AS/400e 系列

ILE 概念

版本 4

SC31-2020-02

AS/400e 系列

ILE 概念

版本 4

SC31-2020-02

第三版（1998 年 9 月）

此版本适用于 “特许程序” IBM Operating System/400 （程序 5769-SS1）的版本 4、发行版 3 、修订版 0 以及所有后续发行版和修订版，直到在新版本另有说明为止。此版本只适用于精简指令集计算机 (RISC) 系统。

此版本将置换 SC41-5606-01。此版本只适用于精简指令集计算机 (RISC) 系统。

目录

关于 ILE 概念 (SC31-2020)

本书描述关于 OS/400 特许程序的“集成语言环境”(ILE) 体系结构的概念和术语。所涵盖的主题包括模块建立、联编、程序的运行和调试以及异常处理。

本书中描述的概念适合于所有 ILE 语言。每种 ILE 语言在实现 ILE 体系结构上可能有某些不同。要确定每种语言到底如何实现此处所述的概念，参考该特定 ILE 语言的程序员指南。

本书还描述直接适用于所有 ILE 语言的 OS/400 函数。还特别解释了关于联编、信息处理和调试的公共信息。

本书没有描述从现存 AS/400 语言至 ILE 语言的迁移。在每本 ILE 高级语言 (HLL) 程序员指南中都包含有此类信息。

谁应阅读本书

您应该阅读本书，如果:

- 您是开发应用程序或软件工具的软件供应商
- 您有在 AS/400 系统上开发混合语言应用程序的经验
- 您不熟悉 AS/400 系统，但是有在其他系统上进行应用程序设计的经验
- 您的程序共享公共过程，而当您更新或增强那些过程时，您不得不重新建立使用它们的那些程序

如果您是主要以一种语言编写程序的 AS/400 应用程序员，那么您应该阅读本书的前四章，以便对 ILE 及其益处有一个总的了解。在这之后，那种 ILE 语言的程序员指南就足够供您进行应用程序开发了。

先决条件和相关信息

使用“AS/400 信息中心”作为您的 AS/400 信息需要的起始点。可通过下列任何一种方法获得它:

- 在 Internet 上的以下“统一资源定位器”(URL) 地址处:

  `http://publib.boulder.ibm.com/html/as400/infocenter.html`

- 在 CD-ROM: *AS/400e series Information Center*, SK3T-2027上。

“AS/400 信息中心”包含有关重要主题（如 Java、程序临时性修改 (PTF) 以及 Internet 安全性）的可浏览信息。它还包含至相关主题的超文本链路，包括至诸如“AS/400 技术室”、“AS/400 软拷贝库”以及 AS/400 主页等 Web 站点的 Internet 链路。

有关相关出版物的列表，参见第151页的『书目』。

如何发表您的意见

您的反馈很重要，它有助于提供最准确和高质量的信息。若对本书或任何其他 AS/400 文档有任何意见，请填写本书末尾的读者意见表。

- 若喜欢通过邮件发送意见，则使用印刷在书后的带地址的读者意见表。若从美国以外的其他国家邮寄读者意见表，则可以将该表交给当地的 IBM 分部或 IBM 代表，以“邮资已付”方式邮寄。
- 若喜欢通过传真发送意见，则使用下列两个号码之一:
  - 美国和加拿大: 1-800-937-3430
  - 其他国家: 1-507-253-5192
- 若喜欢用电子邮件发送意见，则使用下列网络 ID:
  - IBMMAIL（发送至 IBMMAIL(USIB56RZ)）
  - RCHCLERK@us.ibm.com

确保包括下列各项:

- 书名。
- 书的出版号。
- 您的意见所适用的页号或主题。

第1章 集成语言环境介绍

本章定义“集成语言环境*”(ILE*) 模型，描述 ILE 的益处，并且说明如何从先前的程序模型发展了 ILE。

在可能的任何地方，从 RPG 或 COBOL 程序员的角度提供信息，并且以现存的 AS/400* 功能部件进行描述。

什么是 ILE?

ILE 是一组新的工具和相关的系统支持，是为在 AS/400 系统上增强程序开发而设计的。

只能由通过新 ILE 系列的编译器产生的程序开发这种新模型的能力。该系列包括 ILE RPG/400*、ILE COBOL/400*、ILE C/400* 和 ILE CL。

ILE 的益处有哪些?

ILE 提供了比先前程序模型多得多的益处。那些益处包括联编、模块化、可重复使用部件、公共运行期服务、共存和源调试程序。它们还包括对资源的更好控制、对语言交互的更好控制、更好的代码优化、更好的 C 环境，以及未来的基础。

联编

联编的益处是帮助减少与调用程序相关的额外开销。将模块联编在一起可以使调用加速。先前的调用机制仍可用，但还有一个较快的替代项。要区分这两种类型的调用，将前一种方法称为动态或外部程序调用，而将 ILE 方法称为静态或联编的过程调用。

联编能力和调用性能得到的改进一起，使得用高度模块化的方法开发应用程序要实用得多。 ILE 编译器不产生可运行的程序。它产生一个可以和其他模块组合（联编）在一起构成单个可运行单元的模块对象 (*MODULE)；即程序对象 (*PGM)。

正如您可以从 COBOL 程序调用 RPG 程序一样，ILE 允许您联编以不同语言编写的模块。因此，建立包含分别以 RPG、COBOL、C 和 CL 语言编写的模块的单个可运行程序是可能的。

模块化

使用模块化方法进行应用程序设计的益处包括下列各项:

- 更快速的编译时间

  编译的代码段越小，编译器处理它就越快。在维护期间此益处尤为重要，因为通常只有一行或两行需要更改。当更改两行时，可能不得不重新编译 2000 行。那不是一种有效的资源使用。

  如果将代码模块化并利用 ILE 的联编能力，那么可能仅需要重新编译 100 或 200 行。即使包括了联编步骤，此过程也快得多。
- 简化的维护

当更新非常大的程序时，要准确了解正在进行什么是很困难的。当原来的程序员用与您自己不同的风格编写程序时，更是这样。一个较小的代码段倾向于表示单个功能，并且更容易掌握其内部工作。因此，逻辑流变得更加明显，当进行更改时，更不容易导致不想要的副作用。

- 简化的测试

  较小的编译单元使您能够在隔离状态下测试功能。此隔离帮助确保测试范围是完整的；也就是说，所有可能的输入和逻辑路径都被测试。

- 程序设计资源的更好使用

  模块化有助于将工作分得更细。当编写大程序时，要细分工作是困难的（若不是不可能的话）。编码程序的所有部分可能会超过初级程序员的能力，或浪费高级程序员的技能。

- 从其他平台更容易的迁移代码

  在其他平台（如 UNIX**）上编写的程序通常是模块化的。可以将那些模块迁移至 AS/400 系统并合并到 ILE 程序中。

可重复使用的部件

ILE 允许选择可以混合到您自己的程序中的例行程序包。用任何 ILE 语言编写的例行程序可由所有的 AS/400 ILE 编译器用户使用。程序员可以用他们选择的语言编写的事实确保您对例行程序尽可能广的选择。

可将 IBM 和其他供应商向您交付这些包所使用的相同机制用于您自己的应用程序中。您的安装可以开发它自己的标准例行程序集，且以它选择的任何语言来这样做。

您不仅可以在自己的应用程序中使用现成的例行程序。还可以用您选择的 ILE 语言开发例行程序，并将它们卖给任何 ILE 语言的用户。

公共运行期服务

将现成的部件（**可联编的 API**）的选择，作为可合并到您的应用程序中的 ILE 的一部分提供。这些 API 提供如下服务:

日期和时间处理

信息处理

数学例行程序

对屏幕处理的更多控制

动态存储器分配

以后，附加例行程序将添加至此组中，而其他的将可以从第三方供应商获得。

有关随 ILE 一起提供的 API 的更多细节，参见 *System API Reference*。

与现存应用程序共存

ILE 程序可与现存 OPM 程序共存。ILE 程序可以调用 OPM 程序和其他 ILE 程序。类似地，OPM 程序可以调用 ILE 程序和其他 OPM 程序。因此，经过仔细计划，对 ILE 进行逐步的转换是可能的。

源调试程序

源调试程序允许您调试 ILE 程序和服务程序。有关源调试程序的信息，参见第99页的『第9章 调试考虑事项』。

对资源的更好控制

在引入 ILE 之前，程序所使用的资源（例如，打开的文件）被限制为只可由下列各项使用（即由下列各项所拥有）：

分配资源的程序

作业

在许多情况下，这一限制迫使应用程序设计者进行折衷。

ILE 提供了第三种选择。作业的一部分可以占有资源。这种选择是通过 ILE 结构（**激活组**）的使用来实现的。在 ILE 下，资源可以由任何下列各项使用:

程序

激活组

作业

共享的开放数据通路 – 方案

共享的开放数据通路 (ODP) 是资源的一个示例，您可以通过 ILE 的新级别的范围限定来更好地控制这些资源。

为了改进在 AS/400 上的应用程序的性能，程序员决定对客户主文件使用共享的 ODP。“订单项”和“开票”应用程序都使用该文件。

因为将共享的 ODP 限制给该作业，所以这些应用程序的其中之一无意地导致其他应用程序出现问题是十分可能的。实际上，避免这样的问题要求在这些应用程序的开发者中认真地协调。若这些应用程序是从不同的供应商处购买的，甚至可能不能避免问题。

可能发生哪种问题呢？考虑下列方案:

1. 客户主文件是按帐号设定关键字的，并且包含帐号 A1、A2、B1、C1、C2、D1、D2 等等的记录。
2. 操作员复查主文件记录，在请求下一个记录之前按要求更新每一个记录。当前显示的记录是帐号 B1 的记录。
3. 电话振铃。客户 D1 想下一个订单。
4. 操作员按下“转至订单项”功能键，处理客户 D1 的订单，并返回至主文件屏幕。
5. 程序仍正确地显示 B1 的记录，但是当操作员请示下一个记录时，显示哪一个记录呢？

   若您说是 D2，则正确。当“订单项”应用程序读取记录 D1 时，由于共享的 ODP 限制给作业，所以更改了当前文件位置。因此，请求下一个记录意味着在 D1 之后的下一个记录。

在 ILE 下，可通过在专用于“开票”的激活组中运行主文件维护来防止此问题。同样，“订单项”应用程序将在其自己的激活组中运行。每个应用程序仍然可得到共享的 ODP

的益处，但是每个应用程序将拥有其自己的、属于相关激活组的共享的 ODP。此级别的限定范围将防止此示例中所述的那种干扰。

限定资源范围至激活组允许程序员自由开发应用程序，该应用程序是独立于在作业中运行的任何其他应用程序运行的应用程序。它还减少所要求的协调工作，并且增强对现存的应用程序包编写混入信息扩充的能力。

确认控制 – 方案

将共享的开放数据通路 (ODP) 的范围限定至应用程序的能力在确认控制的区域中是有用的。

假设您想使用在确认控制下的文件，但您还需要它以使用共享的 ODP。在没有 ILE 的情况下，若一个程序打开在确认控制下的文件，则作业中的所有程序都必须也这样做。甚至当仅有一个或两个程序需要确认能力时也是这样。

这种情况下的一个潜在问题是，若作业中的任何程序发出一个确认操作，则所有更新都被确认。即使在逻辑上这些更新不是所讨论的应用程序的一部分，也会确认它们。

可通过在单独的激活组中运行该应用程序的每个需要确认控制的部分来避免这些问题。

对语言交互的更好控制

在没有 ILE 的情况下，程序在 AS/400 上运行的方式取决于下列各项的组合:

语言标准（例如，COBOL 和 C 的 ANSI 标准）

编译器的开发者

当您混合语言时此组合可能会导致问题。

混合语言 – 方案

在没有由 ILE 引入的激活组的情况下，OPM 语言中的交互是很难预计的。 ILE 激活组可以解决该困难。

例如，考虑由将 COBOL 和其他语言混合在一起所引起的问题。 COBOL 语言标准包括一个称为**运行单元**的概念。运行单元将程序组合在一起，以便在特定情况下它们作为单个实体来运作。这是一个非常有用的特性。

假设三个 ILE COBOL/400 程序（PRGA、PRGB 和 PRGC）形成一个小应用程序，其中 PRGA 调用 PRGB，PRGB 再调用 PRGC（参见图1）。在 ILE COBOL/400 的规则下，这三个程序在同一个运行单元中。结果，若其中任何一个程序结束，则所有三个程序都将结束，并且控制将返回至调用者 PRGA。

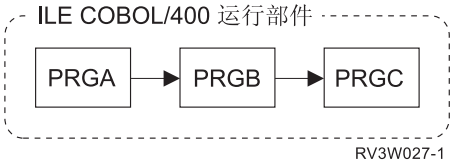

图 1. 在一个运行单元中的三个 ILE COBOL/400 程序

假设我们现在将一个 RPG 程序 (RPG1) 引入至应用程序中，并且 COBOL 程序 PRGB 也调用该 RPG1（参见图2）。 RPG 程序期望其变量、文件和其他资源保持不变，直到程序返回并且最后记录 (LR) 指示灯亮为止。

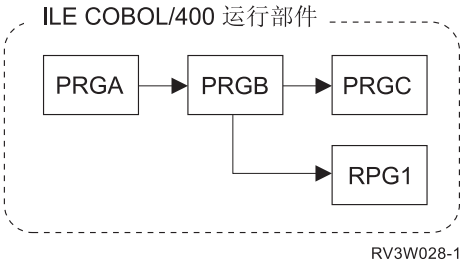

图 2. 在一个运行单元中的三个 ILE COBOL/400 程序和一个 ILE RPG/400 程序

然而，程序 RPG1 是以 RPG 编写的事实并不保证当 RPG1 作为 COBOL 运行单元的一部分运行时，所有的 RPG 语义都适用。若运行单元结束，RPG1 消失而不管其 LR 指示灯设置是什么。在许多情况下，这种情况可能正是您所想要的。然而，若 RPG1 是实用程序，可能控制发票号的发行，则这种情况是不能接受的。

可以通过从 COBOL 运行单元运行单独的激活组中的 RPG 程序来防止此情况（参见图3）。一个 ILE COBOL/400 运行单元本身是一个激活组。

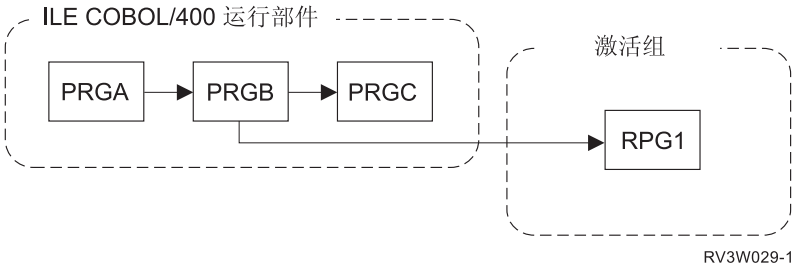

图 3. 在单独激活组中的 ILE RPG/400 程序

有关 OPM 运行单元和 ILE 运行单元之间的不同的信息，参见 *ILE COBOL for AS/400 Programmer's Guide*。

更好的代码优化

ILE 转换器可以比原始程序模型 (OPM) 转换器执行多得多的类型的优化。尽管每个编译器都执行某种优化，但在 AS/400 上的大多数优化都是由转换器来进行的。

启用了 ILE 的编译器不直接产生模块。它首先产生模块的中间形式，然后调用 ILE 转换器以将中间代码转换成可运行的指令。

更好的 C 环境

C 是一种流行的工具构建器语言。因此，更好的 C 语言意味着越来越多的最新应用程序开发工具被迁移到 AS/400。对于您来说，这意味着下列更多的选择:

CASE 工具

第四代语言 (4GL)

其他程序设计语言

编辑器

调试程序

未来的基础

在将来，ILE 提供的益处和功能将更加重要。未来的 ILE 编译器将提供重要的增强功能。当我们转向面向对象的程序设计语言和可视化程序设计工具时，对 ILE 的需要变得更为显著。

程序设计方法越来越依赖于高度模块化的方法。通过将成千上万的可重复使用的小部件组合在一起以组成完整的应用程序，可以构建应用程序。若这些部件不能在它们自身中间快速地传送控制，则结果应用程序将不能工作。

ILE 的历史是什么?

ILE 是 OS/400* 程序模型发展过程中的一个阶段。每个阶段是为满足应用程序员不断变化的需要而发展的。

首次引入 AS/400 系统时提供的程序设计环境称为原始程序模型 (OPM)。在“版本 1 发行版 2”中，引入了“扩展程序模型”(EPM)。

原始程序模型说明

AS/400 上的应用程序开发者将源代码输入源文件中并编译该源文件。若编译成功，则建立了一个程序对象。 OS/400 提供的用来建立和运行程序的函数、过程和规则的集合称为**原始程序模型 (OPM)**。

当 OPM 编译器生成程序对象时，它生成附加代码。附加代码初始化程序变量，并为特定语言所需的特殊处理提供任何必需的代码。特殊处理可能包括处理此程序期望的任何输入参数。当程序将要开始运行时，编译器生成的附加代码成为该程序的起始点（入口点）。

当 OS/400 遇到调用请求时，一般激活程序。在运行期，对另一个程序的调用是**动态程序调用**。动态程序调用所需的资源可能是很大的。应用程序开发者通常设计一个应用程序来包括几个可使动态程序调用数化为最小的大程序。

图 4 说明 OPM 和操作系统之间的关系。正如您所见到的那样，RPG、COBOL、CL、BASIC 和 PL/I 都以此模式运作。

形成 OPM 边界的断开行指示 OPM 是 OS/400 的组成部份。此集成是指通常由编译器编写者提供的许多功能都内置于操作系统中。调用约定的结果标准化允许以一种语言编写的程序自由调用以另一种语言编写的那些程序。例如，以 RPG 编写的应用程序一般包括一些 CL 程序，以发出文件覆盖、执行字符串处理、或发送信息。

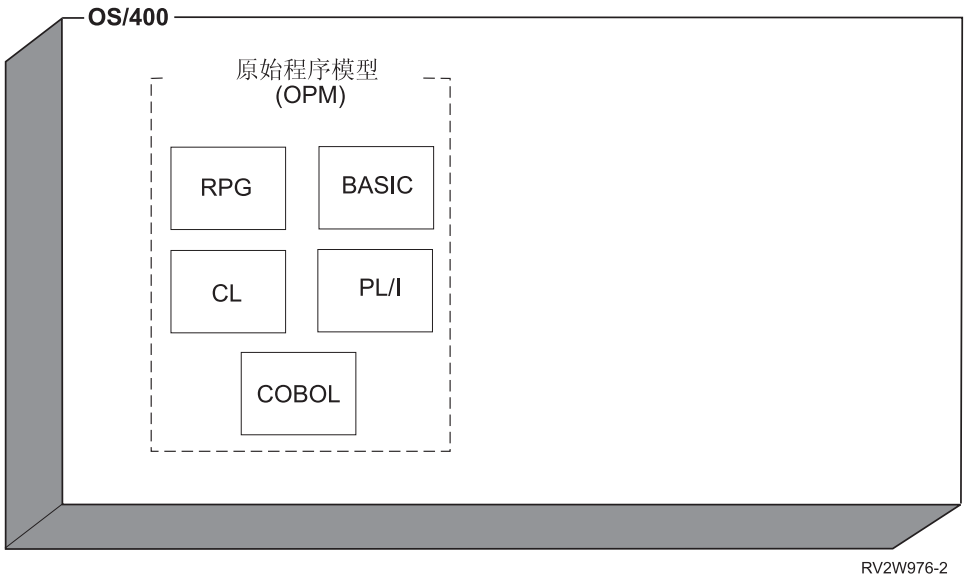

图 4. OPM 至 OS/400 的关系

OPM 的主要特性

下面的列表标识 OPM 的主要特性:

- 对传统的 RPG 和 COBOL 程序有益

  对于支持传统的 RPG 和 COBOL 程序（即相对大的、多功能的程序）来说， OPM 是理想的模型。

- 动态联编

  当程序 A 想调用程序 B 时，就可以调用。此动态程序调用是一种既简单又强大的能力。在运行期，操作系统定位程序 B 并确保用户有使用它的权利。

  OPM 程序只有一个单一入口点，然而，ILE 程序中的每个过程都可以是一个入口点。

- 限制数据共享

  在 OPM 中，内部过程必须与整个程序共享变量，而在 ILE 中，每个过程都可以拥有其自己的局部范围变量。

扩展程序模型说明

OPM 继续为有用的用途服务。然而，OPM 不象 C 语言等语言定义的那样提供对过程的直接支持。**过程**是一组自包含的高级语言 (HLL) 语句，它执行特定的任务然后返回至调用者。各种语言定义过程的方法有所不同。在 C 中，将过程称为函数。

要允许在 AS/400 上运行某些语言，它们定义编译单元之间的过程调用或定义具有局部范围变量的过程，则需要增强 OPM。这些增强称为**扩展程序模型 (EPM)**。如第8页的图5中所示，建立 EPM 是为了支持象 Pascal 的一些语言。随基本 OPM 支持一起，EPM 提供调用位于其他程序中的过程的能力。由于 EPM 使用 OPM 的功能，在 EPM 中的某些过程调用转化成对包含该过程的程序的动态调用。概念上，所调用的 EPM 程序的入口点提供下列功能:

初始化程序变量

调用所标识的过程

由“设置程序信息”(SETPGMINF) 命令提供系统支持以帮助解决对适当程序的过程调用。

尽管 EPM 是与 OPM 不同的程序设计模型，但它与 OPM 联系紧密。 EPM 是作为 AS/400 高层机器接口之上的一个附加层构建的。大多数与 OPM 相关的功能也适用于 EPM。

在本手册的以后几章中，术语 OPM 既指 OPM 又指 EPM。仅适用于 EPM 的功能用术语 EPM 进行特别限定。

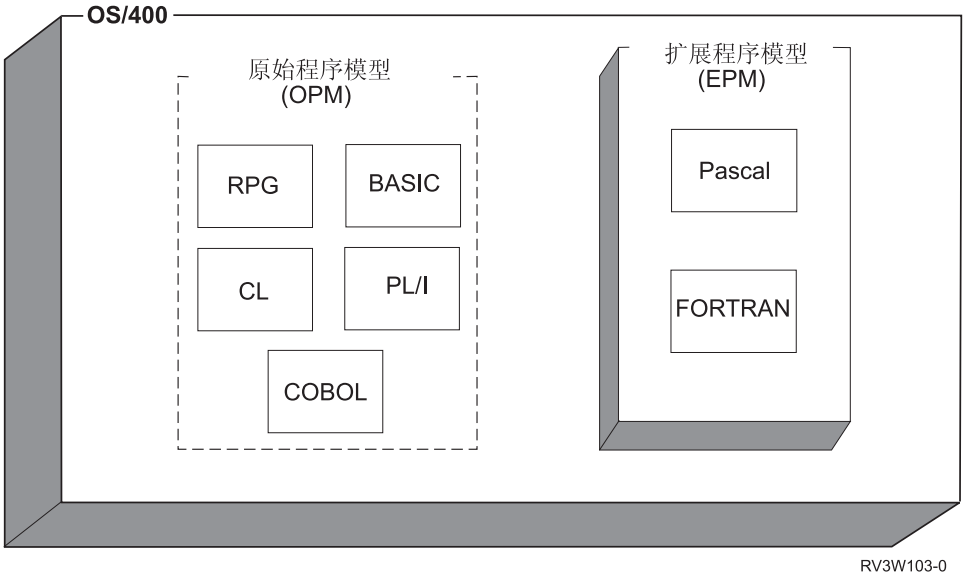

图 5. OPM 和 EPM 至 OS/400 的关系

EPM 块周围的阴影区是指 EPM 不象 OPM 那样合并至 OS/400 中。而是在操作系统上面的一层。它提供基于过程的语言所需的附加支持。

基于过程的语言的主要特性

基于过程的语言有下列特性:

- 局部范围变量

  局部范围变量仅在定义它们的过程内已知。与局部范围变量对应的是，定义表示两个不同数据段的具有相同名称的两个变量的能力。例如，变量 COUNT 在子例程 CALCYR 中可能有 4 位数字长度，而在子例程 CALCDAY 中却有 6 位数字长度。

  当您编写打算复制到几个不同程序中的子例程时，局部范围变量提供相当大的益处。在不使用局部范围变量的情况下，程序员必须使用如根据子例程的名称命名变量这样的模式。

- 自动变量

  在进入过程的任何时候都可以建立自动变量。退出过程时，自动变量被损坏。

- 外部变量

  外部数据是在程序间共享数据的一种方法。若程序 A 说明一个数据项为外部的，则是指程序 A 要将该数据项**调出**至其他想要共享该数据的程序。然后程序 D 可以**调入**该项而完全无需涉及程序 B 和 C。关于调入和调出的更多信息，参见第12页的『模块对象』。

- 多个入口点

  COBOL 和 RPG 程序只有一个单一入口点。在 COBOL 程序中，是 PROCEDURE DIVISION 的开始。在 RPG 程序中，是第一页 (1P) 输出。这是 OPM 支持的模型。

  另一方面，基于过程的语言可有多个入口点。例如，一个 C 程序可以完全由要由其他程序使用的子例程组成。可以将这些过程调出（若需要的话与相关的数据一起）以供其他程序调入。

  在 ILE 中，这种类型的程序称为**服务程序**。它们可以包括来自于任何 ILE 语言的模块。服务程序在概念上与 Windows** 或 OS/2* 中的动态链接库 (DLL) 相似。在第15页的『服务程序』中有关于服务程序的更详细的讨论。

- 频繁调用

  基于过程的语言在本质上是集中调用。尽管 EPM 提供了某些功能以使调用的额外开销减至最低，但在单独的编译单元之间的过程调用仍有相对较高的额外开销。 ILE 将十分有效地改进此类型的调用。

集成语言环境说明

如图6所示，ILE（正如 OPM 一样）紧密集成在 OS/400 中。它对基于过程的语言提供与 EPM 相同类型的支持，但是它执行得更加彻底和始终如一。它是为更传统的语言如 RPG 和 COBOL，以及为未来语言的发展而设计的。

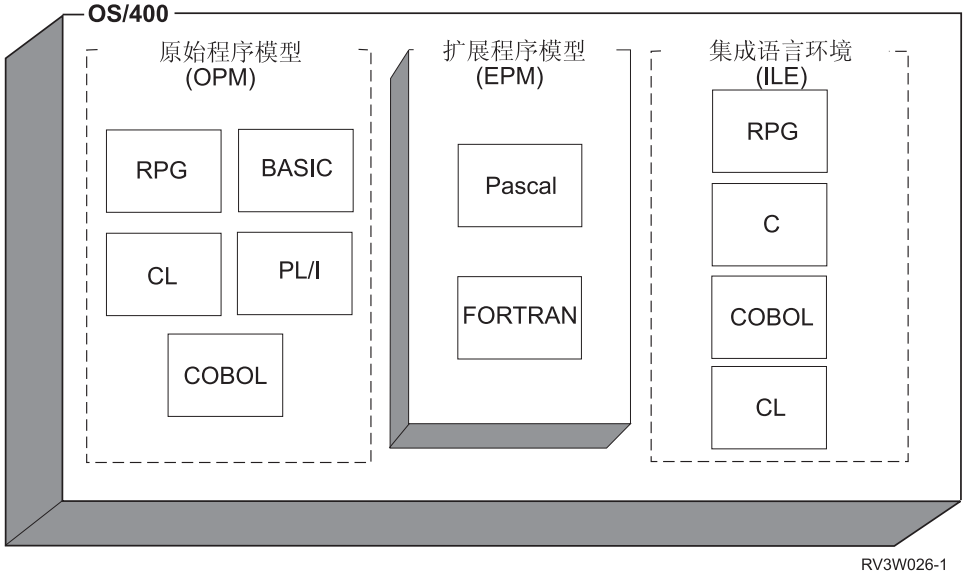

图 6. OPM、EPM 和 ILE 至 OS/400 的关系

第2章 ILE 基本概念

表1比较和对比原始程序模型 (OPM) 和 “集成语言环境” (ILE) 模型。本章简要说明在该表中列示的相同点和不同点。

表 1. OPM 和 ILE 之间的相同点和不同点

OPM	ILE
程序	程序服务程序
可运行的程序中的编译结果	不可运行的模块对象中的编译结果
编译，运行	编译、联编、运行
每种语言的模拟运行单元	激活组
动态程序调用	动态程序调用静态过程调用
单语言焦点	混合语言焦点
特定语言错误处理	公共错误处理特定语言错误处理
OPM 调试程序	源级调试程序

ILE 程序的结构

一个 ILE 程序包含一个或多个模块。依次地，一个模块包含一个或多个过程（参见 图 7）。

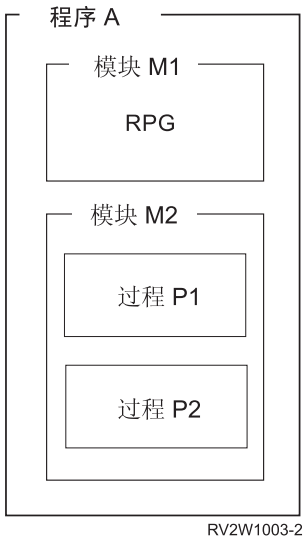

图 7. ILE 程序的结构

过程

过程是一组自包含的高级语言语句的集合，它执行特定的任务然后返回至调用者。例如，一个 ILE C/400 函数是一个 ILE 过程。

模块对象

模块对象是*不可运行的*对象，它是 ILE 编译程序的输出。对系统使用符号 *MODULE 来代表模块对象。模块对象是建立可运行 ILE 对象的基本构建块。这是 ILE 和 OPM 之间的一个显著区别。 OPM 编译程序的输出是*可运行的*程序。

模块对象可由一个或多个过程和数据项规范组成。在一个模块中从另一个 ILE 对象中直接存取过程或数据项是可能的。关于编码能够被其他 ILE 对象直接存取的过程和数据项的有关详情，参见 ILE HLL 程序员指南。

ILE RPG/400、ILE COBOL/400 和 ILE C/400 都有下列公共概念:

- 调出

  调出是过程或数据项的名称，编码在模块对象中，可用于其他 ILE 对象。用调出的名称及其相关的类型（过程或数据）来标识调出。

  调出还可称为**定义**。

- 调入

  调入是对在当前模块对象中未定义的过程或数据项的名称的使用或引用。用调入的名称及其相关的类型（过程或数据）来标识调入。

  调入还可以称为**引用**。

模块对象是 ILE 可运行对象的基本构建块。因此，当建立了一个模块对象时，还可能生成下列:

- 调试数据

  调试数据是调试正运行的 ILE 对象所必需的数据。此数据是可选的。

- 程序入口过程 (PEP)

  程序入口过程是编译程序生成的代码，该代码是动态程序调用上的 ILE 程序的入口点。它类似于为 OPM 程序中的入口点提供的代码。

- 用户入口过程 (UEP)

  用户入口过程（由程序员编写的），是动态程序调用的目标。它是从 PEP 中获得控制的过程。 C 程序的 main() 函数是 ILE 中程序的 UEP。

第13页的图8显示模块对象的概念性视图。在此示例中，模块对象 M1 调出两个过程（Draw_Line 和 Draw_Arc）和一个数据项 (rtn_code)。模块对象 M1 调入一个称为 Draw_Plot 的过程。此特定的模块对象有一个 PEP、一个相应的 UEP（过程 Draw_Arc）和调试数据。

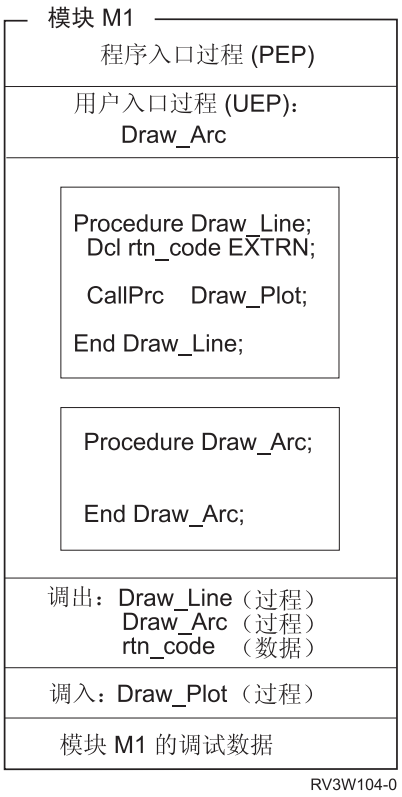

图 8. 模块的概念性视图

*MODULE 对象的特性:

- *MODULE 对象是 ILE 编译程序的输出。
- 它是 ILE 可运行对象的基本构建块。
- 它不是可运行对象。
- 它可能定义了 PEP。
- 若定义了 PEP，则也定义了 UEP。
- 它可以调出过程和数据项名。
- 它可以调入过程和数据项名。
- 它可以定义调试数据。

ILE 程序

ILE 程序与 OPM 程序共享下列特性:

- 程序通过动态程序调用获得控制。
- 程序只有一个入口点。
- 用符号 *PGM 对系统标识程序。

ILE 具有下列 OPM 程序不具有的特性:

- ILE 程序是从一个或多个复制的模块对象中建立的。
- 一个或多个复制的模块可以包含 PEP。
- 您能控制 ILE 程序对象的作为 PEP 使用的模块的 PEP。

当指定了“建立程序”(CRTPGM) 命令时，ENTMOD 参数允许您选择包含 PEP 的哪一个模块是程序的入口点。

忽略与未被选择作为程序入口点的模块相关的 PEP。该模块的所有其他过程和数据项都当作已指定的使用。只忽略 PEP。

当对 ILE 程序执行动态程序调用时，对在程序建立时选择的模块的 PEP 给出控制。PEP 调用相关的 UEP。

当建立了 ILE 程序对象时，ILE 调试程序仅调试那些与包含调试数据的复制的模块相关的过程。调试数据不影响正运行的 ILE 程序的性能。

图9显示 ILE 程序对象的概念性视图。当调用了 PGMEXAMP 程序时，对在复制的模块对象 M3 中定义的该程序的 PEP 给出控制。复制模块 M2 也定义了 PEP，但该程序忽略并且从不使用它。

在此程序示例中，只有两个模块 M1 和 M3 具有新的 ILE 调试程序必需的数据。使用新的 ILE 调试程序不能调试模块 M2 和 M4 中的过程。

分别从服务程序 PRINTS 和 MATHFUNC 将调入的过程 `print` 和 `SIN` 解析为调出的过程。

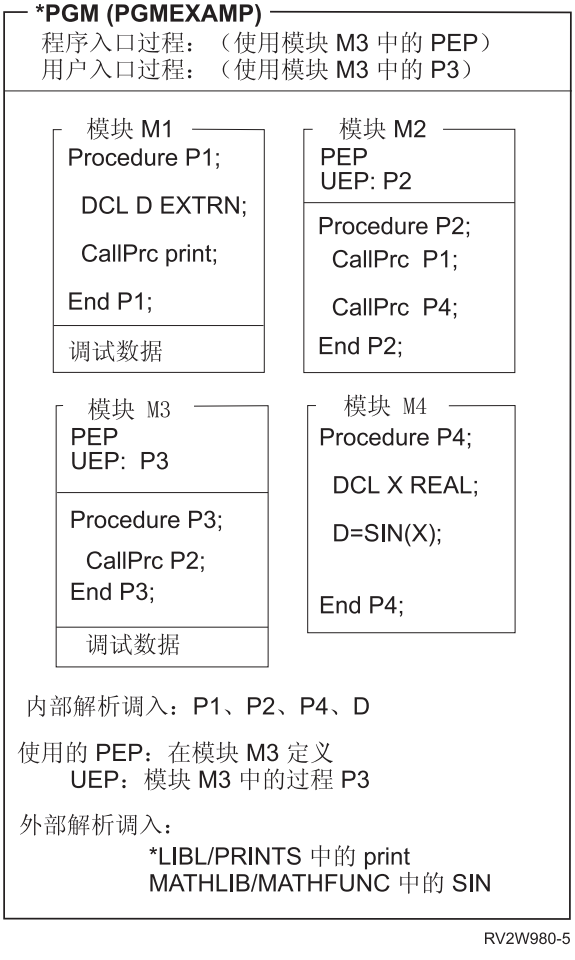

图 9. ILE 程序的概念性视图

ILE *PGM 对象的特性:

- 复制任何 ILE 语言的一个或多个模块以建立 *PGM 对象。

- 建立该程序的人能控制其 PEP 成为该程序的唯一的 PEP 的模块。
- 在动态程序调用上，被选择作为该程序的 PEP 的模块的 PEP 获得控制以运行。
- 与所选择的 PEP 相关的 UEP 是该程序的用户入口点。
- 不能从该程序中调出过程和数据项名。
- 可从模块和服务程序中但不是从程序对象中调入过程或数据项名。关于服务程序的有关详情，参见『服务程序』。
- 模块可有调试数据。
- 程序是可运行对象。

服务程序

服务程序是一个可运行的过程和可用数据项的集合，其他 ILE 程序或服务程序可简单并直接地存取这些过程和数据项。在许多方面，服务程序类似于子例程库或过程库。

服务程序提供其他 ILE 对象可能需要的公共服务（从服务程序名）。由 OS/400 提供的一组服务程序的示例是一种语言的运行期过程。这些运行期过程通常包括这样一些作为数学过程和公共输入／输出过程的项。

服务程序的**公共接口**由可由其他 ILE 对象存取的调出的过程名和数据项名组成。只有那些从组成服务程序的模块对象中调出的项才适合从服务程序中调出。

程序员可以指定哪些过程或数据项可被其他 ILE 对象所知。因此，服务程序可有不可用于其他任何 ILE 对象的隐藏的或私用的过程和数据。

在未重新建立其他使用更新的服务程序的 ILE 程序或服务程序的情况下，更新服务程序是可能的。程序员更改服务程序以控制更改是否与现存支持兼容。

ILE 提供的控制兼容更改的方法是通过使用**联编器语言**。联编器语言允许您定义可调出的过程名和数据项名列表。从过程和数据项的名称以及从在联编器语言中指定它们的顺序中生成**特征符**。要对服务程序进行兼容更改，应将新的过程或数据项名添加至调出列表的末尾。关于特征符、联编器语言和保护您的客户对服务程序的投资的有关详情，参见第56页的『联编器语言』。

第16页的图10显示服务程序的概念性视图。注意，组成该服务程序的模块与在第14页的图9中组成 ILE 程序对象 PGMEXAMP 的是同一组模块。服务程序 SPGMEXAMP 的先前特征符 Sigyy，包含过程 P3 和 P4 的名称。当对服务程序执行了向上兼容更改时，当前特征符 Sigxx 不只包含过程 P3 和 P4 的名称，还包含数据项 D 的名称。使用过程 P3 和 P4 的其他 ILE 程序或服务程序不需要重新建立。

尽管服务程序中的模块可能有 PEP，但这些 PEP 被忽略。服务程序本身没有 PEP。因此，不可以象程序对象那样动态地调用服务程序。

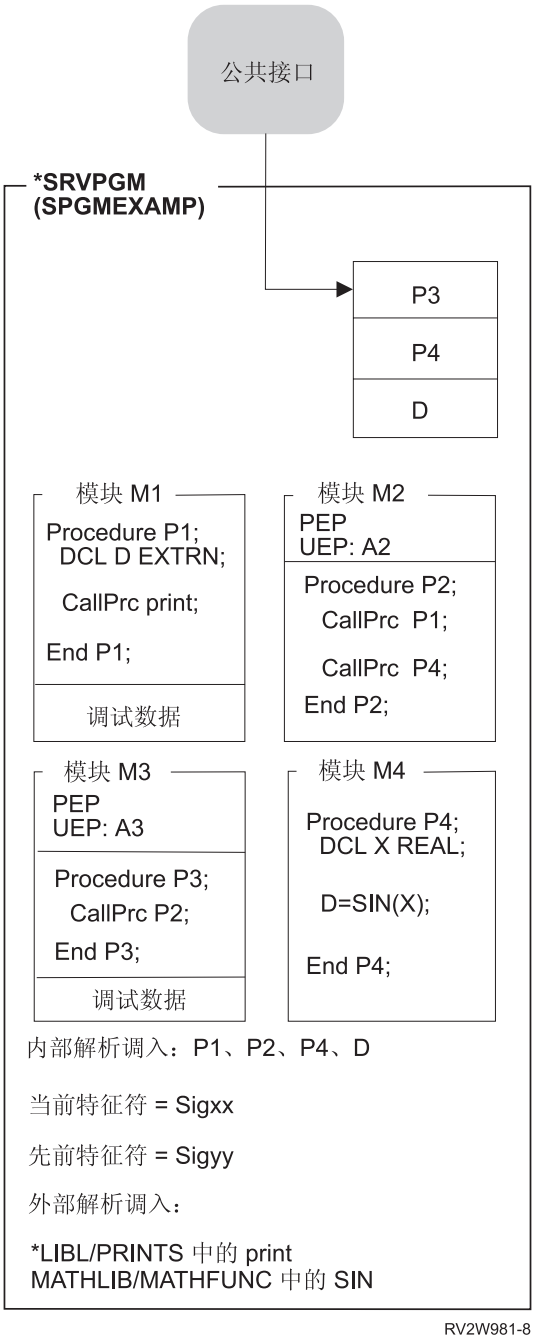

图 10. ILE 服务程序的概念性视图

ILE *SRVPGM 对象的特性:

- 复制来自任何 ILE 语言的一个或多个模块以建立 *SRVPGM 对象。
- 没有与该服务程序相关的 PEP。因为没有 PEP，所以对服务程序的动态程序调用无效。忽略模块的 PEP。
- 其他 ILE 程序或服务程序可使用由公共接口标识的此服务程序的调出。
- 特征符是从由该服务程序调出的过程和数据项名中生成的。
- 只要仍支持先前特征符，就可以置换服务程序而不影响使用它们的 ILE 程序或服务程序。

- 模块可有调试数据。
- 服务程序是可运行的过程和数据项的集合。
- 只可将弱数据调出至激活组。它不能是组成从服务程序调出的公共接口的一部分。关于弱数据的有关详情，参见在第55页的『调入和调出概念』中的“调出”。

联编目录

联编目录包含建立 ILE 程序或服务程序时可能需要的模块和服务程序的名称。仅当在联编目录中所列示的模块和服务程序提供了满足任何当前未解析的调入请求时，才使用它们。联编目录是系统对象，对系统用符号 *BNDDIR 标识它。

联编目录是可选的。使用联编目录的原因的为了便利以及程序的大小。

- 当建立您自己的 ILE 程序或服务程序时，它们提供了将可能需要的模块或服务程序打包在一起的便利的方法。例如，一个联编目录可能包含提供数学函数的所有模块和服务程序。当您想使用其中的某些函数时，您只需指定一个联编目录而不用要使用的指定每个模块或服务程序。

  注： 联编目录所包含的模块或服务程序越多，联编程序所花费的时间就越长。因此，应该只在联编目录中包括那些必需的模块或服务程序。
- 因为不指定那些不使用的模块或服务程序，所以联编目录可以减少程序的大小。

在联编目录中，对入口的限制极少。甚至当一个对象已不存在时，还可以将该模块或服务程序的名称添加至联编目录中。

有关配合联编目录使用的 CL 命令的列表，参见第143页的『附录C. 对 ILE 对象使用的 CL 命令』。

图11显示联编目录的概念性视图。

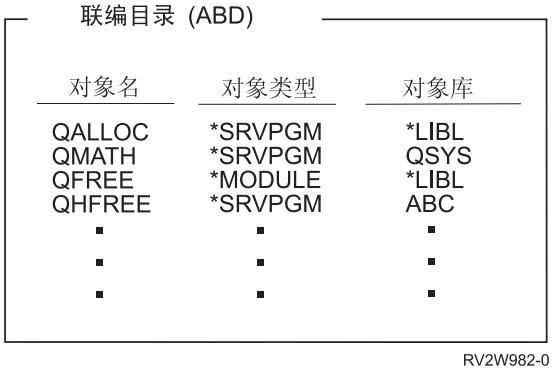

图 11. 联编目录的概念性视图

*BNDDIR 对象的特性:

- 用来分组建立 ILE 程序或服务程序可能需要的服务程序和模块的名称的便利方法。
- 由于联编目录项只是名称，因此所列的对象不必仍存在于系统上。
- 唯一有效的库名称是 *LIBL 或一个特定的库。
- 列表中的对象是可选的。仅当存在未解析的调入并且所命名的对象提供了满足该未解析的调入请求时，才使用所命名的对象。

联编器功能

联编器的功能与连接编辑器提供的功能相似，但也有某些不同。**联编器**处理对指定的模块中的过程名和数据项名的调入请求。然后联编器尝试在指定的模块、服务程序和联编目录中查找匹配调出。

在建立 ILE 程序或服务程序过程中，联编器执行下列类型的联编:

- 通过复制联编

  要建立 ILE 程序或服务程序，复制下列项:

  在模块参数上指定的模块

  从对未解析的调入提供调出的联编目录中选择的任何模块

  当建立 ILE 程序或服务程序时，建立了在复制的模块内使用的所需的过程和数据项的物理地址。

  例如，在第16页的图10中，模块 M3 中的过程 P3 调用模块 M2 中的过程 P2。使过程 M3 知道模块 M2 中的过程 P2 的物理地址，以便于可以直接存取该地址。

- 通过引用联编

  将至对未解析的调入提供调出的服务程序的符号链路保存在所建立的程序或服务程序中。符号连路指向提供调出的服务程序。当激活与服务程序联编的程序对象时，将该链路转换为物理地址。

  第16页的图10显示至服务程序 *MATHLIB/MATHFUNC 中的 SIN 的一个符号链路的示例。当激活与服务程序 SPGMEXAMP 联编的程序对象时，将至符号 SIN 的符号链路转换为物理地址。

在运行期，随着所建立的至过程和数据项的物理链路的使用，下列各项之间的区别较小:

- 存取本地过程或数据项
- 存取与同一个程序联编的不同模块或服务程序中的过程或数据项

第19页的图12和第19页的图13显示如何建立 ILE 程序 PGMEXAMP 和服务程序 SPGMEXAMP 的概念性视图。联编器使用模块 M1、M2、M3 和 M4 以及服务程序 PRINTS 和 MATHFUNC 来建立 ILE 程序 PGMEXAMP 和服务程序 SPGMEXAMP。

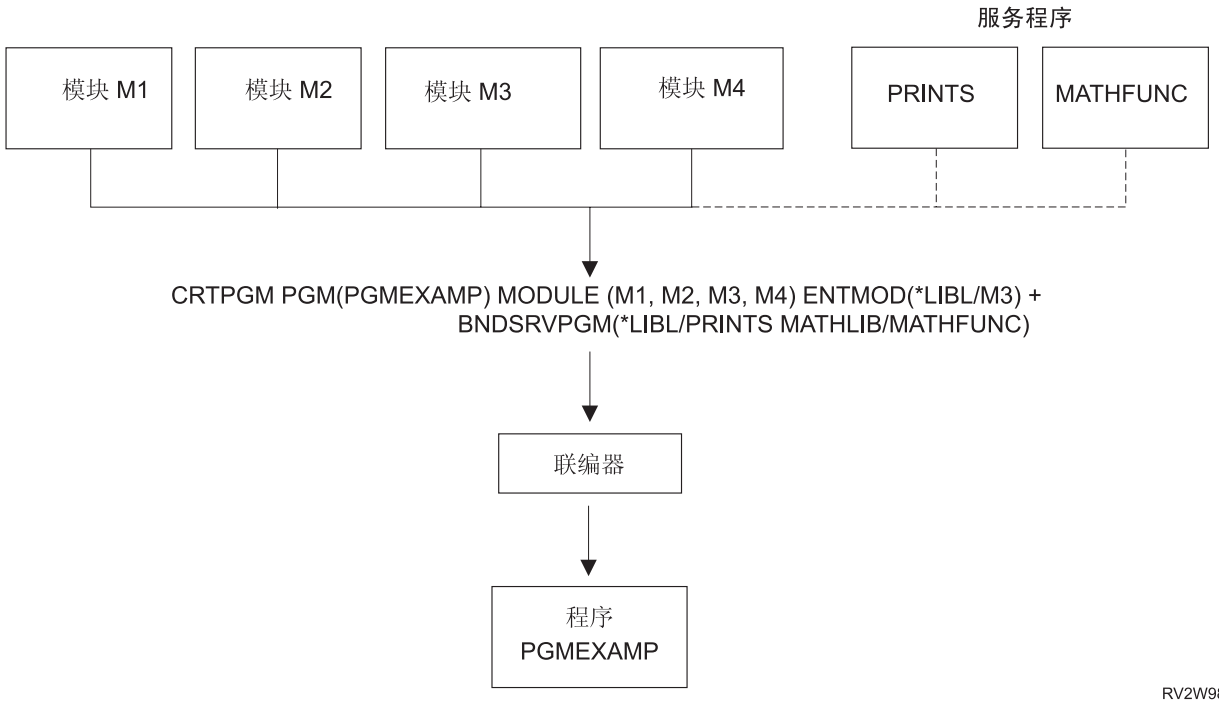

图 12. ILE 程序的建立. 中断行指示该服务程序是通过引用联编的而不是通过复制联编的。

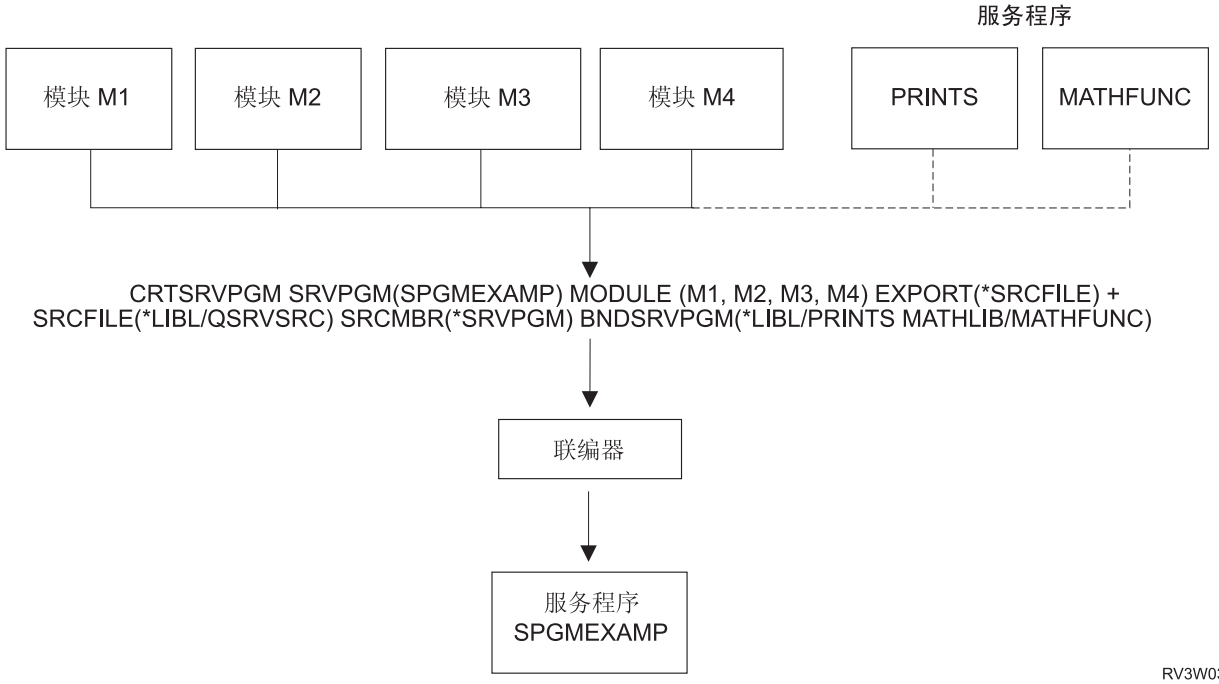

图 13. 服务程序的建立. 中断行指示该服务程序是通过引用联编的而不是通过复制联编的。

有关建立 ILE 程序或服务程序的其他信息，参见第45页的『第4章 程序建立概念』。

对程序和过程的调用

在 ILE 中既可以调用程序也可以调用过程。 ILE 要求调用者标识调用语句的目标是程序还是过程。 ILE 语言通过对于程序和对于过程使用单独的调用语句来传达此需求。因此，当编写您的 ILE 程序时，必须知道要调用的是程序还是过程。

每种 ILE 语言都有唯一的语法允许您在动态程序调用和静态过程调用之间加以区别。每种 ILE 语言中的标准调用语句缺省为动态程序调用或静态过程调用。对于 RPG 和 COBOL，缺省是动态程序调用，而对于 C，缺省则是静态过程调用。因此，标准语言调用在 OPM 或 ILE 中执行相同类型的功能。此约定使从 OPM 语言迁移至 ILE 语言相对容易。

联编器可以处理最多 256 个字符长度的过程名。要确定您的过程名有多长，参见 ILE HLL 程序员指南。

动态程序调用

动态程序调用转换控制至 ILE 程序对象或 OPM 程序对象。动态程序调用包括:

- OPM 程序可以调用另一个 OPM 程序或 ILE 程序，但不能调用服务程序。
- ILE 程序可以调用 OPM 程序或另一个 ILE 程序，但不能调用服务程序。
- 服务程序可以调用 OPM 程序或 ILE 程序，但不能调用另一个服务程序。

静态过程调用

静态过程调用转换控制至 ILE 过程。只可在 ILE 语言中编码静态过程调用。可使用静态过程调用来调用下列任何项:

- 在同一个模块内的过程
- 在同一个 ILE 程序或服务程序内的一个单独模块中的过程
- 在一个单独的 ILE 服务程序中的过程

第21页的图14显示静态过程调用的示例。该图形显示:

- ILE 程序中的过程可以调用在同一个程序或服务程序中已调出的过程。程序 A 中的过程 P1 调用在另一个复制的模块中的过程 P2。程序 C 中的过程 P3 调用服务程序 D 中的过程 P4。
- 服务程序中的过程可以调用在同一个服务程序或另一个服务程序中已调出的过程。程序 B 中的过程 P6 调用在另一个复制的模块中的过程 P7。程序 E 中的过程 P5 调用服务程序 F 中的过程 P4。

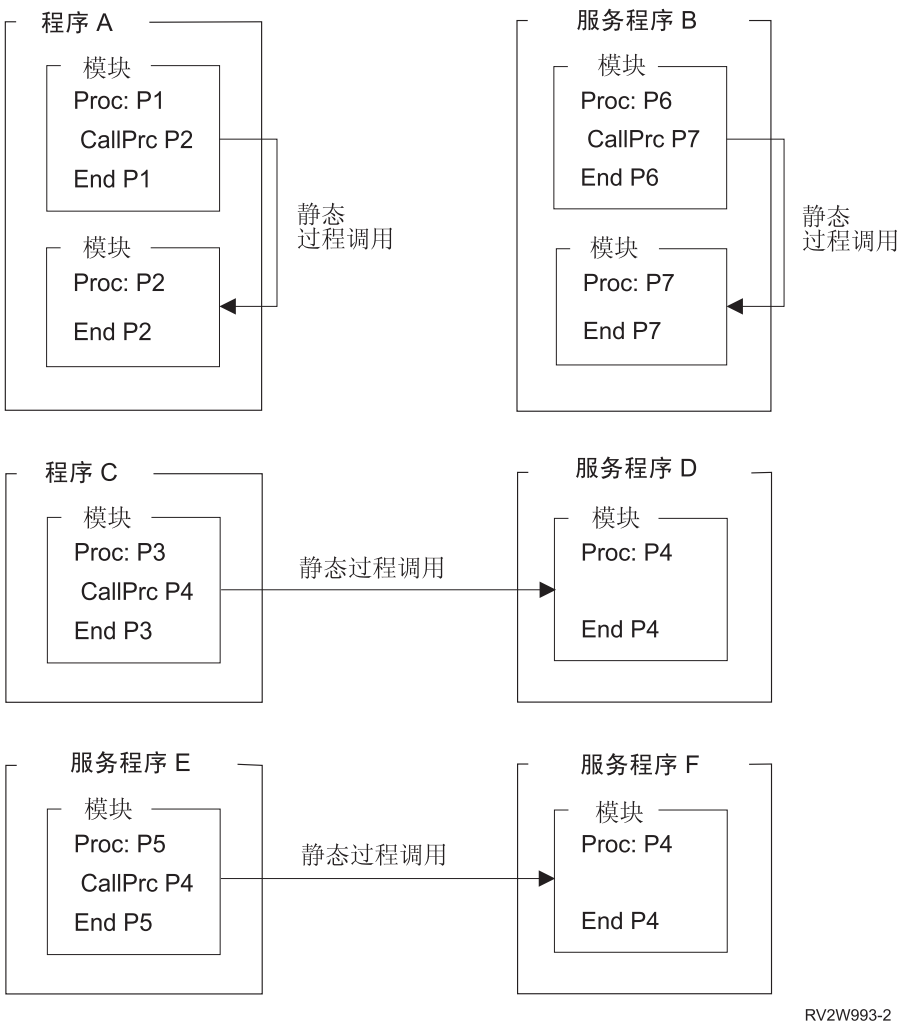

图 14. 静态过程调用

激活

在成功建立 ILE 程序后，您将想要运行您的代码。将使程序或服务程序准备好运行的过程称为**激活**。您不必发出命令来激活程序。激活是当调用程序时由系统来完成的。因为这些服务程序未被调用，所以在调用一个直接或非直接要求其服务的程序期间激活它们。

激活执行下列功能:

- 唯一分配程序或服务程序所需的静态数据
- 将至提供调出的服务程序的符号链路更改为至物理地址的链路

不管有多少个作业正在运行程序或服务程序，只能有那些对象的指令的一个副本驻留在存储器中。然而，每个程序激活都有其自己的静态存储器。因此即使有多个作业并行使用一个程序对象，对每个激活来说静态变量还是单独的。程序还可以被激活在多个激活组中，甚至在同一个作业内，但是对特定的激活组来说激活是局部的。

若下列两者之一为真:

- 激活找不到所需的服务程序

- 服务程序不再支持特征符所代表的过程或数据项

则发生错误并且不能运行应用程序。

关于程序激活的有关详情，参考第26页的『程序激活建立』。

当激活分配程序所使用的静态变量所需的存储器时，从激活组中分配空间。在建立程序或服务程序时，可以指定将要在运行期时使用的激活组。

有关激活组的详情，参考第27页的『激活组』。

错误处理

图15显示 OPM 和 ILE 程序的完整的错误处理结构。在本手册中贯穿使用此图来描述高级错误处理性能。本主题提供了标准语言错误处理性能的简要概述。有关错误处理的其他信息，参考第35页的『错误处理』。

该图显示称为异常信息体系结构的基本层。异常信息可能是当 OPM 或 ILE 程序遇到错误时由系统生成的。异常信息还可用于传达不认为是程序错误的状态信息。例如，通过发送一条状态异常信息来传达一个未找到数据库记录的情况。

每个高级语言都定义了特定于语言的错误处理能力。尽管这些能力因语言而异，但总的来讲，每个 HLL 用户还是能够说明要处理特定错误情况的意图。此意图的说明包括对错误处理例程的标识。当发生异常时，系统定位错误处理例程，并且将控制传递至用户编写的指令。可以执行各种操作，包括结束程序或从错误中恢复和继续。

图15显示 ILE 使用的异常信息体系结构与 OPM 程序所使用的相同。由系统生成的异常信息在 ILE 程序内启动特定于语言的错误处理，就如同它们在 OPM 程序内所做的一样。图中的最低层包括发送和接收异常信息的能力。这可由信息处理器 API 或命令来完成。可在 ILE 和 OPM 之间发送和接收异常信息。

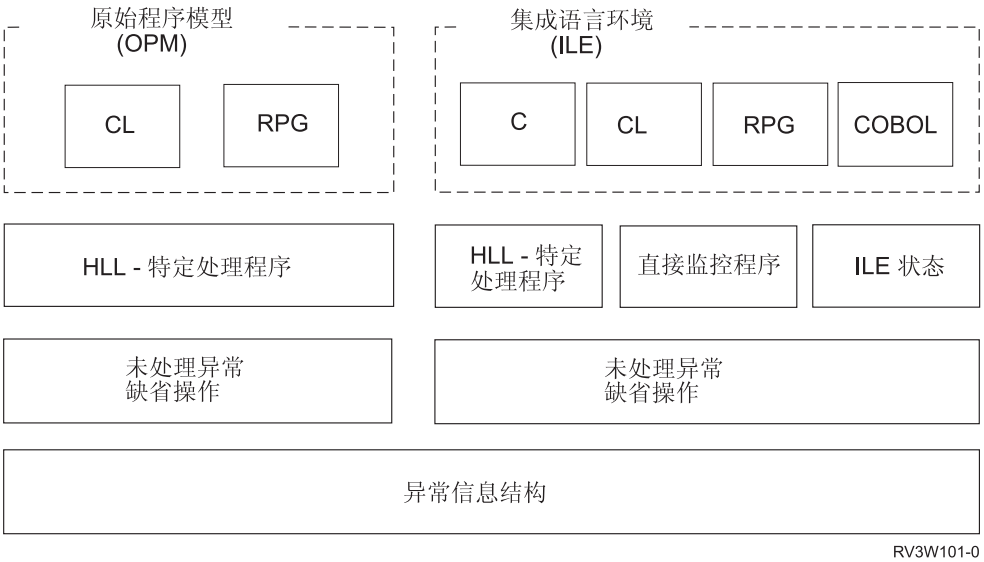

图 15. OPM 和 ILE 的错误处理

特定于语言的错误处理对于 ILE 程序和 OPM 程序所起的作用是类似的，但是还有基本的区别:

- 当系统发送异常信息至 ILE 程序时，使用过程和模块名来限定该异常信息。当发送一条异常信息时，可指定这些相同的限定。当在作业记录中出现一条对 ILE 程序的异常信息时，系统通常提供程序名、模块名和过程名。
- ILE 程序的扩展优化可导致多个与同一个生成的指令相关的 HLL 语句成员。作为优化的结果，出现在作业记录中的异常信息可能包含多个 HLL 语句成员。

其他错误处理能力在第35页的『错误处理』中有述。

优化转换器

在 AS/400 上，**优化**意味着使对象的运行期性能最大化。所有 ILE 语言者具有对由 ILE 优化转换器提供的优化技术的存取权。通常，优化需求越高，建立对象所花的时间就越长。在运行期，高优化的程序或服务程序的运行要比相应的由低优化级别建立的程序或服务程序快。

尽管可以为模块、程序对象和服务程序指定优化，但优化技术只应用于单个的模块。优化级别有:

10 或 *NONE

20 或 *BASIC

30 或 *FULL

40（比级别 30 更多的优化）

由于性能的原因，当使用一个模块用于生产时，您可能想要一个高级优化。在您期望使用的优化级别上测试您的代码。验证是否每件事都如您所期望的那样，然后使您的用户获得代码。

因为在级别 30 (*FULL) 或级别 40 上的优化可能会对您的程序指令影响较大，所以您可能需要了解特定的调试限制和不同编址的异常检测。有关调试考虑事项，参考第99页的『第9章 调试考虑事项』。有关编址错误考虑事项，参考第141页的『附录B. 优化程序中的异常』。

调试程序

ILE 提供允许源级调试的调试程序。可随列表文件一起使用调试程序，并且允许设置中断点、显示变量和进入或跨过一条指令。您可以执行这些操作而无需一定要从命令行输入命令。当使用调试程序时命令行仍可用。

源级调试程序使用系统提供的 API，允许调试程序或服务程序。这些 API 对每个人来说都是可用的，并且允许您编写您自己的调试程序。

对 OPM 程序的调试程序继续存在于 AS/400 系统上，但只可用来调试 OPM 程序。

当调试已优化的模化时，可能导致混乱。当使用 ILE 调试程序来察看或更改由正运行的程序或过程使用的变量时，会发生下列情况。调试程序在存储位置中检索或更新此变量的数据。在级别 20 (*BASIC)、30 (*FULL) 或 40 的优化时，数据变量的当前值可能在调试程序不能存取的硬件寄存器中。（数据变量是否在硬件寄存器中取决于若干因素。这些因素包括如何使用变量、变量的大小和在代码中的何处停止检查或更改数据

变量。）因此，所显示的变量的值可能不是当前值。由于此原因，在开发期间您应该使用优化级别 10 (*NONE)。然后，要获得最好的优化，您可以在生产期间将优化级别更改为 30 (*FULL) 或 40。

有关 ILE 调试程序的详情，参见第99页的『第9章 调试考虑事项』。

第3章 ILE 高级概念

本章描述 ILE 模型的高级概念。在阅读本章之前，您应熟悉第11页的『第2章 ILE 基本概念』中描述的概念。

程序激活

激活是用来准备运行程序的过程。在可运行 ILE 程序和 ILE 服务程序之前，系统必须激活它们。

程序激活包括两个主要步骤:

1. 分配并初始化程序的静态存储器。
2. 完成程序与服务程序的联编。

此题目把注意力放在步骤 1 上。第32页的『服务程序激活』中说明了步骤 2。

第26页的图16显示了两个存储在永久磁盘存储器中的 ILE 程序对象。就象对所有 OS/400 对象那样，在不同 OS/400 作业中运行的多个并行用户可共享这些程序对象。只存在程序代码的一个副本。但是，当调用这些 ILE 程序之一时，必须对每个程序激活分配并初始化程序中说明的某些变量。

如图16所示，每个程序激活都支持这些变量的最少一个唯一副本。同名变量的多个副本可存在于一个程序激活中。若 HLL 允许说明限于个别过程使用的静态变量，则发生此情况。

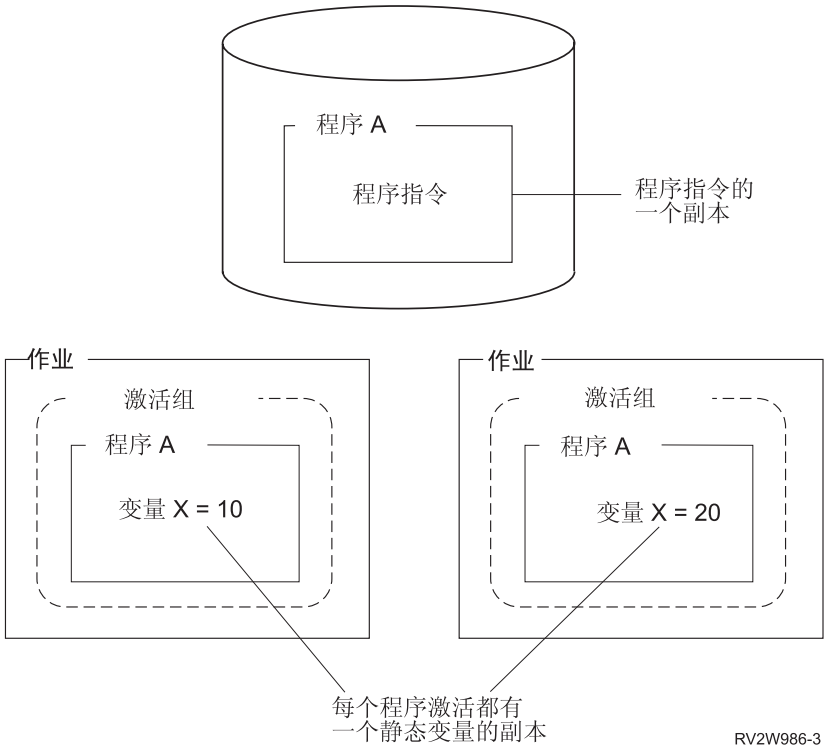

图 16. 每个程序激活有静态变量的一个副本

程序激活建立

ILE 通过跟踪激活组中的程序激活来管理程序激活的过程。参考第27页的『激活组』以获取激活组的定义。在激活组中，只有特定程序对象的一个激活。应用此规则时，将驻留在不同的 AS/400 库中的同名程序认为是不同的程序对象。

当在 HLL 程序中使用动态程序调用语句时，ILE 使用建立该程序时指定的激活组。此属性是通过使用“建立程序” (CRTPGM) 命令或“建立服务程序” (CRTSRVPGM) 命令上的激活组 (ACTGRP) 参数指定的。若此参数指示的激活组中已存在程序激活，则使用它。若从未在此激活组中激活程序，则首先激活该程序，然后运行它。若存在命名的激活组，则可用 UPDPGM 和 UPDSRVPGM 命令上的 ACTGRP 参数更改该名称

程序一旦激活，在删除激活组之前，它就保持激活状态。此规则的结果是，活动程序有可能不在激活组中的调用堆栈上。第27页的图17显示了激活组中有三个活动程序，但三个程序中只有两个在调用堆栈上有过程的示例。在此示例中，程序 A 调用程序 B，导致激活程序 B。程序 B 然后返回至程序 A。程序 A 然后调用程序 C。结果调用堆栈包含程序 A 和 C 的过程，但不包含程序 B 的过程。有关调用堆栈的讨论，参见第79页的『调用堆栈』。

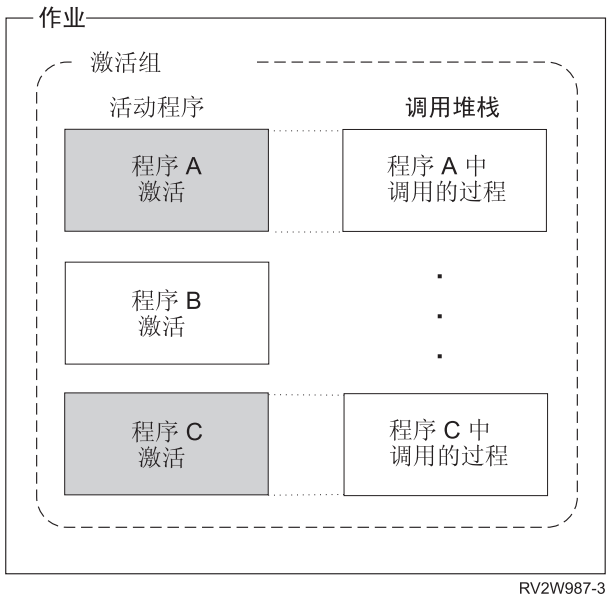

图 17. 程序可能是活动的，但不在调用堆栈上

激活组

所有 ILE 程序和服务程序都在作业的称为**激活组**的子结构中激活。这个子结构包含运行程序所必需的资源。这些资源分成下列一般类别:

静态和自动程序变量

动态存储器

临时数据管理资源

特定类型的异常处理程序和结束过程

对每个激活组的静态和自动程序变量以及动态存储器分配了不同的地址空间。这提供了对意外存取的某种程度的程序隔离和保护。

临时数据管理资源包括下列各项:

打开的文件（开放数据通路或 ODP）

确认定义

本地 SQL 游标

远程 SQL 游标

分层文件系统 (HFS)

用户界面管理器

查询管理实例

开放式通信链路

“公共程序设计接口”(CPI) 通信

激活组之间这些资源的分隔支持一个基本概念。即，这是这样一个概念: 在一个激活组中激活的所有程序都是作为一个协作应用程序开发的。

软件供应商可选择不同的激活组来将他们的程序与同一作业中运行的其他供应商应用程序隔离开。第28页的图18中显示了这种供应商隔离。在此图中，通过集成来自四个

不同供应商的软件包提供了彻底的客户解决方案。激活组通过隔离与每个供应商软件包相关联的资源增加了集成的容易程度。

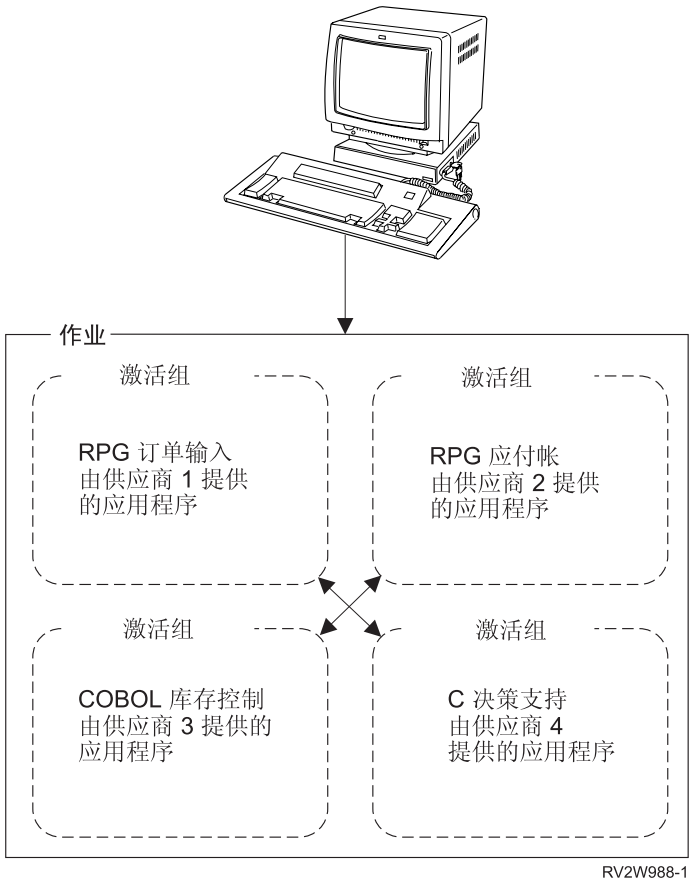

图 18. 隔离每个供应商的应用程序的激活组

对激活组指定上述资源有一个显著的后果。这个后果是删除激活组时，上述所有资源都返回至系统。删除激活组时未关闭的临时数据管理资源由系统关闭。静态和自动程序变量的存储器以及未释放的动态存储器返回至系统。

激活组建立

建立程序或服务程序时，可通过指定激活组属性来控制 ILE 激活组的运行期建立。通过使用 CRTPGM 命令或 CRTSRVPGM 命令上的 ACTGRP 参数指定属性。没有“建立激活组”命令。

所有 ILE 程序都具有下列其中一种激活组属性:

- 用户命名的激活组

  通过 ACTGRP(name) 参数指定。此属性允许您以一个应用程序的形式管理 ILE 程序和 ILE 服务程序的集合。激活组是在首次需要它时建立的。然后，指定同一激活组名的所有程序和服务程序都使用它。
- 系统命名的激活组

通过 CRTPGM 命令上的 ACTGRP(*NEW) 参数指定。此属性允许每当调用程序时建立新的激活组。ILE 为此激活组选择名称。 ILE 指定的名称在作业中是唯一的。对系统命名的激活组指定的名称与您为用户命名的激活组选择的任何名称都不匹配。ILE 服务程序不支持此属性。

- 属性: 使用调用程序的激活组

  通过 ACTGRP(*CALLER) 参数指定。此属性允许建立将在调用程序的激活组中激活的 ILE 程序或 ILE 服务程序。借助此属性，当激活程序或服务程序时，决不会建立新的激活组。

作业中的所有激活组都具有名称。一旦作业中存在激活组，ILE 就使用它来激活指定该名称的程序和服务程序。此设计的结果是，重复的激活组名不能存在于一个作业中。但是，可使用 UPDPGM 和 UPDSRVPGM 上的 ACTGRP 参数来更改激活组的名称。

缺省激活组

OS/400 作业启动时，系统建立两个将由 OPM 程序使用的激活组。为 OS/400 系统代码保留一个激活组。将另一激活组用于所有其他 OPM 程序。不能删除 OPM 缺省激活组。作业结束时，系统删除它们。

若满足以下两个条件，则可在 OPM 缺省激活组中激活 ILE 程序和 ILE 服务程序:

- ILE 程序或 ILE 服务程序是用激活组 *CALLER 选项建立的。
- 对 ILE 程序或 ILE 服务程序的调用发自 OPM 缺省激活组。

因为不能删除缺省激活组，所以 ILE HLL 结束动词不能提供彻底的结束处理。在作业结束之前，系统不能关闭打开的文件。在作业结束之前，ILE 程序使用的静态和堆存储器不能返回至系统。

第30页的图19显示了典型的带有 ILE 激活组和 OPM 缺省激活组的 OS/400 作业。因为使用了特殊值 *DFTACTGRP 来表示两个 OPM 缺省激活组，所以将两个组组合在一起。每个激活组中的方框代表程序激活。

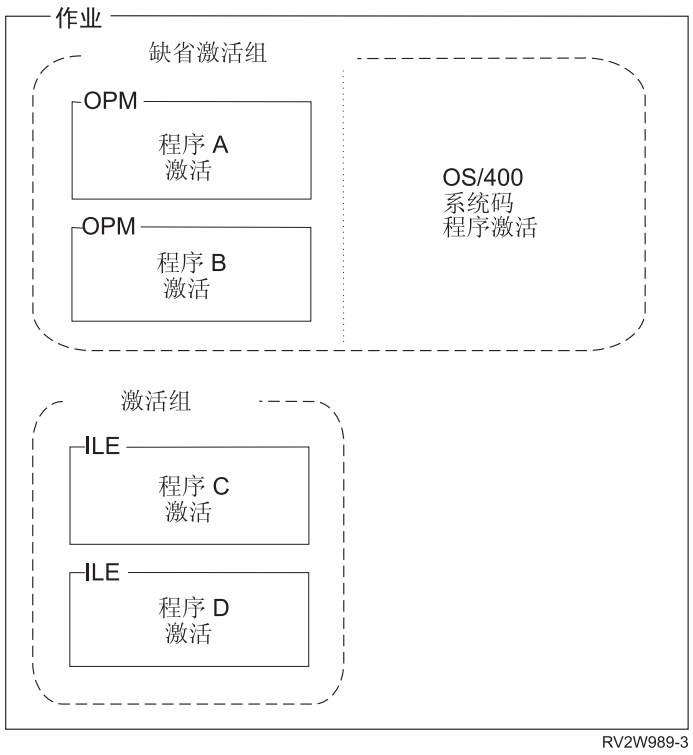

图 19. 缺省激活组和 ILE 激活组

ILE 激活组删除

激活组要求在作业中建立资源。若应用程序可再使用激活组，则可节省处理时间。 ILE 提供了数个选项来允许您在不结束或删除激活组的情况下从激活组返回。删除激活组与否取决于激活组的类型和应用程序的结束方法。

应用程序可通过下列方法离开激活组并返回至在另一激活组中运行的调用堆栈项（参见第79页的『调用堆栈』）：

- HLL 结束动词

  例如，COBOL 中的 STOP RUN 或 C 中的 exit()。
- 未处理的异常

  系统可将未处理的异常移至另一激活组中的调用堆栈项。
- 特定于语言的 HLL 返回语句

  例如，C 中的 return 语句、COBOL 中的 EXIT PROGRAM 语句或 RPG 中的 RETURN 语句。
- 跳过操作

  例如，发送异常信息或转移至不在您的激活组中的调用堆栈项。

可通过使用 HLL 结束动词来从应用程序中删除激活组。未处理的异常也会导致删除激活组。倘若最近的控制边界是激活组中最旧的调用堆栈项（有时称为硬控制边界），则这些操作将总是删除激活组。若最近的控制边界不是最旧的调用堆栈项（有时称为软控制边界），则将控制传送至该控制边界之前的调用堆栈项。但是，不删除激活组。

控制边界是表示应用程序的边界的调用堆栈项。每当在激活组之间进行调用时，ILE 就定义控制边界。参考第33页的『控制边界』以获取控制边界的定义。

可将用户命名的激活组留在作业中，以供稍后使用。对于此类型的激活组，任何正常返回或越过硬控制边界的跳过操作都不删除该激活组。在系统命名的激活组中使用这些操作将删除该激活组。因为不能通过指定系统生成的名称来再使用系统命名的激活组，所以总是删除它们。关于从激活组的最旧调用堆栈项正常返回的、从属于语言的规则，参考 ILE HLL 程序员指南。

图20显示了如何离开激活组的示例。在图中，过程 P1 是最旧的调用堆栈项。对于系统命名的激活组（用 ACTGRP(*NEW) 选项建立），从 P1 正常返回将删除该激活组。对于用户命名的激活组（用 ACTGRP(name) 选项建立），从 P1 正常返回不会删除该激活组。

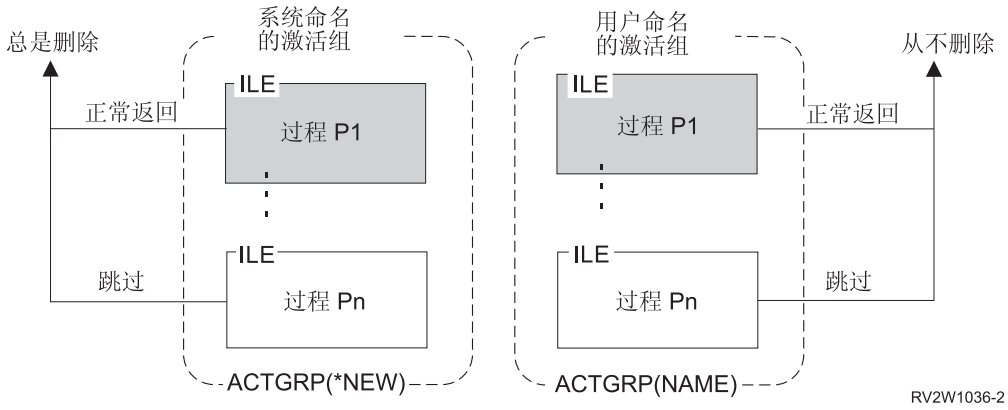

图 20. 离开用户命名的和系统命名的激活组

若将用户命名的激活组留在作业中，则可通过使用“回收激活组”(RCLACTGRP) 命令删除它。此命令允许在应用程序返回后删除命名的激活组。此命令仅可删除不在使用中的激活组。

第32页的图21显示了带有一个不在使用中的激活组和一个当前在使用中的激活组的 OS/400 作业。若在激活组中激活了 ILE 过程的调用堆栈项，则认为该激活组在使用中。在程序 A 或程序 B 中使用 RCLACTGRP 命令删除程序 C 和程序 D 的激活组。

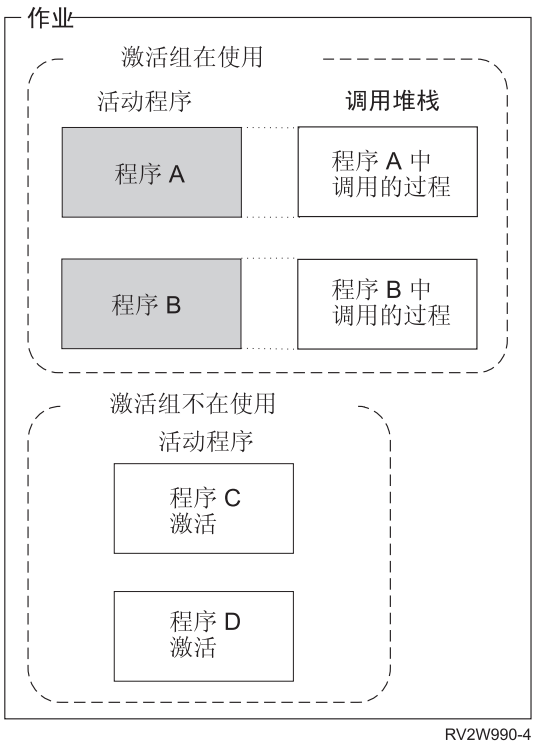

图 21. 使用中的激活组在调用堆栈上有项目

ILE 删除激活组时，发生特定结束操作处理。此处理包括调用用户注册的出口过程、数据管理清理和语言清理（如关闭文件）。参考第41页的『数据管理范围限定规则』以获取有关删除激活组时发生的数据管理处理的细节。

服务程序激活

此题目讨论系统用来激活服务程序的独特步骤。第25页的『程序激活』中描述了用于程序和服务程序的公共步骤。下列激活活动对于服务程序来说是唯一的:

- 服务程序激活是间接地作为对 ILE 程序的动态程序调用的一部分启动的。
- 服务程序激活包括通过将符号链路映射成物理链路来完成程序间的联编连接。
- 服务程序激活包括特征符检查处理。

当在激活组中首次激活 ILE 程序时，检查它与任何 ILE 服务程序的联编。若服务程序已与正在激活的程序联编，则它们也是作为同一动态调用处理的一部分激活的。重复此过程，直到激活了所有必需的服务程序为止。

第33页的图22显示了与 ILE 服务程序 B、C 和 D 联编的 ILE 程序 A。 ILE 服务程序 B 和 C 还与 ILE 服务程序 E 联编。显示了每个程序和服务程序的激活组属性。

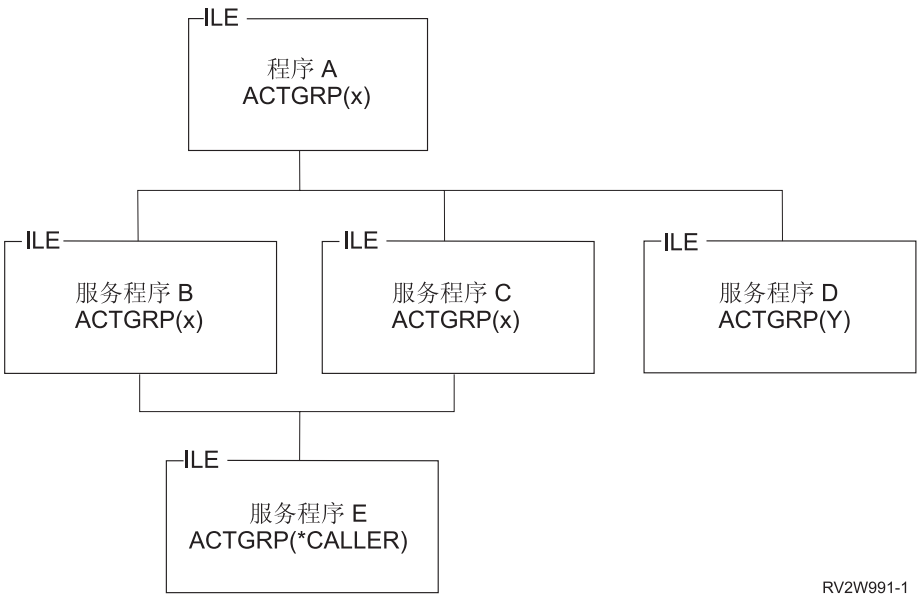

图 22. 服务程序激活

激活 ILE 程序 A 时，发生下列各项:

- 通过使用显式库名或通过使用当前库列表找出服务程序。此选项是由您在建立程序和服务程序时控制的。
- 就象程序那样，服务程序激活在激活组中只发生一次。在图22中，虽然服务程序 B 和 C 都使用服务程序 E，但只激活它一次。
- 为服务程序 D 建立第二个激活组 (Y)。
- 在所有程序和服务程序之间发生特征符检查。

在观念上，可将此处理视为完成建立程序和服务程序时启动的联编处理。CRTPGM 命令和 CRTSRVPGM 命令保存引用的每个服务程序的名称和库。建立程序时，还在客户机程序或服务程序中保存调出过程和数据项的表的索引。服务程序激活过程通过将这些符号引用更改成可在运行期使用的地址来完成联编步骤。

一旦激活服务程序，就处理对另一服务程序中的模块的静态过程调用和静态数据项引用。处理量与已通过复制将模块与同一程序联编时所需的处理量相同。但是，与服务程序相比，通过复制联编的模块需要较少的激活期处理。

激活程序和服务程序要求对 ILE 程序和所有 ILE 服务程序对象具有执行权限。在图22中，使用程序 A 的调用者的当前权限来检查对程序 A 和所有服务程序具有的权限。还使用程序 A 的权限来检查对所有服务程序具有的权限。注: 不使用服务程序 B、C 或 D 的权限来检查对服务程序 E 具有的权限。

控制边界

当发生未处理的功能检查，或使用 HLL 结束动词时，ILE 执行下列操作。 ILE 将控制传送至表示应用程序边界的调用堆栈项的调用者。此调用堆栈项称为**控制边界**。

控制边界有两个定义。第34页的『ILE 激活组的控制边界』和第34页的『OPM 缺省激活组的控制边界』说明了下列定义。

控制边界可以是下列任一项:

- 其紧前面的调用堆栈项在另一非缺省激活组中的任何 ILE 调用堆栈项。
- 其紧前面的调用堆栈项是 OPM 程序的任何 ILE 调用堆栈项。

ILE 激活组的控制边界

此示例显示如何在 ILE 激活组之间定义控制边界。

图23显示了各种调用建立的两个 ILE 激活组和控制边界。过程 P2、P3 和 P6 是潜在的控制边界。例如，当在过程 P7 中运行时，过程 P6 是控制边界。当在过程 P4 或 P5 中运行时，过程 P3 变成控制边界。

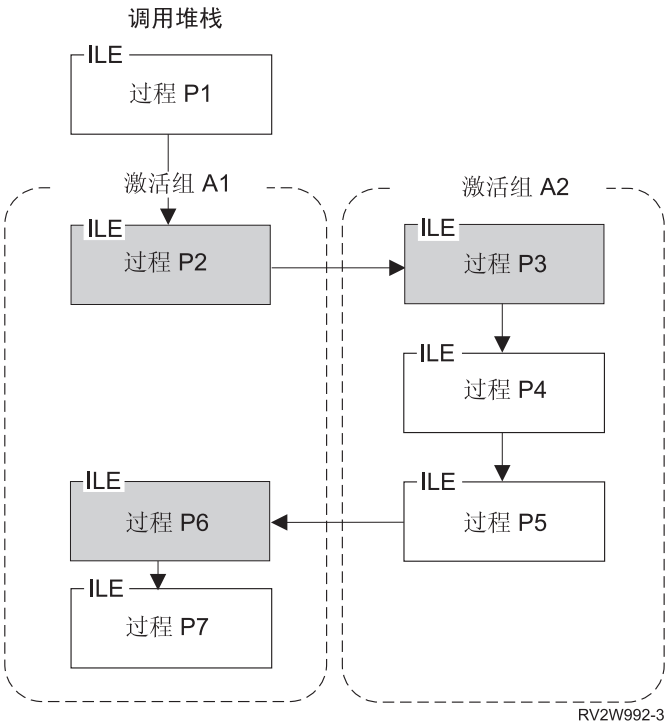

图 23. 控制边界. 带阴影的过程是控制边界。

OPM 缺省激活组的控制边界

此示例显示当在 OPM 缺省激活组中运行 ILE 程序时，控制边界是如何定义的。

第35页的图24显示了三个在 OPM 缺省激活组中运行的 ILE 过程（P1、P2 和 P3）。可通过使用带 ACTGRP(*CALLER) 参数值的 CRTPGM 命令或 CRTSRVPGM 命令建立此示例。因为前面的调用堆栈项是 OPM 程序 A 和 B，所以过程 P1 和 P3 是潜在的控制边界。

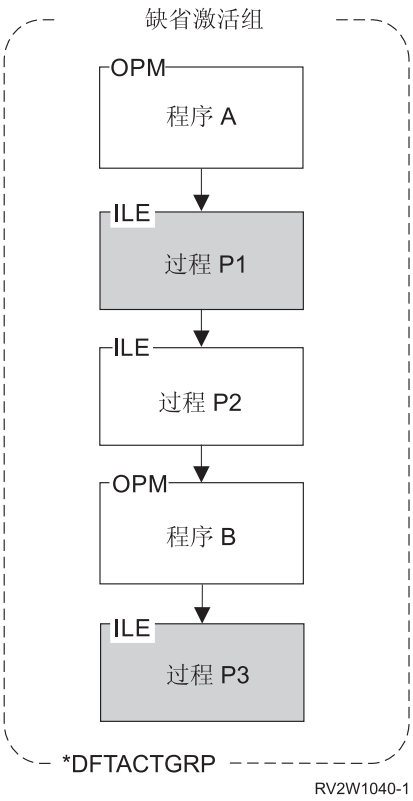

图 24. 缺省激活组中的控制边界. 带阴影的过程是控制边界。

控制边界使用

使用 ILE HLL 结束动词时，ILE 使用调用堆栈上最近的控制边界来确定将控制传送至何处。在 ILE 完成所有结束处理之后，控制边界紧前面的调用堆栈项接收到控制。

当 ILE 过程中发生未处理的功能检查时，使用控制边界。控制边界定义调用堆栈上的一个点，在该处，将未处理的功能检查**促进**为类属 ILE 失败状态。有关附加信息，参考『错误处理』。

当最近的控制边界是 ILE 激活组中最旧的调用堆栈项时，任何 HLL 结束动词或未处理的功能检查都导致删除该激活组。当最近的控制边界不是 ILE 激活组中最旧的调用堆栈项时，控制返回至该控制边界紧前面的调用堆栈项。因为同一激活组中存在较早的调用堆栈项，所以不删除该激活组。

第34页的图23将过程 P2 和过程 P3 显示成其激活组中最旧的调用堆栈项。在过程 P2、P3、P4 或 P5（但不是 P6 或 P7）中使用 HLL 结束动词将导致删除激活组 A2。

错误处理

此题目说明 OPM 和 ILE 程序的高级错误处理能力。要了解这些能力是如何融入异常信息体系结构中的，参考第36页的图25。可在第91页的『第8章 异常和状态管理』中找到特定参考信息和附加概念。第36页的图25显示了错误处理的概述。此主题从此图的底层开始，并向顶层继续。顶层代表可在 OPM 或 ILE 程序中用来处理错误的功能。

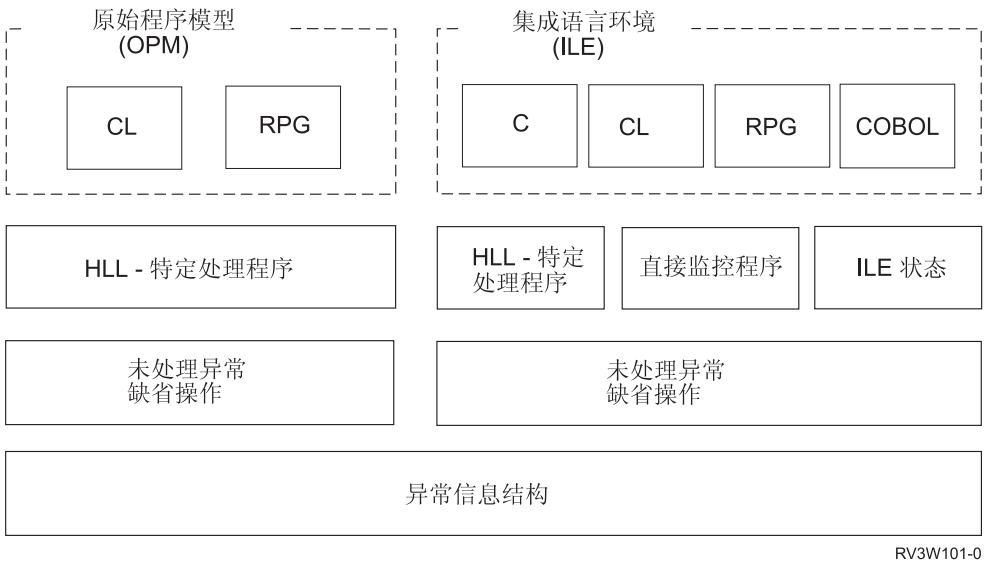

图 25. ILE 和 OPM 错误处理

作业信息队列

对于每个 OS/400 作业中的每个调用堆栈项，都存在一信息队列。此信息队列降低了在运行于调用堆栈上的程序和过程之间发送和接收资料式信息和异常信息的难度。此信息队列称为**调用信息队列**。

调用信息队列由调用堆栈上的 OPM 程序或 ILE 过程的名称标识。可使用该过程名或程序名来指定要发送的信息的目标调用堆栈项。因为 ILE 过程名不是唯一的，所以可可选地指定 ILE 模块名和 ILE 程序或服务程序名。当同一程序或过程有多个调用堆栈项时，使用最近的调用信息队列。

除调用信息队列之外，每个 OS/400 作业还包含一个**外部信息队列**。运行于作业中的所有程序和过程都可通过使用此队列来在交互式作业和工作站用户之间发送和接收信息。

有关如何发送和接收异常信息的更多信息，参考 *System API Reference* 中的信息处理 API。

异常信息以及如何发送它们

此题目描述不同的异常信息类型和发送异常信息的方法。

ILE 和 OPM 的错误处理基于异常信息类型。除非另有限定，否则术语**异常信息**指示任何这些信息类型:

脱离 (*ESCAPE)
指示一错误导致程序异常结束，未完成其工作。在发送脱离异常信息之后，您将不会接收到控制。

状态 (*STATUS)
描述程序正在执行的工作的状态。在发送此信息类型之后，您可能会接收到控制。接收到控制与否取决于接收程序处理状态信息的方法。

通知 (*NOTIFY)

描述要求校正操作或要求调用程序回答的状态。在发送此信息类型之后，您可能会接收到控制。接收到控制与否取决于接收程序处理通知信息的方法。

功能检查

描述程序未期望的结束条件。 ILE 功能检查 CEE9901 是仅由系统发送的特殊信息类型。OPM 功能检查是信息 ID 为 CPF9999 的脱离信息类型。

有关这些信息类型和其他 OS/400 信息类型的信息，参考 *System API Reference*。

异常信息是通过下列方法发送的:

- 由系统生成

  OS/400（包括 HLL）生成异常信息，指示程序设计错误或状态信息。

- 信息处理程序 API

  可使用“发送程序信息”(QMHSNDPM) API 来将异常信息发送至特定调用信息队列。

- ILE API

  可使用指示状态 (CEESGL) 可联编 API 来生成 ILE 状态。此状态导致发送脱离异常信息或状态异常信息。

- 特定于语言的动词

  对于 ILE C/400，raise() 函数生成 C 信号。 ILE RPG/400 和 ILE COBOL/400 都没有类似的函数。

如何处理异常信息

当您或系统发送异常信息时，异常处理开始。此处理继续，直到处理了该异常为止，即是当将异常信息修改为指示它已得到处理时为止。

当对 OPM 调用信息队列调用异常处理程序时，系统修改异常信息，指示它已得到处理。在对 ILE 调用信息队列调用异常处理程序之前，ILE HLL 修改异常信息。结果是，调用处理程序时，特定于 HLL 的错误处理认为该异常信息已得到处理。即使不使用特定于 HLL 的错误处理，ILE HLL 也可处理异常信息，或允许异常处理继续。参考 ILE HLL 参考手册以确定未处理的异常信息的 HLL 缺省操作。

为了允许您旁路特定于语言的错误处理，我们为 ILE 定义了附加的能力。这些能力包括直接监控处理程序和 ILE 状态处理程序。使用这些能力时，您负责修改异常信息，以指示异常已得到处理。若不修改异常信息，则系统通过尝试找出另一异常处理程序来继续异常处理。题目第39页的『异常处理程序的类型』包含有关直接监控处理程序和 ILE 状态处理程序的细节。要修改异常信息，参考 *System API Reference* 中的更改异常信息 (QMHCHGEM) API。

异常恢复

在发送异常之后，您可能想继续处理。从错误恢复是很有用的应用程序工具，它允许传递容错的应用程序。对于 ILE 和 OPM 程序，系统已定义**继续点**这一概念。继续点初始设置为紧跟着异常发生后面的指令。处理异常之后，可在继续点继续处理。有关如何使用和修改继续点的更多信息，参考第91页的『第8章 异常和状态管理』。

未处理的异常的缺省操作

若不在 HLL 中处理异常信息，则系统对未处理的异常执行缺省操作。

第36页的图25显示了未处理的异常的缺省操作，这些缺省操作基于是将该异常发送至 OPM 还是 ILE 程序。 OPM 和 ILE 的不同缺省操作导致错误处理能力存在基本差异。

对于 OPM，未处理的异常生成称为功能检查信息的特殊脱离信息。此信息带有给定的特殊信息 ID CPF9999。将把它发送至生成原始异常信息的调用堆栈项的调用信息队列。若不处理此功能检查信息，则系统除去该调用堆栈项。系统然后将此功能检查信息发送至前一个调用堆栈项。此处理继续，直到此功能检查信息得到处理为止。若此功能检查信息未得到处理，则作业结束。

对于 ILE，将未处理的异常信息渗滤至前一调用堆栈项信息队列。当将异常信息移至前一调用信息队列时，发生**渗滤**。其作用是将同一异常信息发送至前一调用信息队列。发生此情况时，异常处理在前一调用堆栈项处继续。

第39页的图26显示了 ILE 中未处理的异常信息。在此示例中，过程 P1 是控制边界。过程 P1 也是激活组中最旧的调用堆栈项。过程 P4 生成未处理的异常信息。未处理的异常的渗滤继续，直到到达任一控制边界或异常信息得到处理为止。当将未处理的异常渗滤至控制边界时，将其转换为功能检查。若该异常是脱离，则生成功能检查。若它是通知异常，则发送缺省回答、处理该异常，并允许该通知的发送方继续。若它是状态异常，则处理该异常，并允许该状态的发送方继续。（过程 P3 中显示的）继续点用来定义调用堆栈项，功能检查的异常处理应在该处继续。对于 ILE，下一处理步骤是将特殊功能检查异常信息发送至此调用堆栈项。在此示例中，这是过程 P3。

现在可处理功能检查异常信息，或将其渗滤至控制边界。若它已得到处理，则正常处理继续，异常处理结束。若将功能检查信息渗滤至控制边界，则 ILE 认为该应用程序已因预期不到的错误而结束。ILE 对所有语言定义了类属故障异常信息。此信息是 CEE9901，ILE 将其发送至控制边界的调用者。

ILE 中定义的未处理的异常信息的缺省操作允许您从混合语言应用程序中发生的错误状态中恢复。对于意外的错误，ILE 对所有语言强制故障信息为一致的。这改进了集成来自不同源的应用程序的能力。

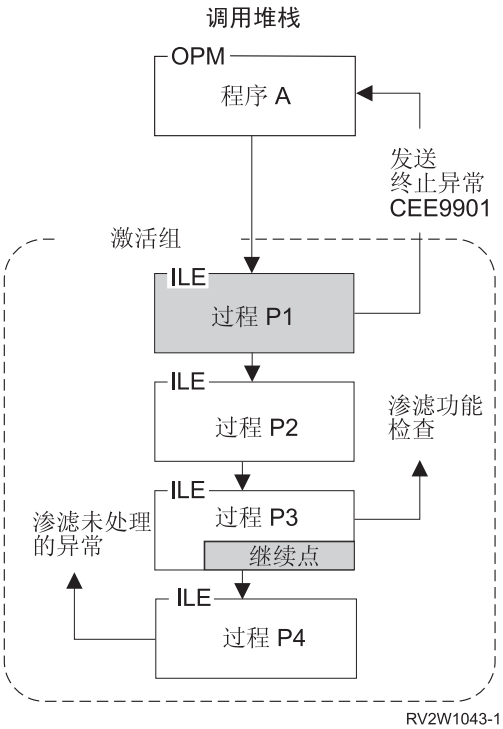

图 26. 未处理的异常缺省操作

异常处理程序的类型

此题目提供了为 OPM 和 ILE 程序提供的异常处理程序的概述。如第36页的图25所示，这是异常信息体系结构的顶层。与 OPM 相比，ILE 提供了附加的异常处理能力。

对于 OPM 程序，特定于 HLL 的错误处理为每个调用堆栈项提供了一个或多个处理例程。当将异常发送至 OPM 程序时，系统调用适当的例程。

ILE 中特定于 HLL 的错误处理提供了相同的能力。但是，ILE 有附加的异常处理程序类型。这些类型的处理程序提供了对异常信息体系结构的直接控制，并允许旁路特定于 HLL 的错误处理。 ILE 的附加处理程序类型是:

直接监控处理程序

ILE 状态处理程序

要确定您的 HLL 是否支持这些类型的处理程序，参考 ILE HLL 程序员指南。

直接监控处理程序允许在有限的 HLL 源语句周围直接说明异常监控程序。对于 ILE C/400，此能力是通过 #pragma 指令启用的。 ILE COBOL/400 并不以与 ILE C/400 相同的方式在有限 HLL 源语句周围直接说明异常监控程序。 ILE COBOL/400 程序不能直接在任意源代码周围编码处理程序的启用和禁用。但是，内部映射了诸如

```
ADD a TO b ON SIZE ERROR imperative
```

之类的语句以使用同一机制。因此，就哪个处理程序首先获取控制的优先级而言，这样的限于语句的条件祈使句在（通过 CEEHDLR 注册的）ILE 状态处理程序之前获取控制。然后，控制转至 COBOL 中的 USE 说明。

ILE 状态处理程序允许在运行期**注册**异常处理程序。 ILE 状态处理程序是对特定调用堆栈项注册的。要注册 ILE 状态处理程序，使用注册用户编写的状态处理程序 (CEEHDLR) 可联编 API。此 API 允许在运行期标识发生异常时应获取控制的过程。CEEHDLR API 需要在语言中说明和设置过程指针的能力。CEEHDLR 是以内部函数形式实现的。因此，不能指定其地址，也不能通过过程指针调用它。可通过调用注销用户编写的状态处理程序 (CEEHDLR) 可联编 API 来**注销** ILE 状态处理程序。

OPM 和 ILE 支持特定于 HLL 的处理程序。**特定于 HLL 的处理程序**是为了处理错误而定义的语言功能部件。例如，可使用 ILE C/400 signal 函数来处理异常信息。 RPG 中特定于 HLL 的错误处理包括编码 *PSSR 和 INFSR 子例程的能力。 COBOL 中特定于 HLL 的错误处理包括用于 I/O 错误处理的 USE 说明，以及限于语句的条件短语（如 ON SIZE ERROR 和 AT INVALID KEY）中的祈使句。

若既使用特定于 HLL 的错误处理又使用附加的 ILE 异常处理程序类型，则异常处理程序优先级变得很重要。

第41页的图27显示了过程 P2 的调用堆栈项。在此示例中，已对单一调用堆栈项定义了全部三种类型的处理程序。虽然这可能不是典型示例，但却是有可能定义全部三种类型的。因为定义了全部三种类型，所以定义异常处理程序优先级。此图显示了此优先级。发送异常信息时，按以下次序调用异常处理程序:

1. 直接监控处理程序

   首先选择调用，然后选择该调用中处理程序的相对次序。在调用中，所有直接监控处理程序和 COBOL 限于语句的条件祈使句都在 ILE 状态处理程序之前获取控制。类似的，ILE 状态处理程序在其他特定于 HLL 的处理程序之前获取控制。

   若已在引起异常的语句周围说明了直接监控处理程序，则在调用特定于 HLL 的处理程序之前调用这些处理程序。例如，若第41页的图27中的过程 P2 带有特定于 HLL 的处理程序，而过程 P1 带有直接监控处理程序，则在考虑 P1 的直接监控处理程序之前考虑 P2 的处理程序。

   可在语法上嵌套直接监控程序。在指定同一优先级号的多个嵌套监控程序中，首先选择嵌套得最深的直接监控程序中指定的处理程序。

2. ILE 状态处理程序

   若已对调用堆栈项注册了 ILE 状态处理程序，则接着调用此处理程序。可注册多个 ILE 状态处理程序。在示例中，过程 P5 和过程 P6 是 ILE 状态处理程序。当对同一调用堆栈项注册了多个 ILE 状态处理程序时，系统按后进先出 (LIFO) 次序调用这些处理程序。若将 COBOL 限于语句使用的条件祈使句归类成特定于 HLL 的处理程序，则那些祈使句优先于 ILE 状态处理程序。

3. 特定于 HLL 的处理程序

   最后调用特定于 HLL 的处理程序。

当将异常信息修改为显示它已得到处理时，系统结束异常处理。若正在使用直接监控处理程序或 ILE 状态处理程序，则修改异常信息是您的责任。可使用数种控制操作。例如，可将处理指定为控制操作。只要异常信息仍未得到处理，系统就使用先前定义的优先级继续搜索异常处理程序。若未在当前调用堆栈项内处理异常，则将其渗滤至前一调用堆栈项。若不使用特定于 HLL 的错误处理，则 ILE HLL 可选择允许异常处理在前一调用堆栈项处继续。

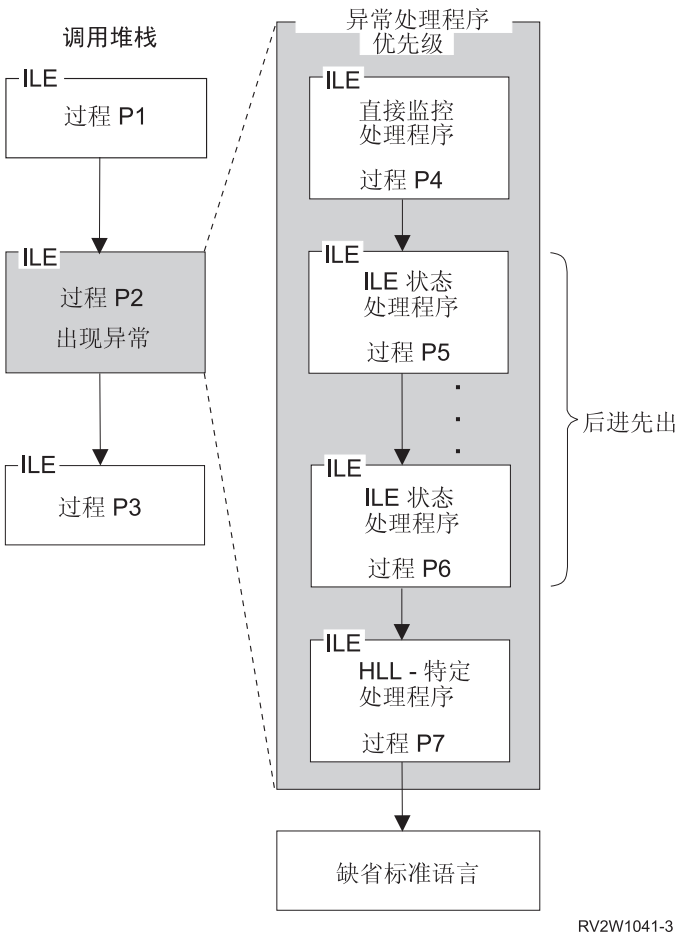

图 27. 异常处理程序优先级

ILE 状态

为了允许更大的系统间一致性，ILE 定义了允许您处理错误状态的功能部件。 ILE **状态**是 HLL 中的错误状态的与系统无关的表示法。对于 OS/400 操作系统，每个 ILE 状态都有相对应的异常信息。 ILE 状态由状态记号表示。**状态记号**是 12 个字节的数据结构，它在多个 SAA 参与系统之间是一致的。此数据结构包含允许您将状态与基础异常信息相关联的信息。

先前描述的 ILE 状态处理程序和渗滤模型与 SAA 体系结构一致。要编写在系统之间一致的程序，需要使用 ILE 状态处理程序和 ILE 状态记号。有关 ILE 状态的更多信息，参考第91页的『第8章 异常和状态管理』。

数据管理范围限定规则

数据管理范围限定规则控制数据管理资源的使用。这些资源是允许程序使用数据管理的临时对象。例如，当程序打开文件时，将建立称为开放数据通路 (ODP) 的对象来将程序与该文件相连。当程序建立覆盖来更改处理文件的方式时，系统建立覆盖对象。

数据管理范围限定规则确定在调用堆栈上运行的多个程序或过程何时可共享资源。例如，许多程序可同时使用用 SHARE(*YES) 参数值建立的打开文件或确认定义对象。共享数据管理资源的能力取决于数据管理资源的范围限定级别。

数据管理范围限定规则还确定资源的存在情况。根据范围限定规则，系统自动删除作业中不使用的资源。这种自动清理操作的结果是，作业使用较少的存储器，作业性能得到改进。

ILE 将 OPM 和 ILE 程序的数据管理范围限定规则的形式分成下列范围限定级别:

调用

激活组

作业

根据正在使用的数据管理资源，可明确地指定一个或多个范围限定级别。若不选择范围限定级别，则系统选择其中一个级别作为缺省级别。

参考第103页的『第10章 数据管理范围限定』以获取有关每种数据管理资源如何支持范围限定级别的信息。有关附加细节，参考 *Data Management* 一书。

调用级范围限定

当数据管理资源与建立该资源的调用堆栈项相连时，发生调用级范围限定。图28显示了示例。对于在缺省激活组中运行的程序来说，调用级范围限定通常是缺省范围限定级。在此图中，OPM 程序 A、OPM 程序 B 或 ILE 过程 P2 可选择在不关闭其各自的文件 F1、F2 或 F3 的情况下返回。数据管理将每个文件的 ODP 与打开该文件的调用级号相关联。可使用 RCLRSC 命令来根据传送给该命令的特定调用级号关闭文件。

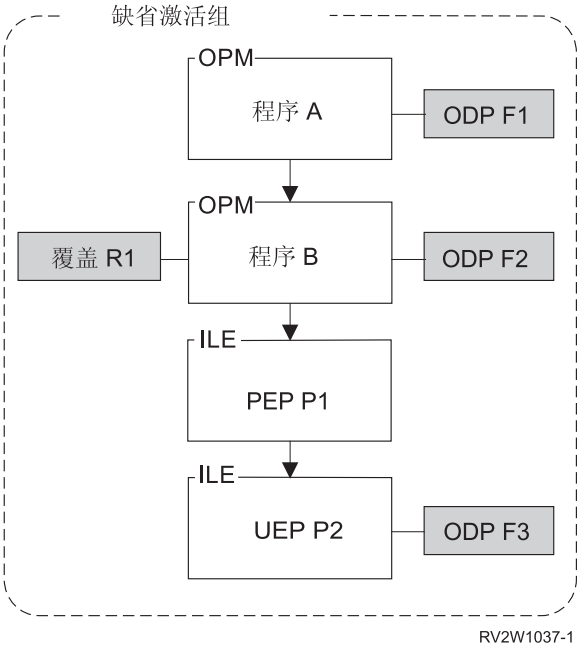

图 28. 调用级范围限定. 可将 ODP 和覆盖的范围限于调用级。

当相对应的调用堆栈项返回时，删除限于特定调用级使用的覆盖。建立覆盖的调用级下面的任何调用堆栈项均可共享该覆盖。

激活组级范围限定

当将数据管理资源与建立该资源的 ILE 程序或 ILE 服务程序的激活组相连时，发生激活组级范围限定。删除激活组时，数据管理关闭与该激活组相关联的、该激活组中运行的程序未关闭的所有资源。 图29显示了激活组级范围限定的示例。对于不在缺省激活组中运行的 ILE 过程使用的大多数数据管理资源类型来说，激活组级范围限定是缺省范围限定级。例如，此图显示了限于激活组使用的文件 F1、F2 和 F3 的 ODP 以及覆盖 R1。

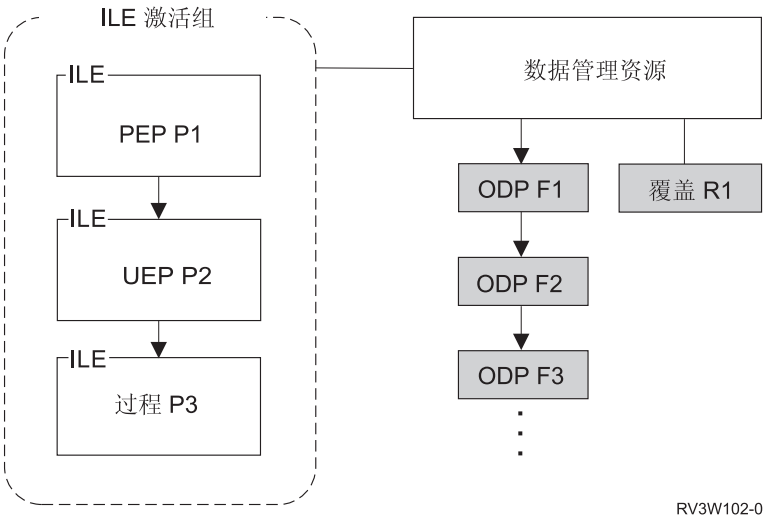

图 29. 激活组级范围限定. 可将 ODP 和覆盖限于某激活组使用。

共享限于激活组使用的数据管理资源的能力限于在该激活组中运行的程序。这提供了应用程序隔离和保护。例如，假设图中的文件 F1 是用 SHARE(*YES) 参数值打开的。同一激活组中运行的任何 ILE 过程均可使用文件 F1。另一激活组中对文件 F1 的另一打开操作导致建立该文件的另一个 ODP。

作业级范围限定

当数据管理资源与作业相连时，发生作业级范围限定。 OPM 和 ILE 程序都可使用作业级范围限定。作业级范围限定允许在运行于不同激活组中的程序之间共享数据管理资源。如前一题目所述，将资源限于某激活组使用导致只有该激活组中运行的程序才可共享该资源。作业级范围限定允许在运行于作业中的所有 ILE 和 OPM 程序之间共享数据管理资源。

第44页的图30显示了作业级范围限定的示例。程序 A 可能已打开文件 F1，并指定作业级范围限定。此文件的 ODP 与作业相连。除非作业结束，否则系统不关闭该文件。若已用 SHARE(YES) 参数值建立了 ODP，则任何 OPM 程序或 ILE 过程均可潜在地共享该文件。

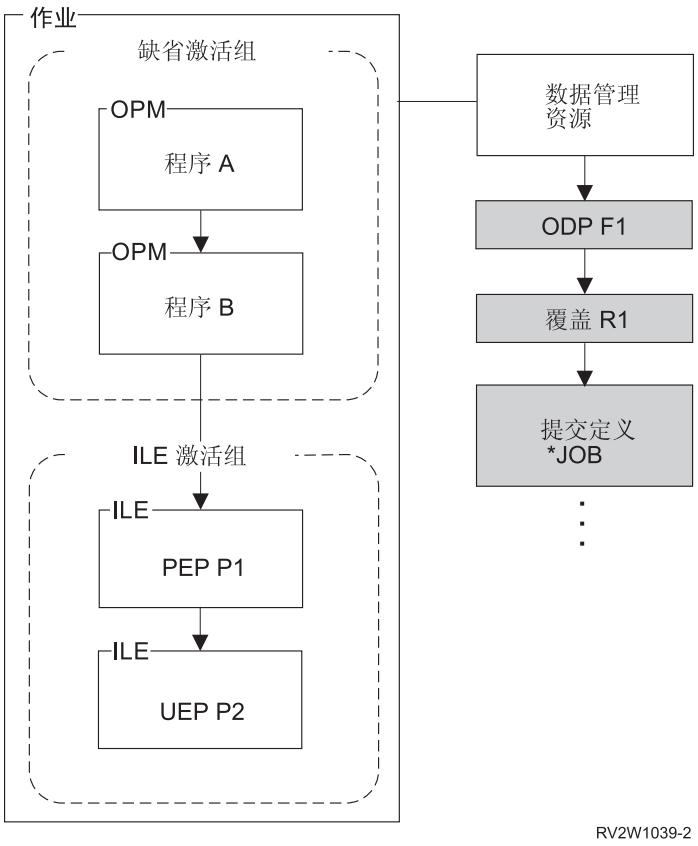

图 30. 作业级范围限定. 可将 ODP、覆盖和确认定义限于作业级。

限于作业级使用的覆盖影响该作业中的所有打开文件操作。在此示例中，过程 P2 可能已建立覆盖 R1。在明确地删除作业级覆盖或作业结束之前，作业级覆盖保持活动。当发生合并时，作业级覆盖是最高优先级的覆盖。这是因为当调用堆栈上存在多个覆盖时，调用级覆盖合并到一起。

可通过使用覆盖命令、确认控制命令上的范围限定参数，以及通过各种 API 来明确地指定数据管理范围限定级。使用范围限定规则的数据管理资源的完整列表在第103页的『第10章 数据管理范围限定』中。

第4章 程序建立概念

在设计和维护应用程序方面，建立 ILE 程序或服务程序的过程提供了更大的灵活性和控制。此过程包括两个步骤:

1. 将源代码编译成模块。
2. 将模块联编成 ILE 程序或服务程序。当运行“建立程序”(CRTPGM) 或“建立服务程序”(CRTSRVPGM) 命令时，联编发生。

本章说明与联编器和与建立 ILE 程序或服务程序的过程相关联的概念。在阅读本章之前，您应熟悉第11页的『第2章 ILE 基本概念』中描述的联编概念。

建立程序和建立服务程序命令

“建立程序”(CRTPGM) 和“建立服务程序”(CRTSRVPGM) 命令看起来类似，并共享许多相同的参数。比较两个命令中使用的参数可帮助理解如何使用每一命令。

表2显示这些命令及其提供了缺省值的参数。

表 2. CRTPGM 和 CRTSRVPGM 命令的参数

参数组	CRTPGM 命令	CRTSRVPGM 命令
标识	PGM(*CURLIB/WORK) MODULE(*PGM)	SRVPGM(*CURLIB/UTILITY) MODULE(*SRVPGM)
程序存取	ENTMOD(*FIRST)	EXPORT(*SRCFILE) SRCFILE(*LIBL/QSRVSRC) SRCMBR(*SRVPGM)
联编	BNDSRVPGM(*NONE) BNDDIR(*NONE)	BNDSRVPGM(*NONE) BNDDIR(*NONE)
运行期	ACTGRP(*NEW)	ACTGRP(*CALLER)
其他	OPTION(*GEN *NODUPPROC *NODUPVAR *WARN *RSLVREF) DETAIL(*NONE) ALWUPD(*YES) ALWRINZ(*NO) REPLACE(*YES) AUT(*LIBCRTAUT) TEXT(*ENTMODTXT) TGTRLS(*CURRENT) USRPRF(*USER)	OPTION(*GEN *NODUPPROC *NODUPVAR *WARN *RSLVREF) DETAIL(*NONE) ALWUPD(*YES) ALWRINZ(*NO) REPLACE(*YES) AUT(*LIBCRTAUT) TEXT(*ENTMODTXT) TGTRLS(*CURRENT) USRPRF(*USER)

两条命令的标识参数命名要建立的对象和复制的模块。两个参数之间的唯一区别是建立对象时使用的缺省模块名。对于 CRTPGM，对模块使用与在程序 (*PGM) 参数上指定的名称相同的名称。对于 CRTSRVPGM，对模块使用与在服务程序 (*SRVPGM) 参数上指定的名称相同的名称。否则，这些参数的外观和操作都相同。

两条命令之间最显著的相似性是联编器在调入和调出之间解析符号的方式。在两种情况下，联编器都处理来自模块 (MODULE)、已联编的服务程序 (BNDSRVPGM) 和联编目录 (BNDDIR) 参数的输入。

两条命令之间最显著的差异是程序存取参数（参见第53页的『程序存取』）。对于 CRTPGM 命令，要对联编器标识的只是哪一个模块具有程序入口过程。一旦建立程序，并对此程序进行动态程序调用，处理就以包含程序入口过程的模块开始。因为 CRTSRVPGM 命令可对其他程序或服务程序提供数个存取点的接口，所以它需要更多的程序存取信息。

使用被采用权限 (QUSEADPAUT)

QUSEADPAUT 系统值定义哪些用户可建立具有使用被采用权限 (USEADPAUT(*YES)) 属性的程序。若用户具有必需的权限，则 QUSEADPAUT 系统值授权的所有用户可建立或更改程序和服务程序，以使用被采用权限。参见 *Security - Reference* 以找出所需的权限。

此系统值可包含授权列表的名称。对此列表检查用户的权限。若用户对命名授权列表至少具有 *USE 权限，则该用户可建立、更改或更新具有 USEADPAUT(*YES) 属性的程序或服务程序。对授权列表的权限不能来自被采用权限。

若授权列表是在系统值中命名的，且该授权列表丢失，则正在尝试的功能将不会完成。发送一条信息指示此情况。但是，若程序是用 QPRCRTPG API 建立的，且在选项模板中指定了 *NOADPAUT 值，则即使授权列表不存在，也将成功地建立该程序。若在命令或 API 上请求多个功能，且授权列表丢失，则不执行该功能。若找不到授权列表时正在尝试的命令是“建立 Pascal 程序”(CRTPASPGM) 或“建立 Basic 程序”(CRTBASPGM)，则结果是功能检查。

表 3. 可能的 QUSEADPAUT 值

值	说明
授权列表名	若所有下列各项均为真，则发出诊断信息，指示程序是用 USEADPAUT(*NO) 建立的: • 对 QUSEADPAUT 系统值指定了授权列表。 • 用户对上面提及的授权列表不具有权限。 • 建立程序或服务程序时，没有其他错误。 若用户对授权列表具有权限，则程序或服务程序是用 USEADPAUT(*YES) 建立的。
***NONE**	若用户具有必需的权限，则 QUSEADPAUT 系统值授权的所有用户可建立或更改程序和服务程序，以使用被采用权限。参见 *Security - Reference* 以找出所需的权限。

有关 QUSEADPAUT 系统值的更多信息，参见 *Security - Reference*。

符号解析

符号解析是联编器为了匹配下列各项而进行的过程:

- 来自要通过复制联编的模块集的调入请求
- 指定的模块和服务程序提供的调出集

可将符号解析期间要使用的调出集认为是有序（顺序编号的）列表。调出的次序是由下列各项确定的:

- CRTPGM 或 CRTSRVPGM 命令的 MODULE、BNDSRVPGM 和 BNDDIR 参数上指定对象的次序
- 从指定模块的语言运行期例行程序的调出

已解析的和未解析的调入

调入和调出都由过程或数据类型和名称组成。**未解析的调入**是类型和名称尚未与调出的类型和名称相匹配的调入。**已解析的调入**是类型和名称与调出的类型和名称完全匹配的调入。

只有来自通过复制联编的模块的调入才进入未解析的调入列表。符号解析期间，使用下一未解析的调入来搜索调出的有序列表以寻找匹配。若在检查有序调出集之后存在未解析的调入，则通常不建立程序对象或服务程序。但是，若在选项参数上指定了 *UNRSLVREF，则可建立带有未解析调入的程序对象或服务程序。若这样的程序对象或服务程序尝试在运行期使用未解析的调入，则发生下列情况:

- 若程序对象或服务程序是为 “版本 2 发行版 3” 的系统建立或更新的，则发出错误信息 MCH3203。该信息为『机器指令中发生函数错误』。
- 若程序对象或服务程序是为 “版本 3 发行版 1” 的系统建立或更新的，则发出错误信息 MCH4439。该信息为『试图使用未解析的调入』。

通过复制联编

MODULE 参数上指定的模块总是通过复制联编的。通过复制联编 BNDDIR 参数指定的联编目录中命名的模块（若需要的话）。在下列任一情况下，需要联编目录中命名的模块:

- 该模块为未解析的调入提供调出
- 该模块提供正在用来建立服务程序的联编器语言源文件的当前调出块中命名的调出

若在联编器语言中找到的调出来自模块对象，则该模块总是通过复制联编的，无论它是在命令行上显式提供的，还是来自联编目录。例如,

```
模块 M1: 调入 P2
模块 M2: 调出 P2
模块 M3: 调出 P3
联编器语言 S1:      STRPGMEXP PGMLVL(*CURRENT)
                      EXPORT P3
                      ENDPGMEXP
联编目录 BNDDIR1:           M2
                            M3
CRTSRVPGM SRVPGM(MYLIB/SRV1) MODULE(MYLIB/M1) SRCFILE(MYLIB/S1)
          SRCMBR(S1) BNDDIR(MYLIB/BNDDIR1)
```

服务程序 SRV1 将有三个模块: M1、M2 和 M3。因为 P3 在当前调出块中，所以复制 M3。

通过引用联编

BNDSRVPGM 参数上指定的服务程序是通过引用联编的。若在联编目录中命名的服务程序向未解析的调入提供调出，则通过引用联编该服务程序。以此方式联编的服务程序不添加新的调入。

注: 要更好地控制与您的程序联编的事项，指定类属服务程序名或特定的库。仅当确切知道与您的程序联编的事项时，才应在用户控制的环境中指定值 *LIBL。不要用 OPTION(*DUPPROC *DUPVAR) 指定 BNDSRVPGM(*LIBL/*ALL)。用 *ALL 指定 *LIBL 可能会在程序运行期导致不可预测的结果。

联编大量的模块

对于 CRTPGM 和 CRTSRVPGM 命令上的模块 (MODULE) 参数，可指定的模块数是有限制的。若要联编的模块数超过此限制，则可使用下列其中一种方法:

1. 使用联编目录来联编提供其他模块所需的调出的大量模块。
2. 使用允许在 CRTPGM 和 CRTSRVPGM 命令上的 MODULE 参数指定类属模块名的模块命名约定。例如，`CRTPGM PGM(mylib/payroll) MODULE(mylib/pay*)`。名称以 `pay` 开始的所有模块都无条件地包括在程序 `mylib/payroll` 中。因此，仔细地挑选命名约定，以使 CRTPGM 或 CRTSRVPGM 命令上指定的类属名不联编不想要的模块。
3. 将模块分组成单独的库，以使值 *ALL 可配合 MODULE 参数上的特定库名使用。例如，`CRTPGM PGM(mylib/payroll) MODULE(payroll/*ALL)`。库 `payroll` 中的每个模块都无条件地包括在程序 `mylib/payroll` 中。
4. 使用方法 2 和 3 中描述的类属名与特定库的组合。
5. 对于服务程序，使用联编源语言。联编源语言中指定的调出导致联编模块（若它满足调出的话）。 RTVBNDSRC 命令可帮助您建立联编源语言。虽然 RTVBNDSRC 命令上的 MODULE 参数限制了可在 MODULE 参数上显式指定的模块数，但可将类属模块名和值 *ALL 与特定库名配合使用。可多次使用 RTVBNDSRC 命令，并将输出引向同一源文件。但是，在此情况下，您可能需要编辑联编源语言。

调出次序的重要性

只需稍稍更改一下命令，就可建立不同的、但具有潜在等效性的程序。仅当下列两项都为真时，MODULE、BNDSRVPGM 和 BNDDIR 参数上指定对象的次序通常才是重要的:

- 多个模块或服务程序正在调出重复的符号名
- 另一模块需要调入符号名

大多数应用程序没有重复的符号，程序员很少需要担心指定对象的次序。对于那些将重复符号既调出又调入的应用程序，考虑在 CRTPGM 或 CRTSRVPGM 命令上列示对象的次序。

下列示例显示符号解析如何工作。将第49页的图31中的模块、服务程序和联编目录用于第50页的图32和第52页的图33中的 CRTPGM 请求。假设所有标识的调出和调入都是过程。

这些示例还显示了在程序建立过程中联编目录的作用。假设库 MYLIB 在 CRTPGM 和 CRTSRVPGM 命令的库列表中。以下命令在库 MYLIB 中建立联编目录 L:

```
CRTBNDDIR BNDDIR(MYLIB/L)
```

以下命令将模块 M1 和 M2 的名称以及服务程序 S 和 T 的名称添加至联编目录 L:

```
ADDBNDDIRE BNDDIR(MYLIB/L) OBJ((M1 *MODULE) (M2 *MODULE) (S) (T))
```

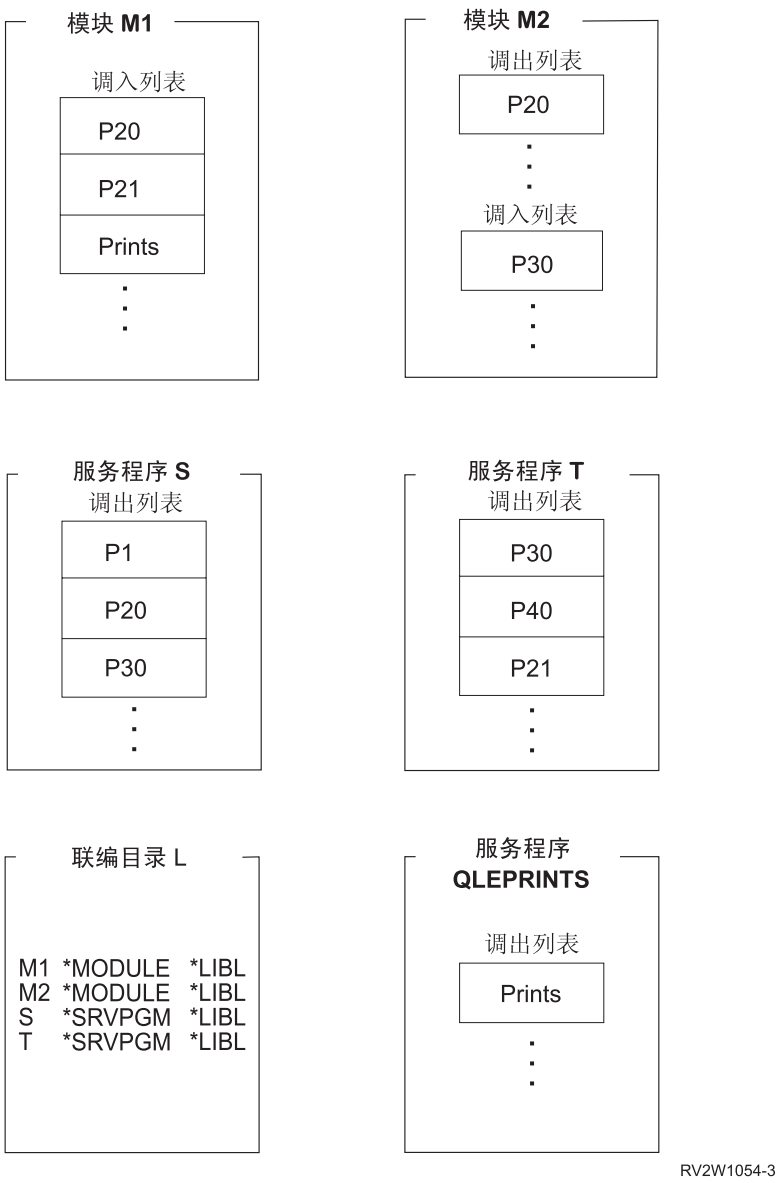

图 31. 模块、服务程序和联编目录

程序建立示例 1

假设使用了以下命令来在第50页的图32中建立程序 A:

```
CRTPGM  PGM(TEST/A)
        MODULE(*LIBL/M1)
        BNDSRVPGM(*LIBL/S)
        BNDDIR(*LIBL/L)
        OPTION(*DUPPROC)
```

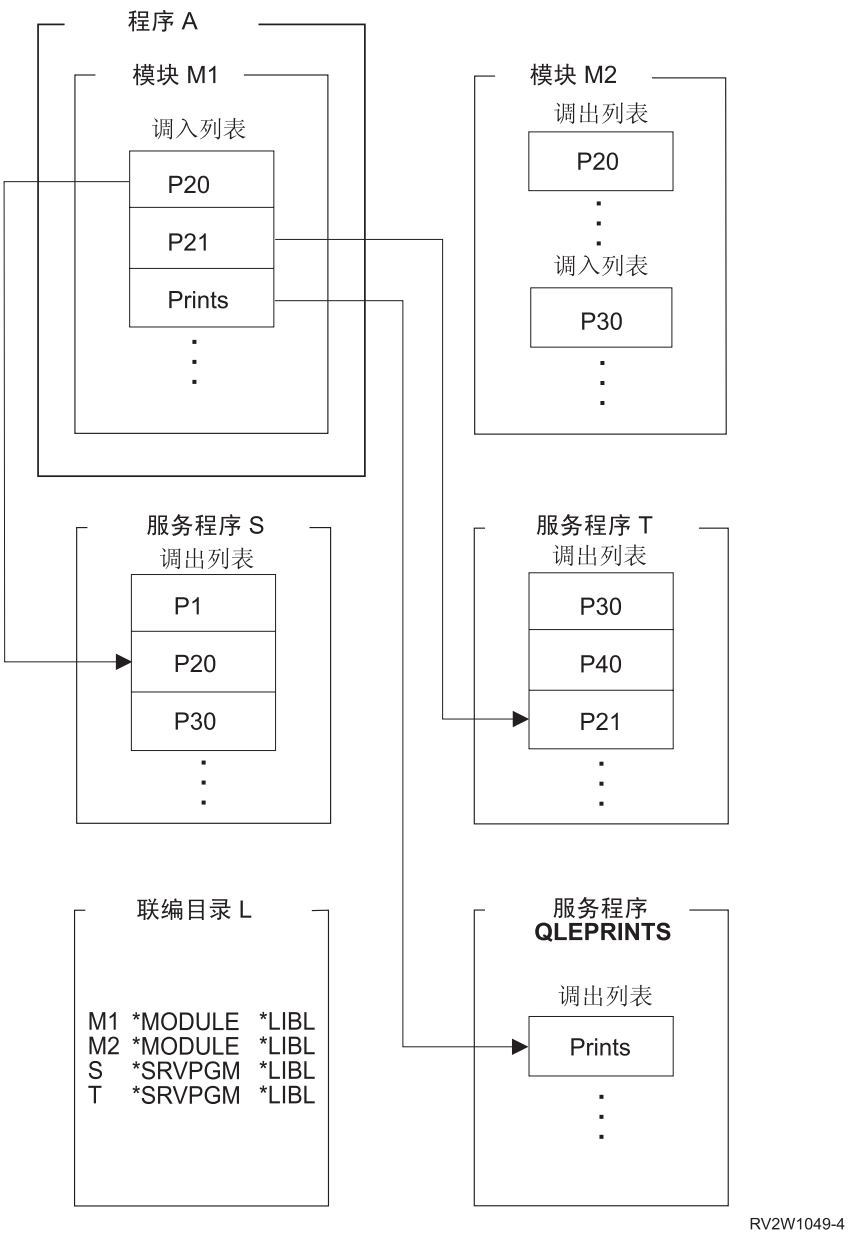

图 32. 符号解析和程序建立: 示例 1

要建立程序 A，联编器以指定的次序处理 CRTPGM 命令参数上指定的对象:

1. 第一个参数 (PGM) 上指定的值是 A，这是要建立的程序的名称。
2. 第二个参数（模块）上指定的值是 M1。联编器从那里开始。模块 M1 包含三个需要解析的调入: P20、P21 和 Prints。
3. 第三个参数 (BNDSRVPGM) 上指定的值是 S。联编器扫描服务程序 S 的调出列表以找出解析任何未解析调入请求的任何过程。因为调出列表包含过程 P20，所以解析该调入请求。
4. 第四个参数 (BNDDIR) 上指定的值是 L。联编器接着扫描联编目录 L。
   a. 联编目录中指定的第一个对象是模块 M1。因为已在模块参数上指定模块 M1，所以当前它是已知的，但它未提供任何调出。

b. 联编目录中指定的第二个对象是模块 M2。模块 M2 提供了调出，但没有一个调出与任何当前未解析的调入请求（P21 和 Prints）相匹配。

c. 联编目录中指定的第三个对象是服务程序 S。在步骤第50页的3中已处理服务程序 S，它未提供任何附加的调出。

d. 联编目录中指定的第四个对象是服务程序 T。联编器扫描服务程序 T 的调出列表。找到过程 P21，它解析该调入请求。

5. 任何参数上都未指定最后一个需要解析的调入 (Prints)。然而，联编器在服务程序 QLEPRINTS 的调出列表中找到 Prints 过程，在此示例中，QLEPRINTS 是编译程序提供的公共运行期例行程序。编译模块时，编译程序指定联编目录（它包含自己的运行期服务程序和 ILE 运行期服务程序）为缺省值。这就是联编器如何知道它应在编译程序提供的运行期服务程序中寻找任何剩余的未解析引用的原因。若在联编器查看运行期服务程序之后有不能解析的引用，则通常联编会失败。但是，若在建立命令上指定 OPTION(*UNRSLVREF)，则建立程序。

程序建立示例 2

第52页的图33显示了类似的 CRTPGM 请求的结果，不同的是已除去 BNDSRVPGM 参数上的服务程序:

```
CRTPGM  PGM(TEST/A)
        MODULE(*LIBL/M1)
        BNDDIR(*LIBL/L)
        OPTION(*DUPPROC)
```

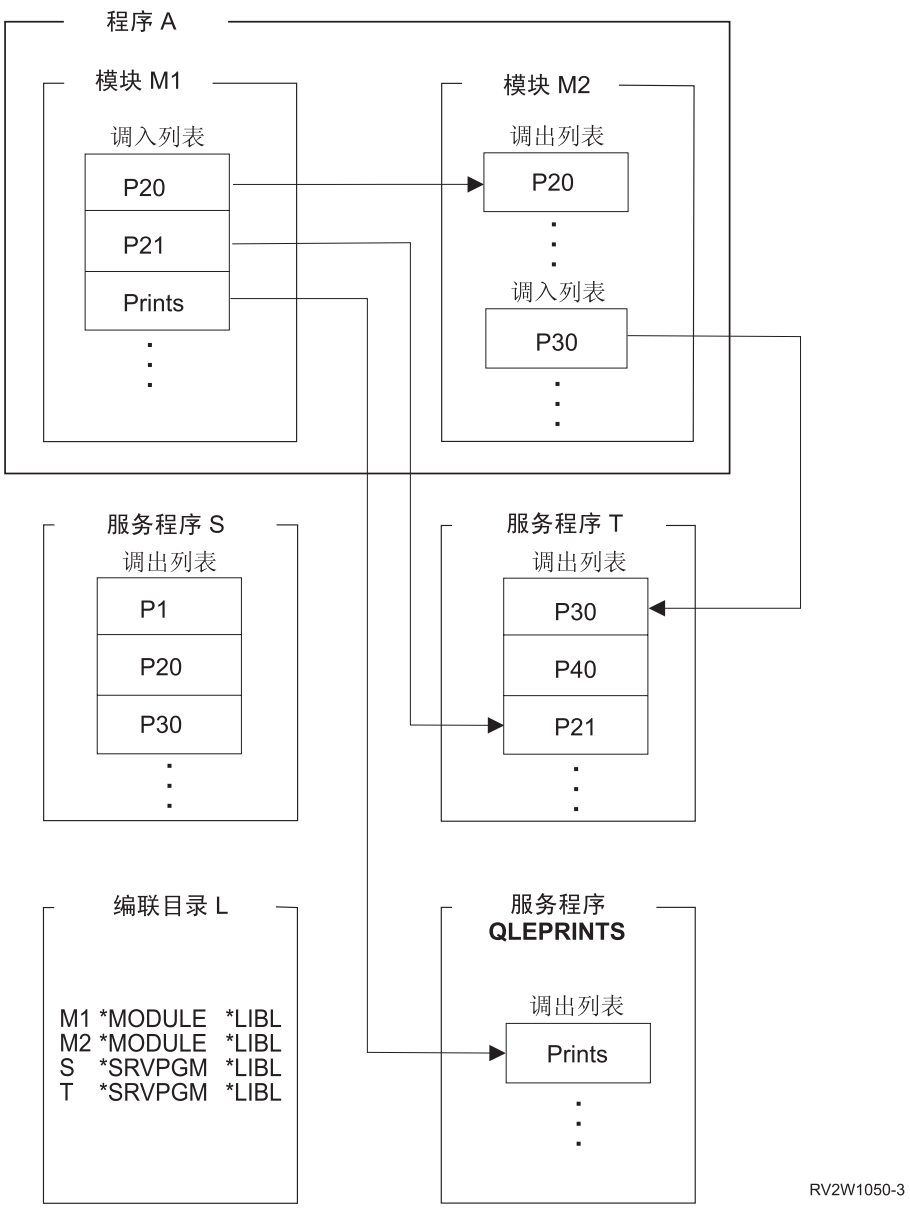

图 33. 符号解析和程序建立: 示例 2

更改要处理对象的定序将更改调出的定序。它还导致建立的程序与示例 1 中建立的程序不同。因为 CRTPGM 命令的 BNDSRVPGM 参数上未指定服务程序 S，所以处理联编目录。模块 M2 调出过程 P20，在联编目录中，它是在服务程序 S 之前指定的。因此，在此示例中模块 M2 被复制至结果程序对象。当将第50页的图32与图33相比较时，您看到下列各项:

- 示例 1 中的程序 A 仅包含模块 M1，且使用服务程序 S、T 和 QLEPRINTS 中的过程。
- 在示例 2 的程序 A 中，两个称为 M1 和 M2 的模块使用服务程序 T 和 QLEPRINTS。

示例 2 中的程序建立如下:

1. 第一个参数 (PGM) 指定要建立的程序的名称。

2. 第二个参数 (MODULE) 上指定的值是 M1，因此联编器再次从那里开始。模块 M1 包含同样三个需要解析的调入：P20、P21 和 Prints。
3. 这次，指定的第三个参数不是 BNDSRVPGM。它是 BNDDIR。因此，联编器首先扫描指定的联编目录 (L)。
   a. 联编目录中指定的第一项是模块 M1。模块参数已处理此库中的模块 M1。
   b. 联编目录中指定的第二项是用于模块 M2 的。联编器扫描模块 M2 的调出列表。因为该调出列表包含 P20，所以解析该调入请求。模块 M2 是通过复制联编的，必须将其调入添加至未解析的调入请求列表以供处理。未解析的调入请求现在是 P21、Prints 和 P30。
   c. 处理继续至联编目录中指定的下一对象，它是服务程序 S。在此情形中，服务程序 S 不为当前未解析的调入请求提供任何调出。处理继续至联编目录中列示的下一对象。
4. 服务程序 T 为未解析的调入提供了调出 P21 和 P30。
5. 与示例 1 相同，未指定调入请求 Prints。但是，可在编写模块 M1 的语言提供的运行期例行程序中找到该过程。

符号解析还受调出的强度影响。关于强调出和弱调出的信息，参见第55页的『调入和调出概念』中的“调出”。

程序存取

建立 ILE 程序对象或服务程序对象时，需要指定其他程序如何可存取该程序。在 CRTPGM 命令上，借助于入口模块 (ENTMOD) 参数做到这一点。在 CRTSRVPGM 命令上，借助于调出 (EXPORT) 参数做到这一点（参见第45页的表2）。

CRTPGM 命令上的程序入口过程模块参数

程序入口过程模块 (ENTMOD) 参数告诉联编器下列各项所在的模块的名称:

程序入口过程 (PEP)

用户入口过程 (UEP)

此信息标识哪一个模块包含对建立的程序进行动态调用时获取控制的 PEP。

ENTMOD 参数的缺省值是 *FIRST。此值指定联编器将它在模块参数上指定的模块列表中找到的第一个包含 PEP 的模块用作入口模块。

若存在下列条件:

在 ENTMOD 参数上指定了 *FIRST

遇到带 PEP 的第二个模块

则联编器将第二个模块复制到程序对象中，并继续联编过程。联编器忽略附加的 PEP。

若在 ENTMOD 参数上指定了 *ONLY，则程序中只有一个模块可包含 PEP。若指定了 *ONLY，而遇到带 PEP 的第二个模块，则不建立对象。

要获得显式的控制，可指定包含 PEP 的模块的名称。任何其他 PEP 都被忽略。若显式指定的模块未包含 PEP，则 CRTPGM 请求失败。

要查看模块是否带程序入口过程，使用显示模块 (DSPMOD) 命令。该信息出现在“显示模块信息”屏幕的*程序入口过程名*字段中。若在该字段中指定 *NONE，则表示此模块不带 PEP。若在该字段中指定名称，则表示此模块带有 PEP。

CRTSRVPGM 命令上的调出参数

调出 (EXPORT)、源文件 (SRCFILE) 和源成员 (SRCMBR) 参数标识正在建立的服务程序的公共接口。这些参数指定服务程序使其可供其他 ILE 程序或服务程序使用的调出（过程和数据）。

调出参数的缺省值是 *SRCFILE。该值指示联编器使用 SRCFILE 参数来引用关于服务程序的调出的信息。此附加信息是其中带有联编器语言源的源文件（参见第56页的『联编器语言』）。联编器找出联编器语言源，并根据指定的要调出名称生成一个或多个特征符。联编器语言还允许您指定您选择的特征符，而不是让联编器生成一个特征符。

“检索联编器源”(RTVBNDSRC) 命令可用来建立源文件，该源文件包含基于一个模块或一组模块的调出的联编器语言源。 RTVBNDSRC 命令建立的文件包含可从模块调出的所有符号，这以联编器语言语法指定。可编辑此文件以仅包括想要调出的符号，然后在 CRTSRVPGM 命令的 SRCFILE 参数上指定此文件。

调出参数的另一可能的值是 *ALL。指定 EXPORT(*ALL) 时，从复制的模块调出的所有符号都从服务程序调出。生成的特征符是由下列各项确定的:

- 调出的符号数
- 调出符号的字母次序

若指定 EXPORT(*ALL)，则无需联编器语言即可定义服务程序的调出。因为不必生成联编器语言源，所以此值最容易使用。但是，一旦其他程序使用了这些调出，指定了 EXPORT(*ALL) 的服务程序可能难于更新或校正。若更改服务程序，则调出的次序或数目也会更改。从而，该服务程序的特征符也会更改。若特征符更改，则必须重建使用更改过的服务程序的所有程序或服务程序。

EXPORT(*ALL) 指示从在服务程序中使用的模块调出的所有符号都从服务程序调出。ILE C/400 可将调出定义为全局的或静态的。只有在 ILE C/400 中作为全局定义的外部变量才可配合 EXPORT(*ALL) 使用。在 ILE RPG/400 中，下列各项可配合 EXPORT(*ALL) 使用:

- RPG 程序名（不要与 *PGM 对象相混淆）
- 用关键字 EXPORT 定义的变量

在 ILE COBOL/400 中，下列语言元素是模块调出:

- 编译单元在词法上处于最外面的 COBOL 程序（不要与 *PGM 对象相混淆）的 PROGRAM-ID 段中的名称。这映射为强过程调出。
- COBOL 编译器生成的，根据前面强调符中的 PROGRAM-ID 段中的名称派生的名称（若该程序不具有 INITIAL 属性）。这映射为强过程调出。关于强调出和弱调出的信息，参见第55页的『调入和调出概念』中的“调出”。
- 作为 EXTERNAL 定义的任何数据项或文件项。这映射为弱调出。

配合源文件和源成员参数使用的调出参数

调出参数上的缺省值是 *SRCFILE。若在调出参数上指定 *SRCFILE，则联编器还必须使用 SRCFILE 和 SRCMBR 参数来找出联编器语言源。

以下示例命令通过使用缺省值找出联编器语言源来联编名为 UTILITY 的服务程序:

```
CRTSRVPGM SRVPGM(*CURLIB/UTILITY)
          MODULE(*SRVPGM)
          EXPORT(*SRCFILE)
          SRCFILE(*LIBL/QSRVSRC)
          SRCMBR(*SRVPGM)
```

要使此命令建立服务程序，源文件 QSRVSRC 中必须要有名为 UTILITY 的成员。此成员因而必须包含联编器将其转换成特征符和调出标识符集的联编器语言源。缺省值是从与服务程序 (UTILITY) 同名的成员中获取联编器语言源。若找不到带这些参数上提供的值的文件、成员或联编器语言源，则不建立服务程序。

SRCFILE 参数的文件的最大宽度

在 V3R7 或更新的发行版中，CRTSRVPGM 或 UPDSRVPGM 命令上“源文件” (SRCFILE) 参数的文件的最大宽度是 240 个字符。若文件大于最大宽度，则出现信息 CPF5D07。对于 V3R2，最大宽度是 80 个字符。对于 V3R6、V3R1 和 V2R3，对最大宽度没有限制。

调入和调出概念

ILE 语言支持下列类型的调出和调入:

- 弱数据调出
- 弱数据调入
- 强数据调出
- 强数据调入
- 强过程调出
- 弱过程调出
- 过程调入

ILE 模块对象可将过程或数据项调出至其他模块。并且 ILE 模块对象可从其他模块调入（引用）过程或数据项。当在 CRTSRVPGM 命令上使用模块对象来建立服务程序时，从服务程序可选地调出它的调出。（参见第54页的『CRTSRVPGM 命令上的调出参数』。）调出的强度（强或弱）取决于程序设计语言。强度确定了解多少关于调出的信息才可设置其特性，如数据项的大小。强调出的特性是在联编期设置的。调出的强度影响符号解析。

- 若有一个或多个弱调出同名，则联编器使用强调出的特性。
- 若弱调出与强调出的名称并不相同，则在激活期之前，不能设置其特性。在激活期，若存在多个同名的弱调出，则程序使用最大的一个。除非同名的已激活弱调出已设置其特性，否则情况均如此。

- 在联编期，若使用联编目录，且找到与弱调入相匹配的弱调出，则将联编它们。但是，仅当有要解析的未解析调入时，才使用联编目录。一旦解析了所有调入，对联编目录项的搜索就停止。不将重复的弱调出标记为重复的变量或过程。联编目录中项目的次序非常重要。

可在程序对象或服务程序外调出弱调出，以便在激活期进行解析。这与强调出相反，对于强调出，仅在服务程序外调出且仅在联编期调出。

但是，不能在程序对象外调出强调出。可在联编期在服务程序外调出强过程调出，以满足下列任一项:

- 通过引用联编服务程序的程序中的调入。
- 通过引用与该程序联编的其他服务程序中的调入。

服务程序通过联编源语言定义其公共接口。

可通过联编源语言来使弱过程调出成为服务程序的公共接口的一部分。但是，通过联编源语言从服务程序调出弱过程调出不再将其标记成弱的。将作为强过程调出来处理它。

仅可将弱数据调出至激活组。不能使其成为通过使用联编器源语言从服务程序调出的公共接口的一部分。在联编器源语言中指定弱数据将导致联编失败。

表4概述了某些 ILE 语言支持的调入和调出的类型:

表 4. ILE 语言支持的调入和调出

ILE 语言	弱数据调出	弱数据调入	强数据调出	强数据调入	强过程调出	弱过程调出	过程调入
RPG IV	否	否	是	是	是	否	是
COBOL[2]	是 [3]	是 [3]	否	否	是 [1]	否	是
CL	否	否	否	否	是 [1]	否	是
C	否	否	是	是	是	否	是
C++	否	否	是	是	是	是	是

注:

1. COBOL 和 CL 仅允许从模块调出一个过程。
2. COBOL 使用弱数据模型。作为外部来定义的数据项变成该模块的弱调出和弱调入。
3. COBOL 要求 nomonocase 选项。在没有此选项的情况下，小写字母被自动转换为大写字母。

有关哪些说明变成特定语言的调入和调出的信息，参见下列其中一本书籍:

- *Licensed Information Document: ILE RPG for AS/400, GI10-4931*
- *Licensed Information Document: ILE COBOL for AS/400, GI10-4932*
- *CL Reference (Abridged)*
- *Licensed Information Document: ILE C for AS/400, GI10-4933*

联编器语言

联编器语言是一小组不可运行的命令，它定义服务程序的调出。当指定 BND 源类型时，联编器语言启用源程序输入实用程序 (SEU) 语法检查程序来提示并验证输入。

注: 不能对包含通配符的联编器源文件使用 SEU 语法检查类型 BND。也不能对包含长于 254 个字符的名称的联编器源文件使用它。

联编器语言由下列命令的列表组成:

1. “启动程序调出”(STRPGMEXP) 命令， 它标识来自服务程序的调出列表的开始
2. “调出符号”(EXPORT) 命令，其每一个 标识可从服务程序调出的符号名
3. “结束程序调出”(ENDPGMEXP) 命令， 它标识来自服务程序的调出列表的结束

图34是源文件中联编器语言的样本:

```
STRPGMEXP PGMLVL(*CURRENT) LVLCHK(*YES)
   .
   .
EXPORT SYMBOL(p1)
EXPORT SYMBOL('p2')
EXPORT SYMBOL('P3')
   .
   .
ENDPGMEXP
   .
   .
   .

STRPGMEXP  PGMLVL(*PRV)
   .
   .
EXPORT SYMBOL(p1)
EXPORT SYMBOL('p2')
   .
   .
ENDPGMEXP
```

图 34. 源文件中联编器语言的示例

“检索联编器源”(RTVBNDSRC) 命令可用来帮助根据一个或多个模块的调出生成联编器语言源。

特征符

一对 STRPGMEXP PGMLVL(*CURRENT) 和 ENDPGMEXP 之间标识的符号，定义服务程序的公共接口。该公共接口由**特征符**表示。特征符是标识服务程序支持的接口的值。

若选择不指定显式特征符，则联编器根据要调出的过程和数据项名的列表，并根据指定它们的次序来生成特征符。因此，特征符提供了容易和方便的方法来验证服务程序的公共接口。特征符不验证服务程序内特定过程的接口。

注: 要避免对服务程序进行不兼容的更改，不可在联编器语言源中除去或重新排列现存过程和数据项名。附加的调出块包含的符号必须与现存调出块包含的符号相同，且次序也必须相同。必须将附加的符号只添加至列表的末尾。

没办法以与现存程序和服务程序兼容的方法除去服务程序调出，原因是与该服务程序联编的程序或服务程序可能需要该调出。

若对服务程序作了不兼容的更改，则仍与它联编的现存程序可能不再正确工作。仅当可保证在对服务程序进行不兼容的更改之后，用 **CRTPGM** 或 **CRTSRVPGM** 重建与其联编的所有程序和服务程序时，才可对其进行不兼容的更改。

启动程序调出和结束程序调出命令

“启动程序调出”(STRPGMEXP) 命令标识来自服务程序的调出列表的开始。“结束程序调出”(ENDPGMEXP) 命令标识来自服务程序的调出列表的结束。

源文件中指定的多对 STRPGMEXP 和 ENDPGMEXP 导致建立多个特征符。STRPGMEXP 和 ENDPGMEXP 对出现的次序并没有意义。

STRPGMEXP 命令上的程序级参数

只有一条 STRPGMEXP 命令可指定 PGMLVL(*CURRENT)，但它不必是第一条 STRPGMEXP 命令。源文件中的所有其他 STRPGMEXP 命令都必须指定 PGMLVL(*PRV)。当前特征符表示指定了 PGMLVL(*CURRENT) 的无论哪一个 STRPGMEXP 命令。若将多个 STRPGMEXP 命令标记为 *CURRENT，则假设第一个是当前的。该命令由当前特征符表示。

STRPGMEXP 命令上的级别检查参数

STRPGMEXP 命令上的级别检查 (LVLCHK) 参数指定联编器是否应自动检查服务程序的公共接口。指定 LVLCHK(*YES)，或让该值缺省为 LVLCHK(*YES)，将导致联编器检查特征符参数。特征符参数确定联编器是使用显式特征符值还是生成非零特征符值。若联编器生成特征符值，则系统验证该值是否与服务程序的客户机已知的值相匹配。若值匹配，则服务程序的客户机可使用公共接口，而不必重新编译。

指定 LVLCHK(*NO) 禁止自动特征符检查。若存在下列条件，则您可决定使用此功能:

- 您知道对服务程序的接口的某些更改不导致不兼容性。
- 您想避免更新联编器语言源文件或重新编译客户机。

小心使用 LVLCHK(*NO) 值，因为它表示您负责人工地验证公共接口是否与先前级别相兼容。仅当您可控制调用服务程序的哪些过程以及它的客户机使用哪些变量时，才指定 LVLCHK(*NO)。若不能控制公共接口，则可能会发生运行期或激活错误。参见第124页的『联编器语言错误』以获取使用联编器语言可能导致的常见错误的说明。

STRPGMEXP 命令上的特征符参数

特征符 (SIGNATURE) 参数允许明确地指定服务程序的特征符。显式特征符可以是十六进制串或字符串。您可能因下列任一原因而要考虑明确地指定特征符:

- 联编器可能会生成您不想要的兼容特征符。特征符基于指定调出的名称及其次序。因此，若两个调出块有相同次序的相同调出，则它们有相同的特征符。作为服务程序提供者，您可能知道两个接口不兼容（例如，因为它们的参数列表不同）。在此情况下，可明确地指定新特征符，而不是让联编器生成兼容特征符。若这样做，则在服务程序中建立了不兼容性，这强制某些或所有客户机重新编译。
- 联编器可能生成您不想要的不兼容特征符。若两个调出块的调出不同或次序不同，则它们的特征符不同。作为服务程序提供者，若您知道两个接口确实兼容（例如，因为函数名已更改，但它仍是同一函数），则可明确地指定与联编器先前生成的相

同的特征符，而不是让联编器生成不兼容的特征符。若指定同一特征符，则维护了服务程序中的兼容性，这允许客户机使用您的服务程序，而不必重新联编。

特征符参数的缺省值 *GEN 导致联编器根据调出的符号生成特征符。

可通过使用“显示服务程序”(DSPSRVPGM) 命令并指定 DETAIL(*SIGNATURE) 来确定服务程序的特征符值。

调出符号命令

“调出符号”(EXPORT) 命令标识可从服务程序调出的符号名。

若调出的符号包含小写字母，则应该用撇号将该符号名引起来，如第57页的图34所示。若不使用撇号，则符号名被转换为全大写字母。在该示例中，联编器搜索名为 P1 的调出，而不是 p1。

还可通过使用通配符（<<< 或 >>>）来调出符号名。若存在符号名，且与指定的通配符相匹配，则调出该符号名。若存在下列任何条件，则指示错误，且不建立服务程序:

- 没有符号名与指定的通配符相匹配
- 多个符号名与指定的通配符相匹配
- 有符号名与指定的通配符相匹配，但它不可调出

必须将通配符说明中的子字符串引在引号内。

特征符是由通配符说明中的字符确定的。即使更改过的通配符说明与同一调出相匹配，更改通配符说明也更改特征符。例如，两个通配符说明『r』>>> 和『ra』>>> 都调出符号『rate』，但它们建立两个不同的特征符。因此，强烈建议使用与调出符号尽可能相似的通配符说明。

注: 不能对包含通配符的联编器源文件使用 SEU 语法检查类型 BND。

通配符调出符号示例

对于下列示例，假设可能调出的符号列表由下列各项组成:

interest_rate

international

prime_rate

下列示例显示选择哪个调出或为何发生错误:

EXPORT SYMBOL (『interest』>>>)

调出符号『interest_rate』，因为它是唯一一个以『interest』开始的符号。

EXPORT SYMBOL (『i』>>>『rate』>>>)

调出符号『interest_rate』，因为它是唯一一个以『i』开始，并且后面含有『rate』的符号。

EXPORT SYMBOL (<<<『i』>>>『rate』)

导致『通配符说明有多个匹配』错误。『prime_rate』和『interest_rate』都包含『i』，且跟着以『rate』结束。

EXPORT SYMBOL (『inter』>>>『prime』)

导致『没有通配符说明的匹配』错误。没有任何符号以『inter』开始，并跟着以『prime』结束。

EXPORT SYMBOL (<<<)

导致『通配符说明有多个匹配』错误。此符号与全部三个符号都匹配，因而无效。一个调出语句仅可产生一个调出的符号。

联编器语言示例

作为使用联编器语言的示例，假设您正在开发具有下列过程的简单金融应用程序:

- Rate 过程

  计算 Interest_Rate，给出 Loan_Amount、Term_of_Payment 和 Payment_Amount 的值。
- Amount 过程

  计算 Loan_Amount，给出 Interest_Rate、Term_of_Payment 和 Payment_Amount 的值。
- Payment 过程

  计算 Payment_Amount，给出 Interest_Rate、Term_of_Payment 和 Loan_Amount 的值。
- Term 过程

  计算 Term_of_Payment，给出 Interest_Rate、Loan_Amount 和 Payment_Amount 的值。

第117页的『附录A. CRTPGM、CRTSRVPGM、UPDPGM 或 UPDSRVPGM 命令的输出列表』中显示了此应用程序的某些输出列表。

在联编器语言示例中，每个模块都包含多个过程。此结构更能代表 ILE C/400（与代表 ILE RPG/400 相比），但这些示例甚至适用于仅包含一个过程的模块。

联编器语言示例 1

Rate、Amount、Payment 和 Term 过程的联编器语言看起来象:

```
FILE: MYLIB/QSRVSRC  MEMBER: FINANCIAL

STRPGMEXP  PGMLVL(*CURRENT)
  EXPORT SYMBOL('Term')
  EXPORT SYMBOL('Rate')
  EXPORT SYMBOL('Amount')
  EXPORT SYMBOL('Payment')
ENDPGMEXP
```

已作了某些初始设计决策，并且三个模块（MONEY、RATES 和 CALCS）提供了必需的过程。

要建立第61页的图35中图示的服务程序，在以下 CRTSRVPGM 命令上指定联编器语言:

```
CRTSRVPGM  SRVPGM(MYLIB/FINANCIAL)
           MODULE(MYLIB/MONEY MYLIB/RATES MYLIB/CALCS)
           EXPORT(*SRCFILE)
           SRCFILE(MYLIB/QSRVSRC)
           SRCMBR(*SRVPGM)
```

注意，库 MYLIB 中的源文件 QSRVSRC（在 SRCFILE 参数中指定）是包含联编器语言源的文件。

还要注意，因为在 MODULE 参数上指定了建立服务程序所需的所有模块，所以无需联编目录。

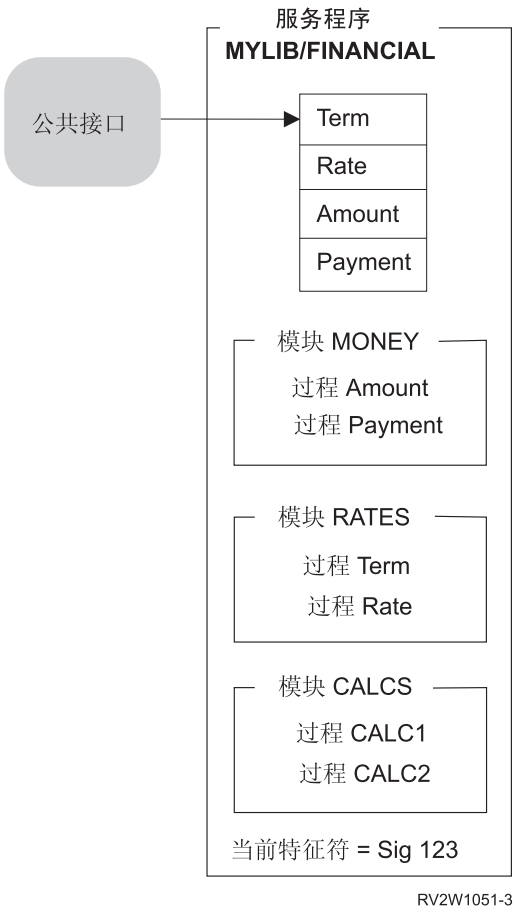

图 35. 通过使用联编器语言建立服务程序

联编器语言示例 2

随着开发应用程序过程的进行，编写了一个称为 BANKER 的程序。 BANKER 需要使用称为 FINANCIAL 的服务程序中的称为 Payment 的过程。第62页的图36中显示了带有 BANKER 程序的结果应用程序。

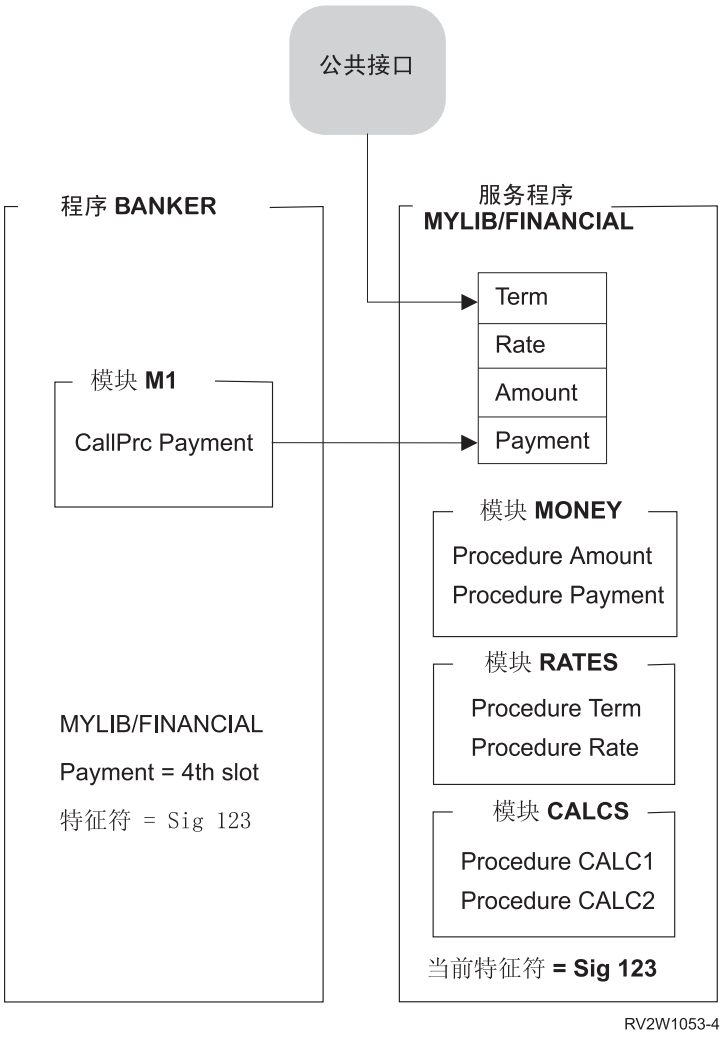

图 36. 使用服务程序 FINANCIAL

建立 BANKER 程序时，在 BNDSRVPGM 参数上提供了 MYLIB/FINANCIAL 服务程序。发现从 FINANCIAL 服务程序的公共接口的第四个槽调出符号 Payment。MYLIB/FINANCIAL 的当前特征符以及与 Payment 接口相关联的槽与 BANKER 程序保存在一起。

在使 BANKER 准备好运行的过程期间，激活验证下列各项:

- 可找到库 MYLIB 中的服务程序 FINANCIAL。
- 服务程序仍支持保存在 BANKER 中的特征符 (SIG 123)。

此特征符检查验证 BANKER 在建立时使用的公共接口在运行期是否仍有效。

如图36中所示，调用 BANKER 时， MYLIB/FINANCIAL 仍支持 BANKER 使用的公共接口。若激活在 MYLIB/FINANCIAL 或服务程序 MYLIB/FINANCIAL 中找不到任一匹配特征符，则发生下列情况:

未能激活 BANKER。

发出错误信息。

联编器语言示例 3

随着应用程序继续增大，需要两个新过程来完成我们的金融软件包。两个新过程 OpenAccount 和 CloseAccount 分别打开和关闭帐户。需要执行下列步骤来更新 MYLIB/FINANCIAL，以便无需重建程序 BANKER:

1. 编写过程 OpenAccount 和 CloseAccount。
2. 更新联编器语言以指定新过程。

更新后的联编器语言支持新过程。它还允许使用 FINANCIAL 服务程序的现存 ILE 程序或服务程序保持不变。联编器语言看起来象:

```
FILE: MYLIB/QSRVSRC  MEMBER: FINANCIAL

STRPGMEXP  PGMLVL(*CURRENT)
  EXPORT SYMBOL('Term')
  EXPORT SYMBOL('Rate')
  EXPORT SYMBOL('Amount')
  EXPORT SYMBOL('Payment')
  EXPORT SYMBOL('OpenAccount')
  EXPORT SYMBOL('CloseAccount')
ENDPGMEXP

STRPGMEXP  PGMLVL(*PRV)
  EXPORT SYMBOL('Term')
  EXPORT SYMBOL('Rate')
  EXPORT SYMBOL('Amount')
  EXPORT SYMBOL('Payment')
ENDPGMEXP
```

当需要对服务程序的更新操作执行下列两项时:

- 支持新过程或数据项
- 允许使用更改过的服务程序的现存程序和服务程序保持不变

必须选择两种替代方法之一。第一种替代方法是执行下列步骤:

1. 复制包含 PGMLVL(*CURRENT) 的 STRPGMEXP、ENDPGMEXP 块。
2. 将复制的 PGMLVL(*CURRENT) 值更改为 PGMLVL(*PRV)。
3. 在包含 PGMLVL(*CURRENT) 的 STRPGMEXP 命令中，将要调出的新过程或数据项添加至列表末尾。
4. 将更改保存至源文件。
5. 建立或重建新的或更改过的模块。
6. 通过使用更新过的联编器语言，从新的或更改过的模块建立服务程序。

第二种替代方法是利用 STRPGMEXP 命令上的特征符参数，并在调出块末尾添加新符号:

```
STRPGMEXP PGMVAL(*CURRENT) SIGNATURE('123')
  EXPORT SYMBOL('Term')
    .
    .
    .
   EXPORT SYMBOL('OpenAccount')
   EXPORT SYMBOL('CloseAccount')
ENDPGMEXP
```

要建立第64页的图37中显示的增强服务程序，在以下 CRTSRVPGM 命令上使用页 63 上指定的已更新联编器语言:

```
CRTSRVPGM  SRVPGM(MYLIB/FINANCIAL)
           MODULE(MYLIB/MONEY MYLIB/RATES MYLIB/CALCS MYLIB/ACCOUNTS))
           EXPORT(*SRCFILE)
           SRCFILE(MYLIB/QSRVSRC)
           SRCMBR(*SRVPGM)
```

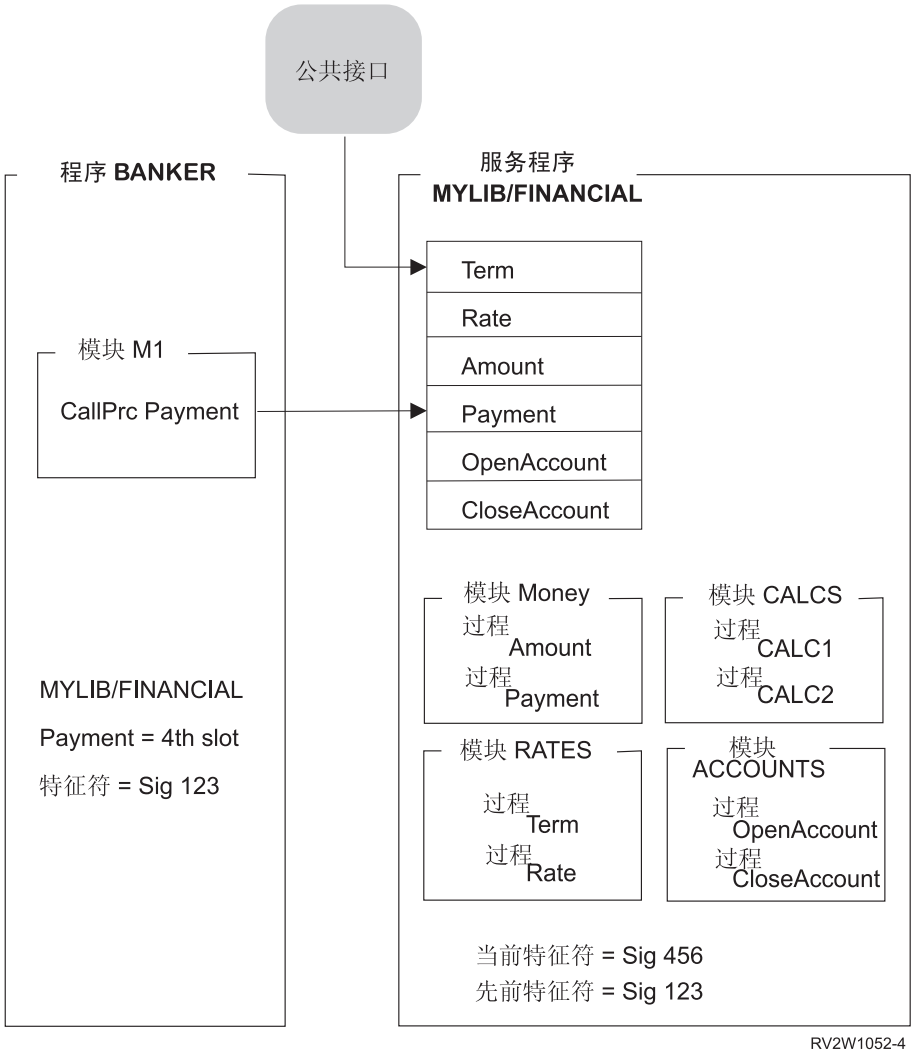

图 37. 通过使用联编器语言更新服务程序

因为仍支持先前的特征符，所以不必更改 BANKER 程序。（参见服务程序 MYLIB/FINANCIAL 中先前的特征符和 BANKER 中保存的特征符。）若由 CRTPGM 命令重建 BANKER，则与 BANKER 一起保存的特征符将是服务程序 FINANCIAL 的当前特征符。重建程序 BANKER 的唯一原因是该程序使用了服务程序 FINANCIAL 提供的其中一个新过程。联编器语言允许增强服务程序，而不必更改使用更改过的服务程序的程序或服务程序。

联编器语言示例 4

在交付更新过的 FINANCIAL 服务程序之后，您接收到根据下列各项建立利率的请求:

Rate 过程的当前参数

申请人的信贷历史

必须在对 Rate 过程的调用上添加称为 Credit_History 的第五个参数。 Credit_History 更新从 Rate 过程返回的 Interest_Rate 参数。另一需求是不必非得更改使用 FINANCIAL 服务程序的现存 ILE 程序或服务程序。若语言不支持传送可变数目的参数，则看来难于同时执行下列两项:

- 更新服务程序
- 避免重建使用 FINANCIAL 服务程序的所有其他对象

但是，幸运的是有办法做到这一点。以下联编器语言支持更新过的 Rate 过程。它仍允许使用 FINANCIAL 服务程序的现存 ILE 程序或服务程序保持不变。

```
FILE: MYLIB/QSRVSRC  MEMBER: FINANCIAL

STRPGMEXP  PGMLVL(*CURRENT)
  EXPORT SYMBOL('Term')
  EXPORT SYMBOL('Old_Rate')  /* Original Rate procedure with four parameters */
  EXPORT SYMBOL('Amount')
  EXPORT SYMBOL('Payment')
  EXPORT SYMBOL('OpenAccount')
  EXPORT SYMBOL('CloseAccount')
  EXPORT SYMBOL('Rate')           /* New Rate procedure that supports +
                                     a fifth parameter, Credit_History */
ENDPGMEXP

STRPGMEXP  PGMLVL(*PRV)
  EXPORT SYMBOL('Term')
  EXPORT SYMBOL('Rate')
  EXPORT SYMBOL('Amount')
  EXPORT SYMBOL('Payment')
  EXPORT SYMBOL('OpenAccount')
  EXPORT SYMBOL('CloseAccount')
ENDPGMEXP

STRPGMEXP  PGMLVL(*PRV)
  EXPORT SYMBOL('Term')
  EXPORT SYMBOL('Rate')
  EXPORT SYMBOL('Amount')
  EXPORT SYMBOL('Payment')
ENDPGMEXP
```

原始符号 Rate 重新命名为 Old_Rate，但仍处于要调出的符号的同一相对位置处。记住这一点很重要。

有注解与 Old_Rate 符号相关联。注解是 /* 和 */ 之间的任何内容。建立服务程序时，联编器忽略联编器语言源中的注解。

还必须调出新过程 Rate，它支持附加参数 Credit_History。将这个更新过的过程添加至调出列表的末尾。

下列两种方法可处理原始的 Rate 过程:

- 将支持四个参数的原始 Rate 过程重新命名为 Old_Rate。重复 Old_Rate 过程（将其称为 Rate）。更新代码以支持第五个参数 Credit_History。
- 更新原始 Rate 过程以支持第五个参数 Credit_History。建立称为 Old_Rate 的新过程。Old_Rate 支持 Rate 的四个原始参数。它还用伪的第五个参数调用新的更新过的 Rate 过程。

  因为维护较简单，且对象的大小较小，所以这是优选的方法。

通过使用更新过的联编器语言和新的支持过程 Rate、Term 和 Old_Rate 的 RATES 模块，建立以下 FINANCIAL 服务程序:

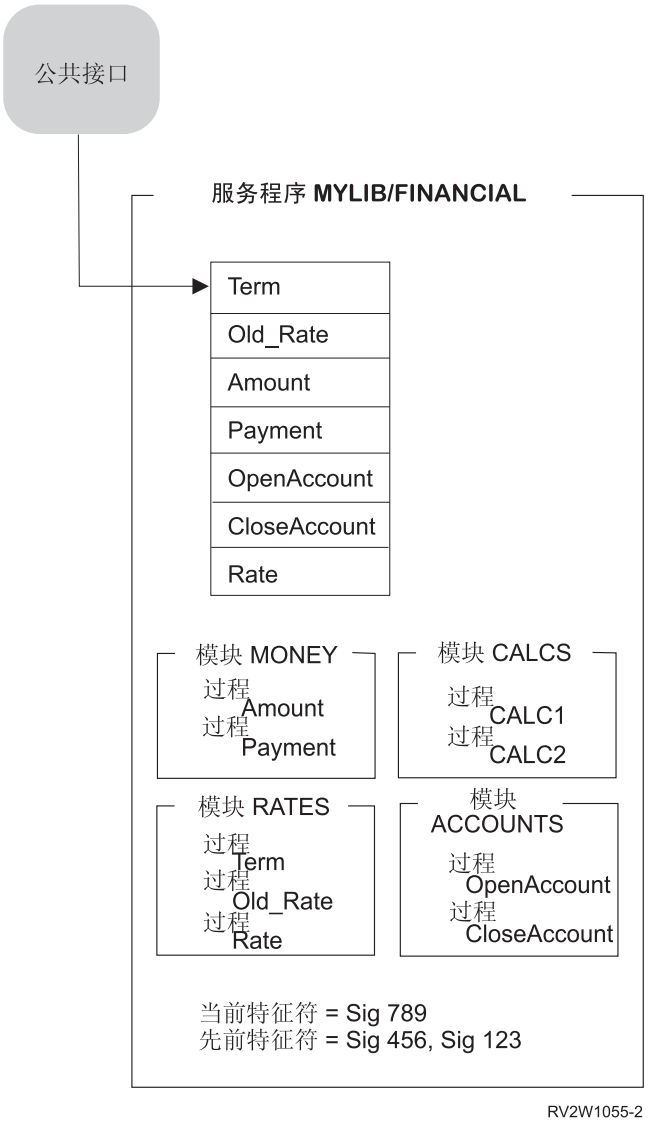

图 38. 通过使用联编器语言更新服务程序

使用 FINANCIAL 服务程序的原始 Rate 过程的 ILE 程序和服务程序转至槽 2。这将调用指向 Old_Rate 过程，因为 Old_Rate 处理原始的四个参数，所以这是有好处的。若需要重建使用原始 Rate 过程的任何 ILE 程序或服务程序，则执行下列其中一项:

- 要继续使用原始的四个参数的 Rate 过程，调用 Old_Rate 过程而不是 Rate 过程。
- 要使用新的 Rate 过程，将第五个参数 Credit_History 添加至对 Rate 过程的每一调用。

当对服务程序的更新必须满足下列需求时:

- 支持更改了它可处理的参数数目的过程
- 允许使用更改过的服务程序的现存程序和服务程序保持不变

需要执行下列步骤:

1. 复制包含 PGMLVL(*CURRENT) 的 STRPGMEXP、ENDPGMEXP 块。

2. 将复制的 PGMLVL(*CURRENT) 值更改为 PGMLVL(*PRV)。
3. 在包含 PGMLVL(*CURRENT) 的 STRPGMEXP 命令中，重新命名原始过程名，但让其仍处于同一相对位置处。

   在此示例中，将 Rate 更改为 Old_Rate，但仍处于要调出的符号列表中的同一相对位置处。
4. 在有 PGMLVL(*CURRENT) 的 STRPGMEXP 命令中，将支持不同数目参数的原始过程名放在列表的末尾。

   在此示例中，Rate 被添加至调出符号列表的末尾，但这个 Rate 过程支持附加参数 Credit_History。
5. 将更改保存至联编器语言源文件。
6. 在包含源代码的文件中，增强原始过程以支持新参数。

   在示例中，这表示将现存 Rate 过程更改为支持第五个参数 Credit_History。
7. 建立一个新的过程，它将原始参数作为输入处理，并用伪额外参数调用新过程。

   在示例中，这表示添加处理原始参数的 Old_Rate 过程，并用伪的第五个参数调用新的 Rate 过程。
8. 保存联编器语言源代码更改。
9. 用新的和更改过的过程建立模块对象。
10. 使用更新过的联编器语言来从新的和更改过的模块建立服务程序。

更改程序: “更改程序” (CHGPGM) 命令更改程序的属性，而不要求重新编译。某些可更改的属性如下:

- 优化属性。
- 用户简要表属性。
- 使用被采用权限属性。
- 性能收集属性。
- 概要数据属性。
- 程序文本。

即使指定的属性与当前属性相同，用户也可强制重新建立程序。通过指定强制程序重新建立 (FRCCRT) 参数来做到这一点。

若更改下面列示的任何参数，运行该程序的其他作业可能会失败:

- “优化程序” 提示（OPTIMIZE 参数）。
- “使用被采用权限” 提示（USEADPAUT 参数）。
- “启用性能收集” 提示（ENBPFRCOL 参数）。
- “概要数据” 提示（PRFDTA 参数）。
- “用户简要表” 提示（USRPRF 参数）。

另外，通过指定 FRCCRT(*YES) 来强制程序重新建立可能会导致运行正在更改的程序的其他作业失败。

程序更新

建立 ILE 程序对象或服务程序之后，您可能必须要校正其中的错误或对其添加增强。但是，在服务对象之后，它可能变得很大，以致于将整个对象交付给客户变得很困难或费用很高。

可通过使用“更新程序”(UPDPGM) 或“更新服务程序”(UPDSRVPGM) 命令缩小装运大小。这些命令仅置换指定的模块，且仅需将更改过的或添加的模块交付给客户。

若使用 PTF 过程，则可使用包含一个或多个对 UPDPGM 或 UPDSRVPGM 命令的调用的出口程序来执行更新功能。将同一模块与多个程序对象或服务程序联编要求对每个 *PGM 和 *SRVPGM 对象运行 UPDPGM 或 UPDSRVPGM 命令。

例如，图39

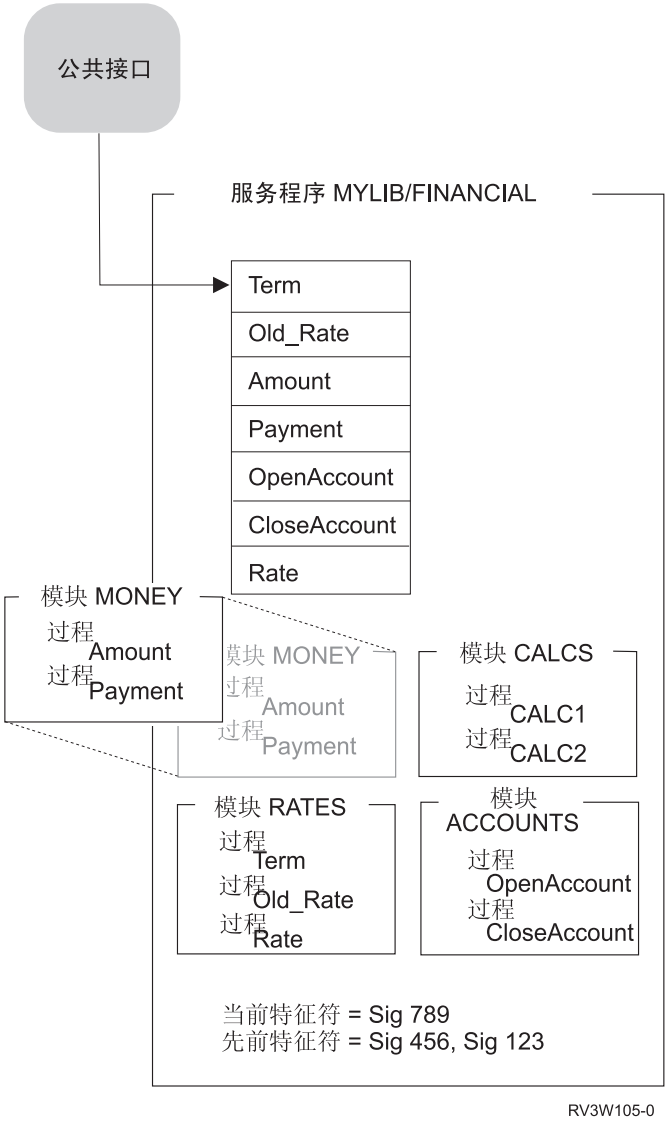

图 39. 置换服务程序中的模块

注意，不要在程序或服务程序于另一作业中仍为激活状态时更新该程序。否则，对该作业的更新保持为不活动，直到被激活回收或注销为止。

CRTPGM 或 CRTSRVPGM 命令上的允许更新 (ALWUPD) 和允许 *SRVPGM 库更新 (ALWLIBUPD) 参数确定是否可更新程序对象或服务程序。通过指定 ALWUPD(*NO)，UPDPGM 或 UPDSRVPGM 命令不能置换程序对象或服务程序中的模块。通过指定 ALWUPD(*YES) 和 ALWLIBUPD(*YES)，可更新程序以使用先前未指定的库中的服务程序。通过指定 ALWUPD(*YES) 和 ALWLIBUPD(*NO)，可更新模块，但不能更新已联编的服务程序库。不能同时指定 ALWUPD(*NO) 和 ALWLIBUPD(*YES)。

UPDPGM 和 UPDSRVPGM 命令上的参数

模块参数上指定的每个模块都置换联编到程序对象或服务程序中的同名模块。若有多个联编到程序对象或服务程序中的模块同名，则使用置换库 (RPLLIB) 参数。此参数指定用来选择要置换的模块的方法。若未将任何同名模块联编到程序对象或服务程序中，则不更新程序对象或服务程序。

已联编服务程序 (BNDSRVPGM) 参数指定在程序对象或服务程序已联编的那些服务程序之外的其他服务程序。若置换模块比起它置换的模块来说，包含较多的调入或较少的调出，则可能需要这些服务程序来解析那些调入。

借助于服务程序库 (SRVPGMLIB) 参数，可指定存储已联编服务程序的库。每次运行 UPDPGM 或 UPDSRVPGM 命令时，都使用指定库中经更新的已联编服务程序。还可在激活程序时更改库名。若使用 ALWLIBUPD(*YES)，则 UPDPGM 或 UPDSRVPGM 命令允许更改库。

联编目录 (BNDDIR) 参数指定包含解析额外调入还可能需要的模块或服务程序的联编目录。

激活组 (ACTGRP) 参数指定将 ILE 应用程序传送至 AS/400 时要使用的激活组名。此参数还允许您在激活程序或服务程序时更改激活组名。只有已命名的激活组才允许此更改。

模块被带有更少调入的模块置换

若模块被另一带有更少调入的模块置换，则总是建立新的程序对象或服务程序。但是，若存在下列情况，则更新的程序对象或服务程序包含隔离的模块:

- 因为现在丢失调入，所以其中一个联编到程序对象或服务程序中的模块不再解析任何调入
- 该模块最初来自 CRTPGM 或 CRTSRVPGM 命令上使用的联编目录

随着时间的推移，带有隔离的模块的程序可能会显著地增大。要除去不再解析任何调入的模块及最初来自联编目录的模块，可在更新对象时指定 OPTION(*TRIM)。但是，若使用此选项，则那些模块包含的调出不可用于将来的程序更新。

模块被带有更多调入的模块置换

若模块被带有更多调入的模块置换，则当已解析那些额外的调入，且下列各项为真时，便可更新程序对象或服务程序:

- 现存的模块集已联编到对象中。
- 服务程序已与对象联编。
- 命令上指定了联编目录。若这些联编目录之一中的模块包含必需的调出，则该模块被添加至程序或服务程序。若这些联编目录之一中的服务程序包含必需的调出，则该服务程序通过引用与程序或服务程序联编。
- 隐式联编目录。**隐式联编目录**是一个联编目录，它包含建立包含模块的程序可能需要的调出。每个 ILE 编译程序都将隐式联编目录的列表构建到它建立的每个模块中。

若不能解析那些额外调入，则除非在更新命令上指定了 OPTION(*UNRSLVREF)，否则更新操作失败。

模块被带有更少调出的模块置换

若模块被另一带有更少调出的模块置换，则当存在下列情况时，发生更新:

- 联编不需要丢失的调出。
- 在 UPDSRVPGM 的情况下，不从服务程序调出丢失的调出。若指定 EXPORT(*ALL)，则服务程序调出是不相同的。

若存在下列情况，则不发生更新:

- 因为丢失调出而不能解析某些调入。
- 在命令上指定的额外服务程序和联编目录中找不到那些丢失的调出。
- 联编器语言指示调出符号，但该调出丢失。

模块被带有更多调出的模块置换

若模块被另一带有更多调出的模块置换，则发生更新操作（若唯一地命名了所有额外的调出的话）。若指定 EXPORT(*ALL)，则服务程序调出是不相同的。

但是，若未唯一地命名一个或多个额外调出，则重复的名称可能会导致问题:

- 若在更新命令上指定了 OPTION(*NODUPPROC) 或 OPTION(*NODUPVAR)，则不更新程序对象或服务程序。
- 若指定了 OPTION(*DUPPROC) 或 OPTION(*DUPVAR)，则发生更新，但为联编选择的具有重复名的调出可能会不同。在 CRTPGM 或 CRTSRVPGM 命令上，若在包含已选择调出的对象前面指定了正被置换的模块，则选择已选择的调出。（若数据项是弱的，则仍可能不选择它。）

建立模块、程序和服务程序的提示

要方便地建立并维护模块、ILE 程序和服务程序，考虑下列各项:

- 遵循为了建立程序或服务程序而将要复制的模块的命名约定。

  带公共前缀的命名策略使得在模块参数上类属地指定模块更容易。
- 为了易于维护，仅在一个程序或服务程序中包括每个模块。若多个程序都需要使用某个模块，则将该模块放在服务程序中。这样，即使必须重新设计模块，也只需在一个地方重新设计它。
- 要确保您的特征符，每当建立服务程序时都使用联编器语言。

  联编器语言使更新服务程序变得容易，而不必重新建立使用的程序和服务程序。

“检索联编器源”(RTVBNDSRC) 命令可用来帮助根据来自一个或多个模块的调出生成联编器语言源。

若下列任一情况存在:

– 服务程序将永不更改

– 特征符更改时，服务程序的用户不介意更改他们的程序

您不需要使用联编器语言。因为对于大多数应用程序来说，不大可能存在此情况，所以考虑对所有服务程序使用联编器语言。

- 若别人将使用您建立的程序对象或服务程序，则在建立它时指定 OPTION(*RSLVREF)。开发应用程序时，您可能想建立带有未解析调入的程序对象或服务程序。但是，在实际生产时，应解析所有调入。

  若指定 OPTION(*WARN)，则在包含 CRTPGM 或 CRTSRVPGM 请求的作业记录中列示未解析的引用。若在 DETAIL 参数上指定列表，则它们也被包括在程序列表上。应保存作业记录或列表。

- 设计新应用程序时，确定是否可标识应进入一个或多个服务程序的公共过程。

  标识和设计新应用程序的公共过程可能是最容易的。若正在将现存应用程序转换为使用 ILE，则确定服务程序的公共过程可能难得多。尽管如此，还是尝试标识应用程序所需的公共过程并尝试建立包含公共过程的服务程序。

- 当将现存应用程序转换为 ILE 时，考虑建立一些大型程序。

  通过一些通常是很小的更改，您可容易地转换现存应用程序以利用 ILE 能力。建立模块之后，将它们组合到一些大型程序中可能是最容易和最经济的转换为 ILE 的方法。

  使用一些大型程序而不是使用许多小程序还有其他的优点，即使用较少的存储器。

- 尝试限制应用程序使用的服务程序数。

  这可能要求根据多个模块建立服务程序。优点是激活时间更短，联编过程更快。

  对于应用程序应使用多少服务程序这个问题，几乎没有什么正确的答案。若程序使用数百个服务程序，则可能使用得太多了。另一方面，只使用一个服务程序也可能并不实际。

  作为示例，为 OS/400 提供的特定于语言的和公共运行期的例行程序提供了大概 10 个服务程序。使用了超过 70 个模块来建立这 10 个服务程序。这个比例看来将性能、可理解性和可维护性平衡得很好。

第5章 激活组管理

本章包含一些示例，说明如何使用激活组来构造应用程序。主题包括:

- 支持多个应用程序
- 将“回收资源”(RCLRSC) 命令与 OPM 和 ILE 程序配合使用
- 使用“回收激活组”(RCLACTGRP) 命令删除激活组
- 服务程序和激活组

在同一作业中运行多个应用程序

用户命名的激活组允许您留下作业中的激活组供以后使用。常规返回操作或跳越操作（比如 ILE C/400 中的 longjmp()）越过控制边界而不删除激活组。

这允许应用程序保持上次被使用时的状态。在应用程序的调用之间保持不更改静态变量和已打开的文件。这可节约处理时间并且对于执行正尝试提供的功能可能是很必要的。

然而，应该准备接受来自同一作业中正在运行的多个单独的客户机的请求。系统并不限制可以联编至 ILE 服务程序的 ILE 程序的数目。因此，可能需要支持多个客户机。

图40显示可用于共享公共服务功能同时还保持用户命名的激活组的性能优点的技术。

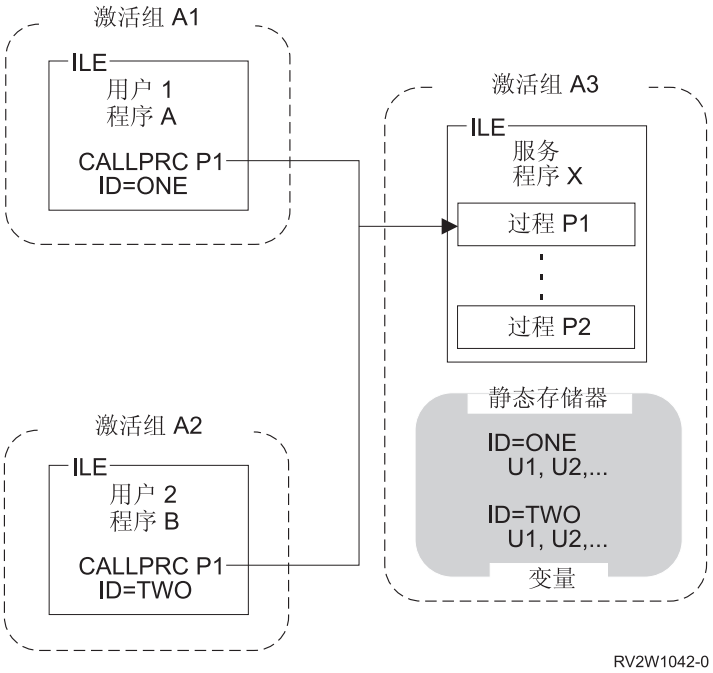

图 40. 在同一作业中运行的多个应用程序

每次调用服务程序 X 中的过程都需要用户句柄。在此示例中，字段 ID 就代表句柄。每个用户都要负责提供此句柄。由您执行初始化例行程序以返回每个用户的唯一句柄。

调用服务程序时，用户句柄用于找出与此用户相关的存储变量。保存激活组建立时间时，可以同时支持多个客户机。

回收资源命令

“回收资源”(RCLRSC) 命令取决于称为**级别号**的系统概念。 级别号是系统为作业中使用的某些资源所指定的唯一的值。下面定义了三种级别号:

调用级别号
每个调用堆栈入口都被给定唯一的级别号

程序激活级别号
每个 OPM 和 ILE 程序激活都被给定唯一的级别号

激活组级别号
每个激活组都被给定唯一的级别号

当作业运行时，系统继续为刚描述的每个新出现的资源指定唯一的级别号。 级别号是按增加值顺序指定的。建立具有较低级别号的资源之后才建立具有较高级别号的资源。

第75页的图41显示在 OPM 和 ILE 程序中使用 RCLRSC 命令的示例。已对此示例中所显示的已打开文件使用了调用级范围。使用调用级范围时，每个数据管理资源都被给定与建立该资源的调用堆栈入口相同的级别号。

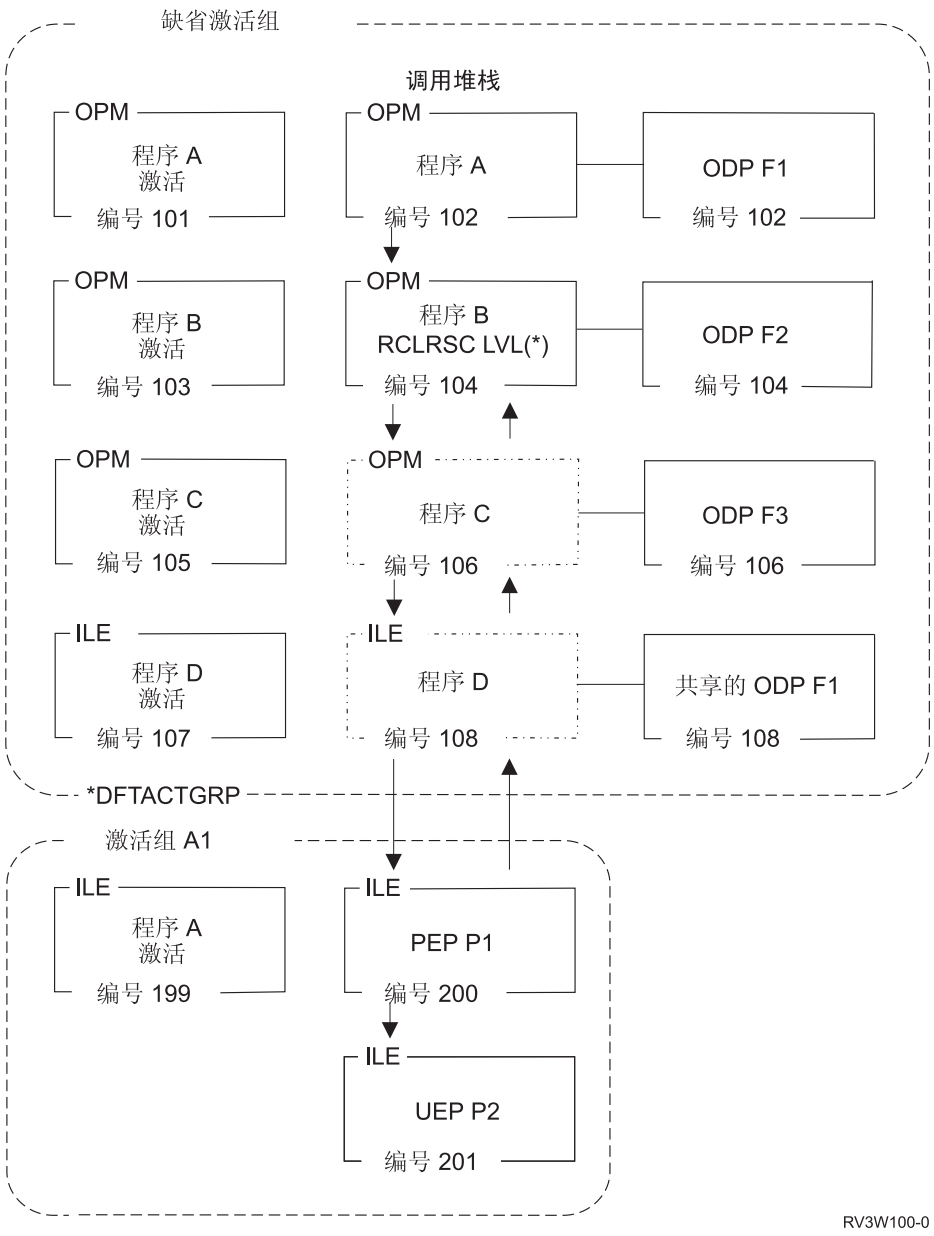

图 41. 回收资源

在此示例中，调用序列是程序 A、B、C 和 D。程序 D 和 C 返回到程序 B。程序 B 将使用带有选项 LVL(*) 的 RCLRSC 命令。 RCLRSC 命令使用级别 (LVL) 参数来清除资源。调用级别号大于当前调用堆栈入口的调用级别号的所有资源都被清除。此示例中，将调用级别号 104 用作起始点。大于调用级别号 104 的所有资源都被删除。注意，调用级别 200 和 201 中的资源不受 RCLRSC 的影响，因为它们在 ILE 激活组中。RCLRSC 仅在缺省激活组中起作用。

另外，程序 C 和 D 中的存储器以及文件 F3 的开放数据通路 (ODP) 被关闭。 文件 F1 是与程序 A 中打开的 ODP 共享的。关闭共享的 ODP，但是文件 F1 仍保持打开。

用于 OPM 程序的回收资源命令

“回收资源”(RCLRSC) 命令可用于关闭已返回但未结束的 OPM 程序的打开文件并释放其静态存储器。一些 OPM 语言（例如 RPG）允许返回而不结束程序。若稍后想关闭程序的文件并释放其存储器，可以使用 RCLRSC 命令。

用于 ILE 程序的回收资源命令

对于由指定了 DFTACTGRP(*YES) 的 CRTBNDxxx 命令所建立的 ILE 程序，RCLRSC 命令将释放静态存储器，就象它对 OPM 程序所做的那样。 对于**不是**由指定了 DFTACTGRP(*YES) 的 CRTBNDxxx 命令所建立的 ILE 程序， RCLRSC 命令将对在缺省激活组中已经建立的任何激活重新进行初始化但不释放静态存储器。使用大量静态存储器的 ILE 程序应该在 ILE 激活组中激活。删除激活组时此存储器将返回系统中。RCLRSC 命令关闭由正在缺省激活组中运行的服务程序或 ILE 程序所打开的文件。RCLRSC 命令不重新初始化服务程序的静态存储器，并且不影响非缺省的激活组。

要直接从 ILE 使用 RCLRSC 命令，可以使用 QCAPCMD API 或 ILE CL 过程。QCAPCMD API 允许直接调用系统命令而不使用 CL 程序。 在第75页的图41 中，直接调用系统命令是很重要的，因为您可能想使用特殊 ILE 过程的调用级别号。某些语言（例如 ILE C/400）还提供允许直接运行 OS/400 命令的系统功能。

回收激活组命令

“回收激活组”(RCLACTGRP) 命令可用于删除未使用的非缺省的激活组。此命令允许选项删除所有入选的激活组或者按名称来删除激活组。

服务程序和激活组

建立 ILE 服务程序时，决定是指定 *CALLER 的选项还是指定 ACTGRP 参数的名称。此选项确定是将服务程序激活到调用者的激活组中还是激活到单独命名的激活组中。两种选择各有优点和缺点。此主题讨论每个选项所提供的内容。

对于 ACTGRP(*CALLER) 选项，服务程序运作如下:

- 静态过程调用的速度很快

  当调用到服务程序中的静态过程在同一激活组中运行时，它将被优化。
- 共享的外部数据

  服务程序可以调出同一激活组中的其他程序和服务程序要使用的数据。
- 共享的数据管理资源

  打开文件和其它数据管理资源可以在服务程序和激活组中的其它程序之间共享。服务程序可能发出影响激活组中的其它程序的确认操作或回滚操作。
- 无控制边界

  服务程序内未处理的异常情况将渗滤到客户机程序中。在服务程序内使用的 HLL 结束动词可以删除客户机程序的激活组。

对于 ACTGRP(name) 选项，服务程序运作如下:

- 用于变量的单独地址空间

客户机程序不能操纵指针来对工作存储器进行寻址。若服务程序正在运行且具有被采用权限，则这可能很重要。

- 单独的数据管理资源

  您具有自己的已打开文件和确认定义。阻止偶然共享已打开的文件。

- 受控的状态信息

  您控制何时删除应用程序存储器。通过使用 HLL 结束动词或常规语言返回语句，可以决定何时删除应用程序。然而，必须管理多个客户机的状态信息。

第6章 调用过程和程序

ILE 调用堆栈和自变量传送方法降低了语言间通信的难度，使您能容易地编写混合语言应用程序。本章讨论了动态程序调用和静态过程调用的不同示例，在第20页的『对程序和过程的调用』中介绍了这些调用。介绍了第三种类型的调用，即过程指针调用。

另外，本章讨论了对 OPM 和 ILE 应用程序设计接口 (API) 的原始程序模型 (OPM) 支持。

调用堆栈

调用堆栈是**调用堆栈项**的后进先出 (LIFO) 列表，每个被调用的过程或程序都有一项。每个调用堆栈项都有关于该过程或程序的自动变量的信息，以及关于限于调用堆栈项范围的其他资源（如状态处理程序和取消处理程序）的信息。

每个作业有一个调用堆栈。调用为每个被调用的过程或程序将新项目添加到调用堆栈上，并将控制传送给被调用的对象。返回将除去堆栈项，并将控制传送回前一堆栈项中的调用过程或程序。

调用堆栈示例

第80页的图42包含调用堆栈的一个段，它有两个程序: 一个 OPM 程序（程序 A）和一个 ILE 程序（程序 B）。程序 B 包含三个过程: 其程序入口过程、其用户入口过程和另一过程 (P1)。 第12页的『模块对象』中定义了程序入口过程 (PEP) 和用户入口过程 (UEP) 的概念。调用流包括下列步骤:

1. 对“程序 A”的动态程序调用。
2. “程序 A”调用“程序 B”，将控制传送给其 PEP。这个对“程序 B”的调用是动态程序调用。
3. PEP 调用 UEP。这是静态过程调用。
4. UEP 调用过程 P1。这是静态过程调用。

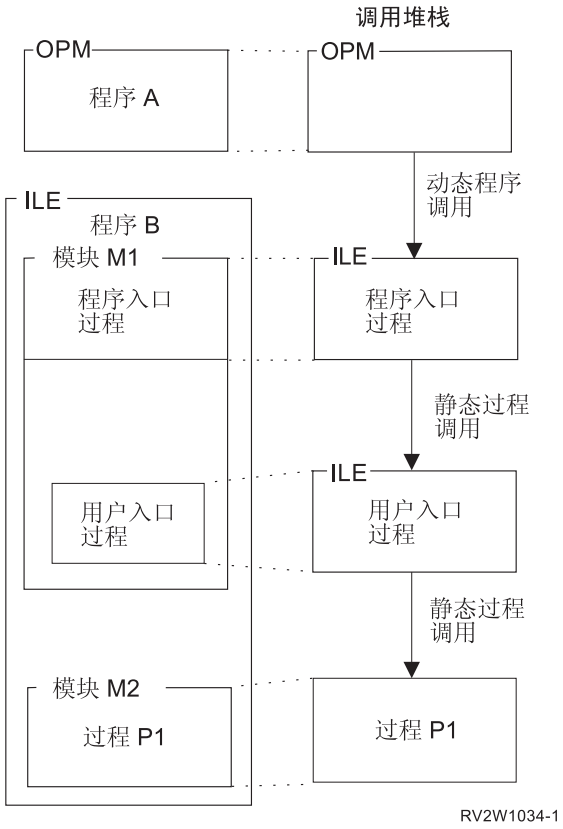

图 42. 调用堆栈上的动态程序调用和静态过程调用

图42说明了此示例的调用堆栈。堆栈上最近调用的项目描述于堆栈的底部。它是当前正在处理的项目。当前调用堆栈项可执行下列任一项:

- 调用另一过程或程序，这将另一项目添加至堆栈的底部。
- 它完成处理后，将控制返回给其调用者，这从堆栈中除去其自身。

假设过程 P1 完成后，无需在“程序 B”中进行更多的处理。过程 P1 将控制返回给 UEP，而 P1 从堆栈中被除去。然后，UEP 将控制返回给 PEP，而 UEP 从堆栈中被除去。最后，PEP 将控制返回给“程序 A”，而 PEP 从堆栈中被除去。只有“程序 A”留在此调用堆栈段上。“程序 A”从它对“程序 B”进行动态程序调用的位置继续处理。

调用程序和调用过程

ILE 运行期间，可进行三种类型的调用: 动态程序调用、静态过程调用和过程指针调用。

激活 ILE 程序时，它的除 PEP 之外的所有过程都可用于静态过程调用和过程指针调用。当动态程序调用调用程序且尚不存在激活时，发生程序激活。激活程序时，还激活与此程序联编的所有服务程序。只有静态过程调用或过程指针调用（而非动态程序调用）才可存取 ILE 服务程序中的过程。

静态过程调用

对 ILE 过程的调用将把新调用堆栈项添加至堆栈的底部，并将控制传送给指定的过程。示例包括下列任何一项:

1. 对同一模块中的过程的调用
2. 对同一 ILE 程序或服务程序的另一模块中的过程的调用
3. 对已从同一激活组中的 ILE 服务程序调出的过程的调用
4. 对已从另一激活组中的 ILE 服务程序调出的过程的调用

在示例 1、2 和 3 中，静态过程调用不跨越激活组边界。调用路径长度（它影响性能）是相等的。此调用路径比对 ILE 或 OPM 程序的动态程序调用的路径短得多。在示例 4 中，调用跨越激活组边界，并执行附加的处理来切换激活组资源。调用路径长度长于激活组内静态过程调用的路径长度，但仍短于动态程序调用的路径长度。

对于静态过程调用，在联编期间，必须将被调用过程与调用过程联编。调用总是存取同一过程。这与通过指针调用过程相反，在那种调用中，每一个调用的目标都可能不同。

过程指针调用

过程指针调用提供了动态调用过程的方法。例如，通过处理过程名或地址的数组或表，可动态地将过程调用按路径发送至不同的过程。

过程指针调用将项目添加至调用堆栈的方式与静态过程调用完全相同。可使用静态过程调用来调用的任何过程都可通过过程指针调用。若被调用过程在同一激活组中，则过程指针调用的成本几乎与静态过程调用的成本完全相同。过程指针调用可另外存取任何已激活的 ILE 程序中的过程。

将自变量传送给 ILE 过程

在 ILE 过程调用中，**自变量**是表达式，表示调用过程传送给调用中指定的过程的值。ILE 语言使用三种方法来传送自变量:

通过值，直接地
: 直接将数据对象的值放入自变量列表。

通过值，间接地
: 将数据对象的值复制至临时位置。将复制的地址（指针）放入自变量列表。

通过引用
: 将指向数据对象的指针放入自变量列表。被调用过程对自变量所作的更改在调用过程中反映出来。

第82页的图43说明了这些自变量传送的形式。并非所有的 ILE 语言都支持直接通过值传送。 ILE HLL 程序员指南中描述了可用的传送形式。

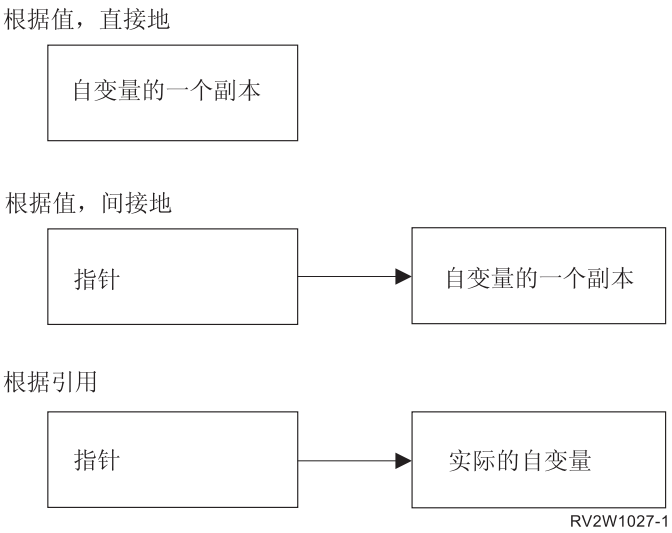

图 43. 将自变量传送给 ILE 过程的方法

HLL 语义通常确定何时通过值传送数据以及何时通过引用传送它。例如，ILE C/400 直接通过值传送和接受自变量，而对于 ILE COBOL/400 和 ILE RPG/400，通常通过引用传送自变量。必须确保调用程序或过程以被调用过程所期望的方式传送自变量。ILE HLL 程序员指南包含了更多有关将自变量传送给不同语言的信息。

静态过程调用上最多允许 400 个自变量。每种 ILE 语言可进一步限制最大自变量数。ILE 语言支持下列自变量传送形式:

- ILE C/400 直接通过值传送和接受自变量，并扩大整数和浮点值。还可通过对被调用的函数指定 #pragma 自变量伪指令来间接地通过值传送自变量。
- ILE COBOL/400 通过引用或间接地通过值传送自变量。 ILE COBOL/400 仅间接地接受参数。
- ILE RPG/400 通过引用传送和接受自变量。
- ILE CL 通过引用传送和接受自变量。

函数结果

要支持允许定义函数（返回结果自变量的过程）的 HLL，该模型假设可能存在特殊函数结果自变量，如第83页的图44中所示。如 ILE HLL 程序员指南所述，支持函数结果的 ILE 语言使用公共机制来返回函数结果。

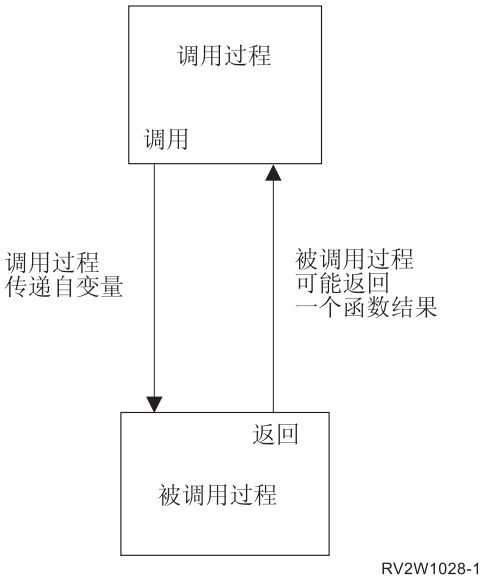

图 44. 程序调用自变量术语

省略的自变量

所有 ILE 语言都可模拟省略的自变量，这允许使用 ILE 状态处理程序和其他运行期过程的反馈码机制。例如，若 ILE C/400 过程或 ILE 可联编 API 期望通过引用传送自变量，则有时可通过在自变量的位置中传送空指针来省略该自变量。关于如何在特定 ILE 语言中指定省略的自变量的信息，参考该语言的程序员指南。 *System API Reference* 指定了对于每个 API 可省略的自变量。

对于未向被调用过程提供内部方法来测试是否省略了自变量的 ILE 语言来说，可使用“测试省略的自变量”(CEETSTA) 可联编 API。

动态程序调用

动态程序调用是对程序对象所作的调用。例如，使用 CL 命令 CALL 时，就是正在进行动态程序调用。

OPM 程序是通过使用动态程序调用来调用的。 OPM 程序还被另外限制为仅进行动态程序调用。

EPM 程序可进行程序调用和过程调用。 EPM 程序也可被其他程序和过程调用。

ILE 程序也可被动态程序调用所调用。可通过使用静态过程调用或过程指针调用来存取激活的 ILE 程序内的过程。然而也必须通过动态程序调用来调用尚未激活的 ILE 程序。

与静态过程调用（它是在编译期联编的）相反，执行调用时，动态程序调用的符号才被解析成地址。结果是，动态程序调用使用的系统资源比静态过程调用使用的要多。动态程序调用的示例包括:

- 对 ILE 程序、EPM 程序或 OPM 程序的调用
- 对不可联编 API 的调用

对 ILE 程序的动态程序调用将控制传送给已标识程序的 PEP，然后它将控制传送给该程序的 UEP。被调用程序完成处理后，控制被传送回跟在调用程序指令后的指令。

在动态程序调用上传送自变量

对 ILE 或 OPM 程序的调用（与对 ILE 过程的调用相反）通常通过引用传送自变量，这表示被调用程序接收自变量的地址。 EPM 程序可接收通过引用、直接通过值或间接通过值传送的自变量。

使用动态程序调用时，您需要知道被调用程序期望的自变量传送的方法，以及在必要时如何模拟它。动态程序调用上最多允许 255 个自变量。每种 ILE 语言可进一步限制最大自变量数。 ILE HLL 程序员指南中包含了有关如何使用不同的传送方法的信息，而 *Pascal User's Guide* (SC09-1844) 中包含了 EPM 中的传送方法的信息。

语言间数据兼容性

ILE 调用允许在用不同的 HLL 编写的过程之间传送自变量。为了降低 HLL 之间的数据共享的难度，某些 ILE 语言添加了数据类型。例如，ILE COBOL/400 添加了 USAGE PROCEDURE-POINTER 作为新数据类型。

要在 HLL 之间传送自变量，您需要知道每种 HLL 对其正在接收的自变量所期望的格式。要求调用过程确保自变量符合被调用过程所期望的大小和类型。例如，即使在参数列表中说明了短整数（2 个字节）， ILE C/400 函数也可能期望的是 4 个字节的整数。 ILE HLL 程序员指南中包含了有关如何匹配传送自变量的数据类型需求的信息。

在混合语言应用程序中传送自变量的语法

某些 ILE 语言提供了将自变量传送给其他 ILE 语言的过程的语法。例如，ILE C/400 提供了 #pragma 自变量来间接地通过值将值自变量传送给其他 ILE 过程。

操作描述符

若正在编写可从用不同的 HLL 编写的过程接收自变量的过程或 API，则操作描述符可能对您很有用。**操作描述符**向被调用过程提供了描述性信息，以防被调用过程不能精确地预见自变量的格式（例如，不同类型的字符串）。附加信息使该过程能正确地解释自变量。

自变量提供值；操作描述符提供关于自变量的大小和类型的信息。例如，此信息可能包括字符串的长度和字符串的类型。

借助于操作描述符，不要求诸如可联编 API 之类的服务对每种 HLL 有各种不同的联编，而 HLL 也不必模拟不兼容的数据类型。一些 ILE 可联编 API 使用操作描述符来适应 HLL 之间公共字符串数据类型缺乏的情况。操作描述符的存在对于 API 用户来说是透明的。

操作描述符支持 HLL 语义，而对于不使用或不期望它们的过程来说，操作描述符是不可见的。每种 ILE 语言都可使用适合于该语言的数据类型。每种 ILE 语言编译器都提供了至少一种生成操作描述符的方法。有关操作描述符的 HLL 语义的更多信息，参考 ILE HLL 参考手册。

操作描述符与您可能熟悉的其他数据描述符有区别。例如，它们与和分布式数据或文件相关联的描述符不相关。

操作描述符的需求

当用另一 ILE 语言 编写的被调用过程期望使用操作描述符时，以及当 ILE 可联编 API 期望使用操作描述符时，就应使用它们。通常，可联编 API 需要用于大多数字符串自变量的描述符。 *System API Reference* 中有关可联编 API 的信息指定了给定的可联编 API 是否需要操作描述符。

缺少必需的描述符

省略必需的描述符是错误的。若某过程需要用于特定参数的描述符，则此需求构成该过程的接口的一部分。若不提供必需的描述符，则该过程在运行期间将失败。

存在不必要的描述符

存在不是必需的描述符并不影响被调用过程存取自变量。若不需要或不期望某操作描述符，则被调用过程忽略它即可。

注: 当无论需要与否都生成描述符时，它们可能会妨碍语言间的通信。描述符增加了调用路径的长度，这可能会降低性能。

用于操作描述符存取的可联编 API

正常情况下，被调用过程根据编写该过程所用的 HLL 的语义直接存取描述符。一旦将过程设计为期望使用操作描述符，程序员通常就无需进行进一步的处理。但是，被调用过程有时在存取所需的描述符之前，需要确定它们是否存在。为此，提供了下列可联编 API:

- “检索操作描述符信息” (CEEDOD) 可联编 API
- “获取字符串信息” (CEESGI) 可联编 API

对 OPM 和 ILE API 的支持

用 ILE 开发新功能或将现存的应用程序转换为 ILE 时，您可能想继续支持 OPM 中的调用级 API。此主题说明了一种在维护用 ILE 编写的应用程序时可用来实现这种双重支持的技术。

ILE 服务程序提供了开发和传递可从所有 ILE 语言进行存取的可联编 API 的方法。要向 OPM 程序提供相同的功能，您需要考虑不能从 OPM 程序直接调用 ILE 服务程序这一事实。

要使用的技术是为计划支持的每个可联编 API 开发 ILE 程序存根模块。您可能想将可联编 API 命名为与 ILE 程序存根模块的名称相同，也可选择不同的名称。每个 ILE 程序存根模块都包含对实际可联编 API 的静态过程调用。

第86页的图45中显示了此技术的示例。

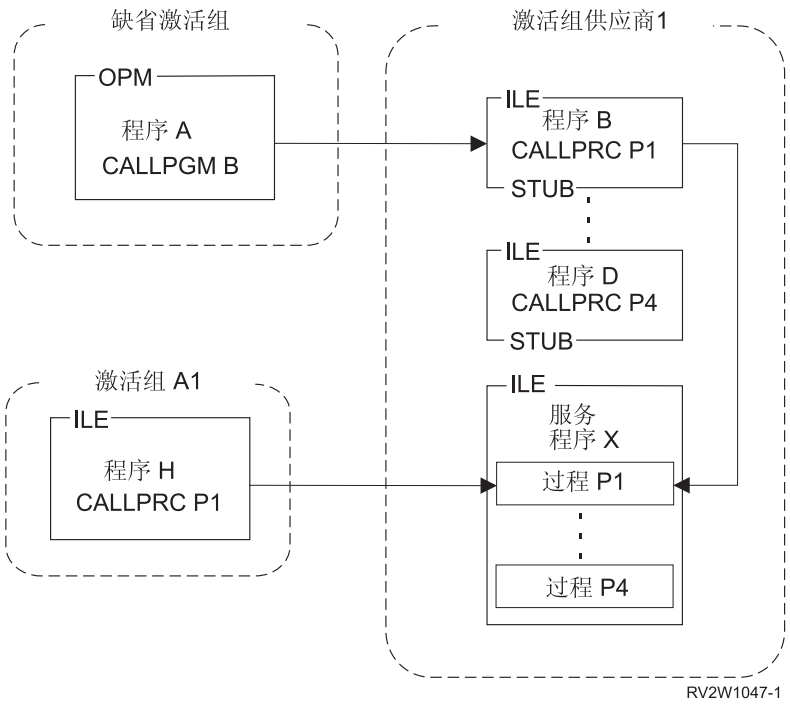

图 45. 支持 OPM 和 ILE API

程序 B 至 D 是 ILE 程序存根模块。服务程序 X 包含每一可联编 API 的实际实现。对每个程序存根模块和服务程序给出了相同的激活组名。在此示例中，选择了激活组名 VENDOR1。

必要时，系统建立激活组 VENDOR1。来自 OPM 程序 A 的动态程序调用在来自 OPM 程序的第一个调用上建立激活组。当激活 ILE 程序 H 时，来自 ILE 程序 H 的静态过程调用建立激活组。激活组一旦存在，就可在程序 A 或程序 H 中使用它。

应在 ILE 过程（在此示例中是过程 P1）中编写 API 的实现。可直接地通过过程调用或间接地通过动态程序调用来调用此过程。不应实现任何依赖于特定调用堆栈结构的功能，如发送异常信息。从程序存根模块或实现过程的正常返回将激活组留在作业中，供以后使用。您可借助于“为每一调用上的程序存根模块或实现过程建立控制边界”这一知识来实现 API 过程。 HLL 结束动词删除激活组，不管调用源于 OPM 程序还是 ILE 程序。

第7章 存储管理

操作系统提供对 ILE 高级语言的存储器支持。此存储器支持除去了每种语言的运行期环境对唯一的存储管理器的需求。它避免了不同的存储管理器与高级语言机制之间的不兼容性。

操作系统提供程序和过程在运行期内所使用的自动、静态和动态存储器。 自动存储器和静态存储器都是由操作系统管理的。也就是说，对自动存储器和静态存储器的需求是在编译期间从程序变量说明中得知的。动态存储器是由程序或过程管理的。仅在运行期才能知道对动态存储器的需求。

当激活程序时，分配并初始化用于程序变量的静态存储器。

开始运行程序或过程时，分配自动存储器。将程序或过程添加到调用堆栈时，为变量扩展自动存储堆栈。

当程序或过程运行时，在程序的控制下分配动态存储器。需要附加存储器时扩展此存储器。您能够控制动态存储器。本章的其余部分着重说明动态存储器以及控制它的方法。

动态存储器

操作系统允许使用动态地建立和废弃的多个堆栈。**堆栈**是用于分配动态存储器的一个存储区域。 应用程序所需要的动态存储器数量取决于使用该堆栈的程序及过程所处理的数据。

堆栈特性

每个堆栈具有下列特性:

- 对堆栈指定堆栈标识符，该堆栈标识符在激活组内是唯一的。

  缺省堆栈的堆栈标识符通常为零。

  由程序或过程调用的存储管理可联编 API 使用堆栈标识符来标识它将执行操作的堆栈。可联编 API 必须在拥有堆栈的激活组内运行。
- 堆栈由建立该堆栈的激活组所拥有。

  因为激活组拥有堆栈， 所以，堆栈的生命期没有拥有该堆栈的激活组的生命期长。堆栈标识符仅在拥有该堆栈的激活组内才有意义并且是唯一的。
- 动态地扩展堆栈大小以满足分配请求。

  堆栈的最大大小为 4 千兆字节减去 512K 字节。这就是最大堆栈大小，若分配的总数（在任何一个时刻）不超过 128 000 的话。
- 堆栈中的任何单个分配的最大大小被限制为 16 千兆字节减去 64K 字节。

缺省堆栈

首次请求激活组内的缺省堆栈中的动态存储器将导致缺省堆栈的建立，存储器分配就是在缺省堆栈中进行。 若堆栈中没有足够的存储器来满足对动态存储器的任何后续请求，则扩展该堆栈并分配附加存储器。

已分配的动态存储器保持分配状态，直到明确地释放它或废弃堆栈为止。仅当拥有激活组结束时才废弃该缺省堆栈。

若已从缺省堆栈中分配了存储器，则同一激活组中的程序自动地共享动态存储器。然而，可以隔离由激活组中的程序和过程所使用的动态存储器。通过建立一个或多个堆栈来进行隔离。

用户建立的堆栈

通过使用 ILE 可联编 API 可以明确地建立和废弃一个或多个堆栈。这使您能够管理堆栈以及从那些堆栈中分配的动态存储器。

例如，在用户建立的堆栈中为激活组内的程序所分配的动态存储器可能是共享的，也可能是不共享的。动态存储器的共享取决于程序引用了哪个堆栈标识符。 可以使用多个堆栈以避免自动共享动态存储器。用此方法可以隔离数据的逻辑组。下面是关于使用一个或多个用户建立的堆栈的一些其他原因:

- 可以将某些存储对象分组在一起 满足一直以来的需求。一旦满足了该需求，就可以释放由单个调用分配给 “废弃堆栈” (CEEDSHP) 可联编 API 的动态存储器。此操作释放动态存储器并废弃堆栈。这样，动态存储器可用于满足其他请求。
- 通过使用 “标记堆栈” (CEEMKHP) 和 “释放堆栈” (CEERLHP) 可联编 API， 可以立即释放多个动态存储器。CEEMKHP 可联编 API 允许您标记堆栈。当准备释放自从标记堆栈以来所分配的组时，使用 CEERLHP 可联编 API。通过使用标记和释放功能可保持堆栈完整，但是要释放已在堆栈中分配的动态存储器。这样可以避免与通过再次使用现存堆栈而建立的堆栈相关联的系统额外开销，从而满足动态存储器需求。
- 存储器需求可能与定义缺省堆栈的存储器属性不匹配。例如，缺省堆栈的初始大小为 4K 字节。然而，您需要大量的动态存储器分配，其总和超过 4K 字节。可以建立初始大小大于 4K 字节的堆栈以减少系统额外开销，否则，当隐式地扩展堆栈以及随后存取该堆栈扩展时，都将产生系统额外开销。类似地，可以使堆栈扩展大于 4K 字节。有关定义堆栈大小的信息，参见第89页的『堆栈分配策略』和对堆栈属性的讨论。

需要使用多个堆栈而不是缺省堆栈可能还有其他原因。存储管理可联编 API 使您能够管理自己建立的堆栈以及在那些堆栈中分配的动态存储器。有关存储管理可联编 API 的说明，参见 *System API Reference*。

单堆栈支持

不具有固有的多堆栈存储器支持的语言（例如 ILE C/400）使用缺省堆栈。不能将 “废弃堆栈” (CEEDSHP)、 “标记堆栈” (CEEMKHP) 或 “释放堆栈” (CEERLHP) 可联编 API 与缺省堆栈同时使用。由缺省堆栈分配的动态存储器只能通过显式释放操作来释放，或当拥有的激活组结束时释放。

这些对使用缺省堆栈帮助的限制可确保在混合语言应用程序中不会无意地释放已分配的动态存储器。对于在具有潜在的各种存储器支持的情况下再次使用现存代码的大型应用程序，释放堆栈和废弃堆栈操作被认为是不安全的。若释放堆栈操作对缺省堆栈有效，则在分开使用时能正确使用标记功能的应用程序的多个部分在一起使用时可能失败。

ILE C/400 堆栈支持

ILE C/400 对系统所提供的堆栈支持提供可选堆栈支持。

若选择使用此可选支持，则下列规则适用:

- 不能使用 CEEFRST 和 CEECZST 可联编 API 来释放或重新分配通过 C 函数 malloc()、calloc() 和 realloc() 分配的动态存储器。
- 可以用 free() 函数来释放由 CEEGTST 可联编 API 分配的动态存储器。
- 可以用 realloc() 函数来重新分配由 CEEGTST 可联编 API 初始分配的动态存储器。

若不选择此可选支持，则既可以使用存储管理可联编 API，也可以使用 malloc()、calloc()、realloc() 和 free() 函数。

其他语言（例如 COBOL）没有堆栈存储器模型。这些语言可通过可联编 API 来存取动态存储器的 ILE 动态存储器模型。

堆栈分配策略

与缺省堆栈相关联的属性是由系统通过缺省分配策略定义的。此分配策略定义属性，例如 4K 字节的堆栈建立大小和 4K 字节的扩展大小。不能更改此缺省分配策略。

然而，可以控制通过“建立堆栈”(CEECRHP) 可联编 API 而明确地建立的堆栈。 还可通过“定义堆栈分配策略”(CEE4DAS) 可联编 API 来定义显式建立的堆栈的分配策略。然后，当明确地建立堆栈时，所定义的分配策略就会提供堆栈属性。这样，就可以为一个或多个显式建立的堆栈定义单独的分配策略。

您可以在未定义分配策略的情况下使用 CEECRHP 可联编 API。在这种情况下，堆栈是由 _CEE4ALC 分配策略类型的属性来定义的。 _CEE4ALC 分配策略类型指定 4K 字节的堆栈建立大小和 4K 字节的扩展大小。 _CEE4ALC 分配策略类型包含下列属性:

```
Max_Sngl_Alloc = 16MB - 64K  /* 单个分配的最大大小 */
Min_Bdy        = 16          /* 任何分配的最小边界校准 */
Crt_Size       = 4K          /* 堆栈的初始建立大小 */
Ext_Size       = 4K          /* 堆栈的扩展大小 */
Alloc_Strat    = 0           /* 用于分配策略的一个选项 */
No_Mark        = 1           /* 组取消分配选项 */
Blk_Xfer       = 0           /* 用于堆栈的块传输的一个选项 */
PAG            = 0           /* 用于在 PAG 中建立堆栈的一个选项 */
Alloc_Init     = 0           /* 用于分配初始化的一个选项 */
Init_Value     = 0x00        /* 初始化值 */
```

此处所显示的属性用于说明 _CEE4ALC 分配策略类型的结构。有关这些属性的完整说明，参见 *System API Reference* 中对 _CEE4ALC 分配策略类型的说明。

存储管理可联编 API

为所有堆栈操作提供可联编 API。可以使用可联编 API、固有语言函数或这两者来编写应用程序。

可联编 API 属于下列类别:

- 基本堆栈操作。这些操作可以用于缺省堆栈和用户建立的堆栈。

  “释放存储器” (CEEFRST) 可联编 API 释放一个先前分配的堆栈存储器。

  “获取堆栈存储器” (CEEGTST) 可联编 API 分配堆栈内的存储器。

  “重新分配存储器” (CEECZST) 可联编 API 更改先前分配的存储器的大小。
- 扩展的堆栈操作。这些操作仅可用于用户建立的堆栈。

  “建立堆栈” (CEECRHP) 可联编 API 建立新堆栈。

  “废弃堆栈” (CEEDSHP) 可联编 API 将废弃现存的堆栈。

  “标记堆栈” (CEEMKHP) 可联编 API 返回一个令牌，可用于 标识要由 CEERLHP 可联编 API 释放的堆栈存储器。

  “释放堆栈” (CEERLHP) 可联编 API 将释放自从指定标记以来在堆栈中分配的所有存储器。
- 堆栈分配策略

  “定义堆栈分配策略” (CEE4DAS) 可联编 API 定义分配策略，它用于确定用 CEECRHP 可联编 API 建立的堆栈的属性。

有关存储管理可联编 API 的特定信息，参见 *System API Reference*。

第8章 异常和状态管理

本章提供了有关异常处理和状态处理的附加细节。在阅读本章之前，先阅读第35页的『错误处理』中描述的高级概念。

OS/400 的异常信息体系结构的作用是: 用来实现异常处理和状态处理。存在异常处理和状态处理交互作用的情况。例如，使用向注册用户编写的状态处理程序 (CEEHDLR) 可联编 API 注册的 ILE 状态处理程序来处理用发送程序信息 (QMHSNDPM) API 发送的异常信息。本章说明了这些交互作用。本章使用术语**异常处理程序**来表示 OS/400 异常处理程序或 ILE 状态处理程序。

处理光标和继续光标

要处理异常，系统使用两个称为处理光标和继续光标的指针。这些指针跟踪异常处理的过程。在某些高级错误处理方案中，您需要了解处理光标和继续光标的使用。在以后的主题中，使用这些概念来说明附加的错误处理功能部件。

处理光标是一个指针，它跟踪 当前的异常处理程序。当系统搜索可用的异常处理程序时，它将处理光标移至每个调用堆栈项定义的异常处理程序列表中的下一处理程序处。此列表可包含:

- 直接监控处理程序
- ILE 状态处理程序
- 特定于 HLL 的处理程序

处理光标顺着异常处理程序列表向较低优先级的处理程序移动，直到处理了该异常为止。若对调用堆栈项定义了的任何异常处理程序都未处理该异常，则处理光标移至前一调用堆栈项的第一个（最高优先级）处理程序。

继续光标是一个指针，它跟踪 异常处理程序在处理异常之后可继续处理的当前位置。正常情况下，系统将继续光标设置为指向跟着异常发生后面的下一指令。对于引起异常的过程上面的调用堆栈项，继续点就紧跟在当前暂停过程或程序的过程或程序调用的后面。要将继续光标移至较早的继续点，使用移动继续光标 (CEEMRCR) 可联编 API。

第92页的图46显示了处理光标和继续光标的示例。

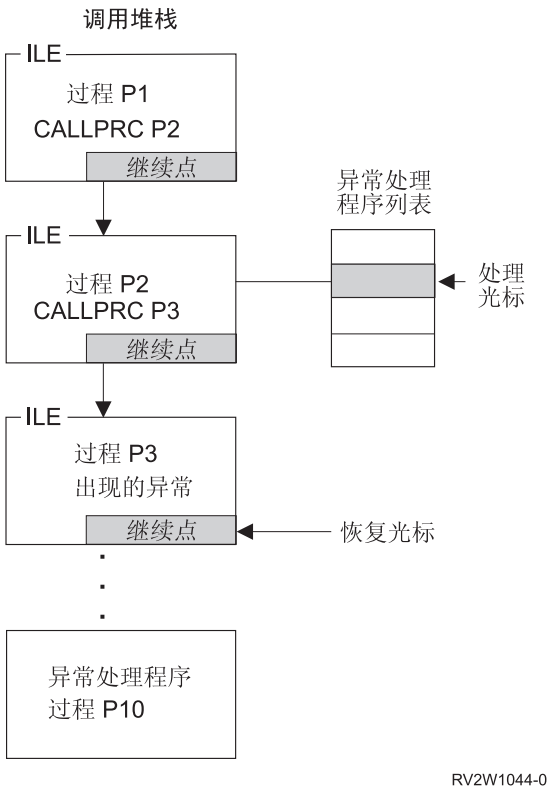

图 46. 处理光标和继续光标示例

处理光标当前在过程 P2 的异常处理程序优先级列表中定义的第二个异常处理程序处。系统当前调用处理程序过程 P10。若过程 P10 处理异常并返回，则控制转至过程 P3 中定义的当前继续光标位置。此示例假设过程 P3 将异常渗滤至过程 P2。

异常处理程序过程 P10 可用移动继续光标 (CEEMRCR) 可联编 API 修改继续光标。随此 API 一起提供了两个选项。异常处理程序可将继续光标修改为下列任一项:

- 包含处理光标的调用堆栈项
- 处理光标前面的调用堆栈项

在图46中，可将继续光标修改为过程 P2 或 P1。在修改了继续光标并将异常标记为已处理之后，从异常处理程序的正常返回将控制返回至新的继续点。

异常处理程序操作

系统调用异常处理程序时，您可采取数种操作来处理异常。例如，ILE C/400 扩展支持控制操作、分支点处理程序，以及由信息 ID 监控。此处描述的可能的操作与下列任何类型的处理程序相关:

- 直接监控处理程序
- ILE 状态处理程序
- 特定于 HLL 的处理程序

如何继续处理

若确定处理可继续，则可在当前继续光标位置处继续。在可继续处理之前，必须更改异常信息以指示它已得到处理。某些类型的处理程序要求明确地更改异常信息以指示该信息已得到处理。对于其他处理程序类型，系统可在调用处理程序之前更改异常信息。

对于直接监控处理程序， 可对异常信息指定要执行的操作。该操作可以是调用处理程序、在调用处理程序之前处理异常，或处理异常并继续程序。即使该操作仅仅是调用处理程序，也仍可通过使用“更改异常信息” (QMHCHGEM) API 或可联编 API CEE4HC（处理状态）来处理异常。可通过使用移动继续光标 (CEEMRCR) 可联编 API 来更改直接监控处理程序内的继续点。进行这些更改之后，通过从异常处理程序返回来继续处理。

对于 ILE 状态处理程序，通过设置返回码值并返回至系统来继续处理。要获取实际的返回码值，请参考 *System API Reference* 中描述的注册用户编写的状态处理程序 (CEEHDLR) 可联编 API

对于特定于 HLL 的处理程序，异常信息被更改以指示在调用处理程序之前，它已得到处理。要确定是否可从特定于 HLL 的处理程序修改继续光标，参考 ILE HLL 程序员指南。

如何渗滤信息

若确定处理程序不识别某异常信息，则可将该异常信息渗滤至下一可用的处理程序。要使渗滤发生，不可将该异常信息认为是已得到处理的信息。同一或前一调用堆栈项中的其他异常处理程序有机会处理该异常信息。渗滤异常信息的技术随异常处理程序类型的不同而有所变化。

对于直接监控处理程序，不要更改异常信息以指示它已得到处理。从异常处理程序的正常返回导致系统渗滤该信息。将把该信息渗滤至调用堆栈项的异常处理程序列表中的下一异常处理程序。若处理程序在异常处理程序列表的末尾，则该信息被渗滤至前一调用堆栈项中的第一个异常处理程序。

对于 ILE 状态处理程序，通过设置返回码值并返回至系统来传递渗滤操作。要获取实际的返回码值，请参考 *System API Reference* 中描述的可联编 API CEEHDLR

对于特定于 HLL 的处理程序，可能不能渗滤异常信息。是否可渗滤信息取决于在调用处理程序之前，HLL 是否将该信息标记为已处理。若不说明特定于 HLL 的处理程序，则 HLL 可渗滤未处理的异常信息。请参考 ILE HLL 参考手册以确定特定于 HLL 的处理程序可处理的异常信息。

如何促进信息

在某些限制的条件下，可选择将异常信息修改为另一信息。此操作将原始的异常信息标记为已处理，并用新的异常信息重新启动异常处理。仅在直接监控处理程序和 ILE 状态处理程序中才允许此操作。

对于直接监控处理程序，使用促进信息 (QMHPRMM) API 来 促进信息。仅可促进状态和脱离信息类型。借助于此 API，您可对在何处放置处理光标以继续异常处理有一定的控制。参考 *System API Reference* 以获取有关此 API 的信息。

对于 ILE 状态处理程序，可通过设置返回码值并返回至系统来传递促进操作。要获取实际的返回码值，参考 *System API Reference* 中描述的注册用户编写的状态处理程序 (CEEHDLR) 可联编 API

未处理的异常的缺省操作

若将异常信息渗滤至控制边界，则系统执行缺省操作。若该异常是通知信息，则系统发送缺省回答、处理该异常，并允许该通知信息的发送程序继续处理。若该异常是状态信息，则系统处理该异常并允许状态信息的发送程序继续处理。若该异常是脱离消息，则系统处理该脱离消息，并将功能检查信息发回至继续光标当前所在的位置。若未处理的异常是功能检查，则取消堆栈上一直到控制边界的所有项目，并将 CEE9901 脱离信息发送至前面的下一堆栈项。

表5 包含了当在控制边界处未处理异常时，系统执行的缺省响应。

表 5. 对未处理的异常的缺省响应

信息类型	状态的严重性	“指示状态”(CEESGL) 可联编 API 生成的状态	任何其他来源产生的异常
状态	0（资料式信息）	返回未处理的状态。	继续，不记录信息。
状态	1（警告）	返回未处理的状态。	继续，不记录信息。
通知	0（资料式信息）	不适用。	记录通知消息，并发送缺省回答。
通知	1（警告）	不适用。	记录通知消息，并发送缺省回答。
脱离	2（错误）	返回未处理的状态。	记录脱离消息，并将功能检查信息发送至当前继续点的调用堆栈项。
脱离	3（严重错误）	返回未处理的状态。	记录脱离消息，并将功能检查信息发送至当前继续点的调用堆栈项。
脱离	4（严重 ILE 错误）	记录脱离消息，并将功能检查信息发送至当前继续点的调用堆栈项。	记录脱离消息，并将功能检查信息发送至当前继续点的调用堆栈项。
功能检查	4（严重 ILE 错误）	不适用	结束应用程序，并将 CEE9901 信息发送至控制边界的调用者。

注: 当未处理的功能检查结束应用程序时，若该控制边界是激活组中最旧的调用堆栈项，则删除激活组。

嵌套异常

嵌套异常是在处理另一异常时发生的异常。发生此情况时，第一个异常的处理临时暂停。系统保存所有相关联的信息，如处理光标和继续光标的位置。以最近生成的异常打头，再次开始异常处理。系统设置处理光标和继续光标的新位置。一旦正确处理了新异常，原始异常的处理活动就正常地继续。

当发生嵌套异常时，下列两项都仍在调用堆栈上:

- 与原始异常相关联的调用堆栈项
- 与原始异常处理程序相关联的调用堆栈项

要减少 异常处理循环的可能性，系统停止在原始异常处理程序调用堆栈项处的嵌套异常的渗滤。然后，系统将嵌套异常促进至功能检查信息，并将该功能检查信息渗滤至同一调用堆栈项。若不处理嵌套异常或功能检查信息，则系统通过调用异常终止 (CEE4ABN) 可联编 API 来结束应用程序。在此情况下，将信息 CEE9901 发送至控制边界的调用者。

若在处理嵌套异常时移动继续光标，则可隐式地修改原始异常。要使此情况发生，执行下列各项:

1. 将继续光标移至早于导致原始异常的调用堆栈项的调用堆栈项
2. 通过从处理程序返回来继续处理

状态处理

ILE 状态是以与系统无关的方式表示的 OS/400 异常信息。 ILE 状态记号用来表示 ILE 状态。**状态处理**指的是允许您处理错误可与特定于语言的错误处理分开进行的 ILE 功能。其他 SAA 系统已实现了这些功能。可使用状态处理来增加应用程序在已实现状态处理的系统之间的可移植性。

ILE 状态处理包括下列功能:

- 动态地注册 ILE 状态处理程序的能力
- 指示 ILE 状态的能力
- 状态记号体系结构
- 可联编 ILE API 的可选状态记号反馈码

下面的主题中描述了这些功能。

如何表示状态

ILE **状态记号**是 12 个字节的复合数据类型，它包含传达状态的各个方面的结构化字段。这些方面可以是其严重性、其相关联的信息号，以及特定于状态的给定实例的信息。状态记号用来将关于状态的此信息通知给系统、信息服务、可联编 API 和过程。信息在所有 ILE 可联编 API 的可选 fc 参数中返回，例如，使用状态记号来通知。

若操作系统或硬件检测到异常，则系统自动构建相对应的状态记号。还可使用构造状态记号 (CEENCOD) 可联编 API 来建立状态记号。然后，可通过借助于指示状态 (CEESGL) 可联编 API 返回记号来向系统指示状态。

状态记号的布局

第96页的图47显示了状态记号图。显示了每个字段的起始位位置。

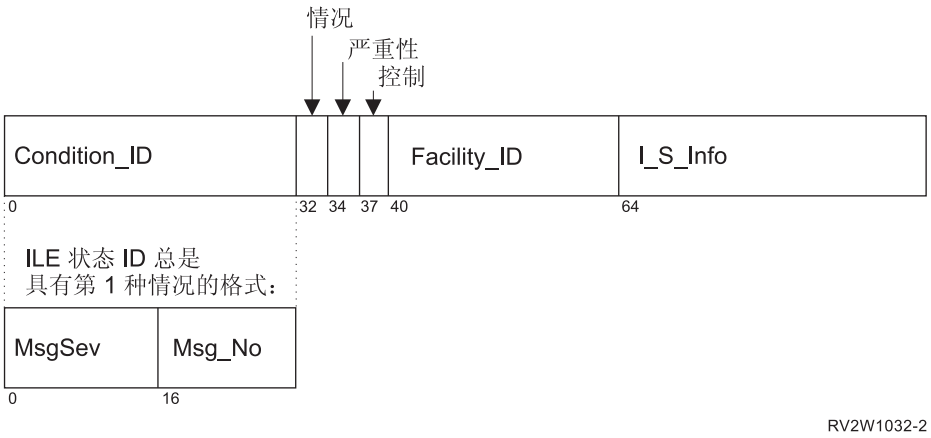

图 47. ILE 状态记号布局

每个状态记号都包含图47中指示的部件:

Condition_ID

4 个字节的标识符，与 Facility_ID 一起描述记号所传达的状态。 ILE 可联编 API 和大多数应用程序生成大小写 1 状态。

Case 2 位的字段，定义记号的 Condition_ID 部分的格式。 ILE 状态总是大小写 1。

Severity

3 位的二进制整数，指示状态的严重性。 Severity 和 MsgSev 字段包含相同的信息。参见第94页的表5以获取 ILE 状态严重性的列表。参见第97页的表7和第97页的表8以获取相对应的 OS/400 信息严重性。

Control

3 位的字段，包含描述或控制状态处理的各个方面的标志。第三位指定 IBM 是否已指定 Facility_ID。

Facility_ID

3 个字符的字母数字字符串，它标识生成状态的设施。 Facility_ID 指示该信息是由系统生成的，还是在 HLL 运行期生成的。第97页的表6列示了 ILE 中使用的设施 ID。

I_S_Info

4 个字节的字段，它标识与状态的给定实例相关联的特定于实例的信息。此字段包含与状态记号相关联的信息的实例的引用关键字。若信息引用关键字是零，则没有相关联的信息。

MsgSev

2 个字节的二进制整数，它指示状态的严重性。 MsgSev 和 Severity 包含相同的信息。参见第94页的表5以获取 ILE 状态严重性的列表。参见第97页的表7和第97页的表8以获取相对应的 OS/400 信息严重性。

Msg_No

2 个字节的二进制数，它标识与状态相关联的信息。 Facility_ID 与 Msg_No 的组合唯一地标识状态。

表6包含了在 ILE 状态记号和 OS/400 信息的前缀中使用的设施 ID。

表 6. 信息和 ILE 状态记号中使用的设施 ID

设施 ID	设施
CEE	ILE 公用库
CPF	OS/400 XPF 信息
MCH	OS/400 机器异常信息

状态记号测试

可测试从可联编 API 返回的状态记号以指示以下项目:

成功 要测试是否成功，确定前 4 个字节是否是零。若前 4 个字节是零，状态记号的剩余部分是零，则指示成功调用了可联编 API。

等价记号

要确定两个状态记号是否等价（即，状态记号的*类型*相同，但状态记号的*实例*不同），将两个状态记号的前 8 个字节作比较。对于给定状态的所有实例来说，这些字节相同。

相等记号

要确定两个状态记号是否相等，（即，它们表示状态的同一实例），则将两个状态记号的全部 12 个字节作比较。对于状态的不同实例，最后 4 个字节可以不同。

ILE 状态与 OS/400 信息的关系

对于在 ILE 中生成的每一状态，都有一信息与之相关联。状态记号包含唯一的 ID，ILE 使用它来将与该状态相关联的信息写至信息文件。

每个运行期信息的格式都是 **FFFxxxx**:

FFF 设施 ID，这是 3 个字符的 ID，在 ILE 和 ILE 语言下生成的所有信息都使用它。参考表6以获取 ID 及相对应的设施的列表。

xxxx 错误信息号。这是十六进制数，它标识与状态相关联的错误信息。

表7和表8显示了 ILE 状态严重性如何映射至 OS/400 信息严重性。

*表 7. 将 AS/400 *ESCAPE 信息严重性映射至 ILE 状态严重性*

从 AS/400 信息严重性	到 ILE 状态严重性	到 AS/400 信息严重性
0-29	2	20
30-39	3	30
40-99	4	40

*表 8. 将 AS/400 *STATUS 和 *NOTIFY 信息严重性映射至 ILE 状态严重性*

从 AS/400 信息严重性	到 ILE 状态严重性	到 AS/400 信息严重性
0	0	0
1-99	1	10

OS/400 信息和可联编 API 反馈码

作为可联编 API 的输入，您可选择编码反馈码，并在过程中将反馈码用作返回（或反馈）码检查。反馈码是状态记号值，提供它的目的是为了灵活地检查从调用向其他过

程的返回。然后，可将反馈码用作状态记号的输入。若调用可联编 API 时省略了反馈码，且出现某状态，则一条异常信息被发送至可联编 API 的调用者。

若在应用程序中编码反馈码参数来接收来自可联编 API 的反馈信息，则当生成某状态时，发生下列事件序列:

1. 将一资料式信息发送至 API 的调用者，传达与该状态相关联的信息。
2. 出现该状态的可联编 API 构建该状态的状态记号。可联编 API 将信息放入特定于实例的信息区。状态记号的特定于实例的信息是资料式信息的信息引用关键字。系统使用它来对状态作出反应。
3. 若检测到的状态是关键的（严重性是 4），则系统将异常信息发送至可联编 API 的调用者。
4. 若检测到的状态不是关键的（严重性小于 4），则状态记号被返回至调用可联编 API 的例行程序。
5. 当状态记号被返回至应用程序时，您有下列选择:
   - 忽略它并继续处理。
   - 使用指示状态 (CEESGL) 可联编 API 指示该状态。
   - 获取、格式化和调度信息以进行显示，使用 获取、格式化和调度信息 (CEEMSG) 可联编 API 。
   - 将信息存储在存储区中，使用获取信息 (CEEMGET) 可联编 API。
   - 使用调度信息 (CEEMOUT) 可联编 API 来将用户定义信息调度至指定的目的地。
   - 当 API 的调用者收回控制时，资料式信息被除去，并且不出现在作业记录中。

若调用可联编 API 时省略了反馈码参数，则可联编 API 将异常信息发送至可联编 API 的调用者。

第9章 调试考虑事项

源调试程序用于调试 OPM 程序、ILE 程序和服务程序。 CL 命令还可用于调试原始程序模型 (OPM) 程序。

本章提出有关源调试程序的几项考虑事项。有关如何使用源调试程序的信息，可以在联机信息和正使用的 ILE 高级语言 (HLL) 的程序员指南中找到。有关用于特定任务（例如建立模块）的命令的信息，可以在 ILE HLL 程序员指南中找到。

调试方式

要使用源调试程序，对话必须处于调试方式。**调试方式**是一种特殊环境，在该环境中除了可以使用常规的系统功能以外，还可使用程序调试功能。

运行“启动调试”(STRDBG) 命令时，对话被置于调试方式。

调试环境

可以在以下两种环境中调试程序:

- OPM 调试环境。除非 OPM 程序被明确添加至 ILE 调试环境，否则所有 OPM 程序都可在此环境中调试。
- ILE 调试环境。所有 ILE 程序都是在此环境中调试的。另外，若 OPM 程序满足下列所有标准，则它也可在此环境中调试:
  - 它是 CL、COBOL 或 RPG 程序。
  - 它是用 OPM 源调试数据编译的。
  - 通过设置 STRDBG 命令的 OPMSRC 参数来指示 *YES。

ILE 调试环境提供源级调试支持。调试能力直接来自于语句、源或代码的列表视图。

一旦 OPM 程序处于 ILE 调试环境中，系统就将通过同一用户界面为 ILE 和 OPM 程序提供无缝调试。有关如何在 ILE 调试环境中对 OPM 程序使用源调试程序的信息，参见关于正用于 OPM 语言的等价的 ILE 高级语言 (HLL)（CL、COBOL 或 RPG）的联机帮助和程序员指南。例如，可以参考 *ILE RPG for AS/400 Programmer's Guide* 以获取有关在 ILE 调试环境中调试 OPM RPG 程序的信息。

添加程序到调试方式

在可以调试程序之前，必须将它添加到调试方式。 OPM 程序、ILE 程序和 ILE 服务程序可同时处于调试方式。在 OPM 调试环境中，可以有多达 20 个 OPM 程序同时处于调试方式。在 ILE 调试环境中，可以同时处于调试方式的 ILE 程序、服务程序和 OPM 程序的数目是不受限制的。然而，同时受支持的最大调试数据量是每模块 16MB。

必须对程序或服务程序具有 *CHANGE 权限才能将它添加到调试方式。当程序或服务程序在调用堆栈上停止时，可以将它添加到调试方式。

源调试程序存取 ILE 程序和服务程序的方式是一次存取一个模块。当您正在调试 ILE 程序或服务程序时，可能需要调试另一个程序或服务程序中的模块。在可以调试第二个程序中的模块之前，必须将第二个程序或服务程序添加到调试方式。

调试方式结束时，将所有程序都从调试方式中除去。

可观察性和优化如何影响调试

模块是不是可观察的和完全优化的将影响调试它的能力。

模块**可观察性**是指可以与某个模块一起存储的数据，该模块允许不用再次编译该数据就可更改它。**优化**是一个过程， 系统在该过程中寻找用于减少生成相同输出所需要的系统资源数量的处理快捷方式。

可观察性

模块可观察性由两种数据类型组成:

调试数据

由 *DBGDTA 值表示。需要此数据才允许对模块进行调试。

建立数据

由 *CRTDTA 值表示。需要此数据来将代码转换为机器指令。要更改模块优化级别，模块必须具有此数据。

一旦编译了模块，就只可以除去此数据。使用“更改模块” (CHGMOD) 命令， 可以从模块中除去其中一种数据类型，或者两种类型都除去。除去所有可观察性将使模块减少到其最小大小（采用压缩）。一旦除去了此数据，就不能用任何方式来更改该模块，除非再次编译该模块并置换该数据。要再次编译它，必须具有对源代码的权限。

优化级别

通常，若模块中含有建立数据，则可以更改优化源代码所在的级别以便在系统中运行。处理快捷方式被转换成机器代码，从而使模块中的过程更有效地运行。优化级别越高，过程在模块中运行的效率也更高。

然而，越优化就越不能更改变量，并且在调试期间可能无法察看变量的实际值。当正在调试时，将优化级别设置为 10 (*NONE)。这为模块中的过程提供最低级别的性能，但是允许您准确地显示和更改变量。完成调试之后，将优化级别设置为 30 (*FULL) 或 40。这为模块中的过程提供最高级别的性能。

调试数据建立和除去

调试数据是与每个模块存储在一起的，并且是在建立模块时生成的。要调试已经建立但是没有调试数据的模块中的过程，必须重新建立模块及调试数据，然后将该模块重新联编至 ILE 程序或服务程序。不需要重新编译具有调试数据的程序或服务程序中的其他所有模块。

要除去模块中的调试数据，重新建立该模块而不调试数据或使用“更改模块” (CHGMOD)命令。

模块视图

可用的调试数据的级别可能随 ILE 程序或服务程序中的每个模块而变化。 模块是单独编译的，并且可能是用不同的编译器和选项来生成的。这些调试数据级别确定哪些**视图**是由编译器生成的，以及哪些视图是由源编译器显示的。其可能值为:

***NONE**
没有生成调试视图。

***STMT**
没有源是由调试程序显示的，但是，可以使用在编译器列表中找到的过程名和语句号来添加断点。与此视图存储在一起的调试数据数量是进行调试所需要的数据的最小数量。

***SOURCE**
若用于编译模块的源文件仍存在于系统中，则源调试程序将显示源。

***LIST** 列表视图是由模块生成和存储的。这允许源调试程序显示源，即使用于建立该模块的源文件不在系统中。若将要更改程序，此视图作为备份副本是很有用的。然而，调试数据的数量可能相当大，特别是如果其他文件也扩展到了此列表中的话，其数量会更大。建立模块时所使用的编译器选项确定所包括的文件是否是扩展的。可以扩展的文件包括 DDS 文件和包括文件（例如 ILE C/400 包括文件、ILE RPG/400 /COPY 文件和 ILE COBOL/400 COPY 文件）。

***ALL** 生成了所有调试视图。对于列表视图，调试数据的数量可能相当大。

ILE RPG/400 还有一个调试选项 *COPY，它既生成源视图又生成副本视图。副本视图是包括所有 /COPY 源成员的调试视图。

跨作业调试

您可能想使用单独的作业来调试正在作业中或批处理作业中运行的程序。 这对于想在不受调试程序屏面干扰的情况下观察程序的功能是相当有用的。例如，在步进期间或在断点处，应用程序所显示的屏面或窗口可能会覆盖调试程序屏面，或者被调试程序屏面覆盖。通过启动服务作业并在与正在调试的作业不同的作业中启动调试程序，可避免此问题。有关这方面的信息，参见 *CL Programming* 一书中关于测试的附录。

OPM 和 ILE 调试程序支持

OPM 和 ILE 调试程序支持通过 “ILE 调试程序 API” 来启用 OPM 程序的源级调试。有关 “ILE 调试程序 API” 的信息，参见 *System API Reference*。 OPM 和 ILE 调试程序支持通过同一用户界面为 ILE 和 OPM 程序提供无缝调试。要使用此支持，必须用 AS/400 RPG、COBOL 或 CL 编译器编译 OPM 程序。必须将 OPTION 参数设置为 *SRCDBG 或 *LSTDBG 以进行编译。不支持 System/36 编译器和 EPM 编译器。

监视支持

当指定的存储位置的内容被更改时， “监视” 支持提供使程序停止执行的能力。存储位置是由程序变量的名称指定的。程序变量被解析为存储位置，并监视此位置中的内容是否发生了更改。若该存储位置中的内容发生了更改，则停止执行。在中断点显示中断的程序源，并且突出显示的源行将在更改该存储位置的语句之后运行。

未监控的异常

当发生未监控的异常时，正在运行的程序发出功能检查并将一条信息发送至作业记录。若正处于调试方式并且程序的模块是用调试数据建立的，则源调试程序将显示“显示模块源”屏幕。若需要的话，将程序添加到调试方式。在该屏幕上显示适当的模块，并且突出显示受影响的行。然后可以调试该程序。

调试时对国家语言支持的限制

若下列条件之一存在:

- 调试作业的编码字符集标识符 (CCSID) 为 290、930 或 5026（日语片假名）
- 用于调试的设备说明的代码页为 290、930 或 5026（日语片假名）

调试命令、函数和十六进制字母应该按大写字母输入。例如:

```
BREAK 16 WHEN var=X'A1B2'
EVAL var:X
```

当在调试命令（例如 EVAL）中使用标识符名时，对“日语片假名”代码页的上述限制不适用。然而，调试 ILE RPG/400、ILE COBOL/400 或 ILE CL 模块时，调试命令中的标识符名被源调试程序转换为大写字母，因此，再显示时可能会有不同。

第10章 数据管理范围限定

本章包含有关 ILE 程序或服务程序可能使用的数据管理资源的信息。在阅读本章之前，应该了解在第41页的『数据管理范围限定规则』中所描述的数据管理范围限定概念。

有关每个资源类型的详细资料,可参阅各自的 ILE HLL 程序员指南。

公共数据管理资源

本主题标识遵循数据管理范围限定规则的所有数据管理资源。跟在每个资源后的是关于如何指定范围限定的简短说明。可在参考出版物中找到每个资源的其他细节。

- 打开文件操作

  打开文件操作导致建立一个临时的 调用开放数据通路 (ODP) 的资源。可以通过使用 HLL 打开动词、 “打开查询文件” (OPNQRYF) 命令或 “打开数据库文件” (OPNDBF) 命令来启动打开功能。将 ODP 的范围限于打开该文件的程序的激活组。对于在缺省激活组中运行的 OPM 或 ILE 程序，将 ODP 的范围限于调用级号。要更改 HLL 打开动词的的范围限定，可使用覆盖。通过在所有覆盖命令、 OPNDBF 命令和 OPNQRYF 命令上使用打开范围 (OPNSCOPE) 参数，可以指定范围限定。关于打开文件操作的更多信息，参阅 *Data Management* 一书。

- 覆盖

  将覆盖的范围限于调用级别、激活组级别或作业级别。要指定覆盖范围限定，在任何覆盖命令上使用覆盖范围 (OVRSCOPE) 参数。若未指定明显的范围限定，则覆盖的范围取决于覆盖是从哪里发出的。若覆盖是从缺省激活组发出的，则将其范围限于调用级。若覆盖是从任何其他激活组发出的，则将其范围限于激活组级。关于覆盖的更多信息，参阅 *Data Management* 一书。

- 确认定义

  确认定义支持范围限于激活组级 和范围限于作业级。通过在 “启动确认控制” (STRCMTCTL) 命令上使用控制范围 (CTLSCOPE) 参数来指定范围限定级别。关于确认定义的更多信息，参见 *Backup and Recovery*。

- 局部 SQL 游标

  可为 ILE 编译器产品建立 SQL 程序。由 ILE 程序使用的 SQL 游标的范围可被限于模块或激活组。可通过在 “建立 SQL 程序” 命令上使用结束 SQL (ENDSQL) 参数来指定 SQL 游标范围限定。关于局部 SQL 游标的更多信息，参阅 *DB2 for AS/400 SQL Programming* 一书。

- 远程 SQL 连接

  随 SQL 游标一起使用的远程连接作为正常 SQL 处理的一部分被隐式地限于激活组。这允许多个会话存在于一个源作业和多个目标作业或系统中。关于远程 SQL 连接的更多信息，参阅 *DB2 for AS/400 SQL Programming* 一书。

- 用户界面管理器

  “开放打印应用程序” (QUIOPNPA) 和 “开放显示应用程序” API 支持应用程序范围参数。可使用这些 API 来将用户界面管理器 (UIM) 应用程序的范围限于激活组或作业。关于用户界面管理器的更多信息，参见 *System API Reference*。

- 开放数据链路（开放文件管理）

  “启用链路”(QOLELINK) API 启用一个数据链路。若从 ILE 激活组中使用此 API，则该数据链路的范围限于该激活组。若从缺省激活组中使用此 API，则该数据链路的范围限于调用级。关于开放数据链路的更多信息，参见 *System API Reference*。

- 公共程序设计接口 (CPI) 通信会话

  启动会话的激活组拥有该会话。通过“启用链路”(QOLELINK) API 启用链路的激活组拥有该链路。关于“公共程序设计接口 (CPI) 通信”会话的更多信息，参见 *System API Reference*。

- 分层的文件系统

  “开放流式文件”(OHFOPNSF) API 管理分层的文件系统 (HFS) 文件。可使用此 API 上的打开信息 (OPENINFO) 参数来控制将范围限于激活组或作业级。关于分层的文件系统的更多信息，参见 *System API Reference*。

确认控制范围限定

ILE 对确认控制引入两个更改:

- 每个作业有多个独立的确认定义。事务可彼此独立地确认和回滚。在 ILE 之前，每个作业只允许一个单个的确认定义。
- 当激活组正常结束时若更改暂挂，则系统隐式确认这些更改。在 ILE 之前，系统不确认这些更改。

确认控制允许您将 对资源（如数据库文件或表）的更改作为一个单个的事务来定义和处理。**事务**是对系统上的对象的一组个别的更改，在用户看来可能是单个不可分的更改。确认控制确保下列其中之一发生在系统上:

- 整组个别更改发生（**确认**操作）
- 没有个别更改发生（**回滚**操作）

在确认控制下，可以使用 OPM 程序和 ILE 程序更改各种资源。

“启动确认控制”(STRCMTCTL) 命令使在作业内运行的程序在确认控制下进行更改成为可能。当使用 STRCMTCTL 命令启动确认控制时，系统建立**确认定义**。每个确认定义只为发出 STRCMTCTL 命令的那个作业所知。确认定义包含关于在该作业内的确认控制下被更改的资源的信息。当确认资源更改时，系统将维护确认定义中的确认控制信息。通过使用“结束确认控制”(ENDCMTCTL) 命令结束确认定义。关于确认控制的更多信息，参见 *Backup and Recovery*。

确认定义和激活组

在一个作业内运行的程序可以启动和使用多个确认定义。作业的每个确认定义都标识一个有相关资源的单独的事务。可以确认或回滚这些资源，而与为该作业启动的所有其他确认定义无关。

注: 只有 ILE 程序能够对不是缺省激活组的激活组启动确认控制。因此，仅当作业正运行一个或多个 ILE 程序时，它才能使用多个确认定义。

原始程序模型 (OPM) 程序在缺省激活组中运行。缺省情况下，OPM 程序使用 *DFTACTGRP 确认定义。对 OPM 程序，可通过在 STRCMTCTL 命令上指定 CMTSCOPE(*JOB) 来使用 *JOB 确认定义。

当使用“启动确认控制”(STRCMTCTL) 命令时，在确认范围 (CMTSCOPE) 参数上指定确认定义的范围。确认定义的**范围**指示在作业内运行的哪些程序使用该确认定义。确认定义的缺省范围限于发出 STRCMTCTL 命令的程序的激活组。只有在该激活组内运行的程序才将使用该确认定义。 限于激活组的确认定义被称为**激活组级**的确认定义。在 OPM 缺省激活组的激活组级启动的确认定义被称为缺省激活组 (*DFTACTGRP) 确认定义。在一个作业的各个激活组内运行的程序可以启动和使用许多激活组级的确认定义。

确认定义的范围也可限于该作业。带有此范围值的确认定义被称为 **作业级**或 *JOB 确认定义。在激活组中运行的、没有在激活组级启动的确认定义的任何程序，使用作业级确认定义。若该作业的另一个程序已经启动了作业级确认定义，则会发生这种情况。只能为一个作业启动一个作业级确认定义。

对于给定的激活组，在该激活组内运行的程序只可以使用单个确认定义。在激活组内运行的程序既可以使用作业级确认定义，也可以使用激活组级确认定义。然而，这些程序不可以同时使用这两种确认定义。

当程序执行确认控制操作时，它不直接指出要对该请求使用哪种确认定义。而是由系统根据该请求程序运行在哪个激活组中来确定要使用哪个确认定义。这是可能的，因为在任何时间点上，在一个激活组内运行的程序只能使用单个确认定义。

结束确认控制

通过使用“结束确认控制”(ENDCMTCTL) 命令，可以对作业级或激活组级确认定义结束确认控制。 ENDCMTCTL 命令对系统指示要结束发出该请求的程序的激活组的确认定义。 ENDCMTCTL 结束该作业的一个确认定义。该作业的所有其他确认定义保持不变。

若结束了激活组级确认定义，则在该激活组内运行的程序将不可以再在确认控制下执行更改。若作业级确认定义已启动或仍存在，则任何指定确认控制的新文件打开操作都使用作业级确认定义。

若作业级确认控制结束，则在该作业内运行的任何使用作业级确认定义的程序将不可以再在确认控制下执行更改。若用 STRCMTCTL 命令再次启动确认控制，则可执行更改。

激活组结束期间的确认控制

当下列条件同时存在时:

- 激活组结束
- 作业未结束

系统自动结束激活组级的确认定义。若下列两个条件都存在:

- 存在激活组级的确认定义的未确认更改
- 激活组正常结束

则在结束确认定义之前，系统对确认定义执行一个隐式确认操作。另外，若存在下列两个条件之一:

- 激活组异常结束
- 当关闭在限于激活组的确认控制下打开的任何文件时，系统遇到错误

则在结束激活组级的确认定义之前，对它执行隐式回滚操作。因为激活组异常结束，所以系统用上一次成功的确认操作更新通知对象。确认和回滚取决于暂挂更改。若无暂挂更改，则没有回滚，但仍更新通知对象。若激活组异常结束并有暂挂更改，则系统隐式回滚这些更改。若激活组正常结束并有暂挂更改，则系统隐式确认这些更改。

在对 *JOB 或 *DFTACTGRP 确认定义的激活组结束处理期间， 从不执行隐式确认操作或回滚操作。这是因为 *JOB 和 *DFTACTGRP 确认定义从不因激活组的结束而结束。相反，这些确认定义或者是由 ENDCMTCTL 命令显式结束，或者是当作业结束时由系统结束。

当激活组结束时，系统自动关闭限于该激活组的任何文件。这包括限于该激活组的在确认控制下打开的任何数据库文件。在对激活组级的确认定义执行的任何隐式确认操作之前，发生对任何这样的文件的关闭操作。因此，在执行任何隐式确认操作之前，驻留在 I/O 缓冲区中的任何记录首先被强制放入数据库。

作为隐式确认操作或回滚操作的一部分，系统对每个 API 确认资源调用 API 确认和回滚出口程序。每个 API 确认资源必须与激活组级的确认定义相关。在调用 API 确认和回滚出口程序之后，系统自动除去 API 确认资源。

若存在下列条件:

- 对因激活组的结束而结束的确认定义执行了隐式回滚操作
- 为确认定义定义了通知对象

则更新通知对象。

第11章 ILE 可联编应用程序设计接口

ILE 可联编应用程序设计接口（可联编 API）是 ILE 的一个重要部分。 在某些情况下，它们提供由特定的高级语言提供的功能之外的附加功能。例如，不是所有的 HLL 都提供操纵动态存储器的固有方法。在那些情况下，您可以通过使用特定的可联编 API 来增补 HLL 功能。若 HLL 提供了与特定的可联编 API 相同的功能，则使用特定于 HLL 的那一个。

可联编的 API 与 HLL 无关。这对于混合语言应用程序来说是有用的。例如，若只将状态管理可联编 API 与混合语言应用程序一起使用，则您对该应用程序将具有统一的状态处理语义。这使得状态管理比在使用多个特定于 HLL 的状态处理程序时更加一致。

可联编的 API 提供了许多种功能，包括:

激活组和控制流管理

状态管理

日期和时间处理

动态屏幕管理

数学函数

信息处理

程序或过程调用管理和操作描述符存取

源调试程序

存储管理

有关 ILE 可联编 API 的详情，参见 *System API Reference*。

可用的 ILE 可联编 API

大多数可联编 API 可用于 ILE 支持的任何 HLL。可联编 API 的命名约定如下:

- 名称以 CEE 开头的可联编 API 基于 “SAA 语言环境*” 规范。这些 API 在 IBM SAA 系统中应为一致的。有关 “SAA 语言环境” 的详情，参见 *SAA CPI Language Environment Reference*。
- 名称以 CEE4 或 CEES4 开头的可联编 API 是特定于 AS/400 系统的。

ILE 提供下列可联编的 API:

激活组和控制流可联编 API

异常结束 (CEE4ABN)

找到控制边界 (CEE4FCB)

注册激活组出口过程 (CEE4RAGE)

注册调用堆栈入口终止用户出口过程 (CEERTX)

指示紧急终止状态 (CEETREC)

未注册调用堆栈入口终止用户出口过程 (CEEUTX)

状态管理可联编 API

构造状态标记 (CEENCOD)

分解状态标记 (CEEDCOD)

处理状态 (CEE4HC)

将“继续光标”移动至“返回点”(CEEMRCR)

注册用户编写的状态处理程序 (CEEHDLR)

检索 ILE 版本和平台 ID (CEEGPID)

返回相对调用号 (CEE4RIN)

指示状态 (CEESGL)

未注册用户状态处理程序 (CEEHDLU)

日期和时间可联编 API

根据 Lilian 日期来计算星期中某日 (CEEDYWK)

将日期转换为 Lilian 格式 (CEEDAYS)

将整数转换为秒 (CEEISEC)

将 Lilian 日期转换为字符格式 (CEEDATE)

将秒转换为字符时间戳记 (CEEDATM)

将秒转换为整数 (CEESECI)

将时间戳记转换为秒数 (CEESECS)

获得当前格林威治标准时间 (CEEGMT)

获得当前本地时间 (CEELOCT)

获得标准世界时间与本地时间的偏差量 (CEEUTCO)

获得标准世界时间 (CEEUTC)

查询世纪 (CEEQCEN)

返回国别的缺省日期和时间字符串 (CEEFMDT)

返回国别的缺省日期字符串 (CEEFMDA)

返回国别的缺省时间字符串 (CEEFMTM)

设置国别 (CEESCEN)

数学可联编 API

每个数学可联编 API 的名称中的 x 指的是下列数据类型之一:

I 32 位二进制整数

S 32 位单精度浮点数

D 64 位双精度浮点数

T 32 位单精度浮点复数（实部和虚部均为 32 位长）

E 64 位双精度浮点复数（实部和虚部均为 64 位长）

绝对函数 (CEESxABS)

反余弦 (CEESxACS)

反正弦 (CEESxASN)

反正切 (CEESxATN)

反余切 (CEESxAT2)

共轭复数 (CEESxCJG)

余弦 (CEESxCOS)

余切 (CEESxCTN)

错误函数及其补码 (CEESxERx)

以 e 为底的指数 (CEESxEXP)

取幂 (CEESxXPx)

阶乘 (CEE4SIFAC)

浮点复数除法 (CEESxDVD)

浮点复数乘法 (CEESxMLT)

伽玛函数 (CEESxGMA)

双曲反正切 (CEESxATH)

双曲余弦 (CEESxCSH)

双曲正弦 (CEESxSNH)

双曲正切 (CEESxTNH)

复数的虚部 (CEESxIMG)

对数伽玛函数 (CEESxLGM)

以 10 为底的对数 (CEESxLG1)

以 2 为底的对数 (CEESxLG2)

以 e 为底的对数 (CEESxLOG)

模块化算术 (CEESxMOD)

最近整数 (CEESxNIN)

最近完整数 (CEESxNWN)

正差分 (CEESxDIM)

正弦 (CEESxSIN)

平方根 (CEESxSQT)

正切 (CEESxTAN)

符号的转换 (CEESxSGN)

截断 (CEESxINT)

其他数学可联编 API:

基本随机数生成 (CEERAN0)

信息处理可联编 API

分配信息 (CEEMOUT)

获取信息 (CEEMGET)

获取、格式化和分配信息 (CEEMSG)

程序或过程调用可联编 API

获取字符串信息 (CEEGSI)

检索操作描述符信息 (CEEDOD)

测试省略的自变量 (CEETSTA)

源调试程序可联编 API

允许程序发出调试语句 (QteSubmitDebugCommand)

启用对话以使用源调试程序 (QteStartSourceDebug)

从一个视图向另一个视图映射位置 (QteMapViewPosition)

注册一个模块的视图 (QteRegisterDebugView)

除去一个模块的视图 (QteRemoveDebugView)

检索源调试对话的属性 (QteRetrieveDebugAttribute)

对程序检索模块和视图列表 (QteRetrieveModuleViews)

检索程序停止的位置 (QteRetrieveStoppedPosition)

从指定的视图中检索源文本 (QteRetrieveViewText)

设置源调试对话的属性 (QteSetDebugAttribute)

采用调试方式以外的作业 (QteEndSourceDebug)

存储管理可联编 API

建立堆栈 (CEECRHP)

定义堆栈分配策略 (CEE4DAS)

废弃堆栈 (CEEDSHP)

释放存储器 (CEEFRST)

获取堆栈存储器 (CEEGTST)

标记堆栈 (CEEMKHP)

重新分配存储器 (CEECZST)

释放堆栈 (CEERLHP)

动态屏幕管理器可联编 API

动态屏幕管理器 (DSM) 可联编 API 是一组屏幕 I/O 接口，它们为 ILE 高级语言提供了建立和管理显示屏幕的一种动态方法。

DSM API 属于下列功能组:

- **低级服务**

  低级服务 API 提供至 5250 数据流命令的直接接口。使用该 API 来查询和操纵显示屏幕的状态；建立、查询和操纵与显示屏幕交互作用的输入和命令缓冲区；以及定义字段并将数据写入显示屏幕。

- **窗口服务**

  使用窗口服务 API 来建立、删除、移动窗口和改变窗口大小；以及在对话期间并行地管理多个窗口。

- **对话服务**

  对话服务 API 提供一般页面调度接口，可以使用它来建立、查询和操纵对话；以及对对话执行输入和输出操作。

有关 DSM 可联编 API 的详情，参见 *System API Reference*。

第12章 高级优化技术

程序简要表编制是一种高级优化技术，它根据运行 ILE 程序和服务程序时收集到的统计数据，在这些程序中将过程重新排序或在过程内编码。这种重新排序可提高指令高速缓存的使用率并减少程序所需的页面调度，因而提高了性能。程序的该语义特性不受程序简要表编制影响。

由程序简要表编制实现的性能改进取决于应用程序的类型。一般来讲，从将大多数时间用于应用程序代码本身、而不是将时间用于运行期或进行输入／输出处理的程序中，您可获得更多的改进。另外，不同的 AS/400 模型具有不同的指令高速缓存能力。具有受限高速缓存能力的模型可能不能与具有扩展高速缓存能力的模型获得一样多的改进。因此，您可能需要考虑用户设置以确定简要表的模型。

程序简要表编制只可用于满足下列条件的 ILE 程序和服务程序:

- 程序是特别为 V4R2M0 或以后发行版建立的。
- 程序的目标发行版与当前的系统发行版级别相同。
- 程序是通过使用 *FULL (30) 或以上的优化级别编译的。

注: 因为优化需求，在使用程序简要表编制之前应完全调试程序。

简要表编制的类型

可以单独使用或组合使用下列两种方法来对程序进行简要表编制:

- 块排序
- 过程排序

块排序简要表编制对采用条件转移的每一方的次数进行记录。它在已联编的模块内重新排序代码，以使较频繁使用的条件不转移。这通过增加下一指令在指令高速缓存中的可能性、减少从主存储器中获取它的需要，来提高指令高速缓存使用率。

过程排序简要表编制对程序内每个过程调用另一个过程的次数进行记录。它在程序内将过程重新排序，以使最频繁调用的过程打包在一起。在将调用过程放入主存储器的同时，通过增加将被调用过程放入主存储器的可能性来减少页面调度。

尽管您可以选择将哪种类型的简要表编制数据用于您的程序，但是仍建议您两种类型都使用。这将产生最好的性能增益。

如何编制程序简要表

编制程序简要表过程有五个步骤:

1. 启用程序以收集简要表编制数据。
2. 使用“开始程序简要表编制”(STRPGMPRF) 命令来在系统上开始程序简要表编制收集。
3. 通过在该程序的高使用率代码路径中运行它来收集简要表编制数据。由于程序简要表编制使用在运行程序时收集到的统计数据来执行这些优化，所以基于应用程序的典型使用来收集此数据是很关键的。

4. 使用“结束程序简要表编制”(ENDPGMPRF) 命令来在系统上结束程序简要表编制收集。
5. 根据所收集的简要表编制数据，通过请求对代码重新排序来将所收集的简要表编制数据应用于程序，以获取最优性能。

程序简要表编制对所使用的优化转换器的级别是敏感的。 IBM 建议在同一机器上执行这些步骤。（否则所收集的数据可能不可用。）

- 在为收集简要表编制数据启用了程序与下一次应用优化转换器 PTF 之间，不要应用优化转换器 PTF。
- 在为收集简要表编制数据启用了程序与下一次应用系统发行版级别之间，不要应用系统发行版级别。

启用程序以收集简要表编制数据

若启用了至少一个联编至某程序中的模块来收集简要表编制数据，则该程序被启用来收集简要表编制数据。启用程序来收集简要表编制数据有下面两种方法：通过更改一个或多个 *MODULE 对象来收集简要表编制数据、然后用这些模块创建或更新该程序，或者通过在创建该程序来收集简要表编制数据之后更改该程序，两种技术都可以产生一个启用了已联编模块的程序，来收集简要表编制数据。

根据所使用的 ILE 语言，在编译器命令上可能存在一个选项来创建模块（启用该模块来收集简要表编制数据）。只要 ILE 语言支持至少为 *FULL (30) 的优化级别，就可以通过在简要表编制数据 (PRFDTA) 参数上指定 *COL，来使用更改模块 (CHGMOD) 命令更改任何 ILE 模块以收集简要表编制数据。

在已经通过使用“更改程序”(CHGPGM) 或“更改服务程序”(CHGSRVPGM) 命令、并同时在简要表编制数据 (PRFDTA) 参数上指定 *COL 来创建了一个可观察的程序之后，可启用该程序来收集简要表编制数据。这将引起所有与具有目标发行版 V4R2M0 或以后版本、并且优化级别为 30 或以上的程序联编的模块被启用来收集简要表编制数据。

注： 可以只为所建立的、与所在的系统具有相同发行版级别的程序和服务程序启用简要表编制数据。所建立的这些程序和服务程序中的模块也必须与该系统具有相同的发行版级别。

要启用模块或程序来收集简要表编制数据，需要重新转换对象。因此，启用模块或程序来收集简要表编制数据所需要的时间与强制重新建立该对象（FRCCRT 参数）所花的时间差不多。另外，由于由优化转换器生成的额外机器指令，该对象的大小可能较大。

一旦启用了程序或模块来收集简要表编制数据，就不能除去建立数据的可观察能力，直到发生下列其中一种情况：

- 所收集的简要表编制数据适用于该程序
- 程序或模块被更改而不能再用来收集简要表编制数据。

使用“显示模块”(DSPMOD)、“显示程序”(DSPPGM) 或“显示服务程序”(DSPSRVPGM) 命令，通过指定 DETAIL(*BASIC)，来确定是否启用了模块或程序来收集简要表编制数据。对于程序或服务程序，使用 DETAIL(*MODULE) 中的选项 5（显示说明）来确定启用哪个（些）联编模块来收集简要表编制数据。有关详情，参见主题“如何分辨程序或模块已编制简要表或已被启用来进行收集”。

注: 若程序已经具有所收集的简要表编制数据（当程序运行时所收集的统计数据），则当重新启用一个程序来收集简要表编制数据时将清除此数据。有关详情，参见“启用管理程序来收集简要表编制数据”。

收集简要表编制数据

为使程序更新简要表编制数据计数，必须在所启用的用来收集简要表编制数据的程序要运行的机器上启动程序简要表编制。这允许启动大型的、运行时间长的应用程序，并且允许在收集简要表编制数据之前达到稳定状态。这将在发生数据收集时为您提供控制。

使用“启动程序简要表编制”(STRPGMPRF) 命令以在机器上启动程序简要表编制。要在机器上结束程序简要表编制，使用“结束程序简要表编制”(ENDPGMPRF) 命令。这两个命令都是随公共权限 *EXCLUDE 交付的。当机器进行 IPL 时，将隐式地结束程序简要表编制。

一旦开始了程序简要表编制，所运行的、同时也是被启用来收集简要表编制数据的任何程序或服务程序都将更新其简要表编制数据计数。无论在发出 STRPGMPRF 命令之前该程序是否是活动的，都会发生这种情况。

若您正从中收集简要表编制数据的程序可以被机器上的多个作业调用，则所有这些作业都将更新该简要表编制数据。若这不是所期望的，则将在一个单独的库中复制该程序的一个复制副本，并且替代地使用该副本。

注:

1. 当在机器上启动了程序简要表编制，则在运行所启用的用来收集简要表编制数据的程序时，简要表编制数据计数将增加。因此，若先前运行了此程序而随后未清除这些计数，则可能添加了“失效”简要表编制数据记数。可使用若干种方法来强制清除这些简要表编制数据计数。有关详情，参见“启用管理程序来收集简要表编制数据”。
2. 并不是每次增加简要表编制数据计数时都将它们写入 DASD，因为这样做可能导致程序运行期的大幅度下降。仅当程序自然调出页面时才将简要表编制数据计数写入 DASD。要确保已将简要表编制数据计数写入 DASD，使用“清除存储池”(CLRPOOL) 命令来清除在其中运行程序的存储池。

应用所收集的简要表编制数据

应用所收集的简要表编制数据可执行下列操作:

1. 指示机器使用所收集的简要表编制数据来在程序中重新排序过程（过程排序简要表编制数据）和／或过程内的代码（基本块简要表编制数据），以获取最优性能。
2. 从程序中除去机器指令，这些指令是先前当启用该程序来收集简要表编制数据时添加的（即，不再启用该程序来收集简要表数据）。
3. 将所收集的简要表编制数据转换为可观察数据; *BLKORD（基本块简要表编制可观察性）和 *PRCORD（过程排序简要表编制可观察性）。

一旦已将所收集的数据应用到该程序，则不能再次应用它。要再次应用简要表编制数据，需要您完成“如何为程序编制简要表”中概述的步骤。**当启用程序来收集简要表编制数据时，废弃先前应用的任何简要表编制数据。**

若想再次应用已收集的数据，则在应用简要表编制数据之前复制该程序。若您试过从每种类型的简要表编制（块排序、过程排序或两者）中获益，则这可能是您所期望的。

要应用简要表编制数据，使用“更改程序”(CHGPGM) 或“更改服务程序”(CHGSRVPGM) 命令，并对简要表编制数据 (PRFDTA) 参数指定应用块排序简要表编制数据 (*APYBLKORD)、过程简要表编制数据 (*APYPRCORD) 或这两种类型的简要表编制数据 (*APYALL)。建议指定 *APYALL。

当对程序应用简要表编制数据时，将创建两种新形式的可观察性能力，并与该程序一起保存。可通过使用“更改程序”(CHGPGM) 和“更改服务程序”(CHGSRVPGM) 命令来除去这些新的可观察性能力。

- 当对程序应用块排序简要表编制数据时，隐式地添加了 *BLKORD 可观察性能力。这允许机器在重新转换程序的情况下，为该程序保留所应用的块排序简要表编制数据。
- 当对程序应用过程排序简要表编制数据时，隐式地添加了 *PRCORD 可观察性能力。这允许机器在重新转换或更新程序的情况下，为该程序保留所应用的过程排序简要表编制数据。

例如，您对程序应用块排序简要表编制数据，然后除去 *BLKORD 可观察性能力。仍为该程序编制块排序简要表。然而，导致您的程序重新转换的任何更改还将导致不再对它编制块排序简要表。

注: 除去 *CRTDTA 可观察性能力还将导致隐式地除去 *BLKORD 可观察性能力。这是因为仅当重新转换程序时才需要 *BLKORD 可观察性能力。由于当除去了 *CRTDTA 可观察性能力时不能重新转换程序，所以不再需要 *BLKORD，并且也将它除去。然而不除去 *PRCORD 可观察性能力。

另外，不建议使用不同版本的优化转换器来重新转换具有 *BLKORD 可观察性能力的程序。该不同版本不同于用来启用程序并应用简要表编制数据的优化转换器的版本。优化转换器 PTF 和操作系统的新发行版可能会使某些基本的块简要表编制数据无效。

启用管理程序来收集简要表编制数据

通过使用“更改程序”(CHGPGM) 或“更改服务程序”(CHGSRVPGM) 命令更改所启用的用来收集简要表编制数据的程序，将在当该更改需要重新转换程序时隐式地导致简要表编制数据清零。例如，若将所启用的用来收集简要表编制数据的程序从优化级别 *FULL 更改为优化级别 40，则任何所收集的简要表编制数据都将被隐式地清除。当复原所启用的用来收集简要表编制数据的程序，并且在“复原对象”(RSTOBJ) 命令上指定了 FRCOBJCVN(*YES *ALL) 时，也是这样。

通过使用“更新程序”(UPDPGM) 或“更新服务程序”(UPDSRVPGM) 命令更新所启用的用来收集简要表编制数据的程序，将在当结果程序仍被启用来收集简要表编制数据时隐式地导致简要表编制计数被清除。例如，程序 P1 包含模块 M1 和 M2。启用了联编在 P1 中的模块 M1 来收集简要表编制数据而不启用 M2。只要启用了这些模块其中之一，用模块 MI 或 M2 更新程序 P1 将导致仍启用的用来收集简要表编制数据的程序。所有简要表编制数据计数将被清除。然而，若通过在“更改模块”(CHGMOD) 命令的简要表编制数据 (PRFDTA) 参数上指定 *NOCOL 来将模块 M1 更改为不再启用来收集简要表编制数据，则用模块 M1 更新程序 P1 将导致不再启用程序 P1 来收集简要表编制数据。

通过在“更改程序”(CHGPGM) 或“更改服务程序”(CHGSRVPGM) 命令的简要表编制数据 (PRFDTA) 参数上指定 *CLR 来从程序中明确地清除简要表编制计数。注意，要使用 *CLR 选项该程序必须不是活动的。

若不再想启用该程序来收集简要表编制数据，则在“更改程序”(CHGPGM) 或“更改服务程序”(CHGSRVPGM) 命令的简要表编制数据 (PRFDTA) 参数上指定 *NOCOL。这将更改该程序，以使它回到启用该程序来收集简要表编制数据之前的状态。还可以通过 CHGMOD 命令或重新编译模块来将该模块的 PRFDTA 值更改为 *NOCOL，并且将该模块重新与程序联编。

通过对程序应用简要表编制数据来管理程序

若通过使用“更改程序”(CHGPGM) 或“更改服务程序”(CHGSRVPGM) 命令更改已应用了简要表编制数据的程序，则若下列两个情况都为真，所应用的简要表编制数据将丢失:

- 更改要求重新转换程序。

  注: 不更改已应用简要表编制数据的程序的优化级别。这是因为简要表编制数据是与优化级别无关的。

- 所需的简要表编制可观察性能力已被除去。

若更改要求是启用该程序来收集简要表编制数据，则不管是否已除去简要表编制可观察性能力，所有已应用的简要表编制数据都将丢失。这样的要求将导致所启用的程序来收集简要表编制数据。

一些示例:

- 程序 A 应用了过程排序和块排序简要表编制数据。已经从程序中除去了 *BLKORD 可观察性能力，但没有除去 *PRCORD 可观察性能力。运行了 CHGPGM 命令来更改程序 A 的性能收集属性，它还要求重新转换程序 A。这种更改请求将导致程序 A 不再是块排序简要表编制。然而，仍然应用过程排序简要表编制数据。
- 程序 A 应用了过程排序和块排序简要表编制数据。已经从程序中除去了 *BLKORD 和 *PRCORD 可观察性能力。运行了 CHGPGM 命令来更改程序 A 的用户简要表属性，它还要求重新转换程序 A。这种更改请求将导致程序 A 不再是块排序或过程排序简要表编制。程序 A 将转回至应用简要表编制数据之前的状态。
- 程序 A 应用了块排序简要表编制数据。已经从程序中除去了 *BLKORD 可观察性能力。运行了 CHGPGM 命令来更改程序 A 的文本，它**不**要求重新转换程序 A。这种更改请求将导致程序 A 仍然是块排序简要表编制。
- 程序 A 应用了过程排序简要表编制数据。未从程序中除去 *PRCORD 可观察性能力。运行了 CHGPGM 命令以启用程序 A 来收集简要表编制数据（这导致重新转换程序 A）。这将导致程序 A 不再是过程排序简要表编制，并将它保留在这样一种状态下: 从未应用过简要表编制数据，但是在已清除了所有简要表编制数据计数的情况下，现在启用该程序来收集简要表编制数据。

如何分辨程序或模块是否已编制简要表或已为收集启用

使用“显示程序”(DSPPGM) 或“显示服务程序”(DSPSRVPGM) 命令，指定 DETAIL(*BASIC)，以确定程序的程序简要表编制数据属性。“简要表编制数据”的值将是下列值其中之一:

- *NOCOL - 未启用程序来收集简要表编制数据。
- *COL - 启用了程序中的一个或多个模块来收集简要表编制数据。此值不指示是否实际收集了简要表编制数据。
- *APYALL - 对此程序应用了块排序和过程排序简要表编制数据。不再启用简要表编制数据的收集。
- *APYBLKORD - 对此程序中的一个或多个已联编模块的过程应用了块排序简要表编制数据。这些应用只是对先前已启用来收集简要表编制数据的那些已联编的模块。不再启用简要表编制数据的收集。
- *APYPRCORD - 对此程序应用了过程排序程序简要表编制数据。不再启用简要表编制数据的收集。

要显示已联编在程序内的模块的程序简要表编制数据属性，使用 DSPPGM 或 DSPSRVPGM DETAIL(*MODULE)，在已联编到程序中的模块上指定选项 5，以在模块级查看此参数的值。“简要表编制数据”的值将是下列值其中之一:

- *NOCOL - 未启用此已联编的模块来收集简要表编制数据。
- *COL - 启用了些已联编的模块来收集简要表数据。此值不指示是否实际收集了简要表编制数据。
- *APYBLKORD - 对此已联编的模块的一个或多个过程应用了块排序简要表编制数据。不再启用简要表编制数据的收集。

另外，DETAIL(*MODULE) 显示下列字段以给出受程序简要表编制数据属性影响的过程数的指示。

- 过程数 – 模块中的过程总数。
- 重新排序块过程数 – 在此重新排序基本块的模块中的过程数。
- 实际的块排序过程数 – 在此已联编模块中的过程数，该已联编模块是指当应用块排序简要表数据时收集了块排序简要表编制数据的已联编模块。当运行基准时，可能是这样的情况: 由于没在该基准中执行指定的过程，所以没有为该过程收集的数据。这样此计数将影响已在该基准中执行的实际过程数。

使用 DSPMOD 命令以确定模块的简要表编制属性。“简要表编制数据”的值将是下列值其中之一。它将从不显示 *APYBLKORD，因为基本块数据只能应用于与程序联编的模块而不能用于独立模块。

- *NOCOL - 未启用模块来收集简要表数据。
- *COL - 启用了模块来收集简要表数据。

附录A. CRTPGM、CRTSRVPGM、UPDPGM 或 UPDSRVPGM 命令的输出列表

此附录显示联编器列表的示例，并说明可能作为使用联编器语言的结果而出现的错误。

联编器列表

“建立程序”(CRTPGM)、“建立服务程序”(CRTSRVPGM)、“更新程序”(UPDPGM) 以及“更新服务程序”(UPDSRVPGM) 命令的联编器列表几乎是相同的。此主题在第60页的『联编器语言示例』中提供了一个 CRTSRVPGM 命令的联编器列表，CRTSRVPGM 命令是用以建立 FINANCIAL 服务程序的。

可以在 CRTPGM、CRTSRVPGM、UPDPGM 或 UPDSRVPGM 命令的细节 (DETAIL) 参数上指定三种列表类型:

*BASIC

*EXTENDED

*FULL

基本列表

若在 CRTPGM、CRTSRVPGM、UPDPGM 或 UPDSRVPGM 命令上指定 DETAIL(*BASIC)，则列表包含下列各项:

- 在 CRTPGM、CRTSRVPGM、UPDPGM 或 UPDSRVPGM 命令上指定的值
- 简要摘要表
- 显示完成联编过程的某些片段所花费的时间长度的数据

图48、图49和第119页的图50显示此信息。

```
                                建立服务程序                          页      1

服务程序  . . . . . . . . . . . . . . . . . . :   FINANCIAL
库  . . . . . . . . . . . . . . . . . . . . . :   MYLIB
调出  . . . . . . . . . . . . . . . . . . . . :   *SRCFILE
调出源文件  . . . . . . . . . . . . . . . . . :   QSRVSRC
库  . . . . . . . . . . . . . . . . . . . . . :   MYLIB
调出源成员  . . . . . . . . . . . . . . . . . :   *SRVPGM
激活组  . . . . . . . . . . . . . . . . . . . :   *CALLER
允许更新  . . . . . . . . . . . . . . . . . . :   *YES
允许联编的 *SRVPGM 库名更新 . . . . . . . . . :   *NO
建立选项  . . . . . . . . . . . . . . . . . . :   *GEN        *NODUPPROC  *NODUPVAR   *DUPWARN
列表细节  . . . . . . . . . . . . . . . . . . :   *FULL
用户简要表  . . . . . . . . . . . . . . . . . :   *USER
置换现存服务程序  . . . . . . . . . . . . . . :   *YES
目标发行版  . . . . . . . . . . . . . . . . . :   *CURRENT
允许重新初始化  . . . . . . . . . . . . . . . :   *NO
权限  . . . . . . . . . . . . . . . . . . . . :   *LIBCRTAUT
文本  . . . . . . . . . . . . . . . . . . . . :

模块        库              模块        库              模块        库              模块        库

MONEY       MYLIB           CALCS       MYLIB
RATES       MYLIB           ACCTS       MYLIB

服务                        服务                        服务                        服务
程序        库              程序        库              程序        库              程序        库

*NONE

联编                        联编                        联编                        联编
目录        库              目录        库              目录        库              目录        库

*NONE
```

图 48. 在 CRTSRVPGM 命令上指定的值

```
                                建立服务程序                          页      3

                                   简要摘要表

程序入口过程 . . . . . . . . . . . :   0

多个强定义 . . . . . . . . . . . . :   0

未解析的引用 . . . . . . . . . . . :   0

                          * * * * *  简  要  摘  要  表  结  束  * * * * *
```

图 49. 简要摘要表

```
                                   建立服务程序                                 页     23

                                      联编统计

符号收集 CPU 时间  . . . . . . . . . . . . . . . . . . :                .018
符号解析 CPU 时间  . . . . . . . . . . . . . . . . . . :                .006
联编目录解析 CPU 时间  . . . . . . . . . . . . . . . . :                .403
联编器语言编译 CPU 时间  . . . . . . . . . . . . . . . :                .040
列表建立 CPU 时间  . . . . . . . . . . . . . . . . . . :               1.622
程序/服务程序建立 CPU 时间  . . . . . . . . . . . . . . :                .178

总 CPU 时间  . . . . . . . . . . . . . . . . . . . . . :               2.761
总经过时间 . . . . . . . . . . . . . . . . . . . . . . :              11.522

                          * * * * *  联   编   统   计   结   束  * * * * *

*CPC5D0B - 在库 MYLIB 中建立的服务程序 FINANCIAL。

                          * * * * *  建  立  服  务  程  序  列  表  结  束  * * * * *
```

图 50. 联编统计

扩展列表

若在 CRTPGM、CRTSRVPGM、UPDPGM 或 UPDSRVPGM 命令上指定 DETAIL(*EXTENDED), 则该列表包括由 DETAIL(*BASIC) 提供的所有信息以及扩展的摘要表。扩展的摘要表显示已解析的调入（引用）数, 以及已处理的调出（定义）数。对于 CRTSRVPGM 或 UPDSRVPGM 命令, 该列表还显示使用的联编器语言、生成的特征符, 以及哪些调入（引用）与哪些调出（定义）匹配。 图51、第120页的图52和第121页的图53显示附加数据的示例。

```
                                   建立服务程序                                 页      2

                                    扩展的摘要表

  有效定义  . . . . . . . . . . . . :          418
  强  . . . . . . . . . . . . . . . :          418
  弱  . . . . . . . . . . . . . . . :            0
  已解析的引用  . . . . . . . . . . :           21
  对强定义  . . . . . . . . . . . . :           21
  对弱定义  . . . . . . . . . . . . :            0

                          * * * * *  扩  展  的  摘  要  表  结  束  * * * * *
```

图 51. 扩展的摘要列表

```
                                建立服务程序                                  页      4

                                联编器信息列表
模块 . . . . . :   MONEY
库 . . . . . :   MYLIB
已联编 . . . . . :   *YES

     编号      符号         引用     标识符                          类型      范围        调出      关键字

     00000001  Def                   main                            Proc      模块        强
     00000002  Def                   Amount                          Proc      SrvPgm      强
     00000003  Def                   Payment                         Proc      SrvPgm      强
     00000004  Ref          0000017F  Q LE AG_prod_rc                 数据
     00000005  Ref          0000017E  Q LE AG_user_rc                 数据
     00000006  Ref          000000AC  _C_main                         Proc
     00000007  Ref          00000180  Q LE leDefaultEh                Proc
     00000008  Ref          00000181  Q LE mhConversionEh             Proc
     00000009  Ref          00000125  _C_exception_router             Proc

模块 . . . . . :   RATES
库 . . . . . :   MYLIB
已联编 . . . . . :   *YES

     编号      符号         引用     标识符                          类型      范围        调出      关键字

     0000000A  Def                   Trem                            Proc      SrvPgm      强
     0000000B  Def                   Rate                            Proc      SrvPgm      强
     0000000C  Ref          0000017F  Q LE AG_prod_rc                 数据
     0000000D  Ref          0000017E  Q LE AG_user_rc                 数据
     0000000E  Ref          00000180  Q LE leDefaultEh                Proc
     0000000F  Ref          00000181  Q LE mhConversionEh             Proc
     00000010  Ref          00000125  _C_exception_router             Proc

模块 . . . . . :   CALCS
库 . . . . . :   MYLIB
已联编 . . . . . :   *YES

     编号      符号         引用     标识符                          类型      范围        调出      关键字

     00000011  Def                   Calc1                           Proc      模块        强
     00000012  Def                   Calc2                           Proc      模块        强
     00000013  Ref          0000017F  Q LE AG_prod_rc                 数据
     00000014  Ref          0000017E  Q LE AG_user_rc                 数据
     00000015  Ref          00000180  Q LE leDefaultEh                Proc
     00000016  Ref          00000181  Q LE mhConversionEh             Proc
     00000017  Ref          00000125  _C_exception_router             Proc

模块 . . . . . :   ACCTS
库 . . . . . :   MYLIB
已联编 . . . . . :   *YES

     编号      符号         引用     标识符                          类型      范围        调出      关键字

     00000018  Def                   OpenAccount                     Proc      SrvPgm      强
     00000019  Def                   CloseAccount                    Proc      SrvPgm      强
     0000001A  Ref          0000017F  Q LE AG_prod_rc                 数据
     0000001B  Ref          0000017E  Q LE AG_user_rc                 数据
     0000001C  Ref          00000180  Q LE leDefaultEh                Proc
     0000001D  Ref          00000181  Q LE mhConversionEh             Proc
     0000001E  Ref          00000125  _C_exception_router             Proc
```

图 52. 联编器信息列表 (1/2)

```
服务程序 . . . . . :   QC2SYS
  库 . . . . . :     *LIBL
  已联编 . . . . . :     *NO

     编号      符号         引用     标识符                          类型        范围        调出        关键字

     0000001F  Def                   系统                            Proc                    强

服务程序 . . . . . :   QLEAWI
库 . . . . . :   *LIBL
已联编 . . . . . :   *YES

     编号      符号         引用     标识符                          类型        范围        调出        关键字

     0000017E  Def                   Q LE AG_user_rc                 数据                    强
     0000017F  Def                   Q LE AG_prod_rc                 数据                    强
     00000180  Def                   Q LE leDefauTtEh                Proc                    强
     00000181  Def                   Q LE mhConversionEh             Proc                    强
```

图 52. 联编器信息列表 (2/2)

```
                                    建立服务程序                                    页     14

                                    联编器语言列表

STRPGMEXP  PGMLVL(*CURRENT)
  EXPORT SYMBOL('Term')
  EXPORT SYMBOL('Rate')
  EXPORT SYMBOL('Amount')
  EXPORT SYMBOL('Payment')
  EXPORT SYMBOL('OpenAccount')
  EXPORT SYMBOL('CloseAccount')
ENDPGMEXP
******** 调出特征符: 00000000ADCEFEE088738A98DBA6E723。
STRPGMEXP  PGMLVL(*PRV)
  EXPORT SYMBOL('Term')
  EXPORT SYMBOL('Rate')
  EXPORT SYMBOL('Amount')
  EXPORT SYMBOL('Payment')
ENDPGMEXP
******** 调出特征符: 000000000000000000ADC89D09E0C6E7。

                          * * * * *  联 编 器 语 言 列 表 结 束  * * * * *
```

图 53. 联编器语言列表

完全列表

若在 CRTPGM、CRTSRVPGM、UPDPGM 或 UPDSRVPGM 命令上指定 DETAIL(*FULL)，则该列表包括为 DETAIL(*EXTENDED) 提供的所有细节，以及交叉引用列表。第122页的图54显示所提供的附加数据的部分示例。

交叉引用列表

标识符	Defs	Refs Ref	Refs Ref	类型	库	对象
xlatewt	000000DD			*SRVPGM	*LIBL	QC2UTIL1
yn	00000140			*SRVPGM	*LIBL	QC2UTIL2
y0	0000013E			*SRVPGM	*LIBL	QC2UTIL2
y1	0000013F			*SRVPGM	*LIBL	QC2UTIL2
Amount	00000002			*MODULE	MYLIB	MONEY
Calc1	00000011			*MODULE	MYLIB	CALCS
Calc2	00000012			*MODULE	MYLIB	CALCS
CloseAccount	00000019			*MODULE	MYLIB	ACCTS
CEECRHP	000001A0			*SRVPGM	*LIBL	QLEAWI
CEECZST	0000019F			*SRVPGM	*LIBL	QLEAWI
CEEDATE	000001A9			*SRVPGM	*LIBL	QLEAWI
CEEDATM	000001B1			*SRVPGM	*LIBL	QLEAWI
CEEDAYS	000001A8			*SRVPGM	*LIBL	QLEAWI
CEEDCOD	00000187			*SRVPGM	*LIBL	QLEAWI
CEEDSHP	000001A1			*SRVPGM	*LIBL	QLEAWI
CEEDYWK	000001B3			*SRVPGM	*LIBL	QLEAWI
CEEFMDA	000001AD			*SRVPGM	*LIBL	QLEAWI
CEEFMDT	000001AF			*SRVPGM	*LIBL	QLEAWI
CEEFMTM	000001AE			*SRVPGM	*LIBL	QLEAWI
CEEFRST	0000019E			*SRVPGM	*LIBL	QLEAWI
CEEGMT	000001B6			*SRVPGM	*LIBL	QLEAWI
CEEGPID	00000195			*SRVPGM	*LIBL	QLEAWI
CEEGTST	0000019D			*SRVPGM	*LIBL	QLEAWI
CEEISEC	000001B0			*SRVPGM	*LIBL	QLEAWI
CEELOCT	000001B4			*SRVPGM	*LIBL	QLEAWI
CEEMGET	00000183			*SRVPGM	*LIBL	QLEAWI
CEEMKHP	000001A2			*SRVPGM	*LIBL	QLEAWI
CEEMOUT	00000184			*SRVPGM	*LIBL	QLEAWI
CEEMRCR	00000182			*SRVPGM	*LIBL	QLEAWI
CEEMSG	00000185			*SRVPGM	*LIBL	QLEAWI
CEENCOD	00000186			*SRVPGM	*LIBL	QLEAWI
CEEQCEN	000001AC			*SRVPGM	*LIBL	QLEAWI
CEERLHP	000001A3			*SRVPGM	*LIBL	QLEAWI
CEESCEN	000001AB			*SRVPGM	*LIBL	QLEAWI
CEESECI	000001B2			*SRVPGM	*LIBL	QLEAWI
CEESECS	000001AA			*SRVPGM	*LIBL	QLEAWI
CEESGL	00000190			*SRVPGM	*LIBL	QLEAWI
CEETREC	00000191			*SRVPGM	*LIBL	QLEAWI
CEEUTC	000001B5			*SRVPGM	*LIBL	QLEAWI
CEEUTCO	000001B7			*SRVPGM	*LIBL	QLEAWI
CEE4ABN	00000192			*SRVPGM	*LIBL	QLEAWI
CEE4CpyDvfb	0000019A			*SRVPGM	*LIBL	QLEAWI
CEE4CpyIofb	00000199			*SRVPGM	*LIBL	QLEAWI
CEE4CpyOfb	00000198			*SRVPGM	*LIBL	QLEAWI
CEE4DAS	000001A4			*SRVPGM	*LIBL	QLEAWI
CEE4FCB	0000018A			*SRVPGM	*LIBL	QLEAWI
CEE4HC	00000197			*SRVPGM	*LIBL	QLEAWI
CEE4RAGE	0000018B			*SRVPGM	*LIBL	QLEAWI
CEE4RIN	00000196			*SRVPGM	*LIBL	QLEAWI
OpenAccount	00000018			*MODULE	MYLIB	ACCTS
Payment	00000003			*MODULE	MYLIB	MONEY
Q LE leBdyCh	00000188			*SRVPGM	*LIBL	QLEAWI
Q LE leBdyEpilog	00000189			*SRVPGM	*LIBL	QLEAWI
Q LE leDefaultEh	00000180	00000007	0000000E	*SRVPGM	*LIBL	QLEAWI
	00000015		0000001C			
Q LE mhConversionEh	00000181	00000008	0000000F	*SRVPGM	*LIBL	QLEAWI
	00000016		0000001D			
Q LE AG_prod_rc	0000017F	00000004	0000000C	*SRVPGM	*LIBL	QLEAWI
	00000013	0000001A				
Q LE AG_user_rc	0000017E	00000005	0000000D	*SRVPGM	*LIBL	QLEAWI
		00000014	0000001B			
Q LE HdlrRouterEh	0000018F			*SRVPGM	*LIBL	QLEAWI
Q LE RtxRouterCh	0000018E			*SRVPGM	*LIBL	QLEAWI
Rate	0000000B			*MODULE	MYLIB	RATES
Term	0000000A			*MODULE	MYLIB	RATES

图 54. 交叉引用列表

示例服务程序的列表

第119页的图50、第120页的图52和第122页的图54显示某些列表数据，这些列表数据是当在第64页的图37中指定了 DETAIL(*FULL) 以建立 FINANCIAL 服务程序时生成的。这些图显示联编统计、联编器信息列表和交叉引用列表。

示例服务程序的联编器信息列表

联编器信息列表 (第120页的图52) 包括下列数据和列标题:

- 处理的模块或服务程序的库和名称。

  若对于模块对象，*已联编*字段显示值 *YES，则将该模块标记为要通过复制联编。若对服务程序，*已联编*字段显示值 *YES，则通过引用联编该服务程序。若对模块对象或服务程序，*已联编*字段显示值 *NO，则该对象不包括在联编中。原因是该对象没有提供满足未解析的调入的调出。

- 编号

  对于每个处理的模块或服务程序，有一个唯一的标识符 (ID) 与每个调出（定义）或调入（引用）相关。

- 符号

  此列标识作为调出 (Def) 或调入 (Ref) 的符号名。

- Ref

  在此列 (Ref) 中所指定的编号是满足该调入请求的那个调出 (Def) 的唯一 ID。例如，在第120页的图52中，调入 00000005 的唯一 ID 匹配调出 0000017E 的唯一 ID。

- 标识符

  它是调出或调入的符号的名称。对唯一 ID 00000005 调入的符号名是 Q_LE_AG_user_rc。对唯一 ID 0000017E 调出的符号名也是 Q_LE_AG_user_rc。

- 类型

  若符号名是过程，则将类型标识为 Proc。若符号名是数据项，则将类型标识为 “数据”。

- 范围

  对于模块，此列标识是在模块层还是在至服务程序的公共接口处存取调出的符号名。若正在建立程序，则仅可在模块层存取调出的符号名。若正在建立服务程序，则在模块层或服务程序 (SrvPgm) 层都可以存取调出的符号名。若调出的符号是公共接口的一部分，则*范围*列中的值必须是 SrvPgm。

- 调出

  此列标识从模块或服务程序中调出的数据项的强度。

- 关键字

  此列包含关于任何弱调出的附加信息。典型情况下，此列为空白。

示例服务程序的交叉引用列表

第122页的图54中的交叉引用列表是查看联编器信息中所提供的数据的另一个方法。交叉引用列表包括下列列标题:

- 标识符

  在符号解析期间处理的调出的名称。

- Def

  与每个调出相关的唯一 ID。
- Ref

  此列中的号码指示对此调出 (Def) 解析的调入 (Ref) 的唯一 ID。
- 类型

  指示该调出是来自于 *MODULE 还是 *SRVPGM 对象。
- 库

  在命令上或联编目录中所指定的库名。
- 对象

  提供调出 (Def) 的对象的名称。

示例服务程序的联编统计

第119页的图50显示建立服务程序 FINANCIAL 的一组统计。该统计标识当联编器处理建立请求时在何处花费时间。您对此节中提供的数据只有间接控制权。某些处理额外开销的数量是无法计算的。因此，*总 CPU 时间*字段中所列示的值大于前面字段中所列示的时间的总和。

联编器语言错误

当在服务程序的建立期间系统处理联编器语言时，可能会出现错误。若在“建立服务程序”(CRTSRVPGM) 命令上指定 DETAIL(*EXTENDED) 或 DETAIL(*FULL)，则可以在假脱机文件中看到错误。

可能出现下列资料信息:

- 填充的特征符
- 截取的特征符

可能出现下列警告错误:

- 当前调出块限制接口
- 调出块重复
- 在先前调出上的重复符号
- 多次禁用了级别检查，忽略它
- 不允许多个当前调出块，假定为先前

可能发生了下列严重错误:

- 当前调出块为空
- 调出块未完成，在 ENDPGMEXP 之前发现了“文件结束”
- 未启动调出块，需要 STRPGMEXP
- 不能嵌套调出块，ENDPGMEXP 丢失
- 调出必须存在于调出块内部
- 不同调出块具有相同的特征符，必须更改调出
- 多个通配符匹配
- 没有当前调出块

- 没有通配符匹配
- 先前调出块为空
- 特征符包含变体字符
- LVLCHK(*NO) 要求 SIGNATURE(*GEN)
- 特征符语法无效
- 要求了符号名
- 符号不允许作为服务程序调出
- 未定义符号
- 语法无效

填充的特征符

图55显示包含此信息的联编器语言列表。

联编器语言列表

```
             STRPGMEXP  SIGNATURE('短特征符')
******** 填充的特征符
             EXPORT     SYMBOL('Proc_2')
             ENDPGMEXP

******** 调出特征符: E2889699A340A289879581A3A4998540。

                              *****  联  编  器  语  言  列  表  结  束  *****
```

图 55. 所提供的特征符少于 16 字节，因此填充它

此为资料信息。

建议的更改

不需要更改。

若希望避免该信息，确保所提供的特征符正好是 16 字节长。

截取的特征符

图56显示包含此信息的联编器语言列表。

联编器语言列表

```
             STRPGMEXP  SIGNATURE('此特征符非常长')
******** 截取的特征符
             EXPORT     SYMBOL('Proc_2')
             ENDPGMEXP

******** 调出特征符: E38889A240A289879581A3A499854089。

                              *****  联  编  器  语  言  列  表  结  束  *****
```

图 56. 仅将所提供的数据的前 16 字节用作特征符

此为资料信息。

建议的更改

不需要更改。

若希望避免该信息，确保所提供的特征符正好是 16 字节长。

当前调出块限制接口

图57显示包含此错误的联编器语言列表。

联编器语言列表

```
STRPGMEXP  PGMLVL(*CURRENT)
  EXPORT SYMBOL(A)
  EXPORT SYMBOL(B)
ENDPGMEXP
******** 调出特征符: 00000000000000000000000000000CD2。
STRPGMEXP  PGMLVL(*PRV)
  EXPORT SYMBOL(A)
  EXPORT SYMBOL(B)
  EXPORT SYMBOL(C)
ENDPGMEXP
******** 调出特征符: 00000000000000000000000000000CDE3。
******** 当前调出块限制接口。

                    * * * * *  联 编 器 语 言 列 表 结 束  * * * * *
```

*图 57. PGMLVL(*PRV) 比 PGMLVL(*CURRENT) 调出了更多符号*

此为警告错误。

PGMLVL(*PRV) 调出块比 PGMLVL(*CURRENT) 调出块指定了更多符号。

若未发生其他错误，则建立服务程序。

若下列两者为真:

- PGMLVL(*PRV) 支持了名为 C 的过程
- 在新服务程序下，不再支持过程 C

，则在此服务程序中调用 C 过程的任何 ILE 程序或服务程序都将在运行期得到一个错误。

建议的更改

1. 确保 PGMLVL(*CURRENT) 调出块比 PGMLVL(*PRV) 调出块有更多要调出的符号。
2. 再次运行 CRTSRVPGM 命令。

在此示例中，将 EXPORT SYMBOL(C) 错误地添加至了 STRPGMEXP PGMLVL(*PRV) 块而不是 PGMLVL(*CURRENT) 块。

重复的调出块

第127页的图58显示包含此错误的联编器语言列表。

```
STRPGMEXP  PGMLVL(*CURRENT)
  EXPORT SYMBOL(A)
  EXPORT SYMBOL(B)
ENDPGMEXP
******** 调出特征符: 00000000000000000000000000000CD2。
STRPGMEXP  PGMLVL(*PRV)
  EXPORT SYMBOL(A)
  EXPORT SYMBOL(B)
ENDPGMEXP
******** 调出特征符: 00000000000000000000000000000CD2。
******** 重复的调出块。

                    * * * * *  联 编 器 语 言 列 表 结 束  * * * * *
```

图 58. 重复的 STRPGMEXP/ENDPGMEXP 块

此为警告错误。

多个 STRPGMEXP 和 ENDPGMEXP 块以完全相同的顺序调出了所有相同的符号。

若未发生其他错误，则建立服务程序。重复的特征符仅在所建立的服务程序中包括一次。

建议的更改

1. 执行下列更改之一:
   - 确保 PGMLVL(*CURRENT) 调出块是正确的。适当地更新它。
   - 删除重复的调出块。
2. 再次运行 CRTSRVPGM 命令。

在此示例中，指定了 PGMLVL(*CURRENT) 的 STRPGMEXP 命令需要在 EXPORT SYMBOL(B) 后面添加下以下源行:

```
EXPORT  SYMBOL(C)
```

在先前调出上的重复符号

图59显示包含重复符号错误的联编器语言列表。


```
STRPGMEXP  PGMLVL(*CURRENT)
  EXPORT SYMBOL(A)
  EXPORT SYMBOL(B)
  EXPORT SYMBOL(A)
******** 在先前调出上的重复符号
  EXPORT SYMBOL(C)
ENDPGMEXP
******** 调出特征符: 00000000000000000000000000000CDED3。

                    * * * * *  联 编 器 语 言 列 表 结 束  * * * * *
```

图 59. 重复调出的符号

此为警告错误。

将一个要从服务程序中调出的符号在 STRPGMEXP 和 ENDPGMEXP 块中指定了多次。

若未发生其他错误，则建立服务程序。只从服务程序中调出第一个重复的符号。所有重复的符号都将影响生成的特征符。

建议的更改

1. 从联编器语言源文件中删除重复源行的其中之一。
2. 再次运行 CRTSRVPGM 命令。

在此示例中，除去第二个 EXPORT SYMBOL(A)。

不能多次禁用级别检查，忽略它

图60显示包含此错误的联编器语言列表。

联编器语言列表

```
STRPGMEXP  PGMLVL(*CURRENT) LVLCHK(*NO)
  EXPORT SYMBOL(A)
  EXPORT SYMBOL(B)
ENDPGMEXP
******** 调出特征符: 00000000000000000000000000000000。
STRPGMEXP  PGMLVL(*PRV) LVLCHK(*NO)
******** 不能多次禁用级别检查，忽略
  EXPORT SYMBOL(A)
ENDPGMEXP
******** 调出特征符: 000000000000000000000000000000C1。

                    * * * * *  联 编 器 语 言 列 表 结 束  * * * * *
```

*图 60. 多个 STRPGMEXP 命令指定了 LVLCHK(*NO)*

此为警告错误。

多个 STRPGMEXP 块指定了 LVLCHK(*NO)。

若未发生其他错误，则建立服务程序。将第二个以及后续的 LVLCHK(*NO) 都假定为 LVLCHK(*YES)。

建议的更改

1. 确保只有一个 STRPGMEXP 块指定了 LVLCHK(*NO)。
2. 再次运行 CRTSRVPGM 命令。

在此示例中，PGMLVL(*PRV) 调出块是唯一指定了 LVLCHK(*NO) 的调出块。从 PGMLVL(*CURRENT) 调出块中除去 LVLCHK(*NO) 值。

不允许多个当前调出块，假定为先前

第129页的图61显示包含此错误的联编器语言列表。

联编器语言列表

```
STRPGMEXP  PGMLVL(*CURRENT)
  EXPORT SYMBOL(A)
  EXPORT SYMBOL(B)
  EXPORT SYMBOL(C)
ENDPGMEXP
******** 调出特征符:  00000000000000000000000000000CDE3。
STRPGMEXP
  EXPORT SYMBOL(A)
******** 不允许多个 “当前” 调出块，假定为 “先前”。
  EXPORT SYMBOL(B)
ENDPGMEXP
******** 调出特征符:  00000000000000000000000000000CD2。

                    * * * * *  联 编 器 语 言 列 表 结 束  * * * * *
```

*图 61. 指定了多个 PGMLVL(*CURRENT) 值*

此为警告错误。

在多个 STRPGMEXP 命令上指定了 PGMLVL(*CURRENT) 值或允许缺省为 PGMLVL(*CURRENT)。将第二个以及后续的具有 PGMLVL(*CURRENT) 值的调出块假定为 PGMLVL(*PRV)。

若未发生其他错误，则建立服务程序。

建议的更改

1. 将适当的源文本更改为 STRPGMEXP PGMLVL(*PRV)。
2. 再次运行 CRTSRVPGM 命令。

在此示例中，第二个 STRPGMEXP 是要更改的那个。

当前调出块为空

图62显示包含此错误的联编器语言列表。

联编器语言列表

```
STRPGMEXP  PGMLVL(*CURRENT)
ENDPGMEXP
******** 调出特征符:  00000000000000000000000000000000。
***错误:  当前调出块为空。

                    * * * * *  联 编 器 语 言 列 表 结 束  * * * * *
```

*图 62. 没有要从 STRPGMEXP PGMLVL(*CURRENT) 块中调出的符号*

此为严重错误。

没有标识为要从 *CURRENT 调出块中调出的符号。

不建立服务程序。

建议的更改

1. 执行下列更改之一:
   - 添加要调出的符号名。
   - 除去空 STRPGMEXP-ENDPGMEXP 块，并且将另一个 STRPGMEXP-ENDPGMEXP 块当作 PGMLVL(*CURRENT)。
2. 运行 CRTSRVPGM 命令。

在此示例中，将以下源行添加至联编器语言源文件中的 STRPGMEXP 和 ENDPGMEXP 命令之间:

```
EXPORT SYMBOL(A)
```

调出块未完成，在 ENDPGMEXP 之前发现“文件结束”

图63显示包含此错误的联编器语言列表。

联编器语言列表

```
STRPGMEXP  PGMLVL(*CURRENT)
***错误:  语法无效。
***错误:  调出块未完成，在 ENDPGMEXP 之前发现了 “文件结束”。

                    * * * * *  联 编 器 语 言 列 表 结 束  * * * * *
```

图 63. 找不到 ENDPGMEXP 命令，但找到了 “源文件的结束”

此为严重错误。

在到达文件的末尾之前未找到 ENDPGMEXP。

不建立服务程序。

建议的更改

1. 执行下列更改之一:
   - 在适当的地点添加 ENDPGMEXP 命令。
   - 除去没有匹配的 ENDPGMEXP 命令的任何 STRPGMEXP 命令，并除去任何要调出的符号名。
2. 运行 CRTSRVPGM 命令。

在此示例中，在 STRPGMEXP 命令之后添加下列行:

```
  EXPORT  SYMBOL(A)
ENDPGMEXP
```

未启动调出块，需要 STRPGMEXP

第131页的图64显示包含此错误的联编器语言列表。

```
ENDPGMEXP
***错误: 未启动调出块, 需要 STRPGMEXP。
***错误: 没有“当前”调出块

                    * * * * *  联 编 器 语 言 列 表 结 束  * * * * *
```

图 64. STRPGMEXP 命令丢失

此为严重错误。

在找到 ENDPGMEXP 命令之前没找到 STRPGMEXP 命令。

不建立服务程序。

建议的更改

1. 执行下列更改之一:
   - 添加 STRPGMEXP 命令。
   - 除去任何已调出的符号和 ENDPGMEXP 命令。
2. 运行 CRTSRVPGM 命令。

在此示例中，将下列两个源行添加至联编器语言源文件中 ENDPGMEXP 命令之前。

```
STRPGMEXP
  EXPORT  SYMBOL(A)
```

不能嵌套调出块，ENDPGMEXP 丢失

图65显示包含此错误的联编器语言列表。


```
STRPGMEXP  PGMLVL(*CURRENT)
  EXPORT SYMBOL(A)
  EXPORT SYMBOL(B)
STRPGMEXP  PGMLVL(*PRV)
***错误: 不能嵌套调出块, ENDPGMEXP 丢失。
  EXPORT SYMBOL(A)
ENDPGMEXP
******** 调出特征符: 000000000000000000000000000000C1。

                    * * * * *  联 编 器 语 言 列 表 结 束  * * * * *
```

图 65. ENDPGMEXP 命令丢失

此为严重错误。

在找到另一个 STRPGMEXP 命令之前没找到 ENDPGMEXP 命令。

不建立服务程序。

建议的更改

1. 执行下列更改之一:
   - 在下一个 STRPGMEXP 命令之前添加 ENDPGMEXP 命令。
   - 除去 STRPGMEXP 命令和任何要调出的符号名。
2. 运行 CRTSRVPGM 命令。

在此示例中，将 ENDPGMEXP 命令添加至联编器源文件中第二个 STRPGMEXP 命令之前。

调出必须存在于调出块内部

图66显示包含此错误的联编器语言列表。

联编器语言列表

```
STRPGMEXP  PGMLVL(*CURRENT)
  EXPORT SYMBOL(A)
  EXPORT SYMBOL(B)
ENDPGMEXP
******** 调出特征符: 00000000000000000000000000000CD2。
  EXPORT SYMBOL(A)
***错误: 调出必须存在于调出块内部。

                    * * * * *  联 编 器 语 言 列 表 结 束  * * * * *
```

图 66. 要调出的符号名在 STRPGMEXP-ENDPGMEXP 块之外

此为严重错误。

未在 STRPGMEXP-ENDPGMEXP 块内定义要调出的符号。

不建立服务程序。

建议的更改

1. 执行下列更改之一:
   - 移动要调出的符号。将它放入 STRPGMEXP-ENDPGMEXP 块内。
   - 除去符号。
2. 运行 CRTSRVPGM 命令。

在此示例中，从联编器语言源文件中除去出错的源行。

不同的调出块具有相同的特征符，必须更改调出

此为严重错误。

从调出不同符号的 STRPGMEXP-ENDPGMEXP 块生成了相同的特征符。这种错误情况极不可能出现。对于任何要调出的非普通符号的集合，每 3.4E28 次尝试才应发生一次这种错误。

不建立服务程序。

建议的更改

1. 执行下列更改之一:
   - 添加要从 PGMLVL(*CURRENT) 块中调出的附加符号。

     首选方法是指定一个已调出的符号。这会导致符号重复的警告错误，但是能够帮助确保特征符的唯一。另一个方法是添加另一个要调出但仍未调出的符号。
   - 从模块中更改要调出的符号的名称，并对联编器语言源文件进行相应的更改。
   - 通过在“开始程序调出”(STRPGMEXP) 命令上使用 SIGNATURE 参数来指定特征符。
2. 运行 CRTSRVPGM 命令。

多个通配符匹配

图67显示包含此错误的联编器语言列表。

联编器语言列表

```
STRPGMEXP PGMLVL(*CURRENT)
EXPORT ("A"<<<)
***错误:  通配符说明的多个匹配
EXPORT ("B"<<<)
ENDPGMEXP
******** 调出特征符:  00000000000000000000000000000FFC2。

                    * * * * *  联 编 器 语 言 列 表 结 束  * * * * *
```

图 67. 通配符说明的多个匹配

此为严重错误。

为调出指定的通配符与多个可用于调出的符号匹配。

不建立服务程序。

建议的更改

1. 更详细地指定通配符，以使所希望的匹配调出是唯一的匹配调出。
2. 运行 CRTSRVPGM 命令。

没有当前调出块

第134页的图68显示包含此错误的联编器语言列表。

联编器语言列表

```
STRPGMEXP  PGMLVL(*PRV)
  EXPORT SYMBOL(A)
ENDPGMEXP
******** 调出特征符: 000000000000000000000000000000C1。
***错误: 没有“当前”调出块

                    * * * * *  联 编 器 语 言 列 表 结 束  * * * * *
```

*图 68. 没有 PGMLVL(*CURRENT) 调出块*

此为严重错误。

在联编器语言源文件中找不到 STRPGMEXP PGMLVL(*CURRENT)。

不建立服务程序。

建议的更改

1. 执行下列更改之一:
   - 将 PGMLVL(*PRV) 更改为 PGMLVL(*CURRENT)。
   - 添加是正确 *CURRENT 调出块的 STRPGMEXP-ENDPGMEXP 块。
2. 运行 CRTSRVPGM 命令。

在此示例中，将 PGMLVL(*PRV) 更改为 PGMLVL(*CURRENT)。

没有通配符匹配

图69显示包含此错误的联编器语言列表。

联编器语言列表

```
STRPGMEXP PGMLVL(*CURRENT)
EXPORT ("Z"<<<)
***错误: 没有通配符说明的匹配
EXPORT ("B"<<<)
ENDPGMEXP
******** 调出特征符: 000000000000000000000000000FFC2。

                    * * * * *  联 编 器 语 言 列 表 结 束  * * * * *
```

图 69. 没有通配符说明的匹配

此为严重错误。

为调出指定的通配符不与任何可用于调出的符号匹配。

不建立服务程序。

建议的更改

1. 指定与调出所期望的符号相匹配的通配符。
2. 运行 CRTSRVPGM 命令。

先前调出块为空

图70显示包含此错误的联编器语言列表。

联编器语言列表

```
STRPGMEXP  PGMLVL(*CURRENT)
  EXPORT SYMBOL(A)
  EXPORT SYMBOL(B)
ENDPGMEXP
******** 调出特征符: 00000000000000000000000000000CD2。
STRPGMEXP  PGMLVL(*PRV)
ENDPGMEXP
******** 调出特征符: 00000000000000000000000000000000。
***错误: 先前调出块为空。

                          * * * * *  联 编 器 语 言 列 表 结 束  * * * * *
```

*图 70. 没有 PGMLVL(*CURRENT) 调出块*

此为严重错误。

找到了 STRPGMEXP PGMLVL(*PRV)，但未指定符号。

不建立服务程序。

建议的更改

1. 执行下列更改之一:
   - 添加符号至空的 STRPGMEXP-ENDPGMEXP 块中。
   - 除去空的 STRPGMEXP-ENDPGMEXP 块。
2. 运行 CRTSRVPGM 命令。

在此示例中，从联编器语言源文件中除去空的 STRPGMEXP-ENDPGMEXP 块。

特征符包含变体字符

图71显示包含此错误的联编器语言列表。

联编器语言列表

```
             STRPGMEXP  SIGNATURE('\!cdefghijklmnop')
***错误: 特征符包含变体字符
             EXPORT     SYMBOL('Proc_2')
             ENDPGMEXP

******** 调出特征符: E05A838485868788899191929394959697。

                          * * * * *  联 编 器 语 言 列 表 结 束  * * * * *
```

图 71. 特征符包含变体字符

此为严重错误。

特征符包含不在所有编码字符集标识符 (CCSID) 中的字符。

不建立服务程序。

建议的更改

1. 除去变体字符。
2. 运行 CRTSRVPGM 命令。

在此特定情况中，需要除去的是 \!。

LVLCHK(*NO) 要求 SIGNATURE(*GEN)

图72显示包含此错误的联编器语言列表。

联编器语言列表

```
             STRPGMEXP  SIGNATURE('ABCDEFGHIJKLMNOP')  LVLCHK(*NO)
             EXPORT     SYMBOL('Proc_2')
***错误:  LVLCHK(*NO) 要求 SIGNATURE(*GEN)
             ENDPGMEXP

******** 调出特征符:  C1C2C3C4C5C6C7C8C9D1D2D3D4D5D6D7。

                          * * * * *  联 编 器 语 言 列 表 结 束  * * * * *
```

*图 72. 若指定了 LVLCHK(*NO)，则显式特征符无效*

此为严重错误。

当指定了 LVLCHK(*NO) 时，还要求 SIGNATURE(*GEN)。

不建立服务程序。

建议的更改

1. 执行下列更改之一:
   - 指定 SIGNATURE(*GEN)
   - 指定 LVLCHK(*YES)
2. 运行 CRTSRVPGM 命令。

特征符语法无效

第137页的图73显示包含此错误的联编器语言列表。

```
             STRPGMEXP  SIGNATURE('"abcdefghijkl "')
***错误:  特征符语法无效
***错误:  特征符语法无效
***错误:  语法无效。
***错误:  语法无效。
             EXPORT     SYMBOL('Proc_2')
             ENDPGMEXP

                                * * * * *  联  编  器  语  言  列  表  结  束  * * * * *
```

图 73. 对特征符值指定的内容无效

此为严重错误。

特征符包含无效字符。

不建立服务程序。

建议的更改

1. 从特征符值中除去无效字符。
2. 运行 CRTSRVPGM 命令。

在此情况下，从特征符字段中除去 ″ 字符。

要求了符号名

图74显示包含此错误的联编器语言列表。


```
STRPGMEXP  PGMLVL(*CURRENT)
  EXPORT SYMBOL(A)
  EXPORT SYMBOL(')
***错误:  要求了符号名。
ENDPGMEXP
******** 调出特征符:  000000000000000000000000000000C1。

                                * * * * *  联  编  器  语  言  列  表  结  束  * * * * *
```

图 74. 没有符号要调出

此为严重错误。

没有找到要从服务程序中调出的符号名。

不建立服务程序。

建议的更改

1. 执行下列更改之一:
   - 从联编器语言源文件中除去有错的行。
   - 添加一个要从服务程序中调出的符号名。
2. 运行 CRTSRVPGM 命令。

在此示例中，从联编器语言源文件中除去源行 EXPORT SYMBOL("")。

符号不允许作为服务程序调出

图75显示包含此错误的联编器语言列表。

联编器语言列表

```
STRPGMEXP  PGMLVL(*CURRENT)
  EXPORT SYMBOL(A)
***错误: 符号不允许作为服务程序调出。
  EXPORT SYMBOL(D)
ENDPGMEXP
******** 调出特征符: 00000000000000000000000000000CD4。

                    * * * * *  联 编 器 语 言 列 表 结 束  * * * * *
```

图 75. 符号名无效，不能从服务程序中调出

此为严重错误。

要从服务程序调出的符号未从要通过复制联编的模块的其中之一中调出。典型情况下，指定要从服务程序调出的符号实际上就是需要由该服务程序调入的符号。

不建立服务程序。

建议的更改

1. 执行下列更改之一:
   - 从联编器语言源文件中除去有错的符号。
   - 在 CRTSRVPGM 命令的 MODULE 参数上，指定具有想要调出的符号的模块。
   - 将符号添加至将通过复制联编的模块的其中之一，并且重新建立模块对象。
2. 运行 CRTSRVPGM 命令。

在此示例中，从联编器语言源文件中除去源行 EXPORT SYMBOL(A)。

未定义符号

图76显示包含此错误的联编器语言列表。

联编器语言列表

```
STRPGMEXP  PGMLVL(*CURRENT)
  EXPORT SYMBOL(A)
  EXPORT SYMBOL(Q)
***错误: 未定义符号。
ENDPGMEXP
******** 调出特征符: 00000000000000000000000000000CE8。

                    * * * * *  联 编 器 语 言 列 表 结 束  * * * * *
```

图 76. 在要通过复制联编的模块中未找到符号

此为严重错误。

在将要通过复制联编的模块中找不到要从服务程序中调出的符号。

不建立服务程序。

建议的更改

1. 执行下列更改之一:
   - 从联编器语言源文件中除去未定义的符号。
   - 在 CRTSRVPGM 命令的 MODULE 参数上，指定具有想要调出的符号的模块。
   - 将符号添加至将通过复制联编的模块的其中之一，并且重新建立模块对象。
2. 运行 CRTSRVPGM 命令。

在此示例中，从联编器语言源文件中除去源行 EXPORT SYMBOL(Q)。

语法无效

此为严重错误。

源成员中的语句不是有效的联编器语言语句。

不建立服务程序。

建议的更改

1. 校正源成员以使它包含有效的联编器语言语句。
2. 运行 CRTSRVPGM 命令。

附录B. 优化程序中的异常

在极少数情况下，在用优化级别 30 (*FULL) 或 40 编译的程序中可能出现 MCH3601 异常信息。本附录说明出现此信息的一个示例。当用优化级别 10 (*NONE) 或 20 (*BASIC) 编译同一个程序时，该程序不接收到 MCH3601 异常信息。此示例中的信息是否出现取决于 ILE HLL 编译器如何为数组分配存储器。对于您的语言来说，此示例可能从不出现。

当请求优化级别 30 (*FULL) 或 40 时， ILE 试图通过计算循环外部的数组索引引用来改进性能。当参考循环中的数组时，通常按顺序存取每个元素。通过保存先前循环迭代中的最后一个数组元素地址，可以改进性能。要完成此性能改进， ILE 计算循环外部的第一个数组元素地址并保存该值以供在循环内部使用。

执行以下示例:

```
DCL ARR[1000] INTEGER;
DCL I INTEGER;

I = init_expression;   /* Assume that init_expression evaluates
                          to -1 which is then assigned to I */

/* More statements */

WHILE ( I < limit_expression )

  I = I + 1;

  /* Some statements in the while loop */

  ARR[I] = some_expression;

  /* Other statements in the while loop */

END;
```

若对 ARR[init_expression] 的引用产生了不正确的数组索引，则此示例将导致一个 MCH3601 异常。这是由于 ILE 试图在进入 WHILE 循环之前计算第一个数组元素地址而引起的。

若在优化级别 30 (*FULL) 或 40 时接收到 MCH3601 异常，查找下列情况:

1. 您有一个循环，在使用变量作为数组元素索引之前，循环增大此变量。
2. 在循环入口处的索引变量的初始值是负数。
3. 对使用变量初始值的数组的引用无效。

当存在这些情况时，为使仍可以使用优化级别 30 (*FULL) 或 40，可能会执行下列操作:

1. 将增大变量的那部分程序移至循环顶部。
2. 按需要更改对变量的引用。

将对先前示例进行如下更改:

```
I = init_expression + 1;

WHILE ( I < limit_expression + 1 )
```

```
    ARR[I] = some_expression;
    I = I + 1;
  END;
```

若此更改是不可能的，则将优化级别从 30 (*FULL) 或 40 减至 20 (*BASIC) 或 10 (*NONE)。

附录C. 对 ILE 对象使用的 CL 命令

下列列表指示可以对每个 ILE 对象使用哪些 CL 命令。

对模块使用的 CL 命令

CHGMOD
更改模块

CRTCMOD
建立 C 模块

CRTCBLMOD
建立 COBOL 模块

CRTCLMOD
建立 CL 模块

CRTRPGMOD
建立 RPG 模块

DLTMOD
删除模块

DSPMOD
显示模块

RTVBNDSRC
检索联编器源

WRKMOD
使用模块

对程序对象使用的 CL 命令

CHGPGM
更改程序

CRTBNDC
建立联编的 C 程序

CRTBNDCBL
建立联编的 COBOL 程序

CRTBNDCL
建立联编的 CL 程序

CRTBNDRPG
建立联编的 RPG 程序

CRTPGM
建立程序

DLTPGM
删除程序

DSPPGM
显示程序

DSPPGMREF
显示程序引用

UPDPGM
更新程序

WRKPGM
使用程序

对服务程序使用的 CL 命令

CHGSRVPGM
更改服务程序

CRTSRVPGM
建立服务程序

DLTSRVPGM
删除服务程序

DSPSRVPGM
显示服务程序

UPDSRVPGM
更新服务程序

WRKSRVPGM
使用服务程序

对联编目录使用的 CL 命令

ADDBNDDIRE
添加联编目录项

CRTBNDDIR
建立联编目录

DLTBNDDIR
删除联编目录

DSPBNDDIR
显示联编目录

RMVBNDDIRE
除去联编目录项

WRKBNDDIR
使用联编目录

WRKBNDDIRE
使用联编目录项

对结构化查询语言使用的 CL 命令

CRTSQLCI
建立结构化查询语言 ILE C/400 对象

CRTSQLCBLI
建立结构化查询语言 ILE COBOL/400 对象

CRTSQLRPGI
建立结构化查询语言 ILE RPG/400 对象

对源调试程序使用的 CL 命令

DSPMODSRC
显示模块源

ENDDBG
结束调试

STRDBG
开始调试

用于编辑联编器语言源文件的 CL 命令

STRPDM
启动程序设计开发管理器

STRSEU
启动源入口实用程序

可将下列不可运行的命令输入到联编器语言源文件中:

ENDPGMEXP
结束程序调出

EXPORT
调出

STRPGMEXP
开始程序调出

附录D. 注意事项

本资料是为在美国提供的产品和服务而设计的。IBM 可能未在其他国家提供本文档中讨论的产品、服务或功能部件。请教您的本地 IBM 代表，以获取关于您所在的地区当前可获得的产品和服务的信息。任何对 IBM 产品、程序或服务的引用并不说明或暗示只能使用 IBM 产品、程序或服务。只要未侵犯 IBM 公司的知识产权或其他受保护的合法权利，任何有同等功能的产品、程序或服务项目均可使用，以替换 IBM 的产品、程序或服务项目。然而，用户需负起评估和验证任何非 IBM 产品、程序或服务的责任。

IBM 可能已经申请或正在申请与本文档有关的各项专利权。提供本文档，并不表示允许您使用这些专利。您可以书面形式提出专利申请要求，请寄往:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

下列段落不适用于英国或这样的规定与当地法律不一致的任何其他国家: 国际商业机器公司“按原样”提供此出版物，不作任何类型的担保，无论是明确的还是暗示的，包括（但不限于）暗示的不侵权、可交易性或适合特定目的的保证。一些国家不允许否认某些事务中的明确或隐含的保证，因此，此声明可能不适用于您。

本资料可能会包含技术性错误或印刷错误。错误信息会定期得到更改；这些更改将编入本出版物的新版本中。IBM 可以在任何时间在不通知的情况下改进和／或更改此出版物中描述的产品和／或程序。

此信息中对非 IBM Web 站点的任何引用都只是为了方便而提供的，不以任何方式充当那些 Web 站点的认可。那些 Web 站点处的材料不是此 IBM 产品的材料的一部分，使用那些 Web 站点的后果由您自己承担。

为了以下目的: (i) 允许在独立建立的程序和其他程序（包括本程序）之间进行信息交换 (ii) 允许对已经交换的信息进行相互使用，而希望获取有关本程序的信息的合法用户，应联系:

IBM Corporation
Software Interoperability Coordinator
3605 Highway 52 N
Rochester, MN 55901-7829
U.S.A.

只要遵守适当的条件和条款，包括某些情形下一定数量的付款，都可获取这方面的信息。

IBM 按照“IBM 顾客协议”、“IBM 国际程序许可证协议”或我们之间的任何等价协议的各项条款提供此信息中描述的特许程序和所有可用于它的特许材料。

与非 IBM 产品有关的信息是从那些产品的供应商、他们发布的公告或其他可公开获取的信息源处获取的。 IBM 尚未测试那些产品，因此不能确认其性能的准确性、兼容性或任何其他与非 IBM 产品相关的声明。关于非 IBM 产品的能力问题应该向那些产品的供应商提出。

有关 IBM 的未来方向或意图的所有声明都可更改或撤销而无需另行通知，并且仅代表目标和目的。

此信息包含日常商业活动中使用的数据和报表的示例。为了尽可能完整地说明它们，示例包括个人、公司、商标和产品的名称。所有这些名称都是虚构的，若与实际商业企业使用的名称和地址有任何雷同，纯属巧合。

版权许可证:

此信息包含源语言的样本应用程序，它说明在各种操作平台上的程序设计技术。为了达到开发、使用、销售或分发符合（对其编写样本程序的）操作平台的应用程序设计接口的应用程序的目的，您可以任何形式复制、修改和分发这些样本程序，而不必向 IBM 付款。未在所有情况下彻底测试这些示例。因此，IBM 不能保证或暗示这些程序的可靠性、可服务性或功能。为了达到开发、使用、销售或分发符合 IBM 的应用程序设计接口的应用程序的目的，您可以任何形式复制、修改和分发这些样本程序，而不必向 IBM 付款。

若您正在察看的是本资料软拷贝，则图片和彩色插图可能不会出现。

商标

下列术语是国际商业机器公司在美国和／或其他国家的商标:

Application System/400
AS/400
C/400
COBOL/400
DB2
DB2/400
IBM
ILE
Integrated Language Environment
Language Environment
Operating System/400
OS/2
OS/400
RPG/400
RPG IV
SAA
System 36
Systems Application Architecture
400

Microsoft、Windows、Windows NT 和 Windows 95 标志是 Microsoft 公司的注册商标。

UNIX 是在美国和经 X/Open 有限公司唯一许可的其他国家的注册商标。

其他公司、产品和服务的名称可能是其他公司的商标或服务标记。

书目

有关与 AS/400 系统上的 ILE 环境相关的主题的其他信息，参考下列 IBM AS/400 出版物:

- *Backup and Recovery*, SC41-5304 一书，提供关于计划备份和恢复策略的信息、可用来保存和复原系统数据的不同类型的媒体，以及如何使用日志来记录对数据库文件的更改和如何将该信息用于系统恢复。本手册描述如何计划和设置用户辅助存储池 (ASP)、镜像保护和校验和，以及其他可用性恢复主题。它还描述如何从备份再次安装系统。
- *CL Programming*, SC41-5721 一书，提供关于 AS/400 程序设计主题的范围广阔的讨论，包括有关对象和库、CL 程序设计、程序间的控制流和通信、使用 CL 程序中的对象和建立 CL 程序的一般讨论。其他主题包括预定义的和即发的信息以及信息处理、定义和建立用户定义的命令和菜单、应用程序测试（包括调试方式、断点、跟踪和显示功能）。
- *CL Reference (Abridged)*, SC41-5722 一书，提供 AS/400 控制语言 (CL) 及其 OS/400 命令的说明。（非 OS/400 命令在相应的特许程序出版物中有述。）它还提供 AS/400 系统的*所有* CL 命令的概述，并且描述要编码这些命令所需的语法规则。
- *Communications Management*, SC41-5406 一书，提供关于通信环境中的工作管理、通信状态、跟踪和诊断通信问题、错误处理和恢复、性能的信息，以及特定线路速度和子系统存储器信息。
- *Data Management*, SC41-5710 一书，提供关于使用应用程序中的文件的信息。此手册包括关于下列主题的信息:
  - 系统上的数据管理支持的基本结构和概念。
  - 覆盖和文件重定向（当运行应用程序时对文件临时进行更改）。
  - 通过使用系统命令将数据从一个地方复制到另一个地方来复制文件。
  - 使用双字节数据剪裁系统。
- *DB2 for AS/400 Database Programming*, SC41-5701 一书，提供有关 AS/400 数据库组织的详细讨论，包括关于如何在系统上建立、描述和更新数据库文件的信息。此手册还描述如何使用 OS/400 数据说明规范 (DDS) 关键字定义系统的文件。
- *Distributed Data Management*, SC41-5307 一书，提供关于远程文件处理的信息。它描述如何对 OS/400 分布式数据管理 (DDM) 定义远程文件、如何建立 DDM 文件、通过 DDM 支持哪些文件实用程序，以及与其他系统相关的 OS/400 DDM 需求。
- *ICF Programming*, SC41-5442 一书，提供编写使用 AS/400 通信和 OS/400 系统间通信功能 (OS/400-ICF) 的应用程序所需的信息。此指南还包含关于数据说明规范 (DDS) 关键字、系统提供的格式、返回码、文件传输支持以及程序示例的信息。
- *ILE C for AS/400 Programmer's Guide*, SC09-2712 一书，提供关于如何使用 ILE C/400 语言开发应用程序的信息。它包括关于建立、运行和调试程序的信息。它还包括对于语言间程序和过程调用、场地、异常处理、数据库文件、外部描述文件和设备文件的程序设计考虑事项。还描述了某些性能提示。附录包括关于将源代码从 EPM C/400 或 System C/400 迁移至 ILE C/400 的信息。
- *ILE C for AS/400 Language Reference*, SC09-2711 一书，提供关于如何编写遵守“系统应用体系结构 C 级别 2 ”定义的程序，以及如何使用 ILE C/400 特定功能（如记录 I/O）的信息。它还提供关于 ILE C/400 机器接口库函数的信息。
- *ILE C for AS/400 Run-Time Library Reference*, SC09-2715 一书，提供关于 ILE /400 命令语法、C 的元素、SAA C 库函数、SAA C 的 ILE C/400 库扩展，以及 ILE C/400 机器接口库扩展的快速参考信息。
- *ILE COBOL for AS/400 Programmer's Guide*, SC09-2540 一书，描述如何在 AS/400 系统上编写、编译、联编、运行、调试和维护 ILE COBOL/400 程序。它提供关于如何调用其他 ILE COBOL/400 和非 ILE COBOL/400 程序、与其他程序共享数据、使用指针以及处理异常的程序设计信息。它还描述如何对外部连接设备、数据库文件、显示文件和 ICF 文件执行输入／输出操作。
- *ILE COBOL for AS/400 Reference*, SC09-2539 一书，描述 ILE COBOL/400 程序设计语言。它提供关于 ILE COBOL/400 程序设计语言的结构和 ILE

COBOL/400 源程序结构的信息。它还描述所有“标识部分”段落、“环境部分”子句、“数据部分”段落、“过程部分”语句和“编译指示”语句。

- *ILE RPG for AS/400 Programmer's Guide*, SC09-2507 一书，是使用 RPG IV 程序设计语言的指南，该程序设计语言是在 AS/400 系统上的“集成语言环境”(ILE) 中 ILE RPG/400 的实现。它包括关于建立和运行程序的信息，以及过程调用和语言间程序设计的考虑事项。本指南还包含调试和异常处理，并且说明如何在 RPG 程序中使用 AS/400 文件和设备。附录包括关于迁移到 RPG IV 和样本编译器列表的信息。它可使人们对数据处理概念以及 RPG 语言有一个基本的了解。
- *ILE RPG for AS/400 Reference*, SC09-2508 一书，提供使用 RPG IV 程序设计语言为 AS/400 系统编写程序所需的信息。本手册逐位置、逐关键字地描述所有 RPG 规范的有效项，并提供所有操作码和内部函数的详细说明。本手册还包含关于 RPG 逻辑循环、数组和表、编辑功能和指示符的信息。
- *Intrasystem Communications Programming*, SC41-5447 一书，提供关于在同一个 AS/400 系统上的两个应用程序之间的交互式通信的信息。本指南描述可以编码至程序中的通信操作，该程序使用系统内通信支持与另一个程序通信。它还提供关于开发使用 OS/400 系统间通信功能 (OS/400-ICF) 的系统内通信应用程序的信息。
- *Security - Basic*, SC41-5301 一书，说明为什么安全性是必需的，定义主要概念，并且提供关于在 AS/400 系统上计划、实现和监控基本安全性的信息。
- *Security - Reference*, SC41-5302 一书，说明可如何使用系统安全性支持，以保护系统和数据免受不具有正确权限的人使用，保护数据免受有意或无意的损坏或破坏，保持安全性信息最新以及在系统上设置安全性。
- *System API Reference*, SC41-5801 一书，为有经验的程序员提供关于如何将应用程序设计接口 (API) 用于如下这些 OS/400 功能的信息:
  - 配置
  - 数据库文件
  - 动态屏幕管理器
  - 信息处理
  - 网络管理
  - 安全性
  - 源调试
  - 假脱机文件
  - 用户接口
  - 用户对象
  - 用户定义的通信
  - 工作管理
  - 使用软件产品

  本书包括原始程序模型 (OPM) 和“集成语言环境”(ILE) API。某些 API 提供 CL 命令的替代项。
- *Work Management*, SC41-5306 一书，提供关于如何建立和更改工作管理环境的信息。其他主题包括：调整系统的说明，收集性能数据（包括关于所收集的数据的记录格式和内容的信息）的说明，使用系统值来控制或更改系统的整体操作的说明，以及如何收集数据以确定谁正在使用该系统和哪些资源正在使用的说明。

激活 (activation). 准备要运行的程序的处理步骤。激活包括为作业中的程序分配和初始化静态存储器，以及完成某些联编的部分。激活之后，程序就已准备好运行了。

激活组 (activation group). 作业中资源的区分。激活组由分配给一个或多个程序的系统资源（程序或过程变量、确认定义和打开文件的存储器）组成。

自变量 (argument). 在高级语言 (HLL) 过程调用中，表示调用过程传送给被调用过程的值的表达式。

微元 (atomic). (1) 在 SQL 中，是数据库数据定义功能的特性，它允许在电源中断或异常结束发生时完成功能或返回至其原始状态。 (2) 在确认控制中，是允许对对象的个别更改以单一更改形式出现的特性。

自动数据 (automatic data). 进入和重新进入过程时以同一个值分配的数据。从该过程退出时，不为下次进入该过程而保留数据的值。自动数据的范围是激活组内的过程调用。对应于*外部数据 (external)*和*本地数据 (local data)*。

自动存储器 (automatic storage). 在 OS/400 应用程序设计接口中，调用或运行程序时由系统建立的区域。每个例行程序都可以有自动存储器或静态存储器。在自动存储器中，每次调用或运行程序时都定义变量。对应于*静态存储器 (static storage)*。

联编 (bind). 通过组合“集成语言环境”(ILE) 编译器建立的一个或多个模块，建立可运行的程序。另见*联编器 (binder)*和*联编 (binding)*。

联编器 (binder). 一个系统部件，它通过打包“集成语言环境”(ILE) 模块并解析那些模块之间传送的符号来建立联编程序。

联编器语言 (binder language). 一小组命令（STRPGMEXP、EXPORT 和 ENDPGMEXP），它们定义服务程序的外部接口（特征符）。这些命令在源文件中，不能单独运行。

联编 (binding). 通过打包“集成语言环境”(ILE) 模块并解析在那些模块之间传送的符号来建立程序的过程。

联编目录 (binding directory). 建立 ILE 程序或服务程序时可能需要的模块和服务程序的名称的列表。联编目录不是模块和服务程序的储存库。而是，它允许通过名称和类型引用它们。

已联编程序 (bound program). 组合“集成语言环境”(ILE) 编译器建立的一个或多个模块的 AS/400 对象。

断点 (breakpoint). 程序（由命令或状态指定）中的位置，在该处，系统停止处理该程序，并将控制交给显示站用户或指定的程序。

调用 (call). 在调用堆栈上为被调用过程或程序添加新项目，并将控制传送给被调用对象。

调用信息队列 (call message queue). 对作业中每一调用堆栈项都存在的信息队列。

调用堆栈 (call stack). 当前对作业启动的所有程序或过程的有序列表。次序是后进先出。可用 CALL 指令明确地启动程序和过程，也可从某些其他事件隐式地启动。

调用堆栈项 (call stack entry). 调用堆栈中的程序或过程。每个调用堆栈项都有关于该过程的局部自动变量的信息，以及关于范围限于调用堆栈项的其他资源（如状态处理程序和取消处理程序）的信息。

确认 (commit). 使自从上一确认或回滚操作后对一个或多个数据库文件所作的所有更改永久化，并使更改过的记录可供其他用户使用。

确认控制 (commitment control). 分组数据库文件操作的方法，允许通过 Commit 命令以单一单元形式处理一组数据库更改，或通过 Rollback 命令以单一单元形式除去一组数据库更改。

确认定义 (commitment definition). 系统用来维护整个路由选择步骤和（在系统故障的情况下）整个初始程序装入 (IPL) 中的确认控制环境的信息。此信息是从“启动确认控制”命令获取的，该命令建立路由选择步骤中的确认控制环境和文件打开信息。

状态 (condition). 在“集成语言环境”(ILE) 模型中，一个在高级语言中与系统无关的错误状态表示法。对于 OS/400 程序，每个 ILE 状态都有相对应的异常信息。

状态记号 (condition token). 12 个字节的数据结构，它在多个“系统应用软件*”(SAA) 参与系统之间是一致的，允许应用程序员将状态与基础异常信息相关联。

控制边界 (control boundary). 当发生未监控的错误，或使用高级语言终止动词时，用作控制被传送所至的点的调用堆栈项。

调试 (debug). 检测、诊断并消去程序中的错误。

调试方式 (debug mode). 一种方式，在这种方式中，程序提供关于其活动的详细输出，以辅助用户检测和校正程序本身的错误或者程序或系统的配置中的错误。

调试程序 (debugger). 用来检测和跟踪计算机程序中的错误的工具。

直接监控处理程序 (direct monitor handler). 一个异常处理程序，它允许应用程序员直接说明围绕某些高级语言源语句的异常监控。对于 ILE C/400，此能力是通过 #pragma 语句启用的。

动态程序调用 (dynamic program call). 在运行期从一个程序或过程对另一程序 (*PGM) 的调用。控制传送至被调用程序。

动态屏幕管理器 (dynamic screen manager). 用于控制屏幕交互作用的一组 API。

动态存储器 (dynamic storage). 在应用程序设计接口中，程序员在运行程序或过程时分配的存储器区域。对应于*自动存储器 (automatic storage)*和*静态存储器 (static storage)*。

EPM. 参见*扩展程序模型 (Extended Program Model)*。

调出 (export). 在模块或程序中定义的可供其他模块或程序使用的外部符号。另见*外部符号 (external symbol)*。对应于*调入 (import)*。

扩展程序模型 (EPM) (Extended Program Model)(EPM). 用来在 AS/400 系统上用定义过程调用的高级语言编译源代码和建立程序的一组功能。

外部数据 (external data). 从一个过程调出并由另一过程调入的数据。对应于*内部数据 (internal data)*。

外部信息队列 (external message queue). 作业中运行的所有程序和过程用来发送和接收作业外的信息（例如，在交互式作业与工作站用户之间）的信息队列。

外部符号 (external symbol). 在高级语言程序中定义的，表示诸如过程或变量之类的东西的项目。解析外部符号是联编器连接模块以构成已联编程序或服务程序的方法。

先进先出 (FIFO)(first-in first-out (FIFO). 一种排队技术，队列中要处理的下一请求是在队列上停留时间最长的最高优先级的请求。对应于*后进先出 (LIFO)(last-in first-out (LIFO))*。

句柄光标 (handle cursor). 跟踪当前异常处理程序的指针。对应于*继续光标 (resume cursor)*。

堆 (heap). 向过程提供动态存储器的对象。该对象是激活组的一部分，在删除激活组时删除它。参见*动态存储器 (dynamic storage)*。

堆标识符 (heap identifier). 标识其激活组中的堆的数字。

ILE. 参见*集成语言环境 (Integrated Language Environment)*。

调入 (import). 对另一模块或程序中定义的外部符号的引用。对应于*调出 (export)*。

特定于实例的信息 (instance-specific information). 包含与状态记号相关联的信息实例的引用关键字的数据。若信息引用关键字是零，则没有相关联的信息。

集成语言环境 (ILE)(Integrated Language Environment (ILE)). 一组结构和接口，为所有与 ILE 一致的高级语言提供公共运行期环境和运行期可联编应用程序设计接口 (API)。

内部数据 (internal data). 只有定义它的过程或 OPM 程序才识别的数据。当过程将控制返回至调用程序或过程时，局部数据被删除。对应于*外部数据 (external data)*。

作业 (job). 计算机单独运行的工作单元。在 ILE 中，作业是资源和数据的集合，且由一个或多个激活组组成。另见*激活组 (activation group)*。

后进先出 (LIFO)(last-in first-out (LIFO)). 一种排队技术，要检索的下一项目就是最近放在队列上的项目。对应于*先进先出 (FIFO)(first-in first-out (FIFO))*。

LIFO. 参见*后进先出 (LIFO)(last-in first-out (LIFO))*。

局部数据 (local data). 只有定义它的过程才识别的数据。局部数据的范围就是包围它的激活组。

模块 (module). 在“集成语言环境”(ILE) 模型中，编译源代码所生成的对象。模块不能运行。要运行，必须将模块联编到程序对象或服务程序中。

嵌套异常 (nested exception). 当处理另一异常时发生的异常。

可观察性 (observability). 从与对象存储在一起的数据派生的对象的特性，它允许从对象检索源、允许在不进行重新编译的情况下重建对象，并允许象征性地调试对象。

ODP. 参见*开放数据通路 (ODP)(open data path (ODP))*。

开放数据通路 (ODP)(open data path (ODP)). 打开文件时建立的控制块。 ODP 包含关于合并的文件属性的信息和输入或输出操作返回的信息。仅当打开文件时，才存在 ODP。

操作描述符 (operational descriptor). 关于自变量的大小、形状和类型的信息，系统将其传送给被调用过程。当被调用过程不能精确地预测自变量的格式（例如，不同类型的字符串）时，此信息很有用。

OPM. 参见*原始程序模型 (OPM)(original program model (OPM))*。

优化级别 (optimization level). 处理程序的效率的级别，由应用程序员确定。当在系统上优化代码时，系统使用处理快捷方式来减少生成同一输出所需的系统资源量。然后，系统将处理快捷方式转换成机器代码，从而使程序能更有效率地运行。

优化 (optimize). 使已编译代码的性能达到最大。

原始程序模型 (OPM)(original program model (OPM)). 在引入“集成语言环境”(ILE) 模型之前，用来在 AS/400 系统上编译源代码和建立高级语言程序的功能集。

参数 (parameter). (1) 在“集成语言环境”中，定义传送给被调用过程的自变量的类型的标识符。 (2) 向命令或程序提供的的值，用作该命令或程序的输入或用来控制该命令或程序的操作。

PEP. *参见程序入口过程 (program entry procedure)*。

渗滤 (percolate). 在“集成语言环境”(ILE) 模型中，指的是拒绝处理状态。将未更改的状态传送至下一状态处理程序。

过程 (procedure). 执行特定任务，然后返回至调用者的一组自包含的高级语言语句。

过程调用 (procedure call). 对已联编程序中模块内的过程所作的调用。另见*静态过程调用 (static procedure call)*和*过程指针调用 (procedure pointer call)*。对应于*程序调用 (program call)*。

过程指针调用 (procedure pointer call). 高级语言调用机制，用于指定要调用的过程的地址。过程指针调用提供了动态调用过程的方法。例如，通过处理过程名或地址的数组或表，应用程序员可动态地将过程调用按路径发送至不同的过程。对应于*静态过程调用 (static procedure call)*。

程序 (program). 在“集成语言环境”(ILE) 模型中，将模块一起联编所生成的可运行对象。

程序调用 (program call). 对 ILE 程序或 OPM 程序所作的调用。另见*动态程序调用 (dynamic program call)*。对应于*过程调用 (procedure call)*。

程序入口过程 (PEP)(program entry procedure (PEP)). 编译器提供的作为动态程序调用上 ILE 程序的入口点的过程。对应于*用户入口过程*。

促进 (promote). 将未处理的状态转换成具有不同含义的新状态。将新状态传送至另一状态处理程序。

公共接口 (public interface). 其他“集成语言环境”(ILE) 对象可存取的已调出过程和数据项的名称。

已解析的调入 (resolved import). 类型和名称与调出的类型和名称完全匹配的调入。

继续光标 (resume cursor). 跟踪异常处理程序在处理异常之后可继续处理的当前位置的指针。

继续点 (resume point). 程序中的指令，处理异常之后，处理在该处继续。

返回 (return). 除去调用堆栈项，并将控制传送回前一调用堆栈项中的调用过程或程序。

回滚 (roll back). 将应用程序或用户更改的数据复原为它在上一确认边界处的状态。

SAA. *参见系统应用体系结构 (SAA)(Systems Application Architecture (SAA))*。

范围 (scope). 语言语句的语义效果达到的范围。范围可以是作业或激活组。

服务程序 (service program). 执行实用程序功能的已联编程序，可由其他已联编程序调用。另见*已联编程序 (bound program)*。

特征符 (signature). 标识服务程序支持的接口的值。特征符基于服务程序中允许的调出和调出的顺序。

源调试程序 (soruce debugger). 通过显示“集成语言环境”(ILE) 程序的源代码的表示法来对这些程序进行调试的工具。

静态过程调用 (static procedure call). 高级语言 (HLL) 调用语句，指定要调用的“集成语言环境”(ILE) 过程的名称。联编期间，过程的名称被解析为它的地址。对应于*过程指针调用 (procedure pointer call)*。

静态存储器 (static storage). 在 OS/400 应用程序设计接口中，在激活程序时系统建立的区域。每个例行程序都可以有自动存储器或静态存储器。在静态存储器内，定义了变量。对应于*自动存储器 (automatic storage)*。

静态变量 (static variables). 为程序激活说明的变量。对于作业中的程序来说，静态变量可能有多个副本，对于在其中激活该程序的每个激活组都有一个副本。

强调出 (strong export). 仅允许联编器使用一个外部符号定义的调出。选择联编器搜索中的第一个定义并废弃重复的定义。

符号解析 (symbol resolution). 联编器用来将要通过复制联编的模块集中的未解析调入与指定模块和服务程序提供的调出集相匹配的过程。

系统应用体系结构 (SAA)(Systems Application Architecture (SAA)). 定义一组规则的体系结构，该组规则用于设计对策略操作系统（如 OS/2 、OS/400、VM 和 MVS 操作系统）的公共用户接口、程序设计接口、应用程序和通信的支持。

事务 (transaction). 对系统上对象的一组个别的更改，应以对用户的整体更改的形式出现。

转换程序 (translator). OS/400 部件，它执行程序或模块编译中的最终步骤。在“集成语言环境” (ILE) 模型中，转换程序称为优化转换程序。

UEP. 参见*用户入口过程 (user entry procedure)*。

未解析的调入 (unresolved import). 类型和名称尚未与调出的类型和名称相匹配的调入。

用户入口过程 (UEP)(user entry procedure (UEP)). 应用程序员编写的入口过程，它是动态程序调用的目标。此过程从程序入口过程 (PEP) 获取控制。对应于*程序入口过程 (program entry procedure))*。

弱调出 (weak export). 允许同一外部符号有数个定义的调出。每个弱调出都有相关联的关键字值，它是数据项的大小。联编器选择具有最大关键字值的弱调出。对应于*强调出 (strong export)*。

通配符 (wildcard). 可用来表示零个或多个字符的字符串的一个字符或一组字符。

本索引按汉语拼音，数字，英文字母和特殊字符顺序排列。

[B]

[C]

[D]

[F]

[G]

[H]

[J]

[K]

[L]

[M]

[Q]

[R]

[S]

[T]

[W]

[X]

[Y]

[Z]

A

C

D

E

H

I

M

O

P

Q

R

S

U

读者意见表

AS/400e 系列
ILE 概念
版本 **4**

SC31-2020-02

姓名

地址

单位及部门

电话号码

读者意见表

SC31-2020-02

请沿此线
撕下或折起

折起并封口　　请勿使用钉书机　　折起并封口

在此
贴上
邮票

IBM CORPORATION
ATTN DEPT 542 IDCLERK
3605 HWY 52 N
ROCHESTER MN 55901-7829

折起并封口　　请勿使用钉书机　　折起并封口

SC31-2020-02

请沿此线
撕下或折起

Printed in China

SC31-2020-02

Spine information:

IBM AS/400e 系列 AS/400 ILE 概念 V4R3 版本 4 SC31-2020-02