分配的卷，就可以创建一个新的数据集。

系统用户希望只写简单的JCL，但是通常可以使用JCL过程来完成复杂的作业。编目过程被一次写成后就可以被许多用户使用。z/OS提供了许多JCL过程，添加本地写的过程也很简单。为了给某个特定的程序提供所需的参数(通常是DD语句)，用户必须懂得如何覆写或者扩展JCL过程中的语句。

本章关键术语		
并置(concatenation)	数据定义语句(DD statement)	执行语句(EXEC statement)
作业控制语句(job control language，JCL)	作业语句(JOB statement)	作业步(job step)
作业名(jobname)	过程(PROC)	SDSF
符号文件名(symbolic file name)	系统库(system library)	实用程序(utility)

6.12 复习题

为了帮助您理解本章的内容，完成下列回顾问题：

1.在第212页，6.7章节”JCL过程(PROC)”中的过程片断和作业中，哪里是COBOL源代码？应用程序的输出数据集可能是什么？输入数据集可能是什么？我们怎样为应用程序的SYSOUT数据集进行JCL 编码？

2.我们有三个DD语句：

```
//DD1   DD   UNIT=3480,…
//DD2   DD   UNIT=0560,…
//DD3   DD   UNIT=560,…
```

这些数字的含义是什么？我们是怎么知道的？

3.JCL可以被提交和启动，它们的不同是什么？

4.解释数据集名，DD名，和程序中的文件名之间的关系。

5.哪一种JCL语句(JOB，EXEC或者DD)有最多的参数？为什么？

6.JCL和JCL PROC之间的区别是什么？使用JCL PROC的好处是什么？

7.在执行PROC的JCL流中覆写JCL PROC语句时，必须要知道什么PROC名？她们在JCL覆写语句中的名字顺序是什么样的？

8.当一个JCL作业有几个EXEC语句时，与每个EXEC语句相关的名字的类型是什么？

6.13 思考题

这些内容最好在课上讨论，这些讨论应当被当作是基础课程的一部分。

1. 为什么数据库系统的出现潜在的改变了大量DD语句的需要？
2. JOB语句中的第一个位置参数是记账区。主机使用中的记账有多重要？为什么？

6.14 练习

这一章的实验练习将会帮助您提高在z/OS中创建批处理作业并提交执行作业的技术。这些技术在本书以后章节的实验练习中也是必须的。

要完成这些实验练习，您或者您的小组需要一个TSO用户ID和密码(可以向老师请求帮助)。

练习内容如下：

- 第223页 “创建一个简单的作业”
- 第225页 “在分屏模式下使用ISPF”
- 第226页 “在ISPF下对文本进行操作”
- 第226页 “提交一个作业并且查看结果”
- 第227页 “创建一个PDS成员”
- 第228页 “复制一个PDS成员”

6.14.1 创建一个简单的作业

1. 在ISPF中，跳至‘Data Set List Utility’面板，在Dsname Level处(前面的练习中有过介绍)输入yourid.JCL。
2. 在yourid.JCL左边(在命令栏)输入e(编辑)。在成员JCLTEST左边输入s(选择)。在编辑器命令行输入RESet。
3. 注意在数据集中只有一行JCL，EXEC PGM=IEFBR14。这是一个不需要任何输入和输出的系统实用程序，设计成运行结束时返回成功码(0)。在命令行输入SUBMIT或者SUB然后按下回车。
4. 输入1回应消息：
   IKJ56700A ENTER JOBNAME CHARACTER(S)-
   结果会显示消息：
   IKJ56250I JOB yourid1(JOB00037) SUBMITTED

注意：当您看到有三个星号(***)出现时，就表示还有更多的信息察看。按回车键继续。

   当作业执行完毕时，您将看到信息：
   $HASP165 yourid1 ENDED AT SYS1 MAXCC=0 CN(INTERNAL)
5. 在您的文件中添加(插入)新的第一行作为JOB语句。JOB语句必须在EXEC语句之前。(提示： 先复制(r)那一行EXEC语句，然后用您的JOB语句覆写EXEC语句。)作业语句如下：

   //youridA JOB 1

将”yourid”替换为您的小组用户ID，保留字母”A”，然后提交这个JCL，然后按下PF3保存文件退出编辑器。

6. 在ISPF主选项菜单中，找到SDSF(第250页，7.9.5章节"使用SDSF"中有介绍)。您可以使用分屏功能打开一个新的会话屏幕，一个用来查看DSLIST，另外一个用来查看SDSF。

7. 在SDSF菜单中，输入PREFIX yourid*，然后输入ST(状态面板)。您提交的两个作业都会在列表中。在任意一个作业的左边输入S(选择)，然后上下翻页查看执行产生的信息。按PF3键退出。

223

8. 再次编辑JCLTEST，在底部插入下面的行：
   ```
   //CREATE DD DSN=yourid.MYTEST,DISP=(NEW,CATLG),
   // UNIT=SYSDA,SPACE=(TRK,1)
   ```

9. 提交上面创建的JCLTEST的内容，按下PF3键(保存并退出编辑)，然后用SDSF查看这个作业的输出。注意您的两个作业有同样的作业名。JOBID数值最大的那个作业是最后执行的作业。
   a. 状态码(condition code)是什么？如果状态码大于0，向下翻页到输出列表的底部找到JCL错误信息。改正JCLTEST然后重新提交。一直重复直到收到cond code=0000。
   b. 跳至'Data Set List Utililty'面板(=3.4)，在DSNAME level处输入yourid.MYTEST。这个数据集存储在哪一个卷上？
   c. 在数据集左边(命令)栏中输入DEL /删除这个数据集。可能会出现一个确认消息来确认您是否真的要删除这个数据集。
   d. 我们刚刚学习批量执行不需要任何输入和输出的程序IEFBR14，如果没有JCL错误，将会返回条件码0(成功)。尽管IEFBR14没有做任何的I/O，系统依旧读取并执行了JCL指令。这个程序对在DD语句中创建(DISP=NEW)和删除(DISP=(OLD,DELETE))数据集非常有用。

10. 在任何ISPF面板中，在命令区输入：TSO SUBMIT JCL(JCLERROR)，您的用户ID是JCL这个数据集的前缀(高级限定符)，这个数据集包含一个成员JCLERROR。
    a. 系统会提示使输入一个后缀字符来生成一个作业卡片。从提交信息中记下作业名和作业号。
    b. 使用SDSF选择这个作业的输出。向下翻页到底部。您看到JCL错误了吗？不正确的和正确的JCL DD操作数分别是什么？在yourid.JCL(JCLERROR)中找到并改正错误。重新提交JCLERROR来验证您的修改。

11. 在任何ISPF面板中，输入TSO SUBMIT JCL(SORT)。您的用户ID是数据集JCL的前缀，这个数据集包含成员SORT。
    a. 系统会提示使输入一个后缀字符来生成一个作业卡片。从提交信息中记下作业名和作业号。
    b. 使用SDSF，在作业名左边输入?。将会显示这个作业产生的一个列表。在SORTOUT左边输入S(选择)，来查看排序输出，然后按PF3退出。选择JESJCL。注意"job statement generated message"和"substitution JCL"消息。

224

12. 我们来清除一些(或者所有)不需要的作业输出。在SDSF中，在任意您想从JES输出队列中清 除的作业键入p(purge)。

13. 在ISPF面板中，输入TSO SUBMIT JCL(SORT)然后查看输出。

14. 在ISPF面板中，输入TSO SUBMIT JCL(SORTPROC) 然后查看输出。也许

您在SDSF ST面板中看不到输出。这是因为作业名不是以yourid开头的。要想看到所有的输出，可以先输入PRE *，再输入OWNER yourid来只查看所有属于您的作业。

15. SORT和SORTPROC之间存在的JCL不同是什么？在两个JCL流中，SYSIN DD语句都引用了排序控制语句。排序控制语句的位置在哪里呢？

    提示：在提交的作业中所有&SYSID的JCL引用都被用户ID替代了

16. 编辑包含排序控制语句的分区数据集成员。把FIELD=(1,3,CH,A)改成FIELD=(6,20,CH,A)。按 PF3键，再后在ISPF面板中输入TSO SUBMIT JCL(SORT)。用SDSF查看作业输出。这是按编码(code)排序的还是按区域(area)排序的？

17. 在ISPF面板中，输入TSO LISTC ALL。默认地，这将列出所有以yourid开头的数据集的目录条目。系统目录将返回数据集名，保存着这些详细信息的目录名，卷位置，以及与JCL UNIT=操作数相同的设备类型号。LISTC是LISTCAT的缩写。

6.14.2 在分屏模式下使用 ISPF

就像我们前面已经讨论过的，大部分ISPF用户喜欢使用分屏。这很容易实现：

1. 将光标移到最底行(或者最顶行)。

2. 按PF2键分屏。

3. 按PF9键在两个屏幕间切换。

4. 按PF3键(或许需要按几次)来退出一个分屏。不一定非要从最底行或者最顶行来分屏。可以把任意一行作为分界行使用PF2键来分屏。可以分出超过两个的屏幕。试着使用这些ISPF命令：

   ```
   START
   SWAP LIST
   SWAP <屏幕号>
   ```


6.14.3 在 ISPF 下对文本进行操作

登录到TSO/E并且进入ISPF以后，看主选项菜单。

1. 输入每个选项并记下它的目的和功能。每个小组都要准备针对ISPF面板的12个功能(选项0-11)中的任意一个功能做一个简明的总结。注意，z/OS装置经常会为了适应需要而把ISPF面板彻底地客户化了。

2. 在分区数据集中创建一个测试成员。输入几行信息，然后试验下面的命令。如需帮助请按PF1。

i	插入一行。
回车键	不输入任何内容就按下回车键可以退出插入模式。
i5	得到5个输入行。
d	删除一行。
d5	删除五行。
dd/dd	删除一块连续的行(在这个块的第一行输入DD，在最后一行再输入一个DD)。
r	重复(或复制)一行。
rr/rr	重复(或复制)一块连续的行(一个RR标记这个块的第一行，另一个RR标记出最后一行)
c连同a或b	将一行复制到另一行前面或者后面。
c5连同a或b	将五行复制到另一行前面或者后面。
cc/cc连同a或b	将一块连续的行复制到另一行前面或者后面。
m，m5，mm/mm	移动行。
x，x5，xx/xx	隐藏行
s	重新显示您隐藏掉的行。
(	左移一列。
)	右移一列。
<	左平移数据。
>	右平移数据。

6.14.4 提交一个作业并且查看结果

编辑yourid.LIB.SOURCE库中的成员COBOL1并且查看COBOL程序。它里面没有JCL。现在编辑yourid.JCL中的成员COBOL1[6]，并且仔细观察JCL。它用了一个JCL过程来编译和运行COBOL程序[7]。按下面的步骤操作：

226

1. 把作业名改成yourid加上附加字符。
2. 把NOTIFY参数改为您的用户ID。
3. 在您的作业卡中添加TYPRUN=SCAN。
4. 在ISPF命令行输入SUB提交作业。
5. 将ISPF分屏，在新屏幕中进入SDSF(在前面的练习中也许您已经做过了)。
6. 在SDSF中用ST(状态)显示并寻找您的作业名。

   可能您需要在SDSF命令行输入PRE或者OWNER命令来查看所有的作业名。(前面的用户可能使用过前缀命令(PRE)来查看某一些作业名)

7. 在您的作业名旁边输入S来查看所有的打印输出：
   - 来自JES2的消息
   - 来自启动程序的消息

[6]匹配成员名(COBOL1)不是必需的；不过它们非常方便。
[7]这并不完全是我们前面讨论过的 COBOL 过程。这些过程的细节有时会由于不同的操作系统版本而有一些改变。

- 来自COBOL编译器的消息
- 来自绑定器的消息
- COBOL程序的输出

8. 当您准备好运行作业时将TYPRUN=SCAN去掉。

9. 使用PF3回到上一级，在您的作业名旁边输入?显示另一种输出格式。教师可以告诉您不同的JES2和启动程序消息代表的不同含义。
   - 在JOB语句中的使用MSGLEVEL=(1,1)重新提交作业。
   - 在JOB语句中的使用MSGLEVEL=(0,0)重新提交作业。

   MSGLEVEL这个参数控制着产生的启动程序消息的数量。

6.14.5 创建一个 PDS 成员

有几种方法可以创建一个新的PDS成员。使用您的用户ID尝试下面的每一种方法。在下面的步骤中TEST3，TEST4，TEST5和TEST6代表新的成员名。在每一个实验中输入几行文本。使用ISPF编辑面板：

- 到ISPF主菜单。
- 进入选项2(编辑)。
- 在'Data Set Name'这一行，输入JCL(TEST3)(没有引号！)
- 输入几行文本然后按下PF3键保存新成员。

当在编辑状态查看成员列表时也可以创建一个新的成员：

- 用选项3.4或者选项2来编辑yourid.JCL。
- 在查看成员列表状态下，在命令行输入S TEST4。
- 输入几行文本，然后按PF3键保存新成员。

当编辑一个已经存在的成员时也可以创建一个新的成员：

- 编辑yourid.JCL(TEST1)或其他任何已经存在的成员。
- 在一部分连续的行的第一行和最后一行输入cc(在行命令区)选择出一块连续的行。
- 在命令行输入CREATE TEST5。这样就会在目前的库中创建一个成员TEST5。

还可以使用JCL来创建一个新成员。在yourid.JCL(TEST5)或者任何方便的地方输入下列JCL:

```
//yourid1 JOB 1,JOE,MSGCLASS=X
//STEP1 EXEC PGM=IEBGENER
//SYSIN DD DUMMY
//SYSPRINT DD SYSOUT=*
//SYSUT2 DD DISP=OLD,DSN=yourid.JCL(TEST6)
//SYSUT1 DD *
This is some text to put in the member
More text
/*
```

将这些JCL保存在这个成员中。后面将会使用到。

6.14.6 复制一个 PDS 成员

复制一个库成员有很多种方法。之前的练习使用ISPF的3.3面板中的功能来复制一个库中的所有成员。同样的功能可以用来复制一个或者多个成员。

当编辑一个库成员时，我们可以将这个库中的另外一个成员复制进来。

- 编辑一个库成员。
- 在这个成员中用a(之后：after)或者b(之前：before)来标记出一行来指明将另一个成员复制到哪里。
- 在命令行输入COPY xxx，这里的xxx是当前数据集中的另一个成员的名字。

228

我们还可以按照如下步骤复制另一个数据集中的成员(或者一个顺序数据集)：

- 编辑一个成员或者一个顺序数据集
- 用A(之后：after)或者B(之前：before)标记处一行，指明新的内容添加的位置。
- 在命令行输入COPY显示Edit/View-Copy面板。
- 在‘Data Set Name’区域输入完整的顺序数据集名(如果需要，使用单引号)或者完整的库名(包括成员名)。

229

230

7

第 7 章 批处理和 JES

目标：作为大型机的职业操作人员，您应当了解系统处理您所在公司的核心程序(比如工资单)的方式。像这样涉及到顺次执行一个或者多个批量任务的工作通常通过批处理程序来完成。

而且，您需要理解作业输入子系统 (JES)是如何使得批处理程序工作的。JES帮助z/OS系统接受作业，调度运行作业，并且指定作业输出的处理方式。

完成本章节后，您应当能够：

- 对系统中批处理过程和作业的启动和管理有总体的概述了解。
- 解释作业输入子系统 是如何通过z/OS系统来管理工作流程的。

231

7.1 什么是批处理？

批处理作业这个术语来源于运行一个或者多个程序时使用穿孔卡片来引导计算机的时代。代表着多个作业的多张卡片通常会叠加集中在读卡机的入口处，然后以批量的形式来执行。

穿孔卡片是一个历史性的词条了，它是赫尔曼·霍尔瑞斯(Herman Hollerith)(1860-1929)在美国人口统计局做统计学家的过程中于1890年发明的。为了帮助为1890年美国人口普查结果制表，霍尔瑞斯设计了一种有80列12行的纸片卡；他设计的纸片卡大小和当时的一美元钞票的大小一样。他在卡片上合适的行/列交汇处打孔来代表一系列的数值。同时，霍尔瑞斯设计了一种电动机械装置来"读"卡片上的洞，由计算设备将这些结果电子信号分类整理并且来制成表。(霍尔瑞斯先生后来组建了计算制表记录公司，而它正是IBM的前身)

批处理作业
那些可以用最少的人机交互执行的程序，通常是在预定的时间执行。

如今，那些可以无需终端用户交互而运行，或者可以在资源允许的情况下按预定执行的作业，都被称为批处理作业。例如，一个读入一个大文件并且生成报告的程序就被认为是一个批处理作业。

在PC或者Unix系统中，并没有z/OS系统中批处理的配对物。批处理是那些需要频繁执行并且需要最少的人机交互的程序。它们通常是按照预定的时间或者基于需要执行的。可能最相近的类比就是UNIX中由AT或者CRON指令执行的过程了，当然它们之间的不同是显而易见的。也许您会认为批处理在某种程度上类似于通常在基于英特尔内核的操作系统上管理的打印队列。用户提交打印请求，打印作业在一个被称为打印池的工作队列中等待被执行，直到按照优先级被选中执行为止。

为了能够处理一个批处理作业，z/OS专业操作人员使用作业控制语言(JCL)来告诉z/OS系统哪些程序需要被执行，执行这些程序需要哪些文件。如我们在第201页第6章学到的"使用JCL和SDSF"，JCL允许用户向z/OS系统描述一个批处理作业的某些属性，比如：

- 您是谁(批处理作业的提交者)
- 哪个程序被执行
- 输入输出的定位
- 作业何时运行

232 用户向系统提交作业后，正常情况下不会再有人机交互，直到作业完成。

7.2 什么是 JES

JES
一个在z/OS系统上处理批量工作负载的程序集合。

z/OS系统使用一个作业输入子系统 (Job Entry Subsystem，JES)接收作业，调度它们在z/OS系统中的处理过程，并且控制它们的输出处理。JES是操作系统的一个组件，它提供了增补性的作业管理，数据管理，和任务管理功能，例如调度、控制作业流程、在辅助存储设备读写输入输出流，它与作业执行并发进行(一个叫

做假脱机操作的进程)。

z/OS把工作分成任务和子任务来管理。交易和批处理作业都与一个基于优先级管理的内部任务队列相关联。JES是z/OS的一个组件，它在程序执行前工作，以准备要执行的工作。在工作执行后，JES也会活跃起来，以帮助工作执行后的清理工作。这些包括了管理由运行的程序生成的输出的打印工作。

更为明确的是，JES管理着输入、输出作业队列和数据。

例如， JES为z/OS管理批处理的以下几个方面：

- 为操作系统接收作业
- 调度作业交付z/OS处理
- 控制它们的输出处理过程

z/OS有两个作业入口系统版本：JES2和JES3. 这之中，JES2是目前最普遍使用的，并且在本文中用作范例。JES2和JES3由很多功能和特性，但是它们最基本的功能如下：

- 接收以各种方式提交的作业
  - 以SUBMIT命令通过ISPF提交
  - 通过网络提交
  - 通过正在运行的程序提交，该程序可以通过JES内部阅读器提交其他作业
  - 通过读卡器提交(非常少见！)
- 将等待执行的作业放入队列中。可以为不同的目的而定义多个队列
- 将作业排队，等待启动程序处理，启动程序是一个系统程序，它请求合适队列中的下一个作业
- 接收运行的作业的输出打印，并且将输出放入队列中。
- 可选的，把输出结果发送到打印机，或者把它保存到spool以便PSF，InfoPrint或者另外一个输出管理器检索

Spooling
在辅助存储设备上的读写(通过JES)输入输出流，与作业执行并发工作。

233

JES为假脱机操作(spooling)使用一个或者多个磁盘数据集，假脱机操作是与作业执行并发的，它是在辅助存储设备上读写输入输出流的过程，采用的格式方便后续处理或输出操作。SPOOL是同时外围联机操作(simultaneous peripheral operations online)的缩略语。

启动程序 Initiator
操作系统的一部分，从系统的输入设备读取并且处理操作控制语言指令。

JES把多个SPOOL数据集(如果存在)合并到一个单一的概念性的数据集中去。内部格式不是标准的访问方法格式，并且不会被应用程序直接读写。输入的作业和来自许多作业的打印输出被存储在单一的(概念性的)SPOOL数据集中。在一个小的z/OS系统中SPOOL数据集可能占据几百个柱面的磁盘空间；在一个大的系统中，它们就可能是磁盘空间中许多完整的卷。

批处理基本元素如图7-1所示：

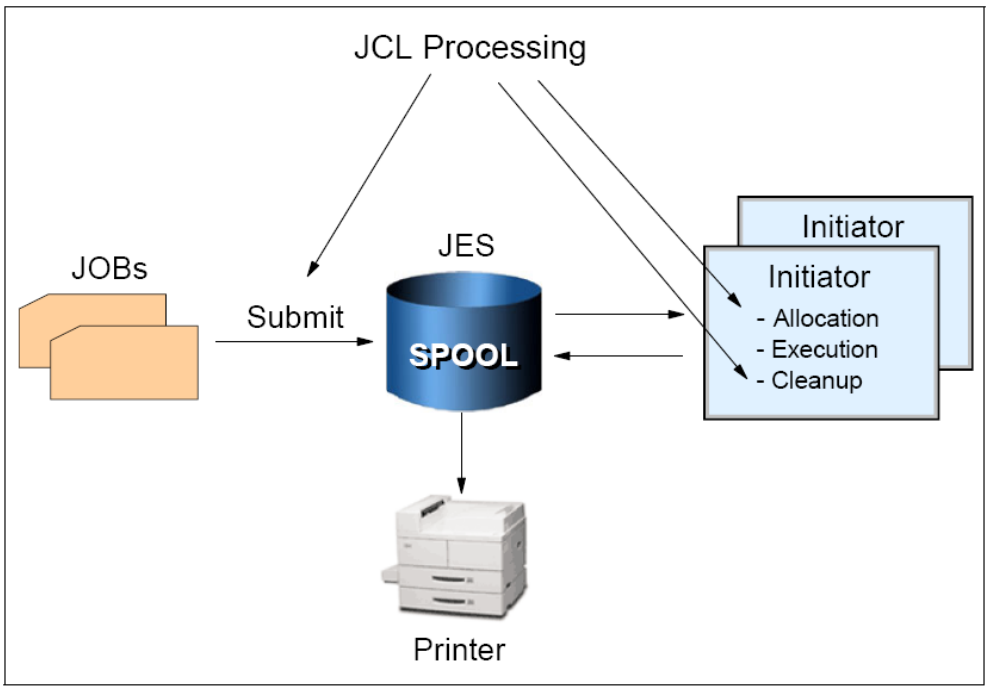


图7-1基本批处理流程

启动程序(initiator)是z/OS中一个完整的部分，它可以读取，解析并且执行JCL语言。通常情况下，它运行在几个地址空间(作为多个启动程序)。一个启动程序管理着批处理作业的运行，在同一地址空间一次只能运行一个作业。如果有10个启动程序都在活动(在10个地址空间)，那么10个批处理作业就可以同时运行。JES也做一些JCL处理，但是启动程序做关键的JCL工作。

图7-1的作业代表了JCL和可能与JCL一起提交的数据。提交给编译器的源代码输入就是一个可能混合着JCL的数据(源语句)的例子。另一个例子是一个为某公司的不同部门准备每周工资表的核算工作(也许对所有部门，工资表应用程序是相同的，但是输入和主总结文件可能不同)。

图中用穿孔卡片代表作业(使用代表穿孔卡片的惯例符号)，尽管现在真实的穿孔卡片输入已经很少见了。通常，一个作业由存储在某分区数据集的某成员中的卡片信息(80个字节固定长度的记录)组成。

7.3 启动程序做什么？

为了异步运行多个作业，系统必须能够执行以下功能：

- 从输入队列中选择作业(JES完成这项工作)
- 确保多个作业不会在数据集使用上冲突(包括TSO用户和其他交互的应用程序)
- 确保单用户设备比如磁带设备，被正确分配
- 为作业找到其所请求的可执行程序
- 在作业运行完毕后完成清理工作并请求下一个作业

上述大部分工作是基于每个作业的JCL信息，由启动程序(initiator)来完成。最复杂的功能是保证没有因为数据集使用而引起的冲突。例如，如果两个作业想同时写入同一个数据集(或者当一个写入的同时另一个读取)，这就是一个冲突[1]。这个事件通常会导致坏数据的产生。JCL的主要目的是告诉启动程序作业需要什么。

数据集冲突的防止对z/OS系统非常关键，并且也是操作系统的标志性的特征。当JCL构造合理时，会自动防止冲突。例如，如果作业A和作业B必须同时写入一个特定数据集，系统(通过启动程序)不会允许两个作业同时运行。作为代替，无论哪个作业先运行都会使拟处理另一个作业的启动程序等待，直到第一个作业运行完毕。

7.4 使用 JES 和启动程序管理作业和输出

让我们看一看JES和z/OS的启动程序是如何共同工作来处理批处理作业的，来看两个场景。

7.4.1 批处理作业场景 1

设想您现在是一个z/OS应用程序员正在为没有专业技术的用户开发一个程序。您的程序假设读取几个文件，写入另外几个文件，并且生成打印的报告。这个程序将会在z/OS上以批处理作业的形式运行。

需操作系统中什么样的功能才能满足您程序的要求呢？并且，您的程序将如何去获取这些系统功能呢？

首先，您需要一类特殊的语言通知操作系统您的需要。在z/OS中，这类语言就是作业控制语言(JCL)。JCL的使用在第201页第6章："使用JCL和SDSF"中详细解释，但是现在，假设JCL提供给您了从操作系统中为批处理作业请求资源和服务的方法。

您也许要为批处理作业请求编译和执行程序需要的功能，以及程序运行时分配存储空间。

使用JCL，您可以指定如下内容：

- 您是谁(由于安全原因非常重要)
- 系统处理您的程序时需要什么样的资源(程序，文件，内存)和服务。例如，您也许需要做如下的事情：
  - 将编译器代码加载到内存。
  - 使编译器可访问您的源代码，也就是说，当编译器请求读取时，您的源代码可以写入编译器内存。
  - 为编译器代码，I/O缓冲区和工作区分配适量的内存。

[1]在一些案例下：这样的用法是正确的，而且可以为这些案例组建 JCL。在简单的批处理作业案例中，这样的冲突通常是不会被接受的。

- 使编译器可以访问输出磁盘数据集以接收目标码，通常被称作‘object deck’或者简单的OBJ。
- 使编译器可以访问一个可以告诉您可能存在的错误的打印文件。
- 条件允许的话，使z/OS将新生成的OBJ加载到内存(但是如果编译失败则跳过这一步)。

236

- 为您的程序分配一些内存使用。
- 使您的程序可以访问所有的输入输出文件。
- 使您的程序可以访问一个打印机来输出最终消息。

这样，您要求操作系统：

- 将JCL转化成描述所需资源的控制块。
- 分配所需资源(程序，内存，文件)。
- 适时的调度执行，例如，您的程序只有当编译成功时才会运行。
- 当程序执行完毕释放资源。

在z/OS中完成以上任务的是JES和一个批处理启动程序程序。

把JES看成是队列中等待执行的作业的管理者。它管理着这作业集的优先级和他们相关的输入数据以及输出结果。一旦JES释放(分配)了作业，启动程序就使用JCL卡片上的语句来指定该作业需要的资源。

上面描述的JCL被称作作业---由编译和执行两个步骤顺序组成。作业中的作业步都会被顺序执行。为了使作业执行必须将其提交到JES。为了使您的任务更加简化，z/OS在一个名为SYS1.PROCLIB的数据集中提供了一个过程集合。一个过程是准备好运行的JCL语句的集合。

过程 (Procedure)
JCL 语句集合

例7-1展示了一个可以编译，连接-编辑，执行(这些步骤在第257页第8章”设计和开发z/OS上的应用程序”中会有介绍)程序的JCL过程。第一个作业步用 “//COBOL EXEC PGM = IGYCRCTL.”这一语句确定COBOL的编译器。而”//SYSLIN DD”这一语句则描述了编译器的输出(目标文件OBJ)。

目标文件(OBJ)是第二个作业步连接-编辑(通过IEWL程序)的输入。连接-编辑用以解决外部引用和引进或连接先前开发的公用程序(一种代码重用方式)。

在第三个作业步，程序被执行。

例7-1 编译，连接-编辑，执行程序的过程

```
000010 //IGYWCLG PROC LNGPRFX='IGY.V3R2M0',SYSLBLK=3200,
000020 //             LIBPRFX='CEE',GOPGM=GO
000030 //*
000040 //*****************************************************************
000050 //*                                                               *
000060 //*  Enterprise COBOL for z/OS and OS/390                         *
000070 //*               Version 3 Release 2 Modification 0              *
```

237

```
000080 //*                                                                  *
000090 //*  LICENSED MATERIALS - PROPERTY OF IBM.                           *
000100 //*                                                                  *
000110 //*  5655-G53 5648-A25 (C) COPYRIGHT IBM CORP. 1991, 2002            *
000120 //*  ALL RIGHTS RESERVED                                             *
000130 //*                                                                  *
000140 //*  US GOVERNMENT USERS RESTRICTED RIGHTS - USE,                    *
000150 //*  DUPLICATION OR DISCLOSURE RESTRICTED BY GSA                     *
000160 //*  ADP SCHEDULE CONTRACT WITH IBM CORP.                            *
000170 //*                                                                  *
000180 //******************************************************************
000190 //*
000300 //COBOL  EXEC PGM=IGYCRCTL,REGION=2048K
000310 //STEPLIB  DD  DSNAME=&LNGPRFX..SIGYCOMP,
000320 //             DISP=SHR
000330 //SYSPRINT DD  SYSOUT=*
000340 //SYSLIN   DD  DSNAME=&&LOADSET,UNIT=SYSDA,
000350 //             DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
000360 //             DCB=(BLKSIZE=&SYSLBLK)
000370 //SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(1,1))
000440 //LKED   EXEC PGM=HEWL,COND=(8,LT,COBOL),REGION=1024K
000450 //SYSLIB   DD  DSNAME=&LIBPRFX..SCEELKED,
000460 //             DISP=SHR
000470 //SYSPRINT DD  SYSOUT=*
000480 //SYSLIN   DD  DSNAME=&&LOADSET,DISP=(OLD,DELETE)
000490 //         DD  DDNAME=SYSIN
000500 //SYSLMOD  DD  DSNAME=&&GOSET(&GOPGM),SPACE=(TRK,(10,10,1)),
000510 //             UNIT=SYSDA,DISP=(MOD,PASS)
000520 //SYSUT1   DD  UNIT=SYSDA,SPACE=(TRK,(10,10))
000530 //GO     EXEC PGM=*.LKED.SYSLMOD,COND=((8,LT,COBOL),(4,LT,LKED)),
000540 //             REGION=2048K
000550 //STEPLIB  DD  DSNAME=&LIBPRFX..SCEERUN,
000560 //             DISP=SHR
000570 //SYSPRINT DD  SYSOUT=*
000580 //CEEDUMP  DD  SYSOUT=*
000590 //SYSUDUMP DD  SYSOUT=*
```

您可以写一些简单的JCL来调用一个过程如例7-2 所示。这个例子中，我们增加了其他的DD语句，例如：

```
// COBOL.SYSIN DD*
```

它包含了COBOL源代码。

238 *例7-2 COBOL程序*

```
000001 //COBOL1 JOB (POK,999),MGELINSKI,MSGLEVEL=(1,1),MSGCLASS=X,
000002 //  CLASS=A,NOTIFY=&SYSUID
000003 /*JOBPARM SYSAFF=*
```

```
000004 // JCLLIB   ORDER=(IGY.SIGYPROC)
000005 //*
000006 //RUNIVP EXEC IGYWCLG,PARM.COBOL=RENT,REGION=1400K,
000007 //             PARM.LKED='LIST,XREF,LET,MAP'
000008 //COBOL.STEPLIB DD DSN=IGY.SIGYCOMP,
000009 //             DISP=SHR
000010 //COBOL.SYSIN DD *
000011        IDENTIFICATION DIVISION.
000012        PROGRAM-ID.    CALLIVP1.
000013        AUTHOR.        STUDENT PROGRAMMER.
000014        INSTALLATION.  MY UNIVERSITY
000015        DATE-WRITTEN.  JUL 27, 2004.
000016        DATE-COMPILED.
000017        /
000018        ENVIRONMENT DIVISION.
000019        CONFIGURATION SECTION.
000020        SOURCE-COMPUTER.  IBM-390.
000021        OBJECT-COMPUTER.  IBM-390.
000022
000023        PROCEDURE DIVISION.
000024            DISPLAY "***** HELLO WORLD *****" UPON CONSOLE.
000025            STOP RUN.
000026
000027 //GO.SYSOUT DD SYSOUT=*
000028 //
```

在执行一个作业步时，程序被z/OS控制，而并非JES(图7-2)。处理流程的此处也需要一个假脱机处理(spooling)功能。

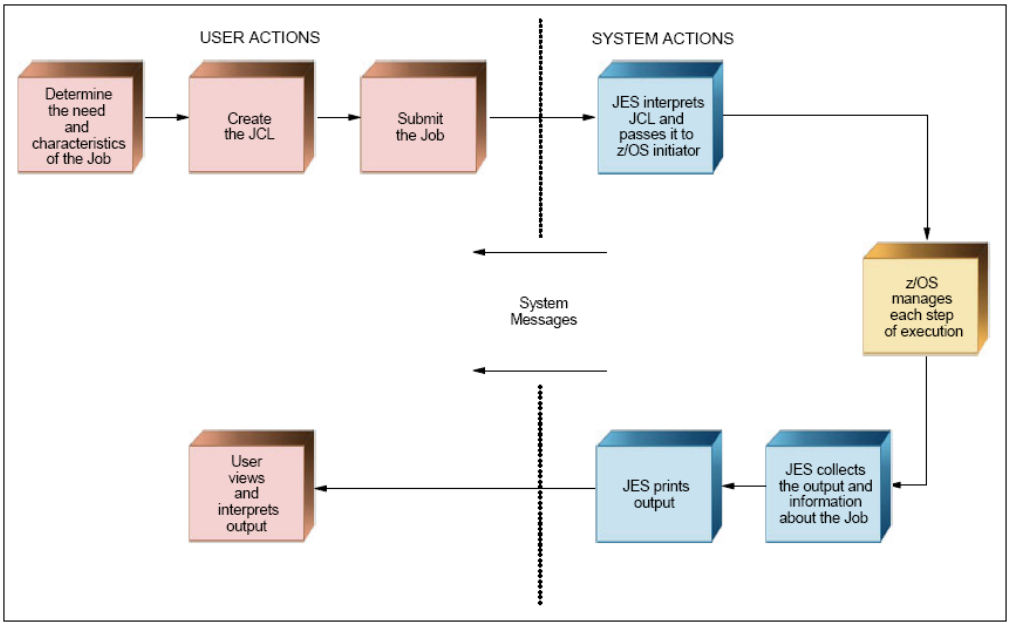


图7-2 对JCL的相关处理行为

Spooling是系统处理其任务时使用的方法，包括：

- 使用直接存取存储设备(DASD)上的存储空间作为缓存来减少在外围设备和即将运行的程序间传送数据时的造成的处理延迟。
- 为后续过程处理或输出在中间设备上读写输入输出流。
- 当计算机忙于其他工作时做诸如打印之类的操作。

有两种spooling：输入和输出。这两种都会改进程序的读输入和写输出的性能。

在JCL中实现输入spooling，您要需要声明//DD *，它会定义一个文件，文件的内容就是JCL中//DD *语句和/*语句之间的内容。所有逻辑记录必须有80个字符。如图7-3所示，这种情况下，文件会读取和存储在一个特定的JES2 spool区域(硬盘上一个巨大的JES文件)。

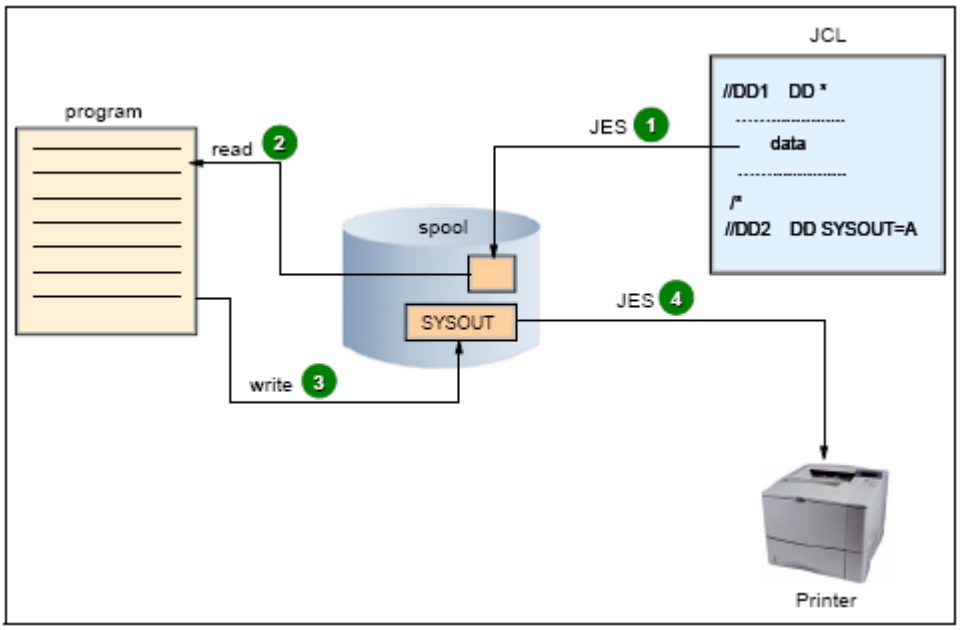


图7-3 Spooling

然后，当执行程序，要求读取该数据时，JES2从spool中选择记录，把它们传输给程序(以硬盘速度)。

为了在JCL中实现输出的spooling，您要在DD语句中指定关键字SYSOUT。SYSOUT在Spool中定义了一个空文件，分配132字节长的逻辑记录，这些记录采用打印格式(EBCDIC/ASCII/UNICODE)。JES在解析到有SYSOUT关键字的DD卡片时分配该文件，并且在后面的作业步程序中使用它。通常，在作业运行的最后，这个文件会由一个JES功能函数打印出来。

7.4.2 批处理作业场景 2

设想您现在想要为一个主文件建立一个备份，并且从另一个文件(更新文件)中读取记录来更新这个主文件。如果这样的话，您需要一个具有两个作业步的作业。第一步，您的作业要读取这个主文件，并将它写入磁带。第二步，另一个程序(可以用COBOL语言来实现)从更新文件中读取记录，并在主文件中查找相对应的记录。

这个程序更新已有的记录(如果找到了对应记录)或者如果需要则增加一条新的记录。

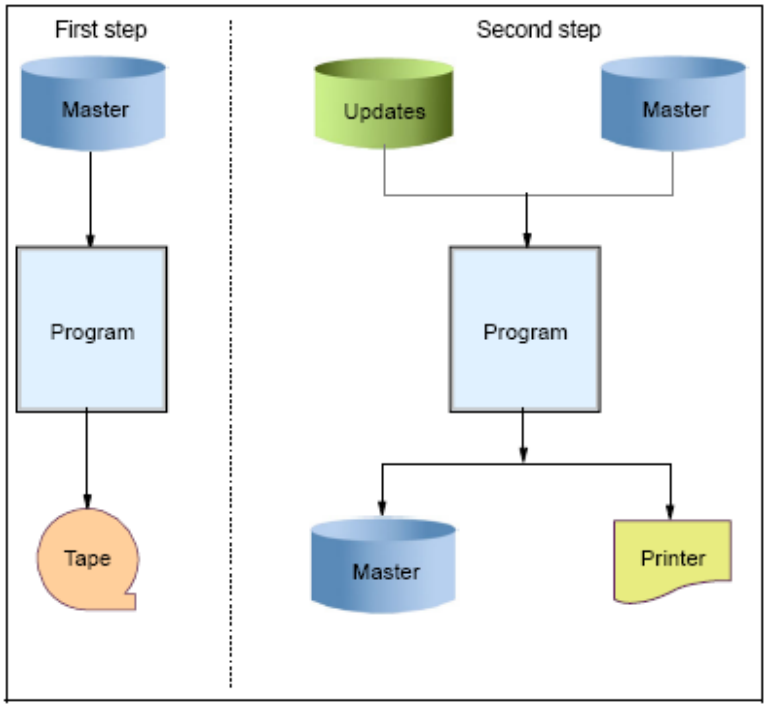
241

这个场景中，需要操作系统中哪些功能来满足您的需求呢？

建立一个两步的作业需要规定如下：

- 您是谁
- 这个作业需要什么资源，例如如下所示：
  - 加载备份程序(您已经编译好的)。
  - 系统需要分配多少内存来容纳备份程序，I/O缓冲区，和工作区域。
  - 使备份程序可以访问输出磁带数据集，来接收备份，复件，和主文件数据集本身。
  - 在程序结尾，向操作系统表明现在您的更新程序要被加载到内存(但是，如果备份程序运行失败，这个步骤就不应当执行)。
  - 使更新程序可以访问更新文件和主文件。
  - 使您的程序可以访问一个打印机来输出最终消息。

您的JCL必须包含两步，第一步指定备份程序所需的资源，第二步指定更新程序所需的资源(图7-4)。



242

图7-4 场景2

逻辑上来说，如果第一步由于任何原因而失败，第二步是不会被执行的。第二步会有一个//DD SYSOUT语句来指出对输出spooling的需求。

只有当有足够的资源可用时，作业才会被允许执行。这样，系统就可以更加高效：JES在程序运行开始前和结束后管理作业；在程序运行过程中，基本控制程序来

管理作业。

z/OS系统提供两种作业输入子系统 ：JES2和JES3。本章节讨论了JES2.请在第246页第7.6章节"JES2和JES3比较"中查看两者简单的比较。

7.5 贯穿系统的作业流程

让我们来详细的看一下一个作业是如何被JES和批处理启动程序程序来共同处理的。

在一个作业的生命周期中，JES2和z/OS的基本控制程序掌握着处理过程的不同阶段。作业队列包含了等待运行的作业，当前正在运行的作业，等待着输出生成的作业，输出正在生成的作业和等待着被清除出系统的作业。

总体来说，一个作业需要经历以下几个阶段：

- 输入
- 转化
- 处理
- 输出
- 打印/打孔(硬拷贝)
- 清除

检查点(Checkpoint)
一个点，在该点记录着系统和作业的状态信息，以使作业步可以稍候重启。

在批处理作业处理过程中，有许多的检查点(Checkpoint)会出现。检查点是在处理过程中的一个点，这个点上记录着系统和作业的状态信息(在一个叫检查点数据集的文件中)。检查点可以让作业步在因为某种错误异常终止后重新开始运行。

图7-5显示了一个作业在批处理过程中的不同阶段。

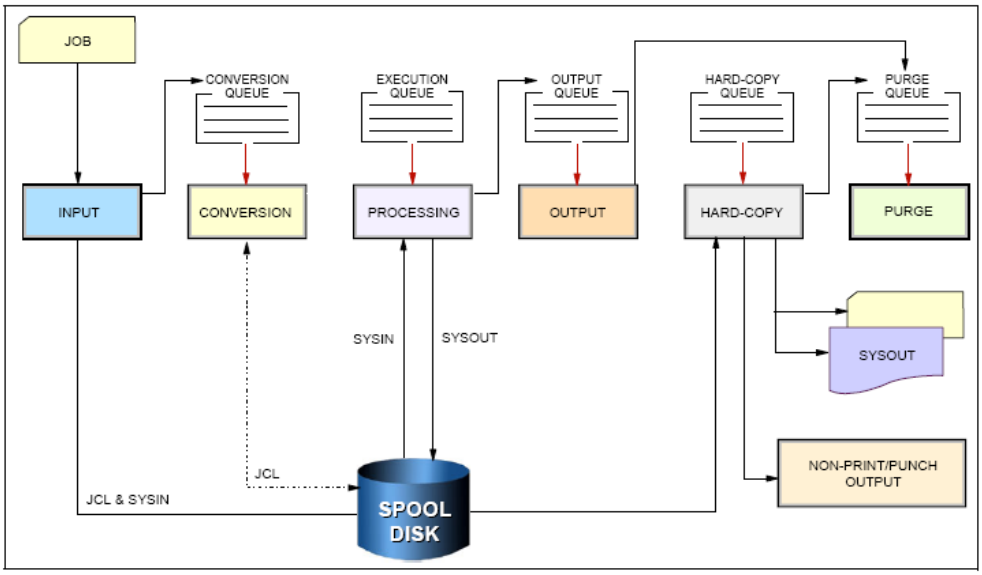


图7-5 贯穿系统的作业流程

1. 输入阶段

JES2以输入流的方式从输入设备，或者以内部阅读器的方式从其他程序，或者从作业入口网络的其他节点来接收作业。

内部阅读器是一个可以被其他程序调用，向JES2提交作业、控制语句和的命令的程序。任何一个运行在z/OS中的作业可以使用内部阅读器来向JES2传送一个输入流。JES2可以通过多个内部阅读器同时接收多个作业。

系统程序员定义了内部阅读器程序可用于处理除启动任务(STC)和TSO请求之外的所有批处理作业。

JES2读取输入流，为每一个作业JCL语句分配一个作业标识符。JES2将作业的JCL，可选择的JES2控制语句和SYSIN数据放置到一个叫spool数据集的DASD数据集。接着JES2从spool数据集中选择作业进行处理及之后的运行。

2. 转化阶段

JES2使用一个转化程序来分析作业的JCL语句。这个转化程序接收作业的JCL并与过程库的JCL进行合并。过程库可以通过JCLLIB JCL语句来定义，或者

系统/用户过程库可以在JES2启动过程中的PROC xx DD语句中声明。然后，JES2将组合好的JCL转换成JES2和启动程序都可以识别的转换器/解释器文本。接着，JES2将这些转换器/解释器文本存储在spool数据集中。如果JES2检测到任何JCL错误，它会发布消息，作业就会被送到输出处理队列而并不被执行。如果没有错误，JES2就会将这个作业放到执行队列中去。

3. 处理阶段

在处理阶段，JES2对来自启动程序的作业请求发出响应。JES2从等待运行的作业队列中选择作业，然后发送到启动程序。

启动程序是属于z/OS的系统程序，但是被JES或z/OS系统中的工作负载管理(Workload management ,WLM)组件控制，它启动作业，分配所需资源，允许该作业与正在运行的作业进行竞争(WLM在第104页第3.5章节”什么是工作负载管理”中详细讨论过)。

JES2启动程序由操作员启动或者当系统初始化的时候由JES2自动启动。它通过JES2初始化语句定义。为了更有效的利用可使用的系统资源，系统会将每一个启动程序与一个或多个作业类相关联。在遵从排列作业优先级情况下，启动程序选择那些作业类与其分配类相匹配的作业运行。

WLM启动程序根据性能目标，批处理作业负载的相对重要性，以及系统处理更多作业的能力由系统自动启动。启动程序基于作业的服务类和它们被指定的执行顺序来选择作业。作业通过JOBCLASS JES2初始化语句来传输到WLM启动程序。

4. 输出阶段

系统输出(SYSOUT)
指定作业输出的目的位置
系统生成的输出。

JES2控制所有的SYSOUT处理。SYSOUT是系统产生的输出；也就是一个作业产生的所有输出。该输出包含了必须被打印的系统信息，用户要求的必须被打印或者打孔的数据集。作业结束后，JES2通过输出类和设备设置要求来分析作业输出的特征；然后JES2将有相似特征的数据集归为一组。JES2将输出放入队列，进行打印或者打孔处理。

5. 硬拷贝阶段

JES2通过输出类、通道代码、优先级和其他标准选择输出进行处理。输出队列包含将要本地处理或者远程处理的输出。为一个特定的作业处理了所有的输出后，JES2将作业放到清理队列中。

245

清理(Purge)
作业完成时，释放分配给该作业的 spool 空间

6. 清理阶段

当一个作业所有的处理过程完成后，JES2会释放分配给作业的spool空间，使得空间可以分配给后续的作业所使用。JES2接着向操作者发送一个消息表明作业已经从系统中清理了。

7.6 JES2 与 JES3 比较

前面提到过，IBM提供了两种作业输入子系统 ：JES2和JES3。许多情况下，JES2和JES3有相似的功能：它们向系统读入作业，将其转化成内部机器可识别的形式，选择作业进行处理，处理作业的输出，然后从系统中清理作业。

在只有一个处理器的主机装置中，JES3为系统用户提供了磁带设置，依赖性作业控制，和最终期限安排，而在同一个系统中的JES2则要求用户通过其他方法来管理这些行为。在多处理器配置的装置中，两者有显著的不同，主要在于JES2是独立控制作业处理功能的。也就是说，在这样的配置中，每一个JES2处理器控制它自身的作业输入，作业调度和作业输出处理。大多数装置使用JES2，就像我们在文章中使用的例子一样。

246

图7-6列出了JES2和JES3的一些不同：

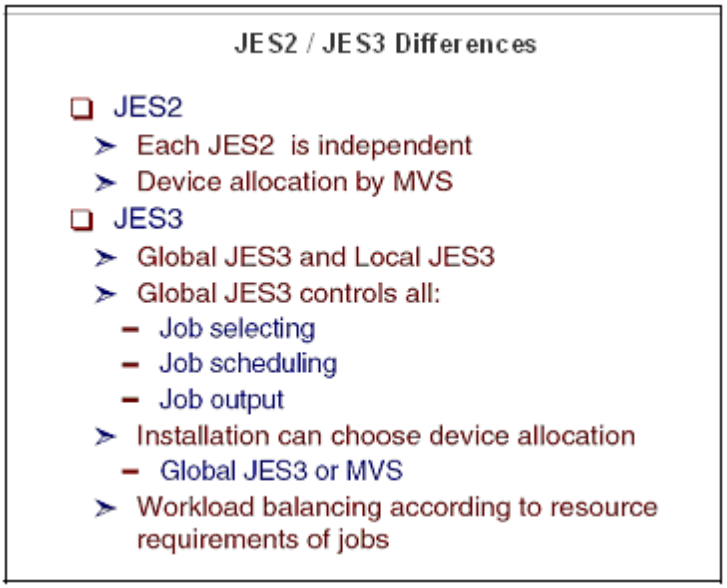


图7-6 JES2/JES3的不同

当多个z/OS系统集群时(一种系统综合体)，在同一个系统综合体中，是可以配置JES2与其他JES2系统来分享spool和检查点数据集的。这种配置称为多通道

spool(Multi-Access Spool, MAS)。相反，JES3通过一个单一的全局JES3处理器来集中控制它所有的处理功能。这个全局处理器为所有其他JES3系统提供了作业选择，调度，以及设备分配功能。

7.7 总结

批处理是z/OS最基本的功能。许多批处理作业是并行运行的，并且JCL用来控制每个作业的操作。正确使用JCL参数(特别是DD语句中的DISP参数)可以让那些可能需要访问同样数据集的作业并行，异步的执行。

启动程序是处理JCL的系统程序，它在地址空间建立必要的环境，并在同一个地址空间中运行批处理作业。多启动程序(每个都有一个地址空间)允许批处理作业并行执行。

操作系统的一个目标是在最佳使用系统资源的情况下处理工作。为了达到这个目标，在关键阶段资源管理要做到如下：

- 在作业处理前，为作业保留输入和输出资源
- 在作业处理中，管理spool中的SYSIN和SYSOUT数据
- 在作业处理后，将已完成作业的使用过的资源全部释放，使其可被其他作业使用

z/OS与JES共同管理作业和资源。JES为系统接收作业，调度它们交由z/OS处理，控制它们的输出处理。JES是等待执行作业队列的管理者。它管理作业的优先级，作业相关的输入数据和输出结果。启动程序使用JCL记录中的声明来指定每一个被JES释放(或者分配)的作业所需要的资源。

IBM提供两种作业输入子系统 ：JES2和JES3。在许多情况下，JES2和JES3有相同的功能。

作业生命周期中，JES和z/OS基础控制程序控制着整个处理过程的不同阶段。作业以队列形式被管理：等待运行的作业(转化队列)，现在正在运行的作业(执行队列)，等待生成输出的作业(输出队列)，已经生成输出的作业(硬拷贝队列)，等待从系统中清理的作业(清理队列)。

本章关键术语		
批处理作业(batch job)	检查点(checkpoint)	启动程序(initiator)
作业输入子系统 (JES)	过程(Procedure)	清理(purge)
假脱机处理(spooling)	SYSIN	SYSOUT

7.8 复习题

为了帮助您理解本章的内容，请完成下列问题：

1. 什么是批处理？

2. z/OS为什么需要JES？

3. 在一个作业的生命周期中，JES2有哪些典型的处理？

4. SPOOL缩写代表什么？

248 5. 启动程序执行哪些作业？

7.9 练习

这些练习包含了以下主题：

- “学习系统卷” 第249页
- “在作业中使用实用程序” 第249页
- “检查TSO登录JCL” 第250页
- “探索主目录” 第250页
- “使用SDSF” 第250页
- “使用TSO REXX和ISPF” 第252页

7.9.1 学习系统卷

使用ISPF功能来探索一些系统卷。下面是我们关心的：

- 检查VSAM数据集的命名。注意单词”DATA”和”INDEX”为最后的限定词。
- 寻找spool区域。这可能需要根据数据集名进行猜测。它有多大？
- 寻找基础系统库，诸如SYS1.PROCLIB和其他类似的。看看它们的成员名。
- 考虑以成员列表的形式显示的ISPF统计区域。这和源库和执行库的有什么不同？

7.9.2 在作业中使用实用程序

z/OS有一个叫做IEBGENER的实用程序可以用来复制数据。它有四个DD语句：

- SYSIN为控制语句。我们可以使用DD DUMMY，因为在这个作业中我们不需要任何控制语句。
- SYSPRINT传送来自程序的消息。在这个实验中我们使用SYSOUT = X。
- SYSUT1是输入数据。
- SYSUT2是输出数据。

程序的基本功能是将SYSUT1指向的数据集复制到SYSUT2指向的数据集。两者必须是顺序数据集或者是库的一个成员。 249

程序自动的从输入数据集获得数据控制模块(DCB)属性并将其应用到输出数据集。写一个JCL作业，将yourid.JCL(TEST1)的成员内容输出到SYSOUT = X。

7.9.3 检查 TSO 登录 JCL

处理TSO登录的密码面板包含了用来创建TSO会话的JCL过程的名字。系统中存在几个带有不同特征的过程。

我们来看ISPFPROC过程。可以请老师帮助找到正确的ISPFPROC所在的库。

- 被执行的基本TSO程序的名字是什么？
- 为什么有如此多的DD语句？注意并置。

寻找IKJACCNT过程。这是一个最小的TSO登录过程。

7.9.4 探索主目录

到ISPF的选项6中，做以下操作：

- 使用LISTC LEVEL(SYS1)命令列出主目录中所有SYS1数据集的基本信息。
- 注意到它们是NONVASM或者CLUSTER(并且关联到DATA和INDEX条目)。CLUSTER是指VASM数据集。
- 使用PA1键结束列表(如需帮助请看第141页，4.3.3节的 “使用PA1键”)。
- 使用LISTC LEVEL(SYS1) ALL命令来得到一个扩展了更多信息的列表。

  注意NONVSAM数据集的卷和设备类型。这是目录的基本信息。

- 使用LISTC LEVEL(xxx) 来观察别名(ALIAS)级别的数据集输出，注意它来自一个用户目录。

注意：如果您键入profile命令时使用了NOPREFIX参数，当您输入LISTC 和LISTC ALL命令，它会生成一个系统范围的结果显示。这些命令允许您看到主目录中的所有条目，包括别名条目。

7.9.5 使用 SDSF

从ISPF主选项菜单上，找到并选择系统显示查找工具(SDSF)。这个实用程序允许
250 您显示输出数据集。ISPF主选项菜单中的选项一般比第一个控制板中列出的选项更多，其上有关于如何显示附加选项的说明。

回到第223页6.14.1”创建一个简单的作业”，如果需要则重复第五步。这样可以为这次练习提供一个作业列表。

SDSF练习1

当浏览输出清单时，假设您想将其永久地保存到数据集以便以后查看。在命令输入行，输入PRINT D。一个窗口会提示您输入您想保存到的数据集名称。您可以使用一个已经存在的数据集或者新创建一个。

在这个例子中，输入yourid.cobol.list来创建一个新的数据集。在‘disposition’区域，键入NEW。按下回车，回到先前的界面。注意屏幕右上角显示PRINT OPENED。这意味着您现在可以打印作业清单了。在命令输入行，输入PRINT。在屏幕的右上角显示的将是打印的行数(xxx LINES PRINTED)。这意味着列表已经被放到您创建的数据集中了。在命令行输入PRINT CLOSE。现在在屏幕右上角，您可以看到PRINT CLOSED.

现在我们看一下您创建的数据集：yourid.cobol.list，并且查看一下作业清单。跳至=3.4然后输入您的用户ID。您的所有数据集的列表将会显示出来。找到yourid.cobol.list，在旁边的命令输入区域键入B，您可以看到当您使用SDSF时看到的作业清单再次显示出来了。您可以回到SDSF ST 中 清除(P)您的作业清单，因为您已经有了永久保存的副本。

回到SDSF主控制板，输入LOG显示系统所有的活动日志。这里您可以看到操作员可见的大部分的信息。例如，在列表的底部，您可以看到突出显示的操作员可以回复的Reply信息。

```
/R xx,/ DISP TRAN ALL
```

滚动到底部查看结果。注意SDSF Log命令中的操作员命令前面必须有斜杠(/)，这样才会被识别为系统命令。

现在在命令输入区输入M，按下F7，这时会显示日志的顶端。输入F和您的用户ID来显示与您的用户ID相关的第一个条目。很有可能就是您登录到TSO的记录。接着输入F youridX，X代表了您前面提交的一个作业。这时您可以看到您的作业被JES2的内部阅读器接收，并且接着有几行信息显示您的作业运行的状态。也许您会看到一个JCL错误，或者‘youridX started | ended’。

SDSF 练习2

这个练习要使用上面的打印功能。将日志按照上一个练习的操作方式保存到数*据集。*

SDSF 练习3

在这个练习中，您要从Log屏幕上输入操作员命令。在命令输入行中输入以下命令，并观察结果显示：

/D A, L	列出系统中所有活动的作业
/D U,,,A80, 24	列出当前在线的DASD 卷
/V A88, OFFLINE	滚动到底部看结果(M F8)
/DU,,, A88, 2	检查它的状态；注意脱机卷不会显示VOLSER。当一个卷脱机时，您可以运行实用程序，例如ICKDSF来格式化一个卷。
/V A88, ONLINE	滚动到底部，查看结果。
/D U,,, A88,2	检查它的状态；现在显示了VOLSER。

/C U = yourid	取消一个作业(这个情况下是您的TSO会话)
Logon yourid	重新登录到您自己的ID

7.9.6 使用 TSO REXX 和 ISPF

在USER.CLIST数据集中，有一个叫做ITSODSN的REXX程序。在任何一个ISPF命令输入区输入：TSO ITSODSN都可以运行这个程序。它会提示您输入您想创建的数据集的名字。您不需要输入yourid，因为如果您的前缀(prefix)是可用的，TSO会将其加入到数据集名。有两种类型的数据集可以供您选择，顺序数据集或者分区数据集。程序还会询问您想把数据集存储到哪一个卷。然后它就会分配数据集，将您的用户id附加到数据集名上。跳至=3.4，找到数据集，使用S选项检查数据集是不是与您想要一样。

REXX 练习1

在REXX程序中，您可以在代码中找到数据集的一些特征，例如LRECL和BLKSIZE。修改程序使其可以提示用户输入任何他们想要的数据集特征。您也可以以您喜欢的任何一种方式来修改程序。提示：在您开始之前为程序做好备份。

REXX 练习2

TSO和批处理下的REXX可以直接访问其他子系统，就像您在程序中已经看到的，它使用放在引号里的TSO命令直接分配了一个数据集。另一个在REXX程序外部执行功能的方法是通过主机命令环境。一些主机命令环境示例如下：

252

TSO	
MVS	使REXX在非TSO环境中运行
ISPEXEC	在TSO下访问ISPF环境

修改REXX程序，使分配数据集以后，使用ISPF Edit命令打开数据集，输入一些数据，使用PF3推出，接着用=3.4来检查您的数据集。记住，如果数据集是分区的(PO)，您必须打开一个成员。您可以用以下形式使用任意成员名：yourid.name(成员名)。

提示：

- 在上述主机命令环境中，第二种格式更加简单。
- 注意REXX的”if then else”逻辑的使用和逻辑中的”do end”。
- 使用命令： ADDRESS ISPEXEC “edit DATASET(…)”。

253

Part 2

第 2 部分 z/OS 上的应用程序编程

在该部分，我们介绍开发一个简单的运行在z/OS上的程序所能使用的工具和实用程序。以下的章节指导学生进行应用程序设计、选择编程语言以及使用一个运行时环境。

255

256

8

第 8 章 设计和开发 z/OS 上的应用程序

目标：作为公司里最新的z/OS应用程序设计者或程序员，您会被要求设计和编写新的程序，或者修改现有的程序以满足您们公司的业务要求。这就要求您能完全理解各种用户的需求，并且知道去使用z/OS的哪一项系统服务。

本章简单概述一个新应用程序的通用设计，代码编写和测试周期。本章中的大部分信息对于计算机的各种平台均基本适用，并非只专用于主机。

在本章结束之后，您应当能够掌握：

- 描述应用程序设计者和应用程序员所扮演的角色。
- 列出设计一个z/OS应用程序所需要考虑的主要事项。
- 描述批处理和在线处理对于应用程序的优缺点。
- 简单描述测试一个z/OS新应用程序的过程。
- 列举z/OS作为新应用程序开发平台的三个优势。

8.1 应用程序设计师与程序员

应用程序设计和应用程序开发的任务是截然不同的，可以在不同的教科书里讲解。在大型的z/OS使用单位，完成这每个任务可能需要不同的部门。本章提供一个对这些工作角色的概述，并且阐述每种技能是如何适合一个z/OS上典型的应用程序开发生命周期的。

应用程序设计者负责为一个重要的业务需求确定最好的编程方案。任何成功的设计都依赖于设计者对业务本身的知识，知晓在主机组织中其他的角色例如编程和数据库设计，并且了解与业务相关的软硬件。简而言之，设计者必须对整个项目有一个全面的了解。

在这个过程中另一个角色是业务系统分析师。分析师负责与一个特定部门(会计、销售、生产控制、制造等等)的用户一起工作以便确定应用程序的业务需求。象应用程序设计者一样，业务系统分析师需要对组织的业务目标以及信息系统的能力有一个广泛地了解。

应用程序
一套文件，组成用户可用的软件。

应用程序设计者从业务系统分析师和终端用户那里收集需求。设计者确定有哪些有用的IT资源可用来支持应用程序。然后应用程序设计师写一份设计说明文档，应用程序编程人员依此执行。

应用程序员负责开发和维护应用程序。也就是说，程序员为终端用户在主机上开发、测试和部署应用程序。根据应用程序设计师的说明文档，程序员利用各种工具编写应用程序。开发过程包括代码改变和编译、应用程序构造和单元测试的多次循环。

在开发的过程中，设计师和程序员必须和企业中的其他人员交流。例如，程序员经常在一个程序员小组内工作，这些程序员为相关应用程序模块开发代码。

当应用程序模块完成时，他们会通过一个包括功能测试、集成测试和系统测试的测试过程。在这个测试过程之后，应用程序员必须接受用户们的测试以确定代码是否真正达到了用户要求。

258

除了编写新的应用程序代码，程序员还负责维护和加强公司现有的主机应用程序。事实上，这经常是现今许多应用程序员在主机上的主要工作。当许多主机上仍然在使用COBOL和PL/1创建新应用程序的同时，用Java这样的语言开发新的应用程序在主机上也变得流行起来，就和在分布式平台一样。

8.2 为 z/OS 设计一个应用程序

设计
为一个给定的业务需求确定最佳编程方案的任务。

在设计的早期阶段，应用程序设计者要确定应用程序的特征。这些决定是基于许多标准，综合这些标准并对其进行详细审核以最终形成一个用户接受的解决方案。这些决定不是相互独立的，一个决定将影响其他的决定，并且所有的决定制定时必须考虑项目全局和项目的约束。

设计一个应用程序运行在z/OS平台上，和设计一个应用程序运行在其他平台上，遵循着很多同样的步骤，这些其他平台包括分布式环境。然而，z/OS引入了一些特殊需要考虑的事项。本章中提供了z/OS应用程序设计师在为一个应用程序设计步骤过程中所做决策的一些例子。下面的列表并不尽善尽美，只是使您了解该过程所涉及的一些方面。

- 260页的"为z/OS设计：批处理还是在线处理"。
- 260页的"为z/OS设计：数据源和访问方法"。
- 260页的"为z/OS设计：可用性和工作负载需求"。
- 261页的"为z/OS设计：意外情况处理"。

除了这些方面的决定，可能影响z/OS应用程序设计的其他因素可以包括对一种或多种编程语言的选择和开发环境选择。本章讨论的其他需要考虑的事项如下所列：

- 在267页"使用EBCDIC字符集"讨论使用主机字符集。
- 在269页"使用应用程序开发工具"里讨论一个交互开发环境(IDE)。
- 在277页第9章"在z/OS上使用编程语言"我们讨论不同语言之间的区别。

牢记最好的设计是那些在开始就牢记结果的设计。在开始设计之前，我们要清楚最终的目标。

8.2.1 为 z/OS 设计：批处理还是在线处理

在为z/OS和主机设计一个应用程序时，一个主要需要考虑的是这个应用程序设计成一个批处理程序还是在线处理程序。在许多情形下，结果是显然的，但大多数的程序都可以使用上述任意一种方式。那么，设计者如何来决定究竟用哪一种方式呢？

使用批处理或是在线处理的原因如下所列：

- 使用批处理的原因
  - 数据是存放在磁带上的。
  - 交易被提交来是做通宵的处理的。
  - 用户不需要在线访问数据。
- 使用在线处理的原因
  - 用户需要在线访问数据。
  - 需要很快的响应速度。

8.2.2 为 z/OS 设计：数据源和访问方法

这部分设计者需要考虑的典型事项包括：

- 哪些数据必须被储存？
- 将如何对数据进行访问？这包括对访问方式的选择。
- 有特别的需求还是可以可预料的需求。
- 我们选择PDS, VSAM还是一个诸如DB2的数据库管理系统。

8.2.3 为 z/OS 设计：可用性和工作负载需求

对于一个将要运行在z/OS上的一个应用程序，设计者必须能够回答以下的问题：

- 需要存取和访问的数据量有多少？
- 有需要共享数据吗？
- 需要的响应时间是多少？
- 对项目开支的约束是什么？
- 一次访问应用程序的用户量有多大？

260

对于应用程序的可用性需求是什么？(一天24小时，一周7天还每个工作日早上8：00到下午5：00，等等)

8.2.4 为 z/OS 设计：意外情况处理

是否会有一些不寻常的条件出现? 如果是这样，就需要在设计中考虑它们以防止在最后的应用程序中出现错误。例如，我们不能总设想着输入会按照预期的进行。

8.3 应用程序开发生命周期概述

一个应用程序是满足特定需求(解决一定问题)的一些程序的集合。从硬件或操作系统的角度来看，解决方案可以驻留在任何平台上，或者是各种平台的混合。

和其他的操作系统一样，在z/OS上应用程序的开发通常包括下面这些阶段：

- 设计阶段
  - 收集需求
    用户的，硬件的和软件的需求
  - 进行分析
  - 重复进行设计
    - 概要设计
    - 详细设计
  - 将设计移交给应用程序员
- 编码和测试应用程序
- 执行用户测试
  用户对系统进行功能和可用性测试
- 执行系统测试
  - 执行集合测试(测试应用程序和别的程序以确保所有程序继续按希望的功能执行)
  - 利用生产数据执行性能(容量)测试
- 上线生产 – 实际进行操作
  确保所有的文档都准备好(用户培训，操作过程)
- 维护阶段 – 继续每天的更改和对应用程序的加强。

开发
构建，测试和发布一个应用程序。

图8-1显示了应用程序开发生命周期的不同阶段的过程流程。

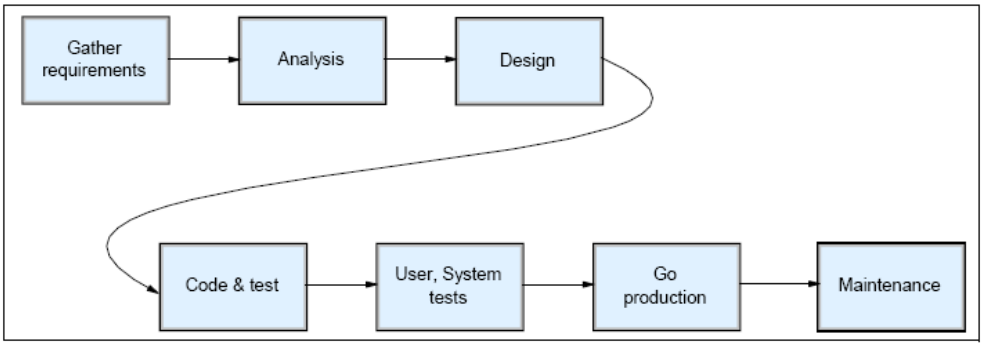


图8-1 应用程序开发的生命周期

图8-2描述了直到开始开发之前的设计阶段。一旦所有的需求都被收集到，分析过，核实过，并且做好了一个设计，这样我们就准备好将编程需求移交给应用程序员进行开发了。

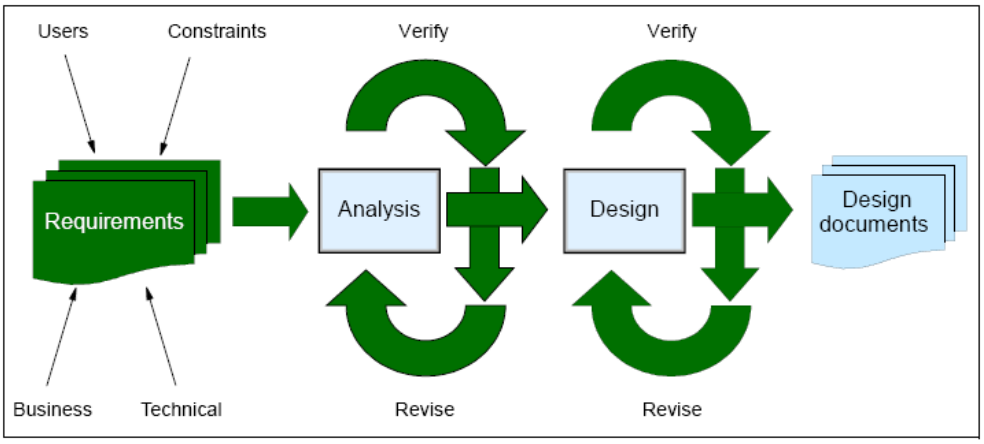


图8-2 设计阶段

程序员们拿到设计文档(编程需求)，就开始了如图8-3所示的编码、测试、修正、再测试的循环往复过程。

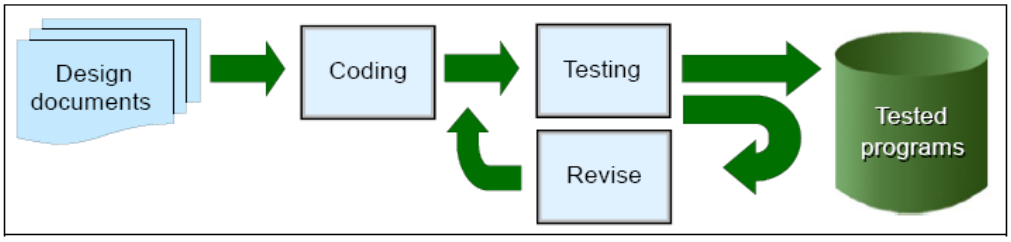


图8-3 开发阶段

在程序被程序员测试过之后，它们就成为了一系列正式用户和系统测试中的一部分。这些测试从一个用户的角度来看，是进行可用性和功能性验证，并且还要在

一个更大的框架中验证该应用程序的功能(如图8-4)。

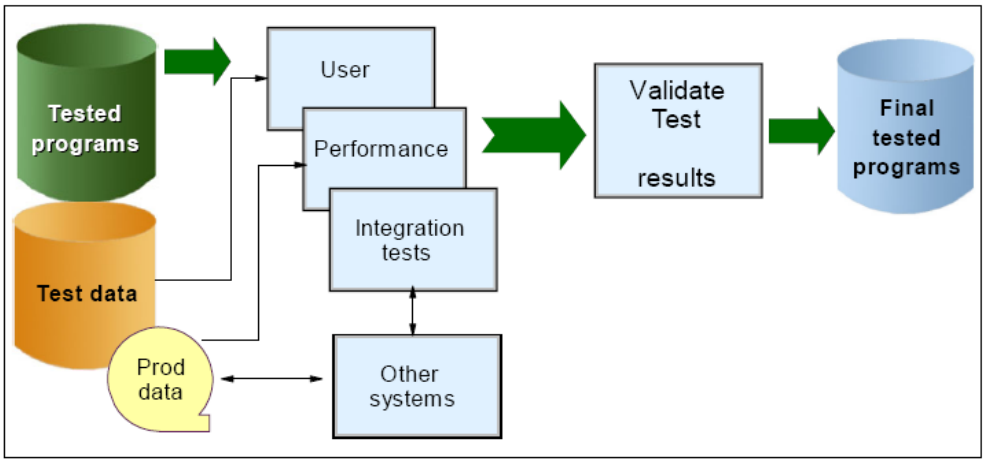


图8-4 测试

开发生命周期中的最后一个阶段就是进入生产环节并且使之保持稳定状态。作为进入生产环节的前提条件，开发团队必须提供相关的文档。这些通常包括用户培训方案以及操作流程。用户培训使用户熟悉新应用程序。操作流程文档能使操作部门接手应用程序，使其持续正常运行。

在生产环节中，对系统的变更和改进工作都由负责维护工作的团队掌控(很可能就是同一开发团队)。在应用程序系统生命周期中的该环节上，系统变更将被严密控制，并且在实际生产环节中加以实现之前必须通过严格的测试(如图8-5)。

263

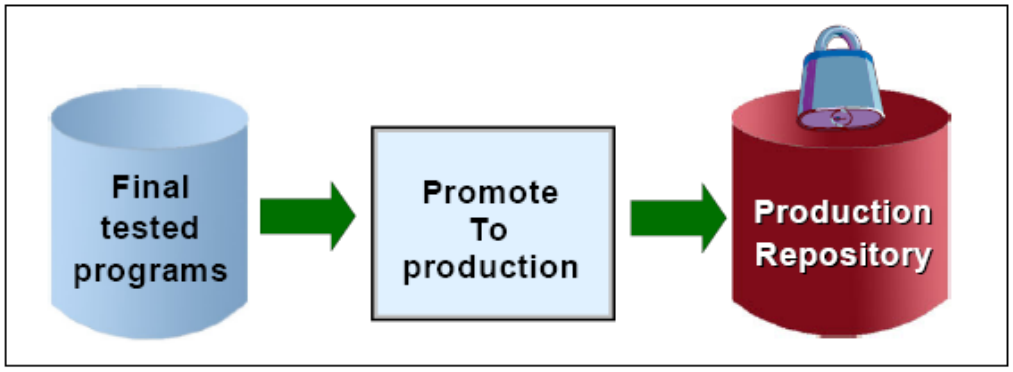


图8-5 生产

正如前面提到的，为满足用户需求或者解决问题，一个应用程序的解决方案可能被设计为能被部署在任何平台或者多种平台的联合体上。如图8-6所示，我们特定的应用程序能被部署在三个环境中的任何一个：Internet、企业网、中央站点。操作系统必须提供访问任意以上三种环境的途径和方法。

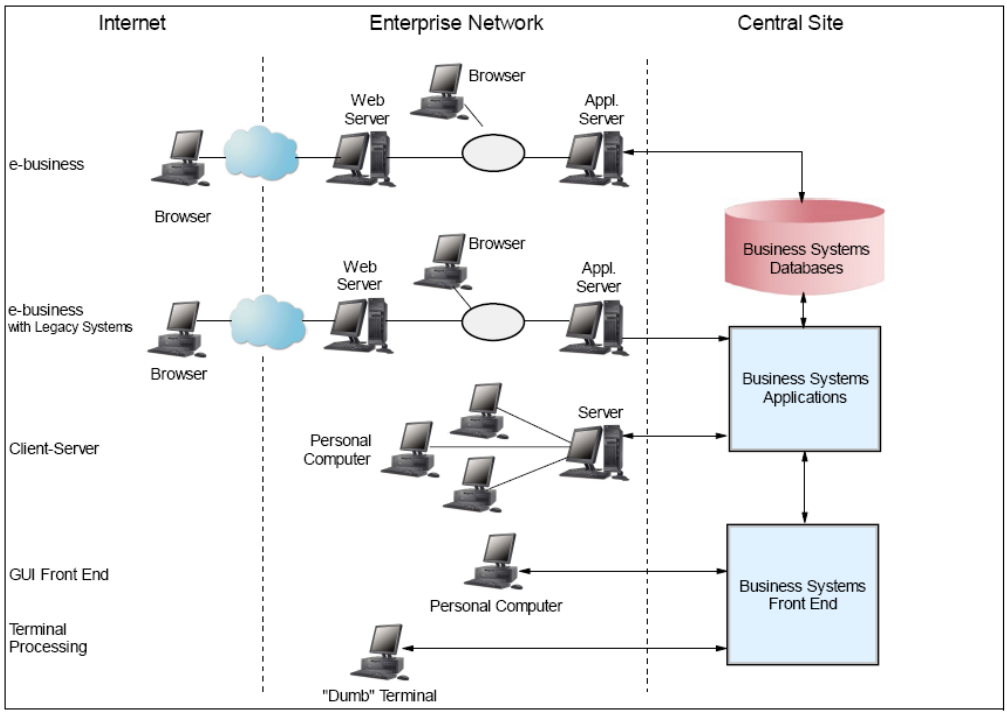


图8-6 增长的基础设施复杂性

要开始设计过程，我们必须首先判定我们需要达到的目标。基于项目的限制，我们确定如何并且通过何种途径来达到项目的既定目标。为此，我们组织与用户(那些需要该解决方案来解决问题的人)以及其他有利益相关者见面交流。

这些见面交流将能明确告诉我们该应用程序项目生命周期中的每一个后续步骤。在项目的某些阶段，我们再次召集用户来确认我们已经理解了他们的需求，确认我们的解决方案符合他们的要求。在这些项目的阶段里程标上，我们同样要求用户对我们已经完成的工作予以签字认可，这样一来我们才能继续到项目的下一个阶段。

8.3.1 为设计收集需求

当设计应用程序时，有很多种方法来分类需求：功能需求、非功能需求、新兴需求、系统需求、流程需求、开发约束以及操作约束——还有很多。

计算机应用程序操作数据，这些数据位于某处并且需要本地或者远程访问。应用程序处理这些数据，并施加以一些处理操作，并将结果呈现给最先请求数据的访问者。

平台
通常指一个操作系统，包含操作系统和硬件(环境)。

这个简单的描述涉及了许多处理过程及操作，它们包含了从计算机到软件产品的许多不同的需求。

尽管每一个应用程序设计都是一个独立的案例，有很多独特的需求，但是它们中的一些对于作为同一系统组成部分的所有应用程序来说都是相同的。不仅仅因为

它们隶属于同一个组成某信息系统的应用程序集合，而且他们是同一安装系统的一部分，连接到相同的外部系统。

多个系统作为一个整体面临的问题之一是：组件散布于各种不同的机器、不同的平台等等，每个都在一个服务器场环境下完成各自工作。

对使用zSeries解决方案而言一个重要的优点在于应用程序能够使用主机系统上的各种工具来维护。一些主机工具使不同平台依照工作负载和优先级，协作并安全地共享数据和资源成为可能。

如下是对应用程序各种不同类型需求的列表。列表项并非各不相同；一些列表项已经包含了别的表项。

- 可达性
- 可恢复性
- 可服务性
- 可用性
- 安全性
- 连通性
- 性能指标
- 资源管理
- 易用性
- 数据备份频率
- 可移植性
- Web服务
- 易改变性
- 交互性
- 故障预防与错误分析

8.4 在主机上开发应用程序

在分析完毕并做出决策方案之后，就交由应用程序员来处理了。开发程序员没有给与自行发挥的自由，而是需要遵循设计师给予的规范。然而，假设设计师本身也许并不是开发程序员，也可能会有由于编程局限导致的一些变更。但到了项目的这一环节上，我们不再讨论有关设计的变更，而仅仅讨论变更程序执行设计者指定任务的方式。

开发过程是迭代的，并且通常是模块层次上的。一个开发程序员通常要遵循以下的步骤：

1. 编写模块代码
2. 测试模块功能
3. 对模块作相应的更正
4. 重复回到以上步骤2，直至成功。

一个模块的测试完成之后，需要登记完成并且实施有效冻结以保证在未来产生相应变更时，它将再次被测试。当足够的多的模块编码和测试完成，他们会在不断增加的复杂度测试中被一起测试。

266 这个流程将被重复直至所有的模块都编码测试完毕。尽管流程图显示测试仅仅在开发完成之后，但实际上测试在开发期间持续不断地发生着。

8.4.1 使用 EBCDIC 字符集

z/OS数据集是以扩展二进制码十进制交换码(EBCDIC)字符集。这是在8位ASCII(美国信息交换标准码)被广泛使用之前发展起来的8位字符集。与之对照的是，z/OS UNIX文件是以ASCII编码的。

绝大多数您所熟悉的系统都是使用ASCII。您需要了解将数据从基于ASCII的系统转移到EBCDIC编码系统时整个编码模式的不同之处。通常来说这种变换是由内部处理的，例如当文本从PC上的3270模拟器发送到一个TSO会话。但是，当传送程序时，决不能使用编码转换，应该指定二进制传输。有时传送文本甚至会由于某个特定字符而发生问题，例如"或"符号(|)或者逻辑"否"，这时程序员就必须查看转译字符的实际值。

关于EBCDIC和ASCII的位分配表见附录D,"EBCDIC-ASCII表"见第593页，这些表可能对我们的讨论有所帮助。ASCII和EBCDIC都是8位字符集。区别在于把位分配给特定字符的方式不同。如下为一些实例：

字符	EBCDIC	ASCII
A	11000001(x'C1')	01000001(x'41')
B	11000010(x'C2')	01000010(x'42')
a	10000001(x'81')	01100001(x'61')
1	11110001(x'F1')	00110001(x'31')
空格	01000000(x'40')	00100000(x'20')

尽管ASCII的安排方式似乎更加符合逻辑，但海量的以EBCDIC存在的数据以及大量的对字符集敏感的程序使将所有这样已经存在的数据和程序都转换成ASCII的想法不切实际。

字符集都有对照序列，对应于字符位的二进制值。例如，A比B的值无论在ASCII还是EBCDIC中都要小。对照序列对于分类和几乎所有需要扫描并且处理字符串的程序而言都非常重要。在两个字符集中公共字符的一般对照序列如下：

	EBCDIC	ASCII
最低值	空格	空格
	标点符号	标点符号
	小写字母	数字

	大写字母	大写字母
最高值	数字	小写字母

267

例如，在EBCDIC中”a”小于”A”，但在ASCII中”a”大于”A”。在ASCII中数字字符小于任何字母字符，但在EBCDIC中却是大于任何的字母。A-Z和a-z在ASCII中是两个连续的序列。在EBCDIC中一些字母却被间隔开。如果我们用Z减去A在ASCII中得到的结果是25。如果我们用Z减去A在EBCDIC中得到的结果却是40。(取决于一些字母之间的间隔的二进制值)

将简单的字符串在ASCII和EBCDIC之间转换很平常。但是当要转换的字符不在目标代码的标准字符集中时，情况将复杂得多。一个典型的例子就是主要的主机编程语言(PL/I)中用到的逻辑”否”符号；在ASCII字符集中没有对应的字符。同样的，一些C语言用到的ASCII字符集同样在最初的EBCDIC字符集中找不到，尽管在晚些时候还是加入到了EBCDIC中。如今依然有些对货币分标号(¢)以及符号(^)和一些晦涩少用的符号持有疑惑。

主机同样使用一些版本的双字节字符集(DBCS)，主要是对于亚洲文字。很多PC程序也使用相同的字符集。

传统的主机编程并不使用特定的符号作为终结符。特别是空字符和换行符(或者说CL/LF字符对)没有被使用。并且没有二进制文件和文本文件的概念。只要适当编程，字节总是可以被解释成为EBCDIC或者ASCII或者其他。当这样的文件被传送到主机打印机，它将尝试地将其解释为EBCDIC，原因在于打印机对字符集敏感。z/OS Web服务器通常存储ASCII文件因为数据将被希望得到ASCII数据的PC浏览器程序解析。倘若没有人尝试用主机打印机来打印ASCII文件(或者在3270上显示)，系统就不关心使用的是何种字符集。

8.4.2 主机上的 Unicode

Unicode作为业界标准，是一种以16位字符集来表示所有的现代语言和I/T协议中的文本和符号。主机(为单字节字符使用EBCDIC)，PC以及各种RISC系统都使用相同的Unicode方案。

Unicode由Unicode协会负责维护。(http://www.unicode.org/)

主机应用程序已日益增加了对Unicode的使用。最新型号的z系列主机包含了很多独特的专为Unicode设计的硬件指令。在本书成书之时，主机上Unicode的主要使用是在Java中。然而，z/OS的中间件产品同样已经开始使用Unicode，而且这是在不久的将来的一个发展方向。

268

8.4.3 z/OS 应用程序员所使用的接口

当操作系统被开发出来以满足计算市场的需求时，应用程序就编写运行在这些操作系统上。多年后，很多的应用程序被开发出来并运行在z/OS之上，以及后来的

UNIX系统上。为了适应用户对UNIX应用程序的需要，z/OS在其传统的接口基础上又添加了一整套完整的UNIX操作系统。z/OS所实现的UNIX接口被共同称为z/OS UNIX系统服务，或者简称为z/OS UNIX.

对z/OS开发者而言最为常见的接口就是TSO/E以及它的面板驱动的接口，ISPF,使用的都是3270终端。通常，开发者在自己的个人计算机上使用3270模拟器，而不是真正的3270终端。模拟器可以提供给开发者一些附加的功能，例如多会话窗口，以及从PC上上传和下载代码和数据。TSO/E同其他z/OS用户接口在第4章详细描述。见第127页"TSO/E，ISPF和UNIX：z/OS的交互工具"。

典型的z/OS程序开发涉及到使用一种行编辑器来操作源代码文件、使用批处理作业编译程序、以及多种不同的机制来测试代码。基于3270终端功能的交互式调试器，可用于常用的语言。本章将介绍开发一个运行于z/OS上的简单程序所需的工具和实用程序。

仅仅是用到z/OS的z/OS UNIX部分的开发可以通过3270或者TSO/E(使用其他编辑器)使用Telnet会话(vi编辑器可用)登录系统，或者从运行X Server的个人计算机上的X Window系统会话登录系统。X Server接口方式比较少被用到。

另一种可选的方法就是和各种中间件产品联合使用。例如，WebSphere产品为个人计算机提供了GUI开发工具。这些工具通过与z/OS的TCP/IP连接方式，自动调用新应用程序开发和测试阶段所需要的主机组件。

本书将在第3部分"z/OS的联机工作负载"讨论联机应用程序和中间件产品的使用，该部分涵盖网络通讯，数据库管理以及Web服务等主题。

8.4.4 使用应用程序开发工具

要编写能经过反复考验的代码需要用到主机上的相关工具。对开发程序员而言最基本的工具就是ISPF编辑器。

当使用诸如COBOL或者PL/1这样的语言开发传统的、面向过程的程序的时候，程序员通常登录到主机系统并使用IDE或者ISPF编辑器来修改代码并编译运行。程序员使用一个公共库(例如IBM软件配置库管理器或者SCLM)来存放开发中的代码。该库允许程序员将代码登入(check in)或者登出(check out)，确保不会影响到他人的工作。SCLM以一个选项的形式包含于ISPF的主菜单中。

269

可执行
一个程序文件在一个特定的环境下准备运行。

为了简单化，源代码可以存放一个分区数据集(PDS)中。然而，使用PDS既不能提供变更控制也不能像SCLM那样防止同一个版本的代码的多个更新。所以，任何地方只要我们写上"登出"或"保存"到SCLM，就意味着您可以将它们相应替换为"编辑一个PDS成员"或"保存一个PDS成员"。

当源代码修改完成时，程序员就提交一个JCL文件来编译源代码、绑定应用程序模块并创建可执行代码进行测试。程序员对程序功能进行"单元测试"。程序员通过作业监控以及检查工具跟踪正在运行的程序、观察输出并对源代码或者其他对象作出适当修正。有时，当失败发生时，程序可能会创建内存转储(DUMP)。程序员

同样可以使用工具解析产生的转储输出并通过追踪可执行代码以确定问题产生点。

一些主机应用程序员目前已经转向于使用应用程序交互式开发环境(IDE)工具以加快整个编辑/编译/测试过程。IDE使应用程序员能在工作站就完成编辑、测试并调试代码的工作，而不必直接在主机系统上进行。IDE特别适用于构建那些"混合型"应用程序，这些系统通常用到基于宿主端的程序或者交易处理系统，但同时也包含了类似于网络浏览器这样的用户接口。

当所有的组件都完成开发和测试之后，应用程序员将其打包成某种适当的部署形式，并交给负责整合产品代码部署的团队。

z/OS上可用的应用程序使能服务包括：

- Language Environment®(语言环境)
- C/C++ IBM Open Class® Library(C/C++ IBM开放类库)
- DCE Application Support1(DCE 应用程序支持1)
- Encina® Toolkit Executive2(Encina 工具包执行器2)
- C/C++ 调试工具
- DFSORT
- GDDM®-PGF
- GDDM-REXX
- HLASM 工具包
- 诸如COBOL, PL/I, and Fortran等传统语言

8.4.5 进行调试会话

应用程序员通过"单元测试"来测试一个正在开发的某特定模块的功能。程序员通过作业监控以及查看软件诸如SDSF(见第215页6.8节"理解SDSF"描述)跟踪编译作业的执行、查 看编译的输出并验证单元测试的结果。如果有必要，程序员还将对源代码或者其他对象进行适当的修正。

有时，程序可能会把问题发生时刻的内存进行转储。此时，z/OS应用程序员可能使用诸如IBM Debug Tool以及IBM Fault Analyzer解析产生的转储输出并通过追踪可执行代码以找到问题或者异常代码。

一个典型的开发会话包括以下步骤：

1. 登录到z/OS系统上。
2. 键入ISPF并从SCLM库(或者PDS)中打开/登出源代码。
3. 对源代码作出适当修改。
4. 提交JCL作业，构建应用程序并进行测试性运行。
5. 转换到SDSF查看作业的执行状态。
6. 在SDSF中查看作业输出并检查有无错误产生。

7. 查看转储输出查找程序缺陷。[1]

8. 重新运行”编译/链接/执行”的作业并查看结果。

9. 验证作业输出的正确性。

10. 将源代码保存到到SCLM(或者PDS)中。

有的应用程序员已经转向于使用应用程序交互式开发环境(IDE)工具以加快整个编辑/编译/测试过程。使用诸如WebSphere Studio Enterprise Developer 这样的IDE可以在工作站上就完成编辑代码的工作，而不必直接在主机系统上，使编译过程”脱离平台”进行，并可以远程调试。

IDE特别适用于构建那些混合型应用程序，这些系统是基于宿主端的COBOL程序或者诸如CICS或者IMS的交易处理系统，同时也包含了类似于网络浏览器这样的用户接口。无论是用高级语言构建联机事务处理(OLTP)组件还是HTML的前端用户界面组件，IDE都提供了统一的开发环境。一旦这些组件的开发和测试完成，它们将被打包成某种适当的部署形式，并交由负责整合产品代码部署的团队。

除了新的应用程序编码工作，应用程序员还要负责对已有的主机应用程序进行维护和改进。实际上，这通常是如今很多主机系统的高级语言程序员的首要工作。在绝大多数的z/OS用户仍在使用COBOL或者PL/1创建新的应用程序的同时，用诸如Java这样的语言编程也在主机平台上逐渐流行起来，就如分布式平台一样。

交易
一种活动或请求。他们会因为订单、改变或添加等要求而更新主文件。

然而，对于我们当中那些对传统语言抱有兴趣的人而言，在主机上使用诸如COBOL或者PL/I这样的高级语言进行的程序开发如今依然十分普遍和广泛。全世界有不计其数的主机系统程序应用于生产领域，而这些程序对于使用它们的企业的日常业务来说，是相当关键和重要的。COBOL和其他高级语言的程序员需要维护现有的这些代码，并对这些程序不断加以更新和修改。

另外，很多的企业继续地使用COBOL或者其他传统语言来构建新的应用程序业务逻辑，同时IBM也在不断的改进和加强高级语言的编译器，使其拥有新的功能和特点，以允许这些语言能继续使用更新的技术和数据形式。

8.4.6 实施系统测试

本阶段所实施的测试和开发阶段所实施的测试工作的区别在于，现阶段我们是把整个应用程序作为一个整体，和其他的应用程序联合起来进行测试。由于我们需要了解整个应用程序的工作全貌如何，而不仅仅是其中的一部分，于是这种测试只有当所有的应用程序编码工作完成后才能执行。

[1] 术语‘编程缺陷’(programming bug)的由来通常源于 1945 年的美国海军中尉霍波。故事的开始，霍波中尉正在哈佛大学测试 Mark II Aiken Relay 计算机。一天，之前正常运行的程序突然神秘地失败了。在检测过程中，操作员发现一只飞蛾被夹在继电器间，造成了短路(早期的计算机占据了大量的空间，有成千上万个真空管组成)。在 1945 年 9 月 9 日的日志里记载了这只飞蛾和下面这句话：“第一次发现了一个真正的臭虫”，并且他们“除掉了机器中的臭虫”。

本阶段要实施的测试包括：

- 用户测试——测试应用程序的功能和易用性。
- 集成测试——将新的应用程序同其他应用程序共同测试，以检验它们之间连接和协同是否同预期一致。
- 性能与压力测试——使用实际的生产数据或者至少达到实际生产数据量来测试应用程序，观察当请求很多时，应用程序的响应情况。

272

用户和集成测试的结果需要确认，以保证它们符合要求。此外，应用程序性能必须符合需求。这些测试所涌现的任何问题都必须在实际进入到生产环节前弄清楚。在测试阶段我们遇到的问题数目能够很好反映我们前期设计工作的好坏。

8.5 在主机上生产上线

“生产上线”所指的动作并不像简单地打开一个上面写着”现在应用程序准备好上线了”的开关那么简单。它比这复杂得多。从一个项目到另一个，程序生产上线的方式也不尽相同。在一些情况中，我们打算替换一个已经存在的系统，我们可能会决定在彻底转为使用新系统之前，先将新旧系统并行运行一段时间。在这种情况下，我们对新旧系统运行处理同样的数据，并比较运行结果。如果在一段时间之后我们对结果满意，我们则转为使用新系统；如果发现问题，我们可以将问题修正后继续并行运行两个系统，直到不再出现任何新问题为止。

在另一些情况中，我们面对的是全新的系统，我们可能有一个切换的日子，从那时开始使用新系统。我们可能只需要不到一天时间就可以开始使用。即使是新系统，我们常常会替换一些系统形式，即便是手工系统，如果有必要我们依然可以做这种并行测试。

无论使用哪一种方法生产上线，在系统能被真正交付运转之前，都依然需要我们关注每一个可能没有妥当的环节。其中的一项工作便是提供系统相关的文档，以及运行和使用的流程说明书。我们也需要培训每一个使用系统的人。

当所有的文档和培训都完成后，我们可以将运行应用程序的职责交给操作组，并把维护应用程序的职责交给维护团队。在一些情况下，开发团队同时就是维护团队。

到了现在的环节，应用程序的开发生命周期已经趋于稳态，我们进入到了应用程序的维护阶段。此后，我们仅仅要负责系统的改进优化和日常变更。由于应用程序已经处于变更控制流程的严格管理之下了，于是在变更应用程序于生产环节之前，所有的变更都需要依照变更控制流程进行测试。通过这个途径能向终端用户保证应用程序正常稳定地运行。

273

8.6 总结

本章描述了应用程序设计师和应用程序员的不同角色。重点讨论了涵盖设计和开发一个运行于主机环境的应用程序所需要的几种类型的决策方案。这并不意味着

这些流程与其他平台上的大不相同，但是其中的一些问题和结论确实有所不同。

本章随后描述了设计和开发一个基于z/OS的应用程序的整个生命周期。整个流程从需求收集阶段开始，该阶段应用程序设计师分析用户需求以决定如何最好地满足用户。可能会存在很多的方法获得解决方案；分析和设计阶段的目标就在于确保所做的选择是最佳方案。尽管时间是任何项目都要考虑的问题，但这里"最佳"并不意味着"最快"，而是指整体上看来最好的方案，综合考虑到了用户需求和问题分析。

EBCDIC字符集不同与ASCII字符集。在字符对应的基础上，两个不同字符集之间的转译并不复杂。当考虑到了对照序列，二者的差异才会更加明显，将一个字符集转换到另一个字符集的转换程序也许会十分简单，也许会非常复杂。在8位ASCII字符集被广泛认可并使用之前，EBCDIC字符集就已经成了确定的行业标准。

在设计阶段的末期，开发程序员这个角色开始接管工作。开发程序员必须将应用程序设计说明书变成没有错误的程序代码。在每个模块在加入到整个系统之前，开发程序员都要测试相关代码，这个过程贯穿于整个开发阶段。开发程序员必须修正每一个已经侦测出来的逻辑错误并将修改更新后的模块加入到已经测试过的程序组中。

一个应用程序很少孤立存在。更多的情况，一个应用程序常常是一组更大的应用程序集的一部分，通常一个系统的输出是另一个系统的输入。为了确认把一个系统加入到更大的应用程序集中协同工作不会产生问题，应用程序员进行系统测试和集成测试。这些测试自行设计，并且很多测试结果由实际的应用程序用户来验证。在系统测试中找到的任何问题，都必须解决并且进行反复测试，一切无误后流程才能进入到下一步处理。

274

在成功完成系统测试之后，应用程序就准备生产上线了。该阶段也有时被称为推广应用程序 。一旦被推广，应用程序代码将被更加严密的控制。在没有确定可靠性之前，公司不希望对其工作系统有任何的更改。在几乎所有的z/OS使用站点，都有一套严格的规则管理应用程序(或者应用程序内部的模块)的推广。以防止未经测试通过的代码污染"纯净"的系统。

在应用程序生命周期的该环节上，生命周期趋于稳态，在对已经生产上线系统所做的变更都仅是改进优化，或者功能性改动(例如，由于税法更改，薪资程序需要更改等)，或者是修正。

本章关键术语		
应用程序(application)	ASCII	数据库(database)
设计(design)	开发(develop)	EBCDIC
可执行的(executable)	平台(platform)	交易(transaction)

8.7 复习题

为帮助测试您对本章内容的理解，请完成如下问题：

1. 应用程序设计师和应用程序员有什么不同？ 哪一个角色需要对整个项目有

全局的了解？

2. 在应用程序开发生命周期的哪一个阶段，设计师召集用户见面？

3. 使用代码仓库管理源代码的目的是什么？

4. 应用程序开发生命周期中有哪些阶段？ 简要叙述每个阶段作了什么？

5. 如果您是某一个特定项目的设计师，并且新应用程序上线的时间要求非常紧迫，您将作何决策以减少整个项目总体时间？

6. 作为系统测试阶段的一部分，您需要完成性能测试。为什么您必须用生产数据来进行这个测试？

7. 给出一些可能的理由以决定为应用程序采用批处理方式而不是联机处理方式。

8. 为什么不把所有的文件都存储为ASCII格式，这样一来不就不用从EBCDIC转换过来了？

9

第 9 章 在 z/OS 上使用编程语言

目标：您作为公司里最新的z/OS应用程序员，您需要了解z/OS能支持哪些编程语言，并且如何根据一组给定的需求，确定哪种语言最为合适。

在本章结束之后，您应当能够掌握：

- 罗列在主机上的几种常见的编程语言
- 解释编译型语言与解释型语言之间的差异
- 创建一个简单的CLIST或者REXX程序
- 为联机应用系统选择合适的数据文件组织方式
- 对比高级语言和汇编语言各自的优点
- 解释数据集名称，DD名以及程序内部文件名之间的关系
- 解释z/OS语言环境的使用是如何影响应用系统设计师的决策

277

9.1 编程语言概述

计算机语言是人同计算机交流的途径。之所以需要它，是由于计算机只能以自身的机器语言(字位或字节)工作。如果人们使用机器语言就会十分的低效和不便。因此，我们用计算机语言来编写程序，这些程序随后会转换成计算机可以处理的机器语言。

编程语言
人们和计算机进行交流的方式。

目前有很多种计算机语言，它们都是由机器语言向更易书写的自然语言演化。一些语言已经改编成适用于它们拟解决的某类应用程序，和设计时使用的某类方法。"代"这个词被用来表明这种演化过程。

计算机语言的分类如下：

1. 机器语言，即第一代编程语言，直接的机器代码。
2. 汇编程序，即第二代编程语言，使用助记符来代表将被汇编程序最终转换成机器语言的指令，例如汇编语言。
3. 过程化语言，即第三代编程语言，也被称为"高级语言"，例如Pascal，FORTRAN，Algol，COBOL，PL/I，Basic以及C等。代码程序，称为源程序，必须经过编译步骤翻译。
4. 非过程化语言，即第四代语言，也被称为"4GL"，用于为数据库应用程序、报表产生以及查询实现预定义功能，例如RPG，CSP，QMF™。
5. 可视化编程语言，使用鼠标以及图标，例如Visual Basic以及Visual C++。
6. 超文本标记语言，用于书写万维网相关文档。
7. 面向对象语言，即OO技术，例如SmallTalk ，Java以及C++ 。
8. 其他语言，例如专用于3D应用程序的语言等。

代
计算机语言演化的阶段。

为了适应不断产生的新标准，每种计算机语言各自进行着自己的演变发展。以下部分将描述z/OS所支持的广泛使用的几种计算机语言。

- 汇编语言-"在z/OS上使用汇编语言"，第280页
- COBOL-"在z/OS上使用COBOL"，第282页
- PL/I-"在z/OS上使用PL/I"，第290页
- C/C++-"在z/OS上使用C/C++"，第294页
- Java-"在z/OS上使用Java"，第294页
- CLIST-"在z/OS上使用CLIST语言"，第296页
- REXX-"在z/OS上使用REXX"，第299页

对于这个列表，我们可以增加Shell脚本以及PERL在z/OS UNIX系统服务环境中的使用。

这里所讨论的编程语言，我们列出了它们的演变过程以及分类。有过程化与非过程化语言、编译型与解释型语言、机器依赖与非机器依赖语言。

汇编语言程序是机器依赖的，因为这种语言只是当前运行程序的机器语言的符号版本。每种机器的汇编语言指令各有不同，所以很可能在一台机器上编写的汇编程序不能直接移植到另一台机器上，而是需要我们使用另一台机器的指令集来重新编写汇编程序。用高级语言(HLL)编写的程序可以运行于其他平台上，但需要将其重新编译成目标平台所使用的机器语言。

本章介绍的大多数高级语言(HLL)是过程化语言。该类型语言非常适合编写结构化程序。非过程化语言，例如SQL和RPG，更适合特殊用途，例如报告生成。

大多数的高级语言(HLL)都被编译成机器语言，但也有一些是被解释执行。程序被最终编译为机器代码后，能被十分高效的反复执行。解释型语言在每次程序运行时都需要被解析、解释并执行。使用解释型语言节约了程序员的时间，但代价是增加了机器资源消耗。

关于编译型语言和解释型语言的优点将在9.11节进一步讨论。第301页"编译型语言对比解释型语言"

9.2 为 z/OS 选择合适的编程语言

在z/OS上进行程序开发，关于如何选择合适的编程语言，您需要考虑如下的方面：

- 这是何种类型的应用程序？
- 响应时间的需求如何？
- 开发和后续支持的预算限制如何？
- 项目的时间约束如何？
- 我们有必要因为某种特定语言较之所选择的一般性语言所具有的优势， 而采用这种语言编写一些子程序例程吗？
- 我们是采用编译型程序还是解释型程序？

279

接下来的部分将对主机均支持的几种语言进行上述的考虑。

9.3 在 z/OS 上使用汇编语言

汇编程序
汇编语言程序的编译器。

汇编语言是一种符号化的编程语言，它能以指令编码的形式代替直接以机器语言编码。它是一种在形式和内容上都与机器语言最为接近的符号编程语言。因此，在如下情况下，汇编语言是编程的最佳选择：

- 您需要在字节或者字位的级别上控制您的程序
- 您需要编写其他的符号编程语言(诸如COBOL、FORTRAN或者PL/I等)所无法提供的功能编写子例程[1]。

[1]子例程是常被其他程序频繁调用的程序，编写时应该始终考虑性能问题。编译语言是编写子例程的好的选择。

汇编语言由表示指令和注释的语句组成。指令语句是语言的有效部分，他们可以分为以下三种类别：

- 机器指令，指令集的机器语言指令的符号化表示，例如：
  - IBM Enterprise Systems Architecture/390 (ESA/390)
  - IBM z/Architecture

编译器
将高级语言语句集转化为低级别表示形式的软件。

称为机器指令是因为编译器会将它翻译成计算机可以执行的机器语言代码。

- 汇编指令，汇编程序在汇编源程序模块时所作特定操作的请求；例如：定义数据常量、保护存储区域以及定义标识原程序模块结束。
- 宏指令(或宏)，请求汇编程序对预定义的一组指令序列(称作"宏定义")进行处理。根据宏定义，汇编程序产生对应的机器和汇编指令并随后执行，这些汇编程序产生的指令如同本身就包含在源程序模块的原始输入中一样，被汇编程序执行时同等对待。

绑定器
绑定(链接编辑)目标卡片叠，生成装入模块。

汇编程序产生一个程序清单，该清单包含了在汇编过程的各个阶段所产生的相关信息[2]。它实际上就是汇编语言程序的编译器。

汇编程序同样也产生相关信息给其他的运行器，例如绑定器(或在更早的操作系统上叫做连接器)。

280

在计算机能够执行您的程序之前，目标代码(被称为目标卡片叠或者简称为OBJ)必须通过另一种处理，以解决对应的指令和数据的地址分配问题。这个处理过程称为"链接-编辑"，由绑定器实施。

装入模块
通过链接编辑器从目标模块中产生，可以直接装载运行。

绑定器或者链接编辑器(更多详细讨论，见10.3.7"如何使用链接编辑器"，第327页)使用目标卡片叠中的信息，并将其结合到装入模块中。在程序提取时间，绑定器产生的装入模块被装入到虚存中。在程序被载入之后，便可以运行。

图9-1展示以上步骤。

[2] 程序列表并非包含汇编过程中产生的全部信息。为了捕获可能在列表中的全部信息(或者更多)，z/OS程序员可以指定编译程序选项ADATA，控制编译程序创建SYSADATA文件作为输出文件。SYSADATA文件内容人们无法识别，只能用工具处理。相较于通过“listing scrapers”提取相似数据的老方法，使用SYSADATA文件更易于工具处理。

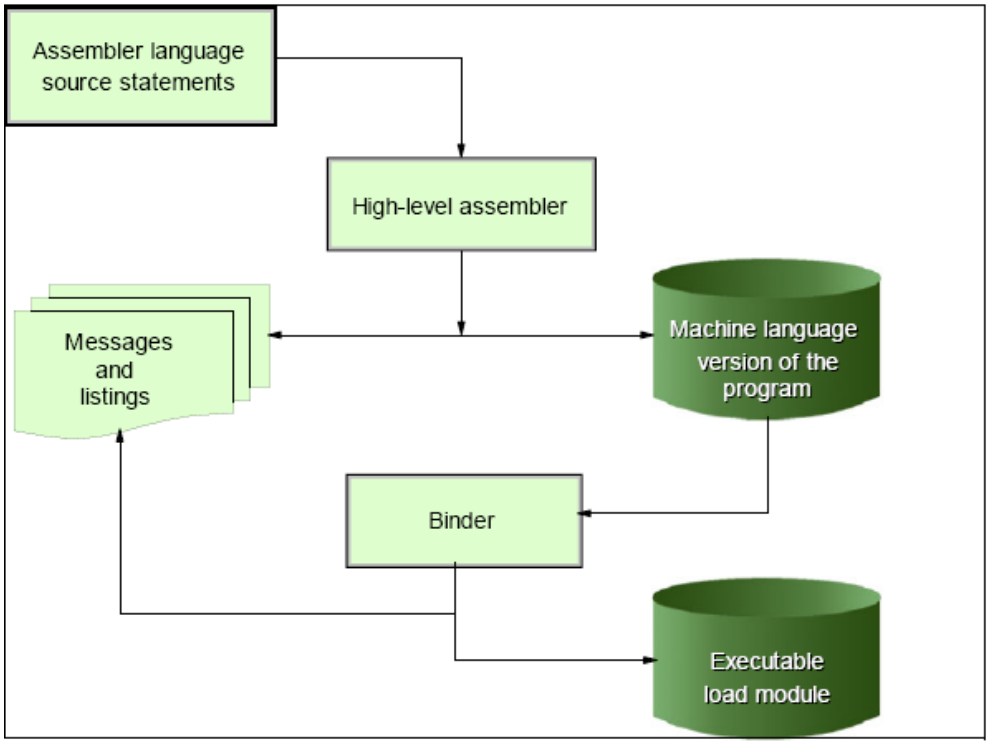


图9-1 编译源代码成为可执行模块

相关阅读：更多有关于在z/OS上使用汇编语言的信息，可以参阅IBM出版物，HLASM General Information, GC26-4943 和 HLASM Language Reference,SC26-4940。以上书籍可在以下网址获得：

http://www-03.ibm.com/systems/z/os/zos/bkserv/find_shelves.html

9.4 在 z/OS 上使用 COBOL

面向商业的通用语言(Common Business-Oriented Language，COBOL)，是一种类似于英语的编程语言，它被广泛使用于商务数据处理领域中开发面向业务的应用系统。COBOL几乎已经成为使用这种计算机语言进行计算机编程开发的通用术语。然而，在本章中，COBOL指的是产品IBM Enterprise COBOL for z/OS and OS/390®。

调试
调试软件意味着在源代码(程序逻辑)中定位错误。

除了COBOL语言所提供的传统特性，该版本的COBOL能通过COBOL功能函数，将COBOL应用系统集成到面向Web的业务流程当中。通过这个发布版本所具有的功能，应用开发者可以完成如下事情：

- 使用调试工具中新的调试功能
- 当应用系统运行在IMS的Java依赖区域(Java-dependtdent region)时，能与Java协同工作
- COBOL程序的组建简单化，并且能够和分布应用程序的Java组件协同操作。
- 增强了在标准的格式下(包括XML和Unicode)数据的交换及使用。

通过z/OS和OS/390的IBM企业版COBOL，COBOL与Java应用程序可以在电子商务的世界里协同工作。

COBOL编译器将产生程序清单，该清单包含了在编译过程的各个阶段所产生的相关信息。编译器同样也产生相关信息给其他的运行器，例如绑定器。

在计算机能够执行您的程序之前，目标卡片叠必须进行另一种处理，以解决对应的指令和数据的定位问题。这个处理过程称为链接编辑，由绑定器实施。

绑定器(更多详细讨论，见10.3.7"如何使用链接编辑器"，第327页)使用目标卡片叠中的信息，并将其组合放入到装入模块中。在程序提取时刻，绑定器产生的装入模块被装入到虚存中。在程序被载入之后，随后便可以运行。

第283页的图9-2展示了COBOL源代码语句转换成为可执行的装入模块的过程。

该过程类似于汇编语言程序。实际上，这个过程适用于所有需要编译的高级语言。

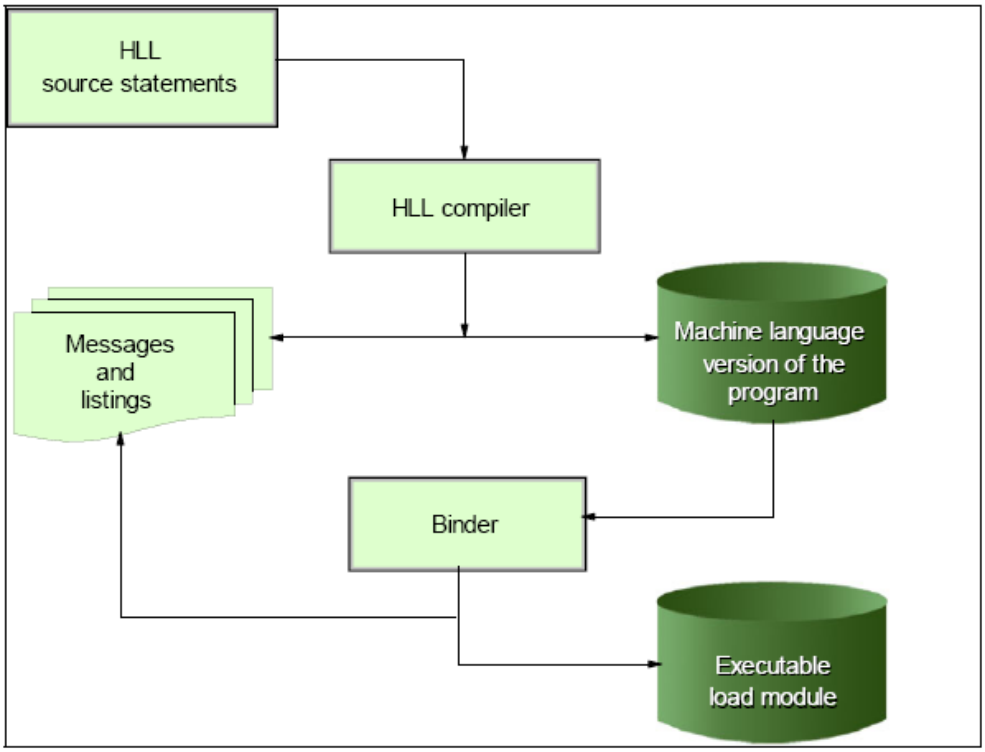


图9-2 从HLL源代码到可执行模块

9.4.1 COBOL 程序格式

除了"COPY"和"REPLACE"以及程序的结束符，一个COBOL源程序的语句、条目、段和节可以划分到以下四个部分中：

- 标识部(IDENTIFICATION DIVISION)，用于标识程序名字，如果需要，您可

以在该部分给出其他的标识信息。

- 环境部(ENVIRONMENT DIVISION)，用于描述程序所需依赖计算机环境的各个方面。
- 数据部(DATA DIVISION)，将您所需要的数据特性定义于该部分的如下节中：
  - 文件节(FILE SECTION) ，用于定义输入-输出操作所用到的数据
  - 连接节(LINKAGE SECTION)，用于描述来自于其他程序的数据。

当需要为程序内部处理定义数据时：

  - 工作存储节(WORKING-STORAGE SECTION)，静态的分配存储，并在整个运行单元的生命周期中保留。
  - 本地存储节(LOCAL-STORAGE SECTION)，在每一次程序被调用时刻分配存储，并在调用运行结束时刻释放存储。
  - 连接节(LINKAGE SECTION)，用于描述来自于其他程序的数据
- 过程部(PROCEDURE DIVIDISION)，用于指定数据操作相关的指令，以及与其他过程的接口。

程序的过程部可以分成不同的节和段，它们包含了句子和语句。如下详述：

  - 节：程序处理逻辑的一个逻辑子部分。一节都有一个节头以及可选的一个或多个段。一个PEFORM语句可以构成一节。一种类型的节用于声明。

    声明是由一组一个或多个用途更为特殊的节组成，在过程部的开头写明，第一节前要有关键字DECLARATIVES，并且最后一个节之后要紧跟关键字END DECLARATIVES。

  - 段：一个节、过程或者程序的子部分。一个语句可以构成一段。
  - 句子：由一个或者多个COBOL语句组成，以句号结尾。
  - 语句：执行COBOL处理的一个确定的步骤，例如将两个数字相加。
  - 短语：语句的子部分。

COBOL各部的示例：

例9-1 标识部：

```
 IDENTIFICATION DIVISION.
Program-ID. Helloprog.
Author. A. Programmer.
Installation.  Computing Laboratories.
Date-Written.  08/21/2002.
```

例9-2 环境部

输入-输出编码示例

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. computer-name.
OBJECT-COMPUTER. computer-name.
SPECIAL-NAMES.
  special-names-entries.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT [OPTIONAL] file-name-1
        ASSIGN TO system-name [FOR MULTIPLE {REEL | UNIT}]
        [.... .
I-O-CONTROL.
    SAME [RECORD] AREA FOR file-name-1 ... file-name-n.
```

关于用户提供信息的解释在例9-3之后。

例9-3 文件控制(FILE-CONTROL)中的输入输出文件

```
 IDENTIFICATION DIVISION.
. . .
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT filename ASSIGN TO assignment-name
    ORGANIZATION IS org ACCESS MODE IS access
    FILE STATUS IS file-status
. . .
DATA DIVISION.
FILE SECTION.
FD  filename
01  recordname
    nn . . . fieldlength & type
    nn . . . fieldlength & type
. . .
WORKING-STORAGE SECTION
01 file-status  PICTURE 99.
. . .
PROCEDURE DIVISION.
    . . .
    OPEN iomode filename
    . . .
    READ filename
    . . .
    WRITE recordname

    . . .
    CLOSE filename
    . . .
    STOP RUN.
```

285

- org代表组织方式，可以是SEQUENTIAL、LINE SEQUENTIAL、INDEXED

或者RELATIVE。

- access代表访问模式，可以是SEQUENTIAL、RANDOM或者DYNAMIC。
- iomode表示INPUT或者OUTPUT模式。如果您只需要从文件里读，用INPUT模式。如果您只需要写它，用OUTPUT或者EXTEND。如果您既要读又要写，指定I-O，组织方式是LINE SEQUENTIAL除外。
- 其他像filename，recordname，fieldname(如示例中的nn)，fieldlength以及type同样被指定。

9.4.2 COBOL 中 JCL 与程序文件之间的关系

示例9-4描述了JCL语句与COBOL程序中文件的关系。由于在程序中不需要引用到数据文件实际的物理位置，我们实现了设备独立性。也就是说，我们可以在不用改动程序的情况下改变数据存放的位置以及名称。我们只需要修改对应的JCL。

示例9-4 COBOL 中JCL 与程序文件之间的关系

```
//MYJOB    JOB
//STEP1    EXEC IGYWCLG
...
   INPUT-OUTPUT SECTION.
   FILE-CONTROL.
     SELECT INPUT ASSIGN TO INPUT1 .....
     SELECT DISKOUT ASSIGN TO OUTPUT1 ...
   FILE SECTION.
     FD INPUT1
        BLOCK CONTAINS...
        DATA RECORD IS RECORD-IN
     01 INPUT-RECORD
...
     FD OUTPUT1
        DATA RECORD IS RECOUT
     01 OUTPUT-RECORD
...
/*

//GO.INPUT1  DD DSN=MY.INPUT,DISP=SHR
//GO.OUTPUT1 DD DSN=MY.OUTPUT,DISP=OLD
```


示例9-4展示了一个COBOL程序编译、链接以及运行作业流，列出了文件程序语句和它们所参考的JCL语句。

COBOL SELECT语句创建了DDNAME(INPUT1和OUTPUT1)，和COBOL的FD名(INPUT1和OUTPUT1)之间对应的联系。COBOL的FD名同组条目INPUT-RECORD以及OUTPUT-RECORD相关联。

DD卡片INPUT1以及OUTPUT1与数据集MY.INPUT以及MY.OUT对应相关。我们这个例子所示联系的最终结果是记录从文件INPUT1读入，即从物理数据集MY.INPUT读入，并写入到文件OUTPUT1当中，即最终写入到物理数据集

MY.OUTPUT1当中。程序完全的独立于数据所存在的位置以及数据集的名字。

图9-3展示了示例9-4中物理数据集，JCL以及程序之间的关系。

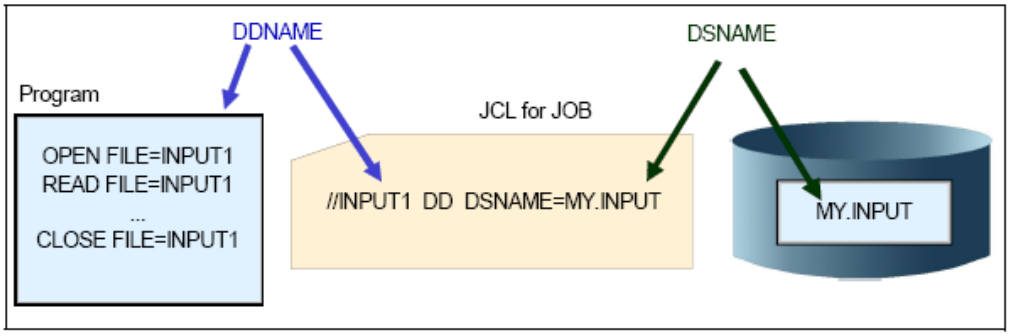


图9-3 数据集，JCL以及程序之间的关系

再次强调，由于程序并没有引用到任何物理数据集，因此在数据集名称或者位置发生改变时，我们不必重新编译程序。

9.4.3 在 UNIX 下运行 COBOL 程序

为了在UNIX环境下运行COBOL程序，您必须使用Enterprise COBOL或者COBOL for OS/390 and VM编译器。这些程序必须是可重入的，因此我们使用编译器和绑定器的选项RENT。

9.4.4 与 Java 方法交互通讯

为了达到与Java之间能够跨语言的协同工作，您必须遵照如下一些规则和指引：

- 使用Java本地接口(JNI)的服务
- 为数据类型编码
- 编译您的COBOL程序

您可以从COBOL程序调用那些用Java编写的方法，您同样也可以从Java程序中调用COBOL编写的方法。为了能够实现基本Java对象能力，您可以使用面向对象的COBOL语言。要使用到更多附加的Java功能，可以调用JNI服务。

由于Java程序可能是多线程，并且使用异步信号，因此需要在编译您的COBOL程序时指定THREAD选项。

9.4.5 创建一个 DLL 或 DLL 应用程序

一个动态链接库(或者称为DLL)是一种文件，包含了可执行代码和程序运行时需要绑定的数据。DLL中的代码和数据可以被几个应用程序所同时共享。创建一个DLL

或者DLL应用程序和创建一个常规的COBOL应用程序类似。包括了编写、编译以及链接源代码。

当编写DLL或者DLL应用程序时，所需要特殊考虑的事项包括：

- 确定装入模块或者应用程序的各部分如何彼此相互关联或者与其他DLL相关联。
- 决定使用何种链接或者调用机制。

根据您是想要一个DLL装入模块，还是想要一个引用另一个独立DLL的装入模块，您需要使用略微不同的编译器及绑定器选项。

9.4.6 构筑 OO 应用程序

您可以通过三种方法中的任一种，来使用面向对象COBOL的语法构筑应用程序。一个面向对象的应用程序通常起始于：

- 一个COBOL程序，可以任意命名。
- 一个包含了main方法的Java类定义。您可以通过Java命令指定包含了main方法的类名，并跟上零个或者多个命令行参数来运行应用程序。
- 一个包含了名为main的工厂方法的COBOL类定义。您可以通过Java命令指定包含了main方法的类名，并跟上零个或者多个命令行参数来运行应用程序。

288

相关阅读：更多有关于如何在z/OS使用COBOL的信息，可以参阅IBM出版物：Enterprise COBOL for z/OS and OS/390 V3R2 Language Reference，SC27-1408和Enterprise COBOL for z/OS and OS/390 V3R2 Programming Guide，SC27-1412。以上书籍可在以下网址获得：

http://www.ibm.com/servers/eserver/zseries/zos/bkserv/find_shelves.html

9.5 高级语言(HLL)中 JCL 与程序文件之间的关系

在第286页的第9.4.2节”COBOL中JCL与程序文件之间的关系”中，我们了解到了如何将COBOL程序与数据集名称以及存放位置隔离开来。通过符号文件引用物理文件的技术并不仅仅局限于COBOL当中；它在所有的高级语言甚至于汇编语言之都有应用。示例9-5是一个通用的高级语言程序通过符号文件名引用物理数据集的例子。

将程序与数据集名称和位置相隔离开来是正常的需求。但是有些情况下程序需要访问直接访问存储设备(DASD)上特定位置的特定数据集，这可以通过汇编语言甚至一些高级语言来实现。

将数据集名称或者其他此类信息”写定”在程序代码里并不是一个好的编程习惯。在程序代码中写定的值不能改变，因此在每次相关值需要改变的时候，程序都需要重新编译。将这些值从程序中脱离出来，就如同我们在程序里用符号名称来引用数据集一样，是一种更有效的编程习惯，这使得即使在数据集名称改变时，程序

依然能够工作运行，而不用重新编译。

示例9-5 高级语言中JCL与程序文件之间的关系

```
//MYJOB    JOB
//STEP1    EXEC CLG
...
   OPEN FILE=INPUT1
   OPEN FILE=OUTPUT1
   READ FILE=INPUT1
...
   WRITE FILE=OUTPUT1
...
   CLOSE FILE=INPUT1


   CLOSE FILE=OUTPUT1
/*
//GO.INPUT1  DD DSN=MY.INPUT,DISP=SHR
//GO.OUTPUT1 DD DSN=MY.OUTPUT,DISP=OLD
```


更多解释有关如何使用符号名称引用文件，参阅第209页6.5节，”z/OS为何使用符号文件名”。

9.6 在 z/OS 上使用 PL/I

PL/I(Programming language/I，发音为”P-L one”)，是一种全功能、通用的高级编程语言。它适合以下的开发：

- 商业应用程序
- 工程/科学应用程序
- 很多其他应用程序

编译一个PL/I源程序然后将目标卡片叠通过链接编辑生成装入模块的流程和COBOL基本一致。参阅第283页的示例9-2，第327页的10.3.7节”如何使用链接编辑器”，以及287页图9-3。

对于PL/I，JCL与程序文件之间的关系同COBOL以及其他的高级语言一致。参阅第287页图9-3以及第289页示例9-5。

9.6.1 PL/I 程序结构

变量
保持赋予它的数值，直到一个新值被赋予。

PL/I 是一种块结构的语言，包括了包、过程、语句、表达式以及内置函数构成，见图9-4。

PL/I程序由块组成。一个”块”可以是子例程，也可以仅仅是一组语句。PL/I块允许

您创建高模块化的应用程序，这是因为块可以包含定义变量名字和存储类的声明语句。于是，您可以将变量的作用范围限制在一个单独的块或者一组块中，或者您也可使其在整个编译单元或装入模块中都生效。

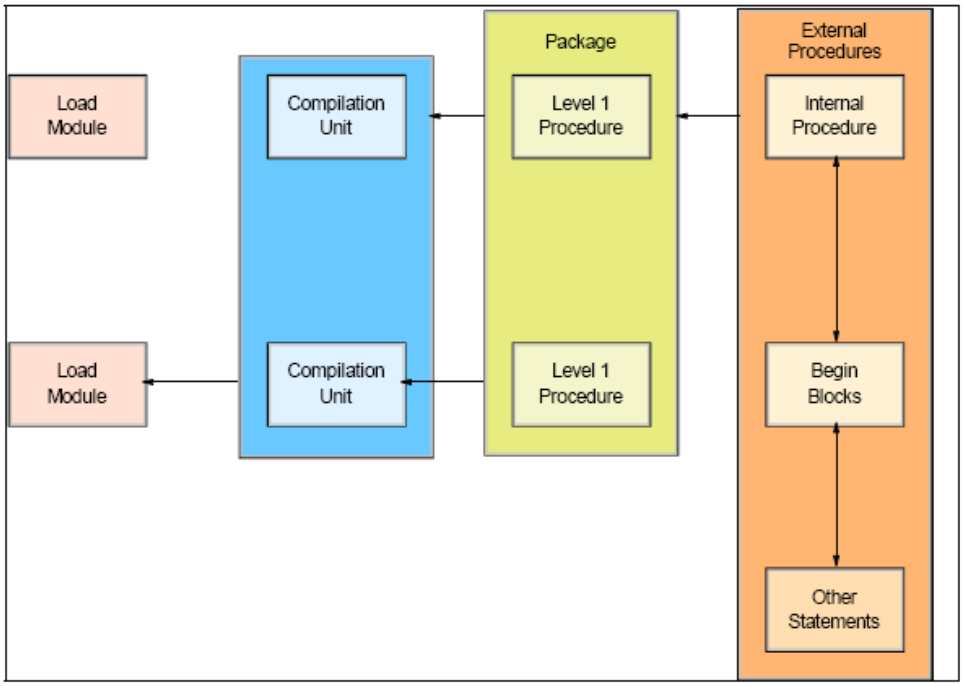


图9-4 PL/I应用程序结构

一个PL/I应用程序包括了一个或多个独立分离的可载入实体，也就是通常所说的"装入模块"。每一个装入模块包含了一个或者多个编译实体，也叫做"编译单元"。除非另外说明，这里"程序"指的是一个PL/I应用程序或者一个编译单元。

一个PL/I块可以是一个"过程"或者一个开始块(begin block)，它们都同样包含了零个或多个语句以及/或者零个或多个块。一个过程是通过一个过程(procedure)语句以及对应的结束语句所划分出的一组语句序列。如示例9-6。一个过程可以是主过程、子例程或者函数。

示例9-6 一个过程块

```
A: procedure;
      statement-1
      statement-2


     .

     .
      statement-n
      end Name;
```

一个开始块是由一个开始(begin)语句和对应的结束语句所划分的一组语句序列，如示例9-7。一个程序在主过程终止时刻终止执行。

示例9-7 开始块

```
B:  begin;
     statement-1
     statement-2

     .

     .

     statement-n
     end B;
```

9.6.2 预处理器

预处理器
在输入被主程序处理之前，执行一些预备操作的软件。

PL/I编译器允许选择一个或多个在程序中所需要使用到的集成预处理器。您可以选择包含预处理器、宏预处理器、SQL预处理器或者CICS预处理器——您同样可以指定这些预处理器被调用的顺序。

每一个预处理器都支持一定数目的选项，以便根据您的需要裁剪定制具体的处理。

- 包含预处理器

  除去使用PL/I的指示符%INCLUDE(%INCLUDE指示符用于将外部文本引用到源程序中)以外，允许您使用包含(include)指示符将外部的源文件引入进程序协同使用。

- 宏预处理器

  宏可使您以隐藏具体实现细节以及操作数据，而仅仅向外暴露操作的方式编写公共使用的PL/I代码。与通用的子例程相对照，宏允许仅产生每个单独应用所需的代码。

- SQL预处理器

  通常来说，无论您是否需要访问DB2数据库，PL/I的编程都是一样的。但是，当需要查找、更新、插入或者删除DB2数据或者使用其他的DB2服务时，您必须使用SQL语句。您可以在PL/I应用程序中使用动态的或者静态的EXEC SQL语句。

  与DB2 交互通讯，您需要做完成以下工作：

  - 编写任何您所需要的SQL语句，并用EXEC SQL标识定界
  - 使用DB2预编译器或者编译时使用PL/I PP(SQL())编译器选项

  在您能充分使用EXEC SQL支持之前，您必须获取访问DB2系统的权限。

注意PL/I SQL预处理器目前不支持DBCS。

- CICS预处理器

  您可以在以交易方式运行于CICS下的PL/I应用程序中使用EXEC CICS语句。

相关阅读：更多有关在z/OS上使用PL/I的信息，可参阅IBM出版物Enterprise PL/I for z/OS V3R3 Language Reference，SC27-1460和Enterprise PL/I for z/OS V3R3 Programming Guide，SC27-1457。以上书籍可在以下网址获得：

http://www.ibm.com/servers/eserver/zseries/zos/bkserv/find_shelves.html

9.6.3 使用 SAX 解析器

PL/I编译器提供了一个叫做PLISAXx(x=A或B)的接口，以提供给您最基本的XML功能。这项支持包括高速XML解析器，该解析器可使程序接受输入的XML信息、检查它们是否符合格式要求，并将他们的内容转换成PL/I的数据结构。

该XML支持并不提供生成XML的功能，这个功能只能通过PL/I程序逻辑代为实现。XML支持并没有特殊的环境需求。它可以在所有的主流的运行时环境下执行，包括CICS、IMS以及MQ系列，另外还有z/OS批处理环境以及TSO。

9.7 在 z/OS 上使用 C/C++

C是一种编程语言，为各种各样的编程目的而设计。包括：

- 系统级代码
- 文本处理
- 图形学

C语言包括了一个简明的语句集，以及通过库所加入的一些功能函数。它使C语言灵活高效。它的另一个优点就是在不同的系统之间，该语言具有很高的一致性。

编译一个C源程序然后链接编辑目标卡片叠生成装入模块的流程和COBOL基本一致。参阅第283页的示例9-2，第327页的10.3.7节”如何使用链接编辑器”，以及287页的图9-3来观察这个过程。C语言中JCL与程序文件之间的关系和PL/I语言一样，同COBOL以及其他的高级语言也一致。参阅第287页的图9-3以及第289页的示例9-5。

相关阅读：更多有关在z/OS上使用C和C++的信息，可参阅IBM出版物C/C++ Language Reference，SC09-4764和C/C++Programming Guide，SC09-4765。以上书籍可在以下网址获得：

http://www.ibm.com/servers/eserver/zseries/zos/bkserv/find_shelves.html

9.8 在 z/OS 上使用 Java

Java是由Sun™ Microsystems™公司开发的一种面向对象的编程语言。Java也可以用于开发传统的主机商用应用程序以及使用标准接口的Internet和Intranet应用程序。

Java是一种日益流行的编程语言，用于多种操作系统上的应用开发。IBM是Java主要的支持者和使用者，横跨了包括z/OS的所有IBM计算平台。z/OS Java产品提供了同所有其他IBM平台一样的全功能Java API。此外，z/OS Java特许程序已经得到增强，允许Java访问z/OS独特的文件系统。在z/OS上诸如Enterprise COBOL和Enterprise PL/I这样的编程语言都向用Java编写的程序提供了接口。这些语言提供了一套接口或者工具，用于同Java程序交互，正如我们前面解释的那样(有关COBOL的内容见第288页9.4.4节"与Java方法交互通讯"，关于PL/I的内容见第293页9.6.3节"使用SAX解析器")。

294

各种各样的针对z/OS的Java软件开发工具包(SDK)特许程序帮助应用程序开发者在z/OS上使用Java API，在多个平台上编写或者运行应用程序，或者使用Java访问位于主机上的数据。其中的一些产品只允许Java应用程序运行在31位地址空间环境下。然而，通过z/OS的64位SDK，以前只能受存储限制在31位地址空间运行的纯Java应用程序，同样可以在64位的环境下执行。同时，一些主机也支持一种运行Java应用程序的专用处理器，称为zSeries应用程序辅助处理器(zAAP)。程序可以交互式地运行在z/OS UNIX下，或者运行在批处理方式下。

9.8.1 z/OS 的 IBM SDK 产品

和基于IBM其他平台的Java SDK一样，z/OS Java SDK特许程序也提供了业界标准API。z/OS SDK产品彼此相互独立，可以单独的订购和使用。

在本书的成书之时，z/OS的Java SDK有如下几种：

- Java SDK 1.3.1产品称之为IBM Developer Kit for OS/390，Java 2 Technology Edition，工作在z/OS和老版本的OS/390下。它是31位产品，很多的z/OS用户已经将它们的Java应用程序转移(或者移植)到Java的最新版本上。
- IBM SDK for z/OS，Java 2 Technology Edition，Version 1.4是Sun Microsystems的Java 软件开发包(SDK)对z/OS平台的IBM的31位接口，并被认证为完全兼容的Java产品。IBM成功地执行了Sun Microsystems公司提供的Java认证包(Java Certification Kit，JCK)1.4。
- IBM SDK for z/OS，Java 2 Technology Edition，Version 1.4运行于z/OS V1R4及其以后的版本，或者z/OS.e Version 1Release 4及其以后的版本上。它提供了同任何其他服务器平台等价的Java执行环境。
- IBM 64-bit SDK for z/OS，Java 2 Technology Edition，Version 1.4可使Java应用程序运行在64位环境中。它运行于z/OS Version 1Release 6及其以后的版本。和31位产品一样，该产品准许使用Java SDK 1.4的API。

更多有关z/OS的IBM Java SDK产品的信息，可查阅如下网站：

http://www.ibm.com/servers/eserver/zseries/software/java/

295

9.8.2 使用 Java 本地接口(JNI)

Java本地接口(JNI)是本地编程语言的Java接口，它是Java开发工具包的一部分。如果标准Java API没有提供您所需要的功能，则可通过JNI允许运行在Java虚拟机(JVM)上的Java代码处理其他语言(例如PL/I)编写的应用程序及库。此外调用API还允许将Java虚拟机嵌入到您的本地PL/I应用程序之中。

Java是一种相当完善的编程语言；然而存在这样一些情况：您需要调用其他编程语言所编写的程序。您可以通过从Java中的一个方法来调用本地语言来实现，也就是所谓”本地方法”。通过JNI编程可以使您使用本地方法实现很多不同的操作。一个本地方法能够：

- 和Java方法一样使用Java对象
- 创建Java对象，包括数组和字符串，然后检查并使用这些对象完成相应的任务。
- 检查并使用由Java应用程序所创建的对象
- 修改更新本地方法创建的或者传递过来的Java对象；Java应用程序可以随后使用这些修改更新过的对象。

最后，本地方法也可以容易地调用已经存在的Java方法，利用已经包含在Java编程框架中的功能。这样，一个应用程序的本地语言端和Java端都可以创建、修改并访问Java对象，并彼此共享使用这些对象。

9.9 在 z/OS 上使用 CLIST 语言

CLIST语言是一种解释型语言。和其他高级解释型语言编写的程序类似，CLIST程序也易于编写和测试。您不必编译或者链接编辑它们。要测试一个CLIST程序，您只需要简单地执行它并更正任何可能产生的错误，直至程序运行不再产生错误。

CLIST和REXX语言是TSO/E上可用的两种命令语言。CLIST语言能使您更加高效地使用TSO/E工作。

术语CLIST(读作”see list”)代表命令序列；之所以这样叫是因为大多数基本CLIST程序都是TSO/E命令序列。当您调用一个这样的CLIST程序时，它按顺序执行TSO/E命令。

296

CLIST编程语言用于：

- 执行日常的例行任务(例如输入TSO/E命令)
- 调用其他的CLIST程序
- 调用其他语言编写的应用程序

- ISPF应用程序(例如显示面板以及控制应用程序流)

9.9.1 CLIST 的类型

一个CLIST程序可以执行范围广泛的各种任务，但是主要归类为如下三类：

- 执行日常例行任务的CLIST
- 作为结构化应用程序的CLIST
- 管理其他语言编写的应用程序的CLIST

以上将在本节予以分别描述。

执行常用例行任务的CLIST

作为TSO/E的使用者，您可能会定期执行某些任务。这些任务可能包括输入TSO/E命令检查数据集状态、为特定的程序分配数据集或者打印文件。

您可以编写相应的CLIST程序，它们能显著减少您在这些日常的例行任务上所花费的时间。通过将执行任务所需要的所有指令集合到一个CLIST程序中，您可以减少花费时间、键盘敲击次数以及任务执行中的错误，最终提升您的工作效率。一个CLIST程序可以由单纯的TSO/E命令组成，或者由TSO/E命令与CLIST语句所混合组成。

作为结构化应用程序的CLIST

CLIST语言包括了编写一个完整的、结构化应用程序的基本工具。任何CLIST都可以调用另外的CLIST程序，这被称为"嵌套的"CLIST程序。CLIST同样包含了独立的子程序，称为"子过程"。嵌套CLIST程序和子过程使您能够将CLIST程序划分成逻辑单元，并将公共功能在某处实现。专门的CLIST语句可使您能够：

- 为子过程或者嵌套CLIST程序定义公共数据。
- 将数据限制在特定子过程或者CLIST程序中。
- 将特定的数据传给子过程或者嵌套CLIST程序。

对于交互式应用程序，CLIST可以发出ISPF命令显示全屏面板。反之，ISPF面板也可以根据用户在面板上的输入，来调用CLIST程序。

管理其他语言编写的应用程序的CLIST

假设您需要访问用其他语言编写的应用程序，但这些应用程序的相关接口可能不易使用或记忆。比重新编写一个新的应用程序更好的方法，是编写CLIST以提供给用户与这样的应用程序之间易于使用的接口。

一个CLIST程序可从终端上收发消息以确定用户需要做什么。基于此，CLIST程序可以建立环境，发出需要的命令来调用程序，执行所请求的任务。

9.9.2 执行 CLIST 程序

要执行一个CLIST程序，可使用EXEC命令。在ISPF的命令行中，命令开头需要键入”TSO”。在TSO/E EDIT或者TEST模式下，在执行EXEC操作时使用EXEC子命令。(执行在EDIT或者TEST模式下的CLIST程序只能够请求执行对应的EDIT或者TEST的子命令和CLIST语句，但您可以使用END子命令以结束EDIT或者TEST模式，使CLIST程序能够请求执行TSO/E命令)。

9.9.3 CLIST 语言的其他应用

除了请求执行TSO/E命令，CLIST程序也可以执行更为复杂的编程任务。CLIST语言包括了您在开发大的结构化应用程序时所需用到的编程工具。CLIST程序可以执行任何复杂度的任务，从显示一系列全屏面板到管理其他语言所编写的应用程序。

CLIST语言特色包括：

- 处理数值数据所需的广泛的算术与逻辑操作符集
- 用于处理字符数据的字符串处理功能
- CLIST语句可用于构建程序、执行I/O操作、定义和修改变量，进行错误处理以及警示中断

9.10 在 z/OS 上使用 REXX

重构的可扩充执行器语言(REXX)是一种结构化语言，它使程序和算法能够以清晰和结构化的方式书写。它是一种解释型和编译型语言。这种解释型语言和其他诸如COBOL语言的不同之处在于，在执行它之前不需要编译REXX命令列。然而，您也可以选择在执行之前编译REXX命令列，以减少它的执行时间。

REXX编程语言的典型应用如下：

- 执行日常例行任务，例如输入TSO/E命令
- 调用其他REXX程序
- 调用其他语言编写的应用程序
- ISPF应用程序(显示面板以及控制应用程序流)
- 对问题的一次性快速解决方案
- 系统编程
- 在一切我们能够使用其他HLL编译语言的地方

REXX也可以用在Java环境中，例如REXX的一个分支叫做NetRexx™能够与Java完全无缝的协同工作。NetREXX程序可以直接使用任何Java类，并可以用来编写任何Java类。它将Java的安全性和性能特点带给了REXX程序，并将REXX的算术运算和简单性的特点带给了Java。于是，作为单一语言的NetREXX，可以同时用

于脚本开发和应用程序开发。

REXX程序结构比较简单。它提供了一个控制结构的常规选择，例如包括IF…THEN…ELSE用于简单的条件处理，
SELECT…WHEN…OTHERWISE…END用于多个可选操作的选择，以及多种不同的DO...END形式用于成组以及重复循环。GOTO指令没有被包含在其中，但是提供了SIGNAL指令用于诸如错误退出时的异常控制转移以及分支转移。

REXX中有关JCL与程序文件之间的关系同COBOL和其他的高级语言相同。参阅第287页图9-3以及第289页示例9-5。

299

9.10.1 编译和执行 REXX 命令列

一个在z/OS之下编译过的REXX程序可以运行在z/VM上。类似的，一个REXX程序在z/VM下编译后同样可以运行在z/OS下。一个在z/OS或者z/VM下编译的REXX程序也能运行在安装了REXX/VSE的z/VSE上。

编译一个REXX源程序然后链接编辑目标卡片叠生成装入模块的流程和COBOL基本一致。参阅第283页的示例9-2，327页的10.3.7节”如何使用链接编辑器”以及287页图9-3。

当使用编译器编译REXX语言时，涉及到三个主要组件：

- IBM Compiler for REXX on zSeries。该编译器将REXX源程序翻译成已编译程序
- IBM Library for REXX on zSeries。该库包含了已编译程序在运行时刻会调用的例程服务。
- 备选库。备选库包含了语言处理器，能将已编译程序转换并通过解释器执行。备选库可以被没有IBM Library for REXX on zSeries的z/OS和z/VM用户使用，以运行已编译程序。

编译器以及运行库运行在具有TSO/E的z/OS系统上，或者运行在z/VM系统的CMS下。IBM Library for REXX in REXX/VSE运行于z/VSE上。

编译器可以产生如下形式的输出：

- 已编译的EXEC

  和解释型的REXX程序十分类似，同样使用系统的EXEC处理器，并且搜索顺序也一样。将解释型程序替换成编译型程序最简单的方法就是产生已编译的EXEC。用户并不需要了解他们所使用的REXX程序是已编译的EXEC还是解释型程序。已编译的EXEC可发送到z/VSE上运行。

- z/OS下的目标卡片叠或z/VM下的TEXT文件
- 一个TEXT文件是一个其外部引用还未确定的目标代码文件(这个术语仅仅用于z/VM之上)。它们在使用之前必须被转换成可执行形式(装入模块)。装入模块以及MODULE文件的调用方式同其他编译器产生的可执行模块的调用方式相同，并且使用相同的搜索顺序。然而，该搜索顺序和解释型REXX程序以及

编译后的EXEC的搜索顺序并不相同。这些装入模块可以用作命令，也可以用作REXX功能包的一部分使用。目标卡片叠或者MODULE文件可以发送到z/VSE上用于构建可执行程序段(phase)。

- IEXEC输出

  该输出包含了被编译的REXX程序的扩展源程序。"扩展"的意思是指通过在编译时刻，将%INCLUDE指示符包含在IEXEC输出中，使其包含了主程序以及其他的所有部分。只有在指定边界以内的文本才能包含在IEXEC输出之中。但需要注意的是，MARGINS的默认设置包含了输入记录中的所有文本内容。

相关阅读：更多有关使用REXX的信息，可参阅以下出版物：

- The REXX Language, 2nd Ed., Cowlishaw, ZB35-5100
- Procedures Language Reference (Level 1), C26-4358 SAA® CPI
- REXX on zSeries V1R4.0 User's Guide and Reference, SH19-8160
- Creating Java Applications Using NetRexx , SG24-2216

同样可以访问以下网站：

http://www.ibm.com/software/awdtools/REXX/language/REXXlinks.html

9.11 编译型语言对比解释型语言

在设计一个应用程序时，您可能需要决定使用编译型语言还是解释型语言来编写应用程序源代码。这两种类型的语言都有它们各自的优缺点。通常确定使用解释型语言是基于开发时间上的限制或者方便未来程序的修改来考虑的。在确定使用解释型语言的同时也付出了代价，即以更高的执行开销换取了开发速度。原因在于每次执行时，解释型语言的每一行在都要被重新翻译解释，这是一个很大的开销。由此，解释型语言通常更加适合于即兴需求，然不是那些预定需求。

9.11.1 编译型语言的优点

汇编语言、COBOL、PL/I 、C/C++都是通过编译器翻译源代码，其产生的高效代码可以被任意多次执行。翻译所花费的开销仅此一次，即在源代码被编译时；此后，仅仅只需要载入执行即可。

与之对照的，解释型语言必须在每次程序运行时刻被解析、解释并执行，这极大的增加了运行程序的开销。基于这个原因，解释型语言通常在效率上不如编译型语言。

有一些编程语言，例如REXX和Java，即是编译型的也是解释型的。

9.11.2 解释型语言的优点

在"编译型语言的优点"中我们讨论了使用编译型语言的原因。"在z/OS中使用CLIST语言"以及"在z/OS中使用REXX语言"中我们讨论了解释型语言的优点。并不能简单的说哪一种语言"更好"——它取决于应用程序本身。甚至在很多应用程序中，我们最后使用到了很多种语言。例如，像CLIST这样的语言所具备的一个优势就是易于编码、测试以及更改。然而，它的效率并不高，我们用机器资源交换了程序员的时间。

记住这一点，我们可以看到在应用程序的高强度部分(很大的资源负载)下，我们使用编译型语言，然而在接口(调用应用程序)以及低强度的部分我们可以选择使用解释型语言来编写。解释型语言同样也适合即兴需求或者是应用程序的原型开发。

设计师的职责之一就是权衡每种语言的利弊并决定应用程序的哪一个部分最适用哪种特定语言。

9.12 什么是 z/OS 的语言环境

正如我们在第257页的第8章"设计和开发z/OS上的应用程序"中提到的，一个应用程序是由一个或者多个协同工作的程序组成的集合，以完成某些特定目标，诸如库存控制或者薪水册管理。应用程序开发的目标包括模块化、共享代码以及在基于工作站的前端开发应用程序。

在z/OS上，语言环境产品为所有相容的高级语言(HLL)产品提供了一个公共的运行环境。这里高级语言(HLL)是指在汇编语言的层次以上并在程序产生器以及查询语言所在层次之下的编程语言。z/OS语言环境为z/OS的应用程序员设置了一个公共的语言开发和运行环境。而相关功能之前已经由各自的语言产品提供，使语言环境免去了维护各个单独语言库的需要。

在过去，编程语言对彼此的相互调用以及在多个不同操作系统上表现一致有很大的局限性。而这个特质也限制了那些希望在同一应用程序中使用多种编程语言的
302 程序的开发。

各种编程语言对实现数据结构、条件控制，与系统服务以及库函数的交互都有各自不同的规则。

通过语言环境以及它所提供的语言相互调用的功能，z/OS应用程序员可以充分使用各种语言的不同功能和特色。

9.12.1 语言环境如何使用

语言环境为所有参与的高级语言设置了一个公共的运行时环境。它集成了最基本的运行时服务，例如运行时消息处理、状态处理以及存储管理等子例程。这些服

务提供了一套对各编程语言都保持一致的接口。应用程序既可以直接调用这些接口，也可以通过语言特定服务来调用这些接口。

通过语言环境，您可以为您的应用程序使用一个运行环境，而不用考虑应用程序的编程语言或者系统资源的需要。

图9-5 展示了语言环境的相关组件，包括：

- 支持启动和停止程序、分配存储、与不同语言编写的程序交互通讯以及标示及处理状态的基本子例程。
- 公共库服务，例如数学运算或者日期与时间服务，系统上运行的程序通常需要这些服务。这些功能通过一个可调用服务库支持。
- 运行库的语言特定部分。

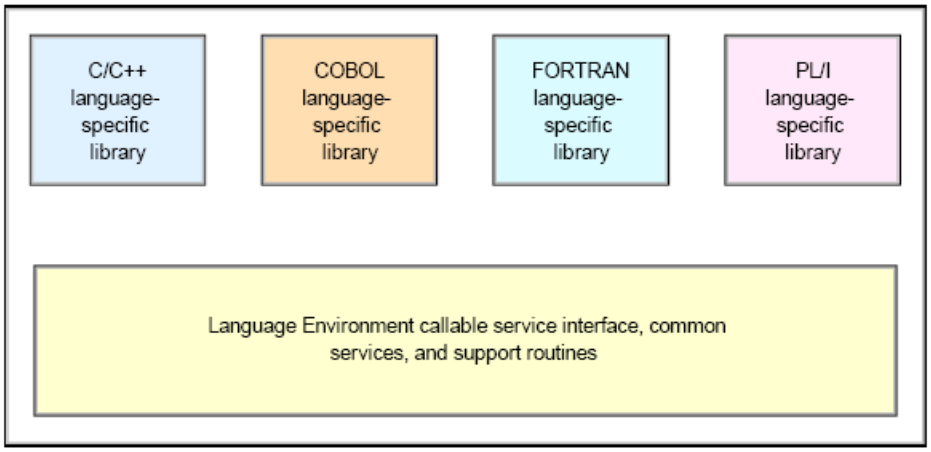


图9-5 z/OS语言环境组件

语言环境是如下的IBM编译器产品所产生的应用程序的必备运行环境：

- z/OS C/C++
- C/C++ Compiler for z/OS
- AD/Cycle® C/370™ Compiler
- VisualAge® for Java, Enterprise Edition for OS/390
- Enterprise COBOL for z/OS and OS/390
- COBOL for z/OS
- Enterprise PL/I for z/OS and OS/390
- PL/I for MVS and VM (以前的AD/Cycle PL/I for MVS and VM)
- VS FORTRAN and FORTRAN IV (在兼容模式下)

在很多情况下，您可以运行以上编译器的先前版本所产生的编译代码。另外，系统还提供了一套汇编宏可以使汇编例程运行在语言环境之下。

9.12.2 语言环境近观

语言环境的语言特定部分提供了语言接口和特定服务，这些服务为每种单独的语言提供支持，并可通过公共调用接口进行调用。在本节，我们将更加详细地讨论

其中的一些接口和服务。

304 图9-6展示了通过语言环境所设定的一个公共运行时环境。

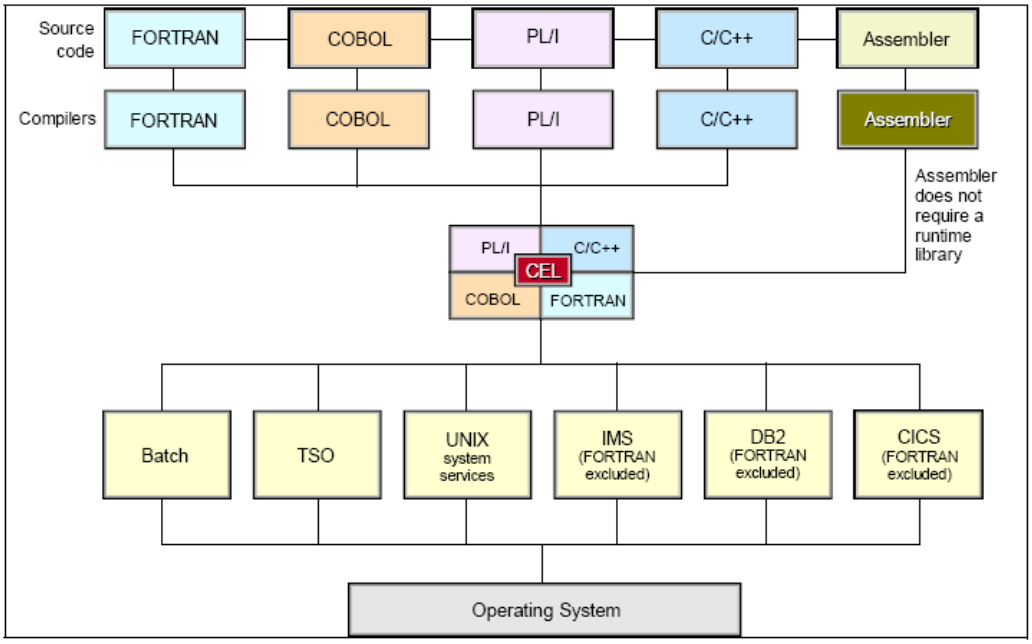


图9-6 语言环境的公共运行环境

语言环境的架构由如下组件的模型构成：

- 程序管理
- 状态处理
- 消息服务
- 存储管理

程序管理模型

语言环境的程序管理模型提供了一个应用程序可以在其中运行的框架。它是组成语言环境的所有组件模型(状态处理、运行时消息服务以及存储管理)的基础。

程序管理模型定义了混合语言应用程序中编程语言的语义有效性，并且集成了事务处理和多线程。

一些用于描述程序管理模型的术语是通用的编程术语；另一些术语的描述却不同于其他语言。您有必要理解这些术语在语言环境语境下的所指的含义，并可将其与其他语境下对比。 305

程序管理

程序管理定义了一个应用程序的程序运行结构，以及和该结构的各种管理组件集合体相关的语义。

三个实体，进程、领域以及线程，是语言环境的程序管理模型的核心所在。

进程

语言环境程序模型的最高级别组件就是进程。一个进程包含了至少一个程序集，并且逻辑上独立于其他的进程。语言环境通常不允许语言文件在程序集之间共享，也不提供访问外部存储数据集合的功能。

程序集

程序管理模型中最为关键的特色就是程序集，是一组构成应用程序的例程的集合。程序集等价于如下的任何之一：

- COBOL中的运行单元
- C/C++中包含了C主函数及其子函数的程序
- PL/I中的主过程及其所有的子例程
- Fortran中的程序及其子例程

在语言环境中，环境通常指得是高级语言在程序集级别的运行时环境。程序集包含了一个主例程以及零个或者多个子例程。主例程在程序集中被最先执行；所有随后的例程都命名为子例程。

线程

每一个程序集至少包括一个线程，即特定例程的一个基本实例。一个线程在程序集初始化的时候创建，并有自己的运行时栈，该栈用于跟踪线程的运行，除了栈，线程还有一个专属的指令计数器、多个寄存器以及状态处理机制。每一个线程代表一个在程序集的资源下运行例程的独立实例。

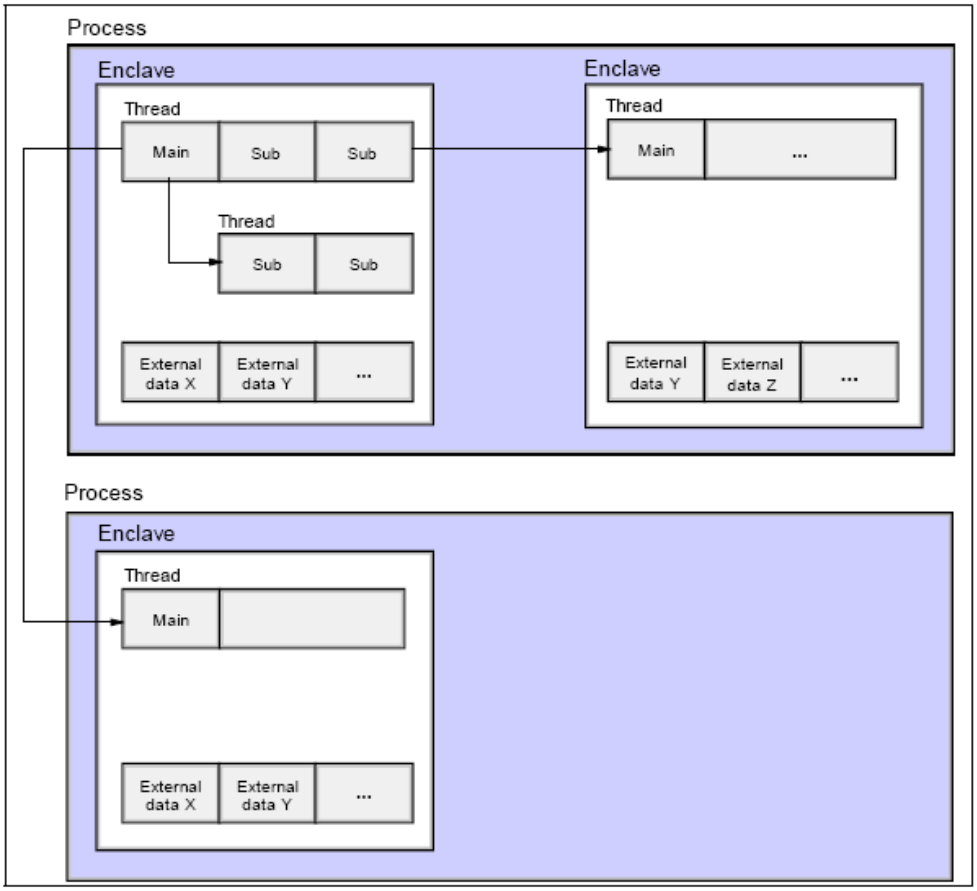


图9-7 完整的语言环境程序模型

图9-7展示了完整的语言环境程序模型，包括多个进程、多个程序集以及多个线程。如图所示，每一个进程都处于自有的地址空间中。一个程序集包含了一个主例程以及若干子例程。

线程可以创建多个程序集，同时程序集亦可以创建更多线程，如此类推。

状态处理模型

对单一语言应用程序和混合语言应用程序，语言环境运行库都提供了具有一致性和预测性的状态处理工具。它并没有取代当前高级语言的状态处理，而是使每种语言都能够响应本身的单一环境和混合语言环境。

语言环境状态管理使您能灵活地直接通过使用可调用服务响应标志状态和这些状态的询问信息，并提供了针对错误诊断、报告以及恢复的相关功能。

消息处理模型以及国家语言支持

语言环境提供了一套能够创建、发送运行时报告信息以及诊断信息的公共消息处理服务。

通过消息处理服务，您可以使用从可调用服务或其他标志状态返回的状态令牌，

并将其转换成消息格式，发送到预定义的输出设备或者缓存区域中。

国家语言支持的可调用服务使得您可以设置国家语言，以影响错误信息以及日子、星期、月份名称的显示语言。您也可以更改国家设置，这将会涉及到默认数据格式、时间格式、货币符号、小数分隔字符以及千位分隔符等。

存储管理模型

公共存储管理服务提供给所有的语言环境兼容的编程语言；语言环境控制运行时刻栈和堆存储的使用。它允许单一语言和混合语言应用程序访问一套集中的存储管理工具，并把多堆存储模型提供给那些目前还没有支持该模型的语言。公共存储模型省去了为每种语言都维护一个单独存储管理器的需要，并避免了不同存储机制所造成的不兼容。

9.12.3 通过语言环境运行程序

在您的程序完成编译之后，您可以：

- 链接编辑并运行一个已存在的目标卡片叠并接受默认的语言环境运行时选项设置
- 链接编辑并运行一个已存在的目标卡片叠并指定新的语言环境运行时选项设置
- 调用一个语言环境服务

接受默认运行时选项设置

为了在批处理方式下运行一个已存在的目标卡片叠并接受所有的默认语言环境运行时选项设置，您可以使用语言环境提供的负责链接编辑及运行的编目过程，
308 CEEWLG(编目过程在第212页6.7节"JCL过程(PROC)"中讨论过)。CEEWLG过程指定了链接编辑和运行您的目标卡片叠所需要的语言环境库。

运行库服务

语言环境库位于数据集中，数据集的高级限定词(HLQ)由安装时指定。例如，SCEERUN包含了在执行C/C++、PL/I、COBOL以及FORTRAN等语言编写的应用程序时所需要的运行库例程。SCEERUN2包含了执行C/C++和COBOL编写的应用程序时所需要的运行库例程。

对于需要语言环境所提供运行库的应用程序来说，它们可以通过使用以下两个方法或者其中之一来访问SCEERUN和SCEERUN2数据集。

- LNKLST
- STEPLIB

在语言环境下链接编辑和运行应用程序之前，有一些必须考虑到的问题。

重要：语言环境库例程被划分为两类：常驻例程和动态例程。常驻例程和应用程序相关联，包含了诸如初始例程、终止例程以及可调用服务的指针等内容。动态例程并不是应用程序的一部分，仅在运行时刻动态载入。

语言环境可调用服务

COBOL应用开发者会发现语言环境的一致性状态处理服务特别有用。而对所有的语言而言，公共的数学运算服务以及日期和时间服务也是同样非常的有用。

语言环境可调用服务分类如下：

- 通讯状态服务
- 状态处理服务
- 日期和时间服务
- 动态存储服务
- 通用可调用服务
- 初始与终止服务
- 本地可调用服务
- 数学运算服务
- 消息处理服务
- 国家语言支持服务

相关阅读：关于可调用服务的有关信息，在以下IBM 出版物中有完全的描述：z/OS Language Environment Programming Reference，SA22-7562。

语言环境调用约定

语言环境的服务可被高级语言库例程、其他语言环境服务以及用户自定义的高级语言请求所调用。在很多情况下，高级语言库例程调用这些服务完成用户指定的功能。如下即为从我们本章所描述的三种语言中调用数学运算服务的例子。同样可以参阅298页9.9.3节"CLIST语言的其他应用"所引用的例子。

示例9-8展示了一个COBOL程序如何调用数学运算可调用服务CEESDLG1计算10为底的对数。

示例9-8 从COBOL中调用数学运算可调用服务的示例

```
77   ARG1RL  COMP-2.
77   FBCODE  PIC X(12).
77   RESLTRL COMP-2.
     CALL "CEESDLG1" USING ARG1RL , FBCODE ,
     RESLTRL.
```

9.13 总结

本章概述了在设计和开发一个运行在z/OS的应用程序所时需确定的事宜。选择适

用的编程语言是应用程序设计阶段最为重要的步骤之一。应用程序设计师必须充分了解每种语言各自的优缺点，然后根据特定的应用程序需求作出最佳决定。

选择语言的一个关键因素就是确定在给定的系统安装中那种语言用得最多。如果在一个系统安装中绝大多数应用程序都使用COBOL，那么它也很可能是在本系统安装中新的应用系统所选用的开发语言。

需要指出当主要的编程语言已经选择好，并不意味着该应用系统的所有程序只能局限于此语言。很可能存在使用多种语言的情况，这样我们可以充分利用某种特定语言的优势专门应对应用系统某部分的开发。这里，可能会将经常被调用的子例程可以用汇编语言编写，以确保应用系统尽可能的高效，而应用系统的其他部分是用COBOL或者其他的高级语言编写。

310

很多z/OS使用站点维护了一个被很多业务所共享的子例程的库。这个库可能包括了诸如日期转换等例程。只要这些子例程编写使用了标准的连接约定，他们就可以被其他语言所调用，而不管这些子例程是由何种语言所编写而成。

每种语言都有其自身固有的优势，应用设计者需要充分使用这些优势。如果某给定的应用系统值得为采用多种语言编程而增加复杂性，应用设计者就应该充分利用每种语言的特色和功能。但是要始终牢记，在应用程序需要修改更新时，其他人必须也能使用这些语言进行编程。这是编程的一条主要规则。最初的程序员可能会长期离开，但应用系统却需要能够一直运行下去。

最后，设计的复杂性通常需要同维护工作的简易性进行相互权衡。

本章中重要术语		
汇编程序(assembler)	绑定器(binder)	编译器(complier)
调试(debugging)	动态链接库(dynamic link library)	代(generation)
输入/输出(I/O，input/output)	解释器(interpreter)	装入模块(load modules)
预处理器(preprocessor)	编程语言(programming language)	变量(variable)

9.14 复习题

为了帮助您检测对本章的理解程度，请完成下列问题：

1. 为什么一个程序可能需要用汇编语言编写？
2. 各公司是否在继续改进和加强COBOL和PL/I的编译器？
3. 为什么CLIST和REXX被称为解释型语言？
4. CLIST和REXX主要适用的领域有哪些？
5. 哪种解释型的语言也可以被编译？
6. 在z/OS应用程序开发中，语言环境的使用是强制性的吗？

7. 哪种数据文件组织形式更加适合联机应用程序？哪种数据文件组织形式更加适合批处理应用程序?

8. 什么是高级语言(HLL)？用高级语言编程相对汇编语言有何优点？

9. 假设程序PROG1使用如下的JCL运行：

```
//job JOB
//STEP010    EXEC PGM=PROG1
//STEPLIB    DD DSN=MY.PROGLIB,DISP=SHR
//INPUT1          DD DSN=A.B.C,DISP=SHR
//OUTPUT1   DD DSN=X.Y.Z,DISP=SHR
```

如果INPUT1 DD语句改为使用数据集A1.B1.C1，我们是否可以使用同一个程序来处理它？假设新数据集和原来的数据集具有相同的属性特征。

9.15 思考题

1. 如果考虑到性能，您应该使用编译型语言还是解释性语言？

2. 如果您需要开发一个交易系统，以下哪一个是您的最佳选择？

    a. 基于CICS的COBOL或者PL/I

    b. 基于CICS的C/C++

    c. 综合使用上述两者？

3. 您将使用何种语言开发一个用于保险业中计算保险费的应用系统？假设该应用系统会被很多其他应用系统调用。

4. 一个COBOL程序能够调用一个汇编程序吗？为何会有这种需要?

10

第 10 章 在 z/OS 上编译和链接编辑程序

目标：作为您所在公司最新的z/OS应用程序员，您可能需要创建一个能在z/OS上运行的新的应用程序。完成这项工作需要您了解如何编译、链接链接以及执行程序。

在本章结束之后，您应当能够：

- 解释说明编译器的作用
- 编译一个源程序
- 解释链接链接编译器和绑定器的不同
- 从已编译程序创建可执行代码
- 解释说明目标卡片叠与装入模块之间的区别
- 在z/OS上运行一个程序

313

10.1 源、目标以及装入模块

一个程序可划分为完成特定功能的逻辑单元。完成一个功能或者几个相关功能的一个代码逻辑单元称为"模块"。将单独的功能编写为单独的模块，这种处理方法称为"模块编程"。每个模块都可以使用最为适合完成其功能的符号语言来编写。

源模块
语言翻译器(编译器)的输入。

目标卡片叠
语言翻译器的输出。

每个模块都被一个语言翻译器所汇编或者编译。语言翻译器的输入为源模块；语言翻译器的输出称为目标卡片叠。在目标卡片叠能够被执行之前，它必须先被绑定器(或者链接编辑器)处理。绑定器的输出称为装入模块，如图10-1所示。

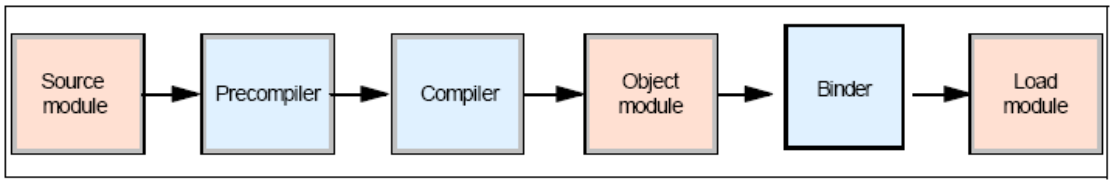


图10-1 源、目标以及装入模块

根据模块的状态不同——源模块、目标卡片叠或者装入模块——它可以存储于相关的库之中。一个库是在可直接访问存储上的分区数据集(PDS)或者扩展分区数据库(PDSE)，PDS和PDSE可以被划分成不同的分区，称为成员，在一个库中，每个成员都包含了一个完整的程序或者其中的一部分。

10.2 什么是源码库？

源程序(或者说源代码)是一组以计算机语言编写的语句集合，参见277页第9章所讨论的"在z/OS上使用编程语言"。对源程序来说，当它们被检查无错后，将被存放在分区数据集之中，也就是我们通常所说的"源码库"中。源码库包含了相关的源代码，这些源代码将来会被提交给编译处理或者被应用程序员取出重作修改。

Copybook
在一个共享的库中，程序员存放公用的程序片断。

*Copybook*是一个包含了预先编写文本的源码库，通常被用来在编译时刻将文本复制到源程序之中，作为避免相同的代码语句被一遍遍重复编写的一种捷径。它通常是一个共享库，在该库中程序员存放了公用的程序片断，这些程序片断将来会被包括在程序员的源代码之中。应当注意不要将其与程序或子例程相混淆。一个copybook成员只是一段文本；它可能并不是真正的编程语言的合法语句。

子例程是指一段通常被调用完成预定功能的例程。使用copybook成员以及子例程背后的目的本质上是相同的，即避免重复编写前面已经完成的代码。然而，子例程是一个小的程序(已被编译、链接编辑并且可执行)，它能被调用并能基于传递给它的信息返回对应的结果。而copybook成员仅仅是一段文本，它将来会被包含在源代码之中并最终与之一起成为可执行的程序。Copybook是一个COBOL中的术语名词，但这个概念却用于绝大多数的编程语言之中。

如果您在将要编译的源程序里使用copybook，您可以通过提供一个SYSLIB的DD

语句，或者为源程序COPY语句中指定的库提供一个DD语句，来将copybook从源库中包含进来。在示例10-1中，我们将DEPT88.BOBS.COBLIB库中的INPUTRCD成员的文本内容插入到即将编译的源程序中。

示例10-1 COBOL源代码中的copybook

```
//COBOL.SYSLIB  DD  DISP-SHR,DSN-DEPT88.BOBS.COBLIB
//SYSIN    DD  *
      IDENTIFICATION DIVISION.
     . . .
      COPY INPUTRCD
      . . .
```

这些库必须位于直接访问存储设备(DASD)之上。在您使用JCL或者在TSO之下进行编译时，这些库不能位于分层文件系统(HFS)之中。

10.3 在 z/OS 上编译程序

链接编辑器
将目标卡片叠转化为可执行的装入模块。

编译器的功能就是将源代码翻译转换成目标卡片叠，目标卡片叠在执行之前必须先被绑定器(或者链接编辑器)处理。在一个源模块的编译过程中，编译器为所有的指令、数据元素以及标签从零开始赋予相对地址。

地址的表示形式是基地址加相对位移，它使得程序可重定位，也就是说，程序不必在每次执行时都载入到内存的同一位置。(关于可重定位程序的更多信息，参阅第332页10.4”为可执行程序创建装入模块”。)所有对外部程序或者子例程的引用都暂时未被解析。这些引用将在目标卡片叠被链接时解析，或者在程序执行时刻被动态解析。

要在z/OS上编译程序，您可以使用批处理作业，或者在TSO/E下通过命令、CLIST或者ISPF面板进行编译。对于C程序，您可以在z/OS UNIX shell中使用c89命令进行编译。对于COBOL程序，您可以在z/OS UNIX shell下使用cob2命令进行编译。

对于通过批处理作业进行编译的方式，z/OS包含了一整套的编目过程，以帮助您省去一些需要自己编写JCL的工作。如果没有编目过程能够满足需求，您就需要自己编写编译过程所需要的所有JCL。

作为编译步骤的一部分，您需要定义编译过程所需要的数据集，并为您的程序和您预期的输出指定所需要的编译器选项。

包含源代码的数据集(库)在SYSIN DD语句处指定，参见示例10-2。

示例10-2 SYSIN DD语句指定源代码位置

```
//SYSIN   DD  DSNAME=dsname,
//            DISP=SHR
```

您也可以将源代码直接置于输入流中，这样需要使用如下的SYSIN DD语句：

```
//SYSIN    DD    *
```

当您使用DD *形式时，源代码必须跟在该语句之后。如果编译过程后有其他作业步，则那个作业步的EXEC语句跟在/*语句或者源代码最后一个语句之后。

10.3.1 什么是预编译器？

有些编译器拥有预编译器或者预处理器，用来处理那些非计算机编程语言的语句部分。如果您的源程序包含了EXEC CICS语句或者EXEC SQL语句，则该源程序必须首先通过预处理将这些语句转换成COBOL，PL/I或者汇编语言的语句，这具体取决于您使用何种语言编写程序。

10.3.2 使用编目过程进行编译

编目过程
放置在称为过程库的 PDS 中的一套作业控制语句。

316

在z/OS下编译程序最为简便的方法就是使用称为"编目过程"的批处理作业。一个编目过程是一组作业控制语句，它们位于被称之为"过程库(PROCLIB)"的分区数据集(PDS)之中。z/OS随带的过程库称为SYS1.PROCLIB。该系统库将在471页16.3.7节"SYS1.PROCLIB"中作更为全面的讨论。对于编目过程的使用，我们可以简单的将其想象为copybook，与之不同的是，编目过程中包含的是JCL语句而不是源程序代码。您不必自己编写JCL语句告诉系统到哪里寻找编目过程，因为它们就位于系统库中，而当执行的JCL中引用了相关过程时，系统库将被自动搜索得到。

用于编译的JCL中需要包含如下信息：

- 作业描述
- 调用编译器的执行语句
- 过程未提供但需要用到的数据集的相关定义

COBOL编译过程

在示例10-3所示的JCL中执行了IGYWC过程，它是一个用于编译源程序的单步过程。它将产生目标卡片叠并将其存储于SYSLIN数据集中，正如我们将在示例10-4中所看到的一样。

示例10-3 编译内嵌的COBOL源程序的基本JCL

```
//COMP     JOB
//COMPILE EXEC IGYWC
//SYSIN    DD   *
    IDENTIFICATION DIVISION (source program)
.
.
.
/*
//
```

SYSIN DD语句指明了源程序的位置。在该示例中，星号(*)表明代码位于本作业的输入流内。

对于PL/I程序，除了要替换源程序之外，编译的EXEC语句也应当替换为：

```
//complie   EXEC IBMZC
```

示例10-4中所示的内容组成了示例10-3中所用到的编目过程IGYWC。如前所述，编译处理的结果，已编译程序，将被置于SYSLIN DD语句所指定的数据集内。

示例10-4 过程IGYWC - COBOL编译

```
//IGYWC  PROC  LNGPRFX-'IGY.V3R2M0',SYSLBLK-3200
//*
//*  COMPILE A COBOL PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE
//*   SYSLBLK   3200
//*   LNGPRFX   IGY.V3R2M0
//*
//*  CALLER MUST SUPPLY //COBOL.SYSIN DD . . .
//*
//COBOL  EXEC PGM-IGYCRCTL,REGION-2048K
//STEPLIB  DD  DSNAME-&LNGPRFX..SIGYCOMP,
//             DISP-SHR
//SYSPRINT DD  SYSOUT-*
//SYSLIN   DD  DSNAME-&&LOADSET,UNIT-SYSDA,
//             DISP-(MOD,PASS),SPACE-(TRK,(3,3)),
//             DCB-(BLKSIZE-&SYSLBLK)
//SYSUT1   DD  UNIT-SYSDA,SPACE-(CYL,(1,1))
//SYSUT2   DD  UNIT-SYSDA,SPACE-(CYL,(1,1))
//SYSUT3   DD  UNIT-SYSDA,SPACE-(CYL,(1,1))
//SYSUT4   DD  UNIT-SYSDA,SPACE-(CYL,(1,1))
//SYSUT5   DD  UNIT-SYSDA,SPACE-(CYL,(1,1))
//SYSUT6   DD  UNIT-SYSDA,SPACE-(CYL,(1,1))
//SYSUT7   DD  UNIT-SYSDA,SPACE-(CYL,(1,1))
```

COBOL预处理器和编译及链接过程

第319页示例10-5所示的JCL执行了DFHEITVL的过程，它是一个由预处理COBOL源程序，编译从预处理步骤的输出，并将其链接到装载库中这三步组成的过程。第一步产生预处理后的源代码，放到临时数据集SYSPUNCH中，该步把所有的CICS调用展开到COBOL源码语句之中。第二步将这个临时数据集作为输入，并产生目标卡片叠，存于临时数据集SYSLIN中，参见319页示例10-6。第三步将SYSLIN临时数据集，以及所有其他可能需要包含的模块作为输入，创建一个装入模块并存于由SYSLMOD DD语句所引用的数据集里。

第319页示例10-5中，您会发现其JCL要比一个简单的编译作业(第317页示例10-3)复杂一些。一旦我们从单步作业转入到多步作业，在我们提供JCL覆写时必须要告诉系统我们所引用的究竟是哪一步。

参看319页示例10-6的JCL，我们可以看到第一个作业步(每个作业步都是一个EXEC语句，并且作业步名就是EXEC所在行的名称)名叫TRN，于是我们将SYSIN DD语句用TRN限制修饰，以确保动作作用于TRN作业步中。

类似的，第四步叫做LKED，于是我们必须用LKED来限制SYSIN DD语句，以确保相关内容应用在LKED作业步中。更多有关覆写编目过程的信息可参见第213页6.7.1节"JCL PROC语句覆写"。

示例10-5中JCL运行的最终结果(假设没有错误产生)将是预处理并编译我们内嵌的源程序，同时链接编辑目标卡片叠，然后将名为PROG1的装入模块存于MY.LOADLIB数据集之中。

示例10-5 用于预处理、编译及链接内嵌式COBOL源程序的基本JCL

```
//PPCOMLNK   JOB
//PPCL       EXEC DFHEITVL,PROGLIB='MY.LOADLIB'
//TRN.SYSIN    DD *
  IDENTIFICATION DIVISION (source program)
    EXEC CICS ...
...
    EXEC CICS ...
...
//LKED.SYSIN DD *
   NAME PROG1(R)
/*
```

示例10-6中的语句组成了在示例10-5中所用到的DFHEITVL编目过程。同其他的编译及链接过程一样，其预处理、编译并链接步骤的结果将产生装入模块，且装入模块将被存于由SYSLMOD DD语句所指定的数据集中。

示例10-6 DFHEITVL过程- COBOL的预处理、编译及链接

```
//DFHEITVL PROC SUFFIX=1$,           Suffix for translator module
//*
//* This procedure has been changed since CICS/ESA Version 3
//*
//* Parameter INDEX2 has been removed
//*
//       INDEX='CICSTS12.CICS', Qualifier(s) for CICS libraries
//       PROGLIB=&INDEX..SDFHLOAD,   Name of output library
//       DSCTLIB=&INDEX..SDFHCOB,    Name of private macro/DSECT lib
//       COMPHLQ='SYS1',             Qualifier(s) for COBOL compiler
//       OUTC=A,                     Class for print output
//       REG=2M,                     Region size for all steps
//       LNKPARM='LIST,XREF',        Link edit parameters
//       STUB='DFHEILIC',            Link edit INCLUDE for DFHECI
//       LIB='SDFHCOB',              Library
//       WORK=SYSDA                  Unit for work data sets
```

```
//*      This procedure contains 4 steps
//*      1.   Exec the COBOL translator
//*           (using the supplied suffix 1$)
//*      2.   Exec the vs COBOL II compiler
//*      3.   Reblock &LIB(&STUB) for use by the linkedit step
//*      4.   Linkedit the output into data set &PROGLIB
//*
//*      The following JCL should be used
//*      to execute this procedure
//*
//*      //APPLPROG EXEC DFHEITVL
//*      //TRN.SYSIN  DD *
//*         .
//*         . Application program
//*         .
//*      /*
//*      //LKED.SYSIN DD *
//*         NAME anyname(R)
//*      /*
//*
//* Where   anyname   is the name of your application program.
//*      (Refer to the system definition guide for full details,
//*      including what to do if your program contains calls to
//*      the common programming interface.)
//*
//TRN    EXEC PGM=DFHECP&SUFFIX,
//            PARM='COBOL2',
//            REGION=&REG
//STEPLIB  DD DSN=&INDEX..SDFHLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=&OUTC
//SYSPUNCH DD DSN=&&SYSCIN,
//            DISP=(,PASS),UNIT=&WORK,
//            DCB=BLKSIZE=400,
//            SPACE=(400,(400,100))
//*
//COB    EXEC PGM=IGYCRCTL,REGION=&REG,
//       PARM='NODYNAM,LIB,OBJECT,RENT,RES,APOST,MAP,XREF'
//STEPLIB  DD DSN=&COMPHLQ..COB2COMP,DISP=SHR
//SYSLIB   DD DSN=&DSCTLIB,DISP=SHR
//         DD DSN=&INDEX..SDFHCOB,DISP=SHR
//         DD DSN=&INDEX..SDFHMAC,DISP=SHR
//         DD DSN=&INDEX..SDFHSAMP,DISP=SHR
//SYSPRINT DD SYSOUT=&OUTC
//SYSIN    DD DSN=&&SYSCIN,DISP=(OLD,DELETE)
//SYSLIN   DD DSN=&&LOADSET,DISP=(MOD,PASS),
//            UNIT=&WORK,SPACE=(80,(250,100))
//SYSUT1   DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT2   DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT3   DD UNIT=&WORK,SPACE=(460,(350,100))
```

```
//SYSUT4   DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT5   DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT6   DD UNIT=&WORK,SPACE=(460,(350,100))
//SYSUT7   DD UNIT=&WORK,SPACE=(460,(350,100))
//*
//COPYLINK EXEC PGM=IEBGENER,COND=(7,LT,COB)
//SYSUT1   DD DSN=&INDEX..&LIB(&STUB),DISP=SHR
//SYSUT2   DD DSN=&&COPYLINK,DISP=(NEW,PASS),
//            DCB=(LRECL=80,BLKSIZE=400,RECFM=FB),
//            UNIT=&WORK,SPACE=(400,(20,20))
//SYSPRINT DD SYSOUT=&OUTC
//SYSIN    DD DUMMY
//*
//LKED   EXEC PGM=IEWL,REGION=&REG,
//            PARM='&LNKPARM',COND=(5,LT,COB)
//SYSLIB   DD DSN=&INDEX..SDFHLOAD,DISP=SHR
//         DD DSN=&COMPHLQ..COB2CICS,DISP=SHR
//         DD DSN=&COMPHLQ..COB2LIB,DISP=SHR
//SYSLMOD  DD DSN=&PROGLIB,DISP=SHR
//SYSUT1   DD UNIT=&WORK,DCB=BLKSIZE=1024,
//            SPACE=(1024,(200,20))
//SYSPRINT DD SYSOUT=&OUTC
//SYSLIN   DD DSN=&&COPYLINK,DISP=(OLD,DELETE)
//         DD DSN=&&LOADSET,DISP=(OLD,DELETE)
//         DD DDNAME=SYSIN
```

COBOL编译和链接过程

示例10-7的JCL执行了IGYWCL过程，该过程是一个编译源程序并将它链接到装载库的两步过程。第一个作业步产生了目标卡片叠并存于SYSLIN临时数据集中，如322页示例10-8所示。第二个作业步是将SYSLIN临时数据集作为输入，并联同所有其他可能需要被包含的模块一起，创建一个装入模块并存放于SYSLMOD DD语句所引用的数据集中。

示例10-7中JCL运行的最终结果(假设没有错误产生)是将我们的内嵌源程序编译、链接编辑目标卡片叠，并将名为PROG1的装入模块存放于MY.LOADLIB的数据集之中。

示例10-7　编译和链接一个内嵌COBOL源程序的基本JCL

```
//COMLNK   JOB
//CL       EXEC IGYWCL
//COBOL.SYSIN    DD *
  IDENTIFICATION DIVISION (source program)
 .
 .

.
/*
//LKED.SYSLMOD DD DSN=MY.LOADLIB(PROG1),DISP=OLD
```


第322页示例10-8所示的内容组成了在示例10-7中所用到的IGYWCL编目过程。同前面提到的一样，编译和链接步骤的结果是产生装入模块，并存放于SYSLMOD DD语句指定的数据集中。

示例10-8 IGYWCL过程- COBOL的编译和链接

```
//IGYWCL PROC  LNGPRFX='IGY.V2R1M0',SYSLBLK=3200,
//             LIBPRFX='CEE',
//             PGMLIB='&&GOSET',GOPGM=GO
//*
//*  COMPILE AND LINK EDIT A COBOL PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE
//*   LNGPRFX   IGY.V2R1M0
//*   SYSLBLK   3200
//*   LIBPRFX   CEE
//*   PGMLIB    &&GOSET          DATA SET NAME FOR LOAD MODULE
//*   GOPGM     GO               MEMBER NAME FOR LOAD MODULE
//*
//*  CALLER MUST SUPPLY //COBOL.SYSIN DD ...
//*
//COBOL  EXEC PGM=IGYCRCTL,REGION=2048K
//STEPLIB  DD  DSNAME=&LNGPRFX..SIGYCOMP,
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,UNIT=VIO,
//             DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
//             DCB=(BLKSIZE=&SYSLBLK)
//SYSUT1   DD  UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT2   DD  UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT3   DD  UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT4   DD  UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT5   DD  UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT6   DD  UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT7   DD  UNIT=VIO,SPACE=(CYL,(1,1))
//LKED   EXEC PGM=HEWL,COND=(8,LT,COBOL),REGION=1024K
//SYSLIB   DD  DSNAME=&LIBPRFX..SCEELKED,
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,DISP=(OLD,DELETE)
//         DD  DDNAME=SYSIN
//SYSLMOD  DD  DSNAME=&PGMLIB(&GOPGM),
//             SPACE=(TRK,(10,10,1)),

//             UNIT=VIO,DISP=(MOD,PASS)
//SYSUT1   DD  UNIT=VIO,SPACE=(TRK,(10,10))
```


COBOL编译、链接和执行过程

第323页示例10-9所执行的IGYWCLG过程，是一个编译源程序、将它链接入装载库并执行装入模块的三步过程。前两个作业步和编译及链接示例中的相同(第321页示例10-7)。然而不同的是，在第321页示例10-7中我们覆写了SYSLMOD DD语句以永久地存储装入模块，而在示例10-9中，我们并不需要将其先存储再执行。这也就是示例10-9中为什么我们把SYSLMOD DD语句的覆写加上了方括号，就是用以表明这条语句是可选的。

如果将这条语句写上，则装入模块PROG1将被永久保存到MY.LOADLIB中。反之如果没有写这条语句，装入模块将被存入一个临时数据集中并在GO作业步完成后被删除。

在示例10-9中，您会发现其中的JCL同简单编译作业(第317页示例10-3)所用的

JCL非常相似。观察示例10-10中的JCL可知，它与第322页示例10-8的JCL的不同之处仅仅在于我们多加了GO作业步。示例10-9中JCL运行的最终结果(假设没有错误产生)将是我们内嵌的源程序被编译、链接编译目标卡片叠生成并存储装入模块(至临时或永久数据集之中)，最后执行装入模块。

示例10-9 编译、链接及执行一个内嵌的COBOL源程序的基本JCL

```
//CLGO    JOB
//CLG     EXEC IGYWCLG
//COBOL.SYSIN    DD *
  IDENTIFICATION DIVISION (source program)
.
.
.
/*
[//LKED.SYSLMOD DD DSN=MY.LOADLIB(PROG1),DISP=OLD]
```

323

示例10-10中的内容组成了示例10-9中用到的IGYWCLG编目过程。

示例10-10　IGYWCLG过程-COBOL编译、链接及执行

```
//IGYWCLG PROC LNGPRFX='IGY.V2R1M0',SYSLBLK=3200,

//            LIBPRFX='CEE',GOPGM=GO
//*
//*  COMPILE, LINK EDIT AND RUN A COBOL PROGRAM
//*
//*  PARAMETER  DEFAULT VALUE    USAGE
//*   LNGPRFX   IGY.V2R1M0
//*   SYSLBLK   3200
//*   LIBPRFX   CEE
//*   GOPGM     GO
//*
//*  CALLER MUST SUPPLY //COBOL.SYSIN DD ...
//*
//COBOL  EXEC PGM=IGYCRCTL,REGION=2048K
//STEPLIB  DD  DSNAME=&LNGPRFX..SIGYCOMP,
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,UNIT=VIO,
//             DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
//             DCB=(BLKSIZE=&SYSLBLK)
//SYSUT1   DD  UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT2   DD  UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT3   DD  UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT4   DD  UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT5   DD  UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT6   DD  UNIT=VIO,SPACE=(CYL,(1,1))
//SYSUT7   DD  UNIT=VIO,SPACE=(CYL,(1,1))
//LKED   EXEC PGM=HEWL,COND=(8,LT,COBOL),REGION=1024K
//SYSLIB   DD  DSNAME=&LIBPRFX..SCEELKED,
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=&&LOADSET,DISP=(OLD,DELETE)
//         DD  DDNAME=SYSIN
//SYSLMOD  DD  DSNAME=&&GOSET(&GOPGM),SPACE=(TRK,(10,10,1)),
//             UNIT=VIO,DISP=(MOD,PASS)
//SYSUT1   DD  UNIT=VIO,SPACE=(TRK,(10,10))
//GO     EXEC PGM=*.LKED.SYSLMOD,COND=((8,LT,COBOL),(4,LT,LKED)),
//             REGION=2048K
//STEPLIB  DD  DSNAME=&LIBPRFX..SCEERUN,
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
```


10.3.3 编译面向对象(OO)的应用程序

如果您使用批处理作业或者TSO/E来编译一个OO COBOL程序或类定义，在通常情况下，将产生目标卡片叠并写入到通过SYSLIN或者SYSPUNCH的DD名所指定的数据集中。

如果COBOL程序或者类定义使用了JNI[1]环境架构访问JNI可调用服务，则将文件JNI.cpy从HFS复制到一个PDS或者PDSE的名为JNI的成员中，并通过SYSLIB DD语句指定库，然后通过使用COPY语句在COBOL源程序中插入COPY JNI。

示例10-11所示，使用SYSJAVA的DD名将产生的Java源代码写入到HFS文件之中。例如：

示例10-11 为Java源文件配置的SYSJAVA DD名

```
//SYSJAVA DD PATH='/u/userid/java/Classname.java',
//           PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//           PATHMODE=SIRWXU,
//           FILEDATA=TEXT
```

10.3.4 什么是目标卡片叠？

一个目标卡片叠是由汇编程序、编译器或者其他语言翻译转换器所产生的一个或者多个编译单元的集合，用作绑定器(或链接编辑器)的输入。

目标卡片叠是可以重新定位的不可执行的机器代码。一个装入模块同样也是可重定位的，但它是可执行的机器代码。装入模块的组织形式使其可以被载入到虚存中，并可通过程序管理器重新定位，程序管理器将装入模块载入到内存的某个特定位置，使其可以运行。

目标卡片叠和装入模块共有一些相同的逻辑结构，包括：

- 控制字典，包含了解析不同模块控制段间所交叉引用的符号，以及重定位地址常量所需用到的相关信息
- 文本，包含了程序指令及数据
- 模块结束标志，在目标卡片叠中以END语句标明，或者在装入模块中以模块终结符标明

目标卡片叠存放于通过SYSLIN或者SYSPUNCH DD语句所指定的分区数据集中，该数据集随后会被作为链接编辑处理的输入。

10.3.5 什么是目标库？

您可以使用目标库来存储目标卡片叠。要被链接编辑的目标卡片叠可从目标库中得到，并被转换成可执行或者可载入的程序。

但使用OBJECT编译器选项时，您可以将目标卡片叠作为一个传统的数据集，如一个UNIX文件，存放在磁盘或者磁带上。SYSLIN DD语句的DISP参数值决定了目标卡片叠将被执行的操作：

[1] Java 本地接口(JNI)是本地编程语言的 Java 接口，属于 Java 开发包的一部分。使用 JNI 编写程序，可以确保您的代码可以移植到多个平台上。

- 在编译之后传递给绑定器(或者链接编辑器)(DISP=PASS)
- 将其编目于某个已存在的目标库中(DISP=OLD)
- 保留(DISP=KEEP)
- 加入到一个新的目标库中，该目标库将在作业步结束时被编目(DISP=CATLG)

通过在SYSLIN DD语句中指定目标卡片叠的数据集及其成员名称，目标卡片叠将成为绑定器的主要输入。在如下的示例之中，在目标库USER.LIBROUT中名为TAXCOMP的成员是主要的输入。USER.LIBROUT是一个已编目的分区数据集：

```
//SYSLIN DD DSNAME=USER.LIBROUT(TAXCOMP),DISP=SHR
```

库成员将被等同于顺序数据集一样被处理。

10.3.6 程序管理如何运作？

尽管程序管理组件提供了很多的服务，但主要还是用在将目标卡片叠转换成可执行程序、存储入程序库并在执行时刻载入到虚存之中。

您可以使用程序管理的绑定器和装载器完成这些任务。这些组件也可以和链接编辑器一起联合使用。一个链接编辑所产生的装入模块可被绑定器接收成为其输入，或者通过程序管理的装载器载入到内存当中执行。链接编辑器同样也能处理绑定器所产生的装入模块。

第327页图10-2说明了程序管理组件是如何协同工作的，并且各自是如何工作以产生可执行程序的。我们已经讨论过其中的一些组件(源模块及目标卡片叠)，所以这里我们来看其他的部分。

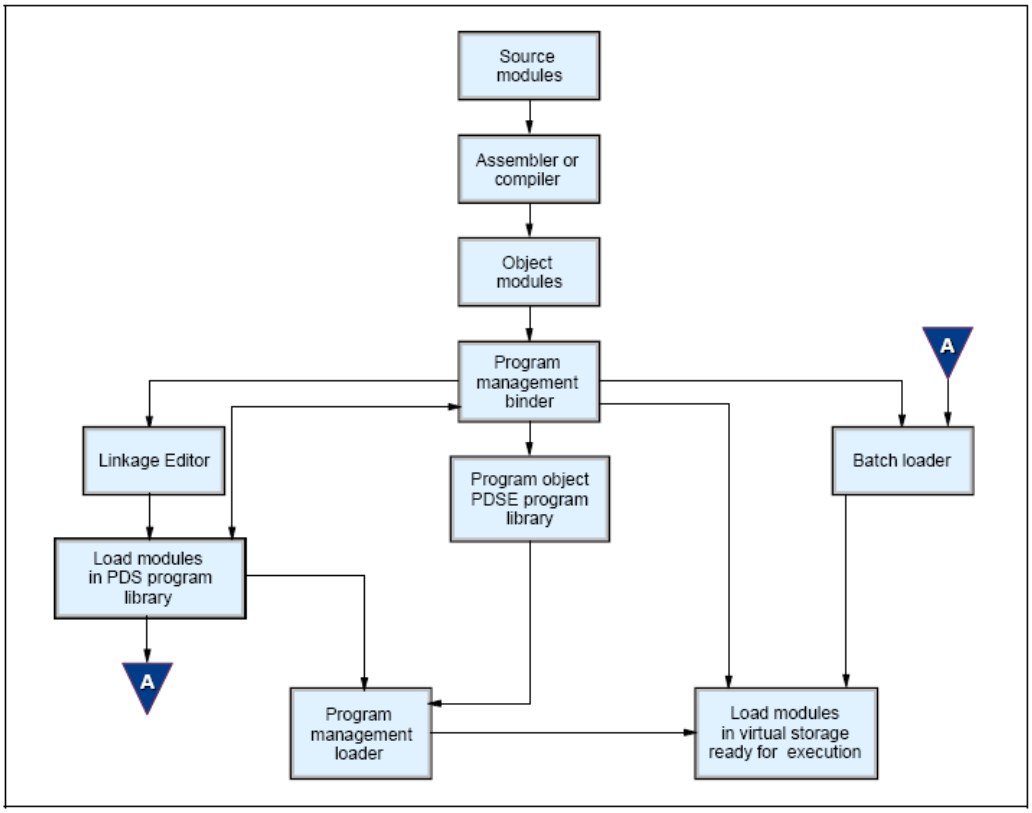


图10-2 使用程序管理组件创建并载入程序

10.3.7 如何使用链接编辑器？

链接编辑器处理过程紧随源程序的汇编或者编译过程。"链接编辑器"既是一个处理程序同时也是一个同语言翻译转换器联合使用的服务程序。

链接编辑器和装载器处理程序将接收语言翻译转换器的输出，并加以处理以准备执行。链接编辑器生成可执行模块，程序管理在执行时刻将可执行模块载入到内存之中。

链接编辑器接收两种主要的输入类型：

- 主输入，包括目标卡片叠以及链接编辑器的控制语句。
- 附加用户指定的输入，包括了目标卡片叠和控制语句，或者装入模块。这部分输入可以由用户指定，或者从一个调用库中由链接编辑器自动合并进来。

链接编辑器的输出有两种：

- 装入模块，作为一个命名成员，存放于库(或分区数据集)中。
- 诊断信息输出，以顺序数据集的形式输出。

装载器将可执行程序装载至内存中并把控制权直接交由程序。

10.3.8 如何创建装入模块？

在处理目标卡片叠和装入模块的过程之中，链接编辑器把连续的相对虚存地址赋予控制段，并解析控制块之间的引用。几种不同的语言翻译转换器所产生的目标卡片叠可以用来构成同一个装入模块。

一个输出的装入模块是由所有被链接编辑器处理过的的输入目标卡片叠和输入装入模块所组成。因此，一个输出模块的控制字典，是由所有链接编辑器输入的控制字典的合成。一个装入模块的控制字典称为复合外部符号字典(CESD)以及重新配位表(RLD)。装入模块同样也包含了每个输入模块的文本内容，以及一个模块结束标志符。

第329页图10-3展示了两个源程序的编译过程：PROGA和PROGB。PROGA是一个COBOL程序而PROGB是一个汇编语言程序。PROGA调用PROGB。在该图中，我们可以看到编译之后，PROGA中对PROGB的引用未被解析。链接编辑两个目标卡片叠的过程解析了这个引用，使得在PROGA执行时能够正确地调用PROGB工作。PROGB将被转向调用并执行，在PROGB被调用完成后，控制权将会交还给PROGA。

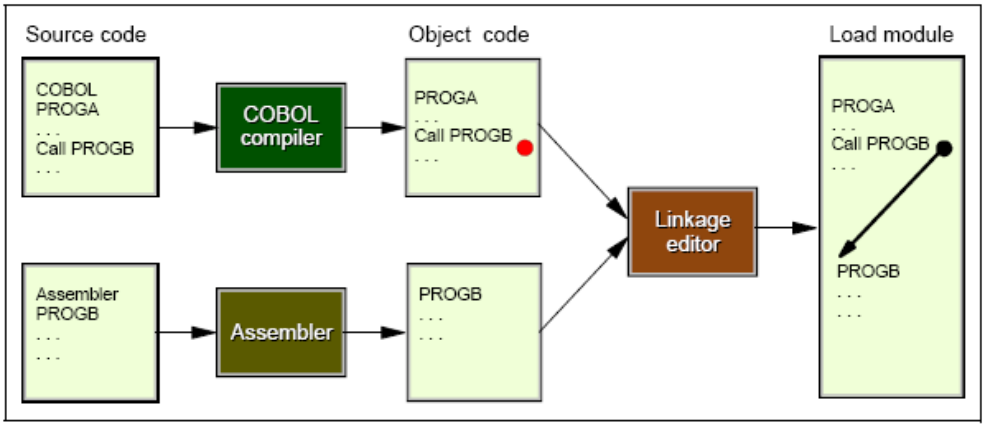


图10-3 在装入模块创建时刻解析引用

使用绑定器

程序库
包含装入模块和程序目标的数据集。

z/OS提供的绑定器可以完成所有链接编辑器的功能。绑定器链接编辑(合成并编辑)组成应用程序的分离的目标卡片叠、装入模块、以及程序对象，并生成一个单独的程序对象或者装入模块，您可以将其装载执行。当需要用到某个程序库成员时，装载器将其载入到虚存中做好执行准备。

您可以使用绑定器：

- 将目标卡片叠或装入模块转换成为一个程序对象并将其存储在扩展分区数据集的(PDSE)程序库中，或者z/OS UNIX文件之中。
- 将目标卡片叠或程序对象转换成为一个装入模块并将其存储在分区数据集(PDS)的程序库中。该功能等价于链接编辑器对目标卡片叠以及装入模块所完

成的功能。

- 将目标卡片叠或可执行模块，或程序对象，转换成一个虚存中可执行的程序并将其执行。该功能等价于批处理装载器对目标卡片叠及装入模块所完成的功能。

绑定器处理目标卡片叠、装入模块和程序对象，将多个模块链接编辑或绑定到一个单独的装入模块或程序对象中。控制语句指定了如何将输入合成为一个或者多个具有连续虚存地址的装入模块或程序对象。每个目标卡片叠都可以单独被绑定器处理，这使得只有那些修改过的模块需要被重新编译或汇编。绑定器可以创建24位、31位以及64位寻址模式的程序。

您可以指定一个寻址模式(AMODE)以指定在程序执行时使用哪一个硬件寻址模式。寻址模式包括：

329

- 24，即表明24位寻址模式必须生效。
- 31，即表明31位寻址模式必须生效。
- 64，即表明64位寻址模式必须生效。
- ANY，即表明24位、31位或者64位寻址模式都可以生效。
- MIN，即需要绑定器为对应的程序模块指定AMODE值。

绑定器将在输入的所有控制段中选择限制性最强的AMODE赋予给程序模块。AMODE值为24时限制性是”最强”的；AMODE值为ANY为限制性”最弱”的。

所有链接编辑器的服务都可以由绑定器来完成。更多有关地址的分布以及地址空间的哪个区域可以被24位、31位和64位寻址，可参见第96页3.4.9”虚拟存储的历史和64位寻址”。

绑定器和链接编辑器

绑定器相比于链接编辑器而言少了很多的限制。绑定器没有链接编辑器中别名最多为64个的限制，允许一个装入模块或程序对象拥有任意所需数量的别名。绑定器的主输入数据集(SYSLIN)可以具有系统支持的任意块大小，而没有链接编辑器中块大小不能超过3200字节的限制。绑定器同样也没有限制外部名称的数量，而链接编辑器限制了最多只能有32767个名称。

顺便提一下，z/OS语言环境提供了另一种称为预链接器的工具，它可以将多个目标卡片叠组合成一个单独目标卡片叠。在预链接之后，您可以将目标卡片叠链接编辑成一个装入模块(存于PDS中)，或者将其绑定到一个装入模块或者程序对象(存于PDS、PDSE或者在zFS文件中)中。然而通过绑定器，z/OS应用程序员不再需要进行预链接，因为绑定器将完成预链接器的所有功能。使用绑定器还是链接编辑器取决于个人的喜好。绑定器是最近使用的创建装入模块的方法。

每一个绑定器作业步所需要的主要输入由DD名为SYSLIN的DD语句定义。主要输入可以是：

- 一个顺序数据集
- 一个分区数据集(PDS)的成员
- 一个扩展分区数据集(PDSE)的成员
- 并置的顺序数据集、分区数据集或扩展分区数据集的成员，或者其组合

330

► 一个z/OS UNIX文件

主数据集可以包含目标卡片叠、控制语句、装入模块以及程序对象。所有的模块和控制语句都被顺序地处理，并且它们的顺序决定了绑定器处理的顺序。然而，处理之后各部分的顺序可能不同于它们的输入顺序。

绑定器实例

示例10-12表示一个能用于链接编辑一个目标卡片叠的作业。LKED作业步的输出将被置于SYSLMOD DD所指定的私有库中。绑定器作业步的输入是由同一个作业中的前一个作业步传递过来的(例如，编译器的输出将直接成为绑定器的输入)。

示例10-12 绑定器JCL示例

```
//LKED     EXEC PGM-IEWL,PARM-'XREF,LIST', IEWL is IEWBLINK alias
//             REGION-2M,COND-(5,LT,prior-step)
//*
//*        Define secondary input
//*
//SYSLIB   DD  DSN-language.library,DISP-SHR           optional
//PRIVLIB  DD  DSN-private.include.library,DISP-SHR optional
//SYSUT1   DD  UNIT-SYSDA,SPACE-(CYL,(1,1))            ignored
//*
//*        Define output module library
//*
//SYSLMOD  DD  DSN-program.library,DISP-SHR            required
//SYSPRINT DD  SYSOUT-*                                required
//SYSTERM  DD  SYSOUT-*                                optional
//*
//*        Define primary input
//*
//SYSLIN   DD  DSN-&&OBJECT,DISP-(MOD,PASS)            required
//         DD  * inline control statements
         INCLUDE  PRIVLIB(membername)
         NAME     modname(R)
/*
```

JCL语句的相关解释如下：

EXEC 绑定一个程序模块并将其存储在一个程序库中。IEWBLINK的可替换名为IEWL、LINKEDIT、EWL以及HEWLH096。

PARM域选项请求产生交叉引用表和模块映射表，放置到诊断输出数据集中。

331

SYSUT1 定义一个临时的可直接访问数据集用作中间数据集。

SYSLMOD 定义一个临时数据集用作输出模块库。

SYSPRINT 定义诊断输出数据集，被赋为输出类A。

SYSLIN 定义主输入数据集，&&OBJECT，包含了输入目标卡

片叠；这个数据集从上一个作业步传递而来并要一直传到该作业步的尾端。

INCLUDE 指定顺序数据集、库成员或z/OS UNIX文件，以用作绑定器的附加输入源(在该例中，为私有库PRIVLIB的成员)。

NAME 指定了由前面输入模块所创建的程序模块，并用作程序模块输入的界定符。(R)表明该程序模块将替换输出模块库中的同名模块。

10.4 为可执行程序创建装入模块

可重定位的
意指可以将可调入模块定位在虚存中的任意地址上。

“装入模块”是存储于分区数据集程序库中的可执行程序。如果仅仅为了执行而要创建装入模块，您需要用到批处理装载器或者程序管理装载器。要创建一个可存储于程序库的装入模块，您需要用到绑定器或者链接编辑器。而在所有的情形下，装入模块都是可重定位的，也就是说它可以定位于驻存模式(RMODE)界限之内的虚存中的任意地址。

一旦程序被载入，控制权连同基地址寄存器中的值将交由程序，即给定了程序开始执行的地址，即载入点，通过基地址加偏移量，可以使得程序的所有地址得以解析。可重定位程序使得程序的同一份拷贝能够载入许多不同的地址空间中，每次都被载入到不同的起始地址点。更多有关可重定位程序的讨论可参阅第315页10.3”在z/OS上编译程序”。

10.4.1 批处理装载器

332

“批处理装载器”将基本的编辑和载入服务(链接编辑器及程序管理器同样也提供)合成到一个作业步中。批处理装载器接收目标卡片叠和装入模块，并将它们载入到虚存中执行。与绑定器和链接编辑器所不同的是，批处理装载器不产生可以存储到程序库中的可执行模块。批处理装载器将可执行程序载入内存中准备执行，并将控制权直接交给程序。

批处理装载器处理是在载入步骤完成，该步骤等价于绑定器或者链接编辑器中的链接编辑以及执行步骤。批处理装载器可以用在编译-装载和装载作业中。它可以包含来自调用库(SYSLIB)、链接装配区(LPA)的模块。和其他程序管理组件类似，批处理装载器支持24位、31位和64位寻址模式下的寻址及驻存模式属性。批处理装载器程序是可重入的，因而它能够驻存于链接装配区中(LPA)。

注意：在更新版本的z/OS中，绑定器已经替代了批处理装载器。

10.4.2 程序管理装载器

程序管理装载器通过加入对程序对象的载入支持，增强了程序管理组件的相关服

务。装载器将程序对象和装入模块读入到虚存中准备运行。它解析程序中的地址常量并指向到虚存的适当位置，程序装载器支持24位、31位以及64位寻址模式。

在处理对象和装入模块的过程中，链接编辑器将连续的相对虚存地址赋予给控制段部分，并解析控制段间的引用。几种不同的语言翻译转换器所产生的目标卡片叠可以用来组成同一个装入模块。

在示例10-13所示为一个编译、链接编辑并执行的作业，该例使用了汇编语言。

示例10-13 编译、链接编辑和执行的JCL

```
//USUAL     JOB   A2317P,'COMPLGO'
//ASM       EXEC  PGM=IEV90,REGION=256K,  EXECUTES ASSEMBLER
//           PARM=(OBJECT,NODECK,'LINECOUNT=50')
//SYSPRINT  DD    SYSOUT=*,DCB=BLKSIZE=3509  PRINT THE ASSEMBLY LISTING
//SYSPUNCH  DD    SYSOUT=B PUNCH THE ASSEMBLY LISTING
//SYSLIB    DD    DSNAME=SYS1.MACLIB,DISP=SHR THE MACRO LIBRARY
//SYSUT1    DD    DSNAME=&&SYSUT1,UNIT=SYSDA,    A WORK DATA SET
//           SPACE=(CYL,(10,1))
//SYSLIN    DD    DSNAME=&&OBJECT,UNIT=SYSDA, THE OUTPUT OBJECT DECK
//           SPACE=(TRK,(10,2)),DCB=BLKSIZE=3120,DISP=(,PASS)
//SYSIN     DD    *                             inline SOURCE CODE
                  .
                  .
                  code
                  .
/*
//LKED      EXEC  PGM=HEWL,                     EXECUTES LINKAGE EDITOR
//           PARM='XREF,LIST,LET',COND=(8,LE,ASM)
//SYSPRINT  DD    SYSOUT=*                       LINKEDIT MAP PRINTOUT
//SYSLIN    DD    DSNAME=&&OBJECT,DISP=(OLD,DELETE) INPUT OBJECT DECK
//SYSUT1    DD    DSNAME=&&SYSUT1,UNIT=SYSDA,    A WORK DATA SET
//           SPACE=(CYL,(10,1))
//SYSLMOD   DD    DSNAME=&&LOADMOD,UNIT=SYSDA,   THE OUTPUT LOAD MODULE
//           DISP=(MOD,PASS),SPACE=(1024,(50,20,1))
//GO        EXEC  PGM=*.LKED.SYSLMOD,TIME=(,30), EXECUTES THE PROGRAM
//           COND=((8,LE,ASM),(8,LE,LKED))
//SYSUDUMP  DD    SYSOUT=*                        IF FAILS, DUMP LISTING
//SYSPRINT  DD    SYSOUT=*,                       OUTPUT LISTING
//           DCB=(RECFM=FBA,LRECL=121)
//OUTPUT    DD    SYSOUT=A,                      PROGRAM DATA OUTPUT
//           DCB=(LRECL=100,BLKSIZE=3000,RECFM=FBA)
//INPUT     DD    *                              PROGRAM DATA INPUT
                  .
                  .
                  data
                  .
/*
//
```

注意：

- 在ASM(编译)作业步中，SYSIN DD用于内嵌的源代码而SYSLIN DD用于输出的目标卡片叠。
- 在LKED(链接-编辑)作业步中，SYSLIN DD用于输入目标卡片叠而SYSLMOD DD用于输出装入模块。
- 在GO(执行程序)作业步中，EXEC JCL语句表明它将执行前面作业步的SYSLMOD DD语句中所指定的程序。
- 该示例没有使用前述COBOL示例中所用的编目过程，而是将所有的JCL内嵌写入到作业里。我们本可以使用一个已存在的JCL过程，或自己编写一个过程，然后只需要提供相应的诸如INPUT DD语句的覆写就可以了。

10.4.3 什么是装载库？

334

“装载库”包含了可执行的程序。装载库可以是如下的任何一种：

- 系统库
- 私有库
- 临时库

系统库

如果作业或者作业步没有指定私有库，当您编写了如下语句时，系统将在系统库中搜索程序：

```
//stepname  EXEC  PGM=program-name
```

系统将查找在对应库中和所指定程序名具有相同名称或别名的成员。使用最广泛的系统库是SYS1.LINKLIB，它包含了被链接编辑器所处理过的可执行程序。更多有关系统库的信息，请参阅第466页16.3.1节”z/OS系统库”。

私有库

每个用户编写的可执行程序都是一个私有库成员。为了向系统表明程序位于私有库中，相关的DD语句可用如下方法中的一种来指定相关库：

- 在JOB语句之后并在第一个EXEC语句之前，使用DD名为JOBLIB的DD语句指定
- 如果库将仅仅用于一个作业步，则通过DD名为STEPLIB的DD语句在作业步中指定

为执行一个私有库中的程序，可以编写：

```
//stepname  EXEC  PGM=program-name
```

当您指定了JOBLIB或者STEPLIB时，系统将在搜索系统库之前，先在由JOBLIB或者STEPLIB语句所定义的库中搜索需要执行的程序。

如果在作业中之前的DD语句指定了程序为私有库成员，则可通过指向那个DD语

句来执行该程序：

```
//stepname    EXEC    PGM=*.stepname.ddname
```

对于那些由于很少被用到而不需要放到系统库中的程序，私有库特别的适用。例如产生季度销售税报表的程序，就很适合放在私有库中。

临时库

“临时库”是分区数据集，被创建用来暂存程序，直到这些程序在同一作业的后续步中被用到。临时库在同一作业中被创建和删除。

在测试一个刚刚编好的程序时，临时库特别适用于存储链接编辑器产生的装入模块，这些模块将在后续的作业步中被执行。在经过全面测试之前，其他的作业不会需要该模块，因此该模块不需要存储在私有库或者系统库之中。在第333页示例10-13中，LKED作业步的SYSLMOD DD语句创建了一个名叫&&LOADMOD的临时库。在GO作业步中，我们通过如下的方式引用了前面的临时库：

```
//GO       EXEC      PGM=*.LKED.SYSLMOD，…….
```

10.5 编译到执行过程概览

由第336页图10-4中所示，我们可以发现目标卡片叠与存在装载库中并将被载入到中央存储中执行的装入模块之间的关系。

从两个程序A和B开始，它们将被编译成两个目标卡片叠，然后这两个目标卡片叠被链接成一个装入模块称为MYPROG，并存于直接访问存储设备上的一个装载库中。装入模块MYPROG将随后被程序管理装载器载入到中央存储中，得到控制权后开始运行。

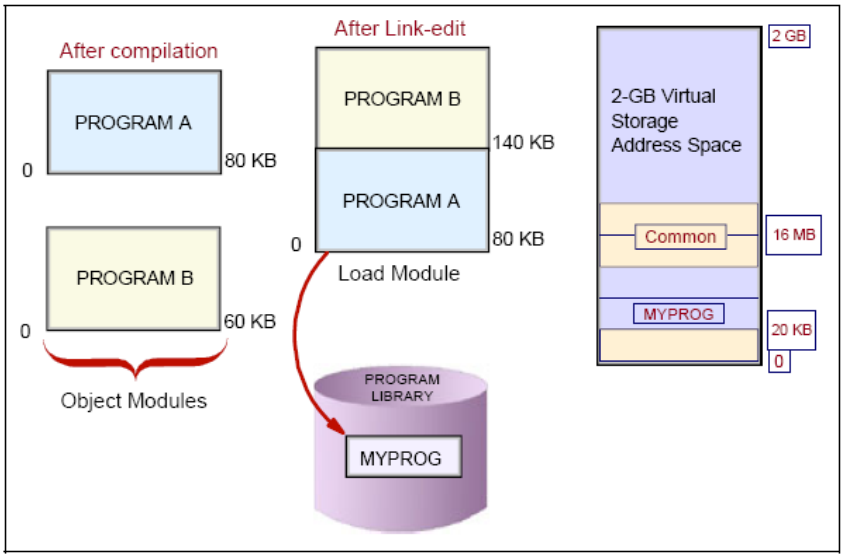


图10-4　程序编译、链接编辑以及执行

10.6 过程的使用

为节省时间和防止错误产生，您可以预先准备几组作业控制语句并将它们存于分区数据集(PDS)或者扩展分区数据集(PDSE)，即我们所说的”过程库”中。它可以用于诸如编译、汇编、链接编辑以及执行程序，如333页示例10-13所示。更多有关JCL过程的深入讨论，请参阅第212页6.7”JCL过程(PROC)”

过程库即是包含了多个过程的库。系统过程库SYS1.PROCLIB(或者安装时定义的过程库)中的一组作业控制语句，称为”编目过程”。(关于SYS1.PROCLIB请参阅第220页6.10”系统库”)

为了在过程存入过程库前先对它进行测试，可将过程加入到输入流中执行；位于输入流中的过程称为内嵌过程。在任何作业中可以编写的内嵌过程的最大数目为15个。为了在输入流中测试过程，过程的结尾必须加上过程结束(PEND)语句。PEND语句表明已到达PROC结尾。这仅仅是内嵌过程所需。在过程库中，您不需要使用PEND语句。

一个内嵌过程必须出现在同一作业中调用它的EXEC语句之前。

示例10-14 过程定义样例

```
//DEF     PROC  STATUS=OLD,LIBRARY=SYSLIB,NUMBER=777777
//NOTIFY  EXEC  PGM=ACCUM
//DD1     DD    DSNAME=MGMT,DISP=(&STATUS,KEEP),UNIT=3400-6,
//            VOLUME=SER=888888
//DD2     DD    DSNAME=&LIBRARY,DISP=(OLD,KEEP),UNIT=3390,
//            VOLUME=SER=&NUMBER
```

在示例10-14的编目过程中定义了三个符号参数：分别是&STATUS、&LIBRARY和&NUMBER。在PROC语句中符号参数给予赋值。在过程被调用并且调用的EXEC语句没有对符号参数赋值的情况下，这些值将会生效。

在示例10-15中我们测试了名为DEF的过程。注意过程是由PROC和PEND语句描述。紧跟过程DEF后的EXEC语句引用了该改过程，随后将被调用执行。在本例中，由于名字DEF对应了本作业内前面写入的过程，因此系统将直接使用该内嵌的过程，而不会再做搜索。

示例10-15 测试一个内嵌的过程

```
//TESTJOB  JOB ....
//DEF      PROC  STATUS=OLD,LIBRARY=SYSLIB,NUMBER=777777
//NOTIFY   EXEC  PGM=ACCUM
//DD1      DD    DSNAME=MGMT,DISP=(&STATUS,KEEP),UNIT=3400-6,
//            VOLUME=SER=888888
//DD2      DD    DSNAME=&LIBRARY,DISP=(OLD,KEEP),UNIT=3390,
//            VOLUME=SER=&NUMBER
//         PEND
//*
//TESTPROC EXEC DEF
//
```

10.7 总结

本章描述将一个源程序翻译转换成可执行的装入模块，并执行该装入模块的相关过程。尽管可能存在编译源程序之前的预处理步骤，但该翻译转换过程的基本步骤就是编译和链接编辑。预处理步骤只有在您的源程序中使用了CICS命令语言调用或者SQL调用时才会需要。预处理步骤的输出随后将传给编译步骤。

编译步骤的目的在验证源代码并将其翻译转换成可重定位的机器语言，以目标卡片叠的形式存在。尽管目标卡片叠是机器语言，但它是不可执行的。它必须被链接编辑器、绑定器或者装载器处理后方可执行。

链接编辑器、绑定器以及装载器将目标卡片叠以及其他装入模块作为输入，产生出一个可执行的装入模块，如果是装载器，装入模块随后还会被执行。该处理过程中解析了目标卡片叠中所有的未解析的引用，并保证该程序执行所需要的所有部分都包含在最终的装入模块中。此时装入模块已准备好可以执行。

要执行一个装入模块，装入模块必须被装载到中央存储之中。绑定器或者程序管理服务将模块载入到内存中并将控制权交给程序以开始执行。将控制权交给模块包括了提供给程序其在内存中的首地址。由于程序的指令和数据是通过基地址和相对基地址的位移来寻址的，因此获取首地址可以让系统在位移范围内对指令和数据进行寻址[2]。

338

本章中的重要术语		
绑定器(binder)	copybook	链接编辑器(linkage editor)
装入模块(load modules)	目标卡片叠(object deck)	面向对象代码 (object-oriented code)
过程(procedure)	过程库(procedure library)	程序库(program library)
可重定位(relocatable)	源模块(source module)	

[2] 每个基地址寄存器的最大位移为 4096(4k)。为了对全部程序寻址，大于 4K 的程序必须有多个基地址寄存器。

10.8 复习题

为了帮助您检测对本章的理解程度，请完成下列问题：

1. 执行一个源程序需要哪些步骤？
2. 如何修改一个目标卡片叠？或一个装入模块？
3. 系统有多少种不同类型的装载库？
4. 什么是过程库？它有什么用途？
5. 链接编辑器和绑定器有什么不同？
6. Copybook和编目过程有哪些相似之处？
7. 编译器的作用是什么？它的输入和输出各是什么？
8. 可重定位的含义是什么？
9. 目标卡片叠和装入模块之间的区别是什么？
10. SYSLMOD DD语句的作用是什么？
11. 为什么一个内嵌的PROC需要一个PEND语句而编目的PROC不需要？

10.9 练习

本章的实验练习将帮助您培养配置和准备运行在z/OS上的程序的技能。这些技能对完成本书中后面的实验练习是必要的。

为完成实验练习，您或者您的团队需要一个TSO用户ID以及密码(在讲师协助下完成)。

练习主要包括：

339

- 第340页”练习：编译和链接一个程序”
- 第342页”练习：执行一个程序”

10.9.1 练习：编译和链接一个程序

在该部分，使用至少两种编程语言来编译和链接；参见JCL：

yourid.LANG.CNTL(language)

这里"language"是指ASM、ASMLE、C、C2、COBOL、COBOL2、PL1、PL12中的一个。

在尝试第342页10.9.2"练习：执行一个程序"之前，请先完成这个练习。该练习中每一个作业成功运行的最终结果为创建多个装入模块，这些装入模块将在下一个练习中使用。

提示：JCL可能需要相应的修改，用来指定提交作业的学生的高级限定词(HLQ)。此外，任何涉及到语言环境数据集的作业都要作相应的修改。更多信息参见注释栏。

要提交作业，可在ISPF命令行上输入SUBMIT命令。一旦作业完成，您需要通过SDSF来查看作业的输出结果。

1. 提交下列的数据集内容以编译并链接一个复杂的汇编语言程序：

   yourid.LANG.CNTL(ASMLE)

提示：学生可能需要修改JCL中以CEE开头的数据集。请询问您的系统程序员语言环境数据集的高级限定词(HLQ)是什么。下列JCL的加强字体部分可能需要修改。

```
//C.SYSLIB      DD DSN=SYS1.MACLIB,DISP=SHR
//                   DD DSN=CEE.SCEEMAC,DISP=SHR
//C.SYSIN       DD DSN=ZUSER##.LANG.SOURCE(ASMLE),DISP=SHR
//L.SYSLMOD     DD DSN=ZUSER##.LANG.LOAD(ASMLE),DISP=SHR
//L.SYSLIB      DD DSN=CEE.SCEELKED,DISP=SHR
//               DD DSN=CEE.SCEELKEX,DISP=SHR
```

2. 提交下列数据集内容以编译并链接一个简单的汇编语言程序：

   yourid.LANG.CNTL(ASM)

3. 提交下列的数据集内容以编译并链接一个复杂的C语言程序

   yourid.LANG.CNTL(C)

提示：学生可能需要修改JCL中以CEE和CBC开头的数据集。向您的系统程序员查询语言环境数据集以及C语言数据集的高级限定词(HLQ)是什么。下列JCL的加强字体部分可能需要修改。

```
//STEP1 EXEC PROC=EDCCB,LIBPRFX=CEE,LNGPRFX=CBC,
// INFILE='ZUSER##.LANG.SOURCE(C)',
// OUTFILE='ZUSER##.LANG.LOAD(C),DISP=SHR'
```

4. 提交下列数据集内容以编译并链接一个简单的C语言程序：

   yourid.LANG.CNTL(C2)

提示：学生可能需要修改JCL中以CEE和CBC开头的数据集。向您的系统程序员查询语言环境数据集以及C语言数据集的高级限定词(HLQ)是什么。下列JCL的加强字体部分可能需要修改。

```
//STEP1 EXEC PROC=EDCCB,LIBPRFX=CEE,LNGPRFX=CBC,
```

```
// INFILE='ZUSER##.LANG.SOURCE(C2)',
// OUTFILE='ZUSER##.LANG.LOAD(C2),DISP=SHR'
```

5. 提交下列的数据集内容以编译并链接一个复杂的COBOL语言程序

   yourid.LANG.CNTL(COBOL)

> **提示：**学生可能需要修改JCL中以CEE开头的数据集。向您的系统程序员查询语言环境数据集的高级限定词(HLQ)是什么。下列JCL的加强字体部分可能需要修改。
>
> //SYSIN DD DSN=ZUSER##.LANG.SOURCE(COBOL),DISP=SHR
> //COBOL.SYSLIB DD DSN=CEE.SCEESAMP,DISP=SHR
> //LKED.SYSLMOD DD DSN=ZUSER##.LANG.LOAD(COBOL),DISP=SHR

6. 提交下列数据集内容以编译并链接一个简单的COBOL语言程序：

   yourid.LANG.CNTL(COBOL2)

341

7. 提交下列的数据集内容以编译并链接一个复杂的PL/I语言程序

   yourid.LANG.CNTL(PL1)

> **提示：**学生可能需要修改JCL中以CEE开头的数据集。向您的系统程序员查询语言环境数据集的高级限定词(HLQ)是什么。下列JCL的加强字体部分可能需要修改。
> //SYSIN DD DSN=ZUSER##.LANG.SOURCE(PL1),DISP=SHR
> //PLI.SYSLIB DD DSN=CEE.SCEESAMP,DISP=SHR
> //BIND.SYSLMOD DD DSN=ZUSER##.LANG.LOAD(PL1),DISP=SHR

8. 提交下列的数据集内容以编译并链接一个简单的PL/I语言程序

   yourid.LANG.CNTL(PL12)

10.9.2 练习：执行一个程序

在该部分，您所选择的语言实例已经在第340页的练习10.9.1”练习：编译和链接一个程序”中完成了编译和链接。如果上述练习没有成功完成，请不要尝试运行以下任何一个作业，因为它们将会以错误结束。

前面编译和链接作业执行后，生成并存储了装入模块。如下的练习包括执行每种语言实例所生成的装入模块的操作。对于解释型语言，您可以直接执行如下源代码成员：

342

youid.LANG.SOURCE(language)
这里”language”是CLIST、REXX中的一种。

提示：JCL可能需要相应的修改，来指定提交作业的学生的高级限定词(HLQ)。要提交作业，可在ISPF命令行上输入SUBMIT命令。一旦作业完成，您需要通过SDSF来查看作业的输出结果。

为使这些作业能够执行成功，学生需要完成第340页练习10.9.1"练习：编译和链接一个程序"，以在如下的数据集中创建装入模块：

ZPROF.LANG.LOAD

如果这些作业不能执行成功，学生可以在SDSF的作业日志中得到类似于以下的信息：

```
CSV003I REQUESTED MODULE ASM     NOT FOUND
CSV028I ABEND806-04 JOBNAME=ZPROF2 STEPNAME=STEP1
IEA995I   SYMPTOM DUMP OUTPUT 238
SYSTEM COMPLETION CODE=806 REASON CODE=00000004
```

模块名、JOBNAME名以及STEPNAME会各不相同，取决于具体提交的哪个作业。

1. 提交以下数据集内容以执行一个复杂的汇编程序：
   yourid.LANG.CNTL(USEASMLE)
   本例访问了z/OS语言环境并打印如下信息：
   "IN THE MAIN ROUTINE"
2. 提交以下数据集内容以执行一个简单的汇编程序：
   yourid.LANG.CNTL(USEASM)
   本例将返回码设置为15并退出。
3. 提交以下数据集内容以执行一个复杂的C语言程序：
   yourid.LANG.CNTL(USEC)
   本例打印了本地日期和时间。
4. 提交以下数据集内容以执行一个简单的C语言程序：
   yourid.LANG.CNTL(USEC2)
   本例打印出信息"HELLO WORLD"。
5. 提交以下数据集内容以执行一个复杂的COBOL语言程序：
   yourid.LANG.CNTL(USECOBOL)
   本例打印了本地日期和时间。

6. 提交以下数据集内容以执行一个简单的COBOL语言程序：
   yourid.LANG.CNTL(USECOBO2)
   本例打印出信息"HELLO WORLD"。
7. 提交以下数据集内容以执行一个复杂的PL/I语言程序：
   yourid.LANG.CNTL(USEPL1)
   本例打印了本地日期和时间。
8. 提交以下数据集内容以执行一个简单的PL/I语言程序：
   yourid.LANG.CNTL(USEPL2)
   本例打印出信息"HELLO WORLD"。
9. 执行以下一个复杂的CLIST语言程序：

yourid.LANG.CNTL(CLIST)
本例提示用户输入高级限定词(HLQ)并对该HLQ产生一个格式化的目录列表。

在ISPF命令行上键入：
TSO EX'yourid.LANG.SOURCE(CLIST)'
当出现提示之后，输入HLQ为yourid

10. 执行以下一个简单的CLIST语言程序：

yourid.LANG. SOURCE(CLIST2)
本例打印出信息"HELLO WORLD"。

在ISPF命令行上键入：
TSO EX'yourid.LANG.SOURCE(CLIST2)'

11. 执行以下一个复杂的REXX语言程序：

yourid.LANG. SOURCE(REXX)
本例提示用户输入高级限定词(HLQ)并对该HLQ产生一个格式化的目录列表。

在ISPF命令行上键入：
TSO EX'yourid.LANG.SOURCE(REXX)'
当出现提示之后，输入HLQ为yourid

12. 执行以下一个简单的REXX语言程序

yourid.LANG. SOURCE(REXX2)
本例打印出信息"HELLO WORLD"。

在ISPF命令行上键入：
TSO EX'yourid.LANG.SOURCE(REXX2)'

Part 3

第 3 部分 z/OS 上的联机工作负载

在该部分，我们研究z/OS上运行的联机或交互式工作负载的主要分类，如交易处理、数据库管理以及Web服务。以下的章节指导学生对网络通讯进行讨论，并介绍了几种常用的中间件产品，包括DB2，CICS和WebSphere。

347

348

11

第 11 章 z/OS 上的交易管理系统

目标：为了更多地了解关于主机开发的相关知识，您必须首先理解主机在当今在线交易系统中所扮演的角色。本章将介绍交易处理的概念与术语，展示用于主机上在线处理的几类主要的系统软件。我们集中介绍两种在z/OS上广泛使用的交易管理产品：CICS和IMS。

在本章结束之后，您应当能够学会：

- 描述大型系统在典型的在线业务中所扮演的角色。
- 列举大多数交易处理系统的共同特性。
- 解释CICS在在线交易系统中所扮演的角色。
- 描述CICS程序，CICS交易和CICS任务。
- 解释什么是会话程序，什么是伪会话程序。
- 解释CICS和Web使能技术。
- 讨论IMS的组件。

11.1 主机上的在线处理

在前面的章节中，我们已经讨论了批处理程序的价值，但它可不是唯一能够在z/OS和主机上运行的程序。正如在本章中所介绍的那样，在线程序也可以在z/OS上运行。我们将讨论什么是在线或交互应用程序，以及他们在主机环境中一些公共基础知识。

我们也将介绍常在应用程序中用于存储数据的数据库。数据库使开发变得更加容易，特别是对于一个关系数据库管理系统(RDBMS)而言。数据库把程序员们从组织和管理数据的负担中解放了出来。在本章的后面部分，我们将讨论几个在基于主机的企业中被广泛使用的交易管理系统。

我们以一个旅行社的例子作为开始。这个例子的需求与许多主机客户所提出的要求有共同之处，即可以为客户们提供对服务的直接访问也可以充分利用基于互联网的商业所具有的优势。

11.2 全球在线处理的示例——全新概览

一家大型的旅行社多年来一直依靠一个基于主机的批处理系统进行管理。在过去的几年中，旅行社的客户们享受到了极好的服务，旅行社也一直在完善它的系统。

当旅行社的商业活动开始的时候，他们的IT人员设计了一些用于支持旅行社内部或外部商务处理的应用程序：雇员信息，顾客信息，与汽车租赁公司的联系，和全世界旅馆的联系，航班的安排等等。起初这些程序以批处理的形式定期得到更新。

然而，这类数据不是静态的，并且越来越倾向于会被频繁地更改。举例而言，由于价格的变动频繁，系统很难在每时每刻维护即时信息。对于那些想要立即获取信息的客户而言，定期执行的批处理程序的更新显然无法满足他们的要求(尤其如果考虑到亚洲，欧洲和美洲之间的时差)。

如果用传统的批处理程序去开发这些应用，那么意味着从接收更改的消息到实际做出更改之间必定存在着时间差。旅行社需要一种方式(通过电话，传真或者电子邮件等等)在消息改变的瞬间立即对小量的数据进行更新，如351页图11-1所示。

350

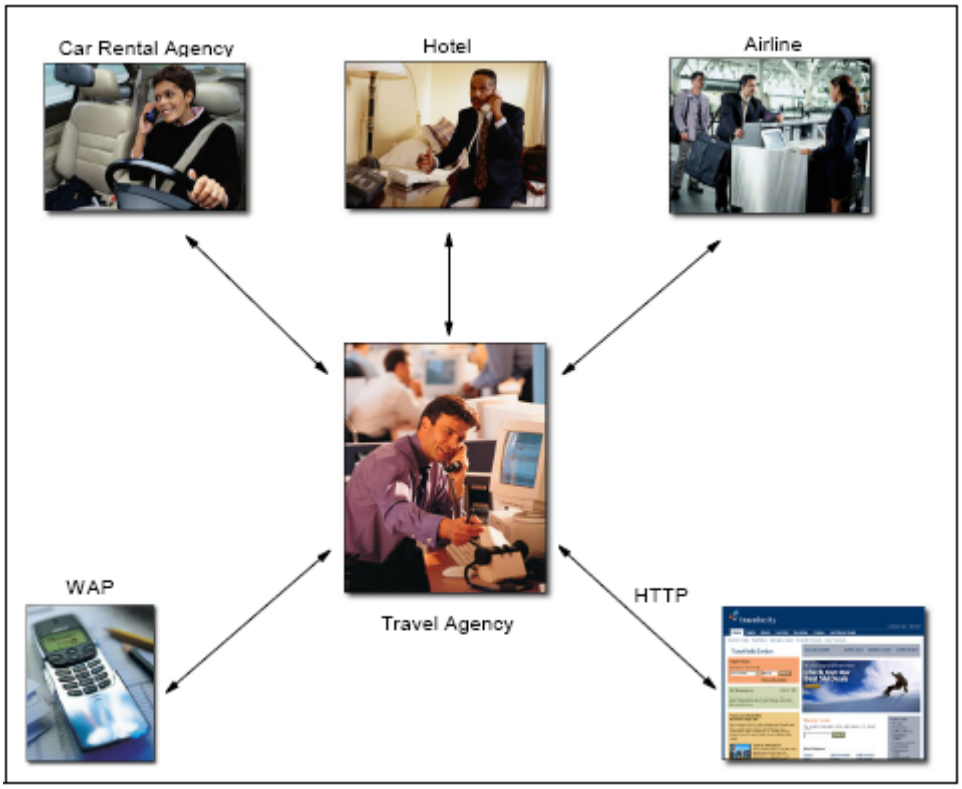


图11-1 一个实例

因此，旅行社的IT人员开发了一些新的程序。为了可以将最新的变化显示在终端用户的面前，这些新程序从本质上来讲必须具备交易的特性。由于系统中这些数据改变的即时性，这种程序被称为交易或者交互应用。

旅行社联系它的供应商去寻求解决的办法。他们需要一种让计算机能进行交互的工作方式。有些航空公司的系统运行在主机上，有些则不是；而且人们都想保留他们自己的程序。

终于，他们找到了一个解决方案！它使通信变得容易：只需要问一个问题，然后过几秒钟就得到答案—好主意。

由于客户也要参与交互，所以还需要有一些新的方案。PC已经非常普及，人们一定想要从互联网上浏览旅游的信息；也有一些客户使用他们的手提电脑作为无线接入点(WAP)。

11.3 主机的交易系统

日常生活中交易无处不在，例如当您买商品或者服务的时候，或者在互联网上搜索的时候，就产生了一个交易。交易是一种交换，通常为一个请求和应答，它频繁地发生，就如同在一个企业日复一日操作中所运行的日常事务一样。

交易有以下几个特性：

- 每个交易只处理和传送少量的数据
- 有数量庞大的用户
- 会被执行很多次

11.3.1 什么是交易程序？

一个商业交易是自包含的。一些交易含有一个简短的会话(比如改变住址)，有些则牵涉到在一段时间内的多个步骤(比如旅行的预定，需要确定汽车，旅馆和航班机票等)。

单个交易可能由许多执行业务流程必需的程序组成。大型的交易系统(例如IBM的CICS产品)依靠z/OS上的多任务(multitasking)与多线程(multithreading)的功能，可以在同一时间处理不止一个任务。系统为每个任务保存特定的变量数据，并且跟踪每个用户正在执行的指令。

对于一个可以让成千上万的用户在同一时间使用的系统而言，多任务是一个必备的基础。当一个多任务的交易系统接收到一个运行交易的请求时，它可以启动一个新的与这个交易相关联的交易运行实例；即，某个交易的一次运行，它有特定的数据集合，通常也只代表一个终端的一个特定客户工作。您可能也认为任务与UNIX上的线程相似。当交易完成的时候，任务随即结束。

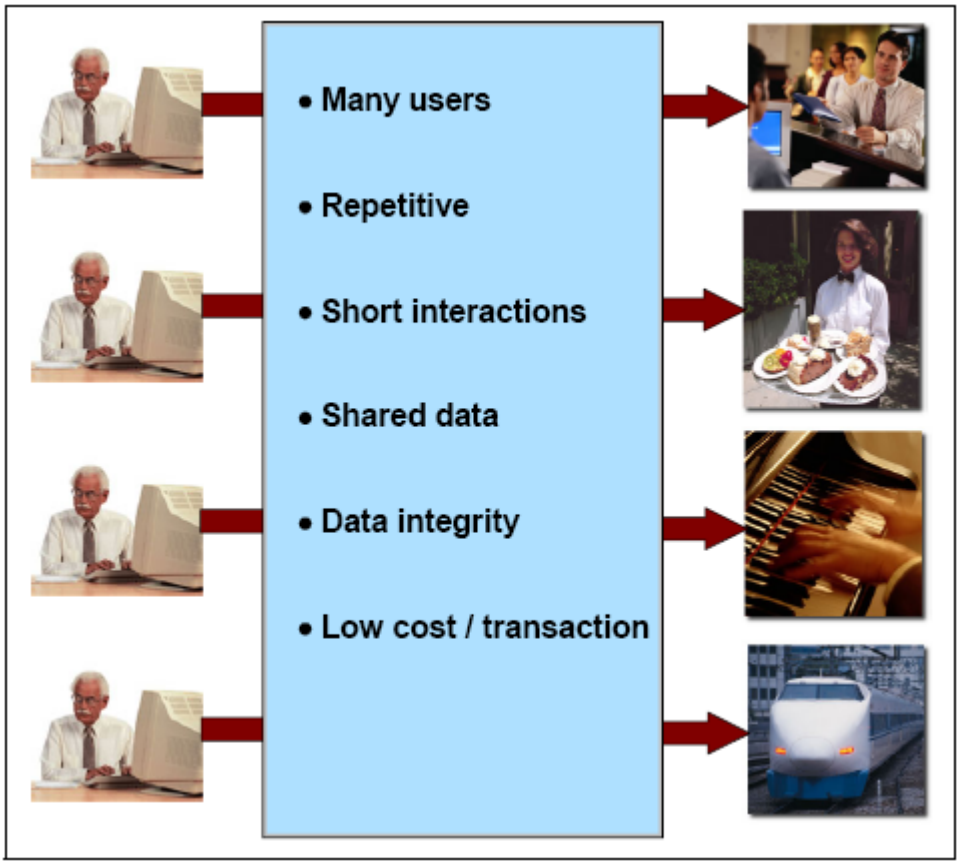
多线程：
一个应用程序的单一备份可以被几个交易并发处理使用。

多线程允许几个交易并发使用某个应用程序的单个备份。多线程需要所有的交易程序都是可重入的；即，在程序的入口和出口之间必须是可串行重复使用的。在程序设计语言的实现上，可重入性是靠每次程序调用时所获取的工作存储区的快速备份(*flash copy*)保证的。

352

11.3.2 什么是交易系统？

353页图11-2显示了交易系统的主要特性。在互联网出现以前，一个交易系统为成百上千的终端提供服务，每秒处理数十或数百次交易。无论从交易的速率还是从交易的融合来讲，这种任务量是相当容易做出估计的。



交易：
一个工作单元，由一个或多个交易程序执行，包含了一个特定集合的输入数据，并启动一个具体的过程或者任务。

图11-2 交易系统的特性

交易系统必须能够支持大量的并发用户和多种交易类型。

交易或者在线系统的主要特性之一是用户与系统之间的交互非常简短。大部分交易在很短的时间内被执行——有时是1秒钟。用户可以通过一个很短的交互完成一次交易，并每次交互都能获得即时响应。这些应用程序要处理关键性的任务；因此，持续的可用性，高性能，数据的保护及其完整性都是必需的。

在线交易处理(Online transaction processing，OLTP)是一种交互的交易过程。它需要满足以下几个条件：

- 即时响应
- 对于终端用户交易界面具有持续的可用性
- 安全性
- 数据完整性

在线交易对很多人来说都是很熟悉的，例如：

- ATM机上的交易，比如存款，取款，查询，转账
- 用借记卡或者信用卡在超市付款
- 从互联网上购买商品

实事上，在线系统具有很多操作系统的特性：

- 管理和调度任务
- 控制用户对系统资源的访问权限
- 内存使用管理
- 管理和控制对数据文件的同步访问
- 提供设备独立性

11.3.3 什么是交易系统的典型要求？

对于一个交易系统，必须遵守四项要求，它们经常被人们合在一起说，记作A-C-I-D或者ACID：

- 原子性：交易所执行的过程需要被视为一个整体。要么全部执行，要么全部不执行。
- 一致性：交易处理过程中需要保证信息的一致性。
- 独立性：来自2个或更多交易的过程需要确保各自的独立性。
- 持久性：交易所做出的改变需要是持久的。

通常，交易由一个和交易系统交互的终端用户通过一个具体的终端机器启动。在过去，交易系统仅仅支持通过远程处理网络连接的终端和设备。现在，交易系统可以处理由以下几个途径所提交的请求：

354

- 网页
- 远程工作站程序
- 另一个交易系统中的程序
- 在预先定义的时间点自动触发

11.3.4 什么是提交和回滚？

在交易系统中，提交和回滚所指的是一系列保证一致性的动作。一个应用程序要么对资源完成一个恢复单元里所有的改变，要么完全不做改变。两阶段提交协议提供了提交和回滚的操作。它验证即使只有一个部分(例如应用程序，系统或者资源管理器)出现了问题，操作也是要么所有的改变都完成，要么不做任何改变。协议允许在系统或者子系统出错的情况下执行重新启动或者恢复程序。

当应用程序准备提交或者回滚时会初始化两阶段提交协议。在此时，负责协调的恢复管理器，也被叫做同步点管理器，给每个资源管理器一个检查他们在该UR中的资源是否一致和是否可以被提交的机会。如果所有的资源通过检查，那么恢复管理器指示所有的资源管理器执行更改。如果任意一个资源管理器发现了问题，恢复管理器将指示它们停止更新操作。这个过程通常表现为两个阶段。

在第一阶段，应用程序向同步点协调器发送同步或者回滚的请求。同步点协调器向所有与该UR有关的资源管理器通过一个PREPARE的命令来发出同步点流程起始指令。作为对PREPARE命令的回应，每个参与到交易中的资源管理器答复

同步点协调器，告诉它自身是否作好了提交的准备。

当同步点协调器接受到了所有的回复之后，第二阶段开始启动。在这个阶段同步点协调器将根据先前收到的答复发出提交或者回滚的命令。如果任何一个资源管理器做出了无法进行提交的答复，那么同步点协调器将让所有的资源管理器回滚已做的改变。

同步点协调器通知所有的资源管理器去提交或者去回滚的时刻被称为原子瞬间(*atomic instant*)。在该时刻之后，无论发生任何错误，同步点协调器将确保所有的交易不是被提交就是被回滚。一个同步点协调器通常会去记录这个时刻所做出的决定。在原子瞬间发生之后，如果出现任何非正常终止，那么负责这部分的资源管理器必须与同步点协调器一起，在该部件重启的时候完成提交或者回滚。

在z/OS上，最主要的同步点协调器被称为资源恢复服务(RRS)。IBM的交易管理产品CICS也含有其内置的同步点协调器。

在协议的第一个阶段内，资源代理器(agent)不知道同步点协调器是否会做出提交或是回滚的决定。这段时间被称为信任时间(*indoubt period*)。根据其所处于两阶段提交过程的哪个阶段，UR被描述为一种特定的状态：

- 在UR对资源做出修改之前，被描述为*In-reset*。
- 当UR提出改变资源的请求时，被描述为*In-flight*。
- 当提交的请求被提出时(第一阶段)，被描述为*In-prepare*。
- 当同步点协调器做出提交的决定时(第二阶段)，被描述为*In-commit*。
- 当同步点协调器做出收回的决定时，被描述为*In-backout*。

图11-3展示了两阶段提交。

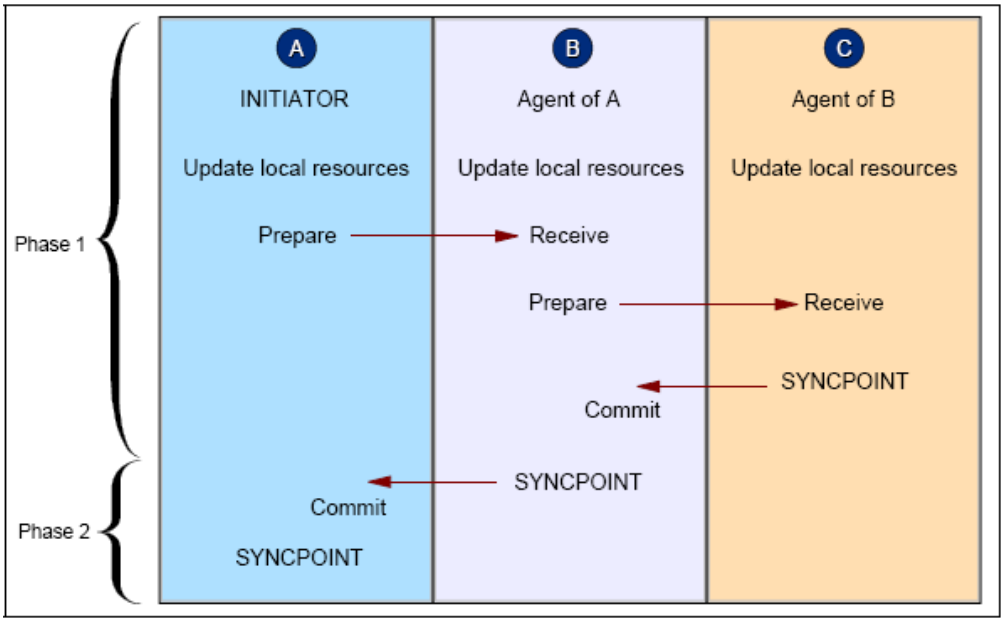


*图**11-3** 两阶段提交*

在z/OS上最常用的交易管理系统，比如CICS或者IMS，都支持两阶段提交协议。

356

CICS与IMS和DB2数据库管理系统的在交易中对完全的两阶段提交协议提供支持，并且在分布式的CICS系统中依然支持此协议。

当应用程序开发人员想开发新的、需要更新多个资源管理器中的资源(或者还跨越多个系统)的应用程序时，他们总面临着很多限制。许多新的应用程序使用了一些不支持两阶段提交协议的技术，比如DB2存储过程，EJB，和CICS或IMS的客户端附属工具。如果一个应用程序使用这些资源管理器中任何一个来更新资源，那么应用程序不一定可以做到全局的同步点协作。

出于以下几个原因，全局同步点协作的欠缺可能会影响一个应用程序的设计：

- 如果不是所有的资源管理器都支持两阶段提交协议，那么程序就不能处理复杂的和分布式的交易。
- 在跨越多个系统时，应用程序不能设计成一个单一的应用程序(或者恢复单元)，而CICS例外。

应用程序开发人员不得不在这种种限制之中进行开发。比如，程序员可以限制在哪里存放商务数据，以保证所有的数据都可以放在一个单一的恢复单元中进行提交。

一但有一个组件出现了问题，这些限制也可能会影响受保护的数据与其一致性的恢复能力。因为资源管理器根本无从知道应该去提交还是去回滚操作。

11.4 什么是 CICS?

CICS是客户信息控制系统(Customer Information Control System)的缩写，它是z/OS操作系统上通用的交易处理子系统。CICS提供了一些系列运行在线程序的服务，可以在同一时间接受许多用户的请求，使用同一个文件和同一段程序去运行同一个应用。

CICS以最快的响应速度管理共享的资源，数据的完整性和运行的优先级。CICS为用户授权，分配资源(实际存储和循环)，并且把来自应用的数据库访问请求传递给合适的数据库管理系统(如DB2)。我们可以说CICS在行为上和在许多功能上与z/OS操作系统非常相似。

357

一个CICS应用是一组用于完成一个业务操作的相关程序的集合，例如处理一个旅行的请求或者准备一个公司的工资表。CICS应用在CICS的控制下运行，使用CICS的服务和接口去访问程序和文件。

传统的CICS应用由提交一个交易请求来驱动。交易的执行包括了运行一个或多个应用程序，来完成所需的功能。在CICS文档中可以发现”CICS应用程序”有时仅仅被称为”程序”，而术语”交易”常含有由多个程序完成的流程的意思。

CICS程序也可以采用EJB的形式。可以从CICS信息中心里找到更多关于CICS环境下JAVA编程的信息。

11.4.1 z/OS 系统中的 CICS

在一个z/OS系统中，CICS提供了一个用于管理交易的功能层，而操作系统依然是与计算机硬件距离最近的接口。CICS根本上从操作系统中分离出了一种特定类型的的应用程序(即，在线程序)，并由它自己处理这些程序。

举例而言，当一个应用程序访问一个终端或者任何设备时，它并不直接地与它们通信。应用程序向CICS发送命令，并使用需要的访问方式与操作系统通信，最终这些访问方式会与终端或者设备进行通信。

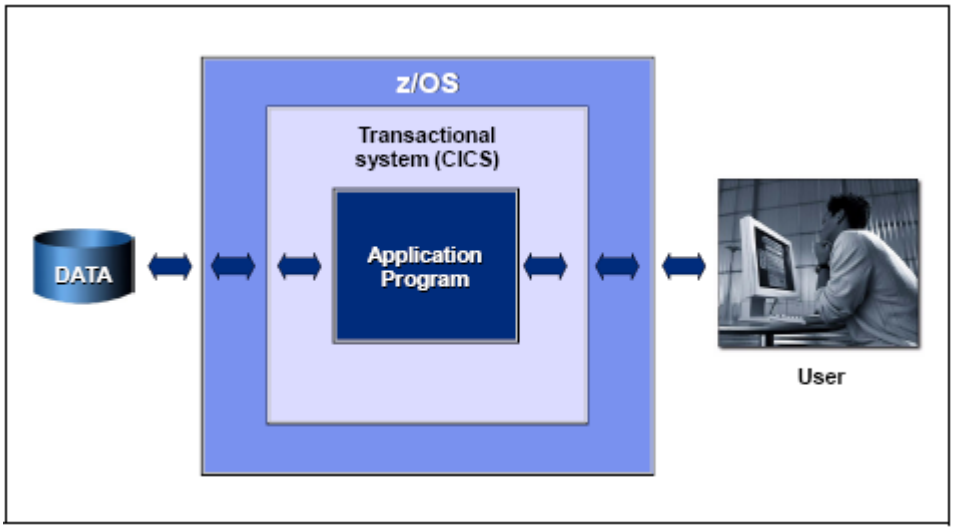


图11-4 交易系统和操作系统

一个z/OS系统上可能会有几份CICS的拷贝同时运行。每个CICS都在一个z/OS上独立的地址空间启动。CICS提供一个叫做多区域操作(multi-region operation，MRO)的选项。这将不同的CICS功能分离到不同的CICS区域中。所以一个(或多个)特定的CICS地址空间可能专门用于终端控制，并被命名为终端所有区域(terminal owning region，TOR)。其他可能的CICS区域还包括程序所有区域(application-owning region，AOR)和文件所有区域(file-owning region，FOR)。

11.4.2 CICS 程序，交易和任务

CICS允许您将应用的逻辑与资源分离。为了开发和运行CICS应用，需要理解CICS程序，交易和任务之间的关系。这些术语在有关CICS的书籍中和在许多命令中都会出现。

- 交易(transaction)

  交易是一个由单独的请求启动的业务流程处理。请求通常在一个终端由用户发出，但也可能由网页，远程工作站，在另一个CICS系统上的程序，或者在预先定义好的时间自动触发的程序发出。《CICS Internet Guide》和《CICS

External Interfaces Guide》描述了运行CICS交易的不同方式。

一个CICS交易用4个字符来命名，被定义在程序控制表(program control table，PCT)中。

359

- 应用程序(application program)

  在运行时，一个单个的交易由一个或多个执行处理的应用程序组成。

  但对于CICS而言，术语"交易"既可以被理解为单独的事件，也可以被理解为所有同一种类的交易。您可以使用交易资源定义(transaction resource definition)向CICS描述交易的类型。交易类型的定义中需要给出交易类型名(交易标识符或TRANSID)，并告诉CICS完成工作而需要的一些信息，比如最先调用的程序是哪一个，在执行交易时需要什么样的权限。

  向CICS提交TRANSID即可运行一个交易。CICS使用记录在TRANSACTION定义中的信息来建立正确的运行环境，并启动交易的第一个程序。

- 工作单元(unit of work)

工作单元
一个交易；
一个可恢复的完整操作。

  用来描述恢复单元的术语"交易"现在已经在IT业界得到广泛的使用，而CICS将UR称为工作单元(unit of work)。这是一个典型的可恢复的完整操作；如果程序命令或者系统出错，它可以作为一个整体地被提交或者被停止。在许多情况下，一个CICS交易的范围也就是一个单独的工作单元，但您应当意识到，在阅读非CICS的出版物时，这两个概念还是有差异的。

- 任务(task)

  您也可以看到"任务"这个词在CICS的书籍中被广泛使用。这个词在CICS中也有特定的含义。当CICS接受到一个运行交易的请求时，它启动一个与交易的运行实例相关的新的任务。即，为了一个特定终端的一个特定用户工作的，拥有一份特定数据的交易的执行。您也可以认为它接近一个线程(thread)。当交易完成时，任务随即结束。

11.4.3 使用编程语言

您可以使用COBOL，面向对象COBOL，C，C++，Java，PL/I，或者汇编语言来编写运行在z/OS平台上的CICS应用程序。大部分的处理逻辑可以使用标准的语句来写，但您需要使用CICS命令或者Java和C++类库来请求CICS服务。

大部分情况下，您使用CICS命令级别的编程接口，EXEC CICS。它是在使用COBOL，面向对象COBOL，C，C++，PL/I和汇编语言时用的。这些命令在*CICS Application Programming Reference*一书中有详细的介绍。

360

对于使用JCICS类库的Java编程，在CICS信息中心里的*Java Applications in CICS component*一书中有详细的介绍。

对于使用CICS C++类的C++编程，*在CICS C++ OO Class Libraries*一书中有详细的介绍。

11.4.4 会话和伪会话程序

会话交易
一个程序，执行和用户间的会话。

在CICS中，当被执行的程序进入到与用户交互的部分时，它被称作一个会话交易(*conversational transaction*)，参见362页图11-5。与之形成对比的是非会话交易(参见363页图11-6)，它只处理一个输入，响应并结束运行。非会话交易不能暂停下来去终端读取第二个输入，这也就是它之所以被称为非会话的原因。

由一系列非会话交易为用户提供一个会话交易的表象，这种CICS中的技术被称为伪会话处理。当用户在等待输入时根本没有交易存在；只有用户在发送输入时CICS才会去读取输入。362页图11-5和363页图11-6在一个银行账户记录更新的例子展示了不同种类的会话。

361

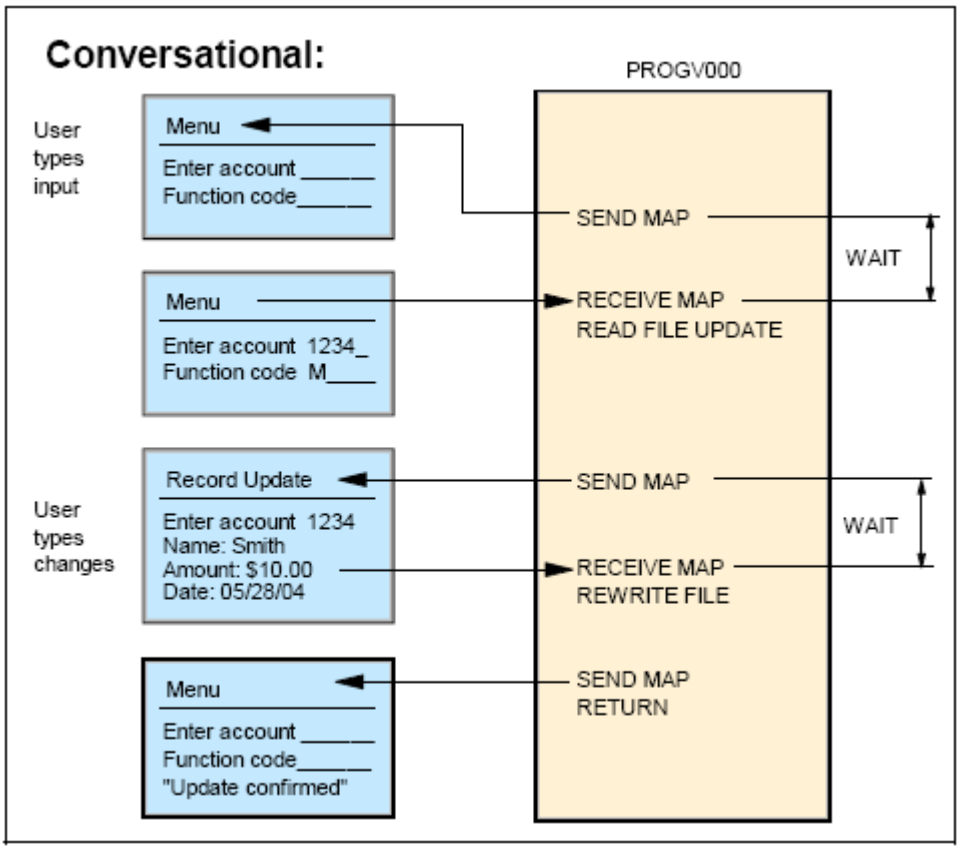


图11-5 会话交易的例子

在一个会话交易中，程序在等待接收数据时依然占用着资源。而在伪会话模型中，在这段等待的时间里并不占用资源(如363页图11-6)。

362

更多关于这部分的主题可以参见*CICS Application Programming Guide*。

伪会话
用一系列非会话交易为用户提供一个会话交易的表象。

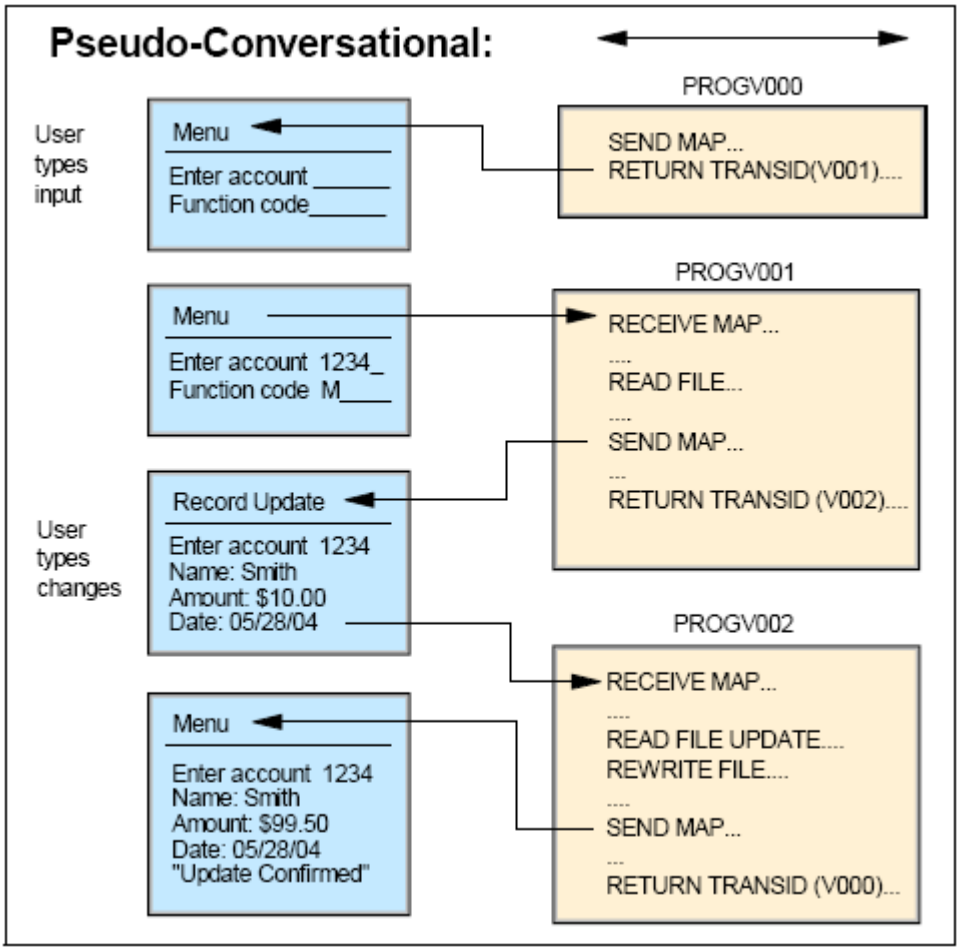


图11-6 伪会话交易的例子

11.4.5 CICS 编程命令

CICS命令的通常格式是EXECUTE CICS(或者EXEC CICS)，再加上命令名和一个或多个可能的选项。

您可以不用知道或者不考虑CICS控制块和存储区的内容，就直接使用CICS命令级的接口来编写应用程序。但您可能需要一些在您的应用程序本地运行环境之外的一些有效信息。

当您需要使用CICS系统的服务，例如，从文件中读取信息的时候，您只需要在您的代码中使用CICS命令即可。例如在COBOL中，CICS命令大致是这样的:

EXEC CICS function option option ... END-EXEC.

这里的**function**指的是您执行的操作。比如读文件就是READ，向终端输出就是SEND等等。

选项**(option)**指的是与**function**有关的一些规格补充。选项以关键字的形式表达。比如，与READ命令有关的选项包括FILE，RIDFLD，UPDATE等。FILE选项通常需要指定一个用于标识或指向文件名的值，用于告诉CICS您希望读取哪个文件。RIDFLD(record identification field，即键值)选项的值用于告诉CICS您需要哪些记录，同样也需要指定值。另一方面，UPDATE选项只是简单的表示您想要更改记录，它不需要指定值。因此，如果我们想要去读取并可能修改一条记录，它定义在CICS中的文件ACCTFIL，并使用存储在工作存储区中的键值ACCTC，我们可以使用如示例11-1中的命令：

示例 11-1 CICS命令示例

```
EXEC CICS
   READ FILE('ACCTFIL')
        RIDFLD(ACCTC) UPDATE ...
END-EXEC.
```

您可以使用ADDRESS和ASSIGN命令去访问这些信息。关于这些命令的编程信息，可以参考*CICS Application Programming Reference*一书。当使用ADDRESS和ASSIGN命令时，您可以读取各种域，但不应该以任何其他方式设置它或者使用它。这意味着您不应该将任何CICS域作为CICS命令的参数使用，因为这些域可能已经被EXEC接口模块改变了。

11.4.6 一个 CICS 交易是如何运行的

为了开始一个CICS的在线会话，用户通常需要从登录开始，这个过程使得CICS可以确认他们的身份。通过登录，用户获得了使用某些交易的权限。登录之后，用户就可以使用那些他们想要用的特定交易了。一个CICS交易通常由一个1至4个字符的交易标识符或TRANSID来标示。TRANSID定义在一个表中，这个表中还记录着该交易所使用的第一个程序。

应用程序存储在直接访问存储设备(DASD)上的一个库中，DASD和某处理器相关联。当系统启动时或者需要它们的时候被载入。如果一个程序在内存中但未被使用，CICS可以释放它所占据的空间。当再次需要这个程序时，CICS重新从库中载入一个它的拷贝。

在处理一个交易时，系统可能从几个终端上接受到消息。对于每条消息，CICS载入应用程序(如果还未被载入)并启动一个新的任务去执行它。因此，多个CICS任务可以并发的运行。

多线程(*Multithreading*)是允许应用程序的一份拷贝被多个交易同时处理的一种技术。例如，一个交易可能开始执行一个程序(一个旅行者请求得到信息)。与此同时，另一个交易可能执行这个程序的同一份拷贝(另一个旅行者请求得到信息)。试着将这种处理方式与单线程(*single-threading*)进行比较。所谓的单线程即指在另一个交易可以使用一个程序之前，该程序必须执行完毕。多线程要求所有的程序都是类似于可重入的，即它们在程序的入口和出口之间必须是可串行重复使用的。CICS命令则自动遵循这项规则执行CICS应用程序。

CICS维护着一个用于控制每个任务的独立的线程。例如，当一个任务在等待磁盘文件读取时，或者在等待终端响应时，CICS就能够把控制权交到另一个任务。任务由CICS的任务控制程序进行控制。

CICS同时管理着多任务和任务本身对服务的请求(关于操作系统服务或CICS的服务)。这使得在任务等待操作系统完成它自己请求的处理时，CICS还能够继续工作。当一个由CICS管理的交易拥有最高的优先级时，其他准备运行的交易就会将其对于处理器的控制权让出。

在您的应用程序运行的时候，它请求不同的CICS工具去处理在它和终端间消息的传递，必要的文件处理和数据库的访问。当应用结束的时候，CICS将控制权交还给终端并进入待机状态。图11-7，11-8和11-9将有助于您对此的理解。

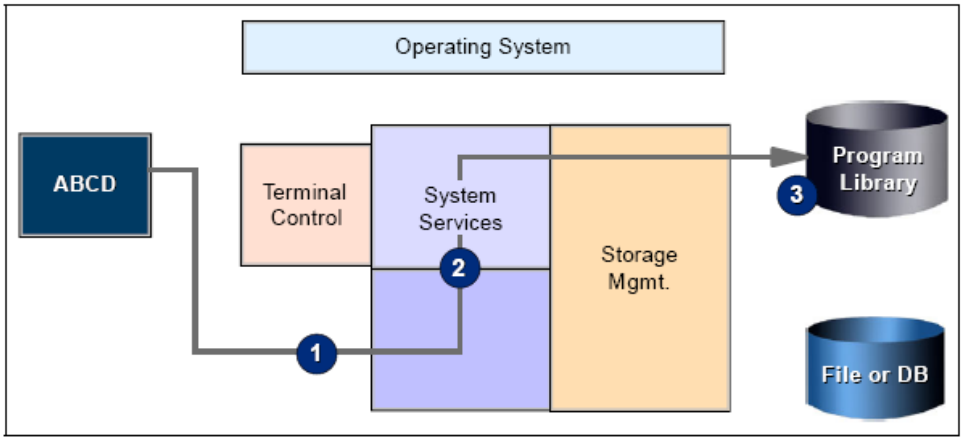
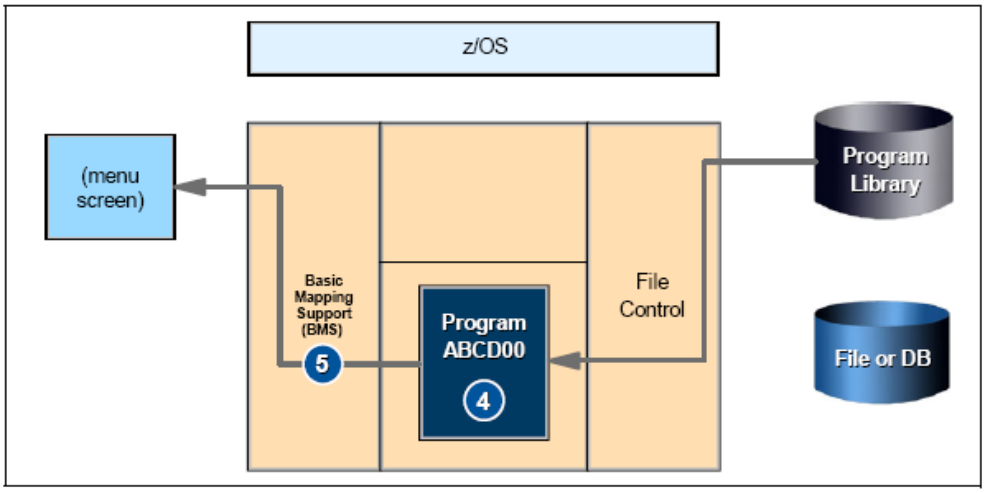


图11-7 CICS交易的处理流程(第一部分)

在一个交易(标记成ABCD)中控制的流程由序列1至8表示(我们只是用这个交易展示一些可能牵涉到的阶段)。这8个阶段的意义如下：

1. 终端控制器接收输入到终端上的字符ABCD，并将其放入工作存储区。
2. 系统服务将交易代号ABCD解释为对应用程序ABCD00的调用。如果终端用户有权限调用这个程序，则需要在内存中找到该程序，或者将该程序装载到内存中。
3. 可执行模块从程序库(*program library*)中被装载到工作存储区中。



366 *图11-8 CICS交易的处理流程(第二部分)*

4. 任务被创建，程序ABCD00被授予控制权。

5. ABCD00调用基本映像支持(Basic mapping support，BMS)和终端控制向终端发送一个菜单，允许用户精确描述他们需要什么样的信息。

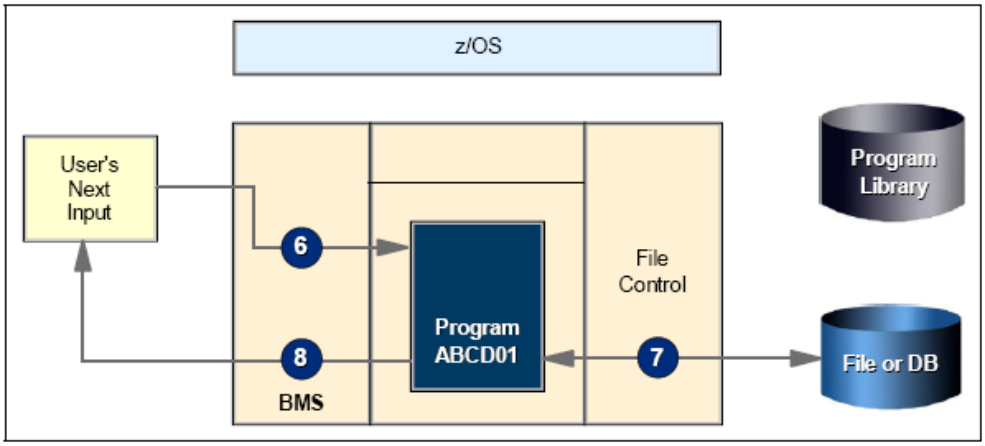


图 11-9 CICS交易的流程(第三部分)

6. BMS和终端控制也处理用户其后的输入并将其交给ABCD01(由ABCD00指定的，负责处理下一个终端响应的程序)。然后ABCD01调用文件控制。

7. 根据终端用户的要求，文件控制(*file control*)读取相应的文件。

8. 最后，ABCD01使用BMS和终端控制将取得的数据格式化并显示到终端上。

11.4.7 CICS 为应用程序提供的服务

CICS应用程序在CICS的控制之下运行，使用CICS服务和接口去访问程序和文件。

应用程序接口

您使用应用程序接口(API)从应用程序中去访问CICS服务。您可以向编写其他程序一样地编写CICS程序。大多数处理逻辑可以用标准语言进行编写，但您可以使用CICS命令去请求CICS服务。

367

终端控制服务

这些服务可以使CICS应用程序与终端设备通信。通过这些服务，可以将信息发送到终端屏幕，也可以从终端获取用户的输入。直接处理终端控制服务并不是一件简单的事。BMS使您可以从一个更高的语言级别去与终端通信。它可以格式化您的数据，您也不需要了解数据流的细节。

文件和数据库控制服务

我们区分以下两种CICS数据管理服务：

1. 无论该数据集是VSAM还是由基本直接访问方法(BDAM)进行管理，CICS文件控制为您提供访问数据集的服务。CICS文件控制允许您对VSAM或BDAM数据集进行数据读取、更新、添加和浏览，以及对VSAM数据集里的数据进行删除。
2. 数据库控制为您提供访问DL/I和DB2数据库的服务。尽管CICS为DL/I数据库提供了两种编程接口，我们推荐您使用更高级的EXEC DL/I接口。CICS为DB2提供了EXEC SQL编程接口。它为操纵表的集合提供了强有力的语句支持，从而减轻了应用程序逐条记录处理(如果是DL/I就是逐段处理)的负担。

其他CICS服务

- 任务控制能够用来控制一个任务的执行。您可以挂起一个任务或者通过顺序的可重用一个任务来调度资源的使用。一个任务的优先级也同样可以被改动。
- 程序控制管理着一个CICS系统中应用程序间的控制流。在程序控制命令中所提及的程序名必须已经在CICS中定义为一个程序。可以使用程序控制命令去连接您的应用程序中从一个到另一个，也可以在控制权不移交回请求程序的前提下将它从一个应用程序转交给另一个。
- 临时存储(Temporary Storage，TS)和临时数据(Transient Data，TD)的控制。CICS临时存储控制机制为程序员提供了将数据存储在临时存储队列中的能力。临时存储队列也许在主存中，也许在直接访问存储设备上的辅存中，或者在耦合设施中。CICS临时数据控制机制则提供了一种更广泛的队列机制，为了随后的或者外部的处理而排队(存储)数据。
- 间隔控制服务提供了与时间相关的一些功能。使用间隔控制命令，您可以在一个特定的时间或者在一段特定的间隔之后启动任务，推迟任务的执行，以及在超过特定的时间时获得通知。

368

- 存储控制设备控制对主存的申请，以提供为处理交易而需要的中间工作区和其他主存。CICS无需应用程序提出申请就为每个程序自动地提供内存，并提供其他关于中间存储(任务内部和任务之间)的设备。除了CICS自动提供的工作内存之外，您还可以使用其他CICS命令去获取和释放内存。
- 转储和跟踪控制(dump and trace control)，当一个应用程序运行时发生异常终止时，转储控制用以提供交易的转储。跟踪控制是一种应用程序员的辅助调试设施，可以在CICS操作顺序中插入跟踪断点进行调试。

11.4.8 程序控制

一个交易(任务)在完成工作的运行期间可能会执行好几个程序。

对于CICS系统中的任何应用程序，程序的定义中包含了每个程序的一个入口。每个入口记录着程序是用何种语言编写的。对于系统中每一个交易标识符，交易的定义都有一个入口。在该定义中重要的信息有交易的标识符和交易运行时所要执行的第一个程序。

以下说明这两组定义，交易和程序，是如何在一起进行工作：

- 用户在终端(或由先前的交易确定)上键入的某个交易标识符。
- CICS在安装好的交易定义表中寻找该标识符。
- 该定义告诉CICS需要调用的第一个程序。
- CICS在安装好的程序定义表中寻找该程序。找到它在哪里，然后装载它(如果它还不在内存中)。
- CICS利用以上这2组定义，为此特定的交易和终端组合创建必要的控制块。对命令级别的COBOL程序而言，这包括了为执行的程序创建一个工作空间的私有拷贝。
- CICS将控制权转交给程序。它使用该控制块开始运行。在交易运行期间如果必要的话，该程序可以将控制权交给另一个在安装好的程序定义表中可以找到的程序。

有两个CICS命令用于在程序之间传递控制权。其一是LINK命令，它与COBOL中的CALL语句相似。其二是XCTL(转移控制)命令，在COBOL没有与之对应的语句。当一个程序去连接另一个程序时，第一个程序依然驻留在内存中。当第二个程序(被连接的程序)执行完毕并放弃控制权时，第一个程序接着执行在LINK之后的语句。被连接的程序可以被认为比连接的程序低一个逻辑层次运行。

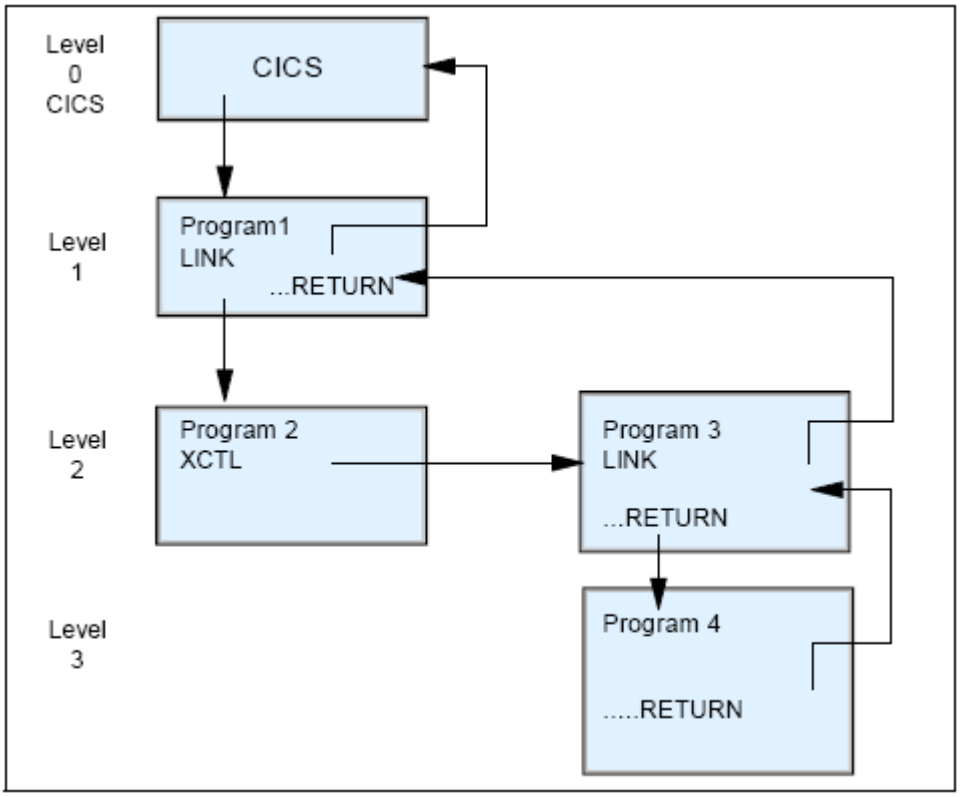


图11-10 在程序之间转移控制权(假设程序正常结束)

与之形成对比的是，当一个程序把控制移交给另一个程序时，第一个程序可以认为已经结束，而第二个程序在和第一个程序相同的层次上运行。当第二个程序结束时，控制权并不返回第一个程序，而返回给最后发出LINK命令的那个程序。

有些人喜欢把CICS本身当作这个过程中的最高层次程序，而把交易中的第一个程序当成下一个层次，依次类推。图11-10展示了这种概念。

370 LINK命令的用法大致如下：

```
EXEC CICS LINK PROGRAM(pgmname)
     COMMAREA(commarea) LENGTH(length) END-EXEC.
```

在这里pgmname表示您希望去连接的程序的名字。commarea表示包含着要传递的数据区域和/或接收返回数据的区域。COMMAREA接口也是一个调用CICS程序时的选项。

一个关于CICS应用设计的好的原则是将表现形式和业务逻辑分离；使用LINK命令可以使这些程序进行通信，而COMMAREA选项则可以使这些程序间可以相互传递数据。这种模块化设计的方式不仅带来了功能的分离，也在采用新的表现形式，将现有的应用程序进行Web使能化方面，提供了更大的灵活性。

11.4.9 CICS 编程路线图

对于那些使用EXEC CICS命令级别编程接口的CICS程序，典型的开发步骤如下：

1. 设计应用程序，标识您将使用到的CICS资源和服务。可以参考*CICS Application Programming Guide*一书的应用程序设计相关章节。
2. 使用您选择的语言编写程序，包括用EXEC CICS命令去请求CICS服务。关于CICS命令，可以参考*CICS Application Programming Reference*一书。

   对于在线程序而言一个必不可少的组件是屏幕定义，即数据显示到屏幕上时的布局(比如网页)。在CICS中我们把它叫做*map*。
3. 您可能只需要依靠编译器编译程序并在CICS中安装它，或者您需要为程序定义翻译器选项然后翻译并编译程序。具体内容可以参考*CICS Application Programming Guide*一书。
4. 使用PROGRAM和TRANSACTION资源定义在CICS中定义您的程序及相关交易。可以参考*CICS Resource Definition Guide*。
5. 定义任何在您的程序中使用的CICS资源，比如文件，队列或者终端。
6. 使用CEDA INSTALL命令在CICS中安装资源。可以参考*CICS Resource Definition Guide*一书。

371

11.4.10 我们的在线程序示例

重新回到我们在349页第11章”z/OS上的交易管理系统”中介绍的旅行社的例子。我们需要做的交易大致是：

- 添加，更新和/或者删除雇员的信息。
- 由汽车出租公司添加，更新和/或者删除可用的车辆。
- 从汽车出租公司获得可用的车的数量。
- 更新出租汽车的价格。
- 由航空公司添加，更新和/或者删除航班。
- 通过目的地或者航空公司获取已销售的票的数目。

图11-11展示了一个用户怎样能够计算部门的平均工资。用户输入部门而交易负责计算平均工资。

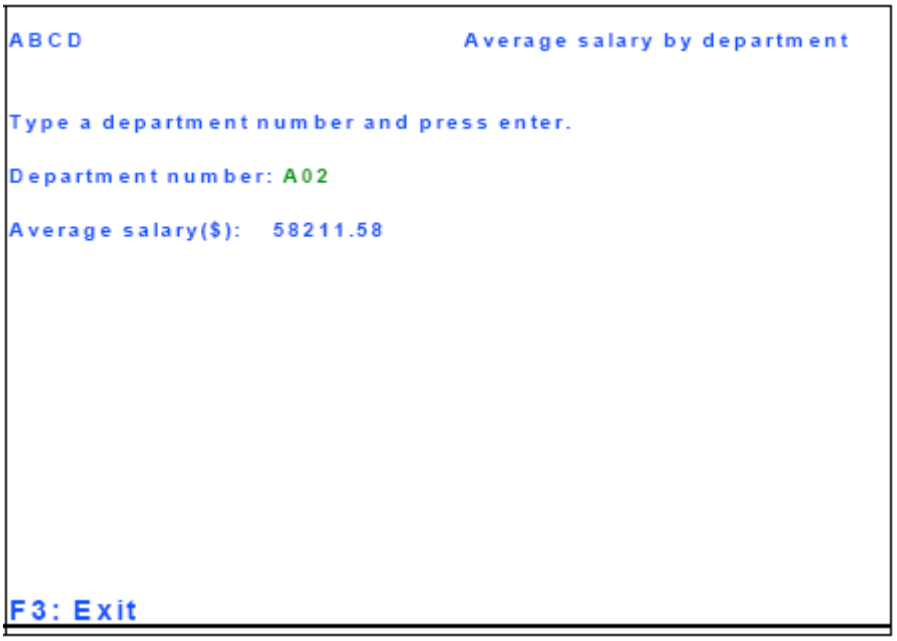


图11-11 CICS应用程序的用户界面

IMS
一个IBM 产品，支持层次型数据库，数据通信，交易处理，和数据库的拆除与复原。

请注意在您的CICS应用程序中可以为用户添加PF功能键的定义。

11.5 什么是 IMS?

IMS作为一个信息管理系统/360创建于1969年，在z/OS上它既是一个交易管理器，又是一个数据库管理器。IMS由三个部分组成：交易管理器(TM)，数据库管理器(DB)和向这两个组件提供公共服务的一系列的系统服务(如373页图11-12)。

372

随着近几年IMS的发展，为了适应商业需求，一些新的接口被添加。现在可以使用许多IMS组件的接口去访问IMS资源。

在本章中，我们将了解IMS的交易管理功能；我们将在381页第12章”z/OS上的数据库管理系统”中更详细地讨论它的数据库功能。

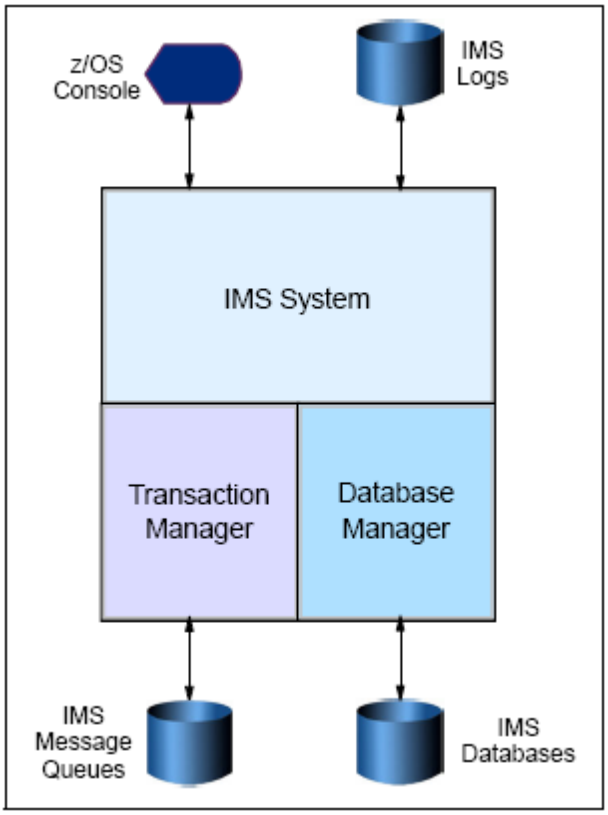


图11-12 IMS产品概览

您可以象编写其他程序一样地编写IMS程序。您可以使用COBOL，面向对象COBOL，C，C++，Java，PL/I，或者汇编语言来编写IMS应用程序。更多关于用Java来编写IMS程序的信息可以参考*IMS Java Guide and Reference*一书。 373

IMS交易管理器

IMS交易管理器为用户提供了一个访问在IMS中运行的应用程序的网络。用户可以是使用终端的人员，工作站或者是其他应用程序，它们可以在同一个z/OS上，在另一个z/OS上的，或者在非z/OS平台。

一个交易是一组触发了特定的业务应用程序执行的输入数据的设置。消息对于一个应用程序而言是预设好了的，而任何结果的返回可以被认为是一个交易。

IMS数据库管理器

IMS的数据库管理器组件提供了一个控制和访问数据的中心点。这些数据都是通过IMS应用处理的。它支持使用IMS层次数据库模型的数据库，并为应用程序提供访问数据库的服务。那些应用程序可以运行在IMS交易管理器之中，CICS交易监控器之中(现称为z/OS交易服务器)，也可以是批处理程序。

数据库管理器组件提供安全设施(撤销，复原)和维护数据库功能。它允许多任务(批处理和/或者在线)同时访问和更新数据，与此同时还保持着那些数据的完整。它也

提供通过重组或者重构来优化数据库的功能。IMS数据库在内部使用一些IMS数据库组织访问方法来组织。数据库的数据通过普通的操作系统访问方式存储在磁盘上。

我们将在381页第12章”z/OS上的数据库管理系统”更详细地介绍IMS的数据库管理组件。

IMS 系统服务

IMS的一些功能对于数据库管理器和交易管理器来说是共有的:

- 从错误中充启或者恢复IMS子系统。
- 安全：控制对IMS资源的访问。
- 管理应用程序：调度任务，装载应用程序，提供锁服务。
- 提供诊断信息和关于性能的信息。
- 提供操作IMS子系统的设施。
- 用IMS程序的接口向其他z/OS子系统提供接口。

11.5.1 z/OS 系统中的 IMS

IMS运行在z系列和更早期的S/390架构，或者与之兼容的主机和z/OS及更早版本的操作系统上。一个IMS子系统运行在一个z/OS系统的几个地址空间上。有一个负责控制的地址空间和几个相依赖的地址空间提供IMS服务与运行IMS应用程序。

由于历史的原因，有些描述IMS的文档使用术语区域(*region*)去描述z/OS上的地址空间。例如，IMS控制区域。在本书中，如果该术语是常用的，我们就使用它。您可以将区域和z/OS地址空间理解为一个概念。

为了充分利用z/OS的功能，IMS作了以下的工作：

- 运行在多个地址空间。IMS子系统(IMS/DB批处理程序和工具库除外)通常由一个控制区域地址空间，几个提供系统服务的依赖地址空间，和为应用程序而存在的依赖地址空间组成。
- 在每个地址空间运行多个任务。特别对于控制区域而言，IMS创建了许多子任务去履行不同的功能。这就使得当一个IMS子任务在等待系统服务时z/OS可以调度其他IMS子任务。
- 使用z/OS的跨内存服务来实现构成一个IMS子系统不同地址空间的通信。还使用z/OS的公共系统区域(CSA)来存储那些经常被IMS地址空间访问的IMS控制块，从而将使用多个地址空间的开销最小化。
- 使用z/OS子系统的特征。IMS将其动态注册为一个z/OS的子系统。这使得它可以使用一些系统工具。当相依赖的地址空间发生错误时可以检测出来。也可以防止删除相依赖的地址空间(并可以与其他子系统诸如DB2，WebSphere MQ进行交互)。
- 可以利用z/OS的系统耦合。多个IMS子系统可以在z/OS上同时运行形成系统耦合，并可以访问相同的IMS数据库。

11.5.2 IMS 交易管理器消息

IMS交易管理器的网络的输入和输出采用消息的形式，这些消息在IMS和物理终端或者网络上的应用程序间传送。这些消息被异步地处理(即当IMS接受到消息时，并不总是立即发送一个回复，尽管有时发生；IMS也会发送主动提供的消息)。

375

消息可能有4种类型：

- 交易—在这种消息中的数据是传送给IMS应用程序进行处理的。
- 传送给其他逻辑目的地(比如网络终端)的消息。
- 让IMS进行处理的命令。
- 让IMS APPC进行处理的消息。虽然IMS对于消息使用异步的通信协议，但APPC使用同步的协议(即，在消息发出后总是可以得到回复)。IMS TM对于APPC的接口对此必须做出特殊的处理以适应它。

如果IMS不能够立即处理一个输入消息，或者不能够立即发出一个输出消息，那么这条消息将被存储在IMS系统外部的一个消息队列中。IMS通常不会从消息队列中删除消息，除非它确认了消息已经被某个应用程序处理过，或者已经到达了它的目的地。

11.6 总结

在本章中我们学到，交易应用总是需要不断地改变，这取决于公司，客户，供应商等的需求。在其他时候，改变是通过新的技术完成的，而可靠的、健壮的应用程序保持不变。通过交易管理器的帮助，与计算机的交互可以以在线方式进行。许多不同的交易管理器和数据库管理器诞生了，但它们的原则是相同的。

CICS是一个交易处理子系统。这意味着它可以根据请求为您运行应用程序。它可以在同一个时间接受许多用户的请求去运行同一个应用程序，使用同一个文件，数据库或者程序。CICS以极快的响应速度管理着资源的共享，数据的完整和执行的优先级。传统地，CICS应用在接收到提交的交易请求时开始运行。交易的运行包括一个或者多个实现具体功能的应用程序的执行。

您可以像编写其他程序一样地编写CICS程序。您可以使用COBOL，C，C++，Java，PL/I或者汇编程序来编写CICS应用程序。大部分的处理逻辑可以使用标准语言的语句进行表达，但您也可以使用CICS命令。CICS命令根据它们的功能进行分类，包括终端交互，文件访问或者程序连接。大多数的CICS资源可以在线地通过CICS提供的交易来定义和改变。其他CICS提供的交易允许您监控CICS系统的情况。互联网的持续发展使得许多企业考虑将它们现有的系统变得可以让用户在互联网上进行使用。我们也简单地介绍了CICS应用网络支持的不同技术。

376

信息管理系统(IMS)由三个组件组成：交易管理器(TM)，数据库管理器(DB)，和为这两个组件提供公共服务的一系列系统服务。您可以像编写其他程序一样地编写IMS程序。您可以使用COBOL，C，C++，Java，PL/I或者汇编程序来编写IMS应用程序。

本章中的重要术语		
IMS TM	会话(conversational)	伪会话(pseudo-conversational)
IRLM	CICS TS	信息管理系统(Information Management System，IMS)
CICS命令(CICS command)	区域(region)	基本映像支持(basic mapping support，BMS)
多线程(multi-threading)	交易(transaction)	工作单元(unit of work)

11.7 复习题

为了帮助您检测对本章内容的理解程度，请完成下列问题：

1. 您经常使用的典型的在线交易有哪些？
2. 为什么对于在线交易处理而言多线程和多任务是那么的重要？
3. 在线交易系统的共同特征是什么？
4. 解释两阶段提交。
5. 描述CICS编程路线图的主要阶段。
6. 商业交易(business transaction)和CICS交易的含义有何不同？
7. 怎样在CICS中定义资源?
8. 什么是IMS的主要组件？他们的任务是什么？
9. IMS消息的四种类型是什么？

11.8 练习：编写一个 CICS 程序

请试着做本练习。

*CICS Application Programming Guide*一书将会对您完成本练习有所帮助。

分析和更新class程序

► 考虑COMMAREA的一个可能的应用。

思考在程序之间使用LINK或者XCTL传递数据。开发一个通用的用于错误处理的程序：所有的调用需要通过COMMAREA来传递错误数据。同样地，返回命令中的COMMAREA选项是被用来在一个伪会话顺序的几个连续交易之间传递数据。

当前资源的状态可以使用第一个交易通过COMMAREA来传递，这是为了通过第二个交易来比较当前状态。有时候在允许其进行更新之前可能需要了解资源的状态在上次交互之后是否被改变了。在Web应用中，一个CICS交易中的商业逻辑可以使用COMMAREA接口来进行调用。

- 几个对class程序的简单更新可以很容易的完成:
  - 在屏幕上增加一个额外的输出域，雇员佣金的最大值就可以是一个例子。

    必须先在map资源中必须定义一个新的域。可能有些常量字段需要改变。将map汇编并产生新的一份文件。修改程序在SQL语句中增加另一列，并在取得数据之后将其移到map的新的输出域中。为用户程序执行准备作业。CICS会话中需要新的程序和map的拷贝。

  - 创建一个与主菜单相似的交易。其中的一个选项用于启动当前的交易。

    在这个交易的map中只需要两个变量域：选项域与消息行。起初只需要一个用于启动交易ABCD的选项即可。新的map可以定义在同一个mapset中。修改交易ABCD，使其RETURN TRANSID到新的交易中。仅需要将以下的资源添加到CICS系统中即可：新的交易，程序(用户程序和map)。

378 - 学习CICS HANDLE CONDITION语句并找到可以在哪里去使用它。

    试着为RECEIVE CICS命令添加错误控制。当在发出RECEIVE命令后未能从终端得到可用的数据时，即会出现MAPFAIL的状况。

商业交易

分析一个典型的商业交易。思考用不同的CICS程序和交易去实现它。画一个图去展示处理的流程。

*CICS Application Programming Primer*里有一个例子可能很合适。一个百货公司主文件里存储着一些客户的帐户信息。应用程序需要做到以下几点：

- 显示客户的帐户记录。
- 添加新的帐户信记录。
- 更新或者删除现有的客户记录。
- 打印一份某客户帐号记录的信息。
- 通过客户的名字访问帐户记录。

379

12

第 12 章 z/OS 上的数据库管理系统

目标：为了处理主机上的在线工作负载，您需要很好的理解一些主要类型的系统软件。在本章中我们将集中介绍两种在z/OS上最常用的数据库管理系统(DBMS)产品：DB2和IMS DB。

在本章结束之后，您应当能够学会：

- 解释数据库在典型的在线交易中的应用。
- 描述大型系统的网络连接的两种模型。
- 解释DB2在在线交易处理中所扮演的角色。
- 列举常见的DB2数据结构。
- 在z/OS上构造简单的SQL查询。
- 概览DB2应用程序开发。
- 解释什么是IMS的组件。
- 描述IMS DB子系统的结构。

12.1 主机上的数据库管理系统

本章将会对基本的数据库(DB)概念作一个概览，介绍他们的作用是什么以及他们的优点是什么。现在有许多的数据库，但我们只局限范围在主机上最常用的两种数据库：层次数据库和关系数据库。

12.2 什么是数据库？

一个数据库为商务数据提供存储和控制。它是独立于一个或者多个应用程序的(但并不与处理过程的需求分离)。如果合理地设计与实现，数据库应该提供一个一致的商务数据的视图，以至于数据可以被集中的控制和管理。

一种用来描述数据集合的逻辑视图的方法是使用实体关系模型(entity relationship model)。数据库记录了特定条目(实体)的细节(属性)信息，以及不同种类实体之间的关系。例如，对于一个应用的仓库管理部分而言，应该有部件，购买订单，客户和客户订单这些实体。每个实体有自己的属性—如部件应该有部件号，名称，单位价格，单位数量等等。

在这些实体之间也存在着关系，例如一个客户应该与一个他所下的订单相关联，而订单应该与其所订购的部件相关联，等等。383页的图12-1展示了一个实体关系模型。

382

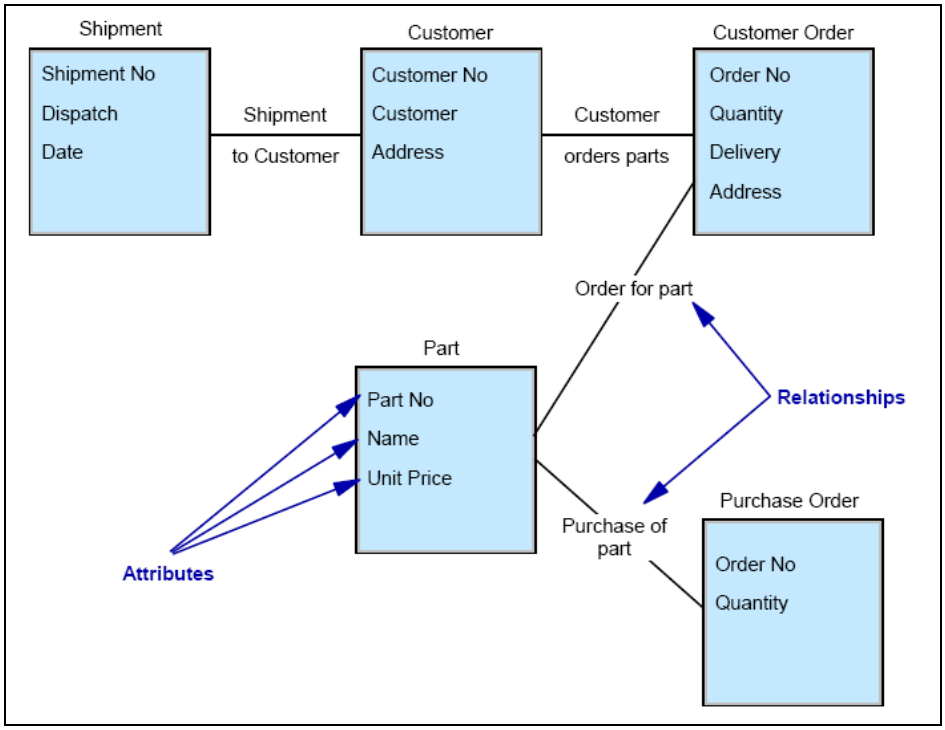


图12-1 实体，属性和关系

一个数据库管理系统(DBMS)，例如IMS数据库管理器组件(IMS/DB)或者DB2产品，提供了在数据库中存储和使用商务数据的方法。

12.3 为什么使用数据库？

DBMS
数据库管理系统，提供在数据库中存储和使用数据的方法。

当计算机系统最初开始发展的时候，数据存储在对于一个应用程序(或者甚至是单个应用程序的一个小部分)而言唯一的独立的一些文件内。但一个合理设计和实现的DBMS比起单纯的PDS文件系统来说有许多优点：

- 它减少了应用程序的编程工作量。
- 比起一个非DBMS的系统而言，它在创建，修改和访问数据方面效率更高。正如您所知道的那样，如果在文件中要增加新的数据元素时，那么所有使用该文件的程序(甚至是不需要使用该新数据元素的程序)都必须重写。而在使用DBMS时，这种情况就不会发生了。尽管程序员可以借助于技巧来将重新编写程序的工作量最小化，但依然是需要为其付出额外精力的。
- 相对于单纯的文件系统，它提供了一个更高的数据安全和保密级别。尤其是，当访问一个无格式文件中的一条逻辑记录时，应用程序可以看到所用的数据元素，甚至包括那些机密的或者本来需要更高访问权限才能看到的数据。为了尽量避免这种情况，许多用户将敏感的数据放入一个单独管理的文件中，当需要的时候将两个文件连在一起进行处理。而这又可能导致数据一致性方面的问题。

Segment(段)
任意的分区，保留区域，局部组件或者大型结构的片段。

如果使用DBMS，敏感的数据可以隔离在独立段(在IMS/DB中)或者视图(在DB2中)中以防止未被授权的应用程序去访问它。但是这些数据元素仍然是逻辑记录的完整的组成部分。

然而，同样的细节信息可能被存放在几个不同的地方；例如，客户的信息可能既用于订单程序中，又用于发票程序中。这导致了一些问题：

- 由于细节信息是分开地存放和处理的，应该是一样的信息(例如，客户的姓名和地址)可能在不同的程序中出现不一致。
- 当共同的信息必须做出改变时，它必须在几个地方进行改变。这导致了很大的工作量。如果数据的任意一个备份遗漏掉，就会导致上一条中所列的数据不完整问题。
- 没有数据的控制中心去保证数据的安全，这既可能导致数据的丢失，也可能导致未经授权的访问。
- 重复的数据会浪费存储介质的空间。

使用像IMS/DB或者DB2这样的数据库管理系统来实现数据库也会有一些额外的好处，DBMS可以做到：

- 允许多个任务同时去访问和更新数据，同时保证数据库的完整性。这在许许多多的用户通过在线程序访问数据时，显得格外重要。
- 提供了相关工具，使得应用程序可以去更新多条数据库的记录。即使在应用程序发生错误时，也能保证应用程序在多条记录间的数据依然保持一致。
- 可以将机密或者敏感的数据放在一个独立段(在IMS中)或者表(在DB2中)中。与之形成对比的是，在一个PDS或者VSAM文件中，程序可以访问逻辑记录的所有数据元素。而有些数据元素可能包含着应该被限制访问的数据。

- 提供控制和实现数据的备份和恢复的实用程序，防止重要商务数据的丢失。
384
- 提供用于监控和调整对数据访问的实用程序。
- 能够改变逻辑记录的结构(通过添加或移除数据域)。这些变更通常会需要那些使用VSAM或者PDS的应用程序进行重新编译或者重新汇编，即使这些程序并不需要使用那些增加的或者被修改过的域。而一个合理设计的数据库能避免应用程序员发生这种情况。

但请记住，仅仅依靠使用数据库和数据库管理系统本身，并不能体现上述的优越性。还需要对数据库有合理的设计和管理，并且要有良好的应用程序开发，才能体现出上述优越性。

12.4 谁是数据库管理员？

数据库管理员(DBA)为子系统中某个指定的数据库负主要的责任。在有些公司里，DBA被给予特殊的组权限，SYSADM，这使得他们几乎可以在DB2子系统中作任何事情，也使他们可以管辖子系统中所有的数据库。而在另一些地方，DBA的权限被限制在单个的数据库中。

DBA创建了数据对象的层次结构，由创建数据库开始，然后创建表空间，表，以及所需要的任何索引或者视图。他们也负责定义参照完整性和建立任何必要的约束。

DBA从本质上实现了数据库的物理设计。这些工作牵涉到空间的计算，决定用于表空间和索引空间的物理数据集的大小，以及分配存储组(也称为storgroups)。

有许多工具能帮助DBA完成这些任务。举例而言，DB2提供了管理员工具和DB2评估器(Estimator)。如果对象大小增长，DBA能够修改某些对象以应对改变。

DBA能够负责为数据库对象进行授权，尽管有时由一个特别的安全管理组去做这件事。

数据的中心化与对数据访问的控制是数据库管理系统与生俱来的能力。中心化的一个好处就是可以对多个应用提供一致的数据。作为结果，它要求对数据及其用法有严格的控制。

DBA要为精确的控制实现负责。事实上，为了使集中数据库的优势最大化，您必须对数据库进行集中控制。由于DBA工作的实际执行是依赖于公司的组织部门的，这里我们仅仅讨论DBA的角色和责任。能够胜任DBA的人员需要有应用程序开发和系统编程的经验。
385

在典型的系统中，DBA需要为以下几点负责：

- 提供数据库及其使用的规范和管理
- 引导，复查和批准新的数据库的设计
- 决定数据访问的规则与监控其安全性
- 确保数据库的完整性和可用性，并监控必要的重组备份和恢复活动
- 基于测试数据的结果，批准在现有的数据库上运行新的程序

总体来说，DBA负责维护数据库中数据的当前信息。起初，这种责任是依靠手动的方法来履行的。但由于这种管理范围的扩大和难度的加深，数据字典程序的使用就变得合理和有必要了。

DBA并不为数据库中的具体内容负责。这是用户的责任。更准确地说，DBA负责执行程序以保证准确，完整和及时地更新数据库。

12.5 怎样设计一个数据库？

对数据库设计过程用最简单的描述就是，为了不同的应用程序而组织数据元素。为了达到以下目的：

- 在现在和可以预见到的将来，数据元素要做好能被不同的应用程序使用的准备。
- 数据元素的存储是高效的。
- 对于那些有特定安全性需要的数据元素，需要制定访问控制。

386

多年来，出现了多种不同的数据库模型(例如层次型，关系型或对象型)，以至于无法找到统一的词汇，用于描述这些概念。

12.5.1 实体

一个数据库包含了关于实体的信息。一个实体(entity)有以下几个特点：

- 能够被唯一地定义。
- 无论是现在还是将来我们都可以从中获取实质的信息。

在实际中，这些定义是受到正在考虑中的应用程序与业务的上下文限制的。实体的例子有部件，工程，订单，客户，卡车等。很清楚的是在数据库的设计过程中定义实体是一个主要的步骤。我们在数据库中存放的有关于实体的信息是通过数据属性来描述的。

12.5.2 数据属性

数据属性(data attribute)是一个信息单元，描述了实体的一个事实细节(fact)。例如，以"部件"这一实体而言，Name=Washer，Color=Green和Weight=143是关于部件的三个事实细节，这样它们就是三个数据属性。

一个数据属性对应着一个名字和值。数据属性的名字描述着记录着什么类型的事实细节；值则是事实细节本身。在这个例子中Name，Color和Weight就是属性的名字，而Washer，Green和143就是属性的值。一个值必须与一个名字相关联，这样它才是有意义的。

一个发生值(occurrence)就是一个特定实体中的一个数据属性的值。一个属性总是

依赖于一个实体。它本身是没有意义的。取决于其用法，一个实体可以用一个或多个数据属性来描述。在理想情况下，一个实体应该由一个数据属性来唯一的定义，例如，一份订单用其订单号来唯一定义。这样的属性被称为实体的键(key)。键作为特定实体值的标识来使用，是实体中一个特殊的属性。键并不总是唯一的。具有相同键值的实体被称为同义实体(synonym)。

举例而言，某人的全名通常不是一个唯一标识。在这种情况下，我们不得不依靠其他属性如完整的地址，生日或者随机的序列号来标识实体。一个更常用的办法是定义一个新的属性来作为唯一键使用，例如，员工号。

12.5.3 实体间的关系

标识的实体之间也需要连接，这被称为关系(relationship)。例如，一个订单可能是针对一些部件的。同样，这些关系仅仅在应用程序和商务环境下才有意义。关系可以是一对一的(即一个实体的一个发生值只对应另一个实体中的一个发生值)，一对多的(即一个实体的发生值对应到另一个实体中的多个发生值)或者是多对多的(即一个实体的多个发生值对应到另一个实体中的多个发生值)。

387

关系可能是回归的，即一个实体的某发生值可能与本实体的其他发生值有关系。例如一个部件，比如紧固件，就可能包含其他几个部件：螺栓，螺母和垫圈。

12.5.4 应用程序功能

数据本身并不是数据库管理系统的终极目标。重要的是应用程序对数据的处理。用最小的应用程序单元去替代用户与数据库的交互行为是最好的用来模拟处理流程的办法。例如，一个单独的订单，一个部件的库存状况。在以后的章节中我们将其叫做应用程序功能(application function)。

功能是由应用程序来操作的。在一个批处理系统中，大量的功能累积到单个的程序中(即，一天的所有的订单)，然后由所需要的一个应用程序去做数据库的处理。而在一个在线系统中，一个单个的程序中也许只有一个或两个功能，以提供一次与用户的交互。

尽管功能总是有区别的，即使在批处理中许多人更喜欢讨论程序而不是功能。然而对功能的清晰理解是优良设计所必须的，特别是在DB环境中。一旦您确定了应用程序的功能需求，您才能决定怎样用CICS或者IMS程序去最好地实现它。在某些方面，功能可以理解为一个特别的用户对应用程序的单独使用。正因为如此，它是DB系统所关注的焦点。

12.5.5 访问路径

每个功能需要从其输入中获得某种标识，来识别相关的实体(例如，访问部件数据库时需要得到部件编号)。这就是功能的访问路径。通常来说，功能需要随机访问，

尽管有时出于性能原因使用顺序访问。尤其是在功能是批处理时，并且相对数据库大小来说其数量众多时，或者需要访问大多数数据库记录的情况下，都需要使用顺序访问。为了提高随机访问的效率，每个访问路径都应该使用实体的键。

12.6 什么是数据库管理系统？

一个数据库管理系统(DBMS)只不过是一个计算机化的数据保存系统。它为用户提供了一系列的工具，使用户可以操作数据库中的数据或者管理数据库本身的结构。数据库管理系统按照它们的数据结构或者类型来分类。

在主机z/OS操作系统上有以下几种数据库类型可以使用：反向列表，层次的，网络的，关系的。

当一个应用程序需要的数据的数据结构(不是数据值)是相对静态时，在主机平台上倾向于使用一个层次型的模型。例如，一个物料清单(Bill of Material，BOM)的数据库结构总是有一个高层次的装配部件号，以及几个层次的拥有子组件的组件。该结构通常会有一个组件预报，成本和价格等等的数据。BOM应用程序的数据结构极少变化，也很少加入新的数据元素(不是数据值)。应用通常从顶端的装配部件号开始，并向下访问到各个细节组件。

Root(根)
一个层次结构的最高层。

两种数据库系统的优点已经在383页第12.3节"为什么使用数据库？"中列出。关系数据库管理系统(RDBMS)比起层次数据库管理系统而言有其额外的显著的优势，即非导航性(non-navigational)。所谓导航性(navigational)，即指在层次数据库中，应用程序的开发人员必须了解数据库的结构。程序必须包含特定的逻辑，从根节点开始，去访问存有某个预期属性值或者元素值的子节点。程序还必须访问这中间介入的所有节点，即使并不需要用到他们。

在本部分的余下部分将讨论关系数据库的结构。

12.6.1 在关系数据库中存在哪些结构？

关系数据库中包含以下结构：

- 数据库

  一个数据库是数据的一个逻辑分组。它包含了一组相关联的表空间和索引空间的集合。典型意义上，一个数据库包含了一个应用程序或者一组相关应用程序所用到的所有数据。例如，您可以有一个工资数据库或者一个库存数据库。

- 表

  一个表是一个由行与列组成的逻辑结构。行没有固定的顺序，所以当您在索取数据时有可能需要为数据排序。而列的顺序就是数据库管理员在建表时所指定的顺序。每个行与列交汇点的特定数据项被称为值，或者更精确地被称为原子值。一个表的命名由表的拥有者的ID加上表名所构成，例如

TEST.DEPT或者PROD.DEPT。数据库中存在以下三种表：

SQL (Structured Query Language) 结构化查询语言
一种用来查询和处理关系数据库中数据的语言。

- 被创建并用来存放持久数据的基本表
- 用来存放中间查询结果的临时表
- 查询表时所返回的结果表

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00
B01	PLANNING	000020	A00
C01	INFORMATION CENTER	000030	A00
D01	DEVELOPMENT CENTER		A00
E01	SUPPORT SERVICES	000050	A00
D11	MANUFACTURING SYSTEMS	000060	D01
D21	ADMINISTRATION SYSTEMS	000070	D01
E11	OPERATIONS	000090	E01
E21	SOFTWARE SUPPORT	000100	E01

图12-2 DB2表示例(部门表)

在这张表中我们使用了：

- 列：列的有序集是DEPTNO，DEPTNAME，MGRNO和ADMRDEPT。在一个给定列中的所有数据必须具有相同的数据类型。
- 行：每个行包含着关于单个部门的数据。
- 值：行与列的交汇处就是值。例如，PLANNING就是在部门B01行中的DEPTNAME列的值。

► 索引

一个索引就是一组对于一个表中的行的引用的有序集合。与表中的行并不排序这一特点不同，在DB2中索引总是有序的。使用索引是为了达到以下两个目的：

- 为了性能。索引可以使我们更快地获取数据
- 为了记录的唯一性

通过为雇员的名字创建索引，可以更快地从表中获取雇员的数据，而不是整个数据库扫描一遍。同样地，通过为员工编号创建唯一键索引，DB2会保证每个员工编号的唯一性。唯一索引是DB2保证记录唯一性的唯一方式。在创建索引的同时系统会自动地创建索引空间(index space)，它是包含索引的数据集。

390

► 键

键是在创建表或索引，或者在定义完整性关系时所被指定的一列或者多列。

- 主键

  一张表只能拥有一个主键，因为它定义了实体。一个主键要满足两个要求：

1．它必须有一个值，即值不能为空。

2．它必须是唯一的，即在该列上必须已经定义了唯一键索引。

- 唯一键

  我们已经知道主键必须是唯一的，但对于一个表来说可以有多个唯一键。在EMP表(见577页"雇员表(EMP)")的例子中，员工号被定义为主键并因此是唯一的。如果表中还有一个社会保险号，并希望这个值也是唯一的。为了保证这点，可以在社会保险号这一列上添加唯一键索引。

- 外键

  外键是一个在参照完整性约束中指定的键，它的存在依赖于另一个表的主键或者唯一键(父键)。

举一个例子是雇员的工作部门号关联到在DEPT表中定义为主键的部门号。该约束是表定义的一部分。

12.7 什么是 DB2？

在第11章已经讨论过了关系数据库管理系统(RDBMS)的基本概念，参见349页"z/OS的交易管理系统"。在本节中的大部分的表示例可以在附录B中找到，参见575页"DB2范例表"。这些表，如EMP和DEPT，是所有平台上DB2产品示例数据库中的一部分。在抓屏图片中使用的是DB2第8个版本。因此，表的拥有者是DSN8810。

DB2管理的元素可以被分成两类：其一是用来管理用户数据的数据结构，其二是受到DB2控制的系统结构。数据结构可被进一步地分为基本结构(basic structure)和模式结构(schema structure)。对于主机平台DB2而言，模式结构是相当新的对象，它是为了与DB2产品家族相兼容才在主机上推出的。模式(schema)就是这些新对象的逻辑组。

391

12.7.1 DB2 中的数据结构

在本章中前面的部分讨论了对DBRM而言通用的大多数基本结构。现在开始讨论几种DB2中特有的结构。

视图

View(视图)
为了在一张表中控制什么用户可以访问什么而提供的一种访问表数据的方式。

视图(view)是另一种察看一张或多张表中数据的方式。它就像在一个幻灯片上放一个覆盖物，以允许人们只能看到该幻灯片的一部分内容。例如，您可以在部门表上创建一个视图，使得用户只能去访问特定部门并更新其工资信息。您不希望让用户看到其他部门的工资信息。您创建的表的视图使用户只能去访问一个部门，而用户像使用表一样地使用该视图。因此，视图的使用是出于安全性的原因。大多数公司并不允许用户直接访问表，而是利用一个视图来完成它。用户通过视图来访问。对于缺乏经验的用户而言，视图可以被用来简化复杂的查询。

表空间

表只是一个逻辑上的构建。它被保存在一个被称为表空间(table space)的实际物理数据集中。表空间是一种可以包含一张或多张表的存储结构。表空间的命名由数据库的名字加上表空间的名字来构成的，比如PAYROLL.ACCNT_RECV。有三种类型的表空间：简单的，分段的和分区的。可以参考DB2 UDB for z/OS:SQL Reference以获取更多的信息。

DB2使用VSAM数据集。即，每个段是一个VSAM数据集。

索引空间

索引空间(index space)是包含单个索引的另外一种存储结构。事实上，当创建一个索引时，DB2会自动地创建一个索引空间。

存储组

392

存储组(storage group)由磁盘(DASD)上的一组卷组成，这些卷上的数据集里保存了表和索引。

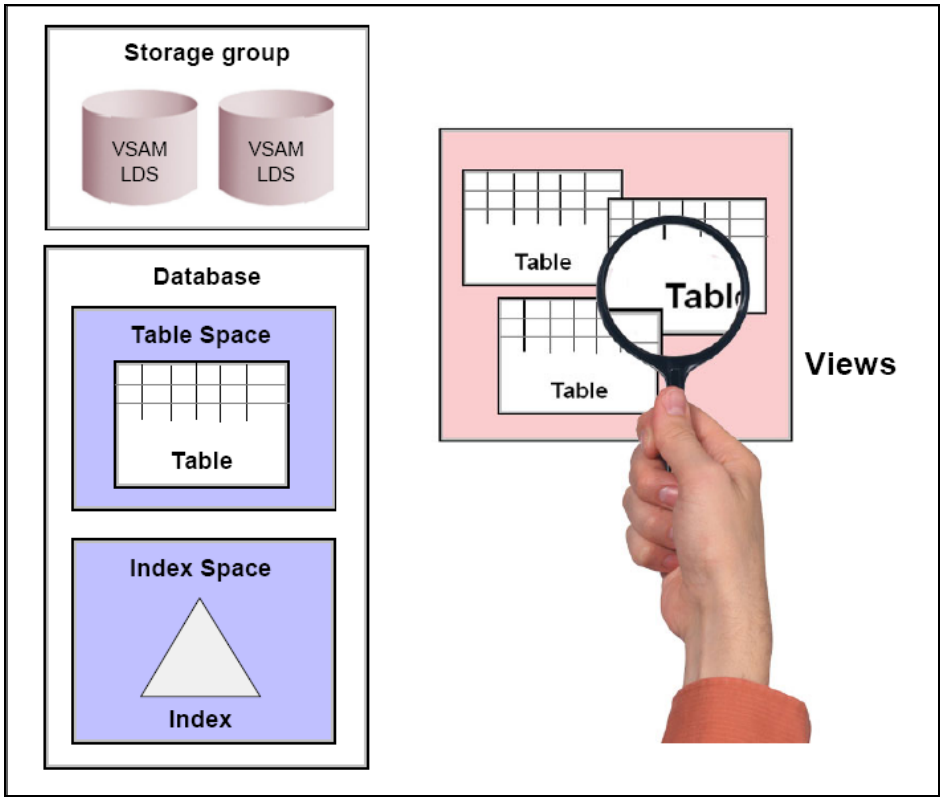


图12-3 DB2子系统中对象的层次模型

12.7.2 模式结构

用户定义数据类型 (UDT)

UDT(user-defined data type)是一种用户在通常的字符和数字类型之上定义自己的数据类型的方式。然而，UDT是基于已经存在的DB2数据类型的。如果您要处理国际货币，很有可能想要区分不同种类的币种。通过UDT，您可以基于‘decimal’数据类型定义欧元(EURO)数据类型，将它作为一种不同的数据类型与日元(YEN)或美元(US_DOLLAR)区分开来。作为结果，将不能将日元与欧元相加，因为他们是不同的数据类型。

用户定义函数 (UDF)

UDF(user-defined function)可以简单地定义在已经存在的DB2函数之上，例如四舍五入函数或者求平均数函数，或者也可以更加复杂编写成一个应用程序可供SQL语句访问。在前面的国际货币的例子中，我们可以通过使用UDF来进行货币之间的转换以便进行算术计算。

触发器

触发器(trigger)定义了一组行为的集合，这些行为在对某张特定的表进行添加，更新或者删除时执行。例如，当每次向EMP表中添加员工的时候，您或许也想将公司统计数据表中雇员的人数增加1。这时就可以定义一个触发器，在向EMP表插入的时候触发它。该触发器将自动在COMPANY_STATS表中正确的列上加1。

大型对象

LOB是DB2用来管理未结构化数据的数据类型。LOB有如下三种类型：

- 二进制大型对象(Binary Large Object，BLOB)，用于照片，图像，音频和视频剪辑。
- 字符大型对象(Character Large Object，CLOB)，用于大型的文本文档。
- 双字节大型对象(Double Byte Character Large Objects，DBCLOB)，用于需要使用双字节字符(如日本文字)的大型文本文档。

LOB存储在特殊的辅助表中，而这些表使用一个特殊的LOB表空间。在EMP基本表中，有可能包括一些关于雇员简历一类的文本材料。由于这是大量的数据，它被存储在它自己的表中。EMP表中可以定义一个CLOB列，保存指向那张特殊的LOB辅助表的指针，辅助表存储于LOB表空间中。每个定义为LOB的列拥有与它相关的辅助表和LOB表空间。

存储过程

存储过程(stored procedure)是很典型的存储和运行在服务器上的用户所编写的程序(也可以为了实现本地目标而运行)。存储过程为”客户端/服务器”环境而特别设计，在此环境中客户端只需向服务器发送一个调用的请求，服务器即可运行存储过程来访问DB2中的数据并返回结果。这省却了通过几个网络调用去运行几个单

独的数据库查询所带来的昂贵代价。

394 可以将存储过程理解为可以被调用的，用于执行一系列相关操作的子程序。它是一个在DB2中定义的，受到DB2管理的应用程序。

系统结构

编目和目录

DB2自身维护着一系列的表，这些表中包含着元数据或关于子系统中所有DB2对象的数据。编目(catalog)保存着所有对象的信息，例如表，视图，索引，表空间等等，而目录(directory)保存着应用程序的相关信息。编目可以通过查询来看对象的信息，而目录却不能。

当创建一张用户表时，DB2自动地记录表名，创建者，表空间和数据库到编目表中，该表称为SYSIBM.SYSTABLES。而表中所定义的所有列会被自动地记录在SYSIBM.SYSCOLUMNS表中。

此外，为了记录表的拥有者对表有授权，DB2会自动地在SYSIBM.SYSTABAUTH表中增加一条记录。在表中创建的任何索引会被记录在SYSIBM.SYSINDEXES表中。

缓冲池

缓冲池(Buffer pool)是DB2用来临时地存放表空间或索引空间页的虚拟存储区域。它们是DB2与数据驻留的物理存储设备间的缓存区。一个数据页从磁盘上被取出放置到缓冲池中的页上。如果需要的数据已经存在于缓冲池内，则可以节省一次昂贵的I/O访问。

活动日志和归档日志

DB2将所有改变数据的行为和其他重要的事件记录在一个日志中(log)。这些信息用于在出现错误时恢复数据，或者用于将数据回滚到前一个同步点。DB2将每个日志记录写到一个叫做活动日志(active log)的数据集中。

当活动日志写满的时候，DB2将其中的内容拷贝到一个磁盘或磁带上的叫做归档日志(archive log)的数据集中。一个引导数据集会追踪这些活动日志和归档日志。在系统恢复时，为了重启系统或者完成任何需要读取日志的活动，DB2会访问这些信息，。一个引导数据集允许进行同步点恢复。

12.7.3 DB2 地址空间

DB2作为一个多地址空间的子系统，至少需要三个地址空间：

- 系统服务
- 数据库服务

395

► 锁管理服务(IRLM)

此外，分布式数据设施(Distributed Data Facility，DDF)被用在DB2子系统之间进行通信。图12-4展示了这些地址空间。

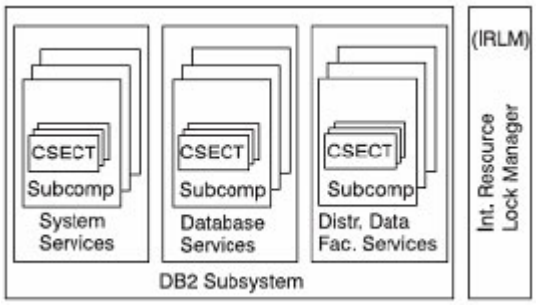


图12-4 DB2至少需要的地址空间

12.7.4 使用 DB2 实用程序

在z/OS上，DBA通过一系列实用程序(utilities)和程序来维护数据库对象，这些实用程序通过JCL作业提交。通常一个公司有一个数据集库，来存放这些DBA所拷贝和使用的JCL。然而DB2也提供了一些工具来生成这些JCL，例如管理工具(AdministrationTool)和可以在DB2I面板中找到的实用程序选项。

实用程序帮助DBA完成他们的工作。实用程序可以分为以下几类：

► 数据组织实用程序

在创建表之后，DBA可以使用LOAD实用程序压缩数量巨大的数据并向表中导入它们。另外，UNLOAD实用程序和DSNTIAUL汇编程序可以帮助DBA在子系统之间移动或者复制数据。

REORG实用程序可以使数据按照一定的顺序存放。随后的插入或者导入可能会打乱这种顺序，DBA必须根据RUNSTATS实用程序所产生的报告来继续使用REORG，RUNSTATS实用程序提供了关于统计和性能相关的信息。甚至可以对系统编目运行RUNSTATS。

► 备份与恢复实用程序

为了恢复数据，对DBA而言，使用COPY实用程序对数据和索引进行镜像备份是极其关键的。DBA可以使用完全备份或者增量备份(仅仅针对数据)。由于只有完全备份才能进行恢复，可以使用MERGECOPY实用程序将多个增量备份和一个完全备份合在一起。RECOVER实用程序可以将数据库恢复回到备份的状态。更多的情况下，数据库恢复回到一个镜像备份，再根据记录在日志中所有的数据改变，将数据库向前恢复到当前状态。不需要镜像备份，可以使用REBUILD INDEX工具创建索引。

► 数据一致性实用程序

CHECK实用程序是几个重要的数据一致性实用程序之一，可以用来检查和帮助修正参考完整性和约束的不一致。特别是在一次增加的数据导入或者一次

恢复之后。

一个典型的使用实用程序的例子是：运行RUNSTATS，然后运行EXPLAIN，接着再次运行RUNSTATS。

12.7.5 使用 DB2 命令

系统管理员和DBA都使用DB2命令来监视子系统。DB2I面板和管理员工具为您提供使用这些命令的简易方法。'-DISPLAY DATABASE'命令会显示出一个数据库中所有的表空间和索引空间的状态。例如，在没有镜像备份时，您的表可以处于"拷贝挂起"状态，需要您进一步运行COPY实用程序。还有其他的一些显示命令，例如DISPLAY UTILITY可以显示出实用程序作业运行的状态，或者也可以去显示缓冲池，线程和日志的信息。

您也可以在TSO会话或者批处理作业中发送一些DSN命令。然而使用DB2I面板中的选项，BIND，DCLGEN，RUN等等来执行这些命令更为简便。(在有些公司，DBA负责数据库的绑定，尽管在通常情况下由程序员在编译的时候进行绑定。)

12.8 什么是 SQL?

结构化查询语言(Structured Query Language，SQL)是一种高级的语言，用户可以使用它来指定所需要的信息，而不需要关心数据是怎样被取得的。数据库负责生成访问路径以获取数据。SQL在一个集合的层面上进行工作，这意味着它被设计为可以获取一条或者多条记录。本质上，它用于从一张或多张表中取得数据，并以一张结果表的形式返回结果。

基于功能，SQL可以被分为以下三类：

- DML—数据操作语言(Data manipulation language)，用于读取和修改数据。
- DDL—数据定义语言(Data definition language)，用于定义，改变或者移除DB2对象。
- DCL—数据控制语言(Data control language)，用于授予或者回收权限。

397

可以使用一些工具来输入和执行SQL语句。在这里我们重点介绍SPUFI，它代表的是使用文件输入执行SQL(SQL Processing Using File Input)。SPUFI是DB2交互(DB2 Interactive，DB2I)菜单面板中的一部分。当DB2安装之后您可以从ISPF面板中选用它。(当然，它取决于系统管理人员是如何设置系统菜单面板的。)

SPUFI是数据库管理员最常用的工具。它允许一次编写和保存一条或者多条SQL语句。DBA使用它来授予或者回收权限；当情况紧急的时候甚至还可以使用它来创建对象。开发人员也经常使用SPUFI来测试他们的查询。这样他们就可以确保查询返回的是他们想要的。

另一个您可能会在主机上使用到的工具是查询管理工具(Query Management Facility，QMF)，它只允许一次编写和保存一条SQL语句。QMF的主要优势在于

它的报表功能[1]。它允许您设计灵活的和可重用的报表格式，包括图像。此外，它提供了提示查询的功能，用来帮助不熟悉SQL的用户去构建简单的SQL语句。另一个是管理工具，它兼有SPUFI和构建查询的功能。

398

图12-5演示了怎样使用SPUFI输入SQL语句。在DB2I面板上，它是第一个选项。请注意这个DB2子系统的名字是DB8H。

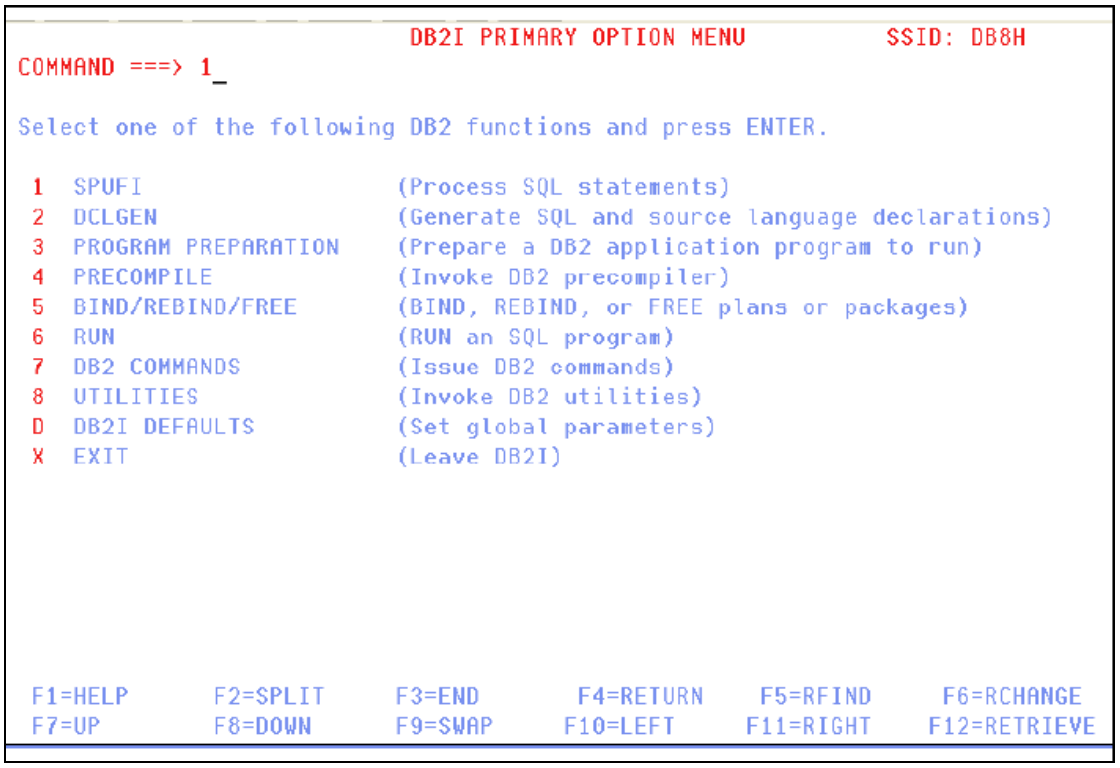


图12-5 使用SPUFI输入SQL

SPUFI使用文件进行输入和输出，所以需要预先定义两个数据集：

- 第一个数据集叫ZPROF.SPUFI.CNTL，是一个分区数据集。它用来将查询作为其成员保存。如果是一个顺序数据集，SQL语句就会被覆写。
- 输出文件叫ZPROF.SPUFI.OUTPUT，必须是一个顺序数据集。这意味着输出结果将在执行下一次查询后被覆写。如果您希望保存结果，必须使用ISPF菜单的编辑功能将文件重命名。

在图12-6中您可以看到这些信息是怎样填入的。

399

[1] QMF 包括一种管理功能来捕捉那些使用了糟糕查询语句而消耗的 CPU 数。

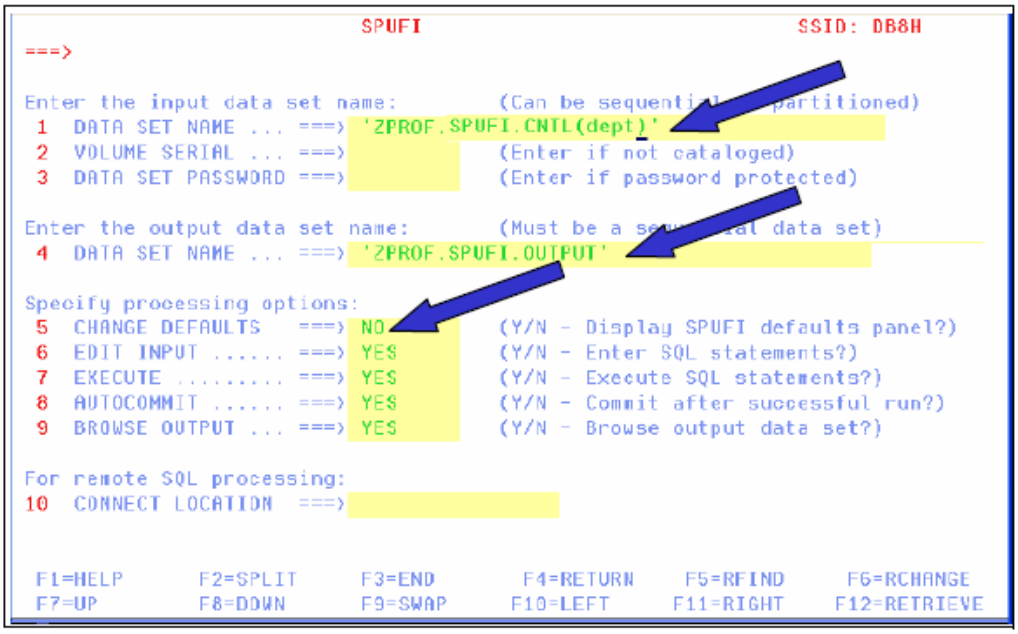


图12-6 指定SPUFI数据集

请注意选项5，可以临时将其改为YES来查看缺省值。您可能会想去改变的一个值是获取记录的最大条数。

当对选项5置为NO的时候，如果按下Enter键，SPUFI就会打开输入文件ZPROF.SPUFI.CNTL(DEPT)，让您输入或编辑SQL语句。在命令行中输入recov on，然后按回车，屏幕上方的警告就会消失。该选项是profile的一部分，在本书前面的部分中曾有所提及。屏幕如401页图12-7所示。

400

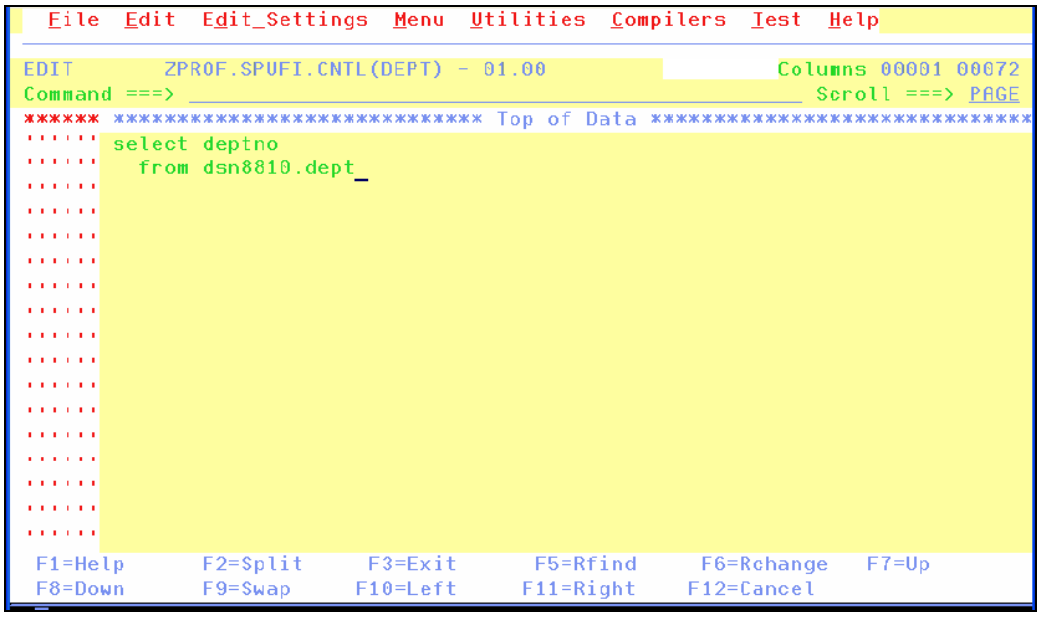


图12-7 编辑输入文件

如果在您的profile中设置了CAPS ON，那么在键入回车键时所输入的SQL语句将自动转为大写。但这并不是必须的。

请注意我们曾经提及过名为DSN8810.DEPT的表，该表名是经过限定的名字，我们通过该表名可以使用由DSN8810创建的示例表DEPT。

如果您只输入一条SQL语句，就不需要使用SQL分隔符，即一个分号。因为这是缺省指定的(但也可以在提及过的选项5中改变)。但如果输入多条SQL语句，就需要在每条语句结束的地方使用分号，以标识有不止一条语句。

之后，需要键入F3去回到最初的SPUFI面板。您将看到402页图12-8所示的界面。 401

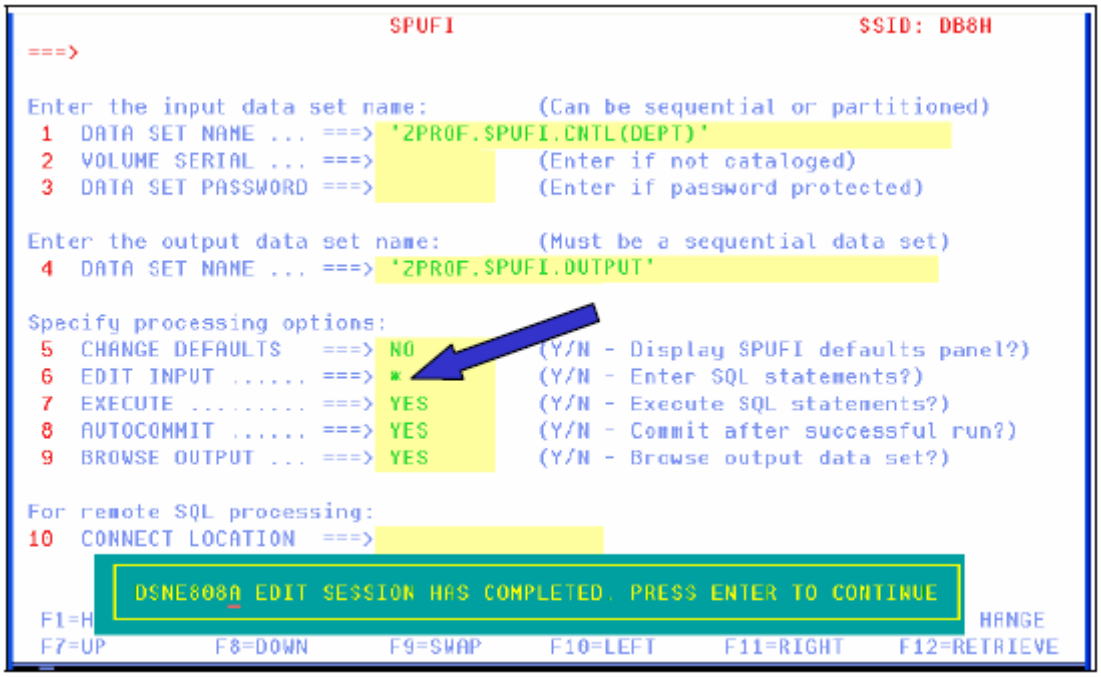


图12-8 回到最初的SPUFI面板

402 请注意由于已经结束了SQL的编辑，在选项6之后出现了一个星号。之后如果输入回车键，将执行输入的SQL语句并自动打开输出文件，这是因为选项BROWSE OUTPUT被设置成了YES。输出的第一部分如403页图12-9所示。

```
 Menu  Utilities  Compilers  Help
 BROWSE    ZPROF.SPUFI.OUTPUT                  Line 00000000 Col 001 080
 Command ===> _                                        Scroll ===> PAGE
******************************** Top of Data ********************************
--------+---------+---------+---------+---------+---------+---------+---------+
select deptno                                                        00010000
  from dsn8810.dept                                                  00020000
--------+---------+---------+---------+---------+---------+---------+---------+
DEPTNO
--------+---------+---------+---------+---------+---------+---------+---------+
A00
B01
C01
D01
D11
D21
E01
E11
E21
F22
G22
  F1=Help    F2=Split   F3=Exit    F5=Rfind   F7=Up      F8=Down    F9=Swap
 F10=Left   F11=Right  F12=Cancel
```

图12-9 SPUFI查询结果的第一部分

键入F8将获取第二部分(在本例中为最后一部分)的结果，如图12-10所示。

```
 Menu  Utilities  Compilers  Help
 BROWSE    ZPROF.SPUFI.OUTPUT                  Line 00000018 Col 001 080
 Command ===> _                                        Scroll ===> PAGE
H22
I22
J22
DSNE610I NUMBER OF ROWS DISPLAYED IS 14
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100
--------+---------+---------+---------+---------+---------+---------+---------+
DSNE617I COMMIT PERFORMED, SQLCODE IS 0
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 0
--------+---------+---------+---------+---------+---------+---------+---------+
DSNE601I SQL STATEMENTS ASSUMED TO BE BETWEEN COLUMNS 1 AND 72
DSNE620I NUMBER OF SQL STATEMENTS PROCESSED IS 1
DSNE621I NUMBER OF INPUT RECORDS READ IS 2
DSNE622I NUMBER OF OUTPUT RECORDS WRITTEN IS 30
****************************** Bottom of Data ******************************
  F1=Help    F2=Split   F3=Exit    F5=Rfind   F7=Up      F8=Down    F9=Swap
 F10=Left   F11=Right  F12=Cancel
```

图12-10 查询结果的第二部分

请注意结果表只有一列。这是由SELECT所指定的，即DEPTNO。我们从表中所有的记录中(14条)获取了DEPTNO。这里还有一些信息。其中一条给出了所取得的记录数目。另一条指出了SQLCODE(SQLCODE是执行SQL之后的返回码，标记成功与否)的值是100，它表示读到了文件的尾端，因此没有更多的结果可以显示。

相关阅读：可以参考IBM出版物DB2 UDB for z/OS:SQL Reference来获取更多的关于SQL的信息。可以在z/OS因特网知识库网站上找到这本书和其他的相关出版

物。

http://www.ibm.com/servers/eserver/zseries/zos/bkserv/

12.9 DB2 应用程序开发

SQL并不是一种纯粹的编程语言，但对于访问和操控DB2数据库的数据而言，它是必不可少的。它是一种为使用DB2数据库在20世纪70年代中期产生的第四代非程序语言。SQL既可以通过一个如SPUFI的解释型程序动态地执行，也可以嵌入并编译或汇编到一个宿主语言中进行执行。

那么怎样才能编写访问DB2数据的应用程序呢？

为了做到这一点，要将SQL嵌入到编程语言的源代码中，如Java,，Smalltalk，REXX，C，C++，COBOL， Fortran，PL/I和高级汇编。在程序中可以使用两种SQL语句：静态的和动态的。

- 静态

  在这种情况下，SQL指的是编写在源代码中的完整的SQL语句。在程序的准备阶段，DB2为语句创建访问路径，并将它们记录在DB2中。在多个运行实例之间，SQL从不发生改变，一直使用同样的路径，DB2也不需要重新创建访问路径。(请注意：所有的SQL必须有一个访问路径。)

- 动态

  在这种情况下，SQL指的是编写在源代码时，部分或者完全不确定的SQL语句。在应用程序的运行时刻，DB2才能知道确切的SQL语句，并决定合适的访问路径。由于在每次执行时语句都会发生变化，所以记录访问路径没有意义。SPUFI就是一个例子。SPUFI是一个接受动态SQL语句的应用程序，即是在输入文件中编辑的SQL语句。每次您使用SPUFI时，SQL都可能会发生变化。因此要在应用程序之中嵌入特殊的SQL准备语句以应对这种情况。

404

我们现在集中介绍静态SQL，去了解使用DB2时所涉及到的过程。我们想要说这个过程可能看上去复杂，但每种行为都有其存在的原因。

12.9.1 DB2 程序准备：流程

传统的程序准备流程包括编译和链接编辑。但对于DB2而言，因为编译器认不出SQL语句，必须要有一些额外的步骤去处理SQL语句。包括编译和链接编辑在内的这些步骤可以使用DB2I面板来完成。但除了DCLGEN之外，通常可以使用一个JCL作业流来完成所有的处理。以下解释请参考407页图12-11。

DCLGEN

DCLGEN是一种自动产生DB2对象代码定义的方式，这些对象将在程序中使用。这些定义设置在DCLGEN库中的一个成员里。您可以选择在代码中包括它们，如

果不包括它们就必须手工编写这些定义。DB2数据库管理员必须根据公司的规则创建它们。在这些步骤中需要有一个运行的DB2系统，因为定义是根据DB2编目所产生的。

预编译

由于编译器无法处理SQL，所以在预编译阶段中，预编译就把SQL语句注释掉，替换为一个调用DB2的CALL语句。该调用传递了一些参数，例如宿主变量地址(用于存放取得的数据)，语句数目，和一个叫做一致性令牌(但通常被称为时间戳)的修改过的时间戳(该参数非常重要！)。在该阶段不需要一个运行的DB2系统——所有的操作都无需访问DB2。

预编译器通过特殊的起始符和终止符来识别SQL，这些标识符是每个SQL语句都必须包含的。起始符EXEC SQL对于所有的编程语言都是相同的，但终止符有所差别，COBOL使用END-EXEC.(请注意句号)，而C和其他语言使用一个分号。以下是一个COBOL的例子：

```
EXEC SQL
    SELECT EMPNO, LASTNAME
    INTO :EMPNO, :LASTNAME
    FROM EMP
END-EXEC.5
```

405

在本例中，数据库中的列EMPNO和LASTNAME被取入到以冒号作为开始的宿主变量中。宿主变量是定义在SQL所嵌入的宿主语言中的变量(如COBOL，PL/1等等)。在DCLGEN阶段，可以定义一系列的这种类型的变量。该示例中宿主变量名与列名相同，但这不是必需的。宿主变量可以任意命名，只需其数据类型与列的数据类型相对应即可。

在预编译之后，程序被分为了两个部分：

- 改动过的源代码；原始代码中的SQL已经被注释掉并被CALL语句所替换。
- 数据库请求模块(database request module，DBRM)；它包含了程序的SQL语句的，通常是一个PDS库的成员。

改动后的代码会交给编译器去编译，再交给链接编辑器创建出一个可执行模块。这个过程就像任何不包括SQL的程序一样。

406

顺便，只要有足够的权限，可以在程序中嵌入任意类型的SQL：DML，DDL，和DCL。

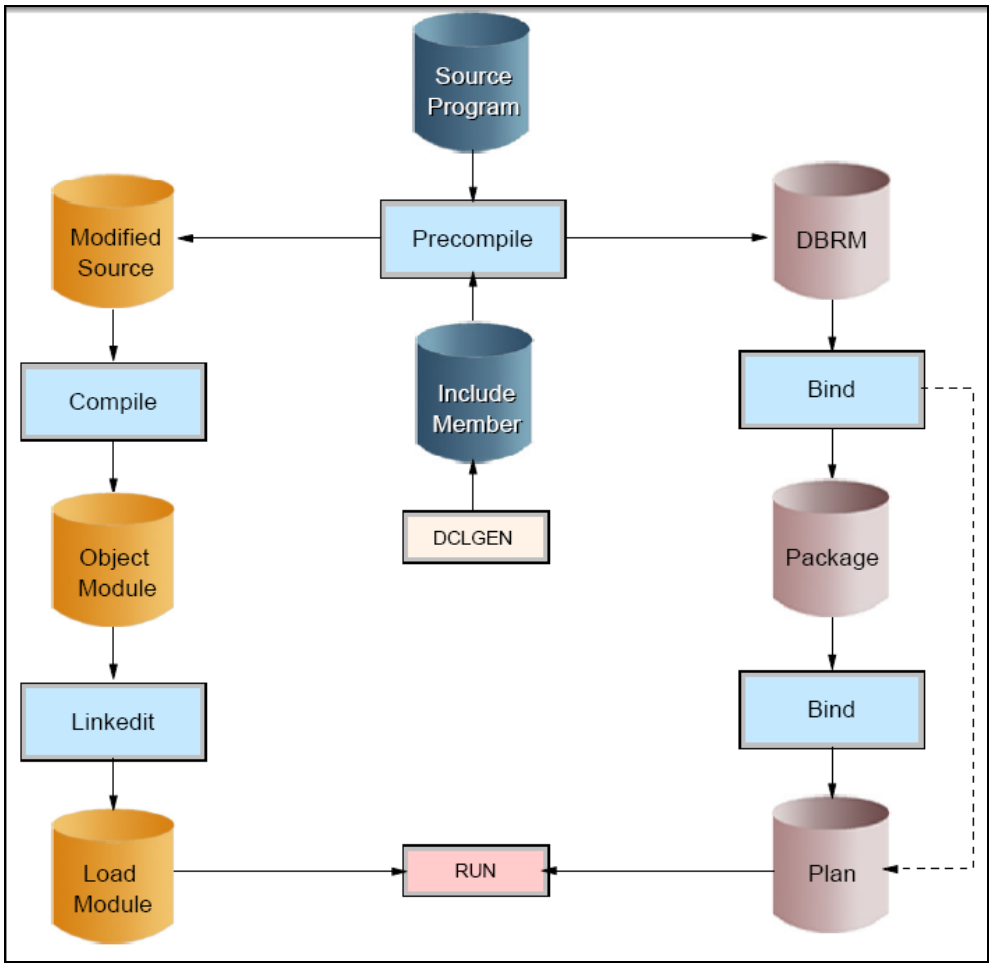


图12-11 程序准备流程

绑定

绑定(BIND)可以被认为是DB2中编译DBRM的流程。绑定时作以下三种工作：

- 语法检查。
- 权限检查。
- 最重要的是，决定语句的访问路径。DB2中有一个叫做优化器的组件，它可以评估各种对数据进行访问的不同方式，如浏览整个表，是否使用索引及使用哪个索引等等。优化器会权衡每种开销并做出最优的决定。这就是一个基于开销(cost-based)的优化器，与之相对的还有基于规则(rule-based)的优化器。

SQL及其访问路径(包括一致性令牌/时间戳)会作为一个包(package)存储在DB2的目录中。其他的一些信息，诸如包的信息和实际的SQL语句则存储在编目(catalog)中。绑定为一个应用程序创建了可执行的SQL代码并放在其对应的包中。现在，DB2就拥有了所有它所需要的信息，可以去获取应用程序所请求的数据了。

程序通常还需要调用子程序，这些子程序也可能包含SQL调用。于是这样的每个子程序也都会有一个包。您需要将所有的DB2信息整合起来，所以就需要另一个

步骤进行另一次绑定，在这次绑定中创建一个计划(plan)。

即使您没有使用一个子程序，您依然需要创建一个计划。计划中可能包含您的程序的范畴之外的信息。在实践中这是很常见的：一个计划中包含了一个项目所用到的所有的包，该项目每次运行时都仅使用这个计划。

为了叙述的完整，我们需要添加说明：最初的DBRM也可以直接绑定到一个计划中(这被称为流内DBRM)。但这样的话，如果我们对众多程序中的一个做出微小的改变，那么整个计划都必须重新绑定。即使仅仅添加一个新索引也要如此。

正如您所知道的那样，在绑定过程中，DB2会更新它的目录和编目。更新意味着防止其他用户进行更新(数据被锁)，所以用户的其他对DB2的操作大多都无法执行。为了避免这种限制，包就被引入了。现在只需要重新绑定一个包，所以更新的时间非常短，对其他用户的影响也就微乎其微了。而现在依然存在流内DBRM的计划，尽管大多数的公司选择将它们转换成由包组成的计划。

计划是主机平台上所独有的；其他平台上没有这一概念。

运行

当您执行应用程序的时候，可执行模块被载入到主存中。当遇到一条SQL语句的时候，系统就转向替换SQL的DB2的CALL语句，并将参数传递给DB2。参数之一便是一致性令牌，该令牌或是时间戳也存在于包中。然后DB2就会在特定计划中寻找具有相应时间戳的包，找到后就进行装载和执行。所以运行时，需要指定一个计划名作为参数。

最后请注意：SQL语句的执行结果通常是一个结果集(多于一条记录)。一个应用程序只能在同一时刻处理一条记录或一行。所以DB2中加了一个额外的特殊结构，
408 叫做游标(cursor)，它本质上是一个指针。在嵌入式SQL中，游标允许您从结果集中每次获取，更新或删除一条记录。

相关阅读：可以参考IBM出版物DB2 UDB for z/OS:Application Programming and SQL Guide来获取更多的信息。

12.10 IMS 数据库管理器的功能

一个数据库管理系统(DBMS)为商务交易和数据存储的访问提供了一系列的工具。DBMS所扮演的角色提供以下的功能：

- 允许多个用户对同一份数据的同时访问。
- 控制对数据的并行访问以维护所有更新的一致性。
- 减少对硬件设备和操作系统访问方式的依赖程度。
- 只维护一份数据，减少数据的冗余程度。

12.11 IMS 数据库子系统的结构

IMS数据库管理器提供了管理和访问应用数据的中心控制。IMS在其产品中提供了一整套的实用程序去提供这些功能。本部分中将描述z/OS几种不同的类型地址空间，以及它们之间的联系。IMS子系统的核心是一个运行在一个z/OS地址空间上的控制区域(control region)。还有一些其他区域或者IMS应用程序，运行在一些依赖地址空间中，为控制区域提供一些额外的服务。

除了控制区域，IMS使用的一些应用程序和工具在独立的批处理地址空间中的运行。它们是与IMS子系统及其控制区域相分离的，并与之没有任何联系。

出于历史的原因，一些描述IMS的文档使用术语"区域"去描述一个z/OS地址空间，例如，IMS控制区域。在本文中，在较常用的地方，我们就使用"区域"这个术语。您可以将该术语理解为z/OS地址空间。

图12-12展示了IMS DB/DC子系统。如果需要更多的信息，可以参考An Introduction to IMS (ISBN 0-13-185671-5)。

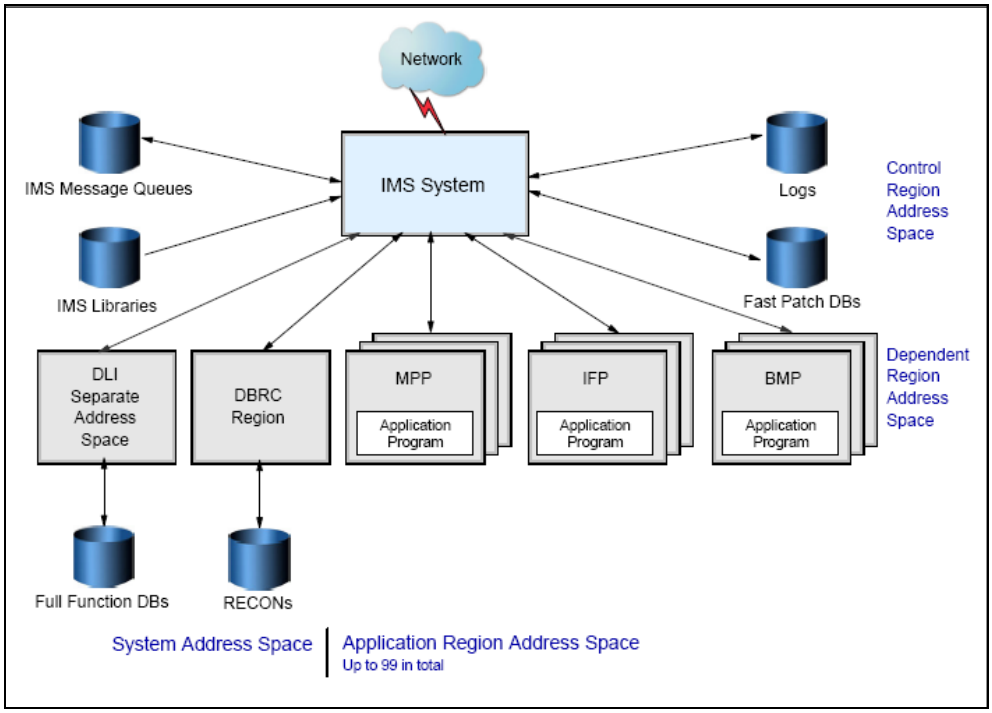


图12-12 IMS DB/DC子系统的结构

12.11.1 IMS 层次数据库模型

IMS使用层次模型作为存储数据的基本方式。这是一种存储数据和实现不同类型实体之间关系的有效方式。

在这个模型中，单个的实体类型作为层次结构中的段(segment)来实现。层次结构由数据库的设计者，根据实体之间的关系和应用程序所需要的访问路径来决定。

请注意在IMS产品中，术语"数据库"与其他DBMS中所谓的数据库有微小的区别。在IMS中，一个数据库通常被用来描述一个层次的实现，所以一个应用程序经常要访问许多个数据库。与关系型模型相比，一个IMS数据库大致相当于关系型模型中的一张表。

DL/I允许数据结构有很大的变化。每个层次型数据结构中最多允许有255个段(segment)类型，而在每个层次型数据结构上最多可以定义15个段层(segment level)。对每个段类型的实例个数亦没有限制，除非受制于物理访问方法的限制。

410

访问段的顺序

遍历层次型模型的顺序是从上到下，从左到右，从前到后(对于孪生兄弟段)。

段的编码号并不考虑孪生兄弟段情况，而对数据库记录的顺序处理是一种层次型的序列。一个数据库记录的所有段都会被包括，所以孪生兄弟段确实在这个层次型序列中有一席之地。段中可能包含着决定它们存储和处理顺序的序列域。

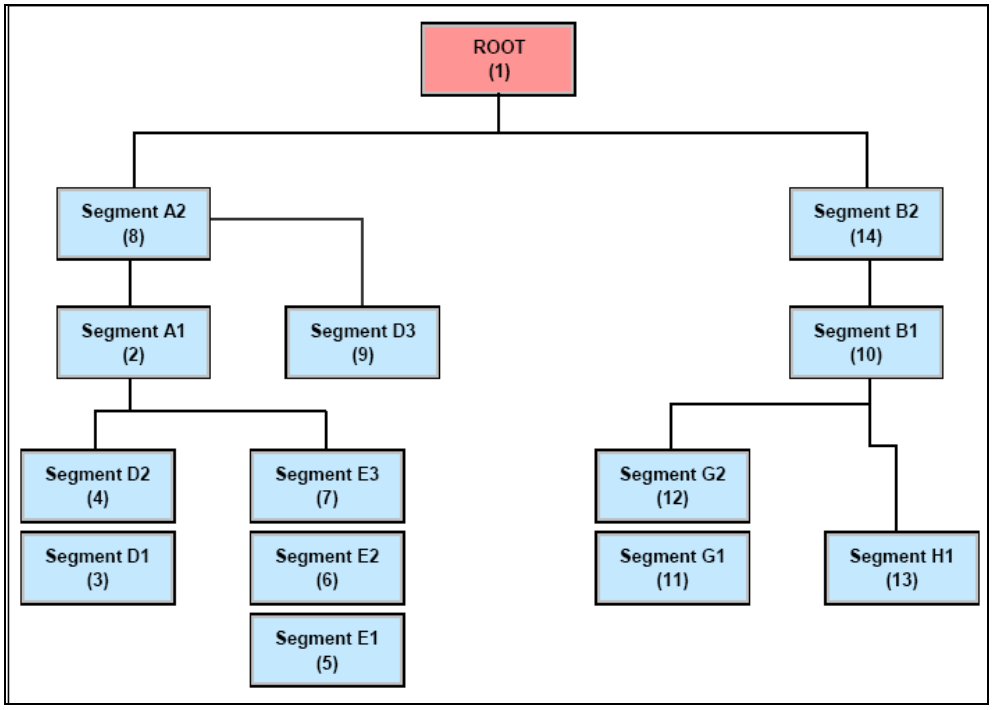


图12-13 序列

图12-13中的层次型数据结构描述了从应用程序的角度来看的一个数据库记录中的数据。它并不表示数据的物理存储。数据的物理存储对应用程序是无关紧要的。

图12-13也展示了层次型数据结构中最基本的构建元素，即数据段之间的父/子关系。

411

12.11.2 IMS 对 z/OS 服务的使用

IMS被设计能充分利用z/OS操作系统的特性。这包括以下几点：

- 它运行在多个地址空间内

  IMS子系统(IMS/DB批处理应用程序和实用程序除外)通常由一个控制区域的地址空间，用于提供系统服务的多个依赖的地址空间，IMS应用程序的多个依赖的地址空间组成。运行在多个地址空间上的特性带来了以下几个好处：

  - 当运行在多处理器CPC上时，可以最大幅度地利用CPU。
  - 在多个不同的CPU上，地址空间可以被并行地调度。
  - 将应用程序与IMS系统的代码隔离开来，减少了由于应用程序失败而引起的系统中断。
- 在每个地址空间上可以运行多个任务

  尤其在控制区域中，IMS为了实现不同的功能而创建了多个z/OS子任务。这就允许当一个IMS子任务在等待系统服务时，z/OS可以调度其他的IMS子任务。

- 使用z/OS的跨内存服务来实现构成IMS子系统的不同地址空间的通信。它也使用z/OS的公共系统区域(Common System Area，CSA)来存储IMS控制块，这些控制块由构成IMS子系统的地址空间频繁使用。这就将运行多个地址空间所带来的开销降到了最低。
- 使用z/OS子系统的特征来检测相依赖的地址空间何时发生崩溃，来防止取消依赖的地址空间，还可以与诸如DB2和WebSphere MQ这样的子系统进行交互。
- 可以利用z/OS的系统综合体(本书稍候讨论)。多个IMS子系统可以在多个z/OS上同时运行，形成一种系统综合体，访问同一个IMS数据库。它带来了以下两个好处：
  - 可用性增强：不需要中断服务，即可在z/OS系统和IMS子系统之间切换。
  - 容量增加：多个IMS子系统可以处理更大的工作量。

12.11.3 IMS 的演化

起初，所有的IMS/DB在线应用程序使用IMS/TM作为数据库的编程接口。但随着DB2的普遍使用，许多客户转向使用DB2作为数据库来开发他们的在线程序，和他们已经存在的运行良好的IMS应用程序共存，这就是您为什么能在现实世界中看到许多混合系统的原因。

412

12.11.4 我们的在线程序示例

回过来看我们在11章349页”z/OS的交易管理系统”中旅行社的例子。IMS交易的例子可以作为航空公司系统的一个组成部分：

- 一些批处理可能需要每日进行更新，例如旅行社和其他客户的付款。
- 另一些批处理可能是一些提示，提醒旅行社和其他客户支付款项。
- 可以使用在线应用程序用来检查预定是否完成(和支付)。
- 检查是否还有剩余座位。

12.12 总结

数据可以存放在一个无格式的文件中，但通常这会导致数据的冗余和不一致。因此，最好创建中心数据库，并能从各个不同的地方来方便地访问(读取或者更改)数据。数据库管理系统来处理一致性，安全性等问题；用户和开发人员就可以不必为此费心。

关系数据库是当前商业世界中最主流的数据组织方式。IBM的DB2实现了关系型的准则，如主键，参考完整性，访问数据库的语言(SQL)，null和范式设计。在关系数据库中，最基础的结构是由行和列组成的表。

DB2的基本对象之间有层次型的依赖关系。在表结构之上可以创建索引和视图，如果表被删除，那么这些对象也随之被删除。表存放在被称为表空间的物理数据集中，表空间与一个数据库相关联，数据库是多个表空间的逻辑分组。DB2中较新的模式对象包括UDT，UDF，LOB，触发器和存储过程。

DB2也有用来帮助管理子系统的系统结构。编目和目录用来存储那些描述RDBMS所有对象的元数据。缓冲池用来存储从磁盘介质上获取的数据页，以提高查询速度。活动日志或归档日志和BSDS用于记录DB2中数据的变化，以便于数据库的恢复。

使用SQL是访问数据库中数据的唯一方式。它并不是一种完整的编程语言，并且工作在集合的级别。当它操纵数据的时候，使用结果表。SQL根据功能可以分为三类：DML，DDL和DCL。在主机上，SPUFI是一种用于输入SQL语句的工具。

由于传统的第三代编译器无法识别SQL，在应用程序中使用SQL需要一些特殊的步骤。预编译器会将程序中的SQL语句注释掉，将它们复制到DBRM中，并用一个一致性令牌标记，然后用将那些SQL语句用DB2的调用语句替换掉。改动后的源代码然后经过编译和链接编辑。DBRM通过绑定过程来决定访问路径，并将该可执行SQL代码放入一个包中。多个包在逻辑上与一个计划相关联。程序运行时，装入模块中的DB2调用将会将其一致性令牌传递给DB2，DB2将在相应计划中找到匹配的包，然后执行SQL。

SQL既可以处理静态语句，又可以处理动态语句。EXPLAIN可以被用来找出优化器为SQL选择的访问路径。

EXPLAIN语句定义了查询的访问路径以期提高效率。EXPLAIN对于多张表，多次访问的数据库查询特别有用。

本章中的重要术语		
全功能数据库 (full-function database)	DL/I	更改的源码(modified source)
SPUFI	SQL	SYSADM
EXPLAIN	视图(view)	DBMS
多任务(multitasking)	多线程(multithreading)	数据库管理员(database administrator，DBA)

12.13 复习题

为了帮助您检测对本章内容的理解程度，请回答下列问题：

1. 数据库管理员(DBA)的职责有哪些？
2. 哪个DB2对象定义了一个物理存储区域？是表吗？
3. 以下的SQL语句有什么问题？

   SELECT * FROM PAYROLL;

4. 如果定义DB2的对象，应该使用哪种SQL？
5. 预编译器是怎样在程序中找到SQL语句的？
6. 装入模块是如何与SQL语句合并在一起的?
7. 怎样才能知道优化器选择了哪条访问路径？在哪个处理过程中创建访问路径？
8. 什么是存储过程？
9. 系统管理员的职责有哪些？
10. 数据库管理员(DBA)的职责有哪些？
11. DB2用哪些方式去保证安全性?
12. 什么是IMS-DB的数据库结构?请描述。

12.14 练习 1 在一个 COBOL 程序中使用 SPUFI

在开始此练习之前请先确保能连接一个DB2。

12.14.1 步骤 1：创建文件

在开始DB2练习之前，需要创建两个PDS：

- ZUSER##.DB2.INCLUDE,，用来存放DCLGEN。

► ZUSER##.DB2.DBRM，用来存放DBRM。

您可以使用ZUSER##.LANG.CNTL作为基库。

另外，还需要一个ZUSER##.SPUFI.OUTPUT文件，文件格式如下：记录格式VB，记录长度4092，块长4096。

12.14.2 步骤 2：DCLGEN

DCLGEN是一个简易地为您在应用程序中使用的DB2信息产生COBOL定义的方法。这些语句可以包含在源程序中。

首先，在DB2I面板中输入D选中‘DB2I Defaults’，如图12-14所示，并按回车键。

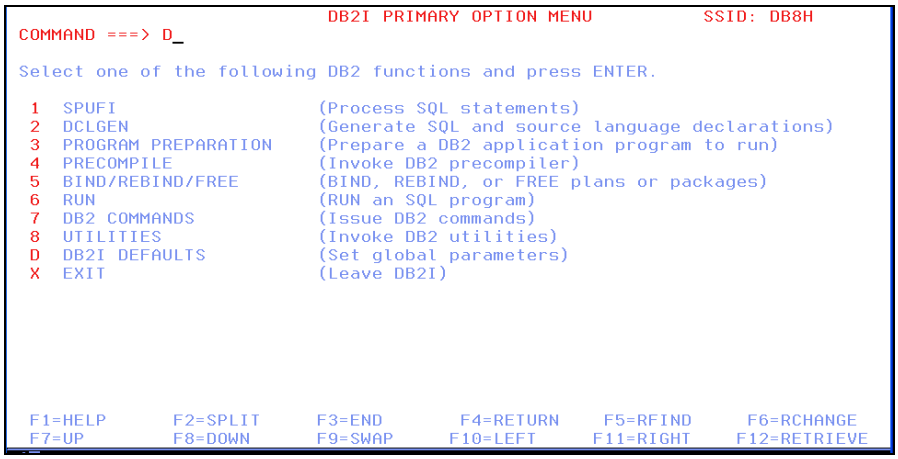


图12-14 DB2I菜单

在‘DB2I DEFAULTS PANEL 1’上，将选项3‘APPLICATION LANGUAGE’指定为‘IBMCOB’(如图12-15)。

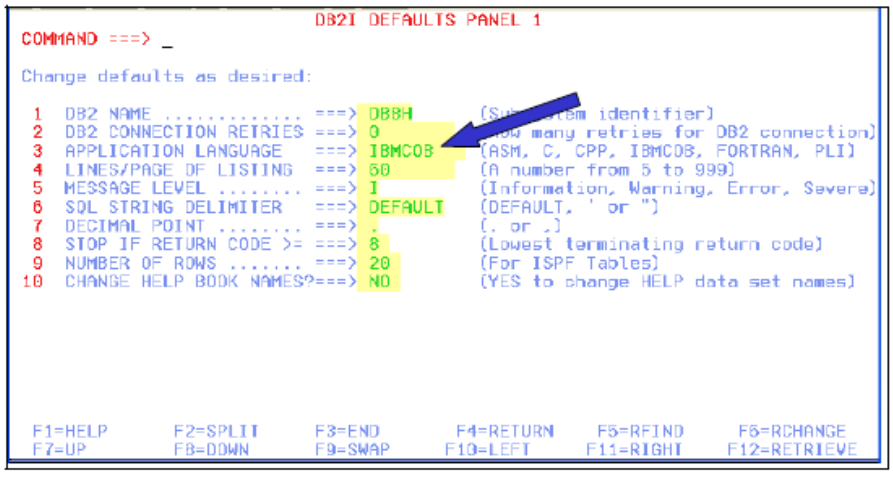


图12-15 DB2I默认值面板1

416

键入回车，在‘DB2I DEFAULTS PANEL 2’上，将选项2‘COBOL STRING DELIMITER’指定为DEFAULT；将选项3‘DBCS SYMBOL FOR DCLGEN’指定为G。键入回车(如图12-16)。

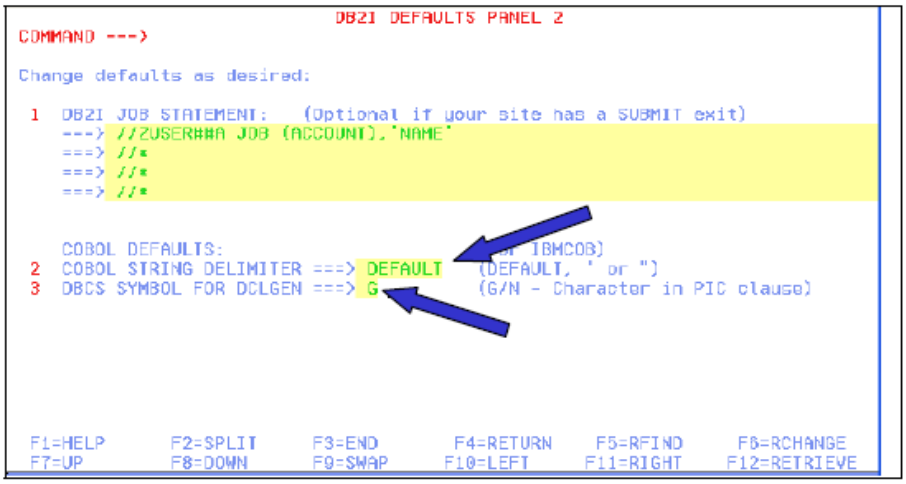


图12-16 DB2I默认值面板2

以上步骤用于确定您使用了正确的语言。

回车之后，回到DB2I主面板(如416页图12-14)。现在选择选项2‘DCLGEN’。

您需要一个已经创建好了的目标数据集来保存DCLGEN所产生的定义(ZUSER##.DB2.INCLUDE)；它应该已经被创建好了。如果您还没有，可以使用ISPF菜单创建一个PDS。

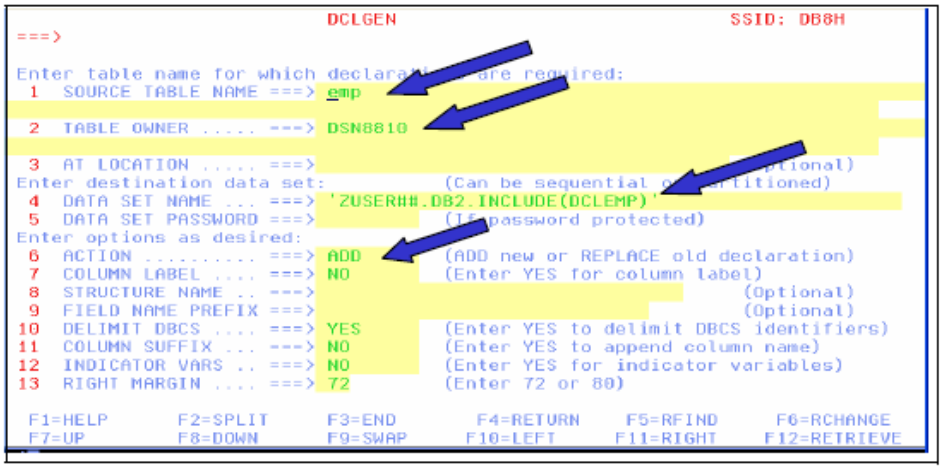


图12-17 DCLGEN

如图12-17所示，您需要指定表名，表的拥有者，您的PDS文件，和指定ADD操作。结果消息应该是：

417

```
EXECUTION COMPLETE, MEMBER DCLEMP ADDED
***
```

如果表的定义发生了改变，您必须要改变DCLGEN的设置，并且使用REPLACE操作。

12.14.3 步骤 3：测试您的 SQL

回到SPUFI；使用您的PDS数据集SPUFI.CNTL。在该PDS中有一个叫做SELECT的成员，在该成员中有您将要在程序中使用的SQL语句。在此处没有where子句，所以您可以看到所有能获得的结果。通过该语句您可以看到在表中有哪些部门。

对于那些更复杂的查询，这是一个验证其正确性的好方法。作为一名应用程序开发员，您必须确定执行了正确的SQL。

12.14.4 步骤 4：创建程序

在此处，您可以创建一个程序，或者使用为您提供的程序LANG.SOURCE(COBDB2)。该示例程序计算了一个部门的平均工资。您只需指定部门即可获得结果。输入999即可退出程序。

更改该程序，加入以下内容：

- 您的变量(包括您在步骤1中所创建的成员)。
- 指定COBOL的SQL分隔符。

如果您用"???"来查找就能看到它们的位置。

12.14.5 步骤 5：完成程序

编辑作业LANG.CNTL(COBDB2)，并按放在作业顶部的说明做出相应的修改。

您会在该作业中找到以下几个作业步：

- 预编译作业步(PC)：它将您的程序分为两个部分：DBRM和修改过的源代码。
- 编译，预链接和链接作业步(COB，PLKED，LKED)：这些步骤将编译和链接您的被改动过的源代码。
- 绑定作业步(BIND)：该步骤绑定包和计划。

  问题：如果您需要修改程序，可以跳过哪一步的绑定？您可以自由地更改程序。您可以将求平均值改为求一个部门的最高或者最低工资(这样的话只需更改SQL)。

► 运行作业步(RUN)：以批处理方式运行程序，求部门A00和D21的平均工资。

12.14.6 步骤 6：从 TSO 中运行程序

不以批处理的方式运行程序，试着从TSO READY提示符处运行程序。如果要这样做，您必须为您的回话分配2个文件(这些操作要在运行作业之前完成)。

键入下面的命令，然后每一行后按一次回车：

```
TSO alloc da(*) f(sysprint) reuse
tso alloc da(*) f(sysin) reuse
```

然后回到了您的DB2I屏幕。

选择选项6‘RUN’。在这里输入文件的名字和计划的名字，如图12-18。

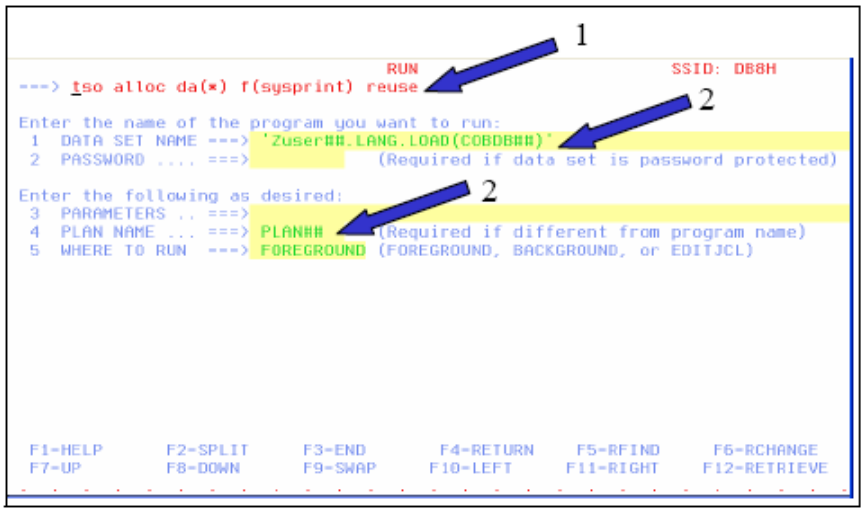
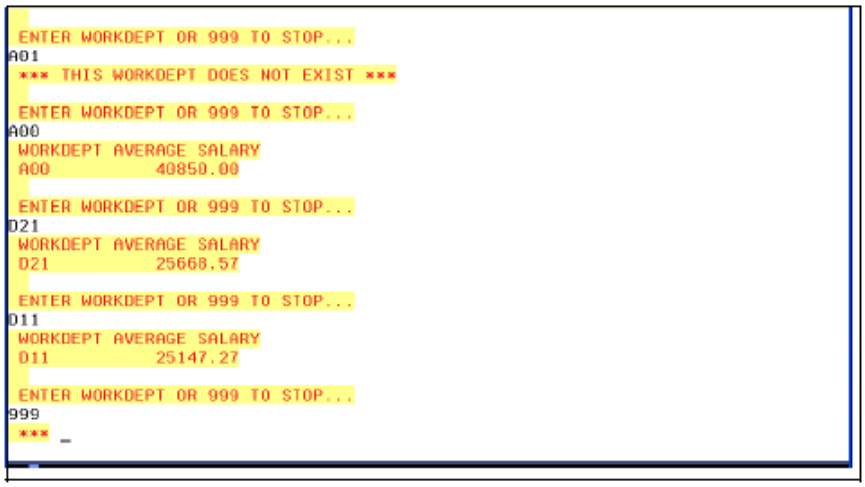


图12-18 准备好运行

419



420

图12-19 程序的执行

13

第 13 章 z/OS 上的 HTTP 服务器

目标：作为主机专业人员，您需要了解如何在z/OS上部署Web应用程序并且应该了解如何使z/OS具有处理Web事务的能力。

在本章结束之后，您应当能够掌握：

- 列举出三种服务器模式。
- 理解静态和动态Web页面。
- 对于下面的每一功能组，列出至少他们所包含的两种功能：基本，安全和缓存。

13.1 z/OS 上 Web 事务的介绍

随着越来越多的企业将他们的应用程序移植到Web环境，主机的技术厂商面临着一个难题，如何来支持和管理新的基于web的联机事务，同时还要面对越来越多的传统的事务处理，比如批处理。

在接下来的几章中将会介绍如何通过使用一些中间件产品所提供的功能来使z/OS具有处理Web事务的能力：

- 第13章(421页)，”z/OS上的HTTP服务器”
- 第14章(433页)，”z/OS上的WebSphere应用服务器”
- 第15章(449页)，”消息和队列机制”

这些章节中的事例都是基于IBM的产品，但现今市场上还有很多类似的产品可供选择。

13.2 什么是 z/OS HTTP 服务器？

z/OS HTTP服务器可以处理静态和动态的Web页面。z/OS HTTP服务器具有和其他Web服务器相同的功能，但它同时还有一些专为z/OS设计的特性。您可以使用以下三种模式中的任何一种来配置和运行z/OS HTTP服务器，每一种模式都有各自在处理Web事务方面的优势：

独立服务器模式 这种模式适用于单纯的基于HTTP服务的应用实现(例如一些简单的Web网站)。它的主要职责就是提供一些发布到Internet上的功能极为有限的Web应用。

扩展服务器模式 这种模式适用于那些交互式的Web网站，这些网站的访问流量经常动态的增加或者降低。它适用于更为复杂的应用环境下，比如需要支持对Servlet和JSP的调用。

多服务器模式 这种模式下，将独立服务器和扩展服务器组合在一起，以此来提高整个系统的可扩展性和安全性。例如，独立模式服务器可以被用做连接到扩展模式服务器的一个网关，在这个网关中可以验证所有请求的身份认证，并且可以把这些请求重新路由到其他服务器。

13.2.1 在 z/OS 上处理静态 Web 页面

通过z/OS上的Web服务器，例如HTTP服务器，用户对于静态页面的处理和其他平台的Web服务器是相似的。用户可以通过向HTTP服务器发送一个HTTP的请求来获取某一特定的文件。HTTP服务器从它的文件存储目录下获取该文件并将它发给用户，同时会在响应的HTTP的头部加入关于文件的一些信息(例如，文件MIME

类型和大小)。

然而，z/OS HTTP服务器同其他的Web服务器有一个最主要的区别，因为z/OS操作系统使用EBCDIC进行编码，所以z/OS上的文件必须被首先转成Internet支持的ASCII形式(但是二进制的文件，例如，图片并不需要转码)。

z/OS HTTP服务器会自动的进行以上的这些转码工作，以省去程序员编程时在转码这一步上所耗费的时间。然而，程序员必须使用FTP将这些文档上传到服务器上。也就是说，程序员在通过FTP上传时要指定以ASCII码传输的形式将文件从EBCDIC码转换成ASCII码。对于二进制的传输，文件不会被转换。

13.2.2 在 z/OS 上处理动态 Web 页面

动态页面是Web商业领域中的重要组成部分。对于每一种的交互方式和个性化的设置都需要通过动态的内容来实现。例如，当一个用户在网站上填好一张表格后，表格中的数据必须被处理，同时相应的返回信息必须呈现给用户。

在z/OS上有两种方式来处理动态Web页面：

- “使用CGI来处理动态Web页面”，423页
- “使用插件接口”，424页

使用CGI来处理动态Web页面

一种处理动态Web页面的方式是通过通用网关接口(CGI)的方式来实现，CGI是HTTP协议的一个组成部分。CGI是一种标准的方法，Web服务器通过CGI将一个Web用户的HTTP请求发给一个应用程序。CGI生成输出再将输出传回给HTTP服务器，HTTP服务器通过HTTP响应的方式将CGI的输出发回给用户(图13-1)。

CGI并不仅仅限于返回HTML的页面；应用程序也可以创建纯文本文档，XML文档，图片，PDF文档等等。MIME类型必须反映出HTTP响应的内容。

CGI有一个主要的缺陷，就是每一个HTTP的请求都需要一个独立的地址空间。当同时有多个请求时会降低效率。

为了避免这一问题，一个称作FastCGI的技术被提出来。基本上来说，HTTP服务器的FastCGI插件是一个程序，它可以在一个单独的地址空间范围内同时管理多个CGI请求，这样会避免对于每个请求都需要很多条程序指令来处理。更多的有关HTTP服务器插件的信息将在424页的”使用插件接口”部分讲述。

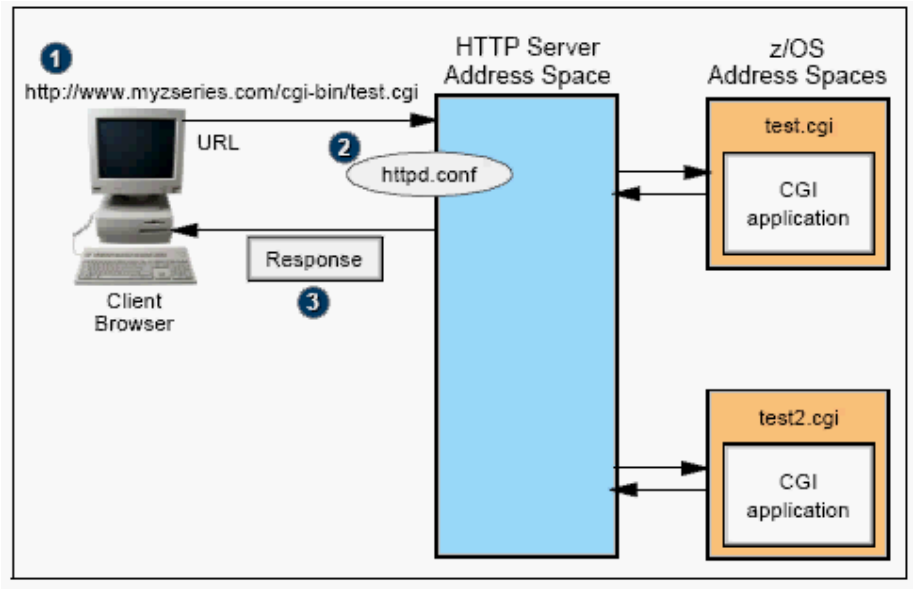


图 13-1 CGI的工作机制

使用插件接口

另一种用来处理动态页面内容的方法是使用HTTP服务器提供的插件接口，插件接口可以将几种产品中的一个同HTTP服务器连接起来。例如，HTTP服务器可以通过以下几种方式将控制权传递给WebSphere服务器。

- WebSphere插件，同一地址空间

  图13-2展示了一种简单的配置，这里不需要有J2EE™服务器。图中的Servlet可以连接到CICS或者IMS服务器，或者通过JDBC™连接到DB2上。然而，并不推荐在servlet中写业务处理逻辑的代码。

424

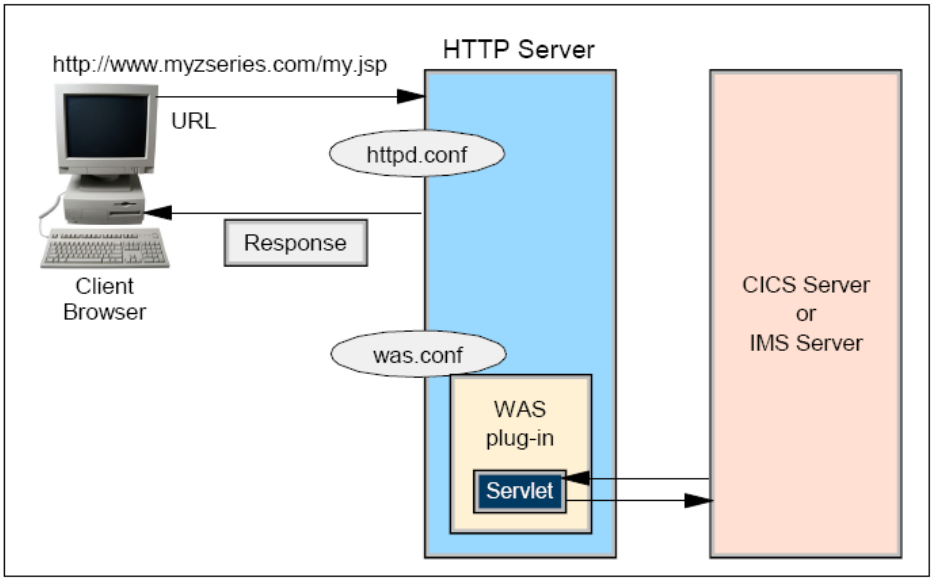


图13-2 使用WebSphere插件访问servlet

► HTTP服务器的Web容器，单独的EJB™容器

图13-3展示了一种更加实用的配置方法，在此种配置方法中servlet运行在与EJB不同的地址空间中，所以EJB是通过远程调用。然后EJB可以从其他的服务器那里获取信息。

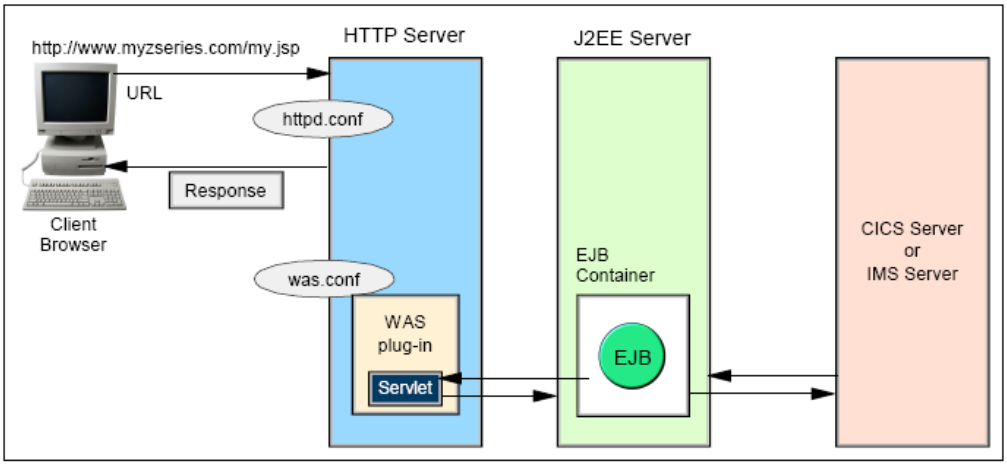


图13-3 通过WebSphere插件访问EJB

► 同时含有Web容器和EJB容器的单独的J2EE服务器

除了在WebSphere插件中通过本地调用的方式来运行servlet，您也可以使用WebSphere插件通过远程调用的方式在Web容器中来运行servlet，如图13-4所示。这可以让您将servlet和EJB放到同一个z/OS的地址空间里，这样调用EJB就不需要通过远程调用的方式来进行，提高了效率。

425

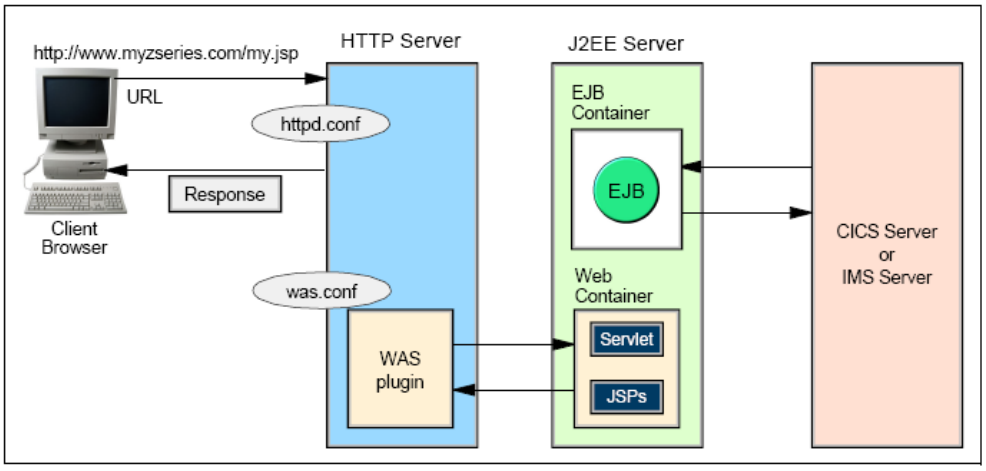


图13-4 使用WebSphere插件访问Web容器中的servlet

如果您准备使用WebSphere应用服务器，HTTP服务器可以不需要，但是这里提供了一些方法，您可以用它们来使HTTP服务器同WebSphere应用服务器进行交互。以上的部分说明了一些可以借鉴的方式。

13.3 HTTP 服务器的功能

HTTP服务器提供了和其他Web服务器相类似的功能，但是同时也提供了和z/OS相关的特有的功能。与z/OS相关的功能可以被分为如下几组：

- 基本功能
- 安全功能
- 文件缓存

13.3.1 基本功能

- EBCDIC/ASCII文件访问

  服务器访问文件并对他们进行转码，如果需要，从EBCDIC转成ASCII码。

- 性能和使用的监控

  作为z/OS特色的一部分，HTTP服务器可以生成系统管理设备(SMF[1])的记录，这些记录可被系统程序员事后用来作为性能和使用情况的分析。

426

- 跟踪和记录日志

  HTTP服务器具有一套完整的记录日志，跟踪和报表的功能，这可以使用户方便地跟踪每一HTTP的请求。

- 服务端包含机制(SSI)

  服务端包含机制可以使您向从服务器发给客户端的文档(静态的或者动态的)中插入一定的信息。这可以是一个变量(比如"最后修改"日期)，或者一个程序的输出，或者是另一个文件的内容。但是需要注意的是，如果仅仅是激活此项功能但是不使用的话，会对性能产生很严重的影响。

- 简单网络管理协议(SNMP)管理信息基础(MIB)

  HTTP服务器提供了一个SNMP MIB和SNMP的子代理服务，所以您可以使用任何具有SNMP功能的网络管理系统来监控您的服务器的运行状态，吞吐量，和运行其上的事务。如果有一些阀值超出了预定的范围，它可以及时通知您。

- Cookie支持

  因为HTTP是一种无状态协议，状态的记录可以通过cookie来实现，cookie中存放了客户端的信息。这项支持对于处理多个网页非常有用，例如要实现一些可定制化的文档或者轮替式广告。

- 多格式处理

  这项特性用于个性化的网页。浏览器将头信息和请求一同发送，包含了一种称为接受头部(accept header)信息。这种信息含有用户使用的语言。HTTP服务器可以利用接受头部的内容来选择恰当的文档内容返回给客户端。

[1] SMF 是 z/OS 操作系统一个可选特性，它提供给用户收集和记录信息的方法，这些信息可以被用来评估系统使用情况，以进行结账，退账和性能调优。

- 持久连接

  利用HTTP/1.1的特性，并不是每个请求都需要建立一个新的连接。持久连接会在一段特定的时间内保持”活动”状态，因此可以使用一个已经建立好的连接来处理另一个请求。

- 虚拟主机

  虚拟主机可以使您只运行一个服务器但是对于客户端来说却好像同时有多个服务器在运行。这是通过对于相同的IP使用不同的DNS名称或者对于同一服务器使用不同的IP地址来实现的。

13.3.2 安全功能

- 线程级别的安全

  对于每一个运行在HTTP服务器上的线程都可以为他们配置一个独立的安全运行环境，这就意味着对于连接到服务器上的每一个客户端都有它自己的安全运行环境。

- HTTPS/SSL支持

  HTTP服务器对安全套接层(SSL)协议完全支持。HTTPS使用SSL作为普通HTTP层的一个子层，以此来加密和解密HTTP的请求和响应。HTTPS使用443服务端口，而HTTP使用80端口。

- LDAP支持

  轻量级目录访问协议(LDAP)规定了一种简化的方式，用一种异步的客户端/服务器类型的协议从一个符合X.500规范的目录下获取信息。

- 证书验证

  作为对SSL支持的一部分，HTTP服务器可以使用证书验证机制，并且作为一个证书的授权机构。

- 代理支持

  HTTP服务器可以作为一种代理服务器。然而您不能使用快速响应缓存加速器(FRCA)。

13.3.3 文件缓存

使用如下可能存在的文件缓存机制可以显著地提高系统性能：

- HTTP服务器缓存HFS文件
- HTTP服务器缓存z/OS数据集
- z/OS UNIX缓存HFS文件
- 快速响应缓存加速器(FRCA)

13.3.4 插件代码

WebSphere HTTP服务器插件是一份可以运行在多种Web服务器上的代码，这些服务器包括：IBM HTTP Server，Apache，IIS，Sun Java™ System。请求通过插件被传递，这些请求在插件中是基于一份配置文件来被处理的。

插件是一份WebSphere提供的代码，它可以运行在不同的HTTP服务器之上。这些HTTP服务器可以是z/OS上的IBM HTTP服务器。当处理任务进入到HTTP服务器时，HTTP服务器配置文件(httpd.conf)中的指示命令被用来做出一个如下的决定：该请求任务应该由HTTP服务器来处理还是应该交给插件本身来处理。

428

一旦进入插件，处理请求的逻辑将由插件的配置文件决定，而不是HTTP服务器的配置文件。这个配置文件的默认名称是plugin-cfg.xml。将要处理该请求的后端应用程序服务器的信息被定义在这个文件中。这个文件是由WebSphere应用服务器所创建并且不需要修改，尽管您也可以灵活的来配置它。

需要注意的是：通常情况下，插件是用来为HTTP服务器提供一些扩展功能的。图13-5是说明了其用途的一个例子，尽管还有许多不同的插件，他们也可以被配置用来帮助您定制系统的Web运行环境。另一种比较流行的插件是轻量级目录访问协议(LDAP)，它被用来用作安全验证。

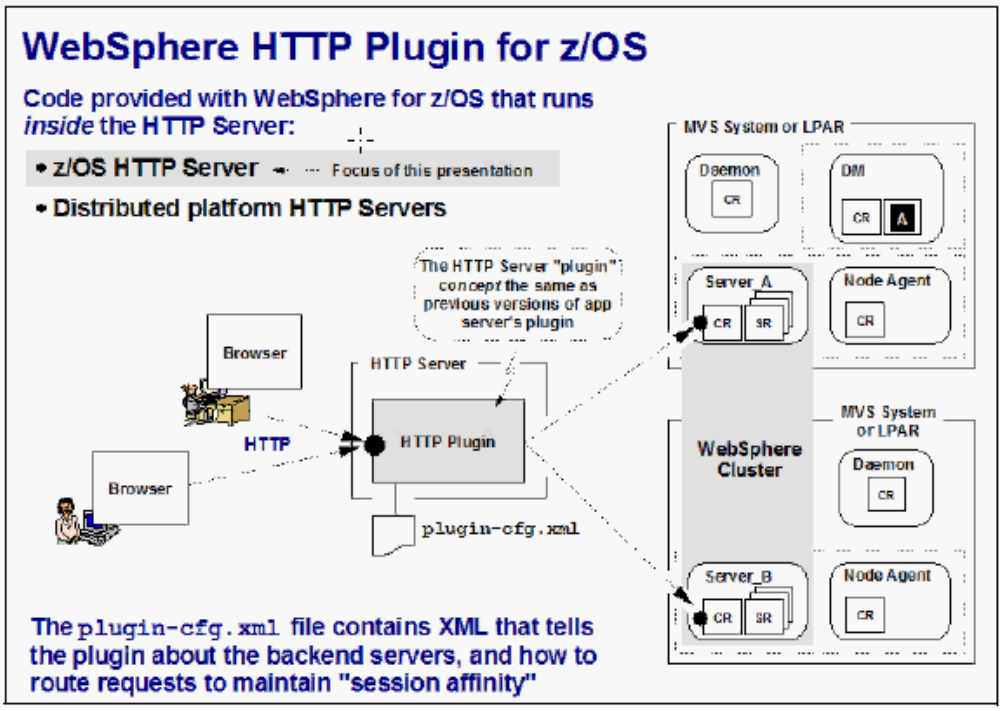


图13-5 一个插件的例子

429

13.4 总结

z/OS提供了HTTP服务器用以处理静态和动态的Web页面。HTTP服务器支持WebSphere插件(它用来处理EJB容器和J2EE)，安全机制和文件缓存。这些特性使它可以更好地处理动态Web页面。

本章中的重要术语		
通用网关接口(CGI)	动态(dynamic)	快速响应缓存加速器(FRCA)
HTTP	J2EE	轻量级目录访问协议(LDAP)
安全套接层(SSL)	静态(static)	

13.5 复习题

为了帮助您检测对本章的理解程度，请完成下列问题：

1. 列出三种服务器模式。
2. 解释静态和动态Web页面的含义。
3. 对于下面每一功能组，列出至少他们所包含的两种功能：基本，安全和缓存。

13.6 练习

使用ISHELL或者OMVS shell完成这个练习。同时，您需要知道：

- HTTP服务器的配置文件httpd.conf的位置
- IP地址或者HTTP服务器的名称

完成如下的步骤并回答问题：

1. 浏览安装在z/OS上的HTTP服务器产品的配置文件httpd.conf。Web文档(F “URL translation rule”)被存放在什么目录下面？并且应该用哪个端口(F “Port directive”)？
2. 打开Web浏览器窗口，显示HTTP服务器下面的类。WebSphere是如何嵌入这个HTTP服务器的(F “Websphere”)？
3. 使用OEDIT命令在Web文档目录下建立一份HTML文档。将它命名成youridtest.html。这有一份现成的例子供参考：


```
<!doctype html public “//W3//Comment//EN”>
<html>
<head>
<META content=“text/html; charset=iso-8859-1”>
<title> This is a simple HTML Exercise</title>
</head>
<body bgcolor=“#FFFFFF”>
<p>Hello World
```

```
</body>
</html>
```

4. 打开Web浏览器查看您的HTML文档，例如：

   www.yourserver.com/yourtestid.html

   如何安装您自己的CGI？

5. 检查httpd.conf文件。HTCounter CGI选项”Date and Time”是否被开启了？如果是，修改您的youridtest.html文件，将如下一行代码加入到HTML的body段中：

   ```
   <img src=“/cgi-bin/datetime?Timebase=local”>
   ```

   保存文件。现在显示的是什么？

14

第 14 章 z/OS 上的 WebSphere 应用服务器

目标：作为主机的专业人员，您应该了解如何在z/OS上部署一个Web应用程序并且应该了解如何使z/OS具有处理Web事务的能力。

在本章结束之后，您应当能够学会：

- 列出J2EE应用程序模型的六种性质。
- 给出三个在z/OS下运行WebSphere应用服务器的原因。
- 说出三个连接CICS，DB2和IMS的连接器。

14.1 什么是 z/OS 的 WebSphere 应用服务器？

随着越来越多的企业将他们的应用程序移植到Web环境，企业面临着一个难题，如何来支持和管理新的基于Web的联机事务，同时还要面对越来越多的传统的事务处理，比如批处理。

WebSphere应用服务器是一个拥有多种功能的，强大的，基于Java2企业版(J2EE)的，基于Web服务技术的应用系统。z/OS上的WebSphere应用服务器提供了标准的软件开发包来支持J2EE的实现，用户可以利用WebSphere应用服务器提供的API来实现他们的J2EE的应用。就像上文所说的那样，WebSphere应用服务器是一个Java应用程序的开发和部署的环境，它面向基于开放标准的技术规约，可以支持这些规约下的所有主要功能，例如，servlet，Java服务器网页(Java server pages，JSP)和企业级Java Bean(EJB)，它提供了对服务和接口的进行集成的最新技术。

z/OS上的WebSphere应用服务器的运行环境与z/OS提供的一些特性与服务进行了高度的集成。应用程序服务器可以和z/OS操作系统上的所有主要子系统进行交互，例如，DB2，CICS和IMS。此外，它还具有一些扩展的属性，例如，安全机制，性能监控机制，可扩展机制和恢复机制。WebSphere应用服务器同时还提供了一些高级的管理和工具化的功能，因此用户可以将它来同任意的数据中心或者服务器环境进行集成。

WebSphere应用服务器是一个电子商务应用程序的部署环境，它提供了一些基于开放标准的技术，例如，CORBA，HTML，HTTP，IIOP和J2EE规范(servlet， JSP和企业级Java Bean)。它提供了基于J2EE标准实现的所有Java API。

当外来任务到达时，WebSphere应用服务器会在它的控制器的地址空间自动启动一个服务域。如图14-1所示，一个应用程序服务器的实例由一个控制域(CR)和一个或者多个的服务域(SRs)所组成。

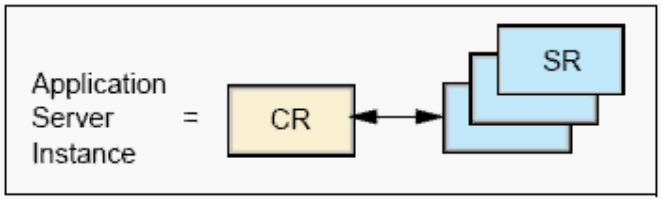


图14-1 一个应用程序服务器的实例

z/OS上的应用程序服务器支持两种配置方式：基本(Base)部署和网络(Network)部署。每一种配置从本质上来说都使用了相同的体系架构层次，由服务器(server)，节点(node)，和单元(cell)组成。然而，单元和节点只在网络部署的配置中有着很重要的作用。

14.2 服务器

服务器是主要的运行时组件，这是用户的应用程序实际的执行环境。服务器提供了容器和服务专门用于运行特定的Java应用程序的组件。每一个应用程序服务器都运行在属于它自己的Java虚拟机(JVM)上。

根据配置，服务器可以分开工作或者组合在一起工作：

- 在基本部署配置中，每一个应用程序服务器的功能是作为一个单独的实体来工作。在服务器之间没有负载的分配或者统一的管理。
- 在网络部署配置中，多个应用程序服务器被一个中心管理单元所维护。

此外，用户可以通过服务器的集群来实现对于负载的分配。

提示：一种称作JMS(Java消息服务器)的特殊的应用程序服务器并不包含在本书所讲的内容中。

14.3 节点(和节点代理)

节点是WebSphere管理下服务器进程的一种逻辑上的集群，这些服务器进程共享一种配置和操作控制。一个节点通常和一个物理上安装的应用服务器想关联。

当您接触到更高级的应用服务器的配置的时候，您还要了解额外的一些概念，它们是，从一个共有的管理服务器配置多个节点和多个节点之间的负载分配。在这些集中化管理配置中，每一个节点都有一个节点代理(node agent)，这些节点代理和部署管理器(Deployment Manager)协同来完成对于节点的管理任务。

14.4 单元

单元是一组节点在单一的管理域下的集群。在基本配置中，一个单元只含一个节点。在这个节点下可能有多个服务器，但是每一个服务器的配置文件都单独进行存放和维护，并且这种维护是基于XML配置管理的。

通过进行网络部署配置，一个单元可以由多个节点组成，并且所有的这些节点都由一个中心节点进行管理。在单元中的所有节点的配置文件和应用程序文件都被集中维护在单元管理器的配置库中。这个集中配置库由部署管理器来维护，并且被同步到每一个节点的本地化备份中去。

在应用程序服务器的地址空间中，有一个容器(container)的概念，对于不同的应用程序执行组件，容器分别为他们提供了独立的运行环境。例如，一个EJB容器，被用来运行企业级Java Bean组件。另一种容器，Web容器，被用来运行Web相关的组件，例如，HTML，GIF文件，servlet，和JSP。这些容器在JVM下共同构成了服务器的运行环境。

14.5 z/OS 上的 J2EE 应用程序模型

z/OS上的J2EE应用程序模型和其他平台是完全一致的，它同样遵守SDK所定义的规约，有如下的一些特点：

- 功能性 – 满足用户的需求
- 可靠性 – 在持续不断变化的环境中仍然可以运行
- 实用性 – 提供了对于应用程序功能的简便访问方式
- 高效性 – 可以广泛地使用系统资源
- 可维护性 – 可以很容易的对应用程序进行修改
- 可移植性 – 可以将应用从一个环境中移植到另一环境中

z/OS上的WebSphere应用服务器支持四种应用程序设计的基本模型：基于Web的计算，企业集成计算，多线程分布式的商业计算，面向服务的计算。所有这些设计模型的关注点集中在将应用程序的逻辑实现同底层的运行环境相分离，也就是说，从物理上的拓扑和外部对于信息系统的访问和应用程序的编程模型是无关的。

z/OS WebSphere应用服务器支持的J2EE编程模型对于新的业务需求下的应用程序的开发来说无疑是更为简便的，因为它将程序的实现细节同底层的运行环境相分离。它支持组建的部署和J2EE提供的面向服务的编程模型。

14.6 在 z/OS 上运行 WebSphere 应用服务器

WebSphere应用服务器是作为一个标准的子系统运行在z/OS操作系统之上。因此它继承了主机平台所具有的所有功能和特性，例如它可以同时并行处理上上百种不同种类的工作负载，并且可以满足用户自定义的服务级别目标。

14.6.1 工作负载的合并

根据前面章节的讨论，主机可以被用来合并多个单独服务器上的工作负载。因此，如果有一个很大的管理上的开销或者出于多个单独服务器的物理容量上的担心，主机可以提供一个单独的服务器环境来管理这些工作负载。对于执行过程中使用主机提供的服务的应用程序，它可以提供单一的管理视图，统一进行性能监控以及恢复。

几个应用程序服务器可以很容易的被迁移到主机上的一个逻辑分区内，因此对于管理和监控来说十分方便(逻辑分区，即LAPR，在第35页的"主机硬件系统与高可用性"章节中有所介绍)。整合同时还允许通过使用工具来收集度量值，这极大地方便了对于容量的分析。

14.6.2 z/OS 上的 WebSphere 的安全机制

基于z系列硬件和软件结合的安全机制以及J2EE标准所提供的安全机制，对非法入侵有着很好的防御效果。产品的安全机制是一个分层架构，它构建于操作系统平台，Java虚拟机(JVM)和Java2的安全机制之上。

z/OS上的WebSphere应用服务器将基础架构和机制结合在一起来，基于工业标准按照端对端的安全视角来保护一些敏感的J2EE资源和一些受管理的资源。

这个开放的架构使得WebSphere可以和以下所有的主机企业信息系统进行安全连接和互操作：

- CICS交易服务器(TS)
- DB2
- Lotus® Domino®
- IBM Directory

437

WebSphere应用服务器将RACF和WebSEAL安全代理(可信赖联合拦截器)集成在一起，以此提供一个统一的，基于策略标准的和基于许可的安全模型，这种模型可以保护所有定义在J2EE规约中的Web资源和企业Java Bean组件。

14.6.3 持续可用性

z/OS上的WebSphere应用了z系列平台的内部错误检测和修正功能。z/OS上的WebSphere有恢复终止管理功能，它可以检测，隔离，更正和恢复软件上的错误。z/OS上的WebSphere可以基于服务等级协议来区分和制定工作任务的优先级。它提供了集群的功能并且可以在不终止软件组件的情况下对其进行变更，例如资源管理器。

对于一个重要的应用，z/OS上的WebSphere实现了z/OS上的故障管理设备的功能，称作自动重启管理器，简称为ARM。这个设备可以用来检测应用中的故障，并且当故障发生时重启服务器。WebSphere使用ARM来恢复应用程序服务器(服务域)。每一个运行在z/OS系统之上的应用程序服务器都注册在一个ARM的重启动组中。

z/OS上的WebSphere拥有一种称作集群的特性。集群技术在WebSphere高可靠性的解决方案中使用的极为广泛，如图14-2所示。

一个集群由多份相同的组件的拷贝所构成，它可以保证当异常出现的时候至少有一份拷贝可以被用来处理服务请求。通常来说，集群可以作为一个单元来工作，在这个单元中各个拷贝之间可以相互协作，以此来保证请求可以被任何一个可以处理此请求的拷贝所接受。

当高可靠性解决方案的设计者确定集群中的成员数量和布置方式时，他们可以制定服务级别。z/OS上的WebSphere可以管理那些被需要用来建立服务级别的集群。当WebSphere集群同其他的集群技术相结合的时候可以显著地提高可用性的

438

服务级别。

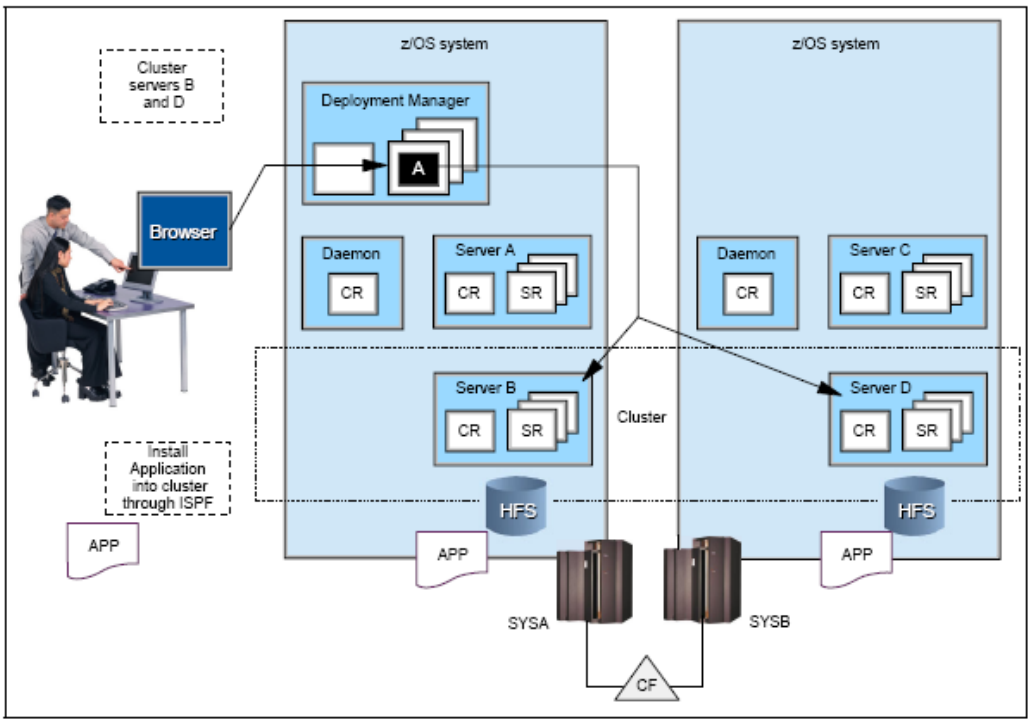


图14-2 在一个单元中的服务器集群

一个WebSphere应用服务器的集群由独立的集群成员所组成，每一个成员都含有相同应用程序的集合。在WebSphere应用服务器集群的前面是一个工作任务负载的分配器，它可以将工作任务路由到独立的集群成员当中去。

集群可以被垂直部署在一个LPAR中(也就是说，两个或者两个以上的成员在一个z/OS系统中)，或者也可以被水平的部署在多个LPAR中，这样可以预防其中一个LPAR中成员失灵的时候，在其他的LPAR有候补的成员可以替换，由此可以获得最高的可靠性。

工作任务在此种情况下可以被其他集群中的成员所接管。同样可以采取一种混合配置的方式将垂直和水平配置相结合，使得在WebSphere集群中同时包含垂直和水平部署的成员(如图14-3)。

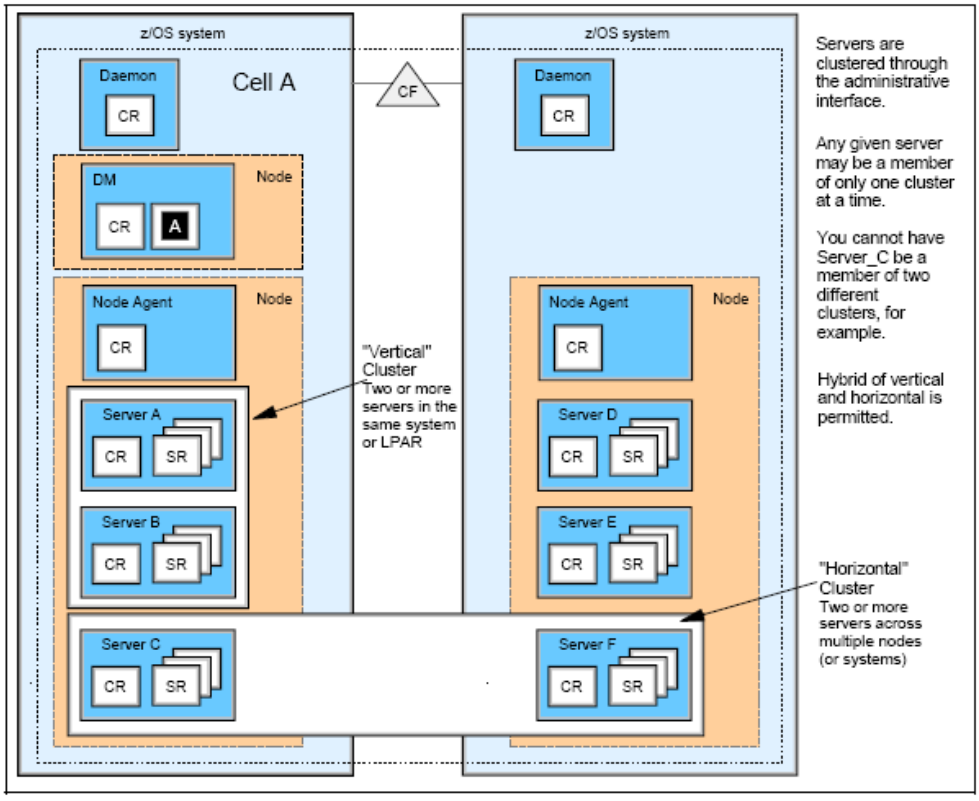


图14-3 垂直和水平部署的集群

您或许可以考虑一下什么时候用垂直部署的集群什么时候用水平部署的集群？您可以用垂直集群来检查在一个独立系统中调度的效率。在垂直集群中，服务器会对共享的资源进行竞争。

14.6.4 性能

性能主要取决于应用程序的设计和编码的质量，而不是运行平台硬件的性能，一个存在缺陷的应用程序不论在z/OS平台还是在其他平台都不会有很好的性能表现。

z/OS上的WebSphere应用服务器充分利用了主机硬件上的特性，也利用了软件上的特性，将工作负载管理规划，动态LPAR配置和并行系统综合体的功能结合在一起。特别地，它使用了z/OS工作负载管理(WLM)的三种不同功能：

- 路由功能

  WLM路由服务功能可以根据测量当前系统的使用率(也被称为性能索引-PI)来将客户端的请求分配到指定的服务器系统上。

- 队列功能

  WLM队列机制被用来将工作任务从控制域到一个或多个服务器域进行派发。可以将工作管理器作为队列管理器注册在WLM中。这会告诉WLM这个服务器

将会用WLM所管理的队列来将工作任务路由到其他的服务器中，这可以使WLM来管理服务器的地址空间，以此来完成对这个工作任务的性能调优目标。

- 优先级功能

  应用程序服务器可以通过启动和停止服务器域来设置工作任务的优先级。这可以使WLM来管理应用程序服务器的实例，以此来实现应用的业务目标。

WLM对每一个服务级别维护了一个性能索引(PI)，它可以测量实际性能与目标性能之间的差异。因为同时存在了不同的目标类型，WLM需要一些方法来比较在一个服务级别中的一个工作任务和另一个工作任务性能的好坏。服务级别(Service Class，SC)是被用来描述在一个工作负载中具有相同性能特性的一组工作任务。

14.7 在 z/OS 上应用服务器的配置

z/OS上应用服务器的配置包括如下几方面：

- 基本服务器节点
- 网络部署管理器

14.7.1 基本服务器节点

基本应用程序服务器节点是z/OS上的WebSphere应用服务器中的最简单的运行结构。它由一个应用服务器和守护服务器(一个节点和一个单元)所组成，如442页的图14-4所示。所有的配置文件和定义信息被保存在为这个基本服务器建立的HFS目录下。守护服务器(deamon server)是一个特殊的服务器，它有一个控制域(Controller Region，CR)。z/OS上的WebSphere的系统架构要求对于每一个z/OS系统或者LPAR，在一个单元中要有一个守护服务器。

每一个基本应用服务器节点都包含对它自己单元域的管理功能和一个独立的配置库。因此，您可以有许多个基本应用服务器，而每一个都和其他的相隔离，他们都根据特定的业务需要制定自己的管理规则。

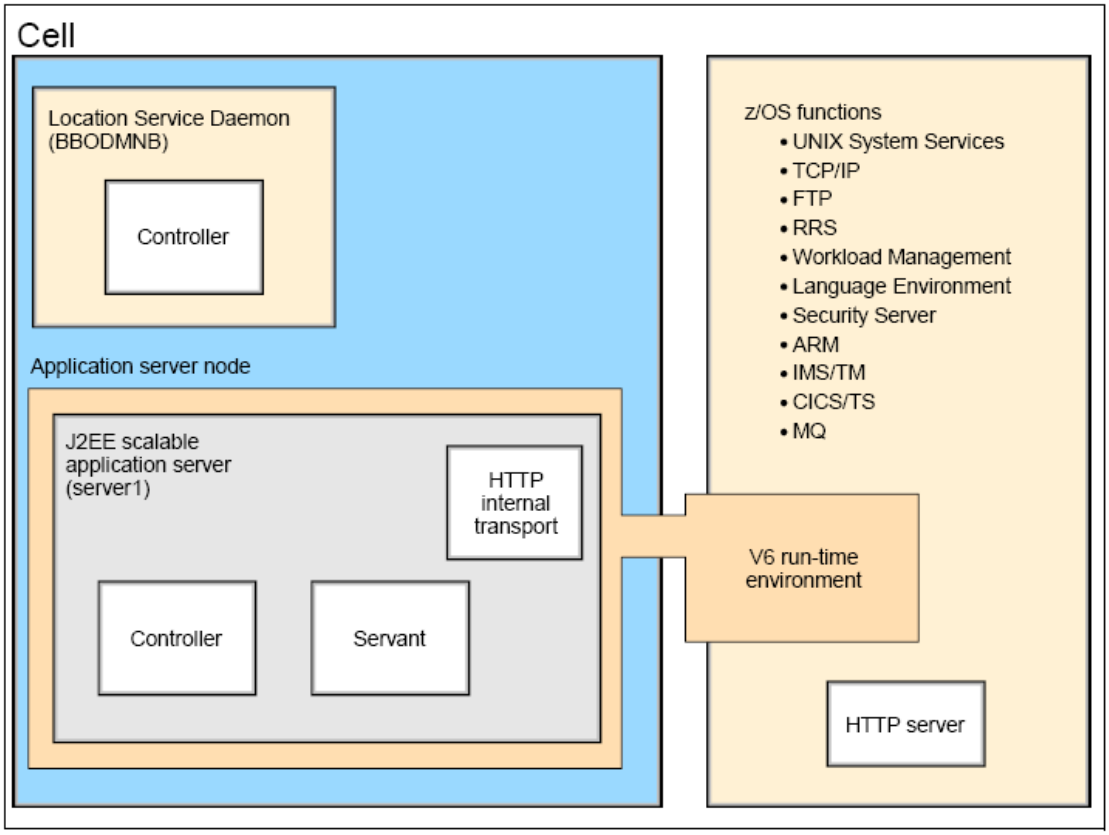


图 14-4 基本服务器节点

14.7.2 网络部署管理器

网络部署管理器(443页，图14-5)是对基本应用服务器的扩展。它允许系统可以从一个中央位置来管理多个应用程序服务器。在这里，应用程序服务器和节点绑定在一起，而多个节点隶属于一个单元。通过部署管理器，垂直和水平系统以及分布式应用都可以被很容易地进行管理。

网络部署管理器也可以管理每一个节点的配置库，可以进行诸如创建，委会和移除配置库的操作。系统用抽取/修改的方式来更新配置。

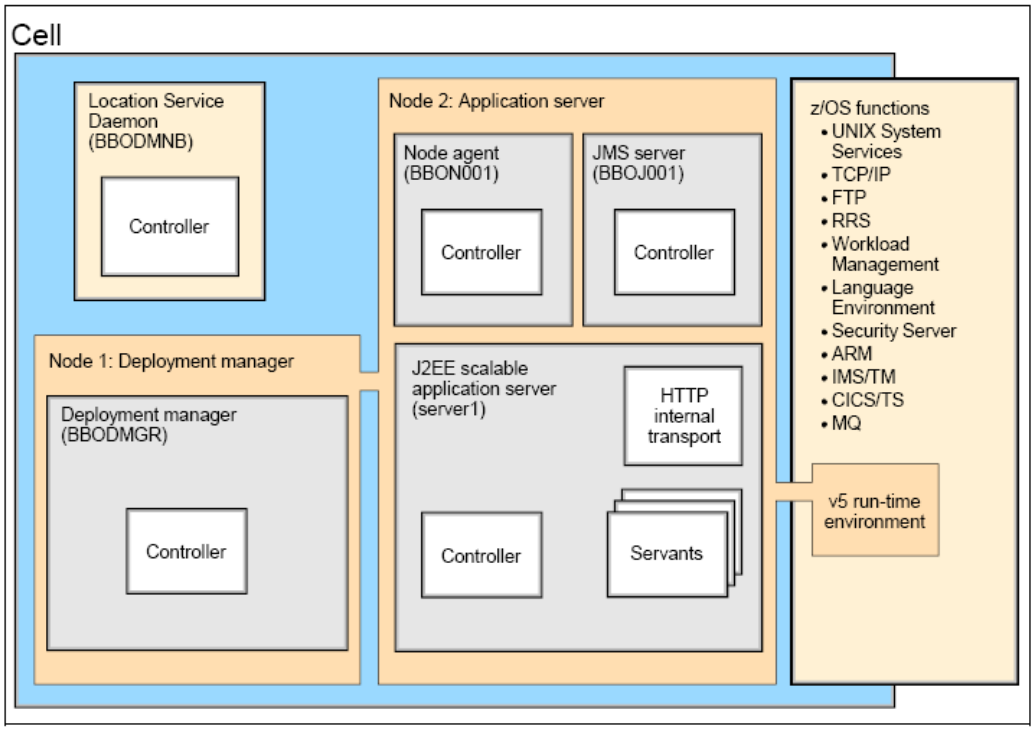


图14-5 网络部署管理器

14.8 企业信息系统连接器

目前企业应用中的一个重要的需求是，应用程序可以和本应用所在服务器进程之外的资源进行交互而且可以高效地利用这些资源。同等重要的是服务供应商可以通过连接和使用企业的资源将他们自己的解决方案植入到企业应用中。

一个应用应该可以访问多种类型的资源，这些资源作为应用程序可以不在相同的机器内。因此，对资源的访问始于一个连接，这个连接是应用程序到资源的一个路径，而这个路径可能是另一个交易管理器或者是数据库管理器。

Java程序对大范围的后端资源的访问是通过资源适配器来完成的。资源适配器是一个系统级的软件驱动程序，它可以被集成到应用程序服务器中并且Java应用程序通过它可以连接到不同的后端资源。

以下的一些考虑因素对于所有的连接来说都是通用的：

- 建立一个连接的开销是非常昂贵的。建立连接的时间与建立连接之后实际使用的时间相比可能需要更久。
- 连接必须是安全的。这个通常是要由应用程序和服务器以及所要访问的资源共同来完成。
- 连接的性能必须要很好。性能对于一个成功的应用来讲至关重要，而且它也是一个应用总体表现的重要功能。
- 连接必须可被监测而且有很好的诊断功能。一个连接诊断功能的优劣取决于

服务器和连接资源的状态信息。

- 连接和操作资源的方法。不同的数据库架构需要应用程序服务器选择不同的访问方法。
- 服务的质量，它现在成为衡量访问应用程序服务器外部资源的好坏的一个因素。当一个实际的应用在交易中操作数据的时候，可能需要满足ACID属性(原子性，一致性，隔离性和持久性)。

企业的资源通常都是一些较为古老的资源，他们都是商业组织很久以前所开发，现在暴露给应用程序服务器的外部接口。每一种资源都有它自己的连接协议和访问此种资源的接口集合。因此，这些资源应该具有"适应性"，也就是说它们可以被应用服务器内部的JVM进程所访问。

WebSphere应用服务器拥有一些同其他的z/OS子系统(例如CICS，DB2和IMS)进行交互的设施。这种交互的功能是通过资源适配器和连接器所实现的。对后端企业信息系统(EIS)资源的访问极大地在原有的业务功能上扩展了应用程序服务器的功能，提供了更强大的业务处理能力。

J2EE连接架构(JCA)定义了应用程序，连接器和部署应用程序的服务器之间的系统规约。应用程序有一个称作资源适配器的组件。资源适配器包含在应用程序的代码中，用来处理同应用程序开发者创建的连接器之间的通讯。

从编程的角度看，这意味着编程人员可以用一个统一的接口来从企业信息系统中获得所需的数据。资源适配器会识别出差异点并且提供一种可靠的编程模型，这种模型是完全独立于EIS系统本身的实现方式和通讯要求的。

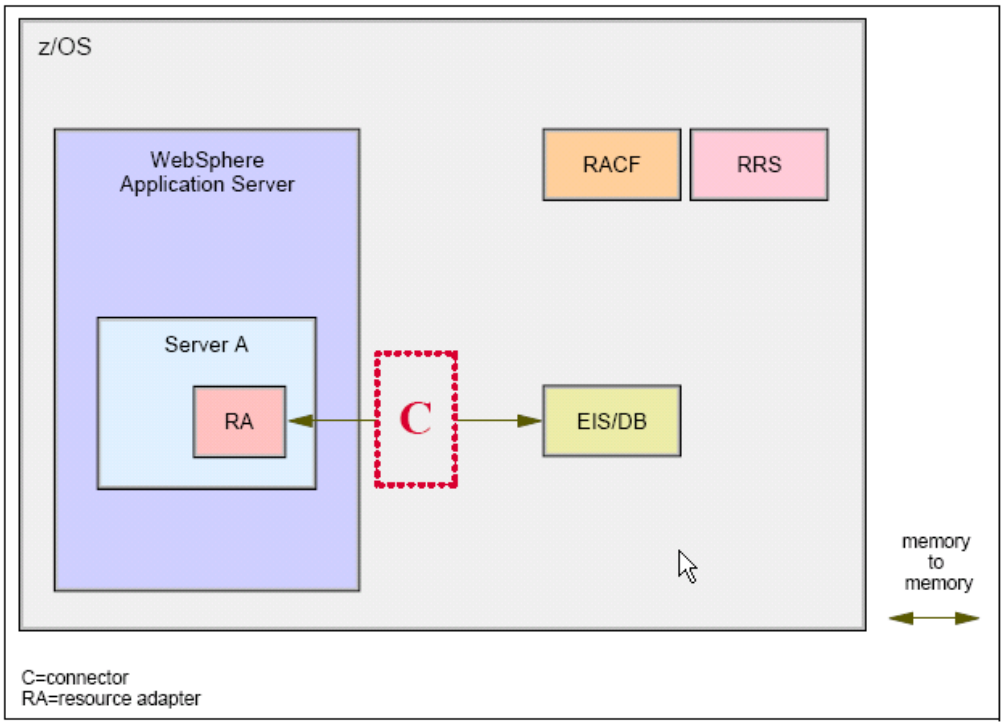


图 14-6 EIS连接器的基本架构

14.8.1 z/OS 上的连接器

z/OS上的WebSphere提供了如下的一些连接器，他们可以使z/OS上的Web应用程序同主机上的中间件产品进行交互，例如，CICS，IMS和DB2：

- CICS交易网关(CICS Transaction Gateway，CTG)
- IMS连接器(IMS Connect)
- 446页的”DB2 JDBC”

CICS交易网关(CTG)

客户信息控制系统(CICS)拥有一个CICS交易网关用于建立从应用程序服务器到CICS的连接。CTG提供了Java程序与CICS应用交易之间的接口。它是一组客户端和服务器的软件组件，包含了多个服务和工具，用于从应用程序服务器访问CICS。CTG在servlet和EJB中使用特定的API和协议来访问CICS交易管理器所提供的服务和功能。

IMS连接器

IMS连接器是一个TCP/IP连接服务器，它可以使应用程序服务器的客户端程序同IMS开放事务管理器访问(OTMA)进行信息间的交换。这个服务器提供了TCP/IP客户端和IMS数据库之间的通讯链路。它支持多TCP/IP客户端到多数据库的访问。为了保证通过TCP/IP传输信息的安全性，IMS连接器提供了对安全套接字层(SSL)的支持。

IMS连接器也在应用程序服务器的客户端与数据库及IMSplex资源之间，或者本地选项客户端与数据库及IMSplex资源之间进行路由。使用TCP/IP连接从TCP/IP客户端，或者使用z/OS程序调用(PC)的本地选项客户端收到的请求信息通过跨系统耦合设施(XCF)的会话被传递给数据库。IMS连接器收到数据库的响应消息然后再把它们传回给先前的TCP/IP客户端或者本地的选项客户端。

IMS连接器支持通过socket调用的TCP/IP客户端的通信，但是它也支持通过不同的输入数据流格式的任意的TCP/IP客户端的通信。用户自定义的消息出口可以在IMS连接器的地址空间内执行，它可以将z/OS安装系统的消息格式转成OTMA的消息格式，然后IMS连接器将消息发给IMS。用户自定义的消息出口也可以将OTMA的消息格式转成z/OS安装系统的消息格式，然后将消息发回给IMS连接器。随后IMS连接器将输出信息发给客户端。

DB2 JDBC

Java数据库连接(JDBC)是一个应用程序编程接口(API)，Java编程语言用JDBC来访问不同种类的表格数据，并且可以访问一些层次化的系统，像IMS。JDBC规约是由Sun Microsystems公司和关系数据库厂商(像Oracle和IBM)共同制定的，以此来保证Java应用程序在不同数据库平台上的可移植性。

JDBC接口不必属于”连接器”的范畴，因为对于它的实现并不需要一个独立的地址空间。这个接口是以一种Java语言的形式呈现的，就像一个Java的类，但是并没

有提供对于接口的具体实现。对于JDBC，真正实现JDBC接口的是数据的供应商，供应商提供的接口被称为"驱动程序"。此种方式为程序的移植带来了方便，因为所有使用JDBC的访问方式都可以通过标准的参数来进行标准化的调用。因此一个应用程序在开发的时候可以几乎忽略使用的数据库的类型，因为所有和数据库平台相关的代码都被封装在JDBC驱动程序之中了。

这样带来的结果是，JDBC所能提供的功能和不能提供的功能必须拥有灵活性，这仅取决于一个事实，即不同的数据库系统有着不同的功能级别。JDBC驱动提供了物理代码，这些代码实现了定义在规约中的对象，方法和数据类型。JDBC标准定义了四种不同类型的驱动，编号为1到4。他们之间的区别取决于驱动的物理实现和它同数据库之间的通讯方式。

z/OS只支持类型2和类型4的驱动程序，如下所示：

- 类型2(Type 2)

  JDBC API调用相应平台上的相应数据的代码来访问数据库。这是最常用的驱动程序类型，并且拥有最佳的性能。然而，因为其驱动程序代码是平台相关的，对于每一种平台都需数据库供应商要提供不同版本的代码。

- 类型4(Type 4)

  类型4驱动是完全基于Java实现的，可以直接使用数据自身的协议来访问目标数据库。(对于DB2来说，协议是DRDA)。因为驱动完全用Java实现，它可以不加改动地被用于任何的平台，只要这个平台支持DBMS协议，因此也可以使应用程序跨平台的来使用它，而不需要做任何的改动。

  一个运行在WebSphere应用服务器上的Java应用程序，同支持两阶段提交的通用类型4的JDBC驱动程序进行通信，而驱动程序通过DRDA同远端的数据库直接通信。通用类型4驱动实现了DRDA应用请求者的功能。

为了访问z/OS上的DB2，IBM提供了类型2驱动程序和一个将类型2和类型4的JDBC实现结合起来的驱动程序。通常来说，JDBC类型2驱动程序被用于和目标DB2子系统运行于同一z/OS系统中的Java程序。JDBC类型4驱动程序被用于和目标DB2子系统运行在不同的z/OS系统中的Java程序。

本章中的重要术语		
单元(cell)	服务级别(SC)	通用网关接口(CGI)
企业信息系统(EIS)	Java管理扩展(JMX ™M)	Java2企业版(J2EE)
服务器域(SR)	集群(cluster)	节点(node)

14.9 复习题

为了帮助您检测对本章的理解程度，请完成下列问题：

1. 列出J2EE应用程序模型的六种性质。
2. 举出三个在z/OS下运行WebSphere应用服务器的原因。
3. 说出三种连接器的名称。

4. z/OS上的HTTP服务器和WebSphere应用服务器的主要区别是什么？

448

5. 在什么情况下连接器不应该被使用？

15

第 15 章 消息和队列机制

目标：作为主机的专业人员，您应该了解z/OS中的消息和队列机制。这些功能在异构平台间的应用程序的通讯方面有着广泛的应用。

在本章结束之后，您应当能够：

- 解释为什么使用消息和队列机制。
- 描述消息传递的异步流程。
- 解释队列管理器的功能。
- 列出三种z/OS相关的适配器

15.1 什么是 WebSphere MQ

如今绝大多数企业的IT系统都是分布部署在不同生产厂商所提供的平台上的，这样就会使得跨系统间的通讯和数据共享变得十分的困难。许多这样的企业还需要和供应商和客户的系统进行数据的共享，而通常供应商和客户的系统和这些企业的系统又是完全不同的。所以我们希望拥有一种消息处理工具，它可以很方便地在跨系统的条件下将一种系统中接收到的数据发送到另一种系统中。

IBM WebSphere MQ可以通过在应用程序和Web服务之间的消息传递来实现应用系统的集成。它可以被用于超过35种的硬件平台上，并且可以利用Java，C，C++和COBOL来实现点对点的消息传递。有四分之三的企业在决定购买实现应用系统间消息通讯的系统时购买了WebSphere MQ。在最大规模的系统安装中，MQ每天可以传递超过2.5亿条消息。

在分布式的环境中，不同系统上的不同数据库的数据需要保持同步，但是以协议的方式来协调更新，删除以及其他的一些操作又是极为有限的。混合环境很难保持一致，通常需要复杂的编程来集成他们。

消息队列，以及管理他们的软件，例如z/OS上的IBM WebSphere MQ，可以实现程序对程序端的通信。对于在线的应用程序来说，消息和队列机制意味着：

- 消息机制意味着程序之间的通信是通过向彼此发送消息(或者数据)来实现的，而不是通过直接的程序调用。
- 队列机制意味着消息可以被存放在存储区的队列中，这样程序就可以以不同的速度，在不同的时间和不同的位置相互独立的运行，而不需要建立一个在他们之间的逻辑上的连接。

15.2 同步通讯

图15-1展示了在同步通讯模型下的，程序对程序的基本通讯机制。

程序A准备好了一条消息，并且把它放在队列1中。程序B从队列1中取得这条消息，然后处理它。对于程序A和程序B，两者通过使用应用编程接口(API)来向一个队列中放置消息，并从这个队列中取得消息。WebSphere MQ API被称作消息队列接口(MQI)。

当程序A在队列1中放置一条消息时，程序B可能不在运行。在程序B启动并且准备获得此消息之前，这个队列安全地保存着这条消息。同样地，当程序B从队列1中取得消息时，程序A可能已经不在运行了。应用此种模型，可以使得两个程序并不需要同时运行来实现彼此间的通讯。

但是这种模型在设计上存在一个很明显的问题，就是程序A在可以进行其他的处理之前需要等待多长的时间。这种设计在某些情况下是适用的，但是当等待时间太长的时候，就不再适用了。应用异步的通讯机制可以很好地解决上述问题。

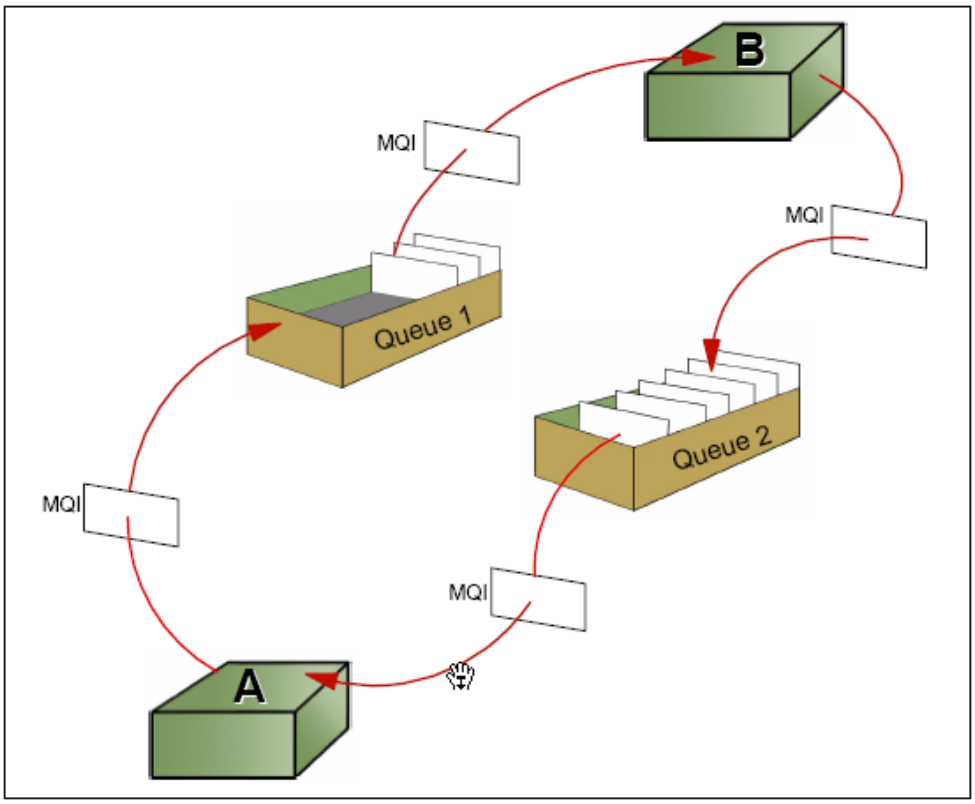


图15-1 同步式应用程序设计模型

15.3 异步通讯

使用异步通讯模型，程序A将消息放到队列1中等待程序B的处理，但是实际上是程序C，和程序A异步运行，并从队列2中取得从B发来的响应消息，并且处理它们。一般地，程序A和程序C是同一个应用系统的一部分。您可以从图15-2中了解异步事务流的执行过程。

451

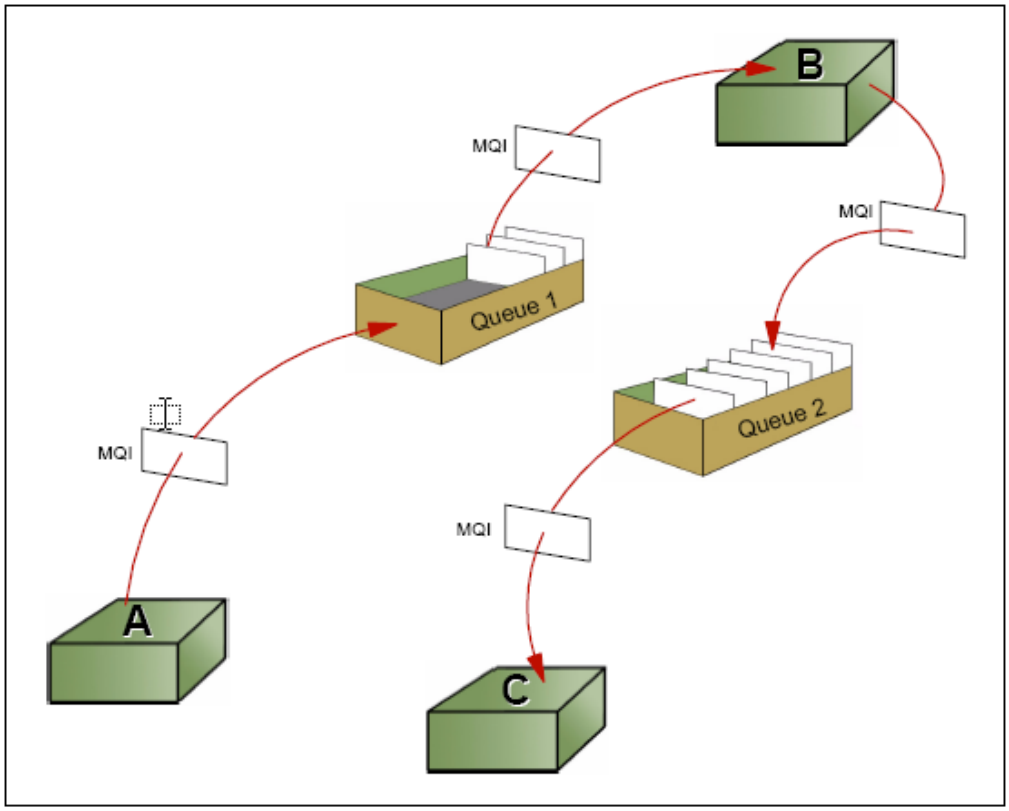


图15-2 异步式应用程序设计模型

WebSphere MQ本身支持这种异步消息处理模型。程序A可以持续地将消息放进队列1中，而不会因为等待每一条消息的响应而被阻塞。它可以继续向队列1中放置消息即便程序B执行失败。如果这样，在程序B重新启动之前，队列1可以一直安全地保存这些消息。

在异步模型的变体中，程序A可以将一组消息序列放置在队列1中，可以选择继续处理其他的任务，然后再回到队列1中取得并且处理先前消息的响应。WebSphere MQ的这种性质，使得进行通讯的应用程序可以不需要同时处于活动的状态，这种性质被称为执行时间上的独立(time independence)。

15.4 消息的种类

452

WebSphere MQ的消息分为以下四种：

数据报消息	不需要响应的消息。
请求消息	需要响应的消息。
应答消息	对于请求消息的响应消息。
报告消息	描述事件的消息，例如描述错误的发生，或者确认消息到达或消息发送。

15.5 消息队列和队列管理器

消息队列被用来存放程序发来的消息。他们是队列管理器定义的对象。

当应用程序将一条消息放到队列中的时候，队列管理器需要保证这条消息：

- 被安全地保存
- 可被恢复
- 发送一次，且仅仅一次，到接收程序

即便消息被发送到另一个队列管理器的队列中，以上对于消息的这种保证机制仍然适用，这也被称为WebSphere MQ的消息保障传递属性。

15.5.1 队列管理器

拥有和管理队列的软件组件被称为队列管理器(QM)。在WebSphere MQ中消息队列管理器被称作MQM，它为应用程序提供消息相关的服务，保证消息被放置在正确的队列中，可以将消息路由到其他的队列管理器中，并且提供了通用的编程接口来处理这些消息，这个通用编程接口被称为消息队列接口(MQI)。

当应用程序或者系统出现故障时队列管理器可以将消息保存起来，留作后续的处理。在通过MQI得到成功执行的响应之前，消息都会被保存在队列中。

队列管理器和数据库管理器有很多的相似之处。队列管理器拥有和控制队列就像数据库管理器拥有和控制他们的数据存储对象一样。他们都提供了访问数据的编程接口，同时提供了安全，授权，恢复和一些管理工具。

然而，他们之间仍然存在着一些关键的差异。数据库是被设计用来提供长时间的数据存储，同时提供了高级的数据检索机制，而队列并不是针对此而设计的。队列中的消息通常反映出一个业务流程并没有被执行完，它或许是一个没有满足的请求，一个没被处理的响应或者是一个没被读取的报告。457页的图15-4反映了队列管理器和数据库管理器中的事务执行流程。

15.5.2 消息队列的种类

消息队列的种类有很多，和本文最相关的有以下几种：

- 本地队列(local queue)

  如果一个队列被应用程序所连接的队列管理器直接管理，则称它为本地队列。它为使用同一队列管理器的程序储存消息。应用程序可以不必和队列管理器运行在同一台机器之上。

- 远程队列(remote queue)

如果一个队列被一个不同的队列管理器所管理，它就成为远程队列。远程队列并不是真正意义上的队列，它仅仅是一个定义在本地队列管理器中的远程队列的定义。程序不能从远程队列中读取消息。多个远程队列和一个传输队列结合在一起。

- 传输队列(transmission queue)

  传输队列是一个本地队列，它有一个特殊的目的：当消息被发送到一个不同的队列管理的队列中时，它被用来作为一个中间传递的媒介。传输队列对于应用程序来说是透明的，也就是说，他们被队列管理器启动队列(queue manager initiation queue)在内部使用。

  这是一个本地队列，队列管理器会向它里面写入一个触发消息(对编程人员是透明的)，前提是当另一个本地队列满足某种条件时，例如，当一条消息被写入一个空的消息队列或者一个传输队列中。WebSphere MQ有两种应用程序负责监控启动队列和读取触发消息，他们分别是触发监视器和通道启动程序。触发管理器可以根据消息来启动相应的应用程序。通道启动程序可以启动队列管理器间的消息传送。

- 死信队列/停用字母队列(dead-letter queue)

  队列管理器(QM)必须能够处理当它无法将消息发送的情况，例如

  - 目标队列已经满了
  - 目标队列不存在
  - 消息不允许被放置在目标队列中
  - 发送者没有被授权使用目标队列
  - 消息过大
  - 消息含有重复的序列号

只要上述情况中的一个发生时，消息就会被写入死信队列/停用字母队列。这个队列当队列管理器启动时就会被定义，并且每个队列管理器拥有一个。它被用来存放那些不能被派发的消息。

15.6 什么是通道？

通道是一个逻辑上的通讯链路。程序对程序的会话通讯方式要求在每一对相互通讯的应用程序之间存在一个通讯的连接。通道可以使应用程序间的通讯不受底层通讯协议的影响。

WebSphere MQ使用两种类型的通道：

- 消息通道

  消息通道通过消息通道代理(MCA)来连接两个队列管理器。消息通道是单向的，它包含两个消息通道代理(一个发送方一个接收方)和一个通讯协议。MCA将消息从一个传输队列传送到一个通讯链路，并从这个通讯链路传送到目标队列。对于双向通讯，需要定义一对包含发送方和接收方的通道。

- MQI通道

MQI通道将WebSphere MQ客户端同队列管理器连接起来。客户端并不需要有他们自己的队列管理器。MQI通道是双向的。

15.7 如何保证交易的完整性

在一个业务流程中，可能需要同时保持两个或两个以上的分布式数据库的数据一致性。WebSphere MQ提供了一种解决方案，可以使多个工作单元异步地执行，如图15-3所示。

图15-3的上半部分显示了两阶段提交的结构，WebSphere MQ的解决方案在下半部分显示，说明如下：

- 第一个应用程序对数据库进行写操作，并将一条消息放到队列中，同时设置一个同步点来提交对以上两种资源中的数据更改。这条消息包含了被用于更新另一个系统的数据库字段的数据。因为这个队列是一个远程队列所以消息仅仅是放在本工作单元的传输队列中。当工作单元被提交后，这条消息可以被负责发送的消息通道代理(MCA)所取得。
- 在第二个工作单元中，负责发送的MCA从传输队列中取得消息并把它发给负责接收的MCA(负责接收的MCA在第二个数据库所在的系统中)，然后负责接收的MCA将消息放置在目标队列中。因为WebSphere MQ的消息确保投递的特性，使得以上的操作可以十分可靠的完成。当这个工作单元被提交后，这条消息可以被第二个应用程序所取得。
- 在第三个工作单元中，第二个应用程序从目标队列中取得消息并且使用这条消息中的数据来更新数据库。

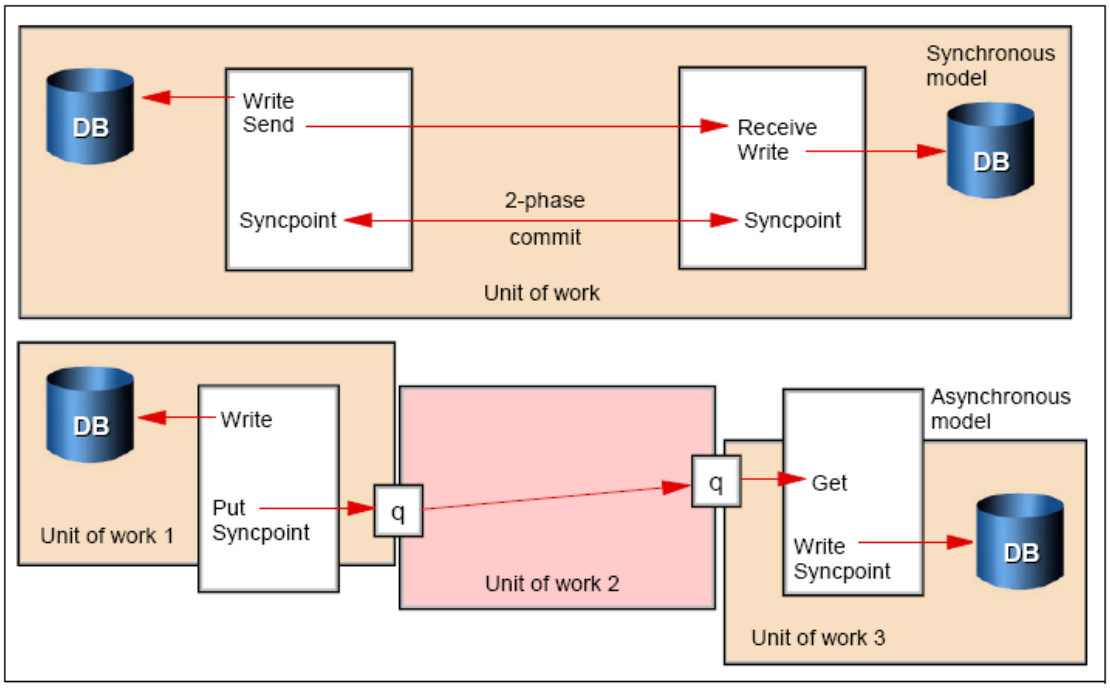


图15-3 数据一致性

工作单元1和3保证了交易的完整性，工作单元2中的WebSphere MQ的一次仅此一次的”确保消息投递”的特性，保证了整个业务交易的完整性。

对于一个更为复杂的业务交易，可能同时有多个工作单元参与到业务流的执行当中来。

15.8 消息和队列机制的应用示例

现在让我们回到早先的那个旅游中介机构的例子上，看一看消息机制是如何在假期预约的业务流程中发挥作用的。假定这个旅游中介机构必须为客户预订飞机票，宾馆客房和租用的汽车，并且所有的这些预订必须全部成功完成才能视为整个业务流程结束(如图15-4)。

通过消息队列管理器，例如WebSphere MQ，应用程序可以同时发送多个请求，它不需要等到一个请求的答复之后再发送下一个请求。消息可以被放置在三种队列的每一个中，这些队列分别为航空预订程序，宾馆预订程序和汽车租赁程序来服务。每一应用程序可以同其他的两个应用程序并行的执行它们自己的任务，然后将应答的消息放到应答队列中去。旅游中介机构的应用程序等待这些应答，然后整理生成一个最终的答案反馈给旅游中介机构。

用这种方式来设计这个系统可以提高系统整体的响应时间。尽管对于旅游中介机构的程序来说它可能只有在全部收到所有的应答之后才能继续下一步的处理，但是程序的逻辑中也可以加入一些在给定的时间内当一部分应答收到时的处理步骤。

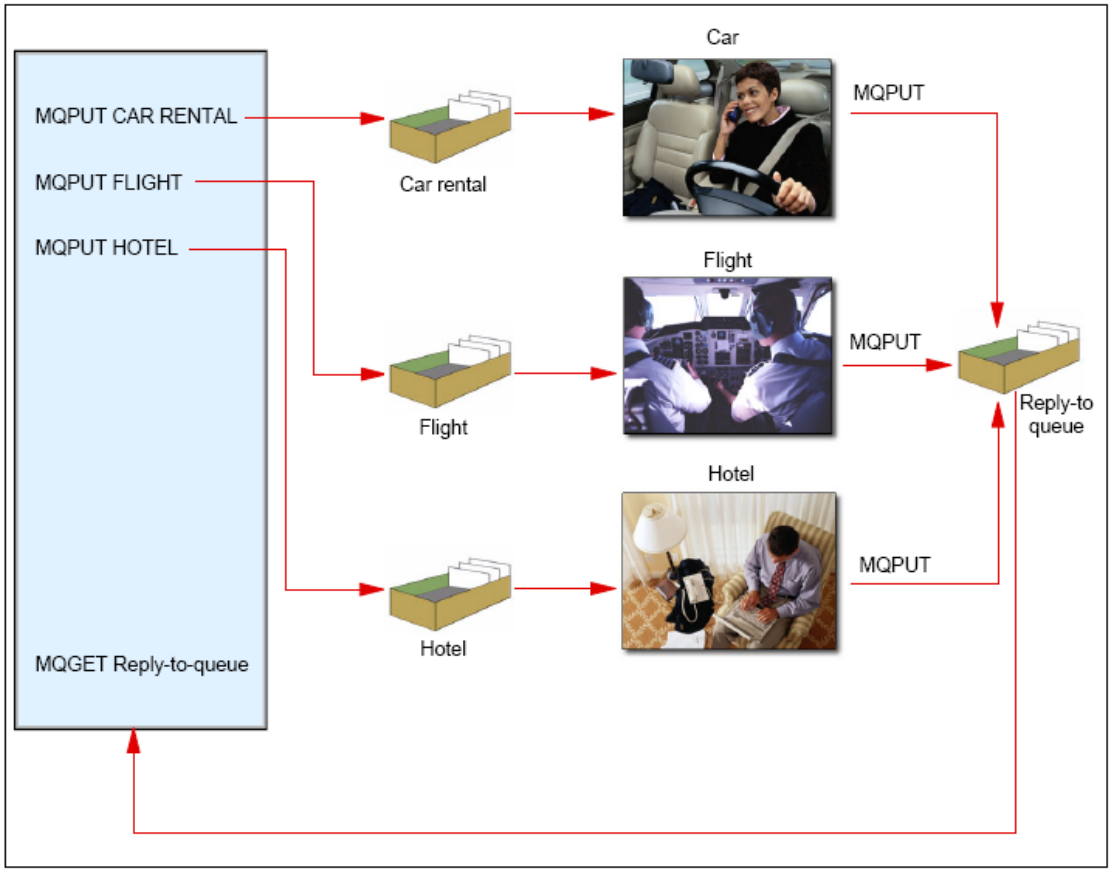


图15-4 并行处理

15.9 同 CICS，IMS，批处理或 TSO/E 的接口

WebSphere MQ可以用于多种平台之上。z/OS上的WebSphere MQ包含了一些适配器来为以下产品提供消息和队列服务：

- CICS：WebSphre MQ-CICS桥接器
- IMS：WebSphere MQ-IMS桥接器
- Batch或者TSO/E

15.10 总结

在一个在线应用系统环境中，消息和队列机制可以使得不同平台上的应用程序进行通讯。z/OS上的IBM WebSphere MQ是一个可以运行在主机和其他环境中的管理消息和队列的软件。通过消息机制，程序可以通过传递消息来彼此进行通讯，而不需要通过直接的程序调用。通过队列机制，消息可以被保存在存储区的队列之中，这样程序就可以彼此间独立地运行(也就是异步地运行)。

以下是WebSphere MQ的一些功能优势：

1. 提供了一个通用的应用编程接口MQI，MQI在多种支持平台上是一致的。
2. 数据的传输可以确保投递。即使系统发生故障，消息也不会丢失，同时也不会进行消息的重复发送。
3. 异步通讯机制。也就是说，进行通讯的应用程序不需要同时处于活动的状态。
4. 以消息驱动处理作为应用程序的设计模式。应用程序可以被分成一些离散的功能模块，这些模块可以运行在不同的系统中，可以在不同的时段被调用，或者并行的执行。
5. 因为屏蔽掉了网络的复杂性，使得应用程序的开发变得十分的快捷。

本章关键术语		
本地队列(local queue)	通道(channel)	消息驱动(message-driven)
消息队列接口(MQI)	异步应用程序 (asynchronous application)	死信队列/停用字母队列 (dead-letter queue)
队列管理器(QM)	远程队列(remote queue)	同步点(syncpoint)

458

15.11 复习题

为了帮助您检测对本章的理解程度，请完成下列问题：

1. 为什么需要在异构平台应用程序的通讯中使用消息和队列机制？
2. 描述消息的异步执行流程。
3. 解释队列管理器的功能。
4. 列出三种z/OS相关的适配器。
5. 使用MQI的目的是什么？
6. 死信队列/停用字母队列有什么用处？

459

Part 4

第 4 部分 z/OS 上的系统编程

在该部分，通过对系统库、安全以及用于启动(IPL)和停止z/OS系统的过程的讨论，我们揭示了z/OS的内部工作机制。该部分的一些章节还涉及硬件细节、虚拟技术以及在一个系统综合体中的多个z/OS系统的集群技术。

461

462

16

第 16 章 系统管理概览

目标：作为一名z/OS系统程序员，您需要了解z/OS的工作原理。

在完成本章后，您将能够：

- 讨论z/OS系统程序员的职责。
- 解释系统库的含义，用法和管理它们内容的方法。
- 列出不同类型的操作控制台。
- 描述IPL系统的过程。

16.1 系统程序员的角色

系统程序员负责管理主机硬件配置；安装、客户化和维护主机操作系统。各种装置都必须确保系统和服务可用且达到服务水平协议。要求24小时7天不间断操作的系统，需要对哪怕最小的业务活动中断作准备。

本章中，我们调查一些想要成为z/OS系统程序员的人感兴趣的方面。虽然本书无法涉及到系统管理的方方面面，但我们必须了解：z/OS系统程序员的工作非常复杂，需要系统很多方面的技能，比如：

- 设备I/O配置
- 处理器配置
- 控制台定义
- 软件所在的系统库
- 系统数据集和它们所在位置
- 用来定义z/OS配置的客户化参数
- 安全管理

如465页的图16-1所示，一定程度上，系统程序员的角色通常包含系统操作的以下方面：

- 466页的‘客户化系统’
- 478页的‘管理系统性能’
- 478页的‘配置I/O设备’
- 479页的‘遵循变化控制流程’
- 482页的‘配置控制台’
- 485页的‘初始化系统’

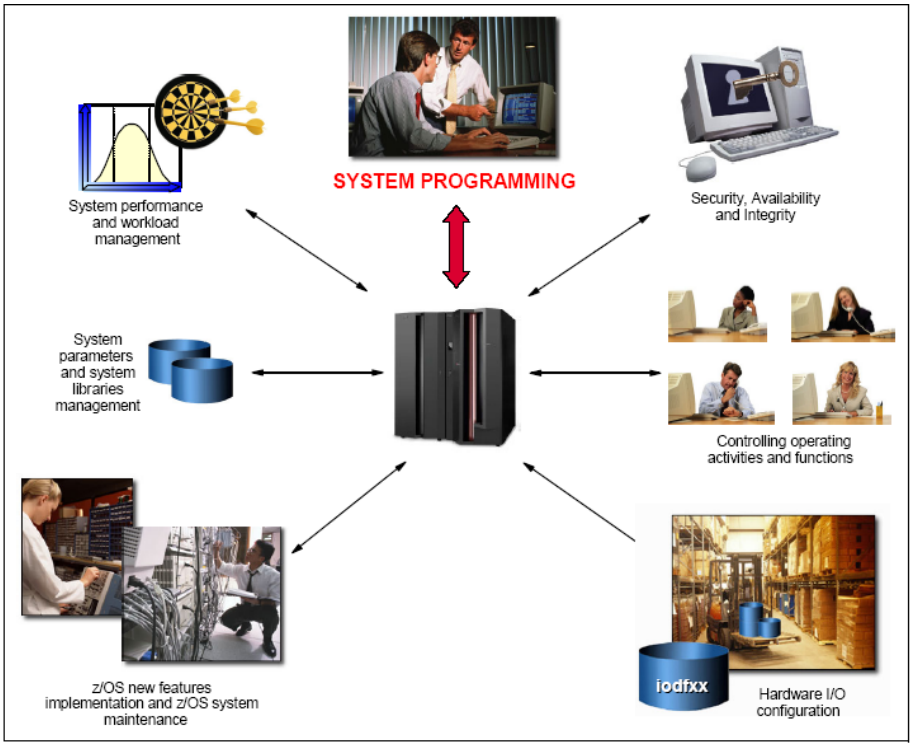


图16-1 系统程序员涉及的一些领域

16.2 什么是职责分离？

在一个大型z/OS系统中，在系统程序员中，或在IT机构系统程序员部门和其他部门之间，都存在着"职责分离"一说。

一个典型的z/OS系统包括以下角色或者更多：

- z/OS系统程序员
- CICS系统程序员
- 数据库系统程序员
- 数据库管理员
- 网络系统程序员
- 自动化专家
- 安全经理
- 硬件管理
- 生产控制分析师
- 系统操作员
- 网络操作员
- 安全管理员
- 服务经理

职责分离的部分原因是由于审计要求——确保一个人在一台系统上没有太大的权力。

例如，当一个系统上要添加新应用程序时，要在完成若干任务之后，该程序才能被终端用户使用。生产控制分析师需要添加批处理程序到批处理调度包中，将新的过程添加至过程库中并建立操作流程。系统程序员需要完成系统本身相关的任务，诸如给予安全特权，将程序加至系统库中。程序员的工作还涉及到为新应用程序建立自动化操作。

但是在测试系统上，一个人可能履行包括操作员在内的所有角色和任务。这常常是一个学习各项工作如何开展的好方法。

16.3 客户化系统

这部分描述了以下主题：

- 软件所在的系统库
- 系统数据集和它们所在位置
- I/O设备配置
- 控制台定义
- 用来定义z/OS配置的客户化参数
- z/OS实施和维护

16.3.1 z/OS 系统库

如467页的图16-2所示，一台系统中存在着不同类型的数据。

首先有IBM提供的z/OS软件。这经常安装到一系列叫做常驻系统卷(SYSRES)的磁盘卷上。

z/OS很多的灵活性是依赖于这些SYSRES卷的。您可以在现行的一套运行生产任务的同时，从生产集中备份一套新的并进行维护。之后从新的一套IPL只需要很短的断供时间----但维护工作已经实现！同样，如果从旧的一套IPL，将停止应用变化。

z/OS的补丁由一个系统更改程序/扩展(SMP/E)的产品管理。使用系统符号可以间接编目，一个特定的库可以以此种方式编目在例如SYSRES卷2上，该卷的名字在系统IPL时可以从系统符号中解析得到。我们将在476页的16.3.11节 “什么是系统符号”中讨论系统符号。

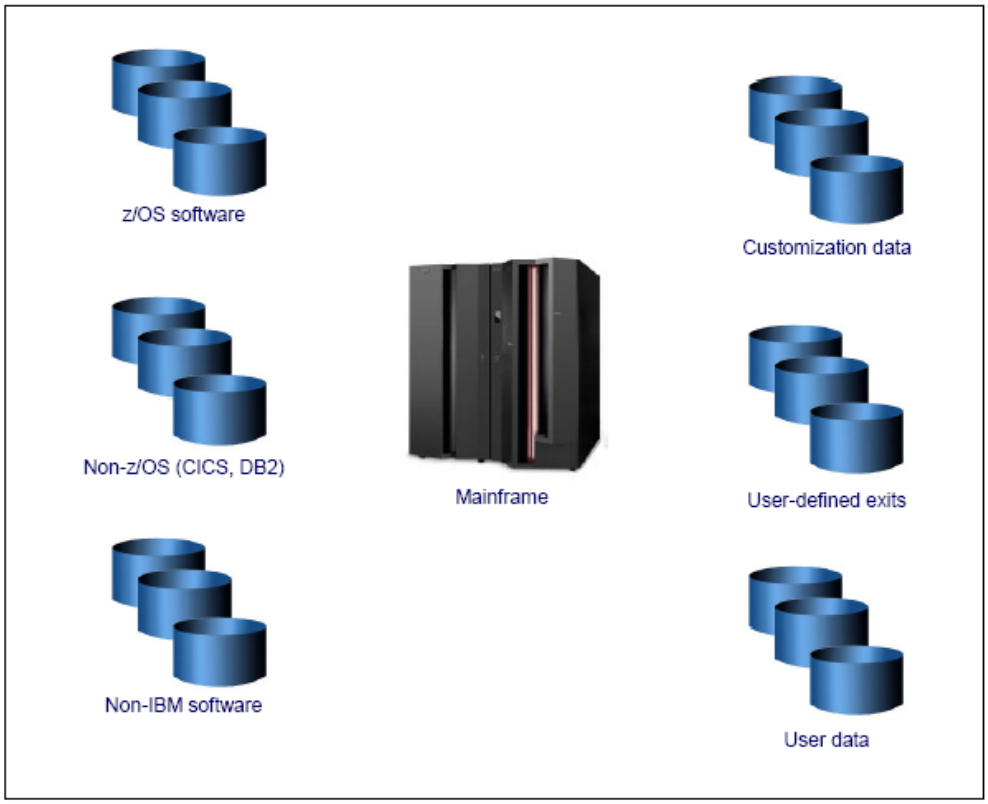


图16-2 数据类型

另外一组卷是非z/OS和非IBM软件卷。这些可以合并成一个卷组。非z/OS软件大多数不在SYSRES卷上，因为SMP/E往往将SYSRES卷集当成一个实体来管理，而将其他软件分开单独管理。因为这些卷并不是组成SYSRES卷的一部分，所以卷上的每个库都只有一个拷贝。我们可以把很多需要的卷添加到组中，给每个卷指定一个自己的磁盘名。

‘客户化数据’指的是诸如SYS1.PARMLIB，SYS1.PROCLIB，主目录，IODF，页数据集，JES SPOOL，/etc目录等之类的项目，这些项目对系统运行至关重要。这也是为管理软件，SMP/E数据所存储的位置。

这些数据集并不总是和IBM提供的z/OS软件存放在不同的DASD卷上；一些系统将PARMLIB和PROCLIB放在第一个SYSRES卷包上，其他的将它们放在主目录所在的卷包上或其他地方。这样一个选择取决于SYSRES卷的管理方式。每个系统都有自己偏好的方法。

在很多系统上，一些IBM提供的默认值并不合适，所以我们需要修改它们。用户出口程序(user exit)和用户更正程序(usermod)直接修改IBM代码，来满足系统需求。更正程序通常由SMP/E管理。

最后是用户数据，这也通常是最大的磁盘卷池。这并不是系统库的一部分，展示在这里主要是为了体现完整性。用户数据包含生产，测试和用户数据。通常它存放在不同的卷池中，由系统管理存储(SMS)管理，SMS可以直接将数据放置到合适的卷上。比如，生产数据可以放置到每天备份的卷上，而用户数据可以放置到

其他一些卷上，这些卷每周备份，并且在一段不活动时间后数据可能会迁移到磁卷上，来释放磁盘空间以存放新的数据。

z/OS有很多标准系统库，比如：SYS1.PARMLIB，SYS1.LINKLIB，SYS1.LPALIB，SYS1.PROCLIB，和SYS1.NUCLEUS。一些系统库和IPL过程相关，另外一些和调用程序的搜索顺序或系统安全相关，描述如下：

- SYS1.PARMLIB包含整个系统的控制参数。
- SYS1.LINKLIB包含很多系统的执行模块。
- SYS1.LPALIB包含系统启动时装载到连接装配区(LPA)的系统执行模块。
- SYS1.PROCLIB包含z/OS发布的JCL过程。
- SYS1.NUCLEUS有系统基本的监控管理模块。
- SYS1.SVCLIB有访管程序。

468

16.3.2 SYS1.PARMLIB

SYS1.PARMLIB是一个分区数据集，它包含了IBM提供的和系统创建的成员。它必须放置在直接存取卷上，可以是系统常驻卷。PARMLIB在z/OS操作系统中是一个重要的数据集，您可以认为它的功能类似于UNIX系统下的/etc目录。

PARMLIB目的在于在一个单独数据集中以事先指定的形式提供初始化参数，这样就减小了操作员输入参数的必要性。

所有SYS1.PARMLIB数据集中的参数和成员在‘z/OS MVS Initialization and Tuning Reference, SA22-7592’一书中有详细描述。本章讨论一些最重要的PARMLIB成员。

16.3.3 连接装配区(LPA)

LPA是一个地址空间公共区的一部分。它存在于系统队列区(SQA)之下，包含可调页的连接装配区(PLPA)，接着是固定连接装配区(FLPA)，如果其中之一存在的话，最后则是修正连接装配区(MLPA)。

LPA模块装载入公共存储区，系统的所有地址空间都可共享它。因为这些模块是可重入的，并且不能自修改，所以任何地址空间中的多个任务都可以同时使用它们。LPA中的模块已经在虚存中了，所以它们不需要进入虚存。

LPA中的任何模块总是在虚存中，FLPA中的模块总是在中央存储中。LPA模块需要经常引用，否则这些页就会被窃取。当LPA中的页(不同于FLPA)没有持续被多个地址空间引用时，它就可能被窃取。

16.3.4 可调页的连接装配区(PLPA)

PLPA是一个公共存储区域，它在IPL时(当冷启动完成且指定CLPA选项时)装载。这块区域包含了只读的系统程序以及系统选择的一些只读的可重入的用户程序，以方便用户共享。PLPA和扩展PLPA包含所有SYS1.LPALIB中的成员和其他在LPALSTxx中指定的有效库成员。LPALSTxx通过IEASYSxx中的LPA参数指定或在系统初始化时由操作员输入指定(后者将会覆盖PARMLIB规范)。

您可以使用一个或多个在SYS1.PARMLIB中的LPALSTxx成员来连接您的系统程序库数据集至SYS1.LPALIB。您也可以使用LPALSTxx成员在PLPA中添加您系统的只读可重入用户程序。系统利用这个连接，即LPALST连接，在内核初始化进程中建立PLPA。SYS1.LPALIB必须放置在直接访问卷上，通常是系统常驻卷。

图16-3 表明一个LPALSTxx成员的例子

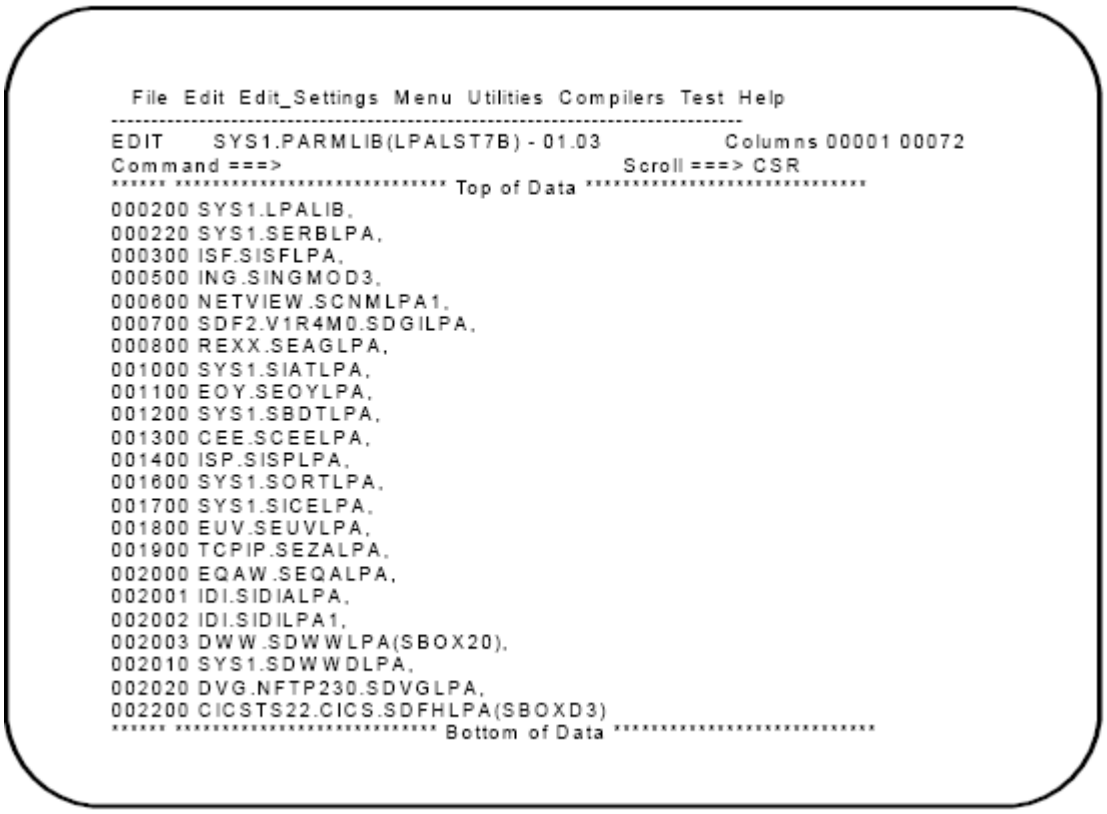


图16-3 LPALST PARMLIB成员例子

16.3.5 固定的连接装配区(FLPA)

FLPA在IPL时载入，其中的模块在SYS1.PARMLIB中有效的IEAFIXxx成员中列出。该区域只为当采用固定模式而非调页模式时可以显著提高系统性能的模块使

用。最合适放入FLPA的模块不经常被使用，但是需要极快的反应时间。

FLPA可以包含来自LPALST连接，linklist连接，SYS1.MIGLIB和SYS1.SVCLIB
470 的模块。FLPA由IEASYSxx中的FIX参数指定，也可以在系统IPL时由操作员输入指定，前者附追加在IEAFIX之后就能组成IEAFIXxx PARMLIB成员。

图16-4表明了一个IEAFIX PARMLIB成员，FLPA中一部分模块属于SYS1.LPALIB库。

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
-------------------------------------------------------------------------------
EDIT      SYS1.PARMLIB(IEAFIX00) - 01.00              Columns 00001
00072
Command ===>                                          Scroll ===> CSR
****** ***************************** Top of Data ******************************
000001 INCLUDE LIBRARY(SYS1.LPALIB) MODULES(
000002          IEAVAR00
000003          IEAVAR06
000004          IGC0001G
000005          )
000006 INCLUDE LIBRARY(FFST.V120ESA.SEPWMOD2) MODULES(
000007          EPWSTUB
000008          )
****** **************************** Bottom of Data ****************************
```

图16-4 IEAFIX PARMLIB成员

16.3.6 修正的连接装配区(MLPA)

MLPA包含APF授权库中可重入的程序(见537页的18.7.1"授权程序")，它是当前IPL过程中连接装配区的可调页扩展。需要注意的是，MLPA只有在IPL时才存在。因此，如果想使用一个MLPA，就要在每次IPL(包括快速启动和热启动IPL)时指定MLPA的模块。当系统搜索一个程序时，先搜索MLPA再搜索PLPA。故在IPL时可以利用MLPA中的新模块或者替换模块来临时修改或更新PLPA中的模块。

16.3.7 SYS1.PROCLIB

SYS1.PROCLIB是一个分区数据集，它包含完成某种系统功能的JCL过程。操作 471
员或程序员可以调用JCL过程来完成系统任务或处理程序任务。

16.3.8 主调度子系统

主调度子系统在操作系统和主要作业输入子系统 (JES2或JES3)之间建立通讯。当您启动z/OS，主要启动程序初始化系统服务，比如系统日志和通讯任务，并启动主调度地址空间，该地址空间将变成第一个地址空间(ASID=1)。然后，主调度器启动作业输入子系统 (JES2或JES3)。JES是主要作业输入子系统 。在很多生产系统上，人们并不马上启动JES，而是由自动化包按照控制顺序启动所有任务。然后启动其他定义好的子系统。所有子系统在PARMLIB库中的成员IEFSSNxx中定义，这些子系统是二级子系统(secondary subsystem)。

在SYS1.LINKLIB库中可以找到初始的MSTJCL00装入模块。如果需要更正，推荐做法是在PARMLIB数据集中创建一个MSTJCLxx成员。后缀由PARMLIB的IEASYSxx成员中MSTRJCL参数指定。MSTJCLxx成员一般称为主JCL。它包含所有系统输入输出数据集的数据定义(DD)语句，这些DD语句用于JES和操作系统之间的通讯。

例16-1显示了一个MSTJCLxx成员的例子。

例16-1主JCL样例

```
File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
-------------------------------------------------------------------------------
EDIT       SYS1.PARMLIB(MSTJCL00) - 01.07               Columns 00001 00072
Command ===>                                              Scroll ===> CSR
****** ***************************** Top of Data ******************************
000100 //MSTRJCL  JOB MSGLEVEL=(1,1),TIME=1440
000200 //         EXEC PGM=IEEMB860,DPRTY=(15,15)
000300 //STCINRDR DD SYSOUT=(A,INTRDR)
000400 //TSOINRDR DD SYSOUT=(A,INTRDR)
000500 //IEFPDSI  DD DSN=SYS1.PROCLIB,DISP=SHR
000600 //         DD DSN=CPAC.PROCLIB,DISP=SHR
000700 //         DD DSN=SYS1.IBM.PROCLIB,DISP=SHR
000800 //IEFJOBS  DD DSN=SYS1.STCJOBS,DISP=SHR
000900 //SYSUADS  DD DSN=SYS1.UADS,DISP=SHR
****** *************************** Bottom of Data ***************************
```


当主调度器不得不启动开始任务时，系统决定START命令所指的是一个过程还是一个作业。如果在MSTJCLxx成员中存在IEFJOBS DD语句，系统在IEFJOBS DD连接中搜索START命令请求的成员。

如果在IEFJOBS DD连接中找不到该成员或IEFJOBS连接根本不存在，系统就搜索IEFPDSI DD语句

，来寻找START命令请求的成员。如果找到该成员，系统检查第一条记录是否是有效的JOB语句，如果有效，系统使用该成员内容作为开始任务的JCL源码。如果成员第一条记录没有有效的JOB语句，系统则假设JCL源码是一个过程，系统将创建JCL来调用该过程。

在JCL源码被创建或被找到后，系统处理该JCL。在本书截稿之际，MSTJCL00

包含一个IEFPDSI DD语句，该语句定义的数据集包含了开始任务的过程JCL源码。一般来说，这个数据集是SYS1.PROCLIB；也可能是一个连接。为了完成有用的工作，SYS1.PROCLIB必须至少包含主JES的过程，下章节将继续讨论这点。

16.3.9 作业过程库

SYS1.PROCLIB包含JES2编目过程。该过程定义了作业相关过程库，如例16-2。

例16-2 怎样在JES2过程中指定过程库

```
//PROC00 DD  DSN=SYS1.PROCLIB,DISP=SHR
//       DD  DSN=SYS3.PROD.PROCLIB,DISP=SHR
//PROC01 DD  DSN=SYS1.PROC2,DISP=SHR
...
//PROC99 DD  DSN=SYS1.LASTPROC,DISP=SHR
...
```

很多系统中，JES过程中有很长的过程库列表。这是因为JCLLIB是一个相对较新的改革。

我们应该关注可以删除这些过程库的用户数，因为一旦某个过程库被删除，JES将无法启动。一般情况下，正在使用中的库是无法被删除的，但是JES并没有占据这些库，虽然它一直在使用这些库。

您可以通过指定以下语句覆盖默认设置：

```
/*JOBPARM PROCLIB=
```


在过程库的名字(PROCLIB)之后，您可以将JES2过程中指向所用库的DD语句的名字写在此处。举例来说，图16-2中，我们假设您在类A中运行一个作业，类A已指定默认PROCLIB为PROC00。如果您想使用放置在SYS1.LASTPROC中的过程，您将需要在JCL中包含以下语句：

```
/*JOBPARM PROCLIB=PROC99
```

另一个指定过程库的方法是使用JCLLIB JCL语句。该语句允许您在不使用系统过程库的情况下编码和使用过程。系统按照您在JCLLIB语句中指定的顺序依次搜索库，然后再搜索未指定的默认系统过程库。

例16-3表明了JCLLIB语句的使用。

例16-3 JCLLIB语句样例

```
//MYJOB   JOB
//MYLIBS  JCLLIB    ORDER=(MY.PROCLIB.JCL,SECOND.PROCLIB.JCL)
//S1      EXEC PROC=MYPROC1
...
```

假设系统默认过程库只包含SYS1.PROCLIB，系统将按照以下顺序搜索过程MYPROC1：

1. MY.PROCLIB.JCL

2. SECOND.PROCLIB.JCL

3. SYS1.PROCLIB

16.3.10 程序搜索顺序

当系统服务(诸如LINK，LOAD，XCTL或ATTACH)使用默认选项请求一个程序时，系统按照以下顺序搜索程序：

1. 作业装配区(JPA)

   JPA中的程序已经装载到请求的地址空间中。如果JPA中的拷贝可用，那就是用它。否则，系统要么搜索新的拷贝，要么延迟请求直到JPA中的拷贝可用为止。(比如，在拟使用JPA中串行化重用模块之前，系统将延迟请求直到在它之前的请求完成工作)

2. TASKLIB

   一个程序可以给TASKLIB连接分配一个或多个数据集。TASKLIB中由未授权任务装载的模块在执行之前必须被装载入虚存的私有区。之前已经被装载入虚存的公共区中的模块(LPA模块或那些由授权程序装载入CSA的模块)在运行之前必须再次被装载入虚存公共区。

3. STEPLIB 或JOBLIB

   这些特定的DD名字可以用来分配数据集，这些数据集在搜索程序时居于默认系统搜索顺序之前。我们可以在JCL中或利用具有动态分配的程序在STEPLIB和JOBLIB中分配数据集。但只搜索两者之一。 如果某个作业步即分配了STEPLIB又分配了JOBLIB，系统会搜索STEPLIB而忽略JOBLIB。

   系统在TASKLIB之后LPA之前搜索任何连接到STEPLIB和JOBLIB的数据集。在STEPLIB和JOBLIB中找到的模块在运行前必须被装载入虚存私有区。之前已经被装载入虚存公共区中的模块(LPA模块或那些由授权程序装载入CSA的模块)在运行之前必须再次被装载入虚存公共区。

4. LPA的搜索顺序如下：

   a. 动态LPA模块，在PROGxx成员中指定。

   b. 固定LPA(FLPA)模块，在IEAFIXxx成员中指定。

   c. 修正LPA(MLPA)模块，在IEALPAxx成员中指定。

   d. 可调页LPA(PLPA)模块，由LPALSTxx或PROGxx中指定的库中载入。

   LPA模块装载入公共存储中，系统中所有地址空间都可以共享。因为模块是可

重入且非自修改的，每个模块都可以同时被任意地址空间内的任意多个任务使用。LPA中的模块不需要被装载入虚存中，因为它们已经在虚存中了。

5. 由PROGxx和LNKLSTxx指定的linklist的库

   默认地，linklist开始于SYS1.LINKLIB，SYS1.MIGLIB和SYS1.CSSLIB。但是您可以使用PROGxx中的SYSLIB来改变这个顺序，并可以将其他库添加到linklist连接中。系统必须在运行这些模块前将它们载入到虚存私有区。

通过改变调用程序的宏的一些选项可以改变程序的默认搜索顺序。影响系统搜索顺序的参数为EP，EPLOC，DE，DCB和TASKLIB。想知道这些参数的更多信息，查看‘z/OS MVS Programming: Assembler Services Guide.’书中关于搜索装入模块的章节。一些IBM子系统(比较显著的是CICS和IMS)和应用程序(诸如ISPF)使用这些工具来建立程序的其他搜索顺序。

16.3.11 什么是系统符号

系统符号是允许不同z/OS系统共享PARMLIB定义，同时又能保持这些定义唯一值的元素。系统符号就好像程序中的变量；它们可以根据程序输入而呈现不同值。当您在共享PARMLIB定义中指定一个系统符号时，系统符号相当于‘占位符’。每个共享定义的系统在初始化时用唯一值替代系统符号。

每个系统符号都有一个名字(以(&)符号开始，以可选的句点(.)结束)和一段置换文本，后者是在每次系统符号出现时，系统用来替换它的字符串。

有两种类型的系统符号：

动态： 置换文本可以在IPL的任意时间点变换。

静态： 置换文本在系统初始化时定义，在整个IPL过程中保持不变。

一些符号留作系统之用。您可以在系统中输入D SYMBOLS命令显示这些符号。例16-4显示了输入命令后的结果。

例16-4 D SYMBLOS命令的部分输出(移除了部分行)

```
HQX7708 ----------------  SDSF PRIMARY OPTION MENU  --
COMMAND INPUT ===> -D SYMBOLS
  IEA007I STATIC SYSTEM SYMBOL VALUES
          &SYSALVL.  = "2"
          &SYSCLONE. = "70"
          &SYSNAME.  = "SC70"
          &SYSPLEX.  = "SANDBOX"
          &SYSR1.    = "Z17RC1"
          &ALLCLST1. = "CANCEL"
          &CMDLIST1. = "70,00"
          &COMMDSN1. = "COMMON"
          &DB2.      = "V8"
          &DCEPROC1. = "."
          &DFHSMCMD. = "00"
          &DFHSMHST. = "6"
          &DFHSMPRI. = "NO"
          &DFSPROC1. = "."
          &DLIB1.    = "Z17DL1"
          &DLIB2.    = "Z17DL2"
          &DLIB3.    = "Z17DL3"
          &DLIB4.    = "Z17DL4"
          &IEFSSNXX. = "R7"

          &IFAPRDXX. = "4A"
```

IEASYMxx PARMLIB成员提供在单独一处指定多系统环境中每个系统参数的方法。IEASYMxx包含定义静态系统符号的语句；还包含指定IEASYSxx PARMLIB成员的语句(SYSPARM语句)，该成员中包含了系统参数。

476

例16-5 显示了一个IEASYMxx PARMLIB成员。

例16-5 部分IEASYMxx PARMLIB成员(移除了部分行)。

```
SYSDEF          SYSCLONE(&SYSNAME(3:2))
                  SYMDEF(&SYSR2='&SYSR1(1:5).2')
                  SYMDEF(&SYSR3='&SYSR1(1:5).3')
                  SYMDEF(&DLIB1='&SYSR1(1:3).DL1')
                  SYMDEF(&DLIB2='&SYSR1(1:3).DL2')
                  SYMDEF(&DLIB3='&SYSR1(1:3).DL3')
                  SYMDEF(&DLIB4='&SYSR1(1:3).DL4')
                  SYMDEF(&ALLCLST1='CANCEL')
                  SYMDEF(&CMDLIST1='&SYSCLONE.,00')
                  SYMDEF(&COMMDSN1='COMMON')
                  SYMDEF(&DFHSMCMD='00')
                  SYMDEF(&IFAPRDXX='00')
                  SYMDEF(&DCEPROC1='.')
                  SYMDEF(&DFSPROC1='.')
  SYSDEF          HWNAME(SCZP901)
                  LPARNAME(A13)
                  SYSNAME(SC70)
                  SYSPARM(R3,70)
                  SYMDEF(&IFAPRDXX='4A')
                  SYMDEF(&DFHSMHST='6')
                  SYMDEF(&DFHSMPRI='NO')
                  SYMDEF(&DB2='V8')
```

本例中，变量&SYSNAME的值将会由关键字SYSNAME指定，本例中是SC70。

因为系统综合体中的每个系统都有唯一的名字，我们可以在允许之处使用&SYSNAME来指定每个系统的唯一资源。举例来说，我们可以指定一个SMF数据集为SYS1.&SYSNAME..MAN1，在系统SC70上运行时名字替换后的结果是SYS1.SC70.MAN1。

您可以使用变量来构造其他变量的值。如例16-5，我们可以看到&SYSCLONE采取了&SYSNAME的一部分值：从位置3开始的长度为2的值。这里，&SYSCLONE的值为70。类似地，我们看到&SYSR2由&SYSR1的前5位字符构造，后缀为2。那&SYSR1在哪里定义的呢？&SYSR1由系统定义的，它的值是IPL卷的卷标VOLSER。如果您回顾一下476页的例16-4，您可以看到&SYSR1和&SYSR2的值。

477

这里我们也可以看到定义在所有系统之上的全局变量——&IFAPRDXX的值是00——它在SC70上重定义为4A。

在多个z/OS系统共享单个PARMLIB时，系统符号有了用武之地。这里，使用符号，通过符号替换，使得可以使用单个成员，而不需要每个系统上都使用唯一不同的成员。LOADxx成员指定系统使用哪个IEASYMxx成员。

16.4 管理系统性能

系统‘调优’任务是一个迭代和持续的过程，它也是在企业中最大直接影响系统资源的所有用户的科目。z/OS工作负载管理(WLM)组件，曾在104页的‘什么是工作负载管理’讨论过，是这个过程重要的组成部分，它包括为多个系统组件和子系统选择合适参数进行初始化调优。

当系统变为可操作，依照作业类和优先级选择作业执行的标准建立之后，WLM按照系统指定的参数来控制可用资源的分配。

然而，WLM只能处理可用资源。如果可用资源无法满足系统需求，那即便再优化的分配也无能为力；应该检查其他系统区域来决定增加可用资源的可能性。当系统需求增加，有必要切换优先级或者获得额外资源(诸如更强处理器，更大存储，更多终端)，系统程序员也需要更改WLM参数来反映改变后的状况。

16.5 配置 I/O 设备

478

我们必须定义操作系统(软件)和通道子系统(硬件)的I/O配置。z/OS硬件配置定义(HCD)组件使得用户在单个交互终端用户接口下就可以处理硬件和软件I/O配置。HCD输出文件时一个I/O定义文件(IODF)，它包含I/O配置数据。IODF用来定义z/OS操作系统中多个硬件和软件配置。

当激活一个新的IODF，HCD定义通道子系统和/或操作系统的I/O配置。使用HCD激活功能或z/OS ACTIVATE操作命令，无须IPL软件或硬件上电重置(POR)就可以在现有配置中实施更改。在系统运行时实施更改叫做动态配置或动态重置。

16.6 遵守变化控制流程

典型情况下，数据中心管理有一组服务经理组成的专家团队，对服务等级协议(SLA)负责。数据中心的变化控制机制和实践在确保满足SLA的情况下实施。

任何变化必须在操作人员的控制之下实施。当一个变化被引入到生产系统并产生问题或不稳定时，操作人员负责观察，汇报，然后管理必要的活动来更正问题或取消变化。

虽然系统程序员通常自己发起并实施变化，但有时变化是来自变化管理系统的请求。任何为操作组或者其他组发起的指令都必须记入变化记录中，并且涉及的每个组都要表示同意。

实施商业应用程序更改一般由一位生产控制分析师处理。应用程序的更改一般发生于测试库中，通过正式的请求(带有审计跟踪)，测试库中的程序可以推广到生产环境中。

变更的流程必须通知到所有相关人员和部门。当所有方都认为变化描述完备时，就可以考虑实施变化，计划变化，延迟甚至可能丢弃变化。

在规划一个变化时需要考虑以下因素：

- 变化实施后带来的好处
- 变化未完成会有什么后果
- 实施变化需要的资源
- 较其他变化来说，变化请求的重要性
- 变化请求间的任何内部依赖性

所有的变化都带来风险。主机的一个优势是它能够提供高可用性。因此所有变化必须精心控制和管理。系统程序员大部分时间都花费在规划变化和风险评估变化上。考虑变化最重要的方面之一是怎样撤销变化，回到系统之前的状态。

16.6.1 风险评估

数据中心管理经常每周召开变化控制会议来讨论，批准或反对变化。这些变化可能是针对应用程序的，系统的，网络的，硬件的或电源等方面。

对于任何变化，一个重要部分就是风险评估，它是指从系统风险角度来考虑和评估变化。低风险变化可能在白天执行，而较高风险变化需要安排在系统不工作的时段。

数据中心通常也会有时期性的高低风险，这会影响决定。比如，如果系统运行信用授权程序，那在主要的公共假日系统通常会很忙碌，可能会冻结任何变化。同样，在零售业每年促销也是非常忙碌的时期，也会造成拒绝变化的决定。

IT机构通过强制执行严格制度化的变化管理流程和政策来达到目的。这些目的包

括：

- 高服务可用性
- 增强的安全性
- 审计工作准备就绪
- 节省成本

16.6.2 变化控制记录系统

一个变化控制记录系统通常用于允许请求，追踪和批准变化。它往往是问题管理系统的伙伴。比如，如果一个生产系统在周一早上有一个严重的问题，那么第一反应就是检查周末实施的变化，决定是否是这些变化带来的问题。

这些记录也显示了系统是可控的，常有必要向审计员证明这点，尤其是对那些非常正规的财务服务部门。美国2002年萨班斯-奥克斯利法强调了企业管理，建立了
480 有效内部控制系统的必要性。证明IT服务具有强大的变化管理和问题管理是部分遵守该措施的。除此之外，欧盟公司法第8法令，在本书撰写之际正在讨论中，也将侧重于和萨班斯-奥克斯利法类似的领域。

正因为这些原因，在最精简的情况下，实施任何变化之前应该定义一套控制文档，也叫变化请求表格，包含以下内容：

- 何人——即是，请求变化的部门，组或人，谁负责实施变化，谁来完成成功测试，如果需要谁负责撤销，谁来宣称变化成功完成。
- 何事——即是，受影响的系统或服务(比如电子邮件，文件服务，域等等)。尽可能越详细越好。理想情况下，应该包括完整的指示，这样在紧急情况下可以由他人完成变化。
- 何时——即是，变化的起始时间和估计持续时间。通常有3个日期：请求日期，计划实施日期和实际实施日期。
- 何处——即是，变化涉及的范围，哪些商务单元，楼房，部门或组会受影响或者需要协助变化。
- 如何——即是，实施规划，如果有需求，也包括撤销变化的规划。
- 优先级——即是，高，中，低，照常营业，紧急，注明日期的(比如时钟调整)。
- 安全——即是，高，中，低
- 影响——即是，如果实施变化将会产生什么样的影响，如果不实施又会如何。其他系统可能会受何种影响，如果不可预料事件发生，又会如何。

16.3.3 生产控制

生产控制通常涉及一个专业人员使用Tivoli®工作负载调度器来管理批处理时序安排，建立和管理一个复杂的批处理计划。该工作可能涉及对应用程序套件以复杂的顺序在每日或每周特定的时间点进行备份。作为计划的一部分，数据库和在线服务也可能会停止后再启动。当变化进行时，生产控制经常需要协同公共假日和其他特别事件比如(以零售业为例)冬季促销。

生产控制也负责将程序员编写的最新程序发布到生产系统中。该任务通常包括：移动源码至一个安全的生产库中，重编译源码产生一个生产装入模块，并将这个模块放置到一个生产装载库中。JCL被拷贝和被升级到生产系统标准，同时被放置到合适的过程库中，应用程序套件被添加到作业调度器中。

481

如果新库需要加入到linklist或需要授权时，那和系统程序员也会有交互。

16.7 配置控制台

z/OS操作涉及管理硬件，诸如处理器和外围设备(包含操作员工作的控制台)，以及软件，诸如z/OS操作控制系统，作业输入子系统 ，控制自动化操作的子系统(比如NetView®)和所有运行在z/OS上的应用程序。

z/OS系统的操作涉及以下方面：

- 消息和命令处理构成了操作员与z/OS交互及z/OS自动化的基础。
- 控制台操作，或如何与z/OS交互操作来监控或控制硬件和软件。

规划z/OS系统操作必须考虑操作员如何使用控制台工作和如何管理消息和命令。系统程序员需要确保操作员在控制台接收到必要的信息来完成他们的工作，选择合适的信息来禁止，自动化或完成其他类型的信息处理。

对于z/OS操作来说，系统如何建立控制台恢复；或为了改变操作选项操作员是否必须要重新IPL系统，都是重要的规划考虑要点。

因为消息是自动化操作的基本，系统程序员需要理解消息处理来规划z/OS自动化操作。

越来越多的主机安装使用多系统环境，这些系统操作活动间的协调就变得异常重要。即使单个z/OS系统上，主机安装也需要考虑控制功能区之间的通信。

在单个系统或多系统环境，操作员在控制台可以输入哪些命令是一个安全关注点，需要仔细协调。作为一名规划人员，系统程序员需要确保当人们与z/OS交互时，正确的人正在做着正确的事。

482

一个控制台配置包含操作员和z/OS通信使用的多种控制台。您的系统最先可以在硬件配置定义(HCD)中定义可以当作控制台使用的I/O设备，HCD是一个在主机上系统程序员为通道子系统和操作系统定义硬件配置的交互接口。

硬件配置管理器(HCM)是一个面向HCD的图形用户接口。HCM与HCD以客户端/服务器关系交互(即是，HCM运行在工作站上，HCD运行在主机上)。主机系统需要一个它们连接设备的内部模型，但是系统程序员在图形化界面下可以更方便有效地维护(和补充)该模型。HCM维护配置数据与主机上IODF同步，这些配置数据在工作站中表现为文件中的图。虽然也可以直接使用HCD配置硬件，但因为HCM是图形界面的，客户更喜欢单独使用HCM。

除了HCD，一旦z/OS定义设备后，就通过在CONSOLxx PARMLIB成员中指定合

适的设备号来告知系统使用哪些设备作为控制台。

通常来说，z/OS系统的操作员接收消息，然后在MCS和SMCS控制台输入命令。他们可以使用其他控制台(比如NetView控制台)来与z/OS交互，但是这里我们只介绍MCS,SMCS和EMCS控制台，主机公司经常使用这些控制台。

- 多控台支持(MCS)控制台是本地连接至z/OS系统的设备，它提供操作员和z/OS间的基本通信。MCS控制台连接至不支持系统网络架构或SNA协议的控制设备上。
- SNA多控台支持(SMCS)控制台提供操作员和z/OS之间的基本通信，不需要一定本地连接至z/OS系统。SMCS控制台使用z/OS通信服务器来提供操作员和z/OS之间的通信，而不是直接I/O至控制设备。
- 扩展多控台支持(EMCS)控制台是操作员或程序员输入命令或接收消息的设备(不是MCS或SMCS控制台)。将EMCS控制台定义作为控制台配置的一部分，允许系统程序员扩展控制台数量，摆脱MCS控制台数量限制，一个系统综合体中的每个z/OS系统最多只能有99个MCS控制台。

系统程序员按照其功能在配置中定义这些控制台。需要反应的重要消息会直接发送给操作员，他可以在控制台上输入命令。另外一个控制台可以当成一个显示消息的监视器，给诸如磁带池库功能区的操作员显示信息，或显示关于您的系统上打印机的信息。

图16-5 显示了一个z/OS系统的控制台配置，包括系统控制台，一个SMCS控制台，NetView和TSO/E。

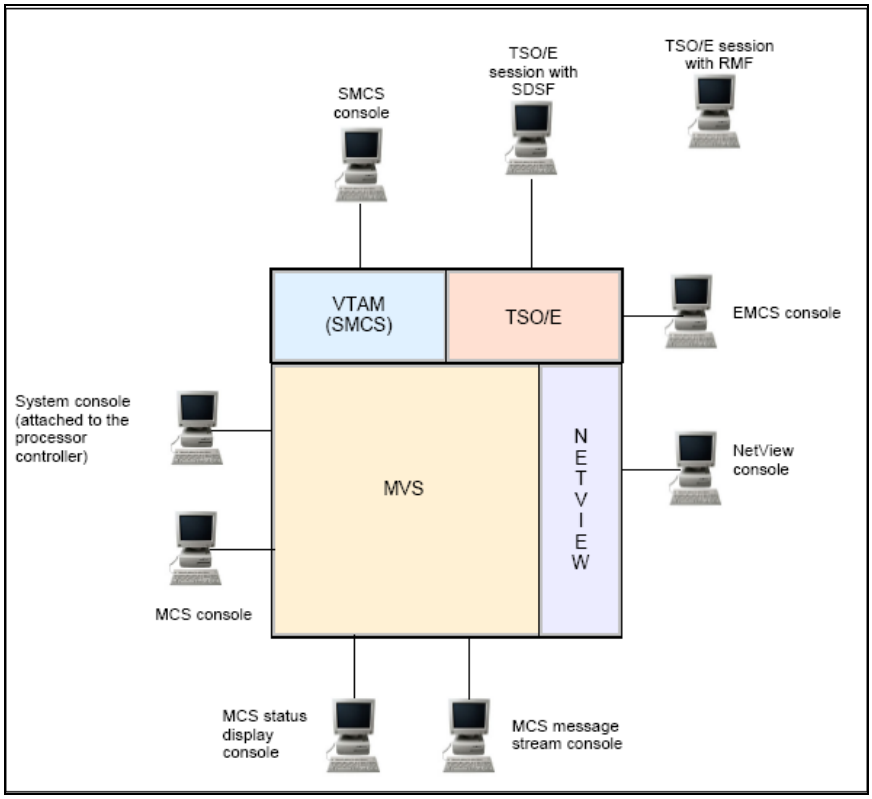


图16-5 z/OS系统中控制台配置样例

系统控制台功能是硬件管理控制台(HMC)的一部分。操作员可以使用系统控制台来启动z/OS和其他系统软件，在其他控制台不可用，需要恢复的情况下也可以使用系统控制台。

除了MCS和SMCS控制台， 图16-5所示的z/OS系统定义了一个NetView控制台。NetView和系统消息，命令列表一起工作，帮助自动化z/OS操作员任务。很多系统操作都可以通过一个NetView控制台控制。

484

用户可以从TSO/E终端监控很多z/OS系统功能。TSO/E用户使用系统显示和搜索工具(SDSF)以及资源评估工具(RMF™)监控z/OS，对工作负载平衡和性能问题作出响应。授权TSO/E用户也可以启动一个扩展MCS控制台会话来与z/OS交互。

图16-5中显示的MCS控制台有：

- 一个MCS控制台，操作员可以通过它查看消息，输入z/OS命令。这个控制台处于全功能模式因为它既可以接收消息又可以接收命令。操作员可以通过MCS或SMCS控制台控制z/OS系统操作。
- MCS状态显示控制台

  操作员可以查看来自DEVSERV，DISPLAY，TRACK或CONFIG命令的系统状态信息。然而，因为这是一个状态显示控制台，操作员无法输入命令。一个在全功能模式控制台上的操作员可以输入这些命令并将命令结果输出到一个状态显示控制台以供查看。

- MCS消息流控制台

  一个消息流控制台可以显示系统消息。操作员可以查看路由到该控制台的消息。然而，因为这是一个消息流控制台，操作员无法输入命令。系统为控制台定义路由代码和消息级别信息，这样就可以将相关消息定向给相应控制台屏幕显示出来。因此，举例来说，负责诸如磁带池库功能区的操作员，可以查看MOUNT消息。

在很多主机系统中，激增的屏幕被操作员工作站所替代，工作站将很多这些屏幕结合到一个窗口中显示。一般来说，硬件控制台是分开的，但是其他很多终端是合成在一起的。系统通过自动化产品的异常状况警告来管理。

合成了控制台控制器的IBM开放系统快速适配器产品(OSA-ICC)是一个连接控制台的现代化手段。在以太网上，OSA-ICC使用TCP/IP连接来接入个人电脑，通过一个TN3270连接(telnet)可以将个人电脑作为控制台使用。

16.8 初始化系统

初始化程序装载(IPL)是一种行为，从磁盘上装载操作系统的一个拷贝到处理器的实存中，并执行它。

485

z/OS系统被设计成在下次重新装载前可持续运行数月，以允许重要的生产工作负载持续可用。变化是重新装载的常见原因，系统的变化级别决定重新装载计划。比如：

- 测试系统可以每日或更经常地IPL。
- 高可用性的银行系统可能一年或更久的时间里重新装载一次，来更新软件级别。
- 外部影响常常是IPL的原因，比如需要测试或维护机房的供电系统。
- 有时设计糟糕的软件耗尽系统资源，只能通过IPL来解决问题。但是这类行为通常用于调查和更正问题。

很多过去需要IPL才能实现的变化，现在可以动态地实现。这样的例子有：

- 在linklist中为子系统，如CICS，添加一个库
- 在LPA中添加模块

我们使用硬件管理控制台(HMC)来IPL z/OS。您需要提供以下信息来IPL z/OS：

- IPL卷的设备地址
- 包含指向系统参数的LOADxx成员
- 包含配置信息的IODF数据集
- IODF卷的设备地址

16.8.1 初始化过程

系统初始化过程(487页的图16-6)准备系统控制程序和系统工作环境。过程主要包括以下：

- 系统和存储初始化，包含系统组件地址空间的创建
- 主调度器初始化和子系统初始化

当系统被初始化且作业输入子系统 被激活后，系统可以使用START，LOGON或MOUNT命令提交处理作业。

系统程序员在硬件管理控制台(HMC)选择LOAD功能开始启动初始化过程。z/OS定位所有在线可用的中央存储，并开始创建各种系统区域。

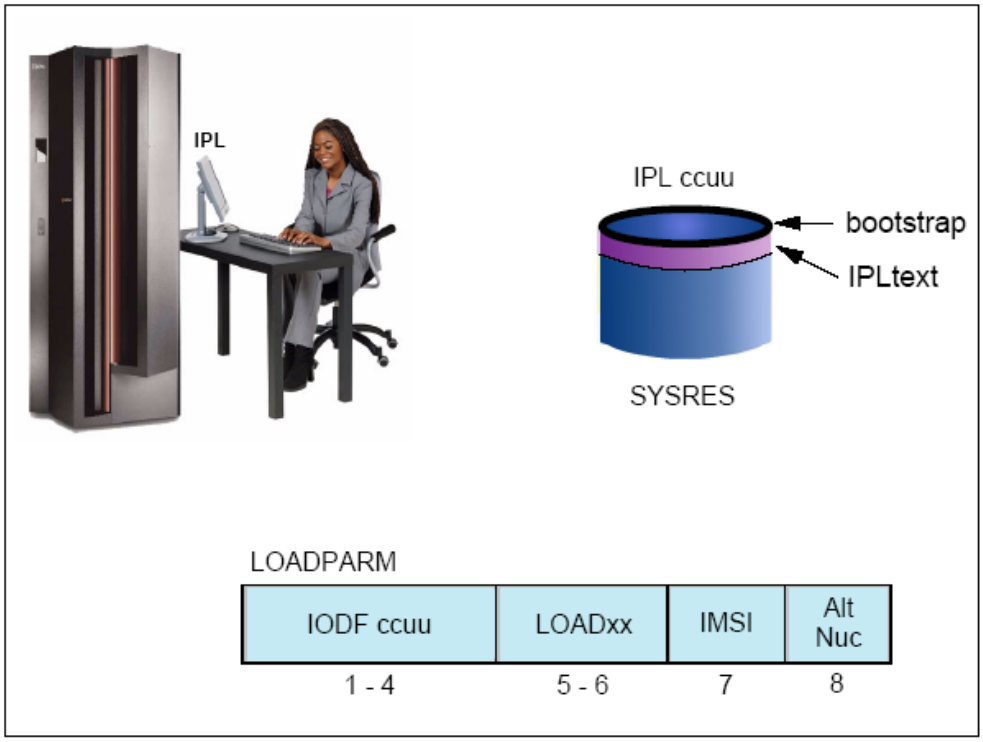


图16-6 IPL机器

并非所有连接到CPU的磁盘都含有装载代码。含有装载代码的磁盘通常指'可IPL的'磁盘，或者说就是SYSRES卷。

可IPL的磁盘在0柱面0磁道处包含一个bootstrap模块。在IPL时，bootstrap被装载到内存的实地址0处，控制权将交给它。然后bootstrap读取IPL控制程序IEAIPL00(也叫IPL文本)并将控制权传递给它。然后这就开始了更复杂的任务来装载操作系统并执行之。

在装载完bootstrap并把控制权传递给IEAIPL00之后，IEAIPL00将准备一个环境，在其中运行程序和模块，组成操作系统，如下：

1. 它将中央存储清空为0，然后为主调度器定义存储区。
2. 它定位在SYSRES卷上的SYS1.NUCLEUS数据集，从中装载一系列的程序，这些程序称为IPL资源初始化模块(IRIM)。
3. 这些IRIM开始创建正常操作系统环境的控制块和子系统。

IRIM完成的一些更重要的任务列举如下：

- 在IPL命令执行时，读取硬件控制台输入的LOADPARM信息。
- 搜索在LOADPARM成员中指定的IODF数据集所在的卷。IRIM将首先试图在SYS0.IPLPARM中定位LOADxx。如果不成功，它就去找SYS1.IPLPARM，如此下去直到SYS9.IPLPARM。如果此时它仍旧没有定位到，则继续搜索SYS1.PARMLIB。(如果LOADxx无法被定位，系统装载一个等待状态。)
- 如果找到LOADxx成员，就打开它并读取它其中的内容，包含内核后缀(除非

被LOADPARM覆盖)，主目录名字，使用的IEASYSxx成员后缀。

- 装载操作系统内核。
- 初始化主调度器地址空间中系统队列区(SQA)，扩展SQA(ESQA)，本地SQA(LSQA)和前缀保存区(PSA)的虚存。在IPL顺序的最后，PSA将会在实存位置0处替代IEAIPL00，然后驻留在该处。
- 初始化实存管理，包含主调度器的段表，公共存储区的段表条目，和页面页框表。

IRIM最后装载内核初始化程序(NIP)的第一部分，来调用资源初始化模块(RIM)，这是启动与IODF定义的NIP控制台通信程序的最早的一部分。

系统继续初始化进程，按照指定的系统参数解析和行动。NIP实现以下的主要的初始化功能：

- 按照SQA系统参数指定的量拓展SQA和ESQA
- 为冷启动IPL创建可调页的连接装配区(PLPA)和扩展PLPA；为快速启动或热启动IPL重置表来匹配现存的PLPA和扩展PLPA。要更多了解快速启动和热启动，请参阅”z/OS MVS Initialization and Tuning Reference”。
- 在固定连接装配区(FLPA)或扩展FLPA中载入模块。需要注意的是，NIP只有在FIX系统参数指定时才能实现这些功能。

488

- 在更改连接装配区(MLPA)和扩展MLPA中载入模块。需要注意的是，NIP只有在MLPA系统参数指定时才能实现这些功能。
- 为共同服务区(CSA)和扩展CSA分配虚存。存储分配量由IPL时CSA系统参数指定的值决定。
- 页保护NUCMAP，PLPA和扩展PLPA，MLPA和扩展MLPA，FLPA和扩展FLPA，以及内核的一部分。

注意：一个系统可以通过在MLPA和FIX系统参数中指定NOPROT来覆盖MLPA和FLPA的页保护。

PARMLIB的一个成员IEASYSnn，包含参数和指针来控制IPL的方向。见例16-6。

例16-6 部分IEASYS00成员内容

```
------------------------------------------------------------------------------
File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
------------------------------------------------------------------------------
EDIT       SYS1.PARMLIB(IEASYS00) - 01.68               Columns 00001 00072
Command ===>                                              Scroll ===> CSR
****** ***************************** Top of Data ******************************
000001 ALLOC=00,
000002 APG=07,
000003 CLOCK=00,
000004 CLPA,
000005 CMB=(UNITR,COMM,GRAPH,CHRDR),
000006 CMD=(&CMDLIST1.),
000007 CON=00,
000008 COUPLE=00,  WAS FK
000009 CSA=(2M,128M),
000010 DEVSUP=00,
000011 DIAG=00,
000012 DUMP=DASD,
000013 FIX=00,
000014 GRS=STAR,
000015 GRSCNF=ML,
000016 GRSRNL=02,
000017 IOS=00,
000018 LNKAUTH=LNKLST,
000019 LOGCLS=L,
000020 LOGLMT=999999,
000021 LOGREC=SYS1.&SYSNAME..LOGREC,
000022 LPA=(00,L),
000023 MAXUSER=1000,
000024 MSTRJCL=00,
000025 NSYSLX=250,

000026 OMVS=&OMVSPARM.,
------------------------------------------------------------------------------
```

要查看您的系统是如何IPL的，您可以输入D IPLINFO命令，例如16-7所示。

例16-7 D IPLINFO命令的输出

```
D IPLINFO
IEE254I  11.11.35 IPLINFO DISPLAY 906
 SYSTEM IPLED AT 10.53.04 ON 08/15/2005
 RELEASE z/OS 01.07.00    LICENSE = z/OS
 USED LOADS8 IN SYSO.IPLPARM ON C730
 ARCHLVL = 2   MTLSHARE = N
 IEASYM LIST = XX
 IEASYS LIST = (R3,65) (OP)
 IODF DEVICE C730
 IPL DEVICE 8603 VOLUME Z17RC1
```

系统地址空间的创建

除了初始化系统区域，z/OS建立系统组件地址空间。它为主调度器建立一个地址空间，同时为其他子系统还有系统组件建立系统地址空间。一些组件地址空间有：*MASTER*，ALLOCAS，APPC，和CATALOG等等。

主调度器的初始化

主调度器的初始化程序初始化诸如系统日志和通信任务的系统服务，且启动主调度器本身。它们也创建作业输入子系统 (JES2或JES3)的系统地址空间，然后启动作业输入子系统 。

子系统的初始化

子系统初始化是一个准备子系统使之在系统中可用的过程。SYS1.PARMLIB中的IEFSSNxx成员包含主要子系统的定义，如JES2或JES3，和一些二级子系统的定义，如NetView和DB2。如果想了解IEFSSNxx成员中包含的次要系统数据的详细信息，请参考特定系统的安装手册。

在系统初始化进程中，定义的子系统被初始化。您应该首先定义主要子系统(JES)，因为其他子系统，诸如DB2，在他们的初始化程序中需要主要子系统提供的服务。如果一个子系统在初始化程序中需要使用其他子系统服务，那么它必须在这些主
490 要子系统之后初始化，否则就会有问题发生。在主要JES初始化后，子系统按照SSN参数指定的PARMLIB中IEFSSNxx成员顺序初始化。比如，SSN=(aa，bb)，那PARMLIB成员IEFSSNaa将在IEFSSNbb之前被处理。

启动/登录/加载处理

在初始化系统且激活作业输入子系统 后，可以提交作业处理。当一个作业通过以下方式被激活：START(批处理作业)，LOGON(分时作业)或MOUNT，系统分配一个新的地址空间。需要注意的是在LOGON之前，操作员必须已经启动VTAM和TSO，它们都有自己的地址空间。

图16-7显示了一些重要的系统地址空间以及VTAM，CICS，TSO，一个TSO用户和一个批处理启始器。每个地址空间默认有2GB的虚存，无论系统运行在31位或64位模式下。

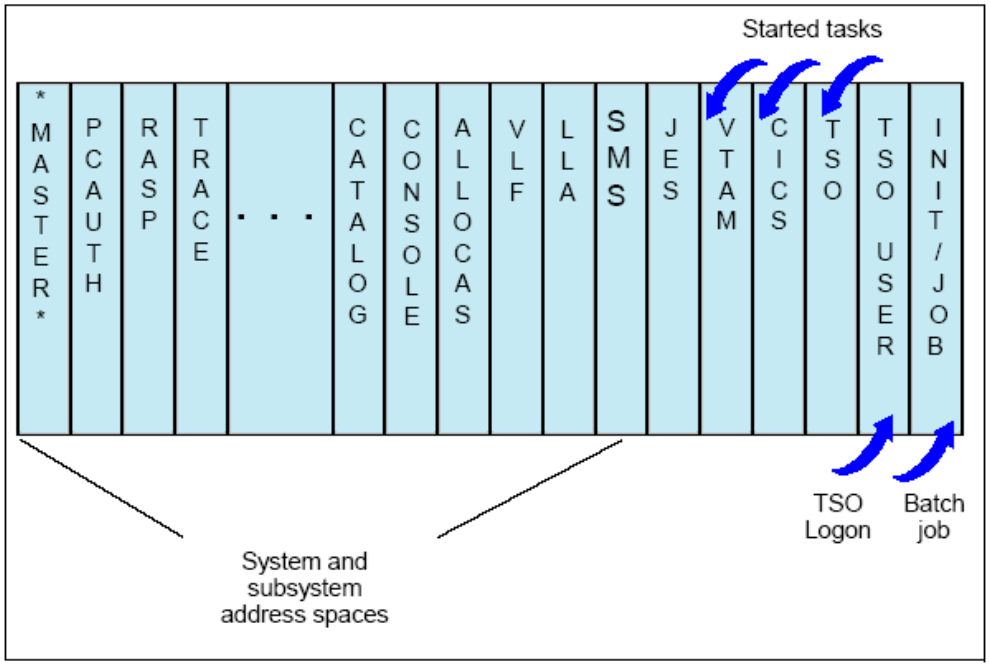


图16-7 多个地址空间的虚存布局

回忆一下101页图3-9”地址空间的存储区域”中每个地址空间的映射。私有区只对那个地址空间可用，但是公共区对所有地址空间都可用。

在z/OS系统初始化过程中，操作员使用系统控制台或与支撑元件(SE)相连的硬件管理控制台(HMC)。在系统控制台上，操作员在内核初始化程序(NIP)阶段初始化系统控制程序。

在NIP阶段，系统可能会提示操作员提供控制z/OS操作的系统参数。系统也会显示信息消息告知操作员初始化进程所在的阶段。

16.8.2 IPL 类型

系统存在多种IPL类型，如下所述：

- 冷启动

  装载(或重新装载)PLPA和清空VIO数据集页的IPL为冷启动。在系统安装完毕之后的第一次IPL总是冷启动，因为这是第一次装载PLPA。之后当PLPA需要重新装载，或需要更改它的内容，或需要恢复它的缺失信息时，这些都是冷启动。在LPA被修改后(比如，当一个新的包含维护信息的SYSRES卷被装载时)，冷启动比较普遍。

- 快速启动

  不需要重装载PLPA，但清空VIO数据集页的IPL为快速启动。(系统重置页表和段表来匹配最新创建的PLPA。)当LPA没有变化但是VIO必须被刷新时，快速启动比较普遍。这阻止了正在使用VIO数据集的作业进行热启动。

- 热启动

  不需要重装载PLPA，保存日志VIO数据集页的IPL为热启动。这将允许在IPL时正在运行的作业之后利用它们的日志VIO数据集重启作业。

  注意：VIO是一种使用内存来存储临时数据集以支持快速访问的方法。然而，和PC上的内存不一样，他们实际上在磁盘上有备份，因此可以用作重启点。很显然，这种方式不应该存储太多的数据，其大小是受限的。

通常来说，推荐做法是做一次冷启动IPL(指定CLPA)。也可以使用其他选项，不过要格外小心避免不可预测的变化或者变化撤销。当您有长时间运行的作业需要在IPL后重新启动时，可以使用热启动，但是还有一种方法即是将长作业分成小块作业，它们传递真实数据集而不是使用VIO。拥有大缓存的现代磁盘控制器减少了长时间保存VIO数据的必要性。

492

不要混淆冷启动IPL(一般使用CLPA来替代术语'冷启动')和JES冷启动。冷启动JES很少被用到，在生产系统中，冷启动JES将会完全毁坏所有在JES中的现存数据。

16.8.3 关闭系统

要关闭系统，必须按照正确顺序依次关闭所有任务。在现代系统上，这些是自动化包应该完成的任务。关闭系统通常只需要一条简单的命令。这能移除自动化本身之外的大多数任务。手动关闭自动化任务，然后输入命令将系统从系统综合体中移除，或发出序列化环。

16.9 总结

z/OS系统程序员的角色是安装，客户化和维护操作系统。

系统程序员必须理解以下方面(或更多)：

- 系统客户化
- 工作负载管理
- 系统性能
- I/O设备配置
- 操作

为了尽可能地提高检索模块任务的性能，z/OS操作系统将一些模块保存在内存中，这些模块包括需要对操作系统快速响应的模块，也包括关键应用程序的模块。连接装配区(LPA)，linklist和授权库是该读取过程的基础。

系统程序员的角色如配置控制台，建立基于消息的自动化也在本章讨论。

系统启动或IPL在以下几点展开了介绍：

- IPL和初始化过程

- IPL类型：冷启动，快速启动和热启动
- IPL的原因

本章关键术语		
HCD	IODF	SYSRES
SMP/E	linklist	IPL
WTOR	PARMLIB	PROCLIB
系统符号(system symbols)	PSA	LPA
内核(nucleus)	LOADPARM	SQA

16.10 复习题

为了帮助您理解本章内容，完成以下复习题：

1. 在473页的例16-2中，假定指定给一个作业某个类，该类有默认PROCLIB连接是PROC00。作业需要调用放置在SYS1.OTHERPRO中的过程。怎么做才能完成这个工作？如果不做任何操作，系统将搜索哪些过程库？
2. 为什么控制台操作常需要自动化？
3. 为什么一个信息和命令架构有助于自动化？
4. 为什么系统重新装载是必须的？
5. 重新装载的3种类型是什么？其中有什么区别？

16.11 思考题

1. 主机被认为是安全的一个原因是它不允许插入式设备；只有系统程序员定义的设备才能被连接和使用。您认为这种观点正确么？
2. 比较474页的”程序搜索顺序”和其他操作系统使用的搜索路径。
3. 联系z/OS和其他您熟悉的操作系统，讨论下列命题：系统程序员的主要目标是避免系统重新装载。

16.12 练习

1. 找出当前IPL使用的是哪个IEASYSxx成员。操作员是否指定后缀使用其他IEASYSxx？
2. 操作员是否指定参数来响应‘SPECIFY SYSTEM PARAMETERS’系统消息？如果是，那找出参数的相关PARMLIB成员，如果操作员不回应消息时系统获得的参数值是什么？
3. 完成以下步骤：
   a. 在您的系统中，找出IPL设备地址和IPL卷。进入SDSF，输入ULOG，然后输入/D IPLINFO。
   b. IODF设备地址是什么？

c. IPL使用的是哪个LOADxx成员？哪个数据集包含这个LOADxx成员？
d. 浏览该成员；系统使用的系统目录的名字是什么？
e. 当前使用的IODF数据集的名字是什么？输入/D IOS,CONFIG
f. 系统参数可以来自于一些PARMLIB数据集。输入/D PARBLIB。您的系统使用的PARMLIB数据集是什么？

495

496

17

第 17 章 使用 SMP/E

目标：作为一名系统程序员，您的责任所在是确保所有软件产品和它们的修正程序都正常安装在系统上。您也要确保所有产品都在合适的级别安装，这样系统元件才能协同工作。首先，这听起来或许不是很困难，但是随着软件配置复杂度的增加，监控所有系统元件的任务也变得更艰难。

SMP/E是z/OS系统中安装和升级软件的主要方法。SMP/E巩固数据安装，允许更灵活地选择安装变化，提供对话框式用户界面，并支持动态分配数据集。

学完本章后，您将可以解释：

- 什么是SMP/E。
- 系统修改是什么。
- SMP/E使用的数据集。
- SMP/E如何帮助您安装和维护产品，监控产品的变化。

497

17.1 什么是 SMP/E?

SMP/E是一个z/OS工具，用于管理z/OS系统上软件产品的安装和追踪这些产品的修正程序。SMP/E通过以下方法在组件级别控制这些变化：

- 从大量潜在变化中选择合适级别的代码安装
- 调用系统实用程序来安装变化。
- 记录安装的变化，提供工具使您能查询软件的状态并在必要时取消变化。

所有的代码和修正程序都可在SMP/E数据库中定位，SMP/E数据库也叫CSI(consolidated software inventory)，它由一个或多个VSAM数据集组成。

SMP/E可以使用批处理作业或使用ISPF/PDF下的对话框运行。通过SMP/E对话框，您可以交互查询SMP/E数据库，提交作业来处理SMP/E命令。我们将在514页的17.11章节"使用SMP/E"中讨论一些基本命令。

相关阅读：SMP/E的标准参考书为IBM出版物，SMP/E User's Guide。您可以在z/OS互联网知识库网站找到这本书和相关出版物：

http://www.ibm.com/servers/eserver/zseries/zos/bkserv/

17.2 从 SMP/E 角度看系统

z/OS系统可能看起来是完整的代码块来驱动CPU。但实际上，z/OS是一个复杂系统，包含了许多不同的小块代码。每一小块代码块都完成系统一个特定的功能(如图17-1)。

498

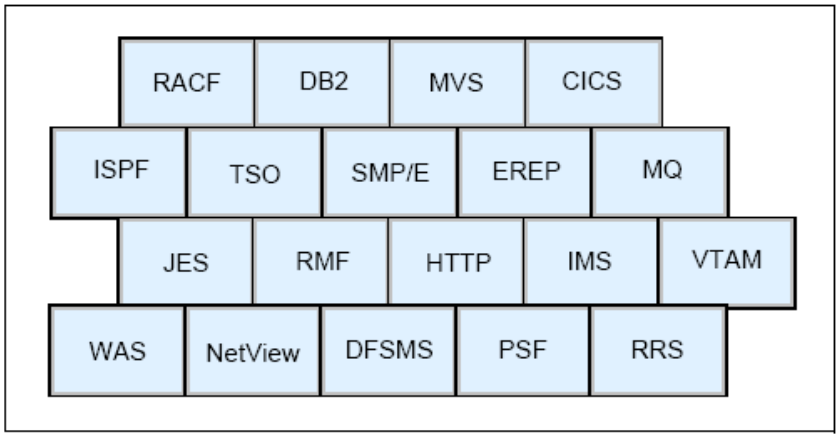


图17-1 从SMP/E角度看系统

例如，z/OS系统中的一些功能包括：

- 基本控制程序(BCP)

- CICS
- DFSMS
- HTTP服务器
- ISPF
- JES2 或JES3
- 开放系统适配器/支持工具 (OSA/SF)
- 资源评估工具 (RMF)
- 系统显示和搜索工具(SDSF)
- SMP/E
- 分时选项/扩展 (TSO/E)
- WebSphere MQ
- z/OS UNIX系统服务(z/OS UNIX)

每个系统功能都由一个或多个装入模块组成。在一个z/OS环境下，一个装入模块代表机器识别的可执行代码的基本单元。一个或多个目标模块组合起来，通过连接-编辑程序处理之后创建为一个装入模块。模块的连接-编辑是一个解析外部引用和地址的过程。系统功能因此就是一个或多个目标模块组合起来并通过连接-编辑创建而成的。

要查看目标模块由何处而来，让我们来看图17-2中的例子。

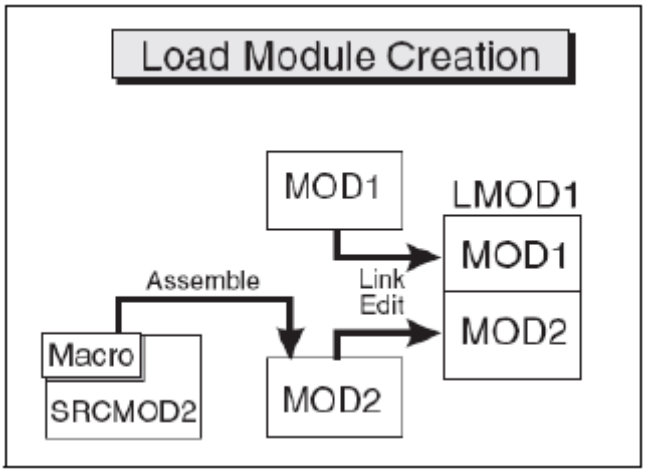


图17-2 装入模块的创建

大多数时候，目标模块作为是产品的一部分发送给您。在本例中，目标模块MOD1被当成是产品一部分发送。其余的时候，您可能需要编译由产品供应商发送的源代码来创建目标模块。您可以更改源码，然后编译成目标模块。本例中，SRCMOD2是源代码，通过编译后创建目标模块MOD2。当编译完成后，您将MOD2和MOD1进行连接-编辑，组成装入模块LMOD1。

除了目标模块和源代码外，大多数产品还发布其他部分，比如宏，帮助面板，CLIST和其他z/OS库成员。这些模块，宏和其他类型的数据和代码都是系统中的基本构建块。所有这些构建块成为叫元件(element)。

元件通常关联并依靠安装在同一z/OS系统上的其他产品或服务。它们描述了软件和安装在同一z/OS系统上的其他产品或服务之间的关系。

17.3 更改系统元件

随着时间的推移，您将会需要改变z/OS系统中的软件。这些改变对提高产品使用性和可靠性是非常必要的。由于各种各样的原因，您或者想要在系统上加入一些新功能，升级一些系统元件，或更改一些元件。软件，无论它是产品或服务，都包含诸如宏，模块，源码和其他类型的数据(比如CLIST或样例过程)的元件。

500

17.3.1 什么是 SYSMOD？

SYSMOD
SMP/E 的输入数据，定义了 z/OS 中的元素的引入，替代或更新。

SMP/E可以安装大量不同的系统更新，前提是这些更新打包成一个系统修改或SYSMOD。实际上，一个SYSMOD是元件和控制信息的打包，SMP/E用这些控制信息来安装和追踪系统修改。

SYSMOD由一些元件的组合以及控制信息组成。它们由以下两部分组成：

- 修正控制语句(MCS)，前两个字母以++开始，MCS告诉SMP/E:
  - 更新或更换哪些元件
  - SYSMOD如何与产品软件和其他SYSMOD相关
  - 其他特定的安装信息
- 修正文本，它们是目标模块，宏和SYSMOD提供的其他元件

17.3.2 SYSMOD 的类型

有四种类型的SYSMOD，每一种都支持一项任务：

FUNCTION(功能) 该类型的SYSMOD在系统中引入一个新产品，一个产品新版本或新发布，或更新一个现有产品的功能。

PTF， 程序临时补丁(PTF)是IBM提供的针对一个报告的问题的修正程序。它们可以在所有环境下安装。PTF是预防服务，用来避免某些已知的问题，尽管这些问题或许没有在您的系统上出现；或者PTF可以用来改正您已经碰到的问题。PTF的安装必须在安装其Function SYSMOD之后，通常也在安装其他PTF之后。

APAR 授权程序分析报告(APAR)是在问题第一次被报告后，用于修正或跳过该问题的临时补丁。一个APAR或许不适用于您的环境。一个APAR的安装必须在安装其Function SYSMOD之后，有时也在安装某个PTF之后。这就是说，一个APAR被设计成在元件的特定的预防服务级别安装。

501

USERMOD 该类型的SYSMOD由用户创建，用来改变IBM代码或

许在系统中添加独立功能。USERMOD的安装必须安装其Function SYSMOD之后，有时也在安装某个PTF，APAR补丁或其他USERMOD之后。

SMP/E追踪每个元件的功能和服务级别，它使用以上的SYSMOD层次来决定某元件的哪些功能和服务级别应该安装以及为元件安装更新的正确次序。

17.4 在系统中引入元件

您可以改变您的系统的一种方式是引入新元件。若使用SMP/E完成该工作，您可以安装一个Function SYSMOD。Function SYSMOD引入新产品、产品的新版本或新发布，或更新现有产品的功能。所有其他类型的SYSMOD依赖于FUNCTION SYSMOD，因为它们都是最初由Function SYSMOD引入的元件的修正程序。

当我们说安装一个Function SYSMOD时，我们指替换在系统数据集或库中的所有产品元件。例如这些库有SYS1.LPALIB，SYS1.MIGLIB，和SYS1.SVCLIB。

502 图17-3显示了在生产系统库中创建可执行代码的过程。

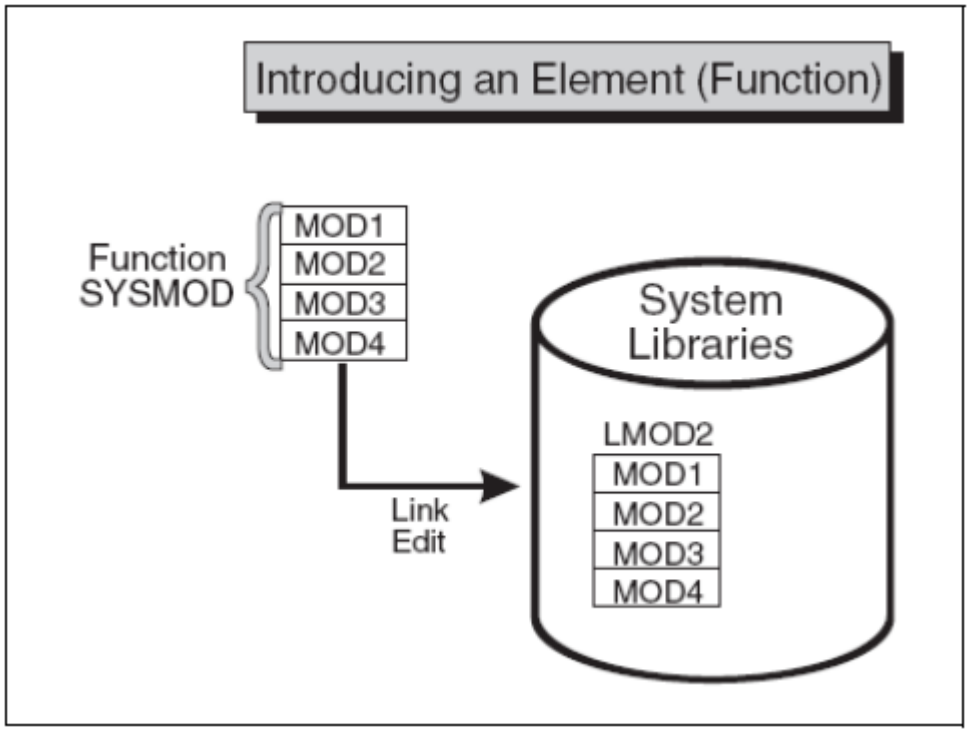


图17-3 引入元件

如图17-3，Function SYSMOD的安装需要连接-编辑目标模块MOD1，MOD2，MOD3和MOD4来创建装入模块LMOD2。通过Function SYSMOD的安装，装入模块LMOD2的可执行代码成功安装在系统库中。

Function SYSMOD分为两种：

- 基本Function SYSMOD添加或替换整个系统功能。例如SMP/E和JES2这两个基本功能。
- 依赖Function SYSMOD对现有系统功能提供附加的功能。‘依赖’是因为它的安装依赖于已经安装好的基本功能。例如SMP/E的语言支持特征就是依赖功能。

这两种SYSMOD都是用来引入新元件的。图17-4显示了一个简单Function SYSMOD引入四个元件的例子。

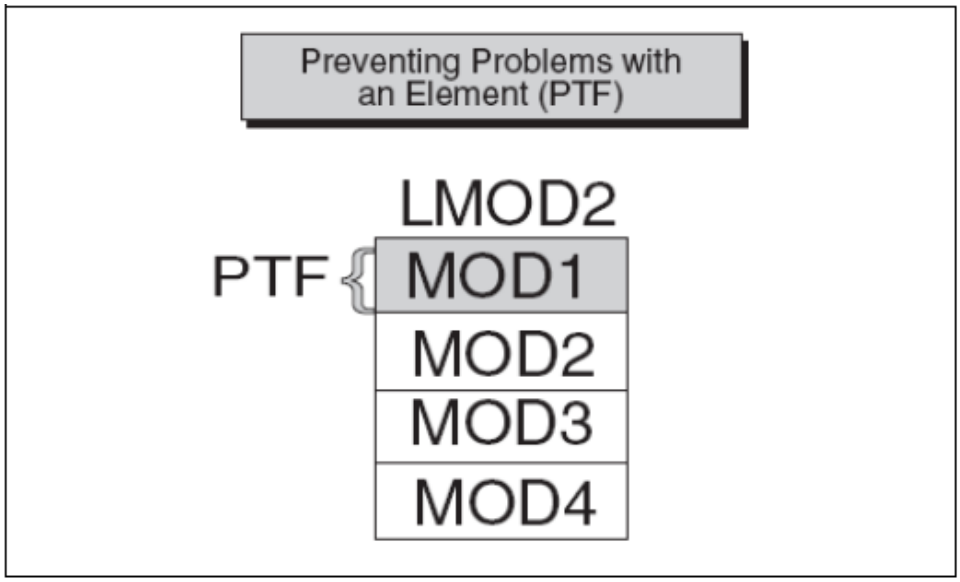

```
++FUNCTION(FUN0001)          /* SYSMOD type and identifier.   */.
++VER(Z038)                  /* For an OS/390 system          */.
++MOD(MOD1)  RELFILE(1)      /* Introduce this module         */
             DISTLIB(AOSFB)  /* in this distribution library. */.
++MOD(MOD2)  RELFILE(1)      /* Introduce this module         */
             DISTLIB(AOSFB)  /* in this distribution library. */.
++MOD(MOD3)  RELFILE(1)      /* Introduce this module         */
             DISTLIB(AOSFB)  /* in this distribution library. */.
++MOD(MOD4)  RELFILE(1)      /* Introduce this module         */
             DISTLIB(AOSFB)  /* in this distribution library. */.
```

图17-4 一个简单的Function SYSMOD例子

17.5 利用元件预防或修正问题

当一个软件元件发现有问题时，IBM提供客户一个经测试的补丁。这个补丁以程序临时补丁(PTF)的形式呈现。虽然您可能没有经历到PTF有意防止的问题，但在系统上安装PTF依旧是明智之举。PTF SYSMOD用来安装PTF，防止您的系统发生对应的问题。

通常来说，PTF替换和更新一个或多个某系统功能的完整元件。一起看一下504页的图17-5。



504

图17-5 利用元件防止问题

在图17-5中，我们看到一个之前已经安装的装入模块，LMOD2。如果我们想要替代MOD1元件，我们应该安装包含模块MOD1的PTF SYSMOD。PTF SYSMOD用正确的元件替换错误的元件。作为PTF SYSMOD安装的一部分，SMP/E重新连接LMOD2让它包含新的正确的MOD1版本。

图17-6显示一个简单的PTF SYSMOD的例子。

```
++PTF(PTF0001)              /* SYSMOD type and identifier.   */.
++VER(Z038) FMID(FUN0001)   /* Apply to this product.        */.
++MOD(MOD1)                 /* Replace this module           */
           DISTLIB(AOSFB)   /* in this distribution library. */.
...
... object code for module
...
```

图17-6 一个简单PTF SYSMOD的例子

PTF SYSMOD总是依赖于Function SYSMOD的安装。有些情况下，PTF SYSMOD也可能依赖于其他PTF SYSMOD的安装。这样的依赖关系叫做先决条件。我们在讨论追踪系统元件的复杂度时会探究一个典型的PTF先决条件。

17.6 利用元件修正问题

APAR
影响到某个用户的IBM 系统控制程序或特许程序中缺陷的临时修正程序。

有时在一个严重问题发生时，您可能认为有必要在PTF发布之前就修正它。在这种情况下，IBM提供您一个授权程序分析报告(APAR)。APAR是一个补丁，它快速修正元件的某个特定区域或替代一个错误元件。您安装APAR SYSMOD来应用一个补丁，从而更新不正确的元件。

505

在图17-7中，阴影部分显示了MOD2区域包含一个错误。

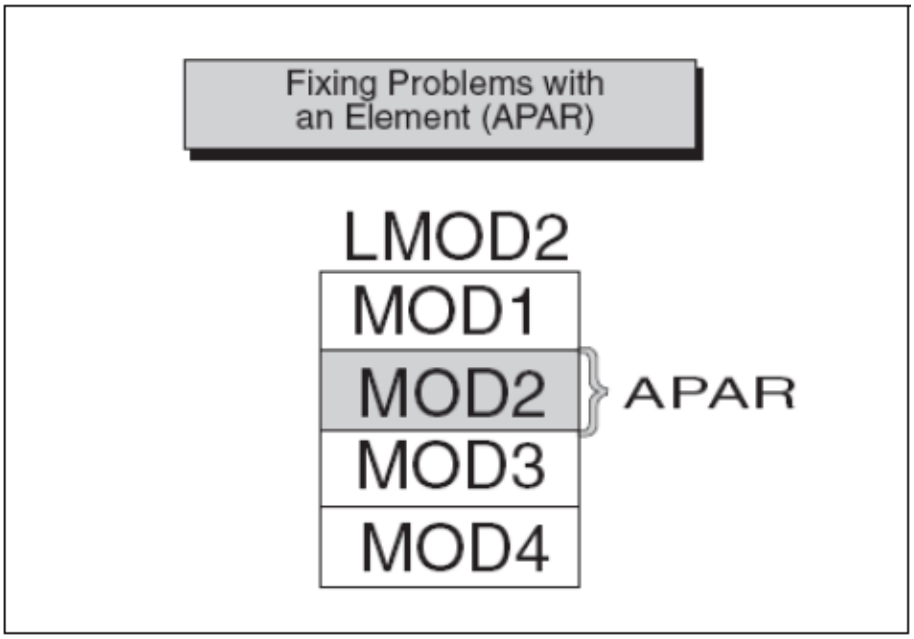


图17-7 利用元件修正问题

APAR SYSMOD的操作是为目标模块MOD2提供一个修正程序。在安装APAR SYSMOD过程中，装入模块LMOD2中的MOD2被更新(和修正)。

图17-8显示了一个简单的APAR SYSMOD的例子。

```
++APAR(APAR001)              /* SYSMOD type and identifier.   */.
++VER(Z038) FMID(FUN0001)    /* Apply to this product         */
            PRE(UZ00004)     /* at this service level.        */.
++ZAP(MOD2)                  /* Update this module            */
            DISTLIB(AOSFB)   /* in this distribution library. */.
...
... zap control statements
...
```

图17-8 一个简单的APAR SYSMOD的例子

506

总是先行安装一个Function SYSMOD是APAR SYSMOD的先决条件，同时它也可以依赖于其他PTF或APAR SYSMOD的安装。

17.7 客户化元件——USERMOD SYSMOD

如果您有这样一个需求，需要产品完成和它的设计方式不一样的任务，那您可能需要客户化系统中的元件。IBM提供您一些模块来允许您裁剪IBM代码来以满足您

特定的需求。在完成您想得到的改变后，您通过安装USERMOD SYSMOD来在系统中添加这些模块。该SYSMOD可以替换或更新某个元件，或在系统中引入一个全新的用户撰写的元件。在这两种情况下，USERMOD SYSMOD都由您构建，用来改变IBM代码或在系统中添加您自己的代码。

在图17-9中，通过USERMOD SYSMOD的安装来更新了MOD3。

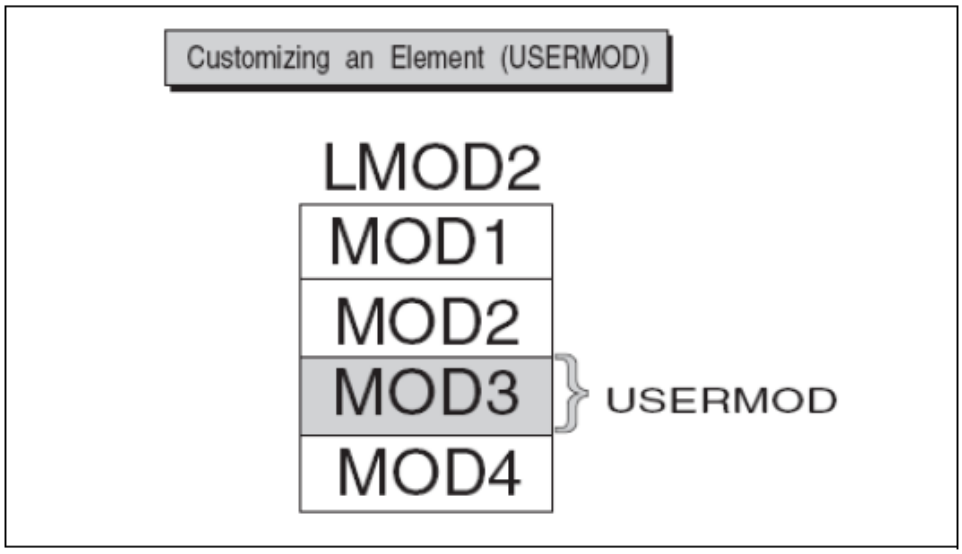


图17-9 客户化元件

图17-10显示了一个简单的USERMOD SYSMOD例子。

```
++USERMOD(USRMOD1)              /* SYSMOD type and identifier.   */.
++VER(Z038) FMID(FUN0001)       /* Apply to this product         */
            PRE(UZ00004)        /* at this service level.        */.
++SRCUPD(JESMOD3)               /* Update this source module     */
            DISTLIB(AOSFB)      /* in this distribution library. */.
...
... update control statements
...
```

图17-10 一个简单的USERMOD SYSMOD例子

USERMOD SYSMOD的先决条件是安装Function SYSMOD，也可能是安装其他PTF，APAR或USERMOD SYSMOD。

17.7.1 SYSMOD 先决条件和并行条件

您已经知道，PTF，APAR和USERMOD SYSMOD都把Function SYSMOD作为先决条件。他们除了都依赖于Function SYSMOD外，还存在以下关系：

- PTF SYSMOD可能还依赖于其他的PTF SYSMOD。

- APAR SYSMOD可能依赖于PTF SYSMOD或其他的APAR SYSMOD。
- USERMOD SYSMOD可能依赖于PTF SYSMOD，APAR SYSMOD和其他的USERMOD SYSMOD。

有时，PTF甚至APAR依赖于其他PTF SYSMOD叫做并行条件。

思考一下这些依赖关系的复杂性。当考虑到许多库中的上百个装入模块时，复杂度就大大增加，对工具，例如SMP/E，的需求就显而易见了。

让我们观察一下这些依赖关系对维护z/OS环境下的软件存在的影响。

17.8 追踪系统元件

当我们观察z/OS维护流程时，追踪系统元件和它们的修正程序的重要之处就变得显而易见起来。通常，一个PTF包含多个元件的替换。

在509页图17-11中示例中，PTF1包含两个模块的替换，MOD1和MOD2。虽然装入模块LMOD2包含四个模块，只有其中两个模块将要被替换。

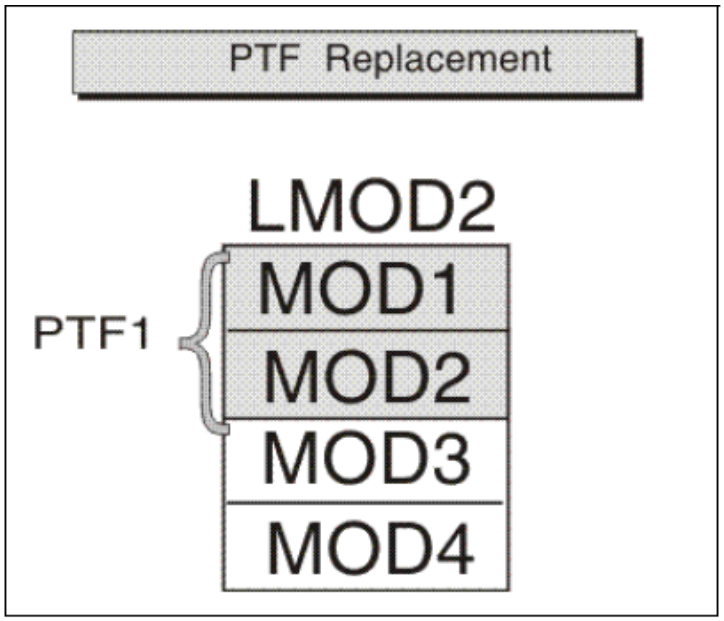


图17-11 PTF替换

但是如果第二个PTF替换一个模块中的一些代码，而这个模块被PTF1替换过了，会发生什么呢？让我们看一下图17-12。

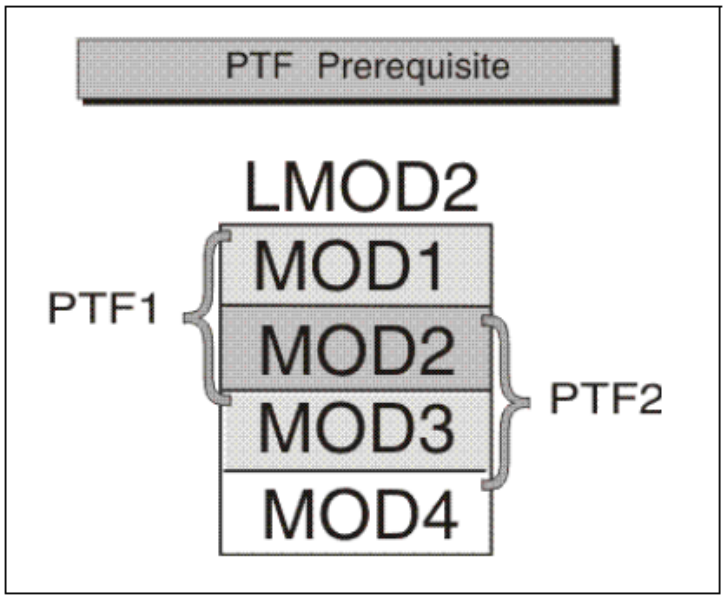


图17-12 PTF先决条件

在这个例子中，PTF2包含MOD2和MOD3模块的替换。为了MOD1，MOD2和MOD3交互顺利，PTF1必须在PTF2之前安装。因为PTF2提供的MOD3可能依赖PTF1版本的MOD1。这个依赖关系就是先决条件。SYSMOD先决条件由SYSMOD包中修正控制语句(MCS)指定，我们在1.1.2话题"改变系统元件"中讨论过。

除了要追踪先决条件，追踪系统元件还有另外一个重要原因：同样的模块通常是不同装入模块的一部分。我们来看看图17-13中的例子。

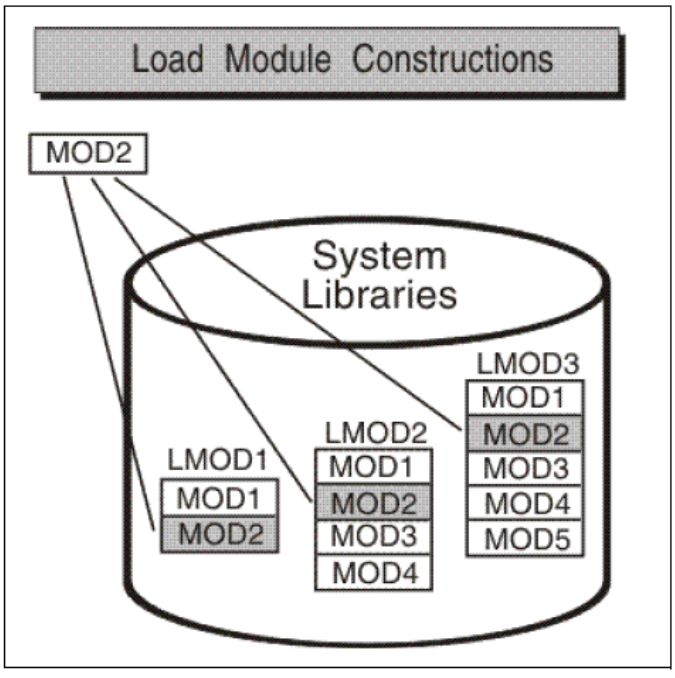


图17-13 装入模块构架

在图17-13中，MOD2模块同时存在于LMOD1，LMOD2和LMOD3中。当引入一个PTF替换MOD2元件时，必须替换所有装入模块中存在的MOD2。因此，我们有必要追踪所有装入模块和它们包含的模块。

510

您现在应该能够理解追踪系统元件和它们的修正程序级别是如何的复杂。我们简略地看一下如何应用SMP/E的追踪功能。

17.9 追踪和控制必要条件

为了成功地追踪和控制元件，所有元件、它们的修正程序和更新都必须在SMP/E中清晰地定义。SMP/E依靠修正程序标识符来完成该任务。与每个元件关联的有三类修正程序标识符，如下：

- 功能修正程序标识符(FMID)标识Function SYSMOD，它将元件引入到系统中。
- 替换修正程序标识符(RMID)标识最后替换某元件的SYSMOD(大多数情况下为PTF SYSMOD)。
- 更新修正程序标识符(UMID)标识自从某元件被替换之后，对它实施更新的SYSMOD。

SMP/E使用这些修正程序标识符来追踪所有安装在您系统上的SYSMOD。这就确保了它们以合适的顺序安装。既然我们意识到元件追踪的必要性，也知道了可用SMP/E追踪的事物的类型，下面我们来看一下SMP/E如何完成追踪功能的。

17.10 SMP/E 如何工作？

我们回顾一下之前对于功能是如何安装到系统中的讨论。我们从诸如模块、宏和源代码的元件开始。这些元件然后被工具，如编译器或连接-编辑器，处理来创建装入模块。装入模块包含机器可识别的可执行代码。

您的在z/OS环境下的生产系统包含z/OS操作系统以及所有日常工作需要的代码。这些东西存放在何处呢？它们如何组织的呢？我们一起来探究。

17.10.1 分配(程序)库和目标库

为了正确执行它的处理工作，SMP/E必须维护大量信息，包括结构，内容和它管理的软件的修正状态。想象一下所有这些SMP/E必须维护的信息就好像它们是放在一个公共图书馆中的信息。

在一个公共图书馆中，您看到书架都放满了书，抽屉里放置了卡片目录，图书馆中每本书都有一张对应的卡片。这些卡片包含这些信息：标题、作者、出版日期、书的类型和书在书架上的具体位置。

511

在SMP/E环境中，有两种类型的”书架”。它们分别代表的是分配(程序)库

(distribution library)和目标库(target library)。和公共图书馆中书架上放置书类似，分配(程序)库和目标库中放置系统的元件。

分配(程序)库包含所有元件，比如模块和宏，这些是运行系统的输入。分配(程序)库一个很重要的用途是备份。如果在生产系统上一个元件发生一个严重的错误，该元件可以被分配(程序)库中具有稳定级别的原件所替代。

目标库包含用来运行系统的可执行代码。

17.10.2 CSI 介绍

您想到图书馆和SMP/E的类似之处时，有一点我们还没有考虑到。在图书馆，有卡片目录来帮助您找到书或者您要寻找的任何信息。SMP/E则以CSI(consolidated software inventory)的形式提供相同类型的追踪机制。

CSI
包含用户系统结构信息的SMP/E数据集。

CSI数据集包含所有SMP/E用来追踪分配(程序)库和目标库的信息。正如卡片目录中每本书都有一张卡片，CSI针对库中每个元件都有一条条目信息。CSI条目包含元件名字，类型，历史信息，元件如何被引入到系统，元件在分配(程序)库和目标库中的具体位置。CSI并不包含元件本身，而是包含元件的描述信息。

让我们来看一看CSI的这些条目具体是如何安排的。

SMP/E区

在公共图书馆的卡片目录中，卡片按照作者的姓氏、书的题目和主题的字母顺序进行排序。在CSI中，元件在分配(程序)库或目标库中的条目按照它们的安装状态分组。即是，代表分配(程序)库中元件的条目在一个分配区(distribution zone)中。代表目标库中元件的条目在目标区(target zone)中。这些区的功能就像是公共图书馆中放置卡片目录的抽屉。

除了分配区和目标区，SMP/E CSI也包含一个全局区。图17-14显示了SMP/E区和库之间的关系。

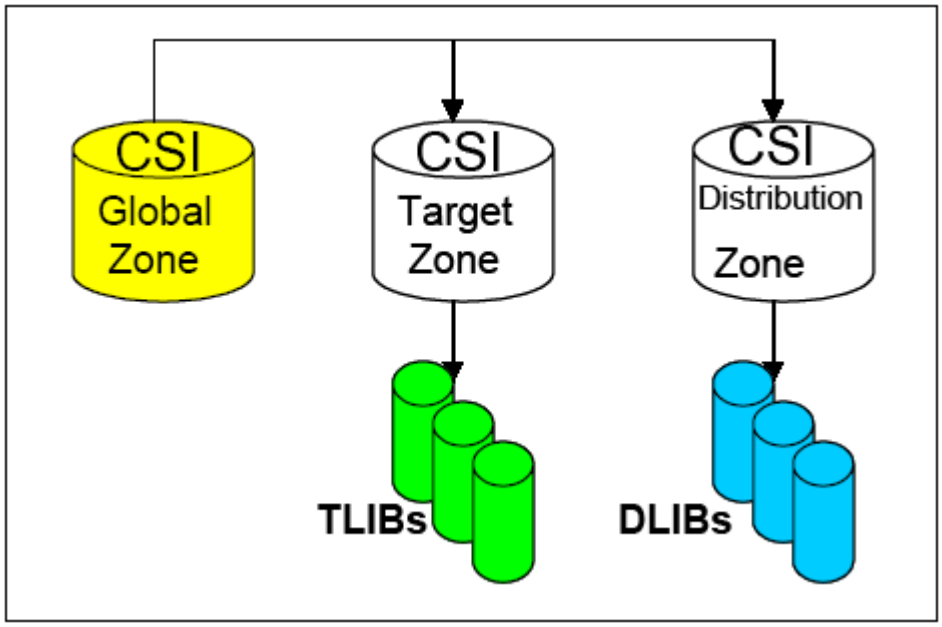


图17-14 SMP/E区和库之间的关系

全局区包含：

- SMP/E用来识别和描述每个目标区和分配区的条目
- SMP/E处理选项的信息
- SMP/E已经开始处理的全部SYSMOD的状态信息
- 需要特殊处理或存在错误的SYSMOD的异常数据。

在SMP/E中，当我们谈及异常数据时，我们常指的是HOLDDATA。HOLDDATA经常在产品中提供，表示某个指定的SYSMOD禁止安装。阻挠SYSMOD安装的原因可能是：

- PTF有错误，在改正之前不该安装 (ERROR HOLD)。
- 在SYSMOD安装前需要某些系统行为 (SYSTEM HOLD)。
- 在SYSMOD安装前，用户想要完成某些行为 (USER HOLD)。

现在您可以看到系统中的所有元件是如何结合的，SMP/E是如何安装，修改和追踪它们的。

513

17.11 使用 SMP/E

现在您已经熟悉SMP/E和它的功能，您可能会思索您还需要知道哪些知识才能开始使用SMP/E。SMP/E流程由3个简单基本命令完成：RECEIVE，APPLY和ACCCEPT。我们来看一下这些命令。

17.11.1 使用 RECEIVE 命令

RECEIVE命令允许您将一个在SMP/E之外的SYSMOD加入SMP/E库域中，并开始构造CSI条目描述它们。之后输入处理中，这些条目可以用来查询它们。最近，RECEIVE的数据源可以是网站上的电子文件，虽然通常源都来自于磁带或第三方生产商的媒介。

该处理完成以下一些任务(图17-15)：

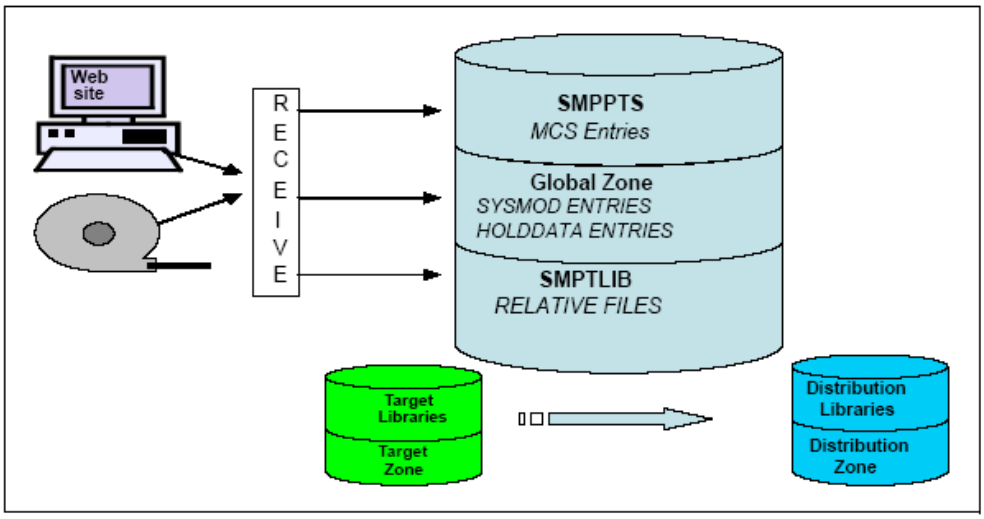


图17-15 SMP/E RECEIVE处理过程

- 构建全局区中描述SYSMOD的条目。
- 确保SYSMOD有效，诸如与CSI中安装产品相关联的修正控制语句(MCS)的语法。
- 在库中安装SYSMOD。例如，PTF临时存放库。
- 评估HOLDDATA，确保错误不被引入。

514

在RECEIVE处理过程中，每个SYSMOD的MCS都被复制到一个SMP/E临时存储区，叫做SMPPTS数据集，SMPPTS包括该SYSMOD的内嵌元件的替换或更新。将相关文件中的元件打包且和MCS分开单独存放在RELFILE中，Function SYSMOD经常使用该方式。相关文件存储在另一个叫SMPTLIB数据集的临时存储区中。

SMP/E用它接收的SYSMOD信息来更新全局区：

在维护系统过程中，您需要安装服务和处理相关的HOLDDATA。比如，假设IBM提供您一盘服务磁带(比如CBPDO或ESO磁带)，您想要在系统上安装它。第一步就是接收磁带上包含的SYSMOD和HOLDDATA，命令如下：

```
SET BDY(GLOBAL).
RECEIVE.
```

以上命令促使SMP/E接收磁带上所有SYSMOD和HOLDDATA。

RECEIVE命令举例

要只接受需要特殊处理或有错误状态的HOLDDATA，使用以下命令：

```
SET BDY(GLOBAL).
RECEIVE HOLDDATA.
```

要只接受SYSMOD安装到全局区中，使用以下命令：

```
SET BDY(GLOBAL).
RECEIVE SYSMODS.
```

要接受某个特定产品(诸如WebSphere Application Server)的所有SYSMOD，包括HOLDDATA，使用以下命令：

```
SET BDY(GLOBAL).
RECEIVE FORFMID(H28W500).
```

17.11.2 使用 APPLY 命令

APPLY命令指定哪些接收到的SYSMOD需要安装在目标库(TLIB)中。SMP/E也确保所有其他要求的SYSMOD(先决条件)都已经安装，或者正在按照正确顺序并行安装。元件源是SMPTLIB数据集，SMPPTS数据集或一些间接库，这取决于它是如何打包的。SMP/E的这阶段处理包含以下几个步骤：

- 根据提供的输入文本和需要改变的目标模块的类型，执行合适的实用程序安装SYSMOD至目标库中。
- 确保目标区中新的SYSMOD和其他SYSMOD之间的关系是正确的。
- 更改CSI来显示更新模块。

APPLY命令更新系统库，在生产系统上要小心使用该命令。推荐做法是最初使用生产目标库和目标区的拷贝。

目标区反应了目标库中的内容。因此，在实用程序执行完并且目标区被更新后，它就能准确反映这些库的状态。

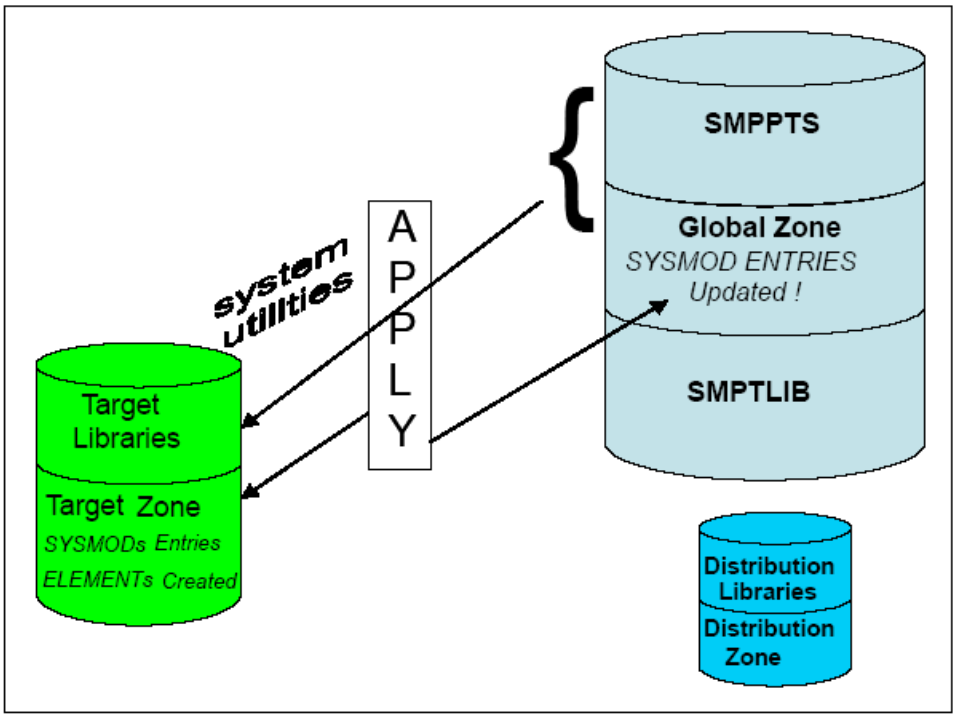


图17-16 SMP/E APPLY处理

516

APPLY处理(图17-16)将精确更新目标区：

- 全局区的所有SYSMOD条目都被更新，反映出SYSMOD已经应用到目标区中。
- 目标区准确反映了每条APPLY处理的SYSMOD条目。元件条目(诸如MOD和LMOD)也在目标区中被创建。
- BACKUP条目在SMPSCDS数据集中被创建，这样可以在需要时恢复SYSMOD。

类似于RECEIVE处理，APPLY命令有很多不同的操作数，增加了查看选择SYSMOD安装在目标库的灵活性，同时也提供输出分类。使用的命令指示SMP/E您想要安装什么。

只安装PTF SYSMOD，输入以下命令：

```
SET BDY(ZOSTGT1).
APPLY PTFS.
```

要选择PTF SYSMOD，在命令中指定名字，例如：

```
SET BDY(ZOSTGT1).
APPLY SELECT(UZ00001, UZ00002).
```

有时，您可能需要在目标库中安装校正的补丁(APAR)或用户修正程序(USERMOD)，比如：

```
SET BDY(ZOSTGT1).
APPLY APARS
```

USERMODS.

其他时候，您可能想从磁带中更新一个选择的产品，例如：

```
SET BDY(ZOSTGT1).
APPLY PTFS
FORFMID(H28W500).
或
SET BDY (ZOSTGT1).
APPLY FORFMID(H28W500).
```

在这两个例子中，SMP/E应用所有FMID对应的可用的PTF。除非您指定其他类型，否则默认的SYSMOD类型是PTF。 517

使用APPLY CHECK

有时，在安装之前您需要查看安装将会包含哪些SYSMOD。您可以在命令中包括CHECK操作数达到此目的，如下：

```
SET BDY(MVSTGT1).
APPLY PTFS
APARS
FORFMID(HOP1)
GROUPEXTEND CHECK.
```

当这些命令完成时，您可以检查SYSMOD状态报告来查看信息：如果您没有指定CHECK操作数，哪些SYSMOD将会被安装。如果您对试运行结果满意，那您可以去掉CHECK操作数再输入命令，这就真正地安装了SYSMOD。

17.11.3 使用 ACCEPT 命令

当一个SYSMOD在目标库中安装后，并且您已经对它进行了测试，然后您就可以通过ACCEPT命令接受变化。这一步将选择的SYSMOD安装至相关的分配(程序)库中。

518 在ACCEPT命令中，您通过指定操作数来指示哪些接收的SYSMOD要安装。这个阶段中，SMP/E也要确保选择每个元件的正确的功能级别。

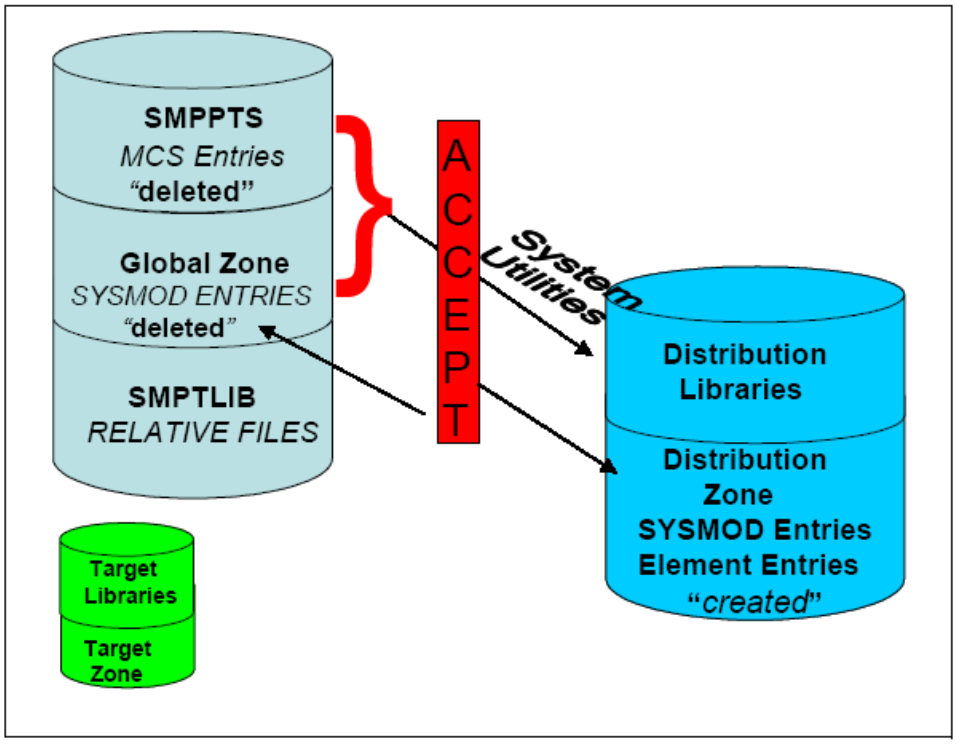


图17-17 SMP/E ACCEPT处理

ACCEPT命令完成以下任务(图17-17)：

- 更新分配区中的目标元件CSI条目。
- 使用SYSMOD内容作为输入，重建或创建分配(程序)库中的目标元件。
- 验证受影响的模块和SYSMOD的目标区CSI条目，确保它们与库中内容一致。
- 完成荒废或过期元件的整理工作。ACCEPT处理会删除这些受影响的SYSMOD所对应的全局区CSI条目，PTS成员和SMPTLIB。例如，对于已经接受到分配区的SYSMOD，ACCEPT删除其对应的全局区SYSMOD条目和SMPPTS数据集中的MCS语句。

作为进一步选项，您可以跳过SMP/E清理全局区的步骤。如果这样，SMP/E保存这些信息。

519

SMP/E提供"停止"ACCEPT处理功能，这样您可以确保安装SYSMOD前所有先决条件都已满足。您可以检视将要发生什么(帮助您发现问题)而无需真正修改分配(程序)库。

重要：如果SYSMOD发生错误，不要ACCEPT它。使用RESTORE过程选取被选中的SYSMOD更新的模块，并将分配(程序)库中指定的模块作为输入，重新建立它们在目标库中的备份。 RESTORE也更新目标区CSI条目来反映SYSMOD已经移走。当ACCEPT操作完成后，就没有办法来恢复系统了，那时变化将是永久的。

在应用SYSMOD至目标区中后，您可以告诉SMP/E在分配区中只安装合格的SYSMOD：

```
SET BDY(ZOSDLB1).
ACCEPT PTFS.
```

要选择特定的PTF SYSMODS安装：

```
SET BDY(ZOSDLB1).
ACCEPT SELECT(UZ00001,UZ00002).
```

当您需要更新某个产品的所有SYSMOD时：

```
SET BDY(ZOSDLB1).
ACCEPT PTFS
FORFMID(H28W500).
```

或

```
SET BDY(ZOSDLB1).
ACCEPT FORFMID(H28W500).
```

注意：在上面的2种情况中，SMP/E为FMID为H28W500的产品接受了其所有可以接受的PTF，该产品位于分配(程序)库ZOSDLB1中。

接受先决条件SYSMOD

当安装SYSMOD时，您可能并不知道它有无先决条件(有时，一个ERROR SYSMOD被停止)。这种情况下，您可以通过指定GROUPEXTEND操作数让
520 SMP/E检查是否有等价的(或接替的)SYSMOD可用：

```
SET BDY(ZOSDLB1).
ACCEPT PTFS
FORFMID(H28W500)
GROUPEXTEND.
```

注意：如果SMP/E找不到需要的SYSMOD，它将寻找并且使用一个替换的SYSMOD。

有一个好方法可以在您安装之前查看包含的SYSMOD，即使用CHECK操作数：

```
SET BDY(ZOSTGT1).
ACCEPT PTFS
FORMFMID(H28W500)
GROUPEXTEND
CHECK.
```

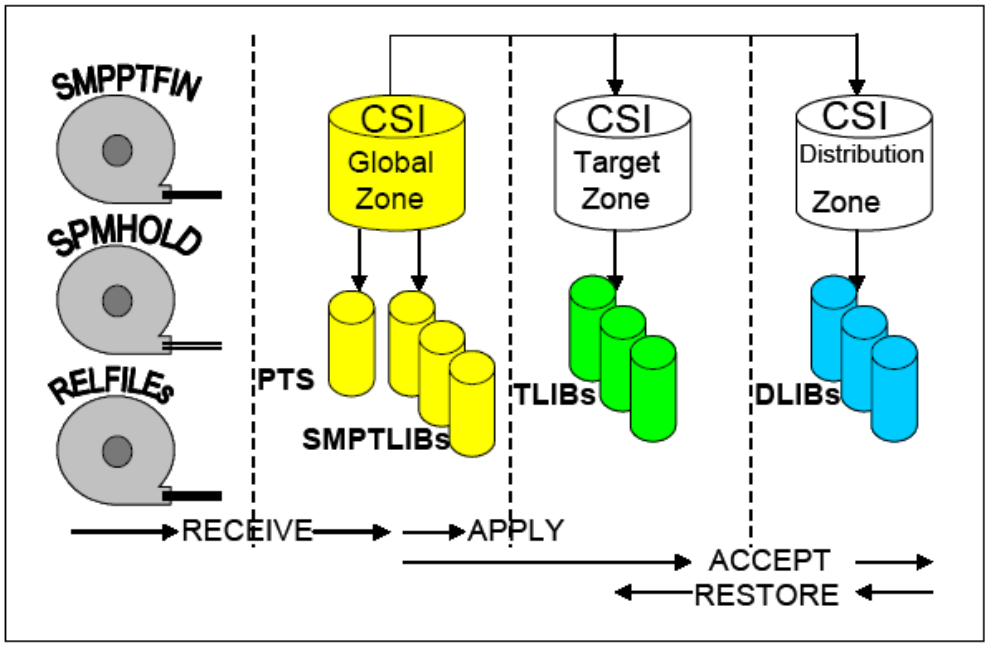
ACCEPT报告

当最后一阶段完成后，以下的报告将会协助您评估结果：

- SYSMOD状态报告(SYSMOD Status Report)——基于您在ACCEPT命令中指定的操作数，提供每个SYSMOD上发生的处理概要。
- 元件概要报告(Element Summary Report)——提供ACCEPT处理中受影响的每个元件和它们所在的库的详细报告。

- 引起者SYSMOD概要报告(Causer SYSMOD Summary Report)——提供引起其他SYSMOD失败的SYSMOD清单，描述要想正常处理必须修正的错误。
- 文件分配报告(File Allocation Report)——提供一个ACCEPT处理使用的数据集列表，提供这些数据集的信息。

521



522 *图17-18 SMP/E处理概览*

```
//DEFINE   JOB   'accounting info',MSGLEVEL=(1,1)
//STEP01   EXEC  PGM=IDCAMS
//CSIVOL   DD    UNIT=3380,VOL=SER=volid1,DISP=SHR
//SYSPRINT DD    SYSOUT=A
//SYSIN    DD    *
  DEFINE CLUSTER(                              -
                 NAME(SMPE.SMPCSI.CSI)         -
                 FREESPACE(10 5)               -
                 KEYS(24 0)                    -
                 RECORDSIZE(24 143)            -
                 SHAREOPTIONS(2 3)             -
         VOLUMES(volid1)                 -
                )                              -
             DATA(                             -
                 NAME(SMPE.SMPCSI.CSI.DATA)    -
                 CONTROLINTERVALSIZE(4096)     -
                 CYLINDERS(250 20)             -
                )                              -
            INDEX(                             -
                 NAME(SMPE.SMPCSI.CSI.INDEX)   -
                 CYLINDERS(5 3)                -
                )                              -
          CATALOG(user.catalog)
/*
```

523

图17-19 CSI VSAM集群定义的例子

```
//SMPJOB   JOB 'accounting info',MSGLEVEL=(1,1)
//SMPSTEP  EXEC SMPPROC
//SMPPTFIN DD  ... points to the file or data set that contains
//*             the SYSMODs to be received
//SMPHOLD  DD  ... points to the file or data set that contains
//*             the HOLDDATA to be received
//SMPTLIB  DD  UNIT=3380,VOL=SER=TLIB01
//SMPCNTL  DD  *
  SET      BDY(GLOBAL)        /* Set to global zone       */.
  RECEIVE  SYSMOD             /* receive SYSMODS and      */
           HOLDDATA           /* HOLDDATA                 */
           SOURCEID(MYPTFS)   /* Assign a source ID       */
                              /*                          */.
  LIST     MCS                /* List the cover letters   */
           SOURCEID(MYPTFS)   /* for the SYSMODs          */
                              /*                          */.
  SET      BDY(TARGET1)       /* Set to target zone       */.
  APPLY    SOURCEID(MYPTFS)   /* Apply the SYSMODs        */
                              /*                          */.
  LIST     LOG                /* List the target zone log */.
/*
```

图17-20 SMP批处理作业例子

17.11.4 其他 SMP/E 工具

放置在全局区(522页的图17-18)中的所有信息，和目标区以及分配区中的信息三者一起组成了SMP/E安装和追踪系统软件(通常是大量数据)所需的数据。您可以使用以下SMP/E工具显示这些信息：

- LIST命令创建系统相关信息的硬拷贝。
- REPORT命令检查，比较和生成区内容信息的硬拷贝。
- ISPF的QUERY对话框。
- SMP/E CSI API，可以用来编写应用程序来查询系统内容。

17.12 SMP/E 使用的数据集

我们回顾一下之前关于SMP/E如何存储系统相关信息的讨论。当SMP/E处理SYSMOD时，它在合适的库中安装元件，并更新它自己已经完成的操作的相关记录。SMP/E在两种库中安装程序元件：

- 目标库包含运行系统需要的可执行代码(比如，用来运行您的生产系统或测试系统中的库)。
- 分配(程序)库(DLIB)包含一个系统的每个元件的重要拷贝。它们被用来作为SMP/E GENERATE命令或系统生成流程的输入，来为一个新系统建立目标库。在目标库的元件不得不被替换或更新时， SMP/E还用分配(程序)库做备份之用。

为了在这些库中安装元件，SMP/E使用由以下几种类型数据集组成的数据库：

- SMPCSI(CSI)数据集是VSAM数据集，它控制安装流程，记录处理结果。一个CSI可以由VSAM键结构分成多个分区。每个分区都是一个区(zone)。

  存在三种类型的区：

  - 单一的全局区(global zone)，用来记录接收到SYPPTS数据集中SYSMOD的相关信息。全局区还包含能让SMP/E访问其他两种类型区的信息，为了从SYSMOD中安装元件SMP/E所调用的系统实用程序的信息，用来允许您裁剪SMP/E流程的信息。
  - 一个或多个目标区(target zone)，用来记录操作系统(或目标)库相关的结构和状态信息。每个目标区也指向相关的分配区，在APPLY，RESTORE和LINK过程中，当SMP/E处理一个SYSMOD并需要检查分配(程序)库中的这些元件级别时使用。
  - 一个或多个分配区(distribution zone)，用来记录分配(程序)库(DLIB)的结构和状态信息。每个DLIB区也指向相关的目标区，在SMP/E打算接受一个SYSMOD而需要检查该SYSMOD是否已经被应用时使用。

  在SMPCSI数据集中可以有很多区(实际上，每个数据集可以有多达32766个区)。例如，一个SMPCSI数据集可以包含一个全局区，多个目标区和多个分配区。这些区也可以在不同的SMPCSI数据集中。一个SMPCSI数据集可以只包含全局区，第二个SMPCSI数据集包含目标区，第三个SMPCSI数据集包含分配区。

- SMPPTS(PTS)数据集临时存储一些等待安装的SYSMOD。PTS作为SYSMOD的存储数据集被严格使用。RECEIVE命令直接存储SYSMOD至PTS，无需更改任何SMP/E信息。PTS和全局区有关，因为这两个数据集都包含接收的SYSMOD的信息。对于某个给定的全局区，只能使用一个PTS。因此，您可以将PTS和全局区看作一对必须并行处理的数据集(比如，删除、保存或修改)。
- SMPSCDS(SCDS)数据集包含APPLY处理过程中更改的目标区条目的备份拷贝。因此，每个SCDS直接与某个特定的目标区相关联，每个目标区也必须有自己的SCDS。SMP/E使用SCDS数据集存储在APPLY处理过程中修改的目标区条目的备份拷贝。因此，每个SCDS直接与特定的目标区相关联，每个目标区也必须有自己的SCDS。

SMP/E也使用以下数据集：

- SMPMTS(MTS)数据集是在安装过程中，当没有别的目标宏库被指定时，SMP/E存储宏备份的库。因此MTS和某个目标区相关联，每个目标区都必须有它自己的MTS数据集。
- SMPSTS(STS)数据集是在安装过程中，当没有别的目标源码库被指定时，SMP/E存储源码备份的库。。因此STS和某个目标区相关联，每个目标区都必须有它自己的STS数据集。
- SMPLTS(LTS)数据集是SMP/E维护一个装入模块基本版本的库。该库中的装入模块为了可以隐式包含模块而指定SYSLIB分配。因此LTS和某个目标区相关联，每个目标区都必须有它自己的LTS数据集。
- 其他实用程序和工作数据集。

SMP/E使用CSI数据集中的信息来为安装选择合适的元件级别，决定哪些库应该包含哪些元件，以及识别安装应该调用哪些系统实用程序。

系统程序员也可以使用CSI数据集来获得系统结构、内容和状态的最新信息。SMP/E以报告，列表和对话框的形式提供这些信息来帮助您：

- 研究功能和服务级别
- 理解SYSMOD(无论已经安装或等待安装)的交集和相互之间的关系
- 为SMP/E处理构建作业流

526

17.13 总结

作为一名z/OS系统程序员，确保所有软件产品和它们的修正程序都在系统上适当安装将是您的职责。在z/OS上，管理系统软件变化的主要方法是SMP/E。

SMP/E可以通过批处理作业或交互系统生产工具/程序开发工具(ISPF/PDF)下的对话框运行。通过SMP/E对话框，您可以交互式地查询SMP/E数据库，创建并提交作业来处理SMP/E命令。

SMP/E安装的软件必须打包成一个系统修改或SYSMOD，SYSMOD包含更新的元件及其控制信息。这些控制信息描述了元件本身和该软件和安装在同一系统上的其他产品或服务之间的关系。

大公司的z/OS系统程序员经常使用SMP/E JCL和命令，然而，系统程序员很少编码SMP/E MCS指令。产品和SYSMOD包会包含必要的MCS语句。

系统程序员一个重大责任就是在一个问题出现在z/OS或IBM可选产品时，与IBM缺陷支持人员一起工作。问题的解决需要系统程序员在公司系统上接收和应用补丁。

本章关键术语		
授权程序分析报告(authorized program analysis report，APAR)	统一服务目录(consolidated service inventory，CSI)	分配(程序)库(distribution library，DLIB)
分配区(distribution zone)	全局区(global zone)	程序临时补丁(program temporary fix，PTF)
SYSMOD	目标库(target library)	目标区(target zone)

17.14 复习题

为了帮助测试您对本章内容的理解，回答以下问题：

1. z/OS系统上，SMP/E的服务目的是什么？
2. 两种类型的Function SYSMOD分别是什么？它们之间有什么区别？

527

3. 在CSI数据集中储存了怎样的信息？
4. APAR和PTF的区别有哪些？
5. 填空：在SMP/E中，您使用_____命令安装所选的SYSMOD至它们相关

的分配(程序)库中。

17.15 思考题

528 在大型企业系统中，使用有序的变化管理流程的重要之处在何处？z/OS环境中，安装和维护如何提供了高可用性的基础？

18

第 18 章 z/OS 上的安全

目标：在z/OS上操作时，需要理解安全的重要性以及z/OS用来实现安全所使用的工具。系统最有价值的资源包括它的数据和应用程序。必须保护它们，防止未授权的内部访问(职员)和外部访问(顾客，商业伙伴和黑客)。

完成本章后，您将能够：

- 解释安全性和完整性的概念。
- 解释RACF以及它和操作系统的接口。
- 给程序授权。
- 讨论完整性的概念。
- 说明变化控制的重要性。
- 解释风险评估的概念。

18.1 为什么要考虑安全问题？

随着时间发展，创建和访问计算机信息变得越来越容易。系统访问不再局限于一些具有高技能的专业程序员。任何人只要稍微花点时间熟悉下比较新的，容易使用的高级查询语言，就可以创建和访问信息。

越来越多的人们开始越来越多地依赖于计算机系统和他们存储在系统中的信息。随着通用计算机文化的提升，使用计算机的人越来越多，对数据安全性的需求也被提到了一个全新的高度。系统也不能再根据没有人知道如何访问数据来简单地保证数据安全性了。

进一步说，保证数据安全并不仅仅意味着机密信息对那些不该看到的人不可访问。它还意味着防止那些有些甚至都不知道自己正在进行不适当操作数据的人由于疏忽，而对文件造成的的破坏。

当一个操作系统在设计，实施和维护时能够防止未授权的访问，并能够达到为该系统指定的安全控制不被折衷的程度，就认为它是具有系统完整性的。特别地，对z/OS来说，这意味着任何未授权的程序，使用任何定义过的或未定义过的系统接口都不可能完成以下这些事情：

- 在授权状态下得到控制权
- 跳过存储或访问保护
- 跳过密码检查

18.2 z/OS 上的安全工具

在接下来的部分，我们会谈到z/OS上提供高级安全性和完整性的工具。

客户数据是一个可以卖给竞争对手的具有价值的资源。因此任何安全策略的目标就是为用户仅仅提供他们需要的访问级别并拒绝未授权的访问。这也是审计员更喜欢给用户或组赋予特定权限，而不是使用一个通用的访问工具。主机安全的传统焦点是在于阻止未授权的人登录到系统，然后确保只有在需要知道的时候才允许用户访问数据。然而，随着主机渐渐成为因特网服务器，提出了额外的安全要求。对于来自外部的威胁，如黑客，病毒和木马程序；安全服务器包括了处理这些威胁的工具。

然而公司内部的主要威胁则是来自内部。公司内部的职员比外部人员有更多的机会获取公司的数据。一个深思熟虑的安全策略通常是第一道防卫线。

更进一步，z/OS提供了一些完整特性来将来自其他程序的蓄意的或无意的损害降到最低。许多系统都运行好几个z/OS副本，通常不允许普通的TSO/ISPF用户来访问生产系统。如果z/OS安全控制配置得好，它可以保护生产环境，防止TSO/ISPF用户(不管是故意的或偶然的)影响重要的生产作业。

18.3 安全角色

在过去，是由系统程序员来管理，决定全面的安全策略和规程。今天各大公司正在寻求更高级别的安全性，所以他们任命了独立的安全经理。系统程序员对安全性也许没有直接责任，但仍有责任就新产品向安全经理提出建议。职责分离对防止任何个人不受控制地访问系统还是很有必要的。

系统管理员分配用户ID和初始密码，并确保密码是非试验的，随机的，并经常更换的。因为用户ID和密码是非常重要的，必须特别注意保护包含了这些信息的文件。

18.4 IBM 安全服务器

许多系统采用一个套装软件，称为IBM安全服务器，通常它被称为RACF，这是它最著名的组件。

z/OS安全防备包括了：

- 控制用户(用户ID和密码)对系统的访问
- 限制授权用户在系统数据文件和程序上可以执行的功能

希望了解更多z/OS上安全管理员可用工具的学生，可以参看下面这个列表，列出了一些z/OS安全功能部件，合起来称为安全服务器：

- DCE安全服务器

  运行在z/OS上，提供具有完整功能的OSF DCE 1.1级别的安全服务器。

- 轻量级目录访问协议(LDAP)服务器

  这个服务器基于客户/服务端模式，允许客户访问LDAP服务器。LDAP目录提供了一种在中央位置维护目录信息的简单方法，包括存储，更新，检索和交换。

- z/OS防火墙技术

  这是z/OS中使用的IPV4网络安全防火墙程序。在本质上，z/OS防火墙既有传统的防火墙的功能，也支持虚拟专用网(VPN)。

  内置防火墙意味着如果需要的话主机可以直接与互联网连接，而不需要硬件介入；并且可以提供需要的安全级别来保护公司的重要资源。使用VPN技术，可以通过互联网建立由客户端到主机的安全加密信道。

- z/OS的网络认证服务

  不需要购买或使用象分布式计算环境(DCE)这样的中间件就可以提供基于Kerberos安全认证服务。

- 企业身份映射(EIM)

  提供了一种低成本地解决方案轻松管理企业中多用户注册和多用户身份的新

方法。

- PKI服务

  允许建立公钥基础设施，为内部或外部用户提供证书授权服务，根据公司的政策来发布和管理数字证书。

- 资源访问控制系统(RACF)

  这是z/OS安全服务最重要的组成部分，在z/OS中用来保护重要的资源。

安全的话题可以单独成为一门课程。在本书中，我们向您介绍RACF组件，以及如何应用它的特性来实现z/OS的安全。

532

18.4.1 RACF

计算机环境下的"访问"意味着对计算机资源的操作能力(如使用，修改或查看资源)，访问控制即是明确允许或限制这种能力的方式。基于计算机的访问控制称为逻辑访问控制。这些保护机制限制用户只能访问对他们而言合适的信息。

逻辑访问控制通常嵌入操作系统，或是作为应用程序逻辑或大的实用程序逻辑的一部分，如数据库管理系统。也可以作为附加安全套装软件安装到操作系统上，这种套装软件可以被各种系统使用，如各种PC机或主机等。另外，也可以在控制计算机和网络通信的专用部件中引入逻辑访问控制。

资源访问控制工具(RACF)是一个附加软件产品，用来向主机系统提供基本的安全。也有其他可用的安全软件包，例如ACF2或Top Secret，两者都由Computer Associates公司提供。

RACF通过仅允许授权用户访问相应的资源来保护系统。在RACF的数据库中以特殊结构形式保存了用户，资源和访问授权等信息，这些特殊结构称为profile，当需要决定允许哪些用户访问受保护的系统资源时，可以查阅这些profile。

为了达成目标，RACF提供了这些功能：

- 识别和鉴定用户
- 授权用户访问受保护的资源
- 对未授权的用户试图访问受保护资源的情况进行日志记录及生成报告
- 控制访问资源的方式
- 允许应用程序使用RACF宏

534页上的图18-1显示了RACF功能的一个简单视图。

RACF使用用户ID和系统加密的密码来完成用户的识别和鉴定。用户ID指示人员为系统的RACF用户。密码来验证用户的身份。通常出口程序(exit)用来加强密码策略，例如规定密码使用的最小长度，不允许使用重复的字符或相接近的键盘字母，并且一定要同时使用数字和密码。一般禁止使用常见的单词如"password"或用户ID来设定密码。

533

另外一个重要的策略是更改密码的频率。如果一个用户ID很长时间没有使用应该

将其收回，下次再用该用户ID登录时要进行一些特殊的处理。当一个人离开公司时，应该使用一个专门的程序来保证从系统中删除相应的用户ID。

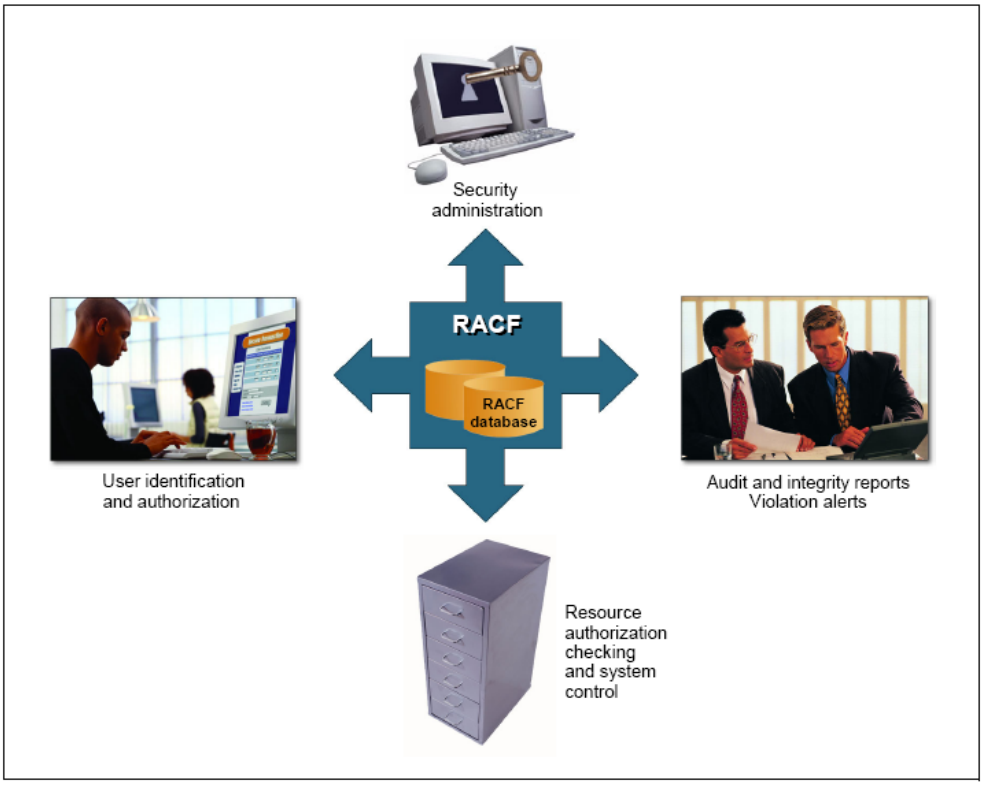


图18-1 RACF功能概览

18.4.2 系统授权工具(SAF)

SAF是z/OS操作系统的一部分，提供调用接口，来执行身份认证，授权和登录服务。

SAF不需要其他先决产品，但是如果与RACF一起协同使用，将会使整个系统的安全性得到极大的加强。SAF的关键部件是SAF路由器。即使RACF不存在，SAF路由器也总是存在的。

534

SAF路由器关注所有提供资源管理的产品，这促使产品和系统间使用公共的控制功能。资源管理组件和子系统在它们的处理过程中调用z/OS路由器作为某决策功能的一部分，如访问控制检验和授权相关的检验。这些功能叫做控制点。

当从资源管理器收到一个请求时，SAF有条件地把控制转交给RACF(如果有RACF)或者用户提供的处理例程，或者转交给两者。

18.5 安全管理

数据安全是保护数据不被无意的或有意的未授权的读取，更改或破坏。根据这个定义，可以看出所有的数据处理系统都会有潜在的安全或控制问题。根据过去的经验用户发现数据安全措施对管理任务和终端用户的请求操作都会产生重要影响。

RACF给具有SPECIAL属性的用户(安全管理员)很多系统级别和组级别的职责。在系统中，安全管理员关注于规划安全，需要：

- 确定使用哪一个RACF功能函数
- 确定RACF保护级别
- 确定RACF要保护的数据
- 确定管理结构和用户

18.5.1 RACF 远程共享工具(RRSF)

RACF远程共享工具(RRSF)用来管理和维护整个企业中分布的RACF数据库。它改进了系统性能，系统管理，系统实用性和可用性。RRSF在整个系统中或网络失败与延迟的情况下确保数据的完整性。它让管理员知道何时发生了关键事件并根据需要返回输出。

18.5.2 RACF 和中间件

主要的子系统如CICS和DB2都使用RACF工具保护交易和文件。为这些子系统配置RACF profile的大多数工作由CICS和DB2 系统程序员完成，因此这些系统程序员必须很好地理解RACF，理解所管理的软件与RACF的联系。

18.6 操作员控台的安全

通过讨论操作员控台这个例子来了解z/OS安全如何影响系统功能。控台安全是指控制操作员在他们的控台上可以输入什么命令来监视并管理z/OS。

如何为您的控台定义命令授权，或者如何控制操作员登录，这些允许您为z/OS系统或系统综合体规划操作安全。在系统综合体中，由于某个系统的操作员输入的命令能影响另一个系统中的处理过程，因此会导致所需的安全措施变得更加复杂，您需要进行相应规划。

对于多控制台支持(MCS)控台，您可以使用以下方法来控制程序员是否可以从一个控台键入命令：

- 通过CONSOLxx的CONSOLE声明中的AUTH关键字
- 通过DEFAULT声明和RACF命令及profile的LOGON关键字

对扩展的MCS控制台，您可以在控制台会话间控制授权的SDSF或TSO/E用户的行为。因为扩展的MCS控制台可以和一名TSO/E用户ID相关联，而不是物理的控制台，所以您也许想使用RACF不仅限制用户可以输入的z/OS命令，也限制用户从哪个TSO/E终端可以输入这些命令。

18.7 完整性

z/OS有很多专门设计的特性和工具，用来保护一个程序不被其他程序有意或无意影响。这也是z/OS具有程序完整性和安全性的原因。

这一节中我们将讨论：

- 授权程序工具(APF)
- 存储保护
- 跨内存通信

18.7.1 授权程序

z/OS有一个叫做授权程序工具(APF)的特征允许所选的程序访问敏感系统功能。设计APF是为了避免破坏完整性。安装系统时指定哪些库包含这些特定功能或程序，这些库称为APF库。

具有APF授权的程序几乎可以做它想做的任何事情。从本质上说它是操作系统的扩展。它可以把自己置于超级用户状态或系统键。它可以修改系统控制块。可以执行特权指令(在超级用户状态中)。它可以关闭日志来掩盖踪迹。当然，这种授权必须很谨慎地授予并需要仔细监控。

您可以使用APF来指定可以使用敏感系统功能的系统或用户程序，如，APF：

- 将敏感系统管理程序调入(SVC)程序(和敏感用户SVC程序，如果需要的话)的使用局限于APF授权程序。
- 允许系统在一个授权的作业步任务中，只能从授权库中获取所有模块，以防止程序在授权作业步任务模块流中伪造模块。

很多系统功能如管理程序调入(SVC)或通过SVC的特殊路径是敏感的，必须限制只有授权程序才能访问这些功能，以免危及系统安全性和完整性。

当正在运行的程序具有如下特征时，系统为该任务授权：

- 运行在超级用户状态下(程序状态字(PSW)的第15位是0)。在75页第3章”z/OS概述”中讨论了PSW。
- PSW关键字为0到7下运行(PSW的第8到11位包含0到7中的一个值)。
- 同一作业之前运行的所有程序都是APF程序。

包含授权程序的库称为授权库。APF授权库必须位于以下授权库之一中：

- SYS1.LINKLIB
- SYS1.SVCLIB
- SYS1.LPALIB
- 由系统所指定的授权库

537

18.7.2 存储保护

主机硬件有存储保护的功能。这通常用来防止未授权地改变存储内容。它也可以用来防止对存储区的未授权读取，尽管z/OS只是使用这种方法保护很小的区域。存储保护是基于4K页面的。存储保护只针对实存，不处理虚存的情况。当从磁盘上拷贝虚存中的一个页面到主存时，z/OS会在主存的页面中设置合适的存储保护关键字。

多地址空间出现以前，存储保护是很重要的。当在单一地址空间(或在虚存出现以前的实存)下存在多名用户和多个任务时，保护用户的存储不被破坏(或不适当的数据偷窥)是至关重要的。在z/OS里，保护用户存储的主要措施是通过多地址空间隔离各个用户的存储。

应用程序不可以更改存储保护关键字。对于常规应用程序(即非授权程序)，无法使用存储保护功能，来保护它的虚存部分，不受同一个地址空间里的其他应用程序部分的影响。

附加的存储保护位(实存的4K页)叫做页保护位。这样甚至可以防止系统程序(以关键字0运行，通常可以存储在任何地方)写入这个页面。在LPA页面中使用这个”位”来防止系统程序对它的意外损害。

18.7.3 跨内存通信

使用正确的操作系统页表管理，可以使得不同地址空间里的用户和应用程序相互完全隔离。这种隔离的一个例外是公共区域。另一个例外是跨内存通信。

通过操作系统正确的配置，有可能使得一个地址空间里的程序可以和另一个地址空间中的程序通信。可能会存在一些跨内存能力，但通常使用的是下面的两种：

- 调用处于不同地址空间的程序的能力
- 访问(获取，存储)另一个地址空间中的虚存的能力

538

第一种情况使用的是程序调用(PC)指令。一旦z/OS完成了正确的配置，只需要一条硬件指令就可以调用另一个地址空间中的程序。常见的例子就是DB2，它是主要的IBM数据库产品。DB2的不同组成部分占用了四个地址空间。DB2的用户可能是TSO用户，批处理作业和其他中间件(例如网络服务器)。当这些用户发出DB2 SQL指令，应用程序中的SQL接口使用程序调用来从DB2地址空间中获得服务。

交叉存储设计可能更复杂，而且必须与z/OS的安全控制协调一致。事实上几乎所

有跨内存通信的使用都在主要的中间件产品中，典型的应用程序很少直接使用。

常规应用编程很少在这个领域冒险。主机硬件体系结构和z/OS内在设计都在避免非正确地使用这些功能，并且与跨内存能力相关的安全性和完整性还没有得到足够的重视。

18.7.4 z/OS 防火墙技术

传统的防火墙功能是作为内部网(一种安全的，内部私有网络)和另一种网络(不安全)或因特网之间的封锁。防火墙的目的是为了防止发生不希望的或未授权的安全网络对内或对外的通信。防火墙有两个任务：

- 允许自己网络中的用户在不危害本网络中数据和其他资源的情况下使用外网中的授权资源。
- 防止外网中的用户进入您的网络，进行危害或攻击。

防火墙可以通过几种方法保护网络。防火墙可以提供屏蔽服务，根据象用户名，主机名和TCP/IP这些来拒绝或授权访问。防火墙还可以提供大量服务，让授权用户通过，阻止未授权用户，同时保证因特网和您网络之间的所有通信看起来就在防火墙处终止，拒绝外部世界看到您的网路结构。

为了控制您的企业内部网和因特网之间的互相访问，同时允许授权交易，z/OS防火墙技术提供了三个关键技术：网络服务器，过滤器和地址转换，还有虚拟专用网。

18.8 总结

数据安全并不仅仅意味着让不应该查看的人无法访问机密信息；还意味着要注意用户无意中的误操作可能导致的数据破坏。在实践中如果不重视数据的安全问题，随着技术的发展，就很有可能让未授权用户有意或无意的访问，修改或破坏数据。z/OS上安全服务器由一系列的特征构成，提供安全实施。

系统授权程序(SAF)是z/OS操作系统的一部分，它提供了对可调用服务的接口，来执行身份识别，授权和登录等服务。

资源访问控制工具(RACF)是z/OS安全服务器的一部分，它负责控制对所有受保护z/OS资源的访问。RACF通过仅允许授权用户访问相应的受保护资源来进行相关的保护工作，并且将与用户，资源和访问权限相关的信息存储在特定的profile中。

RACF提供工具和数据库，允许z/OS的特许程序，如TSO，CICS和DB2，检查和核实一个用户的访问级别，以此决定允许或者拒绝用户对数据集，交易或者数据库视图的使用。

RACF使机构能定义使用RACF保护的个体和组。例如，对机构里的一个秘书来说，安全管理员使用RACF来定义一个用户profile，里面定义了秘书的用户ID，初始密

码和其他信息。

为了完成这个目标，RACF使您能够：

- 识别和鉴定用户
- 授权用户访问受保护的资源
- 对未授权的用户试图访问受保护资源的情况进行日志记录和生成报告
- 控制访问资源的方式

z/OS系统操作包括了以下方面：

- 控制台操作，即操作员如何和z/OS交互以监视和控制硬件和软件
- 信息处理和命令处理，构成了操作员和z/OS系统之间的交互基础和z/OS自动化的基础

操作z/OS包括管理硬件例如处理器和外围设备，和管理软件如z/OS操作控制系统，作业输入子系统 和运行在z/OS上的所有应用程序。

540

当执行控制台安全时，系统可以控制操作员在控制台输入哪些命令来监控和管理z/OS。当然首先要在RACF和PARMLIB的CONSOLxx子句中客户化配置。

本章关键术语		
授权库(authorized libraries)	授权程序工具(APF)	加密(encryption)
防火墙(firewall)	黑客(hacker)	页保护位(page protection)
密码(password)	资源访问控制工具(RACF)	安全策略(security policy)
职责分离(separation of duties)	系统完整性(system integrity)	用户ID(user ID)

18.9 复习题

为了帮您更好得理解本章内容，完成这些问题：

1. 下面的语句对还是错？

   资源profile里的访问信息只能在组级别上设定。这意味着如果某用户连接的组对某数据集仅有只读属性的话，该用户对该数据集就不可能具有更新属性。

2. 在下面的情况中，如果没有调用授权的SVC或特殊功能，程序将会发生什么？
   a. 一个程序连接-编辑返回码AC=0
   b. 从APF授权库中运行

3. 在下面的例子中，运行MPRES2卷上的SYS1.LINKLIB库的程序可能会出现哪些问题？

```
D PROG,APF,ENTRY=1
CSV450I 05.24.55 PROG,APF DISPLAY 979
FORMAT=DYNAMIC
ENTRY VOLUME DSNAME
      1   MPRES1 SYS1.LINKLIB?
```

18.10 思考题

1. 在其他平台上，是如何保护数据集或文件的？有没有方法阻止某个特定应用程序的运行？

2. RACF能够把组管理的权限授予用户。这样就有可能实现分散管理。讨论这样做的利弊。

18.11 练习题

1. 把登录过程IKJACCNT改为IKJACCN1后，尝试登录TSO。应该会看到如下信息：

```
IKJ56483I THE PROCEDURE NAME IKJACCN1 HAS NOT BEEN
AUTHORIZED FOR THIS USERID
```

2. 用您的TSO用户ID(缺省登录过程为IKJACCNT)尝试删除数据集ZPROF.JCL.NOT.DELETE，这是由提供的JCL标准作业建立的数据集。这是个受保护的数据集，您只能读取它的内容。
3. 执行下面这个样例JCL，得到DSMON实用程序报告，该报告中有当前RACF组的树形结构(可用的JCL样例在成员DSMON中)。

```
//DSMONRPT JOB
(POK,999),'DSMONREPORT',MSGLEVEL=(1,1),MSGCLASS=X,
// CLASS=A,NOTIFY=&SYSUID
/*JOBPARM SYSAFF=*
//*
//* NOTE:
//* REMEMBER THAT ICHDSM00 MUST BE RUN BY A USER WITH
AUDITOR ATTRIBUTE
//*
//STEPNAME EXEC PGM=ICHDSM00
//SYSPRINT DD SYSOUT=A
//SYSUT2 DD SYSOUT=A
//SYSIN DD *
FUNCTION RACGRP
/*
```

4. 确认SYS1.LINKLIB库是一个APF授权库。
   - 使用‘DISPLAY APF’命令来显示整个APF列表。
   - 在‘DISPLAY APF’命令中使用‘ENTRY=操作数’。
   - 在‘DISPLAY APF’命令中使用‘DSNAME=操作数’。确认命令中的整个数字都在系统日志中显示出来。
5. 下面的JCL样例可以用来调用ADRDSSU实用程序，并向控制台发送一条WTOR消息。WTOR命令能够写一条ADR112A消息到系统控制台中。ADR112A消息要求操作员执行某种操作然后发送一个回复，比如，可以用WTOR来请求操作员在您的DFSMS作业继续处理前，安装一个请求的卷或终止一个数据库(可用的JCL样例在成员ADRDSSU中)。


```
//WTORTEST JOB (POK,999),'USER',MSGLEVEL=(1,1),MSGCLASS=X,
```

```
// CLASS=A,NOTIFY=&SYSUID
// EXEC PGM=ADRDSSU
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
  WTOR 'TEST'
/*
```

DFSMSdss会把下面的路由代码写到WTOR消息中：

1 Primary operator action

DFSMSdss会把下面的描述符代码写到WTOR消息中：

2 Immediate action required.

在SDSF主屏幕上，选择SR选项(系统请求)，给出任何您想给的回复。 543

19

第 19 章 z/OS 上的网络通信

目标：在使用z/OS网络通信的过程中，您需要与TCP/IP和SNA网络交互。

本章中，您将学习：

- 如何比较多种通信网络模型。
- z/OS通信服务器产品的软件组件
- 如何比较SNA分区和APPN网络拓扑
- 如何使用IP网络来在SNA应用程序之间传输数据。
- 常见的TCP/IP和VTAM命令。

19.1 z/OS 通信

网络通信包含软件和硬件两个方面，在大型企业中，软件和硬件通信职责通常被分开。即便是网络技术专家也需要理解这两个方面。本章我们简单概述主机上的通信软件。

作为一名系统管理员，网络专家必须把对z/OS通信软件的透彻理解带到每一个和公司网络相关的项目中去。然而网络硬件技术员拥有特定的技能和工具来支持物理网络，他们通常无需掌握z/OS通信软件。当一个全国零售连锁商开张一家新店时，z/OS系统程序员和网络硬件技术员必须协同工作来完成新店的开张。

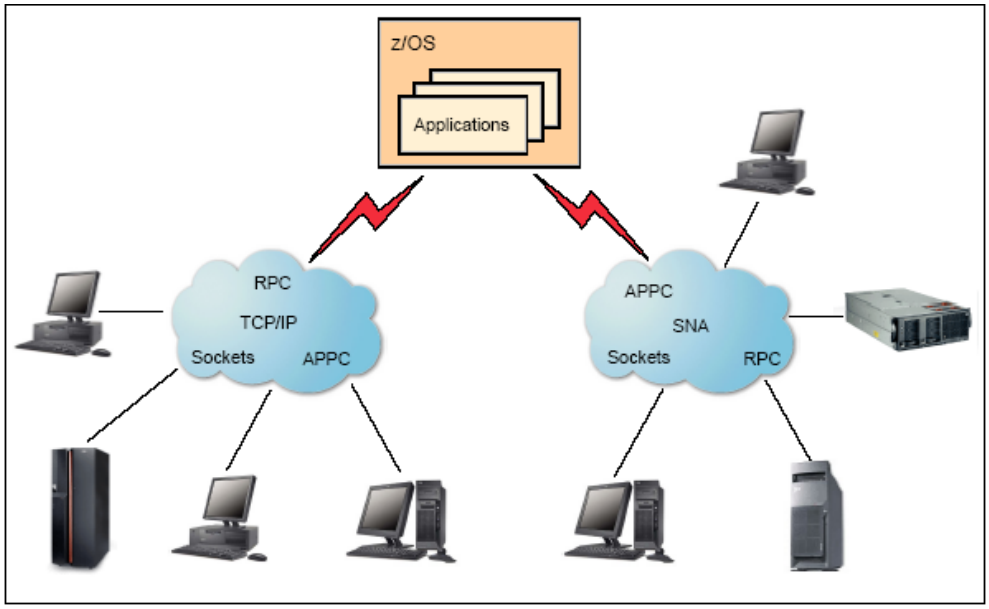


图19-1 IBM通信服务器

z/OS包含支持多协议网络的功能完备的通信服务器。本章从一个z/OS上可用网络技术概述开始，然后在550页的19.3章节"z/OS通信服务器"中讨论操作系统的通信服务器操作的主要方面。

546

19.2 数据网络简史

TCP/IP建立于1969年，比系统网络体系结构(SNA)早5年。然而，SNA被立即被公众使用，而TCP/IP一开始限制给军方和研究机构使用在互连网络中，这构成了因特网的先驱。

除此之外，通过同步数据链路控制(SDLC)协议，SNA包含了网络管理控制，这起初在TCP/IP中不包含。在20世纪80年代，SNA被大公司广泛应用，这是由于SNA允许公司的IT机构将中央计算能力扩展到全球范围，同时保证了合理的响应时间

和可靠性。比如，大范围使用SNA允许零售产业在销售点将新的公司信用卡账号提供给客户。

1983年，TCP/IP在伯克利BSD UNIX中进入公共领域。TCP/IP完备性，应用程序和接受度通过一个开放标准委员会，互联网工程任务组(IETF)，使用请求注解(RFC)机制，而取得领先地位。

术语互联网(internet)是TCP/IP网络的通用术语，不要把它和因特网(Internet)混淆起来，因特网包括了大型国际骨干网络，将所有连接因特网骨干的TCP/IP主机连接起来。

TCP/IP为互相连接的网络(一个互联网)而设计，看似容易建立；与此相对，SNA设计为层次结构，中央的主机位于层次结构的顶端。SNA设计包含网络管理，数据流控制和为特定数据工作负载分配"服务类别"优先级的能力。

1983年，自治SNA网络间通信面世。在那之前，无法容易地进行SNA网络之间的通信。独立的SNA网络共享商业应用程序和网络资源的能力叫做SNA网络互联(SNI)。

19.2.1 z/OS 上的 SNA 和 TCP/IP

IBM开发了系统网络体系结构(SNA)。SNA使得企业在全国范围各地分公司之间可以通信。为了做到这点，SNA包含了诸如虚拟远程通信存取方法(VTAM)，网络控制程序(NCP)，终端控制器和同步数据链路控制(SDLC)协议之类的产品。20世纪80年代SNA在各大公司使用的情况就好比90年代TCP/IP和因特网在公众中的使用情况。

传输控制协议/因特网协议(TCP/IP)是一个工业标准，是一套非专有的通信协议，在运行在不同类型互联网络上的应用程序之间提供可靠的端对端的连接。TCP/IP在因特网发展成熟后就被大力推广，因为它在耗费相对较小的开销的情况下允许对远程数据访问和处理。TCP/IP和因特网带来了小型计算机和通信设备的大量生产，用于聊天，电子邮件，处理商务，下载或上传数据。

拓展SNA持有的数据和程序范围到大量的小型电脑和通信设备(客户家中或小型办公室中等等)上很具商业潜力，大型SNA企业已经意识到这点。

19.2.2 分层网络模型

TCP/IP和SNA都是分层的网络模型。每个都可以间接地对应国际开放系统互联(OSI)网络模型(如图19-2)。

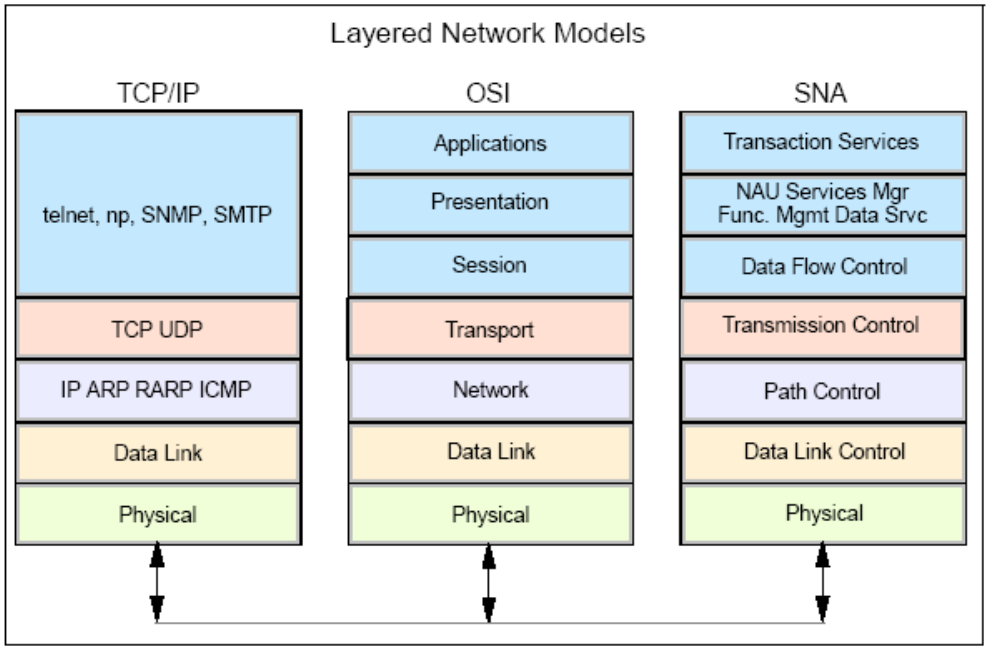


图19-2 开放系统互连(OSI)网络模型

548

OSI网络模型描述了端对端数据通信相关技术的个体元素的结构。如图19-2，OSI网络模型为TCP/IP和SNA提供一些共同点。虽然这两种技术都不能直接映射到OSI网络模型(TCP/IP和SNA在OSI网络模型正式形成之前就存在了)，但由于定义的模型层次它们之间依旧存在共同点。

OSI网络模型被分为7层。OSI第7层(应用层)间接映射到SNA和TCP/IP栈的顶层。OSI第1层(物理层)和第2层(数据链路层)映射到SNA和TCP/IP栈的底层。

典型情况下，两个物理分开的终端软件应用程序通过一个分层的网络模型连接彼此。数据由一端程序发出，由另外一端程序接收。应用程序可以安置在主机，PC，销售点(POS)设备，ATM，终端或打印机控制器上。SNA终端又叫逻辑单元(LU)，而IP终端称为应用程序端口(简称端口)。

思考一下该模型如何在大型杂货连锁店的情况下用作网络通信？每次一个客户为买杂货在某一处杂货店销售点付款，要用到2次分层网络模型：

- POS应用程序位于本地分层网络栈的顶部。
- 负责记录销售细节和授权完成的应用程序位于远程分层网络栈顶部。

本地网络栈可能运行在连接有POS设备的非主机系统上，远程网络栈经常会运行在主机上，处理接收自所有商店位置的交易。付款方式，购买的商品，商店地址和时间都被主机应用程序记录下来，并且授权打印销售单通过这两个分层网络栈返回以最终完成销售。

该交易模型通常叫做请求/服务器或客户机/服务器关系。

19.2.3 网络可靠性和可用性

假如本例中杂货连锁店的网络或连接的主机某种程度上变得不可用了呢？大部分现今使用的POS系统具备在智能商店POS控制器或小型商店处理器中累积交易的能力。之后当故障检修完毕，累积的交易可以成批地发送给主机。

在之前的例子中，恢复交易对防止商店和连锁店总部的簿记和库存问题举足轻重。未记录，或不准确的记录的累积作用可以轻易摧毁一个公司。因此，可靠性，可用性和可服务性(RAS)在网络设计中和在主机本身中一样重要。

549

19.2.4 SNA 持续使用的因素

对于全球关键商业程序，SNA是稳定，可信赖的和可靠的。在全球，大量的公司数据由z/OS常驻的SNA应用程序处理[1]。SNA一个独特的优势在于它是面向连接的，具有很多计时器和控制机制可以确保数据传输的可靠性。

尽管TCP/IP和基于网络的商务很吸引人，但主机IT公司常常对从SNA迁移表示犹豫和怀疑。而这种犹豫常是合理的。对稳定，调优完毕的商务应用程序重新编写来改变SNA程序接口到TCP/IP套接字既费时又费钱。

很多公司选择使用网络使能的技术使大量集中的数据对基于TCP/IP的网络环境可用，同时保持SNA API不变。这种”两全其美”的方法确保了SNA和VTAM在可预见的未来前景良好。

19.3 z/OS 通信服务器

z/OS包含通信服务器，它集成了一系列软件组件，使得z/OS上运行的应用程序可以进行网络通信。通信服务器提供外部网络和在z/OS上运行的商业应用程序之间的数据传输通道。

z/OS通信服务器提供一套通信协议，支持本地和广域网内(其中包括最常用的广域网络，因特网)的点对点连接功能。z/OS通信服务器也能够提升性能，很多TCP/IP程序都能从中受益；它还包含了一些常用的应用程序。

通信服务器包含一些复杂的产品和功能，主要的服务如下：

- IP，使用传输控制协议/因特网协议(TCP/IP)
- 系统网络体系结构(SNA)，使用虚拟远程通信存取方法(VTAM)

550

通信存储管理器(CSM)组件提供一个共享I/O缓冲区用于存储数据流。CSM功能允许授权的主机应用程序无需物理上移动数据就可以共享数据。

[1]运行在 z/OS 上的 SNA 应用程序也叫 VTAM 应用程序。

通信服务器集成了TCP/IP功能和SNA功能，可以实施在除z/OS之外的多种平台上，比如AIX，Microsoft®Windows和Linux。结果是z/OS程序员可以在完全不同的平台系统上使用通信的技术优势(信息访问，电子商务和协作)。

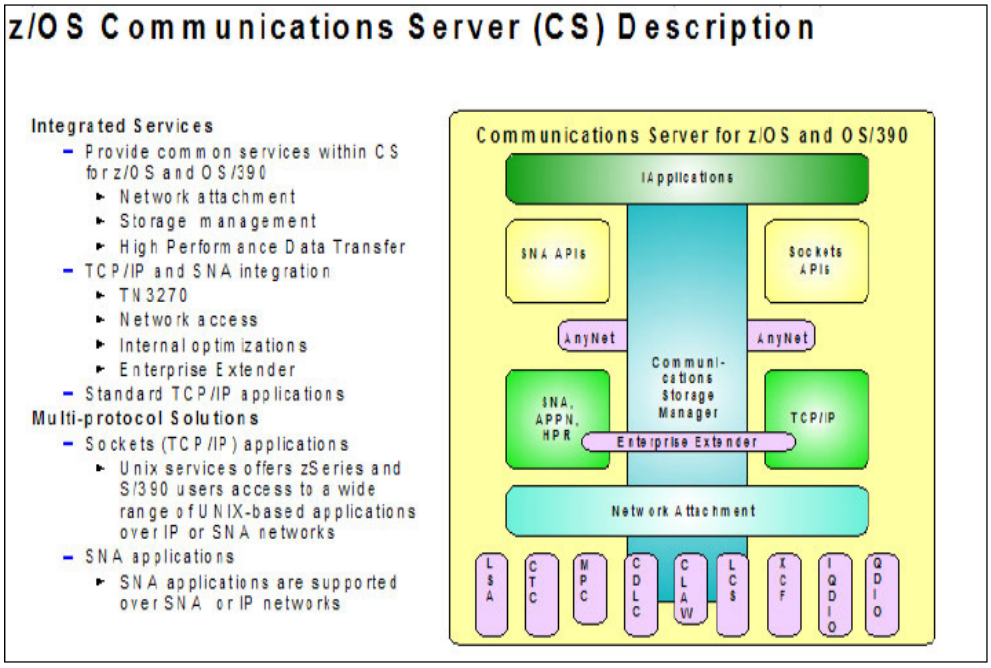


图19-3 z/OS通信服务器

19.4 TCP/IP 概述

TCP/IP是通用术语，用来描述组成因特网基础的一套协议。它第一次出现是在由加利福尼亚大学伯克利分校提供的UNIX系统中，现在基本上全球所有有网络功能的计算机都会用到它。

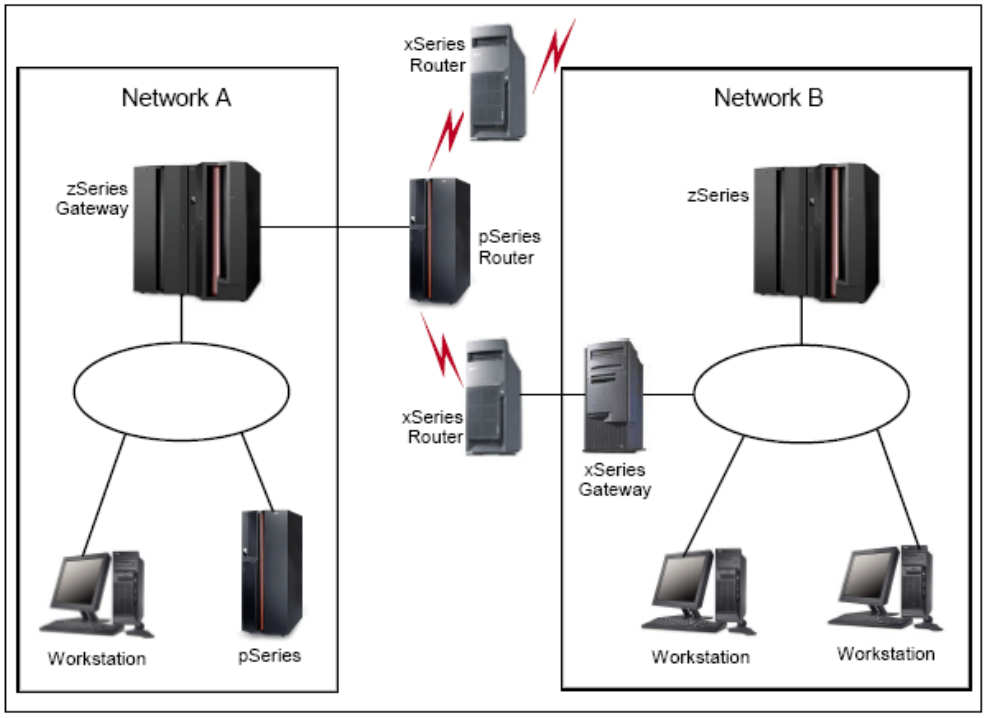


图19-4 TCP/IP介绍

在TCP/IP网络中，所有系统，无论大小，在其他系统看来都是一样的。TCP/IP可以用在本地网(LAN)硬件中，使用大多数常用协议，也可以用在广域网(WAN)中。

在TCP/IP网络环境中，运行TCP/IP的机器叫做主机(host)。一个TCP/IP网络包含了一个或多个主机，他们之间通过各种通信链路连接在一起。任何主机都可以通过直接建立通信来寻址其他所有主机。网络间的连接对于一个与主机通信的应用程序来说是透明的。系统不关心程序是一个使用MVS服务的传统的CICS应用程序，还是一个使用UNIX组件的新下载的应用程序。系统管理员可能必须选择使用哪个组件和程序接口。但是对应用程序来说，所用的组件是透明的。

19.4.1 使用命令监控 TCP/IP

z/OS支持其他操作系统中使用的TCP/IP命令，如下：

- NETSTAT
- PING
- TRACERTE
- NSLOOKUP

552

您可以从授权TSO会话输入这些命令，通过以下途径输入：

- TSO Ready 提示符
- ISPF命令shell
- 任何ISPF命令行下，在命令前添加前缀TSO

► 批处理程序

例19-1显示了您从TSO READY提示符下输入NETSTAT命令后可能看到的结果。

例19-1 NETSTAT输出样例

```
**** NETSTAT ROUTE output ****
MVS TCP/IP NETSTAT CS V1R5       TCPIP Name: TCPIP          21:38:18
Destination      Gateway          Flags     Refcnt  Interface
-----------      -------          -----     ------  ---------
Default          5.12.6.92        UGS       000001  OSA2380LNK
5.12.6.0         0.0.0.0          US        000000  OSA2380LNK
5.12.6.66        0.0.0.0          UH        000000  OSA2380LNK
5.12.6.67        0.0.0.0          UH        000000  STAVIPA1LNK
15.1.100.0       0.0.0.0          US        000000  IQDIOLNKOA016443
15.1.100.4       0.0.0.0          UHS       000000  IQDIOLNKOA016443
15.1.100.42      0.0.0.0          UHS       000000  IQDIOLNKOA016443

**** NETSTAT BYTE output ****
MVS TCP/IP NETSTAT CS V1R5       TCPIP Name: TCPIP          21:41:18
User Id  Conn     Local Socket           Foreign Socket         State
-------  ----     ------------           --------------         -----
DASU8G25 000048BF 0.0.0.0..523           0.0.0.0..0             Listen
D8G2DIST 00000072 0.0.0.0..38062         0.0.0.0..0             Listen
D8G2DIST 00000031 0.0.0.0..38060         0.0.0.0..0             Listen
FTPMVS1  00000022 5.12.6.66..21          0.0.0.0..0             Listen
FTPOE1   00000024 5.12.6.67..21          0.0.0.0..0             Listen
INETD4   00000083 0.0.0.0..7             0.0.0.0..0             Listen
INETD4   00000081 0.0.0.0..19            0.0.0.0..0             Listen
```

19.4.2 使用控制台命令管理 TCP/IP

您可以使用DISPLAY TCPIP和VARY TCPIP命令来管理z/OS上网络和TCP/IP应用程序。

这些命令包含额外选项，如下：

- DISPLAY TCPIP命令可以用来显示以下信息：
  - NETSTAT对某个TCP/IP栈的报告输出
  - 一个动态路由后台程序OMPROUTE的信息
  - TCP/IP栈存储利用率
  - TCP/IP栈系统综合体状态
  - TN3270服务器信息
- VARY TCPIP命令使用选项可以：
  - 停止指定套接字连接
  - 开始或停止通信设备
  - 建立信息包和套接字追踪
  - 清除ARP或相邻的高速缓冲存储器条目
  - 更改TCP/IP栈的系统综合体状态
  - 控制TN3270服务器

- 改变TCP/IP网络配置

相关读物：IP命令的更多信息，请参考IBM出版物”Communications Server IP System Administrators Commands”。

19.4.3 为可用性和负载平衡使用 VIPA

z/OS上的TCP/IP允许网络管理员定义虚拟IP地址或VIPA。VIPA地址由z/OS内在的低级通信协议管理。这样一个地址可以同时自动映射到多个z/OS系统。它可以从一个z/OS系统动态转移到另一个z/OS系统中。它可以动态地从一个IP应用程序转移到另一个IP应用程序，即使那个程序存在于另外一台z/OS主机中。所有这些对于网络和大多数应用程序来说都是透明的。

除此之外，VIPA地址可以与z/OS主机上的所有物理适配器相关联。z/OS主机可以支持的物理适配器的数量是没有限制的。

同时，VIPA地址可以和z/OS的工作负载管理服务一起工作，将接入的IP连接转移到拥有最多系统可用资源的z/OS主机上。

19.4.4 TN3270，通往 z/OS 的网关

在z/OS主机存在于仅支持SNA的环境的时候，专用终端控制器和”哑”终端(不可编程的)是和z/OS通信的基础。用来在哑终端和z/OS之间通信的数据协议叫做3270数据流(在之后我们更完整地讨论VTAM的时候将更多地涉及到)。随着TCP/IP逐渐成为全球标准，3270数据流改造后使其适用于TCP/IP网络。融合3270和现存的telnet标准，Telnet 3270标准问世了。今天，TN3270被进一步优化成TN3270加强版，也叫TN3270E。这个标准在RFC 2355中定义。

554

为了通过TN3270和z/OS通信，需要诸如Personal Communications的客户端在工作站上运行。客户端使用TCP建立TN3270会话。在z/OS这边，TN3270服务器将TN3270协议转换成SNA协议，然后使用VTAM连接到应用程序。SNA部分由TN3270服务器完成，它获取一个逻辑单元(LU)代表一个TN3270客户机。然后，一个LU-LU会话在TN3270服务器和目标应用程序间建立。我们将在本章之后讨论VTAM的时候更多讨论LU-LU会话。

19.5 VTAM 概述

在z/OS中，VTAM提供SNA层次网络通信栈来完成程序和终端用户间的数据传输。VTAM管理SNA定义的资源，在这些资源间建立会话，并追踪会话活动。

VTAM完成以下网络任务，比如：

- 监控和控制资源的激活和连接。
- 建立连接并管理会话的流速。

- 提供应用程序编码接口(比如，为LU 6.2编程提供APPC API)允许通过用户编写的应用程序和IBM提供的子系统访问网络。
- 为分时共享选项(TSO)提供交互终端支持。
- 为本地和远程连接的资源提供支持。

z/OS只运行一个VTAM地址空间。每个使用VTAM的应用程序，例如CICS/TS，都需要一个VTAM定义。使用这个定义，应用程序和VTAM建立和其他应用程序或终端用户的连接。

每个SNA会话终端叫做一个逻辑单元(LU)。一个LU是一个设备或程序，终端用户(应用程序，终端操作员，或输入输出机制)通过它访问SNA网络。VTAM建立的会话叫做LU对LU会话。在一个SNA网络中，比如CICS/TS就被认为是一个LU，通常和其他LU(诸如显示器，打印机，POS设备和其他远程CICS/TS区)建立很多会话。系统分配给每个LU一个唯一的网络可寻址单元(NAU)来达成通信。

2实际上，现今所有客户都运行TN3270E，但是术语TN3270可以用来描述任一协议。

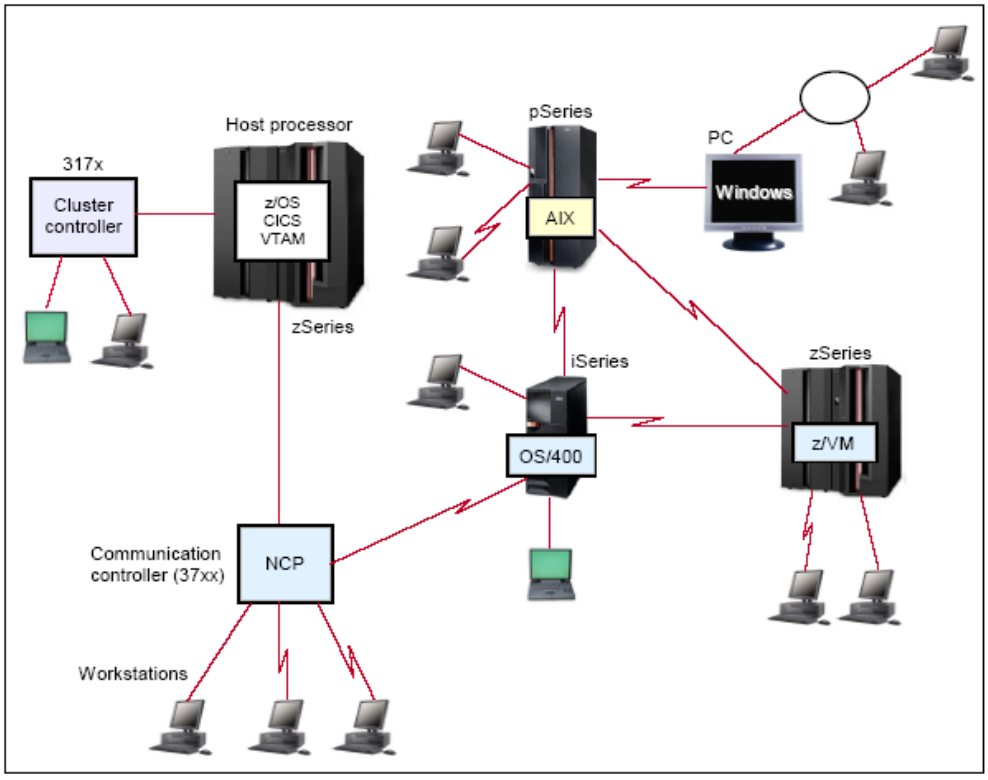


图19-5 VTAM概述——SNA环境

物理单元(PU)控制一个或多个LU。PU并不按照字义解释为网络中的物理单元。它更可能是某个设备的一部分(通常是程序编程，电路系统，或者两者兼有)，为它所处的设备完成一些控制功能；某些情况下，也完成对连接在该PU包含设备上的其他设备的控制功能。

PU存在于一个SNA网络中的每个节点，用于管理和监控该节点的资源(比如附带的链路和邻近链站)。

PU可以存在于设备中，也可以存在于一个附加的控制设备中。VTAM在激活和拥有连接在PU上的每个LU之前必须激活该PU。

主机本身也是某种类型的PU，例如CICS/TS就是它附加的一个LU 。有三种类型的PU：

556

- PU 类型 5 是主机。
- PU 类型 4 是广域网通信控制器。
- PU Type 2 是外围通信控制器。它们可以直接附加到主机上或类型4的PU上。

19.5.1 VTAM 支持的网络拓扑

虽然目前为止，TCP/IP是与z/OS主机进行网络通信最通用的方法，但一些环境仍旧使用SNA方法，并且许多环境现在在UDP/IP中运送(封装)SNA传输。SNA服务器的等级结构可以服务于大型企业需要的集中式数据处理。该等级的最顶层为VTAM。VTAM为以下种类的网络拓扑服务：

- 子区
- 高级点对点网络(APPN)
- 子区/APPN混合

VTAM中管理子区拓扑的部分叫做系统服务控制点(SSCP)。VTAM中管理APPN拓扑的部分叫做控制点(CP)。

VTAM子区网络在时间上早于APPN。在许多大型企业，从子区网络迁移到APPN拓扑是一个理想的技术目标。如果企业扩展器(EE)也添加到网络中，那将提供一些额外功能，包含用TCP/IP分组封装SNA应用程序数据。通过现存TCP/IP通信网络重定向SNA数据流，可以巩固老的SNA特定的通信设备。同时，VTAM管理工作和在硬件软件员工间的沟通协调工作将会大大减少，这也得益于一个纯APPN拓扑结构相比子区网络显得愈加灵活。

所有三种VTAM配置类型(子区，APPN和混合子区/APPN)在全球大型企业中都存在。

19.5.2 什么是子区网络拓扑？

VTAM子区网络的显著特性在于SNA资源的拥有和共享。一个子区是一些由单个PU类型5或PU类型4节点控制的SNA资源的集合。

单个VTAM和它拥有的SNA资源叫做一个域。跨域资源管理器(CDRM)允许SNA网络中的各个VTAM域之间的通信。当一个LU请求与另一个独立VTAM域中的LU建立会话时，两个VTAM相互合作来建立一个跨域会话。

557

第558页中的图19-6显示了一个纯粹的VTAM子区网络。以该图举例，它可能可以代表一家商业公司，该公司基地位于纽约，在洛杉矶有大量店铺，后来扩张到芝

加哥。图19-6包含了3个VTAM域和6个子区。芝加哥子区后来变成控制它的VTAM域的一部分。

总体来说，将老的子区拓扑全部迁移到APPN拓扑是理想的技术目标，因为既可以使用更新的IP网络设施，也可以减少由SNA网络设备产生的成本。除此之外，这样通过动态能力也能简化VTAM网络管理。

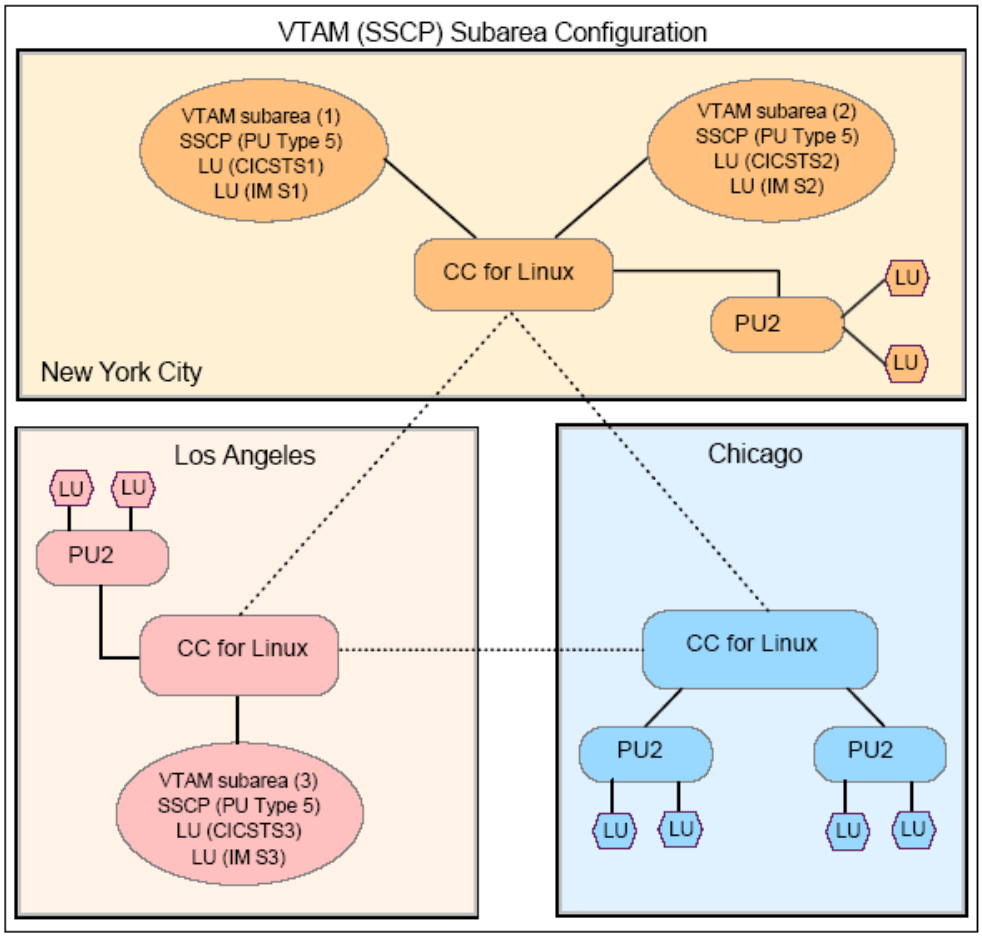


图19-6 纯粹的VTAM子区网络

19.5.3 什么是 APPN 网络拓扑

高级点对点网络®(APPN)是一种数据通信支持，它可以在一个网络的2个或更多系统间路由数据，而无需直接连接系统。

APPN拓扑没有子区号码，也不独占SNA资源。每个参与APPN的VTAM都被包含于一些地理上分散的共享SNA资源中，这样也就减少了跨域资源管理器建立会话的必要性。

APPN具有一个高性能路由(HPR)方法，即通过现存的TCP/IP网络设备传输SNA

应用程序数据。APPN包含一个叫做企业扩展器(EE)的功能，有时也叫做HPR/IP。EE确保SNA应用程序可以使用最新的IP网络技术提供的服务。

HPR并不仅仅局限于TCP/IP网络上的SNA/APPN。事实是，HPR是一项APPN功能，它能提供通过APPN网络的高性能数据分发，同时，在网络出现故障时，通过会话的动态路由实现高可用特性。但是HPR在大多数APPN连接类型中都支持(不仅仅是TCP/IP上的APPN)。企业扩展器(EE)是APPN/HPR功能，它能允许SNA会话和其他APPN功能(比如HPR)在TCP/IP网络上(而不是在SNA网络上)工作。

假设第558页上图19-6中的公司后来从子区网络拓扑迁移到APPN拓扑，第560页的图19-7显示了完成迁移工作后的同一公司的网络情况。

559

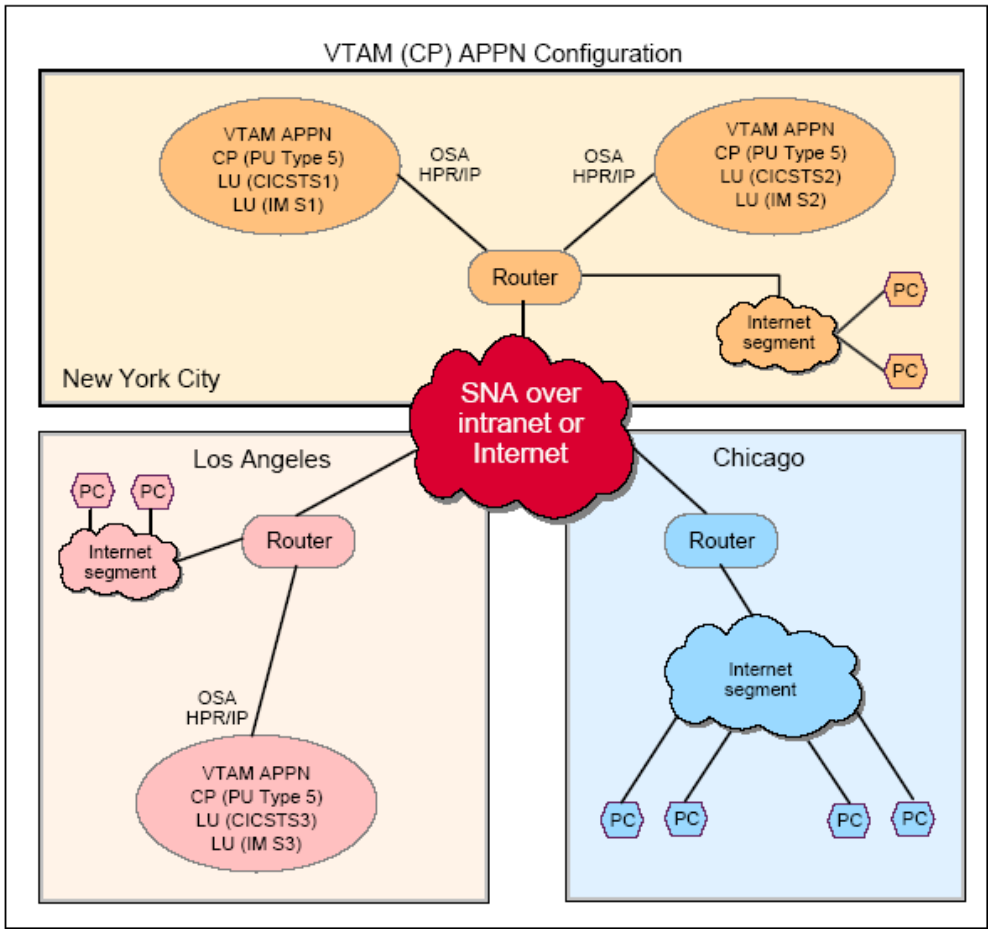


图19-7 APPN拓扑

19.5.4 VTAM 拓扑总结

VTAM可以是一个子区SSCP，一个APPN CP，或是SSCP和CP的结合，构成一个混合网络。较新的APPN拓扑能直接参与现存的IP设施中，因此它是一个理想的架构。

原先的子区SSCP VTAM将会自然而然地演化成混合的子区SSCP和APPN CP，这样就能利用EE HPR/IP功能，并减少连接网络的SNA通信设备的成本。这将极可能导致下面的决定：将剩余的子区迁移到APPN拓扑结构，以降低网络复杂度。

560

19.5.5 使用命令来监控 VTAM

下面的列表是关于VTAM命令的一个小样例，这些命令用来收集VTAM环境相关的信息。

- 使用DISPLAY(D NET,)命令列出VTAM资源的状态，比如：

D NET,VTAMOPTS 显示VTAM启动选项。

D NET,CSM[,OWNERID=ALL]

显示通信存储使用情况。

D NET,APPLS 显示定义的应用程序的状态(ACB)。

D NET,MAJNODES 显示将要列出的所有由VTAM激活的主节点的状态。

D NET,TOPO,LIST=SUMMARY

显示APPN拓扑信息。

D NET,CPCP 显示APPN CP-CP会话状态。

D NET,SESSIONS 显示子区SSCP-SSCP会话状态，

LU-LU会话(包含CP-CP会话)状态，

SSCP-LU会话和SSCP-PU会话的状态。

D NET,CDRMS 显示子区跨域资源管理器的状态。

D NET,EXIT 显示VTAM出口程序的状态。

- 使用VARY (V NET,)命令来激活VTAM资源或使VTAM资源无效。
- 使用MODIFY (F VTAM,)命令来更改VTAM环境。

z/OS系统程序员使用诸如Tivoli NetView的产品来监控和汇报VTAM资源的状态。

相关阅读：关于更多VTAM命令的信息，请参考IBM出版物”Communications Server SNA Operations”。

19.5.6 背景：3270 数据流

3270数据流对于SNA程序和LU-LU会话中的设备，就好比是HTML对于网络应用程序和浏览器。专门命令被嵌入在显屏设备和打印机的数据中。3270数据流就是带有这些嵌入指令和数据域描述符的数据。3270数据流命令由SNA应用程序创建并读入，物理单元(PU)控制器管理显示器，打印机和AIX及PC操作系统上可用的TN3270模拟器。

561

3270数据流最显著的一个优势在于：在按下回车键或PF键时，整个屏幕的数据条目和修改都发送到接收的SNA应用程序中。

3270数据流包括数据域的行列地址，以及数据描述符信息：比如颜色，保护的屏幕区和未保护的屏幕区。

当SNA应用程序发送数据到显示屏时，它包含每个数据域的行列位置布局，数据域的描述符，以及光标的屏幕位置。3270数据流允许在发送给SNA程序之前完成数据录入工作，这大大减少了CPU不必要的中断。相反，VT100 vi会话中，CPU需要注意每个敲键动作。当在VT100中输入CNTRL-G时，系统必须理解敲键动作并依此显示状态行信息。

对于SNA网络成功地集中管理成千上万个地理上分散的显示屏和打印机来说，SNA3270数据流非常重要。

19.6 总结

企业网络在设计，客户华，操作和支持时，通过使用z/OS，AIX，Windows，Linux和Linux on zSeries上的通信服务器，来采用SNA和TCP/IP网络分层的综合特性和功能。

众多大型企业使用3270和SNA应用程序，无需重写商业应用程序API。结果，VTAM在与诸如APPN，HPR和EE的技术结合后继续被支持。除此之外，TCP/IP使用VTAM进行内存管理，设备通信(所有IP设备都通过VTAM)和TN3270会话。

对于选择的SNA工作负载，企业可以使用通信服务器产品来替代一些陈旧的SNA设施组件，例如IBM 3725/45 (NCP)通信控制器硬件或者其他的连接通道的SNA控制器。

本章关键术语		
APPN	通信服务器 (communications server)	因特网(Internet)
LU对LU(LU-to-LU)	NCP	OSI
SDLC	SNA	TCP/IP
PU	VTAM	LU
栈(stack)	因特网段(Internet segment)	子区(subarea)

19.7 复习题

为了帮助您测试对本章内容的理解，完成以下问题：

1. SNA和TCP/IP网络层中，有哪些组件是共有的？
2. 多数全球企业数据是否由z/OS SNA应用程序服务？
3. 公司是否需要重新编写SNA商务应用程序，令该应用程序可以使用网络？
4. SNA子区网络和APPN拓扑之间有什么区别？

5. 为什么APPN拓扑相比SNA子区网络更理想？
6. HTML和3270数据流有什么共同之处？
7. IP地址和SNA的"网络可寻址单元"(NAU)有什么共同之处？
8. TCP/IP和VTAM共享了哪些z/OS通信服务器资源？
9. z/OS通信服务器中的什么组件为TCP/IP和VTAM提供了共享的I/O数据缓冲器？

19.8 实验和练习

1. 从SDSF中，输入TCP/IP命令/D TCPIP,,NETSTAT,HOME，从ISPF中输入TSO NETSTAT HOME。

   两个命令的输出一样吗？该z/OS系统的本地IP地址(可能多个)是什么？

2. 从SDSF输入VTAM命令/D NET,CSM。
   - 有多少空间TOTAL ALL SOURCES为INUSE？
   - 有多少空间TOTAL ALL SOURCES为FREE？
   - 有多少空间TOTAL ALL SOURCES为AVAILABLE？

563

3. 从SDSF中，输入以下VTAM命令：

   ```
   /D NET,APPLS
   /D NET,MAJNODES
   /D NET,TOPO,LIST=SUMMARY
   /D NET,CPCP
   /D NET,SESSIONS
   /D NET,SESSIONS,LIST=ALL
   /D NET,TSOUSER,ID=yourid
   ```

   写下您的IP地址___.___.___.___

   简单介绍一下这些命令的输出怎样才是有用的。

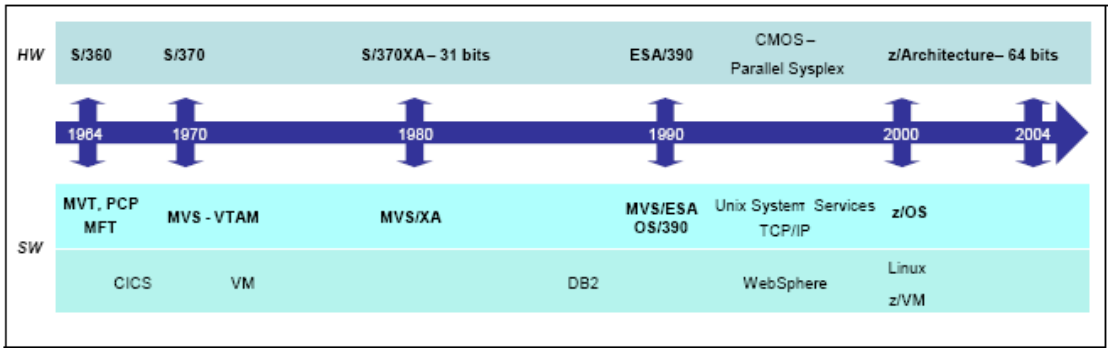
4. 从ISPF中使用TSO OMVS命令启动z/OS UNIX shell。
   - 输入netstat -h.
     是否得到了与练习1相同的信息？
   - 注意您也可以在z/OS UNIX shell(前缀o)使用TCP/IP命令。
   - 输入ping your.ip.addr.ess.
   - 输入traceroute your.ip.addr.ess.
   - 退出z/OS UNIX shell (exit).

564

A

附录 A IBM 主机历史的简要回顾

本附录将讨论从1964年至今IBM主机的发展。整个历程如图A-1所示。



图A-1 IBM主机发展史

在1964年4月7日，IBM推出了360系统(System/360)。该系列的5款功能逐个增强的强劲的计算机，可以运行同一种操作系统，使用相同的44种外围设备。与S/360同时诞生的还有I/O子系统概念(定义指定的处理器去处理内存和I/O设备的数据传输)和并行通道(并行地向I/O设备传输数据的通道)概念。

565

图A-2 S/360型号40

这是第一次，公司可以在一个非常安全的平台为商业活动运行关键的业务程序。

1968年，IBM发布了客户信息控制系统(CICS)。CICS允许车间人事部门在线输入、更新和检索信息。迄今为止，CICS仍然是业界最普遍使用的交易处理器之一。

1969年，Apollo 11号成功的登录月球依靠了数台System 360，信息管理系统(IMS)360，和IBM软件。

1970年夏，IBM发布了一系列增强指令集的机器，成为System/370。这些机器可以在同一系统下使用多于一个的处理器(初始为2个)，并分享内存空间。20世纪70年代，主机体积变得越来越大，运算速度越来越快，并且多处理器系统开始普及。370系列145型号机成为首个拥有全集成单片存储器(电路中，所有相同元素——电阻器，电容器，二极管——都是在一个单一一个硅片上制作而成)和多个128位双极芯片，超过1400块微处理电路元素被排列在一块1/8英寸见方的芯片上。

566

图A-3 S/370™型号165

由于可以运行System/360程序，System/370可以减轻客户的升级压力。System/370同样是第一批具备虚拟存储技术的计算机。这种技术为了延伸计算机空间性能而研发，将硬盘空间模拟为内存以适应软件需求。

3081处理器在1980年得到发展。3081比之前的主机处理器3033在内部性能上提升了2倍。它同时以热传导模块(Thermal Conduction Modules，TCMs)为特点，可以显著节省空间占用，保证散热，以及减少电力使用。

567

图A-4 3081处理器联合体

1982年左右，地址寻址空间从24位增加到31位(370XA)。

1984年，IBM发布了1兆的硅铝金属氧化物半导体(Silicon and Aluminum Metal Oxide Semiconductor，SAMOS)芯片。虽然兆意味着百万，芯片实际上在比小孩手指甲还小的空间上存储了1,048,576位的信息。

1988年系统开始扩展支持多个地址空间。同样在1988年，使用主机，除了决策支持系统，客户还可以将DB2数据库系统部署进核心交易流程中。这降低了CPU的开销并极大地增强了并发行。

在这期间，IBM推出了逻辑分区概念，可以从逻辑上将主机分成若干独立的处理器共享同一套硬件设备。

然而一些工业权威人士并不认为主机可以在20世纪90年代早期生存下来。他们预言个人计算机和小型化服务器的飞速发展将会使得‘大铁锚’(big iron，主机的业界昵称)变得过时。但是IBM相信那些重要的、安全性要求极高的、具有产业优势的计算将一直是需要的，因此诞生了System/390。IBM坚持发展主机，但是从内部对其进行彻底改造，为主机提供全新的技术核心并且不断降低主机的价格。

568

图A-5 S/390 G5和G6

IBM推出系统集群和数据分享的概念，并且推出了System/390并行系统综合体，为实现高级系统可用性提供了可能。

基于互补金属氧化物半导化(Complementary Metal Oxide Semiconductor，CMOS)的处理器被引入主机环境，替代了双极技术并且为主机技术指引了新的方向。CMOS芯片相对于只使用一种晶体管的芯片所需的能量更低。

同样在这十年间，IBM通过企业系统连接(Enterprise System Connectivity，ESCON)引入了并行通道技术，并且开始将网络适配器整合到主机中，成为开放系统适配器(Open System Adapter，OSA)。

1998年，IBM引入了一种新的处理器模块，突破了1,000MIPS的界限，使其成为世界上最强大的主机之一。同样在这期间，逻辑分区扩展到可以支持15个分区。

1999年，按需扩容(Capacity Upgrade on Demand，CUoD)初次出现在System/390上。CUoD提供了额外的处理器，作为备用容量，在业务需要时可以对其进行启用。这提供了一个关键的工具可以帮助公司更好地管理工作负载的峰值需求和处理不可预测的变化。

同样在1999年，IBM推出了首个企业级服务器使用IBM革新的铜芯片技术。这可以扩展客户系统的能力以处理百万级的在线商务事务和大规模企业资源规划应用程序。一种新的概念在这时候产生，即可能在不需要停止机器的情况下对主机进行扩容。

569

FICON，一种新的光纤通道，是ESCON通道容量的8倍。也是在1999年，System/390主机上第一次出现了Linux系统。

2000年10月，IBM宣布第一代z系列主机诞生。z/Architecture系列主机是ESA/390的扩展，并且支持64位寻址技术。动态通道管理技术和专用加密技术同时出现。主机成为开放的平台，并可以运行Linux；专用的处理器(IFL)也因此被研制出来。

z900在2000年发布，它是IBM首个为了电子商务而设计的服务器。

图A-6 z900

z990在z900之后出现。z990有着高达9000 MIPS的运行速度；可用逻辑分区从15个增加到30个，进一步增强了系统的可扩展性。每个操作系统镜像仍然有256个通道的限制，但是z990可以拥有多达1024个通道，他们可以分布在4个逻辑通道子系统中(LCSS)。当前的主机型号也提供了IFL，这是一种专用处理器，为Linux管理集群而设计，同时主机型号也提供了为处理Java而设计的专用处理器zAAP。

570

图A-7 z990

z系列(zSeries)是基于64位z/Architecture设计而成，这种设计可以有效减少内存和存储的瓶颈，并且可以通过智能资源导向器(Intelligent Resource Director，IRD)自动为优先级高的工作负载提供系统资源。IRD是z/Architecture的一个重要特点。并行系统综合体技术和IRD一起被设计用来提供随需应变的商业工作负载所需的系统灵活性以及快速响应能力。

z990提供了多区域(multibook)系统结构，支持1-4个区域(book)配置。每个book由具有12个处理器的多芯片模块(Multiple Chip Module，MCM)组成，MCM中的8个处理器可以配置成标准处理器；除此以外，book中还有内存卡，每个book可以支持最多64G字节大小的内存；以及高性能的自计时互联设备。z900最多可以支持32个处理器。

为了对有高度伸缩性的多区域系统设计提供支持，通道子系统(Channel SubSystem，CSS)通过逻辑通道子系统(Logical Channel SubSystem，LCSS)得以加强，它可以在3个I/O框架上最多安装1024个ESCON通道。有了跨越通道(Spanned Channel)的支持，HiperSockets™，ICB，ISC-3，OSA-Express以及FICON Express可以在LCSS间共享，以提供更强的灵活性。TCP/IP通讯的高速互联，也就是所谓的HiperSockets，允许不同分区以及不同虚拟服务器间建立拥有内存级别速度的TCP/IP通信，而不是一般的网络速度。

最新诞生的主机，IBM System z9 109(也被称作z9-109)，是IBM在主机家族中的下一步发展。它使用z/Architecture的架构，以及z900和z990服务器的指令集(进行了一些扩展)。(这种架构，以前被称为是ESAME架构，通常认为是64位寻址架构，虽然它提供了远远超越64位寻址的能力。)外观上z9-109服务器和z990服务器非常相似。但是，除了对zSeries技术进一步扩展，z9-109在诸多领域如性能表现，可扩展性，可用性，安全性以及虚拟化技术上都进行了加强。

571

图A-8 z9-109

主机在z9-109上进一步的演变发展举例如下：

- 模块化的multi-book设计，支持1-4个book，每个服务器最多拥有54个处理器单元(客户可用的PU)
- 完全64位实存和虚存支持，任何逻辑分区可以被定义为31位或者64位寻址模式
- 最高512G字节的系统内存
- 最多60个逻辑分区

之前主机产品中，系统的I/O设备的数量由通道的数量、每个通道的控制单元数量以及每个控制单元上的设备数量所限制。寻址结构同样造成了一种限制。早期系统的固定3字节地址(每个字节分别对应通道，控制单元和设备)已升级为4字节，支持最多64K设备地址范围。z9-109服务器继续这一增长，提供了多重子系统套件(Multiple Subchannel Sets，MSS)，可以支持最多128K设备地址范围。

572

通道技术从并行通道，ESCON通道技术，向FICON通道技术发展。z9-109服务器继续这一发展，提供了更为显著优越的通道编程技术。

部分服务器工作负载被分担到隔离的处理器上，譬如SAP，ICF，IFL和zAAP。z9-109服务器通过为PR/SM提供单独的内存池，来处理共享的ICF，IFP，和zAAP专用处理器，以提高系统管理能力。

基础的‘真实’系统发展成为虚拟系统，并且，这种发展已经扩展到了系统，处理器，内存，I/O设备，LAN接口等各个层面。z9-109服务器继续这一发展方向，使用新的指令来提高虚拟机器QDIO操作的性能。这是通过创建一种机器转移归向的架构来实现的，该架构的设计用来降低主机编程的开销，并且避免因为适配器中断而导致用户进程的中断。

近来的主机产品扩展了机器指令集，包括和其他平台更为兼容的指令(譬如双浮点数)，用来更好地执行常用编程语言的指令(譬如C/C++的字符串处理指令)，增加寄存器使用的指令(譬如相对指令的和立即指令，以及长位移指令)等等。z9-109服务器进一步扩展了指令，包括新指令和更改的指令。

硬件加密辅助已经在多种早期系统中实现，最近的服务器中该点被进一步突出强调。z9-109服务器，通过扩展基础加密指令的功能，并且通过将多个安全选项(安全协助处理器和加速器)集成到一个系统特色中去，进一步发展了硬件加密处理技术。上面所述的2个安全选项也可以分别单独视为系统的特色。

透明的硬件恢复能力是主机设计的一个发展基石并且在多方面得到了发展。z9-109进一步发展演变，将这种透明恢复功能机制扩展到从I/O框架到系统内存的路径上。

实现并发维护是现代主机的一个主要设计目标，它常需要在组件冗余，以及在芯片和MCM上进行高度集成之间进行平衡。在系统升级或者维修过程中，z9-109服务器允许在一个多区域(multi-book)的配置中，对单个的区域(book)进行并发实时的卸载和重新安装，而不影响其他区域。

573

B

附录 B DB2 范例表

对于381页的第12章"z/OS上的数据库管理系统"，其中大多数例子使用到的数据库表请参照此附录。这些表包含了雇员和部门的相关信息，由这些表组成的应用程序说明了DB2的绝大多数特性。

部门信息表(DEPT)

部门信息表描述了企业中的每一个部门，标识了部门经理，并且展示了它的管理部门信息。这张表存放在名为DSN8D81A.DSN8S81D的表空间中，创建此表的代码如下：

```
CREAT TABLE DSN8810.DEPT
          (DEPTNO     CHAR(3)          NOT   NULL ,
          DEPTNAME  VARCHAR(36)    NOT NULL   ,
          MGRNO       CHAR(3)                        ,
          ADMRDEPT  CHAR(3)          NOT NULL   ,
          LOCATION   CHAR(16)                       ,
          PRIMARY     KEY (DEPTNO)                  )
IN DESN8D81A.DSN8S81D
CCSID EBCDIC;
```

因为此表是自参考的，并且是依赖循环的一部分，它的外键必须使用以下的声明后来加上：

575

```
ALTER TABLE DSN8810.DEPT
          FOREIGN KEY RDD (ADMRDEPT) REFERENCES DSN8810.DEPT
                    ON DELETE CASCADE;
ALTER TABLE DSN8810.DEPT
          FOREIGN KEY RFE (MGRNO) REFERENCES DSN8810.EMP
                    ON DELETE SET NULL;
```

部门信息表字段内容

列号	列名	描述
1	DEPTNO	部门 ID，主键
2	DEPTNAME	部门名称，描述了该部门的主要业务
3	MGRNO	部门经理的员工编号(EMPNO)
4	ADMRDEPT	管理部门 ID，即此部门应上报的部门 ID，最高一级的部门向它自己汇报
5	LOCATION	远程位置的名称

部门信息表中的索引

名称	所在列	索引类型
DSN8810.XDEPT1	DEPTNO	主键索引，升序索引
DSN8810.XDEPT2	MGRNO	升序索引
DSN8801.XDEPT3	ADMRDEPT	升序索引

576

部门信息表的内容

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT	LOCATION
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00	----------
B01	PLANNING	000020	A00	----------
C01	INFORMATION CENTER	000030	A00	----------
D01	DEVELOPMENT CENTER	------	A00	----------
E01	SUPPORT SERVICES	000050	A00	----------
D11	MANUFACTURING SYSTEMS	000060	D01	----------
D21	ADMINISTRATION SYSTEMS	000070	D01	----------
E11	OPERATIONS	000090	E01	----------
E21	SOFTWARE SUPPORT	000100	E01	----------
F22	BRANCH OFFICE F2	------	E01	----------
G22	BRANCH OFFICE G2	------	E01	----------
H22	BRANCH OFFICE H2	------	E01	----------
I22	BRANCH OFFICE I2	------	E01	----------
J22	BRANCH OFFICE J2	------	E01	----------

与其他表间的关系

此表是自参考的：对于管理部门的取值必须是部门ID。这张表是以下表的父表：

- 员工表，对应的外键为WORKDEPT列。
- 项目表，对应的外键为DEPTNO列。同时它依赖于员工表，相应的外键为MGRNO列。

员工信息表(EMP)

员工信息表通过员工号来标识所有的员工，并且列出了他们的基本信息。此表存放在分区表空间DSN8D81A.DSN8S81E中。因为它有对于DEPT表的外键引用，所以DEPT表和它主键上的索引必须首先被创建。然后EMP表可以通过如下的代码来创建：

577

```
CREATE TABLE DSN8810.EMP
        (EMPNO          CHAR(6)         NOT NULL,
        FIRSTNME   VARCHAR(12)    NOT NULL,
        MIDNME          CHAR(1)         NOT NULL,
        LASTNAME  VARCHAR(15)    NOT NULL,
        WORKDEPT CHAR(3),
        PHONENO   CHAR(4)          CONSTRANT NUMBER CHECK
                  (PHONENO >= '0000' AND
                  PHONENO <= '9999')      ,
        HIREDATE DATE                                   ,
        JOB CHAR(8)                                     ,
        EDLEVEL SMALLINT                                ,
        SEX    CHAR(1)                                  ,
        BIRTHDATE DATE                                  ,
        SALARY DECIMAL(9,2)                             ,
        BONUS    DECIMAL(9,2)                           ,
        COMM DECIMAL(9,2)                       ,
        PRIMARY KEY (EMPNO)                             ,
        FOREIGN KEY RED (WORKDEPT) REDERENCES DSN8810.DEPT
        ON DELETE SET NULL                              )
EDITPROC DSN8EAE1
IN DSN8D81A.DSN8S81E
CCSID EBCDIC;
```

578

员工信息表列字段的描述

列号	列名	描述
1	EMPNO	员工号(主键)
2	FIRSTNME	员工名
3	MIDINIT	员工中间名字
4	LASTNME	员工姓
5	WORKDEPT	员工工作部门的 ID
6	PHONENO	员工电话号码
7	HIREDATE	雇佣日期
8	JOB	员工负责的工作
9	EDLEVEL	受正规教育的年数
10	SEX	员工性别(用 M 或者 F 表示)
11	BIRTHDATE	出生日期
12	SALARY	以美元为单位的年薪
13	BONUS	以美元为单位的年终奖金
14	COMM	每年的佣金

员工信息表的索引

名称	所在列	索引类型
DSN8810.XEMP1	EMPNO	主键索引，分区索引，升序索引
DSN8810.XEMP2	WORKDEPT	升序索引

与其他表间的关系

此表是以下表的父表：

- 部门表，对应的外键为MGRNO列。
- 项目表，对应的外键为RESPEMP列。同时它依赖于部门表，相应外键为WORKDEPT列。

579

580

C

附录 C 实用程序

目前对z/OS实用程序的具体构成并没有明确的定义，但是z/OS提供的程序普遍使用的仅包括了极个别的，我们称之为实用程序(utility)。与之对照的是，UNIX界将很多标准命令作为实用程序。它包括了编译器、备份程序、过滤器以及很多其他类型的程序。而在z/OS界中这些程序叫做”应用程序”(application)或者”程序”(program)，而不是”实用程序”[1]。不同之处仅在术语名称不同而已，然而却可能造成新z/OS用户的混淆和困惑。

z/OS实用程序是批处理程序(尽管它们同样可以通过适当的ALLOC命令运行于TSO前台)，它们往往有着类似的JCL需求，包括SYSPRINT、SYSIN、SYSUT1以及SYSUT2等DD语句。绝大多数的z/OS用户都熟悉IEFBR14、IEBGENER以及IEBCOPY。VSAM用户一定也会对IDCAM很熟悉。

相比于z/OS所具有的范围广泛的功能和处理能力，系统提供的实用程序的数量却很少。这导致了大量用户编写的实用程序的出现(尽管很多用户都避免将其命名为实用程序)，其中的很多都在用户社区被广泛的共享使用。独立的软件提供商同样提供了很多类似的产品(需要付费)。其中的一些可以被归类到实用程序；其中的一些和IBM实用程序功能相同，互相竞争；而更多其他的产品提供了IBM实用程序中所没有包含的功能。

这里提到的绝大多数基本和系统实用程序都在相关的IBM出版的书籍”z/OS DFSMSdfp Utilities”中都有详细描述。本附录旨在提供那些实用程序的概要介绍，并提供了多数基本实用程序功能的简单示例。

基本实用程序

一些在批处理作业中广泛使用到的实用程序(这里用传统的术语)，将在此处具体描述。

IEFBR14

该程序的唯一功能就是提供了一个为零(0)的完成码。它是用来”执行JCL”的一个安全方法。”执行JCL”或许不是一个合适的术语，但能很好地表达出语意。例如，考

[1] 对于实用程序，z/OS UNIX 用户使用通用的 UNIX 术语。

虑如下的作业：

```
//OGDEN1 JOB 1, BILL, MSGCLASS=X
//         EXEC PGM=IEFBR14
//A DD DSN=OGDEN.LIB.CNTL, DISP=(NEW,CATLG),VOL=SER=WORK02,
//              UNIT=3390,SPACE=(CYL,(3,1,25))
//B DD DSN=OGDEN.OLD.DATA, DISP=(OLD,DELETE)
```

这是一个很有用的作业，尽管它所执行的程序(IEFBR14)什么也没做。在准备运行这个作业时，启动程序分配了数据集"OGDEN.LIB.CNTL"并在作业结束时将其保留。同时在作业结束时，删除数据集"OGDEN.OLD.DATA"。DD名A和B没有特别含义，仅仅是因为DD语句的语法需要一个DD名字。

当然，可以通过诸如ISPF的工具来完成创建一个数据集和删除另一个数据集的相同功能。然而，这些操作可能需要用作一个大的批处理作业执行序列的一部分。

提示：这里名称IEFBR14非常有趣。一个编写早期OS/360的IBM小组使用IEF作为他们编写的所有模块的前缀。在汇编语言中，BR代表转向到寄存器中的地址。转向14号通用寄存器是结束一个程序的标准做法。尽管并不是一个十分高明的名字，但实际中所有的z/OS专用用户都能轻易的记住IEFBR14。

IEFBR14不是一个实用程序，之所以这样说原因在于它并不包含在实用程序"Utilities"的使用然而对这个十分有用的程序却没有其他的特别分类，因此我们姑且把它放到实用程序的分类中。

582

IEBGENER

IEBGENER实用程序可以将一个顺序数据集内容复制到另一个数据集中。(记住分区数据集的成员同样可以看作一个顺序数据集。)它同样可以完成过滤筛选数据、修改LRECL和BLKSIZE、产生记录、以及一些其他的功能。然而，最为常用的功能就是复制数据集。一个典型的作业如下：

```
//OGDEN2 JOB 1,BILL,MSGCLASS=X
//   EXEC   PGM=IEBGENER
//SYSIN DD DUMMY
//SYSPRINT DD SYSOUT=X
//SYSUT1 DD DISP=SHR,DSN=BILL.SEQ.DATA
//SYSUT2 DD DISP=(NEW,CATLG),DSN=BILL.COPY.DATA,UNIT=3390,
//       VOL=SER=WORK02,SPACE=(TRK,3,3))
```

IEBGENER需要4个如上例中指定的DD名对应的DD语句。SYSIN DD语句用于读取控制参数；对于简单应用不需要使用任何控制参数，则可使用DD DUMMY指定。SYSPRINT语句用于保存IEBGENER实用程序产生的消息。SYSUT1语句用于输入而SYSUT2语句用于输出。本例读取了一个已存在的数据集并将其内容复制到一个新的数据集中。

如果输出数据集是新的并且没有指定DCB参数，则IEBGENER将复制并使用输入数据集的DCB参数。(DCB参数包括LRECL、RECFM以及BLKSIZE，参见第173页5.8"数据集记录格式")

另一个常用的例子如下面的作业：

```
//OGDEN2 JOB 1,BILL,MSGCLASS=X
//   EXEC   PGM=IEBGENER
//SYSIN DD DUMMY
//SYSPRINT DD SYSOUT=X
//SYSUT2 DD DISP=OLD,DSN=BILL.TEST.DATA
//SYSUT1 DD *
       This is in-stream data. It can be as long
       as you like. It appears to an application as
       LRECL=80, RECFM=F, BLKSIZE=80. You would
       want to have the SYSUT2 data set allocated with
       a better blocksize.
/*
```

本例假设BILL.TEST.DATA已经创建。该作业将使用SYSUT1输入流内容覆盖该数据集。由于输出数据集已经存在，因此IEBGENER将使用数据集已经存在的DCB属性。

IEBGENER是z/OS提供的最为基本的复制或列表程序。从OS/360的最早版本便开始提供该程序。

IEBCOPY

该实用程序通常用于以下几种目的：

- 将一个分区数据集中选定的(或所有的)的成员复制到另一个分区数据集中。
- 将一个分区数据集复制成一个独特的顺序结构，称为已卸载(unloaded)分区数据集。作为一个顺序数据集后，它可以写入到磁带上、通过FTP传送[2]、或者像一个简单顺序数据集一样进行操作。
- 读取一个转储的分区数据集(是一个顺序文件)并将其还原成为原来的分区数据集。另一种可选的操作是，只恢复选中的成员。
- 就地压缩一个分区数据集以恢复所丢失的空间。

大多数的z/OS软件产品都是以转储的分区数据集的方式发布的。ISPF复制选项(选项3.3)即是隐式地使用了IEBCOPY。将一个PDS或者PDSE从一个卷移动到另一个卷，可以通过IEBCOPY方便地完成。如果需要在批处理作业中操作分区数据集，IEBCOPY就可能需要使用到。在TSO下的相应等价操作(使用ISPF)也间接地使用到了IEBCOPY。

一个简单的IEBCOPY作业可能如下：

```
//OGDEN5 JOB 1,BILL,MSGCLASS=X
//       EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=*
//SYSIN DD DUMMY
//SYSUT1 DD DISP=SHR,DSN=OGDEN.LIB.SOURCE
//SYSUT2 DD DISP=(NEW,KEEP),UNIT=TAPE,DSN=OGDENS.SOURCE,
//                  VOL=SER=123456
```

该作业将会卸载OGDEN.LIB.SOURCE(假设其为分区数据集)并将其写入到磁带上。(这里名称TAPE假设为一个”内部名称”，系统将其与磁带驱动器相关联。)默认情况下IEBCOPY将从SYSUT1复制到SYSUT2。注意磁带上的数据集名称同作为输入的数据集名称不相同(可以使用相同的名称，但这里没有必要这样做)。下面

[2] 输出数据集的记录格式一般是 V 或者 VB，并且通过 FTP 发送 V 或者 VB 数据集有额外的考虑。

的作业可以用来将该PDS恢复到另一个盘卷上：

```
//JOE6 JOB 1,JOE,MSGCLASS=X
//        EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=*
//SYSIN DD DUMMY
//SYSUT1 DD DISP=OLD,UNIT=TAPE,DSN=OGDENS.SOURCE,VOL=SER=123456
//SYSUT2 DD DISP=(NEW,CATLG),DSN=P390Z.LIB.PGMS,UNIT=3390,
//                  SPACE=(TRK,(10,10,20)),VOL=SER=333333
```

584

本例中IEBCOPY将侦测到输入的数据集是一个转储的分区数据集。这里我们需要额外的知识来确定数据集需要10个磁道且应该有20个目录块。SYSIN处可以不使用DD DUMMY语句，而代之为：

```
//SYSIN DD *
            COPY OUTDD=SYSUT2,INDD=SYSUT1
            SELECT MEMBER=(PGM1,PGM2)
/*
```

OUTDD和INDD参数用于指定将被使用到的DD名。在本例中我们简单地使用了默认名，但这并不是必须的。SELECT语句指定了将被处理的成员名称。

将一个分区数据集从一个已卸载拷贝中恢复过来将自动地压缩(恢复丢失空间)该数据集。

IEBDG

IEBDG实用程序用于创建记录，该记录中的域(field)可以通过各种类型的数据填充。IEBDG的一个典型使用就是用于创建测试数据，产生各种不同的域和数据，并且可以通过轮换、反复、移位、翻转以及其他的域置换等操作改变每个记录的域。IEBDG可以接受输入数据记录并使用产生的数据覆盖三输入内容中的指定域。

如下为IEGDB使用的一个简单示例：

```
//OGDEN7 JOB 1,BILL,MSGCLASS=X
//         EXEC PGM=IEBDG
//SYSPRINT DD SYSOUT=*
//OUT DD DISP=(NEW,CATLG),DSN=OGDEN.TEST.DATA,UNIT=3390,
//             VOL=SER=WORK01,SPACE=(CYL,(10,1)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=8000)
//SYSIN DD *
      DSD OUTPUT=(OUT)
      FD NAME=FIELD1,LENGTH=30,FORMAT=AL,ACTION=RP
      FD NAME=FIELD2,LENGTH=10,PICTURE=10,'TEST DATA '
      FD NAME=FIELD3,LENGTH=10,FORMAT=RA
      CREATE QUANTITY=90000,NAME=(FIELD1,FIELD2,FIELD3)
      END
/*
```

该作业创建了一个新的数据集OGDEN.TEST.DATA，拥有90,000条记录。每条记录80字节长，通过DD语句的DCB参数指定。控制语句指定了占用每条记录前50个字节的三个域。默认情况下，IEBDG将剩余的字节用二进制零填充。这三个域分别是：

585

- 字母域(‘ABCDEF…’)，30字节长，每条记录产生之后进行一次轮换(轮转左边的一个字节)。

- 第二个域包含了10字节的确定常量‘TEST DATA ’。
- 第三个域包含了10字节的随机二进制数据。

该实用程序可以产生更加复杂的数据模式，而该例仅是一个典型的简单应用。它同样也可展示了关于数据所需磁盘空间量的估算，如下：

- 我们知道3390磁道的空间大概有57K，考虑到记录之间间隙所占用的空间，实际可用的空间会更小。
- 我们知道DCB参数(在JCL中指定)为LRECL=80，BLKSIZE=8000，并且RECFM=FB。我们不知道为何选定这些DCB参数，但我们假定它们同即将使用这些测试数据的程序有关。
- 我们可以估算每6个8000大小的块可能填入一个磁道。这不是一个高效的块大小，因为磁道的空间可能被浪费掉，但是这个大小却足够使用。
- 每个块包含100条记录，每条记录80字节。每个磁道包含600条记录。(对于FB类型的记录，每个块的空间会被充分利用。)
- 一个柱面包含了15个磁道，因此一个柱面可以存放9000条这样的记录。
- 基于以上考虑，我们需要10个柱面来存放这90,000条记录。我们指定10个柱面作为JCL中的首次分配空间，同时二次分配增量指定为1个柱面。我们应该不需要任何的二次分配，但如此指定可以更保险。我们可以请求150个磁道取代10个柱面，其结果也是相同的。

IDCAMS

IDCAMS程序并不在z/OS Utilities手册中记录的基本z/OS实用程序集中。IDCAMS程序主要用于创建和操作VSAM数据集。它也有其他的功能(诸如目录更新)，但还是同VSAM的联系最为紧密。它提供了很多复杂的功能，要描述其全部的功能需要一整本的手册。在本书的成书之时，对应的基本IBM手册是”DFSMS Access Method Services for Catalogs”。

IDCAMS简单使用的一个典型示例如下：

```
//OGDEN12 JOB 1,BILL,MSGCLASS=X
//DEL      EXEC PGM=IDCAMS
//SYSPRINT   DD SYSOUT=*
//SYSIN DD *
                DELETE OGDEN.DATA.VSAM CLUSTER
/*
//LOAD EXEC PGM=IDCAMS
//SYSPRINT DD *
//DATAIN DD DISP=OLD,DSN=OGDEN.SORTOUT
//SYSIN DD *
DEFINE CLUSTER (NAME (OGDEN.DATA.VSAM) -
VOLUMES(WORK02) CYLINDERS(1 1) -
RECORDSIZE (72 100) KEYS(9 8) INDEXED)
REPRO INFILE(DATAIN) OUTDATASET(OGDEN.DATA.VSAM) ELIMIT(200)
/*
```


该示例要点说明如下：

- 有两个作业步。第一个作业步删除了一个数据集，该数据集将在第二步中被创建。这是一个预清理作用。该数据集可能在此刻并不存在，于是在执行完第一个作业步后可能会得到一个运行失败的完成码，将其忽略。
- 注意这里没有用于VSAM数据集的DD语句。IDCAMS将执行动态分配来创建

需要的JCL。

- 第二个作业步完成了两个功能。首先创建了一个VSAM数据集(通过DEFINE CLUSTER命令)，然后从顺序文件中载入数据到数据集里(通过REPRO命令)。该顺序数据集需要一个DD语句。
- DEFINE CLUSTER命令延续了三个记录行。续行符同TSO命令下使用的符号相同。
- VSAM数据集位于WORK02卷上，并且使用了一个柱面作为首次分配空间、一个柱面作为二次分配量。平均的记录大小是72字节同时最大的记录长度为100字节。(VSAM数据集总是使用变长记录。)主键(用于在数据集中访问记录)是8字节长并且起始于每条记录的9字节位移处。
- ELIMIT参数指定了REPRO在终止操作之前忽略的错误记录数。一条错误记录常常是由键值重复造成。

IDCAMS的很多功能可以通过键入TSO命令完成。例如DEFINE CLUSTER可以用作是一个TSO命令。然而，这种方法并不值得推荐，原因在于命令本身可能会很复杂而且遇到的错误也可能会很复杂。通过批处理作业输入IDCAMS命令可使得命令和结果信息通过SDSF查看输出的方式，在需要时能反复的回顾和检查。

587

提示：有些用户将这个程序发音为"id cams"(两个音节)，而另一些用户读作"I D cams"(三个音节)。

IEBUPDTE

IEBUPDTE实用程序用于在一个分区数据集中创建多个成员，或更新一个成员的记录。虽然它也可用于其他类型的记录，但其主要用途是创建或维护一个JCL过程库或汇编宏库。现在，该实用程序多数用于z/OS特许程序的发布或维护。TSO用户很少使用它。

以下基本示例包括将两个JCL过程加入到MY.PROGLIC中，我们可通过ISPF方便的完成该功能，但这里我们假设发送如下的作业到磁带上，这样其实用性则更加明显：

```
//OGDEN10 JOB 1,BILL,MSGCLASS=X
//       EXEC PGM=IEBUPDTE
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DISP=OLD,DSN=MY.PROCLIB
//SYSUT2 DD DISP=OLD,DSN=MY.PROCLIB
//SYSIN DD DATA
./     ADD LIST=ALL,NAME=MYJOB1
//STEP1 EXEC=BILLX1
//PRINT DD SYSOUT=A
//                  ( more JCL for MYJOB1)
//SYSUDUMP DD SYSOUT=*        (last JCL for MYJOB1)
./           REPL LIST=ALL,NAME=LASTJOB
//LIST EXEC PGM=BILLLIST
//           (more JCL for this procedure)
//* LAST JCL STATEMENT FOR LASTJOB
./                 ENDUP
/*
```

本例注释如下：

- 当一个库将被要被更新，SYSUT1和SYSUT2都指向到那个库。(如果它们指

向了不同的库，则SYSUT1库将被复制到SYSUT2库中，然后再被更新。)

- SYSIN DD DATA形式表明位于输入流内的数据在第一、二列包含了"//"符号，它将不会被解析为JCL。输入流的结束用/*表示。
- IEBUPDTE实用程序的控制语句在开始两列中使用./表示。
- 名为MYJOB1的成员加入到MY.PROCLIB中；该成员不能已经存在于此库中。
- 名为LASTJOB的成员将被作业中的新内容所代替。

588

IEBUPDTE实用程序也可基于语句中的序号来加入或替代成员中的语句。它是为数不多的仍然操作JCL或源代码中序号的程序之一。

再次强调，IEBUPDTE典型地用于程序发布和维护。例如软件提供商的产品需要在用户的过程库中加入15个JCL过程，提供商可能将这些过程打包在一个IEBUPDTE作业中。这样做的一个优点在于所有的材料都以源码的形式提供，用户可以在运行作业之前方便地检查其内容。

面向系统的实用程序

本节所讨论的程序提供给系统管理员一些基本的实用功能，这里仅作简要描述。

IEHLIST

IEHLIST实用程序用于列出分区数据集的目录内容或者磁盘卷的VTOC信息。它通常用于列出VTOC内容并提供位级别的信息。IEHLIST在大多数的系统设置中并不常用，但在VTOC损坏的极少数情况下还是需要用到。有时该实用程序会与SUPERZAP程序一同使用来修补或者整理损坏的VTOC。

IEHINITT

IEHINITT实用程序用于在磁带上写入标准标签。如果需要，它可以用来为单个磁带或者大批的磁带写上标签。很多大型的z/OS系统不允许在系统中使用没有标签的磁带。

IEHPROGM

IEHPROGM实用程序几乎已经过时。它主要用于通过程序而不是JCL来管理编目、重命名数据集以及删除数据集，主要在系统安装配置或者主要程序产品的安装配置过程中使用。这些功能可能包含了几十个(或几百个)这样的编目和数据集动作。将命令预先准备(通过IEHPROGM批处理作业)的方法比动态方法产生错误的概率小很多。IEHPROGM的大多数功能在IDCAMS中都有提供，并且后者是现在完成编目和数据集功能的首选实用程序。

589

ICKDSF

ICKDSF实用程序主要用于初始化磁盘卷。该程序最小的功能包括创建磁盘标签记录和VTOC。ICKDSF也能够扫描卷以确保其可用性、重新格式化所有磁道、写入宿主地址(home address)和R0记录等等。

SUPERZAP

SUPERZAP 程序(实际的名字已经更改过很多次)能够用来修补磁盘记录。它能解析PDS库中可执行模块的格式，以用于修补这样的可执行模块，这也是该程序最为普遍的使用需求。SUPERZAP如今在系统维护方面已经不再常用；它更多地用在早期版本的操作系统中。

SUPERZAP用于修补VTOC，可执行程序或者几乎任何其他的磁盘记录。实际上，它最为普遍的用途是修补可执行程序。该程序早期被广泛地用于在程序中安装小补丁。

例如，某产品发布新的版本，该新版本可能已经通过磁带发布给成千上万的用户。在所有这些磁带被交付后，开发者可能会发现一个很小的程序问题，它需要通过修改一些指令来修正这个问题。为了避免制作新的发布磁带并将其交付到所有用户手中(对主流软件产品而言这是一个巨大而昂贵的负担)，开发者可以创建一个SUPERZAP的解决方案并将其以邮件/传真/FTP的方式发送给用户。

SUPERZAP是一个位级别的工具。在相对很少的位或者字节需要修改时，可以用到该工具。如下即为一个SUPERZAP的示例：

```
//OGDEN15 JOB 1,BILL,MSGCLASS=X
//     EXEC PGM=AMASPZAP
//SYSPRINT DD SYSOUT=*
//SYSLIB DD DISP=OLD,DSN=OGDEN.LIB.LOAD
//SYSIN DD *
NAME QSAM1
VERIFY 004E 4780
REP 004E 4700
/*
```

SYSLIB DD语句必须指向包含了需修改的装入模块的数据集。NAME控制语句指定了要修改的可执行模块(即PDS成员名称)。VERIFY语句指示检查模块中位移为x'004E'的位置并验证其包含了数据x'4780'。如果验证通过则将模块该位移处的值修改为x'4700'。 这种修改将等于转向指令改为了空操作指令，我们假设这修改了程序逻辑。

590

我们可以制作如此的SUPERZAP补丁，是因为我们可以得到程序的汇编列表，并且可以从中找出需修改指令在模块内部的精确定位。即使不使用汇编列表，我们同样可以通过阅读十六进制的内存转储并以此来重构机器语言操作，但这样的做法比起前者显然要复杂得多。

值得注意的是在磁盘上存放的可执行程序的格式十分复杂，而且载入到内存后并不是一个简单的程序镜像。(可重定位数据、外部符号以及优化后的磁盘载入格式都构成了其复杂性。)SUPERZAP能够理解该磁盘格式并允许用户操作可执行程序就如同它是一个内存镜像一样。

我们讨论了SUPERZAP，但示例中使用的程序为AMASPZAP，这是该程序目前的名字，尽管它为人熟知的名字是SUPERZAP。

应用级实用程序

有很多的应用程序可以被看作为实用程序。这里简要介绍几个十分常用的实用程序。这些程序在使用上比上述的基本实用程序复杂得多，因此这里的介绍不包括其应用示例。

ADRDSSU

该程序是z/OS所提供的主要”磁盘转储”和”磁盘恢复”程序。它可以筛选和挑选哪些数据集进行转储或者恢复，但它主要还是用于整盘转储。磁盘转储的目的通常在于提供一份内容备份，以供日后需要时能恢复使用。通常是将整个卷进行转储但只需要恢复那些意外受损的特定数据集。

备份通常存入到磁带上，但也同样可以写入到一个磁盘数据集中。一个磁盘可以一个磁道接一个磁道地转储(即物理转储)，也可以一个数据集接一个数据集地转储(即逻辑转储)。当完成了逻辑转储后，多个数据集分区将合并成为一个单一的分区，分区数据集将被压缩，并且所有空闲空间都放于一个单一的分区中。

RMF

资源度量工具(Resource Measurement Facility，RMF)是一个可选的IBM特许程序，用于度量系统性能的各个方面。不同的RMF模块分别提供长时间运行统计收集、瞬时数据、长时间运行报告、批处理类型报告、基于TSO的报告等等。硬件I/O系统负责维护多个统计计数器，这些计数器记录每个I/O设备的排队时间、每个设备的活动量、以及其他底层的相关信息。RMF访问这些硬件计数器并将其内容包含到最后的报告之中。

591

592

D

附录 D EBCDIC-ASCII 表

EBCDIC - ASCII table

Hx	Dec	E	A	Hx	Dec	E	A	Hx	Dec	E	A	Hz	Dec	E	A
00	00		NUL	20	32		SP	40	64	SP	@	60	96	-	'
01	01			21	33		!	41	65		A	61	97	/	a
02	02			22	34		"	42	66		B	62	98		b
03	03			23	35		#	43	67		C	63	99		c
04	04			24	36		$	44	68		D	64	100		d
05	05			25	37		%	45	69		E	65	101		e
06	06			26	38		&	46	70		F	66	102		f
07	07			27	39		'	47	71		G	67	103		g
08	08			28	40		(	48	72		H	68	104		h
09	09			29	41		)	49	73		I	69	105		i
0A	10			2A	42		*	4A	74	^	J	6A	106	|	j
0B	11			2B	43		+	4B	75	.	K	6B	107	,	k
0C	12			2C	44		,	4C	76	<	L	6C	108	%	l
0D	13			2D	45		-	4D	77	(	M	6D	109	_	m
0E	14			2E	46		.	4E	78	+	N	6E	110	>	n
0F	15			2F	47		/	4F	79	|	O	6F	111	?	o
10	16			30	48		0	50	80	&	P	70	112		p
11	17			31	49		1	51	81		Q	71	113		q
12	18			32	50		2	52	82		R	72	114		r
13	19			33	51		3	53	83		S	73	115		s
14	20			34	52		4	54	84		T	74	116		t
15	21			35	53		5	55	85		U	75	117		u
16	22			36	54		6	56	86		V	76	118		v

Hx	Dec	E	A	Hx	Dec	E	A	Hx	Dec	E	A	Hz	Dec	E	A
17	23			37	55		7	57	87		W	77	119		w
18	24			38	56		8	58	88		X	78	120		x
19	25			39	57		9	59	89		Y	79	121		y
1A	26			3A	58		:	5A	90	!	Z	7A	122	:	z
1B	27			3B	59		;	5B	91	$	[	7B	123	#	{
1C	28			3C	60		<	5C	92	*	\	7C	124	@	|
1D	29			3D	61		=	5D	93	)	]	7D	125	'	}
1E	30			3E	62		>	5E	94	;	^	7E	126	=	~
1F	31			3F	63		?	5F	95	not	_	7F	127	"	
80	128			A0	160			C0	192	{		E0	224	\	
81	129	a		A1	161			C1	193	A		E1	225		
82	130	b		A2	162	s		C2	194	B		E2	226	S	
83	131	c		A3	163	t		C3	195	C		E3	227	T	
84	132	d		A4	164	u		C4	196	D		E4	228	U	
85	133	e		A5	165	v		C5	197	E		E5	229	V	
86	134	f		A6	166	w		C6	198	F		E6	230	W	
87	135	g		A7	167	x		C7	199	G		E7	231	X	
88	136	h		A8	168	y		C8	200	H		E8	232	Y	
89	137	i		A9	169	z		C9	201	I		E9	233	Z	
8A	138			AA	170			CA	202			EA	234		
8B	139			AB	171			CB	203			EB	235		
8C	140			AC	172			CC	204			EC	236		
8D	141			AD	173	[		CD	205			ED	237		
8E	142			AE	174			CE	206			EE	238		
8F	143			AF	175			CF	207			EF	239		
90	144			B0	176			D0	208	}		F0	240	0	
91	145	j		B1	177			D1	209	J		F1	241	1	
92	146	k		B2	178			D2	210	K		F2	242	2	
93	147	l		B3	179			D3	211	L		F3	243	3	
94	148	m		B4	180			D4	212	M		F4	244	4	
95	149	n		B5	181			D5	213	N		F5	245	5	
96	150	o		B6	182			D6	214	O		F6	246	6	
97	151	p		B7	183			D7	215	P		F7	247	7	
98	152	q		B8	184			D8	216	Q		F8	248	8	
99	153	r		B9	185			D9	217	R		F9	249	9	
9A	154			BA	186			DA	218			FA	250		
9B	155			BB	187			DB	219			FB	251		
9C	156			BC	188			DC	220			FC	252		
9D	157			BD	189	]		DD	221			FD	253		
9E	158			BE	190			DE	222			FE	254		
9F	159			BF	191			DF	223			FF	255		

E

附录 E Class 程序

这里的所有练习都涉及一个员工文件(或数据库)；该文件中所有的员工都有唯一的员工编号标识，文件还包括了基本员工信息。

该练习将部门编号作为输入数据，根据部门编号选中该部门的所有员工记录，把这些记录的工资(salary)字段值加起来。最后再计算出平均工资值并显示给用户。

下面的练习由各种不同的语言编写，在不同的环境中运行，操作不同的数据源，但他们都实现了以上所述的功能。这里提供了源代码，准备作业，以及相关指令。

我们假设学生已经安装并配置好了3270模拟器，并拥有合适的z/OS平台上的TSO，CICS，DB2和WebSphere的相关访问权限。请注意每个练习中可能需要的系统定义(比如HLQ，DB2数据库名字等等)。

COBOL-CICS-DB2 程序

源代码

Map定义

该定义放在LUISM.TEST.SAMPLIB库的TMAP01成员中。

```
          PRINT NOGEN
TMAPSET  DFHMSD TYPE=&SYSPARM,                                         X
               LANG=COBOL,                                             X
               MODE=INOUT,                                             X
               TERM=3270-2,                                            X
               CTRL=FREEKB,                                            X
               STORAGE=AUTO,                                           X
               TIOAPFX=YES
TMAP01   DFHMDI SIZE=(24,80),                                          X
               LINE=1,                                                 X
               COLUMN=1,                                               X
               MAPATTS=COLOR
         DFHMDF POS=(1,1),                                             X
               LENGTH=9,                                               X
               ATTRB=(NORM,PROT),                                      X
               COLOR=BLUE,                                             X
               INITIAL='ABCD txid'
         DFHMDF POS=(1,26),                                            X
               LENGTH=28,                                              X
               ATTRB=(NORM,PROT),                                      X
               COLOR=GREEN,                                            X
               INITIAL='Average salary by department'
         DFHMDF POS=(9,1),                                             X
               LENGTH=41,                                              X
                ATTRB=(NORM,PROT),                                      X
               INITIAL='Type a department number and press enter.'
         DFHMDF POS=(11,1),                                            X
               LENGTH=18,                                              X
               ATTRB=(NORM,PROT),                                      X
               COLOR=GREEN,                                            X
               INITIAL='Department number:'
DPTONO   DFHMDF POS=(11,20),                                           X
               LENGTH=3,                                               X
               ATTRB=(NORM,UNPROT,IC),                                 X
               COLOR=TURQUOISE,                                        X
               INITIAL='___'
         DFHMDF POS=(11,24),                                           X
               LENGTH=1,                                               X
               ATTRB=ASKIP
         DFHMDF POS=(13,1),                                            X
               LENGTH=18,                                              X
               ATTRB=(NORM,PROT),                                      X
               COLOR=GREEN,                                            X
               INITIAL='Average salary($):'
AVGSAL   DFHMDF POS=(13,20),                                           X
```

```
               LENGTH=11,                                              x
               ATTRB=(NORM,PROT),                                      X
               COLOR=TURQUOISE
MSGLINE  DFHMDF POS=(23,1),                                            x
                 LENGTH=78,                                           x
               ATTRB=(BRT,PROT),                                       x
               COLOR=BLUE
         DFHMDF POS=(23,79),                                           x
               LENGTH=1,                                               x
               ATTRB=(DRK,PROT,FSET),                                  x
               INITIAL=' '
         DFHMDF POS=(24,1),                                            x
               LENGTH=7,                                               x
               ATTRB=(NORM,PROT),                                      x
               COLOR=RED,                                              x
               INITIAL='F3=Exit'
         DFHMSD TYPE=FINAL
         END
```

程序代码

该程序位于LUISM.TEST.SAMPLIB库的XYZ2成员中。

```
IDENTIFICATION DIVISION.
      *-------------------------------------------------------
      *    COBOL-CICS-DB2 PROGRAM ZSCHOLAR RESIDENCY
      *    OBTAINS THE AVERAGE SALARY OF EMPLOYEES OF A GIVEN DEPART.
      *-------------------------------------------------------
      *----------------------
       PROGRAM-ID.    XYZ2.
      /
       ENVIRONMENT DIVISION.
      *--------------------
       CONFIGURATION SECTION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
       DATA DIVISION.
      *-------------
       FILE SECTION.
      /
      *-------------
       WORKING-STORAGE SECTION.
      ************************************************
      * WORKAREAS                                    *
      ************************************************
       01 SWITCH.
```

```
  05 DATA-IS         PIC X VALUE 'Y'.
     88 DATA-IS-O           VALUE 'Y'.
  05 SEND-IND        PIC X.
     88 SEND-IND-ERASE     VALUE '1'.
     88 SEND-IND-DATAO     VALUE '2'.
     88 SEND-IND-ALARM     VALUE '3'.
 01 COMMUNICATION-AREA PIC X.
 01 MSGLINET.
    02 MSGSQLC      PIC X(8).
    02 FILLER       PIC X.
    02 MSGREST      PIC X(69).
************************************************
* DB2 HOST VARIABLES DECLARATION                *
************************************************
 01  WORKDEPT-HV   PIC X(3).
 01  SALARY-HV     PIC X(11).
 01  SALARY-IN     PIC S9(4) COMP-5.
************************************************
* SQLCA DECLARATION                             *
************************************************
     EXEC SQL INCLUDE SQLCA END-EXEC.
************************************************
* DFHAID                                        *
************************************************
    COPY DFHAID.
************************************************
* MAP COPY                                      *
************************************************
    COPY MAPONL.
************************************************
* DECLARE OF DB2 TABLE                          *
************************************************
     EXEC SQL
      DECLARE DSN8810.EMP TABLE
      (EMPNO     CHAR(6)     NOT NULL,
       FIRSTNAME VARCHAR(12) NOT NULL,
       MIDINIT   CHAR(1)     NOT NULL,
       LASTNAME  VARCHAR(15) NOT NULL,
       WORKDEPT  CHAR(3)             ,
       PHONENO   CHAR(4)             ,
       HIREDATE  DATE                ,
       JOB       CHAR(8)             ,
       EDLEVEL   SMALLINT            ,
       SEX       CHAR(1)             ,
       BIRTHDATE DATE                ,
       SALARY    DECIMAL(9,2)        ,
       BONUS     DECIMAL(9,2)        ,
       COMM      DECIMAL(9,2)        )
     END-EXEC.
```

598

```
*****************************************************************
 LINKAGE SECTION.
*****************************************************************
 01 DFHCOMMAREA    PIC X.
/
 PROCEDURE DIVISION USING DFHCOMMAREA.
*****************************************************************
* MAIN ROGRAM ROUTINE
*****************************************************************
 MAINLINE.
*****************************************************************
* 2000-PROCESS
*****************************************************************
 2000-PROCESS.
     EVALUATE TRUE
      WHEN EIBCALEN = ZERO
         MOVE LOW-VALUE TO TMAP010
         SET SEND-IND-ERASE TO TRUE
         PERFORM 2000-10-SEND
      WHEN EIBAID = DFHCLEAR
         MOVE LOW-VALUE TO TMAP010
         SET SEND-IND-ERASE TO TRUE
         PERFORM 2000-10-SEND
      WHEN EIBAID = DFHPA1 OR DFHPA2 OR DFHPA3
         CONTINUE
      WHEN EIBAID = DFHPF3
          EXEC CICS RETURN
         END-EXEC
         GOBACK
      WHEN EIBAID = DFHENTER
         PERFORM 2000-00-PROCESS
      WHEN OTHER
         MOVE LOW-VALUE TO TMAP010
         MOVE 'WRONG KEY' TO MSGLINE0
         SET SEND-IND-ALARM TO TRUE
         PERFORM 2000-10-SEND
     END-EVALUATE.
*
     EXEC CICS RETURN TRANSID('ABCD')
         COMMAREA(COMMUNICATION-AREA)
     END-EXEC.
     GOBACK.
 2000-00-PROCESS.
     EXEC CICS RECEIVE MAP('TMAP01')
                       MAPSET('TMAPSET')
                       INTO(TMAP01I)
     END-EXEC.
     IF DPTONOL = ZERO OR DPTONOI = SPACE
        MOVE 'N' TO DATA-IS
```

```
           MOVE 'ENTER A VALID DEPARTMENT NUMBER' TO MSGLINEO
        END-IF.
        IF DATA-IS-O
           MOVE DPTONOI TO WORKDEPT-HV
           PERFORM 2000-01-DB2
        END-IF.
        IF DATA-IS-O
           SET SEND-IND-DATAO TO TRUE
           PERFORM 2000-10-SEND
        ELSE
           SET SEND-IND-ALARM TO TRUE
           PERFORM 2000-10-SEND
        END-IF.
*
      2000-01-DB2.
           EXEC SQL SELECT CHAR(DECIMAL(SUM(SALARY),9,2))
                INTO :SALARY-HV :SALARY-IN
                FROM DSN8810.EMP
                WHERE WORKDEPT=:WORKDEPT-HV END-EXEC.
        IF SQLCODE = 0
        THEN
           IF SALARY-IN = -1
           THEN
             MOVE 'N' TO DATA-IS
             MOVE 'NO EMPLOYEES EXIST IN THIS DEPARTMENT' TO MSGLINEO
             MOVE SPACES TO AVGSALO
           ELSE
             MOVE SALARY-HV TO AVGSALO
             MOVE SPACES TO MSGLINEO
           END-IF
        ELSE
           MOVE 'O' TO DATA-IS
           MOVE SPACES TO AVGSALO
           MOVE 'SQLSTATE' TO MSGSQLC
           MOVE SQLSTATE TO MSGREST
           MOVE MSGLINET TO MSGLINEO
        END-IF.
*
       2000-10-SEND.
        EVALUATE TRUE
        WHEN SEND-IND-ERASE
          EXEC CICS SEND MAP('TMAPO1')
               MAPSET('TMAPSET')
               FROM (TMAPO10)
               ERASE
          END-EXEC
        WHEN SEND-IND-DATAO
          EXEC CICS SEND MAP('TMAPO1')
               MAPSET('TMAPSET')
```

600

```
            FROM (TMAP010)
            DATAONLY
      END-EXEC
    WHEN SEND-IND-ALARM
      EXEC CICS SEND MAP('TMAP01')
           MAPSET('TMAPSET')
           FROM (TMAP010)
           DATAONLY
           ALARM
      END-EXEC
    END-EVALUATE.
```

准备作业

编译和链接-编辑Map

该作业放在LUISM.TEST.SAMPLIB库的MAPASSEM成员中。作业中调用的2个过程均放在SYS1.PROCLIB中。

```
//LUISM01 JOB (999,POK),'BMS Compilation',
//             CLASS=A,MSGCLASS=T,MSGLEVEL=(1,1)
//*********************************************************************
//*        ASSEMBLE MAP SET                                           *
//*********************************************************************
//STEP01  EXEC PROC=DFHASMV1,PARM.ASSEM='SYSPARM(MAP)'
//SYSLIN   DD DSN=LUISM.OBJET0,DCB=(LRECL=80),
//         SPACE=(2960,(10,10)),UNIT=SYSDA,DISP=(NEW,PASS)
//SYSIN    DD DSN=LUISM.TEST.SAMPLIB(TMAP01),DISP=SHR
/*
//*********************************************************************
//*        LINK EDIT                                                  *
//*********************************************************************
//STEP02  EXEC PROC=DFHLNKV1,PARM='LIST,LET,XREF'
//SYSLIN  DD DSN=LUISM.OBJET0,DISP=(OLD,DELETE)
//        DD *
          MODE RMODE(ANY)
          NAME TMAPSET(R)
/*
```

生成Map Copy文件

该作业放在LUISM.TEST.SAMPLIB库的MAPCOPYGM成员中。


```
//LUISM02 JOB (999,POK),'BMS COPY',
//             CLASS=A,MSGCLASS=T,MSGLEVEL=(1,1)
//*********************************************************************
//*        MAP COPY GENERATION                                        *
//*********************************************************************
//STEP01  EXEC PROC=DFHASMV1,PARM.ASSEM='SYSPARM(DSECT)'
//SYSLIN   DD DSN=LUISM.TEST.SAMPLIB(MAPCOPY),DISP=OLD
//SYSIN    DD DSN=LUISM.TEST.SAMPLIB(TMAP01),DISP=SHR
/*
```

准备程序

该作业位于LUISM.TEST.SAMPLIB库的CICSDB2P成员中。

```
//LUISM03 JOB (999,POK),'Cobol-CICS-DB2',
//             CLASS=A,MSGCLASS=T,MSGLEVEL=(1,1)
//*********************************************************************
//*  DB2 precompile, CICS translation, COBOL compile, pre-link,      *
//*  and link edit. Also DB2 Bind.                                   *
//*********************************************************************
//*********************************************************************
//*        DB2 Precompile                                           *
//*********************************************************************
//PC       EXEC  PGM=DSNHPC,PARM='HOST(IBMCOB)'
//SYSIN    DD  DSN=LUISM.TEST.SAMPLIB(XYZ2),DISP=SHR
//DBRMLIB  DD  DISP=SHR,
//             DSN=DB8HU.DBRMLIB.DATA(XYZ2)
//STEPLIB  DD  DISP=SHR,DSN=DB8H8.SDSNEXIT
//         DD  DISP=SHR,DSN=DB8H8.SDSNLOAD
//SYSCIN   DD  DSN=&&DSNHOUT,DISP=(MOD,PASS),UNIT=SYSDA,
//             SPACE=(800,(500,500))
//SYSLIB   DD  DISP=SHR,DSN=DB8HU.SRCLIB.DATA
//SYSPRINT DD  SYSOUT=*
//SYSTERM  DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSUT1   DD  SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//SYSUT2   DD  SPACE=(800,(500,500),,,ROUND),UNIT=VIO
/*
//*********************************************************************
//*        CICS Translator                                          *
//*********************************************************************
//TRN    EXEC PGM=DFHECP1$,
//            COND=(4,LT,PC)
//STEPLIB  DD DSN=CICSTS23.CICS.SDFHLOAD,DISP=SHR
```

```
//SYSPRINT DD SYSOUT=*
//SYSPUNCH DD DSN=&&SYSCIN,
//            DISP=(MOD,PASS),UNIT=SYSDA,
//            DCB=BLKSIZE=400,
//            SPACE=(400,(400,100))
//SYSUDUMP DD  SYSOUT=*
//SYSIN    DD  DSN=&&DSNHOUT,DISP=(OLD,DELETE)
//*
//*******************************************************************
//*        Compile                                                  *
//*******************************************************************
//COB      EXEC PGM=IGYCRCTL,
//             PARM=(NOSEQUENCE,QUOTE,RENT,'PGMNAME(LONGUPPER)'),
//             COND=(4,LT,TRN)
//SYSPRINT DD  SYSOUT=*
//SYSLIB   DD  DSN=CICSTS23.CICS.SDFHCOB,DISP=SHR
//         DD  DSN=CICSTS23.CICS.SDFHMAC,DISP=SHR
//         DD  DSN=CICSTS23.CICS.SDFHSAMP,DISP=SHR
//         DD  DSN=LUISM.TEST.SAMPLIB,DISP=SHR
//SYSTERM  DD  SYSOUT=*
//SYSLIN   DD  DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=VIO,
//             SPACE=(800,(500,500))
//SYSIN    DD  DSN=&&SYSCIN,DISP=(OLD,DELETE)
//SYSUT1   DD  SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//SYSUT2   DD  SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//SYSUT3   DD  SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//SYSUT4   DD  SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//SYSUT5   DD  SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//SYSUT6   DD  SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//SYSUT7   DD  SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//*******************************************************************
//*        Prelink                                                  *
//*******************************************************************
//PLKED   EXEC PGM=EDCPRLK,COND=(4,LT,COB)
//STEPLIB  DD  DISP=SHR,DSN=CEE.SCEERUN
//SYSMSGS  DD  DISP=SHR,
//             DSN=CEE.SCEEMSGP(EDCPMSGE)
//SYSIN    DD  DSN=&&LOADSET,DISP=(OLD,DELETE)
//SYSMOD   DD  DSN=&&PLKSET,UNIT=SYSDA,DISP=(MOD,PASS),
//             SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSDEFSD DD  DUMMY
//SYSOUT   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSTERM  DD  SYSOUT=*
//*******************************************************************
//*        Linkedit                                                 *
//*******************************************************************
//LKED     EXEC PGM=IEWL,PARM='LIST,XREF',
```

603

```
//         COND=(4,LT,PLKED)
//SYSLIB   DD  DISP=SHR,DSN=CEE.SCEELKED
//         DD  DISP=SHR,DSN=DB8H8.SDSNLOAD
//         DD  DISP=SHR,DSN=CICSTS23.CICS.SDFHLOAD
//         DD  DISP=SHR,DSN=ISP.SISPLOAD
//         DD  DISP=SHR,DSN=GDDM.SADMMOD
//SYSLMOD  DD  DSN=CICSTS23.CICS.SDFHLOAD(XYZ2),
//             DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSUT1   DD  SPACE=(1024,(50,50)),UNIT=VIO
//SYSLIN   DD  DSN=&&PLKSET,DISP=(OLD,DELETE)
//         DD  DDNAME=SYSIN
//CICSLOAD DD  DSN=CICSTS23.CICS.SDFHLOAD,
//             DISP=SHR
//SYSIN    DD  *
          INCLUDE CICSLOAD(DSNCLI)
          MODE RMODE(ANY)
          NAME XYZ2(R)
/*
//*********************************************************************
//*        Bind                                                       *
//*********************************************************************
//BIND     EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT,LKED)
//STEPLIB  DD DSN=DB8H8.SDSNLOAD,DISP=SHR
//DBRMLIB  DD DSN=DB8HU.DBRMLIB.DATA,DISP=SHR
//SYSUDUMP DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
   GRANT BIND, EXECUTE ON PLAN XYZP TO PUBLIC;
//SYSTSIN  DD *
DSN SYSTEM(DB8H)
BIND PACKAGE (DSN8CC81) MEMBER(XYZ2) -
     ACT(REP) ISO(CS) ENCODING(EBCDIC)
BIND PLAN(XYZP)  PKLIST(DSN8CC81.*) -
     ACT(REP) ISO(CS) ENCODING(EBCDIC)
RUN  PROGRAM(DSNTIAD) PLAN(DSNTIA81) -
     LIB('DB8HU.RUNLIB.LOAD')
END
/*
```

CICS定义

所有的CICS资源都通过CEDA交易程序在线定义。相应的组为PAZSGROU。

Resource	Tipo
ABCD	Transaction
XYZ2	Program
TMAPSET	Program (map)
TMAPSET	Mapset
XYZE	DB2 entry (correlates ABCD transaction and XYZP planname)

程序执行

在CICS屏幕上输入ABCD并按回车。然后，输入一个部门编号并按回车。结束后，按PF3键。

```
ABCD txid                Average salary by department

 Type a department number and press Enter.

 Department number: ___

 Average salary($):

 F3=Exit
```

COBOL-Batch-VSAM 程序

程序代码

该程序放在LUISM.TEST.SAMPLIB库的XYZ3成员中。

```
 IDENTIFICATION DIVISION.
*-------------------------------------------------------------
*    COBOL VSAM PROGRAM ZSCHOLAR RESIDENCY
*-------------------------------------------------------------
*-----------------------
 PROGRAM-ID.    XYZ3.
/
 ENVIRONMENT DIVISION.
*--------------------
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT I-FILE
      ASSIGN TO KSDATA
      ORGANIZATION IS INDEXED
      ACCESS IS DYNAMIC
      RECORD KEY IS I-FILE-RECORD-KEY
```

```
           ALTERNATE RECORD KEY IS I-FILE-ALTREC-KEY
           FILE STATUS IS FSTAT-CODE VSAM-CODE.
          SELECT DPTONO
           ASSIGN TO SYSIN
           ORGANIZATION IS SEQUENTIAL
           ACCESS IS SEQUENTIAL
            FILE STATUS IS DPTONO-CODE.
       DATA DIVISION.
      *-------------
       FILE SECTION.
       FD I-FILE
           RECORD CONTAINS 101 CHARACTERS.
         01 I-FILE-RECORD.
           05 I-FILE-RECORD-KEY           PIC X(6).
           05 FILLER                      PIC X(32).
           05 I-FILE-ALTREC-KEY           PIC X(3).
           05 FILLER                      PIC X(42).
           05 SALARY                      PIC S9(7)V9(2) COMP-3.
           05 FILLER                      PIC X(13).
       FD DPTONO
           RECORDING MODE F
           BLOCK 0 RECORDS
           RECORD 80 CHARACTERS
           LABEL RECORD STANDARD.
         01 DPTONO-RECORD     PIC X(80).
      /
       WORKING-STORAGE SECTION.
       01 STATUS-AREA.
         05 FSTAT-CODE                   PIC X(2).
           88 I-O-OKAY                   VALUE ZEROES.
         05 VSAM-CODE.
           10 VSAM-R15-RETURN-CODE       PIC 9(2) COMP.
            10 VSAM-FUNCTION-CODE        PIC 9(1) COMP.
            10 VSAM-FEEDBACK-CODE        PIC 9(3) COMP.
       77 DPTONO-CODE                    PIC XX.
       01 WS-DPTONO-RECORD.
         05 DPTONO-KEYED       PIC X(3).
         05 FILLER             PIC X(77).
       01 WS-SALARY            PIC S9(7)V9(2) COMP-3 VALUE 0.
       01 WS-SALARY-EDITED     PIC $ZZ,ZZZ,ZZ9.99.
      /
       PROCEDURE DIVISION.
           OPEN INPUT DPTONO.
           READ DPTONO INTO WS-DPTONO-RECORD.
           DISPLAY DPTONO-KEYED.
           OPEN INPUT I-FILE.
           IF FSTAT-CODE NOT = "00"
606          DISPLAY "OPEN INPUT VSAMFILE FS-CODE: " FSTAT-CODE
             PERFORM VSAM-CODE-DISPLAY
```

```
      STOP RUN
    END-IF.
    MOVE DPTONO-KEYED TO I-FILE-ALTREC-KEY.
    PERFORM READ-FIRST.
    IF FSTAT-CODE = "02"
      PERFORM READ-NEXT UNTIL FSTAT-CODE = "00"
    END-IF.
    IF FSTAT-CODE = "23"
      DISPLAY "NO RECORDS EXISTS FOR THIS DEPARTMENT"
    END-IF.
    MOVE WS-SALARY TO WS-SALARY-EDITED.
    DISPLAY WS-SALARY-EDITED.
    CLOSE DPTONO.
    CLOSE I-FILE.
    STOP RUN.

READ-NEXT.
    READ I-FILE NEXT.
    IF FSTAT-CODE NOT = "00" AND FSTAT-CODE NOT = "02"
       DISPLAY "READ NEXT I-FILE FS-CODE: " FSTAT-CODE
       PERFORM VSAM-CODE-DISPLAY
    ELSE
       ADD SALARY TO WS-SALARY
    END-IF.

READ-FIRST.
    READ I-FILE RECORD KEY IS I-FILE-ALTREC-KEY.
    IF FSTAT-CODE NOT = "00" AND FSTAT-CODE NOT = "02"
       DISPLAY "READ I-FILE FS-CODE: " FSTAT-CODE
       PERFORM VSAM-CODE-DISPLAY
    ELSE
       ADD SALARY TO WS-SALARY
    END-IF.

VSAM-CODE-DISPLAY.
      DISPLAY "VSAM CODE -->"
              " RETURN: " VSAM-R15-RETURN-CODE,
              " COMPONENT: " VSAM-FUNCTION-CODE,
              " REASON: " VSAM-FEEDBACK-CODE.
```

准备作业

创建VSAM环境

该作业位于LUISM.TEST.SAMPLIB库的VSAMDEF成员中。
作业执行了如下作业步：

- 将员工DB2表卸载到一个顺序文件中。
- 删除/定义VSAM KSDS文件。
- 定义辅助索引(alternative index)(通过部门编号)。
- 定义路径。
- 从顺序文件(步骤1)对VSAM文件进行repro操作。
- 完成BLDINDEX。

```
//LUISM06 JOB (999,POK),'UNLTAB/DEFVSAM/REPRO',
//             CLASS=A,MSGCLASS=T,MSGLEVEL=(1,1)
//*****************************************************************
//*        UNLOAD DB2 TABLE                                       *
//*****************************************************************
//STEP00   EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD DSN=DB8H8.SDSNLOAD,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSREC00 DD DSN=LUISM.EMP.TABLE.UNLOAD,
//            SPACE=(TRK,(1,1)),DISP=(,CATLG)
//SYSPUNCH DD DSN=LUISM.EMP.TABLE.SYSPUNCH,
//            SPACE=(TRK,(1,1)),DISP=(,CATLG),
//            RECFM=FB,LRECL=120
//SYSIN    DD *
DSN8810.EMP
/*
//SYSTSIN  DD *
DSN SYSTEM(DB8H)
RUN  PROGRAM(DSNTIAUL) PLAN(DSNTIB81) -
     LIB('DB8HU.RUNLIB.LOAD')
END
/*
//*****************************************************************
//*        DELETE THE KSDS FILE                                   *
//*****************************************************************
//STEP01   EXEC PGM=IDCAMS,COND=(4,LT,STEP00)
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD *
 DELETE LUISM.KSDATA
/*
//*****************************************************************
//*        DEFINE A KSDS FILE                                     *
//*****************************************************************
//STEP02   EXEC PGM=IEFBR14,COND=(4,LT,STEP01)
//DEFINE   DD  DSN=LUISM.KSDATA,DISP=(NEW,KEEP),
//             SPACE=(TRK,(1,1)),AVGREC=U,RECORG=KS,
//             KEYLEN=6,KEYOFF=0,LRECL=101
/*
//*****************************************************************
```

608

```
//*        DEFINE ALTERNATE INDEX                                      *
//*********************************************************************
//STEP03   EXEC PGM=IDCAMS,COND=(4,LT,STEP02)
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD *
 DEFINE ALTERNATEINDEX                    -
       (NAME(LUISM.ALTINDEX)              -
        RELATE(LUISM.KSDATA)              -
       NONUNIQUEKEY                      -
        KEYS(3 38)                        -
        RECORDSIZE(23 150)                -
        VOLUMES(TOTSSI)                   -
        KILOBYTES(100 100)                -
        UPGRADE)
/*
//*********************************************************************
//*        DEFINE PATH                                                 *
//*********************************************************************
//STEP04   EXEC PGM=IDCAMS,COND=(4,LT,STEP03)
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD *
 DEFINE PATH                              -
       (NAME(LUISM.PATH)                  -
        PATHENTRY(LUISM.ALTINDEX))
/*
//*********************************************************************
//*        REPRO INTO THE KSDS FROM DB2 UNLOAD SEQ FILE                *
//*********************************************************************
//STEP05   EXEC PGM=IDCAMS,COND=(4,LT,STEP04)
//SEQFILE  DD  DSN=LUISM.EMP.TABLE.UNLOAD,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD *
 REPRO INFILE(SEQFILE)                    -
       OUTDATASET(LUISM.KSDATA)           -
       REPLACE
/*
//*********************************************************************
//*        BLDINDEX                                                    *
//*********************************************************************
//STEP06   EXEC PGM=IDCAMS,COND=(4,LT,STEP05)
//BASEDD   DD  DSN=LUISM.KSDATA,DISP=SHR
//AIXDD    DD  DSN=LUISM.ALTINDEX,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD *
 BLDINDEX INFILE(BASEDD)                   -
          OUTFILE(AIXDD)           -
          SORTCALL
/*
```

准备程序

该作业位于LUISM.TEST.SAMPLIB库的BATVSAMP成员中。

```
//LUISM07 JOB (999,POK),'Cobol-VSAM',
//             CLASS=A,MSGCLASS=T,MSGLEVEL=(1,1)
//*******************************************************************
//*        Compile the IBM COBOL program                            *
//*******************************************************************
//COB      EXEC PGM=IGYCRCTL,
//             PARM=(NOSEQUENCE,QUOTE,RENT,'PGMNAME(LONGUPPER)')
//SYSPRINT DD  SYSOUT=*
//SYSTERM  DD  SYSOUT=*
//SYSLIN   DD  DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=VIO,
//             SPACE=(800,(500,500))
//SYSIN    DD  DSN=LUISM.TEST.SAMPLIB(XYZ3),DISP=SHR
//SYSUT1   DD  SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//SYSUT2   DD  SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//SYSUT3   DD  SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//SYSUT4   DD  SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//SYSUT5   DD  SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//SYSUT6   DD  SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//SYSUT7   DD  SPACE=(800,(500,500),,,ROUND),UNIT=VIO
//*******************************************************************
//*  PRELINK STEP.                                                  *
//*******************************************************************
//PLKED   EXEC PGM=EDCPRLK,COND=(4,LT,COB)
//STEPLIB  DD  DISP=SHR,DSN=CEE.SCEERUN
//SYSMSGS  DD  DISP=SHR,
//             DSN=CEE.SCEEMSGP(EDCPMSGE)
//SYSIN    DD  DSN=&&LOADSET,DISP=(OLD,DELETE)
//SYSMOD   DD  DSN=&&PLKSET,UNIT=SYSDA,DISP=(MOD,PASS),
//             SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSDEFSD DD  DUMMY
//SYSOUT   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSTERM  DD  SYSOUT=*
//*******************************************************************
//*        Linkedit                                                 *
//*******************************************************************
//LKED     EXEC PGM=IEWL,PARM='LIST,XREF',
//         COND=(4,LT,PLKED)
//SYSLIB   DD  DISP=SHR,DSN=CEE.SCEELKED
//         DD  DISP=SHR,DSN=ISP.SISPLOAD
//         DD  DISP=SHR,DSN=GDDM.SADMMOD
//SYSLMOD  DD  DSN=LUISM.TEST.LOADLIB(XYZ3),
//             DISP=SHR


//SYSPRINT DD  SYSOUT=*
//SYSUT1   DD  SPACE=(1024,(50,50)),UNIT=VIO
//SYSLIN   DD  DSN=&&PLKSET,DISP=(OLD,DELETE)
//         DD  DDNAME=SYSIN
//SYSIN    DD  *
           MODE RMODE(ANY)
          NAME XYZ3(R)
/*
```

程序执行

该作业位于LUISM.TEST.SAMPLIB库的RUNXYZ3成员中。

```
//LUISM08 JOB (999,POK),'EJEC. COB-VSAM',
//            CLASS=A,MSGCLASS=T,MSGLEVEL=(1,1)
//*****************************************************************
//STEP01   EXEC PGM=XYZ3
//STEPLIB  DD  DSN=LUISM.TEST.LOADLIB,DISP=SHR
//KSDATA   DD  DSN=LUISM.KSDATA,DISP=SHR
//KSDATA1  DD  DSN=LUISM.PATH,DISP=SHR
//OUTPUTFI DD  SYSOUT=*
//SYSIN    DD  *
E01
/*
```

下面的是部门E0的输出：

```
******************************** TOP OF DATA ***********************************
E01
$    40,175.00
******************************** BOTTOM OF DATA ********************************
```

对于没有员工的部门，输出结果如下：

```
******************************** TOP OF DATA ***********************************
A01
READ I-FILE FS-CODE: 23
VSAM CODE --> RETURN: 08 COMPONENT: 2 REASON: 016
NO RECORDS EXISTS FOR THIS DEPARTMENT
$         0.00
******************************** BOTTOM OF DATA ********************************
```


DSNTEP2 实用程序

本PL/I程序动态地执行了从SYSIN中输入的SQL语句。该应用程序也可以执行非SELECT语句。

执行作业

该执行作业位于LUISM.TEST.SAMPLIB库的DSNTEP2成员中。

```
//LUISM04 JOB (999,POK),'Dsntep2',
//            CLASS=A,MSGCLASS=T,MSGLEVEL=(1,1)
//*****************************************************************
//*        DSNTEP2                                                *
//*****************************************************************
//DSNTEP2  EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD DSN=DB8H8.SDSNLOAD,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
DSN SYSTEM(DB8H)
RUN  PROGRAM(DSNTEP2) PLAN(DSNTEP81) -
     LIB('DB8HU.RUNLIB.LOAD')
END
/*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSIN    DD *
    SELECT CHAR(DECIMAL(SUM(SALARY),9,2))
    FROM DSN8810.EMP
    WHERE WORKDEPT='A00'
/*
```

下面的是作业输出：

```
PAGE    1
***INPUT STATEMENT:
     SELECT CHAR(DECIMAL(SUM(SALARY),9,2))
     FROM DSN8810.EMP
     WHERE WORKDEPT='A00'
                                                    +-------------+
                                                    |             |
                                                    +-------------+
                                                  1_|  0204250.00 |
                                                    +-------------+
SUCCESSFUL RETRIEVAL OF        1 ROW(S)
```


QMF 批处理方式执行

该练习显示了批处理方式执行的QMF过程/查询/表单。EMPQRY查询包含了我们Class程序的SQL语句。EMPPRO过程调用执行了该查询模块，并打印了相关报告。该作业调用了QMF过程并将部门编号传递给它；同时也指定了执行模式(批处理方式，M=B)以及DB2子系统。

作业位于LUISM.TEST.SAMPLIB库的QMFBATCH成员中。

QMF通过SC47TS系统中ISPF选项Q7，COMMAND域为ISPQMF71调用。

执行作业

```
//LUISM10 JOB (999,POK),'QMF in batch',
//            CLASS=A,MSGCLASS=T,MSGLEVEL=(1,1)
/*JOBPARM SYSAFF=SC47
//*********************************************************************
//QMFBAT   EXEC PGM=DSQQMFE,
//         PARM='M=B,I=LUISM.EMPPRO(&&DEP=''A00''),S=DB7D'
//STEPLIB  DD  DISP=SHR,DSN=DB7DU.SDSQLOAD
//         DD  DISP=SHR,DSN=DB7D7.SDSNLOAD
//         DD  DISP=SHR,DSN=DB7D7.SDSNEXIT
//ADMGGMAP DD  DSN=DB7DU.DSQMAPE,DISP=SHR
//DSQPRINT DD  SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=1330)
//DSQDEBUG DD  SYSOUT=A,DCB=(RECFM=FBA,LRECL=121,BLKSIZE=1210)
//DSQUDUMP DD  SYSOUT=A,DCB=(RECFM=VBA,LRECL=125,BLKSIZE=1632)
//DSQSPILL DD  DSN=&&SPILL,DISP=(NEW,DELETE),
//     UNIT=VIO,SPACE=(TRK,(100),RLSE),
//     DCB=(RECFM=F,LRECL=4096,BLKSIZE=4096)
//*
```

QMF过程

```
RUN QUERY EMPQRY (&&D=&DEP FORM=EMPFORM
PRINT REPORT
```

QMF查询

```
SELECT CHAR(DECIMAL(SUM(SALARY),9,2))
FROM DSN8710.EMP
WHERE WORKDEPT=&D
```


批处理 C 程序来访问 DB2

源代码

该程序位于GMULLER.TEST.C库的CDB2成员中。

示例E-1 访问DB2的C源代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

EXEC SQL INCLUDE SQLCA;
EXEC SQL INCLUDE SQLDA;

EXEC SQL
    DECLARE DSN8810.EMP TABLE
    (EMPNO     CHAR(6)     NOT NULL,
    FIRSTNAME VARCHAR(12) NOT NULL,
    MIDINIT   CHAR(1)     NOT NULL,
    LASTNAME  VARCHAR(15) NOT NULL,
    WORKDEPT  CHAR(3)             ,
    PHONENO   CHAR(4)             ,
    HIREDATE  DATE                ,
    JOB       CHAR(8)             ,
    EDLEVEL   SMALLINT            ,
    SEX       CHAR(1)             ,
    BIRTHDATE DATE                ,
    SALARY    DECIMAL(9,2)        ,
    BONUS     DECIMAL(9,2)        ,
    COMM      DECIMAL(9,2)        );

EXEC SQL BEGIN DECLARE SECTION;
  long sum;
  long count;
  char deptno[4];
EXEC SQL END DECLARE SECTION;

int avg_sal(char*);
int record_read(FILE*,char*);

void main()
{
  FILE* cardin; /* for DD card CARDIN */
  int avgsal;
  char dept[4];

614
  cardin = fopen("DD:CARDIN","rb,recfm=FB,lrecl=80,type=record");
  if(cardin == NULL)
```

```
  {
    printf("Error opening DD CARDIN\n");
    exit(-2);
  }

  while(record_read(cardin, dept) != 0)
  {
    avgsal = avg_sal(dept);
    if(avgsal > 0)
      printf("Average salary of %s is %d\n",dept, avgsal);

  }
  fclose(cardin);
}

int avg_sal(char* dept)
{
  int avgsal;
  count = 0;
  strncpy(deptno, dept, 3);
  deptno[3] = 0;

  EXEC SQL SELECT SUM(SALARY), COUNT(*) INTO :sum, :count
           FROM DSN8810.EMP
           WHERE WORKDEPT = :deptno;

  if(count != 0)
  {
    avgsal = sum/count;
    return avgsal;
  } else
  {
    printf("DEPT %s does not exist\n", deptno);
    return -1;
  }
}

int record_read(FILE* file, char* dept)
{
  int readbytes;
  char linebuf[81], linebuf2[80];
  readbytes = fread(linebuf, 1, 81, file);
  strncpy(dept, linebuf, 3); /* first 3 bytes are dept. number */
  dept[3]=0; /* terminate string */
  return readbytes;
}
```

准备程序

该JCL位于GMULLER.TEST.CNTL库的CDB2成员中。

示例E-2 GMULLER.TEST.CNTL(CDB2)

```
//GMULLERC JOB 1,GEORG,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//* PRECOMPILE AND COMPILE THE SAMPLE C FILE
//PROCLIB JCLLIB ORDER=DB8HU.PROCLIB
/*JOBPARM SYSAFF=SC04
//STEP1 EXEC PROC=DSNHC,MEM=CDB2,
//      PARM.PC=('HOST(C),CCSID(1047)')
//PC.DBRMLIB DD DSN=DB8HU.DBRMLIB.DATA(CDB2),DISP=SHR
//PC.SYSLIB  DD DSN=GMULLER.TEST.C,DISP=SHR
//PC.SYSIN   DD DSN=GMULLER.TEST.C(&MEM),DISP=SHR
//LKED.SYSLMOD DD DSN=GMULLER.TEST.LOAD(&MEM),DISP=SHR
//LKED.SYSIN DD *
 INCLUDE SYSLIB(DSNELI)
/*
//*************************************************
//* BIND AND RUN THE PROGRAM                      *
//*************************************************
//BIND     EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//STEPLIB  DD DSN=DB8H8.SDSNLOAD,DISP=SHR
//DBRMLIB  DD DSN=DB8HU.DBRMLIB.DATA,DISP=SHR
//SYSUDUMP DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//CARDIN   DD *
D11
XYZ
A00
/*
//SYSIN    DD *
   GRANT BIND,EXECUTE ON PLAN CDB2 TO PUBLIC;
//SYSTSIN  DD *
 DSN SYSTEM(DB8H)
 BIND PACKAGE (CDB2PAK) MEMBER(CDB2) -
      ACT(REP) ISO(CS) ENCODING(EBCDIC)
 BIND PLAN(CDB2)  PKLIST(CDB2PAK.*) -
      ACT(REP) ISO(CS) ENCODING(EBCDIC)
 RUN  PROGRAM(CDB2) PLAN(CDB2) LIB('GMULLER.TEST.LOAD')
 END
/*
```

- 该作业需要RECFM=U的分区数据集GMULLER.TEST.LOAD。
- 语句"/*JOBPARM SYSAFF=SC04"指向DB2运行的系统，必须相应更改(或者删除，前提是不在一个系统综合体中)。

- DB8H必须用本地DB2名字替代。
- DB2库的HLQ可能不同

输出

示例E-3 CDB2的输出

```
Average salary of D11 is 25147
DEPT XYZ does not exist
Average salary of A00 is 40850
```

运行程序

该JCL位于GMULLER.TEST.CNTL库的RUNJCL成员中。

示例E-4 GMULLER.TEST.CNTL(RUNJCL)

```
//GMULLERR JOB 1,GEORG,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//* PRECOMPILE AND COMPILE THE SAMPLE C FILE
/*JOBPARM SYSAFF=SC04
//*************************************************
//* RUN THE PROGRAM                               *
//*************************************************
//BIND     EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//STEPLIB  DD DSN=DB8H8.SDSNLOAD,DISP=SHR
//DBRMLIB  DD DSN=DB8HU.DBRMLIB.DATA,DISP=SHR
//SYSUDUMP DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//CARDIN   DD DISP=SHR,DSN=GMULLER.TEST.CNTL(CARDIN)
//SYSTSIN  DD *
 DSN SYSTEM(DB8H)
 RUN  PROGRAM(CDB2) PLAN(CDB2) LIB('GMULLER.TEST.LOAD')
 END
/*
```

- 该作业需要GMULLER.TEST.CNTL库的CARDIN成员。
- DB2库的HLQ可能不同。

输入

示例E-5 GMULLER.TEST.CNTL(CARDIN)

```
D11
A00
XYZ
C01
ABC
E21
```


输出

示例E-6 RUNJCL的输出

```
Average salary of D11 is 25147
Average salary of A00 is 40850
DEPT XYZ does not exist
Average salary of C01 is 29722
DEPT ABC does not exist
Average salary of E21 is 24086
```

通过 Java Servlet 访问 DB2

Servlet源代码

示例E-7 SalaryServlet.java

```
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.sql.DataSource;

public class SalaryServlet extends HttpServlet {

    private DataSource ds;
    private boolean dbProblem = false;

    public void init() throws ServletException {
        super.init();
        try { // get DataSource from Container
```

618

```
            Context context = new InitialContext();
            ds = (DataSource) context.lookup("jdbc/DB8H");
        } catch (NamingException e) {
            e.printStackTrace();
            this.dbProblem = true;
        }
    }

    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {

        resp.setContentType("text/html");
        String deptno = req.getParameter("deptno"); // get from request string

        PrintWriter out = resp.getWriter();
        out.println("<html>\n<head>\n <title>Average
Salary</title>\n</head>\n<body>");
        out.println("<h1>Average Salary</h1>");
        out.println("<form action=\"salary\" method=\"get\">");
        out.println("Dept. No.: <input type=\"text\" name=\"deptno\" />");
        out.println(" <input type=\"submit\" />\n</form>");

        if (deptno != null) {
            try {
                int avgSal = getAvgSal(deptno);
                out.println("The average salary of <b>" + deptno + "</b> is
<b>$ " + avgSal
                        + "</b><br>");
            } catch (Exception e) {
                out.println("<b>Error: " + e.getMessage() + "</b><br>");
            }
        }
        out.println("</html>");
    }

    private int getAvgSal(String deptno) throws Exception {
        String sqlStatement = "SELECT SUM(salary), COUNT(*) "
                + "FROM DSN8810.EMP WHERE WORKDEPT = '" + deptno + "'";
        // Connect to database
        Connection con = null;
        try {
            con = ds.getConnection();
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery(sqlStatement); // Execute SQL
            // statement

            rs.next(); // get Values from result set
            int sum = rs.getInt(1);
            int count = rs.getInt(2);
```

```
            if (count == 0)
                throw new Exception("Department " + deptno
                        + " does not exist");
            return sum / count;

        } catch (SQLException e) {
            throw new Exception(e.getMessage());
        } finally {
            try {
                con.close();
            } catch (SQLException e) {}
        }
    }
}
```

- 该Servlet需要一个在Web容器中定义的数据源(这里JNDI名称为"jdbc/DB8H")，它指向DB2数据库。

部署描述程序

示例E-8 web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app id="WebApp_ID">
    <display-name>Salary</display-name>
    <servlet>
        <servlet-name>Salary</servlet-name>
        <servlet-class>SalaryServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Salary</servlet-name>
        <url-pattern>/salary</url-pattern>
    </servlet-mapping>
</web-app>
```

C 程序访问 MQ

MQPUT在队列中写入一条消息(在TSO中输入)。

程序由TSO CALL 'ZSCHOLAR.PROGRAM.LOAD(MQPUT)'启动。然后您必须输入一条消息。

MQGET得到该消息并把它显示在屏幕上。

程序由TSO CALL 'ZSCHOLAR.PROGRAM.LOAD(MQGET)'启动，然后您必须输入一条消息。

也可以使用630页"Java程序访问MQ"中的Java程序接受消息。

MQPUT

示例E-9 ZSCHOLAR.PROGRAM.SRC(MQPUT)

```
#pragma csect(code,"CSQ4BCK1")
/*                                                                  */
/* Define static CSECT name                                         */
/*                                                                  */
#pragma csect(static,"BCK1WS")

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <cmqc.h>

/*                                                                  */
/* Function prototypes                                              */
/*                                                                  */
void usageError( char* programName );
void errorMessage( char* msgStr, MQLONG CC, MQLONG RC );

int main( int argc, char** argv )
{
   /*                                                               */
   /* API variables                                                 */
   /*                                                               */
   MQHCONN HConn = MQHC_DEF_HCONN;
   MQHOBJ  HObj;
   MQLONG  OpenOptions;
   MQMD    MsgDesc = { MQMD_DEFAULT };
   MQOD    ObjDesc = { MQOD_DEFAULT };
   MQPMO   PutMsgOpts = { MQPMO_DEFAULT };
   MQLONG  CompCode;
   MQLONG  Reason;

   /*                                                               */
   /* Parameter variables                                           */
   /*                                                               */
   MQCHAR48 qMgr;
   MQCHAR48 qName;
```

```
char     msgBuffer[255];
int      msgLength;
char     persistent = 'N';
long     rc = 0;

printf("Please enter message text:\n");
fgets(msgBuffer, 255, stdin);
msgLength = strlen(msgBuffer);

strcpy( qMgr,  "MQ8H\0" );
strcpy( qName, "GMULLER\0" );
/*
memset( qMgr,  '\0', MQ_Q_MGR_NAME_LENGTH );
memset( qName, '\0', MQ_Q_NAME_LENGTH );
*/

/*                                                                  */
/* Connect to Queue Manager (MQCONN)                                */
/*                                                                  */
MQCONN( qMgr,
        &HConn,
        &CompCode,
        &Reason     );
/*                                                                  */
/* If connect failed then display error message and exit            */
/*                                                                  */
if( MQCC_OK != CompCode )
   {
   errorMessage( "MQCONN", CompCode, Reason );
   return Reason;
   }

printf( "MQCONN SUCCESSFUL\n" );

/*                                                                  */
/* Open Queue for output (MQOPEN). Fail the call if the queue       */
/* manager is quiescing.                                            */
/*                                                                  */
OpenOptions = MQOO_OUTPUT +
              MQOO_FAIL_IF_QUIESCING;

strncpy( ObjDesc.ObjectName, qName, MQ_Q_NAME_LENGTH );
MQOPEN( HConn,
        &ObjDesc,
        OpenOptions,
        &HObj,
        &CompCode,
        &Reason     );
/*                                                                  */
```

```
/* If open failed then display error message,                    */
/* disconnect from the queue manager and exit                    */
/*                                                               */
if( MQCC_OK != CompCode )
   {
   errorMessage( "MQOPEN", CompCode, Reason );
   rc = Reason;
   MQDISC( &HConn,
           &CompCode,
           &Reason    );
   return rc;
   }

printf( "MQOPEN SUCCESSFUL\n" );

/*                                                               */
/* Set persistence depending on parameter passed                 */
/*                                                               */
if( 'P' == persistent )
   MsgDesc.Persistence = MQPER_PERSISTENT;
else
   MsgDesc.Persistence = MQPER_NOT_PERSISTENT;

/*                                                               */
/* Put String format messages                                    */
/*                                                               */
strncpy( MsgDesc.Format, MQFMT_STRING, MQ_FORMAT_LENGTH );

/*                                                               */
/* Set the put message options to fail the call if the queue     */
/* manager is quiescing.                                         */
/*                                                               */
PutMsgOpts.Options = MQPMO_FAIL_IF_QUIESCING;


strncpy( MsgDesc.MsgId,    MQMI_NONE, MQ_MSG_ID_LENGTH );
strncpy( MsgDesc.CorrelId, MQCI_NONE, MQ_CORREL_ID_LENGTH );

MQPUT( HConn,
       HObj,
       &MsgDesc,
       &PutMsgOpts,
       msgLength,
       msgBuffer,
       &CompCode,
       &Reason    );
/*
/* If put failed then display error message                   */
/* and break out of loop                                      */
```

```
   /*                                                                  */
   if( MQCC_OK != CompCode )
      {
      errorMessage( "MQPUT", CompCode, Reason );
      rc = Reason;
      }

   printf("MESSAGE PUT TO QUEUE\n");

   free( msgBuffer );

   /*                                                                  */
   /* Close the queue and then disconnect from the queue manager       */
   /*                                                                  */
   MQCLOSE( HConn,
            &HObj,
            MQCO_NONE,
            &CompCode,
            &Reason    );
   if( MQCC_OK != CompCode  )
      {
      errorMessage( "MQCLOSE", CompCode, Reason );
      rc = Reason;
      }
   else printf( "MQCLOSE SUCCESSFUL\n" );


   MQDISC( &HConn,
           &CompCode,
           &Reason    );
   if( MQCC_OK != CompCode )
      {
      errorMessage( "MQDISC", CompCode, Reason );
      return Reason;
      }
   else
      {
      printf( "MQDISC SUCCESSFUL\n" );
      return rc;
      }

   return(rc);
} /*end main*/

/********************************************************************/
/* Functions to display error messages                              */
/********************************************************************/
void errorMessage( char* msgStr, MQLONG CC, MQLONG RC )
{

   printf( "************************************************\n" );
   printf( "* %s\n", msgStr );
   printf( "* COMPLETION CODE : %09ld\n", CC );
   printf( "* REASON CODE     : %09ld\n", RC );
   printf( "************************************************\n" );
}
```

下面的JCL用于编译：

示例E-10 ZSCHOLAR.PROGRAM.CNTL(MQPUT)

```
//GMULLERT JOB 1,GEORG,MSGCLASS=H,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//* COMPILE MQ PROGRAM
//STEP1    EXEC PROC=EDCCB,
//            INFILE='ZSCHOLAR.PROGRAM.SRC(MQPUT)',
//            OUTFILE='ZSCHOLAR.PROGRAM.LOAD(MQPUT),DISP=SHR'
//SYSLIB   DD DSN=MQ531.SCSQC370,DISP=SHR
//BIND.CSQBSTUB  DD DSN=MQ531.SCSQLOAD(CSQBSTUB),DISP=SHR
//BIND.SYSIN DD *
  INCLUDE CSQBSTUB
/*
```

MQGET

源代码

示例E-11 ZSCHOLAR.PROGRAM.SRC(MQGET)

```
#pragma csect(code,"CSQ4BCK1")
/*                                                                 */
/* Define static CSECT name                                        */
/*                                                                 */
#pragma csect(static,"BCK1WS")

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <cmqc.h>

#define maxMessageLength  65536

/*                                                                 */
/* Function prototypes                                             */
/*                                                                 */
void usageError( char* programName );
void errorMessage( char* msgStr, MQLONG CC, MQLONG RC );

int main( int argc, char** argv )
```

```
{
   /*                                                               */
   /* API variables                                                 */
   /*                                                               */
   MQHCONN HConn = MQHC_DEF_HCONN;
   MQHOBJ  HObj;
   MQLONG  OpenOptions;
   MQMD    MsgDesc = { MQMD_DEFAULT };
   MQOD    ObjDesc = { MQOD_DEFAULT };
   MQGMO   GetMsgOpts = { MQGMO_DEFAULT };
   MQLONG  CompCode;
   MQLONG  Reason;

   /*                                                               */
   /* Parameter variables                                           */
   /*                                                               */
   MQCHAR48 qMgr;
   MQCHAR48 qName;
   char     msgBuffer[maxMessageLength];
   int      msgLength = maxMessageLength;
   char     persistent = 'N';
   long     rc = 0;
   long     dataLength;
   char     browseGet = 'D'; /* destructive get */
   char     syncpoint = 'N'; /* no Syncpoint */

   memset( msgBuffer, '\0', msgLength );

   strcpy( qMgr,  "MQ8H\0" );
   strcpy( qName, "GMULLER\0" );

   /*                                                               */
   /* Connect to Queue Manager (MQCONN)                             */
   /*                                                               */
   MQCONN( qMgr,
           &HConn,
           &CompCode,
           &Reason     );
   /*                                                               */
   /* If connect failed then display error message and exit         */
   /*                                                               */
   if( MQCC_OK != CompCode )
      {
      errorMessage( "MQCONN", CompCode, Reason );
      return Reason;
      }

   printf( "MQCONN SUCCESSFUL\n" );
```

626

```
/*                                                                  */
/* Open Queue for input shared and browse. Fail the call if the     */
/* queue manager is quiescing.                                      */
/*                                                                  */
OpenOptions = MQOO_INPUT_SHARED +
              MQOO_BROWSE +
              MQOO_FAIL_IF_QUIESCING;

strncpy( ObjDesc.ObjectName, qName, MQ_Q_NAME_LENGTH );
MQOPEN( HConn,
        &ObjDesc,
        OpenOptions,
        &HObj,
        &CompCode,
        &Reason     );
/*                                                                  */
/* If open failed then display error message,                       */
/* disconnect from the queue manager and exit                       */
/*                                                                  */
if( MQCC_OK != CompCode )
   {
   errorMessage( "MQOPEN", CompCode, Reason );
   rc = Reason;
   MQDISC( &HConn,
           &CompCode,
           &Reason    );
   return rc;
   }

printf( "MQOPEN SUCCESSFUL\n" );

/*                                                                  */
/* Set persistence depending on parameter passed                    */
/*                                                                  */
if( 'P' == persistent )
   MsgDesc.Persistence = MQPER_PERSISTENT;
else
   MsgDesc.Persistence = MQPER_NOT_PERSISTENT;

/*                                                                  */
/* Set GetMsgOpts .. don't wait if there are no messages on the     */
/* queue, truncate the message if it does not fit into our          */
/* buffer, perform data conversion on the message if required       */
/* and if possible, and fail the call if the queue manager is       */
/* quiescing.                                                       */
/*                                                                  */
GetMsgOpts.Options = MQGMO_NO_WAIT +
                     MQGMO_ACCEPT_TRUNCATED_MSG +
                     MQGMO_CONVERT +
```

627

```
                    MQGMO_FAIL_IF_QUIESCING;

strncpy( MsgDesc.MsgId,    MQMI_NONE, MQ_MSG_ID_LENGTH );
strncpy( MsgDesc.CorrelId, MQCI_NONE, MQ_CORREL_ID_LENGTH );

/*
/* Set additional GetMsgOpts depending on parameters passed
/* into program.
/*
if( ('S' == syncpoint) && ('B' != browseGet) )
   GetMsgOpts.Options += MQGMO_SYNCPOINT;
else
   GetMsgOpts.Options += MQGMO_NO_SYNCPOINT;

if( ('B' == browseGet) )
   GetMsgOpts.Options += MQGMO_BROWSE_FIRST;

MsgDesc.Encoding = MQENC_NATIVE;
MsgDesc.CodedCharSetId = MQCCSI_Q_MGR;

/* GET */
MQGET( HConn,
       HObj,
       &MsgDesc,
       &GetMsgOpts,
       msgLength,
       msgBuffer,
       &dataLength,
       &CompCode,
       &Reason    );

if( (MQCC_FAILED == CompCode) )
   {
   errorMessage( "MQGET", CompCode, Reason );
   rc = Reason;
   }
else
   {
   /*                                                              */
   /* Only character data messages are correctly displayed         */
   /* by this code                                                 */
   /*                                                              */
   if (MQRC_TRUNCATED_MSG_ACCEPTED == Reason)
      {
      msgBuffer??( msgLength - 1 ??) = 0;
      printf( "Message received (truncated):\n%s\n",
             msgBuffer );
      }
   else
```

```
        {
        msgBuffer??( dataLength ??) = 0;
        printf( "Message received:\n%s\n",
               msgBuffer );
        }
     }

  free( msgBuffer );

  /*                                                               */
  /* Close the queue and then disconnect from the queue manager    */
  /*                                                               */
  MQCLOSE( HConn,
           &HObj,
           MQCO_NONE,
           &CompCode,
           &Reason    );

  if( MQCC_OK != CompCode  )
     {
     errorMessage( "MQCLOSE", CompCode, Reason );
     rc = Reason;
     }
  else printf( "MQCLOSE SUCCESSFUL\n" );


  MQDISC( &HConn,
          &CompCode,
          &Reason    );
  if( MQCC_OK != CompCode )
     {
     errorMessage( "MQDISC", CompCode, Reason );
     return Reason;
     }
  else
     {
     printf( "MQDISC SUCCESSFUL\n" );
     return rc;
     }

  return(rc);
} /*end main*/

/*****************************************************************/
/* Functions to display error messages                           */
/*****************************************************************/
void errorMessage( char* msgStr, MQLONG CC, MQLONG RC )
{
  printf( "***********************************************\n" );
  printf( "* %s\n", msgStr );
  printf( "* COMPLETION CODE : %09ld\n", CC );
  printf( "* REASON CODE     : %09ld\n", RC );
  printf( "***********************************************\n" );
}
```

629

用来编译的JCL如下

示例E-12 ZSCHOLAR.PROGRAM.CNTL(MQGET)

```
//GMULLERT JOB 1,GEORG,MSGCLASS=H,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//* COMPILE MQ PROGRAM
//STEP1    EXEC PROC=EDCCB,
//            INFILE='ZSCHOLAR.PROGRAM.SRC(MQGET)',
//            OUTFILE='ZSCHOLAR.PROGRAM.LOAD(MQGET),DISP=SHR'
//SYSLIB   DD DSN=MQ531.SCSQC370,DISP=SHR
//BIND.CSQBSTUB  DD DSN=MQ531.SCSQLOAD(CSQBSTUB),DISP=SHR
//BIND.SYSIN DD *
  INCLUDE CSQBSTUB
/*
```

Java 程序访问 MQ

Java程序从一个队列中接受一条消息。MessageHandle类也包含一个发送消息的类。
您必须把com.ibm.mq.jar和connector.jar加入到您的CLASSPATH中。
所有文件都在 “program sample\mq”中。
利用命令java -jar mqconnect.jar运行程序。

示例E-13 MQReceiver.java

```
import com.ibm.mq.MQException;

public class MQReceiver {

    public static void main(String[] args) {

        // Connection settings
        String hostname = "wtsc04.itso.ibm.com";
        String queueName = "GMULLER";
        int port = 1598; // mq port
        String channel = "GMULLER.SERV";
```

```
        MessageHandler handler = new MessageHandler(hostname, port, queueName,
                channel);

        String message;
        try {
            System.out.println("Sending message...");
            handler.sendMessage("Hello");
            //System.out.println("Receiving message...");
            //message = handler.receiveMessage();
            //System.out.println("Message: " + message);
            System.out.println("Finished");
        } catch (MQException e) {
            if (e.reasonCode == MQException.MQRC_NO_MSG_AVAILABLE)
                System.out.println("No message in queue");
            else {
                System.out.println("Error getting message");
                e.printStackTrace();
            }
        }
    }
}
```

示例E-14 MessageHandler.java

```
import java.io.IOException;

import com.ibm.mq.*;

public class MessageHandler {

    private String hostname;
    private String queueName;

    public MessageHandler(String hostname, int port, String queueName, String
channel) {
        MQEnvironment.hostname = hostname;
        MQEnvironment.port = port;
        MQEnvironment.channel = channel;
        this.queueName = queueName;
    }

    public String receiveMessage() throws MQException {
        try {
            MQQueueManager mqm = new MQQueueManager(hostname);

            int openOptions = MQC.MQOO_INPUT_AS_Q_DEF + MQC.MQOO_OUTPUT;
```

631

```
            MQQueue queue = mqm.accessQueue(queueName, openOptions);
            // create new Message for receiving
            MQMessage message = new MQMessage();

            // get message from queue
            queue.get(message);
            // get the whole message string
            String messageString =
message.readString(message.getMessageLength());
            // close queue;
            queue.close();
            // disconnect from queue manager
            mqm.disconnect();

            return messageString;

        } catch (IOException e) {
            e.printStackTrace();
            return null;
        }
    }

    public void sendMessage(String messageString) throws MQException {
        try {
            MQQueueManager mqm = new MQQueueManager(hostname);

            int openOptions = MQC.MQOO_INPUT_AS_Q_DEF + MQC.MQOO_OUTPUT;



            MQQueue queue = mqm.accessQueue(queueName, openOptions);
            // create new Message for receiving
            MQMessage message = new MQMessage();


            // write message
            message.writeString(messageString);

            message.encoding = MQC.MQENC_NATIVE;
            message.characterSet = MQC.MQCCSI_INHERIT;

            // put message onto the queue
            queue.put(message);

            // close queue;
            queue.close();
            // disconnect from queue manager
            mqm.disconnect();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

632

633

注意事项

本文针对于在美国提供的产品和服务。

IBM 可能在除美国外的其他国家不提供文档中提到的产品、服务或者功能。有关您所在的地区提供的产品和服务的信息，请咨询您所在地的 IBM 代表。任何有关 IBM 产品、程序以及服务的引用并不是为了表述或者暗示只有 IBM 的产品、程序或者服务可以使用。在不侵犯 IBM 知识产权的情况下，任何其他同样功能的产品、程序或者服务都可以替代使用。然而，对其他非 IBM 的产品、程序或是服务的实施情况进行评价和核实是用户自己的责任。

IBM 可能对本文中提到的技术拥有专利或者正在申请专利。本文的提供并不意味着将这些专利授权给您。您可以写信来咨询专利许可，地址如下：
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

下面一段文字并不适用于英国以及任何以下条例与当地法律法规相冲突的其他国家：**IBM 公司按本书内容提供该出版物，未作出任何形式的明示或暗含的保证，包括但不局限于对于不侵权、适销性、适合于某特殊目的性的暗含保证。**有一些国家不允许在某些事务中不承诺明示或暗含的保证，因此这个声明也许并不适用于您。

特此明确声明其未作出任何种类的明示或默示的陈述或保证，包括对于适销性、适合于某特殊目的性、不侵权、本网站的运营或本网站内容的任何保证。

本文可能包含技术错误或排版错误。因此，我们会对本文作定期修改；这些修改将出现在本书的新版本中。IBM 可以在任何时间在不通知的情况下，对本出版物中描述的产品和/或程序进行完善和/或更改。

本文中任何对非 IBM 网站的参考的提供仅出于方便，并没有以任何形式认可这些网站。这些网站提供的资料并不属于 IBM 的产品资料，对这些网站使用的风险由用户自己承担。

在不给您带来麻烦的情况下，IBM 可能以任何合适的方式使用或者发布您提供的信息。

关于非 IBM 产品的信息来自这些产品的生产厂家，他们公开的声明或者其他公开可用的资料。IBM 没有测试过这些产品也不能确保其性能的精准性，产品的兼容性以及其他对非 IBM 产品的声明。对非 IBM 产品功能的任何问题应该向这些产品的供应商咨询。

本文包含了日常业务操作中使用的数据和报告示例。为了尽可能完整地对他们进行描述，例子中包括了个人名字、公司名称、品牌名称以及产品名称。所有这些名字均是虚构的，如有雷同，均属巧合。

版权许可：

本文包括了样例应用程序的源代码，它展示了在不同操作平台上的编程技术。为了实现开发、使用、营销或者发布应用程序的目的，您可以依照样例程序所用的操作平台的应用程序编程接口，以任何形式复制、修改和发布这些样例程序，而不需要付费给 IBM。这些样例没有进行各种条件下的充分测试，因此 IBM 并不能保证或者暗示这些程序的可靠性，可服务性以及程序的功能。

商标

以下这些术语是美国 IBM 公司的商标或者其他地区的 IBM 公司的商标或者两者都是：

Advanced Peer-to-Peer Networking®
AD/Cycle®
AIX®
C/370™
CICS®
CICSPlex®
Domino®
DB2®
DFS™
DFSMSdfp™
DFSMSdss™
DFSMShsm™
DFSORT™
DRDA®
Encina®
Enterprise Storage Server®
Enterprise Systems Architecture/390®
ECKD™
ESCON®
FlashCopy®
FICON®
Geographically Dispersed Parallel Sysplex™
GDDM®
GDPS®
HiperSockets™
IBM®
IMS™
Language Environment®
Lotus®
Multiprise®
MVS™
MVS/ESA™
MVS/XA™
NetRexx™
NetView®
Open Class®
OS/390®
Parallel Sysplex®
Processor Resource/Systems Manager™
PR/SM™
QMF™
Redbooks™
RACF®
RMF™
S/360™
S/370™
S/390®
Sysplex Timer®
System z9™
System/360™
System/370™
System/390®
SAA®
Tivoli®
VisualAge®
VSE/ESA™
VTAM®
WebSphere®
z/Architecture™
z/OS®
z/VM®
z/VSE™
zSeries®
z9™

以下术语是其他公司的商标：

EJB，Java，JDBC，JMX，JSP，JVM，J2EE，RSM，Sun，Sun Java，Sun Microsystems，VSM，并且所有基于 Java 的商标都是美国或其他地区(或两者都有)的 Sun Microsystems 公司的商标。

Microsoft，Visual Basic，Windows，和 Windows 标志是美国或者其他地区(或两者都有)的微软公司的商标。

Intel，Intel 标志，Intel Inside 标志和 Intel 迅驰标志是美国或者其他地区(或两者都有)的 Intel 公司或者它的子公司的商标或注册商标。

UNIX 是美国或其他地区的 Open Group 的注册商标。

Linux 是美国或其他地区(或两者都有)的 Linus Torvalds 公司的商标。

636

其他的公司、产品或者服务名称可能是其他公司的商标或服务标志。

F

附录 F 附属资料

本附录包含以下内容：

相关出版物

此处列举的出版物尤为适合本教材相关主题的详细讨论。

IBM在互联网上提供了z/OS手册。 可以在z/OS互联网知识库中浏览，搜索和打印这些手册，网址如下：

http://www.ibm.com/servers/eserver/zseries/zos/bkserv

主机体系结构参考资料

- *z/Architecture Principles of Operation*, SA22-7832

z/OS数据管理参考资料

- *z/OS DFSMS: Using Data Sets*, SC26-7410

z/OS JCL和实用程序Utilities参考资料

- *z/OS MVS JCL Reference*, SA22-7597
- *z/OS MVS JCL User's Guide*, SA22-7598
- *z/OS DFSMSdfp Utilities*, SC26-7414

z/OS系统编程参考资料

- *z/OS MVS System Data Set Definition*, SA22-7629
- *z/OS MVS Initialization and Tuning Reference*, SA22-7592
- *z/OS MVS Initialization and Tuning Guide*, SA22-7591
- *JES2 Initialization and Tuning Guide*, SA22-7532

z/OS UNIX参考资料

- *z/OS UNIX System Services Command Reference,* SA22-7802
- *z/OS UNIX System Services User's Guide,* SA22-7801
- *z/OS UNIX System Services Planning,* GA22-7800

z/OS通信服务器参考资料

- *z/OS Communications Server IP Configuration Guide,* SC31-8775
- *z/OS Communications Server IP Configuration Reference,* SC31-8776

编程语言参考资料

- *HLASM General Information,* GC26-4943
- *HLASM Installation and Customization Guide,* SC26-3494
- *HLASM Language Reference, SC26-4940*
- *Enterprise COBOL for z/OS and OS/390 V3R2 Language Reference,* SC27-1408
- *Enterprise COBOL for z/OS and OS/390 V3R2 Programming Guide,* SC27-1412
- *Enterprise PL/I Language Reference,* SC27-1460
- *Enterprise PL/I for z/OS V3R3 Programming Guide,* SC27-1457
- *C/C++ Language Reference,* SC09-4764
- *C/C++ Programming Guide,* SC09-4765
- *IBM SDK for z/OS V1.4 Program Directory,* GI11-2822
- *z/OS V1R5.0 Language Environment Concepts Guide,* SA22-7567
- *z/OS V1R5.0 Language Environment Programming Guide,* SA22-7561
- *The REXX Language,* 2nd Ed., Cowlishaw, ZB35-5100
- *Procedures Language Reference (Level 1),* C26-4358 SAA CPI
- *REXX on zSeries V1R4.0 User's Guide and Reference,* SH19-8160
- *Creating Java Applications Using NetRexx ,* SG24-2216

想获得在线信息，可以访问：
http://www.ibm.com/software/awdtools/REXX/language/REXXlinks.html

CICS参考资料

- *CICS Application Programming Primer,* SC33-0674
- *CICS Transaction Server for z/OS - CICS Application Programming Guide,* SC34-6231
- *CICS Transaction Server for z/OS - CICS System Programming Reference,* SC34-6233

IMS参考资料

- *An Introduction to IMS,* ISBN 0-13-185671-5
- *IMS Application Programming: Design Guide,* SC18-7810
- *IMS Application Programming: Database Manager,* SC18-7809
- *IMS Application Programming: Transaction Manager,* SC18-7812

- *IMS Java Guide and Reference,* SC18-7821

想获得在线的信息，可以访问：
http://www.ibm.com/ims

639

DB2参考资料

- *DB2 UDB for z/OS: Administration Guide,* SC18-7413
- *DB2 UDB for z/OS: Application Programming and SQL Guide,* SC18-7415
- *DB2 UDB for z/OS: SQL Reference,* SC18-7426

WebSphere MQ参考资料

- *WebSphere MQ Application Programming Guide,* SC34-6064
- *WebSphere MQ Bibliography and Glossary,* SC34-6113
- *WebSphere MQ System Administration Guide,* SC34-6068

想获得在线的信息，可以访问：
http://www.ibm.com/software/integration/mqfamily/library/manualsa/

IBM 红皮书

想定购这些出版物，可以参考FY-641页的"怎样得到IBM红皮书"。请注意这里提及的有些文档可能只提供电子版。

- *ABCs of z/OS System Programming, Volume 1*：介绍z/OS和存储概念，TSO/E，ISPF，JCL，SDSF，MVS交付和安装。
- *ABCs of z/OS System Programming, Volume 2*：z/OS实现和日常维护，定义子系统，JES2和JES3，LPA，linklist，授权库和编目。
- *ABCs of z/OS System Programming, Volume 3*：介绍DFSMS和存储管理。
- *ABCs of z/OS System Programming, Volume 4*：通信服务器，TCP/IP和VTAM。
- *ABCs of z/OS System Programming, Volume 5*：基本和并行系统综合体，系统日志，全局域资源串行化，z/OS系统操作，自动重启管理，硬件管理控制台与性能管理。
- *ABCs of z/OS System Programming, Volume 6*：RACF，PKI，LDAP，密码学，Kerberos协议和防火墙技术。
- *ABCs of z/OS System Programming, Volume 7*：Infoprint服务器，语言环境和SMP/E。
- *ABCs of z/OS System Programming, Volume 8*：z/OS问题诊断。
- *ABCs of z/OS System Programming, Volume 9*：z/OS UNIX系统服务。
- *ABCs of z/OS System Programming, Volume 10*：介绍z体系结构，zSeries处理器设计，zSeries连通性，LPAR概念和HCD。
- *IBM WebSphere Application Server V5.1 System Management and*
- *Configuration WebSphere Handbook Series,* SG24-6195 (IBM红皮书)
- *_ z/OS WebSphere Application Server V5 and J2EE 1.3 Security*
- *Handbook,* SG24-6086 (IBM红皮书)

640

在线资料

这些IBM网站是一些相关的信息中心：

- z/OS基本技术信息中心：
  http://publib.boulder.ibm.com/infocenter/zoslnctr/v1r7/index.jsp
- z/OS网站：
  http://www.ibm.com/servers/eserver/zseries
- z/OS互联网知识库：
  http://www.ibm.com/servers/eserver/zseries/zos/bkserv
- z/OS通信服务器网站：
  http://www.software.ibm.com/network/commserver/support/
- IBM术语
  http://www.ibm.com/ibm/terminology/

怎样得到 IBM 红皮书

您可以在以下网站搜索，察看或者下载红皮书，红皮论文，相关提示和技巧，草案和附加材料，也可以定购印刷版的红皮书或者CD-ROM：

ibm.com/redbooks

从 IBM 获得帮助

IBM支持和下载网站如下

ibm.com/support

641

642

附录 G 词汇表

A

abend
异常中止。

abnormal end （异常中止）
由于在任务执行过程中发生了无法恢复的错误，而导致的一个任务，作业或子系统的结束。参见 *abnormal termination（异常中止）* 。

Abnormal termination （异常中止）
（1）一个程序处理在预期终止之前结束；（2）一个由系统故障或者操作员操作导致一个作业以失败结束。与 *abend*，*abnormal end* 同义。

ACB
访问控制块（Access Control Block）。

ACCEPT
一个 SMP/E 命令，用于在分配(程序)库中安装各种 SYSMOD。

Accept
在 SMP/E 中，用于在分配(程序)库中安装各种 SYSMOD。通过 ACCEPT 命令实现。

accepted SYSMOD （接受的 SYSMOD）
一个被 SMP/E ACCEPT 命令成功安装的 SYSMOD。它不带有 ERROR 标记集，并且可以在分配区（distribution zone）中找到对应的 SYSMOD 条目。

access authority （访问权限）
每一种权限都涉及到一种请求，此请求为访问一类已被保护的资源。在 RACF 当中，访问权限分为无权限（none），只读（read），更新（update），变更（alter）以及执行（execute）。

access list （访问列表）
在 profile 中的一个列表，保存了所有被授权的用户以及他们各自的访问权限。

access method （访问方式）
一种用于在主存和 I/O 设备之间移动数据的技术。

ACID properties （ACID 属性）
一个交易的属性：原子性（atomicity），一致性（consistency），分离性（isolation），和持久性（durability）。在 CICS 当中，ACID 属性应用于一个工作单元（UoW）。

address （地址）
系统为连接到网络的每个设备、工作站或者系统所分配的唯一编码。

address space （地址空间）
一个程序可以利用的所有范围的地址。在 z/OS 中，系统为用户创建的连续虚拟存储地址组成的地址空间可达 16EB。一个地址空间包含用户数据和程序，也包含系统数据和程序，其中有些系统数据和程序对于所有地址空间是公用的。参见 *Virtual storage address space（虚存地址空间）*。

addressing mode (AMODE) （寻址模式）
一个程序属性，指当程序开始时希望生效的地址长度。在 z/OS 中，地址长度可以是 24 位，31 位或者 64 位。

administrator （管理员）
一个角色，负责管理性任务例如访问权限和内容管理。另外，管理员可以为用户授予不同等级的权限。

allocate **在执行任务时，分配资源以供使用。**

ALLOCATE command
在 z/OS 中，一个用来连接文件的逻辑名字（ddname）及其物理名字（数据集名字）的

TSO/E 命令。

alphanumeric character
一个字母或是一个数字。

ASCII
美国资讯交换标准码（American Standard Code for Information Interchange）。

AMODE
寻址模式（addressing mode）。

ANSI
美国国家标准学会（American National Standards Institute）。

AOR
应用程序所有区域（application-owning region）。

APAR
授权程序分析报告（authorized program analysis report）。

APAR fix
一个针对影响特定用户的 IBM 系统控制程序或特许程序的缺陷所发布的临时更正。一个 APAR fix 通常会在之后被一个称为 PTF 的永久更正所替代。对于 APAR fix，可以通过使用++APAR 语句来使 SMP/E 识别它们。

APF
授权程序工具（authorized program facility）.

API
应用程序编程接口（application programming interface）。

APPC
高级程序间通信（advanced program-to program communications）。

application （应用程序）
一个或一批用来执行一个任务的程序，例如计算薪水册、库存管理以及字处理应用程序。

application-owning region (AOR)
在CICSPlex®配置中，专用于处理应用程序的CICS区域。

application program （应用程序）
用来在计算机上实现特定类型作业的软件组件的集合，比如一个用来控制库存或是薪水册的程序。

APPLY
用来向目标库（target library）安装 SYSMOD 的 SMP/E 命令。

apply
在 SMPE 中，用来向目标库（target library）安装 SYSMOD。这通过 APPLY 命令实现。

APPN
高级对等网络（advanced peer-to-peer network）。

ARM
自动重启管理器（automatic restart manager）。

ASCII
美国资讯交换标准码（American Standard Code for Information Interchange）。

ASID
地址空间标识符（address space identifier）。

ASSEM entry
一个 SMP/E 条目，包含汇编语句，可以通过编译创建目标模块。

assembler （汇编编译器）
可以将汇编语言指令转换成为二进制机器语言（目标代码）的计算机程序。

assembler language （汇编语言）
一种符号编程语言，包含基本的计算机操作指令，指令根据数据格式、存储结构和计算机寄存器构造。

asynchronous processing （异步处理）
一组操作与发出操作请求的作业分开执行；例如在工作站交互式的作业中提交一个批处理作业。参见 ***synchronous processing***

ATM
自动取款机（automated teller machine）

audit （审计）
审阅和检查数据处理系统的活动，主要是检验处理过程针对于数据安全和数据精确的合格性和有效性。

authority
访问对象，资源或功能的权利。

authorization checking （授权检查）
检验一个用户是否被准许访问一个 RACF 保护的资源。

authorized program analysis report (APAR)
为更正一个由于当前版本程序的缺陷引起的问题而提出的请求。其更正称为一个 APAR fix。

authorized program facility (APF)
允许对授权使用受限制功能的程序进行鉴定的工具。

automated operations （自动化操作）
在系统和网络操作中替代或是简化操作者工作的自动化过程。

automatic call （自动调用）
在所有主要输入被处理之后，被链接编辑器用来决定剩下的未确定的外部符号的过程。

automatic call library （自动调用库）
包含装入模块或目标卡叠，作为链接编辑器的二次输入，在所有主要输入被处理之后，来解析遗留的未定义的外部符号。
自动调用库可能是：
- 包括目标卡叠（链接编辑控制语句可有可无）的库。
- 包括装入模块的库。
- 包括语言环境运行时程序的库。

automatic restart （自动重启）
在当前运行的同时发生的一次重启，而不必重新提交作业。一个自动重启动可以发生在一个作业步的当中或开头。对照 *deferred restart*。参见 *checkpoint restart（检验点重启动）*。

automatic restart management （自动重启管理）
一个 z/OS 用来提高批处理作业和开始任务的可用性的恢复功能。当一个作业失败，或者当系统在运行时遇到不可预期的失败，z/OS 可以在没有操作员参与的情况下自动重启动作业。

auxiliary storage （辅存）
除去处理器存储之外的所有可寻址存储。

B

background （后台）
（1）指在多道程序设计中，低优先级的程序所运行的环境；（2）在 TSO/E 环境中，作业通过 SUBMIT 命令提交，或是执行 SYSIN。每次只有一个作业步被分配到中央存储的域中，并且驻留在中央存储中直到作业步结束。对照 *foreground（前台）*。

background job （后台作业）
（1）一个低优先级的作业，通常是批处理作业或者没有交互的作业；（2）在 TSO 下，通过 SUBMIT 命令或是 SYSIN 提交的作业。对照 *foreground job（前台作业）*。

backout （回退）
一个请求，去除自从上次提交或者回退后对资源的所有变更，如果是第一个恢复单元，就应该是退回到应用程序的开始。backout 又称为 *rollback* 或是 *abort*。

backup （备份）
对于一个数据集，创建一份拷贝来应对意外损失。

BAL
基本汇编语言（Basic Assembler Language）。

base function （基本功能）
在 SMP/E 中，定义当前未存在于目标库的基本 z/OS 系统元素或其他产品元素的 SYSMOD。在 SMP/E 中，基本功能用 ++FUNCTION 语句来定义。SMP/E 本身就是一个 z/OS 基本功能的实例。

base level system
SMP/E 中，系统生成时创建的目标系统模块、宏、源代码和 DLIB 的等级，功能和服务修正

程序可以应用在其上。

batch
处理一组记录或数据的作业放在一起进行处理或者传输。适合于很少或者不需要用户参与的作业。对照 *interactive（交互）*。

batch job （批处理作业）
提交到系统的一组预定义好的处理动作，很少或者不需要用户与系统之间进行交互。对照 *interactive job（交互作业）*。

batch message processing (BMP) program （批量消息处理程序）
一个能够访问在线数据库和消息队列的 IMS 批处理程序。BMP 在线运行，但是就如运行在批处理环境下的程序一样，他们通过作业控制语言（JCL）来启动。

batch processing （批处理）
一种运行一个或者一组程序的方法，在这些程序中一个或多个记录（一批）在很少或不需要用户或者操作员干预的情况下处理。对照 *interactive processing（交互处理）*。

BCP
基本控制程序（base control program）。

Big endian
存储二进制数据的格式，最高位字节首先存储。Big endian 被大多数硬件体系结构采用，包括 z/Architecture。对照 *little endian*。

binary data （二进制数据）
（1）任何不适合人们直接阅读的数据。二进制数据可能包含在文本字符范围之外的不可打印的字符。（2）一类以 0 和 1 按位的形式存储的数值数据。二进制数据能够使得很大的数值被存放在较小的存储空间之内。

bind （绑定）
（1）意指联接一个或多个控制段或程序模块组成单一程序模块，解决它们之间的引用问题。（2）在 SNA 中，用来激活两个逻辑单元（logical units，LUs）之间的一个会话的请求。

binder （绑定器）
意指用来处理语言转换器和编译器的输出，形成一个可执行程序（装入模块或程序目标）的 z/OS 程序。它替代了应用于早先版本 z/OS 操作系统（如 MVS 和 OS/390）中的链接编辑器和批装载器（batch loader）。

BLK
DD 语句中 SPACE 参数的子参数。它指定以块为单位分配空间的大小。

BLKSIZE
块大小（block size）。

BLOB
大型二进制对象（binary large object）。

block size （块大小）
（1）一个块中数据元素的个数；（2）一个块大小的量度，通常以记录、字、计算机字或字符为单位指定；（3）与块长度（*block length*）同义。（4）与物理记录大小同义（*physical record size*）。

BPAM
基本分区访问方法（basic partitioned access method）。

BSAM
基本顺序访问方法（basic sequential access method）。

buffer （缓存）
一部分用来临时保存输入或输出的存储区域。

bypass （跳过）
意指在 SMP/E 中，避免可能导致 SYSMOD 处理失败的错误。通过在一个 SMP/E 命令中使用 BYPASS 操作数来采用这一方式。

byte （字节）
存储中可寻址的基本单元。长度为 8 位。

byte stream （比特流）
存储在一个流文件中的简单字节序列。参见 *record data（记录数据）*。

C

C Language （C 语言）
一种用于以紧凑有效的代码来开发软件应用

程序的高级语言，只需要进行最小限度的更改就可以在不同类的计算机上运行。

cable “in inventory”
意指未使用的电缆。

cache （高速缓冲存储器）
一种随机访问电子存储器，用来存储常用数据，使他们被通道以更快的速度访问。

cache structure （缓存结构）
一种耦合设施的结构，允许缓存的数据在一个系统综合体的多系统应用程序之间进行高性能的共享。
应用程序可以使用一个缓存结构来执行几个不同种类的缓存系统，包括 store-through 和 store-in 缓存。

called routine （可调用程序）
被另外一个例程或程序所调用的例程或程序。

carriage control character （托架控制字符）
在输入数据记录中的一个可选的字符，可指定写、置空或跳过操作。

carriage return (CR) （回车）
（1）一个按键，通常用来指示一个命令行的结束；（2）在文本数据中，该动作指示在下一行左边空白处继续打印；（3）一个会引起打印开始的字符，该打印动作开始于回车键按下的同一物理行的起始位置。

CART
命令和响应令牌（command and response token）。

case-sensitive
大小写字母敏感。

catalog （编目/目录）
（1）关于文件和库的目录，记录了它们的存放位置；（2）向一个目录中输入关于文件或库的信息；（3）所有数据集的索引的集合，控制程序使用它定位包含某个数据集的物理卷。

cataloged data set （已编目数据集）
通过一个索引或者多个索引的层次结构能够定位一个数据集，那么这个数据集就是已编目的数据集。

cataloged procedure （编目过程）

放置于一个库中的一批 JCL 语句，可以用其名字来调用。

CCW
通道命令字（channel command word）。

CEMT
CICS 提供的交易，允许从一个控制台或者 CICS 终端会话来检查终端、连接以及其他 CICS 实体的状态。

central processor (CP) （中央处理器）
计算机的一部分，具有对于指令执行的排序和处理能力、初始化程序负载(IPL)以及其他机器操作。

central processor complex (CPC) （中央处理器联合体）
包含主存、一个或更多中央处理器、计时器以及通道这几种硬件的物理集合。

central processing unit (CPU) （中央处理单元）
与 *processor* 同义。

central storage （中央存储）
（1）在 z/OS 中，一个计算机系统的存储。在此存储中，中央处理单元可以直接得到指令和数据，并且直接将结果返回到其中。（之前称其为”实存”。）（2）与 *processor storage（处理器存储）* 同义。

CF （耦合设施）
Coupling Facility，简写为 CF，是一个运行在 s/390 或 z 系列处理器上的特殊的逻辑分区（LPAR），被用于系统综合体的多系统间来支持数据共享。

CFRM
Coupling Facility resource management，耦合设施资源管理。

CGI
公共网关接口，Common Gateway Interface。

channel adapter （通道适配器）
以电子方式聚合两个或更多控制器通道接口的设备。

channel connection address(CCA) （通道连接地址）
在 I/O 操作当中唯一指定一个通道 I/O 设备的 I/O 地址。

channel interface （通道接口）
存储控制中的电路，为主机通道关联存储路径。

channel path interface （通道路径接口）
物理处理器中的通道的逻辑等价物。

channel subsystem (CSS) （通道子系统）
在主存与 I/O 设备之间传输数据流的子通道的集合。逻辑分区通过子通道来与 I/O 设备进行通信。一个处理器所支持的最大通道子系统数取决于处理器的类型。如果处理器支持多于一个的通道子系统，那么每一个通道子系统都有一个处理器唯一的十六进制数字表示的通道子系统标示符(CSS ID)。

channel-to-channel (CTC)
一个通道至通道适配器两端的程序之间进行通信（数据传输）。

channel-to-channel adapter (CTCA) （通道至通道适配器）
一个输入输出设备，支持不同系统内程序之间的通信。

channel-to-channel (CTC) connection （通道至通道连接）
在同一个或不同处理器上两个 CHPID 之间的连接，可以是直接连接或是通过一个 switch 来实现连接。当通过一个 switch 来实现连接时，两个 CHPID 必须通过同一个或是链接的 switch 来连接在一起。

character （字符）
一个字母、数字或者其他符号，用于数据组织，控制或者呈现的组成部分。一个字符通常以连接或者相邻的笔画的空间排列形式出现。

checkpoint （检验点）
（1）例程中的一个位置，该点用于检查，或者为重启而进行的数据记录点；（2）一个点，处在该点的作业和系统状态会被记录下来，之后可以在该点进行作业步的重启动。

checkpoint data set （检验点数据集）
一个数据集，用于记录一个作业和系统状态，之后用它可以进行作业步的重启动。

checkpoint write （检验点写操作）
任何对检验点数据集的写的操作。是一个用于更新检验点数据集的初始的、中间的和最终的写操作的一个通用术语。

CHPID
通道路径标识符（channel path identifier）。

CI
控制区间（control interval）。

CICS
客户信息控制系统（Customer Information Control System）。

CICSplex （CICS 集群）
意指一个多 CICS 系统互联的配置，其中每个系统专用于总体工作负载的多个主要元素中的一个。参见 *application owning region（应用所有区域）*和 *terminal owning region（终端所有区域）*。

CKD
计数索引数据（count-key data）。

client （客户端）
从服务器端接受共享服务的功能单元。参见 *client-server（客户—服务器网络结构）*。

client-server （客户—服务器网络结构）
在 TCP/IP 中，一种远程数据处理的交互模型。在这种方式下，一方的程序发送请求到另外一方的程序并且等待响应。发起请求的程序称为客户端，应答的程序称为服务器端。

CLIST
命令表（command list）。

CLOB
大型字符对象（character large object）。

CLPA
创建链接装配区（create link pack area）。

COMS
互补金属氧化物半导体（Complementary Metal Oxide Semiconductor）。

CMS
会话监控系统（Conversational Monitor System）。

COBOL
面向商业的通用语言（Common Business-Oriented Language）。

code page （代码页）
（1）对于各个代码点，图形字符标志和控制功能的分配方法，例如，对于一个 8 位代码，256 个代码点的字符和方法的分配，对于一个 7 位代码，128 个代码点的字符和方法的分配；（2）一个对于图形字符的 16 位标志的特定分配。

code point （代码点）
一个字节的代码代表 256 个潜在字符之一。

coexistence （共存）
位于不同层次的两个或更多系统（例如软件层、服务层或操作层）之间共享资源。在共存的 2 个共享资源的系统中，一个系统可以按以下几种方法响应由另一个系统所带来的新功能：忽略、终止或支持。

command （命令）
意指实现一个操作或运行一个程序的请求。当参数、标记或其他操作数与一个命令联合时，其结果字符串则形成一条命令。

commander and response token (CART) （命令和响应令牌）
一个 WTO、WTOR、MGCRE、某些 TSO/E 命令以及 REXX exec 的参数，允许用户将命令和命令相关的响应信息连接起来。

command prefix （命令前缀）
1～8 个字符的命令标志。命令前缀可以区分命令属于某个应用程序或是子系统，而不能区分是属于哪个 z/OS 系统。

COMMAREA
对于运行在 CICS 之下的应用程序，一个可用的通信区域。

commit
执行自上一次提交或回滚之后对资源所有变更的请求。如果这是第一个恢复单元，则执行从应用程序开始以来所有的变更。

Common Business-Oriented Language (COBOL) （面向商业的通用语言）
一种基于英语的、主要用于商业应用的高级语言。

common service area(CSA) 公共服务区域
在 z/OS 中公共区域的一部分，包含可以被所有地址空间寻址的数据区域。

compatibility （兼容性）
在系统中工作的能力；或是与其他设备或程序协同工作的能力。

compilation unit （编译单元）
一部分足够完整、可以被正确编译的计算机程序。

compiler （编译器）
可将源程序翻译称为可执行程序（目标卡片叠）的程序。

compiler options （编译器选项）
在编译过程中，可以指定的关键字，来控制编译的某些方面。编译器选项可以控制编译器产生的装入模块的性质、将产生的输出打印的类型、高效使用编译器以及存放错误信息的目的地。也称 compiler-time options（编译时选项）。

complementary metal oxide semiconductor (CMOS) （互补金属氧化物半导体）
一种结合正负电极电压需求的电子属性的技术，相对于其他类型半导体使用少很多的电力。

component （组件）
一个操作系统的功能部件，例如调度程序或者

管理程序。

condition code （条件码）
一个反映之前 I/O、运算或逻辑操作结果的代码。

configuration （配置）
依照计算机系统或网络中各功能单元的性质、数量以及主要特性来进行的规划安排。

connection （连接）
意指在 TCP/IP 中两个应用程序协议中间的路径，提供可靠的数据流传输服务。在互联网通信中，一个连接由一个系统中的 TCP 应用程序延伸到另外一个系统中的 TCP 应用程序。

consistent copy （一致性拷贝）
一个单一时间点上对整个数据实体内容（例如一个逻辑卷）的一份拷贝。

console （控制台）
任何操作者可以在其上输入命令及接受信息的设备。

console group （控制台组）
在 z/OS 中，CNGRPxx 里面定义了一组控制台，里面的每一个成员在控制台或硬拷贝恢复时可以作为替代控制台，或者用来显示同步信息。

control block （控制块）
一个被计算机程序用来保存控制信息的存储区域。

control interval (CI) （控制间隙）
意指一个定长的区域或磁盘空间，VSAM 在其中存储记录并创建分散的自由空间。也指在一个键顺序数据集或文件中，顺序集索引记录中的一个条目所指向的记录集。CI 是一个 VSAM 与磁盘之间信息传输的单位。一个 CI 经常包含整数个的物理记录。

control region （控制区域）
意指包含子系统工作管理器或子系统资源管理器的控制程序的主存区域。

control section (CSECT) （控制段）
程序员指定一部分程序作为一个可重定位的单元，其中所有元素都将被装载入毗邻的主存位置。这部分程序就称为控制段。

control statement （控制语句）
在编程语言当中，控制语句用来改变多条语句的连续的顺序执行次序。一个控制语句可以是一个条件语句，比如 IF；或者是一个强制性的语句，比如 STOP。在 JCL 中，指在作业中用来向系统标志该作业或描述该作业需求的语句。

control unit (CU) （控制单元）
每个物理控制器都包含一个或更多控制单元，用来将处理器和设备之间的高级请求翻译成低级请求。与 *device control unit（设备控制单元）* 同义。

control unit address （控制单元地址）
存储控制地址的高位，用来标志宿主系统的存储控制。

controller （控制器）
意指一种设备，能够将处理器上的高级请求转换成到 I/O 设备的低级请求，或是能将 I/O 设备上的低级请求转换成到处理器的高级请求。每个物理控制器都包含一个或更多的逻辑控制单元、通道、设备接口和电源。控制器可以被分解为一个独立环节，或是被组合到一个子系统中。

conversation （会话）
意指两个程序之间通过一个 6.2 类型的会话逻辑单元建立起的逻辑连接。在处理交易的时候，该会话逻辑单元允许二者之间进行通信。

conversational （会话的）
属于需要与终端用户进行对话的一个程序或是系统，会话将会交替地接收输入然后迅速针对输入作出响应，以便于用户进行一连串连续的操作。

conversational monitor system (CMS) （会话监控系统）
意指一个虚拟机操作系统，可以提供通用的交互时间的分享能力、问题解决能力以及程序开发能力，并且只能在 VM/370 控制程序的控制之下进行操作。

CORBA
通用对象请求代理体系结构（Common Object Request Broker Architecture）。

corequisite SYSMODs （相依 SYSMODs）
意指只有在其他 SYSMOD 存在的条件下才可以被正常安装的 SYSMOD。相依 SYSMODs 可以用包含 REQ 操作数的 ++VER 语句来定义。

corrective service （纠错服务）
意指用来选择性的修复系统问题的 SYSMOD。通常纠错服务就是指 APAR 补丁。

count-key data （计数键数据）
意指一种磁盘存储器，其所存储的数据安照以下格式进行存储：计数段后面接键段，后面再接该条记录的实际数据。除了其他信息，计数段包含了记录的地址和数据的长度。其中记录的地址以 CCHHR 的格式保存（CC 是两位的柱面序号，HH 是两位的头[head]序号，而 R 是一位的记录序号），键段包含记录的键值。

couple data set （耦合数据集）
意指一种数据集，通过 XCF 耦合数据集格式实用程序创建，并且根据它指定的类型，可以被一个系统综合体之下的部分或所有 z/OS 操作系统共享。参见 *sysplex couple data set（系统综合体耦合数据集）*。

Coupling Facility （耦合设施）
一个特殊的逻辑分区，在一个系统综合体之中提供高速缓存、列表处理、和锁功能。

Coupling Facility channel （耦合设施通道）
一个以实现数据共享为目的，在耦合设施和直接与之相连的中央处理器集群之间提供高速连接的高带宽的光缆通道。

coupling services （耦合服务）
意指在系统综合体中，XCF 在各个 z/OS 系统成员之间传输数据和状态的功能。

CP
中央处理器（central processor）。

CPC
中央处理器集群（central processor complex）。

CPU
中央处理单元（central processing unit）。

create link pack area (CLPA) （创建链接装配区）
在系统 IPL 期间的一个选项，指定初始化可调页的链接装配区。

crossbar switch （纵横制交换机）
一种连接控制器到有并行（总线和标签）接口处理器的静态交换机。纵横制交换机顶部包含一组通道接口，可连接到其上的目标如处理器或其他纵横制交换机。纵横制交换机在边上也包含一组控制单元接口，可连接到其下的目标如控制器。

cross-memory linkage （跨内存联结）
一种在不同地址空间调用程序的方法。其调用过程与呼叫方同步。

cross-system coupling facility (XCF) （跨系统耦合设施）
意指 z/OS 操作系统的一个组件，可以提供种种功能来支持运行在同一个系统综合体之下的已授权程序之间的协同工作。

cross-system extended services (XES) （跨系统扩展服务）
意指一套 z/OS 操作系统服务，允许在系统综合体下不同系统中运行的一个应用或子系统的多个实例通过一个耦合设施实现高性能、高可用性的数据共享。

cross-system restart （跨系统重启动）
如果系统失败，自动重启动管理重新启动同一系统综合体内另一个符合条件的系统。

cryptographic key （加密键）
意指决定在普通文本与加密文本之间进行加密转换的参数。

cryptography （加密）
意指对数据的转换，隐藏原有的含义。

CSA
公共服务区（common service area）。

CSI
统一软件库（consolidated software inventory）。参见 *SMPCSI（主 CSI）*。

CSS
通道子系统（channel subsystem）。

CSECT
控制段（control section）。

CTC
通道至通道（channel-to-channel）。

CTC connection
通道至通道连接（channel-to-channel connection）。

cumulative service tape （累计服务磁带）
一个与新的功能清单同时派送的磁带，内含当前所有针对此功能清单的 PTF。

Customer Information Control System (CICS) 客户信息控制系统
一个在线交易处理（OLTP）系统，可以提供对数据库、文件以及终端的专用接口，从而支持商务贸易应用。客户信息控制系统允许用户开发的应用程序同时处理多个远程终端提交的交易。

D

daemon （镜像）
意指在 UNIX 系统中，一个无需看守的具有很长生命周期的进程，在系统范围内执行持续的或是间断的功能，例如网络控制。一些镜像会被自动触发来执行它们的任务；而另外一些则是间歇性的运转。一个例子就是 cron 镜像，它会间歇性地执行在 crontab 文件中列出的任务。z/OS 操作系统就等同于一个已经开始的任务。

DASD
直接访问存储设备（direct access storage device）。

DASD volume （DASD 卷）
一个以公用标号标志的直接访问存储设备，可以被一些有关的地址所访问。参见 *volume（卷）*。

data class （数据类）
每一个数据类都是文件分配与空间的属性的集合，是由存储管理员定义的，当分配一个新的 SMS 管理的数据集的时候会自动生效。

data control block (DCB) （数据控制块）
在存储或提取数据时，数据控制块被访问方式例程（access method routines）所用到。

data definition name (ddname) （数据定义名称）
（1）指数据定义声明的名字，与包含相同名字的数据控制块相对应；（2）该符号代表可放置于 DD 语句里面 name 段的名字。

data definition (DD) statement （数据定义语句）
意指描述与一个特定作业步相关联的数据集的作业控制语句。

data definition name （数据定义名称）
参见 *ddname（数据定义名）*。

data definition statement （数据定义语句）
意指一个 JCL 控制语句，可将一个文件的逻辑名称（ddname）与文件的物理名称（数据集名）连接起来。

data division （数据部）
在 COBOL 程序中，数据部描述了该程序所要用到的文件以及文件之中所包含的记录。它还描述了所有需要的 WORKING-STORAGE、LINKAGE-STORAGE 以及 LOCAL-STORAGE 的数据条目。

Data Facility Sort (DFSORT) （数据快速排序）
意指一个 0IBM 许可的高速数据处理功能。DFSORT 提供了排序、合并和拷贝操作的方法，同时还支持在记录、域以及比特级别的通用数据操作。

data in transit （传输中的数据）
意指正在被传送到恢复系统 DASD 卷中去的

应用系统 DASD 卷上的更新数据。

data integrity （数据完整性）
意指对于数据，没有意外或故意的破坏、更改或丢失现象发生。

data set （数据集）
意指在 z/OS 中的命名集合，以一个指定好的名字来储存和提取的相关数据的所有记录。

data set backup （数据集备份）
通过备份来避免个别数据集的丢失。

data set label （数据集标签）
（1）描述数据集属性的信息的集合，通常被存放于与数据集所在的相同的卷上。（2）同时代表数据集控制块和磁带数据集标签的术语。

data sharing （数据共享）
意指一种功能，允许并发的子系统或应用程序同时进行对于同一数据的直接访问和改变，同时保证数据的完整性。

data stream （数据流）
（1）所有通过一个数据链路传送的信息（数据和控制命令），通常是进行单一的读或写操作。（2）指正在传输的持续的数据元素流，或是指使用定义好的格式、以字符或是二进制码形式进行的传输。

data type （数据类型）
表现数据特征的属性和内部代表。

data warehouse （数据仓库）
意指一个为组织机构提供重要商务信息的系统。数据仓库系统将数据进行净化来保证数据的准确性和流通性，然后再将数据呈现给决策者，使之可以有效果的、高效率的翻译和使用数据。

database （数据库）
表的集合，或是表空间和索引空间的集合。

database administrator (DBA) （数据库管理员）
意指一个负责设计、开发、操作、保障、维护和使用数据库的个人。

database management system (DBMS)（数据库管理系统）
意指一个软件系统，可以控制数据库的创建、组织和调整，以及对储存于其中的数据进行访问控制。

DBCS
双字节字符集（double-byte character set）。

DBMS
数据库管理系统（database management system）。

DB2
DATABASE 2；通常指 IBM 关系型数据库管理系统的一个家族，也特指运行在 z/OS 下的子系统。

DB2 data sharing group （DB2 数据共享组）
一个或更多并发的 DB2 子系统的集合，允许其中每个子系统直接访问相同的数据，同时保证数据完整性。

DCB
数据控制块（data control block）。

DCLGEN
声明产生器（declarations generator）。

ddname
数据定义名称（data definition name）。

DD statement
DD 语句（data definition statement）。

deadlock （死锁）
（1）意指一个进程不能继续的错误条件，原因是在进程的两个元素之中，每一个元素都在等待另一个元素的动作，或是在等待来自另一个元素的响应。（2）指对于资源的不可调解的争夺。（3）意指这样一种僵局：当多个进程都在等待一个资源的时候，该资源正在被另外一个进程控制着，而且同样处在等待的状态。

deallocate （取消分配）
意指将已分配给指定任务的资源释放出来。

declarations generator (DCLGEN) （声明

发生器）
一个 DB2 的子组件，用来生成 SQL 表的声明，以及按照表生成 COBOL、C 或是 PL/I 数据结构声明。其声明产生自 DB2 系统编目信息。

dedicated （专属）
意指为一个应用或目标进行的系统资源（设备、程序或是整个系统）的分配。

default （默认）
意指当用户没有专门指定其他选择时，所使用的值或所采取的动作。

deferred restart （延期重启）
当用户重新提交一个作业的时候，系统执行的重启动。操作员通过系统输入读取机来将重启动叠卡提交到系统中去。参见 *checkpoint restart（检验点重启动）*。对照 *automatic restart（自动重启动）*。

deleted function （已删除的功能）
指在 SMP/E 中，当另外一个功能被安装到系统的时候，被删除的功能。已删除的功能会在 SYSMOD 的子条目 DELBY 中被记录下来。

destination （目的地）
节点名与以下几种之一的组合：用户 ID、远程打印机或远程打孔机、特殊的本地打印机、或是 LOCAL（当仅指定了一个节点名的条件下所采用的默认值）。

destination node （目的节点）
意指为已被授权的外部用户提供应用服务的节点。

device （设备）
计算机的外围设备或是同样出现在应用程序外围的对象。

device address （设备地址）
意指一个 ESCON 设备层结构的域，用来在控制单元镜像上面选择一个指定的设备。

device control unit （设备控制单元）
意指一个硬件设备，用来控制数据在一个或更多 I/O 设备或终端上的读、写以及显示。

device number （设备号码）
一个四位十六进制的标识符，例如 13A0，用来标识一个设备，以便于程序与主机操作员之间的通信。也指标识一个子通道的设备号码。

Device Support Facilities program (ICKDSF) （设备支持功能程序
意指一个用来在安装过程中初始化 DASD 卷以及执行介质维护的程序。

DFSMS
数据工具存储管理子系统（Data Facility Storage Management Subsystem）。

DFSMShsm
IBM 的一种产品，用于备份和恢复数据，并管理存储层次结构中磁盘卷的空间。

device type （设备类型）
意指对于一类设备的通用名，例如 3390。

DFS
分布式文件服务（Distributed File Service）。

DFSORT
数据排序功能（Data Facility Sort）。

dialog （对话框）
意指交互式的弹出窗口，包含很多选项，这些选项允许浏览或是调整信息、对所选的对象做出专门的动作、或是访问其他对话框。例如，HCM 提供了一系列的对话框来帮助用户创建、编辑、删除以及连接对象，同时操纵配置图表。

direct access storage device (DASD) （直接访问存储设备）
意指一种存储设备，其数据的访问时间不依赖于数据的存放位置。

directory （目录）
（1）意指一类文件，包含有其他文件或是目录的名称和控制信息。目录还可以包含子目录，同时子目录还可以包含更下一层的子目录。（2）一个包含目录条目的文件。在同一个目录下，不允许存在任何两个具有相同名称的目录条目。（3）一个指向文件或其他目录的文件。（4）意指被控制程序所使用的索引，用来定位存储与同一直接访问存储之上，但却分散在不同区域的一个数据集的数据块。

disaster recovery （灾难恢复）
从一次使系统损毁或瘫痪的灾难中恢复，例如一场火灾。典型的灾难恢复技术就是将数据备份至第二系统（恢复系统），然后使用恢复系统替代已损毁或瘫痪的应用系统。参见 *recovery（恢复）*，*backup（备份）*以及 *recovery system（恢复系统）*。

DISP
意指一个 JCL DD 语句的参数：部署。

display console （显示控制台）
意指在 z/OS 中，一个允许用户控制输入输出功能的 MCS（Multiple Console Support，多控制台支持）控制台。

distributed computing （分布式计算）
意指需要两个或更多机器通过网络互联协作的计算。数据和资源将在各个计算机之间实现共享。

Distributed Computing Environment (DCE) 分布式计算环境
意指一套综合的、集成的服务，用来支持分布式应用的开发、使用以及维护。分布式开发环境独立于操作系统和网络；它提供了在不同平台之间的协作性和移植性。

distributed data （分布式数据）
意指存放在数据库管理系统，而不是存放在本地系统上的数据。

Distributed File Service (DFS) （分布式文件服务）
意指一个分布式计算环境的组件。分布式文件服务将几个文件服务器组合成为本地文件系统，使得这些文件对于所有分布式文件服务的客户端机器来说都是同等可用的。不论文件在服务器上的物理位置怎样，分布式文件服务都允许网络中任何位置的用户来访问和共享存储在文件服务器上的文件。所有文件的命名都从属于同一个全局性的命名空间，这样一来不论在网络中的任何地方，只要通过相同的名字就能找到对应的文件。

distribution library (DLIB) （分配(程序)库）
意指包含系统中所有元件的原版拷贝的库。分配程序库可以用来创建或备份目标程序库。

distribution zone （分配区）
意指在 SMP/E 中统一软件库（CSI）数据集中的一组记录，描述了 SYSMOD，以及分配程序库中的各个元素。

DLIB
分配(程序)库（distribution library）。

DLL
动态链接库（dynamic link library）。

double-byte character set (DBCS) 双字节字符集
意指一套字符，其中的每个字符都代表一个双字节码。诸如中文、日文和韩文等语言都包含有更多字符，不能够仅由 256 个代码点完全表达出来，而是需要依靠双字节字符集来表达。因为每个字符都需要两个字节，所以双字节字符集的字符需要硬件和程序来支持其输入、显示以及打印。对照 *single-byte character set（单字节字符集）*。

doubleword （双字）
意指一系列组成 8 字节长（2 个 4 字节字）的二进制位或字符，作为一个单元来引用。

downwardly compatible （向下兼容）
意指应用程序的一种能力，允许它们运行在早先版本的 z/OS 上。

drain
意指在停止打印机设备之前，允许它完成当前的工作。

driving system （驱动系统）
用来安装程序的系统。对照 target system（目标系统）。

dsname
数据集名（data set name）。

DSORG
数据集组织结构（Data set organization），数据类定义中 DCB 和 DD 语句的参数。

dump （转储）

意指展现存储内容的报告。转储通常在一个程序失败之后生成，用来辅助问题的诊断。

dynamic allocation （动态分配）
意指在程序执行时，而非程序装载至中央存储时，对此程序分配系统资源。

dynamic link library (DLL) （动态链接库）
意指一个文件，包含了可执行代码，以及在装载时或运行时绑定到程序的数据。动态链接库的代码和数据可以被几个应用程序在同一时刻进行共享。

dynamic reconfiguration （动态重配置）
意指在系统运行时，对通道子系统和操作系统进行变更同时立即生效，而不需要重新启动系统。

E

e-business （电子商务）
（1）意指通过如 Internet 这样的电子媒介进行的商务交易。（2）意指通过使用互联网技术来转换关键交易步骤。

EB
2 的 60 次方。参见 *exabyte（2 的 60 次方）*。

EBCDIC
广义二进制编码的十进制交换码（Extended Binary Coded Decimal Interchange Code）。

EC
工程变更（engineering change）。

ECSA
扩展公共服务区（extended common service area）。

EDT
合格设备表（eligible device table）。

element （元件）
意指在 SMP/E 中，产品的一个部分，例如宏、模块、对话面板或是样本代码。

eligible device table (EDT) （合格设备表）
意指在安装期间定义好的表，其中的条目列举了所有合格的可以进行配置的设备。合格配置表定义了这些设备之间小范围内的和普遍的关系。在 IPL 期间，安装过程识别出 z/OS 所使用的合格设备表。IPL 之后，在被选定的合格设备表中，作业可以请求任何小范围的设备组进行设备配置。每个合格设备表都会被用一个特有的 ID（两位）来唯一标识，并且都会包含一个或更多的小范围组和普遍组。

enclave
意指在一个或更多地址空间可跨多个可调度单元（的交易

encrypt （加密）
系统地将数据进行编码，使这些数据在不知道加密玄机的情况下就不会被读懂。

endian
数据表现得一个属性，反映了多字节数据在内存中的存储形式。参见 *big endian* 和 *little endian*。

enterprise （企业）
意指构成商务相关的所有可操作的实体、功能以及资源的总和。

Enterprise System Connection (ESCON) （企业系统连接）
意指一套产品和服务，可使用光缆作为传输媒介，提供可动态连接的环境。

entry area （输入区）
意指在 z/OS 中，控制台屏幕可供操作员输入命令或输入命令回复的区域。

entry name （入口名）
意指在汇编语言里面，一个由程序员指定的名字，用来在控制段里面标志一个程序的入口点，同时可以被任何控制段引用。参见 *entry point（程序入口点）*。

entry point （程序入口点）
意指当一个例程被输入执行的时候，其第一个指令的地址或标签。在一个装入模块中，当该模块被调用的时候，控制就转移至其程序入口点上。

esoteric
esoteric （或 **esoteric** 设备组）是一个由

安装定义和命名的 I/O 设备的集合，集合内的 I/O 设备通常属于同一设备组。合格设备表定义了这些设备小范围内和普遍的关系。分配给一个 **esoteric** 的名字将被用在 JCL DD 语句中。之后该作业将从此 **esoteric** 设备组中选择一个设备来进行分配，替代一个给定的设备代码或是普遍设备组。

EOF
文件的结束（End of file）。

ESCON
企业系统连接（Enterprise Systems Connection）。

ETR
外部时间引用（External Time Reference）。

exabyte
意指对于处理器来说，实存与虚存的容量或是通道量：2 的 60 次方。

exception SYSMOD （例外 SYSMOD）
意指出错的或是在安装之前需要进行特别处理的 SYSMOD。++HOLD 和++RELEASE 语句可以用来标识例外 SYSMOD。

EXCP
可执行通道程序（execute channel programs）。

executable （可执行）
意指一个已经被读入内存等待执行的装入模块或程序对象。

executable program （可执行程序）
（1）意指一个程序，其形式适合用计算机来执行。该程序可以是一个应用程序或是一个脚本程序。（2）意指一个已经被编辑链接好的，可以在处理器中运行的程序。（3）意指可以被当作一个完备的过程来执行的程序。它由一个主程序和非必要的一个或更多子程序组成。（4）参见 *executable file（可执行文件）*, *load module（装入模块）*。

Extended Binary-Coded Decimal Interchange Code (EBCDIC) （广义二进制编码的十进制交换码）
意指在z/OS环境中用来表现字符数据的编码方案。对照*ASC II（美国信息交换标准码）*和*Unicode（统一字符编码标准）*。

extended MCS console （扩展MCS控制台）
意指在z/OS中的一个控制台而不是MCS（Multiple Console Support，多控制台支持）控制台，可以允许操作员或程序输入系统命令和接收信息。扩展MCS控制台可以通过OPERPARM段来定义。

extended remote copy (XRC) （扩展远程复制）
意指基于硬件和软件的远程复制服务选项。针对用于灾难恢复、设备移植和工作负载移植的交叉存储子系统，扩展远程复制支持它们的异步卷复制。

external reference （外部引用）
意指在一个对象叠卡中，由其他程序或模块定义的，对一个象征符号的引用，例如一个程序入口点的名字。

F

feature （特性）
一个 IBM 产品的一部分，可以被客户单独的选购。

feature code （特性代码）
被 IBM 用来处理硬件和软件采购清单的四位代码。

fetch （获取）
意指对于一个过程的动态读取。

Fiber Connection Environment (FICON) （光纤连接环境）
意指光纤通信方法，为主机系统提供高数据传输速率、高带宽、更远的传输距离以及使每个控制单元上可以连接更多的设备。FICON 可以与 ESCON 协同工作，或是替代 ESCON。

fiber link （光纤链路）
意指在光纤发送端与接收端之间的物理光纤连接以及传输媒介。在光纤管理柜里面，一个光纤链路可以包含一个或更多光缆和 patchport。在光纤链路中，每个连接都可以

是永久的或是易变的。

FICON
光纤连接环境（Fiber Connection Environment）。

FIFO
先进先出（first in, first out）。

file （文件）
意指在 z/OS 中的命名集合，以一个指定好的名字来储存和提取的相关数据的所有记录。

FILEDEF
文件定义语句（file definition statement）。

first in, first out （先进先出）
意指一种排序方式，其下一个被取出的项是最早进入队列的项。

firewall （防火墙）
意指一种中间服务器，其功能是从不安全的网络中隔离出一个可靠的网络。

fix （补丁）
意指对程序中所存在错误的更正，通常是一个临时的更正，或是绕开有缺陷的代码部分。

fixed-length record （定长记录）
意指逻辑上或物理上有关联的记录，拥有相同的长度。对照 *variable-length record（变长记录）*。

FlashCopy （快速复制）
意指 point-in-time 复制服务功能，可以迅速从源位置向目标位置复制数据。

FMID
功能更改标识符（function modification identifier）。

foreground （前台）
（1）指在多道程序设计中，高优先级的程序被执行的环境。（2）指在 TSO 下的，程序在主存中交换进出的环境，允许终端用户共享 CPU 时间。所有的命令处理器程序都在前台执行。对照 *background（后台）*。

foreground job （前台作业）
（1）一个高优先级的作业，通常是一个实时作业。（2）指在 TSO 下，任何在中央存储的交换区执行的作业，例如一个命令处理器或一个终端用户的程序。对照 *background job（后台作业）*。

foreign key （外键）
指在一个约束关系中的一列或几列子表。键必须拥有相同的列数同时有相同的描述，就像父表的主键。每个外键值必须在相关的父表中匹配一个父键值，否则就要置为空。

fork
意指创建和开始一个子进程。fork 的功能与创建和配属一个地址空间相似。它创建一个父进程的副本，包括打开文件描述符。

Fortran
一种高级语言，主要被用来开发包含数字计算的应用程序。在早先的用法中，这种语言的名字都是以大写字母表示的，即 FORTRAN。

frame （框架）
对于一个主机微处理器簇来说，一个框架包含一个到两个中央处理集群（CPC）、支撑基础以及交流电源配给。

FTP
文件传输协议（File Transfer Protocol）。

fullword （整字长）
指一系列位或字符，组成的 4 字节长（一个字），被当作一个单元引用。

fullword boundary （整字长边界）
指地址可被 4 整除的存储位置。

function （功能）
指在 SMP/E 中，根据需要可以被安装到用户系统的产品。功能可以通过++FUNCTION 语句被 SMP/E 识别。每个功能必须有个特有的功能限定标识符（FMID）。

function modification identifier (FIMD) 功能更改标识符
意指一个代码，用来标志得到 z/OS 许可的程序的版本等级。

G

gateway node （网关节点）
指作为网络之间的接口的节点。

GB
十亿字节（gigabyte）。

GDG
世代数据集（generation data group）。

generalized trace facility (GTF) 广义跟踪工具
指类似系统跟踪的功能，收集用来决策和诊断系统操作期间问题的信息。然而与系统跟踪不同的是，**广义跟踪工具**可以被定制记录一个指定的系统和用户程序事件。

generation data group （世代数据集）
意指这样的一种集合：包含了历史上相关的非 VSAM 数据集，这些数据集是按照历史顺序排列的；每一个这样的数据集就成为一个世代数据集。

generic （类别）
指 z/OS 定义的一组设备，这些设备具有近似的属性。例如，设备类型 3270-X、3277-2、3278-2、-2A、-3、-4 和 3279-2a、-2b、-2c、-3a、-3b 属于同类别。每一个类别都有一个类别名，被用在 JCL DD 语句中来分配设备。z/OS 将这个名字译为"选定该组的所有设备"。在一个给定的 z/OS 配置中，每个合格设备表（EDT）都含有相同的类别表。

Geographically Dispersed Parallel Sysplex （地理分散的并行系统综合体）
指一种集成了并行系统集群技术和远程复制技术的应用，从而提高应用的可用性和灾难恢复能力。GDPS 拓扑结构是将一个并行系统集群分散到两个不同的地点，在两个地点上，所有的重要数据形成镜像式的备份。GDPS 在一个点上，集中控制管理远程复制的定制和存储子系统、使并行系统集群的操作性任务自动化以及使故障恢复自动化。

gigabyte （十亿字节）
2 的 30 次方，1073741824 字节。这接近于美式英语中的一个 billion 字节。

global access checking （全局访问检查）
意指一种能力，允许一个安装在存储器内依照默认值建立一个表，该表为所选的资源定义了授权级别。

global resource serialization （全局资源串行化）
指一个 z/OS 的功能，该功能提供了在多个 z/OS 镜像中对于资源（尤其以数据集为代表）的排序机制。

global resource serialization complex （全局资源串行化复合体）
指一个或更多 z/OS 系统使用全局资源排序来对共享资源（例如位于共享直接访问存储器卷上的数据集）进行访问排序。

global zone （全局区）
指 CSI 数据集中的一组记录，用来记录来自特定系统的 SYSMOD 的信息。全局区同时包含以下信息：（1）使 SMP/E 可以访问该特定系统的目标和分配区。（2）使用户可以定制 SMP/E 处理的各个方面。

Gregorian calendar （公历）
指自从 1582 年 10 月 15 日起，开始在世界普遍使用的日历。

group （组）
指 RACF 用户的集合，集合内的用户可以共享对于受保护资源的访问授权。

H

hardcopy log （硬拷贝日志）
指在具有多控制台支持（MCS）的系统中，一个对于系统活动的永久的记录。

hardware （硬件）
意指物理设备，与计算机程序或使用方法相反；例如，机械的、磁性的、电的或是电子的设备。对照 *software（软件）*。

hardware configuration dialog (HCD) （硬件配置会话）
指在 z/OS 中的面板程序，属于硬件配置定义的一部分。该程序允许安装过程为 z/OS 系统

配置定义设备。

Hardware Management Console (HMC)（硬件管理控制台）
指用来监视和控制硬件（如主机微处理器）的控制台。

hardware unit （硬件单元）
意指中央处理器、存储元件、通道路径、设备以及其他等等。

HASP
休斯顿自动假脱机优先系统（Houston Automatic Spooling Priority）。

HCD
硬件配置定义（Hardware Configuration Definition）。

head of string （字符串头）
指在一个字符串中，设备的第一个单元，包含了连接到控制器设备接口的字符串接口。

hexadecimal （十六进制的）
以 16 为基数的计数系统。十六进制位的范围是从 0 到 9（十进制的 0 到 9）和 A 到 F（大小写不敏感，指十进制的 10 到 15）。

HFS
分层文件系统（hierarchical file system）。

hierarchical file system (HFS) （分层文件系统）
意指一个包含 POSIX-complaint 分级文件系统的数据集。POSIX-complaint 分级文件系统是以分级结构组织的文件和目录的集合，可以通过 z/OS UNIX 系统服务功能进行访问。

high-level language (HLL) （高级语言）
指在汇编语言级别之上，又在程序产生器和查询语言级别之下的编程语言。例如 C、C++、COBOL、Fortran 和 PL\I。

high parallel （高并行度）
指在一个并行系统中的多系统操作，每一个系统都拥有多个处理器。参见 *n-way*。

HMC
硬件管理控制台（Hardware Management Console）。

HOLDDATA
指在 SMP/E 中的一个或更多 MCS 控制台，用来指示某个包含错误或是在被安装之前需要进行特别处理的 SYSMOD。++HOLD 和 ++RELEASE 语句可以用来定义 HOLDDATA。受 HOLDDATA 影响的 SYSMOD 被称为例外 SYSMOD。

Houston Automatic Spooling Priority (HASP) （休斯顿自动假脱机优先系统）
指计算机程序，可以提供辅助作业管理、数据管理以及任务管理功能，例如：控制作业流、定制任务以及假脱机操作。参见 *JES2（作业输入子系统）*。

I/O
输入输出。

I/O cluster （I/O 簇）
指一个拥有逻辑分区处理器配置的系统集群，同时该逻辑分区处理器配置带有可控制通道路径。

I/O device （I/O 设备）
如打印机、磁带驱动器和硬盘驱动器等等。

IBM Support Center （IBM 支持中心）
指负责软件服务的 IBM 组织。

IBM system engineer (SE) （IBM 系统工程师）
指一类 IBM 服务代表，负责为已经投放进市场的 IBM 软件执行维护服务。

ICSF
集成加密服务工具（Integrated Cryptographic Service Facility）。

IDCAMS
指一个用来处理访问方式服务命令的 IBM 程序。它可以从 TSO 终端或是内嵌在用户应用程序里，以作业或是作业步的形式进行调用。

image （镜像）
指 z/OS 操作系统的一个实例。

IMS

信息管理系统（Information Management System）。

IMS DB
信息管理系统数据库管理（Information Management System Database Manager）。

IMS DB data sharing group （信息管理系统数据共享组）
一个或更多并行的 IBM DB 子系统，可以直接访问和变更相同的数据，同时保持数据的一致性。

Information Management System (IMS)（信息管理系统）
指一种 IBM 产品，支持分级数据库、数据通信、转化处理以及数据库回退和恢复功能。

initial program load (IPL) （初始程序装载）
指 z/OS 操作系统的初始化过程，使 z/OS 操作系统开始执行。在 IPL 期间，系统程序被载入存储，同时 z/OS 已经准备执行工作。与 *boot（导入）*、*load（装载）*同义。

initial storage allocation （初始化存储分配）
指将被分配给一个逻辑分区的中央存储和扩展存储的量。

initiator （启动器）
指操作系统的一部分，可以从系统输入设备读取操作控制语言并进行处理。

initiator/terminator （启动器/终结器）
作业计划功能，可以选定作业或作业步来执行，并为它们分配输入输出设备，将它们置于任务控制之下，同时在作业结束之后，为作业在系统输出单元所进行的输出显示提供控制信息。

input/output configuration data set (IOCDS) （输入输出配置数据集）
指一个文件，包含了不同对于选定的处理器的配置定义。同一时刻只能有一个输入输出配置数据集是生效的。当主处理器被用于通道子系统时，对于与主处理器上的处理器控制器，输入输出配置数据集包含了与其相关联的文件的 I/O 配置数据。通道子系统使用此配置数据来控制 I/O 请求。输入输出配置数据集创建于 IODF（输入输出定义文件）。

input/output definition file (IODF) （输入输出定义文件）
指一个 VSAM 线性数据集，内含 I/O 定义信息，包括处理器和操作系统的 I/O 定义，以及所有逻辑对象和它们在硬件配置中的连通性。

install （安装）
指在 SMP/E 中，将一个 SYSMOD 应用于目标库或接收到分配(程序)库。

instruction line （提示行）
指在 z/OS 中，控制台屏幕的一部分，包含关于控制台控制信息和输入错误信息。

interactive （交互）
隶属于一个程序或系统，交替的接收输入和响应输入。指在一个交互系统中，一个存在于用户和系统之间的持续的对话。对照 batch（批处理）。

interactive problem control system (IPCS)（交互式故障控制系统）
指一个 z/OS 组件，支持在线故障管理、交互式故障分析、在线 dump 纠错、故障跟踪以及故障报告。

Interactive System Productivity Facility (ISPF) （交互式系统生产设备）
指一个对于交互式应用的会话管理器。可控制和服务于会话中得到许可的操作。

internal reader （内部阅读器）
指一种向作业输入子系统传递作业的功能。

interrupt （中断）
指挂起一个进程（例如一个计算机程序的执行），该挂起动作是由一个进程外部的事件所引发的。被挂起的进程还可以被恢复。

IOCDS
输入输出配置数据集（input/output configuration data set）。

IODF
输入输出定义文件（input/output definition

file）。

IPCS
交互式故障控制系统（Interactive Problem Control System）。

IPL
初始程序装载（initial program load）。

IPv6
国际互联网协议版本 6（Internet protocol Version 6）。

ISMF
交互式存储管理工具（interactive storage management facility）。

ISPF
交互式系统生产设备（Interactive System Productivity Facility）

ISPF/PDF
交互式系统生产设备/程序开发设备（Interactive System Productivity Facility/Program Development Facility）

IVP
安装确认过程（installation verification procedure）。

J

JCL
作业控制语言（job control langrage）。

JES
作业输入子系统 （job entry subsystem）。

JES2 （作业输入子系统 2）
指 z/OS 的一个子系统，可以接收作业至系统、将作业转换成内部格式、选定要执行的作业、处理作业的输出以及将作业清楚出系统。在多处理器的模式下，每个 JES2 处理器独立控制自己的作业输入、计划以及输出处理。对照 JES3（作业输入子系统 3）。

JES3 （作业输入子系统 3）
指 z/OS 的一个子系统，可以接收作业至系统、将作业转换成内部格式、选定要执行的作业、处理作业的输出以及将作业清楚出系统。在包含几个松耦合处理单元的集群中，由 JES3 程序来管理处理器，这样所有的处理器都将得到集中控制，并且通过一个普遍的作业队列来向各个处理器分配作业。对照 JES2（作业输入子系统 2）。

job （作业）
指操作系统的工作单元。作业可以通过 JCL 语句来定义。

job class （作业类别）
指可以为作业指定的众多作业类别中的任何一个类别。有了作业分类，加上为启动器/终结器指定可以启动的作业类别，系统就可能控制多个不同类别的作业混合并发执行。

job control language (JCL) （作业控制语言）
指一系列的命令，用来向系统确定一个作业，并描述作业的需求。

job entry subsystem (JES) （作业输入子系统）
指可以进行排存、作业排序以及管理 I/O 的系统功能。

job entry subsystem 2
参见 *JES2（作业输入子系统 2）*。

job entry subsystem 3
参见 *JES3（作业输入子系统 3）*。

job priority （作业优先级）
指一个分配给作业的值，当作业之间竞争系统资源的时候用来量度作业的相对重要性。

job separator pages （作业分页）
指界定作业的那些打印输出页。

job step （作业步）
意指那些请求和控制程序的执行的作业控制语言语句，或是那些指定运行该程序所需的资源的作业控制语言语句。一个作业步的 JCL 语句包括一个 EXEC 语句，来指定将被调用的程序或过程，后面接一个或更多的 DD 语句，来指定程序可能需要的数据集或是 I/O 设备。

Julian date （儒略日）
指一种日期格式，其第 1 和 2 位置包含年，3 到 5 位置包含日。日是以 1 到 366 的数字来代表的，以取代月。

jumper cable （分号电缆/跨接电缆）
指用来在 patchport 之间进行可变链接的光纤。

K

kernel （内核）
指操作系统的一个部分，执行基本功能，例如分配硬件资源。

key-sequenced data set (KSDS) （键顺序数据集）
指一种 VSAM 文件或是数据集，其数据是依其键的升序顺序载入的，用一个索引来进行控制。记录依照键或是地址的方式来被取出和存储，而新记录按照键顺序和分配的自由空间插入的。由于控制间隔或控制区是分开的，故其相对字节地址是可变的。

keyword （关键字）
指命令操作数的一部分，由指定的字符串组成（例如 DSNAME=）。

KSDS
键顺序数据集（key-sequenced data set）。

L

LAN
局域网网络（local area network）。

Language Environment （语言环境）
z/OS 语言环境的缩写。指一系列构架和接口，为 C、C++、COBOL、Fortran、PL/I、Visual PL/I 和 Java 等语言的应用程序（由语言环境适应编译器编译）提供通用的运行时环境和运行时服务。

last in, first out (LIFO) （后进先出）
指一种排序技术，其下一个要被取出的项最晚进入队列。

LCSS
逻辑通道子系统（logical channel subsystem）。

LCU
逻辑控制单元（logical control unit）。

LDAP
轻量级目录访问协议（Lightweight Directory Access Protocol）。

library （库）
指一个分区数据集（PDS），包含一个成员命名相关的集合。参见 *partitioned data set（分区数据集）*。

LIC
许可的内部代码（Licensed Internal Code）。

licensed internal code (LIC) （许可的内部代码）
指 IBM 不作为机器的一部分销售的微代码，但是客户是经过许可的。许可的内部代码在内存的一部分中执行，却并不能被用户的程序寻址。有些 IBM 产品用内部许可代码作为硬件电路的替代方法来执行功能。

licensed program （特许程序）
指一个软件包，可以从程序库中订购，例如 IBM 软件分配(程序)库（ISMD）。IMS 和 CICS 就是特许程序的两个例子。

Lightweight Directory Access Protocol (LDAP) （轻量级目录访问协议）
指一个互联网协议标准，基于 TCP/IP 协议，运行访问和操纵数据在目录信息树（Directory Information Tree）中的组织。

LIFO
后进先出（last in, first out）。

link library （链接库）
指一个数据集，包含链接编辑对象模块。

link pack area (LPA) （链接装配区）
指虚拟存储的一个区，包含在 IPL 期间载入的可重入的例程，可以被系统中所有任务同时使用。

linkage editor （链接编辑器）
指一个操作系统的组件，可以解决分开编译或集合的模块之间的交叉引用，然后指定最终地址，来创建一个可重定位的载入模块。链接编辑器在之后将载入模块保存至硬盘上的一个装载库。

linked list （链表）
指一个表，其中的元素可以是分散的，而每一个元素都包含指向下一个元素的地址信息。与 chained list（链表）同义。

link-edit （链接-编辑）
指通过链接编辑器或绑定器来创建一个可载入的计算机程序。

list structure （列表结构）
指一个耦合设施的结构，允许 sysplex 中的多系统应用程序可以共享表或队列结构的信息。一个表结构由一系列表和一个可选的锁表所组成，锁表可被用来在表结构中对资源进行序列化。每个表包含一个由表的条目构成的队列。

little endian
指一种二进制数据存储格式，在这种格式中，最不重要的字节被放在第一个位置。little endian 用于 Intel 硬件架构。对照 big endian。

LMOD
指在 SMP/E 中装入模块（load module）的缩写。

load module （装入模块）
指一种存储在分区数据集程序库中的可执行程序。参见 program object（程序对象）。

local area network （局域网）
意指一种网络，其通信范围被限定在中等大小的地理区域（1 到 10 千米），例如一个办公楼、商店或是校园。 一般情况下局域网不延伸到外公用网络。依靠中高档数据传输速率通信媒介的本地网络，运行出错的概率很低。

local system queue area (LSQA) （本地系统队列区）
指在 z/OS 中，与虚拟存储区联合的一个或多个段。其中每个虚拟存储区都包含作业相关的系统控制块。

lock structure （锁结构）
意指一种耦合设施的结构，允许 sysplex 中的应用针对一系列应用定义的资源执行定制的锁协议。锁结构支持共享、排他以及应用定义的锁状态， 同时支持普遍竞争管理和恢复协议。

logical control unit (LCU) （逻辑控制单元）
指一个可能带有附加设备的单一控制单元，或一组共享设备的控制单元。在通道子系统中，一个逻辑控制单元代表一系列物理或逻辑上普遍附加有 I/O 设备的控制单元。

logical partition (LP) （逻辑分区）
指被定义来支持一个操作系统的处理器硬件的一个子集，参见 logically partitioned (LPAR) mode（逻辑分区模式）。

logical partitioning （逻辑分区功能）
指操作系统的功能，运行创建逻辑分区。

logical subsystem （逻辑子系统）
指存储控制器的逻辑功能，允许一个或更多主 I/O 接口访问一系列设备。控制器根据对联合 I/O 接口的寻址机制来集合设备。一个存储控制器上存在有一个或更多逻辑子系统。一般来说，控制器可与一系列给定的设备联合，但仅与一个逻辑子系统联合。

logical unit (LU) （逻辑单元）
指在 SNA 中的一个端口，终端用户可以通过它访问 SNA 网络，从而与其他终端用户实现通信。同时终端用户还可以通过它访问系统服务控制点（system services control points）所提供的功能。

logical unit type 6.2 （逻辑单元类型 6.2）
指支持协作处理环境下通信的一类 SNA 逻辑单元。

logically partitioned (LPAR) mode （逻辑分区模式）

logoff （登出）

（1）指用户终止终端会话的过程。（2）指在 VTAM 中，请求从 VTAM 应用程序断开一个终端。

logon （登录）
（1）指用户启动终端会话的过程。（2）指在 VTAM 中，请求从 VTAM 应用程序连接一个终端。

loop （循环）
指一个或一组指令重复执行的状态。

loosely coupled （松耦合）
指一个多系统结构，当处理一个工作负载时，多个 z/OS 镜像之间需要进行较低等级的交互和协作。参见 *tightly coupled（紧耦合）*。

LP
逻辑分区（logical partition）。

LPA
链接装配区（link pack area）。

LPAR
逻辑分区（模式）（logically partitioned/logically partitioned mode）。

LRECL
逻辑记录长度（logical record length）。

LSQA
本地系统队列区（local system queue area）。

LU
逻辑单元（logical unit）。

M

machine check interruption （机器检查中断）
指一种中断，由设备故障或错误产生。

machine readable （机器可读）
指计算机可以从存储设备、数据媒介或其他来源处理解或翻译的数据。

macro （宏）
源语言中的一种指令，将会被一个定义好的相同源语言编写的指令队列替代。

main task （主任务）
指在 z/OS 多任务并行处理的情况下的主程序。

MAS
并行访问池配置（multi-access spool configuration）。

master catalog （主目录）
指一种包含广泛的数据集和卷的信息的编目。这些信息被 VSAM 用来定位数据集、分配和取消分配存储空间、验证程序或操作者访问数据集的权限以及收集数据集的使用统计信息。

master IODF （主 IODF）
指在中央保存的 IODF，包含有多个系统甚至是一整个管理架构的 I/O 定义。主 IODF 可协助维护 I/O 数据一致性，以及提供综合报告。

master trace （主跟踪）
指主计划的集中数据跟踪能力，用来服务于 z/OS 的信息处理部分。

MB
兆字节（megabyte）。

MCS
（1）多控制台支持。（2）配置控制语句（在 SMP/E 中）。

MCS console （MCS 控制台）
指一个定义到 z/OS 的非 SNA 设备，在本地添加到 z/OS 系统之中，用来输入命令和接收消息。

megabyte (MB) （兆字节）
2 的 20 次方字节，1048576 字节。

member （成员）
指分区数据集或扩展分区数据集的一个分区。

message processing facility (MPF) （消息处理能力）
指一个用来控制消息的保持、抑制和表达的功能。

message queue （消息队列）
指一个等待处理或是等待被发送到终端的消

息队列。

message text （消息文本）
消息的一部分，包含了发送给终端用户或程序的实际信息。

microcode （微码）
存储的微指令，对用户不可用，可以执行特定的功能。

microprocessor （微处理器）
指配置给一个或少量芯片的处理器。

migration （移植）
指一种工作，通常由系统编程人员执行，涉及到升级一个新版本的程序或是替代早先版本的程序。这些活动的完成可以确保系统中的应用程序和资源可以在新的版本水平正常工作。

modification control statement (MCS) （修正控制语句）
指一个 SMP/E 控制语句，用来将 SYSMOD 打包。配置控制语句描述了程序的元素和程序与其他可能安装在同一系统中的程序之间的关系。

modification level （修正等级）
自从上一个修正等级以来，发送的所有临时补丁。修正等级的改变并不添加新的功能，而对于修正等级所适用的版本，也并不会变更所支持的程序类别。对照 *release（可能包含新主题的发行）*和 *version（同一发布主题的不同版本）*。一旦派送出一个新的程序 release，那么修正等级将被置 0，而每当发布一个新的程序 version 时，修正等级加 1。

module （模块）
编译源代码之后产生的对象。一个模块是不能运行的。模块必须被绑定到一个程序中才可以运行。

monoplex
指使用 sysplex couple 数据集，但却只含有一个系统的系统综合体。

multi-access spool configuration （并行访问缓冲配置）
意指多个系统共享作业输入子系统 2 的输入队列、作业队列以及输出队列（通过一个检验点数据集或耦合设施来实现）。

multiple console support (MCS) （多控制台支持）
指 z/OS 系统中的操作接口。

Multiple Virtual Storage (MVS) （多虚存储）
早期版本的 z/OS 操作系统。

multiprocessing （多重处理）
指同时执行一个或更多计算机程序或指令队列。参见 *parallel processing（并行处理）*。

multiprocessor (MP) （多处理器）
指可以被物理上分割成两个工作处理器联合的一个中央处理器集群，

multisystem application （多系统应用程序）
指在多系统环境下，一个拥有多种跨 z/OS 镜像功能的应用程序。

multisystem console support （多系统控制台支持）
在一个 sysplex 中，多个控制台支持着一个或更多个系统。多系统控制台支持允许 sysplex 中不同系统的控制台之间进行通信（发送消息和接收命令）。

multisystem environment （多系统环境）
指在一个或更多处理器上布置了两个或更多的 z/OS 镜像的环境，其中不同镜像的程序之间可以进行通信。

multisystem sysplex （多系统 sysplex）
指一个 sysplex，其中的两个或更多个 z/OS 镜像允许被初始化来构成 sysplex 的一部分。

multitasking （多任务处理）
指支持并行、交叉存取的操作模式，允许两个或更多任务或线程同时执行。*与 multithreading（多线程）同义*。

MVS
多虚存储（Multiple Virtual Storage）。

MVS/ESA™

多虚存储/企业系统架构（Multiple Virtual Storage/Enterprise Systems Architecture）

N

n-way
指在一个中央处理器集群中，中央处理器的数目。例如，一个 6-way 中央处理器集群包含 6 个中央处理器。

NCP
网络控制程序（Network Control Program）。

network （网络）
数据处理产品的集合之间，通过通信线路连接起来，实现信息交换。

Network File System （网络文件系统）
z/OS 的一个组件，允许工作站、个人电脑，或任何其他在 TCP/IP 网络上的系统远程访问 z/OS 主处理器数据。这些系统使用客户端软件来适应网络文件系统协议。

network job entry (NJE) （网络作业输入）
指作业输入子系统 2 的一个功能，在可通信的作业输入子系统之间支持选定的作业、系统输出数据、操作命令以及消息的传递。其可通信的作业输入子系统是通过双同步通信线路、通道至通道适配器和共享队列连接的。

network operator （网络操作员）
（1）负责控制电信网络的操作的人员。（2）指一个拥有足够权限提交网络操作命令的 VTAM 应用程序。

next sequential instruction （下一顺续指令）
指在没有任何程序分支或控制传递的情况下，将被按顺序执行的下一个指令。

NIP
内核初始化程序（nucleus initialization program）。

nonpageable region （不可页调度区域）
指在 MVS 中，不可页调度的动态区域的一部分，用来分配给执行期间不会进行页调度的作业步或系统任务。在不可**页调度**区域中，每个虚拟地址都与其实际地址相同。*与 V=R region（V=R 区）同义*。

nonreentrant （不可重入程序）
指一类不能被多用户共享的程序。

nonstandard labels （非标准标签）
指不符合美国国家标准标签规范或 IBM System/370 标准标签规范的标签。

nucleus （内核）
控制程序的一部分，通常存在于中央存储之中。

nucleus initialization program (NIP) （内核初始化程序）
指 z/OS 初始化控制程序的阶段；允许操作者对于初始化期间的某个选项作出最后一刻变更（last minute change）。

null
空；没有任何意义。

O

object deck （目标卡片叠）
由汇编程序或编译器产生的一个或更多控制段的集合，用来作为链接编辑器或绑定器的输入。也被称为对象代码或简单的称为 OBJ。

object module （对象模块）
从语言转换器（例如编译器或汇编程序）输出的模块。对象模块以可重定位的格式存在，包含不可执行的机器码。要执行对象模块，必须先用链接编辑器进行处理。

offline （脱机）
指设备不受处理器控制的状态。

offset （偏移量）
从记录、区域或是控制块中的一个预期的起始点到另外一个点的度量单位的数目。

online （在线）
指用户可以与计算机进行交互这样一种状态。

operating system （操作系统）

控制程序运行的软件；另外，操作系统可能提供诸如资源分配、计划、I/O 控制以及数据管理等等服务。尽管操作系统主要是软件实现的，但是在局部以硬件实现也是有可能的。

operations log （操作日志）
在 z/OS 中，操作日志记录了 sysplex 中每个系统的通信和系统故障信息，是一个核心的记录。

operator commands （操作员命令）
系统操作员用来获取信息、变更操作、启动新操作或是终止操作的语句。

operator message （操作员消息）
来自操作系统的消息，引导操作者完成指定的功能，例如插入磁带；或提示操作者系统特定的状态，例如一个错误状态。

OS/390
早期版本的 z/OS 操作系统。

output group （输出组）
一系列作业的输出数据集，具有相同的特征，例如级别、目的地以及外部复写器。

output writer （输出复写器）
作业计划的一部分，将指定的输出数据集转录至系统输出设备。该设备不依赖于产生输出数据集的程序。

overlay （覆盖）
在存储中现有数据之上重写。

P

page （页）
（1）在虚拟存储系统中，意指存储在一个定长块中的指令、数据或是两者都有，可以在中央存储和外部页存储之间传输。（2）指在中央存储和外部页存储之间传输指令、数据或是两者都有。

page fault （页错误，也称缺页）
意指 z/OS 或 System/390 虚拟存储系统中的一个程序中断，当一个活动页引用一个标记为"不在中央存储"的页的时候，该中断就会发生。

pageable region （可页调度区）
意指 MVS 中动态可页调度区的一部分，用来分配给在执行期间可进行页调度的作业步或系统任务。*与 V=V region（V=V 区）同义。*

paging （页面调度）
在 z/OS 中，意指在中央存储和外部页存储之间传输页的处理过程。

paging device （页面调度设备）
意指在 z/OS 中存放页（也有可能是其他数据）的 DASD。

parallel processing （并行处理）
许多服务器同时处理工作单位。工作单位可以是交易或是大量工作的一部分（批量工作）。参见 *highly parallel（高并行度）*。

Parallel Sysplex （并行系统综合体）
拥有一个或更多 CF 的 sysplex。

parameter （参数）
例程接收的数据项。

parmlib （参数库）
SYS1.PARMLIB PDS 数据集中所有成员，其中包含了 z/OS 行为的限制和控制参数。

parmlib member （参数库成员）
SYS1.PARMLIB PDS 数据集的成员之一，包含了 z/OS 行为的限制和控制参数。

partially qualified data set name （部分限定的数据集名）
限定符并未输入全的数据集名。星号和百分号可用来代替这些未定义的限定符。

partitionable CPC （可分割的中央处理器集群）
指一个可以被分割为两个独立中央处理器集群的中央处理器集群。参见 physical partition（物理分割），single-image mode（单镜像模式），side。

partitioned data set (PDS) （分区数据集）
直接寻址存储中的数据集，可分成被称为成员的多个区，每个成员中可以包含一整个程序、

程序的一部分、或数据。与 *program library（程序库）*同义。对照 *sequential data set（顺序数据集）*。

partitioned data set extended (PDSE)（扩展分区数据集）
一个由系统管理的数据集，就像一个 PDS（分区数据集）一样包含索引目录和成员。扩展分区数据集可以用来代替分区数据集。

partitioning
从一个配置中形成多个配置的过程。

password （密码）
仅有系统和用户知晓的特有的字符串。用户必须通过输入密码之后才可访问系统和更改密码内容。

PC
个人电脑（personal computer）。

PCHID
物理通道标识符（physical channel identifier）。

PE
参见 *program error PTF（程序错误临时补丁）*。

peer-to-peer remote copy (PPRC) （对等远程复制）
意指 DASD 控制器子系统见得直接连接，主要用来提供热等待能力。这些点对点的连接可以是 DASD 控制器之间互连，或者他们可以通过开关传递，就如从 CHPID 到控制单元之间的连接。

percolate （过滤）
当状态处理器返回的值指示它已不能掌握该状态的时候，状态管理器所采取的对应动作。之后该状态会被传送至下一个状态处理器。

performance administration （性能管理）
在明确的安装的商业目标指示下，定义和调整工作负载管理的目标和资源组的过程。

permanent data set （永久数据集）
一个用户命名的数据集，一般保留的比作业或交互对话的持续时间长一些。对照 *temporary data set（临时数据集）*。

PFK capability （PFK 能力）
在一个显示控制台上，指示所支持的程序功能键。程序功能键在系统产生时已指定好。

physical channel identifier (PCHID) （物理通道标识符）
硬件中通道路径的物理地址。逻辑 CHPID 拥有相应的物理通道。实际的 I/O 硬件是通过物理通道添加到处理器上的。通道拥有一个物理通道标识符（PCHID），决定其在处理器中的物理地址。PCHID 是一个分配给处理器的 3 位 16 进制数。

physical partition （物理分区）
中央处理器集群的一部分，像一个独立的中央处理器集群一样运作，拥有自己操作系统。

physical unit (PU) （物理单元）
（1）SNA 终端的控制单元或簇控制器。（2）控制单元或簇控制器的一部分，使物理单元满足 SNA 为其定义的角色。

physically partitioned (PP) mode （物理分区模式）
指当硬件单元被分为 2 个隔离的操作配置时，处理器集群所处的状态。处理器控制器的 A 侧控制 0 侧，处理器控制器的 B 侧控制 1 侧。对照 *single-image (SI) configuration （单镜像配置）*。

PL/I （PL/I 语言）
主要用在科研和商业的高级语言。PL/I 是一个强大的面向过程语言，尤其适合解决复杂的科研问题，或是长时间运行的复杂商业交易和保存记录应用程序。

platform （平台）
程序运行的操作系统环境。

PLPA
可调页的连接装配区(pageable link pack area)。

pointer （指针）
一个位置的地址或是其他关于地址的表示。

portability （移植性）
将一个应用程序从一个平台迁移到另外一个平台，对其源代码基本不作改动。

Portable Operating System Interface (POSIX) （可移植的操作系统接口）
意指计算机环境下的可迁移的操作系统接口，由 IEEE 管理的接口标准，基于 UNIX。可迁移的操作系统接口并不是一个产品，而是一个进化的标准家族，描述了从 C 语言到系统管理外壳接口等广泛的操作系统组件。

POSIX
可**移植**的操作系统接口（Portable Operating System Interface）。

PPRC
对等远程复制（peer-to-peer remote copy）。

PPT
z/OS 中的程序属性表。

preprocessor （预处理器）
为将要执行的预处理程序语句检查应用程序源代码的例程，会对源代码进行变更。

preventive service （预防服务）
（1）PTF 的大量安装，以避免再次发现这些 PTF 修正的 APAR。（2）在程序更新磁带上发布的 SYSMOD。

preventive service planning (PSP) （预防服务计划）
对于一个产品或是一个服务等级的安装推荐和 HOLDDATA。预防服务计划可以从 IBM 支持中心获得。

primary key （主键）
一个数据记录中的一个或更多字符，用来标识数据记录或控制数据记录的使用。主键必须是特有的。

printer （打印机）
一种将系统输出数据写到纸上或是其他媒介的设备。

procedure （过程）
一系列自我包含的高级语言语句，执行一个特定的任务，再返回给调用者。，每种计算机语言都为过程概念指定了不同的名字。在 C 语言中，过程被称为函数。COBOL 语言中，过程是一个节或是一个段，只能在程序内实现。在 PL/I 语言中，过程是一个被命名的代码块，可以从外部调用，通常通过 call 来调用。

processor （处理器）
指物理处理器或是机器，拥有与其联合的一个序列号、一系列通道以及一个逻辑处理器。逻辑处理器有很多通道路径 ID，或 CHPID，逻辑上与通道数量相等。逻辑处理器可以被分为很多个逻辑分区。

processor storage
参见 *central storage（中央存储）*。

program error PTF （程序错误临时补丁）
一个用来抑制错误的临时补丁。PE-PTF 可以通过++HOLD ERROR 语句连同第一次报告该错误的 APAR 来标志。

program fetch （获取程序）
将程序读取到特定的存储位置，重新调整每个可重定位的恒定地址，为程序的执行做准备。

program library （程序库）
分区数据集或是扩展分区数据集，通常包含已命名的成员。

program management （程序管理）
意指一系列的任务，包括为程序执行做准备、保存程序、装入模块或程序对象到程序库中去、以及在操作系统上执行程序。

program module （程序模块）
绑定器的输出。程序对象和装入模块的统称。

program object （程序对象）
所有或一部分计算机程序，具有适合被读入虚拟存储从而准备执行的形式。程序对象存储在扩展分区数据集程序库中，比装入模块的约束更少。程序对象由绑定器产生。

processor controller （处理器控制器）
为中央处理器提供支持和故障分析功能的硬件。

Processor Resource/System Manager™ (PR/SM)　（处理器资源/系统管理器）
允许处理器同时使用几个 z/OS 镜像，提供逻辑分区能力的特性。参见 LPAR（逻辑分区）。

profile
一种数据集，用来描述用户、用户组或一个或更多计算机资源的显著特性。

program function key (PFK)　（程序功能键）
显示设备键盘上的键，向程序传递一个信号，调用一个特定的程序操作。

program interruption　（程序中断）
程序因为某个时间中端执行，例如：例外操作、例外指数溢出或例外寻址。

program level　（程序等级）
指修正等级、可能包含新主题的发行、同一发布主题的不同版本以及补丁等级。

program management　（程序管理）
系统内部功能，当一个程序被调用时，为其执行所需的激活和调用工作，使其能够应用于运行时环境。

program mask　（程序掩码）
在程序状态字的 20 到 23 位，一个 4 位的结构，每一位分别控制着定点溢出、浮点溢出、指数溢出以及可能导致程序中端的显著例外。程序掩码的位可以被操纵，来允许或禁止程序中断的存在。

program number　（程序号）
7 位代码（以 xxxx-xxx 的形式），被 IBM 用来标识每个特许程序。

program object　（程序对象）
所有或一部分计算机程序，具有适合被读入虚拟存储从而准备执行的形式。程序对象存储在扩展分区数据集程序库中，比装入模块的约束更少。程序对象由绑定器产生。

program status word (PSW)　（程序状态字）
中央存储内的 64 位结构，用来控制指令执行顺序，以及保持和指示计算系统对于特定程序的状态。

program temporary fix (PTF)　（程序临时补丁）
对于一个 IBM 检测出的故障的临时解决办法或是避开该故障的方法。故障是由当前版本程序的缺点造成的。

PSP
预防服务计划（preventive service planning）。

PSW
程序状态字（program status word）。

PTF
程序临时补丁（program temporary fix）。

Q

QSAM
排队顺序访问方式（queued sequential access method）。

qualified name　（限定名）
一个数据集名，以字符串和分隔字符串的句点组成；例如，”TREE.FRUIT.APPLE”就是一个限定名。

qualifier　（限定符）
限定名的修饰成分，而不是最右边的名字。例如，”TREE” 和 ”FRUIT” 是”TREE.FRUIT.APPLE”的限定符。

queue　（队列）
一列或表，由很多条目组成，在系统中等待处理。

queued sequential access method (QSAM)　（排队顺序访问方式）
基本顺序访问方式的扩展版本（BSAM）。等待处理的输入数据块或等待传输至辅存的暑促数据块被放至系统队列中，来使 I/O 操作延迟最小化。

R

RACF

资源访问控制工具（Resource Access Control Facility）。

RDW
记录描述字（record descriptor word）。

read access （读访问）
允许读取信息。

reader （阅读器）
一个从输入设备或数据库文件读取作业，并将作业放入作业队列的程序。

real address （实际地址）
指在虚拟存储系统中，中央存储某个位置的地址。

real storage （实际存储）
参见 *central storage（中央存储）*。

reason code （原因码）
描述尝试操作失败或仅部分成功的原因的返回码。

receive （接收）
在 SMP/E 中，从 SMPPTFIN 和 SMPHOLD 中读取 SYSMOD 和其他数据集，同时为了随后的 SMP/E 处理而将它们存储于全局区。以上操作可使用 RECEIVE 命令完成。

RECEIVE processing （RECEIVE 处理）
对于安装新产品库而言必需的 SMP/E 处理。在此过程中，以未载入的分区数据集的形式组织起来的代码将被载入临时的 SMPTLIB 数据集。SMP/E RECEIVE 处理会自动分配对应于磁带上文件的临时分区数据集，并将磁带上的文件从磁带载入。

RECFM
记录格式（record format）。

record （记录）
（1）一组联系的数据、字或当作一个单位对待的域，例如一个名字、地址以及电话号码。（2）一个关于单一对象的自我包含的信息集合。一条记录是由很多截然不同的条目构成的，被称作域。很多外壳程序（例如 awk、join 和 sort）被设计用来处理记录组成的数据，记录是用新的行分隔的，其中每条记录包含很多的域，域之间是由空格或某些其他字符分隔的。awk 还可以处理由字符而不是新的行分隔的记录。参见 *fixed-length record（定长记录）*，*variable-length record（变长记录）*。

record data （记录数据）
具有面向记录的结构的数据，可以一条接一条记录的访问。这种数据结构是 z/OS 和其他主机操作系统上典型的数据集结构。参见 *byte stream（比特流）*。

recording format （记录格式）
对磁带卷来说，磁带上数据的格式。例如 18、36、128 或 256 个 track。

recovery （恢复）
在数据受损或毁坏之后重新建立数据的过程。通常是通过将日志中记录的交易重新应用于数据的备份版本的方法进行数据恢复。

recovery system （恢复系统）
当主应用系统不再可用的时候，用来替代主应用系统的系统。在恢复系统中，应用系统的数据必须可用。通常是同错备份和恢复技术来实现，或是通过不同 DASD 之间的复制技术，例如远程复制。

recursive routine （回归例程）
可被本身进行调用的例程，或是可被已经调用过的例程反过来调用自己的例程。

redundant array of independent disk (RAID) （独立磁盘冗余阵列）
一种磁盘子系统架构，将两个或更多物理磁盘存储设备整合为一个单独的逻辑设备，来完成数据冗余。

reenterable （可重入的）
一种复用属性，允许一个程序同时被一个或更多任务同时使用。如果适当的序列存在，可以防止资源冲突，那么可复用的模块可以修正自己的数据或其他共享的资源。

reentrant （可重入的）
一个例程或应用程序的属性，允许多于一个的用户共享一个装入模块的副本。

refreshable （可刷新）
一种复用属性，允许程序被一个新的副本替代（刷新），却并不影响其运行。在执行期间，可刷新的模块不能被本身或是其他模块所修改。参见 *reusability（复用性）*。

register （寄存器）
计算机内部组件，可以存储指定数量的数据，并可以迅速的接收和传输该数据。

register save area (RSA) （寄存器保存区）
用来保存寄存器内容的主存的区域。

related installation materials (RIMs) （相关安装材料）
IBM 为供客户定制而派送的材料，由 IBM 开发，包括面向任务的文档、作业、样本退出例程、过程、参数以及实例。

release （发行）
一个新产品的发行，或对于一个已存在的产品推出新功能以及 APAR 补丁。对照 *modification level （修正等级）*和 *version （同一发布主题的不同版本）*。

remote copy （远程拷贝）
基于存储的灾难恢复和工作负载迁移功能，可以实时的从远程地点复制数据。远程复制有两个选项可用。参 *peer-to-peer remote copy（同级远程复制）*和 *extended remote copy（扩展远程复制）*。

remote job entry (RJE) （远程作业输入）
提交自远程终端的作业控制语句和作业数据，使所描述的作业被列入计划和执行。

remote operations （远程操作）
来自远程地点的操作。

reversed storage allocation （逆向存储分配）
可以被动态的配置的中央存储和扩展存储的数量，对一个逻辑分区可以指定以上存储是在线还是脱机。

residency mode (RMODE) （常驻模式）
程序模块的属性，指定当载入一个模块后，该模块是否必须存放于 16MB 虚拟地址线以下，或是可以存放在虚拟存储的任何位置。

Resource Access Control Facility (RACF) （资源访问控制工具）
IBM 安全管理产品，通过对系统标志和验证一个用户、对保护的资源进行访问授权、记录探测到的未授权的进入系统的企图以及记录探测到的对保护资源进行的访问来提供访问控制能力。

resource recovery services (RRS) （资源恢复服务）
z/OS 系统组件，提供了资源管理器用来保护资源的服务。RRS 是 z/OS 系统级别的同步管理器。

RESTORE
SMP/E 命令，用来从目标库移除已应用的 SYSMOD。

restore （移除）
在 SMP/E 中，使用 RESTORE 命令从目标库移除已应用的 SYSMOD。

restructured extended executor (REXX) （重构的可扩充执行器）
一个为终端用户个人编程使用的多种用途的过程化语言，为方便普通用户和计算机专家而设计。同时，它还可用于应用程序宏。REXX 拥有通过这些宏和过程向操作系统输入命令的能力。

resynchronization （同步）
以 track 为单位，从首选卷至次要卷进行的镜像复制，只复制上次复制以来被更改的 track。

return code （返回码）
一个例程产生的代码，用来指示它的执行成功或失败。可以被用来影响之后指令或程序的执行。

reusability （复用性）
模块或段的属性，指示它可被地址空间内多个任务所重用或共享的内容。参见 *refreshable（可刷新）*，*reenterable（可重入）*，以及 *serially reusable（可连续复用）*。

RIM
相关安装材料（related installation material）。

RJE
远程作业输入（remote job entry）。

RMF
资源度量工具（Resource measurement Facility）。

RMODE
驻留模式（residency mode）。

rollback （回滚）
回复被应用程序更改的数据至上次保存的状态。

routine （例程）
（1）被程序调用的程序或指令队列。例程通常有多种用途，经常被用到。CICS 和编程语言都使用例程。（2）封装了过程化逻辑语句和 SQL 语句的数据库对象，存储在数据库服务器上，可以通过 SQL 语句或 CALL 语句调用。例程主要分 3 类：过程、函数和方法。（3）指在 REXX 中的一系列指令，通过 CALL 指令或包含 CALL 的函数来调用。一个例程可以在用户程序之中，也可以在用户程序之外。（4）程序中的一系列语句，使系统执行一个或一系列相关的操作。

routing （路由）
通信路径的分配，通过所分配的路径，消息可以被送达目的地。

routing code （路由码）
分配给操作者的消息的代码，用来将消息发送至适当的控制台。

RSA
寄存器保存区（register save area）。

run （运行）
执行程序、实用程序或其他机器功能。

run time （运行时）
包含正在运行的程序的实例。与 *execution time（执行时间）*同义。

run-time environment （运行时环境）
用来支持程序执行的一系列资源。与 execution environment（执行环境）同义。

S

SAF
系统授权工具（system authorization facility）。

SAP
系统协助处理器（System Assistance Processor）。

save area （保存区）
主存的一个区域，用来保存寄存器内容。

SDSF
系统显示和搜索工具（System Display and Search Facility）。

security administrator （安全管理员）
一个编程人员，负责管理、保护和控制对敏感信息的访问。

sequential data set （顺序数据集）
（1）记录按照连续物理位置组织的数据集，例如磁带。对照 *direct data set（直接访问数据集）*。（2）内容是按照连续物理顺序排列的数据集，作为一个实体保存。数据集可以保护数据、文本、整个或部分程序。对照 *partitioned data set (PDS)（分区数据集）*。

serially reusable （可连续复用）
可重用属性，允许程序被队列中多于一个的任务执行。可连续复用的模块不能在上一任务退出之前再允许新的任务进入。参见 *reusability（复用性）*。

server （服务器）
（1）在网络中包含程序、数据或提供服务的计算机，相同网络中的其他计算机可以进行访问。（2）接收远程过程调用的一方。对照 *client（客户）*。

server address space （服务器地址空间）
任何作为交易管理器或资源管理器工作的地址空间。例如，服务器地址可以是 CICS AOR，或是一个 IMS 控制区。

service （服务）

PTF 和 APAR 补丁。

service level （服务等级）
一个组件的 FMID、RMID 以及 UMID 值。服务等级标志着组件的拥有者、替换组件的最新的 SYSMOD 以及上次替换动作以来，所有对组件进行更新的 SYSMOD。

service level agreement (SLA) （服务等级协议）
提供给计算机安装用户的关于信息系统服务而签署的协议。

service processor （服务处理器）
处理器集群中负责维护集群的部分。

service unit （服务单位）
一个工作请求所消费的服务的数量，依服务定义系数、CPU、SRB、I/O 以及存储服务单元来计算。

session （会话）
（1）一个时间段，在此时间内终端可以与交互系统进行通信；通常，经历的时间从终端登录系统至终端登出系统。（2）程序或设备之间可以进行通信的时间段。（3）在 VTAM 中，节点连接到一个应用程序的时间段。

severity code （严重性提示码）
操作者消息的一部分，指示错误状态的严重程度（I、E 或 S）。

shared DASD option （共享 DASD 选项）
一个选项，使独立运行的计算机系统可以共同使用存放于共享 DASD 上的通用数据集。

shared storage （共享存储）
存储的一个区域，该区域对于每个虚拟地址空间而言都是相同的。存储于共享存储的信息可以被共享，不需要被加载到用户区，因为共享存储对于所有用户是一个相同的空间。

side （站点）
物理分区形成的配置之一。

SIGP
信号处理器（signal processor）。

simultaneous peripheral operations online (spool) （同时外围联机操作）
当作业正在运行时，以利于之后的处理或输出操作的格式，在辅存设备上并行的读写输入输出流。

single point of control （单点控制）
sysplex 的一个特性，可以允许用户在一个工作站上完成给定的一系列任务，即使需要多个 IBM 和其他厂商的产品来完成某一部分。

single-image (SI) mode （单镜像模式）
一种多处理器系统的运作方式，允许其像一个中央处理器集群一样工作。依定义，在单镜像模式中只有一个单处理器在工作。对照 physical partitioned (PP) configuration（物理分区配置）。

single-processor complex （单处理器集群）
一种处理环境，仅有一个处理器（计算机）访问缓冲区并且包含所有的节点。

single system image （单系统镜像）
一种产品特性，指产品的多个镜像可以当作一个镜像来显示和管理。

single-system sysplex （单系统 sysplex）
一种 sysplex，在这种 sysplex 中只有一个 z/OS 可被启动。在单系统 sysplex 中，XCF 在系统中提供 XCF 服务，却不提供多个 z/OS 系统之间的信号传输服务。参见 multisystem sysplex（多系统 sysplex）。

SLA
服务等级协议（service level agreement）。

small computer system interface (SCSI) （小型计算机系统接口）
标准硬件接口，允许多种外围设备之间进行通信。

SMF
系统管理工具（system management facilities）。

SMP/E
系统修改/扩展（System Modification Program/Extended）。

SMPCSI
SMP/E 数据集，包含关于用户系统的结构，以及在用户系统上安装操作系统所需的信息。SMPCSI DD 语句特别涉及到包含全局区的 CSI。也被称为主 CSI。

SMS
存储管理子系统（Storage Management Subsystem）。

SNA
系统网络架构（System Network Architecture）。

software （软件）
（1）数据处理系统的整个或部分程序、过程、规则以及相关联的文档。（2）有关于数据处理系统的一系列程序、过程，还有可能是关联的文档。例如，编译器、程序库、使用手册和电路图。对照 *hardware（硬件）*。

sort/merge program （排序/合并程序）
处理程序，用来对指定的队列进行排序或合并。

source code （源代码）
由源程序语言编写的，作为编译器或汇编器的输入。

source program （源程序）
以一种编程语言编写的一系列指令，在运行之前必须被转换为机器语言。

spin data set
在关闭的时候被取消分配的一个数据集（可被打印）。Spin off 数据集支持是为作业结束之前创建的输出数据集提供的。

spool
同时外围联机操作（simultaneous peripheral operations online）。

spooled data set （spooled 数据集）
被写到辅存设备的数据集，被 JES 管理。

spooling （假脱机处理）
当作业正在运行时，以利于之后的处理或输出操作的格式，在辅存设备上并行的读写输入输出流。

SPUFI
使用文件输入的 SQL 处理（SQL Processing Using File Input）。

SREL
系统版本标识符（system release identifier）。

SRM
系统资源管理器（system resources manager）。

SSID
子系统标识符（subsystem identifier）。

started task （启动的程序）
指通过 START 命令启动的一个 z/OS 中的地址空间，以无人看守的方式运行。启动的程序通常用于具有相当重要性的应用程序。在 UNIX 中与之对应的概念是 Daemon。

status-display console （状态显示控制台）
MCS 控制台，可以接受系统状态的显示，但操作者不能向其中输入命令。

step restart （作业步重启动）
从一个作业步的最初开始的重启动。这种重启动可以是自动的或是延期的，而延期的重启动可能导致作业的重新提交。对照 *checkpoint restart（检验点重启动）*。

storage administrator （存储管理员）
数据处理中心的工作人员，负责定义、执行以及维护存储管理的原则。

storage class （存储类别）
存储属性的集合，确定性能目标和可用性需求，是由存储管理员定义的，用来选定一个可以满足以上目标和需求的设备。

storage group （存储组）
存储的卷和属性的集合，由存储管理员定义。集合可以是一组 DASD 卷或是磁带卷，或者是当作单一对象存储存储层次对待的一组 DASD 卷、光学存储或磁带卷。

storage management （存储管理）
意指这样一些活动，包括数据集的分配、存放、

监视、迁移、备份、召回、恢复以及删除。这些活动可以通过手动操作，或是使用自动进程来完成。当进行存储资源优化时，存储管理子系统可以为用户自动化这些处理。参见 *Storage Management Subsystem（存储管理子系统）*。

Storage Management Subsystem (SMS)　（存储管理子系统）
一种功能，用来自动化和集中化存储管理。存储管理员用 SMS 来描述数据分配的特性，性能以及可用性目标，数据备份和保持需求，以及通过数据类别、存储类别、管理类别、存储组和 ACS routine 定义体现的存储需求。

string　（字符串）
一个或多个 I/O 设备的集合。该术语通常指物理上的一连串单元，但也可能指集成至一个控制单元的 I/O 设备的集合。

structure　（结构）
在 CF/耦合设施上，z/OS 用来映射和管理存储的构造。参见 *cache structure（缓存结构）*，*list structure（表结构）*，和 *lock structure（锁结构）*。

subchannel set　（子通道装置）
由安装指定的结构，定义了有关通道子系统或操作系统的设备的布局。

subpool storage　（子池存储）
在特定任务的子池数之下分配的所有存储块。

subsystem　（子系统）
次要的或是附属的系统，或是编程支持，通常可以独立运作，或是与一个控制系统异步运行。例如 CICS 和 IMS。

subsystem interface (SSI)　（子系统接口）
意指一种组件，提供在 z/OS 与其作业输入子系统之间的通信。

subtask　（子任务）
z/OS 多任务的情况下，被一个更高调用次序的任务（主任务）启动和终止的任务。子任务运行并行函数，程序的这些部分可以独立于主任务和其他子任务单独运行。

superuser　（超级用户）
（1）没有操作限制的系统用户。超级用户拥有执行管理任务所需的特殊权限。z/OS 中相应的超级用户应该是处于特权模式或者超级用户模式的用户。（2）可以绕过所有 z/OS UNIX 安全检查的系统用户。超级用户拥有管理进程和文件所需的权限。

superuser authority　（超级用户权限）
指可以不受限制的访问和修正操作系统的任何部分的能力，通常与管理系统的用户相关联。

supervisor　（超级用户）
z/OS 中调整资源使用和维护处理单元操作流的部分。

support element　（支撑元件）
为中央处理器集群提供通信、监控以及故障诊断功能的硬件单元。

suspended state　（挂起状态）
由于永久误差条件或用户命令引起的，双重复制或远程复制的一对卷中只有一个设备被更新时，整个任务所处的状态。对于剩下的还在工作的设备所进行的写操作会被记录进日志。这样，当两个卷都被激活的时候，可以允许两个卷自动重新同步。

SVC
访管指令（supervisor call instruction）。

SVC interruption　（访管指令中断）。
由访管指令的执行引起的中断，会导致控制转交回管理员。

SVC routine　（访管指令例程）
控制程序例程，执行或开始一个访管指令所指定的控制程序服务。

SWA
调度程序工作区（scheduler work area）。

swap data set　（交换数据集）
用于交换操作的数据集。

switch　（开关）
提供连通能力设备，也可以将任何两个

ESCON 或 FICON 连接在一起或者断开。

synchronous messages （同步消息）
在某个恢复情形下，z/OS 系统发送的 WTO 或 WTOR 消息。

syncpoint manager （同步点管理器）
对于所保护的资源，协调处理提交的两个阶段的功能，这样所有对于数据的变更可以被提交或是收回。在 z/OS 中，RRS 可以作为系统级别的同步点管理器。同步点管理器也被称作交易管理器、同步点协调器或提交调整器。

syntax （语法）
约束一种编程语言结构的规则，同时规定编程语言语句的解释。

SYSIN
一个系统输入流；同时，在输入流中用作数据集的数据定义名。

SYSLIB
（1）用来标识目标库的子条目，在目标库中存有元素。（2）被汇编器所使用的宏的库的连接。（3）一系列被链接编译器用来确定外部引用的例程。

SYSLOG
系统日志（system log）。

SYSMOD
系统修改（system modification）。

SYSOUT
系统输出流；也是一个指示器，用于数据定义语句中，表示将要被写到系统输出单元的数据集。

SYSOUT class （SYSOUT 类）
意指这样的一类输出：以指定的特性输出，同时打印到指定输出设备上。每个系统有其自己的 SYSOUT 类，以字母 A 到 Z、数字 0 到 9 和*号指定。

Sysplex （系统综合体）
通过某个多系统硬件组件和软件服务实现相互通信和协作，来处理客户工作负载的 z/OS 系统。参见 *Parallel Sysplex（并行 Sysplex）*。

sysplex couple data set （sysplex couple 数据集）
意指一种的 couple 数据集，包含了关于整个 sysplex 范围内使用 XCF 服务的系统、组以及成员的数据。所有 sysplex 中的 z/OS 系统必须能够连接到 sysplex couple 数据集。参见 *couple data set（couple 数据集）*。

Sysplex Timer （Sysplex 时钟）
指一个 IBM 单元，在多处理器或处理器站点之间同步系统时钟。

SYSRES
系统常驻磁盘（system residence disk）。

system （系统）
硬件配置和软件操作系统的联合。通常把 z/OS 系统简称为系统。

system abend （系统异常中止）
操作系统无力处理一个例程时，产生的异常中止；也可能是由程序源代码的逻辑错误引起的。

system console （系统控制台）
指 z/OS 中添加到处理器控制器的控制台，用来启动一个 z/OS 系统。

system control element (SCE) （系统控制元件）
一种硬件，负责处理数据传输，以及处理关联到处理器各个元件之间的存储请求的控制信息。

system data （系统数据）
z/OS 或其子系统初始化所需的数据集。

system library （系统库）
存储着部分操作系统的数据集或文件的集合。

system-managed data set （系统管理的数据集）
被指定了存储类别的数据集。

system-managed storage （系统管理的存储）
由 z/OS 中的存储管理子系统管理的存储。

system management facilities (SMF)

（系统管理工具）
z/OS 组件，为评估系统使用率提供收集和记录信息的方法。

system modification (SYSMOD) （系统修改）
SMP/E 的输入数据，定义了操作系统中各元素的介绍、替代或更新，以及将要安装的相关联的分配(程序)库。系统修正是由一系列 MCS 定义的。

System Modification Program Extended (SMP/E) （系统修改/扩展）
指一个 IBM 产品，或者 OS/390 和 z/OS 的一个元件，用来在 z/OS 系统上安装软件以及软件的变更。SMP/E 统一了安装数据、允许更灵活的安装变更、提供对话接口，同时支持数据集的动态分配。SMP/E 是控制 z/OS 操作系统的主要方法。

Systems Network Architecture (SNA) （系统网络架构）
对所要传输的信息单元的逻辑结构、格式、协议以及可操作队列进行的描述，同时也指出了网络的配置控制和操作控制。

system queue area (SQA) （系统队列区）
指 z/OS 中虚拟存储为系统相关控制块所保留的区域。

T

tape volume （磁带卷）
磁带的存储空间，标记有卷标签，包含数据集或者对象以及可用的释放空间。磁带卷是在单独的一个磁带盒或者轴上的记录空间。参见 *volume（卷）*。

target library （目标库）
在SMP/E中，用于存放不同部分操作系统的数据集集合。这些数据集有时也称为系统库。

target zone （目标域）
在SMP/E中，一组VSAM记录，用以描述目标系统的宏、模块、汇编、装入模块、源模块以及从系统生成过程中的DLIB中拷贝出来的库，和在目标系统中应用的SYSMOD。

task （任务）
在多道程序或多进程环境中，由控制程序处理的一个或多个指令序列，由计算机完成一部分的工作。

task control block (TCB) （任务控制块）
一个包含与在进行中任务的有关信息和指针的数据结构。

TCB
任务控制块（task control block）。

TCP/IP
传输控制协议/网际协议（Transmission Control Protocol/Internet Protocol）。

temporary data set （临时数据集）
创建并在同一个工作中删除的数据集。

terminal （终端）
一种设备，通常配备有键盘和某种显示器，可以通过链路传送和接受信息。

terminal owning region (TOR) （终端所有区域）
专门用于管理终端网络的CICS区域。

TGTLIB
目标库（target library）。

tightly coupled （紧耦合）
多CP同时共享存储并且有一份z/OS所控制。参见*loosely coupled（松散耦合）*、*tightly coupled multiprocessor（紧耦合多处理器）*。

tightly coupled multiprocessing （紧耦合多处理）
两个计算系统同时在一个控制程序下进行操作，并共享资源。

tightly coupled multiprocessor （紧耦合多处理器）
拥有多个CP的CPU。

Time Sharing Option/Extensions (TSO/E) （分时选项/扩展）
在z/OS中允许用户交互地共享计算机时间和资源的工具。

timeout （超时）
在物理对话结束前，存储控制能够保留的"长时间忙碌"状态的时间。

TLIB
目标库（target library）。

transaction （交易）
由一个或多个交易程序执行的工作单位，包含明确的数据集合并开始明确的流程或者工作。

Transmission Control Protocol/Internet Protocol (TCP/IP)传输控制协议/网际协议
一种用于两台物理上分开的电脑之间硬件无关的通信协议。为使得两个位于不同物理网络的计算机更方便的进行通信而设计。

Transport Layer Security (TLS) （传输层安全）
一种通过网际网提供通信保密的协议。

TRK
在DD语句中，SPACE参数的子参数。它明确了空间将按磁道进行分配。

trunk cable （中继电缆）
用在两个柜之间永久连接的电缆，即便不用时也会被保留。

TSO (TSO/E)
分时选项。参见*Time Sharing Option/ Extensions（分时选项/扩展）*。

TSO/E
分时选项/扩展。

U

UCB
单元控制块（unit control block）。

UCLIN
在SMP/E中，用以开始改变SMP/E数据集的命令。实际上，改变是由随后的UCL语句所做的。

UIM
单元信息模块。

Unicode Standard （Unicode 标准）
一种全球性的字符编码标准，支持当今世界任何语言的文本的交换、处理和显示。也能支持许多中古典历史文字，并在不断的被扩展。

uniprocessor (UP) （单处理器）
只含有一个中央处理器的处理器联合体。

unit of recovery (UR) （恢复单元）
在一个节点上进行的或者作为ACID交易的一部分的改变集合。当资源管理器第一次访问节点上的保护资源时，UR就隐式地开始了。当更改资源的ACID交易的两阶段提交结束时，UR就结束了。

UNIX. See
参见*z/OS UNIX System Services（z/OS UNIX系统服务）*。

UNIX file system （UNIX 文件系统）
UNIX文件树的一部分，物理上存放在一个单独的设备或者磁盘分区上，可分别的被加载、卸载和管理。参见*hierarchical file system（分级文件系统）*。

UNLOAD
SMP/E命令，以UCL语句的形式将数据从SMP/E数据集条目中拷贝出来。

unload （卸载）
在SMP/E中，以UCL语句的形式将数据从SMP/E数据集条目中拷贝出来。使用UNLOAD命令。

unused cable （未使用电缆）
最近被分离但未被存放起来的物理电缆。

upwardly compatible （向上兼容）
无需重编译或者重连接，使得应用程序能够在后发布的z/OS上运行的能力。

user abend （用户异常终止）
由用户代码向操作系统所执行的异常终止程序的请求。对照*system abend（系统异常终止）*。

user catalog （用户目录）
与主目录同样的可选编目，并由主目录指向。

它也可以减少主目录的压力，有利于卷的更改迁移。

user exit （用户出口程序）
在应用程序的指定点进行控制的代码。用户出口程序经常用以提供额外的初始化和终止功能。

user ID （用户ID）
用户标识。

user identification (user ID) （用户标识）
由1-8个字符组成的符号，标识系统用户。

user modification (USERMOD) （用户修正程序）
由用户构建的用以修改、增加已存在功能或者增加用户定义功能的变更。USERMODs在SMP/E中使用++USERMOD语句标识。

USERMOD
用户**修正程序**。

V

V=R region
V=R 区域，与*nonpageable region（不可页调度区域）*同义。

V=V region
V=V 区域，与*pageable region（可页调度区域）*同义。

variable-length record （变长记录）
记录的长度与其他在逻辑或物理上相关的记录长度无关。对照*fixed-length record（定长记录）*。

VB
可变长块（Variable blocked）。

vendor （供应商/生产商）
给其他个人或公司提供服务或产品的个人或公司。

version （版本）
一个单独的特许程序，基于已经存在的特许程序，并通常有较大的新代码或新功能。对照*release（发布）*和*modification level（修正等级）*。

VIO
虚拟输入/输出（virtual input/output）。

virtual address space （虚拟地址空间）
在虚拟存储系统中，指派给工作、终端用户或者系统任务的虚拟存储。同见*地址空间*。

virtual input/output (VIO) （虚拟输入/输出）
只在页式存储中的数据集分配。

Virtual Storage Access Method (VSAM) （虚拟存储存取方式）
直接或线性处理存放在直接访问设备上的定长和变长记录的存取方式。在VSAM数据集或文件中的记录可以在逻辑上根据键值域（键值序列）顺序组织起来，也可以在物理上根据他们所处于的数据集或文件的位置（条目序列）或相对记录数序组织。

virtual storage （虚拟存储）
（1）可被计算机系统用户认为是可寻址主存的存储空间，其中的虚拟地址会被映射到真实的地址。虚拟存储的大小受限于计算机系统的寻址模式和可用辅存的容量，而不是由主存单元的真实数量决定的。（2）一种允许外部磁盘存储作为主存出现的寻址模式。

virtual telecommunications access method (VTAM) （虚拟远程通信存取方法）
在z/OS下，维护终端和应用程序之间通讯控制的程序集。

VM
虚拟机。

VOLSER..
卷序列号（volume serial number）。

volume （卷）
（1）在DASD、磁带或光设备上的存储空间，由卷标所标识。（2）以一种读/写机制访问的那部分存储，比如，一个磁鼓，一个磁盘组套，或者部分的磁盘存储器模块。（3）可装载和卸载的记录媒介的单位，比如，一盘磁带或者一个磁盘组套。

volume backup （卷备份）
整个卷的备份，以防卷的遗失。

volume serial number （卷序列号）
当卷在系统中准备被使用时在卷标上所指派的数字。

volume table of contents (VTOC) （卷目录表）
在直接存取存储设备（DASD）卷上的表，用来描述在该卷上数据集的位置、大小和其他特性。

VPN
虚拟专用网（virtual private network）。

VSAM
虚拟存储访问方法（virtual storage access method）。

VTAM
虚拟远程通讯访问方法（Virtual Telecommunications Access Method）。

VTOC
卷目录表（volume table of contents）。

W

WAP
无线接入点**(**wireless access point)。

wait state （等待状态）
与*waiting time（等待时间）*同义。

waiting time （等待时间）
（1）为了进入准备条件，而依赖一个或多个事件的任务条件。（2）所有操作都被暂停时处理部件的条件。

wild carding （通配符）
在分类规则中，以星号（*）代替若干个字符的使用方法。

WLM..
工作负载管理器（workload manager）。

workload （工作负载）
作为单位，被跟踪、管理和报告的工作组。

work request （工作请求）
一种工作，例如请求服务，批处理，APPC、CICS或IMS交易，TSO LOGON或TSO命令。

wrap mode （覆盖模式）
当新消息加入时，允许新旧消息之间的分隔线整屏下移的一种控制台显示模式。当屏幕已满并有新消息添加时，分隔线将覆盖最旧的消息，新的消息立即显示在线之后。

write-to-operator (WTO) message （发给操作员的消息）
传送给操作员控制台的一种信息，通知操作员可能需要改正的错误和系统情况。

write-to-operator-with-reply (WTOR) message （发给操作员且需应答的消息）
传送给操作员控制台的一种信息，通知操作员可能需要改正的错误和系统情况。操作员必须键入应答消息。

WTO
发给操作员（write-to-operator）。

WTOR
发给操作员且需应答（write-to-operator-with-reply）。

X

XA
扩展架构（Extended Architecture）。

XCF
跨系统耦合设施（cross-system coupling facility）。

Z

zAAP
zSeries应用程序协处理器

z/Architecture
IBM为大型计算机和其外设设计的体系架构。zSeries家族服务器使用的就是z/Architecture。

zFS
zSeries文件系统（zSeries file system）。

z/OS
IBM主机上被广泛使用的64位中央存储操作系统。

z/OS Language Environment （z/OS语言环境）
一种IBM软件产品，为使得与高级语言编译器保持一致而提供的常用运行时环境和运行时服务。

z/OS UNIX System Services (z/OS UNIX)（z/OS UNIX系统服务）
支持类UNIX环境的z/OS服务。用户可在传统的TSO/E界面和shell界面间转换。擅长UNIX的用户，可使用其熟悉的标准命令和工具与系统进行交互。擅长z/OS的用户，可以使用他熟悉的TSO/E命令和交互菜单来创建、管理分级系统文件，在z/OS数据集和文件间进行复制，以此与系统进行交互。应用程序程序员和用户可从他们两个界面中选择，也可以进行适当的折中，选择混合的界面。

zSeries Application Assist Processor (zAAP) （zSeries应用程序协处理器）
一种在选定的zSeries机器上配置允许运行Java程序的专用协处理部件。

zSeries File System (zFS) （zSeries 文件系统）
一种以VSAM线性数据集形式存放的z/OS UNIX文件系统。

Numerics

3270 pass-through mode （3270传递模式）
允许程序从z/OS内核传送和接受3270数据流或者发送TSO/E命令的一种模式。


IBM
新型主机介绍：z/OS基础

新型主机介绍：z/OS基础

主机基础概念，包括主机应用和主机系统架构

适合学生和初学者的z/OS基础知识

主机硬件和外围设备

这本IBM红皮书使学生了解信息系统技术，提供相应的背景知识以及一些用来使用主机基本工具的必备技能。这是向学生介绍主机概念的系列丛书中的第一本，以帮助他们为在大型系统计算领域从事相关职业做好准备。

为了取得最佳的教学效果，学生应该已经学习过计算机系统概念的导论性课程，如计算机组成原理和体系结构，操作系统，数据管理或者数据通信。他们也需要学习过一种或者多种编程语言，并熟悉PC环境。

本教材也可以用作高级课程的前修课程，或者为实习和专题研究打下基础。它并不期望覆盖主机操作的方方面面，也不是详细讨论主机工具的每个特点和选项的参考性书籍。

另外一些人也会从阅读本书中获益，这些人包括非主机平台上经验丰富的数据处理专家，或者熟悉主机的某些方面而想对主机环境的其他方面有更全面了解的人员。

国际技术支持组织

实践凝练技能

IBM红皮书由IBM国际技术支持组织负责组织撰写。来自全球的IBM专家，客户和业务合作伙伴基于真实案例编写出了这些前沿的技术文档。为您的环境提供针对性的建议，以帮助您更加高效地实施IT解决方案。

更多信息参见：
ibm.com/redbooks

SG24-6366-00 ISBN 0738495972