BC　ABAP/4　用户指南

版　本　3.0

目　　录

SAP 专用术语及图标说明

惯例约定	是用于
屏幕文本	你在屏幕上看到的单词和字母（包括系统提示信息、字段 名称、屏幕标题、菜单名称和菜单项目）。
用户输入	准确的用户输入。你在键盘上键入的单词和字母要完全与文档中的相同。
<可变用户输入>	可变的用户输入。尖括号表示你可以用适当的键盘输入替换这些变量。
全部大写	报表名、程序名、项目代码、表格名、 ABAP/4语言要素、文件名和目录。
书目标题	与期它的书相互参照。
键　标	键盘上的键。通常功能键（例如**F2和ENTER键）是用这种方式表示的。**

下列图标...	有助于明确...
Ê¾ Àý	一个例子。例子有助于理解复杂的概念或操作。.
×¢ Êí	一个注释。注释包含了重要的信息，例如需要特别考虑的 情况或例外。
×¢ Òâ	一个注意。注意有助于你避免错误，例如可能导致造成数 据丢失的错误。

一个具有概况信息的标题。该图标一般用于指定各章的概览的。

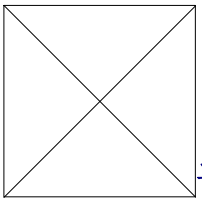
一个具有过程信息的标题。这些标题可告诉你 R/3 系统每一

步处理过程。

一个具有处理过程信息的标题。该标题描述了 R/3 的业务过

程。

一个具有概念信息的标题。用于明确你所需要的概念和背景资料，以便完成一向业务过程。



读者注意事项

ABAP/4 用户指南是关于 ABAP/4 编程语言的手册。其目的在于

- 向新的 ABAP/4 程序员提供从基本语言组件到复杂报表和事务编码的全面指导
- 向资深 ABAP/4 程序员提供解决特殊问题的参考方案

为满足这些需求，特提供一个整体概述并本指南分成三个部分，循序渐进，逐步深入

- ABAP/4 的基本组件
- 如何编写报表程序
- 如何编写事务

ABAP/4 用户指南目的不在替换 ABAP/4 关键字文档。可以通过如下方式调用该文档

- 从“ABAP/4 编辑器：初始屏幕”（事务 SE38）中选择“实用程序 -> ABAP/4 关键字文档
- 选择 ABAP/4 编辑器中的“帮助”
- 将光标放在 ABAP/4 编辑器中的关键字上，然后按下 F1 键

ABAP/4 用户指南的真正目的在于说明如何使用以及为何使用 ABAP/4 关键字。并为此提供了许多示例，构成用户自己测试程序的基础。

要获取详细信息，可以进行以下某种操作：

- 关于关键字的详细信息，请参阅关键字文档。
- 关于 ABAP/4 环境术语的详细信息，请从“ABAP/4 编辑器：初始屏幕”（事务 SE38）中选择“实用程序 -> ABAP/4 关键字文档”。然后就会出现与解释文本相链接的主题层次。要找出这些文本用于哪些主题，请选择“查看 -> 标题和关键字”。屏幕右侧就会出现一个名称列表。对于所有以 ABEN 而非 TITL 开头的名称，如果双击树结构中的主题，则会出现文本屏幕。
- 关于如何提高 ABAP/4 任务性能的技巧，请从“ABAP/4 开发工作台”上选择事务 SE30 或“测试 -> 运行时间分析”，然后单击“提示和技巧”。

ABAP/4 简介

SAP 最初开发 ABAP/4（高级商业应用程序设计）语言仅为内部使用，为应用程序员提供优化的工作环境。经过不断的改进和修改以满足商业领域的需要。现在，ABAP/4 已成为 SAP 开发所有自己的应用程序的仅有工具。
SAP 客户用 ABAP/4 进行其自身开发。这样的开发对 R/3 标准解决方案适应特殊问题非常重要。ABAP/4 开发工作平台包含所有用于创建和维护 ABAP/4 程序的工具。
ABAP/4 的可解释特性使其易于生成、测试并运行程序的中级版本，以便将来生成最终版本。此过程也叫做早期原形处理，意味着您不必丢弃中级版本。
ABAP/4 是第四代支持结构化程序设计的语言。它合并了所有通常的控制结构和模块化概念。
在 ABAP/4 用户指南的第一部分，对 ABAP/4 的基本特性进行了详细描述。这些特性包括：

- ABAP/4 包括
  - 带有各种类型和结构的声明数据的声明元素
  - 数据制作的操作元素
  - 控制程序流的控制元素
  - 反应外部事件的事件元素
- ABAP/4 支持多种语言。文本摘要（例如，标题、页眉和其他文本）将根据程序代码分别存储。您可以随时在不改变程序代码的情况下更改、转换和维护这些文本摘要。
- ABAP/4 支持商业数据类型和操作。您可以用特殊日期和时间字段进行计算。系统会自动执行必需的类型转换。
- ABAP/4 提供一系列功能处理字符串。
- ABAP/4 包含一个叫 Open SQL 的 SQL 子集。用 Open SQL，您可以读取和访问数据库表，与所用的数据库系统无关。
- ABAP/4 允许您定义和处理内部表，该表只在运行程序时存在。内部表使其更容易处理数据表，并帮助使用程序中的复杂数据结构。
- ABAP/4 允许您定义和调用子程序。也可以调用其他程序的子程序。参数能够以各种方式从子程序传递或传递到子程序。
- ABAP/4 包含一种特殊子程序，叫做功能模块。您可以在中央库中创建和维护功能模块。在调用程序和子程序之间功能模块有一个明确定义的数据接口。它们能够以调用程序的独立模式进行分别测试。

这里有两种主要的 ABAP/4 程序类型：

- 报表程序
  报表程序用于分析数据库表中的数据。这种分析的结果可以显示在屏幕上或发送到打印机上。逻辑数据库支持报表程序。逻辑数据库是特殊的 ABAP/4 程序，使开发者不必编

---

bc01e:0
ABAP/4 简
bc01e.rtf001
ABAP/4, 简介;ABAP/4, 用户指南概述;用户指南, 概述;
overview

码所有的数据库访问。关于报表程序的详细信息，参见《ABAP/4 用户指南》的第二部分。

- 对话程序
  将对话程序组织为包含对话模块的模块池。每个动态程序（由一个屏幕及其流逻辑组成的“动态程序”）都基于一个 ABAP/4 对话程序。流逻辑包含对 ABAP/4 对话模块的调用。关于对话程序的详细信息，参见《ABAP/4 用户指南》的第三部分。

语法约定

本文档中涉及的语法语句约定如下：

关键字	定义
语句	关键字和语句选项大写。
<变量>	变量或代表所填入值的词用尖括号括起来。在所使用的值中不要包括角括号（例外：字段符号）
[]	方括号表明用户没有使用、使用一个或多个附加选项。在用户选项中不要包括方括号。
|	两选项之间的竖条表明用户可以使用一个或另一个选项。
()	应将括号作为命令的一部分键入
,	逗号表示用户可选择多个选项，用逗号隔开，作为命令的一部分键入。
<f_1> <f_2>	带索引的变量表明用户可列出多个变量。变量之间要采用与前两个一样的符号隔开。
.....	点号表明用户可在此处放置上下文中允许的任何东西。

在语法指令中，关键字大写，变量放进角括号中。在程序中键入关键字时可以忽略大小写。**WRITE**、**Write** 和 **write** 完全一样。
输出屏幕上的输出或者采用屏幕弹出形式或者采用如下格式：

```
Screen output.
```

第一部分　ABAP/4 基础

目　　录

第 1 章　创建简单的 ABAP/4 程序

概览

内容

本节描述如何创建简单的 ABAP/4 程序。了解如何创建 ABAP/4 程序将有助于了解本指南中的其它主题。
创建简单的 ABAP/4 程序涉及下列基本步骤：
1. 命名程序
2. 指定程序属性
3. 编写程序代码
4. 测试程序

本节也描述如何显示或更改现有程序以及如何从编辑器中启动程序。
此处描述的创建新 ABAP/4 程序的过程适用于报表和短培训程序。在开始编写报表程序之前，用户也许想先创建报表和短培训程序以熟悉 ABAP/4 语法。要为新事务创建模块存储，可以采用不同于报表程序的方式进行。关于如何为事务创建模块存储的详细信息，参见 ABAP/4 模块池。
关于 ABAP/4 编辑器和调试过程的详细信息，参见文档 *ABAP/4 工作台工具*（页 **Error! Not a valid link.**）。
本节讲述下列主题：

命名程序

要创建 ABAP/4 程序，请进行如下操作：

1. 在“SAP R/3”初始屏幕上选择“工具 -> ABAP/4 工作台”。
   出现“ABAP/4 开发工作台”屏幕
2. 选择“ABAP/4 编辑器”
   “ABAP/4 编辑器初始屏幕”如下所示：
3. 为在“程序”字段中创建的程序输入名称（关于创建程序名称的详细信息，参见 *命名程序规 则*（页 457））。

4. 选择“创建”。

不论在“对象组件”下选择什么，都出现“ABAP/4：程序属性”屏幕。

当命名并创建程序后，可以定义其属性（关于定义程序属性的详细信息，参见 *指定程序属性*（页 459））。

> 创建 ABAP/4 程序还有其它过程。例如，可以：
>
> 1. 选择“ABAP/4 开发工作台”屏幕上的“对象浏览”。
> 2. 选择“对象列表”下的“程序”。
> 3. 选择“单一对象”下的“程序对象”。
> 4. 输入程序名并单击“显示”。
>
> 如果程序不存在，则询问是否要创建它。关于对象浏览器及创建程序其它过程的详细信息，参见文档 *ABAP/4 工作台工具*（页 **Error! Not a valid link.**）。

命名程序规则

当创建程序名称时请遵循如下规则：

- 使用至少 1 个但不超过 8 个字符。
- 不要使用下列字符
  - 句点（.）
  - 逗号（，）
  - 空格（ ）
  - 括号 ’（’ ’）’
  - 单引号（’）
  - 双引号（″）
  - 等号（=）
  - 星号（*）
  - 元音变音（_，_，_，_，_，_）和 ’_’
  - 百分号（%）和下划线（_）：因为这些符号是 SQL 语句的通配符，所以也会导致问题（参见 为在程序中选定行指定条件）。SAP 建议在程序名称中不要使用它们。

创建程序名称时请遵守这些命名约定：

- 报表程序（以列表格式输出数据分析）：**Yaxxxxxx** 或 **Zaxxxxxx**。用应用程序区的分类字母替换 **a**。用任何有效字符替换 **x** 。注意 SAP 报表程序遵守相似的命名约定：**Raxxxxxx**。
- 任何其它 ABAP/4 程序（培训程序或事务程序）： **SAPMYxxx** 或 **SAPMZxxx**。用有效字符替换 **x** 。注意标准 SAP ABAP/4 程序遵守相似的命名约定：**SAPMaxxx**，其中 **a** 代表某应用程序区。

指定程序属性

程序属性决定程序属于哪种应用程序以及程序所链接的逻辑数据库。必须谨慎输入属性以便系统能正确处理程序（有关程序属性的详细信息，参见 *重要的程序 属性*（页 461））。已经将名称分配给程序并选择“ABAP/4 编辑器初始屏幕”上的“创建”时，出现“ABAP/4：程序属性”屏幕。

要输入程序属性，请进行如下操作：

1. 在字段“标题”中输入程序标题。选择描述程序功能的标题。系统自动将标题与文本摘要合并。如果以后要更改标题，请按如下操作进行：
   - 选择“ABAP/4 编辑器初始屏幕”上的“文本摘要”或“属性”。

- 选择“更改”。

2. 完成两个强制字段：
   - 如果创建报表程序，则在“类型”字段中输入 **1**，如果创建模块存储，则在“类型”字段中输入 **M**。关于可能类型的列表，请单击可能条目箭头。
   - 在“应用程序”字段中为应用程序输入分类字母，如财务会计输入 **F** 。
3. 如果创建报表（类型 = 1），请选择“确定”。
   系统将特定报表属性自动插入输入字段。然后看见附加字段“逻辑数据库”、“从应用程序”和“选择屏幕”（版本）。

4. 指定与程序相关的所有其他属性（有关程序属性的详细信息，参见 *重要的程序 属性* (页 461) ）。
5. 选择“保存”以保存属性。
6. 出现“维护对象目录条目”窗口。
7. 输入开发类。
   如果为培训或测试目的创建程序（如某私有对象），则输入开发类 $TMP 或选择“逻辑对象”。
8. 选择“保存”以保存开发类。
   这将关闭“维护对象目录条目”窗口并返回“ABAP/4：程序属性”屏幕。
9. 选择“后退”离开屏幕。如果要直接转到 ABAP/4 编辑器，请选择“转向 -> 源代码”。

> 如果通过预定义开发类在“对象浏览器”中创建程序，则跳过步骤 6 到 8。

重要的程序属性

最重要的程序如下所述。关于其它属性的详细信息（或关于此处描述的属性的详细信息），请选择相关的输入字段，并单击可能的条目箭头。
注意下述某些属性只适用于报表程序并不适用于其他 ABAP/4 程序。

类型

除了类型 1（如同报表的独立程序）和 M（模块存储），还应该注意类型 I（包含程序）。包含程序是个独立的程序，它有两个主要特征。首先，它包含程序代码，不同程序都可使用该代码。其次，它用于模块化程序源代码，该代码分成逻辑相关部分。其中每个部分都存储在不同的包含程序中。包含程序改善源代码的可读性并有助于维护。(有关包含程序的详细信息，参见 包含程序)。

应用程序

“应用程序”字段包括应用程序的缩写，如，财务会计缩写为 **F** 。该必需条目使系统能将程序分配给适当的业务区。

开发类

开发类对系统之间进行传输非常重要。执行传输时，可以将分配给某个开发类的工作台对象组合起来。
如果用户在某组中工作，也许要将程序分配给现有开发类或创建新的开发类。分配给开发类 $TMP 的程序是私有对象并且不能传输到其他系统。
但是，也可以通过选择“ABAP/4 编辑器初始屏幕”上的“程序 -> 重新分配...”更改分配给程序的开发类。

应用程序中的逻辑数据库（仅报表程序）

这些属性决定报表使用哪个逻辑数据库检索数据，以及逻辑数据库属于哪个应用程序。应用程序中数据库名称必须是唯一的。但是，整个系统可以包括数个同名数据库。因此指定应用程序很重要。关于逻辑数据库的详细信息，参见 逻辑数据库的特性和维护。
如果报表程序直接读取数据，而未使用逻辑数据库（关于此主题的详细信息，参见 访问带逻辑数据库的数据库表），则应该象通常一样指定应用程序，但将“逻辑数据库”字段置空。

选择屏幕版本（仅报表程序）
如果没有指定选择屏幕版本，则系统在逻辑数据库选择标准、报表特定参数及选择选项的基础上自动创建选择屏幕（关于选择屏幕的详细信息，参见 使用选择屏幕）。
如果想使用自己的选择屏幕，请在此字段中输入号码。该号码必须小于 1000。可以通过按 F4 或在 DBxxxSEL 程序中找到现有选择屏幕号码（详细信息，参见 逻辑数据库选择）。
大写/小写
显示和存储时，如果要让 ABAP/4 编辑器将程序代码与输入时保持相同，则请将该字段置空。如果选择该字段，则所有的程序代码（除了引号中的文本及注释外）都转换成大写字母。屏幕显示依赖于所使用的编辑器模式（详细信息，参见文档 *ABAP/4 工作台工具*（页 Error! Not a valid link.）。
编辑器锁定
如果设置此属性，则其他用户则不能修改、改名或删除程序。只有用户可以更改程序、维护属性、文本摘要及文档；或者解除锁定。
定点算术
如果设置此属性，系统则根据小数点位数将类型 P 字段四舍五入，或用零填补（关于类型 P 字段的详细信息，参见*数字数据类 型*（页 3－4））。这种情况下，无论在用户主记录中指定了什么，小数点字符总是句点（.）。
通过变式启动（仅报表程序）
如果设置该属性，则用户只可以使用变式启动报表程序。在启动程序之前，必须创建至少一个报表变式（关于创建变式的详细信息，参见 使用变式预设置选择）。

编写程序

在 ABAP/4 编辑器中编写 ABAP/4 程序。
可以直接从“ABAP/4 程序属性”屏幕切换到编辑器，也可以从“ABAP/4 编辑器初始屏幕”中调用编辑器。
请进行如下操作：

1. 在“ABAP/4 程序属性”屏幕上选择“转向 -> 源代码”或“源代码”。
   选定“源代码”并选择“ABAP/4 编辑器初始屏幕”上的“更改”。
   出现“ABAP/4 编辑器编辑程序”屏幕。
   系统自动输入第一个 ABAP/4 语句，如
   REPORT <report name> or **PROGRAM <program name>.**
   对于 <report/program name>，系统使用在“ABAP/4 编辑器初始屏幕”上输入的名称。

   > 语句 REPORT 和 PROGRAM 实际上具有相同功能。它们使系统能识别报表程序或任何其他 ABAP/4 程序并允许为输出列表指定一定的标准：
   >
   > REPORT 或 PROGRAM 语句可以有不同的参数，如 LINE-SIZE、LINE-COUNT 或 NO STANDARD PAGE HEADING。这些参数主要适用于报表程序，用来分析数据并输出结果列表。
   >
   > 关于列表的详细信息，参见 创建列表 或参见 REPORT 和 PROGRAM 上的关键字文档。

2. 输入程序代码。
3. 选择“检查”进行语法检查。然后系统扫描程序代码寻找语法错误及不相容处。如果检查出错误，则出现消息报告它并且有可能的话将建议解决方案或更正。将光标放在适当的位置上。
4. 选择“保存”保存代码。
   源文本存储在程序库中。

测试程序

测试程序检查代码是否正确工作。
为测试目的要运行程序，选择“ABAP/4 编辑器编辑程序”屏幕上的“程序 -> 执行”。
系统执行该程序，如同从“ABAP/4 编辑器初始屏幕”中启动的一样。例如，创建报表程序之后，则首先出现从中输入条件的选择屏幕，然后是结果列表。
如果正创建 ABAP/4 程序模块，则必须在运行程序之前创建事务代码（关于创建事务代码的详细信息，参见 对话编程简介）。

> 出于测试目的，可以先不进行保存，直接运行某个不是模块存储部件的程序。编辑器保留一个包含更改的临时版本。但是，测试结束后必须返回编辑器以确保所有更改都已保存。

显示或更改程序

本节主要说明如何显示或更改现有程序。此处相关的是技术步骤而不是实际代码更改。
要显示或更改程序，请进行如下操作：

1. 在“ABAP/4编辑器初始屏幕”上的“程序”字段中输入要更改的程序名称。
2. 选择“源代码”并选择“显示”或“更改”。
   如果选择“更改”则继续步骤 3。
   如果选择“显示”，则看到“ABAP/4编辑器显示程序”屏幕。此处也显示源代码，但不能进行更改。可以通过选择“显示 -> 更改”切换到更改模式。
3. 如果其他用户锁定该程序对编辑器的访问，则系统显示下列消息：
   User <name> forbade all changes.
   在这种情况下，不能更改程序。要更改程序，必须将它复制到新程序中并用新程序名再开始步骤 1。否则，继续步骤 5。
4. 输入对程序代码的更改。
5. 选择“检查”检查语法。
6. 保存程序的更改版本。
   如果更改程序的私有版本（开发类 $TMP），则立即保存更改版本。
   如果从 $TMP 之外的开发类更改程序，则会出现一个窗口，从中可以指定更正号。
7. 运行程序测试它（关于测试的详细信息，参见 *测试程序* (页 11) ）。

将事务代码分配给程序

可以将事务代码分配给类型 1 的单独程序，以后就将这些程序作为事务代码处理。在编写 ABAP/4 事务 中对事务有说明。
进行如下操作：

1. 按照本节上述主题所述，创建单独程序。
2. 在“ABAP/4 开发工作台”屏幕上选择“开发 -> 其它工具 -> 事务”。
3. 填入事务名称并选择“维护事务”屏幕上的“创建”
4. 在下列对话屏幕上选择“报表事务”
5. 在下列“创建报表事务”屏幕上填入所需条目“事务文本”和“程序”。
6. 将事务代码保存在开发类中。

第二章 ABAP/4 程序语法和格式

概览

内容

本节说明 ABAP/4 语法，并且提供关于如何在 ABAP/4 中编程的建议。同时解释如何提高程序的清楚性，以及使用已有程序代码模块使编程变得更加容易。本节的主题包括：

本节只是概述。关于单个 ABAP/4 组件的详细信息，参见本指南的相应主题。

语法元素

ABAP/4 编程语言包括下列元素类型：

语句

ABAP/4 程序包括单个 ABAP/4 语句。每条语句以关键字开头，以句号结束。

```
PROGRAM SAPMTEST.
WRITE 'First Program'.
```

该示例包含两条语句，每行一条。关键字是 PROGRAM 和 WRITE。程序在屏幕上显示输出（称为列表）。此示例中，列表包括“第一个程序”行。

关键字

关键字是语句的第一个词。它决定整个语句的意义。有四种不同类型的关键字：

说明性关键字

这些关键字定义数据类型，或者说明程序可以访问的数据对象。说明性关键字示例：

TYPES, DATA, TABLES

系统在生成程序期间处理说明性关键字，而不是在运行时。在程序代码中独立于其位置处理它们。为清楚起见，应该在程序开头的“说明部分”指定所有说明性关键字。

关于说明关键字的详细信息，参见*声明数据*（页 **错误！链接无效。**）。

事件关键字

这些关键字在 ABAP/4 程序中定义处理块。处理块是当特定事件发生时进行处理的语句组。事件关键字示例如下：

AT SELECTION SCREEN, START-OF-SELECTION, AT USER-COMMAND

关于事件关键字的详细信息，参见用事件控制 ABAP/4 程序流。

控制关键字

这些关键字根据特定条件控制 ABAP/4 程序流。控制关键字示例如下：

IF, WHILE, CASE

关于控制关键字的详细信息，参见*控制 ABAP/4 程序流*（页 **错误！链接无效。**）。

操作关键字

当某事件（由事件关键字触发）和条件（由控制关键字定义）发生时，这些关键字则处理数据（由说明性关键字定义）。操作关键字示例如下：

WRITE, MOVE, ADD

关于控制关键字的详细信息，参见*处理数据*（页 **错误！链接无效。**）。

注释

注释是写在 ABAP/4 程序语句之间用来向读者解释其目的的文本摘要。注释由导致系统忽略它们的特殊字符标记。应该使用注释在内部证明程序。注释帮助其他用户理解和更改程序。

```
*********************************************
*  PROGRAM SAPMTZST                         *
*  CREATED BY CARL BYTE, 06/27/1995         *
*  LAST CHANGE BY RITA DIGIT, 10/01/1995    *
*  PURPOSE: DEMONSTRATION                   *
*********************************************
PROGRAM SAPMTEST.
*********************************************
* DECLARATION PART                          *
*********************************************
DATA ......
......
```

```
**********************************************
* OPERATION PART                              *
**********************************************
......
```

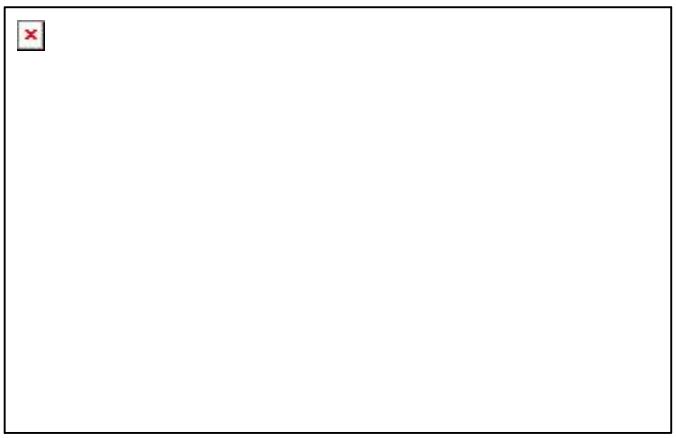
所有以星号（*）开始的行都是注释，并且被系统忽略（关于注释的详细信息，参见*注释结构*（页 399））。

语法结构

ABAP/4 程序是具有特定结构的不同语句的顺序。
可以在语句间插入注释。
分开的但相似的语句顺序可以组成链语句。

语句结构

下列图表显示 ABAP/4 语句的结构。



ABAP/4 没有格式限制。可以自由格式输入语句。这意味着可以缩排语句、在一行中写几条语句或者一条语句跨越几行。
在语句中必须以至少一个空格分开词。系统也将行结束符解释为空格。

程序段

```
PROGRAM SAPMZTST.
WRITE 'This is a statement'.
```

也可以如下编写：

```
PROGRAM SAPMTEST. WRITE 'This is a statement'.
```

或者如下：

```
              PROGRAM
              SAPMTEST.
                        WRITE
                       'This is
                        a statement'.
```

应该使用自由格式提高程序的可读性，但是应避免使用复杂格式。

注释结构

可以在程序的任意处插入注释行。在程序中有两种方法表明注释：

如果要将整行变为注释，则在行开始处输入星号（*）。

如果要将某行的一部分变为注释，则在注释之前输入双引号（"）。系统将由双引号标明的注释解释为空格。

```
PROGRAM SAPMTEST.
* The following line contains a WRITE statement
WRITE 'First Program'. " Output on List
```

本程序的第二行是不执行的注释。注释由行开始处的星号（*）标明。

在第三行，双引号（"）之后全部都是注释并且不执行。

程序的其余部分为带关键字 PROGRAM 和 WRITE 的可执行语句。

连接相似语句

ABAP/4 编程语言允许将带相同起始部分的连续语句连入链语句。

要连接分开的语句，只要写一次相同部分，并且在其后设置冒号（:）。在冒号后，列出语句的其余部分并用逗号(,)分开。请保证在最后部分之后设置句号以通知系统链的结束处。

语句顺序：

```
WRITE SPFLI-CITYFROM.
WRITE SPFLI-CITYTO.
WRITE SPFLI-AIRPTO.
```

链语句：

```
WRITE: SPFLI-CITYFROM, SPFLI-CITYTO, SPFLI-AIRPTO.
```

在链中，冒号将语句的开始部分与可变部分分开。可以在冒号（或逗号）之前或之后插入任意个空格。

例如，可以写如下相同语句：

```
WRITE:    SPFLI-CITYFROM,
          SPFLI-CITYTO,
          SPFLI-AIRPTO.
```

在链语句中，第一部分（冒号之前）不受语句关键字的限制。

语句顺序：

```
SUM = SUM + 1.
SUM = SUM + 2.
SUM = SUM + 3.
SUM = SUM + 4.
```

链语句：

```
SUM = SUM + : 1, 2, 3, 4.
```

ABAP/4 程序格式

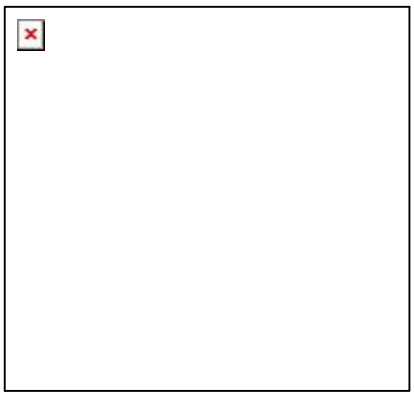
要编写高质量的程序，不仅应该遵循命名约定（参见*命名程序规则*（页 **错误!链接无效。**）），而且要保持一定的 ABAP/4 程序格式标准。
一开始定义数据就应该注意这些标准。在构建程序流时注意下列主题中的建议，并且尽可能地使用信息注释。如果遵照这些建议，程序将

- 更具有可读性
- 更容易测试和更改
- 更可靠

要提高程序的质量，请使用下列技术：
ABAP/4 编辑器包括更容易分配程序格式的工具。该工具称为整齐打印程序。

缩排语句块

应该将属于一组的语句组成单一块。关于语句块的详细信息，参见 *控制 ABAP/4 程序流*（页 **错误！链接无效。**）。
每块至少缩排两列，如下所示。



使用模块化工具

要编出好的程序，应该使用 ABAP/4 模块化工具（参见 模块化 ABAP/4 程序）。
如果将大处理块写成子程序，则程序的逻辑结构容易识别。允许根据执行的任务排序子程序。
子程序可能增加程序的总长度，但是您将发现该方法大大增加清晰性，特别是在复杂程序的情况下。如果按系统执行的顺序排列子程序，则程序代码很容易读。

正确插入程序注释

应该避免在语句行中放置注释。在注释行放置它们将提高程序的可读性。
要在程序中插入子程序标题和注释，请使用 ABAP/4 编辑器可用的已有结构。在子程序标题中，解释调用程序的目的并且提供足够的信息和参考。关于已有结构的详细信息，参见 *插入已有关 键字结构*（页 9）和在 *ABAP/4 工作台工具*（页 Error! Not a valid link.）中的 ABAP/4 编辑器文档。

整齐打印程序

使用整齐打印程序可以更容易地遵循 ABAP/4 格式指南。
整齐打印程序是 ABAP/4 编辑器的可选功能（有关详细信息，参见 *ABAP/4 工作台工具*（页 Error! Not a valid link.）中的 ABAP/4 编辑器文档）。
要从 ABAP/4 编辑器调用整齐打印程序，请选择“程序 -> 整齐打印程序”。

这里是整齐打印程序如何工作的示例。

下面显示在使用整齐打印程序之前程序的格式：

```
PROGRAM SAPMTEST.
                DATA: SUM1 TYPE I, SUM2 TYPE I, SUM3 TYPE I.
             IF  SUM1 = SUM2.
       WRITE 'Case 1'.
            ELSEIF SUM1 = SUM3.   WRITE 'Case 2'.
                                                    ENDIF.
```

使用整齐打印程序之后，相同程序如下所示：

```
PROGRAM SAPMTEST.
DATA: SUM1 TYPE I, SUM2 TYPE I, SUM3 TYPE I.
IF  SUM1 = SUM2.
  WRITE 'Case 1'.
ELSEIF SUM1 = SUM3.
  WRITE 'Case 2'.
ENDIF.
```

插入已有结构

已有结构可以简化 ABAP/4 程序的编码。它提供确切的语法，并且遵循 ABAP/4 格式指南。使用 ABAP/4 编辑器时可以在程序中插入两种已有结构：

关于已有结构的详细信息，参见 *ABAP/4 工作台工具*（页 Error! Not a valid link.）中的 ABAP/4 编辑器文档。

插入已有关键字结构

要在代码中插入已有关键字结构，请进行下列操作：

1. 将光标放在要插入结构的行上。
2. 选择“退出 -> 插入语句”，或者选定“模式”。
3. 在出现的对话框中，选择带单选按钮的语句，或者在“其它结构”字段中输入：

要显示所有可用的已有关键字结构的列表，请将光标放在“其它结构”字段上，并且单击输入字段右侧的可能条目箭头。所有首字符为星号（*）的语句都是已有注释行（关于已有注释行的详细信息，参见 *插入已有注 释行*（页 11））。

如果在对话框的“其它结构”字段中输入 CASE，则系统在程序中插入如下行：

```
CASE f.
  WHEN w1.
    ......
  WHEN w2.
    ......
  WHEN OTHERS.
```

```
    ......
ENDCASE.
```

插入已有注释行

要在代码中插入已有注释行，请执行下列操作：

1. 遵照 *插入已有关 键字结构* (页 17) 中的 1 到 2 步骤。
2. 在对话框的“其他结构”字段的可能条目列表中选择带首字符为星号（*）的结构。
3. 系统将注释行插入到程序中。

如果在对话框的“其他结构”字段中输入 **3，则系统在程序中插入如下行：

```
**********************************************************
*                                                        *
*                                                        *
*                                                        *
**********************************************************
```

第三章 ABAP/4 程序语法和格式

概览

内容

本节说明 ABAP/4 语法，并且提供关于如何在 ABAP/4 中编程的建议。同时解释如何提高程序的清楚性，以及使用已有程序代码模块使编程变得更加容易。本节的主题包括：

本节只是概述。关于单个 ABAP/4 组件的详细信息，参见本指南的相应主题。

语法元素

ABAP/4 编程语言包括下列元素类型：

语句

ABAP/4 程序包括单个 ABAP/4 语句。每条语句以关键字开头，以句号结束。

```
PROGRAM SAPMTEST.
WRITE 'First Program'.
```

该示例包含两条语句，每行一条。关键字是 PROGRAM 和 WRITE。程序在屏幕上显示输出（称为列表）。此示例中，列表包括“第一个程序”行。

关键字

关键字是语句的第一个词。它决定整个语句的意义。有四种不同类型的关键字：

说明性关键字

这些关键字定义数据类型，或者说明程序可以访问的数据对象。说明性关键字示例：

TYPES, DATA, TABLES

系统在生成程序期间处理说明性关键字，而不是在运行时。在程序代码中独立于其位置处理它们。为清楚起见，应该在程序开头的“说明部分”指定所有说明性关键字。

关于说明关键字的详细信息，参见*声明数据*（页 **错误！链接无效。**）。

事件关键字

这些关键字在 ABAP/4 程序中定义处理块。处理块是当特定事件发生时进行处理的语句组。事件关键字示例如下：

AT SELECTION SCREEN, START-OF-SELECTION, AT USER-COMMAND

关于事件关键字的详细信息，参见用事件控制 ABAP/4 程序流。

控制关键字

这些关键字根据特定条件控制 ABAP/4 程序流。控制关键字示例如下：

IF, WHILE, CASE

关于控制关键字的详细信息，参见*控制 ABAP/4 程序流*（页 **错误！链接无效。**）。

操作关键字

当某事件（由事件关键字触发）和条件（由控制关键字定义）发生时，这些关键字则处理数据（由说明性关键字定义）。操作关键字示例如下：

WRITE, MOVE, ADD

关于控制关键字的详细信息，参见*处理数据*（页 **错误！链接无效。**）。

注释

注释是写在 ABAP/4 程序语句之间用来向读者解释其目的的文本摘要。注释由导致系统忽略它们的特殊字符标记。应该使用注释在内部证明程序。注释帮助其他用户理解和更改程序。

```
*********************************************
*  PROGRAM SAPMTZST                         *
*  CREATED BY CARL BYTE, 06/27/1995         *
*  LAST CHANGE BY RITA DIGIT, 10/01/1995    *
*  PURPOSE: DEMONSTRATION                   *
*********************************************
PROGRAM SAPMTEST.
*********************************************
* DECLARATION PART                          *
*********************************************
DATA ......
......
```

```
*********************************************
* OPERATION PART                            *
*********************************************
......
```

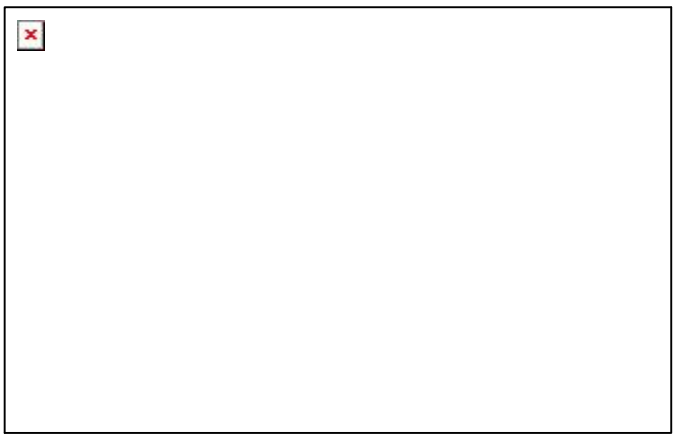
所有以星号（*）开始的行都是注释，并且被系统忽略（关于注释的详细信息，参见*注释结构*（页 15））。

语法结构

ABAP/4 程序是具有特定结构的不同语句的顺序。
可以在语句间插入注释。
分开的但相似的语句顺序可以组成链语句。

语句结构

下列图表显示 ABAP/4 语句的结构。



ABAP/4 没有格式限制。可以自由格式输入语句。这意味着可以缩排语句、在一行中写几条语句或者一条语句跨越几行。
在语句中必须以至少一个空格分开词。系统也将行结束符解释为空格。

程序段

```
PROGRAM SAPMZTST.
WRITE 'This is a statement'.
```

也可以如下编写：

```
PROGRAM SAPMTEST. WRITE 'This is a statement'.
```

或者如下：

```
              PROGRAM
              SAPMTEST.
                        WRITE
                       'This is
                        a statement'.
```

应该使用自由格式提高程序的可读性，但是应避免使用复杂格式。

注释结构

可以在程序的任意处插入注释行。在程序中有两种方法表明注释：

如果要将整行变为注释，则在行开始处输入星号（*）。

如果要将某行的一部分变为注释，则在注释之前输入双引号（"）。系统将由双引号标明的注释解释为空格。

```
PROGRAM SAPMTEST.
* The following line contains a WRITE statement
WRITE 'First Program'. " Output on List
```

本程序的第二行是不执行的注释。注释由行开始处的星号（*）标明。

在第三行，双引号（"）之后全部都是注释并且不执行。

程序的其余部分为带关键字 PROGRAM 和 WRITE 的可执行语句。

连接相似语句

ABAP/4 编程语言允许将带相同起始部分的连续语句连入链语句。

要连接分开的语句，只要写一次相同部分，并且在其后设置冒号（:）。在冒号后，列出语句的其余部分并用逗号(,)分开。请保证在最后部分之后设置句号以通知系统链的结束处。

语句顺序：

```
WRITE SPFLI-CITYFROM.
WRITE SPFLI-CITYTO.
WRITE SPFLI-AIRPTO.
```

链语句：

```
WRITE: SPFLI-CITYFROM, SPFLI-CITYTO, SPFLI-AIRPTO.
```

在链中，冒号将语句的开始部分与可变部分分开。可以在冒号（或逗号）之前或之后插入任意个空格。

例如，可以写如下相同语句：

```
WRITE:    SPFLI-CITYFROM,
          SPFLI-CITYTO,
          SPFLI-AIRPTO.
```

在链语句中，第一部分（冒号之前）不受语句关键字的限制。

语句顺序：

```
SUM = SUM + 1.
SUM = SUM + 2.
SUM = SUM + 3.
SUM = SUM + 4.
```

链语句：

```
SUM = SUM + : 1, 2, 3, 4.
```

ABAP/4 程序格式

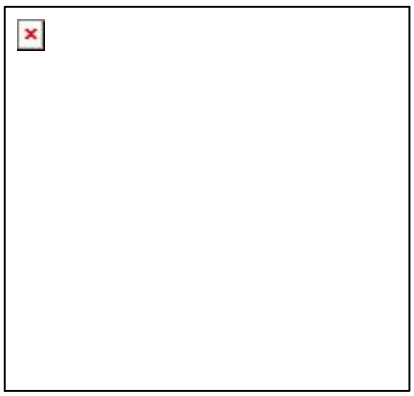
要编写高质量的程序，不仅应该遵循命名约定（参见*命名程序规则*（页 **错误!链接无效。**）），而且要保持一定的 ABAP/4 程序格式标准。
一开始定义数据就应该注意这些标准。在构建程序流时注意下列主题中的建议，并且尽可能地使用信息注释。如果遵照这些建议，程序将

- 更具有可读性
- 更容易测试和更改
- 更可靠

要提高程序的质量，请使用下列技术：
ABAP/4 编辑器包括更容易分配程序格式的工具。该工具称为整齐打印程序。

缩排语句块

应该将属于一组的语句组成单一块。关于语句块的详细信息，参见 *控制 ABAP/4 程序流*（页 **错误！链接无效。**）。
每块至少缩排两列，如下所示。



使用模块化工具

要编出好的程序，应该使用 ABAP/4 模块化工具（参见 模块化 ABAP/4 程序）。
如果将大处理块写成子程序，则程序的逻辑结构容易识别。允许根据执行的任务排序子程序。
子程序可能增加程序的总长度，但是您将发现该方法大大增加清晰性，特别是在复杂程序的情况下。如果按系统执行的顺序排列子程序，则程序代码很容易读。

正确插入程序注释

应该避免在语句行中放置注释。在注释行放置它们将提高程序的可读性。
要在程序中插入子程序标题和注释，请使用 ABAP/4 编辑器可用的已有结构。在子程序标题中，解释调用程序的目的并且提供足够的信息和参考。关于已有结构的详细信息，参见 *插入已有关 键字结构*（页 17）和在 *ABAP/4 工作台工具*（页 Error! Not a valid link.）中的 ABAP/4 编辑器文档。

整齐打印程序

使用整齐打印程序可以更容易地遵循 ABAP/4 格式指南。
整齐打印程序是 ABAP/4 编辑器的可选功能（有关详细信息，参见 *ABAP/4 工作台工具*（页 Error! Not a valid link.）中的 ABAP/4 编辑器文档）。
要从 ABAP/4 编辑器调用整齐打印程序，请选择“程序 -> 整齐打印程序”。

这里是整齐打印程序如何工作的示例。

下面显示在使用整齐打印程序之前程序的格式：

```
PROGRAM SAPMTEST.
                DATA: SUM1 TYPE I, SUM2 TYPE I, SUM3 TYPE I.
             IF  SUM1 = SUM2.
       WRITE 'Case 1'.
            ELSEIF SUM1 = SUM3.    WRITE 'Case 2'.
                                                    ENDIF.
```

使用整齐打印程序之后，相同程序如下所示：

```
PROGRAM SAPMTEST.
DATA: SUM1 TYPE I, SUM2 TYPE I, SUM3 TYPE I.
IF  SUM1 = SUM2.
  WRITE 'Case 1'.
ELSEIF SUM1 = SUM3.
  WRITE 'Case 2'.
ENDIF.
```

插入已有结构

已有结构可以简化 ABAP/4 程序的编码。它提供确切的语法，并且遵循 ABAP/4 格式指南。使用 ABAP/4 编辑器时可以在程序中插入两种已有结构：

关于已有结构的详细信息，参见 *ABAP/4 工作台工具*（页 Error! Not a valid link.）中的 ABAP/4 编辑器文档。

插入已有关键字结构

要在代码中插入已有关键字结构，请进行下列操作：

1. 将光标放在要插入结构的行上。
2. 选择“退出 -> 插入语句”，或者选定“模式”。
3. 在出现的对话框中，选择带单选按钮的语句，或者在“其它结构”字段中输入：

要显示所有可用的已有关键字结构的列表，请将光标放在“其它结构”字段上，并且单击输入字段右侧的可能条目箭头。所有首字符为星号（*）的语句都是已有注释行（关于已有注释行的详细信息，参见 *插入已有注 释行*（页 18））。

如果在对话框的“其它结构”字段中输入 CASE，则系统在程序中插入如下行：

```
CASE f.
  WHEN w1.
    ......
  WHEN w2.
    ......
  WHEN OTHERS.
```

```
    ......
ENDCASE.
```

插入已有注释行

要在代码中插入已有注释行，请执行下列操作：

1. 遵照 *插入已有关 键字结构* (页 24) 中的 1 到 2 步骤。
2. 在对话框的“其他结构”字段的可能条目列表中选择带首字符为星号（*）的结构。
3. 系统将注释行插入到程序中。

如果在对话框的“其他结构”字段中输入 **3，则系统在程序中插入如下行：

```
**********************************************************
*                                                        *
*                                                        *
*                                                        *
**********************************************************
```

第四章 将数据输出到屏幕

概览

内容

本节说明如何在屏幕上创建简单输出列表。为此，请使用 WRITE 语句。
下面将介绍：
ABAP/4 允许在屏幕和纸张上，生成比这里看到的更复杂和更有效的输出列表。这些介绍是后面章节（例如，创建列表）的基础。

WRITE 语句

在屏幕上输出数据的基本 ABAP/4 语句是 WRITE。

语法
WRITE <f>.
该语句以其标准格式，将字段 <f> 输出到当前列表中。默认情况下，该列表显示在屏幕上。
字段 <f> 可以是

- 任何数据对象（参见 数据对象 （页 **错误！链接无效。**））
- 字段符号或公式参数（参见使用字符符号 (页使用字段符号)）
- 文本符号（参见 *Working with Text Elements*（页 **错误！链接无效。**））

选择“打印”，可以直接从输出屏幕打印当前输出列表。如果为程序定义选项屏幕（参见 使用选择屏幕），就可以在选择屏幕上选择“执行和打印”，然后，该列表不输出到屏幕上，而是直接发送到打印机。

```
PROGRAM SAPMZTST.
WRITE 'Hello, here I am!'.
```

当启动该程序时，系统离开当前屏幕（这可能是“ABAP/4 编辑器：初始屏幕”），并如下所示转到输出屏幕：

输出屏幕与在程序属性中指定的程序标题有相同的名称（参见*指定程序属性*（页 **错误！链接无效。**））。

屏幕上的第一行包含列表表头。默认情况下，该列表表头与程序标题相同。但是，可以由实际程序外的程序标题单独维护列表表头。详细信息，参见（*使用文本摘要*（页 **错误！链接无效。**））。当前页号（1）出现在右边。

列表表头后紧跟着一条水平线，然后显示输出。

可以选择“搜索”，以搜索特定模式。

在屏幕上，输出通常是左对齐的。如果使用几个 WRITE 语句，输出字段就一个接一个显示，输出之间由列分开（如一个空格）。如果当前行没有足够空间，则开始新行。

```
PROGRAM SAPMTEST.
TABLES SPFLI.
.............
WRITE: 'COMPANY: ', SPFLI-CARRID.
```

注意冒号和逗号的用法（参见 *语法结构*（页 **错误！链接无效。**））。

该示例中的程序段向屏幕输出两个字段：文字“COMPANY:”和表格工作区 SPFLI 的组件 CARRID：

COMPANY: AA。

输出屏幕上的数据字段格式，依赖数据类型（参见 *预定义的基本数据类型*（页 **错误！链接无效。**））。

予定义数据类型的输出格式

数据类型	输出长度	定位
C	字段长度	左对齐
D	8	左对齐
F	22	右对齐
I	11	右对齐
N	字段长度	左对齐

P	2 * 字段长度（＋1）	右对齐
T	6	左对齐
X	2 * 字段长度	左对齐

数字数据类型 F、I 和 P 是右对齐的，左边用空格填充。如果有足够的空间，也输出千位分隔符。如果类型 P 字段包含小数位，则默认输出长度增加一位。

对数据类型 D,日期的内部格式与输出格式不同。当输出数据使用 WRITE 语句时，系统自动以用户主记录中指定的格式（例如，DD/MM/YYYY 或 MM/DD/YYYY），输出数据类型 D。

```
PROGRAM SAPMTEST.
DATA NUMBER TYPE P VALUE '-1234567.89' DECIMALS 2.
WRITE: 'Number', NUMBER, 'is packed'.
```

输出如下：

```
Number      1,234,567.89- is packed
```

字段 NUMBER 总长为 13，即，9 位数字（包括小数点）、前导负号和作为分隔符的两个逗号。因为类型 P 字段的字段长度为 8,所以 NUMBER 字段的输出长度为 2*8+1=17。剩余的位置用四个空格填充。这意味着，在文字‘Number’和数字自身之间有五个空格。

在屏幕上定位 WRITE 输出

如下所示，通过制定字段名称前面的格式规范，可以在屏幕上定位 WRITE 语句的输出：

语法

```
WRITE AT [/][<pos>][(<len>)] <f>.
```

此处

斜线‘/’表示新的一行

<pos> 是最长为三位数字的数字或变量，表示在屏幕上的位置

<len> 是最长为三位数字的数字或变量，表示输出长度

如果格式规范只包含直接值（即，不是变量），可以忽略关键字 AT。

```
WRITE 'First line.'.
WRITE 'Still first line.'
WRITE / 'Second line.'
WRITE /13 'Third line.'
```

这在屏幕上生成如下输出：

```
First Line. Still first line.
Second line.
            Third line.
```

如果指定某一个位置 <pos>，则无论在该位置是否有可用的空间，或写有其它字段，总是在该位置输出字段。

```
DATA: LEN TYPE I VALUE 10,
      POS TYPE I VALUE 11,
      TEXT(10)   VALUE '1234567890'

WRITE 'The text ---------- appears in the text.'.

WRITE AT POS(LEN) TEXT.
```

这在屏幕上生成如下输出：

```
The text -1234567890- appears in the text.
```

如果输出长度 <len> 太短，则显示几个字符。左边截断数字字段，并用星号（*）作前缀。右边截断所有其它字段，但是没有给出该字段较短的指示。

```
DATA: NUMBER TYPE I VALUE  1234567890,
      TEXT(10)      VALUE 'abcdefghij'.

WRITE: (5) NUMBER, /(5) TEXT.
```

输出如下：

```
*7890
abcde
```

格式化选项

对 WRITE 语句，可以使用不同的格式化选项。

语法

WRITE .... <f> <选项>.

所有数据类型的格式化选项

选项	用途
LEFT-JUSTIFIED	输出左对齐。
CENTERED	输出居中。

RIGHT-JUSTIFIED	输出右对齐。
UNDER <g>	输出直接开始于字段 <g> 下。
NO-GAP	忽略字段 <f> 后的空格。
USING EDIT MASK <m>	指定格式模板 <m>。
USING NO EDIT MASK	撤消对 ABAP/4 词典中指定的格式模板的激活。
NO-ZERO	如果字段仅包含零，则用空格代替它们。对类型 C 和 N 字段，将自动代替前导零。

数字字段的格式化选项

选项	用途
NO-SIGN	不输出前导符号。
DECIMALS <d>	<d> 定义小数点后的数字位数。
EXPONENT <e>	在类型 F 字段中，在 <e> 中定义幂数。
ROUND <r>	用10**(-r) 乘类型P 字段，然后取整。
CURRENCY <c>	按表格 TCURX 中的货币 <c> 格式化。
UNIT <u>	按表格 T006 中为类型 P 字段所指定的单位 <u> 固定小数位数。

日期字段的格式化选项

选项	用途
DD/MM/YY	用户主记录中定义的分隔符
MM/DD/YY	用户主记录中定义的分隔符
DD/MM/YYYY	用户主记录中定义的分隔符
MM/DD/YYYY	用户主记录中定义的分隔符
DDMMYY	无分隔符。
MMDDYY	无分隔符。
YYMMDD	无分隔符。

关于格式选项和这些选项内例外原则的详细信息，参见 WRITE 语句的关键字文档。
以下是格式选项的一些示例。要获得更多的示例，参见章节创建列表。在用户主记录中，定义了数字字段的小数点字符和千位分隔符（分号或逗号）。

ABAP/4 代码	屏幕输出
DATA: G(5) VALUE 'Hello', F(5) VALUE 'Dolly'. WRITE: G, F. WRITE: /10 G,	Hello Dolly Hello

	/ F UNDER G. WRITE: / G NO-GAP, F.	Dolly HelloDolly
	DATA TIME TYPE T VALUE '154633'. WRITE: TIME, /(8) TIME USING EDIT MASK '__:__:__'.	154633 15:46:33
	WRITE: '000123', / '000123' NO-ZERO.	000123 123
	DATA FLOAT TYPE F VALUE '123456789.0'. WRITE FLOAT EXPONENT 3.	123456.789E+03
	DATA PACK TYPE P VALUE '123.456' DECIMALS 3. WRITE PACK DECIMALS 2. WRITE: / PACK ROUND -2, / PACK ROUND -1, / PACK ROUND 1, / PACK ROUND 2.	123.46 12,345.600 1,234.560 12.346 1.235
	WRITE: SY-DATUM, / SY-DATUM YYMMDD.	06/27/1995 950627

除了上表中显示的格式化选项外，还可以用 FORMAT 语句的格式化选项。这些选项允许指定输出的亮度和颜色。详细信息，参见FORMAT 语句。

在屏幕上输出符号和图标

使用下列语法，可以在屏幕上输出符号和 R/3 图标：

语法
```
WRITE <symbol-name> AS SYMBOL.
WRITE <icon-name>   AS ICON.
```
符号和图标的名称（<符号名> 和 <图标名>）是定义系统的常量，这些常量在包含程序 <SYMBOL> 和 <ICON>（尖括号是名称的一部分）中指定。这些包含程序也包含符号和图标的简短说明。输出符号和图标最简单的方法是使用语句结构（参见 *通过语句结 构使用 WRITE* (页 442) 中的示例）。
要使符号和图标对程序可用，必须在程序中输入恰当的包含程序或更易理解的包含程序 <LIST>。关于输入包含程序的详细信息，参见使用包含程序。

```
INCLUDE <SYMBOL>.
INCLUDE <ICON>.
WRITE: / 'Phone Symbol:', SYM_PHONE AS SYMBOL.
SKIP.
WRITE: / 'Alarm Icon: ', ICON_ALARM AS ICON.
```

输出如下：

上面两个 INCLUDE 语句可以用单个 INCLUDE 语句替代

```
INCLUDE <LIST>.
```

输出屏幕上的线和空行

用下列语法，可以在输出屏幕上生成水平线：

语法

```
ULINE [AT [/][<pos>][(<len>)]].
```

它等同于

```
WRITE [AT [/][<pos>][(<len>)]] SY-ULINE.
```

AT 后的格式规范，与在 *在屏幕上定 位 WRITE 输出* (页 28) 中为 WRITE 语句说明的格式规范完全一样。

如果没有格式规范，系统则开始新的一行，并用水平线填充该行。否则，只按指定输出水平线。

生成水平线的另一种方法，是在 WRITE 语句中键入恰当数量的连字符，如下所示：

```
WRITE [AT [/][<pos>][(<len>)]] '-----...'.
```

垂直线

用下列语法，可以在输出屏幕上生成垂直线：

语法

```
WRITE [AT [/][<pos>]] SY-VLINE.
```

或

```
WRITE [AT [/][<pos>]] '|'.
```

空行

用下列语法，可以在输出屏幕上生成空行：

语法

```
SKIP [<n>].
```

该语句从当前行开始，在输出屏幕上生成 <n> 个空行。如果没有指定 <n> 的值，就输出一个空行。

要将输出定位在屏幕的指定行上，请使用：

语法

```
SKIP TO LINE <n>.
```

该语句允许将输出位置向上或向下移动。
要获得更多的信息和示例，参见 创建列表。

将字段内容作为复选框输出

使用下列语法，可以将字段的第一个字符，作为复选框输出到输出屏幕上：

语法
```
WRITE <f> AS CHECKBOX.
```
如果字段〈f〉的第一个字符是一个“X”，就显示复选框已填充。如果字段〈f〉的第一个字符是 SPACE，就显示复选框为空。
该语句创建的复选框，默认状态是可输入的。就是说，用户可以通过单击鼠标来填充它们或使其为空。如何使输出字段能输入或撤消输入，将在 使字段接受输入 下加以说明。能输入字段在允许用户对话的交互式列表中是很重要的（参见交互列表）。

```
DATA: FLAG1    VALUE ' ',
      FLAG2    VALUE 'X',
      FLAG3(5) VALUE 'Xenon'.

WRITE: / 'Flag 1', FLAG1  AS CHECKBOX,
       / 'Flag 2', FLAG2  AS CHECKBOX,
       / 'Flag 3', FLAG3  AS CHECKBOX.
```

输出列表如下：

对 FLAG2 和 FLAG3，因为这些字段的第一个字符是“X”，所以填充复选框。通过单击鼠标，用户可以改变复选框的内容。

通过语句结构使用 WRITE

R/3 系统包含试验 WRITE 语句的所有选项和输出格式及将它们插入到程序中的有用工具。要做到这一点，在 ABAP/4- 编辑器中选择“编辑 -> 插入语句...”，然后在相关对话框中选择 WRITE（参见 *插入已有关键字结构*（页 **错误！链接无效。**））。

当用 *Enter* 确认选项后，看到如下屏幕：

在该屏幕上，可以

通过在字段“字段”中输入内部字段的名称或文字，确定其输出格式。然后通过选择“其它格式选项”，在该屏幕上或者在可访问的屏幕上选择格式选项。
通过选择合适的字段，生成符号、图标、线段和复选框的 WRITE 语句。
生成在 ABAP/4- 词典中定义的结构组件的 WRITE 语句。这很有用，比如，在执行 SELECT 语句后（参见从数据库表读取数据）。

在屏幕“装配 WRITE-语句”上，选择单选按钮“符号”和“显示”，随后将看到如下对话框：

在此，可以选择符号，例如 SYM_FOLDER。下一对话框在输出屏幕上显示相应的 WRITE 语句和结果输出。

同时显示注释，通知用户在程序中需要包含程序（参见 *在屏幕上输 出符号和图 标* (页 31) ）。

按下“继续”后，就会看到“装配 WRITE 语句”屏幕上的字段“符号”现在包含了一个值：

现在，如果选择“执行”，就会将下列文本插入到程序中：

```
WRITE SYM_FOLDER AS SYMBOL.
```

在“装配 WRITE 语句”屏幕上，选择单选按钮“结构”，并在恰当的输入字段中输入：

然后，选择“选择组件”。在下一屏幕上，可以选择希望用 WRITE 语句输出的 ABAP/4- 词典结构 SFLIGHT 的组件，例如：

如果采用该选择，就会将下面的 WRITE 语句插入到程序中：

```
WRITE: SFLIGHT-CARRID,
       SFLIGHT-CONNID,
       SFLIGHT-FLDATE,
       SFLIGHT-PRICE,
       SFLIGHT-PLANETYPE,
       SFLIGHT-SEATSOCC.
```

第五章 处理文本摘 要

概览

内容

ABAP/4 编程环境支 持在多种语 言中创建和 维护程序。 可以将程序 输出到屏幕 上的所有文 本，作为文 本池中的文 本摘要存储 起来。对不 同的语言， 可以创建自 己的文本池 。更改文本 时，就不必 改程序代码 ，而只须更 改恰当的文 本摘要。
本章将介绍 ： 图1 （略）

文本摘要— 概念

文本摘要包 括在选择屏 幕或 ABAP/4 程序的输出 屏幕上出现 的任何说明 性文本。
文本摘要包 含：

程序标题 。标题属于 程序属性
（ 参见 *指定程序属性*（页 **错误！链接无效。**））
输出列表 页眉的列表 标题和列标 题
（参见 创建列表）
选择屏幕 上出现的选 择文本
（参 见 使用选择选择屏幕）
可在 WRITE 语句中使用 的文本符号
（参见 *WRITE 语句*（页 **错误！链接无效。**））。

可以在语言 相关的文本 池中，将这 些文本摘要 存储在程序 之外。程序 自动使用用 户登录语言 的文本摘要 。
不用更改程 序代码，就 可以创建和 维护文本摘 要。可以创 建标准文本 池（可以将 标准文本池 从一个程序 复制到另一 个程序）。 如果在 WRITE 语句中只是 处理符号， 而不使用串 文字，程序 就与语言无 关。只有初 始语言文本 池的文本摘 要，才必须 翻译成其它 语言。
ABAP/4 工作台完全 支持文本摘 要的翻译。 翻译器可以 从初始语言 的现有文本 池，创建其 它不同语言 的文本池。

创建和更改 文本摘要

如果希望创 建或更改程 序特有的文 本摘要，请 进行如下操 作：

1. 在“ ABAP/4 编辑器初始 屏幕”（SE38） 上，在“程 序”字段中 ，输入希望 维护其文本 摘要的程序 名称。
2. 选择 “文本摘要 ”并选择“ 显示”或“ 更改”。 图2 （略）

进入“ABAP/4 文本摘要” 屏幕。 图3 （略）

现在，选择 希望为程序 维护的文本 摘要类型。
如果已更改 程序的源代 码，但还没 有生成程序 ，系统询问 是否要生成 程序。要更 新文本摘要 与程序的关 系，则必须 生成程序。
如果登录语 言与程序的 初始语言不 一样（比如 ，登录语言 是创建程序 的语言）， 则下列特性 适用于所有 文本摘要：

在显示模 式中，可以 看到表示初 始语言与登 录语言不同 的警告。文 本摘要以登 录语言显示 。如果某些 文本摘要在 初始语言中 存在，但在 登录语言中 不存在，则 它们以初始 语言显示， 并用语言标 识符在右边 做上相应的 标记。这允 许定位未翻 译的文本摘 要（参见 *翻译文本摘 要*（页 8）中的示例） 。

在更改模 式中，系统 询问是否希 望在初始语 言中维护文 本摘要，或 是否希望更 改初始语言 。更改初始 语言时，就 从旧初始语 言中获取新 的初始语言 中不存在的 文本，但不 做标记。

下面的主题 更详细地讨 论不同的文 本摘要：

标题和表头

每个程序都 必须有标题 。在指定程 序属性（参 见*指定程序属性*（页 **错误！链接无效。**））时，输入 程序标题。 可以随意更 改标题。

可以创建或 更改程序输 出列表的表 头行，以及 列表中不同 列的列标题 。

更改程序的 标题

要更改程序 的标题，从 “ABAP/4 文本摘要” 屏幕中选择 “标题和表 头”，然后 选择“更改 ”。

在“标题” 字段中，可 以输入最长 为 70 个字符的标 题。

图 4 （略）

选择“保存 ”保存更改 。

创建和更改 列表及列表 头

要在输出中 创建或更改 标题，从“ ABAP/4 文本摘要” 屏幕中选择 “标题和表 头”，然后 选择“更改 ”。

在“列表表 头”字段中 ，可以输入 最长为 70 个字符的列 表表头，在 “列标题” 字段的四行 中，可以输 入最长为 255 个字符的列 标题。 图 5 （略）

可以用“编 辑”菜单的 选项格式化 标题。如果 没有指定任 何列表表头 ，就在屏幕 上显示程序 标题。

选择“保存 ”保存更改 。 图 6 （略）

> 假设有下列 程序：
>
> ```
> PROGRAM SAPMZTST.
>
> DATA: NUM1 TYPE I, NUM2 TYPE P DECIMALS 2.
>
> DO 5 TIMES.
>   NUM1 = SY-INDEX ** 2. NUM2 = SQRT( SY-INDEX ).
>   WRITE: / SY-INDEX, NUM1, NUM2.
> ENDDO.
> ```
>
> 如果创建如 上所示的表 头，则输出 屏幕如下： 图7 （略）
>
> 对于包含程 序（参见 包含程序），只能维 护程序标题 。

选择文本

可以用文本 摘要替代屏 幕上出现的 参数及选择 标准的标准 文本（参见 使用选择屏幕）。

要更改选择 屏幕上的文 本，从“ABAP/4 文本摘要” 屏幕中选择 “选择文本 ”，然后选 择“更改” 。

在下列屏幕 上，列“名 称”已经包 含程序的参 数名称及程 序的选择标 准名称（参 见下面的示 例）。

现在 ，对每个参 数和选择标 准，可以输 入最长为 30 个字符的选 择文本。

如果已创建 ABAP/4 词典字段的 选择标准， 可以选择“ 实用程序 -> 复制 DD 文本”。系 统用作为属 性分配给 ABAP/4 词典字段的 短文本，自 动填充这些 选择文本。 文本摘要保 持不变，但 可以修改。 图 8 （略）

> 如果在将短 文本作为选 择文本复制 到文本池中 之后，再在 ABAP/4 词典中对该 文本进行更 改，则不将 该更改自动 转移到文本 池中。

选择“保存 ”保存更改 。

如果在维护 选择文本之 后，再更改 或删除任何 参数或选择 标准，然后 再调用选择 文本维护， 则可以在程 序中每个不 再需要的文 本右边出现 带标记的复 选框。这样 有助于删除 任何不使用 的选择文本 。如果试图 删除程序中 使用的选择 文本，则系 统输出警告 信息。在存 储选择文本 时，系统再 次提示存在 不使用的文 本，并将其 显示出来以 供删除。

> 假定有下列 程序：

```
PROGRAM SAPMZTST.

TABLES SBOOK.

PARAMETERS: TEXT(10).

SELECT-OPTIONS: SEL1 FOR SBOOK-CARRID,
                SEL2 FOR SBOOK-CONNID.
```

更改选择文 本的屏幕如 下： 图9 （略）

选择“实用 程序 -> 复制 DD 文本”之后 ，它变为： 图10 （略）

现在，可以 将光标放在 （例如）SEL1 上，选择“ 实用程序 -> 复制 DD 文本”更改 其选择文本 。 图11 （略）

也更改了参 数 TEXT 的选择文本 。

在保存选择 文本并启动 SAPMZTST 后，选择屏 幕如下： 图12 （略）

现在，用参 数 PARA 代替程序中 的参数 TEXT：

```
PROGRAM SAPMZTST.

TABLES SBOOK.

PARAMETERS: PARA(10).

SELECT-OPTIONS: SEL1 FOR SBOOK-CARRID,
                SEL2 FOR SBOOK-CONNID.
```

然后，如果 再次调用选 择文本维护 ，则生成程 序后，屏幕 如下：

显示新参数 PARA， 并将旧参数 TEXT 标记为“未 使用”。

文本符号

文本符号是 在程序外输 入和维护的 文本常数。 在程序的最 终版本中， 应该用文本 符号而不是 文本文字， 以保持程序 与语言无关 并易于维护 。
要创建或更 改文本符号 ，在“ABAP/4 文本摘要” 屏幕上选择 “文本符号 ”，并选择 “更改”。
对于每个文 本符号，应 该指定一个 三字符长度 的标识符， 该标识符不 包含空格， 也不以字符 ‘%’ 打头 。可以给每 个文本符号 分配最长为 132 个字符的文 本。下划线 （_）代表 空格。但是 ，不能用文 本符号在屏 幕上输出下 划线。
图（略）
可以用“编 辑”菜单的 功能格式化 文本符号。
要删除文本 符号，请将 光标放在文 本符号上， 并选择“删 除”。
选择“保存 ”保存更改 。
要在程序中 包含文本符 号，请按如 下格式使用 WRITE 语句：

语法
WRITE ... TEXT-<idt> ...
系统在文本 池中查找带 标识符 <idt> 的文本符号 ，并将分配 的文本写到 屏幕上。如 果文本符号 <idt> 不存在，则 系统跳过 WRITE 语句。
为避免跳过 WRITE 语句，可以 在程序中定 义文本。如 果文本符号 不存在，系 统就将该文 本写到屏幕上。为此， 请按如下格 式使用 WRITE 语句：

语法
WRITE ... '<text>'(<idt>) ...
如果存在文 本符号 <idt>， 系统就使用 文本符号。 否则，系统 将串文字 <text> 写到屏幕上 。

图略

不能对文本 符号使用偏 移量规范（ 参见 *为数据对象指定偏移值* (页 **错误！链接无效。**) ）。

```
PROGRAM SAPMZTST.

WRITE:   TEXT-010,
       / TEXT-AAA,
       / TEXT-020,
       / 'Default Text 030'(030),
       / 'Default Text 040'(040).
```

如果将上述 文本符号与 该程序链接 ，则输出如 下：

文本符号 020 和 040 不存在。在 020 的情况下， 系统跳过 WRITE 语句。在 040 的情况下， 则将程序中 定义的文本 输出到屏幕 上。

复制文本摘 要

可以将文本 摘要从一个 ABAP/4 程序复制到 另一个 ABAP/4 程序。复制 功能允许传 输具有自己 的标准文本 摘要的文本 池，这些标 准元素是在 不同程序中 使用的。为 此，请在“ ABAP/4 文本摘要” 屏幕上选择 “复制”。
在“源程序 ”字段中， 输入希望从 其中复制文 本的程序名 称，在“目 标程序”字 段中，输入 希望将文本 复制到其中 的目标程序 名称。（图略）

要复制源程 序的所有文 本，选择“ 复制”。
如果只希望 从源程序中 复制某些文 本，则选择 “选择部分 ”。现在， 可以指定希 望复制的对 象。（图略）
选择相关的 复选框并选 择“复制” 。

比较文本符 号

如果在程序 代码中插入 新的文本符 号，或更改 现有文本符 号，则不自 动将这些文 本符号复制 到文本池中 。要更新该 列表并消除 任何差异， 可以使用“ ABAP/4 文本摘要” 屏幕上的“ 比较文本符 号”功能。
为程序调用 该功能后， 系统在屏幕 上显示程序 代码中的所 有文本符号 ，其文本在 程序中和文 本池中定义 。在右边， 可以看到在 程序和文本 池中这些文 本是否一样 （“P=T” ）。如果文 本不一样， 则用 T 标记文本池 中的文本， 并用 P 标记程序中 定义的文本 。
在列 +/- 中，可以指 定
“-”， 从文本池中 删除文本符 号
“+”， 将文本符号 添加到文本 池中
“ ”，文本符 号保持不变
如果选择“ 调整”，则 系统根据这 些设置，刷 新文本池中 的文本符号 。这不改变 程序代码。

图略

编写如下程 序：

```
PROGRAM TEXTTEST.

WRITE: TEXT-010,
       'Placeholder'(020),
       TEXT-030.
```

不创建任何 文本符号。 文本符号列 表为空。在 “ABAP/4 文本摘要” 屏幕上选择 “比较文本 符号”后， 如果没有生 成程序，就 出现对话框 。

图略

在这种情况 下，单击“ 生成”。就 可以看到文 本池中的文 本符号和程 序代码的比 较。

图略

在程序中指 定的文本符 号 010、020 和 030，在 文本池中没 有定义。要 将所有这些文本符号复 制到文本池 ，使列 +/- 中的符号保 持为“+” ，并选择“ 调整”。在 做完这些后 ，在显示模 式中，文本 符号列表如 下：

图略

系统已在文 本池中创建 了文本符号 010、020 和 030。它 没有为 010 和 030 分配任何文 本，并将在 程序中定义 的文本分配 给 020。现 在，可以更 改文本符号 ，如 *文本符号* (页 37) 中所述。

将程序代码 更改为

```
PROGRAM TEXTTEST.

WRITE: TEXT-010,
       'Placeholder'(020),
       'Text Symbol'(030).
```

生成程序， 并再次在“ ABAP/4 文本摘要” 屏幕上选择 “比较文本 符号”。现 在，比较显 示如下：

图略

首先，看到 两个正确定 义的文本符 号 010 和 020，然 后看到文本 池和程序中 文本符号 030 文本有区别 的消息。“ +”和“-” 符号允许定 义希望保留 哪个文本。

翻译文本摘 要

要将文本摘 要翻译为其 它语言，从 “ABAP/4 开发工作台 ”屏幕开始 ，选择“实 用程序 → 翻译 → 短/长文本 ”。在后续 的“翻译短 /长文本” 屏幕上，选 择“翻译 → 短文本 → 程序”，然 后选择要翻 译的文本摘 要。出现下 列屏幕：

输入文本摘 要所属的程 序名称、初 始语言和目 标语言，并 选择“翻译 ”。
文本摘要以 其输入语言 显示。可以 输入注释， 并使用文本 摘要下面的 行输入翻译 （参见下面 的示例）。
翻译完当前 类型的所有 文本摘要后 ，请使用“ 定位”菜单 中的功能翻 译其它类型 的文本摘要 。
翻译完所有 文本摘要后 ，选择“保 存”保存翻 译。
采用这种方 式可以创建 不同语言的 完整文本池 。
如果存在不 同语言的文 本池，可以 使用以下选 项影响程序 的输出语言

登录语言 ：在缺省情 况下，系统 自动使用用 户的登录语 言。

SET LANGUAGE 语句：借助 该 ABAP/4 语句，可以 在程序中显 式地指定输 出语言，而 不考虑登录 语言。

语法

```
SET LANGUAGE <lg>.
```

语言 <lg> 可以是文字 或变量。

选择某种语 言（通过登 录语言或显 式规定）时 ，系统在该 语言的文本 池中查找包 含在程序中 的文本摘要 。如果在该 文本池中找 不到文本摘 要，则系统 将程序代码 中的文本输 出到屏幕上 ，或跳过相 关的 WRITE 语句（参见 *文本符号* (页 37))。系统不 使用其它语 言的文本摘 要。

图略

假定程序如 下所示，初 始语言为英 语。

```
PROGRAM TRANTEST.

WRITE TEXT-010.
```

并有如下输 出：

图略

如果用德语 登录到系统 上，并且不 存在英语文 本符号，在 显示程序的 文本符号时 ，就看到下 列行：

图略

右边的字符 “E”表示 存在英语文 本符号 010。

在 ABAP/4 开发工作台 中，将文本 摘要从英语 翻译为德语 ，如下所示 ：

标题和表头 ：

图略

文本符号：

图略

如果登录 R/3 系统并指定 语言“D” ，程序 TRANTEST 的输出如下 ：

图略

通过按如下 格式更改程 序代码，可 以将程序 TRANTEST 的输出语言 修改为德语 。

```
PROGRAM TRANTEST.
SET LANGUAGE 'D'.
WRITE TEXT-010.
```

现在，无论 采用何种语 言登录，输 出始终为德 语

第六章 处理数据

概览

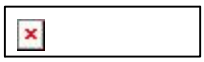
内容

本节描述如 何使用（处 理）数据对 象。涉及到 以下主题：

赋值

在 ABAP/4 中，可以在 声明语句和 操作语句中 给数据对象 赋值。
在声明语句 中，将初始 值赋给声明 的数据对象 。为此，可 以在 DATA、 常量或 STATICS 语句中使用 VALUE 参数（参见 *创建数据对象和数据类型* (页 **错误！链接无效。**) ） 。
要在操作语 句中给数据 对象赋值， 可以使用：

MOVE 语句，对应 于赋值运算 符（=）
WRITE TO 语句



本节提到的 大多数操作 不仅适用于 程序中的内 部字段，而 且适用于程 序参数、表工作区、系 统字段、字 段符号和形 式参数，以 及不更改数 据对象情况 下的常量和 文字。当讨 论有关字段 的操作时， 参照相对于 一般字段， 而并非仅是 内部字段。

用 MOVE 赋值

本节主题描 述如何使用 MOVE 语句或赋值 运算符（= ）。其中包 括：

基本赋值操 作
要将值（文 字）或源字 段的内容赋 给目标字段 ，可以使用 MOVE 语句或赋值 运算符（= ）。

MOVE 语句的语法 如下所示：

语法
MOVE <F1> TO <F2>.
MOVE 语句将源字 段<F1> 的内容传送 给目标字段 <F2>。 <F1>可 以是任何数 据对象。<F2> 必须是变量 ，不能是文 字或常量。 传送后，<F1> 的内容保持 不变。
赋值运算符 （＝）的语 法如下所示 ：

语法
<F2> = <F1>.
MOVE 语句和赋值 运算符功能 相同。
如下所示， 可以多重赋 值：
<F4> = <F3> = <F2> = <F1>
ABAP/4 按从右到左 的顺序进行 处理，如下 所示：
MOVE <F1> TO <F2>.
MOVE <F2> TO <F3>.
MOVE <F3> TO <F4>.

在以上语句 中，不管用 户主记录如 何，小数点 必须用句点 （.）指定 。

源字段和目 标字段可以 是不同数据 类型。其它 程序设计语 言通常将不 同数据类型 之间的赋值 限制在少数 可能组合之 内，而 ABAP/4 则提供大量 自动类型转 换。
例如，可以 将基本数据 类型的源字 段内容赋给 其它基本数 据类型的目 标字段（除 了数据类型 D 无法赋给数 据类型 T，反之亦 然）。ABAP/4 也支持结构 化数据和基 本数据对象 之间或结构 不同的数据对象之间的 赋值。
对于每一赋 值语句（用 MOVE 或赋值运算 符），系统 都检查源字 段和目标字 段的数据类 型。如果定 义了此类组 合的类型转 换，则将源 字段内容转 换为目标字 段数据类型 并将其值赋 给目标字段 。要获得可 能数据类型 转换的综述 以及在 ABAP/4 中是如何定 义的等信息 ，请参见*类型转换* （页 21）。

```
DATA: T(10),
      NUMBER TYPE P DECIMALS 1,
      COUNT  TYPE P DECIMALS 1.

T = 1111.
MOVE '5.3'     TO NUMBER.
COUNT = NUMBER.
```

赋值结果是 字段T、NUMBER 和 COUNT 分别包含值 ‘1111 ’、5.3 和 5.3。请 注意，当赋 值数值 1111 时，系统将 它转换为长 度为 10 的字符串。

要在运行时 指定源字段 和目标字段 ，必须按使用字段符号所 述，使用字 段符号。因 此无法使用 MOVE 语句（或赋 值运算符） 。

用指定偏移 量赋值
可以在每条 ABAP/4 语句中为基 本数据类型 指定偏移量 和长度（参 见*指定数据对 象的偏移量* （页 347））。在这 种情况下， MOVE 语句语法如 下：

语法
MOVE <F1>[+<o1>][(<l1>)] TO <F2>[+<o2>][(<l2>)].
赋值运算符 语法如下：

语法
<F2>[+<o2>][(<l2>)] = <F1>[+<o1>][(<l1>)]。
将字段 <F1> 从 <o1>+1 位置开始且 长度为 <l1> 的段内容赋 给字段 <F2> ，覆盖从 <o2>+1 位置开始且 长度为 <l2> 的段。

在 MOVE 语句中，所 有偏移量和 长度指定都 可为变量。 这同样适用 于采用赋值 运算符的语 句，只要也 可以写做 MOVE 语句。在赋 值运算符之 后未指定字 段名称的语句中，（例 如在数学表 达式中），所有偏移量 和长度指定 都必须是无 符号数值。 详细信息， 参见*数值运算* （页 388））。

SAP 建议只在非 数值字段之 间才采用指 定偏移量和 长度赋值。 对于数值字 段，结果毫 无意义。

```
DATA: F1(8) VALUE 'ABCDEFGH',
      F2(20).

F2+6(5) = F1+3(5).
```

在该示例中 ，赋值运算 符功能如下 ：

```
DATA: F1(8) VALUE 'ABCDEFGH',
      F2(8).

DATA: O TYPE I VALUE 2,
      L TYPE I VALUE 4.

MOVE F1      TO F2.      WRITE   F2.
MOVE F1+O(L) TO F2.      WRITE / F2.
MOVE F1      TO F2+O(L). WRITE / F2.

CLEAR F2.
MOVE F1      TO F2+O(L). WRITE / F2.
MOVE F1+O(L) TO F2+O(L). WRITE / F2.
```

该过程输出 如下：

```
ABCDEFGH

CDEF

CDABCD

  ABCD

  CDEF
```

首先，未指 定偏移量将 F1 内容赋给 F2。然后 ，再指定偏 移量和长度 执行同样操 作。后面三 条 MOVE 语句覆盖偏 移量为2. 的 F2 内容。请注 意，根据 *源类型字符* （页 441）中的规则， F2 的右边用空 格填充。

在字段串组 件之间复制 值

描述的 MOVE 语句赋值方 法适用于基 本数据对象 和结构化数 据对象。另 外，还有一 种 MOVE 语句变体， 允许将源字 段串组件内 容复制到目 标字段串组 件中。语法 如下：

语法

```
MOVE-CORRESPONDING <string1> TO <string2>.
```

该语句将字 段串 <string1> 组件的内容 赋给有相同 名称的字段 串 <string2> 组件。

对于每对名 称，系统都 执行 MOVE 语句，如下 所示：

```
MOVE STRING1-<component> TO STRING2-<component>.
```

系统分别执 行所有必要 类型转换。 该处理不同 于包括整个 字段串的赋 值。在这种 情况下，应 用*不兼容的字 段串和基本 字段*（页 385）所述的转 换规则。

```
DATA: BEGIN OF ADDRESS,
         FIRSTNAME(20) VALUE 'Fred',
         SURNAME(20) VALUE 'Flintstone',
         INITIALS(4) VALUE 'FF',
         STREET(20) VALUE 'Cave Avenue,
         NUMBER TYPE I VALUE '11'.
         POSTCODE TYPE N VALUE '98765'.
         CITY(20) VALUE  'Bedrock',
      END OF ADDRESS.

DATA: BEGIN OF NAME,
         SURNAME(20),
         FIRSTNAME(20),
```

```
        INITIALS(4),
        TITLE(10) VALUE 'Mister',
      END OF NAME.

MOVE CORRESPONDING ADDRESS TO NAME.
```

在该示例中 ，将 NAME-SURNAME、 NAME-FIRSTNAME 和 NAME-INITIALS之 值设置为 Flintstone、 Fred 和 FF。NAME-TITLE 仍为值 Mister。

用 WRITE TO 赋值

用 WRITE TO 给数据对象 赋值时，可 以用 WRITE 语句的格式 化选项（参 见 *WRITE 语句* (页 **错误！链接无效。**))。
本节主题描 述

WRITE TO 语句的基本 形式
要将值（文 字）或源字 段内容写入 目标字段， 可以使用 WRITE TO 语句：

语法
WRITE <F1> TO <F2> [<option>].
WRITE TO 语句将源字 段 <F1> 内容写入目 标字段 <F2>。 <F1> 可以是任何 数据对象。 <F2>必 须是变量， 不能是文字 或常量。写 入后，<F1> 内容保持不 变。
对于 <option>， 可以使用 WRITE 语句的所有 格式化选项 （UNDER 和 NO-GAP 除外）(参 见*格式选项* (页 **错误！链接无效。**))。
WRITE TO 语句总是检 查用户主记 录中的设置 。例如，这 些设置指定 是将小数点 显示为句号 (.)，还 是逗号 (,) 。
WRITE TO语句并不遵 循 *类型转换* (页 21) 中所述的转 换规则。目 标字段解释 为类型 C 字段。系统 总是将源字 段内容转换 为类型 C,它不将 结果字符串 转换为目标 字段的数据 类型,而直 接写入目标 字段。因此 ， 不应使用 数值数据类 型的目标字 段。

```
DATA: NUMBER TYPE F VALUE '4.3',
      TEXT(10),
      FLOAT TYPE F,
      PACK  TYPE P DECIMALS 1.

WRITE NUMBER.

WRITE NUMBER TO TEXT EXPONENT 2.
WRITE / TEXT.

WRITE NUMBER TO FLOAT.
WRITE / FLOAT.

WRITE NUMBER TO PACK.
WRITE / PACK.

MOVE NUMBER TO PACK.
WRITE / PACK.
```

该过程输出 如下：

```
 4.30000000000000E+00

 0.043E+02

1.50454551753894E-153

20342<33452;30,3

                4.3
```

第一输出行 以类型 F 字段的标准 输出格式显 示字段 NUMBER 内容。第二 输出行显示字符串,该 字符串产生 于用格式化 选项 EXPONENT 2 将字段 NUMBER 写入长度为 10 的类型 C 字段中。第 三和第四输 出行显示不 适合于使用 数值数据类 型的目标字 段。第五输 出行显示 MOVE 语句不同于 WRITE TO 语句在于将 类型F 字段正确地 转换为类型 P （有关该转 换的详细信 息，参见*源类型浮点 数* (页 440) ） 。

运行时指定 源字段
可以使用 WRITE TO 语句在运行 时指定源字 段。为此， 请用括号将 包含源字段 名的数据对 象名括起来 ，并将其放 在源字段位 置：

语法
WRITE (<f>) TO <g>.
系统将赋给 <f> 的数据对象 值放到 <g> 中。
然而，如果 要在运行时 指定目标字 段，则必须 按使用字段符号中所述， 使用字段符 号。

```
DATA: NAME(10)   VALUE 'SOURCE',
      SOURCE(10) VALUE 'Antony',
      TARGET(10).

...
WRITE (NAME) TO TARGET.
WRITE: TARGET.
```

则输出

Antony

字段名和字 段内容之间 的连接如下 图所示。

用指定偏移 量写入值
可以指定每 条 ABAP/4 语句中基本 数据对象的 偏移量和长 度（参见*指定数据对 象的偏移量* （页 347） ） 。WRITE TO 语句语法如 下：

语法
WRITE <F1>[+<o1>][(<l1>)] TO <F2>[+<o2>][(<l2>)].
将字段 <F1> 中从 <o1>+1 位置开始且 长度为 <l1> 的内容赋给 字段 <F2>， 覆盖从 <o2>+1 位置开始且 长度为 <l2> 的段。

在 WRITE TO 语句中，目 标字段的偏 移量和长度 指定可为变 量。

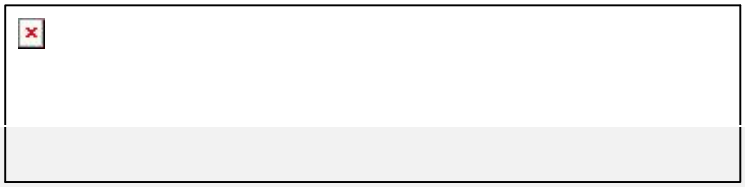
SAP 建议只在非 数值字段之 间采用偏移 量和长度指 定赋值。对 于数值字段 ，结果毫无 意义。

```
DATA: STRING(20),
      NUMBER(8) TYPE C VALUE '123456',
      OFFSET TYPE I VALUE 8,
      LENGTH TYPE I VALUE 12.

WRITE NUMBER+(6) TO STRING+OFFSET(LENGTH) LEFT-JUSTIFIED.
WRITE: / STRING.
CLEAR STRING.

WRITE NUMBER+(6) TO STRING+OFFSET(LENGTH) CENTERED.
WRITE: / STRING.
CLEAR STRING.

WRITE NUMBER TO STRING+OFFSET(LENGTH) RIGHT-JUSTIFIED.
WRITE: / STRING.
CLEAR STRING.
```

输出为：

```
        123456
           123456
              123456
```

第一条 WRITE 语句将字段 NUMBER 左对齐的前 6个位置写 入字段 STRING 的后12个 位置。

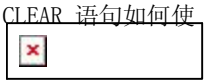


第二条 WRITE 语句将 NUMBER 居中的前6个 位置写入 STRING 的后12个 位置。最后 ，第三条 WRITE 语句将右对 齐的 NUMBER 的前6个位 置写入 STRING的 后12个位 置。

将值重置为 默认值

可以用 CLEAR 语句重置任 何数据对象 值，如下所 示：

语法
CLEAR <f>.
该语句将数 据对象 <f> 的内容重置 为初始默认 值。可以区 分
基本数据 类型
对于基本数 据类型的数 据对象的默 认值列表， 参见*预定义的基本数据类型*（页 **错误！链接无效。**）中的表 。系统将变 量值重置为 它们的初始 默认值，而 不是用DATA 语句的 VALUE 参数所赋的 初始值。无 法重置常量 。
字段串
如果将 CLEAR 语句应用于 字段串，则 将单个组件 的内容重置 为它们的初 始默认值。
内表
关于 CLEAR 语句如何使 用内表的解 释，参见*初始化内表*（页 **Error! Not a valid link.**）。



```
DATA NUMBER TYPE I VALUE '10'.
WRITE NUMBER.
CLEAR NUMBER.
WRITE / NUMBER.
```

输出为：

```
        10
         0
```

CLEAR 语句将字段 NUMBER 的内容从10 重置为默认 值 0。

数值运算

要处理数值 数据对象并 将结果值赋 给数据对象 ，可以用 COMPUTE 语句或赋值 运算符（＝ ）。
COMPUTE 语句语法如 下所示：

语法
COMPUTE <n> = <expression>.
关键字 COMPUTE 可选。换句 话说，该语 句也可以写 成：

语法
<n> = <expression>.
两条语句效 果等同。
在 <expression> 中指定的数 学运算结果 赋给字段 <n>。赋 值运算符（ ＝）的作用 如*基本赋值操 作*（页 41）中所述。 如果运算结 果与 <n> 的数据类型 不同，则系 统自动进行 类型转换（ 参见*类型转换*（页 21）。
在数学表达 式中，可以 按任何排列 组合运算并 用括号进行 指定。
求值顺序是 ：

1. 括号 中的表达式
2. 函数
3. ** （求幂）
4. *、 /、MOD、 DIV （乘法、除 法）
5. +、 - （加法、减 法）

以下主题对 <expression> 中指定的数 学运算加以 描述：
处理以下数 据类型特别 重要：

执行算术运 算

要定义算术 运算，可以 使用适当的 算术运算符 。使用算术 运算进行

基本算术运 算

ABAP/4 支持四种基 本算术运算 ，同时还支 持乘方计算 。可以用数 学表达式指 定以下算术 运算符：

+	加法
-	减 法
*	乘法
/	除 法
DIV	整 除
MOD	求 余
**	求幂

可以用关键 字 ADD、SUBTRACT、 MULTIPLY 和 DIVIDE 进 行基本算术 运算，而不 使用数学表 达式中的运 算符。

下表说明如 何表示 ABAP/4 中的基本算 术运算：

运算	用数学 表达式的语 句	用关键字的 语句
加法	<p> = <n> + <m>.	ADD <n> TO <m>.
减 法	<p> = <m> - <n>.	SUBTRACT <n> FROM <m>.
乘 法	<p> = <m> * <n>.	MULTIPLY <m> BY <n>.
除 法	<p> = <m> / <n>.	DIVIDE <m> BY <n>.
整 除	<p> = <m> DIV <n>.	---
除 法余数	<p> = <m> MOD <n>.	---
求 幂	<p> = <m> ** <n>.	---

在 用关键字的 语句中，将 运算结果赋 给 <m>。

运算数 <m>、 <n>、 <p> 可以是任何 数值字段。 如果字段类 型不同，则 系统自动进 行必要类型 转换（有关 类型转换的 详细信息， 参见*类型转换* (页 21) ）。

使用数学表 达式时,请 注意,运算 符 +、-、*、**、/ 以及前括号 、后括号是 ABAP/4 关键字，前 面和后面都 必须有空格 。

在除法运算 中,如果被 除数不为零 ，则除数不 能为零。对 于整除,用 运算符 DIV 或 MOD 代替 / 。用 DIV 获得整数商 并用 MOD 获得余数。

如果将几种 数学表达式 组合在一起 ，对于相同 优先级的运 算符，从左 到右进行计 算（求幂计 算除外，它 是从右到左 进行)。因 此，<n> ** <m> ** <p> 与 <n> ** （ <m> ** <p> ）相同，但 与（ <n> ** <m> ） ** <p>不同 。

```
DATA: COUNTER TYPE I.
COMPUTE COUNTER = COUNTER + 1.
COUNTER = COUNTER + 1.
ADD 1 TO COUNTER.
```

在此，三条 运算语句进 行相同算术 运算，例如 ，将1加到 字段 COUNTER 内容上并将结果赋给 COUNTER。

```
DATA: PACK TYPE P DECIMALS 4,
      N TYPE F VALUE '+5.2',
      M TYPE F VALUE '+1.1'.
PACK = N / M.
WRITE PACK.
```

```
PACK = N DIV M.
WRITE  / PACK.

PACK = N MOD M.
WRITE /PACK.
```

输出为：

```
4.7273
4.0000
0.8000
```

该示例说明 除法的不同 类型。

执行字段串 的算术运算

类似于用 MOVE-CORRESPONDING 语句在字段 串之间复制 值(参见*在字段串组 件之间复制 值* (页 43))，可以用以 下关键字， 执行字段串 的算术运算 ：

ADD-CORRESPONDING
SUBTRACT-CORRESPONDING
MULTIPLY-CORRESPONDING
DIVIDE-CORRESPONDING

对于所有同 名字段串组 件，ABAP/4 进行相应算 术运算。然 而，仅当所 有相关组件 都是数值型 数据类型时 ，该运算才 有意义。

有关这些关 键字的详细 信息，参见 ABAP/4 关键字文档 。

```
DATA: BEGIN OF RATE,
         USA TYPE F VALUE '0.6667',
         FRG TYPE F VALUE '1.0',
         AUT TYPE F VALUE '7.0',
      END OF RATE.

DATA: BEGIN OF MONEY,
         USA TYPE I VALUE 100,
         FRG TYPE I VALUE 200,
         AUT TYPE I VALUE 300,
      END OF MONEY.

MULTIPLY-CORRESPONDING MONEY BY RATE.

WRITE / MONEY-USA.
WRITE / MONEY-FRG.
WRITE / MONEY-AUT.
```

输出如下：

```
    67
   200
 2,100
```

在此，用 RATE-USA乘 以 MONEY-USA， 以此类推。

添加字段顺 序

除*基本算术运 算* (页 47)中所述的 基本加法之 外，ADD 语句有几个 变体，用于 添加字段顺 序。例如：

添加字段 顺序并将结 果赋给另一 个字段

语法

ADD $\langle n_1 \rangle$ THEN $\langle n_2 \rangle$ UNTIL $\langle n_z \rangle$ GIVING $\langle m \rangle$.

如果 $\langle n_1 \rangle$、$\langle n_2 \rangle$、 . . . 、$\langle n_z \rangle$ 是在内存中 相同类型和 长度的等距 字段序列， 则进行求和 计算并将结 果赋给 $\langle m \rangle$

添加字段 顺序并将结 果添加到另 一个字段的 内容中

语法

ADD $\langle n_1 \rangle$ THEN $\langle n_2 \rangle$ UNTIL $\langle n_z \rangle$ TO $\langle m \rangle$.

该语句除了 将字段总和 添加到 $\langle m \rangle$ 的旧内容中 之外，与上 面语句的工 作方式相同 。

有关其它相 似变体的信 息，参见有 关 ADD 语句的关键 字文档。

```
DATA: BEGIN OF SERIES,
         N1 TYPE I VALUE 10,
         N2 TYPE I VALUE 20,
         N3 TYPE I VALUE 30,
         N4 TYPE I VALUE 40,
         N5 TYPE I VALUE 50,
         N6 TYPE I VALUE 60,
      END OF SERIES.

DATA SUM TYPE I.

ADD SERIES-N1 THEN SERIES-N2 UNTIL SERIES-N5 GIVING SUM.
WRITE SUM.

ADD SERIES-N2 THEN SERIES-N3 UNTIL SERIES-N6 TO SUM.
WRITE / SUM.
```

输出如下：

150

350

在此，将 N1 到 N5 组件内容求 和并将其值 赋给字段 SUM。然 后，将 N2 到 N6组件求 和并将其添 加到 SUM 的值中。

使用数学函 数

要指定数学 表达式，可 以从 ABAP/4 内部函数集 中选择运算 。

语法
[COMPUTE] <n> = <func>(<m>).
在括号之间 的空格和参 数 <m> 是必须的。
将调用的参 数为 <m> 的函数 <func> 结果赋给 <n>。
在以下主题 中描述可用 函数：

所有数值数 据类型的函 数
以下内部函 数使用所有 三种数值数 据类型（F、 I 和 P）作为参 数。

所有数值数 据类型的函 数

函数	结果
ABS	参 数的绝对值 。
SIGN	参 数符号： 1 X > 0 SIGN(X) = 0 if X = 0 -1 X < 0
CEIL	不 小于参数的 最小整数值 。
FLOOR	不 大于参数的 最大整数值 。
TRUNC	参 数的整数部 分。
FRAC	参 数的分数部 分。

```
DATA N TYPE P DECIMALS 2.
DATA M TYPE P DECIMALS 2 VALUE '-5.55'.

N = ABS( M ).   WRITE:   'ABS:  ', N.
N = SIGN( M ).  WRITE: / 'SIGN: ', N.
N = CEIL( M ).  WRITE: / 'CEIL: ', N.
N = FLOOR( M ). WRITE: / 'FLOOR:', N.
```

```
N = TRUNC( M ). WRITE: / 'TRUNC:', N.
N = FRAC( M ).  WRITE: / 'FRAC: ', N.
```

输出如下：

```
ABS:              5.55
SIGN:             1.00-
CEIL:             5.00-
FLOOR:            6.00-
TRUNC:            5.00-
FRAC:             0.55-
```

这些函数的 参数不必为 数值数据类 型。如果选 择其它类型 ，则被转换 为数值类型 。然而，由 于性能原因 ， 应尽可能 使用正确类 型。

```
DATA: T1(10),
      T2(10) VALUE '-100'.
T1 = ABS( T2 ).
WRITE T1.
```

输出为：

```
       100
```

执行了两种 转换。首先 ，将类型 C 字段的内容 T2 转换为类型 P 并且系统用 转换结果处 理函数 ABS。然 后，在类型 C 字段 T1 的赋值期间 ，将函数结 果转换回类 型 C。

浮点函数

以下内部函 数用浮点数 据类型（F） 作为参数。

浮点数据类 型的函数

函数	解释
ACOS、 ASIN、 ATAN; COS、SIN、 TAN	三角函数。
COSH、 SINH、 TANH	反三角函数 。
EXP	基 数为 e 的求幂函数 （e=2.7182818285） 。
LOG	基 数为 e 的自然对数 。
LOG10	基 数为 10 的对数。
SQRT	平 方根。

对于 所有函数， 正常数学约 束（例如 平方根只适 用于正数） 适用。否则 ， 会产生运 行错误。

这些函数的 参数不必为 浮点数。如 果选择其它 类型，则被 转化为类型 F，如*类型转换* (页 21) 中所 述。

```
DATA: RESULT TYPE F,
      PI(10) VALUE '3.141592654'.
```

```
RESULT = COS( PI ).
WRITE RESULT.
```

输出为 -1.00000000000000E+00。 在进行计算 之前，自动 将字符字段 PI 转换为类型 F 字段。

处理压缩数

如果未设置 程序属性“ 定点算法” ，则将类型 P 字段解释为 无小数位的 正数。DATA 语句的参数 DECIMALS 只影响 WRITE 输出的格式 。
因此，SAP 建议在使用 类型 P 字段时，总 是设置程序 属性“定点 算法”（参 见*指定程序属性* （页 **错误！链接无效。**） ）。
然后，在将 类型 P 字段输出到 输出列表中 时，关键字 DECIMALS 不仅指小数 点位置，而 且考虑算术 运算中的小 数位。
对于中间结 果，ABAP/4 计算不超过 31 位（在小数 点之前和之 后）。如果 设置“定点 算法”，则 ABAP/4 中对于压缩 数的计算与 袖珍计算器 计算方式相 同。

```
DATA: P TYPE P.
P = 1 / 3 * 3.
WRITE P.
```

如果未设置 程序属性“ 定点算法” ，则结果为 0，这是因 为除法结果 被内部取整 为 0。

如果设置程 序属性“定 点算法”， 则结果为 1，这是因 为除法结果 被内部存储 0.33333333333333333333333333333333， 精度达 31 位数字。

处理日期和 时间字段

日期和时间 字段数据类 型不是数值 型。但是， 由于进行自 动类型转换 ，可以采用 类似于数值 型字段的处 理方法，处 理日期和时 间字段（参 见*基本数据类 型的可转换 性* （页 437） ）
在处理日期 和时间字段 时，使用偏 移量指定常 常十分有用 （参见*指定数据对 象的偏移量* （页 347））。

```
DATA: ULTIMO TYPE D.
ULTIMO      = SY-DATUM.
ULTIMO+6(2) = '01'.          " = first day of this month
ULTIMO      = ULTIMO - 1.    " = last day of last month
```

在此，将上 月的最后一 天赋给日期 字段 ULTIMO。 为此：

1. 用当 前日期填充 ULTIMO。
2. 用指 定偏移量， 将日期更改 为当前月的 第一天。
3. 从 ULTIMO 减1，将其 内容更改为 上月的最后 一天。在进 行减法之前 ， 系统将 ULTIMO 转换为从 01.01.0001 起的天数并 将结果转换 为日期。

```
DATA: DIFF TYPE I,
      SECONDS TYPE I,
      HOURS TYPE I.
DATA: T1 TYPE T VALUE '200000',
      T2 TYPE T VALUE '020000'.
DIFF = T2 - T1.
SECONDS = DIFF MOD 86400.
HOURS = SECONDS / 3600.
```

最后三行可 由以下行替 换

```
HOURS = ( ( T2 - T1 ) MOD 86400 ) / 3600.
```

计算 02:00:00 和 20:00:00 之间的小时 数。

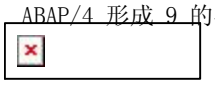
首先，计算 时间字段之 差，结果为 －64800。 这是因为 T1 被内部转换 为 72000， T2 被内部转换 为 7200。

其次，借助 运算 MOD，将 该负差转换 为总的秒数 。请注意， 如果该差为 正数，则进 行 MOD 运算后结果 保持不变。

第三、将秒 数除以 3600， 计算小时数 。

在某些情况 下（例如按 递减顺序对 日期进行排 序），用关 键字 CONVERT 将日期从格 式 D 转换为相反日期非常有 用。

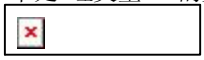
语法
```
CONVERT DATE <d1> INTO INVERTED-DATE <d2>.
CONVERT INVERTED-DATE <d1> INTO DATE <d2>.
```
这些语句将 字段〈d1〉从格式 DATE 或 INVERTED-DATE 转换为格式 INVERTED-DATE 或 DATE 并将其赋给〈d2〉。
对于该转换 ， ABAP/4 形成 9 的补数。



```
DATA: ODATE TYPE D VALUE '19955011',
      IDATE LIKE ODATE.
DATA  FIELD(8).
FIELD = ODATE.    WRITE / FIELD.
CONVERT DATE ODATE INTO INVERTED-DATE IDATE.
FIELD = IDATE.    WRITE / FIELD.
CONVERT INVERTED-DATE IDATE INTO DATE ODATE.
FIELD = ODATE.    WRITE / FIELD.
```
输出如下：
```
19955011
80049488
19955011
```

处理字符串

ABAP/4 提供多个处 理类型 C 的数据对象 的关键字， 即字符串。



在字符串运 算期间，系 统不进行类 型转换。用 下述主题中 所述的字符 串处理关键字，系统将 所有运算数 都当作类型 C 字段，而不 考虑它们的 实际类型。

处理字符串 的方法有：
有关 MOVE 语句变体（ 只使用字符 串）的解释 ，参见

移动字段内 容

要移动字段 内容，请使 用以下主题 中所述的 SHIFT 语句的各种 变体。SHIFT 允许按字节 移动字段内 容（或在文 本字段按字 符移动）。
对于 SHIFT 语句，可以 执行以下功 能：

按给定位置 数移动字段 串
要按给定位 置数移动字 段内容，请 使用 SHIFT 语句，用法 如下：

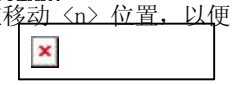
语法
```
SHIFT <c> [BY <n> PLACES] [<mode>].
```

该语句将字 段 <c> 移动 <n> 个位置。如 果省略 BY <n> PLACES， 则将 <n> 解释为一个 位置。如果 <n> 是 0 或负值，则 <c> 保持不变。 如果 <n> 超过 <c> 长度，则 <c> 用空格填充 。<n> 可为变量。
对不同（<mode>） 选项，可以 按以下方式 移动字段 <c>：

LEFT:
向左移动 <n> 位置，右边 用 <n> 个空格填充 （默认设置 ）。

RIGHT:
向右移动 <n> 位置，左边 用 <n> 个空格填充 。

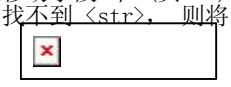
CIRCULAR:
向左移动 <n> 位置，以便 左边 <n> 个字符出现 在右边。



```
DATA: T(10) VALUE 'abcdefghij',
      STRING LIKE T.

STRING = T.
WRITE STRING.

SHIFT STRING.
WRITE / STRING.

STRING = T.
SHIFT STRING BY 3 PLACES LEFT.
WRITE / STRING.

STRING = T.
SHIFT STRING BY 3 PLACES RIGHT.
WRITE / STRING.

STRING = T.
SHIFT STRING BY 3 PLACES CIRCULAR.
WRITE / STRING.

输出为：

abcdefghij

bcdefghij

defghij

   abcdefg

defghijabc
```

移动字段串 到给定串
要移动字段 内容以到给 定串，则使 用 SHIFT 语句，用法 如下：

语法
SHIFT <c> UP TO <str> <mode>.
ABAP/4 查找 <c> 字段内容直 到找到字符 串 <str> 并将字段 <c> 移动到字段 边缘。 <mode> 选项与*按给定位置 数移动字段 串* (页 52) 中所 述相同。<str> 可为变量。
如果 <c> 中找不到 <str>， 则将 SY-SUBRC 设置为 4 并且不移动 <c>。否 则，将 SY-SUBRC 设置为 0。

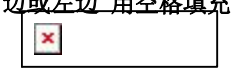


```
DATA: T(10) VALUE 'abcdefghij',
      STRING LIKE T,
      STR(2) VALUE 'ef'.

STRING = T.
WRITE STRING.

SHIFT STRING UP TO STR.
WRITE / STRING.

STRING = T.
SHIFT STRING UP TO STR LEFT.
WRITE / STRING.

STRING = T.
SHIFT STRING  UP TO STR RIGHT.
WRITE / STRING.

STRING = T.
SHIFT STRING  UP TO STR CIRCULAR.
WRITE / STRING.
```

```
输出如下：
abcdefghij
efghij
efghij
    abcdef
efghijabcd
```

根据第一个 或最后一个 字符移动字 段串
假设第一个 或最后一个 字符符合一 定条件，则 可用 SHIFT 语句将字段 向左或向右 移动。为此 ，请使用以 下语法：

语法
SHIFT <c> LEFT DELETING LEADING <str>.
SHIFT <c> RIGHT DELETING TRAILING <str>.
假设左边的 第一个字符 或右边的最 后一个字符 出现在 <str> 中，该语句 将字段 <c> 向左或向右 移动。字段 右边或左边 用空格填充 。<str> 可为变量。

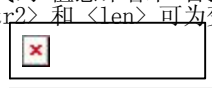


```
DATA: T(14) VALUE '    abcdefghij',
      STRING LIKE T,
      STR(6) VALUE 'ghijkl'.

STRING = T.
WRITE STRING.

SHIFT STRING LEFT DELETING LEADING SPACE.
WRITE / STRING.

STRING = T.
SHIFT STRING RIGHT DELETING TRAILING STR.
WRITE / STRING.

该过程输出 如下：
    abcdefghij
abcdefghij
        abcdef
```

替换字段内 容

要用其它字 符串替换字 段串的某些 部分，请使 用 REPLACE 语句。

语法
REPLACE <str1> WITH <str2> INTO <c> [LENGTH <l>].
ABAP/4 搜索字段 <c> 中模式 <str1> 前 <l> 个位置第一 次出现的地 方。如果未 指定长度， 按全长度搜 索模式 <str1>。
然后，语句 将模式 <str1> 在字段 <c> 中第一次出 现的位置用 字符串 <str2> 替换。如果 指定长度<l>，则只替换模 式的相关部 分。
如果将系统 字段 SY-SUBRC 的返回代码 设置为 0， 则说明在 <c> 中找到 <str1> 且已用<str2> 替换。非 0 的返回代码 值意味着未 替换。
<str1>、 <str2> 和 <len> 可为变量。



```
DATA: T(10) VALUE 'abcdefghij',
      STRING LIKE T,
      STR1(4) VALUE 'cdef',
      STR2(4) VALUE 'klmn',
      STR3(2) VALUE 'kl',
      STR4(6) VALUE 'klmnop',
      LEN TYPE I VALUE 2.
```

```
STRING = T.
WRITE STRING.

REPLACE STR1 WITH STR2 INTO STRING.
WRITE / STRING.

STRING = T.
REPLACE STR1 WITH STR2 INTO STRING LENGTH LEN.
WRITE / STRING.

STRING = T.
REPLACE STR1 WITH STR3 INTO STRING.
WRITE / STRING.

STRING = T.
REPLACE STR1 WITH STR4 INTO STRING.
WRITE / STRING.
```

该过程的输 出如下：

```
abcdefghij

abklmnghij

abklmnefgh

abklghij

abklmnopgh
```

请注意，在 最后一行中 ，字段 STRING 是如何在右 边截断的。 长度为 4 的搜索模式 ‘cdef’ 用长度为6的 ‘klmnop’ 替换。然后 ，填充字段 STRING 的剩余部分 直到字段结 尾。

转换大/小 写并替换字 符

可以将字母 转换大/小 写或使用替 换规则。
要转换大/小 写，请使用 TRANSLATE 语句，用法 如下：

语法
```
TRANSLATE <c> TO UPPER CASE.
TRANSLATE <c> TO LOWER CASE.
```
这些语句将 字段 <c> 中的所有小 写字母转换 成大写或反 之。

使用替换规 则时，请使 用以下语法 ：

语法
```
TRANSLATE <c> USING <r>.
```
该语句根据 字段 <r> 中存储的替 换规则替换 字段 <c> 的所有字符 。<r> 包含成对字 母，其中每 对的第一个 字母用第二 个字母替换 。<r> 可为变量。
有关包含更 复杂替换规 则的 TRANSLATE 语句的更多 变体，参见 关键字文档 。

```
DATA: T(10) VALUE 'AbCdEfGhIj',
      STRING LIKE T,
      RULE(20) VALUE 'AxbXCydYEzfZ'.

STRING = T.
WRITE STRING.

TRANSLATE STRING TO UPPER CASE.
WRITE / STRING.

STRING = T.
TRANSLATE STRING TO LOWER CASE.
WRITE / STRING.

STRING = T.
TRANSLATE STRING USING RULE.
WRITE / STRING.
```

该过程的输 出如下：

```
AbCdEfGhIj
```

```
ABCDEFGHIJ
abcdefghij
xXyYzZGhIj
```

转换为可排 序格式

可以将字符 字段转换为 可按字母顺 序排列的格 式：

语法
CONVERT TEXT <c> INTO SORTABLE CODE <sc>.
该语句为字 符字段 <c> 填充可排序 目标字段 <sc>。 字段 <c> 必须是类型 C 且字段 <sc> 必须是类型 X ，最小长度 为 <c> 长度的 16 倍 。
该语句目的 是为字符字 段 <c> 创建相关字 段 <sc>， 作为 <c> 的按字母顺 序排列的排 序关键字。 在内表的内 容和已解压 缩的数据中 进行排序（ 参见*排序内表* (页 **Error! Not a valid link.**) 和排序抽象数据集)。
如果对未转 换的字符字 段进行排序 ，则系统创 建与各字母 的特定平台 内部编码相 对应的顺序 。在对目标 字段进行排 序之后，转 换 CONVERT TEXT 按这样的方 式创建目标 字段，相应 的字符字段 顺序按字母 排序。例如 ，在德语中 ，顺序为‘ Miller、 Moller、 M ller、 Muller’ ，而不是‘ Miller、 Moller、 Muller、 M ller’ 。
转换方法依 赖于运行 ABAP/4 程序的文本 环境。文本 环境在用户 主记录中定 义。例外的 是可以使用如下语句， 在程序中设 置文本环境 ：

语法
SET LOCALE LANGUAGE <lg> [COUNTRY <cy>] [MODIFIER <m>].
该语句根据 语言 <lg> 设置文本环 境。对于选 项 COUNTRY， 只要特定国 家语言不同 ，就可以在 语言以外指 定国家。对 于选项 MODIFIER， 只要一个国 家内语言不 同，就可以 指定另一个 标识符，例 如，排序顺 序在电话簿 和词典之间 不同。
字段 <lg>、 <cy> 和 <m> 必须是类型 C 且长度必须 与表 TCP0C 的关键字段 长度相等。 表 TCP0C 是一个表格 ，从中进行 平台相关的 文本环境维 护。在语句 SET LOCALE 期间，系统 根据 TCP0C 中 的条目设置 文本环境。 除了内部传 送的平台特 性之外，用 SET 语句指定表 关键字。如 果 <lg> 等于 SPACE，则系统根 据用户主记 录设置文本 环境。如果 对于指定的 关键字在表 中无条目， 则系统将产生运行错误 。
文本环境影 响 ABAP/4 中依赖于字 符集的所有 操作。
有关该主题 的详细信息 ，参见 CONVERT TEXT 和 SET LOCALE LANGUAGE 的关键字文 档。
关于如何根 据语言排序 的示例，参 见*排序内表* (页 **Error! Not a valid link.**) 。

覆盖字符字 段

要用另一字 符字段覆盖 字符字段， 请使用 OVERLAY 语句，用法 如下：

语法
OVERLAY <c1> WITH <c2> [ONLY <str>].
该语句用 <c2> 的内容覆盖 字段 <c1> 中包含 <str> 中字母的所 有位置。<c2> 保持不变。如果省略 ONLY <str>， 则覆盖字段 <c1> 中所有包含 空格的位置 。
如果至少要 替换 <c1> 中的一个字 符，则将 SY-SUBRC 设置为 0。对于所 有其它情况 ，将 SY-SUBRC 设置为 4。如果 <c1> 比 <c2> 长，则只覆 盖 <c2> 中的长度。

```
DATA: T(10) VALUE 'a c e g i ',
      STRING LIKE T,
      OVER(10) VALUE 'ABCDEFGHIJ',
      STR(2) VALUE 'ai'.

STRING = T.
WRITE STRING.
WRITE / OVER.

OVERLAY STRING WITH OVER.
WRITE / STRING.

STRING = T.
OVERLAY STRING WITH OVER ONLY STR.
WRITE / STRING.
```

该过程的输 出如下：

```
a c e g i
ABCDEFGHIJ
aBcDeFgHiJ
```

```
A c e g I
```

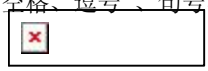
搜索字符串

要搜索特定 模式的字符 串，请使用 SEARCH 语句，用法 如下：

语法
SEARCH <c> FOR <str> <options>.
该语句在字 段 <c>中搜索<str>中的字符串 。如果成功 ，则将 SY-SUBRC 的返回代码 值设置为 0 并 将 SY-FDPOS 设置为字段 <c> 中该字符串 的偏移量。 否则将 SY-SUBRC 设置为 4。
搜索串 <str> 可为下列格 式之一：

<str>	目 的
<pattern>	搜 索 <pattern>（ 任何字符顺 序）。忽略 尾部空格。
.<pattern>.	搜 索 <pattern> ，但是不忽 略尾部空格 。
*<pattern>	搜 索以 <pattern> 结尾的词。
<pattern>*	搜 索以 <pattern> 开始的词。

单词之间用 空格、逗号 、句号、分 号、冒号、 问号、叹号 、括号、斜 杠、加号和 等号等分隔 。



```
DATA STRING(30) VALUE 'This is a little sentence.'.

WRITE: / 'Searched', 'SY-SUBRC', 'SY-FDPOS'.
ULINE /1(26).

SEARCH STRING FOR 'X'.
WRITE: / 'X', SY-SUBRC UNDER 'SY-SUBRC',
              SY-FDPOS UNDER 'SY-FDPOS'

SEARCH STRING FOR 'itt   '.
WRITE: / 'itt   ', SY-SUBRC UNDER 'SY-SUBRC',
                   SY-FDPOS UNDER 'SY-FDPOS'

SEARCH STRING FOR '.e .'.
WRITE: / '.e .', SY-SUBRC UNDER 'SY-SUBRC',
                  SY-FDPOS UNDER 'SY-FDPOS'.

SEARCH STRING FOR '*e'.
WRITE: / '*e ', SY-SUBRC UNDER 'SY-SUBRC',
                SY-FDPOS UNDER 'SY-FDPOS'.

SEARCH STRING FOR 's*'.
WRITE: / 's* ', SY-SUBRC UNDER 'SY-SUBRC',
                SY-FDPOS UNDER 'SY-FDPOS'.
```

该过程的输 出如下：

```
SEARCHED SY-SUBRC SY-FDPOS
X            4        0
itt          0       11
.e .         0       15
*e           0       10
s*           0       17
```

搜索字符字 段 <c> 的各种选项 （<options>） 如下
ABBREVIATED
在字段 <c> 中搜索包含 <str> 中指定字符 串的单词， 其中字符可 能被其它字 符隔开。单 词和字符串 的第一个字 母必须相同 。
STARTING AT <n1>
在字段 <c> 中搜索从 <n1> 开始的 <str> 。结果 SY-FDPOS 参照相对于 <n1> 的偏移量而 不是字段的 开始。
ENDING AT <n2>
在字段 <c> 搜索 <str> 直到位置 <n2>。

AND MARK
如果找到搜 索串，则将 搜索串中的 所有字符（ 和使用 ABBREVIATED 时的所有字 符）转换为 大写形式。

```
DATA: STRING(30) VALUE 'This is a fast first example.',
      POS TYPE I,
      OFF TYPE I.

WRITE / STRING.

SEARCH STRING FOR 'ft' ABBREVIATED.
WRITE: / 'SY-FDPOS:', SY-FDPOS.

POS = SY-FDPOS + 2.
SEARCH STRING FOR 'ft' ABBREVIATED STARTING AT POS AND MARK.
WRITE / STRING.
WRITE: / 'SY-FDPOS:', SY-FDPOS.
OFF = POS + SY-FDPOS -1.
WRITE: / 'Off:', OFF.
```

该过程的输 出如下：

```
This is a fast first example.
SY-FDPOS:    10
This is a fast FIRST example.
SY-FDPOS:    4
Off:         15
```

请注意，在 找到单词‘ fast’ 之后，为了 查找包含‘ ft’的第 二个单词， 必须在偏移 量 SY-FDPOS 上加2，然 后从位置 POS 开始查找。 否则，会再 次找到单词 ‘fast’ 。要获得‘ first’ 相对于字段 STRING 开始的偏移 量，从 POS 和 SY-FDPOS 计算。

获得字符串 长度

要决定字符 串到最后一 个字符而不 是 SPACE 的长度，请 使用内部函 数 STRLEN， 用法如下：

语法
[COMPUTE] <n> = STRLEN(<c>).
STRLEN 将操作数 <c> 作为字符数 据类型处理 ，而不考虑 其实际类型 。不进行转 换。
关键字 COMPUTE 可选。有关 内部函数的 详细信息， 参见 *使用数学函 数* （页 49） 。

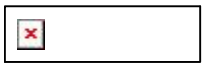
```
DATA: INT TYPE I,
      WORD1(20) VALUE '12345'.
      WORD2(20).
      WORD3(20) VALUE '   4         '.

INT = STRLEN( WORD1 ). WRITE   INT.

INT = STRLEN( WORD2 ). WRITE / INT.

INT = STRLEN( WORD3 ). WRITE / INT.
```

结果分别是 5，0 和 4。

压缩字段内 容

要删除字符 字段中多余 空格，请使 用 CONDENSE 语句，用法 如下：

语法
CONDENSE <c> [NO-GAPS].
该语句去除 字段 <c> 中的前导空 格并用一个 空格替换其 它空格序列 。结果是左 对齐单词， 每个单词用 空格隔开。 如果指定附 加的 NO-GAPS， 则去除所有 空格。



```
DATA: STRING(25) VALUE ' one  two   three    four',
      LEN TYPE I.

LEN = STRLEN( STRING ).
WRITE: STRING, '!'.
WRITE: / 'Length: ', LEN.

CONDENSE STRING.
LEN = STRLEN( STRING ).
WRITE: STRING, '!'.
WRITE: / 'Length: ', LEN.

CONDENSE STRING NO-GAPS.
LEN = STRLEN( STRING ).
WRITE: STRING, '!'.
WRITE: / 'Length: ', LEN.
```

该过程的输 出如下：

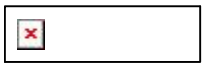
```
 one  two   three    four !
Length:          25
one two three four        !
Length:          18
onetwothreefour           !
Length:          15
```

请注意，字 段 STRING 的总长度保 持不变（惊 叹号！）， 但删除的空 格再次出现 在右边。

连接字符串

要将单个字 符串连接成 一体，请使 用 CONCATENATE 语句，用法 如下：

语法
CONCATENATE <c1> ... <cn> INTO <c> [SEPARATED BY <s>].
该语句连接 字符串 <c1> 与 <cn> 并将结果赋 给 <c>。
该操作忽略 尾部空格。
附加 SEPARATED BY <s> 允许指定字 符字段 <s>，它 放置在单个 字段间已定 义的长度中 。
如果结果符 合 <c>，则 将 SY-SUBRC 设置为 0。 然而，如果 结果必须被 截断，则将 SY-SUBRC 设 置为 4。



```
DATA: C1(10)  VALUE  'Sum',
      C2(3)   VALUE  'mer',
      C3(5)   VALUE  'holi ',
      C4(10)  VALUE  'day',
      C5(30),
      SEP(3)  VALUE ' - '.

CONCATENATE C1 C2 C3 C4 INTO C5.
WRITE C5.

CONCATENATE C1 C2 C3 C4 INTO C5 SEPARATED BY SEP.

WRITE / C5.
```

该过程的输 出如下：

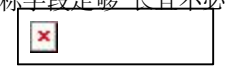
```
Summerholiday

Sum - mer - holi - day
```

在 C1 到 C5 之间，忽略 尾部空格。 分隔符 SEP 保留尾部空 格。

拆分字符串

要将字符串 拆分成两个 或更多小串 ，请使用 SPLIT 语句，用法 如下：

语法
SPLIT <c> AT <del> INTO <c1> ... <cn>.
该语句在字 符字段 <c> 搜索分界字 符串 <del>， 并将分界符 之前和之后 的部分放到 目标字段 <c1> ... <cn> 中。
要将所有部 分放到不同 目标字段中 ，必须指定 足够目标字 段。否则， 用字段 <c> 的剩余部分 填充最后目 标字段并包 含分界符。
如果所有目 标字段足够 长且不必截 断任何部分 ，则将 SY-SUBRC 设置为 0。 否则，将其 设置为 4。



```
DATA: STRING(60),
      P1(20) VALUE '++++++++++++++++++++',
      P2(20) VALUE '++++++++++++++++++++',
      P3(20) VALUE '++++++++++++++++++++',
      P4(20) VALUE '++++++++++++++++++++',
      DEL(3) VALUE '***'.
STRING = ' Part 1 *** Part 2 *** Part 3 *** Part 4 *** Part 5'.
WRITE STRING.

SPLIT STRING AT DEL INTO P1 P2 P3 P4.

WRITE / P1.
WRITE / P2.
WRITE / P3.
WRITE / P4.
```

该过程的输 出如下：

```
 Part 1 *** Part 2 *** Part 3 *** Part 4 *** Part 5

 Part 1

 Part 2

 Part 3

 Part 4 *** Part 5
```

请注意，字 段 P1 ...P4 的内容全部 被改写且用 尾部空格填 充。

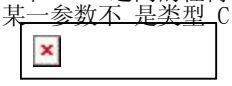
也可以将组 成原始串的 部分放到内 表中。

语法
SPLIT <c> AT <del> INTO <itab>.
对于字符串 的每一部分 ，系统添加 新表行（有 关内表的详 细信息，参 见*创建并处理内表*（页 Error! Not a valid link.））。

分配字符串 部分

MOVE 语句的以下 变体只使用 类型 C 字段：

语法
MOVE <c1> TO <c2> PERCENTAGE <p> [RIGHT].
将左对齐的 字符字段 <c1> 的百分比 <p>（或 如果用 RIGHT 选项指定， 则为右对齐 ）复制到 <c2>。
<p> 值可为 0 和 100 之间的任何 数。将要从 <F1> 复制的长度 取整为下一 个整数。
如果语句中 某一参数不 是类型 C，则忽略 参数 PERCENTAGE。



```
DATA C1(10) VALUE 'ABCDEFGHIJ',
     C2(10).

MOVE C1 TO C2 PERCENTAGE 40.

WRITE C2.

MOVE C1 TO C2 PERCENTAGE 40 RIGHT.

WRITE / C2.
```

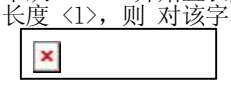
该过程的输 出如下：

```
ABCD

      ABCD
```

指定数据对 象的偏移量

在 ABAP/4 中，可以在 所有处理数 据对象的语 句中指定基 本数据对象 的偏移量值 。
为此，请在 语句中指定 数据对象名 称，如下所 示：

语法
<f>[+<o>][(<l>)]
对字段 <f> 中从 <o>+1 开始且长度 <l> 的部分执行 该语句的操 作。
如果未指定 长度 <l>，则 对该字段 <o> 和结尾之间 的所有位置 进行处理。



通常，必须 将偏移量 <o> 和长度 <l> 指定为无符 号数值。能 够在以下例 外情况下 可 以使用变量 动态指定：

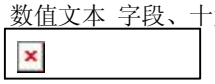
用 MOVE 或赋值运算 符给字段赋 值时
（参见*用指定偏移 量赋值* (页 42) ）

用 WRITE TO 语句向字段 写入值时
（ 参见*用指定偏移 量写入值* (页 45) ）

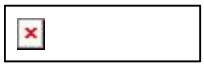
用 ASSIGN 将字段赋给 字段符号时
（参见使用字段符号）

用 PERFORM 将数据传送 给子程序时
（参见用参数传递数据）。

对字符字段 、数值文本 字段、十六 进制字段及 日期和时间 字段指定偏 移量是有意 义的。



对于类型 F、I 和 P 的数值字段 或字段串不 要使用指定 偏移量。它 们不能与文 字或文本符 号一同使用 （有关文本 符号的详细 信息，参见*文本符号* (页 **错误！链接无效。**) ）。



```
DATA TIME TYPE T VALUE '172545'.

WRITE TIME.
WRITE / TIME+2(2).
CLEAR TIME+2(4).
WRITE / TIME.
```

该过程的输 出如下：

172545

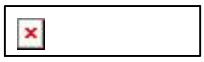
25

170000

首先，在 WRITE 语句中通过 指定偏移量 选择分钟数 。然后，通 过在清除语 句中指定偏 移量将分钟 数和秒数设 置为初始默 认值。

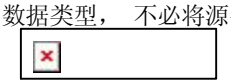
类型转换

可以将一种 数据类型的 数据对象内 容赋给另一 种数据类型 的数据对象 。在这种情 况下，所涉 及的数据类 型必须是可 转换的。



作为一般规 则，所有兼 容的数据类 型都是可转 换的
（有关兼容 数据类型的 详细信息， 参见*数据类型兼容性*（页 **错误！链接无效。**））。

对于兼容的 数据类型， 不必将源字 段值转换为 目标字段的 数据类型， 因为所有技 术属性都一 样。

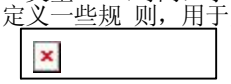


在 ABAP/4 中，定义好 转换规则之 后，两种不 兼容的数据 类型可相互 转换。

当且仅当两 种数据类型 可转换时， 可以用 MOVE 语句或赋值 运算符（＝ ）将一种数 据类型的数 据对象值赋 给另一种数 据类型的数 据对象。将 源对象值转 换为目标对 象的数据类 型。该转换 遵循系统中 定义的以及 在以下主题 中描述的规 则。
对于在数据 对象之间进 行赋值的所 有 ABAP/4 操作（例如 算术运算或 填充内表） ，与用 MOVE 语句一样， 系统都进行 所有必要的 类型转换。 如果两个字 段的数据类 型不可转换 ，但试图将 一个字段内 容赋给另一 字段，则系 统在语法检 查时报告错 误或产生运 行错误。
以下主题描 述 ABAP/4 中定义的不 兼容数据类 型之间的转 换规则。
对于不同对 象之间传送 数据的一些 ABAP/4 语句，数据 对象的排列 也起重要作 用。有关数 据对象排列 的信息，参 见

基本数据类 型的可转换 性

预定义的基本数据类型（页 **错误！链接无效。**）中的表列出 ABAP/4 中预定义的 八种数据类 型。
这些基本数 据类型之间 有 64 种可能的类 型组合。ABAP/4 支持所有自 动类型转换 和长度调整 ，类型 D （日期）和 类型 T （时间）字 段除外，它 们彼此无法 转换。
以下转换表 定义一些规 则，用于为 源字段和目 标字段的所 有可能组合 转换基本数 据类型。



总是按处理 无小数位的 类型 P 的相同方式 处理类型 I。所以， 提到类型 P 时，同样适 用于类型 I 字段。如果 为某一程序 设置“定点 算法”属性 （*指定程序属性*（页 **错误！链接无效。**）），则系 统根据小数 位数截取类 型 P 字段或用零 进行填充。

源类型字符

源类型 C 的转换表

目标	**转换**
C	左 对齐目标字 段中的数据 。如果字段 太长，则用 空格从右进 行填充。如 果太短，则在右边将其 内容截断。
D	字符字段 应包含 YYYYMMDD 格式的 8 字符日期。
F	源字段内 容必须是*文字*（页 **错误！链接无效。**）中 所述的类型 F 字段的有效 表示。
N	只 复制源字段 中的数字。 右边被压缩 ，左边用零 填充。
P	源 字段必须包 含十进制数 的表示，例 如，用可选 符号和只有 一个小数点 的数字序列 。源字段可 包含空格。 如果目标字 段太短，则 产生溢出。 这可能导致 系统终止程序。
T	字符 字段应包含 HHMMSS 格式的 6 字符时间。
X	既然字符 字段应包含 十六进制字 符串，则唯 一有效字符 是 0、1、2、 3、4、5、 6、7、8、 9、A、B、 C、D、E、 F。该字符 串压缩为十 六进制数， 左对齐传送 并且右边用 零填充或截 断。

源类型日期

源类型 D 的转换表

目标	**转换**
C	未 转换左对齐 传送日期。

D	未转换传 送。
F	将日 期转换为压 缩数，然后 将压缩数转 换为浮点数 （参见相关 表）。
N	将日期转 换为字符字 段，然后将 字符字段转 换为数字文 本字段（参 见相关表） 。
P	将日期 转换为从 01.01.0001 开始的天数 。
T	不支持 。导致语法 检查错误信 息或导致运 行错误。
X	将 日期转换为 以十六进制 数表示的从 01.01.0001 开始的天数 。

源类型浮点 数

源类型 F 的转换表

目标	转换
C	将 浮点数转换 为 <mantissa>E<exponent>， 然后传送给 字符字段。 如果尾数不 是零，则尾 数值界于 1 和 10 之间。指数 总是带符号 的。如果目 标字段太短 ，则尾数被 取整。字符 字段长度至 少应为 6 字节。
D	将 源字段转换 为压缩数， 然后将压缩 数转换为日 期字段（参 见相关表） 。
F	未转换 传送。
N	将 源字段转换 为压缩数， 然后将它转 换为数字文 本字段（参 见相关表） 。
P	将浮点 数转换为整 数或定点值 ，并在必要 时将取整。
T	将源字段 转换为压缩 数，然后将 压缩数转换 为时间字段 （参见相关 字段）。
X	将 源字段转换 为压缩数， 然后将压缩 数转换为十 六进制数（ 参见相关表 ）。

源类型数字 文本

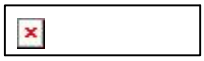
源类型 N 的转换表

目标	转换
C	将 数值字段看 作字符字段 处理。保留 前导零。
D	将 数值字段转 换为字符字 段，然后将 字符字段转 换为日期字 段（参见相 关表）。
F	将 数值字段转 换为压缩数 ，然后将压 缩数转换为 浮点数（参 见相关表） 。
N	右对齐 传送数值字 段，并在左 边用零填充 或截断。
P	压 缩数值字段 ，然后右对 齐带负号传 送。如果目 标字段太短 ，则程序可 能终止。
T	将 数值字段转 换为字符字 段，然后将 字符字段转 换为时间字 段（参见相 关表）。
X	将 数值字段转 换为压缩数 ，然后将压 缩数转换为 十六进制数 （参见相关 表）。

源类型压缩 数

源类型 P 的转换表

目标	转换
C	如 果需要小数 点，则将压 缩字段右对 齐传送给字 符字段。第 一个位置留 着符号位。前导零以空 格出现。如 果目标字段 太短，对于 正数则省略 符号。如果 仍不够，则 在左边截断 该字段。 ABAP/4 用星号（*） 标识截断。 如果要在字 符字段中出 现前导零，则请使用 UNPACK ，而不是 MOVE
D	压 缩字段值表 示从 01.01.0001 起的天数并 转换为 YYYYMMDD 格式的日期 。
F	接受压 缩字段并作 为浮点数传 送。
N	如果 必要，请取 整压缩字段 ，解压缩字 段，然后右 对齐传送。 省略符号。 如果必要，则在左边用 零填充目标 字段。
P	右 对齐传送压 缩字段。如 果目标字段 太短，则产 生溢出。
T	压 缩字段值代 表从午夜起 的秒数并转 换为 HHMMSS 格式的时间 。
X	如果必 要，则将压 缩字段取整 ，然后将其 转换为十六 进制数。用 2 的补数表示 负数。如果 目标字段太 短，则在左 边截断该数 。



总按处理不 带小数位的 类型 P 的方式处理 类型 I。

源类型时间

类型 T 的转换表

目标	转换
C	未 转换将日期 左对齐传送 。
D	不支持 。导致语法 检查出错信 息并导致运 行错误。
F	将 时间转换为 压缩数，然 后将压缩数 转换为浮点 数（参见相 关表）。
N	将 日期转换为 字符字段， 然后将字符 字段转换为 数字文本字 段（参见相 关表）。
P	将 日期转换为 从午夜起的 秒数。
T	未 转换传送。
X	将日期转 换为按十六 进制格式的 从午夜起的 秒数。

源类型十六 进制数

源类型 X 的转换表

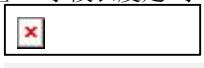
目标	转换
C	将 十六进制字 段中的值转 换为十六进 制字符串， 左对齐传送 给目标字段 并用零填充 。
D	源字段 值表示从 01.01.0001 起的天数并 转换为 YYYYMMDD 格式的日期 。
F	将源字 段转换为压 缩数，然后 将压缩数转 换为浮点数 （参见相关 表）。
N	将 源字段转换 为压缩数， 然后将压缩 数转换为数 字文本字段 （参见相关 表）。
P	将 源字段的值 解释为十六 进制数。转 换为压缩的 十进制数， 然后按右对 齐传送给目标字段。如 果十六进制 字段长于 4 字节，只转 换后四字节 。如果太短 ，则产生运 行错误。
T	源 字段值表示 从午夜起的 秒数并转换 为 HHMMSS 格式的时间 。
X	左对齐 传送值，如 有必要，用 X'00' 进行填充。

字段串的可 转换性

对于字段串 ，在下列方 面存在区别 ：

兼容的字符 串
根据一般规 则，兼容的 字段串总是 可转换的。
MOVE 语句一个组 件一个组件 的传送可兼 容的字段串 。

不兼容的字 段串和基本 字段
ABAP/4 包含以下转 换规则:
字段串到 不兼容的字 段串
基本字段 到字段串
字段串到 基本字段
对于上述各 种情况，系 统首先将所 有字段串转 换为类型 C 字段，然后 在两种剩余 的基本字段 间进行转换 。类型 C 字段长度是 字段串组件 长度之和。



对齐字段串 之后（参见*对齐数据对 象* (页 386)），也将填 充字段添至 类型 C 字段长度中 。

不对齐以下 字段串：

应该记住该 规则用于所 有字段串的 所有操作。 但只适用于 包含**非内表** 作为组件的字段串。

如果将长字 段串转换为 短字段，则 忽略多余部 分。如果将 短字段串转 换为长字段 ，则根据其 类型不对结 尾部分进行 初始化，而 用空格填充 。
例如，如果 短字段串的 结构对应于 长字段的开 头的结构， 则在不兼容 字段串之间 的转换将很 有意义。

```
DATA: BEGIN OF FS1,
         INT      TYPE I            VALUE 5,
         PACK     TYPE P DECIMALS 2 VALUE '2.26',
         TEXT(10) TYPE C            VALUE 'Fine text',
         FLOAT    TYPE F            VALUE '1.234e+05',
         DATA     TYPE D            VALUE '19950916',
      END OF FS1.

DATA: BEGIN OF FS2,
         INT      TYPE I            VALUE 3,
         PACK     TYPE P DECIMALS 2 VALUE '72.34',
         TEXT(5)  TYPE C            VALUE 'Hello',
      END OF FS2.

WRITE: / FS1-INT, FS1-PACK; FS1-TEXT, FS1-FLOAT, FS1-DATE.
WRITE: / FS2-INT, FS2-PACK, FS2-TEXT.

MOVE FS1 TO FS2.
WRITE: / FS2-INT, FS2-PACK, FS2-TEXT.
```

该过程的输 出如下：

```
        5          2.26 Fine text   1.234000000000000E+05 09161995

        3         72.34 Hello

        5          2.26 Fine
```

该例子定义 两个不同字 段串：FS1 和 FS2。最 初两个组件 的字段串结 构相同。将 FS1 赋给 FS2 之后，只有 最初两个组 件的结果好 象是一个组 件一个组件 进行移动的 。用 FS1-TEXT 的最初五个 字符填充 FS2-TEXT。 省略 FS1 的所有其它 位置。

用不兼容字 段串之值填 充的字段串 的数字组件 可以包含导 致运行错误 的无意义值或非法值。

用内表作为 组件的字段 串
包含内表作 为组件的字 段串仅当它 们兼容时才 可转换。

内表的可转 换性

内表只能转 换为其它内 表。内表不 能转换为字 段串或基本 字段。如果 内表的行类 型是可转换 的，则内表 可转换。
这会产生以 下结果：

如果定义 行类型的内 表是可转换 的，则将内 表作为行类 型的内表是 可转换的。
只有当字 段串是**可兼 容的**时，行 类型是将内 表作为组件 的字段串的 内表才是可 转换的（参见*用内表作为 组件的字段 串*（页 66））。

有关内表的 详细信息， 参见*创建并处理内表*（页 Error! Not a valid link.）。

对齐数据对 象

如果字段是 类型 I 或 F，则将它 们**对齐**。如 果字段串包 含类型 I 或类型 F 组件，也将 它们对齐并 在其前部插 入填充字段 。出现此类 字段或结构 对齐的原因 在于类型 I 和类型 F 字段占用内 存中特定平 台相关地址 。例如，类 型 I 字段的地址 必须能被 4 整除，类型 F 字段的地址 必须能被 8 整除。系统在进行 说明期间通 常自动对齐 字段和结构 。

只须注意下 列情况中的 对齐：

- 如果将字 段或结构作 为参数传递 给需要另一 类型作为参 数的子程序 （参见用参数传递数据）
- 如果更改 字段符号的 类型（参见定义字段符号的数据类型 ）或定义结 构化的字段 符号（参见定义结构化的字段符号 ）
- 在使用 Open SQL 语句时，工 作区域与表 工作区域类 型不同（参 见数据库表和SQL 概念）。

第七章　　控制 ABAP/4 程序流

概览

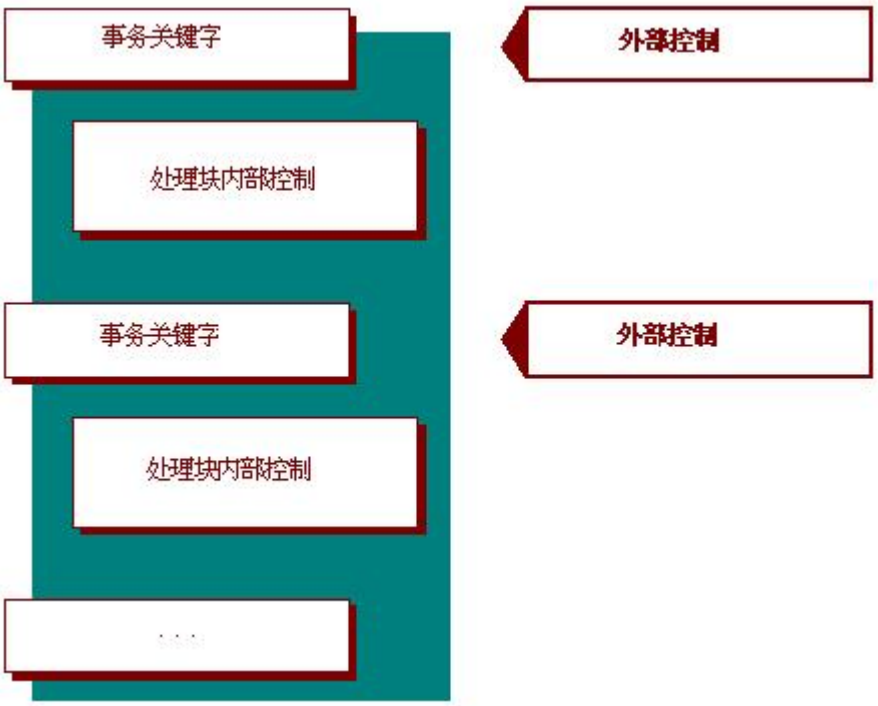
内容

要根据一定 条件执行程 序组件，或 要将重复语 句序列组合 在循环中， 可以使用 ABAP/4 提供的用于 控制程序流 的标准关键 字。

ABAP/4 也包含重要 的附加特征 。

ABAP/4 程序流可以 内部控制和 外部控制。 ABAP/4 程序流的**内 部控制**和**外 部控制**之间的差别在*ABAP/4中 流控制的概 念* (页 62) 中 解释。

本节主要讨 论**内部流控 制。**为此， 可使用其他 编程语言（ 例如，C、 FORTRAN、 PASCAL 等等）中熟 悉的标准控 制关键字。



ABAP/4中 流

于 控

分支
循环

制。

内部控
外部控
（例如
句处理

输入
行语

属 于
归属
相干。

生 时
照前

示例

假设当用户 选择一行后 ，生成列表 且提供一些 细分设备的 报表程序必 须作出反应（详细信息 ，参见 交互列表）。需要为 该事件处理 的代码必须 插入事件关 键字 AT LINE-SELECTION 之后。

```
AT LINE-SELECTION.
   MOVE 'X' TO FLAG.
   .....
```

无论何时用 户通过单击 鼠标或按 F2 键在列表中 选择一个项 目时，AT LINE-SELECTION 和下一个事 件关键字之 间的所有语 句将被处理

关于**外部控 制**的详细信 息，如事件 及其如何与 ABAP/4 程序相互作 用的信息， 参见 C 通过事件控制 ABAP/4 程序流。

该节说明如 何使用**外部 控制**控制处 理块内的程 序流。

要控制 ABAP/4 程序中的内 部流，请遵 循结构化编 程原则并将 程序模块划 分为单个逻辑相关语句 块（这些组 成控制结构 ）。其中的 每个语句块 都执行主任 务的一部分 。

注释

要使程序易 于阅读，应 该缩排控制 结构中的语 句块。出于 布局需要， 可以使用 ABAP/4 编辑器功能 “编辑 -> 插入语句...” 和“程序 -> 整齐打印程 序”(详细 信息，参见 *ABAP/4 程序布局* (页 2 - 6))。

可以用 IF、CASE、 DO 和 WHILE 之类的关键 字控制处理 块中不同语 句之间的程 序流。这些 语句允许编 制条件和无 条件的分支 和循环。条 件使用逻辑 表达式，可 以是真，也 可以为假。

编程逻辑表 达式

使用条件控 制程序中的 内部流。要 用公式指定 条件，请使 用比较数据 字段的逻辑 表达式，如 下所示：

语法

```
.... <F1> <operator> <F2> ...
```

该表达式比 较两个字段 。可能为真 ，也可能为 假。在带关 键字 IF、 CHECK 和 WHILE 的条件语句 中使用逻辑 表达式。

根据操作数 <F1> 和 <F2> 的数据类型 ，可以使用 不同的逻辑 运算符。允 许进行：

除上述比较 之外，还可 以执行测试 以检查数据 字段是否完 全满足一定 条件。可以 使用这些测 试：

另外，可以 将几个逻辑 表达式组合 为一个简单 逻辑表达式 。

比较所有的 字段类型

要比较所有 的字段类型 ，可以在逻 辑表达式中 使用下列运 算符：

<运算符>	含 义
EQ	等于

=	等于
NE	不 等于
<>	不 等于
><	不 等于
LT	小 于
<	小于
LE	小 于等于
<=	小 于等于
GT	大 于
>	大于
GE	大 于等于
>=	大 于等于

操作数可以 是数据库字 段、内部字 段、文字或 常数。

除基本字段 外，还可以 将结构数据 类型和上表 中的运算符 结合起来作 为操作数。 字段串逐个 组件进行比 较，嵌套的 结构分为基 本的字段。 关于比较内 表的详细信 息，参见 *比较内表*（页 **Error! Not a valid link.**）。

如果有意义 ，可以对不 同数据类型 的字段进行 比较。如果 字段可转换 ，就可以进 行比较。在 比较之前， 系统将按照 下列层次规 则执行自动 类型转换（ 参见 *键入转换*（页 6－36）：

1. 如果 操作数之一 是浮点数（ 类型 F），则系 统将其它操 作数转换为 类型 F。
2. 如果 操作数之一 是压缩字段 （类型 P），则系 统将其它操 作数转换为 类型 P。
3. 如果 操作数之一 是日期字段 （类型 D）或时间 字段（类型 T），则系 统将其它操 作数转换为 类型 D 或 T。不支持 日期和时间 字段之间的 比较，这会 导致程序中 断。
4. 如果 操作数之一 是字符字段 （类型 C）且其它 操作数是十 六进制字段 （类型 X），则系 统将类型 X 的操作数转 换为类型 C。
5. 如果 操作数之一 是字符字段 （类型 C），其它 操作数为数 字字段（类 型 N），则系 统将这两种 操作数都转 换为类型 P。

示例

```
DATA: F    TYPE F VALUE '100.00',
      P    TYPE P VALUE '50.00' DECIMALS 2,
      I    TYPE I VALUE '30.00'.

WRITE 'The following logical expressions are true:'.

IF F >= P .
   WRITE: / F,'>=',P.
ELSE.
   WRITE: / F,'<',P.
ENDIF.

IF I EQ P .
   WRITE: / I,'EQ',P.
ELSE.
   WRITE: / I,'NE',P.
ENDIF.
```

这生成如下 输出：

```
The following logical expressions are true:

 1.000000000000000E+02 >=             50.00

        30 NE             50.00
```

这里，在 IF 语句中使用 两个逻辑表 达式。如果 逻辑表达式 为真，则屏 幕上显示出来。如果逻 辑表达式为 假，则将相 反表达式显 示在屏幕上 。

比较字符串 和数字串

要比较字符 串（类型 C）和数字 文本（类型 N），可以 在逻辑表达 式中使用下 列运算符。

<运算符>	含 义
CO	仅包 含
CN	不仅 包含
CA	包 含任何
NA	不 包含任何
CS	包 含字符串
NS	不 包含字符串
CP	包含模 式
NP	不包 含模式

因为除类型 N 和 C 外，系统不 能执行任何 其它类型转 换，所以， 在进行包含 这些运算之 一的比较时 ，操作数应 该是类型 N 或 C。

运算符的功 能如下：

CO（仅包含）

如果 <F1> 仅包含 <F2> 中的字符， 则逻辑表达 式

<F1> CO <F2>

为真。该比 较区分大小 写，并包括 尾部空格。如果比较结 果为真，则 系统字段 SY-FDPOS 包括 <F1> 的长度。如 果为假，则 SY-FDPOS 包含 <F1> 中第一个未 在 <F2> 内出现的字符的偏移量 。

CN（不仅包含 ）

如果 <F1> 还包含 <F2> 之外的其他 字符，则逻 辑表达式

<F1> CN <F2>

为真。该比 较区分大小 写，并包括 尾部空格。如果比较结 果为真，则 系统字段 SY-FDPOS 包含 <F1> 中第一个未 同时在 <F2> 中出现的字 符的偏移量 。如果为假 ，SY-FDPOS 包含 <F1> 的长度。

CA（包含任何 ）

如果 <F1> 至少包含 <F2> 的一个字符 ，则逻辑表 达式

<F1> CA <F2>

为真。该比 较区分大小 写。如果比 较结果为真 ，则系统字 段 SY-FDPOS 包含 <F1> 中第一个也 在 <F2> 中出现的字 符的偏移量 。如果为假 ，SY-FDPOS 包含 <F1> 的长度。

NA（不包含任 何）

如果 <F1> 不包含 <F2> 的任何字符 ，则逻辑表 达式

<F1> NA <F2>

为真。该比 较区分大小 写。如果比 较结果为真 ，则系统字 段 SY-FDPOS 包含 <F1>的 长度。如果 为假，则 SY-FDPOS 包含 <F1> 中在 <F2> 内出现的第 一个字符的 偏移量。

CS（包含字符 串）

如果 <F1> 包含字符串 <F2>， 则逻辑表达 式

<F1> CS <F2>

为真。忽略 尾部空格并 且比较不区 分大小写。 如果比较结 果为真，则 系统字段 SY-FDPOS 包含 <F2> 在 <F1> 中的偏移量 。如果为假 ， SY-FDPOS 包含 <F1> 的长度。

NS (不包含字 符串)

如果 <F1> 不包含字符 串 <F2>， 则逻辑表达 式

<F1> NS <F2>

为真。忽略 尾部空格且 比较不区分 大小写。如 果比较为真 ， 系统字段 SY-FDPOS 包含 <F1> 的长度。如 果为假，系 统字段 SY-FDPOS 包含 <F2> 在 <F1> 中的偏移量 。

CP (包含模式)

如果 <F1> 包含模式 <F2>， 则逻辑表达 式

<F1> CP <F2>

为真。如果 <F2> 属于类型 C，则可以 在 <F2> 中使用下列 通配符：

- * 用于任何字 符串
- + 用于任何单 个字符

忽略尾部空 格且比较不 区分大小写 。如果比较 结果为真，系统字段 SY-FDPOS 包含 <F2> 在 <F1> 中的偏移量 。如果为假 ， SY-FDPOS 包含 <F1> 的长度。

如果要对 <F2> 中的特殊字 符进行比较 ，请将换码 字符 # 放到其前面 。可以使用 换码字符 # 指定

- 大小写字 符
- 通配符 "*"(输 入 #*)
- 通配符 "+" (输入 #+)
- 换码符号 本身 (输入 ##)
- 字符串结 尾的空格 (输入 #___)

NP (不包含模 式)

如果 <F1> 不包含模式 <F2>， 则逻辑表达 式

<F1> NP <F2>

为真。在<F2>中 ，可以使用 与 CP 相同的通配 符和换码字 符。

忽略尾部空 格且比较不 区分大小写 。如果比较 结果为真， 则系统字段 SY-FDPOS 包含 <F1>. 的长度，如 果为假，SY-FDPOS 包含 <F2> 在 <F1> 中的偏移量 。

示例

```
DATA: F1(5) TYPE C VALUE <F1>,
      F2(5) TYPE C VALUE <F2>.
IF F1 <operator> F2.
   WRITE: / 'Comparison true, SY-FDPOS=', SY-FDPOS.
ELSE.
   WRITE: / 'Comparison false, SY-FDPOS=', SY-FDPOS.
ENDIF.
```

下表列出该 程序的执行 结果，取决 于所用的运 算符和 F1 / F2 字段。

<F1>	<operator>	<F2>	Result	SY-FDPOS
'BD '	CO	'ABCD '	真	5
'BD '	CO	'ABCDE'	假	2
'ABC12'	CN	'ABCD '	真	3
'ABABC'	CN	'ABCD '	假	5
'ABcde'	CA	'Bd '	真	1
'ABcde'	CA	'bD '	假	5
'ABAB '	NA	'AB '	假	0

'ababa'	NA	'AB '	真	5
'ABcde'	CS	'bC '	真	1
'ABcde'	CS	'ce '	假	5
'ABcde'	NS	'bC '	假	1
'ABcde'	NS	'ce '	真	5
'ABcde'	CP	'*b*'	真	1
'ABcde'	CP	'*#b*'	假	5
'ABcde'	NP	'*b*'	假	1
'ABcde'	NP	'*#b*'	真	5

二进制位结 构的比较

要将逻辑表 达式初始操 作数第一字 节的二进制 位结构与第 二操作数的 进行比较， 请使用下列 操作符。

<运算符>	含 义
O	二进制 位是 1
Z	二进制 位是 0
M	混合二 进制位

第二 操作数的长 度应为一个 字节。

没有必要对 第二操作数 使用长度为 1 的十六进制 字段（类型 X），但却 较方便，因 为其长度为 一个字节且 数字值直接 与二进制位 结构相关。

操作符功能 如下：

O（二进制位 是 1）

如果 <hex> 中二进制位 是 1 的位置，在 <f> 中是 1，则逻辑 表达式

<f> O <hex>

为真。

Z（二进制位 是0）

如果 <hex> 中二进制位 是 1 的位置，在 <f> 中是 0，则逻辑 表达式

<f> Z <hex>

为真。

M（混合二进 制位）

如果从 <hex> 中二进制位 是 1 的位置起， <f> 中至少一个 是 1，一个是 0，则逻辑 表达式

<f> M <hex>

为真。

示例

```
DATA: C VALUE 'C',
      HEXDEC TYPE X,
      I TYPE I.

HEXDEC = 0.

DO 256 TIMES.

   I = HEXDEC.

   IF C 0 HEXDEC.
      WRITE: / HEXDEC, I.
   ENDIF.

   HEXDEC = HEXDEC + 1.

ENDDO.
```

产生下列输 出：

```
00          0
01          1
02          2
03          3
40         64
41         65
42         66
43         67
```

这里，使用 运算符 0 将字符‘C’ 的二进制位 结构与所有 ‘0’与‘ FF’（10 进制为 255）之 间的 16 进制数进行 比较。HEXDEC 的10 进制值 I ，在HEXDEC 分配到I 期间，通过 使用自动类 型转换确定 。如果比较 结果为真， 则在屏幕上 显式 16 进制数及其 10 进制值。下 表列出这些 数的二进制 位结构。

16 进制	10 进制	二进制 位结构
00	0	00000000
01	1	00000001
02	2	00000010
03	3	00000011
40	64	01000000
41	65	01000001
42	66	01000010
43	67	01000011

字 符‘C’的 二进制位结 构由其 ASCII 码 67 为当前硬件 平台进行定 义。与用户 看到的一样 ，列表输出 中出现的数 字，填写 1 的二进制位 位置与 67 的二进制位 结构一样。

检查字段是 否属于某一 范围

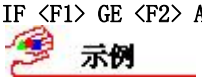
要检查值是 否属于特定 范围，请使 用下列带有 BETWEEN 参数的逻辑 表达式：

语法

```
.... <F1> BETWEEN <F2> AND <F3> .....
```

如果 <F1> 在 <F2> 和 <F3> 之间的范围 内发生，则 表达式为真 。它是下列 逻辑表达式 的短格式：

```
IF <F1> GE <F2> AND <F1> LE <F3>.
```



```
DATA: NUMBER TYPE I,
      FLAG.

....

NUMBER = ...

....

IF NUMBER BETWEEN 3 AND 7.
   FLAG = 'X'.
ELSE.
   FLAG = ' '.
ENDIF.
```

这里，如果 NUMBER 的内容在 3 和 7 之间出现， 字段 FLAG 设置为“X” 。

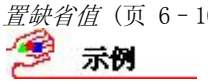
检查初始值

要检查字段 是否设置为 初始值，请 如下使用带 有 IS INITIAL 参数的逻辑 表达式。

语法

```
.... <f> IS INITIAL .....
```

如果 <f> 包含本身类 型的初始值 ，则表达式 为真。通常 情况下，任 何字段，基 本的或结构 化的（字符 串和内表） ，在 CLEAR <f> 语句执行后 ，都包含其 初始值（参 见 *重置缺省值* （页 6－10） ）。



```
DATA FLAG VALUE 'X'.

IF FLAG IS INITIAL.
   WRITE / 'Flag is initial'.
ELSE.
   WRITE / 'Flag is not initial'.
ENDIF.

CLEAR FLAG.

IF FLAG IS INITIAL.
   WRITE / 'Flag is initial'.
ELSE.
   WRITE / 'Flag is not initial'.
ENDIF.
```

这产生如下 输出：

```
Flag is not initial

Flag is initial.
```

这里，DATA 语句之后， 字符串 FLAG 不包含初始 值，因为用 VALUE 参数设置为 ‘X’ 。执 行 CLEAR 语句之后， 将其重置为 初始值。

检查选择条 件

要检查字段 内容是否与 选择表中的 选择条件匹 配，可以如 下使用带有 IN 参数的逻辑表达式：

语法

```
... <f> IN <seltab> ....
```

如果字段 <f> 内容的符合 选择表 <seltab> 中的条件， 则表达式为 真。

关于选择条 件的详细信 息，参见 使用选择标准。

关于逻辑表 达式中检查 选择条件的 详细信息， 包括示例， 参见 使用逻辑表达式中的选择表。

组合几个逻 辑表达式

通过使用逻 辑连接运算 符 AND、OR 和 NOT，可 以将几个逻 辑表达式组 合为单个表 达式：

- 要将几个 逻辑表达式 组合为单个 表达式，且 该表达式仅 当所有组件 表达式为真 时才为真，则表达式之 间要用 AND 连接。
- 要将几个 逻辑表达式 组合为单个 表达式，且 只要其中一 个组件表达 式为真时， 该表达式即为真，则表 达式之间要 用 OR 连接。
- 要转化逻 辑表达式的 结果，请在 其前面指定 NOT。

NOT 优先于 AND，AND 优先于 OR。但是 ，可以用任 何括弧组合 指定处理顺 序。由于 ABAP/4 将括弧解释 为单字，前 面或后面必 须至少有一 个空格。

ABAP/4 从左到右处 理逻辑表达 式。如果确 定组件表达 式之一是真 或假，就不 再执行该组 件中其余的 比较或检查 。这意味着 采取这种方 法组织逻辑 表达式可以 提高性能，就是将经常 为假的比较 放置在 AND 链的开头， 而将费时的 比较，如字 符串查找放 等到最后。

示例

```
DATA: F    TYPE F VALUE '100.00',
      N(3) TYPE N VALUE '123',
      C(3) TYPE C VALUE '456'.

WRITE 'The following logical expression is true:'.

IF ( C LT N ) AND  ( N GT F ).
   WRITE: / '(',C,'lt',N,') AND (',N,'gt',F,')'.
ELSE.
   WRITE: / '(',C,'ge',N,') OR (',N,'le',F,')'.
ENDIF.
```

这产生如下 输出：

下列逻辑表达式为真：

(456 ge 123) OR (123 le 1.000000000000000E+02)

在本例中， 在 IF 语句中使用 逻辑表达式 。如果逻辑 表达式为真 ，则将其在 屏幕上显示 出来。如果 为假，则屏 幕上出现相 反的表达式 。

编程分支和 循环

可以在程序 中定义条件 和无条件分 支和循环。 为此，ABAP/4 提供了几个 在下列主题 中描述的语 句。

分支

循环

使用 IF 的条件分支

IF 语句允许依 据条件将程 序流转到特 定的语句块 中。该语句 块包括 IF 语句及其后 面的 ELSEIF、 ELSE 或 ENDIF 之间的所有 命令。

语法

```
IF <condition1>.
   <statement block>
ELSEIF <condition2>.
   <statement block>
ELSEIF <condition3>.
   <statement block>
.....
ELSE.
   <statement block>
ENDIF.
```

如果第一个 条件是真， 系统将执行 所有语句直 到第一个语 句块结束， 然后继续处 理 ENDIF 语句之后的 程序。要采 用选择性条 件，可以使 用 ELSEIF 语句。如果 第一个条件 是假，系统 使用与 IF 语句一样的 方式处理后 面的 ELSEIF 语句。如果 IF 或 ELSEIF 条件都为假 ，则执行 ELSE 开始的语句 块。最后的 语句块必须 用 ENDIF 结束。

要用公式表 达 IF 和 ELSEIF 语句中的条 件，可以使 用任何 *编程逻辑表 达式* (页 68) 描述的逻辑 表达式。

注释

ABAP/4 允许无限地 嵌套 IF - ENDIF 语句块，但 是必须在相 同的处理块 中终止。换句话说，就 是 IF - ENDIF 块不能包含 事件关键字 。

示例

```
DATA: TEXT1(30) VALUE 'This is the first text',
      TEXT2(30) VALUE 'This is the second text',
      TEXT3(30) VALUE 'This is the third text',
      STRING(5) VALUE 'eco'.

IF TEXT1 CS STRING.
   WRITE / 'Condition 1 is fulfilled'.
ELSEIF TEXT2 CS STRING.
   WRITE / 'Condition 2 is fulfilled'.
ELSEIF TEXT3 CS STRING.
   WRITE / 'Condition 3 is fulfilled'.
ELSE.
   WRITE / 'No condition is fulfilled'.
ENDIF.
```

产生如下输 出：

```
Condition 2 is fulfilled.
```

这里，第二 个逻辑表达 式 TEXT2 CS STRING 是真，因为 字符串“eco” 存在于 TEXT2 中。

使用 CASE 的条件分支

要根据特殊 数据字段的 内容执行不 同的语句块 ，可以如下 使用 CASE 语句：

语法

```
CASE <f>.
   WHEN <F1>.
        <statement block>
```

```
   WHEN <F2>.
        <statement block>
   WHEN <F3>.
        <statement block>
   WHEN ...
   ......
   WHEN OTHERS.
        <statement block>
ENDCASE.
```

系统执行 WHEN 语句之后的 语句块，如 果 <f> 内容等于 <f_i> 的内容，且 继续处理 ENDCASE 语句后面的 语句。如果 <f> 的内容不等 于 <f_i> 的内容，则 执行选项 WHEN OTHERS 后面的语句 块。最后的 语句块必须 用 ENDCASE 结束。

使用 CASE 的条件分支 是带 IF 的类似处理 的短格式：

```
IF <f> = <F1>.
   <statement block>
ELSEIF <f> = <F2>.
   <statement block>
ELSEIF <f> = <F3>.
   <statement block>
ELSEIF <f> = ...
   ...
ELSE.
   <statement block>
ENDIF.
```

注释

在 ABAP/4 中，可以嵌 套 CASE - ENDCASE 块，且可以 与 IF - ENDIF 结合使用， 但必须在同 一处理块中 终止。

示例

```
DATA: TEXT1   VALUE 'X',
      TEXT2   VALUE 'Y',
      TEXT3   VALUE 'Z',
      STRING  VALUE 'A'.

CASE STRING.
  WHEN TEXT1.
     WRITE: / 'String is', TEXT1.
  WHEN TEXT2.
     WRITE: / 'String is', TEXT2.
  WHEN TEXT3.
     WRITE: / 'String is', TEXT3.
  WHEN OTHERS.
     WRITE: / 'String is not', TEXT1, TEXT2, TEXT3.
ENDCASE.
```

产生如下输 出：

```
String is not X Y Z
```

这里，执行 WHEN OTHERS 后面的语句 块，因为 STRING 的内容“A” 不等于“X” 、“Y”或 “Z”，。

使用 DO 的无条件循 环

如果想要多 次执行语句 块，则可以 如下使用 DO 语句编程循 环：

语法

```
DO [<n> TIMES] [VARYING <f> FROM <F1> NEXT <F2>].
   <statement block>
ENDDO.
```

在发现 EXIT、 STOP 或 REJECT 语句之前， 系统继续执 行由 DO 引导、ENDDO 结束的语句 块（参见 *终止循环* (页 63) ）。

可以使用 TIMES 选项限制循 环次数。<n> 可以是文字 或变量。如 果 <n> 是 0 或负数，系 统不执行该 循环。

系统字段 SY-INDEX 中包含已处 理过的循环 次数。

警告

使用 DO 语句时要避 免死循环。 如果不使用 TIMES 选项，则在 语句块中至 少应包含一 个 EXIT、 STOP 或 REJECT 语句，以便 系统能够退 出循环。

示例

本例显示 DO 循环的基本 格式。

```
DO.
WRITE SY-INDEX.
   IF SY-INDEX = 3.
      EXIT.
   ENDIF.
ENDDO.
```

产生如下输 出：

```
1          2          3
```

这里，处理 3 次循环，然 后在 EXIT 语句后退出 循环。

可以任意嵌 套 DO 循环，也可 以与其他循 环组合使用 。

示例

本例显示 2 个嵌套循环 ，都使用 TIMES 选项。

```
DO 2 TIMES.
   WRITE SY-INDEX.
   SKIP.
   DO 3 TIMES.
      WRITE SY-INDEX.
   ENDDO.
   SKIP.
ENDDO.
```

产生如下输 出：

```
1

1          2          3

2

1          2          3
```

外部循环执 行 2 次。每次执 行外部循环 时，内部循 环都执行 3 次。注意系 统字段 SY-INDEX 记录每个循 环各自的循 环次数。

可以使用 VARYING 选项在每次 循环中给变 量 <f> 重新赋值。 <F1>、 <F2>、 <F3>、 ... 必需是内存 中类型相同 和长度相等 的一系列等 距字段。第 一次循环中 ，将 <F1> 分配给 <f>，第 二次循环中 ，将 <F2> 分配给 <f>，以 此类推。可 以在一个 DO 语句中使用多个 VARYING 选项。

如果在 DO 循环中改变 控制变量 <f>，则 系统将自动 改变相应的 字段 <f_i>。

注释

应保证循环 次数不超过 涉及到的变 量 <F1>、 <F2>、 <F3> 的数量。

示例

该示例说明 如何在 DO 循环中使用 VARYING 选项。

```
DATA: BEGIN OF TEXT,
      WORD1(4) VALUE 'This',
      WORD2(4) VALUE 'is',
      WORD3(4) VALUE 'a',
      WORD4(4) VALUE 'loop',
      END OF TEXT.
DATA: STRING1(4), STRING2(4).

DO 4 TIMES VARYING STRING1 FROM TEXT-WORD1 NEXT TEXT-WORD2.
   WRITE STRING1.
   IF STRING1 = 'is'.
      STRING1 = 'was'.
   ENDIF.
ENDDO.

SKIP.

DO 2 TIMES VARYING STRING1 FROM TEXT-WORD1 NEXT TEXT-WORD3
           VARYING STRING2 FROM TEXT-WORD2 NEXT TEXT-WORD4.
    WRITE: STRING1, STRING2.
ENDDO.
```

这产生如下 输出：

```
This is   a    loop

This was  a    loop
```

字段串 TEXT 代表内存中 四个等距字 段序列。每 次执行第一 个 DO 循环时，都 依次将其组 件分配到 STRING1 中。如果 STRING1 包含“is” ，则将其改 变为“was” ，而且自动 将 TEXT-WORD2 改变为“was” 。每次执行 第二个 DO 循环时，将 TEXT 的组件传递 给STRING1和 STRING2。

使用 WHILE 的条件循环

如果只要条 件为真，就 不止一次执 行语句，可 以如下使用 WHILE 语句编程：

语法

```
WHILE <condition> [VARY <f> FROM <F1> NEXT <F2>].
   <statement block>
ENDWHILE.
```

只要 <condition> 是真，或系 统发现 EXIT、 STOP 或 REJECT 语句，系统 将继续执行 由 WHILE 语句引导、 ENDWHILE 结束的语句 块（参见 *终止循环* （页 63） ）。

对于 <condition>， 可以使用 *编程逻辑表 达式* (页 68) 中描述的任 何逻辑表达 式。

系统字段 SY-INDEX 中包含已执 行的循环次 数。

可以任意嵌 套 WHILE 循环，也可 与其他循环 结合使用。

WHILE 语句的 VARY 选项与 DO 循环的 VARYING 选项工作方 式一样（参 见*使用 DO 的无条件循 环* (页 77))。允许 每次执行循 环时为变量 <f> 重新赋值。 <F1>、 <F2>、 <F3>、 ... 必需是内存 中类型相同 和长度相等 的一系列等 距字段。第 一次循环时 ， 将 <F1> 分配给 <f>,第 二次循环时 ,将 <F2> 分配给 <f>,以 此类推。可 以在一个 WHILE 语句中使用 多个 VARY 选项。

警告

使用 WHILE 语句要避免 死循环。请 记住，在一 段时间之后 ， WHILE 语句条件应 变为假，或 者系统能够 找到 EXIT、 STOP 或 REJECT 语句退出循 环。

示例

```
DATA:   LENGTH     TYPE I VALUE 0,
        STRL       TYPE I VALUE 0,
        STRING(30) TYPE C VALUE 'Test String'.

STRL = STRLEN( STRING ).

WHILE STRING NE SPACE.
   WRITE STRING(1).
   LENGTH = SY-INDEX.
   SHIFT STRING.
ENDWHILE.

WRITE: / 'STRLEN:          ', STRL.

WRITE: / 'Length of string:', LENGTH.
```

产生如下输 出：

```
T e s t   S t r i n g

STRLEN:                    11

Length of string:          11
```

此处使用 WHILE 循环确定字 符串的长度 。做法是： 每次执行循 环时，都将 字符串左移 一位，直到 仅包含空格 。选择本例 主要是为了 说明 WHILE 语句。但是 ，确定字符 串长度更简 便和有效的 办法是通过 使用内置的 函数 STRLEN， 这在示例中 也可看出。

终止循环

要终止循环 过程，请使 用下列关键 字之一。

关键字	用途
无条件终止循环过程 (页 41) CONTINUE	
有条件终止循环过	

程 (页 439) CHECK	
完全终止循环 (页 26) EXIT	

注释

在循环中只 能使用 CONTINUE， 但在循环外 还可使用关 键字 CHECK 和 EXIT， 分别完成不 同的功能。 例如，可以 终止子程序 或整个程序 块。关于 CHECK 和 EXIT 语句及其如 何在循环外 使用的详细 信息，参见终止子系统 和 终止处理块 。

以下主题说 明如何在 DO 和WHILE 循环以及下 列循环中使 用 CONTINUE、 CHECK 和 EXIT：

- LOOP - ENDLOOP 循环，用于 处理内表 （ 参见 *循环处理* (页 Error! Not a valid link.) ）。
- SELECT - ENDSELECT 循环，用于 从数据表中 读取数据 （ 参见 从几行选择所有数据）。

无条件终止 循环过程

要立即无条 件终止循环 ，请如下使 用 CONTINUE 语句：

语法

CONTINUE.

在CONTINUE 语句之后， 系统跳过当 前语句块中 所有剩余语 句块，继续 该语句后面 的循环。

示例

```
DO 4 TIMES.
   IF SY-INDEX = 2.
      CONTINUE.
   ENDIF.
   WRITE SY-INDEX.
ENDDO.
```

产生如下输 出：

```
1          3          4
```

此处系统不 处理 WRITE 语句就终止 第二次循环 。

有条件终止 循环过程

要有条件终 止循环过程 ，请如下使 用 CHECK 语句：

语法

CHECK <condition>.

如果条件是 **假**，系统跳 过当前语句 块中所有剩 余语句块， 继续后面的 循环过程。 对于<condition>， 可使用 *编程逻辑表 达式* (页 68) 中描述的任 何逻辑表达 式。

示例

```
DO 4 TIMES.
   CHECK SY-INDEX BETWEEN 2 and 3.
   WRITE SY-INDEX.
ENDDO.
```

产生如下输 出：

```
         2          3
```

此处系统不 处理 WRITE 语句就终止 第一个和第 四个循环， 因为 SY-INDEX 不在 2 和 3 之间。

完全终止循 环

要无条件完 全终止循环 ，请如下使 用 EXIT 语句：

语法

EXIT.

EXIT 语句之后， 系统立即退 出循环，继 续结束语句 (ENDDO、 ENDWHILE、 ENDSELECT) 后 面的处理。 在嵌套循环 中，系统仅 退出当前循 环。

示例

```
DO 4 TIMES.
   IF SY-INDEX = 3.
      EXIT.
   ENDIF.
WRITE SY-INDEX.
ENDDO.
```

产生如下输 出：

```
         1          2
```

此处系统不 处理 WRITE 语句或第四 个循环过程 ，就在第三 个循环过程 中完全终止循环。

概览

内容

要根据一定 条件执行程 序组件，或 要将重复语 句序列组合 在循环中， 可以使用 ABAP/4 提供的用于控制程序流 的标准关键 字。
AB P/4 也包含重要 的附加特征 。
ABAP/4 程序流可以 内部控制和 外部控制。ABAP/4 程序流的**内 部控制**和**外 部控制**之间 的差别在*ABAP/4 中 流控制的概 念* (页 67) 中 解释。
本节主要讨 论**内部流控 制。**为此， 可使用其他 编程语言（ 例如，C、 FORTRAN、 PASCAL 等等）中熟 悉的标准控 制关键字。
下列主题描 述

ABAP/4中 流控制的概 念

与其他高级 编程语言一 样（如，C、 FORTRAN 和 PASCAL 等等），ABAP/4 提供也用于 控制程序流 的标准关键 字。这些关 键字用于

- 分支 (IF、CASE)
- 循环 (DO、WHILE)

然而，ABAP/4 与其他编程 语言不同之 处在于其具 有程序流的 **内部控制**和 **外部控制**。

- **内部控制** 由上述标准 关键字引导 。在程序代 码中对其进 行定义。
- **外部控制** 由事件引导 。事件由 ABAP/4 程序（系统 程序或用户 程序）或者 交互式用户 输入（例如 ，使用鼠标 在屏幕上单 击）生成。 系统不必按 ABAP/4 程序中列出 的顺序对进 行语句处理 。这使得 ABAP/4 成为事件驱 动编程语言 ，类似于 Microsoft 的Visual Basic。

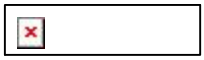
时间事件和 ABAP/4 程序的连接 由事件关键 字提供。ABAP/4 程序中的每 个语句都属 于特别的事 件关键字。 即使在程序 中没有指定 任何事件关 键字，所有 语句都自动 归属于标准 事件关键字（START-OF-SELECTION） ，与程序中 事件语句的 次序完全不 相干。
所有属于某 一特定事件 关键字的语 句形成一个 处理块。处 理块是一个 在事件发生 时执行的模 块。处理块 内的程序流 依赖于**内部 控制。**系统 按顺序处理 语句，或按 照前面提到 的标准关键 字定义的顺 序处理。
ABAP/4 程序常规结 构如下：

假设当用户 选择一行后 ，生成列表 且提供一些 细分设备的 报表程序必 须作出反应（详细信息 ，参见 交互列表）。需要为 该事件处理 的代码必须 插入事件关 键字 AT LINE-SELECTION 之后。

```
AT LINE-SELECTION.
   MOVE 'X' TO FLAG.
   .....
```

无论何时用 户通过单击 鼠标或按 **F2** 键在列表中 选择一个项 目时，AT LINE-SELECTION 和下一个事 件关键字之 间的所有语 句将被处理

关于**外部控 制**的详细信 息，如事件 及其如何与 ABAP/4 程序相互作 用的信息， 参见 C 通过事件控制 ABAP/4 程序流。
该节说明如 何使用**外部 控制**控制处 理块内的程 序流。
要控制 ABAP/4 程序中的内 部流，请遵 循结构化编 程原则并将 程序模块划 分为单个逻 辑相关语句 块（这些组 成控制结构 ）。其中的 每个语句块 都执行主任 务的一部分 。



要使程序易 于阅读，应 该缩排控制 结构中的语 句块。出于 布局需要， 可以使用 ABAP/4 编辑器功能 “编辑 -> 插入语句...” 和“程序 -> 整齐打印程 序”(详细 信息，参见 *ABAP/4 程序布局* (页 2 - 6))。

可以用 IF、CASE、 DO 和 WHILE 之类的关键 字控制处理 块中不同语 句之间的程 序流。这些 语句允许编 制条件和无 条件的分支 和循环。条 件使用逻辑 表达式，可 以是真，也 可以为假。

编程逻辑表 达式

使用条件控 制程序中的 内部流。要 用公式指定 条件，请使 用比较数据 字段的逻辑 表达式，如 下所示：

语法
.... <F1> <operator> <F2> ...
该表达式比 较两个字段 。可能为真 ，也可能为 假。在带关 键字 IF、 CHECK 和 WHILE 的条件语句 中使用逻辑 表达式。
根据操作数 <F1> 和 <F2> 的数据类型 ，可以使用 不同的逻辑 运算符。允 许进行：
除上述比较 之外，还可 以执行测试 以检查数据 字段是否完 全满足一定 条件。可以 使用这些测 试：
另外，可以 将几个逻辑 表达式组合 为一个简单 逻辑表达式 。

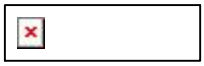
比较所有的 字段类型

要比较所有 的字段类型 ，可以在逻 辑表达式中 使用下列运 算符：

<运算符>	含 义
EQ	等于
=	等于
NE	不 等于
<>	不 等于
><	不 等于
LT	小 于
<	小于
LE	小 于等于
<=	小 于等于
GT	大 于
>	大于
GE	大 于等于
>=	大 于等于

操作数可以 是数据库字 段、内部字 段、文字或 常数。
除基本字段 外，还可以 将结构数据 类型和上表 中的运算符 结合起来作 为操作数。 字段串逐个 组件进行比 较，嵌套的 结构分为基 本的字段。 关于比较内 表的详细信 息，参见 *比较内表* (页 **Error! Not a valid link.**) 。
如果有意义 ，可以对不 同数据类型 的字段进行 比较。如果 字段可转换 ，就可以进 行比较。在 比较之前， 系统将按照 下列层次规 则执行自动 类型转换 (参见 *键入转换* (页 6 - 36)):

1. 如果 操作数之一 是浮点数 (类型 F)，则系 统将其它操 作数转换为 类型 F。
2. 如果 操作数之一 是压缩字段 （类型 P)，则系 统将其它操 作数转换为 类型 P。
3. 如果 操作数之一 是日期字段 （类型 D）或时间 字段（类型 T)，则系 统将其它操 作数转换为 类型 D 或 T。不支持 日期和时间 字段之间的 比较，这会 导致程序中 断。
4. 如果 操作数之一 是字符字段 （类型 C）且其它 操作数是十 六进制字段 （类型 X)，则系 统将类型 X 的操作数转 换为类型 C。
5. 如果 操作数之一 是字符字段 （类型 C)，其它 操作数为数 字字段（类 型 N)，则系 统将这两种 操作数都转 换为类型 P。



```
DATA: F    TYPE F VALUE '100.00',
      P    TYPE P VALUE '50.00' DECIMALS 2,
      I    TYPE I VALUE '30.00'.

WRITE 'The following logical expressions are true:'.

IF F >= P .
   WRITE: / F,'>=',P.
ELSE.
   WRITE: / F,'<',P.
ENDIF.

IF I EQ P .
   WRITE: / I,'EQ',P.
ELSE.
   WRITE: / I,'NE',P.
ENDIF.
```

这生成如下 输出：

```
The following logical expressions are true:
 1.000000000000000E+02 >=             50.00

        30 NE              50.00
```

这里，在 IF 语句中使用 两个逻辑表 达式。如果 逻辑表达式 为真，则屏 幕上显示出来。如果逻 辑表达式为 假，则将相 反表达式显 示在屏幕上 。

比较字符串 和数字串

要比较字符 串（类型 C）和数字 文本（类型 N），可以 在逻辑表达 式中使用下 列运算符。

<运算符>	含 义
CO	仅包 含
CN	不仅 包含
CA	包 含任何
NA	不 包含任何
CS	包 含字符串
NS	不 包含字符串
CP	包含模 式
NP	不包 含模式

因为除类型 N 和 C 外，系统不 能执行任何 其它类型转 换，所以， 在进行包含 这些运算之 一的比较时 ，操作数应 该是类型 N 或 C。
运算符的功 能如下：

CO （仅包含）
如果 <F1> 仅包含 <F2> 中的字符， 则逻辑表达 式
<F1> CO <F2>
为真。该比 较区分大小 写，并包括 尾部空格。 如果比较结 果为真，则 系统字段 SY-FDPOS 包括 <F1> 的长度。如 果为假，则 SY-FDPOS 包含 <F1> 中第一个未 在 <F2> 内出现的字 符的偏移量 。

CN （不仅包含 ）
如果 <F1> 还包含 <F2> 之外的其他 字符，则逻 辑表达式
<F1> CN <F2>
为真。该比 较区分大小 写，并包括 尾部空格。 如果比较结 果为真，则 系统字段 SY-FDPOS 包含 <F1> 中第一个未 同时在 <F2> 中出现的字 符的偏移量 。如果为假 ，SY-FDPOS 包含 <F1> 的长度。

CA（包含任何 ）
如果 <F1> 至少包含 <F2> 的一个字符 ，则逻辑表 达式
<F1> CA <F2>
为真。该比 较区分大小 写。如果比 较结果为真 ，则系统字 段 SY-FDPOS 包含 <F1> 中第一个也 在 <F2> 中出现的字 符的偏移量 。如果为假 ，SY-FDPOS 包含 <F1> 的长度。

NA（不包含任 何）
如果 <F1> 不包含 <F2> 的任何字符 ，则逻辑表 达式
<F1> NA <F2>
为真。该比 较区分大小 写。如果比 较结果为真 ，则系统字 段 SY-FDPOS 包含 <F1>的 长度。如果 为假，则 SY-FDPOS 包含 <F1> 中在 <F2> 内出现的第 一个字符的 偏移量。

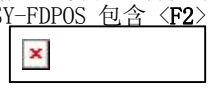
CS（包含字符 串）
如果 <F1> 包含字符串 <F2>， 则逻辑表达 式
<F1> CS <F2>
为真。忽略 尾部空格并 且比较不区 分大小写。 如果比较结 果为真，则 系统字段 SY-FDPOS 包含 <F2> 在 <F1> 中的偏移量 。如果为假 ，SY-FDPOS 包含 <F1> 的长度。

NS（不包含字 符串）
如果 <F1> 不包含字符 串 <F2>， 则逻辑表达 式
<F1> NS <F2>
为真。忽略 尾部空格且 比较不区分 大小写。如 果比较为真 ，系统字段 SY-FDPOS 包含 <F1> 的长度。如 果为假，系 统字段 SY-FDPOS 包含 <F2> 在 <F1> 中的偏移量 。

CP（包含模式 ）
如果 <F1> 包含模式 <F2>， 则逻辑表达 式
<F1> CP <F2>
为真。如果 <F2> 属于类型 C，则可以 在 <F2> 中使用下列 通配符：
- * 用于任何字 符串
- + 用于任何单 个字符

忽略尾部空 格且比较不 区分大小写 。如果比较 结果为真， 系统字段 SY-FDPOS 包含 <F2> 在 <F1> 中的偏移量 。如果为假 ，SY-FDPOS 包含 <F1> 的长度。
如果要对 <F2> 中的特殊字 符进行比较 ，请将换码 字符 # 放到其前面 。可以使用 换码字符 # 指定
- 大小写字 符
- 通配符 "*"(输 入 #*)
- 通配符 "+" (输入 #+)
- 换码符号 本身 (输入 ##)
- 字符串结 尾的空格 (输入 #___)

NP（不包含模 式）
如果 <F1> 不包含模式 <F2>， 则逻辑表达 式
<F1> NP <F2>
为真。在<F2>中 ，可以使用 与 CP 相同的通配 符和换码字 符。
忽略尾部空 格且比较不 区分大小写 。如果比较 结果为真， 则系统字段 SY-FDPOS 包含 <F1>. 的长度，如 果为假，SY-FDPOS 包含 <F2> 在 <F1> 中的偏移量 。



```
DATA: F1(5) TYPE C VALUE <F1>,
      F2(5) TYPE C VALUE <F2>.
IF F1 <operator> F2.
   WRITE: / 'Comparison true, SY-FDPOS=', SY-FDPOS.
ELSE.
   WRITE: / 'Comparison false, SY-FDPOS=', SY-FDPOS.
ENDIF.
```

下表列出该 程序的执行 结果，取决 于所用的运 算符和 F1 / F2 字段。

<F1>	<operator>	<F2>	Result	SY-FDPOS
'BD '	CO	'ABCD '	真	5
'BD '	CO	'ABCDE'	假	2

'ABC12'	CN	'ABCD '	真	3
'ABABC'	CN	'ABCD '	假	5
'ABcde'	CA	'Bd '	真	1
'ABcde'	CA	'bD '	假	5
'ABAB '	NA	'AB '	假	0
'ababa'	NA	'AB '	真	5
'ABcde'	CS	'bC '	真	1
'ABcde'	CS	'ce '	假	5
'ABcde'	NS	'bC '	假	1
'ABcde'	NS	'ce '	真	5
'ABcde'	CP	'*b*'	真	1
'ABcde'	CP	'*#b*'	假	5
'ABcde'	NP	'*b*'	假	1
'ABcde'	NP	'*#b*'	真	5

二进制位结 构的比较

要将逻辑表 达式初始操 作数第一字 节的二进制 位结构与第 二操作数的 进行比较， 请使用下列 操作符。

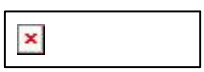
<运算符>	含 义
O	二进制 位是 1
Z	二进制 位是 0
M	混合二 进制位

第二 操作数的长 度应为一个 字节。
没有必要对 第二操作数 使用长度为 1 的十六进制 字段（类型 X），但却 较方便，因 为其长度为 一个字节且 数字值直接 与二进制位 结构相关。
操作符功能 如下：

O（二进制位 是 1）
如果 <hex> 中二进制位 是 1 的位置，在 <f> 中是 1，则逻辑 表达式
<f> O <hex>
为真。

Z（二进制位 是0）
如果 <hex> 中二进制位 是 1 的位置，在 <f> 中是 0，则逻辑 表达式
<f> Z <hex>
为真。

M（混合二进 制位）
如果从 <hex> 中二进制位 是 1 的位置起， <f> 中至少一个 是 1，一个是 0，则逻辑 表达式
<f> M <hex>
为真。



```
DATA: C VALUE 'C',
      HEXDEC TYPE X,
      I TYPE I.

HEXDEC = 0.
```

```
DO 256 TIMES.
   I = HEXDEC.
   IF C O HEXDEC.
      WRITE: / HEXDEC, I.
   ENDIF.
   HEXDEC = HEXDEC + 1.
ENDDO.
```

产生下列输 出：

```
00          0
01          1
02          2
03          3
40         64
41         65
42         66
43         67
```

这里，使用 运算符 O 将字符‘C’ 的二进制位 结构与所有 ‘0’与‘ FF’（10 进制为 255）之 间的 16 进制数进行 比较。HEXDEC 的10 进制值 I ，在HEXDEC 分配到I 期间，通过 使用自动类 型转换确定 。如果比较 结果为真， 则在屏幕上 显式 16 进制数及其 10 进制值。下 表列出这些 数的二进制 位结构。

16 进制	10 进制	二进制 位结构
00	0	00000000
01	1	00000001
02	2	00000010
03	3	00000011
40	64	01000000
41	65	01000001
42	66	01000010
43	67	01000011

字 符‘C’的 二进制位结 构由其 ASCII 码 67 为当前硬件 平台进行定 义。与用户 看到的一样 ，列表输出 中出现的数 字，填写 1 的二进制位 位置与 67 的二进制位 结构一样。

检查字段是 否属于某一 范围

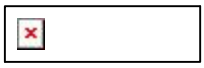
要检查值是 否属于特定 范围，请使 用下列带有 BETWEEN 参数的逻辑 表达式：

语法

```
.... <F1> BETWEEN <F2> AND <F3> .....
```

如果 <F1> 在 <F2> 和 <F3> 之间的范围 内发生，则 表达式为真 。它是下列 逻辑表达式 的短格式：

```
IF <F1> GE <F2> AND <F1> LE <F3>.
```



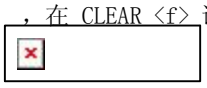
```
DATA: NUMBER TYPE I,
      FLAG.

....

NUMBER = ...

....

IF NUMBER BETWEEN 3 AND 7.
   FLAG = 'X'.
ELSE.
   FLAG = ' '.
ENDIF.
```

这里，如果 NUMBER 的内容在 3 和 7 之间出现， 字段 FLAG 设置为“X” 。

检查初始值

要检查字段 是否设置为 初始值，请 如下使用带 有 IS INITIAL 参数的逻辑 表达式。

语法
```
.... <f> IS INITIAL .....
```
如果〈f〉包含本身类 型的初始值 ，则表达式 为真。通常 情况下，任 何字段，基 本的或结构 化的（字符 串和内表） ，在 CLEAR 〈f〉语句执行后 ，都包含其 初始值（参 见 *重置缺省值* （页 6－10） ）。



```
DATA FLAG VALUE 'X'.

IF FLAG IS INITIAL.
   WRITE / 'Flag is initial'.
ELSE.
   WRITE / 'Flag is not initial'.
ENDIF.

CLEAR FLAG.

IF FLAG IS INITIAL.
   WRITE / 'Flag is initial'.
ELSE.
   WRITE / 'Flag is not initial'.
ENDIF.
```

这产生如下 输出：

```
Flag is not initial

Flag is initial.
```

这里，DATA 语句之后， 字符串 FLAG 不包含初始 值，因为用 VALUE 参数设置为 ‘X’。执 行 CLEAR 语句之后， 将其重置为 初始值。

检查选择条 件

要检查字段 内容是否与 选择表中的 选择条件匹 配，可以如 下使用带有 IN 参数的逻辑 表达式：

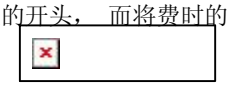
语法
```
... <f> IN <seltab> ....
```
如果字段〈f〉内容的符合 选择表〈seltab〉中的条件， 则表达式为 真。
关于选择条 件的详细信 息，参见 使用选择标准。
关于逻辑表 达式中检查 选择条件的 详细信息， 包括示例， 参见 使用逻辑表达式中的选择表。

组合几个逻 辑表达式

通过使用逻 辑连接运算 符 AND、OR 和 NOT，可 以将几个逻 辑表达式组 合为单个表 达式：

- 要将几个 逻辑表达式 组合为单个 表达式，且 该表达式仅 当所有组件 表达式为真 时才为真，则表达式之 间要用 AND 连接。
- 要将几个 逻辑表达式 组合为单个 表达式，且 只要其中一 个组件表达 式为真时， 该表达式即为真，则表 达式之间要 用 OR 连接。
- 要转化逻 辑表达式的 结果，请在 其前面指定 NOT。

NOT 优先于 AND，AND 优先于 OR。但是 ，可以用任 何括弧组合 指定处理顺 序。由于 ABAP/4 将括弧解释 为单字，前 面或后面必 须至少有一 个空格。
ABAP/4 从左到右处 理逻辑表达 式。如果确 定组件表达 式之一是真 或假，就不 再执行该组 件中其余的 比较或检查 。这意味着 采取这种方 法组织逻辑 表达式可以 提高性能， 就是将经常 为假的比较 放置在 AND 链的开头， 而将费时的 比较，如字 符串查找放 等到最后。



```
DATA: F    TYPE F VALUE '100.00',
      N(3) TYPE N VALUE '123',
      C(3) TYPE C VALUE '456'.

WRITE 'The following logical expression is true:'.

IF ( C LT N ) AND  ( N GT F ).
   WRITE: / '(',C,'lt',N,') AND (',N,'gt',F,')'.
ELSE.
   WRITE: / '(',C,'ge',N,') OR (',N,'le',F,')'.
ENDIF.
```

这产生如下 输出：

下列逻辑表达式为真：

(456 ge 123) OR (123 le 1.000000000000000E+02)

在本例中， 在 IF 语句中使用 逻辑表达式 。如果逻辑 表达式为真 ，则将其在 屏幕上显示 出来。如果 为假，则屏 幕上出现相 反的表达式 。

编程分支和 循环

可以在程序 中定义条件 和无条件分 支和循环。 为此，ABAP/4 提供了几个 在下列主题 中描述的语 句。
分支
循环

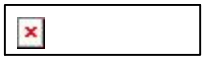
使用 IF 的条件分支

IF 语句允许依 据条件将程 序流转到特 定的语句块 中。该语句 块包括 IF 语句及其后 面的 ELSEIF、ELSE 或 ENDIF 之间的所有 命令。

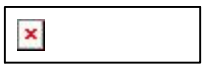
语法

```
IF <condition1>.
   <statement block>
ELSEIF <condition2>.
   <statement block>
ELSEIF <condition3>.
   <statement block>
.....
ELSE.
   <statement block>
ENDIF.
```

如果第一个 条件是真， 系统将执行 所有语句直 到第一个语 句块结束， 然后继续处 理 ENDIF 语句之后的 程序。要采 用选择性条 件，可以使 用 ELSEIF 语句。如果 第一个条件 是假，系统 使用与 IF 语句一样的 方式处理后 面的 ELSEIF 语句。如果 IF 或 ELSEIF 条件都为假 ，则执行 ELSE 开始的语句 块。最后的 语句块必须 用 ENDIF 结束。
要用公式表 达 IF 和 ELSEIF 语句中的条 件，可以使 用任何 *编程逻辑表 达式* (页 84) 描述的逻辑 表达式。



ABAP/4 允许无限地 嵌套 IF - ENDIF 语句块，但 是必须在相 同的处理块 中终止。换句话说，就 是 IF - ENDIF 块不能包含 事件关键字 。



```
DATA: TEXT1(30) VALUE 'This is the first text',
      TEXT2(30) VALUE 'This is the second text',
      TEXT3(30) VALUE 'This is the third text',
      STRING(5) VALUE 'eco'.

IF TEXT1 CS STRING.
   WRITE / 'Condition 1 is fulfilled'.
ELSEIF TEXT2 CS STRING.
   WRITE / 'Condition 2 is fulfilled'.
ELSEIF TEXT3 CS STRING.
   WRITE / 'Condition 3 is fulfilled'.
ELSE.
   WRITE / 'No condition is fulfilled'.
ENDIF.
```

产生如下输 出：

```
Condition 2 is fulfilled.
```

这里，第二 个逻辑表达 式 TEXT2 CS STRING 是真，因为 字符串“eco” 存在于 TEXT2 中。

使用 CASE 的条件分支

要根据特殊 数据字段的 内容执行不 同的语句块 ，可以如下 使用 CASE 语句：

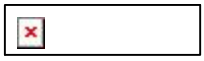
语法

```
CASE <f>.
   WHEN <F1>.
        <statement block>
   WHEN <F2>.
        <statement block>
   WHEN <F3>.
        <statement block>
   WHEN ...
   ......
   WHEN OTHERS.
        <statement block>
ENDCASE.
```

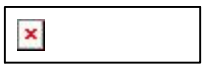
系统执行 WHEN 语句之后的 语句块，如 果 <f> 内容等于 <f_i> 的内容，且 继续处理 ENDCASE 语句后面的 语句。如果 <f> 的内容不等 于 <f_i> 的内容，则 执行选项 WHEN OTHERS 后面的语句 块。最后的 语句块必须 用 ENDCASE 结束。

使用 CASE 的条件分支 是带 IF 的类似处理 的短格式：

```
IF <f> = <F1>.
   <statement block>
ELSEIF <f> = <F2>.
   <statement block>
ELSEIF <f> = <F3>.
   <statement block>
ELSEIF <f> = ...
   ...
ELSE.
   <statement block>
ENDIF.
```



在 ABAP/4 中，可以嵌 套 CASE - ENDCASE 块，且可以 与 IF - ENDIF 结合使用， 但必须在同 一处理块中 终止。



```
DATA: TEXT1   VALUE 'X',
      TEXT2   VALUE 'Y',
      TEXT3   VALUE 'Z',
      STRING  VALUE 'A'.

CASE STRING.
  WHEN TEXT1.
     WRITE: / 'String is', TEXT1.
  WHEN TEXT2.
     WRITE: / 'String is', TEXT2.
  WHEN TEXT3.
     WRITE: / 'String is', TEXT3.
  WHEN OTHERS.
     WRITE: / 'String is not', TEXT1, TEXT2, TEXT3.
ENDCASE.
```

产生如下输 出：

```
String is not X Y Z
```

这里，执行 WHEN OTHERS 后面的语句 块，因为 STRING 的内容“A” 不等于“X” 、“Y”或 “Z”，。

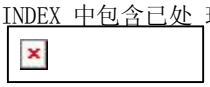
使用 DO 的无条件循 环

如果想要多 次执行语句 块，则可以 如下使用 DO 语句编程循 环：

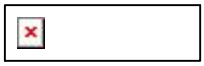
语法

```
DO [<n> TIMES] [VARYING <f> FROM <F1> NEXT <F2>].
   <statement block>
ENDDO.
```

在发现 EXIT、 STOP 或 REJECT 语句之前， 系统继续执 行由 DO 引导、ENDDO 结束的语句 块（参见 *终止循环* (页 80) ）。
可以使用 TIMES 选项限制循 环次数。<n> 可以是文字 或变量。如 果 <n> 是 0 或负数，系 统不执行该循环。
系统字段 SY-INDEX 中包含已处 理过的循环 次数。



使用 DO 语句时要避 免死循环。 如果不使用 TIMES 选项，则在 语句块中至 少应包含一 个 EXIT、 STOP 或 REJECT 语句，以便 系统能够退 出循环。



本例显示 DO 循环的基本 格式。

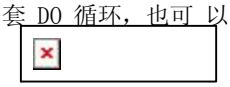
```
DO.

WRITE SY-INDEX.

   IF SY-INDEX = 3.
      EXIT.
   ENDIF.

ENDDO.
```

产生如下输 出：

```
        1          2          3
```

这里，处理 3 次循环，然 后在 EXIT 语句后退出 循环。

可以任意嵌 套 DO 循环，也可 以与其他循 环组合使用 。



本例显示 2 个嵌套循环 ，都使用 TIMES 选项。

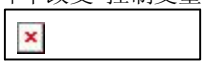
```
DO 2 TIMES.
   WRITE SY-INDEX.
   SKIP.
   DO 3 TIMES.
      WRITE SY-INDEX.
   ENDDO.
   SKIP.
ENDDO.
```

产生如下输 出：

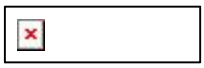
```
        1
        1          2          3
        2
        1          2          3
```

外部循环执 行 2 次。每次执 行外部循环 时，内部循 环都执行 3 次。注意系 统字段 SY-INDEX 记录每个循 环各自的循 环次数。

可以使用 VARYING 选项在每次 循环中给变 量〈f〉重新赋值。〈F1〉、〈F2〉、〈F3〉、 ... 必需是内存 中类型相同 和长度相等 的一系列等 距字段。第 一次循环中 ，将〈F1〉分配给〈f〉，第 二次循环中 ，将〈F2〉分配给〈f〉，以 此类推。可 以在一个 DO 语句中使用 多个 VARYING 选项。
如果在 DO 循环中改变 控制变量〈f〉，则 系统将自动 改变相应的 字段〈f_i〉。



应保证循环 次数不超过 涉及到的变 量〈F1〉、〈F2〉、〈F3〉的数量。



该示例说明 如何在 DO 循环中使用 VARYING 选项。

```
DATA: BEGIN OF TEXT,
      WORD1(4) VALUE 'This',
      WORD2(4) VALUE 'is',
      WORD3(4) VALUE 'a',
      WORD4(4) VALUE 'loop',
      END OF TEXT.
DATA: STRING1(4), STRING2(4).

DO 4 TIMES VARYING STRING1 FROM TEXT-WORD1 NEXT TEXT-WORD2.
   WRITE STRING1.
   IF STRING1 = 'is'.
      STRING1 = 'was'.
   ENDIF.
ENDDO.

SKIP.
```

```
DO 2 TIMES VARYING STRING1 FROM TEXT-WORD1 NEXT TEXT-WORD3
          VARYING STRING2 FROM TEXT-WORD2 NEXT TEXT-WORD4.
    WRITE: STRING1, STRING2.
ENDDO.
```

这产生如下 输出：

```
This is   a    loop
This was  a    loop
```

字段串 TEXT 代表内存中 四个等距字 段序列。每 次执行第一 个 DO 循环时，都 依次将其组 件分配到 STRING1 中。如果 STRING1 包含“is” ，则将其改 变为“was” ，而且自动 将 TEXT-WORD2 改变为“was” 。每次执行 第二个 DO 循环时，将 TEXT 的组件传递 给STRING1和 STRING2。

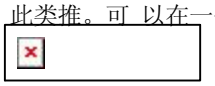
使用 WHILE 的条件循环

如果只要条 件为真，就 不止一次执 行语句，可 以如下使用 WHILE 语句编程：

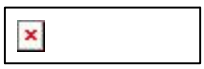
语法

```
WHILE <condition> [VARY <f> FROM <F1> NEXT <F2>].
   <statement block>
ENDWHILE.
```

只要 <condition> 是真，或系 统发现 EXIT、 STOP 或 REJECT 语句，系统 将继续执行 由 WHILE 语句引导、 ENDWHILE 结束的语句 块（参见 *终止循环* (页 80) ）。
对于 <condition>， 可以使用 *编程逻辑表 达式* (页 84) 中描述的任 何逻辑表达 式。
系统字段 SY-INDEX 中包含已执 行的循环次 数。
可以任意嵌 套 WHILE 循环，也可 与其他循环 结合使用。
WHILE 语句的 VARY 选项与 DO 循环的 VARYING 选项工作方 式一样（参 见*使用 DO 的无条件循 环* (页 92) ）。允许 每次执行循 环时为变量 <f> 重新赋值。 <F1>、 <F2>、 <F3>、 ... 必需是内存 中类型相同 和长度相等 的一系列等 距字段。第 一次循环时 ，将 <F1> 分配给 <f>，第 二次循环时 ，将 <F2> 分配给 <f>，以 此类推。可 以在一个 WHILE 语句中使用 多个 VARY 选项。



使用 WHILE 语句要避免 死循环。请 记住，在一 段时间之后 ，WHILE 语句条件应 变为假，或 者系统能够 找到 EXIT、 STOP 或 REJECT 语句退出循 环。



```
DATA:   LENGTH     TYPE I VALUE 0,
        STRL       TYPE I VALUE 0,
        STRING(30) TYPE C VALUE 'Test String'.

STRL = STRLEN( STRING ).

WHILE STRING NE SPACE.
   WRITE STRING(1).
   LENGTH = SY-INDEX.
   SHIFT STRING.
ENDWHILE.

WRITE: / 'STRLEN:          ', STRL.

WRITE: / 'Length of string:', LENGTH.
```

产生如下输 出：

```
T e s t   S t r i n g

STRLEN:                   11

Length of string:         11
```

此处使用 WHILE 循环确定字 符串的长度 。做法是： 每次执行循 环时，都将 字符串左移 一位，直到 仅包含空格 。选择本例 主要是为了 说明 WHILE 语句。但是 ，确定字符 串长度更简 便和有效的 办法是通过 使用内置的 函数 STRLEN， 这在示例中 也可看出。

终止循环

要终止循环 过程，请使 用下列关键 字之一。

关键 字	用途
无条 件终 止 循 环过 程 （页 81) CONTINUE	
有条 件终 止 循 环过 程 (页 81) CHECK	
完全 终止 循 环 (页 82) EXIT	

在循环中只 能使用 CONTINUE， 但在循环外 还可使用关 键字 CHECK 和 EXIT， 分别完成不 同的功能。 例如，可以 终止子程序 或整个程序 块。关于 CHECK 和 EXIT 语句及其如 何在循环外 使用的详细 信息，参见终止子系统 和 终止处理块 。

以下主题说 明如何在 DO 和 WHILE 循环以及下 列循环中使 用 CONTINUE、 CHECK 和 EXIT：

- LOOP - ENDLOOP 循环，用于 处理内表 （ 参见 *循环处理* (页 **Error! Not a valid link.**) ）。
- SELECT - ENDSELECT 循环，用于 从数据表中 读取数据 （ 参见 从几行选择所有数据）。

无条件终止 循环过程

要立即无条 件终止循环 ，请如下使 用 CONTINUE 语句：

语法

CONTINUE.

在 CONTINUE 语句之后， 系统跳过当 前语句块中 所有剩余语 句块，继续 该语句后面 的循环。

```
DO 4 TIMES.
   IF SY-INDEX = 2.
      CONTINUE.
   ENDIF.
   WRITE SY-INDEX.
ENDDO.
```

产生如下输 出：

```
1          3          4
```

此处系统不 处理 WRITE 语句就终止 第二次循环 。

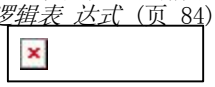
有条件终止 循环过程

要有条件终 止循环过程 ，请如下使 用 CHECK 语句：

语法

CHECK <condition>.

如果条件是 **假，**系统跳 过当前语句 块中所有剩 余语句块， 继续后面的 循环过程。 对于<condition>，可使用 *编程逻辑表 达式* (页 84) 中描述的任 何逻辑表达 式。



```
DO 4 TIMES.
   CHECK SY-INDEX BETWEEN 2 and 3.
   WRITE SY-INDEX.
ENDDO.
```

产生如下输 出：

```
         2          3
```

此处系统不 处理 WRITE 语句就终止 第一个和第 四个循环， 因为 SY-INDEX 不在 2 和 3 之间。

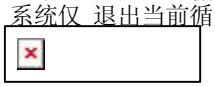
完全终止循 环

要无条件完 全终止循环 ，请如下使 用 EXIT 语句：

语法

EXIT.

EXIT 语句之后， 系统立即退 出循环，继 续结束语句 (ENDDO、 ENDWHILE、 ENDSELECT)后 面的处理。 在嵌套循环 中，系统仅 退出当前循 环。



```
DO 4 TIMES.
   IF SY-INDEX = 3.
      EXIT.
   ENDIF.
WRITE SY-INDEX.
ENDDO.
```

产生如下输 出：

```
         1          2
```

此处系统不 处理 WRITE 语句或第四 个循环过程 ，就在第三 个循环过程 中完全终止循环。

第八章　　创建和处理 内表

概览

内容

本节讨论内 表。除字段 串外，内表 还构成 ABAP/4 提供的另一 种结构化数 据类型。
本节主题描 述

在处理内表 中大量的数 据时,花费 的计算机时 间对性能来 说非常关键 。例如,要 获得最佳性 能，在“ABAP/4 开发工作台 ”初始屏幕 （或事务 SE30） 上选择“测 试 -> 运行时间分 析 ”，在“内 表”下选定 “提示 & 技巧.”， 就会出现有 关如何改进 性能的不同 任务示例。

什么是内表

下列主题提 供内表简介 ：

内表的目的

在 ABAP/4 中，主要使 用表格。表 格是 R/3 系统中的关 键数据结构 。长期使用 的数据存储 在关系数据 库表格中。 关于如何读 取和处理数 据库表格的 详细信息， 参见 读取并处理数据库表。
除了数据库 表格，还可 以创建仅在 程序运行时 间内存在的 内表。ABAP/4 提供了针对 内表的不同 操作。例如 ，可以搜索 、附加、插 入或删除行 。
内表中的行 数并不固定 。根据需求 ，系统可实 时增加内表 的大小。例 如，如果想 将某个数据 库表格读入 内表，不必 事先知道数 据库表格的 大小。该特 征项使得内 表使用起来 十分方便， 同时还支持 动态编程。
可以使用内 表在数据库 表格的子集 上执行表格 计算。例如 ，可以将数 据库表格的 某个部分读 入内表（参 见 将数据读入内表）。然后可 以从内表中 计算总和或 生成次序表 。
内表的另一 种用处是根 据程序需要 重新组织数 据库表格的 内容。例如 ，可以从一 个或多个大 客户表格特 定的数据中 将与创建电 话清单有关 的数据读入 内表中。然 后可在程序 运行期间直 接访问该清 单，而不用 每次调用时 都执行耗时 的数据库查 询。
除了在使用 来自数据库 表格的数据 时使用内表 外，内表还 是 ABAP/4 中用于在程 序中实现许 多复杂数据 结构的重要 特征项（参 见 *结构化数据类型* （页 3 - 6） ）。

内表的结构

在 ABAP/4 中，可以区 别内表数据 类型（定义 内表的结构 ）和内表数 据对象（实 际的内表而 且可以用数 据进行填充 ）。内表数 据类型是数 据结构（可 用于将数据 对象说明为 内表）的抽 象定义。关 于数据类型 和数据对象 之间区别的 详细信息， 参见 *声明数据* （页 3 - 1） 。

数据类型
内表是 ABAP/4 中两种结构 化数据类型 中的一种。 其它结构化 数据类型是 字段串（参 见 *结构化数据类型*（页 3－6））。内表包 括任意数据 类型相同的 行。行的数 据类型可以 是基本的或 结构化的。 该定义打开 了多种内表 结构，范围 从包含一个 字段的行到 包含字段串 将内表作为 组件的行。
可以用带 OCCURS 参数的 TYPES 语句将数据 类型定义为 内表（参见 *创建内表数 据类型*（页 84））。定义数 据类型时不 占用内存。

数据对象
数据对象包 含定义为内 表的数据类 型，是实际 使用的内表 。数据对象 占用内存， 可以对其行 进行填充或 读取。
可以使用带 OCCURS 参数的 DATA 语句，或使 用 TYPE 或 LIKE 参数引用另 一个内表将 数据对象创 建为内表（ 参见 *创建内表数 据对象*（页 63））。

标识表格行

为了访问表 格的某一行 ，必须指定 可用于标识 该行的字段 或组合字段 。在关系数 据模型（用 于在 R/3 系统中存储 长期使用的 数据）中， 用于该目的 的最小组合 称为**关键字** 。定义关键 字的字段称 为**关键字段** 。
在关系数据 模型中，每 个表格至少 有一个关键 字（参见文 档 *ABAP/4 词典*（页 **Error! Not a valid link.**））。特殊唯 一关键字段 的该概念不 用于内表。
但是 ABAP/4 提供了下列 特征项以便 用户访问内 表行：

内表索引
索引是表格 行的序列号 ，不是表格 字段，但由 系统自动创 建和管理。
可以用 DELETE、 INSERT、 MODIFY、 LOOP 和 READ 语句来使用 索引。在这 些语句中， 可以将索引 指定为文字 或变量。
处理完内表 的特定行后 ，系统字段 SY-TABIX 一般包含该 行的索引。

内表关键字
有两种类型 的内表关键 字。

自定义关键 字
使用 READ 语句从内表 中读取行时 ，可以指定 自定义关键 字（参见 *用自定义关 键字读取单 行*（页 42））。

缺省关键字
根据定义， 内表的关键 字段是非数 字（类型 F、I 和 P）和非内 表的字段。 这些关键字 段形成内表 的标准关键 字。
要获得带嵌 套结构（包 含字段串作 为组件的表 格行）的内 表标准关键 字，系统将 子结构分为 基本字段层 次。

根据填充内 表的方式不 同，内表可 以包含带相 同标准关键 字的多行。

用 COLLECT、 READ、 SORT 和 SUM 语句使用内 表的关键字 段。如果标 准关键字段 是内表行的 第一个组件 ，这有助于 提高这些语 句的效率。 创建内表时 请记住这一 点。

访问内表

内表是行 进行访问的 。必须使用 某个工作区 域作为与表 格互相传输 数据的接口 。

工作区域对 内表的行必 须是可转换 的（关于可 转换性的详 细信息，参 见 *类型转换*（页 6－36））。

从内表中读 取数据时， 已定址的表 格行内容覆 盖工作区域 的内容。然 后可以在程 序中引用工 作区域的内 容。将数据 写入内表时 ，必须首先 在工作区域 （从中系统 可以将数据 传输给内表 ）中输入数 据。

为了避免不 一致，最好 是工作区域 与内表行有 相同的数据 类型。创建 与内表兼容 的工作区域 的一种安全 步骤是在说 明内表和工 作区域时使 用相同的数 据类型。
在该环境中 ，可以区分 ABAP/4 中两种类型 的内表的差 别：

- 带表头行 的内表
- 不带表头 行的内表

如果创建带 表头行的内 表（参见 *创建内表*（页 35），系统自 动创建与内 表行数据类 型相同的工 作区域。该 工作区域称 为表头行或 表格工作区 域，对内表 的作用与由 TABLES 语句创建的 数据库表格 工作区域相 同（参见 *TABLES 语句*（页 3－20））。表格工 作区域和内 表本身同名 。

在所有用于 访问内表的 ABAP/4 语句中，可 以指定要使 用的工作区 域（参见 *使 用内表* （页 96） ）。对于带 表头行的内 表，可以忽 略这一指定 。这样，系 统隐式使用 表格工作区 域：

不带表头行 的内表没有 可以隐式使 用的表格工 作区域。要 访问没有表 头行的内表 ，必须在相 应的 ABAP/4 语句中显式 指定工作区 域。
决定创建的 内表是否带 表头行时， 必须考虑是 喜欢隐式还 是显式用于 内表访问的 工作区域。

请记住，对 于带表头行 的内表，内 表本身和表 格工作区域 同名。如果 在语句中使 用该名称， 系统将其解 释为表格工 作区域的名 称，而不是 表格本身（ 对于将数据 读入内存或 从内存中读 取数据的语 句，这一点 例外，参见 ABAP/4 内存中的数据簇 ）。但在某 些语句中， 可以在名称 之后输入方 括号，定址 内表而不是 表格工作区 域，如下所 示

：<name>[]。

创建内表

创建内表时 ，可以决定 是想先用 TYPES 语句创建内 表数据类型 ，然后再创 建具有该类 型的数据对 象，还是想 直接使用 DATA 语句创建内 表数据对象 。既可以创 建带表头行 的内表数据 对象，又可 创建不带表 头行的内表 数据对象。
下列主题描 述

创建内表数 据类型

要创建内表 数据类型， 请使用 TYPES 语句，用法 如下（参见 *TYPES 语句* （页 3－22） ）：

语法
```
TYPES <t> <type> OCCURS <n>.
```
该语句通过 使用 TYPES 语句的 OCCURS 选项创建一 个内表数据 类型 <t>。内 表中行的数 据类型在 <type> 中指定。要 指定行的数 据类型，可 以使用 TYPE 或 LIKE 参数（参见 *DATA 语句的基本格式* （页 3－14） ）。
通过使用 LIKE 参数引用 ABAP/4 词典中定义 的对象，可 以创建内表 ，其行结构 与存储在词 典中的对象 相同且反映 数据库表格 的结构。这 在读取和处 理数据库表 格时非常重 要（参见 读取并处理数据库表）。
<n> 指定行的初 始号。将第 一行写入用 类型 <t> 创建的内表 数据对象之 后，就为指 定行保留了 内存。如果 添加到内表 中的行比 <n> 指定的要多 ，则自动扩 展保留的内 存。如果内 存中没有足 够空间可用 于内表，则 将其写入缓 冲区或磁盘 （分页区域 ）。

```
TYPES VECTOR TYPE I OCCURS 10.
```
本示例创建 内表数据类 型 VECTOR， 其行包含基 本类型 I 字段。

```
TYPES: BEGIN OF LINE,
         COLUMN1 TYPE I,
         COLUMN2 TYPE I,
         COLUMN3 TYPE I,
       END OF LINE.
TYPES ITAB TYPE LINE OCCURS 10.
```
本示例创建 内表数据类 型 ITAB， 其行与字段 串 LINE 结构相同。

```
TYPES VECTOR TYPE I OCCURS 10.
TYPES: BEGIN OF LINE,
         COLUMN1 TYPE I,
         COLUMN2 TYPE I,
         COLUMN3 TYPE I,
       END OF LINE.
```

```
TYPES ITAB TYPE LINE OCCURS 10.

TYPES: BEGIN OF DEEPLINE,
          TABLE1 TYPE VECTOR,
          TABLE2 TYPE ITAB,
       END OF DEEPLINE.

TYPES DEEPTABLE TYPE DEEPLINE OCCURS 10.
```

本示例创建 与上例相同 的内表数据 类型（VECTOR 和 ITAB） 。然后创建 数据类型 DEEPLINE 作为字段串 ，包含这些 内表作为组 件。通过该 字段串，数 据类型 DEEPTABLE 被创建为内 表。因此该 内表的元素 本身就是内 表。

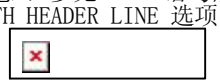
创建内表数 据对象

要创建内表 数据对象， 可以有几种 方式使用 DATA 语句（参见 *DATA 语句* （页 3－14））。可将 DATA 语句用于

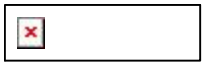
对于前两种 情况，创建 表头行可选 ，但对于用 新的行结构 创建的表格 ，则必须要 有表头行。

通过引用另 一个表格来 创建内表
要通过引用 现有内表数 据类型或数 据对象来创 建内表数据 对象，可使 用 DATA 语句，用法 如下：

语法
DATA <f> <type> [WITH HEADER LINE].
通过使用 TYPE 或 LIKE， 可以使用 <type> 选项来引用 表格数据类 型或表格数 据对象（关 键这些选项 的详细信息 ，参见 *DATA 语句的基本格式* （页 3－14））。将数据 对象 <f> 说明为结构 相同的内表 。如果使用 WITH HEADER LINE 选项，则创 建的内表带 工作区域 <f> （参见 *访问内表* （页 98））。



如果想创建 带表头行的 内表，行类 型不能直接 是内表。但 可以是用内 表作为组件 的结构。



```
TYPES: BEGIN OF LINE,
          COLUMN1 TYPE I,
          COLUMN2 TYPE I,
          COLUMN3 TYPE I,
       END OF LINE.

TYPES ITAB TYPE LINE OCCURS 10.

DATA TAB1 TYPE ITAB.

DATA TAB2 LIKE TAB1 WITH HEADER LINE.
```

同 *创建内表数 据类型* （页 99）中所示，该 示例创建数 据类型 ITAB 作为内表。 通过使用 DATA 语句的 TYPE 参数引用 ITAB， 使数据对象 TAB1 与 ITAB 结构相同。 通过使用 DATA 语句的 LIKE 参数引用 TAB1， 使数据对象 TAB2 结构相同。 创建的 TAB2 带表头行。因此，可以 在程序中使 用 TAB2-COLUMN1、TAB2-COLUMN2 和 TAB2-COLUMN3 等定位表格 工作区域 TAB2。

通过引用结 构来创建内 表
要通过引用 现有行结构 创建内表数 据对象，请 使用 DATA 语句，用法 如下：

语法
DATA <f> <type> OCCURS <n> [WITH HEADER LINE].
该语句通过 使用 DATA 语句的 OCCURS 选项创建内 表 <f>。内 表中行的数 据类型在 <type> 中指定。要 指定数据类 型，可以使 用 TYPE 或 LIKE 参数（关于 这些参数的 详细信息， 参见 *DATA 语句的基本格式* （页 3－14））。
通过使用 LIKE 参数引用 ABAP/4 词典中定义 的对象，可 以创建内表 ，其行结构 与存储在词 典中的对象 相同，并反 映数据库表 格的结构。 这在读取和 处理数据库 表格时非常 重要（参见 读取并处理数据库表）。

<n> 指定行的初 始号。将第 一行写入用 类型 <t> 创建的内表 数据对象之 后，就为指 定行保留内 存。如果添 加到内表中 的行比 <n> 指定的要多 ，则自动扩 展保留的内 存。如果内 存中没有足 够空间可用 于内表，则 将其写入缓 冲区或磁盘 （分页区域 ）。
上述特征项 与用 TYPES 语句创建内 表数据类型 的特征项相 同（参见 *创建内表数 据类型* （页 99） ）。作为附加特 征项，可以 在 DATA 语句中使用 WITH HEADER LINE 选项。这样 创建的表格 工作区域 <f> 与内表 <f> 的行结构相 同（关于表 头行的详细 信息，参见 *访问内表* （页 98） ）。

```
DATA FLIGHT_TAB LIKE SFLIGHT OCCURS 10.
```

本示例创建 数据对象 FLIGHT_TAB， 其结构与数 据库表格 SFLIGHT 相同。

本示例介绍 如何采用两 种不同的步 骤创建同一 内表。

```
TYPES VECTOR_TYPE TYPE I OCCURS 10.
DATA VECTOR TYPE VECTOR_TYPE WITH HEADER LINE.
```

在此创建一 个内表数据 类型 VECTOR_TYPE， 其行包含首 先创建的基 本类型 I 字段。然后 ，通过引用 VECTOR_TYPE 创建数据对 象 VECTOR。 通过使用 WITH HEADER LINE 选项还创建 表格工作区 域 VECTOR。 在这种情况 下，表格工 作区域包含 一种类型 I 字段，可以 通过名称 VECTOR 定位。

```
DATA VECTOR TYPE I OCCURS 10 WITH HEADER LINE.
```

在这种情况 下，通过直 接在 DATA 语句中使用 OCCURS 选项创建完 全一样的数 据类型 VECTOR。

创建带新结 构的内表
要创建既不 引用现有对 象，也不引 用现有行结 构的内表数 据对象，使 用 DATA 语句，用法 如下：

语法

```
DATA: BEGIN OF <f> OCCURS <n>,
              <component declaration>,
              ..............
      END OF <f>.
```

这定义内表 <f> 并在 <component declaration> 中说明其行 组件。除 CCURS 参数外，语 法与定义字 段串相同 （ 参见 *字段串的 DATA 语句* （页 3－18） ）。
该语句**通常** 表格工作区 域 <f>（参 见 *访问内表* （页 98） ）。因此， 其作用与先 创建字段串 <f>，然 后再创建与 该字段串行 结构相同的 内表 <f> 相同。
<n> 指定行的初 始号。将第 一行写入用 类型 <t> 创建的内表 数据对象之 后，就为指 定行保留内 存。如果添 加到内表中 的行比 <n> 指定的要多 ，则自动扩 展保留的内 存。如果内 存中没有足 够空间可用 于内表，则 将其写入缓 冲区或磁盘 （分页区域 ）。

```
DATA: BEGIN OF ITAB OCCURS 10,
         COLUMN1 TYPE I,
         COLUMN2 TYPE I,
         COLUMN3 TYPE I,
       END OF ITAB.
```

本示例创建 内表及其相 应的表格工 作区域 ITAB。

使 用内表

下列主题描 述如何使用 内表。本节 解释：

填充内表

要填充内表 ，既可逐行 添加数据， 也可复制另 一个表格的 内容。
要逐行填充 内表，可以 使用 APPEND、 COLLECT 或 INSERT 语句。

- 要将内表 仅用于存储 数据，出于 性能方面的 考虑，建议 使用 APPEND。 用 APPEND 也可以创建 序列清单。
- 要计算数 字字段之和 或要确保内 表中没有出 现重复条目 ，请使用 COLLECT 语句，它根 据标准关键 字处理行。
- 要在内表 现有行之前 插入新行， 请使用 INSERT 语句。

要将内表内 容复制到另 一个内表中 ，请使用 APPEND、 INSERT 或 MOVE 语句的变式 。

- 要将内表 行附加到另 一个内表中 ，请使用 APPEND 语句的变式 。
- 要将内表 行插入另一 个内表中， 请使用 INSERT 语句的变式 。
- 要将内表 条目内容复 制到另一个 内表中，并 且覆盖该目 标表格，请 使用 MOVE 语句。

关于如何使 用 SELECT 语句用数据 库表格中的 数据填充内 表的详细信 息，参见 将数据读入内表。

附加行

要将行附加 到内表中， 请使用 APPEND 语句，用法 如下：

语法

```
APPEND [<wa> TO|INITIAL LINE TO] <itab>.
```

该语句将新 行附加到内 表 <itab> 中。
通过使用 <wa> TO 选项，指定 要附加的源 区域 <wa>。 对于带表头 行的表格， 可以忽略 TO 选项。这样 ，表格工作 区域就成了 源区域。
可以使用 INITIAL LINE TO 选项替代 <wa> TO，将用 其类型的正 确值初始化 的行添加到 表格中。
APPEND 不考虑是否 存在标准关 键字相同的 行（参见*标识表格行*（页 98）） 。这样，可 能会出现相 同条目。
系统字段 SY-TABIX 在每个 APPEND 语句之后包 含附加行的 索引。

```
DATA: BEGIN OF ITAB OCCURS 10,
        COL1 TYPE C,
        COL2 TYPE I,
      END OF ITAB.

DO 3 TIMES.
  APPEND INITIAL LINE TO ITAB.
  ITAB-COL1 = SY-INDEX. ITAB-COL2 = SY-INDEX ** 2.
  APPEND ITAB.
ENDDO.

LOOP AT ITAB.
  WRITE: / ITAB-COL1, ITAB-COL2.
ENDLOOP.
```

本示例创建 带表头行和 两列的内表 ITAB。 表格用 DO 循环填充。 每次通过循 环时附加初 始化行，然 后用循环索 引填充表格 工作区域并 且附加循环 索引的平方 根。其输出 为：

```
          0
1         1
          0
2         4
          0
3         9
```

```
DATA: BEGIN OF LINE1,
         COL1(3) TYPE C,
         COL2(2) TYPE N,
         COL3    TYPE I,
      END OF LINE1.

DATA TAB1 LIKE LINE1 OCCURS 10.

DATA: BEGIN OF LINE2,
         FIELD1(1)  TYPE C,
         FIELD2     LIKE TAB1,
      END OF LINE2.

DATA  TAB2 LIKE LINE2 OCCURS 1.
```

```
LINE1-COL1 = 'abc'. LINE1-COL2 = '12'. LINE1-COL3 = 3.
APPEND LINE1 TO TAB1.

LINE1-COL1 = 'def'. LINE1-COL2 = '34'. LINE1-COL3 = 5.
APPEND LINE1 TO TAB1.

LINE2-FIELD1 = 'A'. LINE2-FIELD2 = TAB1.
APPEND LINE2 TO TAB2.

REFRESH TAB1.

LINE1-COL1 = 'ghi'. LINE1-COL2 = '56'. LINE1-COL3 = 7.
APPEND LINE1 TO TAB1.

LINE1-COL1 = 'jkl'. LINE1-COL2 = '78'. LINE1-COL3 = 9.
APPEND LINE1 TO TAB1.

LINE2-FIELD1 = 'B'. LINE2-FIELD2 = TAB1.
APPEND LINE2 TO TAB2.

LOOP AT TAB2 INTO LINE2.
   WRITE: / LINE2-FIELD1.
   LOOP AT LINE2-FIELD2 INTO LINE1.
      WRITE: / LINE1-COL1, LINE1-COL2, LINE1-COL3.
   ENDLOOP.
ENDLOOP.
```

其输出为：

```
A

abc 12          3

def 34          5

B

ghi 56          7

jkl 78          9
```

本示例创建 两个不带表 格工作区域 的内表（TAB1 和 TAB2） 。TAB2 有深层结构 ，因为 LINE2 的第二个组 件包含内表 TAB1 的结构。LINE1 被填充并附 加到 TAB1。 然后，将 LINE2 填充并附加 到 TAB2。 用 REFRESH 语句清除 TAB1 之后（参见 *初始化内表* （页 89） ），再重复 相同步骤。 请注意， TAB2 中的行数仅 在 OCCURS 参数中指定 为1。TAB2 的内容输出 。

根据标准关 键字附加行

要用有唯一 标准关键字 的行填充内 表，请使用 COLLECT 语句，用法 如下：

语法

```
COLLECT [<wa> INTO] <itab>.
```

该语句通过 使用 INTO 选项指定想 附加的源区 域 <wa>。 如果表格有 表头行，则 可以忽略 INTO 选项。这样 ，表格工作 区域就成了 源区域。

系统检查表 格条目的标 准关键字是 否相同（所 有非数字字 段，参见 *内表关键字* （页 98） ）。如果没有，COLLECT 语句的作用 与 APPEND 语句相似， 并将新行添 至表格中。

如果存在关 键字相同的 条目，COLLECT 语句不附加 新行，但将 工作区域中 数字字段的 内容添加到 现有条目中 数字字段的 内容中。系 统字段 SY-TABIX 包含处理过 的行的索引 。

为 COLLECT 指定的工作 区域必须与 内表的行类 型兼容，不 仅仅是可转 换为内表的行类型。COLLECT 语句无法用 于带深层结 构的内表， 例如，将内 表作为组件 的行。

如果仅使用 COLLECT 语句填充内 表，则不会 出现重复条 目。因此要 填充没有重 复条目的内 表，应该使 用 COLLECT 而不是 APPEND 或 INSERT。

```
DATA: BEGIN OF ITAB OCCURS 3,
         COLUMN1(3) TYPE C,
         COLUMN2(2) TYPE N,
         COLUMN3    TYPE I,
      END OF ITAB.
```

```
ITAB-COLUMN1 = 'abc'. ITAB-COLUMN2 = '12'. ITAB-COLUMN3 = 3.
COLLECT ITAB.
WRITE / SY-TABIX.

ITAB-COLUMN1 = 'def'. ITAB-COLUMN2 = '34'. ITAB-COLUMN3 = 5.
COLLECT ITAB.
WRITE / SY-TABIX.

ITAB-COLUMN1 = 'abc'. ITAB-COLUMN2 = '12'. ITAB-COLUMN3 = 7.
COLLECT ITAB.
WRITE / SY-TABIX.

LOOP AT ITAB.
   WRITE: / ITAB-COLUMN1, ITAB-COLUMN2, ITAB-COLUMN3.
ENDLOOP.
```

其输出为：

```
         1
         2
         1
abc 12          10
def 34           5
```

本示例创建 带表格工作 区域的内表 ITAB。前两个 COLLECT 语句和 APPEND 语句一样。在第三个 COLLECT 语句中，修 改了 ITAB 的第一行。 下面的数字 图表显示三 个步骤：

插入行

要在内表行 之前插入新 行，请使用 INSERT 语句，用法 如下：

语法

```
INSERT [<wa> INTO|INITIAL LINE INTO] <itab> [INDEX <idx>].
```

该语句通过 使用 INTO 选项指定想 插入的源区 域 <wa>。 如果表格有 表头行，则 可以忽略 INTO 选项。这样 ，表格工作 区域就成了 源区域。

可以使用 INITIAL LINE TO 选项替代 <wa> TO，将用 其类型的正 确值初始化 的行添至表 格中。

如果使用 INDEX 选项，则将 新行插入到 有索引 <idx> 的行之前。 插入之后， 新条目索引 为 <idx>，下行索引加 1。

如果表格包 含 <idx> - 1 条目，系统 将新条目附 加到最后的 现有表格行 之后。如果 表格的条目 小于 <idx> - 1，系统无 法插入条目 并将 SY-SUBRC 设置为 4。 如果操作成 功，将 SY-SUBRC 设置为 0。

如果使用不 带 INDEX 选项的 INSERT 语句，系统 只能在 LOOP - ENDLOOP 循环内通过 在当前行 (例如带 SY-TABIX 返回索引的 行）前插入 新条目来处 理它。

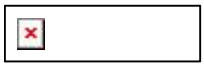
```
DATA: BEGIN OF LINE,
         COL1 TYPE I,
         COL2 TYPE I,
      END OF LINE.

DATA ITAB LIKE LINE OCCURS 10.

DO 2 TIMES.
   LINE-COL1 = SY-INDEX.
   LINE-COL2 = SY-INDEX ** 2.
   APPEND LINE TO ITAB.
ENDDO.

LINE-COL1 = 11. LINE-COL2 = 22.

INSERT LINE INTO ITAB INDEX 2.

INSERT INITIAL LINE INTO ITAB INDEX 1.

LOOP AT ITAB INTO LINE.
   WRITE: / SY-TABIX, LINE-COL1, LINE-COL2.
ENDLOOP.
```

其输出为：

```
         1          0          0
```

```
          2          1          1
          3         11         22
          4          2          4
```

本示例创建 内表 ITAB， 并用两行对 其进行填充 。在第二行 之前插入包 含值的新行 。然后，在 第一行之前 插入一个初 始化行。



```
DATA: BEGIN OF LINE,
         COL1 TYPE I,
         COL2 TYPE I,
       END OF LINE.

DATA ITAB LIKE LINE OCCURS 10.

DO 2 TIMES.
   LINE-COL1 = SY-INDEX.
   LINE-COL2 = SY-INDEX ** 2.
   APPEND LINE TO ITAB.
ENDDO.

LOOP AT ITAB INTO LINE.
   LINE-COL1 = 3 * SY-TABIX. LINE-COL2 = 5 * SY-TABIX.
   INSERT LINE INTO ITAB.
ENDLOOP.

LOOP AT ITAB INTO LINE.
   WRITE: / SY-TABIX, LINE-COL1, LINE-COL2.
ENDLOOP.
```

其输出为：

```
          1          3          5
          2          1          1
          3          6         10
          4          2          4
```

本示例创建 内表 ITAB， 并且用两行 对其进行填 充。在第一 个 LOOP - ENDLOOP 循环的每个 现有行之前 插入一个新 行。下图显 示示例是如 何工作的：

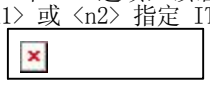
附加内表行

要将部分或 全部内表附 加到另一个 内表中，请 使用 APPEND 语句，用法 如下：

语法

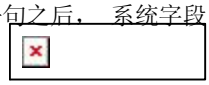
```
APPEND LINES OF <itab1> [FROM <n1>] [TO <n2>] TO <itab2>.
```

如果没有 FROM 和 TO 选项，该语 句将整个表 格 ITAB1 附加到 ITAB2 中。如果使 用这些选项 ，则可通过 索引 <n1> 或 <n2> 指定 ITAB1 中要附加的 第一或最后 一行。



用该方式将 表格行附加 到另一个表 格中的速度 比在循环中 逐行进行附 加快3到4倍 。

在 APPEND 语句之后， 系统字段 SY-TABIX 包含附加的 最后一行的 索引。



```
DATA: BEGIN OF ITAB OCCURS 10.
        COL1 TYPE C,
        COL2 TYPE I,
      END OF ITAB.
```

```
DATA JTAB LIKE ITAB OCCURS 10 WITH HEADER LINE.

DO 3 TIMES.
  ITAB-COL1 = SY-INDEX. ITAB-COL2 = SY-INDEX ** 2.
  APPEND ITAB.
  JTAB-COL1 = SY-INDEX. JTAB-COL2 = SY-INDEX ** 3.
  APPEND JTAB.
ENDDO.

APPEND LINES OF JTAB FROM 2 TO 3 TO ITAB.

LOOP AT ITAB.
  WRITE: / ITAB-COL1, ITAB-COL2.
ENDLOOP.
```

本示例创建 两个相同类 型的内表 ITAB 和 JTAB， 且都有表头 行。在 DO 循环中，用一系列平方 数填充 ITAB， 用一系列立 方数填充 JTAB。 然后将 JTAB 的最后两行 附加到 ITAB 中。ITAB 的输出如下 所示：

```
1         1
2         4
3         9
2         8
3        27
```

插入内表

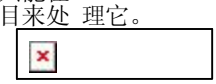
要将部分或 全部内表插 入到另一个 内表中，请 使用 INSERT 语句，用法 如下：

语法

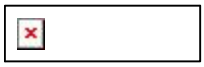
```
INSERT LINES OF <itab1> [FROM <n1>] [TO <n2>]
                INTO <itab2> [INDEX <idx>].
```

如果没有 FROM 和 TO 选项，该语 句将整个表 格 ITAB1 附加到 ITAB2 中。如果使 用这些选项 ，则可通过 索引 <n1> 或 <n2> 指定 ITAB1 中要附加的 第一或最后 一行。

如果使用 INDEX 选项，将 <itab1> 的行插入到 <itab2> 中索引为 <idx> 的行之前。 如果不使用 INDEX 选项，系统 只能在 LOOP - ENDLOOP 块中通过在 当前行（例 如，其索引 在 SY-TABIX 中返回的行 ）之前插入 新条目来处 理它。



取决于表格 大小和插入 地点的不同 ，用该方式 将一个表格 插入到另一 个中的速度比在循环中 逐行进行插 入要快近20倍 。



```
DATA: BEGIN OF LINE,
        COL1 TYPE I,
        COL2 TYPE I,
      END OF LINE.

DATA ITAB LIKE LINE OCCURS 10,
     JTAB LIKE LINE OCCURS 10.

DO 3 TIMES.
  LINE-COL1 = SY-INDEX. LINE-COL2 = SY-INDEX ** 2.
  APPEND LINE TO ITAB.

  LINE-COL1 = SY-INDEX. LINE-COL2 = SY-INDEX ** 3.
  APPEND LINE TO JTAB.
ENDDO.

INSERT LINES OF ITAB INTO JTAB INDEX 1.

LOOP AT JTAB INTO LINE.
  WRITE: / SY-TABIX, LINE-COL1, LINE-COL2.
ENDLOOP.
```

本示例创建 两个类型相 同的内表： ITAB 和 JTAB， 并且用 3 行对其进行 填充。然后 将整个表格 ITAB 插入到 JTAB 的第一行之 前。JTAB 的输出如下 所示：

```
1          1          1
2          2          4
3          3          9
4          1          1
5          2          8
6          3         27
```

复制内表
如果想一次 将内表的全 部内容复制 到另一内表 中，请使用 MOVE 语句或赋值 操作符 (=)，用 法如下：

语法
MOVE <itab1> TO <itab2>.
该语句等价 于：
<itab2> = <itab1>.
也可进行多 重赋值，例 如，
<itab4> = <itab3> = <itab2> = <itab1>.
也是可能的 。ABAP/4 从右到左进 行处理：
<itab2> = **<itab**1>.
<itab3> = **<itab**2>.
<itab4> = **<itab**3>.
这些语句执 行完整操作 。复制整个 表格内容， 包括作为表 格组件的任 何其它内表 的数据。覆 盖目标表格 原来的内容 。
语法与复制 基本字段相 同（详细信 息，参见 *基本分配操作* (页 6－2))。
对于有表头 行的表格， 表格工作区 域和表格本 身同名。要 在上述语句 中进行区分 ，必须在名 称之后输入 两个方括号 ([]) 来定位内表 而不是表格 工作区域。

可以仅将内 表复制到其 它内表中。 内表不能转 换为字段串 或基本字段 。要将内表复制到其它 内表中，其 行类型必须 是可转换的 （关于转换 的详细信息 ，参见 *内表的可转换性* (页 6－43)) 。

```
DATA: BEGIN OF LINE,
         COL1,
         COL2,
      END OF LINE.

DATA ETAB LIKE LINE OCCURS 10 WITH HEADER LINE.
DATA FTAB LIKE LINE OCCURS 10.

LINE-COL1 = 'A'. LINE-COL2 = 'B'.
APPEND LINE TO ETAB.

MOVE ETAB[] TO FTAB.

LOOP AT FTAB INTO LINE.
   WRITE: / LINE-COL1, LINE-COL2.
ENDLOOP.
```

其输出为：

```
A B
```

本示例创建 两个行结构 LINE 相同的内表 ：ETAB 和 FTAB。 创建的ETAB 包含表格工 作区域。在 用 APPEND 语句填充 ETAB 之后，将其 内容复制到 FTAB。 请注意方括 号([]) 的 用法。

```
DATA: ITAB TYPE I OCCURS 10,
      FTAB TYPE F OCCURS 10,
      FL   TYPE F.

DO 3 TIMES.
   APPEND SY-INDEX TO ITAB.
ENDDO.

FTAB = ITAB.

LOOP AT FTAB INTO FL.
   WRITE: / FL.
ENDLOOP.
```

其输出为：

1.000000000000000E+00

2.000000000000000E+00

3.000000000000000E+00

其中，内表 ITAB 具有基本行 类型 I，FTAB 具有基本行 类型 F。这些行 类型都是可 转换的（关 于可转换性 的详细信息 ，参见 *基本数据类型的可转换性* （页 6－37）） 并且可将 ITAB 复制到 FTAB 中。

```
DATA: BEGIN OF ILINE,
         NUM TYPE I,
      END OF ILINE,

      BEGIN OF FLINE,
         NUM TYPE F,
      END OF FLINE,

      ITAB LIKE ILINE OCCURS 10,
      FTAB LIKE FLINE OCCURS 10.

DO 3 TIMES.
   ILINE-NUM = SY-INDEX.
   APPEND ILINE-NUM TO ITAB.
ENDDO.

FTAB = ITAB.

LOOP AT FTAB INTO FLINE.
   WRITE: / FLINE-NUM.
ENDLOOP.
```

其输出为：

6.03823403895813E-154

6.03969074613219E-154

6.04114745330626E-154

其中，内表 ITAB 和 FTAB 的行类型都 是字段串， 每个都包含 类型 I 或 F 的组件。这 些字段串可 转换，但不 兼容。因此 ，将 ITAB 赋给 FTAB 之后，将表 格 ITAB 的内容转换 为类型 C 字段，然后 写入 FTAB（ 关于字段串 可转换性的 详细信息， 参见 *字段串的可转换性* （页 6－41））。系统将 传输的数据 解释为类型 F 字段，并获 得毫无意义 的结果。

读取内表

要读取内表 的内容以便 进一步处理 ，可以使用 LOOP 或 READ 语句。
使用 LOOP 语句逐行读 取内表。
可以用 READ 语句选定单 行：
读取单行后 ，可以作如 下工作
可以使用 DESCRIBE 语句确定内 表属性。

逐行读取内 表
要将内表逐 行读入工作 区域，可以 使用 LOOP 语句编一个 循环。语法 如下所示：

语法
```
LOOP AT <itab> [INTO <wa>] [FROM <n1>] [TO <n2>]
                           [WHERE <condition>].
  .....
ENDLOOP.
```
用 INTO 选项指定目 标区域 <wa>。 如果表格有 表头行，则 可以忽略 INTO 选项。这样 ，表格工作 区域就成了 目标区域。
逐行将内表 <itab> 读入 <wa> 或表格工作 区域 <itab>。 对于读取的 每一行，系 统都处理以 LOOP 开始，以 ENDLOOP 结束的语句 块。可以用 控制关键字 AT 在 LOOP - ENDLOOP 块内控制语 句块流（参 见 *循环处理* (页 **84**) ）。
在语句块内 ，系统字段 SY-TABIX 包含当前行 的索引。处 理完表格的 所有行之后 循环结束。 在 ENDLOOP 语句之后， 如果至少读 取了一行， 则将系统字 段 SY-SUBRC 设置为 0。 否则，将其 设置为 4。
可以使用 FROM、 TO 或 WHERE 选项限制要 在循环中进 行处理的行 数。

- 使用 FROM 选项，可以 用 <n1> 指定要读取 的第一行
- 使用 TO 选项，可以 用 <n2> 指定要读取 的最后一行 。
- 用 WHERE 选项，可以 指定 <condition> 的任何逻辑 表达式（参 见 *编写逻辑表达式* (页 7 - 3) ）。第一个 操作数必须 是内表行结 构的组件。 如果在循环 中使用控制 关键字 AT ，则不能使 用 WHERE 选项。

FROM 和 TO 选项限制系 统必须读取 的行数。WHERE 选项仅避免 对工作区域 进行不必要 的填充。用 WHERE 选项，系统 必须读取所 有行。为了 提高效率， 应该尽可能 使用 FROM 和 TO 选项。在某 些条件下用 EXIT 语句（参见 *完全终止循环* (页 7 - 22) ）而不是 WHERE 选项跳出循 环也十分有 效。

```
DATA: BEGIN OF LINE,
         COL1 TYPE I,
         COL2 TYPE I,
      END OF LINE.

DATA ITAB LIKE LINE OCCURS 10.

DO 30 TIMES.
   LINE-COL1 = SY-INDEX.  LINE-COL2 = SY-INDEX * SY-INDEX.
   APPEND LINE TO ITAB.
ENDDO.

LOOP AT ITAB INTO LINE FROM 10 TO 25 WHERE COL2 > 400.
   WRITE: / SY-TABIX, LINE-COL2.
ENDLOOP.
```

其输出为：

```
        21        441
        22        484
        23        529
        24        576
        25        625
```

在此，根据 字段串 LINE 创建内表 ITAB。在 DO 循环中用1和 30之间的 数字以及这 些数字的平 方对表格进 行填充。用 LOOP 语句逐行读 取表格。将 要读取的行 的索引限制 在10和25之 间，并将每 行第二个组 件的内容限 制为大于400的 数字。

用索引读取 单行
要用索引从 内表中读取 单行，请使 用 READ 语句，用法 如下：

语法
```
READ TABLE <itab> [INTO <wa>] INDEX <idx>.
```
用 INTO 选项指定目 标区域 <wa>。 如果表格有 表头行，可 以忽略 INTO 选项。这样 ，表格工作 区域就成了 目标区域。

系统用索引 <idx> 从表格 <itab> 中读取行。这比用关键 字访问表格 要快(参见 *用关键字读取单行* (页 50))。
如果找到有 指定索引的 条目，则将 系统字段 SY-SUBRC 设置为 0， 而且 SY-TABIX 包含该行的 索引。否则 ， SY-SUBRC 包含非 0 值 。

如果 <idx> 小于或等于 0，则会发 生实时错误 。如果 <idx> 超过表格大 小，系统将 SY-SUBRC 中的返回代 码值设置为 4。

```
DATA: BEGIN OF ITAB OCCURS 10,
        COL1 TYPE I,
        COL2 TYPE I,
      END OF ITAB.

DO 20 TIMES.
   ITAB-COL1 = SY-INDEX.
   ITAB-COL2 = 2 * SY-INDEX.
   APPEND ITAB.
ENDDO.

READ TABLE ITAB INDEX 7.

WRITE: SY-SUBRC, SY-TABIX.
WRITE: / ITAB-COL1, ITAB-COL2.
```

其输出为：

```
    0         7

          7         14
```

在此创建有 表头行的内 表 ITAB， 并用20行 对其进行填 充。读取并 输出索引为 7 的行。

读取有关键 字的单行
按如下方式 从内表中读 取有关键字 的单行：用
读取有关键 字的单行时 ，可以用二 分法搜索代 替顺序搜索 。关于二分 法搜索的详 细信息，参 见

用自定义关 键字读取单 行
要从有自定 义关键字的 内表中读取 单行，请使 用 READ 语句的 WITH KEY 选项，用法 如下：

语法
READ TABLE <itab> [INTO <wa>] WITH KEY <key> [BINARY SEARCH].
用 INTO 选项可以指 定目标区域 。如果表格 有表头行， 则可以忽略 INTO 选项。这样 ，表格工作 区域就成了 目标区域。
系统读取 <itab> 中匹配 <key> 中所定义的 关键字的第 一个条目。关于 二分法搜索 选项的详细 信息，参见 *二分法搜索* (页 **321**) 。
如果找到有 适当关键字 的条目，则 将系统字段 SY-SUBRC 设置为 0， 并且 SY-TABIX 包含该行的 索引。否则 ，将 SY-SUBRC 设置为非 0 值 。
如下所述， 可以定义多 个关键字 <key>：

定义一系列 关键字段
要定义自己 的一系列关 键字段，请 使用 WITH KEY 选项，用法 如下：

语法
....WITH KEY <k_1> = <F1> ... <k_n> = <f_n> ...
自定义关键 字包含表格 组件 <k_1>...<k_n>。 字段 <F1>...<f_n> 是关键字段 的内容必须 匹配的值。
如果 <f_i> 的数据类型 与数据类型 <k_i> 不兼容，则 <f_i> 转换为类型 <k_i>。
可以用 (<n_i>) 代替 <k_i> 来实时设置 关键字段。 关键字段是 字段 <n_i> 的内容。如 果在运行时 <ni> 为空，则系 统忽略该关 键字段。如 果 <n_i> 包含无效的 组件名称， 则发生实时 错误。
用户可以为 任何在关键 字中使用的 组件指定偏 移量和长度 （参见 *指定数据对象的偏移值* (页 6 - 35))。

将整行定义 为关键字
通过使用 WITH KEY 选项可将内 表整行定义 为其关键字 ，如下所示 ：

语法
....WITH KEY = <value> ...
如果〈value〉的数据类型 与表格行的 数据类型不 兼容，则将〈value〉转换为表格 行的数据类 型。对于此类关 键字，也可 以选择由某 个基本数据 类型或内表 直接定义的 ，而不是由 字段串直接 定义的特定 内表行。

将行首定义 为关键字
要将内表的 行首定义为 关键字，请 使用 WITH KEY 选项，用法 如下：

语法
....WITH KEY <k> ...
系统将（左 对齐）的行 首与〈k〉进行比较。〈k〉不能包含内 表或包含内 表的结构。 与上面两个 选项不同之 处在于用〈k〉的数据类型 进行比较。

```
DATA: BEGIN OF LINE,
         COL1 TYPE C,
         COL2 TYPE P DECIMALS 5,
         COL3 TYPE I,
         COL4 TYPE I,
       END OF LINE.

DATA ITAB LIKE LINE OCCURS 10.

DO 10 TIMES.
   LINE-COL1 = SY-INDEX.
   LINE-COL2 = SQRT( SY-INDEX ).
   LINE-COL3 = SY-INDEX ** 2.
   LINE-COL4 = SY-INDEX ** 3.
   APPEND LINE TO ITAB.
ENDDO.

READ TABLE ITAB INTO LINE WITH KEY COL3 = 9 COL4 = 36.
WRITE: / SY-SUBRC, SY-TABIX.

READ TABLE ITAB INTO LINE WITH KEY COL3 = 9 COL4 = 27.
WRITE: / SY-SUBRC, SY-TABIX.

READ TABLE ITAB INTO LINE WITH KEY '2'.
WRITE: / SY-SUBRC, SY-TABIX.
```

其输出为：

```
    4          0
    0          3
    0          2
```

在此，创建 包含四列的 内表。对表 格的10行 进行填充之 后，用自定 义关键字读 取单行。有 关键字段 COL3、COL4 的自定义序 列的第一个 READ 语句失败， 第二个 READ 语句找到索 引为 3 的行。第三 个 READ 语句搜索以 “2”开始 的表格行并 找到索引为 2的行。下 图显示主要 步骤：

```
DATA ITAB   TYPE I OCCURS 10,
DATA SQUARE TYPE I.

DO 30 TIMES.
   SQUARE = SY-INDEX ** 2.
   APPEND SQUARE TO ITAB.
ENDDO.

READ TABLE ITAB INTO SQUARE WITH KEY = 25.

WRITE: SY-SUBRC, SY-TABIX.
```

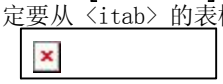
其输出为：

```
    0          5
```

在此创建包 含基本类型 I 行的内表。 填充完表格 之后，读取 值为 25 且索引为 5 的行。

读取有标准 关键字的单 行
要从内表中 读取有特定 标准关键字 的第一行， 请使用 READ 语句，用法 如下：

语法
READ TABLE <itab> [INTO <wa>] [BINARY SEARCH].
用户必须指 定要从 <itab> 的表格工作 区域中读取 行的关键字 。



读取语句的 该变式只能 用于有表头 行的内表。

系统在表格 中搜索第一 个条目以匹 配表格工作 区域中的所 有标准关键 字段并将该 行读入表格 工作区域。 如果使用 INTO 选项，则将 该行读入工 作区域 <wa>。
标准关键字 包含 *内表关键字* (页 98) 中所述的全 部关键字段 ，其中不包 含 SPACE。 关于“二分法搜索”选项的详细 信息，参见 *二分法搜索* (页 **321**) 。
如果找到有 匹配关键字 的条目，则 将系统字段 SY-SUBRC 设置为 0 并 且 SY-TABIX 包含该行的 索引。 否则 ，将 SY-SUBRC 设置为 4。

```
DATA: BEGIN OF LINE,
        COL1 TYPE C,
        COL2 TYPE P DECIMALS 5,
      END OF LINE.

DATA ITAB LIKE LINE OCCURS 10 WITH HEADER LINE.

DO 5 TIMES.
   LINE-COL1 = SY-INDEX.  LINE-COL2 = SQRT( SY-INDEX ).
   APPEND LINE TO ITAB.
ENDDO.

ITAB-COL1 = '2'.

READ TABLE ITAB INTO LINE.

WRITE LINE-COL2.
```

其输出为：

```
          1.41421
```

在此创建有 表头行的内 表 ITAB，并用 1 到 5 之间数值的 平方根对其 进行填充。由于该表格 时类型 C，所以， 其缺省关键 字是 COL1。 在将“2” 赋给表格工 作区域的 COL1 之后，读取 内表的各行 并且输出 2 的平方根。 下图显示主 要步骤：

二分法搜索
用关键字读 取单行时， 可以执行二 分法搜索以 代替标准顺 序搜索。为 此，请使用 READ 语句的 二分法搜索 选项。

语法
READ TABLE <itab> ..... BINARY SEARCH.
如果使用 二分法搜索 选项，则必 须按关键字 中指定的次 序对内表进 行排序。
如果系统找 到匹配指定 关键字的多 行，则读取 索引最低的 行。
二分法搜索 比线性搜索 要快。因此 ，应尽可能 将内表排序 并且使用 二分法搜索 选项。

比较单行的 内容
要将使用 READ 语句读取的 单行内容与 目标区域的 内容进行比 较，可使用 READ 语句的 COMPARING 选项，用法 如下：

语法
READ TABLE <itab> [INTO <wa>] <key-option> COMPARING <fields>.
系统读取由 关键字或 <key option> 中的索引指 定的单行。 读取行之后 ，将 <fields> 中指定的组 件与目标区 域中的相应 组件进行比 较。可以用 INTO 选项指定目 标区域 <wa>。 如果表格有 表头，则可以忽略 INTO 选项。这样 ，表格工作 区域就成了 目标区域。
对于 <field>， 可以编写一 系列组件
... <F1> ...<fn>。
也可以用
... ALL FIELDS
指定所有组 件。
如果系统找 到包含指定 <key-option> 的条目，且 进行比较的 字段内容相 同，则将 SY-SUBRC 设置为 0。 如果进行比 较的字段内 容不同，则 返回值 2。如果系 统找不到条 目，则包含 4。
如果系统找 到条目，则 无论比较结 果如何，都 将其读入目 标区域。关 于 READ 语句的详细 信息，参见 READ 的关键字文 档。

```
DATA: BEGIN OF LINE,
         COL1 TYPE I,
         COL2 TYPE I,
         COL3 TYPE I,
      END OF LINE.

DATA ITAB LIKE LINE OCCURS 10.

DO 10 TIMES.
   LINE-COL1 = SY-INDEX.
   LINE-COL2 = SY-INDEX ** 2.
   LINE-COL3 = SY-INDEX ** 3.
   APPEND LINE TO ITAB.
ENDDO.

LINE-COL1 = 2. LINE-COL2 = 4. LINE-COL3 = 8.
```

```
READ TABLE ITAB INTO LINE INDEX 4 COMPARING COL1 COL2.
WRITE: / SY-SUBRC, SY-TABIX.

READ TABLE ITAB INTO LINE INDEX 4 COMPARING COL1 COL2.
WRITE: / SY-SUBRC, SY-TABIX.
```

其输出为：

```
    2          4

    0          4
```

在此创建并 填充内表 ITAB 。在第一个 READ 语句之后， 将 SY-SUBRC 设置为2， 因为找到了 索引为 4 的行，但内 表和目标区 域 LINE 的 COL1 和 COL2 内容不同。 但将内表读 入 LINE 并在下一个 READ 语句之后 SY-SUBRC 返回0。下 图显示主要 步骤：

读取部分单 行
要读取部分 单行，请使 用 READ 语句的 TRANSPORTING 选项，用法 如下：

语法
READ TABLE <itab> [INTO <wa>] <key-option> TRANSPORTING <fields>.
系统读取由 关键字或 <key option> 中索引指定 的单行。读 取行之后， 将 <fields> 中指定的组 件传输给目 标区域。可 以使用 INTO 选项指定目 标区域 <wa>。 如果表格有 表头行，可 以忽略 INTO 选项。这样 ，表格工作 区域就成了 目标区域。
对于 <fields>， 可以用
... <F1> ...<f_n>
指定一系列 组件。
也可以用
... NO FIELDS
指定不传输 任何组件。
对于后一种 情况，READ 语句只影响 系统字段 SY-SUBRC 和 SY-TABIX。
关于 READ 语句的详细 信息，参见 关键字文档 。

```
DATA: BEGIN OF LINE,
         COL1 TYPE I,
         COL2 TYPE I,
```

```
          COL3 TYPE I,
        END OF LINE.

DATA ITAB LIKE LINE OCCURS 10.

DO 10 TIMES.
   LINE-COL1 = SY-INDEX.
   LINE-COL2 = SY-INDEX ** 2.
   LINE-COL3 = SY-INDEX ** 3.
   APPEND LINE TO ITAB.
ENDDO.

CLEAR LINE.
READ TABLE ITAB INTO LINE INDEX 5 TRANSPORTING COL2.
WRITE: / LINE-COL1, LINE-COL2, LINE-COL3.
```

其输出为：

```
         0         25          0
```

在此创建并 填充内表 ITAB。 READ 语句仅将 COL2 字段读入字 段串 LINE。

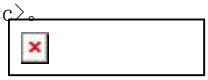
确定内表属 性

如果在处理 过程中想知 道内表一共 包含多少行 ，或者想知 道定义的 OCCURS 参数的大小 ，请使用 DESCRIBE 语句，用法 如下：

语法

```
DESCRIBE TABLE <itab> [LINES <lin>] [OCCURS <occ>].
```

如果使用 LINES 参数，则将 填充行的数 量写入变量 <lin>。 如果使用 OCCURS 参数，则将 行的初始号写入变量 <occ>。



```
DATA: BEGIN OF LINE,
         COL1 TYPE I,
         COL2 TYPE I,
       END OF LINE.

DATA ITAB LIKE LINE OCCURS 10.

DATA: LIN TYPE I, OCC TYPE I.

DESCRIBE TABLE ITAB LINES LIN OCCURS OCC.
WRITE: / LIN, OCC.

DO 1000 TIMES.
   LINE-COL1 = SY-INDEX.
   LINE-COL2 = SY-INDEX ** 2.
   APPEND LINE TO ITAB.
ENDDO.

DESCRIBE TABLE ITAB LINES LIN OCCURS OCC.
WRITE: / LIN, OCC.
```

其输出为：

```
         0         10
     1.000         10
```

在此创建内 表 ITAB。 在填充表格 前后执行 DESCRIBE 语句。更改 当前行号， 但无法更改 初始行号。

更改和删除 内表行

要修改已填 充的内表内 容，可以

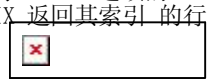
更改行

删除行

用 MODIFY 更改行

要用 MODIFY 语句更改行 ，请使用：

语法
MODIFY <itab> [FROM <wa>] [INDEX <idx>].
FROM 选项中指定 的工作区域 <wa> 代替 <itab> 中的行。如 果表格有表 头行，可以 忽略 FROM 选项。这样 ，表格工作 区域就代替 行。
如果使用 INDEX 选项，则新 行代替索引 为 <idx> 的现有行。 如果替换成 功，则将 SY-SUBRC 设置为 0。 如果内表包 含的行少于 <idx>， 则不更改任 何行并且 SY-SUBRC 包含 4。
如果使用没 有 INDEX 选项的 MODIFY 语句，则系 统只能在 LOOP - ENDLOOP 块中通过更 改当前行（ 例如由 SY-TABIX 返回其索引 的行）来处 理它。

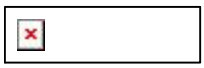


```
DATA: BEGIN OF LINE,
         COL1 TYPE I,
         COL2 TYPE I,
      END OF LINE.

DATA ITAB LIKE LINE OCCURS 10.

DO 3 TIMES.
   LINE-COL1 = SY-INDEX.
   LINE-COL2 = SY-INDEX ** 2.
   APPEND LINE TO ITAB.
ENDDO.

LINE-COL1 = 10. LINE-COL2 = 10 ** 2 .

MODIFY ITAB FROM LINE INDEX 2.

LOOP AT ITAB INTO LINE.
   WRITE: / SY-TABIX, LINE-COL1, LINE-COL2.
ENDLOOP.
```

其输出为：

```
         1         1          1
         2        10        100
         3         3          9
```

在此创建内 表 ITAB， 并用三行对 其进行填充 。用字段串 LINE 的内容代替 第二行。



```
DATA: BEGIN OF LINE,
         COL1 TYPE I,
         COL2 TYPE I,
      END OF LINE.

DATA ITAB LIKE LINE OCCURS 10.

DO 3 TIMES.
   LINE-COL1 = SY-INDEX.
   LINE-COL2 = SY-INDEX ** 2.
   APPEND LINE TO ITAB.
ENDDO.

LOOP AT ITAB INTO LINE.
   IF SY-TABIX = 2.
      LINE-COL1 = SY-TABIX * 10.
      LINE-COL2 = ( SY-TABIX * 10 ) ** 2 .
      MODIFY ITAB FROM LINE.
   ENDIF.
ENDLOOP.

LOOP AT ITAB INTO LINE.
   WRITE: / SY-TABIX, LINE-COL1, LINE-COL2.
ENDLOOP.
```

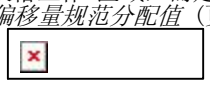
其输出为：

```
         1         3          5
         2        20        400
         3         3          9
```

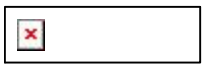
在此创建内 表 ITAB， 并用两行对 其进行填充 。在第一个 LOOP - ENDLOOP 块中，用字 段串 LINE 的内容代替 第二行。

用 WRITE TO 更改行
要用 WRITE TO 语句更改行 ，请使用下 列语法：

语法
WRITE <f>[+<o1>][(<l1>)] TO <itab>[+<o2>][(<l2>)] INDEX <idx>.
将字段 <f> 中偏移量为 <o1>， 长度为 <l1> 部分的内容 复制到索引 为 <idx> 的表格行中 ，**覆盖**偏移量为 <o2>， 长度为 <l2> 的部分。请 注意，即使 对于有表头 行的表格， 带 INDEX 选项的 WRITE TO 语句也不访 问表格工作 区域，而是 访问表格的 某一行。
该语句是 *用偏移量规范分配值* (页 6 - 3) 中所述的 WRITE TO 语句的变式 。



WRITE TO 语句不能识 别表格行的 结构。SAP 建议只在（ 例如）转换 已知其确切 位置的标志 时才使用该 语句。另一 种情况是用 一个基本字 符字段定义 的内表。该 结构的表格 非常重要， 例如，用于 程序的动态 生成（参见 动态生成程序）。



```
DATA CODE(72) OCCURS 10 WITH HEADER LINE.

CODE = 'This is the first line.'.
APPEND CODE.

CODE = 'This is the second line. It is ugly.'.
APPEND CODE.

CODE = 'This is the third and final line.'.
APPEND CODE.

WRITE 'nice.' TO CODE+31 INDEX 2.

LOOP AT CODE.
   WRITE / CODE.
ENDLOOP.
```

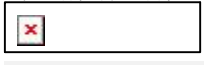
其输出为：

```
This is the first line.

This is the second line. It is nice.

This is the third and final line.
```

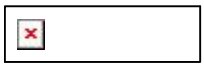
在此用72个 字符长的基 本类型 C 字段定义内 表 CODE。 用三行对表 格进行填充 之后，使用 WRITE TO 语句更改第 二行。单词 “ugly” 由单词“nice” 代替。

在循环中删 除行
要在循环中 从内表中删 除行，请使 用 DELETE 语句，用法 如下：

语法
DELETE <itab>.
系统只能在 LOOP - ENDLOOP 块中处理该 语句（参见 *逐行读取内表* (页 36))。这删除 当前行（例 如有 SY-TABIX 返回的索引 的行)。



删除第一行 后，可以取 消当前行的 定义并取消 其对 SY-TABIX 内容的赋值 。要在该循 环内进一步 处理行，请 仅使用有 INDEX 选项的语句 。



```
DATA: BEGIN OF LINE,
         COL1 TYPE I,
         COL2 TYPE I,
      END OF LINE.

DATA ITAB LIKE LINE OCCURS 10.

DO 30 TIMES.
   LINE-COL1 = SY-INDEX.
   LINE-COL2 = SY-INDEX ** 2.
   APPEND LINE TO ITAB.
ENDDO.

LOOP AT ITAB INTO LINE.
   IF LINE-COL1 < 28.
      DELETE ITAB.
   ENDIF.
ENDLOOP.

LOOP AT ITAB INTO LINE.
   WRITE: / SY-TABIX, LINE-COL1, LINE-COL2.
ENDLOOP.
```

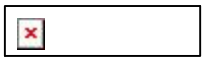
其输出为：

1	28	784
2	29	841
3	30	900

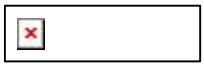
在此创建内 表 ITAB并 用30行对 其进行填充 。在第一个 LOOP - ENDLOOP 块中，删除 COL1 字段中所有 条目小于28的 行。

用索引删除 行
要使用索引 删除行，请 使用有 INDEX 选项的 DELETE 语句，用法 如下：

语法
DELETE <itab> INDEX <idx>.
如果使用 INDEX 选项，则从 ITAB 中删除索引 为 <idx> 的行。删除 行之后，下 面行的索引 减 1。
如果操作成 功，则将 SY-SUBRC 设置为 0。 否则，如果 不存在索引 为 <idx> 的行，则 SY-SUBRC 包含 4。



如果在 LOOP - ENDLOOP 块中删除某 一条目，则 当前行及其 对 SY-TABIX 内容的赋值 可成为未定 义。要在该 循环内进一 步处理行， 请仅使用有 INDEX 选项的语句 。



```
DATA: BEGIN OF LINE,
         COL1 TYPE I,
         COL2 TYPE I,
      END OF LINE.

DATA ITAB LIKE LINE OCCURS 10.

DO 5 TIMES.
   LINE-COL1 = SY-INDEX.
   LINE-COL2 = SY-INDEX ** 2.
   APPEND LINE TO ITAB.
ENDDO.

DELETE ITAB INDEX: 2, 3, 4.

WRITE: 'SY-SUBRC',SY-SUBRC.

LOOP AT ITAB INTO LINE.
   WRITE: / SY-TABIX, LINE-COL1, LINE-COL2.
ENDLOOP.
```

其输出为：

```
SY-SUBRC     4

         1          1          1
         2          3          9
         3          5         25
```

在此创建内 表 ITAB并 用五行对其 进行填充。 然后处理一 系列语句以 删除索引为 2、3 和 4 的三行。删 除索引为 2 的行之后， 下面行的索 引减 1。因此， 下次删除则 删除初始索 引为 4 的行。第三 次删除则会 失败，因为 表格现在仅 包含 3 行，SY-SUBRC 返回 4。

删除邻近的 重复条目

要删除邻近 重复条目， 请使用 DELETE 语句，用法 如下：

语法

DELETE ADJACENT DUPLICATE ENTRIES FROM <itab> [COMPARING <comp>].

系统从内表 <itab> 中删除所有 邻近重复条 目。

完成以下比 较标准之后 ，条目就会 重复：

- 如果没有 COMPARING 选项，则标 准关键字段 的内容必须 相同（参见 *内表关键字* （页 98） ）。
- 如果有 COMPARING 选项

  .... COMPARING <F1> <F2> ... ，

  指定字段 <**F1**>〈**F2**〉... 的内容必须 相同。也可 以通过写入 （<name>）代替 <**F1**>在运行时在 括号中指定 字段名。字 段 <name> 包含排序关 键字段的名 称。如果 <name> 在运行时为 空，则系统 将其忽略。 如果包含无 效的组件名 ，则会发生 实时错误。
- 如果有 COMPARING 选项

  .... COMPARING ALL FIELDS ，

  所有字段的 内容必须相 同。

如果系统找 到并删除至 少一个重复 条目，则将 SY-SUBRC 设置为 0。 否则，将其 设置为 4。

如果表格根 据指定的比 较标准进行 过排序，则 可使用该语 句从内表中 删除**所有**重 复条目。

```
DATA: BEGIN OF LINE,
        COL1 TYPE I,
        COL2 TYPE C,
      END OF LINE.

DATA ITAB LIKE LINE OCCURS 10.

LINE-COL1 = 1. LINE-COL2 = 'A'. APPEND LINE TO ITAB.
LINE-COL1 = 1. LINE-COL2 = 'A'. APPEND LINE TO ITAB.
LINE-COL1 = 1. LINE-COL2 = 'B'. APPEND LINE TO ITAB.
LINE-COL1 = 2. LINE-COL2 = 'B'. APPEND LINE TO ITAB.
LINE-COL1 = 3. LINE-COL2 = 'B'. APPEND LINE TO ITAB.
LINE-COL1 = 4. LINE-COL2 = 'B'. APPEND LINE TO ITAB.
LINE-COL1 = 5. LINE-COL2 = 'A'. APPEND LINE TO ITAB.

LOOP AT ITAB INTO LINE.
   WRITE: / LINE-COL1, LINE-COL2.
ENDLOOP.

DELETE ADJACENT DUPLICATES FROM ITAB COMPARING ALL FIELDS.

SKIP TO LINE 3.
LOOP AT ITAB INTO LINE.
   WRITE: /14 LINE-COL1, LINE-COL2.
ENDLOOP.

DELETE ADJACENT DUPLICATES FROM ITAB COMPARING COL1.

SKIP TO LINE 3.
LOOP AT ITAB INTO LINE.
   WRITE: /28 LINE-COL1, LINE-COL2.
ENDLOOP.

DELETE ADJACENT DUPLICATES FROM ITAB.

SKIP TO LINE 3.
LOOP AT ITAB INTO LINE.
   WRITE: /42 LINE-COL1, LINE-COL2.
ENDLOOP.
```

其输出为：

```
1 A          1 A          1 A          1 A
1 A          1 B          2 B          2 B
1 B          2 B          3 B          5 A
2 B          3 B          4 B
3 B          4 B          5 A
4 B          5 A
5 A
```

在此第一个 DELETE 语句从 ITAB 中删除第二 行，因为第 二行与第一 行的内容相 同。第二个 DELETE 语句从剩余 表格中删除 第二行，因 为字段 COL1 的内容与第 一行中的相 同。第三个 DELETE 语句从剩余 表格中删除 第三和第四 行，因为缺 省关键字段 COL2 的内容与第 二行中的相 同。尽管第 一行与第五 行缺省关键 字的内容相 同，但并不 删除第五行 ，因为与第 一行不相邻 。

删除选定行
要删除一组 选定行，使 用 DELETE 语句，用法 如下：

语法
DELETE <itab> [FROM <n_1>] [TO <n_2>] [WHERE <condition>].
用户必须至 少指定三个 选项之一。 如果使用没 有 WHERE 选项的该语 句，则系统 从 <itab> 中删除所有 索引在 <n_1> 和 <n_2> 之间的行。 如果不使用 FROM 选项，则系 统从第一行 开始删除。 如果不使用 TO 选项，则系 统删除所有 行直到最后 一行。
如果使用 WHERE 选项，则系 统仅从 <itab> 中删除满足 条件 <condition> 的行。对于 <condition>， 可指定任何 逻辑表达式 （参见 *编写逻辑表达式* (页 7 - 3))。第一个 操作数必须 是内表行结 构的组件。
如果系统至 少删除一行 ，则将 SY-SUBRC 设置为 0。 否则，将其 设置为 4。

```
DATA: BEGIN OF LINE,
        COL1 TYPE I,
        COL2 TYPE I,
      END OF LINE.

DATA ITAB LIKE LINE OCCURS 10.

DO 40 TIMES.
   LINE-COL1 = SY-INDEX.
   LINE-COL2 = SY-INDEX ** 2.
   APPEND LINE TO ITAB.
ENDDO.

DELETE ITAB FROM 3 TO 38 WHERE COL2 > 20.

LOOP AT ITAB INTO LINE.
   WRITE: / LINE-COL1, LINE-COL2.
ENDLOOP.
```

其输出为：

```
 1          1
 2          4
 3          9
 4         16
39      1.521
40      1.600
```

在此，如果 COL2 之值大于 20，则系 统删除 ITAB 中 3 行和 39 行之间的所 有条目。

内表排序

要将内表排 序，请使用 SORT 语句，用法 如下：

语法
SORT <itab> [<order>] [AS TEXT]
  [BY <F1> [<order>] [AS TEXT] ... <f_n> [<order>] [AS TEXT]].
如果不使用 BY 选项，则根 据其标准关 键字对内表 <itab> 进行排序（ 参见 *内表关键字*（页 98）。要定义 不同的排序 关键字，使 用 BY 选项。系统 就根据指定 组件 <F1>...<f_n> 对数据集进 行排序。这 些字段可以 是任何类型 ，包括类型 P、I 和 F 字段，或者 表格。排序 关键字段的 数目限制在 250 以内。如果 指定多个关 键字段，则 系统首先根 据 <F1>，然后根据 <F2>， 以此类推对 记录进行排 序。系统使 用 BY 之前指定的 选项作为 BY 之后指定的 所有字段的 缺省选项。 在单个字段 之后指定的 选项覆盖选 项在 BY 之前指定的 这些字段。
如果在运行 时排序标准 仍然未知， 可以通过写 入 (<name>) 代替 <f_i> 进行动态设 置。字段 <name> 包含排序关 键字段的名 称。如果 <name> 在运行时为 空，系统就 将其忽略。 如果包含无 效的组件名 ，则发生实 时错误。对 于任何在排 序字段中使 用的字段， 用户都可指 定偏移量和 长度（参见 *为数据对象指定偏移值*（页 6－35））。
用户可以通 过在 <order> 选项中输入 DESCENDING 或 ASCENDING 来指定排序 顺序。标准 顺序是升序 。用户可以用 选项 AS TEXT 影响字符字 段的排序方 式。如果没 有 AS TEXT， 则系统二分 排序字符字 段并根据它 们的平台相 关内部编码 。如果有选 项 AS TEXT， 系统根据当 前文本环境 按字母顺序 排序字符字 段。用户可 用语句 SET LOCAL LANGUAGE 设置文本环 境，这是例 外。使用选 项 AS TEXT， 用户可免除 在排序之前 将字符字段 转换为可排 序格式之劳 （参见 *转换成可排序格式*（页 6－26））。此类转 换仅在下列 情况下才有 必要：

- 首先按字 母顺序对内 表进行排序 ，然后二分 法进行搜索 （参见 *二分法搜索*（页 321））。
- 按字母 顺序排序后 的内表次序 与按二分法 排序后的次 序不同。
- 用字母关 键字多次对 内表进行排 序。在这种 情况下效率 更佳，因为 只进行一次 转换。
- 在程序中 为数据库表 格创建字母 索引。

如果在 BY 之前指定 AS TEXT， 则选项仅影 响排序关键 字中的字符 字段。如果 在字段名之 后指定 AS TEXT， 则该字段必 须为类型 C。

如果自己指 定排序关键 字，通过使 关键字相对 短些可提高 效率。但是 ，如果排序 关键字包含 内表，则排 序进程可能 会慢很多。

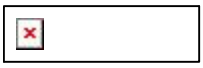
排序并不稳 定。这意味 着也许没有 必要保留排 序关键字相 同的行的旧 次序。
如果主内存 中没有足够 的空间用于 排序，系统 就将数据写 入临时外部 文件。该文 件名在 SAP 参数文件参 数 DIR_SORTTMP 中定义。

```
DATA: BEGIN OF ITAB OCCURS 10,
         LAND(3)  TYPE C,
         NAME(10) TYPE C,
         AGE      TYPE I,
         WEIGHT   TYPE P DECIMALS 2,
      END OF ITAB.

ITAB-LAND = 'USA'. ITAB-NAME   = 'Nancy'.
ITAB-AGE  = 35.    ITAB-WEIGHT = '45.00'.
APPEND ITAB.

ITAB-LAND = 'USA'. ITAB-NAME   = 'Howard'.
ITAB-AGE  = 40.    ITAB-WEIGHT = '95.00'.
APPEND ITAB.

ITAB-LAND = 'GB'.  ITAB-NAME   = 'Jenny'.
ITAB-AGE  = 18.    ITAB-WEIGHT = '50.00'.
APPEND ITAB.

ITAB-LAND = 'F'.   ITAB-NAME   = 'Michele'.
ITAB-AGE  = 30.    ITAB-WEIGHT = '60.00'.
APPEND ITAB.

ITAB-LAND = 'G'.   ITAB-NAME   = 'Karl'.
ITAB-AGE  = 60.    ITAB-WEIGHT = '75.00'.
APPEND ITAB.

SORT ITAB.

LOOP AT ITAB.
   WRITE: / ITAB-LAND, ITAB-NAME, ITAB-AGE, ITAB-WEIGHT.
ENDLOOP.

SKIP.

SORT ITAB DESCENDING BY LAND WEIGHT ASCENDING.

LOOP AT ITAB.
   WRITE: / ITAB-LAND, ITAB-NAME, ITAB-AGE, ITAB-WEIGHT.
ENDLOOP.
```

其输出为：

```
F    Michele            30         60.00
G    Karl               60         75.00
GB   Jenny              18         50.00
USA  Howard             40         95.00
USA  Nancy              35         45.00

USA  Nancy              35         45.00
USA  Howard             40         95.00
GB   Jenny              18         50.00
G    Karl               60         75.00
F    Michele            30         60.00
```

在此创建有 表头行的内 表 ITAB 并用 5 行对其进行 填充。首先 根据其标准 关键字（LAND 和 NAME） 进行排序。 然后根据定 义为 LAND 和 WEIGHT 的排序关键 字进行排序 。一般排序 顺序定义为 降序，但对 于 WEIGHT， 定义为升序 。这就是为 什么在第二 个 SORT 语句之后包 含 NAME 字段“NANCY” 的行在包含 NAME 字段“HOWARD”的行之前输 出。



```
DATA: BEGIN OF ITAB OCCURS 10,
        TEXT(6),
        XTEXT(160) TYPE X,
      END OF ITAB.

ITAB-TEXT = 'Muller'.
CONVERT TEXT ITAB-TEXT INTO SORTABLE CODE ITAB-XTEXT.
APPEND ITAB.

ITAB-TEXT = 'M_ ller'.
CONVERT TEXT ITAB-TEXT INTO SORTABLE CODE ITAB-XTEXT.
APPEND ITAB.

ITAB-TEXT = 'Moller'.
CONVERT TEXT ITAB-TEXT INTO SORTABLE CODE ITAB-XTEXT.
APPEND ITAB.

ITAB-TEXT = 'Miller'.
CONVERT TEXT ITAB-TEXT INTO SORTABLE CODE ITAB-XTEXT.
APPEND ITAB.

SORT ITAB BY TEXT.
LOOP AT ITAB.
  WRITE / ITAB-TEXT.
ENDLOOP.
SKIP.

SORT ITAB BY XTEXT.
LOOP AT ITAB.
  WRITE / ITAB-TEXT.
ENDLOOP.
SKIP.

SORT ITAB AS TEXT BY TEXT.
LOOP AT ITAB.
  WRITE / ITAB-TEXT.
ENDLOOP.
```

本示例演示 字符字段的 字母顺序排 序。内表 ITAB 包含有字符 字段的列和 有相应可按 字母排序的 二分代码的 列。二分代 码由 CONVERT 语句创建（ 参见 转换成可排序格式）。对表格 进行三次排 序。首先根 据 TEXT 字段用二分 法排序。然 后根据 XTEXT 字段用二分 法排序。最 后根据 TEXT 字段按字母 排序。如果 使用 German 文本环境，则输出为：

```
Miller
Moller
Muller
M_ ller

Miller
Moller
M_ ller
Muller
```

```
Miller
Moller
M_ ller
Muller
```

第一次排序 后，“M_ ller” 跟在“Muller” 之后，因为 字母“'_'”的内部 代码在“u” 的代码之后 。其它两个 排序是按字 母顺序进行 的。根据 XTEXT 用二分法排序的结果与 根据字段 TEXT 排序的结果 相同。

要创建次序 表，也可以 使用 APPEND 语句的 SORTED BY 选项代替 SORT 语句（参见*创建排序的列表* (页 49))。在 用内表定义数据 下可以找到 使用 SORT 对数据库表 格数据进行 排序的示例 。

创建次序表

内表适合于 生成次序表 。为此，从 空的内表开 始，然后使 用 APPEND 语句的 SORTED BY 选项，用法 如下：

语法
```
APPEND [<wa> TO] <itab> SORTED BY <f>.
```
如果使用有 SORTED BY 选项的 APPEND 语句(参见 *附加行* (页 102)),则并不 将新行附加 为内表 <itab> 的最后一行 。而是系统 插入新行， 这样内表 <itab> 就根据字段 <f> 以降序排序 。
要生成包含 多达 100 个条目的次 序表，则应 该使用 APPEND 语句。在处 理更大的表 时，由于效 率方面的原 因，建议用 SORT 语句对表格 进行排序 (参见 *排序内表* (页 43))。

如果使用 SORTED BY 选项，表格 只能包含 OCCURS 参数中指定 的行数。这 是一般规则 的一个例外 （参见*创建内表数 据类型* (页 99)) 。 如果添加的 行数比指定 的要多，则 丢弃最后一 行。这对于 创建长度有 限的次序表 十分有用(例如“Top Ten”) 。

使用 APPEND 语句的 SORTED BY 选项指定的 工作区域必 须与内表的 行类型兼容 。可转换性 对该选项不 充分。

```
DATA: BEGIN OF ITAB OCCURS 2,
        COLUMN1 TYPE I,
        COLUMN2 TYPE I,
        COLUMN3 TYPE I,
      END OF ITAB.

ITAB-COLUMN1 = 1. ITAB-COLUMN2 = 2. ITAB-COLUMN3 = 3.
APPEND ITAB SORTED BY COLUMN2.

ITAB-COLUMN1 = 4. ITAB-COLUMN2 = 5. ITAB-COLUMN3 = 6.
APPEND ITAB SORTED BY COLUMN2.

ITAB-COLUMN1 = 7. ITAB-COLUMN2 = 8. ITAB-COLUMN3 = 9.
APPEND ITAB SORTED BY COLUMN2.

LOOP AT ITAB.
  WRITE: / ITAB-COLUMN2.
ENDLOOP.
```

其输出为：

```
         8

         5
```

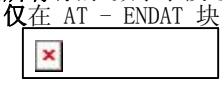
在此，用 SORTED BY 选项将三行 附加到有表 头行的内表 中。请注意 ，COLUMN2 字段内容最 小的行被丢 失，因为 OCCURS 参数中指定 的行数为 2。

循环处理

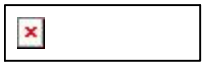
使用 LOOP 语句将行从 内表读入工 作区域（参 见*逐行读取内表* (页 36))。
下列主题描 述

求和
要在循环处 理期间计算 内表中数字 字段的内容 之和，请使 用 SUM 语句。用法 如下：

语法
```
SUM.
```
系统只能在 LOOP - ENDLOOP 块中处理该 语句。
如果在 AT - ENDAT 块中使用 SUM，则 系统计算当 前行组中**所 有**行的数字 字段之和并 将其写入工 作区域中相 应的字段（ 参见 *使用行组的控制级别* (页 51) 中的示例） 。
如果在 AT - ENDAT 块之外使用 SUM 语句（参见 *使用行组的控制级别* (页 51)），则系统 计算**每个**循 环途径中内 表**所有**行的 数字字段之 和并将其写 入工作区域 中相应的字 段（参见示 例)。
因此，应该 **仅**在 AT - ENDAT 块中使用 SUM 语句。



如果某个内 表行的组件 是另一个表 格，则不要 使用 SUM 语句。



本示例显示 SUM 语句在 AT - ENDAT 块之外如何 工作。**不**应 按以下方式 使用：

```
DATA: BEGIN OF LINE,
         COL1 TYPE I,
         COL2 TYPE I,
      END OF LINE.

DATA ITAB LIKE LINE OCCURS 10.

DO 3 TIMES.
   LINE-COL1 = SY-INDEX.
   LINE-COL2 = SY-INDEX ** 2.
   APPEND LINE TO ITAB.
ENDDO.

LOOP AT ITAB INTO LINE.
  WRITE: / LINE-COL1, LINE-COL2.
  SUM.
  WRITE: / LINE-COL1, LINE-COL2.
ENDLOOP.
```

其输出为：

```
         1          1
         6         14
         2          4
         6         14
         3          9
         6         14
```

在此创建内 表 ITAB 并用三行对 其进行填充 。在 LOOP - ENDLOOP 块中，工作 区域 LINE 的内容在 SYM 语句之前和 之后输出。 计算每个循 环途径中所 有行之和。

使用行组的 控制级别
本主题描述 如何如何在 LOOP - ENDLOOP 块中使用控 制级别语句 来创建仅处 理特定表格 行的语句块 。因此可以 使用 SUM 语句计算所 有行的子集 之和（参见 *计算总计* (页 90))。
用控制级别 语句 AT 可以打开语 句块，用控 制级别语句 ENDAT 可以关闭它 。语法如下 所示：

语法
```
AT <line>.
   <statement block>
ENDAT.
```
在其中处理 AT - ENDAT 内语句块的 行条件 <line> 可以是：

<line>	含 义
FIRST	内 表的第一行
LAST	内 表的最后一 行
NEW <f>	行组 的开头，与 字段 <f> 和 <f> 剩余字段中 的内容相同

END Of <f>	行组 的结尾，与 字段〈f〉和〈f〉剩余字段中 的内容相同

AT - ENDAT 块中的语句 块使用这些 行条件代表 预定义的控 制结构。用 户可以使用 它们处理内 表中的控制 断点，而不 必使用 *编写分支和循环* (页 7-14) 中所述的控 制语句自己 编程。
在 AT - ENDAT 语句块中， 工作区域没 有用当前表 格行进行填 充。初始化 所有不是标 准关键字部 件的字段（ 参见 *标识表格行* (页 98))。对于行 条件 FIRST 和 LAST， 系统用星号 (*) 改写所有标 准关键字段 。对于行条 件 NEW〈f〉和 END OF〈f〉，系 统用星号 (*) 改写所有出 现在工作区 域中指定字 段〈f〉右边的标准 关键字段。 用户可根据 自己的需求 在 AT - ENDAT 语句块中填 充工作区域 。

不要在 LOOP 语句的作用 受 FROM、 TO 或 WHERE 限制的循环 中使用控制 级别语句（ 参见*逐行读取内表* (页 36))，因为没 有很好地定 义交互作用

```
vDATA: BEGIN OF LINE,
         COL1 TYPE C,
         COL2 TYPE I,
         COL3 TYPE I,
       END OF LINE.

DATA ITAB LIKE LINE OCCURS 10.

LINE-COL1 = 'A'.
DO 3 TIMES.
   LINE-COL2 = SY-INDEX.
   LINE-COL3 = SY-INDEX ** 2.
   APPEND LINE TO ITAB.
ENDDO.

LINE-COL1 = 'B'.
DO 3 TIMES.
   LINE-COL2 = 2 * SY-INDEX.
   LINE-COL3 = ( 2 * SY-INDEX ) ** 2.
   APPEND LINE TO ITAB.
ENDDO.

LOOP AT ITAB INTO LINE.

WRITE: / LINE-COL1, LINE-COL2, LINE-COL3.

   AT END OF COL1.
      SUM.
      ULINE.
      WRITE: / LINE-COL1, LINE-COL2, LINE-COL3.
      SKIP.
   ENDAT.

   AT LAST.
      SUM.
      ULINE.
      WRITE: / LINE-COL1, LINE-COL2, LINE-COL3.
   ENDAT.

ENDLOOP.
```

其输出为：

```
A          1          1
A          2          4
A          3          9
_____________________________
A          6         14
B          2          4
B          4         16
B          6         36
_____________________________
B         12         56

_____________________________
```

```
*          18          70
```

在此创建内 表 ITAB并 用六行对其 进行填充。 在 LOOP - ENDLOOP 块中，为每 一循环途径 输出工作区 域 LINE。 通常在更改 COL1 内容以及系 统处于最后 一次循环时计算所有数 字字段之和 。请将本例 与 *计算总计* (页 **90**) 中的例子进 行比较。

比较内表

可以将内表 用作逻辑表 达式的操作 数（参见 编写逻辑表达式 ）：

语法
.... <itab1> <operator> <itab2> ...
对于 <operator>，可以使用 *比较所有字段类型* (页 7－4) 中的表格内 列出的所有 操作符(EQ、=、NE、<>、 ><、 GE、 >=、 LE、 <=、 GT、 >、 LT、 <)。
进行内表比 较的第一个 条件是它们 包含的行数 。内表包含 的行数越多 ，则内表就 越大。如果 两个内表行 数相同，则 逐行、逐个 组件进行比 较。如果表 格行的组件 本身就是内 表，则进行 递归比较。如果使用等 于操作符以 外的操作符 ，则系统找 到一对不相 等的组件后 就立即停止 比较并返回 该结果。
对于有表头 行的内表， 则可在表格 名之后使用 方括号 （ ） 以将表格工 作区域和表 格体区别开来。

```
DATA: BEGIN OF LINE,
         COL1 TYPE I,
         COL2 TYPE I,
      END OF LINE.

DATA: ITAB LIKE LINE OCCURS 10,
      JTAB LIKE LINE OCCURS 10.

DO 3 TIMES.
   LINE-COL1 = SY-INDEX.
   LINE-COL2 = SY-INDEX ** 2.
   APPEND LINE TO ITAB.
ENDDO.

MOVE ITAB TO JTAB.

LINE-COL1 = 10. LINE-COL2 = 20.
APPEND LINE TO ITAB.

IF ITAB GT JTAB.
   WRITE / 'ITAB GT JTAB'.
ENDIF.

APPEND LINE TO JTAB.

IF ITAB EQ JTAB.
   WRITE / 'ITAB EQ JTAB'.
ENDIF.

LINE-COL1 = 30. LINE-COL2 = 80.
APPEND LINE TO ITAB.

IF JTAB LE ITAB.
   WRITE / 'JTAB LE ITAB'.
ENDIF.

LINE-COL1 = 50. LINE-COL2 = 60.
APPEND LINE TO JTAB.

IF ITAB NE JTAB.
   WRITE / 'ITAB NE JTAB'.
ENDIF.

IF ITAB LT JTAB.
   WRITE / 'ITAB LT JTAB'.
ENDIF.
```

其输出为：

```
ITAB GT JTAB

ITAB EQ JTAB
```

JTAB LE ITAB

ITAB NE JTAB

ITAB LT JTAB

在此创建两 个内表：ITAB 和 JTAB。 用 3 行填充 ITAB 并将其复制 到 JTAB 中。然后将 另一行附加 给 ITAB 并且第一个 逻辑表达式 测试 ITAB 是否大于 JTAB。 在将同一行 附加给 JTAB 之后，第二 个逻辑表达 式测试两个 表格是否相 等。然后将 另一行附加 给 ITAB ， 第三个逻 辑表达式测 试 JTAB 是否小于或 等于 ITAB。 然后将一行 附加给 JTAB， 其此组件内 容与 ITAB 最后一行中 组件的内容 不相等。下 一逻辑表达 式测试 ITAB 是否与JTAB 不相等。从 中找到 ITAB 和 JTAB 不同之处的 第一个组件 是最后一个 表格行中的 COL1。 ITAB 是30 列而 JTAB 则是50 列。因此在 最后一个逻 辑表达式中 ， ITAB 小于 JTAB。

初始化内表

要初始化有 或没有表头 的内表，请 使用 REFRESH 语句，用法 如下：

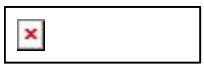
语法
REFRESH <itab>.
该语句将内 表重置为填 充它以前的 状态。这意 味着表格将 不包含任何 行。
如果使用没 有表格工作 区域的内表 ，可以使用 CLEAR 语句代替 REFRESH 语句，用法 如下：

语法
CLEAR <itab>.
如果使用有 表头行的内 表，CLEAR 语句如 *重置缺省值* (页 6 - 10) 中所述，仅 清除表格工 作区域。要重置整个内 表而不清除 表格工作区 域，使用 REFRESH 语句或 CLEAR 语句，用法 如下：

语法
CLEAR <itab>[].
内表名称之 后的方括号 指内表体。
使用 REFRESH 或 CLEAR 初始化内表 后，系统保 持在内存中 保留的空间 。可以用 FREE 语句释放内 存，用法如 下：

语法
FREE <itab>.
也可以使用 FREE 语句重置内 表并直接释 放其内存， 而不必先使 用 REFRESH 或 CLEAR。 与 REFRESH 一样，FREE 在表格体上 ，而不在表 格工作区域 上工作。
在 FREE 语句之后， 可以再次定 位内表。这 样，系统就 再次保留内 存空间。
可以使用如 下逻辑表达 式检查内表 是否为空：

语法
... <itab> IS INITIAL ...
（参见*检查内部值* (页 7 - 12) ）。



```
DATA: BEGIN OF LINE,
         COL1,
         COL2,
      END OF LINE.

DATA ITAB LIKE LINE OCCURS 10.

LINE-COL1 = 'A'. LINE-COL2 = 'B'.
APPEND LINE TO ITAB.

REFRESH ITAB.

IF ITAB IS INITIAL.
   WRITE 'ITAB is empty'.
   FREE ITAB.
ENDIF.
```

其输出为：

```
ITAB is empty.
```

在该程序中 ，先对内表 ITAB进 行填充，然 后再用 REFRESH 初始化。在 IF 语句中，用 带 IS INITIAL 参数的逻辑 表达式检查 ITAB 是否为空。 如果空着， 则释放内存 。

第九章　模块化 ABAP/4 程序

概览

内容

本章将介绍 如何模块化 ABAP/4 程序。如果 程序包含相 同或相似的 语句块，或 者希望多次 处理相同函 数，应用模块化技术可 以避免冗余 。
通过模块化 ABAP/4 程序，使得 程序易读， 并改善程序 的结构。与 未模块化的 程序相比， 模块化程序 更易于维护和更新。
模块化原则 ：

ABAP/4 提供下列模 块化技术：

源代码模块

如果模块化 源代码，程 序将更易维 护。例如， 如果希望避 免在程序中 多次重复相 同的语句， 就应使用模 块化技术。 但是，不应 采用这些方 法模块化任 务和函数。 要模块化任 务和函数， 应使用子程 序和功能模 块。
通过定义宏 ，可以在 ABAP/4 程序中创建 可调用的程 序代码模块 。
也可以在库 中创建包含 程序。这些 程序包含源 代码模块。

宏的定义与 调用

要定义包含 部分源代码 的宏，请使 用 DEFINE 语句，用法 如下：

语法
```
DEFINE <macro>.
   <statements>
END-OF-DEFINITION.
```
这就定义了 宏 <macro>。 必须在 DEFINE 和 END-OF-DEFINITION 之间指定完 整的语句。 这些语句最 多可以包含 九个占位符 （&1，&2， …，&9） 。
完成宏定义 之后，就可 以进行调用 ，方法如下 ：

语法
```
<macro> [<p1> <p2> ... <p9>].
```
在生成程序 期间，系统 用已定义的 语句替换 <macro>， 用 <p_i> 替换每个占 位符 &i。可以 从宏中调用 另一个宏，但宏不能调 用自己。

```
DATA: RESULT TYPE I,
      N1     TYPE I VALUE 5,
      N2     TYPE I VALUE 6.

DEFINE OPERATION.
   RESULT = &1 &2 &3.
   OUTPUT   &1 &2 &3 RESULT.
END-OF-DEFINITION.

DEFINE OUTPUT.
   WRITE: / 'The result of &1 &2 &3 is', &4.
END-OF-DEFINITION.

OPERATION 4 + 3 .
OPERATION 2 ** 7.
OPERATION N2 - N1.
```

输出如下：

```
The result of 4 + 3 is          7
The result of 2 ** 7 is       128
The result of N2 - N1 is        1
```

这样就定义 了两个宏： OPERATION 和 OUTPUT。 OUTPUT 嵌套在 OPERATION 中。OPERATION 先后三次被 其他参数调 用。应注意 占位符 &1、&2、 … 在宏中是如 何被替换的 。

包含程序

如果要在多 个程序中使 用相同的语 句序列，则 可一次性将 其代码编写 到包含程序 中。例如， 当希望在不 同程序中使 用的数据声 明较长时， 这一点将非 常重要。
以下主题说 明：

不能与包含 程序进行显 式数据传递 ，因为包含 程序的目的 是模块化源 代码。如果 想与模块进 行数据传递 ，请使用子 程序或功能 模块。

创建包含程 序
要创建包含 程序，请按 以下章节所 述步骤进行 ：
创建简单的 ABAP/4 程序 (页 1 - 1)
对于程序属 性类型，必 须使用类型 I。具体说 明参见：
重要的程序属性 (页 1 - 4) 通过双击 ABAP/4 程序中 INCLUDE 语句后的程 序名，还可 以创建或更 改包含程序 （参见*使用包含程 序* (页 104))。 这样既可以 创建新程序 ，又可以更 改现有程序 。
不能单独运 行包含程序 ，而必须从 其它程序中 调用。
可以从其它 包含程序中 调用包含程 序。
编写 INCLUDE 程序源代码 的唯一限制 如下：

- 包含程序 不能包含 PROGRAM 或 REPORT 语句。
- 包含程序 不能调用自 身。
- 包含程序 必须包含完 整语句。

但是，必须 确保包含程 序语句在逻 辑上符合调 用它的程序 源代码。在 编辑器中编 辑包含程序 时，选择“ 检查”通常 还无法保证 这一点。

```
***INCLUDE INCL-TST.
TEXT = 'Hello!'.
```

在此，语法 检查报告错 误，因为没 有声明 TEXT 字段。但是 ，在用合适 的类型定义 了字段 TEXT 的任何程序 中，都可以 调用程序 INCL-TST。

要想从语法 检查中获得 有意义的结 果，必须为 调用包含程 序的程序进 行语法检查 （参见 *使用包含程 序* (页 104))。

使用包含程 序
要从另一个 ABAP/4 程序中调用 包含程序， 请使用 INCLUDE 语句，用法 如下：

语法
INCLUDE <incl>.
在语法检查 期间和生成 期间，该语 句将源代码 <incl> 插入到 ABAP/4 程序中。INCLUDE 语句的功能 类似于把 <incl> 源代码复制 到调用程序 中语句的位 置上。

INCLUDE 语句必须单 独占一行， 并且不能扩 展到多行。

包含程序不 是运行时装 载的，而是 在程序生成 前就被解析 。程序生成 后，则包含 所用全部包 含程序的源 代码。

假设编写了 如下的包含 程序：

```
***INCLUDE STARTTXT.
WRITE: / 'Program started by', SY-UNAME,
       / 'on host', SY-HOST,
         'date:', SY-DATUM, 'time:', SY-UZEIT.
ULINE.
```

可以从任何 其它 ABAP/4 程序调用该 程序。例如 ，如果希望 编写输出功 能模块的标 准表头，请 在 PROGRAM 或 REPORT 之后插入 INCLUDE 语句，方法 如下：

```
PROGRAM SAPMZTST.
INCLUDE STARTTXT.

.............
```

输出结果如 下：

```
Program started by FRED

on host hs1077   date: 07/19/1995 time: 09:00:39
_____________________________________________________

.................
```

子程序

子程序是可 以从 ABAP/4 中调用的程 序。定义子 程序的目的 是为了避免 重复编写程 序中频繁使 用的某些部 分或频繁使 用的算法。 可以与子程 序进行显式 数据传递。
有两种类型 的子程序：

- 内部子程 序：
  内部子程序 的源代码与 调用程序（ 内部调用） 位于同一 ABAP/4 程序中。
- 外部子程 序：
  外部子程序 的源代码位 于 ABAP/4 程序中而不 是调用程序 （外部调用 ）中。

尽管内部子 程序主要用 于模块化及 结构化单个 程序，某个 ABAP/4 程序中内部 调用的子程 序可以从另 一个 ABAP/4 程序中外部 调用。另外 ，也可以创 建只包含子 程序的 ABAP/4 程序。这些 程序不能自 己运行，而 是被其它 ABAP/4 程序作为外 部子程序库 来调用。
下列主题说 明

定义子程序

子程序是以 FORM 开头、以 ENDFORM 结尾的代码 块。要定义 子程序，请 使用下列语 法：

语法

```
FORM <subr> [<pass>].
   <statement block>
ENDFORM.
```

<subr> 定义子程序 名。<pass> 选项用于指 定如何与子 程序进行数 据传递。

- 对于内部 子程序，不 必使用 <pass> 选项，但子 程序可以访 问主 ABAP/4 程序中声明 的所有数据 对象。
- 对于外部 子程序，必 须决定是使 用 <pass> 选项，还是 在内存公用 部分声明数 据对象。

有关与子程 序进行数据 传递的详细 信息，参见 *在调用程序 与子程序之 间进行数据 传递* (页 38)
如果在同一 程序中定义 内部子程序 ，并且不使 用事件关键 字，则应将 其分组集中 在程序末尾 ，以免影响 程序流程（ 参见*定义过程块* (页 Error! Not a valid link.))。

子程序不能 包含嵌套的 FORM-ENDFORM 块。

```
FORM HEADER.

   WRITE: / 'Program started by', SY-UNAME,
          / 'on host', SY-HOST,
            'date:', SY-DATUM, 'time:', SY-UZEIT.
   ULINE.

ENDFORM.
```

这样就创建 了子程序 HEADER， 该子程序可 用于创建输 出列表的表 头（请与*使用包含程 序* (页 129) 中 的示例进行 比较）。

调用子程序

可以调用代 码位于同一 ABAP/4 程序中的子 程序(内部 调用)，也 可调用代码 位于其它 ABAP/4 程序中的子 程序(外部 调用)。
可以运行时 指定子程序 名，或从给 定列表中调 用子程序。
不仅可以从 子程序中调 用子程序（ 嵌套调用） ，子程序也 可以调用自 己（递归调 用)。

调用内部子 程序
要调用内部 子程序，请 使用 PERFORM 语句，用法 如下：

语法

```
PERFORM <subr> [<pass>].
```

调用子程序 <subr>。 在 <pass> 选项中，指 定如何与子 程序进行数 据传递。如 果不使用 <pass> 选项，子程 序也可以访 问主 ABAP/4 程序中声明 的所有数据 类型和对象 。该数据叫 做全局数据 ，如果不被 同名局域数 据隐藏，该 数据对子程 序可见（关 于数据传递 的详细信息 ，参见*在调用程序 与子程序之 间进行数据 传递* (页 38))。

```
PROGRAM SAPMZTST.

DATA: NUM1 TYPE I,
      NUM2 TYPE I,
      SUM  TYPE I.

NUM1 = 2. NUM2 = 4.
PERFORM ADDIT.

NUM1 = 7. NUM2 = 11.
PERFORM ADDIT.

FORM ADDIT.
   SUM = NUM1 + NUM2.
   PERFORM OUT.
ENDFORM.

FORM OUT.
   WRITE: / 'Sum of', NUM1, 'and', NUM2, 'is', SUM.
ENDFORM.
```

输出如下：

```
Sum of          2 and          4 is          6

Sum of          7 and         11 is         18
```

在该示例中 ，程序末尾 定义了两个 内部子程序 ：ADDIT 和 OUT。ADDIT 从该程序中 调用，OUT 从 ADDIT 调用。子程 序自动具有 字段 NUM1、 NUM2 和 SUM 的访问权限 。

调用外部子 程序

要调用外部 子程序，请 使用 PERFORM 语句，用法 如下：

语法

```
PERFORM <subr>(<prog>) [<pass>] [IF FOUND].
```

调用程序 <prog> 中定义的子 程序 <subr>。 如果希望与 子程序进行 数据传递， 必须定义 <pass> 选项或者使 用公用部分 （关于进行 数据传递的 详细信息， 参见*在调用程序 与子程序之 间进行数据 传递* (页 38))。
如果使用 IF FOUND 选项，并且 程序 <prog> 中没有子程 序 <sub>， 系统就忽略 PERFORM 语句。
启动调用外 部子程序的 程序时，如 果定义了子 程序的程序 不在内存中 ，ABAP/4 就 将其装载到 内存中。为 了节省存储 空间，尽量 将不同程序 中定义的子 程序数目限 制到最小。

假定程序包 含下列子程 序：

```
PROGRAM FORMPOOL.

FORM HEADER.
   WRITE: / 'Program started by', SY-UNAME,
          / 'on host', SY-HOST,
            'date:', SY-DATUM, 'time:', SY-UZEIT.
   ULINE.
ENDFORM.
```

可以从程序 中调用子程 序，方法如 下：

```
PROGRAM SAPMZTST.

PERFORM HEADER(FORMPOOL) IF FOUND.
```

在该示例中 ，调用程序 与子程序之 间没有数据 传递。与 *使用包含程 序* (页 129) 中的示例一 样，该子程 序执行输出 语句。

运行时指定 子程序名

运行时，可 以指定要调 用的子程序 名以及存储 子程序的程 序名。为此 请使用 PERFORM 语句，用法 如下：

语法

```
PERFORM (<fsubr>) [IN PROGRAM (<fprog>)] [<pass>] [IF FOUND].
```

系统执行字 段 <fsubr> 中存储的子 程序。如果 使用 IN PROGRAM 选项，系统 在字段 <fprog> 中存储的程 序内查找子 程序（外部 调用）。否 则，系统在 当前程序中 查找子程序 （内部调用 ）。
使用该语句 还可在程序 代码中指定 子程序名和 外部程序名 。为此请忽 略括弧。
<pass> 选项指定如 何与子程序 进行数据传 递（关于数 据传递的详 细信息，参 见 *在调用程序 与子程序之 间进行数据 传递* (页 38))。如果使 用 IF FOUND 选项，找不 到子程序 <sub>时 ，系统就忽 略 PERFORM 语句。

假定程序包 含子程序：

```
PROGRAM FORMPOOL.

FORM SUB1.
   WRITE: / 'Subroutine 1'.
ENDFORM.

FORM SUB2.
   WRITE: / 'Subroutine 2'.
ENDFORM.
```

可在运行时 指定子程序 名，如下所 示：

```
PROGRAM SAPMZTST.

DATA: PROGNAME(8) VALUE 'FORMPOOL',
      SUBRNAME(8).

SUBRNAME = 'SUB1'.
PERFORM (SUBRNAME) IN PROGRAM (PROGNAME) IF FOUND.

SUBRNAME = 'SUB2'.
PERFORM (SUBRNAME) IN PROGRAM (PROGNAME) IF FOUND.
```

输出如下：

```
Subroutine 1

Subroutine 2
```

字符字段 PROGNAME 包含程序名 ，该程序包 含子程序。 子程序名被 分配给字符 字段 SUBRNAME。

从列表中调 用子程序
要从列表中 调用特定子 程序，请使 用 PERFORM 语句，用法 如下：

语法
PERFORM <idx> OF <form$_1$> <form$_2$> .... <form$_n$>.
系统执行子 程序列表中 位于 <idx> 处的子程序 。PERFORM 语句的这一 变量只对内 部调用有效 。字段 <idx> 既可以是变 量也可以是 文字。

```
PROGRAM SAPMZTST.

DO 2 TIMES.
  PERFORM SY-INDEX OF SUB1 SUB2.
ENDDO.

FORM SUB1.
   WRITE / 'Subroutine 1'.
ENDFORM.

FORM SUB2.
   WRITE / 'Subroutine 2'.
ENDFORM.
```

输出如下：

```
Subroutine 1

Subroutine 2
```

在该示例中 ，从列表中 连续调用两 个内部子程 序：SUB1 和 SUB2。

在调用程序 与子程序之 间进行数据 传递

就内部和外 部子程序而 言，处理数 据的缺省方 法有所不同 。

- 对于内部 子程序，如 果调用程序 的全局数据 未被同名局 部数据隐藏 ，可直接从 子程序进行 访问。该全 局数据由内 部数据对象 和类型以及 ABAP/4 词典字段组 成。ABAP/4 词典字段由 程序中的 TABLES 引用。（关于该语句的 详细信息， 参见*BC ABAP/4 用户指南*（页 **Error! Not a valid link.**））。
- 如果希望 采用与内部 子程序相同 的方式，从 外部子程序 访问调用程 序数据，就 必须在调用 程序和包含外部子程序 的程序中， 将数据声明 为公用部分 （参见 *将数据声明 为公共部分*（页 43））。

在子程序中 使用全局数 据时，可以 将全局数据 复制到局部 数据堆栈上 。这一点对 内部子程序 中的全局数 据和声明为 公共部分的 数据有效。为此，必须 使用字段符 号（关于该 主题的详细 信息，参见*分配全局字段的本地副本*（页 10 - 16））。

应该只对简 单模块化程 序使用公共 部分。尤其 在嵌套子程 序调用中使 用功能模块 时(关于功 能模块的详 细信息，参 见 *功能模块* (页 90))，访问公 共部分的规 则会很复杂 。

为了使程序 更透明，可 移植性更好 ，在调用程 序和子程序 之间进行数 据传递时， 应尽量选择 下列方法：
在内部和外 部子程序中 ，明确指定 所需的及可 能更改的数 据。为此， 在定义和调 用子程序期 间，可以使 用参数。这 些参数是在 FORM （参见 *定义子程序* (页 130) ）和 PERFORM （参见 *调用子程序* (页 130) ）语句的 <pass> 选项中定义 的。
下列主题说 明

将数据声明 为公共部分
要将数据声 明为公共部 分，请使用 DATA 语句，用法 如下：

语法

```
DATA: BEGIN OF COMMON PART [<name>],
               <data declaration>,
               ..............
      END OF COMMON PART [<name>].
```

在 <data declaration> 中，按 *DATA 语句* (页 3 - 14) 所述，对要 包含到公共 部分中的所 有数据加以 声明。

子程序和调 用程序自动 共享 TABLES 语句定义的 表格工作区 。

要在调用程 序和子程序 中使用公共 部分，必须 在所涉及的 全部程序中 都使用完全 相同的声明 。因此，应 将公共部分 声明放到 INCLUDE 程序中（参 见*包含程序* (页 129))。
一个程序中 可使用多个 公共部分。 但必须给每 个公共部分 分配一个名 称 <name>。 如果每个程 序中只使用 一个公共部 分，则名称 <name> 可选。

为了避免在 具有不同公 共部分声明 的程序之间 发生冲突， 公共部分的 名称应始终 保持唯一。

假定某个 INCLUDE 程序 INCOMMON 包含公共部 分 NUMBERS 的声明。公 共部分由三 个数字字段 组成： NUM1、 NUM2 以及 SUM：

```
***INCLUDE INCOMMON.

DATA: BEGIN OF COMMON PART NUMBERS,
         NUM1 TYPE I,
         NUM2 TYPE I,
         SUM  TYPE I,
      END OF COMMON PART NUMBERS.
```

假定程序 FORMPOOL 包含 INCOMMON， 同时包含子 程序 ADDIT 和 OUT：

```
PROGRAM FORMPOOL.

INCLUDE INCOMMON.

FORM ADDIT.
   SUM = NUM1 + NUM2.
   PERFORM OUT.
ENDFORM.

FORM OUT.
   WRITE: / 'Sum of', NUM1, 'and', NUM2, 'is', SUM.
ENDFORM.
```

假定调用程 序 SAPMZTST 时包含 INCOMMON， 并从程序 FORMPOOL 中调用 ADDIT。

```
PROGRAM SAPMZTST.

INCLUDE INCOMMON.

NUM1 = 2. NUM2 = 4.
PERFORM ADDIT(FORMPOOL).

NUM1 = 7. NUM2 = 11.
PERFORM ADDIT(FORMPOOL).
```

启动 SAPMZTST 后，输出如 下：

```
Sum of          2 and          4 is          6

Sum of          7 and         11 is         18
```

该示例与 *调用内部子 程序* (页 130) 中的示例功 能相同。在 当前示例中 ，必须将程 序所用数据对象声明为 公共部分， 因为子程序 ADDIT 和 OUT 都被外部调 用。

通过参数进 行数据传递
可以用参数 在调用程序 和子程序之 间进行数据 传递。
- 在定义子 程序期间用 FORM 语句定义的 参数叫做**形 式参数**。
- 在调用子 程序期间用 PERFORM 语句指定的 参数叫做**实 参数**。

可以区分不 同种类的参 数：
- 输入参数 用于向子程 序传递参数
- 输出参数 用于从子程 序传递参数
- 输入/输 出参数用于 与子程序进 行参数传递

在 FORM 和 PERFORM 语句的〈pass〉选项中定义 参数，方法 如下：

语法
```
FORM <subr> [TABLES <formal  table list>]
            [USING <formal  input list>]
         [CHANGING <formal output list>]....
PERFORM <subr>[(<prog>)] [TABLES <actual  table list>]
                         [USING <actual  input list>]
                      [CHANGING <actual output list>]....
```
选项 TABLES、 USING 和 CHANGING 必须按上述 顺序编写。
列表中 USING 和 CHANGING 后面的参数 可以是所有 类型的数据 对象（参见 *声明数据* (页 3－1)）和字段符 号（参见 *使用字畏_ 号* (页 10－1)）。列表中 TABLES 后面的参数 既可以是有 表头行的内 表，也可以 是不带表头 行的内表。 可以采用 TABLES、 USING 或 CHANGING 传送内表。

> 可以用**变量** 偏移量和长 度规范指定 实参数（关 于偏移量规 范的详细信 息，参见 *指定数据对 象的偏移量* (页 6－35)）。对于形 式参数，不 能采取这种 方法。要参 考形式参数 的一部分， 请将该部分 分配给字段 符号，然后 再传递该字 段符号（参 见*使用字畏_ 号* (页 10－1)）。
>
> 实参数的偏 移量规范功 能与字段符 号的偏移量 规范相同（ 参见 *具有偏移量 说明的静_ ASSIGN* (页 10－7)）。可以选 择内存区， 该内存区位 于指定的实 参数边界之 外。

对于 FORM 语句中 USING 和 CHANGING 后 面列表中的 每个形式参 数，可以指 定不同数据 传递方式：
- **通过参考 值调用：**在 子程序调用 期间，仅将 实参数的地 址传送给形 式参数。形 式参数本身 没有内存。 在子程序中 使用调用程 序的字段。 更改形式参 数之后，调 用程序中的 字段内容也 会变动。
- **通过值调 用：**在子程 序调用期间 ，形式参数 是作为实参 数的副本创 建的。形式 参数有自己 的内存。更 改形式参数 并不影响实 参数。
- **通过值和 结果调用：** 在子程序调 用期间，形 式参数是作 为实参数的 副本创建的 。形式参数 有自己的内 存空间。在 子程序末尾 ，将对形式 参数的更改 复制给实参 数。

由 TABLES 传递的内表 均通过参考 值调用。
关于如何指 定形式参数 数据类型的 信息，参见
指定形式参 数的数据类 型对于向子 程序传递结 构化数据非 常重要。有 关如何传递 结构化数据 的内容（字 段串和内表 ）在以下章 节中有专门 解释：

通过参考传 递
要通过参考 值在调用程 序和子程序 之间进行数 据传递，请 使用 FORM 和 PERFORM 语句〈pass〉选项的 USING 或 CHANGING， 用法如下：

语法
```
FORM ..... [USING <fi1> ... <fin>] [CHANGING <fo1> ... <fon>] ...
PERFORM.. [USING <ai1> ... <ain>] [CHANGING <ao1> ... <aon>] ...
```
在 USING 和 CHANGING 后面的列表 中指定形式 参数和实参 数，而不需 附加任何内 容。
FORM 语句中形式 参数的名称 可以与 PERFORM 语句中实参 数〈ai_1〉... 〈ai_n〉 and 〈ao_1〉 ... 〈ao_n〉 的名称不同 。
PERFORM 语句列表中 的第一个参 数传递给 FORM 语句相应列 表中的第一 个参数，以 此类推。
对于通过参 考值调用，USING 和 CHANGING 完全相同。对于文档，USING 用于子程序 中的固定输 入参数，而 CHANGING 则用于子程 序中变化的 输出参数。

> 子程序中变 化的输入参 数在调用程 序中也会变 化。要避免 这一点，必 须通过值传 递参数。
>
> 假定程序 FORMPOOL 包含两个子 程序 ADDIT 和 OUT：
>
> ```
> PROGRAM FORMPOOL.
>
> FORM ADDIT USING ADD_NUM1 ADD_NUM2 CHANGING ADD_SUM.
>    ADD_SUM = ADD_NUM1 + ADD_NUM2.
>    PERFORM OUT USING ADD_NUM1 ADD_NUM2 ADD_SUM.
> ENDFORM.
>
> FORM OUT USING OUT_NUM1 OUT_NUM2 OUT_SUM.
> ...WRITE: / 'Sum of',OUT_NUM1,'and',OUT_NUM2,'is',OUT_SUM.
> ENDFORM.
> ```
>
> 假定某个调 用程序调用 ADDIT 和 OUT：
>
> ```
> PROGRAM SAPMZTST.
>
> DATA: NUM1 TYPE I,
>       NUM2 TYPE I,
>       SUM  TYPE I.
> ```

```
NUM1 = 2. NUM2 = 4.
PERFORM ADDIT(FORMPOOL) USING NUM1 NUM2 CHANGING SUM.

NUM1 = 7. NUM2 = 11.
PERFORM ADDIT(FORMPOOL) USING NUM1 NUM2 CHANGING SUM.
```

启动 SAPMZTST 后，输出如 下：

```
Sum of          2 and          4 is          6

Sum of          7 and         11 is         18
```

该示例与 *将数据声明 为公共部分* (页 133) 中的示例功 能相同。在 当前示例中 ，通过参考 值将实参数 NUM1、 NUM2 和 SUM 从 SAPMZTST 传递给子程 序 ADDIT 的形式参数 。更改 ADD_SUM 后，后面的 参数就传递 给子程序 OUT 的形式参数 OUT_NUM1、 OUT_NUM2 和 OUT_SUM。

通过值传递

要确保输入 参数在调用 程序中保持 不变（即使 子程序中已 被更改)， 可以通过值 将数据传递 给子程序。 为此，请使 用 FORM 和 PERFORM 语句 <pass> 选项的 USING， 用法如下：

语法

```
FORM ..... USING ...VALUE(<fi>) ..
PERFORM... USING .......<ai> ..
```

通过对 FORM 语句中 USING (关于 USING 的详细信息 ，参见 *通过参考传 递* (页 134)) 后面列表 中的形式输 入参数写入 VALUE(<f_i>)而 不是 <f_i>， 相应参数就 通过值进行 传递。与实 字段 <a_i> 属性相同的 PERFORM 语句调用该 子程序时， 就会创建一 个新的局域 字段 <f_i>(关于局域字 段的信息，参见 *在子程序中 定义局部数 据类型和对 象* (页 35))。系统处 理独立于调 用程序中参 考字段的该字段。

假定 FORMPOOL 程序包含子 程序 FACT：

```
PROGRAM FORMPOOL.

FORM FACT USING VALUE(F_NUM) CHANGING F_FACT.
   F_FACT = 1.
   WHILE F_NUM GE 1.
      F_FACT = F_FACT * F_NUM.
      F_NUM = F_NUM - 1.
   ENDWHILE.
ENDFORM.
```

假定程序 SAPMZTST 调用子程序 FACT：

```
PROGRAM SAPMZTST.

DATA: NUM  TYPE I VALUE 5,
      FAC  TYPE I VALUE 0.

PERFORM FACT(FORMPOOL) USING NUM CHANGING FAC.

WRITE: / 'Factorial of', NUM, 'is', FAC.
```

在启动 SAPMZTST 后，输出如 下：

```
Factorial of          5 is        120
```

在该示例中 ，计算数字 NUM 的阶乘。输 入参数 NUM 被传递给子 程序的形式 参数 F_NUM。 尽管 F_NUM 在子程序中 已被更改， 实参数 NUM 仍然保持其 初始值。输 出参数 FAC 通过参考值 进行传递。

通过值和结 果进行传递

如果仅希望 在子程序运 行成功之后 ，才将更改 过的输出参 数从子程序 返回给调用 程序，请使 用 FORM 和 PERFORM 语句 <pass> 选项的 CHANGING， 用法如下：

语句

```
FORM ..... CHANGING ...VALUE(<fi>) ..
PERFORM... CHANGING .......<ai> ..
```

通过对 FORM 语句中 CHANGING (参见 *通过参考传 递* (页 134)) 后面列表 中的形式输 入参数写入 VALUE(<f_i>)而不是 <f_i>， 相应参数通 过值和结果 调用进行传 递。与实字 段 <a_i> 属性相同的 PERFORM 语句调用该 子程序时， 就会创建一 个新的局域 字段 <f_i>(关于局域字 段的信息， 参见 *在子程序中 定义局部数 据类型和对 象* (页 35))。系统处 理独立于调 用程序中参 考字段的字 段。

仅在运行到 ENDFORM 语句时，系 统才将 <fi_i> 的当前值传 递给 <ai_i>。 如果子程序 因为某个对 话信息（关 于信息的详 细资料，参 见 *处理错误和 消息* （页 Error! Not a valid link.））而中断， 则实参数 <ai_i> 保持不变。

仅在编写对 话程序（参 见*编写 ABAP/4 事务* （页 Error! Not a valid link.））时，通 过对话信息 终止子程序 才有意义。 在报表程序 中，子程序 中的对话信 息终止整个 程序。在时 间事件 AT SELECTION SCREEN（ 参见 *通过事件控制 ABAP/4 程序流* （页 Error! Not a valid link.））期间，或 交互报表（ 参见 *列表中的消 息* （页 Error! Not a valid link.））期间，该 规则存在例 外。交互报 表的选项屏 幕和工具使 用现存对话 程序。该对 话程序是作 为 ABAP/4 中的关键字 提供的。

在用 PERFORM 调用子程序 时，可以用 USING 代替 CHANGING。 但对于文档 ，则应使用 FORM 语句中相同 的字。

```
PROGRAM SAPMZTST.

DATA: OP1  TYPE I,
      OP2  TYPE I,
      RES  TYPE I.

OP1 = 3.
OP2 = 4.

PERFORM MULTIP USING OP1 OP2 CHANGING RES.

WRITE: / 'After subroutine:',
       / 'RES=' UNDER 'RES=', RES.

FORM MULTIP USING VALUE(O1) VALUE(O2) CHANGING VALUE(R).
    R = O1 * O2.
    WRITE: / 'Inside subroutine:',
           /  'R=', R, 'RES=', RES.
ENDFORM.
```

在启动 SAPMZTST 后，输出如 下：

```
Inside subroutine:
R=          12 RES=           0
After subroutine:
               RES=          12
```

在该示例中 ，从调用程 序调用内部 子程序 MULTIP。 参数 OP1 和 OP2 通过值传递 给形式参数 O1 和 O2。输出 参数 RES 通过值和结 果传递给形 式参数 R。通过将 R 和 RES 从子程序内 部写到屏幕 上，证明 RES 在 ENDFORM 语句之前未 被更改。从 子程序中返 回之后，其 内容已更改 。

键入形式参 数

要确保子程 序的形式参 数属于某个 类型，可在 FORM 语句中指定 该类型。为 此，请在 TABLES、 USING 或 CHANGING 后面列表中 的形式参数 后输入 TYPE <t> 或 LIKE <f>（参 见 *DATA 语句的基本 格式* （页 3 - 14））。该类型 规范可选。

用 PERFORM 调用子程序 时，系统检 查 PERFORM 语句中实参 数的类型是 否与分配给 形式参数的 类型兼容。 下表给出了 兼容规则。 不存在类型 转换。如果 类型不兼容 ，系统会在 内部子程序 调用的语法 检查期间输 出错误信息 ，或在外部 子程序调用 时出现运行 错误。

以下兼容规 则应用于确 定形式参数 类型：

确定类型	实 参数的语法 检查
无类型 规范 TYPE ANY	系统 接受任意类 型的实参数 。将实参数 的所有属性 传递给形式 参
TYPE TABLE	系 统检查实参 数是否为内 表。将表格 的所有属性 和结构从实 参数参数。
TYPE C、 N、 P 或 X	系统检查 实参数的类 型是否为 C、 N、 P 或 X。将参数 的长度和其范（对类 型 P）从实参 数传递给形 式参数。
TYPE D、F、 I或T, LIKE <f>, TYPE <ud>	完 全确定这些 类型。系统 检查实参数 的数据类型 是否完全与 形式相兼容 。

（<ud> 是用户定义 的）

```
REPORT SAPMZTST.

DATA:
  DATE1      TYPE D,              DATE2      TYPE T,
  STRING1(6) TYPE C,              STRING2(8) TYPE C,
```

```
  NUMBER1    TYPE P DECIMALS 2,  NUMBER2    TYPE P,
  COUNT1     TYPE I,             COUNT2     TYPE I.

PERFORM TYPETEST USING DATE1 STRING1 NUMBER1 COUNT1.
SKIP.
PERFORM TYPETEST USING DATE2 STRING2 NUMBER2 COUNT2.

FORM TYPETEST USING NOW
                    TXT TYPE C
                    VALUE(NUM) TYPE P
                    INT TYPE I.
   DATA: T.
   DESCRIBE FIELD NOW TYPE T.
   WRITE: / 'Type of NOW is', T.
   DESCRIBE FIELD TXT LENGTH T.
   WRITE: / 'Length of TXT is', T.
   DESCRIBE FIELD NUM DECIMALS T.
   WRITE: / 'Decimals of NUM are', T.
   DESCRIBE FIELD INT TYPE T.
   WRITE: / 'Type of INT is', T.
ENDFORM.
```

在 SAPMZTST 后，输出如 下：

```
TYPE of NOW is D
Length of TXT is 6
Decimals of NUM are 2
Type of INT is I

TYPE of NOW is T
Length of TXT is 8
Decimals of NUM are 0
Type of INT is I
```

采用不同的 实参数，两 次调用内部 子程序 TYPETEST 。所有实参 数和形式参 数均兼容， 在语法检查 中没有出现 错误信息。 例如，如果 在调用程序 中将 COUNT2 声明为 TYPE F 而不是 TYPE I ，则在语法 检查中会报 告错误，因 为形式参数 INT 是用 TYPE I 指定的。请 注意，类型 规范不同，每次调用子 程序时，形 式参数的类 型和属性也 可以有所不 同。

详细信息， 请参见 FORM 的关键字文 档。

向子程序传 递字段串

如果要向子 程序传递字 段串，并访 问子程序中 字段串的组 件，则必须 指定相应形 式参数的类 型（参见 *键入形式参 数* (页 136) ）。这里使 用的数据类 型必须与字 段串类型相 同。

对于内部子 程序，可以 使用 TYPE 或 LIKE 参考要直接 传递的字段 串的结构。 对于外部子 程序，还必 须在包含子程序的程序 中指定结构 定义。为此 ，可以使用 下列任何一 种：

- 包含程序
- 类型组
- ABAP/4 词典结构

下例就上述 内容加以说 明：

包含程序

可以在包含 程序中定义 结构（参见 *包含程序* (页 129) ）。该方法 适用于这种 结构：只用 于几个子程 序中，并且 仅由几个开 发者使用。

包含程序 DECLARE：

```
***INCLUDE DECLARE.

TYPES: BEGIN OF LINE,
          NAME(10)   TYPE C,
          AGE(2)     TYPE N,
          COUNTRY(3) TYPE C,
       END OF LINE.
```

程序 FORMPOOL 包含程序 DECLARE 和 COMPONENTS 子程序：

```
PROGRAM FORMPOOL.

INCLUDE DECLARE.

FORM COMPONENTS CHANGING VALUE(PERSON) TYPE LINE.
   WRITE: / PERSON-NAME, PERSON-AGE, PERSON-COUNTRY.
   PERSON-NAME = 'Mickey'.
   PERSON-AGE = '60',
   PERSON-COUNTRY = 'USA'.
ENDFORM.
```

程序 SAPMZTST 包含程序 DECLARE， 并调用子程 序 COMPONENTS：

```
REPORT SAPMZTST.
INCLUDE DECLARE.
DATA WHO TYPE LINE.
WHO-NAME = 'Karl'. WHO-AGE = '10'. WHO-COUNTRY = 'D'.
PERFORM COMPONENTS(FORMPOOL) CHANGING WHO.
WRITE: / WHO-NAME, WHO-AGE, WHO-COUNTRY.
```

SAPMZTST 随后输出：

```
KARL       10 D
MICKEY     60 USA
```

实参数 WHO （具有用户 定义的、结 构化数据类 型 LINE） 传递给形式 参数 PERSON。 形式参数 PERSON 的类型由 TYPE LINE 确定。因为 LINE 是由用户定 义的数据类 型，因此 PERSON 的类型就被 完全指定。 子程序访问 并更改 PERSON 的组件。然 后返回给调 用程序中 WHO 的组件。

类型组
可在类型组 中定义结构 （参见 *使用类型组* （页 3－27））。该方法 适用于几个 开发者共同 工作的大型 程序所用的结构中。

DECLA 类型组：

```
TYPE-POOL DECLA .
TYPES: BEGIN OF DECLA_LINE,
          NAME(10)    TYPE C,
          AGE(2)      TYPE N,
          COUNTRY(3)  TYPE C,
       END OF DECLA_LINE.
```

程序 FORMPOOL 使用 DECLA 类型组 ，并包含子 程序 COMPONENTS 。

```
PROGRAM FORMPOOL.
TYPE-POOLS DECLA.
FORM COMPONENTS CHANGING VALUE(PERSON) TYPE DECLA_LINE.
   WRITE: / PERSON-NAME, PERSON-AGE, PERSON-COUNTRY.
   PERSON-NAME = 'Mickey'.
   PERSON-AGE = '60'.
   PERSON-COUNTRY = 'USA'.
ENDFORM.
```

程序 SAPMZTST 使用 DECLA 类型组，并 调用子程序 COMPONENTS ：

```
REPORT SAPMZTST.
TYPE-POOLS DECLA.
DATA WHO TYPE DECLA_LINE.
WHO-NAME = 'Karl'. WHO-AGE = '10'. WHO-COUNTRY = 'D'.
PERFORM COMPONENTS(FORMPOOL) CHANGING WHO.
WRITE: / WHO-NAME, WHO-AGE, WHO-COUNTRY.
```

SAPMZTST 随后输出：

```
KARL       10 D
MICKEY     60 USA
```

除了类型定 义不在包含 程序中出现 ，而在 DECLA 类型组中出 现之外，该 示例与上例 完全一样。

ABAP/4 词典结构
可以使用 ABAP/4 词典中的表 格结构。该 方法始终可 选，因为总 能访问 ABAP/4 词典结构。

程序 FORMPOOL 包含子程序 COMPONENTS ：

```
PROGRAM FORMPOOL.

FORM COMPONENTS CHANGING VALUE(CITIES) LIKE SPFLI.
   WRITE: / CITIES-CITYFROM, CITIES-CITYTO.
   CITIES-CITYFROM = 'New York'.
   CITIES-CITYTO   = 'San Francisco'.
ENDFORM.
```

程序 SAPMZTST 调用子程序 COMPONENTS：

```
REPORT SAPMZTST.

DATA FLIGHT LIKE SPFLI.

FLIGHT-CITYFROM = 'Berlin'. FLIGHT-CITYTO = 'London'.

PERFORM COMPONENTS(FORMPOOL) CHANGING FLIGHT.

WRITE: / FLIGHT-CITYFROM, FLIGHT-CITYTO.
```

SAPMZTST 随后输出：

```
Berlin                 London
New York               San Francisco
```

在子程序 COMPONENTS 中，用 LIKE SPFLI 确定形式参 数 CITIES 的类型。在 ABAP/4 词典中，CITIES 结构与表格 SPFLI 相同。在程 序 SAPMZTST 中，使用相 同结构对字 段串 FLIGHT 加以说明， 然后填充和 传递给子程 序 COMPONENTS 的结构 CITIES 组件。该子 程序处理组 件并将其传 递给调用程序。

向子程序传 递内表

用USING 和 CHANGING 传递
在 FORM 和 PERFORM 语句中，可 以将内表当 作 USING 或 CHANGING 后面列表中 的参数传递 。如果要访 问内表组件 ，则必须指 定相应形式 参数的类型 （参见 *键入形式参 数*（页 136））。否则， 只能在子程 序中执行行 操作。
同时还必须 区分有表头 行或无表头 行的内表。 对于有表头 行的内表， 必须在表格 名之后用方 括号（[]） 指定表格体 ，以便与表 头行区分开 （参见 *访问内表*（页 8－4））。
对于内部子 程序，可以 用 TYPE 或 LIKE 参考要直接 传递的内表 。对于外部 子程序，还 必须在包含 子程序的程 序中定义结 构。为此， 请按照定义 字段串结构 的方法进行 （参见 *向子程序传 递字段串*（页 137））。

```
PROGRAM SAPMZTST.

DATA: BEGIN OF LINE,
         COL1 TYPE I,
         COL2 TYPE I,
      END OF LINE.

DATA ITAB LIKE LINE OCCURS 10.

PERFORM FILL CHANGING ITAB.
PERFORM OUT  USING    ITAB.

FORM FILL CHANGING F_ITAB LIKE ITAB.
   DATA F_LINE LIKE LINE OF F_ITAB.
   DO 3 TIMES.
      F_LINE-COL1 = SY-INDEX.
      F_LINE-COL2 = SY-INDEX ** 2.
      APPEND F_LINE TO F_ITAB.
   ENDDO.
ENDFORM.

FORM OUT USING VALUE(F_ITAB) LIKE ITAB.
   DATA F_LINE LIKE LINE OF F_ITAB.
   LOOP AT F_ITAB INTO F_LINE.
      WRITE: / F_LINE-COL1, F_LINE-COL2.
   ENDLOOP.
ENDFORM.
```

启动 SAPMZTST 后，输出如 下：

```
         1          1
         2          4
         3          9
```

在该示例中 ，定义并调 用了两个子 程序： FILL 和 OUT。无 表头行的内 表 ITAB 传递给子程 序。通过参 考值传递给 FILL， 通过值传递 给 OUT。请 注意，在两 个子程序中 都将工作域 F_LINE 创建为局部 数据对象（ 有关局部字 段的信息， 参见 *在子程序中 定义局部数 据类型和对 象* (页 35) ）。如果 ITAB 为有表头行 的表格，则 在 PERFORM 和 FORM 中必须用 ITAB[] 代替 ITAB。

用 TABLES 传递
在 FORM 和 PERFORM 语句中，可 以将所有内 表当作 TABLES 后面列表中 的参数进行 传递。如果 要在子程序 中访问表格 行的组件， 则必须指定 形式参数的 数据类型（ 参见 *向子程序传 递字段串* (页 137))。否则， 只能执行整 行操作。由 TABLES 传递的内表 总是通过参 考值调用。
如果传递有 表头行的内 表，则将表 格体和表格 工作域传递 给子程序。 如果传递无 表头行的内 表，则在子 程序中自动 创建作为局 部数据对象 的表头行（ 关于局部字 段的详细信 息，参见 *在子程序中 定义局部数 据类型和对 象* (页 35))。

```
PROGRAM SAPMZTST.

TYPES: BEGIN OF LINE,
         COL1 TYPE I,
         COL2 TYPE I,
       END OF LINE.

DATA: ITAB TYPE LINE OCCURS 10 WITH HEADER LINE,
      JTAB TYPE LINE OCCURS 10.

PERFORM FILL TABLES ITAB.
MOVE ITAB[] TO JTAB.
PERFORM OUT  TABLES JTAB.

FORM FILL TABLES F_ITAB LIKE ITAB[].
   DO 3 TIMES.
      F_ITAB-COL1 = SY-INDEX.
      F_ITAB-COL2 = SY-INDEX ** 2.
      APPEND F_ITAB.
   ENDDO.
ENDFORM.

FORM OUT TABLES F_ITAB LIKE JTAB.
   LOOP AT F_ITAB.
      WRITE: / F_ITAB-COL1, F_ITAB-COL2.
   ENDLOOP.
ENDFORM.
```

启动 SAPMZTST 后，输出如 下：

```
         1          1

         2          4

         3          9
```

在该示例中 ，内表 ITAB 有表头行， 而内表 JTAB 无表头行。 ITAB 被传递给子 程序 FILL， 从中使用表 格工作域 F_ITAB 进行填充。 在调用程序 中，将 ITAB 的表格体复 制到 JTAB 后，JTAB 被传递给子 程序 OUT。请 注意，在这 种情况下，将无表头行 的实表格传 递给有表头 行的形式表 格，并在子 程序中使用 了表格工作 域 F_ITAB。

在子程序中 定义局部数 据类型和对 象

局部数据类 型和对象仅 在声明它们 的过程中出 现。对于子 程序，可以 区分
- 仅在执行 子程序时才 存在的动态 数据类型和 对象
- 存在于子 程序之外的 静态数据对 象。在下次 调用同一子 程序之前， 该数据对象 一直调用和 保留其值。
- 显式定义 的局部数据 对象，可以 用于保留全 局数据对象 值

特殊种类的 局部数据是 局部数据堆 栈上的全局 数据副本。 使用字段符 号定义和访 问（关于该 主题的详细 信息，参见*分配全局字段的局部副 本* (页 10 - 16))。

定义动态局 部数据类型 和对象
可以用 TYPES 和 DATA 语句，按照 *创建数据对象和数据类型* (页 3 - 12) 中的说明， 在子程序内 创建局部数 据类型和数 据对象。为 每个子程序 调用新近定 义的这些类 型和对象将 在退出子程 序时删除。
每个子程序 均有自己的 局域命名空 间。如果用 与全局数据 类型或对象 相同的名称 定义局部数 据类型或对 象，则不能 在子程序中 访问全局数 据类型或对 象。局部数 据类型或数 据对象会隐 藏同名全局 数据类型或 对象。这意 味着，如果 在子程序中 使用某个数 据类型或对 象名，则总 定址局部声 明的对象（ 如果存在） ，否则，定 址全局声明 的对象。

```
PROGRAM SAPMZTST.

TYPES WORD(10) TYPE C.
DATA  TEXT TYPE WORD.

TEXT = '1234567890'. WRITE / TEXT.

PERFORM DATATEST.

WRITE / TEXT.

FORM DATATEST.
   TYPES WORD(5) TYPE C.
   DATA  TEXT TYPE WORD.
   TEXT = 'ABCDEFGHJK'. WRITE / TEXT.
ENDFORM.
```

启动 SAPMZTST 后，输出如 下：

```
1234567890

ABCDE

1234567890
```

该示例中， 类型为 WORD 的数据类型 WORD 以及数据对 象 TEXT 都在调用程 序中给出全 局声明。在 给 TEXT 分配值并将 其写到屏幕 上之后，调 用内部子程 序 DATATEST。 在子程序内 部，类型为 WORD 的数据类型 WORD 以及数据对 象 TEXT 都给出局部 声明。它们 隐藏全局类 型和对象。 仅在退出子 程序后，全 局定义才恢 复有效。

如果要避免 隐藏全局数 据类型和对 象，则必须 给局部类型 和对象分配 不同的名称 。例如，所 有局部名称 都以前缀 ‘ F_’ 开头 。

定义静态局 部数据对象

如果想在退 出子程序后 保留局部数 据对象之值 ，必须使用 STATICS 语句而非 DATA 语句进行声 明(参见 *STATICS 语句* (页 3－20))。使用 STATICS 声明已全局 定义、但仅 在定义它的 子程序中局 部可见的数 据对象。

```
PROGRAM SAPMZTST.

PERFORM DATATEST1.
PERFORM DATATEST1.

SKIP.

PERFORM DATATEST2.
PERFORM DATATEST2.

FORM DATATEST1.
   TYPES F_WORD(5) TYPE C.
   DATA  F_TEXT TYPE F_WORD VALUE 'INIT'.
   WRITE F_TEXT.
   F_TEXT = '12345'.
   WRITE F_TEXT.
ENDFORM.

FORM DATATEST2.
   TYPES    F_WORD(5) TYPE C.
   STATICS  F_TEXT TYPE F_WORD VALUE 'INIT'.
   WRITE F_TEXT.
   F_TEXT = 'ABCDE'.
   WRITE F_TEXT.
ENDFORM.
```

启动 SAPMZTST 后，输出如 下：

```
INIT  12345 INIT  12345

INIT  ABCDE ABCDE ABCDE
```

在该示例中 ，定义了两 个类似的子 程序：DATATEST1 和 DATATEST2。 在 DATATEST2 中，用 STATICS 语句而不是 DATA 语句声明数 据对象 F_TEXT。 每次调用 DATATEST1 期间，F_TEXT 都被再次初 始化，但都 为 DATATEST2 保留值。STATICS 语句的 VALUE 选项仅在第 一次调用 DATATEST2 时起作用。

显式定义局 部数据对象

要避免全局 数据对象值 在子程序内 被更改，请 使用 LOCAL 语句，用法 如下：

语法
LOCAL <f>.
只能在 FORM 语句和 ENDFORM 语句之间使 用该语句。 对于 LOCAL，可以保存 无法被子程 序内的数据 声明隐藏的 全局数据对 象值（参见 *定义动态局 部数据类型 和对象*（页 140））。
例如，在子 程序内，用 TABLES 语句（参见 *TABLES 语句*（页 3－20））无法声明 由另一个 TABLES 语句定义的 表格工作域 。如果想局 部使用表格 工作域，但 保留其在子 程序外部的 内容，则必 须使用 LOCAL 语句。
关于该语句 的详细信息 ，参见 LOCAL 关键字文档 。

如下所示， 假定程序 FORMPOOL 包含两个子 程序 TABTEST1 和 TABTEST2：

```
PROGRAM FORMPOOL.

TABLES SFLIGHT.

FORM TABTEST1.
   SFLIGHT-PLANETYPE = 'A310'.
   SFLIGHT-PRICE = '150.00'.
   WRITE: / SFLIGHT-PLANETYPE, SFLIGHT-PRICE.
ENDFORM.

FORM TABTEST2.
   LOCAL SFLIGHT.
   SFLIGHT-PLANETYPE = 'B747'.
   SFLIGHT-PRICE = '500.00'.
   WRITE: / SFLIGHT-PLANETYPE, SFLIGHT-PRICE.
ENDFORM.
```

如下所示， 假定程序 SAPMZTST 调用 TABTEST1 和 TABTEST2：

```
PROGRAM SAPMZTST.

TABLES SFLIGHT.

PERFORM TABTEST1(FORMPOOL).
WRITE: / SFLIGHT-PLANETYPE, SFLIGHT-PRICE.

PERFORM TABTEST2(FORMPOOL).
WRITE: / SFLIGHT-PLANETYPE, SFLIGHT-PRICE.
```

启动 SAPMZTST 后，输出如 下：

```
A310                 150.00

A310                 150.00

B747                 500.00

A310                 150.00
```

该示例中， 通过参照 ABAP/4 词典结构 SFLIGHT 创建了一个 表格工作域 。在 TABTEST1 和 TABTEST2 中，给表格 工作域 SFLIGHT 指定不同的 值。TABTEST1 中指定的值 全局有效， 而 TABTEST2 中指定的值 仅局部有效 。

中断子程序

要中断子程 序，所用方 法与使用 EXIT 或 CHECK 语句中断循 环的方式类 似（参见 *终止循环*（页 7－20））。

- 用 EXIT 无条件中断 子程序。
- 用 CHECK 根据条件中 断子程序。

如果用 EXIT 或 CHECK 中断子程序 ，则系统在 该点中断子 程序、传递 参数，并继 续 PERFORM 语句后的语 句。

在 FORM 例程中，如 果在循环内 使用 EXIT 或 CHECK， 则循环的中 断条件适用 ，说明参见 *终止循环*（页 7－20）。中断循环 时， EXIT 语句和 CHECK 语句的工作 方式不同， 但中断子程 序时，方式 相同。

无条件中断 子程序

要无条件中 断子程序， 请使用 EXIT 语句，用法 如下：

语法

EXIT.
执行 EXIT 语句后，系 统立即退出 子程序，进 行 PERFORM 语句后的处 理。

```
PROGRAM SAPMZTST.

PERFORM TERMINATE.
```

```
WRITE  'The End'.

FORM TERMINATE.
   WRITE '1'.
   WRITE '2'.
   WRITE '3'.
   EXIT.
   WRITE '4'.
ENDFORM.
```

启动 SAPMZTST 后，输出如 下：

1 2 3 The End

在该示例中 ，子程序 TERMINATE 在第三个 WRITE 语句后中断 。

有条件地中 断子程序

要有条件中 断子程序， 请使用 CHECK 语句，用法 如下：

语法

CHECK <condition>.

如果条件不 满足，则系 统退出子程 序，并开始 PERFORM 语句后的处 理。对于 <condition>， 可以使用*编程逻辑表 达式* (页 7-3) .中 说明的任意 逻辑表达式 。

```
PROGRAM SAPMZTST.

DATA: NUM1 TYPE I,
      NUM2 TYPE I,
      RES  TYPE P DECIMALS 2.

NUM1 = 3. NUM2 = 4.
PERFORM DIVIDE USING NUM1 NUM2 CHANGING RES.

NUM1 = 5. NUM2 = 0.
PERFORM DIVIDE USING NUM1 NUM2 CHANGING RES.

NUM1 = 2. NUM2 = 3.
PERFORM DIVIDE USING NUM1 NUM2 CHANGING RES.

FORM DIVIDE USING N1 N2 CHANGING R.
   CHECK N2 NE 0.
   R = N1 / N2.
   WRITE: / N1, '/', N2, '=', R.
ENDFORM.
```

启动 SAPMZTST 后，输出如 下：

```
         3 /          4 =          0.75
         2 /          3 =          0.67
```

该示例中， 由于 N2值为零 ，故系统在 CHECK 语句后的第 二次调用期 间，退出子 程序 DIVIDE。

功能模块

功能模块是 存储在中央 库内的特殊 外部子程序 。R/3 系统提供大 量预定义的 功能模块， 可以从 ABAP/4 程序中进行 调用，而且 可以创建自 己的功能模 块。

功能模块和 一般 ABAP/4 子程序的显 著差异在于 一个明确定 义的、用于 与功能模块 进行数据传 递的接口。 对于功能模 块，不能将 数据声明为 公共部分。 对于 ABAP/4 词典表格， 调用程序和 被调用的功 能模块都有 单独的工作 域。

接口有助于 输入和输出 参数的传递 并将其标准 化。例如， 可以给功能 模块的输入 值分配缺省 值。可以决 定通过值还 是参考值进 行数据传递 （关于进行 数据传递的 详细信息， 参见 *通过参数进 行数据传递* (页 134))。接口也支持例外处 理。借助例 外，可以处 理可能发生 的错误。

用 ABAP/4 工作台调用 、创建和维 护功能模块 。例如，通 过从 ABAP/4 程序外的事 务屏幕调用 功能模块， 可以对其进 行测试。

在功能库中 ，可以将多 个功能模块 组合成功能 组。因为可 以定义功能 组中所有功 能模块都能 访问的全局 数据，因此 ，在一个功 能组中包含 操作相同数 据（如内表 ）的功能模 块比较合理 。

下列主题就 现有功能模 块的使用原 则以及如何 创建新的功 能模块等加 以说明。关 于功能模块 和功能库的 详细信息， 参见文档*工作台工具* (页 Error! Not a valid link.)。。

使用现有功 能模块

下面主题说 明，

调用可用功 能模块列表
要获得可用 功能模块列 表，请执行 以下操作：

1. 通过 “ABAP/4 开发工作台 ”屏幕，在 应用工具条 中，选择“ 开发 -> 功能库”， 或者选择“ 功能库”。
   显示“功能 库：维护功 能模块”屏 幕。
2. 选择 “查找”。
   显示“ABAP/4 资源库信息 系统：功能 模块”屏幕 。
3. 输入 所有已知的 搜索标准（ 例如 **string_sp***） ，并选择“ 输入”。
   显示符合输 入标准的所 有功能模块 列表。
4. 在该 列表中，双 击相应的复 选框，选择 所需的功能 模块。
   例如：
   - 要更改 功能模块， 选择“更改 ”（关于该 主题的详细 信息，参见 文档 *ABAP/4 工作台工具*（页 **Error! Not a valid link.**）） 。
   - 要显示 关于功能模 块的文档， 选择“显示 ”（关于该 主题的详细 信息，参见 *显示功能模 块的属性*（页 98）） 。
   - 要测试 功能模块， 选择“执行 ”（关于该 主题的详细 信息，参见 *测试功能模 块*（页 101）） 。

显示功能模 块的属性
要显示功能 模块的属性 ，请将功能 模块名输入 到“功能库 ：维护功能 模块”屏幕 。例如：
然后，选择 要显示属性 的单选按钮 ，并选择“ 显示”。如 果要更改属 性，可以在 “ABAP/4 功能库：初 始屏幕” 上 选择“更改 ”，或在下 面的屏幕上 选择“显示 <-> 更改”。
例如，重要 属性有：

文档
“文档”屏 幕显示如下 ：
文档就功能 模块的目的 加以说明，列出与模块 进行数据传 递的参数以 及例外。参 数类型 I 的参数为输 入参数，用 于向功能模 块传递数据 。参数类型 E 的参数为输 出参数，用 于从功能模 块向调用函 数传递数据 。例外主要 对功能模块 中可能出现 的错误情况 进行说明。

接口
选择“输入 /输出参数 接口”之后 ，就会出现 下列屏幕：
选择“表格 参数/例外 接口”之后 ，就会出现 下列屏幕：
上述屏幕列 出功能模块 中用于数据 传递的所有 形式参数。 传递参数的 过程与子程 序的过程类 似（参见 *通过参数进 行数据传递*（页 134））。如果选 择“参考值 ”复选框， 就通过参考 值传递参数 。否则，如 果不选择该 复选框，就 通过值传递 参数。

- 输入参数 与子程序的 形式输入参 数相对应。 用于将数据 从调用程序 传递到功能 模块。即使 输入参数是 通过参考值 传递的，也 不能改写。
- 输出参数 与子程序的 形式输出参 数相对应。 用于将数据 从功能模块 传递回调用 程序。
- 通过参考 值或值和结 果（参见 *通过值和结 果进行传递*（页 135））传递更改 参数。更改 参数即是输 入参数又是 输出参数。 更改传递给 功能模块的 值，并将其 返回给调用 程序。
- 表格参数 是内表。内 表处理方式 与更改参数 一样，总是 通过参考值 传递。
- 例外用于 处理功能模 块中可能发 生错误的情 况。调用程 序检查是否 发生了错误 ，然后采取 相应的措施 。

输入参数、 更改参数和 表格参数是 “可选”项 。这就是说 ，在调用程 序中调用函 数时，可以 忽略相应的 实参数。如 果参数可选 ，并且不指 定实参数， 则可以指定 在功能模块 中使用的缺 省值。输出 参数总是可 选。
与子程序一 样（参见 *键入形式参 数*（页 136）），可以在 字段“参考 值类型”中 指定形式参 数的数据类 型。在字段 “参考值结 构”中，可 以指定 ABAP/4 词典参考结 构或字段。 然后，在运 行时，系统 对照结构或 字段检查当 前参数。

源代码
ABAP/4 编辑器屏幕 显示功能模 块的 ABAP/4 源代码。可 以按照 *显示或更改 程序*（页 1－7）中说明的处 理正常 ABAP/4 程序的方式 ，处理源代 码。

测试功能模 块
可以选择“ 单个测试” ，通过“功 能库：维护 功能模块” 屏幕对功能 模块进行测 试，不用从 ABAP/4 程序中调用 。在“测试 功能模块” 屏幕上，可 以为输入参 数分配值。
选择“执行 ”后，出现 “功能模块 ：结果屏幕 ”：
随后，可以 发现功能模 块 STRING_SPLIT 的工作方式 与 ABAP/4 关键字 SPLIT（*拆分字符串*（页 6－33））相同。 在 DELIMITER "-" 处，将输入 参数 STRING "FUNCTION-MODULE" 的值分为“ FUNCTION” 和“MODULE” ，并将其分 配给输出参 数 HEAD 和 TAIL。 请注意，STRING 的内容改为 大写字母， 因为“测试 功能模块” 屏幕上的“ 大写/小写 ”复选框空 着。

调用功能模 块
要从 ABAP/4 程序调用功 能模块，请 使用 CALL 语句，用法 如下：

语法

```
CALL FUNCTION <module>
     [EXPORTING  F1 = a1 .... fn = an]
     [IMPORTING  F1 = a1 .... fn = an]
     [CHANGING   F1 = a1 .... fn = an]
     [TABLES     F1 = a1 .... fn = an]
     [EXCEPTIONS e1 = r1 .... en = rn [OTHERS = ro]].
```

可以将功能 模块 <module> 的名称指定 为文字或变 量。通过将 实参数显式 指定给 EXPORTING、 IMPORTING、 TABLES 或 CHANGING 选项后面列 表中的形式 参数，与功 能模块之间 进行参数传 递。

分配总是有 以下格式：
<formal parameter> = <actual parameter>

- EXPORTING 选项允许将 实参数 a_i 传递给形式 输入参数 f_i。在功 能模块中， 必须将形式 参数声明为 **输入** 参数。
- IMPORTING 选项允许将 形式**输出**参 数 f_i 传递给实参 数 a_i。在功 能模块中， 必须将形式 参数声明为 **输出** 参数。
- CHANGING 选项允许将 实参数 a_i 传递给形式 参数 f_i ，并在处理 功能模块后 ，系统将（ 更改过的） 形式参数 f_i 传递回实参 数 a_i。在功 能模块中， 必须将形式 参数声明为 更改参数。
- TABLES 选项允许在 实参数和形 式参数间传 递表格。借 助该选项， 内表总是通 过参考值传 递。

EXPORTING、 IMPORTING 和 CHANGING 选项的参数 可以是任意 类型的数据 对象。这些 选项的功能 类似于子程 序的 FORM 和 PERFORM 语句中（参 见 *通过参数进 行数据传递* (页 134) ）的 USING 和 CHANGING 选项。TABLES 选项的参数 必须为内表 。TABLES 选项对应于 FORM 和 PERFORM 语句的 TABLES 选项（参见 *向子程序传 递内表* (页 139) ）。用 EXCEPTIONS 选项，可以 处理功能模 块处理过程 中发生的例 外。例外是 功能模块的 特殊参数。 有关如何定 义和出现例 外的内容，*创建和编写 功能模块* (页 44) .中有专门 说明。如果 出现 e_i 例外，则系 统停止执行 功能模块， 并且不将任 何值从功能 模块传递给 程序，通过 参考值传递 的值例外。 如果在 EXCEPTION 选项中指定 了 e_i，则调用程 序通过将 r_i 分配给 SY-SUBRC 来处理例外 。必须将 r_i 指定为数字 文字。可以 使用 EXCEPTION 列表中的 OTHERS 处理列表中 没有显式指 定的所有例 外，并且可 将同一数值 r_i 用于多个例 外。

通过 ABAP/4 编辑器屏幕 选择“编辑 ->插入语句” 是在程序中 包括功能模 块调用的最 简单方法。 可以在出现 的对话窗口 中选择“调 用函数”， 并输入要调 用的功能模 块名。

然后选择“ 回车”，则 系统将 CALL 语句（包含 指定功能模 块的所有可 能选项）插 入程序代码 中。

对于功能模 块 STRING-SPLIT， 插入的代码 如下显示：

```
CALL FUNCTION 'STRING_SPLIT'
     EXPORTING
          DELIMITER =
          STRING    =
     IMPORTING
          HEAD      =
          TAIL      =
     EXCEPTIONS
          NOT_FOUND = 01
          NOT_VALID = 02
          TOO_LONG  = 03
          TOO_SMALL = 04.
```

现在，可以 按需要更改 该代码。

```
REPORT SAPMZTST.

DATA: TEXT(20),
      FRONT(20),
      END(20).

TEXT = 'Testing:String_Split'.

CALL FUNCTION 'STRING_SPLIT'
     EXPORTING DELIMITER = ':' STRING  = TEXT
     IMPORTING HEAD = FRONT TAIL = END
     EXCEPTIONS NOT_FOUND = 1 OTHERS = 2.

CASE SY-SUBRC.
   WHEN 1.        WRITE / 'Not found'.
   WHEN 2.        WRITE / 'Other errors'.
   WHEN OTHERS.   WRITE: / FRONT, / END.
ENDCASE.
```

启动 SAPMZTST 后，输出如 下：

```
Testing
String_Split.
```

该示例中， 调用了功能 模块 STRING_SPLIT。 通过使用 EXPORTING 选项，将实 参数“:” 和 TEXT 传递给形式 参数 DELIMITER 和 STRING。 如果这一过 程成功完成 ，则通过 IMPORTING 选项将形式参数 HEAD 和 TAIL 传递给实参 数 FRONT 和 END。

如果 STRING_SPLIT 中出现 NOT_FOUND 例外，则将 SY-SUBRC 的值设置为 1。如果出 现另一个例 外，则将 SY-SUBRC 的值设置为 2。用 CASE 语句处理该 例外。例如 ，如果将某 个 TEXT 中不包括的 文字分配给 EXPORTING 列表中的形 式参数 DELIMITER， 则输出：

```
Not found
```

创建和编写 功能模块

下列主题说 明如何创建 简单功能模 块。关于该 主题的详细 信息，参见 文档*工作台工具* (页 Error! Not a valid link.) 。

创建功能模 块
要创建新的 功能模块， 请执行以下 操作：

1. 通过“功 能库：维护 功能模块” 屏幕，输入 功能模块名 并选择“创 建”。
2. 出现“功 能模块创建 ：管理”屏 幕。必须在 该屏幕上为 模块输入功 能组 <fgrp>。 也可以为模 块输入一段 短文本。
   属于某个功 能组的所有 功能模块都 被组合到一 个 ABAP/4 程序 SAPL<fgrp>中 。通过在“ 功能库：维 护功能模块 ”屏幕上选 择“转向 -> 功能组 -> 创建组”， 创建功能模 块（参见文 档 *ABAP/4 工作台工具* （页 **Error! Not a valid link.**））。
   然后，选择 “保存”保 存功能模块 ，并选择“ 返回”退出 屏幕。
3. 选择 “输入/输 出参数接口 ”单选按钮 ，通过“功 能库：维护 功能模块” 屏幕定义接 口，然后选 择“更改” 。在“功能 模块更改： 输入/输出 参数”屏幕 上，键入输 入、输出和 更改参数。
   要输入例外 ，请选择“ 转向 -> 表格/例外 接口”。通 过“功能模 块更改：表 格参数/例 外”屏幕， 可以输入例 外名称。
   选择“保存 ”保存接口 ，并选择“ 返回”退出 屏幕。
4. 选定 “源代码” 单选按钮， 通过“功能 库：维护功 能模块”屏 幕输入“源 代码”，然 后选择“更 改”。出现 ABAP/4 编辑器，其 中包含您的 程序。定义 的参数以注 释模式出现 。
   现在，可以 输入并保存 程序代码（ 参见 *编写功能模 块* （页 60） ）
5. 选择 “功能模块 -> 激活”，通 过“功能库 ：维护功能 模块”屏幕 功能模块， 然后按照*测试功能模 块* （页 144） 中 的说明进行 测试。

编写功能模 块
要编写功能 模块，必须 如下在 FUNCTION 和 ENDFUNCTION 语句间包含 语句：

语法
```
FUNCTION <module>.
  <statements>
ENDFUNCTION.
```
要输入程序 代码，方法 与正常子程 序一样，但 下列情况例 外：

功能模块中 的数据处理
不必在功能 模块的源代 码中声明输 出和输入参 数。系统在 INCLUDE 程序中执行 该任务。将 已定义的参 数列表当作 注释行插入 到源代码中 （参见 *创建功能模 块* （页 146） 中的步骤 4）中。
与子程序一 样，可以在 功能模块中 声明局部数 据类型（参 见 *在子程序中 定义局部数 据类型和对 象* （页 140））。选择“全局 数据”，通 过“功能库 ：维护功能 模块”屏幕 为 INCLUDE 程序 L<fgrp>TOP 打开 ABAP/4 编辑器。INCLUDE 程序被作为 第一个语句 包含到程序 SAPL<fgrp> 中。 SAPL<fgrp> 是功能组 <fgrp> （由该组的 所有功能模 块组成）的 主程序。单 个功能模块 又包含在 INCLUDE 程序中（参 见文档*工作台工具* （页 **Error! Not a valid link.**））。
可以用 TYPES 和 DATA 语句将数据 声明写入 L<fgrp>TOP。 该数据对功 能组的所有 模块是全局 数据，对该 组是局部数 据。调用第 一个模块之 后，系统立 即创建他们 ，并始终保 留最后一次 模块调用的 值。

从功能模块 调用子程序
可以从功能 模块调用不 同的子程序 。

- 可以在模 块的 ENDFUNCTION 语句后面直 接写入内部 子程序的程 序代码。可 以从功能组 的所有模块 中调用该子 程序。但是 ，要想更清 晰，则应该 仅从功能模 块（在该模 块后写子程 序）调用该 子程序。
- 要创建可 以从功能组 <fgrp>的所有模块 调用的内部 子程序，请 使用特殊的 INCLUDE 程序 L<fgrp>F<xx> 。在“功能 库：维护功 能模块”屏 幕上选择“ 主程序”后 ，在主程序 SAPL<fgrp> 中双击 INCLUDE 程序名，则 可以打开这 些程序。关 于该主题的 详细信息， 参见文档*工作台工具* （页 **Error! Not a valid link.**）。
- 可以调用 任意外部子 程序。

产生例外
为产生例外 ， ABAP/4 提供两个仅 能在功能模 块中使用的 语句：

语法
```
RAISE <exc>.
MESSAGE ..... RAISING <exc>.
```
这些语句的 效果依赖于 是调用程序 自己处理例 外，还是将 例外留给系 统处理。如 果在 CALL FUNCTION 语句的 EXCEPTION 选项中指定 例外的名称 <exc> 或 OTHERS（ 参见 *调用功能模 块* （页 144）），则调用 程序自己处 理例外。如果系统处 理例外，

- RAISE 语句中断程 序，并切换 到调试模式
- MESSAGE..... RAISING 语句显示一 条信息。系 统按照信息 类型继续处 理。在 INCLUDE 程序 L<fgrp>TOP 的第一个语 句中，必须 指定 MESSAGE-ID。 关于在报表 中处理信息 的信息，参 见 *列表中的消 息* （页 **Error! Not a valid link.**），关于在事 物编程中处 理信息的信 息，参见*处理错误和 消息* （页 **Error! Not a valid link.**）。。

如果调用程 序自己处理 例外，两个 语句都直接 将程序的控 制权返回给 调用程序， 并且系统不 将任何数据 从功能模块 传递给程序 ，通过参考 值调用的参 数除外。MESSAGE..... RAISING 语句不显示 任何信息， 但填充在程 序中可以访 问的下列字 段：

- SY-MSGID （信息标识 符）
- SY-MSGTY （信息类型 ）
- SY-MSGNO （信息号码 ）
- SY-MSGV1 到 SY-MSGV4（ 字段 <**F1**> 到 <**F4**> 的内容，包 含在信息中 ）

（参见 MESSAGE 的关键字文 档）

假定有下面 的功能模块 ：

```
FUNCTION MY_DIVIDE.
```

```
*"----------------------------------------------------------------------
*" Local interface:
*"       IMPORTING
*"             N1
*"             N2
*"       EXPORTING
*"              R
*"       EXCEPTIONS
*"             DIV_ZERO
*"----------------------------------------------------------------------

   IF N2 EQ 0.
      RAISE DIV_ZERO.
   ELSE.
      R =  N1 / N2.
   ENDIF.

ENDFUNCTION.
```

该功能模块 中，如果 N2 不等于零， 则用 N1 除以 N2。否则 ，产生例外 DIV_ZERO。

假定下面的 程序 SAPMZTST 调用 MY_DIVIDE：

```
REPORT SAPMZTST.

DATA: RES TYPE P DECIMALS 2.

CALL FUNCTION 'MY_DIVIDE'
     EXPORTING   N1 = 5 N2 = 4
     IMPORTING   R  = RES
     EXCEPTIONS  DIV_ZERO = 11.

IF SY-SUBRC EQ 0.
   WRITE: / 'Result =', RES.
ELSE.
   WRITE 'Division by zero'.
ENDIF.
```

启动 SAPMZTST 后，输出如 下：

```
RESULT =           1.25
```

如果在 EXPORTING 列表中用 N2 = 0 代替 N2 =， SAPMZTST 通过将 11 传递给 SY-SUBRC， 处理例外 DIV_ZERO， 输出如下：

```
Division by zero
```

第十章 使用字段符号

概览

内容

在 ABAP/4 程序中，字段符号是现有字段的占位符。字段符号本身不直接为字段保留空间，而只是指向一个字段（该字段在程序运行前还未可知）。字段符号可以与程序语言 C 中的指针概念相比较（即，用内容操作符 * 表示的指针）。然而，在 ABAP/4 中，从变量的意义上说，与指针不是真正等价的，此处，变量包含内存地址，并且不用内容操作符即可使用。您只能使用字段符号指向的数据对象。

本节介绍

字段符号的概念

有时仅知道运行时要处理哪个字段和怎样处理它。

为此，可以在程序中创建字段符号。运行时，可以将实际字段分配给这种字段符号。 用字段符号编程的所有操作将用分配的字段来执行。成功地完成分配后，在 ABAP/4 中无论参考字段符号或参考字段本身，都没有任何区别。

在 ABAP/4 中字段符号能指向任何数据对象，也能指向 ABAP/4 字典中定义的结构。

不管是否有类型说明，都可以创建字段符号。如果没有说明，字段符号采用分配字段的所有属性。如果有说明，在分配过程中系统将检查被分配的字段是否与字段符号类型相匹配。

无论哪一种应用，在程序中使用字段符号之前，必须首先分配一个字段给字段符号。

字段符号提供了一些特性，使得它们具有很大的灵活性：

- 可以将所分配字段的偏移量和长度指定为变量。
- 可以将字段符号分配给其它字段符号，甚至指定其偏移量和长度。
- 对字段符号的分配可以扩展到字段边界之外。这允许对规则存储的数据进行有效的访问。
- 可以强制字段符号具有不同于分配字段的类型和小数位。

_ 字段符号可有一个结构用来指向结构的单个组件。

尽管字段符号的这些灵活性使您找到一些问题的完美解答，但使用起来要十分小心。由于可以将直到运行时才可知的数据对象分配给字段符号，所以对涉及到字段符号的操作，语法和安全性检查的有效性是非常有限的。这可能导致运行错误或不正确的数据分配。

运行错误指明明显的问题，而不正确的数据分配是很危险的，因为它们很难被检测出来。因此，只有在绝对确信所做事情，或者没有其它 ABAP/4 语句能解决问题时才使用。

例如，如果正在处理字符串，您可能只想处理一个字符串的一部分，其位置和长度取决于字符串内容。可以使用字段符号做到这一点。然而，自从 R/3 系统的 3.0 版本后，您也可以使用具有变量偏移量和长度说明的 MOVE 语句（参见 *用指定偏移 量赋值*（页 6－3））。使用 MOVE 语句（可能与一些辅助变量结合使用）比使用字段符号更安全。字段符号优点在于某些情况下能提高响应时间。

定义字段符号

可以为任何内部数据对象定义字段符号
可以为内部和外部结构定义结构化的字段符号
可以在子程序和功能模块中局部地使用字段符号

为内部字段定义字段符号

要为内部数据对象定义字段符号，请如下使用 FIELD-SYMBOLS 语句：

语法
```
FIELD-SYMBOLS <FS> [<type>].
```
该语句定义字段符号 <FS>。

> 对字段符号，角括弧是语法部分。它们标识程序代码中的字段符号。

不管是否有类型说明，都可以定义字段符号。

没有类型说明的字段符号

要定义没有类型说明的字段符号，不能使用 FIELD-SYMBOLS 语句中的 <type> 选项：

语法
```
FIELD-SYMBOLS <FS>。
```
字段符号 <FS> 没有任何属性。可以在运行时将任何数据对象分配给 <FS>（参见*将数据对象 分配给字段 符号*（页 130））。分配过程中，字段符号继承数据对象的所有属性。分配的数据对象的数据类型成为字段符号的实际数据类型。

键入字段符 号

可以使用 FIELD-SYMBOLS 语句中 <type> 选项键入字段符号：

语法
```
FIELD-SYMBOLS <FS> <type>.
```
对 <type>，可以输入 TYPE <t> 或 LIKE <f>（参见 *DATA 语句的基本 格式*（页 3－14））。

当将数据对象分配到没有类型说明但已键入类型的字段符号 <FS> 时，系统将检查分配的数据对象的类型是否与字段符号的键入类型相兼容。兼容性规则由下表给出。如果类型不兼容，可能的话，系统在语法检查期间输出错误信息，或以运行错误响应。
另一方面，如果不管分配数据对象，而要字段符号保持它指定的类型，则不会出错。这里，必须将带类型说明的数据对象分配给已键入类型的字段符号。分配期间指定的类型必须与字段符号的键入类型相匹配（参见 *定义字段符 号的数据类 型*（页 131））。
下列兼容规则应用于字段符号的键入类型：

键入字段	分配语法检查
无类型说明， TYPE ANY	系统接受任何字段类型。字段的所有属性分配给字段符号。
TYPE TABLE	系统检查字段是否为内表。该表的所有属性和结构分配给字段符号。
TYPE C、N、P 或 X	系统检查字段是否属于类型 C、N、P 或 X。字段符号继承字段长度及其 DECIMALS 说明（对于类型 P）。
TYPE D、F、I、或 T LIKE <f>, TYPE <ud>	这些类型完全指定。系统检查字段数据类型是否完全与字段符号类型一致。

（<ud> 是用户定义的）
与对子程序形式参数的检查一样，用选项 < type > 来执行检查。(参见*键入形式参 数*（页 9 - 18））。

```
DATA DAT(8) VALUE '09161995'.
FIELD-SYMBOLS <FS> TYPE D.
ASSIGN DAT TO <FS>.
WRITE <FS>.
```

因为 DAT 与字段符号 <FS> 的键入类型不兼容，该程序在语法检查时导致错误。

```
DATA DAT(8) VALUE '19951609'.
FIELD-SYMBOLS <FS> TYPE D.
ASSIGN DAT TO <FS> TYPE 'D'.
WRITE <FS>.
```

在该程序中，ASSIGN 语句指定了与 <FS> 中键入类型相同的类型。因此，程序是可执行的，并产生如下结果：

```
09161995
```

关于 ASSIGN 语句的详细信息，参见 *将数据对象 分配给字段 符号*（页 Error! Not a valid link.）。

定义结构化的字段符号

可以用 FIELD-SYMBOLS 语句定义结构化的字段符号，如下所示：

语法

FIELD-SYMBOLS <FS> STRUCTURE <s> DEFAULT <f>.

该语句定义了初始指向字段 <f> 的结构化字段符号 <FS>。必须将初始字段分配给结构化字段符号 <FS>，但是以后可以更改这种分配（参见*将数据对象 分配给字段 符号*（页 130））。字段符号 <FS> 继承了 <S> 的结构。结构 <S> 可以是任何字符串或 ABAP/4 字典中定义的结构。不必用 TABLES 语句声明字典结构。必须通过输入不带引号的名称指定结构 <s>。说明在运行时不允许。

如果 <s> 不包含类型 I 或类型 F 的组件，<f> 可以是至少与结构 <s>的长度相等。如果 <f> 比 <s> 短，则语法检查时会出错。如果在后面的程序中将另一个字段分配给 <FS>，系统还将检查被分配的字段是否足够长。如果在运行时动态分配较短字段，（参见 *动态 ASSIGN*（页 140）），则发生运行错误。

如果 <s> 包含类型 I 或类型 F 组件，请注意该结构是对齐的（参见 *对齐数据对象*（页 6－43））。如果将数据对象分配给具有这种对齐结构的字段符号，则数据对象也必须相应地对齐。否则发生运行错误。在这种情况下，建议将数据对象只分配给结构化字段符号，该结构化字段符号至少在结构长度上保持与字段符号同样的结构。

定义结构化字段后，可以用符号给结构的单个组件加上参考数目。为此，可用由连字符分隔的字段符号名称作为组件字段名称的前缀。

例如，要为不同结构创建一个工作区或为同一结构创建几个工作区，可以使用结构化字段符号。

```
DATA: WA(100) VALUE '001LH 123419950627'.

DATA: BEGIN OF LINE1,
         COL1(6),
         COL2(4),
         COL3(8),
      END OF LINE1.

DATA: BEGIN OF LINE2,
         COL1 TYPE I VALUE 1,
         COL2 TYPE I VALUE 2,
      END OF LINE2.

DATA LINE3 LIKE LINE2.

DATA ITAB LIKE LINE2 OCCURS 10 WITH HEADER LINE.

LINE3-COL1 = 11. LINE3-COL2 = 22.

FIELD-SYMBOLS: <F1> STRUCTURE SBOOK DEFAULT WA,
               <F2> STRUCTURE LINE1 DEFAULT WA,
               <F3> STRUCTURE ITAB  DEFAULT LINE3.

WRITE: / <F1>-MANDT, <F1>-CARRID, <F1>-CONNID, <F1>-FLDATE,
       / <F2>-COL1,  <F2>-COL2, <F2>-COL3.
       / <F3>-COL1,  <F3>-COL2.
```

输出显示如下：

```
001 LH  1234 27.06.1995

001LH  1234 19950627

        11          22
```

该示例定义三个字段串 LINE1、 LINE2 和 LINE3 以及内表 ITAB（具有表头行）。然后，定义三个字段符号 <F1>、<F2> 和 <F3> 如下：

- <F1> 与 ABAP/4 字典结构 SBOOK 有同样的结构并指向字符串 WA。
- <F2> 与 LINE1 有同样的结构，并且也指向 WA。
- <F3> 与表头行 ITAB 有同样的结构，且指向 LINE3。<F3> 不能指向 WA，因为 ITAB 是对齐的。

当将字段符号输出到屏幕时，所分配字段的内容按照字段符号类型格式化。例如，注意 <F1>-FLDATE 具有数据类型 D。

定义局部字段符号

可用与局部数据类型和对象相似的方式在子程序和功能模块中定义局部字段符号（参见*在子程序中 T寰植渴_ 据类型和_ 象* (页 9 - 25)）。
在子程序或功能模块中声明的任何字段符号都是局部的。对于局部字段符号，可应用下列规则：

- 在子程序或功能模块外部，不能引用局部字段符号。
- 每次调用子程序或功能模块，即使前一次执行过 ASSIGN，也没有字段分配给局部字段符号。
- 局部字段符号可以与全局定义的字段符号或其它子程序、功能模块中的局部字段符号同名。局部字段符号隐藏全局字段符号。
- 结构化字段符号也能成为局部的。它们可以具有局部（内部）结构，可以分配局部字段给它们。

将数据对象分配给字段符号

在使用字段符号之前必须先分配数据对象给它。对于结构化字段符号，必须在定义中包括这个分配（参见*定义结构化 的字段符号* (页 150)）。对于非结构化字段符号，可以自由决定何时何地首次分配数据对象。在程序中，不管字段符号是否结构化，都可以将不同的数据对象分配给同样的字段符号。
要给字段符号分配数据对象，可以使用 ASSIGN 语句。ASSIGN 语句有几个变量和参数。下列主题说明：

ASSIGN 语句的基本格式

ASSIGN 语句的基本格式包含两个静态变量和两个动态变量。

静态 ASSIGN
如果运行之前知道想要分配给字段符号的数据对象的名称，则如下使用 ASSIGN 语句：

语法
ASSIGN <f> TO <FS>.
分配之后，字段符号 <FS> 具有数据对象 <f> 的属性，并指向相同的内存区。

```
FIELD-SYMBOLS: <F1>, <F2> TYPE I.

DATA: TEXT(20)  TYPE C VALUE 'Hello, how are you?',
      NUM       TYPE I VALUE 5,
      BEGIN OF LINE1,
         COL1 TYPE F VALUE '1.1e+10',
         COL2 TYPE I VALUE '1234',
```

```
      END OF LINE1,
      LINE2 LIKE LINE1.

ASSIGN TEXT TO <F1>.
ASSIGN NUM TO  <F2>.
DESCRIBE FIELD <F1> LENGTH <F2>.
WRITE: / <F1>, 'has length', NUM.

ASSIGN LINE1 TO <F1>.
ASSIGN LINE2-COL2 TO <F2>.
MOVE <F1> TO LINE2.
ASSIGN 'LINE2-COL2 =' TO <F1>.
WRITE: / <F1>, <F2>.
```

输出如下：

```
Hello, how are you?  has length         20

LINE-COL2 =      1,234
```

示例定义两个字段符号 <F1> 和 <F2>。因为 <F2> 的类型指定为 I，它只能指向 I 类型字段。在示例教程中 <F1> 和 <F2> 指向几个不同的数据对象。

具有偏移量说明的静态 ASSIGN

通过使用下列 ASSIGN 语句，可以指定要分配给字段符号的字段的偏移值：

语法：

```
ASSIGN <f>[+<o>][(<l>)] TO <FS>.
```

正如在*用指定偏移 量赋值* (页 6－3) 中所描述的，具有偏移量 <o> 和长度 <l> 的 <f> 部分被分配给字段符号 <FS>。在 ASSIGN 语句中，偏移量的说明有下列特殊特征：

- <o> 和 <l> 可以是变量
- 系统不检查所选部分是否位于字段 <f> 内部。偏移量 <o> 和 长度 <l> 都可以比 <f> 的长度大。写地址时，可以超出 <f> 的限制，但不能超出所定义内存区（参见 *运行检查* (页 131) ）。
- 如果没有指定长度 <l>，系统自动输入字段 <f> 的长度。如果 <o> 大于零，<FS> 总是指向超出 <f> 限制的区域。
- 如果<o>小于<f> 的长度，可以为 <l> 指定一个星号 （*） 来阻止 <FS> 参照 <f> 限制之外的区域。

```
FIELD-SYMBOLS <FS>.

DATA: BEGIN OF LINE,
       STRING1(10) VALUE '0123456789',
       STRING2(10) VALUE 'abcdefghij',
      END OF LINE.

WRITE / LINE-STRING1+5.

ASSIGN LINE-STRING1+5 TO <FS>.
WRITE / <FS>.
```

```
ASSIGN LINE-STRING1+5(*) TO <FS>.
WRITE / <FS>.
```

输出如下：

56789

56789abcde

56789

该示例中，可以看到 WRITE 语句和 ASSIGN 语句中的偏移量说明的区别。对于 WRITE，在 LINE-STRING1 端部截短了输出。如果指定偏移量大于9，将导致语法检查时出错。在第一个 ASSIGN 语句中，将 LINE-STRING1 中以偏移量 5开头长度为 10 的内存区域分配给字段符号 <FS>。因为程序中明确定义了 LINE-STRING1 后面的内存区，所以这将导致很有意义的输出结果。在第二个 ASSIGN语句中，避免了 LINE-STRING1 边界后面的内存分配。

```
FIELD-SYMBOLS <FS>.

DATA: BEGIN OF LINE,
         A VALUE '1', B VALUE '2', C VALUE '3', D VALUE '4',
         E VALUE '5', F VALUE '6', G VALUE '7', H VALUE '8',
      END OF LINE,
      OFF TYPE I,
      LEN TYPE I VALUE 2.

DO 2 TIMES.
   OFF = SY-INDEX * 3.
   ASSIGN LINE-A+OFF(LEN) TO <FS>.
    <FS> = 'XX'.
ENDDO.

DO 8 TIMES.
   OFF = SY-INDEX - 1.
   ASSIGN LINE-A+OFF(1) TO <FS>.
   WRITE <FS>.
ENDDO.
```

输出如下：

1 2 3 X X 6 X X

该示例中，可以看到字段符号是如何方便访问和规则字段串的处理的。然而，请注意，这种处理字段限制范围之外内存内容的灵活方法也有其危险性，可能导致运行错误。

动态ASSIGN

如果仅在运行时才知道要分配给字段符号的数据对象名称，请使用下列 ASSIGN 语句：

语法
```
ASSIGN (<f>) TO <FS>.
```
这个语句将名称包含在 <f> 中的字段分配给字段符号 <FS>。对（<f>）不能指定偏移量。运行时，系统按下列顺序搜索给定的字段：

1. 如果分配是在子程序或功能模块中执行的，系统将在局部数据的子程序或功能模块中搜索字段。
2. 如果分配是在子程序或功能模块之外执行的，或者在其中未找到字段，则系统将在程序的全局数据中搜索字段。
3. 如果在全局数据中未发现此字段，系统将在表工作区中搜索，该表工作区是指当前程序组的主程序中 TABLES 语句声明的表工作区。程序组由主程序和包含外部子程序的定义的所有程序组成，这些外部子程序被主程序调用（包括被嵌套的程序）。

> 由于这种搜索对响应时间有负面效应，只有绝对必要时，才可使用动态 ASSIGN。如果在运行之前知道只分配表工作区，则可以使用在 *表工作区的 动态分配* （页 132）中所描述的 ASSIGN 语句的变量。

如果搜索成功并且字段可分配给字段符号，则 SY-SUBRC 设为 0。否则，返回 4。为安全起见，动态 ASSIGN 之后总要检查 SY-SUBRC 的值，以避免字段符号指向未定义区域。

> 假设主程序 SAPMZTST 如下：
>
> ```
> PROGRAM SAPMZTST.
> TABLES SBOOK.
> SBOOK-FLDATE = SY-DATUM.
> PERFORM FORM1(MYFORMS1).
> ```
>
> 假设还调用两个程序 MYFORMS1 和 MYFORMS2：
>
> ```
> PROGRAM MYFORMS1.
> FORM FORM1.
>    PERFORM FORM2(MYFORMS2).
> ENDFORM.
> ```
>
> 和
>
> ```
> PROGRAM MYFORMS2.
> FORM FORM2.
>    DATA NAME(20) VALUE 'SBOOK-FLDATE'.
>    FIELD-SYMBOLS <FS>.
>    ASSIGN (NAME) TO <FS>.
>    IF SY-SUBRC EQ 0.
>       WRITE / <FS>.
>    ENDIF.
> ENDFORM.
> ```
>
> 执行 SAPMZTST 之后，输出显示如下：
>
> 08.02.1995
>
> 该示例中，程序组由 SAPMZTST、MYFORMS1 和 MYFORMS2 组成。在 MYFORMS2 中定义字段符号 <FS>。动态 ASSIGN 语句之后，指向主程序 SAPMZTST 中声明的表工作区的组件 FLDATE。用静态 ASSIGN 语句，这

是不可能的。换句话说，不能用 SBOOK-FLDATE 替代 (NAME)，因为语法检查时将导致错误。

表工作区的动态分配

如果运行之前知道要将表工作区分配给字段符号，但不知道表区的名称，请使用动态 ASSIGN 语句的下列变量：

语法

```
ASSIGN TABLE FIELD (<f>) TO <FS>.
```

系统仅在表工作区中搜索要分配给字段符号的数据对象，该表工作区是由当前程序组的主程序中 TABLES 语句声明的。即，系统仅执行在 *动态 ASSIGN* (页 154) 所描述的搜索的第三步。

如果搜索成功，并且可以分配字段给字段符号，则 SY-SUBRC 设为 0。否则返回 4。

```
TABLES SBOOK.

DATA: NAME1(20) VALUE 'SBOOK-FLDATE',
      NAME2(20) VALUE 'NAME1'.

FIELD-SYMBOLS <FS>.

ASSIGN TABLE FIELD (NAME1) TO <FS>.
WRITE: / 'SY-SUBRC:', SY-SUBRC.

ASSIGN TABLE FIELD (NAME2) TO <FS>.
WRITE: / 'SY-SUBRC:', SY-SUBRC.
```

输出如下：

```
SY-SUBRC:       0

SY-SUBRC:       4
```

在第一个 ASSIGN 语句中，系统发现表工作区 SBOOK 和 SY-SUBRC 的组件 FLDATE 被设为 0。第二个 ASSIGN 语句中，系统没有发现字段 NAME1，因为它只被 DATA 语句声明，而不是被 TABLES 语句声明。在这种情形，SY-SUBRC 返回 4。

将字段符号分配给其它字段符号

除了使用数据对象的名称外，还可以将字段符号分配给 ASSIGN 语句的所有变量中的字段符号。为此，编码静态 ASSIGN 如下：

语法

```
ASSIGN <FS1>[+<o>][(<l>)] TO <FS2>.
```

可按下列方式编码动态 ASSIGN：

语法

```
ASSIGN [TABLE FIELD] (<f>) TO <FS2>.
```

字段 <f> 包含字段符号 <FS1> 的名称。
<FS1> 和 <FS2> 可以相同。

```
DATA: BEGIN OF S,
         A VALUE '1', B VALUE '2', C VALUE '3', D VALUE '4',
         E VALUE '5', F VALUE '6', G VALUE '7', H VALUE '8',
      END OF S.

DATA  OFF TYPE I.

FIELD-SYMBOLS <FS>.

ASSIGN S-A TO <FS>.

DO 4 TIMES.
   OFF = SY-INDEX - 1.
   ASSIGN <FS>+OFF(1) TO <FS>.
   WRITE <FS>.
ENDDO.
```

输出如下：

```
1 2 4 7
```

该示例中，创建八个字段作为字符串 S 的组件 S-A 到 S-H，并用“1”到“8”填充。这些字符串在内存中有规律的存储。组件 S-A 初始分配给字段符号 <FS>。循环语句中有下列结果：

循环途径 1：
<FS> 指向 S-A，OFF 为 0，并且 S-A 被分配给 <FS>

循环途径 2：
<FS> 指向 S-A，OFF 为 1，并且 S-B 被分配给 <FS>

循环途径 3：
<FS> 指向 S-B，OFF 为 2，并且 S-D 被分配到 <FS>

循环途径 4：
<FS> 指向 S-D，OFF 为 3，并且 S-G 被分配给 <FS>

分配字段串组件

可以用 ASSIGN 语句将字段串的特殊组件分配给字段符号，过程如下：

语法
```
ASSIGN COMPONENT <comp> OF STRUCTURE <s> TO <FS>.
```
系统将字符串 <s> 的组件 <comp> 分配给字段符号 <FS>。可以指定 <comp> 为文字或变量。如果 <comp> 属于类型 C 或字段串（象组件一样没有内表），它指定组件的名称。如果 <comp> 有任何其它基本数据类型，将被转化为类型 I（参见 *类型转换* (页 6 - 36)），并且指定组件号。
如果分配成功，SY-SUBRC 设为 0。否则，返回 4。

```
DATA: BEGIN OF LINE,
         COL1 TYPE I VALUE '11',
         COL2 TYPE I VALUE '22',
         COL3 TYPE I VALUE '33',
      END OF LINE.

DATA COMP(5) VALUE 'COL3'.

FIELD-SYMBOLS: <F1>, <F2>, <F3>.

ASSIGN LINE TO <F1>.
ASSIGN COMP TO <F2>.

DO 3 TIMES.
   ASSIGN COMPONENT SY-INDEX OF STRUCTURE <F1> TO <F3>.
   WRITE <F3>.
ENDDO.

ASSIGN COMPONENT <F2> OF STRUCTURE <F1> TO <F3>.
WRITE / <F3>.
```

输出如下：

```
11          22          33

33
```

该示例中，<F1> 指向字段串 LINE，<F2> 指向字段 COMP。循环中，LINE 的组件由其号码指定，并且一个接一个地被分配到 <F3>。 循环之后，LINE 的组件 COL3 由其名称指定，并被分配到 <F3>。

定义字段符号的数据类型

可以使用 ASSIGN 语句的 TYPE 选项定义字段符号的数据类型，过程如下：

语法

```
ASSIGN ..... TO <FS> TYPE <t>.
```

可以用 ASSIGN 语句的所有变量使用 TYPE 选项。

分配数据对象给带类型说明的字段符号时，必须区分下列符号：

- 未定义类型的字段符号
  对于 TYPE，未定义类型的字段符号 <FS> 不继承所分配数据对象的数据类型和输出长度，但继承在 <t> 中指定的数据类型。对于 <t>，可以使用任何预定义的数据类型（参见*基本数据类 型 - 预 T 义*_(页 3 - 3)）。关于预定义基本数据类型输出长度的详细信息，参见 *WRITE 语句* (页 4 - 1) 。类型 <t> 可以是文字或变量。
- 定义类型的字段符号
  如果要分配的数据对象的数据类型与字段符号的键入类型不兼容，使用具有定义类型的字段符号的 TYPE 选项很有意义，但要避免结果错误消息。在这种情况下，所指定的类型 <t> 和字段符号的键入类型必须兼容。不管分配的数据对象怎样，字段符号保持它的数据类型。详细信息和示例，参见 *键入字段符 号* (页 149) 。

如果程序中分配之后使用字段符号，则分配的数据对象不被转化为指定类型 <t>。因此，数据对象内容应该解释为类型 <t> 的字段。

如果

_ 指定的数据类型未知
_ 指定数据类型的长度与分配的字段类型不匹配
_ 有对齐错误（参见*对齐数据对象* (页 6 - 42)）

则系统反应为运行错误。

```
DATA TXT(8) VALUE '19950606'.
DATA MYTYPE(1) VALUE 'X'.
FIELD-SYMBOLS <FS>.
ASSIGN TXT TO <FS>.
WRITE / <FS>.
ASSIGN TXT TO <FS> TYPE 'D'.
WRITE / <FS>.
ASSIGN TXT TO <FS> TYPE MYTYPE.
WRITE / <FS>.
```

输出如下：

```
19950606
06061995
3139393530363036
```

该示例中，三次将字符串 TXT 分配给 <FS>。第一次 <FS> 的类型保持不变，第二次 <FS> 的类型变为类型 D，第三次 <FS> 的类型变为类型 X。请注意，当成对考虑时，最后输出行的数字表示 TXT 中字符的十六进制代码。第二个输出行的格式取决于用户的主记录，而在第三行中则取决于由操作系统使用（在这种情形为 ASCII码）的字符表示法。

更改小数位

可以更改字段符号的小数位，该字段符号在分配时指向类型 P 字段。为此，请如下使用 ASSIGN 语句中 DECIMALS 选项：

语法

```
ASSIGN ..... TO <FS> DECIMALS <d>.
```

可以使用 DECIMALS 选项，该选项具有 ASSIGN 语句的所有变量。用 DECIMALS 选项，字段符号 <FS> 的小数位变为 <d>。一般地，这会导致字段符号和分配的字段有不同的数值。<d> 可以是文字或变量。如果 <d> 没有位于 0 到 14 之间，或者分配的字段不属类型 P，则会出现运行错误。

```
DATA: PACK1 TYPE P DECIMALS 2 VALUE '400',
      PACK2 TYPE P DECIMALS 2,
      PACK3 TYPE P DECIMALS 2.
```

```
FIELD-SYMBOLS: <F1>, <F2>.

WRITE: / 'PACK1', PACK1.

ASSIGN PACK1 TO <F1> DECIMALS 1.
WRITE: / '<F1> ', <F1>.

PACK2 = <F1>.
WRITE: / 'PACK2', PACK2.

ASSIGN PACK2 TO <F2> DECIMALS 4.
WRITE: / '<F2> ', <F2>.

PACK3 = <F1> + <F2>.
WRITE: / 'PACK3', PACK3.

<F2> = '1234.56789'.
WRITE: / '<F2> ', <F2>.
WRITE: / 'PACK2', PACK2.
```

输出如下：

```
PACK1              400.00

<F1>             4,000.0

PACK2            4,000.00

<F2>              40.0000

PACK3            4,040.00

<F2>           1,234.5679

PACK2         123,456.79
```

在该示例中，所有类型 P 字段有两个小数位。字段符号 <F1> 和 <F2> 分别有一个和四个小数位。请注意，对于字段符号及其分配字段，数值是不同的。

分配全局字段的局部副本

使用子程序时（参见 *子程序* (页 9 - 5)），可以在数据堆栈创建全局数据的局部副本。为此，请如下使用 ASSIGN 语句：

语法

```
ASSIGN LOCAL COPY OF ..... TO <FS>.
```

系统在堆栈上放置指定全局数据的副本。在子程序中，不必通过写字段符号 <FS> 地址更改全局数据，就能访问和更改该副本。

可以与 ASSIGN 语句中所有变量（在 *分配字段串 组件* (页 157) 中描述的除外）一起使用该选项。

假设主程序 SAPMZTST 如下：

```
PROGRAM SAPMZTST.
```

```
DATA: BEGIN OF COMMON PART,
        TEXT(5) VALUE 'Text1',
      END OF COMMON PART.

PERFORM ROUTINE(FORMPOOL).

WRITE TEXT.
```

假设包含子程序 ROUTINE 的主程序 FORMPOOL 如下：

```
PROGRAM FORMPOOL.

DATA: BEGIN OF COMMON PART,
        TEXT(5) VALUE 'Text1',
      END OF COMMON PART.

FORM ROUTINE.
   FIELD-SYMBOLS <FS>.
   ASSIGN LOCAL COPY OF TEXT TO <FS>.
   WRITE <FS>.
   <FS> = 'Text2'.
   WRITE <FS>.
   ASSIGN TEXT TO <FS>.
   WRITE <FS>.
   <FS> = 'Text3'.
ENDFORM.
```

启动 SAPMZTST 之后，输出如下：

```
Text1 Text2 Text1 Text3
```

示例中，字段 TEXT 被声明为主程序 SAPMZTST 和程序 FORMPOOL 的共用数据区。通过在程序 FORMPOOL 的子程序 ROUTINE中将 TEXT 分配给 <FS>，将字段 TEXT 的副本放置到局部数据堆栈上。通过写 <FS> 地址，可以读取和更改该副本。全局字段 TEXT 不受局部字段操作的影响。

运行检查

运行时，系统执行某些检查，以避免由在定义数据区之外的错误分配引起的内存数据的非控制丢失。

定义如下的数据区：

- 内表的表存储区。存储区的大小取决于表行数，这种表行数未固定，而运行时，经常动态扩展（参见*创建和处理 内表* (页 8 - 1)）。
- 其它数据对象的 DATA 存储区。该存储区的大小在数据声明期间是固定的（参见 *声明数据* (页 3 - 1)）。

字段符号不能指向这些区域之外的地址。如果系统在运行时遇到这类问题，将停止处理程序。

某些系统信息，诸如内表的控制块，也存储在 DATA 存储区中。因此，尽管有运行检查，还是有可能无意地覆盖一些字段，并引起后续错误（如表头损坏）。

```
PROGRAM SAPMZTST.
```

```
DATA: TEXT1(10), TEXT2(10), TEXT3(5).
FIELD-SYMBOLS <FS>.
DO 25 TIMES.
   ASSIGN TEXT1+SY-INDEX(1) TO <FS>.
ENDDO.
```

启动 SAPMZTST 之后，发生运行错误。系统反应如下：

DATA 存储区宽度为 25。最后的循环途径试图分配该范围之外的地址，因为偏移量是 25。直到第 24 循环途径，未发生错误。如果试图在 ASSIGN 语句中用 TEXT2 替换 TEXT1，将在第 15 循环中发生错误。

第十一章　读取并处理 数据库表

概览

内容

本节讲述下 列主题：

ABAP/4 中的 SQL 概念

ABAP/4 的开放式 SQL

ABAP/4 自身的 SQL

锁定和授权

> 在处理来自 数据库表的 大量数据时 ，程序的运 行时间就成 了影响性能 的重要因素 。有关如何 达到最优性 能的示例， 请选择 ABAP/4 开发工作台 中的“测试→ 实时分析” （或使用事 务 SE30） ，并选定“ 提示和策略 ”。在“SQL 界面”下， 将可找到关 于本节的适 当示例。

数据库表和 SQL 概念

在 R/3 系统中，需 长期保存的 数据都存储 在关系数据 库表中。关 于各种类型 的数据库表 以及如何创 建和维护它 们的信息， 参见文档 *ABAP/4 词典*（页 **Error! Not a valid link.**）。

结构化查询 语言 (SQL) 是为访问关 系数据库创 建的。SQL 有两个语句 类型：数据 定义语言 (DDL) 语句和数据 操作语言 (DML) 语句。

现在，SQL 还没有完全 标准化。要 访问指定的 数据库系统 ，必须查阅 该系统的文 档，以获取 可用的 SQL 语句及其正 确的语法的 列表。
要在 ABAP/4 程序中包含 这些 SQL 语句，请使 用自身的 SQL（参 见 *在 ABAP/4 程序中使用 本地的 SQL 语句* (页 51) ）。
为了避免不 同的数据库 表之间的不 兼容性，并 使 ABAP/4 程序能在应 用中独立于 数据库系统 ，SAP 创建了一套 叫作开放式 SQL 的独立 SQL 语句。开放 式 SQL 包含了一套 标准的 SQL 语句以及一 些专用于 SAP 的增强语句 。使用开放 式 SQL，可 以访问对 SAP 系统有效的 任何数据库 表，而不用 考虑它的创 建者。
下图显示了 开放式 SQL 与自身的 SQL 之间的区别 ：

数据库界面 可将 SAP 的开放式 SQL 语句翻译成 专用于现正 使用的数据 库的 SQL 命令。自身 的 SQL 直接访问数 据库。

开放式 SQL 关键字

关键字	用途
SELECT	*从数据库表 读取数据* (页 156)
INSERT	*向数据库表 添加行* (页 149)
UPDATE	*在数据库表 中更改行* (页 149)
MODIFY	*添加或更改 行* (页 149)
DELETE	*从数据库表 中删除行* (页 152)
OPEN CURSOR, FETCH, CLOSE CURSOR	*使用光标从 数据库表中 读取行* (页 144)
COMMIT WORK, ROLLBACK WORK	*确认或取消 对数据库表 的更改* (页 159)

在 ABAP/4 程序中使用 开放式 SQL 语句时，必 须保证：

1) 被定址的数 据库系统必 须是 SAP 支持的。
2) 必须已经在 ABAP/4 词典中定义 了被定址的 数据库表。

下列系统字 段在开放式 SQL 操作中起着 重要的作用 ：

- SY-SUBRC
  和使用其它 ABAP/4 语句一样， 系统字段 SY-SUBRC 中的返回代 码值表示在 每个开放式 SQL 操作之后该 操作是否成 功。如果操 作是成功的 ，SY-SUBRC 的值就等于 0；如果操 作是失败的 ，SY-SUBRC 的值就不等 于 0。
- SY-DBCNT
  SY-DBCNT 字段中的值 表明受该操 作影响的行 数，或已被 处理的行数 。

可以使用 ‘ SQL 跟踪’实用 程序监视 SQL 或 ABAP/4 功能的性能 。为此，请 选择“系统→ 实用程序→*SQL*跟 踪”。详细 信息，参见 文档 *ABAP/4 工作台工具* (页 **Error! Not a valid link.**) 。

从数据库表 读取数据

要从数据库 表读取数据 ，请使用 SELECT 语句。

语法

```
SELECT <result> FROM <source> [INTO <target>] [WHERE <condition>]
              [GROUP BY <fields>] [ORDER BY <sort_order>].
```

该语句有几 个基本子句 。下表中列 出了每一个 子句。

子句	说明
SELECT <result>	SELECT 子句定义选 择的结果是 单行还是一 个表、选择 的是哪些 列 、以及是否 将排除相同 的行。 *定义选择的 结果* (页 128)
FROM <source>	FROM 子句指定即 将从中选取 数据的数据 库表或视图 <source>。 *指 定将读取的 数据库表* (页 141)
INTO <target>	INTO 子句确定即 将读入选定 数据的目标 区 <target>。 该子句也可以放在FROM 子句之前。 如果没有指 定 INTO 子句，系统 将使用表工作区。表工 作区是由 TANLES 语句自动创 建的表头行 。 *为选定数据 指定目标区* (页 143)
WHERE <condition>	WHERE 子句指定将 按照指定的 条件读取哪 些行来作为 选择。 *选择即将读 取的行* (页 149)
GROUP BY <fields>	GROUP-BY 子句从几行 组成的组中 产生了作为 结果的单行 。一个组是 在 <fields> 中列出的列 中有相同值 的行的集合 。 *给行分组* (页 48)
ORDER BY <sort_order>	ORDER-BY 子句为选定 的行定义顺 序 <sort_order>。 *指定行的顺 序* (页 161)

关于 SELECT 语句及其子 句性能的重 要信息，参 见关键字文 档。

定义选择的 结果

SELECT 子句定义是 选择单行还 是选择多行 、是否去掉 重复行以及 将选择哪些 列。
图中显示了 可能的选择 ：

有关叙述 SELECT 子句的三个 变式的主题 ，参见

选择多行中 的所有数据

要从数据库 表中读取所 有列和多行 ，请按如下 方式在循环 中使用 SELECT 语句。

语法

```
SELECT [DISTINCT] * .....
  ....
ENDSELECT.
```

必须用 ENDSELECT 语句结束该 循环。
该循环依次 读取所选行 ，并为每个 读取的行执 行循环中的 ABAP/4 语句。SELECT 循环的结果 是与被读取 的数据库表 的格式完全 相同的表。
DISTINCT 选项将自动 的去掉重复 的行。

如果至少读 取了一行， 系统字段 SY-SUBRC 就返回 0。如果没 有读取，系 统字段 SY-SUBRC 就返回 4。系统字 段 SY-DBCNT 给读取的行 计数。每执 行一次 SELECT 语句，SY-DBCNT 都加 1。

```
TABLES SPFLI.

SELECT * FROM SPFLI WHERE CITYFROM EQ 'FRANKFURT'.
  ...
  WRITE: / SPFLI-CARRID, SPFLI-CONNID,
           SPFLI-CITYFROM, SPFLI-CITYTO.
  ...
ENDSELECT.
```

该 SELECT 循环从 SPFLI-CITYFROM 字段中包含 “FRANKFURT” 的 SPFLI 表中读取所 有的行（由 WHERE 子句指定） 。读取的每 一行的数据 都由循环中 的 WRITE 语句写入输 出列表中。

输出列表如 下所示：

选择单行中 的所有数据

要从数据库 表中读取单 个行的所有 列，请按如 下方式使用 SELECT 语句：

语法
```
SELECT SINGLE [FOR UPDATE] * ..... WHERE <condition> .....
```
该语句的结 果是一个单 行。为了保 证清楚地指 定了一行，就必须在 WHERE 子句的条件 <condition> 中用 AND 链接形成数 据库表主码 主码的所有 字段。（关 于 WHERE 子句的信息 ，参见 *选择即将读 取的行* （页 149） ）。
如果系统没 有找到具有 指定关键字 的行，系统 字段 SY-SUBRC 将设置为 4。如果系 统找到了一 个完全符合 指定条件的 行，SY-SUBRC 就返回 0。
可以使用 FOR UPDATE 选项锁定在 数据库表中 选定的行。 程序将一直 等到接收到 该锁定。如 果数据库检 测到或怀疑 有一个死锁 定，将产生 运行时间错 误。

FOR UPDATE 选项不是将 SAP 锁定机制与 ENQUEUE/DEQUEUE 功能模块一 起使用的替 代品。例如 ，显示一个 新的屏幕时 ，所有用 FOR UPDATE 锁定的行都 将自动解锁 。要保证在 显示新屏幕 时锁定的行 保留锁定状 态，就必须 使用 SAP 锁定机制。 这是使锁定 的行一直到 事务的结束 都保留其锁 定状态的唯 一方法。详 细信息，参 见 *在 ABAP/4 程序的执行 过程中锁定 数据库对象* （页 157）

```
TABLES SPFLI.

SELECT SINGLE * FROM SPFLI WHERE CARRID EQ 'LH'
                             AND CONNID EQ '2407'.
WRITE: / SPFLI-CARRID, SPFLI-CONNID,
         SPFLI-CITYFROM, SPFLI-CITYTO.
```

该 SELECT 语句只从 SPFLI 中读取 CARRID 字段中包含 “LH”并 且 CONNID 字段中包含 “2407” 的行。

输出屏幕如 下所示

选择并处理 指定列中数 据
要读取显式 地包含规定 的列的行， 或要得到关 于数据库表 特定列的摘 要信息，请 按如下方法 一起使用 SELECT 语句与列表 ：

语法
SELECT [SINGLE [FOR UPDATE]] [DISTINCT] <s_1> <s_2> .....
其中每个 <s_i> 都具有下列 形式之一

- <a_i>
  <a_i> 是数据库表 的字段或表 单的总计表 达式：
  <aggregate>([DISTINCT] <a>)
  关于总计表 达式的说明 ，参见下列 内容。
- <a_i> AS <b_i>
  <b_i> 是结构化目 标区的第 i 个组件的可 选名称。可 使用该可选 名称将读取 或处理指定 行的结果写 到目标区的 组件 <b_i> 中。为此， 还必须使用 INTO 子句的 CORRESPONDING FIELDS 选项（关于 该选项的详 细信息及示 例，参见 *逐个组件地 读取数据* (页 58) ）。

DISTINCT 选项将自动 地去掉重复 行。
如果指定了 SINGLE 选项，选择 的结果将由 一单行的列 <a_1> <a_2>... 组成。将如 *选择单行中 的所有数据* (页 166) 中的描述选 择该行。
可按如下方 法书写 SELECT 语句以在运 行时指定列 ：

语法
SELECT [SINGLE [FOR UPDATE]] [DISTINCT] (<itab>).
如果内表 <itab> 包含列表 <s_1> <s_2> ....， 该语句的操 作与上述情 况一样。因 此，内表 <itab> 的行类型必 须是最大长 度为 72，类型 为 C 的字段。如 果内表为空 ，那么，该 语句就按照 指定的是星号 (*) 而 不是 <itab> 来进行操作 。

总计表达式
使用总计表 达式，可从 数据库表的 列 <a> 中摘录特征 数据。有效 的总计表达 式是：

- MAX： 返回列 <a> 的最大值
- MIN： 返回列 <a> 的最小值
- AVG： 返回列 <a> 的平均值
- SUM： 返回列 <a> 的总计
- COUNT： 按下列方式 给值或行计 数：
  - COUNT(DISTINCT <a>) 返回列 <a> 的不同值的 个数。
  - COUNT(*) 返回选定行 的总数。

在括号和参 数之间必须 包含空格。 算术操作符 AVG 和 SUM 只能对数字 字段进行操 作。

SELECT 子句和 INTO 子句中的列 表
如果在 SELECT 子句中有一 个列表，就 必须将 INTO 子句与 SELECT 语句一起使 用。和相关 主题中叙述 的一样，可 以将工作区 <wa> 或内表 <itab> 用作参数。 注意，如果 SELECT 子句包含了 一个列表，那么就根据 工作区 <wa> 或内表 <itab> 的结构将选 择的数据从 左至右输出 到目标区中 。这是一个 例外。通常 情况下，选 择的数据是 根据表工作 区的结构从 左至右输出 到目标区中 ，而不考虑 目标区的结 构（关于 INTO 子句的详细 信息，参见 *为选定数据 指定目标区* (页 143))。
SELECT 子句中有一 个列表时， 可在 INTO 子句中将列 表 <F1>, <F2>, ... 用作参数：
SELECT <a_1> <a_2> .... INTO (<F1>, <F2>, ...).
SELECT 子句和 INTO 子句中的列 表必须包含 相同数目的 元素。如果 可能，在传 输数据过程 中，应将值 转换成目标 字段的数据 类型（数据 库表和 ABAP/4 数据类型之 间可转换性 的信息，参 见有关 INTO 子句的关键 字文档)。 如果在 SELECT 子句的列表 中除总计表 达式以外还 有一个或多 个数据库字 段，就必须 在 GROUP-BY 子句中列出 所有数据库 字段（详细 信息，参见 *给行分组* (页 48))。

```
TABLES SPFLI.

DATA: LIST(72) OCCURS 1,
      LINE(72).

LINE = ' CITYFROM CITYTO '.
APPEND LINE TO LIST.

SELECT DISTINCT (LIST)
       INTO CORRESPONDING FIELDS OF SPFLI FROM SPFLI.
   WRITE: / SPFLI-CITYFROM, SPFLI-CITYTO.
ENDSELECT.
```

在该示例中 ，将列 CITYFROM 和 CITYTO 写到内表 LIST 中，并且只 从 SPFLI 中选择这些 列。选择结 果将写到表 工作区 SPFLI 中。关于 INTO 子句的 CORRESPONDING FIELDS 选项的信息 ，参见 *逐个组件地 读取数据* (页 58)。因为使用 了 DISRINCT 选项，所以 将去掉重复 的行。

假设下列数 据库表 TEST 由 10 行组成：

COL_1	COL_2
1	3
2	1
3	5
4	7
5	2
6	3
7	1
8	9
9	4
10	3

要摘录并处 理列数据， 可使用下列 程序：

```
TABLES TEST.
DATA RESULT TYPE P DECIMALS 2.
SELECT <agg>( [DISTINCT] COL_2 ) INTO RESULT FROM TEST.
WRITE RESULT.
```

下表显示了 根据总计表 达式 <agg> 和 DISTINCT 选项的不同 组合所得到 的该程序的结果。

使用的总计 表达式	包含 的DISTINCT	选项结果
MAX	no	9.00
MAX	yes	9.00
MIN	no	1.00
MIN	yes	1.00
AVG	no	3.80
AVG	yes	4.43
SUM	no	38.00
SUM	yes	31.00
COUNT	yes	7.00
COUNT(*)	---	10.00

指 定将读取的 数据库表

要指定将用 SELECT 语句读取的 数据库表或 视图，可使 用 FROM 子句。
参见

在程序中指 定数据库表 名
要在程序中 指定即将读 取的数据库 表或视图， 请按下列格 式使用 FROM 子句：

语法
```
..... FROM <dbtab> [CLIENT SPECIFIED] [BYPASSING BUFFER]
                   [UP TO <n> ROWS].........
```
数据库表或 视图 <dbtab> 必须对 ABAP/4 词典有效， 并且必须在 ABAP/4 程序中包含 相应的 TRABLES 语句。
要关闭自动 集团处理（ 关于集团处 理的详细信 息，参见 *为数据库表 处理指定集 团* (页 51)），请使用 CLIENT SPECIFIED 选项。
BYPASSING BUFFER 选项使得不 用读取 SAP 表缓冲区就 可直接读取 数据库。

如果要保证 操作的是数 据库表的最 新版本，该 选项就是非 常重要的。

在 ABAP/4 词典中定义 一个表时， 可以指定 SAP 必须将自己 的本地缓冲 区用于该表 。该缓冲区 被异步更新 。由于 SELECT 语句通常使 用该缓冲区 ，所以没有 必要使用数 据库的最新 版本。要保 证使用的是 最新版本， 请使用 BYPASSING BUFFER 选项。
如果最多只 需从数据库 表 <dbtab> 中读取 <n> 行，请使用 可选规范 UP TO <n> ROWS。 如果 <n> = 0，系统将 读取所有行 ，如果 <n> 小于 0，将产生 运行时间错 误。
如果将 UP TO <n> ROWS 选项与 ORDER BY 子句组合起 来，系统将 首先给这些 行排序，然 后再处理前 <n> 行（关于 ORDER BY 子句的详细 信息，参见 *指定行的顺 序* (页 161))。

```
TABLES SPFLI.

SELECT * FROM SPFLI.
   .....
ENDSELECT.
```
在该示例中 ，将从数据 库表 SPFLI 中读取所有 行。

在运行时指 定数据库表 的名称
可以在运行 时指定数据 库表的名称 。为此，请 按如下格式 使用 FROM 子句：

语法
```
......FROM (<dbtabname>) [CLIENT SPECIFIED] [BYPASSING BUFFER]
                         [UP TO <n> ROWS].. INTO <target> .....
```
该格式的 FROM 子句只能与 INTO 子句一起使 用（参见 *为选定数据 指定目标区* (页 143))。
字段 <dbtabname> 的内容确定 了数据库表 的名称。在 该情况下， 程序不必包 含 TABLES 语句。选项 CLIENT SPECIFIED、 BYPASSING BUFFER 和 UP TO <n> ROWS 同样是用于 在程序中指 定数据库表 的名称（参 见 *在程序中指 定数据库表 名* (页 168))。

```
DATA: BEGIN OF WA,
        LINE(240),
      END OF WA.

DATA NAME(10).

NAME = 'SPFLI'.

SELECT * FROM (NAME) INTO WA.
  WRITE: / WA-LINE.
ENDSELECT.
```
数据库表名 称 SPFLI 被赋给字符 字段 NAME。 SELECT 语句将所有 的行从 SPFLI 中读到目标 区 WA 中。在该示 例中，WA 与 SPFLI 的结构并不 相同，每一 行都将自动 地转换成字 符字段（详 细信息，参 见 *为选定数据 指定目标区* (页 143) 和 *Convertibility of Field Strings* (页 6-41)) 。

因为系统在 ABAP/4 词典中找不 到数据库表 SPFLI， 所以不能用 **NAME = 'spfli'** 代替 **NAME = 'SPFLI'**。

输出列表的 如下部分表 明将数据写 入文本字段 错误地解释 了一些列：

为选定数据 指定目标区

要为选定数 据指定目标 区，请使用 SELECT 语句的 INTO 子句。只有 在需要指定 一个与表工 作区不同的 目标区时才 需要使用该 子句。表工 作区通常由 TABLES 语句自动生 成。如果需 要使用数据 库光标从数 据库表中读 取行，也可 在 FRTCH 语句中指定 INTO 子句(参见 *使用光标从 数据库表中 读取行* (页 144))。
INTO 子句有三个 主要变式。其中两个用 于将数据读 到工作区中 ，而另一个 用于将数据 读到内表中 。这些变式 将在下列主 题中说明。
参见
第三个变式 用于将数据 读到列表中 。它只能与 SELECT 子句中的列 表一起使用 ，这些将在 *选择并处理 指定列中数 据* (页 166) 中进行描述 。如果现在 是将 INTO 子句与 SELECT 子句的列表 一起使用，那么将根据 目标区结构 将选择的列 从左至右输 出。
在 SELECT 语句中没有 列表（SELECT *）的情况 下，所选数 据将从左至 右输出到目 标区中。系 统根据表工 作区的结构 写入数据， 而不考虑目 标区的结构 。要保证能 访问数据库 表单独的列 ，就只能使 用与数据库 表具有相同 结构的目标 区。目标区 的大小至少 要能容下将 读取的行。

ABAP/4 词典中的数 据类型与 ABAP/4 编程语言中 的数据类型 不同。如果 在 SELECT 子句中指定 一个列表， 或在 INTO 子句中使用 CORRESPONDING FIELDS 选项，那么 数据库表的 ABAP/4 词典字段必 须能够转换 成 ABAP/4 编程语言的 目标字段。 有关可转换 的数据类型 的列表，参 见 INTO 子句的关键 字文档。

将数据读到 工作区中
可以将数据 从数据库表 读到与 TABLES 语句中定义 的默认工作 区不同的工 作区(通常 是字段串) 中。为此，请按 照如下格式 在 SELECT 语句的 INTO 子句中指定 工作区：

语法
```
SELECT ... INTO <wa> .............
```
必须为工作 区〈wa〉声明一个至 少与将读取 的行一样大 的数据对象 。

```
TABLES SPFLI.

DATA WA LIKE SPFLI.

SELECT * FROM SPFLI INTO WA.
   WRITE: / WA-CITYFROM, WA-CITYTO.
ENDSELECT.
```

在该示例中 ，因为工作 区 WA 的数据类型 是由 DATA 语句中的 LIKE SPFLI 定义的，所 以它有与数 据库表 SPFLI 相同的结构 。因为在 SELECT 循环中使用 了 INTO 子句，所以 将填充工作 区 WA，而不 是 TABLES 语句中所指 定的标准工 作区 SPFLI。SPFLI 的所有字段 ，即 SPFLI-CITYFROM 和 SPFLI-CITYTO， 都保留为空 。

将数据读到 内表中
可以在单个 操作中将数 据库表中行 选择的结果 集写入内表 中。
为此，请按 照如下格式 在 SELECT 语句的 INTO 子句中指定 内表：

语法
```
SELECT ..... INTO TABLE <itab>.
```
在该情况下 ，SELECT 并不启动循 环，并且不 允许使用 ENDSELECT 语句。
如果内表〈itab〉不是空的， 那么，SELECT 语句将用读 取的数据覆 盖其中的内 容。

```
TABLES SPFLI.
```

```
DATA ITAB LIKE SPFLI OCCURS 10 WITH HEADER LINE.

SELECT * FROM SPFLI INTO TABLE ITAB
                   WHERE CARRID = 'LH'.

LOOP AT ITAB.
   WRITE: / ITAB-CONNID, ITAB-CARRID.
ENDLOOP.
```

在该示例中 ，数据库表 SPFLI（ 在该数据库 表中 CARRID 字段包含“ LH”）的 所有行都将 读到内表 ITAB 中，并在那 里进一步处 理。

将内表指定 为选定行的 目标区时， 可以在包中 处理这些行 或将这些行 附加到内表 中，而不覆 盖表中的内 容。这些选 项的描述如 下。

在包中处理 这些行
如果需要将 所选行按已 预定义大小 的包的形式 读到内表中 ，请按如下 格式使用 INTO 子句的 PACKAGE SIZE 选项：

语法
SELECT * ..... INTO TABLE <itab> PACKAGE SIZE <n>.....
该语句打开 一个循环。 必须使用 ENDSELECT 语句结束该 循环。对于 读取的每个 包含 <n> 行的包，系统都将执行 一次循环。 如果 <n> 小于或等于 零，将出现 运行时间错 误。

在 SELECT 循环外，无 法确定内表 中的内容。 所以如果希 望进一步处 理所选行， 就必须在循 环中编写相 应的 ABAP/4 语句。

```
TABLES SPFLI.

DATA ITAB LIKE SPFLI OCCURS 5 WITH HEADER LINE.

SELECT * FROM SPFLI INTO TABLE ITAB PACKAGE SIZE 5
                    WHERE CARRID = 'LH'.

   LOOP AT ITAB.
      WRITE: / ITAB-CARRID, ITAB-CONNID.
   ENDLOOP.

   SKIP 1.

ENDSELECT.
```

在该示例中 ，数据库表 SPFLI （在该数据 库表中 CARRID 字段包含“ LH”）的 所有行都将 按大小为 5 行的包读到 内表 ITAB 中。在 SELECT 循环内，又 有一个循环 将这些包写 入输出列表 。输出列表 如下所示：

将行附加到 内表中
为了避免覆 盖内表中的 内容，可将 所选行附加 到此表中。 为此，请按 如下格式使 用 APPENDING 子句，而不 是 INTO 子句：

语法
SELECT ..... APPENDING TABLE <itab>.....
与上面描述 的 INTO 子句的唯一 区别是它是 将行附加到 内表 <itab> 中，而不是 其内容。也 可以在该语 句中使用 PACKAGE SIZE 选项。

逐个组件地 读取数据
要一个组件 接一个组件 地将数据读 到目标区中 ，请使用 INTO 子句的 CORRESPONDING FIELDS 选项。语法 如下所示：

语法
对于将数据 读到工作区 中：
```
SELECT ... INTO CORRESPONDING FIELDS OF <wa> ........
```
对于将数据 读到内表中 ：
```
SELECT ... INTO CORRESPONDING FIELDS OF TABLE <itab> ........
```
对于将数据 附加到内表 中：
```
SELECT ... APPENDING CORRESPONDING FIELDS OF TABLE <itab> ........
```
这些语句不 会将所选行 的所有字段 放进目标区 中。系统只 将列的内容 （对于该列 ，目标区中 具有同名组 件）传送到 目标区的对 应组件中。 如果可能， 在传送过程 中，可将值 转换成目标 字段的数据 类型（关于 数据库表和 ABAP/4 数据类型之 间的可转换 性的信息， 参见 INTO 子句的关键 字文档）。

```
TABLES SPFLI.

DATA: BEGIN OF WA,
        NUMBER TYPE I VALUE 1,
        CITYFROM LIKE SPFLI-CITYFROM,
        CITYTO   LIKE SPFLI-CITYTO,
      END OF WA.

SELECT * FROM SPFLI INTO CORRESPONDING FIELDS OF WA.
   WRITE: / WA-NUMBER, WA-CITYFROM, WA-CITYTO.
ENDSELECT.
```

输出如下所 示：

在该示例中 ，系统只将 数据库表 SPFLI 中选定行的 列 CITYFROM 和 CITYTO 传送到 WA 中。WA 中的组件 NUMBER 保持不变。

```
TABLES SBOOK.

DATA: BEGIN OF LUGGAGE,
        AVERAGE TYPE P DECIMALS 2,
        SUM     TYPE P DECIMALS 2,
      END OF LUGGAGE.

SELECT AVG( LUGGWEIGHT ) AS AVERAGE
       SUM( LUGGWEIGHT ) AS SUM
       INTO CORRESPONDING FIELDS OF LUGGAGE FROM SBOOK.

WRITE: / 'Average:', LUGGAGE-AVERAGE, / 'Sum:', LUGGAGE-SUM.
```

该示例为数 据库表 SBOOK 中的所有行 计算了字段 LUGGWEIGHT 的平均值和 总和。总计 表达式 AVG 和 SUM 的结果用选 择性的名称 写入结构 LUGGAGE 的组件 AVERAGE 和 SUM 中。关于总 计表达式和 可选名称的 详细信息， 参见 *选择并处理 指定列中数 据* （页 166）。

选择即将读 取的行

如果只希望 访问数据库 表中的符合 某些条件的 行，请使用 SELECT 语句中的 WHERE 子句。
参见

该主题中说 明的符合开 放式 SQL 的 WHERE 子句不仅用 在 SELECT 语句中，还 用在 UPDATE、 MODIFY 和 DELETE 语句中。

在程序中为 行选择指定 条件
要在程序中 为行选择指 定条件，请 按如下格式 使用 WHERE 子句：

语法
..... WHERE <condition> ....

基本 WHERE 条件
有六个基本 条件可用于 限制行选择 。描述如下 ：

1. <f> <operator> <g>

<f> 是不带作为 前缀的表名 称的数据库 字段名（数 据库表的列 ），<g> 是任意字段 或字母。字 段名称和操 作符必须用 空格隔开。
对于 <operator>， 可使用下列 字符或字符 串：

<operator>	含 义
EQ	等于
=	等于
NE	不 等于
<>	不 等于
><	不 等于
LT	小 于
<	小于
LE	小 于或等于
<=	小 于或等于
GT	大 于
>	大于
GE	大 于或等于
>=	大 于或等于

```
........ WHERE CARRID = 'UA'.
选 定字段 CARRID 的值为“UA” 的所有行。

........ WHERE NUM GE 15.
选定 字段 NUM 中的数字大 于或等于 15 的所有行。

........ WHERE CITYFROM NE 'FRANKFURT'.
选 定字段 CITYFROM 中的字符串 不等于“FRANKFURT” 的所有行。
```

2. <f> NOT BETWEEN <g₁> AND <g₂>

数据库字段 <f> 的值必须（ 不能）处于 字段或字母 <g₁> 和 <g₂> 的值之间才 符合该条件 。

```
........ WHERE NUM BETWEEN 15 AND 45.
选定 字段 NUM 中的数字位 于 15 和 45 之间的所有 行。

........ WHERE NUM NOT BETWEEN 1 AND 99.
选定 字段 NUM 中的数字不 位于 1 和 99 之间的所有 行。

........ WHERE NAME NOT BETWEEN 'A' AND 'H'.
选 定字段 NAME 中的字符串 的字母顺序 不位于“A” 和“H”之 间的所有行 。
```

3. <f> NOT LIKE <g> [ESCAPE <h>]

该条件只能 用于字符类 型字段。

要符合该条 件，数据库 字段 <f> 中的值必须 （不能）符 合 <g> 的模式。在 指定 <g> 时，可使用 下列两个通 配符：

_ （下 划线）表示 单个字符

% （ 百分号）表 示任意字符 串，包括空 字符串

例如，ABC_EFG% 能与字符串 ABCxEFGxyz 和 ABCxEFG 匹配，但不 能与 ABCEFGxyz 匹配。

```
..... WHERE  CITY LIKE '%town%'.
选定所有包 含“town” 的城市名。
..... WHERE  NAME NOT LIKE '_n%'.
仅选定第二 个字母不为 “n”的名 称。
```

如果在比较 中要用到两 个通配符， 可使用 ESCAPE 选项。ESCAPE <h> 指定一个忽 略符号 <h>。如 果通配符前 面有 <h>，那 么通配符和 忽略符号本 身都失去了 它在模式 <g> 中的常用功 能。

```
..... WHERE  FUNCNAME LIKE 'EDIT#_%' ESCAPE '#'.
选定所有以 “EDIT_” 开始的功能 名称。
```

```
在 LIKE 条件中使用 通配符 _ 和 % 也符合 SQL 标准。但是 ， ABAP/4 比较操作符 CP 和 NP 将识别不同 的通配符（ + 和 *）（关于 通配符的详 细信息，参 见比较字符串和数字串 (页 7 - 6) ）。
```

4. <f> NOT IN (<g₁>, ..., <gₙ>)

要满足该条 件，数据库 字段 <f> 中的值必须 （不能）等 于括号内列 表中的一个 值。

```
在该变式中 ，在括号和 比较字段 <gᵢ> 之间不能有 空格，但在 比较字段之 间可以有空 格。
```

```
..... WHERE CITY IN ('Berlin', 'New York', 'London').
选定城市“ Berlin” 、“New York” 和“London” 。
..... WHERE CITY NOT IN ('Frankfurt', 'Rome').
选定除“Frankfurt” 和“Rome” 以外的城市 。
```

5. <f> IS [NOT] NULL

数据库字段 <f> 中的值必须 （不能）等 于 NULL 值。

6. <f> NOT IN <seltab>

要满足该条 件，数据库 字段 <f> 的值必须（ 不能）符合 选择表 <seltab> 中指定的条 件。选择表 是一个特殊 的内表，报 表用户可在 选择屏幕上 填充它。通 常是使用 SELECT-OPTIONS 或 RANGES 来创建

选择 表，但是也 可以按照 *创建和处理 内表* （页 8－1）中的说明定 义。关于选 择表的结构 和示例，参 见 *使用选择标 准* （页 Error! Not a valid link.）。

使用逻辑链 接操作符组 合条件
可以使用逻 辑链接操作 符 AND、OR 和 NOT 按照任意顺 序来组合六 个基本 WHERE 条件。
如果希望指 定几个必须 同时满足的 条件，可按 照下列方法 用 AND 组合它们：
....WHERE <condition$_1$> AND <condition$_2$> AND <condition$_3$> AND...
如果需要指 定几个条件 ，至少要满 足其中的一 个条件，就 可按照下列 方法用 OR 组合它们：
....WHERE <condition$_1$> OR <condition$_2$> OR <condition$_3$> OR...
如果只希望 选择那些不 符合指定条 件的表条目 ，就可用 NOT 转化条件， 如下所示：
....WHERE NOT <condition>
NOT 的优先级比 AND 高，AND 的优先级比 OR 高。但是， 可使用括号 来定义处理 的顺序。这 些括号都必 须加空格。

```
.......WHERE ( NUMBER = '0001' OR NUMBER = '0002' ) AND
                    NOT ( COUNTRY = 'F' OR COUNTRY = 'USA' ).
```

在该示例中 ，只选择那 些 NUMBER 字段中包含 “0001” 或“0002” 并且 COUNTRY 字段中不包 含“F”或 “USA” 的行。

运行时指定 行选择的条 件
可在运行时 为行选择指 定完整的条 件或部分条 件。

由于系统只 有在运行时 才能执行语 法检查或生 成内部控制 块，因此运 行时指定条件比在程序 中指定它们 需要花费更 多的 CPU 时间。并且 ，动态 WHERE 条件只能用 于 SELECT 语句。

在运行时指 定完整的条 件

语法
SELECT.....WHERE (<itab>) .....
只能在仅包 含一个类型 为 C 且最大长度 为 72 的字段的内 表 <itab> 中指定条件 。表名称必 须在括号中 指定，但在 括号和名称 之间没有空 格。
可使用 *在程序中为 行选择指定 条件* （页 172）一节中叙述 的语法将本 节中叙述的 所有条件写 到 <itab> 中。但是也 有下列例外 ：
- 使用字母 。不能使用 变量。
- 不能将操 作符 IN 与选择表一 起使用。

内表也可保 持为空。

```
TABLES SPFLI.

DATA ITAB(72) OCCURS 10 WITH HEADER LINE.

PARAMETERS: CITY1(10) TYPE C, CITY2(10) TYPE C.

CONCATENATE    'CITYFROM = ''' CITY1 '''' INTO ITAB.
APPEND ITAB.

CONCATENATE 'OR CITYFROM = ''' CITY2 '''' INTO ITAB.
APPEND ITAB.

CONCATENATE 'OR CITYFROM = ''' 'BERLIN' '''' INTO ITAB.
APPEND ITAB.

LOOP AT ITAB.
   WRITE ITAB.
ENDLOOP.

SKIP.
```

```
SELECT * FROM SPFLI WHERE (ITAB).
   WRITE / SPFLI-CITYFROM.
ENDSELECT.
```

在启动该程 序时，PARAMETERS 语句将显示 一个选择屏 幕。这将在 *为变量了_ 输入字 _*（页 Error! Not a valid link.）中详细说明 。

在该示例中 ，将在选择 屏幕上要求 用户输入参 数 CITY1 和 CITY2。 假设该程序 的用户按下 列形式填写 了选择屏幕 的输入区域 ：

那么将在屏 幕上产生下 列输出：

前三行显示 内表 ITAB 的内容。输 出列表表明 仅选定了那 些 CITYFROM 字段中包含 “FRANKFURT” 、“NEW YORK” 或“BERLIN” 的行。

在运行时只 指定条件的 一部分
要在运行时 指定 SELECT 语句中的条 件的一部分 ，请按下列 格式使用 WHERE 子句：

语法
SELECT......WHERE <condition> AND (<itab>) ....
按照 *在程序中为 行选择指定 条件*（页 172）中的说明在 程序中指定 条件 <condition>。
必须用 AND 将在运行时 希望在 <itab> 中指定的部 分条件附加 在 <condition> 之后。
可按照上述 方法指定内 表 <itab>。

指定条件的 列表
要在运行时 指定一系列 条件来选择 一定数目的 特定行，请 在 SELECT 语句中使用 下列 WHERE 子句的特殊 变式：

语法
SELECT......FOR ALL ENTRIES IN <itab> WHERE <condition> .....
在条件 <condition> 中，可按前 面的叙述将 内部字段或 字母指定为 比较值。也 可以将内表 <itab> 的列或字母 用作比较值 。在 WHERE 条件中，这 些列将用作 占位符。
该 SELECT 语句的结果 集是 SELECT 语句的所有 结果集的联 合，这些结 果集是用 <itab> 中的相应值在每一行上 替换占位符 的结果。
将从结果集 中删除重复 行。

数据库字段 与内表中的 关联比较字 段必须具有 相同的类型 和长度。

不要在数据 库字段和表 字段之间的 比较中使用 操作符 LIKE、 BETWEEN 和 IN。

如果使用了 WHERE 子句的该变 式，就不要 使用 ORDER BY 子句。

```
TABLES SPFLI.

DATA: BEGIN OF ITAB OCCURS 10,
        CITYFROM LIKE SPFLI-CITYFROM,
        CITYTO   LIKE SPFLI-CITYTO,
      END OF ITAB.

ITAB-CITYFROM = 'FRANKFURT'.
ITAB-CITYTO   = 'BERLIN'
APPEND ITAB.

ITAB-CITYFROM = 'NEW YORK'.
ITAB-CITYTO   = 'SAN FRANCISCO'.
APPEND ITAB.
```

```
SELECT * FROM SPFLI FOR ALL ENTRIES IN ITAB WHERE
                                  CITYFROM = ITAB-CITYFROM AND
                                  CITYTO = ITAB-CITYTO.
   WRITE: / SPFLI-CITYFROM, SPFLI-CITYTO.
ENDSELECT.
```

该示例从 SPFLI 中选择下列 行的联合

- CITYFROM 字段中包含 “FRANKFURT” ，并且 CITYTO 字段中包含 “BERLIN” 。
- CITYFROM 字段中包含 “NEW YORK” ，并且 CITYTO 字段中包含 “SAN FRANCISCO” 。

给行分组

要将数据库 表的一组行 中的内容组 合到单个行 中，请按如 下格式使用 SELECT 语句的 GROUP BY 子句：

语法

```
SELECT  [DISTINCT] <a1> <a2> ..
        FROM clause INTO clause  GROUP BY <F1> <F2> ....
```

该组由 <F1> <F2> ..... 指定的列中 包含有相同 值的行组成 。
按如下格式 编写 SELECT 语句就可在 运行时指定 这些列：

语法

```
SELECT  [DISTINCT] <a1> <a2> ...
        FROM clause INTO clause  GROUP BY (<itab>)
```

如果内表 <itab> 包含有列表 <F1> <F2> ...，那 么该语句与 上述语句的 功能相同。 因此，<itab> 的行类型必 须是 C 类型，并且 最大长度为 72 的字段。

只有在 SELECT 子句中使用 了一个列表 时 GROUP BY <F1> <F2> ... 子句才有效 （参见 *选择并处理 指定列中数 据* (页 166) ）。

如果将总计 功能与 GROUP BY 子句一起使 用，那么在 SELECT 子句列表中 总计表达式以外的所有 数据库字段 也必须在GROUP BY 子句中出现 。

```
TABLES SFLIGHT.

DATA CARRID LIKE SFLIGHT-CARRID.

DATA: MINIMUM TYPE P DECIMALS 2, MAXIMUM TYPE P DECIMALS 2.

SELECT  CARRID MIN( PRICE ) MAX( PRICE )
  INTO (CARRID, MINIMUM, MAXIMUM) FROM SFLIGHT
                                  GROUP BY CARRID.

WRITE: / CARRID, MINIMUM, MAXIMUM.

ENDSELECT.
```

在该示例中 ，将根据 CARRID 字段中的内 容给数据库 表 SFLIGHT 分组，并且 为每组行返 回 PRICE 字段中的最 小值和最大 值，如下所 示：

指定行的顺 序

在读取行的 集合时，可 使用 SELECT 语句的 ORDER BY 子句指定这 些行的顺序 ，按该顺序 将这些行传送给 ABAP/4 程序。
如果不使用 ORDER BY 子句，将不 定义选择的 行的顺序。
可根据主码 或明确指定 的字段给这 些行排序。

按主码排序
要按主码给 所选行排序 ，请使用下 列形式的 ORDER BY 子句：

语法
```
SELECT * ..... ORDER BY PRIMARY KEY.
```
如果使用 ORDER BY PRIMARY KEY 选项，系统 将按主码以 升序给所选 行排序。

按指定的字 段排序
可按字段而 不是主码给 所选行排序 。为此，请 使用下列语 法：

语法
```
.. ORDER BY <F1> [ASCENDING|DESCENDING] <F2> [ASCENDING|DESCENDING] ...
```
按指定的表 字段 <F1>, <F2>, ... 给所选行排 序。可以在 每个字段名 后指定选项 ASCENDING 或 DESCENDING 为每个表字 段明确地指 定排序顺序 。标准的排 序顺序是升 序。
如果指定了 多个字段， 那么系统首 先按 <F1> 给所选行排 序，然后是 <F2>， 依此类推。
如下格式书 写 ORDER BY 子句，也可 在运行时指 定字段：

语法
```
.. ORDER BY (<itab>)
```
如果内表包 含列表 <F1>[ASCENDING|DESCENDING]<F2>...，那 么该语句的 功能与前面 说明的语句 一样。这里 ，<itab> 的行必须是 C 类且最大长 度为 72。

```
TABLES SPFLI.

SELECT * FROM SPFLI ORDER BY CITYFROM ASCENDING
                             CITYTO   DESCENDING.
WRITE: / SPFLI-CITYFROM, SPFLI-CITYTO.
ENDSELECT.
```

在该示例中 ，首先根据 字段 CITYFROM 中的内容按 升序给所选 行排序，然 后根据字段 CITYTO 中的内容按 降序给所选 行排序：

更改数据库 表的内容

可以使用下 列操作更改 数据库表的 内容：

操作	ABAP/4 关键字
添加 行	INSERT 可使用 INSERT 语句添加新 行，即添加 主码在数据 库表中尚不 存在的行。
更改行	UPDATE 可使用 UPDATE 语句更改已 经在数据库 表中存在的 行，即更改 主码已在数 据库中存在 的行。
添加或更改 行	MODIFY 如果不存在 带有即将添 加的主码的 行，请使用 MODIFY 语句添加新 行。否则， 更改现存的 行。
删除行	DELETE 可使用 DELETE 语句从数据 库表中删除 行。

向数据库表 添加行

要向数据库 表添加新行 ，请使用 INSERT 语句。INSERT 语句允许插 入特定内表 中的一行或 几行。
如果因为不 知道要插入 行的主码是 否已经存在 ，从而不能 确定是否可 使用 INSERT 语句，那么 ，请使用 MODIFY 语句。

如果视图指 向单个表， 那么就只能 用视图在数 据库表中插 入行。视图 的维护状态 必须在 ABAP/4 词典中定义 成“无限制 ”。
参见

添加一单行
要在数据库 表中添加一 单行，可使 用下列 INSERT 语句的任一 变式。

基本格式
INSERT 语句的基本 格式如下所 示：

语法
INSERT INTO <dbtab> [CLIENT SPECIFIED] VALUES <wa>.
工作区 <wa> 中的内容将 写进数据库 表 <dbtab> 中。必须在 程序中使用 TABLES 语句声明该 数据库表。工作区 <wa> 的长度至少 要等于数据 库表的表工 作区长度。 为了保证工 作区具有与 数据库表相 同的结构， 可通过 DATA 或 TYPES 语句用 LIKE <dbtab> 选项来定义 它（参见 *DATA 语句的基本格式* (页 3 - 14) ）。
如果数据库 表中没有哪 一行的主码 与工作区中 指定的相同 ，那么该操 作就成功地 完成了，并 且 SY-SUBRC 将设置为 0。否则， 该参数将设 置为 4。
CLIENT SPECIFIED 选项用来关 闭自动集团 处理（关于 集团处理的 详细信息， 参见 *为数据库表 处理指定集 团* (页 51) ）。
如果需要在 运行时指定 数据库表名 ，请使用下 列语法：
INSERT INTO (<dbtabname>) [CLIENT SPECIFIED] VALUES <wa>.
可按照 *指 定将读取的 数据库表* (页 168) 中对 SELECT 语句的说明 指定数据库 表（<dbtabname>） 。

短格式
也可用下列 短格式代替 上述语法：

语法
INSERT <dbtab> [CLIENT SPECIFIED] FROM <wa>.
或者如果需 要在运行时 间指定数据 库表名：
INSERT (<dbtabname>) [CLIENT SPECIFIED] FROM <wa>.
该语句与基 本格式的结 果相同。
一个更短的 格式是：

语法
INSERT <dbtab> [CLIENT SPECIFIED].
在这种情况 下，没有指 定工作区 <wa>。 相反，而是 将表工作区 <dbtab> 中的内容写 进数据库表 中。如果在运行 时间指定了 数据库表名 ，那么，就 必须使用选 项 FROM <wa>。 使用表工作 区允许。

```
TABLES SPFLI.

DATA WA LIKE SPFLI.

WA-CARRID = 'LH'.
WA-CITYFROM = 'WASHINGTON'.
  .....

INSERT INTO SPFLI VALUES WA.

WA-CARRID = 'UA'.
WA-CITYFROM = 'LONDON'.
  .....

INSERT SPFLI FROM WA.

SPFLI-CARRID = 'LH'.
SPFLI-CITYFROM = 'BERLIN'.
  .....

INSERT SPFLI.
```

在该示例中 ，用与数据 库表 SPFLI 相同的结构 定义了工作 区 WA。将值 赋给工作区 WA，并且 使用 INSERT 语句的三种 可能的变式 将工作区插 入到数据库 表 SPFLI 中。

除了在最后 一行中使用 **INSERT SPFLI** 以外，还可 使用更长的 格式 **INSERT SPFLI FROM SPFLI** 或 **INSERT INTO SPFLI VALUES SPFLI.**

从内表中添 加几行
要使用 INSERT 语句将几行 从内表中添 加到数据库 表中，请使 用下列语法 ：

语法
```
INSERT <dbtab> [CLIENT SPECIFIED] FROM TABLE <itab>
[ACCEPTING DUPLICATE KEYS].
```
如果需要在 运行时间指 定数据库表 名，请使用 下列语法：
```
INSERT (<dbtabname>) [CLIENT SPECIFIED] FROM TABLE <itab>
[ACCEPTING DUPLICATE KEYS].
```
此语句在单 个操作中将 内表 <itab> 中的所有行 添加到数据 库表中。
内表 <itab> 中的行长度 至少要等于 数据库表的 长度。为了 保证内表具 有与数据库 表相同的结 构，请通过 DATA 或 TYPES 语句用 LIKE <dbtab> 选项来定义 它（参见 *DATA 语句的基本格式* (页 3 - 14))。
如果操作成 功完成，就 将 SY-SUBRC 设置为 0。如果因 为至少有一 具有相同的 主码的行已 在 <dbtab> 中存在，从 而使得部分 行不能插入 ，那么，系 统将返回一 个运行时间 错误。要避 免该运行时 间错误，请 使用 ACCEPTING DUPLICATE KEYS 选项，它将 跳过这些行 而不插入， 并将 SY-SUBRC 设置为 4。如果需 要更改现存 的行而不是 跳过它们， 请使用 MODIFY 语句（参见 *添加或更改 行* (页 149))。
系统字段 SY-DBCNT 的值始终都 为已插入行 的数目，而 不考虑 SY-SUBRC 中的值。
命名数据库 表和 CLIENT SPECIFIED 选项的步骤 与 *添加一单行* (页 179) 所讲述的相 同。

> 因为处理行 集合更有效 ，所以与处 理单行相比 较，总是优 先采用处理 行集合。

```
TABLES SPFLI.

DATA ITAB LIKE SPFLI OCCURS 10 WITH HEADER LINE.

ITAB-CARRID = 'UA'. ITAB-CONNID = '0011'. ITAB-CITYFROM = ..
APPEND ITAB.

ITAB-CARRID = 'LH'. ITAB-CONNID = '1245'. ITAB-CITYFROM = ..
APPEND ITAB.

ITAB-CARRID = 'AA'. ITAB-CONNID = '4574'. ITAB-CITYFROM = ..
APPEND ITAB.

.................

INSERT SPFLI FROM TABLE ITAB ACCEPTING DUPLICATE KEYS.

IF SY-SUBRC = 0.
   .....
ELSEIF-SUBRC = 4.
   .....
ENDIF.
```
在该示例中 ，声明了一 个内表 ITAB， 该内表具有 与数据库表 SPFLI 相同的结构 。填写 ITAB 之后，然后 将它直接插 入 SPFLI 中。使用 IF 语句来检测 操作是否成 功，并根据 检测的结果 继续运行程 序。

在数据库表 中更改行
要在数据库 表中更改行 ，请使用 UPDATE 语句。它允 许更改单行 或多行。
如果因为不 知道要插入 行的主码是 否已经存在 ，从而不能 确定是否可 使用 UPDATE 语句，那么 ，请使用 MODIFY 语句。MODIFY 语句可用于 更改现存的 行和插入还 不存在的行 （参见 *添加或更改 行* (页 149))。
如果视图指 向单个表， 那么就只能 使用视图在 数据库表中 插入行。视 图的维护状 态必须在 ABAP/4 词典中定义 为“无限制 ”。
参见

更改单行
要使用 UPDATE 语句更改单 行，可使用
- UPDATE 语句的短格 式
- 带有 SET 子句并指定 了完整的 WHERE 条件的 UPDATE 语句（参见 *更改多行* (页 145) ）。

UPDATE 语句的短格 式如下所示 ：

语法
UPDATE <dbtab> [CLIENT SPECIFIED] FROM <wa>.
和
UPDATE <dbtab> [CLIENT SPECIFIED].
在第一个语 句中，工作 区 <wa> 中的内容将 覆盖数据库 表 <dbtab> 的行，该数 据库表与 <wa> 具有相同的 主码。必须 在程序中用 TABLES 声明该数据 库表。
在第二个语 句中，没有 指定工作区 <wa>。但表工作区 <dbtab> 中的内容将 覆盖具有相 同主码的数 据库表的行 。
工作区 <wa> 的长度至少 要等于表工 作区 <dbtab> 的长度。为 了保证字段 串的结构与 数据库表的 结构相同， 必须通过 DATA 或 TYPES 语句用 LIKE <dbtab> 选项定义（ 参见 *DATA 语句的基本格式* (页 3 - 14))。
CLIENT SPECIFIED 选项用于关 闭自动集团 处理（关于 集团处理的 详细信息， 参见*为数据库表 处理指定集 团* (页 51))。
如果操作成 功完成，就 将 SY-SUBRC 设置为 0，将 SY-DBCNT 设置为 1。否则 SY-SUBRC 返回 4 而 SY-DBCNT 返回 0。
如果需要在 运行时间指 定数据库表 的名称，请 使用下列语 法：
UPDATE (<dbtabname>) [CLIENT SPECIFIED] FROM <wa>.
可按照 *指 定将读取的 数据库表* (页 168) 中对 SELECT 语句的说明 指定数据库 表（<dbtabname>） 。如果在运行 时间指定数 据库表的名 称，必须使 用 FROM <wa> 选项。它不 能使用表工 作区。

```
TABLES SPFLI.

DATA WA LIKE SPFLI.

MOVE 'AA'         TO WA-CARRID.
MOVE '0064'       TO WA-CONNID.
MOVE 'WASHINGTON' TO WA-CITYFROM.
  .....

UPDATE SPFLI FROM WA.

MOVE 'LH'     TO SPFLI-CARRID.
MOVE '0017'   TO SPFLI-CONNID.
MOVE 'BERLIN' TO SPFLI-CITFROM.
  .....

UPDATE SPFLI.
```

CARRID 和 CONNID 是表 SPFLI 的主码字段 。主码字段 是“AA” 和“0064” ，或“LH” 和“0017” 的行中的所 有字段都由 工作区 WA 或表工作区 SPFLI 中对应字段 中的值所替 换。主码。

更改多行
要使用 UPDATE 语句在数据 库表中更改 多行，请使 用下列语法 ：

语法
UPDATE <dbtab> [CLIENT SPECIFIED] SET $<S_1>$ .. $<S_n>$ [WHERE <condition>].
使用 WHERE 子句（参见 *选择即将读 取的行* (页 172) ）选定要更 改的行。如 果不指定 WHERE 子句，将更 改所有的行 。必须在程 序中用 TABLES 语句声明该 数据库表。 CLIENT SPECIFIED 选项用于关 闭自动集团 处理（参见*为数据库表 处理指定集 团* (页 51))。
可以识别要 更改的行的 列，并使用 SET 子句的列表 $<S_1>$ .. $<S_n>$给 每一列赋值 。该列表中 的元素可以 是不同的 SET 命令。有下 列三种 SET 命令：

- **<f> = <g>**
  将所 有选定行中 的 <f> 列中的值设 置为值 <g>。
- **<f> = <f> + <g>**
  将所 有选定行中 的 <f> 列中的值加 上值 <g>。
- **<f> = <f> - <g>**
  将所 有选定行中 的 <f> 列中的值减 去值 <g>。

<g> 可以是变量 或字母。关 于数据库字 段和 ABAP/4 字段之间的 可转换性的 信息，参见 SELECT 语句 INTO 子句的关键 字文档。
可在 SY-DBCNT 中找到已更 改行的数目 。如果至少 更改了一行 ，SY-SUBRC 就返回 0。如果没 有更改任何 行，就返回 4。

如果使用 SET 子句更改数 据库表中的 行，就不能 在运行时间 指定数据库 表的名称，而必须在程 序中指定。

```
TABLES SFLIGHT.
UPDATE SFLIGHT SET PLANETYPE = 'A310'
                   FLPRICE = FLPRICE - '100.00'
                   WHERE CARRID = 'LH'.
```

对于 SFLIGHT 中 CARRID 字段包含“ LH”的所 有行，PLANETYPE 字段将改为 “A310” ，并且 FLPRICE 字段将减去 100.00。 详细示例， 请参考 UPDATE 语句的关键 字文档。

使用内表更 改多行

要通过 UPDATE 语句使用内 表在数据库 表中更改多 行，请使用 下列语法：

语法

```
UPDATE <dbtab> [CLIENT SPECIFIED] FROM TABLE <itab>.
```

如果需要在 运行时间指 定数据库表 的名称，就 要使用下列 语法：

```
UPDATE (<dbtabname>) [CLIENT SPECIFIED] FROM TABLE <itab>.
```

内表 <itab> 中的行将覆 盖数据库表 中具有相同 主码的行。

内表中行的 长度至少要 等于数据库 表中行的长 度。为了保 证内表的结 构与数据库 表的结构相 同，就必须 通过 DATA 或 TYPES 语句使用 LIKE <dbtab> 选项来定义 （参见 *DATA 语句的基本格式* (页 3 - 14) ）。

如果因为不 存在带有指 定关键字的 行而使系统 不能更改行 ，那么，系 统不会终止 整个操作， 而是继续处 理内表中的 下一行。

如果已经处 理了内表中 的所有行， 就将 SY-SUBRC 设置为 0。否则设 置为 4。在稍后 的时候，通 过从内表的 总行数中减 去 SY-DBCNT 中给出的实 际已处理行 数，就可计 算系统尚未 处理的行数 。如果内表 为空，那么 SY-SUBRC 和 SY-DBCNT 都将设置为 0。

因为操作行 集合更有效 率，所以与 操作单行相 比，总是优 先操作行集 合。

```
TABLES SPFLI.
DATA ITAB LIKE SPFLI OCCURS 10 WITH HEADER LINE.
ITAB-CARRID = 'UA'. ITAB-CONNID = '0011'. ITAB-CITYFROM = ..
APPEND ITAB.
ITAB-CARRID = 'LH'. ITAB-CONNID = '1245'. ITAB-CITYFROM = ..
APPEND ITAB.
ITAB-CARRID = 'AA'. ITAB-CONNID = '4574'. ITAB-CITYFROM = ..
APPEND ITAB.
.................
UPDATE SPFLI FROM TABLE ITAB.
```

在该示例中 ，给内表 ITAB 定义了与数 据库表 SPFLI 相同的结构 。在填写 ITAB 后，如果数 据库表中某 行的主码字 段（CARRID 和 CONNID） 的内容与内 表中某行的 主码内容相 同，那么， 就覆盖数据 库表中相应 的行。

添加或更改 行

要在数据库 表中插入一 行，而不考 虑该行的主 码是否已经 存在，请使 用 MODIFY 语句。

有两种可能 ：
- 如果数据 库表中没有 哪一行的主 码与即将插 入行的主码 相同，那么 ，MODIFY 的操作就与 INSERT 的操作相同 ，即添加该 行。
- 如果数据 库中已经包 含主码与即 将插入行的 主码相同的 行，那么， MODIFY 的操作就与 UPDATE 的操作相同 ，即更改该 行。

参见

出于性能的 原因，只有 在 ABAP/4 程序不能区 分这两个选 项时，才使 用 MODIFY。

插入单行
要插入单行 ，请使用下 列语法：

语法
MODIFY <dbtab> [CLIENT SPECIFIED] [FROM <wa>].
如果需要在 运行时间指 定数据库表 的名称，请 使用下列语 法：
MODIFY (<dbtabname>) [CLIENT SPECIFIED] [FROM <wa>].
该语句将工 作区 <wa> 中的内容写 入数据库表 中。将 SY-SUBRC 设置为 0，将 SY-DBCNT 设置为 1。
如果在工作 区 <wa> 中指定的主 码在数据库 表中不存在 ，那么，上 面的语句就 将 <wa> 添加到数据 库表中，如 *添加一单行* (页 179) 中所述。
如果在工作 区 <wa> 中指定的主 码在数据库 表中已经存 在，上面的 语句就更改 数据库表中 相应的行， 如 *更改单行* (页 180) 中所述。

插入多行
要插入多行 ，请使用下 列语法：

语法
MODIFY <dbtab> [CLIENT SPECIFIED] FROM TABLE <itab>.
如果需要在 运行时间指 定数据库表 的名称，请 使用下列语 法：
MODIFY (<dbtabname>) [CLIENT SPECIFIED] FROM TABLE <itab>.
内表 <itab> 中的行将覆 盖数据库表 中具有相同 主码的行。 内表中的其 它行将添加 到数据库表 。
SY-SUBRC 设置为 0，而将 SY-DBCNT 设置为内表 中的行数。
对于添加， 上述语句按 *从内表中添 加几行* (页 180) 中所作的说 明操作。
对于覆盖， 上述语句按 *使用内表更 改多行* (页 182) 中所作的说 明操作。

相对而言， 操作多行的 集合比操作 单行更有效 。

从数据库表 中删除行
要从数据库 表中删除行 ，请使用 DELETE 语句。DELETE 语句允许删 除单行或多 行。
如果视图指 向单个表， 就只能使用 视图删除数 据库表中的 行。视图的 维护状态必 须在 ABAP/4 词典中定义 成“无限制 ”。
参见

删除单行
要使用 DELETE 语句删除单 行，可以：
- 将 DELETE 语句与完整 的 WHERE 条件一起使 用（参见 *删除多行* (页 41) ）。
- 使用 DELETE 语句的短格 式。

DELETE 语句的短格 式如下所示 ：

语法
DELETE <dbtab> [CLIENT SPECIFIED] FROM <wa>.
和
DELETE <dbtab> [CLIENT SPECIFIED].
在第一个语 句中，将从 数据库表 <dbtab> 中删除主码 与 <wa> 中指定的主 码相同的行 。必须在程 序中用 TABLES 语句声明该 数据库表。

在第二个语 句中，没有 指定工作区 <wa>。 而是从数据 库表中删除 主码与表工 作区 <dbtab> 中指定的主 码相同的行 。
工作区 <wa> 的长度至少 要等于数据 库表的主码 的长度。CLIENT SPECIFIED 选项用于关 闭自动集团 处理（关于 集团处理的 详细信息， 参见 *为数据库表 处理指定集 团*（页 51））。
如果成功地 删除了行， SY-SUBRC 将设置为 0。如果没 有哪一行的 主码与指定 的主码匹配 ，就将它设置为 4。
如果需要在 运行时间指 定数据库表 的名称，请 使用下列语 法：
DELETE (<dbtabname>) [CLIENT SPECIFIED] FROM <wa>.
可按照 *指 定将读取的 数据库表*（页 168）中对 SELECT 语句的说明 指定数据库 表（<dbtabname>） 。
如果在运行 时间指定了 数据库表， 就必须使用 选项 FROM <wa>， 而不能使用 表工作区。

```
TABLES SPFLI.

DATA WA LIKE SPFLI.

MOVE 'AA'          TO WA-CARRID.
MOVE '0064'        TO WA-CONNID.
DELETE SPFLI FROM WA.

MOVE 'LH'      TO SPFLI-CARRID.
MOVE '0017'    TO SPFLI-CONNID.
DELETE SPFLI.
```

CARRID 和 CONNID 是表 SPFLI 的主码字段 。将删除所 有主码字段 为“AA”和“0064”、或“LH” 和“0017” 的行。

删除多行
要在单个操 作中从数据 库表 <dbtab> 中删除多行 ，可使用如 下格式的 DELETE 语句：

语法
DELETE FROM <dbtab> [CLIENT SPECIFIED] WHERE <conditions>.
如果需要在 运行时间指 定数据库表 的名称，请 使用下列语 法：
DELETE FROM (<dbtabname>) [CLIENT SPECIFIED] WHERE <conditions>.
使用 WHERE 条件指定即 将删除的行 （参见 *选择即将读 取的行*（页 172））。
将从数据库 表中删除所 有符合该条 件的行。

要仔细检查 WHERE 条件，确保 DELETE 语句没有删 除数据库表 中所有的行 。

CLIENT SPECIFIED 选项关闭自 动集团处理 （关于集团 处理的详细 信息，参见 *为数据库表 处理指定集团*（页 51））。关于在 运行时间设 置数据库表 的名称的信 息，参见 *指 定将读取的 数据库表*（页 168）。
在该操作后 ，SY-DBCNT 中将包含删 除的行的数 目。如果至 少删除了一 行，SY-SUBRC 就设置为 0。如果没 有删除任何 行，SY-SUBRC 就返回 4。

```
TABLES SFLIGHT.

DELETE FROM SFLIGHT WHERE PLANETYPE = 'A310'
                    AND      CARRID = 'LH'.
```

在该示例中 ，将从 SFLIGHT 中删除 PLANETYPE 字段中包含 “A310” 且 CARRID 字段中包含 “LH”的 所有行。

使用内表删 除多行
要通过 DELETE 语句用内表 从数据库表 中删除多行 ，请使用下 列语法：

语法
```
DELETE <dbtab> [CLIENT SPECIFIED] FROM TABLE <itab>.
```
如果需要在 运行时间指 定数据库表 的名称，请 使用下列语 法：
```
DELETE (<dbtabname>) [CLIENT SPECIFIED] FROM TABLE <itab>.
```
这些语句从 数据库表中 删除主码与 内表〈itab〉中的某行相 同的行。
内表的长度 至少与数据 库表的主码 的长度相同 。
如果因为数 据库表中没 有哪一行的 主码与指定 的相同，从 而系统没有 删除任何行 ，那么，系 统不会终止 整个操作， 而是继续处 理内表的下 一行。
如果已经处 理了内表中 所有的行， SY-SUBRC 将设置为 0。否则， 设置为 4。在稍后 的时候，从 内表的总行 数中减去 SY-DBCNT 中给出的实 际已删除的 行数，就可 计算系统尚 未删除的行 数。如果内 表是空的， 那么 SY-SUBRC 和 SY-DBCNT 都将设置为 0。

> 相对而言， 操作多行的 集合比操作 单行更有效 。

```
TABLES SPFLI.

DATA ITAB LIKE SPFLI OCCURS 10 WITH HEADER LINE.

ITAB-CARRID = 'UA'. ITAB-CONNID = '0011'.
APPEND ITAB.

ITAB-CARRID = 'LH'. ITAB-CONNID = '1245'.
APPEND ITAB.

ITAB-CARRID = 'AA'. ITAB-CONNID = '4574'.
APPEND ITAB.

.................

DELETE SPFLI FROM TABLE ITAB.
```

在该示例中 ，用与数据 库表 SPFLI 相同的结构 定义了内表 ITAB。 填写 ITAB之 后，SPFLI 中主码（CARRID 和 CONNID） 与内表的某 行相同的行 都将被删除 。

使用光标从 数据库表中 读取行

可使用数据 库光标从数 据库表中读 取行。为此 ，必须执行 下列操作：
有关如何使 用光标从数 据库表中读 取数据的示 例，参见

打开光标

使用光标几 乎可从任何 SELECT 语句的结果 集中获得下 一行（或一 些行）。为 此，请按如 下格式使用 OPEN CURSOR 语句将光标 与相关的 SELECT 语句链接起 来：

语法
```
OPEN CURSOR [WITH HOLD] <c> FOR SELECT ........
                               [WHERE <conditions>].
```
首先必须将 光标〈c〉定义为类型 CURSOR。
如果使用 WITH HOLD 选项，那么 在自身的 SQL 数据库提交 之后，该光 标将保留打 开状态（参 见 *在 ABAP/4 程序中使用 本地的 SQL 语句* （页 51） ）。

> 不能使用带 有下列子句 的 SELECT 语句：
>
> _ SELECT SINGLE ......
>
> _ SELECT .... $\langle a_1 \rangle \langle a_2 \rangle$ ..... 如果该列表 包含总计功 能，但没有 单个字段。
>
> 可使用 *定义选择的 结果* （页 165） 中说明的其 它 SELECT 子句。

用光标读取 数据

在打开光标 后，就可使 用 FETCH 语句从 OPEN CURSOR 语句生成的 结果集中读 取下一行（ 或一些行），如下所示 ：

语法
```
FETCH NEXT CURSOR <c> INTO <target>.
```
所选行都将 读到 INTO 子句指定的 目标区中（ 参见 *为选定数据 指定目标区* (页 170))。

如果 FETCH 语句没有读 取任何行， SY-SUBRC 就设置为 4，否则设 置为 0。关闭光标

必须使用 CLOSE CURSOR 语句关闭不 再需要的光 标，如下所 示：

语法
```
CLOSE CURSOR <c>.
```
下列情况将 自动关闭光 标：

- 执行 COMMIT WORK 或 ROLLBACK WORK 语句时(参 见 *确认或取消 对数据库表 的更改* (页 159))。
- 当执行自 身的 SQL 数据库提交 或取消时（ 参见 *在 ABAP/4 程序中使用 本地的SQL 语句* (页 Error! Not a valid link.) ）。
- 更改屏幕 时
- 执行远程 功能调用时

> 如果在 OPEN CURSOR 语句中使用 WITH HOLD 选项，自身 的 SQL 中的数据库 提交将不会 关闭光标。

使用光标读 取数据的示 例

下面是使用 光标读取数 据的示例。

```
TABLES SPFLI.

DATA: C1 TYPE CURSOR, C2 TYPE CURSOR.

DATA: WA1 LIKE SPFLI,
      WA2 LIKE SPFLI.

DATA: FLAG1, FLAG2.

OPEN CURSOR: C1 FOR SELECT * FROM SPFLI WHERE CARRID = 'LH',
             C2 FOR SELECT * FROM SPFLI WHERE CARRID = 'AA'.

DO.
  IF FLAG1 NE 'X'.
    FETCH NEXT CURSOR C1 INTO WA1.
    IF SY-SUBRC <> 0.
      CLOSE CURSOR C1. FLAG1 = 'X'.
    ELSE.
      WRITE: / WA1-CARRID, WA1-CITYFROM, WA1-CITYTO.
    ENDIF.
  ENDIF.
  IF FLAG2 NE 'X'.
    FETCH NEXT CURSOR C2 INTO WA2.
    IF SY-SUBRC <> 0.
      CLOSE CURSOR C2. FLAG2 = 'X'.
    ELSE.
      WRITE: / WA2-CARRID, WA2-CITYFROM, WA2-CITYTO.
    ENDIF.
  ENDIF.
  IF FLAG1 = 'X' AND FLAG2 = 'X'. EXIT. ENDIF.
ENDDO.
```

输出列表如 下所示：

用不同条件 控制的两个 光标读取数 据库表 SPFLI。 在 DO 循环中将相 继处理 CARRID 字段中包含 “LH”或 “AA”的 行。如果没 有找到与该 条件匹配的 行，就关闭 光标 C1 或光标 C2，并将 FLAG1 或 FLAG2 设置为“X” 。

确认或取消 对数据库表 的更改

有时，在继 续处理前需 要确认对数 据库表所作 的更改。另 一方面，在 永久存储之 前可能需要 取消一些对 数据库表所 作的更改。
要确认对数 据库表的更 改，请使用 COMMIT WORK 语句。要在 永久存储之 前取消一些 更改，请使 用 ROLLBACK WORK 语句。
这些语句在 对话编程（ 即 SAP 事务、数据 库事务、打 开或关闭屏 幕等等，有 关对话编程 的概述，参 见*编写 ABAP/4 事务* (页 Error! Not a valid link.)）中起着 十分重要的 作用。在本 主题中，您将了解到如 何在 ABAP/4 报表中使用 COMMIT WORK 和 ROLLBACK WORK（ 参见*编写 ABAP/4 报表* (页 Error! Not a valid link.)）。
在 ABAP/4 报表中，可 能有几个作 业联系在一 起形成作业 的逻辑单元 （LUW）。通常，可 能要处理 LUW 中的所有动 作，或什么 也不要处理 。假设 LUW 中包含了将 五行插入到 数据库表中 的进程。如 果事务成功 ，所有五行 都将存储在 数据库表中 （这包含一 个更新请求 和一个数据 库事务）。 在显示新屏幕之前，数 据库事务将 自动结束（ ABAP/4 报表的结束 ）。在 ABAP/4 报表中不能 以任何方式 影响该进程
如果要保证 数据库中当 前所作的更 改立即被确 认，那么， 就必须使用 COMMIT WORK 语句结束 LUW。COMMIT WORK 在程序代码 中标记了 LUW 结束并启动 更新任务（ 参见 *COMMIT WORK 处理* (页 Error! Not a valid link.)）。在 COMMIT WORK 语句以后， 对数据库所 作的所有更 改都不能再 取消。
但是，如果 在 LUW 中出现了错 误，就必须 取消已经执 行的部分。 这意味着当 前没有任何 插入的行能 永久地保存 在数据库中 。要撤销当 前 LUW 对数据库的 更改，请使 用 ROLLBACK WORK， 它将取消前 一次数据库 提交后的所 有更改。
要确认对数 据库表的更 改并使它们 不能再取消 ，请按如下 格式使用 COMMIT WORK 语句：

语法
```
COMMIT WORK [AND WAIT].
```
如果使用 AND WAIT 选项，那么 在程序继续 执行以前， 它要等到更 新任务的结 束。如果更 新是成功的 ，SY-SUBRC 就设置为 0。如果 SY-SUBRC 返回一个非 零值，就没 有成功的存 储所作的更 改。
要在保存之 前取消对数 据库表所作 的更改，请 按下列形式 使用 ROLLBACK WORK 语句：

语法
```
ROLLBACK WORK.
```
如果对更改 的取消是成 功的，SY-SUBRC 就设置为 0。如果 SY-SUBRC 返回一个非 零值，就没 有成功地取 消所作的更 改

使用 COMMIT WORK 和 ROLLBACK WORK 语句的结果 是将丢失所 有的数据库 光标。因此 ，在 SELECT 循环中或在 处理 SQL 语句之前不 允许使用这 些语句。

关于 COMMIT WORK 和 ROLLBACK WORK 的详细信息 ，请参考关 键字文档。

```
TABLES SPFLI.

DATA FLAG.

SPFLI-CARRID = 'UA'. SPFLI-CONNID = '0011'.
SPFLI-CITYFROM = ............
INSERT SPFLI.
IF SY-SUBRC <> 0.
   FLAG = 'X'.
ENDIF.

SPFLI-CARRID = 'LH'. SPFLI-CONNID = '1245'.
SPFLI-CITYFROM = ............
INSERT SPFLI.
IF SY-SUBRC <> 0.
```

```
   FLAG = 'X'.
ENDIF.

SPFLI-CARRID = 'AA'. SPFLI-CONNID = '4574'.
SPFLI-CITYFROM = ............
INSERT SPFLI.
IF SY-SUBRC <> 0.
   FLAG = 'X'.
ENDIF.

  ................

  ................

IF FLAG = 'X'.
   ROLLBACK WORK.
ELSE.
   COMMIT WORK.
ENDIF.
```

在该示例中 ，LUW 将在 SPFLI 中插入一系 列特定的行 。在每个 INSERT 语句后，程 序将检查操 作是否成功 ，或检查具 有相应的主 码字段（CARRID 和 CONNID） 的行是否已 经在 SPFLI 中存在。稍 后，程序将 FLAG 设置为“X” 。如果对于 每一个 INSERT 语句，SY-SUBRC 没有设置为 0，那么最 后一个 IF 语句中的 ROLLBACK WORK 语句将取消 对数据库的 所有更改。 否则，用 COMMIT WORK 语句确认这 些更改。

为数据库表 处理指定集 团

自动集团处 理是标准的 开放式 SQL 语句，它用 于访问**集团 专用表**，只 读取并处理 当前集团中 的数据。
它不能

- 在 WHERE 子句中为集 团指定条件 ，否则系统 将在以后的 语法检查中 报告一个错 误。
- 在表工作 区中为INSERT、 UPDATE 或 DELETE 操作填充集 团字段，如 果这样做， 什么也不会 发生，这是 因为 ABAP/4 运行时间系 统将在执行 开放式 SQL 调用之前用 当前集团覆 盖表工作区 中的集团字 段。

如果用户需 要自己为表 处理指定集 团，就必须 将相应的开 放式 SQL 语句（SELECT、 INSERT、 UPDATE、 MODIFY 或 DELETE） 与下列附加 选项一起使 用：

语法
```
... CLIENT SPECIFIED ....
```
该选项必须 紧跟在数据 库表名称的 后面。
如果使用 CLIENT SPECIFIED 选项，就关 闭了自动集 团处理。然 后就可以使 用 WHERE 条件或表工 作区填充 ABAP/4 程序中的集 团字段。

```
TABLES SPFLI.

SELECT * FROM SPFLI CLIENT SPECIFIED
         WHERE MANDT BETWEEN '001' AND '003'.
  ...
ENDSELECT.
```

在该示例中 ，将从数据 库表 SPFLI 中读取集团 “001” 到“003” 的所有行， 而不会考虑 用户选择的 用来访问 R/3 系统的集团 标识。

在 ABAP/4 程序中使用 本地的SQL 语句

开放式 SQL 允许访问 SAP 系统能识别 的所有数据 库表，而不 用考虑数据 库的创建者 。但是，有 时也需要在 ABAP/4 程序中用到 数据库专用 的 SQL 语句（自身 的 SQL）。

通常，带有 数据库专用 SQL 语句的 ABAP/4 程序不能与 不同的数据 库一起运行 。如果希望 程序能与不 同的数据库 一起运行， 请只使用开 放式 SQL 语句。

要使用自身 的 SQL 语句，该语 句必须以 EXEC SQL 语句打头， 并以 ENDEXEC 语句结尾， 如下所示：

语法
```
EXEC SQL [PERFORMING <form>].
   <Native SQL statement>[;]
ENDEXEC.
```
自身的 SQL 语句后面的 分号（;） 是可选的， 但不能使用 句号（.） 。并且在自 身的 SQL 语句中，引 号（″）并 不表示后面 是注释，这 与通常的 ABAP/4 语法规则不 同。
使用自身的 SQL 语句时，定 址的数据库 不一定必须 由 ABAP/4 词典识别。 因此，ABAP/4 程序不一定 包含相应的 TABLES 语句。同样 ，因为启动 SAP 系统时当前 数据库是自 动连接的， 所以不必明 确地为该数 据库使用 CONNECT 语句。一旦 连接了数据 库表或字段 ，知道定址 的数据库系 统是否可区 分大小写就 非常重要。

> 在自身的 SQL 中没有自动 集团处理（ 参见 *为数据库表 处理指定集 团*（页 188））。始终需 要指定集团 。

数据通过主 机变量在数 据库表和 ABAP/4 程序之间传 送。这些变 量在 ABAP/4 程序中声明 ，在自身的 SQL 语句中，这 些变量前面 要有一个冒 号（:）。 可将基本字 段和结构化 字段用作主 机变量。
如果 SELECT 语句的结果 是表，那么 ，就使用 PERFORMING <form> 选项在闭循 环中逐行读 取数据。对于每一行， 都将调用一 次子程序 <form>。 在该子程序 中，可通过 将数据附加 到内表以便 对其作进一 步的处理。
有关自身的 SQL 的详细信息 ，参见 EXEC SQL 的关键字文 档。

```
DATA: BEGIN OF WA,
       CLIENT(3),
       ARG1(3),
       ARG2(3),
     END OF WA.

DATA F3 VALUE ' 1 '.

EXEC SQL PERFORMING LOOP_OUTPUT.
SELECT CLIENT, ARG1 INTO :WA FROM TABLE_001 WHERE ARG2 = :F3
ENDEXEC.

FORM LOOP_OUTPUT.
   WRITE: / WA-CLIENT, WA-ARG2.
ENDFORM.
```

在该示例中 ，工作区 WA 和字符字段 **F3** 都在自身的 SQL SELECT 语句中用作 主机变量。 WA 是目标区， 所选数据将 读入该目标 区。**F3** 用在 WHERE 条件中。在 子程序 LOOP_OUTPUT 中，将读入 WA 中的数据写 到屏幕上。

在 ABAP/4 程序的执行 过程中锁定 数据库对象

在 SAP 系统中，应 用程序必须 锁定它们使 用的对象。 这是因为每 个程序都由 两个任务组 成：
- 对话任务 ，在该任务 中输入或更 改数据
- 更新任务 ，在该任务 中更改数据 库

对话任务将 用户输入的 数据传送给 更新任务。 然后更新任 务读取即将 修改的数据 并相应地修 改数据库。
通常，对话 任务和更新 任务是异步 操作的。这 意味着对话 任务并不等 待更新任务 根据用户的 输入修改数 据库，而是 立即返回到 和用户的对 话中。
因为对话任 务和更新任 务是异步操 作的，所以 必须锁定所 有涉及的对 象，直到已 成功地更改 了数据库。
在 ABAP/4 程序的执行 过程中，必 须锁定程序 中的所有对 象。
例如，可能 必须锁定客 户主记录行 来修改相应 的商业区数 据。在该情 况下，只要 锁定是活动 的，其它用 户就不能修 改该客户主 记录行。其 它用户的对 话必须等到 该应用程序 （包括更新 任务）结束 。
要完全锁定 一个数据对 象，请使用 SAP 锁定机制（ ENQUEUE/DEQUEUE） 。
每个应用程 序都提供了 大量可用于 锁定数据对 象的功能模 块。
要获得一个 可用于锁定 或解锁的功 能模块的列 表，请选择 “ABAP/4 开发工作台 ”屏幕上的 “功能库” 。这将进入 “ABAP/4 功能库：初 始屏幕”” ，在这里选 择“实用程 序 -> 查找”。在 后续“ABAP/4

库信息系统 ：功能模块 ”屏幕中， 在“功能模 块”字段中 输入 *queue*， 然后单击“ 执行”。此 操作开始搜 索名称中包 含 ENQUEUE 或 DEQUEUE 的功能模块 。
另外，可直 接搜索用于 处理要锁定 的数据库对 象的功能模 块。请在“ 功能模块” 区域中输入 该对象的名 称（包括用 于类属搜索 的星号）。
详细信息， 参见文档*系统服务*（页 Error! Not a valid link.）。

检查 ABAP/4 程序用户的 权限

与使用逻辑 数据库读取 数据不同， 运行时在 ABAP/4 程序中使用 SQL 语句处理数 据不会触发 权限检查（ 参见 *逻辑数据库的优点* （页 Error! Not a valid link.））。因为开 放式 SQL 和自身的 SQL 语句允许无 限制地访问 所有数据库 表，所以这 将带来问题 。

> 不是要为所 有用户授予 在 ABAP/4 程序中使用 SQL 语句访问所 有数据的权 限。但是，发布程序之 后，所有具 有该程序使 用权限的用 户都能启动 它。所以， 作为编写 SQL 报表程序的 程序员有责 任检查调用 该程序的用 户是否有权 访问将在其 中处理的数据。

要在 ABAP/4 程序中检查 用户权限， 请使用 AUTHORITY-CHECK 语句，如下 所示：

语法

```
AUTHORITY-CHECK OBJECT '<object>'
                       ID '<name1>' FIELD <F1>
                       ID '<name2>' FIELD <F2>
                          .............
                       ID '<name10>' FIELD <F10>.
```

<object> 是即将检查 的授权对象 的名称。必 须在 ID 后列出在 <object> 中定义的所 有授权字段 的名称（<name1>, <name2>....） 。必须在 <F1>, <F2>.... 中为即将检 查的权限输 入值以作为 变量或字母 。然后，该 语句将为命 名的对象搜 索该用户的 参数文件， 以检查该用 户是否有 <f> 中所有值的 授权。然后 SY-SUBRC 将设置为 0。可用 DUMMY 替换. FIELD <f> 跳过字段的 检查。必须 检查系统字 段 SY-SUBRC 中的内容来 检查 AUTHORITY-CHECK 的结果并作 出相应的动 作。有关可 能的返回代 码值的列表 和详细信息 ，请参考 AUTHORITY-CHECK 的关键字文 档。关于常 用授权的概 念，参见文 档*用户、权限和系统安全性*（页 Error! Not a valid link.）。

> 假设存在权 限对象 F_SPFLI 和字段 ACTVT、 NAME 及 CITY。
>
> ```
> SELECT * FROM SPFLI.
>    AUTHORITY-CHECK OBJECT 'F_SPFLI'
>                          ID 'ACTVT'  FIELD '02'
>                          ID 'NAME' FIELD SPFLI-CARRID
>                          ID 'CITY'   DUMMY.
>    IF SY-SUBRC NE 0. EXIT. ENDIF.
> ENDSELECT.
> ```
>
> 如果用户有 下列关于 F_SPFLI 的权限：
>
> ACTVT 01-03、 NAME AA-LH、 CITY none,
>
> 并且 SPFLI-CARRID 不处于“AA” 和“LH” 之间，权限 检查将终止 SELECT 循环。

第十二章　以簇方式存储数据对象

概览

内容

可以用数据簇方式对 ABAP/4 程序的任何复杂内部数据对象进行分组保存，并将其临时存储在 ABAP/4 内存中，或长时间存储在数据库中。

在下列主题中，您将学到更多有关在内存和数据库中存储数据簇的知识

ABAP/4 内存中的数据簇

可以在 ABAP/4 内存中存储数据簇。ABAP/4 内存是分配给特定事务的存储区，任何模块都是用关键词 CALL 或 SUBMIT 从中进行调用的。关于事务流的详细信息，参见*编写 ABAP/4 事务*（页 Error! Not a valid link.）。
ABAP/4 内存与在事务期间生成该内存的 ABAP/4 程序或程序模块无关。这意味着在同一事务中，存储在 ABAP/4 内存中的对象可由任何 ABAP/4 程序重新读取。但本节所说的 ABAP/4 内存与不受事务限制的全局 SAP 内存不同（对于示例，参见 *将 SPA/GPA 参数传送到 事务*（页 Error! Not a valid link.））。
ABAP/4 内存允许跨越多个程序层次，在不同模块化单元之间进行数据传递。例如，可以在下列单元之间传递数据：

- 报表和其他 SUBMIT 调用的报表
- 事务和报表
- 不同对话模块
- 程序和功能模块

等等。
离开事务后，就释放该内存。
使用 EXPORT TO MEMORY 语句在内存中存储数据对象。

使用 IMPORT FROM MEMORY 语句从内存中读取数据对象。
使用 FREE MEMORY 语句从内存中删除数据簇。

在 ABAP/4 内存中存储数据对象

要将数据对象从 ABAP/4 程序写入 ABAP/4 内存，请使用下列语句：

语法
EXPORT <F1> [FROM <g_1>] <F2> [FROM <g_2>] ... TO MEMORY ID <key>.
此语句将列表中指定的数据对象存储为 ABAP/4 内存中的数据簇。如果忽略选项 FROM <g_i>，则将数据对象 <f_i> 存储到自己的名称之下。如果使用该选项，则将数据对象 <g_i> 存储到 <f_i> 下面。 ID <key> 用于标识内存数据，不得超过 32 个字符。
EXPORT 语句总是完全改写 ID <key> 相同的任何现有数据簇的内容。

> 对于有表头行的内表，只可以存储表本身，而不能存储表头行。在 EXPORT 语句中，将表名解释为表。这是例外。通常情况下，语句将表名解释为表工作区（参见 *访问内表* （页 8 - 4））。

```
PROGRAM SAPMZTS1.

DATA TEXT1(10) VALUE 'Exporting'.

DATA ITAB LIKE SBOOK OCCURS 10 WITH HEADER LINE.

DO 5 TIMES.
  ITAB-BOOKID = 100 + SY-INDEX.
  APPEND ITAB.
ENDDO.

EXPORT TEXT1
       TEXT2 FROM 'Literal'
  TO MEMORY ID 'text'.

EXPORT ITAB
  TO MEMORY ID 'table'.
```

在此示例中，文本字段 TEXT1 和 TEXT2 存储到程序 SAPMZTS1 的 ABAP/4 内存的 ID “文本”之下，内表 ITAB 则存储到 ID “表”中。

从内存中读取数据对象

要将 ABAP/4 内存中的数据对象读到 ABAP/4 程序中，请使用下列语句：

语法
IMPORT <F1> [TO <g_1>] <F2> [TO <g_2>] ... FROM MEMORY ID <key>.

此语句从 ABAP/4 内存的数据簇中读取列表中指定的数据对象。如果忽略选项 TO <g_i>，则将内存中的数据对象 <f_i> 赋给程序中的同名数据对象。如果使用此选项，则将内存中的数据对象 <f_i> 写入字段 <g_i> 中。 ID <key> 用于标识内存数据，不得超过 32 个字符。
不必读取存储在特定 ID <key> 下的所有对象。相反，可以从名称 <f_i> 中进行选择。如果内存中不包含指定 ID <key> 下的对象，则将 SY-SUBRC 设置为 4。但是，如果内存中存在带此 ID 的数据簇，无论数据对象 <f_i> 是否也存在，SY-SUBRC 之值总是为 0。如果簇中不存在数据对象 <f_i>，则目标字段保持不变。
此语句不进行这种检查：即内存中的对象结构与要写入的结构是否匹配。因为数据是按位进行传送的，所以不匹配的结构可能会引起不一致。

```
PROGRAM SAPMZTS1.

DATA TEXT1(10) VALUE 'Exporting'.

DATA ITAB LIKE SBOOK OCCURS 10 WITH HEADER LINE.

DO 5 TIMES.
  ITAB-BOOKID = 100 + SY-INDEX.
  APPEND ITAB.
ENDDO.

EXPORT TEXT1
       TEXT2 FROM 'Literal'
  TO MEMORY ID 'text'.

EXPORT ITAB
  TO MEMORY ID 'table'.

SUBMIT SAPMZTS2 AND RETURN.

SUBMIT SAPMZTS3.
```

程序的第一部分对应于*在 ABAP/4 内存中存储 数据对象* (页 192) 中的示例。当前示例也以 SUBMIT 调用程序 SAPMZTS1 和 SAPMZTS2。通过在 ABAP/4 编辑器中双击程序名生成和维护 SUBMIT 后面指定的程序。关于 SUBMIT 的详细信息，参见*调用报表* (页 Error! Not a valid link.)。

SAPMZTS2 的示例：

```
PROGRAM SAPMZTS2.

DATA: TEXT1(10),
      TEXT3 LIKE TEXT1 VALUE 'Initial'.

IMPORT TEXT3 FROM MEMORY ID 'text'.
WRITE: / SY-SUBRC, TEXT3.

IMPORT TEXT2 TO TEXT1 FROM MEMORY ID 'text'.
WRITE: / SY-SUBRC, TEXT1.
```

SAPMZTS3 的示例：

```
PROGRAM SAPMZTS3.

DATA JTAB LIKE SBOOK OCCURS 10 WITH HEADER LINE.

IMPORT ITAB TO JTAB FROM MEMORY ID 'table'.
```

```
LOOP AT JTAB.
  WRITE / JTAB-BOOKID.
ENDLOOP.
```

输出位于两个连续的屏幕上，如下所示：

和

SAPMZTS2 试图从数据簇“文本”中读取不存在的数据对象 TEXT3。因此，目标字段 TEXT3 保持不变。现有数据对象 TEXT2 被放到 TEXT1 之后。两种情况中， 因为簇“文本”包含数据，SY-SUBRC 都被设置为 0。

SAPMZTS3 将内表 ITAB 从簇“表”中写入内表 JTAB。两个表结构一样，同为 ABAP/4 词典表 SBOOK 结构。

删除内存中的数据簇

要删除 ABAP/4 内存中的数据对象，请使用下列语句：

语法
FREE MEMORY [ID <key>].
如果不附加 ID <key>，则此语句删除整个内存，包括此前用 EXPORT 存储到 ABAP/4 内存中的所有数据簇。附加 ID <key> 之后，该语句只删除用此名称命名的数据簇。

因为删除整个内存会导致任何系统例程内存内容的丢失，所以只应使用附加有 ID 的 FREE MEMORY 语句。

```
PROGRAM SAPMZTST.

DATA: TEXT(10) VALUE '0123456789',
      IDEN(3)  VALUE 'XYZ'.

EXPORT TEXT TO MEMORY ID IDEN.

TEXT = 'xxxxxxxxxx'.
IMPORT TEXT FROM MEMORY ID IDEN.
WRITE: / SY-SUBRC, TEXT.

FREE MEMORY.
```

```
TEXT = 'xxxxxxxxxx'.
IMPORT TEXT FROM MEMORY ID IDEN.
WRITE: / SY-SUBRC, TEXT.
```

此示例的输出为：

```
0  0123456789
4  xxxxxxxxxx
```

FREE MEMORY 语句删除数据簇“XYZ”。因此，在下一个 IMPORT 语句之后，系统字段 SY-SUBRC 被设置为 4，而目标字段保持不变。

数据库中的数据簇

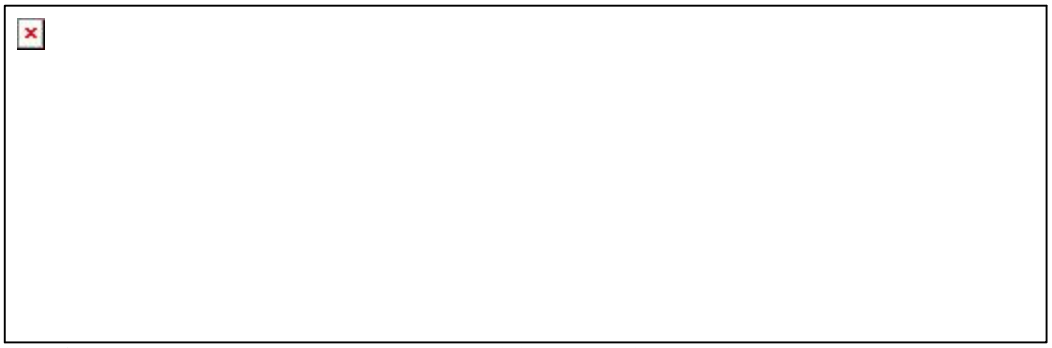
可以将数据簇存储到 ABAP/4 词典的特定数据库中。就是所谓的 ABAP/4 簇数据库，其预定义结构为：
该方法允许单步存储任何具有深结构的复杂数据对象，而不必将其调整为关系数据库的平面结构。这样，在整个系统中都可使用该数据对象，并且每个用户都可对其进行访问。要使访问成功，必须知道存储对象的数据类型。
在簇数据库中存储数据，对于支持有关关系数据库信息的分析结果十分有用。例如，如果要从所有分支机构的人员数据中生成销售额最高的客户清单或者完整的通讯录，就可以编写 ABAP/4 程序，让程序来解决此类问题，并将结果存储为数据簇。如果需要刷新存储的数据簇，可以在后台定期运行这些程序。要使用该结果，可以使用只访问该数据簇的其他程序。因为不必在每次使用结果时都访问关系数据库中的分布式数据，并且也不必每次都重新生成结果，所以，此方法可以很大程度上减少系统的响应时间。
存储数据簇是专就 ABAP/4 而言。尽管也可以使用 SQL 语句访问簇数据库，但是，只有 ABAP/4 语句能够对已存储的数据簇结构进行解码。
使用 EXPORT TO DATABASE 语句将数据对象存储到簇数据库中。
使用 IMPORT FROM DATABASE 语句为数据簇生成目录表，并从簇数据库中读取数据对象。
使用 DELETE FROM DATABASE 语句从簇数据库中删除数据簇。
关于使用开放式 SQL 语句访问簇数据库的信息，参见

簇数据库

簇数据库是 ABAP/4 词典中的特殊数据库。用于存储数据簇。其行结构被划分为部分标准化的开始区（由多个字段组成）和一个用于存储数据的大的区域。
下列主题介绍建立簇数据库的规则，同时还就系统定义的簇数据库 INDX 进行讨论。

簇数据库的结构

簇数据库的结构如下所示：



建立簇数据库的规则如下所述。必须创建第一点到第四点中列出的关键字段。上述数据类型都是 ABAP/4 词典类型。

1. 如果该表是针对客户的，第一个字段必须这样定义：名称为 MANDT，类型为 CHAR，长度为 3 字节，用于存储客户 ID。存储数据簇时，系统既可自动使用当前客户填写字段 MANDT，还可使用 EXPORT 语句中显式指定的客户进行填写。
2. 下一字段（对于与客户无关的表，这是第一个字段）必须这样定义：名称为 RELID，类型为 CHAR，长度为 2 字节。该字段包含区域 ID。簇数据库被分成不同的区域。存储数据簇时，系统用 EXPORT 语句中指定的区域 ID 填写字段 RELID。
3. 下一字段类型为 CHAR，长度可变。它包含簇的名称 <key>，存储数据簇时，在程序中用 EXPORT 语句的附加 ID 指定了该簇。因为后面的字段要对齐，所以系统应最多使用 3 个未用字节填充在字段 RELID 的结尾。如果创建自己的簇数据库，应该相应地定义此字段的长度。
4. 下一字段必须名称为 SRTF2，类型为 INT4，长度为 4。单个数据簇可以扩展到数据库表的好几行中。在理论上，每个簇可能有 2**31 行。字段 SRTF2 包含存储的数据簇内行的顺序号码，可以是 0 和 2**31-1 之间的任何值。存储数据簇时，系统自动填写此字段（参见第 7 点）。
5. SRTF2 的后面可以是任何数目的数据字段，这些字段名称和类型可任意交换。存储数据簇时，系统并不自动填写这些字段。必须在程序中的 EXPORT 语句之前将值显式分配到这些字段。通常包含诸如程序名、用户 ID 等控制信息。
6. 行上的倒数第二个字段名称必须为 CLUSTR，类型为 INT2，并且长度必须为 2。它包含后面的字段 CLUSTD 中的数据长度。存储数据簇时，系统自动填写此字段。
7. 行上的最后一个字段必须名称为 CLUSTD，类型为 VARC。其长度可以任意，但通常为 1000 个字节左右。存储数据簇时，系统按压缩格式用实际数据填写此字段。如果 CLUSTD 的长度不足以存储簇数据，则数据就被分布到多行上。这些行在字段 SRTF2 中进行编号（参见上面的第 4 点）。

既可以按照上述规则创建自己的簇数据库（此时参见文档 *ABAP/4 词典* (页 **Error! Not a valid link.**) ），也可以使用系统定义的簇数据库 INDX：

簇数据库的 示例 (页 186)

簇数据库的示例

数据库 INDX 是簇数据库的示例，是系统中所包含的标准安装的一部分。从用户应用的角度考虑，由于没有必要先创建新的簇数据库，所以，比较实用。而且，所有用户都可以访问存储在这里的数据，并且还可更改或删除。

要在 ABAP/4 编辑器中查看数据库 INDX 的结构，请选择“编辑 -> 详细功能 -> 命令条目”，然后输入 **SHOW INDX**，或者双击 INDX，例如在 TABLES 语句中。

对于每个字段，都将看到 ABAP/4 词典数据类型和相应的 ABAP/4 编程语言的数据类型（也就是 TABLES 生成的表工作区中的组件数据类型）。

头四个字段是表 INDX 的关键字段，并且与 *簇数据库的 结构* (页 195) 中介绍的完全相同。此处簇名的第三个字段的名称是 SRTFD，长度为 22 字节，也就是说，对于 INDX，ABAP/4 程序中以 EXPORT 语句的附加 ID 指定的名称 <key> 最多可以有 22 个字符。

后七个字段是非标准字段，用于用户输入，例如：

- AEDAT：最后修改的日期
- USERA：用户名
- PGMID：程序名

最后两个字段也是预定义的。在表 INDX 中，存储实际数据簇的字段 CLUSTD 的长度为 2886 个字节。

有关表 INDX 用法的示例，参见

在簇数据库 中存储数据 对象 (页 188)

创建数据簇 目录表 (页 186)

从簇数据库 中读取数据 对象 (页 177)

从簇数据库 中删除数据 簇 (页 188)

在簇数据库中存储数据对象

要在簇数据库中存储 ABAP/4 程序的数据对象，请使用下列语句：

语法

```
EXPORT <F1> [FROM <g1>] <F2> [FROM <g2>] ...
       TO DATABASE <dbtab>(<ar>) [CLIENT <cli>] ID <key>.
```

此语句将列表中指定的数据对象存储为簇数据库 <dbtab> 中的簇。必须用 TABLES 语句对 <dbtab> 加以声明。如果不附加 FROM <g_i>，则将数据对象 <f_i> 存储在自己的名称之下。如果有附件项，则将数据对象 <g_i> 存储到名称 <f_i> 之下。

<ar> 是存储数据库的簇的两字符区域 ID。

（参见 *簇数据库的 结构* (页 Error! Not a valid link.) *簇数据库的 结构* (页 Error! Not a valid link.) 下的第 2 点）。

<key> 标识数据库中的数据，其最大长度取决于 <dbtab> 中名称字段的长度。

（参见 *簇数据库的结构* (页 195) 下面的第 3 点）。

在处理特定客户的簇数据库时可以使用选项 CLIENT <cli> 关闭自动客户处理，然后自己指定客户。必须在指定数据库名称之后立即指定此选项。

（参见 *簇数据库的结构* (页 195) 下面的第 1 点）。

EXPORT 语句也将表工作区 <dbtab> 的用户字段内容传输到数据库表。根据需要，可以预先填写这些字段。

（参见 *簇数据库的结构* (页 195) 下面的第 5 点）。

在具有相同名称 <key> 的相同工作区 <ar> 和相同客户系统 <cli> 中，EXPORT 语句总是完全改写任何现有数据簇的内容。

> 对于含有表头行的内表，只可以存储表本身，而不能存储表头行。在 EXPORT 语句中，将表名解释为表。这是例外。通常将表名解释为表工作区（参见 *访问内表* (页 8 - 4) ）。

```
PROGRAM SAPMZTS1.

TABLES INDX.
```

```
DATA: BEGIN OF ITAB OCCURS 100,
        COL1 TYPE I,
        COL2 TYPE I,
      END OF ITAB.

DO 3000 TIMES.
  ITAB-COL1 = SY-INDEX.
  ITAB-COL2 = SY-INDEX ** 2.
  APPEND ITAB.
ENDDO.

INDX-AEDAT = SY-DATUM.
INDX-USERA = SY-UNAME.
INDX-PGMID = SY-REPID.

EXPORT ITAB TO DATABASE INDX(HK) ID 'Table'.

WRITE: '    SRTF2',
     AT 20 'AEDAT',
     AT 35 'USERA',
     AT 50 'PGMID'.
ULINE.

SELECT * FROM INDX WHERE RELID = 'HK'
                   AND   SRTFD = 'Table'.

  WRITE: / INDX-SRTF2 UNDER 'SRTF2',
           INDX-AEDAT UNDER 'AEDAT',
           INDX-USERA UNDER 'USERA',
           INDX-PGMID UNDER 'PGMID'.

ENDSELECT.
```

使用 3000 行填写了内表 ITAB，并且，在给 INDX 的某些用户字段赋值之后，ITAB 被输出到 INDX。

因为 INDX 是关系数据库，所以可以使用开放式 SQL 语句定位单独行。使用 SELECT 语句，辅以适当的 WHERE 条件就可选择用 EXPORT 存储的行。

数据库字段的输出如下所示：

其中，输出表明用户字段 AEDAT、USERA 和 PGMID 由 EXPORT 传送，并且包含 ITAB 的数据簇扩展了 6 行。如果在 DO 语句中更改数目 3000，则此数据簇占用的行数也会发生变化。

创建数据簇目录表

要从 ABAP/4 簇数据库中创建数据簇目录表，请使用下列语句：

语法

```
IMPORT DIRECTORY INTO <dirtab>
                FROM DATABASE <dbtab>(<ar>)
                [CLIENT <cli>] ID <key>.
```

此语句在存储于数据库 <dbtab> 中的数据簇中创建一系列数据对象，并将其放到表 <dirtab> 中。必须使用 TABLES 语句声明 <dbtab> 。

要将数据簇存储到数据库中，通常使用 EXPORT TO DATABASE 语句（参见 *在簇数据库 中存储数据 对象* (页 197)）。关于数据库表 <dbtab> 结构的详细信息，参见 *簇数据库的 结构* (页 195) 。

<ar> 是即将存储数据库的簇的两字符 ID。<key> 标识数据库中的数据，其最大长度取决于 <dbtab> 中名称字段的长度。在处理特定客户的簇数据库时，可以使用选项 CLIENT <cli> 关闭自动客户处理，然后自己指定客户。必须在指定数据库名称之后立即指定此选项。

IMPORT 语句也自动从数据库表中读取表工作区 <dbtab> 的用户字段内容。

如果可以创建某个目录表，则把 SY-SUBRC 设置为 0。否则，设置为 4。

必须按照 ABAP/4 词典结构 CDIR 建立内表 <dirtab>。为此，请使用 DATA 语句的附件 LIKE（参见 *DATA 语句的基本格式* (页 3 - 14))。结构 CDIR 包含下列组件：

字段名	类型	说明
NAME	CHAR	在簇中存储的对象名称
OTYPE	CHAR	对象类型： F 表示基本字段 R 表示字段串 T 表示内表
FTYPE	CHAR	对象的数据类型。结构化的数据类型是类型 C。
TFILL	INT4	已填写行的数目（针对内表）。
FLENG	INT2	字段或结构的长度。

```
PROGRAM SAPMZTS2.

TABLES INDX.

DATA DIRTAB LIKE CDIR OCCURS 10 WITH HEADER LINE.

IMPORT DIRECTORY INTO DIRTAB FROM DATABASE
                                    INDX(HK) ID 'Table'.

IF SY-SUBRC = 0.
  WRITE: / 'AEDAT:', INDX-AEDAT,
         / 'USERA:', INDX-USERA,
         / 'PGMID:', INDX-PGMID.
  WRITE  / 'Directory:'.
  LOOP AT DIRTAB.
    WRITE: / DIRTAB-NAME,  DIRTAB-OTYPE, DIRTAB-FTYPE,
             DIRTAB-TFILL, DIRTAB-FLENG.
  ENDLOOP.
```

```
ELSE.
  WRITE 'Not found'.
ENDIF.
```

此程序创建数据簇的目录表，该数据簇是用 *在簇数据库 中存储数据 对象* （页 197）中的示例程序存储的。其输出如下所示：

内容 DIRTAB 的表含有一行，该行表明数据簇包含名为 ITAB 的内表，表具有 3000 个长度为 8 个字节的已填写行。

从簇数据库中读取数据对象

要将数据对象从 ABAP/4 簇数据库读入 ABAP/4 程序中，请使用下列语句：

语法

```
IMPORT <F1> [TO <g1>] <F2> [TO <g2>] ...
       FROM DATABASE <dbtab>(<ar>)
       [CLIENT <cli>] ID <key>|MAJOR-ID <maid> [MINOR-ID <miid>].
```

此语句从数据库 <dbtab> 中的数据簇中读取列表中指定的数据对象。必须用 TABLES 语句声明 <dbtab> 。如果不附加 TO $<g_i>$，则将数据库的数据对象 $<f_i>$ 分配给程序中的同名数据对象。如果不附加此选项，则将数据库的数据对象 $<f_i>$ 写入字段 $<g_i>$。

要将数据簇存储到数据库中，通常使用 EXPORT TO DATABASE 语句（参见 *在簇数据库 中存储数据 对象* （页 197））。关于数据库表 <dbtab> 结构的详细信息，参见 *簇数据库的 结构* （页 195） 。

<ar> 是即将存储数据库的簇的两字符区域 ID。<key> 标识数据库中的数据，其最大长度取决于 <dbtab> 中名称字段的长度。可以用 MAJOR-ID <maid> 代替附加 ID <key>。然后，就选定名称的第一部分与 <maid> 相符的数据簇。如果指定具有 MAJOR-ID 的附加 MINOR-ID <miid>，则选择名称的第二部分（也就是 <maid> 长度之后的位置）大于或等于 <miid> 的数据簇。在处理特定客户簇数据库时，可以使用选项 CLIENT <cli> 关闭自动客户处理，然后自己指定客户。必须在输入数据库名之后立即指定此选项。

IMPORT 语句也自动从数据库表中读取表工作区 <dbtab> 的用户字段内容。

不必读取存储在特殊名称 <key> 之下的所有对象，但可以使用名称 $<f_i>$ 作出选择。如果数据库不包含具有指定关键字 <ar>、<key> 和 <cli> 的对象，则将 SY-SUBRC 设置为 4。但是，如果数据库中存在具有这些关键字的数据簇，那么，无论是否存在数据对象 $<f_i>$，SY-SUBRC 之值总是为 0。如果簇中没有数据对象 $<f_i>$，则目标字段保持不变。

运行时，系统检查此语句以查看数据库中对象的结构是否与要写入的结构相符。如果不符合，将出现运行时间错误。类型 C 字段是此规则的例外，也可显示在结构数据字段结尾。可以加长、缩短、附加或忽略。

```
PROGRAM SAPMZTS3.

TABLES INDX.

DATA: BEGIN OF JTAB OCCURS 100,
        COL1 TYPE I,
```

```
        COL2 TYPE I,
      END OF JTAB.

IMPORT ITAB TO JTAB FROM DATABASE INDX(HK) ID 'Table'.

WRITE: / 'AEDAT:', INDX-AEDAT,
       / 'USERA:', INDX-USERA,
       / 'PGMID:', INDX-PGMID.

SKIP.
WRITE 'JTAB:'.

LOOP AT JTAB FROM 1 TO 5.
  WRITE: / JTAB-COL1, JTAB-COL2.
ENDLOOP.
```

此程序将内表 JTAB（用 *在簇数据库 中存储数据 对象* (页 197) 中的示例程序存储）从簇数据库 INDX 读入内表 JTAB。INDX 的某些用户字段输出和 JTAB 的头五行如下所示：

从簇数据库中删除数据簇

要从簇数据库中删除数据簇，请使用下列语句：

语法
DELETE FROM DATABASE <dbtab>(<ar>) [CLIENT <cli>] ID <key>.
此语句删除数据库表 <dbtab> 中区域为 <ar> 和名称为 <key> 的整个数据簇。必须用 TABLES 语句对 <dbtab> 进行声明。
在处理特定客户簇数据库时，可以使用选项 CLIENT <cli> 关闭自动客户处理，然后自己指定客户。必须在输入数据库名称之后立即指定此选项。
要将数据簇存储到数据库中，通常使用 EXPORT TO DATABASE 语句（参见 *在簇数据库 中存储数据 对象* (页 197)）。关于数据库表 <dbtab> 结构的详细信息，参见 *簇数据库的 结构* (页 195) 。
此 DELETE 语句从簇数据库中删除被指定数据簇覆盖的所有行。
如果可以删除具有指定关键字的数据簇，则把 SY-SUBRC 设置为 0。否则，其值为 4。

```
PROGRAM SAPMZTS4.

TABLES INDX.

DATA DIRTAB LIKE CDIR OCCURS 10.

IMPORT DIRECTORY INTO DIRTAB FROM DATABASE
                                    INDX(HK) ID 'Table'.
WRITE: / 'SY-SUBRC, IMPORT:', SY-SUBRC.
```

```
DELETE FROM DATABASE INDX(HK) ID 'Table'.
WRITE: / 'SY-SUBRC, DELETE:', SY-SUBRC.

IMPORT DIRECTORY INTO DIRTAB FROM DATABASE
                                    INDX(HK) ID 'Table'.
WRITE: / 'SY-SUBRC, IMPORT:', SY-SUBRC.
```

此程序删除用 *在簇数据库 中存储数据 对象* (页 197) 中的示例程序存储的数据簇。如果启动程序时存在数据簇，则输出如下所示：

```
SY-SUBRC, IMPORT:      0

SY-SUBRC, DELETE:      0

SY-SUBRC, IMPORT:      4
```

在第一个 IMPORT 语句中，数据簇依然存在。然后成功地执行 DELETE 语句。在第二个 IMPORT 语句中，数据簇不再存在。

用开放式 SQL 语句访问簇数据库

簇数据库是在 ABAP/4 词典中定义的关系数据库，ABAP/4 以特殊方法使用该词典。因此，原则上说，也可以使用 *寥（4. _ 数据库表* (页 11－1) 中介绍的开放式 SQL 语句对其进行访问。
对簇数据库表，为了有意义地使用开放式 SQL 语句，必须清楚数据库表的特殊结构（参见 *簇数据库的 结构* (页 195))。
例如，用 SELECT 语句读取字段 CLUSTR 和 CLUSTID，或用 UPDATE 语句对其进行更改，都毫无意义。这些字段包含由系统进行编码的数据簇，要对其进行正确的处理，只能使用 EXPORT TO DATABASE 和 IMPORT FROM DATABASE 语句。
如果数据簇语句的特定组合导致超时运行，则只应使用开放式 SQL 语句 UPDATE、MODIFY 和 DELETE。一定不要在数据簇中使用开放式 SQL 语句 INSERT。
可以使用开放式 SQL 语句维护簇数据库。例如，SELECT 语句允许从簇数据库表中查看特定数据簇。其中，也可以使用用户数据字段中的信息（参见 *在簇数据库 中存储数据 对象* (页 197) 中的示例）。IMPORT FROM DATABASE 语句不适合此目的。

```
PROGRAM SAPMZTS5.

DATA COUNT TYPE I VALUE 0.

TABLES INDX.

SELECT * FROM INDX WHERE RELID = 'HK'
                   AND   SRTF2 = 0
                   AND   USERA = SY-UNAME.

DELETE FROM DATABASE INDX(HK) ID INDX-SRTFD.

IF SY-SUBRC = 0.
    COUNT = COUNT + 1.
  ENDIF.
```

```
ENDSELECT.
WRITE: / COUNT, 'Cluster(s) deleted'.
```

此示例程序从表 INDX 中删除区域“HK”（其中的字段 USERA 包含当前程序用户名称）中的所有数据簇。用 SELECT 语句填写表工作区 INDX 的字段 SRTFD，并在 DELETE 语句中使用。在 WHERE 子句中指定 SRTF2 = 0，可以保证只对每个数据簇进行一次处理。

请不要混淆开放式 SQL 命令集中的 DELETE 语句（参见 *从数据库表 中删除行*（页 11－35））和数据簇的 DELETE 语句（参见 *从簇数据库 中删除数据 簇*（页 201））。删除数据簇中的数据时，应该始终是删除所有行，而不仅仅指特定行。

下例说明如何使用开放式 SQL 语句 UPDATE 更改数据库表中的数据簇名称和区域。用簇语句 EXPORT、IMPORT 和 DELETE 来解决此问题显得太繁琐。

```
PROGRAM SAPMZTS5.
TABLES INDX.
DATA DIRTAB LIKE CDIR OCCURS 10 WITH HEADER LINE.
UPDATE INDX SET RELID = 'NW'
            SRTFD = 'Internal'
                                WHERE RELID = 'HK'
                                AND   SRTFD = 'Table'.
WRITE: / 'UPDATE:',
       / 'SY-SUBRC:', SY-SUBRC,
       / 'SY-DBCNT:', SY-DBCNT.
IMPORT DIRECTORY INTO DIRTAB FROM DATABASE
                                    INDX(NW) ID 'Internal'.
WRITE: / 'IMPORT:',
       / 'SY-SUBRC:', SY-SUBRC.
```

此程序更改了使用 *在簇数据库 中存储数据 对象*（页 197）中的示例程序存储的数据簇。启动程序时，如果存在数据簇，并且在 UPDATE 语句中没有其他错误，则输出如下所示：

```
UPDATE:
SY-SUBRC:      0
SY-DBCNT:      6
IMPORT:
```

```
SY-SUBRC:      0
```

UPDATE 语句更改属于指定数据簇的数据库表 INDX 的六行，正如 *更改嘈_*（页 11 - 31）中所述。然后，MPORT DIRECTORY 语句在名称“内部”之下的区域“NW”中查找数据簇。

概览

内容

ABAP/4 允许使用应 用服务器或 演示服务器 上的顺序文 件。

例如，这些 文件可以用 作数据的临 时存储设备 或本地程序 与 SAP 系统的接口 。

使用应用服 务器上的文 件

ABAP/4 提供一些语 句，以使用 存储在应用 服务器顺序 文件中而不 是存储在数 据库中的数 据。下列主 题说明：

文件和文件 路径的物理 定址是与平 台相关的。 R/3 系统提供了 允许使用平 台相关文件 名的功能模 块和事务：

ABAP/4 的文件处理

ABAP/4 提供了三种 文件处理语 句：

OPEN DATASET 语句用于打 开文件
CLOSE DATASET 语句用于关 闭文件
DELETE DATASET 语句用于删 除文件

打开文件

要在应用服 务器上打开 文件，请使 用 OPEN DATASET 语句。关于 OPEN DATASET 语句的基本 形式说明， 参见：

OPEN DATASET 语句的基本 形式 (页 169)

OPEN DATASET 语句有若干 个包括大量 任务选项。 可以：

关于其他选 项的详细信 息，参见 OPEN DATASET 语句的关键 字文档。

OPEN DATASET 语句的基本 形式

要在应用服 务器上打开 文件，请如 下使用 OPEN DATASET 语句：

语法

OPEN DATASET <dsn> [options].

此语句打开 文件 <dsn>。如果不指 定任何模式 选项，则文 件将按二进 制模式打开 （参见下述主题）。如 果系统不能 打开文件， 则将系统字 段 SY-SUBRC 设置为 0，否则 SY-SUBRC 返回 8。

可以将文件 名 <dsn> 指定为字母 或包含文件 名的字段。如果未指定 路径，则系 统将在应用 服务器上 SAP 系统运行的 目录中打开 文件。要打 开文件，运 行 SAP 系统的用户 必须在操作 系统级有相 应的权限。

文件名是平 台相关的。 必须根据运 行 SAP 系统的操作 系统规则指 定文件名或 路径。要编 写与操作系 统中不相关 的程序，可 以使用逻辑 文件名（关 于逻辑文件 名的详细信 息，参见 *使用平台独 立的文件名* （页 172））。

```
DATA FNAME(60).
FNAME = '/tmp/myfile'.
OPEN DATASET 'myfile'.
OPEN DATASET FNAME.
```

如果 SAP 系统在 UNIX 系统下运行 ，则此示例 可以运行。 此程序在运 行 SAP 系统的目录 中以及在“ /tmp” 路径中打开 文件“myfile” 。对于其它 操作系统， 必须替换其 它文件名。 例如，对 OpenVMS 系统，可以 指定下述内 容：

```
FNAME = '[TMP]myfile.BIN'
OPEN DATASET 'myfile.BIN'.
```

接受操作系 统消息

尝试打开文 件后，要接 受操作系统 消息，请如 下使用 OPEN DATASET 语句的 MESSAGE 选项：

语法

```
OPEN DATASET <dsn> MESSAGE <msg>.
```

系统将在变 量 <msg> 中放置相关 的操作系统 消息。

要进行错误 处理，请与 系统字段一 起使用此选 项。

```
DATA: MESS(60),
      FNAME(10) VALUE 'hugo.xyz'.
OPEN DATASET FNAME MESSAGE MESS.
IF SY-SUBRC <> 0.
  WRITE: 'SY-SUBRC:', SY-SUBRC,
       / 'System Message:', MESS.
ENDIF.
```

如果 SAP 系统在 UNIX 下运行且不 存在“hugo.xyz” 文件，则此 示例输出如 下：

打开文件读 取

要打开文件 进行读访问 ，请如下使 用 OPEN DATASET 语句的 FOR INPUT 选项：

语法

```
OPEN DATASET <dsn> FOR INPUT.
```

此语句打开 文件用于读 取。文件必 须已经存在 ，否则系统 将 SY-SUBRC 设置为 8 并且忽略此 命令。

如果文件已 打开（可能 用于读、写 、或追加） ，系统将复 位到文件的 起始位置。 但是在重新打开文件之 前使用 CLOSE 语句是良好 的编程风格 （关于关闭 文件的详细 信息，参见 *关闭文件* （页 158） ）。

```
DATA FNAME(60) VALUE 'myfile'.

OPEN DATASET FNAME FOR INPUT.

IF SY-SUBRC = 0.
  WRITE / 'File opened'.
  .....
ELSE.
  WRITE / 'File not found'.
ENDIF.
```

在此示例中 ，打开“myfile” 文件以读取 。

打开文件写 入

要打开文件 进行写访问 ，请如下使 用 OPEN DATASET 语句的 FOR OUTPUT 选项：

语法

```
OPEN DATASET <dsn> FOR OUTPUT.
```

此语句打开 文件用于写 入。如果文 件不存在，则创建文件 。如果文件 已存在但处 于关闭状态 ，则删除其 内容。如果 文件已存在 且已打开（ 可能为读、 写、或追加 ），则复位 到文件的起 始位置。如 果系统可以 打开文件， 则 SY-SUBRC 设置为 0，否则 SY-SUBRC 返回 8。

```
DATA: MESS(60),
      FNAME(10) VALUE '/tmp'.

OPEN DATASET FNAME FOR OUTPUT MESSAGE MESS.

IF SY-SUBRC <> 0.
  WRITE: 'SY-SUBRC:', SY-SUBRC,
       / 'System Message:', MESS.
ENDIF.
```

如果 SAP 系统在UNIX 下运行，此 示例输出如 下：

```
系统不能打 开该文件， 因为它是目 录。
```

下列程序将 演示当打开 文件用于写 入时其位置 如何设置。 但在重新打 开已打开文件之前使用 CLOSE 语句是良好 的编程风格 （关于关闭 文件的详细 信息，参见*关闭文件* （页 158） ）。

```
DATA FNAME(60) VALUE 'myfile'.
DATA NUM TYPE I.

OPEN DATASET FNAME FOR OUTPUT.

DO 10 TIMES.
  NUM = NUM + 1.
  TRANSFER NUM TO FNAME.
ENDDO.

PERFORM INPUT.

OPEN DATASET FNAME FOR OUTPUT.
```

```
NUM = 0.
DO 5 TIMES.
  NUM = NUM + 10.
  TRANSFER NUM TO FNAME.
ENDDO.

PERFORM INPUT.

CLOSE DATASET FNAME.

OPEN DATASET FNAME FOR OUTPUT.

NUM = 0.
DO 5 TIMES.
  NUM = NUM + 20.
  TRANSFER NUM TO FNAME.
ENDDO.

PERFORM INPUT.

FORM INPUT.
  SKIP.
  OPEN DATASET FNAME FOR INPUT.
  DO.
    READ DATASET FNAME INTO NUM.
    IF SY-SUBRC <> 0.
      EXIT.
    ENDIF.
    WRITE / NUM.
  ENDDO.
ENDFORM.
```

此程序输出 如下：

```
         1
         2
         3
         4
         5
         6
         7
         8
         9
        10

        10
        20
        30
        40
        50
         6
         7
         8
         9
        10

        20
        40
        60
        80
       100
```

在此示例中 ，“myfile” 文件

1. 打开 以写入。
2. 用 10 个整数填写 （关于 TRANSFER 语句的详细 信息，参见 *向文件中写 入数据* （页 183））。
3. 同时 打开以读取 ，这将复位 到文件的起 始位置。
4. 被读 入字段 NUM（关 于 READ DATASET 语句的详细 信息，参见 *从文件中读 取数据* （页 182））。NUM 的值写到输 出屏幕。
5. 重新 打开以写入 。这将复位 到文件的起 始处位置。
6. 用 5 个整数填写 。这些整数 将改写文件 的以前内容 。
7. 重新 打开以读取 。这将复位 到文件的起 始位置。
8. 被读 入字段 NUM。NUM 的值将写到 输出屏幕。

9. 关闭 （关于 CLOSE DATASET 语句的详细 信息，参见 *关闭文件*（页 158））。
10. 重 新打开以写 入。这将删 除现有文件 的内容。
11. 用 5 个整数填写 。
12. 同 时被打开以 读取。这将 复位到文件 的起始位置 。
13. 被 读入字段 NUM。NUM 的值将写到 输出屏幕。

打开文件追 加

要打开文件 追加数据， 请如下使用 OPEN DATASET 语句的 FOR APPENDING 选项：

语法

```
OPEN DATASET <dsn> FOR APPENDING.
```

此语句打开 文件在文件 末尾写入数 据。如果文 件不存在， 则创建文件 。如果文件 已存在但处于关闭状态 ，系统将打 开文件并定 位到文件末 尾。如果文 件已存在且 已打开（可 能为读、写 、或追加） ，将定位设 置到文件末 尾。SY-SUBRC 总是返回 0。

在再次打开 已经打开的 文件之前使 用 CLOSE 语句是良好 的编程风格 （关于关闭 文件的详细 信息，参见 *关闭文件*（页 158））。

```
DATA FNAME(60) VALUE 'myfile'.

DATA NUM TYPE I.

OPEN DATASET FNAME FOR OUTPUT.
DO 5 TIMES.
   NUM = NUM + 1.
   TRANSFER NUM TO FNAME.
ENDDO.

OPEN DATASET FNAME FOR INPUT.

OPEN DATASET FNAME FOR APPENDING.
NUM = 0.
DO 5 TIMES.
   NUM = NUM + 10.
   TRANSFER NUM TO FNAME.
ENDDO.

OPEN DATASET FNAME FOR INPUT.
DO.
  READ DATASET FNAME INTO NUM.
  IF SY-SUBRC <> 0.
    EXIT.
  ENDIF.
  WRITE / NUM.
ENDDO.
```

此示例输出 如下：

```
           1
           2
           3
           4
           5
          10
          20
          30
```

```
          40
          50
```

在此示例中 ，打开文件 以写入并用 从 1 ~ 5 中的五个整 数填写（关 于 TRANSFER 语句的详细 信息，参见 *向文件中写 入数据*（页 183））。下一语 句 OPEN DATASET 将复位到起 始位置，然 后打开文件 以追加并且 定位设置到 末尾位置。 10 ~ 50 的五个整数 将写入文件 。最后读取 文件内容并 写到屏幕。

指定二进制 模式

要用二进制 模式处理文 件，请如下 使用 OPEN DATASET 语句中的 IN BINARY MODE 选项：

语法

```
OPEN DATASET <dsn> IN BINARY MODE [FOR ....].
```

如果从以二 进制模式打 开的文件中 读取数据或 向此类文件 中写入数据 ,则系统将 逐字节地传输数据。在 传输期间， 系统不解释 文件内容。 在将文件内 容写入到另 一文件时， 系统将传输源文件的所 有字节。在 从文件读取 数据到字段 时，传输的 字节数目取 决于目标字 段大小。在读取之时或 之后，可以 用其它 ABAP/4 语句给目标 字段定址， 系统将根据 数据类型解 释字段内容 。

```
DATA FNAME(60) VALUE 'myfile'.

DATA: NUM1     TYPE I,
      NUM2     TYPE I,
      TEXT1(4) TYPE C,
      TEXT2(8) TYPE C,
      HEX      TYPE X.

OPEN DATASET FNAME FOR OUTPUT IN BINARY MODE.

NUM1  = 111.
TEXT1 = 'TEXT'.
TRANSFER NUM1  TO FNAME.
TRANSFER TEXT1 TO FNAME.

OPEN DATASET FNAME FOR INPUT IN BINARY MODE.

READ DATASET FNAME INTO TEXT2.
WRITE / TEXT2.

OPEN DATASET FNAME FOR INPUT IN BINARY MODE.

READ DATASET FNAME INTO NUM2.
WRITE / NUM2.

OPEN DATASET FNAME FOR INPUT IN BINARY MODE.

SKIP.
DO.
  READ DATASET FNAME INTO HEX.
  If SY-SUBRC <> 0.
    EXIT.
  ENDIF.
  WRITE  HEX.
ENDDO.
```

此示例输出 如下：

```
###oTEXT

111

00 00 00 6F 54 45 58 54
```

在按二进制 模式为写入 数据而打开 文件“myfile” 后，将字段 NUM1 和 TEXT1 的内容写入 文件（关于 TRANSFER 语句的详细 信息，参见 *向文件中写 入数据*（页 183））。然后打 开文件以读 取并且其全 部内容传输 到字段 TEXT2 （关于 READ DATASET 语句的详细 信息，参见 *从文件中读 取数据*（页 182））。字符串 TEXT2 的前四位没 什么意义， 因为各字节 是整数 111 表示法（平 台相关）。 系统企图将 所有字节解 释为字符， 但此解释工 作只对后四 个字节有效 。在用 OPEN 语句复位位 置之后，文 件的前四个 字节将读入 NUM2 。因为 NUM2 与 NUM1 的值有相同 的数据类型 ，所以 NUM2

的值有意义 。最后，将 文件的八个 字节逐字节 地从 HEX 的屏幕输出 读入到 HEX 字段中，可 以看到实际 文件内容十 六进制表示 法。后四个 字节为字 TEXT 中字符的 ASCII 表示法。

指定文本模 式

要用文本模 式处理文件 ，请如下使 用 OPEN DATASET 语句的 TEXT MODE 选项：

语法

```
OPEN DATASET <dsn> FOR .... IN TEXT MODE.
```

如果从以文 本模式打开 的文件中读 取数据或向 此类文件中 写入数据， 数据将逐行 传输。系统假定文件为 行结构。

对于每个 TRANSFER 语句（关于 TRANSFER 语句的详细 信息，参见 *向文件中写 入数据* (页 183) ），系统向 文件中传输 除结尾空白 之外的所有 字节，并在 其后作行结 束标记。

对于每个 READ DATASET 语句（关于 READ DATASET 语句的详细 信息，参见 *从文件中读 取数据* (页 182) ），系统将 读取下一个 行结束标记 之前的所有 数据。如果 目标字段太 小，则截断行的多余部 分。如果目 标字段长于 一行，则右 边各位填入 空格。

如果要将字 符串写入文 件或已知现 存文件是基 于行的格式 ，则应使用 文本模式。 例如，使用文本模式， 可以读取在 应用服务器 上用专用文 本编辑器创 建的文件。

为运行在 UNIX 系统（使用 ASCII 表示法）下 的 SAP 系统写入如 下演示程序 。

```
DATA FNAME(60) VALUE 'myfile'.

DATA: TEXT(4),
      HEX TYPE X.

OPEN DATASET FNAME FOR OUTPUT IN TEXT MODE.

TRANSFER '12        ' TO FNAME.
TRANSFER '123456  9' TO FNAME.
TRANSFER '1234      ' TO FNAME.

OPEN DATASET FNAME FOR INPUT IN TEXT MODE.

READ DATASET FNAME INTO TEXT.
WRITE / TEXT.
READ DATASET FNAME INTO TEXT.
WRITE  TEXT.
READ DATASET FNAME INTO TEXT.
WRITE  TEXT.

OPEN DATASET FNAME FOR INPUT IN BINARY MODE.

SKIP.
DO.
  READ DATASET FNAME INTO HEX.
  If SY-SUBRC <> 0.
    EXIT.
  ENDIF.
  WRITE HEX.
ENDDO.
```

此程序输出 如下：

```
12   1234 1234

31 32 0A 31 32 33 34 35 36 20 20 39 0A 31 32 33 34 0A
```

在此示例中 ，以文本方 式打开“myfile” 文件以写入 。三个字符 串文字（每 个长度为 10 ）传送到该 文件。在按 文本模式打 开文件以读 取之后，长 度为 4 的字段 TEXT 读入存储行 。第一行从 右端开始填 入两个空格 ，第二行的 后五位被截 断。以二进 制模式打开 文件并将其 读入十六进 制字段，表 明其实际结 构：在 31 ~ 36 之间的数是 数值字符 1 ~ 6 的 ASCII 表示法， 20 表示空格字 符，在每行 结尾标有 0A 标记。注意 ：字符串文 字的结尾空 格不写入文 件。

在指定位置 打开文件

要在指定位 置打开文件 ，请如下使 用 OPEN DATASET 语句的 AT POSITION 选项：

语法

```
OPEN DATASET <dsn> [FOR ....] [IN ... MODE] AT POSITION <pos>.
```

此语句打开 文件 <dsn> 并为读写数 据定位位置 <pos>， 该位置从文 件起始处起 按字节计算 。在文件起 始处以前， 不能指定位 置。

以二进制模 式工作时， 指定位置 <pos> 相当有用， 因为文本文 件的物理表 示法取决于 操作系统。

```
DATA FNAME(60) VALUE 'myfile'.

DATA NUM TYPE I.

OPEN DATASET FNAME FOR OUTPUT AT POSITION 16.
DO 5 TIMES.
  NUM = NUM + 1.
  TRANSFER NUM TO FNAME.
ENDDO.

OPEN DATASET FNAME FOR INPUT.
DO 9 TIMES.
  READ DATASET FNAME INTO NUM.
  WRITE / NUM.
ENDDO.

OPEN DATASET FNAME FOR INPUT AT POSITION 28.
SKIP.
DO 2 TIMES.
  READ DATASET FNAME INTO NUM.
  WRITE / NUM.
ENDDO.
```

此示例输出 如下：

```
        0
        0
        0
        0
        1
        2
        3
        4
        5

        4
        5
```

在此示例中 ，以默认二 进制模式打 开文件“myfile” 。写入时起 始位置指定 为 16，读取时起 始位置指定 为 28 。如本例所 示，如果位 置可被 4 整除，为写 入或读取整 数而指定位 置相当有用 。

关闭文件

要在应用服 务器上关闭 文件，请如 下使用 CLOSE DATASET 语句：

语法

```
CLOSE DATASET <dsn>.
```

此语句关闭 文件 <dsn> 。文件命名 在*打开文件* (页 205) 中有说明 。

只有在下次 为写入打开 文件（详细 信息和示例 ，参见*打开文件写 入* (页 207) ）期 间要删除文 件内容时， 关闭文件才 有必要。

为避免错误 并使程序易 于阅读，在 使用下一条 OPEN DATASET 语句之前应 关闭文件。通过 CLOSE 语句，可将 文件分成逻 辑块并易于 维护。

```
DATA FNAME(60) VALUE 'myfile'.
OPEN DATASET FNAME FOR OUTPUT.
  .....
CLOSE FNAME.
OPEN DATASET FNAME FOR INPUT.
  .....
CLOSE FNAME.
OPEN DATASET FNAME FOR INPUT AT POSITION <pos>.
  .....
CLOSE FNAME.
```

尽管 CLOSE 语句在此示 例中没有必 要，但它可 以改善程序 格式。

删除文件

要在应用服 务器上删除 文件，请如 下使用 DELETE DATASET 语句：

语法

```
DELETE DATASET <dsn>.
```

此语句删除 文件 <dsn> 。文件命名 在*打开文件* (页 205) 中有说明 。

如果系统可 以删除文件 <dsn> ，则 SY-SUBRC 返回 0，否则 SY-SUBRC 返回 4。

```
DATA FNAME(60) VALUE 'myfile'.
OPEN DATASET FNAME FOR OUTPUT.
OPEN DATASET FNAME FOR INPUT.
IF SY-SUBRC = 0.
  WRITE / 'File found'.
ELSE.
  WRITE / 'File not found'.
ENDIF.
DELETE DATASET FNAME.
OPEN DATASET FNAME FOR INPUT.
IF SY-SUBRC = 0.
  WRITE / 'File found'.
ELSE.
  WRITE / 'File not found'.
ENDIF.
```

此示例输出 如下：

```
File found
File not found
```

在此示例中 ，如果文件 “myfile” 不存在，则 在打开文件 以写入时创 建该文件。当打开文件 以写入时， 系统不能查 找此文件。 在 DELETE DATASET 语句之后， 系统不再查 找文件。

向文件中写 入数据

要向在应用 服务器上的 文件写入数 据，请如下 使用 TRANSFER 语句：

语法

```
TRANSFER <f> to <dsn> [LENGTH <len>].
```

此语句将字 段〈f〉的值写入文 件〈dsn〉。可以用 OPEN DATASET 语句指定传 输模式。如 果不能打开 文件以写入 ，则系统将 尝试用二进 制模式打开 文件，或为 此文件使用 OPEN DATASET 语句的选项 。但是只用 OPEN DATASET 语句打开文 件是良好的 编程风格。 OPEN DATASET 语句和文件 命名在*打开文件* (页 205) 中有 说明。字段〈f〉的数据类型 可以是基本 型，或者是 不包含作为 组件的内表 格的字段字 符串。内表 格不能在一 次执行中写 入文件。

通过 LENGHT 选项，可以 指定传输数 据的长度〈len〉。系统将第 一个〈len〉字节写入文 件。如果〈len〉太小，则截 断超出的字 节。如果〈len〉太大，系统 将在传输行 的右端各位 填入空格。

下列程序将 演示如何向 文件中写入 内表格：

```
DATA FNAME(60) VALUE 'myfile'.

TYPES: BEGIN OF LINE,
         COL1 TYPE I,
         COL2 TYPE I,
       END OF LINE.

TYPES  ITAB TYPE LINE OCCURS 10.

DATA: LIN TYPE LINE,
      TAB TYPE ITAB.

DO 5 TIMES.
  LIN-COL1 = SY-INDEX.
  LIN-COL2 = SY-INDEX ** 2.
  APPEND LIN TO TAB.
ENDDO.

OPEN DATASET FNAME FOR OUTPUT.
LOOP AT TAB INTO LIN.
  TRANSFER LIN TO FNAME.
ENDLOOP.
CLOSE DATASET FNAME.

OPEN DATASET FNAME FOR INPUT.
DO.
  READ DATASET FNAME INTO LIN.
  IF SY-SUBRC <> 0.
    EXIT.
  ENDIF.
  WRITE: / LIN-COL1, LIN-COL2.
ENDDO.

CLOSE DATASET FNAME.
```

此示例输出 如下：

```
1          1
2          4
3          9
4         16
5         25
```

在此示例中 ，以行类型 LINE 创建字段字 符串 LIN 与内表格 TAB。在 填完内表格 TAB 之后，它将 逐行写入文 件“ myfile ”。然后将 文件读入字 段字符串 LIN 并将 LIN 内容写到输 出屏幕。

为运行在 UNIX 系统（使用 ASCII 表示法）下 的 SAP 系统编写如 下演示程序 。

```
DATA FNAME(60) VALUE 'myfile'.

DATA: TEXT1(4) VALUE '1234',
      TEXT2(8) VALUE '12345678',
      HEX  TYPE X.
```

```
OPEN DATASET FNAME FOR OUTPUT IN TEXT MODE.
TRANSFER: TEXT1 TO FNAME LENGTH 6,
          TEXT2 TO FNAME LENGTH 6.
CLOSE DATASET FNAME.

OPEN DATASET FNAME FOR INPUT.
DO.
  READ DATASET FNAME INTO HEX.
  IF SY-SUBRC <> 0.
    EXIT.
  ENDIF.
  WRITE  HEX.
ENDDO.
CLOSE DATASET FNAME.
```

此示例输出 如下：

```
31 32 33 34 20 20 0A 31 32 33 34 35 36 0A
```

在此示例中 ，将两个字 符串 TEXT1 和 TEXT2 按文本模式 写入文件“ myfile” 。输出长度 值指定为 6。通过将 文件逐字节 地读入十六 进制字段， 可以看到如 何存储文件 ：数值 31 ~ 36 是数值字符 1 ~ 6 的 ASCII 表示法，20 代表空格并 且 0A 用作行结束 标志。从 TEXT1 右端开始填 入两个空格 ，并将 TEXT2 的两个位置 截掉。

从文件中读 取数据

要从应用服 务器上的文 件中读取数 据，请如下 使用 READ DATASET 语句：

语法

```
READ DATASET <dsn> INTO <f> [LENGTH <len>].
```

此语句从文 件 <dsn> 中读取数据 赋值给变量 <f> 。要决定把 从文件中读 取的数据赋 值给哪个变 量，必须清 楚文件结构 。

可以用 OPEN DATASET 语句指定传 输模式。如 果没有打开 文件以读取 ，则系统将 尝试以二进 制模式打开 文件或为此 文件使用 OPEN DATASET 语句的选项 。但是只用 OPEN DATASET 语句打开文件是良好的 编程风格。 OPEN DATASET 语句和文件 命名在*打开文件* (页 205) 中有 说明。

在读取操作 成功后，SY-SUBRC 将返回 0。当到文 件末尾时， SY-SUBRC 将返回 4。当不能 打开文件时 ， SY-SUBRC 将返回 8。

如果以二进 制模式工作 ，则可以使 用 LENGTH 选项查找实 际传输给字 段 <f> 的数据长度 。系统可用 变量 <len> 值设置该长 度。

```
DATA FNAME(60) VALUE 'myfile'.

DATA: TEXT1(12) VALUE 'abcdefghijkl',
      TEXT2(5),
      LENG TYPE I.

OPEN DATASET FNAME FOR OUTPUT IN BINARY MODE.
TRANSFER TEXT1 TO FNAME.
CLOSE DATASET FNAME.

OPEN DATASET FNAME FOR INPUT IN BINARY MODE.
DO.
  READ DATASET FNAME INTO TEXT2 LENGTH LENG.
  WRITE: / SY-SUBRC, TEXT2, LENG.
  IF SY-SUBRC <> 0.
    EXIT.
  ENDIF.
ENDDO.
CLOSE DATASET FNAME.
```

此示例输出 如下：

```
    0 abcde         5
    0 fghij         5
    4 kl###         2
```

在此示例中 ，用字段 TEXT1 的 12 个字节填写 文件“myfile” 。然后将读 取 5 个字节赋值 给字段 TEXT2 。注意：在 到文件末尾 之后系统将 用空格填写 TEXT2 的后三个字 节，在字段 LENG 中给出实际 传输的字节 数。

如果以文本 模式工作， 可以使用 LENGTH 选项查找文 件当前行的 实际长度。 系统将用变 量 <len> 值设置行长 度。为此， 系统将计算 文件中从当 前位置到行 结束标志符 之间的字节 数。

```
DATA FNAME(60) VALUE 'myfile'.

DATA: TEXT1(4) VALUE '1234    ',
      TEXT2(8) VALUE '12345678',
      TEXT3(2),
      LENG TYPE I.

OPEN DATASET FNAME FOR OUTPUT IN TEXT MODE.
   TRANSFER: TEXT1 TO FNAME,
             TEXT2 TO FNAME.
CLOSE DATASET FNAME.

OPEN DATASET FNAME FOR INPUT IN TEXT MODE.
   DO 2 TIMES.
      READ DATASET FNAME INTO TEXT3 LENGTH LENG.
      WRITE: / TEXT3, LENG.
   ENDDO.
CLOSE DATASET FNAME.
```

此示例输出 如下：

```
12          4

12          8
```

在此示例中 ，将字符串 TEXT1 和 TEXT2 按文本模式 写入文件“ myfile” 。然后读取 它们并赋值 给长度 2 的字符串 TEXT3 。行的存储 长度将写入 字段 LENG 。

使用平台独 立的文件名

在 ABAP/4 语句中为使 用文件指定 的文件名是 物理文件名 。这意味着 它们恰好满 足运行有 SAP 系统的操作 系统的要求 。按特定文 件名和路径 从 ABAP/4 程序中创建 文件后，在 操作系统级 登录后，可 发现此文件 恰好位于该 位置。

在各操作系 统之间，因 为文件和路 径的命名规 则不同，除 非如下使用 工具，否则 ABAP/4 程序就不能 从一个操作 系统移植到 另一个操作 系统。

要使程序可 以移植，SAP 系统提供了 逻辑文件名 和逻辑路径 的概念。逻 辑文件和路 径连接到物 理文件和路 径。在特殊 表格中建立 这种连接。 可以根据需 要维护这些 表格。在 ABAP/4 程序中，请 使用功能模 块 FILE_GET_NAME 从逻辑文件 名中创建物 理文件名。

平台独立的 文件名维护 是定制的一 部分。通过 选择“工具 -> 定制 -> 实现工程 -> 显示完整的 IMG.” 可以发现综 合说明。在 出现的屏幕 上，选择“ 基础 -> 常规基础 -> 平台独立的 文件名分配 ”。

关于功能模 块 FILE_GET_NAME 的说明，请 在“ABAP/4 功能库：简 介”屏幕上 输入其名称 并选择文档 。在出现的 屏幕上，单 击“功能模 块文档”。

下列主题提 供了关于如 何使用平台 独立文件名 的简要概述 。

首先解释上 面提到的表 格维护，您 将了解：

然后将了解 如何使用功 能模块 FILE_GET_NAME 以将 ABAP/4 程序中的逻 辑文件名转 换成物理文件名。

维护语法组

语法组包含 所有操作系 统（遵循相 同的文件命 名语法）的 名称。要显 示或创建语 法组，请调用事务 SF05 ，将出现下 列屏幕：

在此屏幕上 ，可以看到 可用语法组 的列表。如 果需要，可 通过单击“ 新条目”定 义新的语法组。关于如 何将操作系 统分配到语 法组的详细 信息，参见*将操作系统 分配到语法 组*（页 187）。

将操作系统 分配到语法 组

调用事务 SFO4 以显示或维 护操作系统 到现有语法 组的分配， 将出现如下 屏幕：

在此屏幕上 ，可以看到 当前受支持 的操作系统 列表。要创 建新条目， 请单击“新 条目”。要 将现有条目 分配到语法 组，请标志 操作系统并 单击“详情 ”。将出现 如下屏幕：

在此屏幕上 ，操作系统 HP-UX 分配到语法 组 UNIX 。

创建和定义 逻辑路径

每个逻辑文 件名都可与 逻辑路径连 接。逻辑路 径给逻辑文 件名提供平 台相关的物 理路径。要创建逻辑路 径，请调用 事务 SFO2 。在出现的 屏幕上，请 选择“新条 目”并定义 新的逻辑路 径，如下例 所示：

通过选择“ 保存”，保 存逻辑路径 。

调用事务 SFO3 以将现有逻 辑路径与物 理路径和语 法组连接。 在出现的屏 幕上，请选 择逻辑路径 维护它与语 法组的连接 ，或单击“ 新条目”以 创建新连接 。例如，语 法组 UNIX 的物理路径 可按如下定 义：

对于输入字 段，可以使 用可能条目 的“逻辑路 径和语法组 ”列表。要 创建“物理 路径”，可 使用运行时 用当前值替 换的保留字 （用尖括弧 括起）。将 光标放在输 入字段上并 选择“帮助 ”，可以获 得保留字列 表。在物理 路径中必须 包括保留字 <FILENAME> 。它的实际 值由使用逻 辑路径的逻 辑文件名定 义。在当前 示例中使用 的保留字 <PARAM_1> 的实际值是 功能模块 FILE_GET_NAME 的输入参数 。

创建和定义 逻辑文件名

要创建逻辑 文件名，请 调用事务 SFO1 并在出现的 屏幕上选择 “新条目” 。定义逻辑 文件名如下 例所示：

如上所述， 既可以将逻 辑文件与逻 辑路径连接 ，也可以在 输入字段“ 物理文件” 中输入物理文件名全称 。在后一种 情况下，逻 辑文件名仅 适用于一种 操作系统。 输入完整物 理文件的规则与为逻辑 路径定义物 理路径的规 则相同。有 关详细信息 和可能保留 字的列表， 请选择“帮助”。

当连接到逻 辑路径时， 逻辑文件适 用于所有为 逻辑路径维 护的语法组 。输入到“ 物理文件” 中的文件名 将替换分配 到逻辑路径 的物理路径 中的保留字 <FILENAME> 。要保持此 名称独立于操作系统， 请选择以字 母开始的名 称，该名称 最多由 8 个字符组成 且不包含特 殊字符。

通过选择“ 保存”，保 存定义。

在ABAP/4 程序中使用 逻辑文件

可以使用功 能模块 FILE_GET_NAME 从 ABAP/4 程序中的逻 辑文件名创 建物理文件 名。要在程 序中插入调 用功能模块 ，请在 ABAP/4 编辑器屏幕 中选择“编 辑 -> 插入语句” 。在出现的 对话框窗口 中，标志“ 调用功能” 并键入 FILE_GET_NAME。 该功能模块 的参数列表 如下。

输入参数

参数	功能
CLIENT	逻 辑文件和路 径的维护表 格是客户相 关的。所以 可输入所需 客户。当前 客户存储在 系统字段 SY-MANDT 中。

LOGICAL_FILENAME	将 要转换的大 写字母输入 逻辑文件名 。
OPERATING_SYSTEM	可 以输入事务 SF04 列表中包含 的任何操作 系统（参见*将操作系统分配到语法 组*（页 217））。根据与 操作系统连 接的语法组 创建物理文 件名。默认 参数是系统 字段 SY-OPSYS 的值。
PARAMETER_1 PARAMETER_2	如 果指定这些 输入参数， 则物理路径 名中的保留 字 <PARAM_1> 和 <PARAM_2> 将由输入值 替换。
USE_PRESENTATION _SERVER	用 此标志可决 定是否输入 显示服务器 的操作系统 而不是由参 数 OPERATING_SYSTEM 输入的操作 系统。
WITH_FILE_EXTENSION	如 果设置此标 志不等于 SPACE ，则为逻辑 文件名定义 的文件格式将附加到物 理文件名。

输出参数

参数	功能
EMERGENCY_FLAG	如 果此参数不 等于 SPACE，则在逻辑 路径中不定 义物理名称 。紧急物理 名称可从表 格 FILENAME 和参数文件 参数创建。
FILE_FORMAT	此 参数是为逻 辑文件名定 义的文件格 式。例如， 可以使用此 参数决定按 何种模式打 开文件。
FILE_NAME	此 参数是物理 文件名（要 使用文件， 可以通过 ABAP/4 语句使用此物理文件名 ）。

例外参 数

参数	功能
FILE_NOT_FOUND	如 果没有定义 逻辑文件， 则出现此例 外。
OTHERS	如 果发生其它 错误，则出 现此例外。

假定逻辑文 件 MYTEMP 和逻辑路径 TMP_SUB 在前述主题 中已定义， 并假定下列 程序：

```
DATA: FLAG,
      FORMAT(3),
      FNAME(60).

WRITE SY-OPSYS.

CALL FUNCTION 'FILE_GET_NAME'

     EXPORTING
          LOGICAL_FILENAME        = 'MYTEMP'
          OPERATING_SYSTEM        = SY-OPSYS
          PARAMETER_1             = '01'

     IMPORTING
          EMERGENCY_FLAG          = FLAG
          FILE_FORMAT             = FORMAT
          FILE_NAME               = FNAME

     EXCEPTIONS
          FILE_NOT_FOUND          = 1
          OTHERS                  = 2.

IF SY-SUBRC = 0.
  WRITE: / 'Flag      :', FLAG,
         / 'Format    :', FORMAT,
         / 'Phys. Name:', FNAME.
ENDIF.
```

此程序输出 如下：

```
HP-UX
FLAG      :
FORMAT    : BIN
Phys. Name: /tmp/TEST01
```

在此示例中 ，SAP 系统运行于 操作系统 HP-UX 之下，该操 作系统是 UNIX 语法组的成 员之一。如 同为 UNIX 语法组定义 的那样，逻 辑文件名 MYTEMP 和逻辑路径 TMP_SUB 一起转换为 物理文件名 /tmp/TEST01 。字段 FNAME 可用于以后 的程序流以使用目录 /tmp 中的文件 TEST01 。

假定具有物 理文件名 TEST 的逻辑文件 名 EMPTY 与没有指定 物理路径的 逻辑路径连接。如果在 上例中用 ‘ EMPTY’ 替换输出参 数‘MYTEMP’ ，则输出结 果如下：

HP-UX

FLAG : X

FORMAT :

Phys. Name: /usr/sap/S11/SYS/global/TEST

系统创建紧 急文件名， 其路径取决 于当前 SAP 系统的安装 。

使用演示服 务器上的文 件

要使用演示 服务器上的 文件，请使 用功能库中 提供的特殊 功能模块。 必须使用内 表格作为程序与功能模 块之间的接 口。下列主 题说明：

物理文件名 取决于演示 服务器的操 作系统。如 同 *使用平台独 立的文件名* (页 216) 所述，可以 使用逻辑文 件名编写平 台相关的 ABAP/4 程序。

通过用户对 话向演示服 务器写入数 据

要通过用户 对话从内表 格向演示服 务器写入数 据,请使用 功能模块 DOWNLOAD。最重要的参 数列表如下 。有关详细 信息，参见 事务 SE37 中的功能模 块文档。

重要输入参 数

参数	功能
BIN_FILESIZE	二 进制文件长 度
CODEPAGE	仅 用于在 DOS 中下载：值 IBM
FILENAME	文 件名（用户 对话的默认 值）
FILETYPE	文 件类型（用 户对话的默 认值）
ITEM	用 户对话窗口 标题
MODE	写 入模式（ ‘ empty’ = 改写， ‘A’ = 追加）

用 FILETYPE 可指定传输 模式。可能 值有：

BIN
二进制文件 ：必须指定 文件长度， 并且内表格 必须包含有 数据类型 X 的一列。
ASC
ASCII 文件.
DAT
Excel 文件：列由 定位符分隔 ，行由回车 符和换行码 分隔。
WK1
Excel 文件和 Lotus 文件：数据 将写入 WK1 电子表格。

重要输出参 数

参数	功能
ACT_FILENAME	文 件名（在用 户对话期间 输入）
ACT_FILETYPE	文 件类型（在 用户对话期 间输入）
FILESIZE	传 输字节数

表 格参数

参数	功能
DATA_TAB	内 部源表格

例 外参数

参数	功能
INVALID_FILESIZE	无 效参数 BIN_FILESIZE
INVALID_TABLE_WIDTH	无 效表格结构
INVALID_TYPE	参 数 FILETYPE 无效值

假定表示所 用的操作系 统是 WINDOWS NT 并假定下列 程序：

```
PROGRAM SAPMZTST.

DATA: FNAME(128), FTYPE(3), FSIZE TYPE I.

TYPES: BEGIN OF LINE,
         COL1 TYPE I,
         COL2 TYPE I,
       END OF LINE.

TYPES  ITAB TYPE LINE OCCURS 10.

DATA: LIN TYPE LINE,
      TAB TYPE ITAB.

DO 5 TIMES.
   LIN-COL1 = SY-INDEX.
   LIN-COL2 = SY-INDEX ** 2.
   APPEND LIN TO TAB.
ENDDO.

CALL FUNCTION 'DOWNLOAD'

     EXPORTING
        CODEPAGE            = 'IBM'
        FILENAME            = 'd:\temp\saptest.xls'
        FILETYPE            = 'DAT'
        ITEM                = 'Test for Excel File'

     IMPORTING
        ACT_FILENAME        = FNAME
        ACT_FILETYPE        = FTYPE
        FILESIZE            = FSIZE

     TABLES
        DATA_TAB            = TAB

     EXCEPTIONS
        INVALID_FILESIZE    = 1
        INVALID_TABLE_WIDTH = 2
        INVALID_TYPE        = 3.

WRITE: 'SY-SUBRC:', SY-SUBRC,
     / 'Name    :', (60) FNAME,
     / 'Type    :', FTYPE,
     / 'Size    :', FSIZE.
```

启动程序后 ，将出现如 下用户对话 窗口：

在此窗口中 ，用户可更 改默认值。 在单击“确 定”后，系 统将从内表 格 TAB （已在程序 中填入数据 ）向文件 d:\temp\saptest.xls 传送数据。 如果该文件 已经存在，则系统询问 用户是否替 换此文件。 系统将在列 之间设置分 隔符，以及 在每行结尾设置回车符 和换行。

此示例输出 如下：

```
SY-SUBRC:     0
Name    : d:\temp\saptest.xls
Type    : DAT
Size    :         27
```

在演示服务 器上可以从 Excel 打开 d:\temp\saptest.xls 文件。出现 如下 Excel 屏幕：

不通过用户 对话向演示 服务器写入 数据

在不使用用 户对话的情 况下，要向 演示服务器 写入数据， 请使用功能 模块 WS_DOWNLOAD 。最重要的 参数列表如 下。有关详 细信息，参 见事务 SE37 中的功能模 块文档。

重要输入参 数

参数	功能
BIN_FILESIZE	二 进制文件长 度
CODEPAGE	仅 在 DOS 中下载：值 IBM
FILENAME	文 件名
FILETYPE	文 件类型
MODE	写 模式（‘empty’ = 改写，‘A’ = 追加）

使用 FILETYPE 可以指定传 输模式。可 能值有：

BIN
二进制文件 ：必须指定 文件长度， 并且内表格 必须包含有 数据类型 X 的一列。
ASC
ASCII 文件，系统 在每行之后 设置行结束 标志。
DAT
Excel 文件：系统 用定位符分 隔列，用回 车符和换行 码分隔行。
WK1
Excel 文件和 Lotus 文件：数据 将写入 WK1 电子表格。

输出参数

参数	功能
FILELENGTH	传 输字节数

表 格参数

参数	功能
DATA_TAB	内 部源表格

例 外参数

参数	功能
FILE_OPEN_ERROR	系 统不能打开 文件
FILE_WRITE_ERROR	系 统不能向文 件写入数据
INVALID_FILESIZE	无 效的参数 BIN_FILESIZE
INVALID_TABLE_WIDTH	无 效的表格结 构
INVALID_TYPE	无 效的参数 FILETYPE 值

假定用于表 示的操作系 统是 WINDOWS NT 并假定下列 程序：

```
PROGRAM SAPMZTST.
DATA: FLENGTH TYPE I.
DATA TAB(80) OCCURS 5.
```

```
APPEND 'This is the first line of my text. ' TO TAB.
APPEND 'The second line.                   ' TO TAB.
APPEND '      The third line.              ' TO TAB.
APPEND '          The fourth line.         ' TO TAB.
APPEND '            Fifth and final line. ' TO TAB.

CALL FUNCTION 'WS_DOWNLOAD'

     EXPORTING
        CODEPAGE            = 'IBM'
        FILENAME            = 'd:\temp\saptest.txt'
        FILETYPE            = 'ASC'

     IMPORTING
        FILELENGTH          = FLENGTH

     TABLES
        DATA_TAB            = TAB

     EXCEPTIONS
        FILE_OPEN_ERROR     = 1
        FILE_WRITE_ERROR    = 2
        INVALID_FILESIZE    = 3
        INVALID_TABLE_WIDTH = 4
        INVALID_TYPE        = 5.

WRITE: 'SY-SUBRC   :', SY-SUBRC,
     / 'File length:', FLENGTH.
```

此示例输出 如下：

```
SY-SUBRC   :      0
File length:        140
```

系统将向 ASCII 文件 d:\temp\saptest.txt. 传输表格 TAB 的五行。使 用 WINDOWS “ 文件管理器 ”，如下所 示可以检查 文件的存在 及长度：

现在可以使 用演示服务 器上的任何 编辑器打开 文件。在下 示例中，使 用 DOS 编辑器：

通过用户对 话从演示服 务器中读取 数据

要通过用户 对话将数据 从演示服务 器读取到给 内表格中， 请使用功能 模块 UPLOAD 。最重要的 参数列表如 下。有关详 细信息，参 见事务 SE37 中的功能模 块文档。

重要输入参 数

参数	功能
CODEPAGE	仅 在 DOS 中加载：值 IBM
FILENAME	文 件名（用户 对话默认值 ）
FILETYPE	文 件类型（用 户对话默认 值）
ITEM	用 户对话窗口 标题

使用 FILETYPE 可以指定传 输模式。可 能值有：

BIN
二进制文件 。
ASC
ASCII 文件：具有 行结束标志 的文本文件 。
DAT
作为文本文 件保存的 Excel 文件，列由 定位符分隔 ，行由回车 符和换行码 分隔。
WK1
作为 WK1 电子表格保 存的 Excel 文件和 Lotus 文件。

重要输出参 数

参数	功能

FILESIZE	传 输字节数
ACT_FILENAME	文 件名（在用 户对话期间 输入）
ACT_FILETYPE	文 件类型（在 用户对话期 间输入）

表 格参数

参数	功能
DATA_TAB	内 部目标表格

例外参数

参数	功能
CONVERSION_ERROR	在 数据转换中 的错误
INVALID_TABLE_WIDTH	无 效的表格结 构
INVALID_TYPE	不 正确参数 FILETYPE

假定用于表 示的操作系 统是 WINDOWS NT 并假定如下 Excel 表格：

如果此表格 作为文本文 件（列之间 有定位符） 保存到 d:\temp\mytable.tab， 则下列程序 可以读取表 格：

```
PROGRAM SAPMZTST.

DATA: FNAME(128), FTYPE(3), FSIZE TYPE I.

TYPES: BEGIN OF LINE,
         COL1(10) TYPE C,
         COL2(10) TYPE C,
         COL3(10) TYPE C,
       END OF LINE.

TYPES  ITAB TYPE LINE OCCURS 10.

DATA: LIN TYPE LINE,
      TAB TYPE ITAB.

CALL FUNCTION 'UPLOAD'

     EXPORTING
          CODEPAGE            = 'IBM'
          FILENAME            = 'd:\temp\mytable.tab'
          FILETYPE            = 'DAT'
          ITEM                = 'Read Test for Excel File'

     IMPORTING
          FILESIZE            =  FSIZE
          ACT_FILENAME        =  FNAME
          ACT_FILETYPE        =  FTYPE

     TABLES
          DATA_TAB            =  TAB

     EXCEPTIONS
          CONVERSION_ERROR    = 1
          INVALID_TABLE_WIDTH = 2
          INVALID_TYPE        = 3.

WRITE: 'SY-SUBRC:', SY-SUBRC,
     / 'Name    :', (60) FNAME,
     / 'Type    :', FTYPE,
     / 'Size    :', FSIZE.

SKIP.
LOOP AT TAB INTO LIN.
   WRITE: / LIN-COL1, LIN-COL2, LIN-COL3.
ENDLOOP.
```

在启动此程 序后，将出 现用户对话 窗口：

在此窗口中 ，用户可以 更改默认值 。单击“确 定”后，系 统将数据从 文件 d:\temp\mytable.tab 传输到内表 格 TAB 。

此示例输出 如下：

```
SY-SUBRC:     0
Name    : d:\temp\mytable.tab
Type    : DAT
Size    :         69

Billy       the         Kid
My          Fair        Lady
Herman      the         German
Conan       the         Barbarian
```

内表格 TAB 的内容等于 初始 Excel 表格的内容 。

不通过用户 对话从演示 服务器中读 取数据

不通过用户 对话，要将 数据从演示 服务器读取 到内表格， 请使用功能 模块 WS_UPLOAD 。最重要的 参数列表如 下。有关详 细信息，参 见事务 SE37 中的功能模 块文档。

重要输出参 数

参数	功能
CODEPAGE	仅 在 DOS 中下载：值 IBM
FILENAME	文 件名
FILETYPE	文 件类型

使用 FILETYPE 可以指定传 输模式。可 能值有：

BIN
二进制文件 。
ASC
ASCII 文件：具有 行结束标志 的文本文件 。
DAT
作为文本文 件保存的 Excel 文件，列由 定位符分隔 ，行由回车 符和换行码 分隔。
WK1
作为 WK1 电子表格保 存的 Excel 文件和 Lotus 文件。

输出参数

参数	功能
FILELENGTH	传 输字节数

表 格参数

参数	功能
DATA_TAB	内 部目标表格

例外参数

参数	功能
CONVERSION_ERROR	在 数据转换中 的错误
FILE_OPEN_ERROR	系 统不能打开 文件
FILE_READ_ERROR	系 统不能从文 件中读取数 据
INVALID_TABLE_WIDTH	无 效的表格结 构
INVALID_TYPE	无 效的参数 FILETYPE 值

假定用于表 示的操作系 统是 WINDOWS NT 并假定如下 文本文件：

下列程序将 读取该文本 文件：

```
PROGRAM SAPMZTST.

DATA: FLENGTH TYPE I.

DATA: TAB(80) OCCURS 5 WITH HEADER LINE.

CALL FUNCTION 'WS_UPLOAD'

     EXPORTING
          CODEPAGE            = 'IBM'
          FILENAME            = 'd:\temp\mytext.txt'
          FILETYPE            = 'ASC'

     IMPORTING
          FILELENGTH          =  FLENGTH

     TABLES
          DATA_TAB            =  TAB

     EXCEPTIONS
          CONVERSION_ERROR    = 1
          FILE_OPEN_ERROR     = 2
          FILE_READ_ERROR     = 3
          INVALID_TABLE_WIDTH = 4
          INVALID_TYPE        = 5.

WRITE: 'SY-SUBRC:', SY-SUBRC,
     / 'Length  :', FLENGTH.

SKIP.
LOOP AT TAB.
  WRITE: / TAB.
ENDLOOP.
```

此示例输出 如下：

检查演示服 务器上的文 件

要获取演示 服务器上的 文件和演示 服务器操作 系统的信息 ，请使用功 能模块 WS_QUERY 。最重要的 参数列表如 下。有关详 细信息，参 见事务 SE37 中的功能模 块文档。

重要输入参 数

参数	功能
FILENAME	查 询命令‘FE’ 、‘FL’ 、和‘DE’ 的文件名
QUERY	查 询命令

输入 参数 QUERY 定义查询命 令。一些重 要查询命令 列表如下：

CD：查 询当前目录
EN：查 询环境变量
FL：查 询用 FILENAME 指定文件的 长度
FE：查 询用 FILENAME 指定文件的 存在性
DE：查 询用 FILENAME 指定目录的 存在性
WS：查 询演示服务 器的窗口系 统
OS：查 询演示服务 器的操作系 统

输出参数

参数	功能
RETURN	查 询结果（ ‘ 0’表示‘ no’， ‘ 1’表示‘ yes’）

例外参数

参数	功能
INV_QUERY	QUERY 或 FILENAME 的错误值

假定用于表 示的操作系 统是 WINDOWS NT,并假 定文件 SYSTEM.INI 存在(如下 所示):

下述程序决 定操作系统 及此文件的 属性:

```
PROGRAM SAPMZTST.

DATA:  FNAME(60), RESULT(30), FLENGTH TYPE I.

FNAME = 'C:\WINNT35\SYSTEM.INI'.

CALL FUNCTION 'WS_QUERY'
     EXPORTING
          QUERY       = 'OS'
     IMPORTING
          RETURN      = RESULT
     EXCEPTIONS
          INV_QUERY   = 1.

IF SY-SUBRC = 0.
  WRITE: / 'Operating System:', RESULT.
ENDIF.

CALL FUNCTION 'WS_QUERY'
     EXPORTING
          QUERY       = 'WS'
     IMPORTING
          RETURN      = RESULT
     EXCEPTIONS
          INV_QUERY   = 1.

IF SY-SUBRC = 0.
  WRITE: / 'Windows:', RESULT.
ENDIF.

CALL FUNCTION 'WS_QUERY'
     EXPORTING
          FILENAME    = FNAME
          QUERY       = 'FE'
     IMPORTING
          RETURN      = RESULT
     EXCEPTIONS
          INV_QUERY   = 1.

IF SY-SUBRC = 0.
  WRITE: / 'File exists ?', RESULT.
ENDIF.

CALL FUNCTION 'WS_QUERY'
     EXPORTING
          FILENAME    = FNAME
          QUERY       = 'FL'
     IMPORTING
          RETURN      = FLENGTH
     EXCEPTIONS
          INV_QUERY   = 1.

IF SY-SUBRC = 0.
  WRITE: / 'File Length:', FLENGTH.
ENDIF.
```

此程序输出 如下:

```
Operating System: NT

Windows: WN32

File exists ? 1

File Length:        210
```

WN32 窗口系统是 WINDOWS NT 的窗口系统 。关于缩写 的详细信息 ，请将光标 放在功能模 块文档屏幕 的 QUERY 字段上并选 择“帮助” 。

第二部分　编写ABAP/4报表

目　　录

第一章 用逻辑数据库访问数据库表

概览

内容

可采用两种方法，用报表程序访问数据库表的数据进行分析。
下列主题包括

关于逻辑数据库的功能和维护的信息，即如何显示、更改或创建逻辑数据库（事务 SLDB 或 SE36），参见 *逻辑数据库 的特征和维护*（页 Error! Not a valid link.）。

用 SELECT 访问数据

SELECT 语句是 SAP 开放式 SQL （标准 SQL 的一个子集）的一部分。关于 SELECT 语句的详细信息，参见 *从数据库表读取数据*（页 **错误！链接无效。**）。
用 SELECT 语句及其不同子句，可以读取和分析 SAP 系统已知的所有数据库表的数据。仅使用 SELECT 语句的报表程序称为 SQL 报表。
在 SQL 报表中，不一定需要事件的外部流控制。如果不使用事件关键词（参见 *通过事件控制 ABAP/4 程序流*（页 Error! Not a valid link.）），通常可以将 SQL 报表看作连续处理的程序。
可是，如果要使用逻辑数据库访问数据，则必须使用事件。

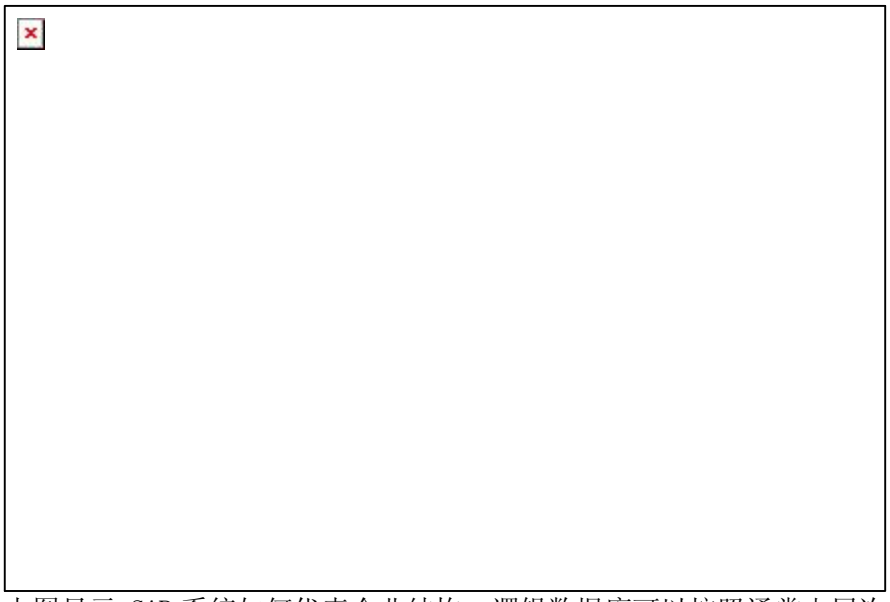
用逻辑数据库访问数据

逻辑数据库访问数据库表中数据的方法与采用 SELECT 语句访问数据的方法有明显差别。

逻辑数据库是特殊的 ABAP/4 程序，将一定数据库表的内容组合在一起。可以将逻辑数据库链接为 ABAP/4 报表程序的属性。这样，逻辑数据库就可向报表程序提供一组层次结构表格行。该层次结构表格行可从不同数据库表中提取。

> “逻辑数据库”不仅指程序本身，还指它供应的数据集。

在 SAP 系统中，许多表格都由外来关键字相关链接（详细信息，参见文档 *ABAP/4 词典*（页 Error! Not a valid link.））。其中部分相关性形成树状层次结构。使用逻辑数据库有助于读取形成此类结构组件的数据表格。



上图显示 SAP 系统如何代表企业结构。逻辑数据库可以按照通常由层次结构定义的顺序逐行将表格行读入报表程序中。

ABAP/4 开发工作台包括创建和显示逻辑数据库的便利工具（既可调用事务 SLDB，也可选择“工具 -> ABAP/4 开发工作台 -> 开发 -> 编程环境 -> 逻辑数据库”）。要查看逻辑数据库 <ldb> 的层次结构，请在 ABAP/4 编辑器的命令区中键入 SHOW DATABASE <ldb>。关于维护逻辑数据库的详细信息，参见 *逻辑数据库 的特征和维 护*（页 Error! Not a valid link.）。

将逻辑数据库链接到报表程序以访问数据时，顺序程序流已不再满足要求。相反，必须根据事件编写一个顺序（参见 *ABAP/4 中 流控制的概 念*（页 **错误！链接无效。**）中的图表）。逻辑数据库提供报表程序的外部流控制事件。与逻辑数据库连接的最重要事件是 GET（参见 *事件及其事 件关键字*（页 Error! Not a valid link.））。

也可在已与逻辑数据库链接的报表程序中使用 SELECT 语句。

访问方法比较

下例对使用 SELECT 语句的报表程序和使用逻辑数据库的报表进行比较。本节中的所有示例都使用逻辑数据库 F1S。必须在程序属性中指定该名称（参见 *指令程序属 性*（页 **错误！链接无效。**））。逻辑数据库 F1S 的结构如下（编辑器命令 SHOW DATABASE F1S）：

下表对从层次结构数据库 SPFLI、SFLIGHT 和 SBOOK 中读取数据的两个报表程序进行比较。

用 SELECT 语句的报表	用逻辑数据库的报表
REPORT	REPORT .....

TABLES: SPFLI, SFLIGHT, SBOOK.	TABLES SPFLI, SFLIGHT,SBOOK.
SELECT * FROM SPFLI WHERE ..	GET SPFLI.
<processing block>	<processing block>
SELECT * FROM SFLIGHT WHERE ..	GET SFLIGHT.
<processing block>	<processing block>
SELECT * FROM SBOOK WHERE ..	GET SBOOK.
<processing block>	<processing block>
ENDSELECT.	
ENDSELECT.	
ENDSELECT.	

左栏的报表程序用嵌套的 SELECT 循环读取数据；右栏的报表程序则使用逻辑数据库 F1S。请注意，这两种情况都必须在程序中用 TABLES 语句对数据库表加以说明。

与左栏的报表程序一样，右栏的报表程序使用借助 SELECT 语句从数据库表读取数据的逻辑数据库程序。逻辑数据库程序的主要结构如下：

逻辑数据库程序
REPORT SAPDB...
TABLES: SPFLI, SFLIGHT, SBOOK.
SELECT * FROM SPFLI WHERE ..
SELECT * FROM SFLIGHT WHERE ..
SELECT * FROM SBOOK WHERE ..
.....
ENDSELECT.
ENDSELECT.
ENDSELECT.

总而言之，逻辑数据库程序用 SELECT 语句从数据库表读取数据。这意味着包含 SELECT 语句的代码从报表程序转出到逻辑数据库程序

关于逻辑数据库程序精确结构的信息，参见*逻辑数据库的数据库程序*（页 Error! Not a valid link.）。

逻辑数据库的优点

使用逻辑数据库可以免于编写从数据库表中检索数据的程序。在报表程序中，不必定义如何检索信息，只需说明任何在屏幕上显示该信息。在 GET 事件后的处理块中，只需指定分析数据和将结果写到屏幕的语句。

传递完每个表格行后，发生 GET 事件，报表程序通过激活相应的处理块就能对其进行处理。如果在程序属性中没有指定逻辑数据库，则永远不会发生 GET 事件。

使用逻辑数据库时，不必编写供用户输入信息的选择屏幕程序，因为这是自动创建的（参见 *选择屏幕和逻辑数据库*（页 Error! Not a valid link.））。但是，如果仅使用 SELECT 语句，则必须自己编写选择屏幕程序（参见 *使用选择屏幕*（页 Error! Not a valid link.））。

一个报表只能使用一个逻辑数据库，但是每个逻辑数据库可以由多个报表使用。这一点大大优于使用 SELECT 语句将数据库访问集成到每个报表程序中。这意味着对于相同访问路径，只用编写一次代码。这同样适用于编写授权检查代码。

使用逻辑数据库时，大多数报表可受益于下列特征：

容易使用的标准用户界面

用于检查用户输入完整性、正确性和真实性的检查功能

重要的数据选择
数据库访问的中央授权检查
保持由应用程序逻辑决定的层次数据视图时，具有良好的读取访问性能（例如，借助视图）

即便使用中央逻辑数据库时，也有很大的灵活性，因为能够：

创建和设计自己的选择屏幕版本
在 ABAP/4 报表程序中使用特定的事件关键字以便启用扩展的用户对话（例如，进行详细授权或真实性检查）。

从报表程序中控制数据库访问

报表程序能影响由逻辑数据库程序读取的数据量。可以随时对当前正处理的数据进行动态检查，以确保满足任何选择标准。如果不是这种情况，逻辑数据库跳过层次结构的下级部分。如果不选择选定特定表进行处理，系统自动忽略树的相关部分

如果系统从逻辑数据库表读取值，则会发生 GET 事件。在报表程序中，可以给 GET 分配一个处理块。例如，如果系统从表 SPFLI 中读取值，则执行分配给 GET SPFLI 的处理块。不必为表格用于定址字段的每个报表定义处理块。不论是否定义处理块，系统都读取表格。但是，如果数据库表是专为逻辑数据库中的字段选择设计的，则系统行为会有所不同（参见 *GET 〈表格〉*（页 **Error! Not a valid link.**））。

处理块既包含已读表格字段，也包含当前访问路径上所有超级表格字段。例如，在事件 GET SBOOK 中，可以定址表格 SPFLI、SFLIGHT 和 SBOOK 的字段。这是因为逻辑数据库程序用嵌套的 SELECT 循环读取表格（参见 *访问方法比 较*（页 230））。

但是，并非总是要从所有表格来访问这些字段，这样，每次读取属于逻辑数据库的所有表格都将浪费宝贵的 CPU 时间。因此，读取程度取决于位于层次结构最低等级处与表格相对应的 GET 语句。

假定要生成航班连接列表。此时系统只需从表格 SPFLI 读取字段。

```
TABLES: SPFLI.
  ...
  GET SPFLI.
```

系统不从表格 SBOOK 读取数据。

要获取与预定相关的文件，则需要表格 SPFLI 和 SBOOK。

```
TABLES: SPFLI, SBOOK.
  ...
  GET SBOOK.
```

系统也读取表格 SFLIGHT，因为它位于表格 SBOOK 的访问路径上。可是，如果用 TABLES 说明报表适当的表格工作区，则只能访问 SFLGHT 的数据。

第二章 使用选择屏 幕

概览

内容

选择屏幕是 报表程序的 一部分，您 可以设计选 择屏幕以便 交互输入字 段值和选择 标准。在报 表程序启动 之后，用户 在该屏幕中 输入值。关 于选择屏幕 概念的详细 信息，参见
关于选择屏 幕与逻辑数 据库之间关 系的简要介 绍，参见
在 ABAP/4 程序中，可 以使用下列 语句设计选 择屏幕：

PARAMETERS， 为变量定义 输入字段
SELECT-OPTIONS， 为选择标准 定义输入字 段
SELECTION-SCREEN， 格式化选择 屏幕

下列主题讨 论

选择屏幕是 什么？

正如您从 *声明数据* （页 **错误！链接无效。**）部分所知道 的，您可以 给内部变量 分配初始值 。但是，不 象 C 语言有 GETC， FORTRAN 语言有 READ， 或者 BASIC 语言有 INPUT， ABAP/4 没有关键字 可以允许您 在程序流的 任何一点上 交互地输入 值。
要创建交互 式的 ABAP/4 程序，必须 使用在*编写 ABAP/4 事务* （页 Error! Not a valid link.）. 中描述的对 话编程方法 。这意味着 您必须创建 事务、编写 ABAP/4 模块存储程 序，并用屏 幕绘制器和 菜单绘制器 设计屏幕（ 参见文档 *ABAP/4 工作台工具* （页 Error! Not a valid link.）。
通过选择屏 幕，ABAP/4 还为报表程 序提供了一 个交互元素 。您可以定 义选择屏幕 而不必影响 对话编程所 要求的所有 细节。例如 ，您可以用 简单语句创 建字段、复 选框或单选 按钮，并设 计屏幕布局 。系统自动 为您处理屏 幕绘制器的 实际任务。
总是在报表 程序启动后 直接处理选 择屏幕（参 见 *通过事件控制 ABAP/4 程序流* （页 Error! Not a valid link.））。用户可 以在该屏幕 中输入字段 值和选择标 准。对于每 个报表程序 ，输入值集 可以按变体 （参见 *使用变式预设置选择* （页 Error! Not a valid link.））创建和存 储。选择屏 幕上的文本 可以按依赖 于语言的文 字摘要（参 见 *选择文本* （页 **错误！链接无效。**））进行维护 。
选择屏幕的 主要目的是 使用户能够 控制报表程 序的数据库 选择。如果 报表程序是 用 SUBMIT 语句（参见*调用报表* （页 Error! Not a valid link.））从另一个 ABAP/4 程序中启动 的，则选择 屏幕对象还 起到数据接 口的作用，
通过在报表 程序中定义 的选择屏幕 ，您可以使 用户能够

通过 PARAMETERS 语句给变量 分配值
通过 SELECT-OPTIONS 语句确定选 择标准

逻辑数据库 程序通常也 包括 PARAMETERS 和 SELECT-OPTIONS 语句（参见 *逻辑数据库 的特征和 护*（页 **Error! Not a valid link.**））。如果您 写通过报表 属性与逻辑 数据库相连 接的报表程 序，报表选 择屏幕将自 动包括相应 的输入字段 。

您应该尽可 能广泛地使 用逻辑数据 库提供的选 择标准。只 有当它们不 能满足您的 需要时才在 您的程序中 使用 PARAMETERS 或 SELECT-OPTIONS 语句。由这 些语句定义 的输入字段 将在报表选 择屏幕上作 为逻辑数据 库的附加选 择标准出现 （示例参见 *避免将选择 标准传递到 逻辑数据库*（页 191））

选择屏幕与 逻辑数据库

每个报表程 序都与一个 逻辑数据库 相链接，该 逻辑数据库 决定选择屏 幕的格式。 如果在程序 属性中没有 指定逻辑数 据库，系统 将使用标准 数据库，该 数据库格式 化选择屏幕 ，但是不读 取任何数据 。

选择屏幕包 含逻辑数据 库选择和在 报表程序中 定义的选择 。为逻辑数 据库选择显 示的输入字 段取决于您 在程序中用 TABLES 语句声明的 数据库表。 标准数据库 的选择屏幕 只包含在报 表程序中定 义的选择。 关于逻辑数 据库选择的 详细信息， 参见 *逻辑数据库 的特征和 护*（页 **Error! Not a valid link.**）。

逻辑数据库 F1S 被附加到下 列报表程序 。F1S 的结构是：

假设下列报 表程序：

```
REPORT SAPMZTST.

TABLES SPFLI.
```

启动 SAPMZTST 后，选择屏 幕如下所示 ：

这些是选择 标准的输入 字段和数据 库表 SPFLI 各列的参数 。在逻辑数 据库程序中编码定义该 屏幕的语句 （ SELECT-OPTIONS 和 PARAMETERS ）。

现在，假设 下列报表程 序：

```
REPORT SAPMZTST.

TABLES SBOOK.
```

启动 SAPMZTST 后，选择屏 幕如下所示 ：

系统不仅显 示与数据库 表 SBOOK 相连的选择 标准的输入 字段，而且 显示那些与 表 SPFLI 和 SFLIGHT 相连的选择 标准。

出现在逻辑 数据库选择 屏幕中的输 入字段通常 是在逻辑数 据库程序中 用 SELECT-OPTIONS 语句（参见 *定义选择标 准*（页 213））定义的。 使用这些语 句来限制对 数据库的访 问，例如通 过 SELECT 语句的 WHERE 子句(参见 *在 WHERE 子句中使用 选择表*（页 185））。示例参 见 *逻辑数据库 示例*（页 **Error! Not a valid link.**）。

逻辑数据库 可以提供一 个特征以便 报表用户指 定动态选择 ，这些动态 选择在逻辑 数据库程序 中没有用 SELECT-OPTIONS 语句编码。 如果逻辑数 据库提供动 态选择，用 户通过单击 选择屏幕上 应用工具条 的“用户选 择”访问这 些选择。将 显示新的选 择屏幕或屏 幕，用户可 以在其中选 择为要其指 定选择标准 的数据库字 段。

对于逻辑数 据库 F1S，动 态选择的屏 幕可能如下 所示：

动态选择通 过使用动态 WHERE 条件限制对 逻辑数据库 程序(参见*运行时指 行选择的条 件*（页 **错误！链接无效。**））对数据库 进行访问。

必须在逻辑 数据库程序 中编码动态 选择的可能 性（参见 *编辑选择*（页 **Error! Not a valid link.**））。如果逻 辑数据库允 许动态选择 （“动态选 择”出现在 应用工具条 中），则可 以使用 ABAP/4 集成开发环 境定义允许 用户在哪些 数据库表和 列上定义动 态选择（选 择视图）。 要找出哪些 数据库表提 供动态选择 ，请选择“ 工具 -> ABAP/4 开发工具台 -> 开发 -> 编程环境 -> 逻辑数据库 -> 细节 -> 动态选择” 。在下一个 屏幕中，您 将看到这些 数据库表名 称的列表。

对于逻辑数 据库 F1S，只 有数据库表 SPFLI 提供动态选 择：

为变量定义 输入字段

如果您想使 报表程序的 用户能在选 择屏幕中输 入变量值， 则必须用 PARAMETERS 语句定义变 量。每个用 PARAMETERS 语句定义的 变量将在选 择屏幕上作 为输入字段 出现。
PARAMETERS 语句具有一 些在报表程 序环境中很 重要的变体 。这些变体 在下列主题 中进行解释 ：
在逻辑数据 库程序中使 用的 PARAMETERS 语句的附加 变体也是有 效的。(参 见关键字文 档和 *逻辑数据库 的特征和 护* (页 **Error! Not a valid link.**))。

PARAMETERS 语句基本格 式

可以按照用 DATA 语句（参见 *DATA 语句* (页 **错误！链接无效。**) ）声明字段 的方式用 PARAMETERS 语句声明字 段。用 PARAMETERS 语句声明的 字段称为参 数。通常， 输入字段的 所有参数都 将出现在选 择屏幕上。 系统处理输 入屏幕时， 报表用户在 这些输入字 段中键入的 值将被分配 给相应的参 数。
要声明参数 及其数据类 型，请使用 PARAMETERS 语句，如下 所示：

语法
PARAMETERS <p>[(<length>)] <type> [<decimals>].
该语句创建 参数 <p>。附 加项 <length>、 <type> 和 <decimals> 与 DATA 语句相同 (参见 *DATA 语句的基本格式* (页 **错误！链接无效。**))。
用户启动报 表程序时， <p> 的输入字段 将出现在选 择屏幕中。 您可以如 *选择文本* (页 **错误！链接无效。**) 中描述的那 样通过使用 文字摘要来 更改输入字 段左侧的说 明。

```
REPORT SAPMZTST.

TABLES SPFLI.

PARAMETERS: WORD(10) TYPE C,
            DATE TYPE D,
            NUMBER TYPE P DECIMALS 2,
            CONNECT LIKE SPFLI-CONNID.
```

该示例创建 四个字段， 即字符字段 WORD， 长度为 10；日期字段 DATE， 长度为 8；组合式数 据字段 NUMBER， 两位小数； 字段 CONNECT， 引用 ABAP/4词 典结构 SPFLI-CONNID。 用户启动报 表 SAPMZTST 时，声明的 四个字段的 输入字段将 出现在选择 屏幕中，如 下所示：

例如,您可 以使用参数 控制程序流 ,或者与 SELECT 语句相连接 ,使报表用 户能够决定 数据库访问 的选择标准 （参见 *选择即将读取的行* (页 **错误！链接无效。**))。

```
TABLES SPLFI.

PARAMETERS: LOW  LIKE SPFLI-CARRID,

            HIGH LIKE SPFLI-CARRID.

SELECT * FROM SPLFI WHERE CARRID BETWEEN LOW AND HIGH.
   ........
ENDSELECT.
```

在该示例中 ，系统从数 据库表 SPFLI 读入所有行 ，其中字段 CARRID 的内容在界 限 LOW 和 HIGH 之间。报表 用户可以在 选择屏幕中 输入 LOW 和 HIGH 字段。

如果使用 PARAMETERS 语句定义 WHERE 条件，则必 须在报表程 序中编码所 有可能的选 择选项。对 于复杂选择 ，最好使用 选择标准， 它们存储在 内表中。（ 参见 *选择标准是 什么？* （页 182））。

给参数分配 缺省值

要为将显示 在选择屏幕 上的输入字 段分配缺省 值，请使用 PARAMETERS 语句的 DEFAULT 选项。语法 如下：

语法
```
PARAMETERS <p> ..... DEFAULT <f> .....
```
<f> 可以是文字 或字段名。 如果指定字 段名，则系 统将按缺省 值处理该字 段的内容。 报表用户可 以在选择屏 幕中更改缺 省值。
系统在时间 事件 INITIALIZATION （参见 *INITIALIZATION* (页 **Error! Not a valid link.**)）之前将缺省值传输给 参数。因此 ，对于那些 在用户启动 程序时已经 被填充的字 段，应该使 用字段名而 不是文字作 为字段的缺 省值。例如 ，这些字段 是系统字段 （参见 *系统 T 宓 数据韵* （页 **错误！链接无效。**））。

```
REPORT SAPMZTST.

PARAMETERS: VALUE TYPE I DEFAULT 100,
            NAME LIKE SY-UNAME DEFAULT SY-UNAME,
            DATE LIKE SY-DATUM DEFAULT '19950627'.
```

如果报表用 户名为 FRED， 选择屏幕如 下所示：

请注意，字 段 DATE 的缺省值在 选择屏幕中 按用户主记 录中的格式 出现。

禁止参数显 示

要禁止在选 择屏幕中显 示参数，请 使用 PARAMETERS 语句的 NO-DISPLAY 选项。语法 如下：

语法
```
PARAMETERS <p> ..... NO-DISPLAY .....
```
创建参数， 并通过 DEFAULT 选项在时间 事件 INITIALIZATION （参见 *INITIALIZATION* (页 **Error! Not a valid link.**)）中内部赋 值，或者， 如果报表是 用 SUBMIT 启动的，则 由调用程序 进行外部赋 值。如果想只在 某种条件下 才显示参数 ，例如，根 据报表用户 在选择屏幕 的其它输入 字段中输入 的值决定是 否显示参数 ，则请您不 要使用 NO-DISPLAY 选项。如果 使用了这个 选项，参数 就不是选择 屏幕的元素 ，而且您也 不能用 MODIFY SCREEN 语句（参见 *给修改组分 配参数* (页 219)）使它可见 。要使参数成 为选择屏幕 的隐藏元素 ，请不带 NO-DISPLAY 选项声明该 参数，并通 过使用 MODIFY SCREEN 语句禁止显 示它。

允许参数接 受大小写

要允许用户 用大写或小 写字母输入 参数值，请 使用 PARAMETERS 语句的 LOWER CASE 选项。语法 如下：

语法
```
PARAMETERS <p> ..... LOWER CASE .....
```
如果没有 LOWER CASE 选项，系统 将所有输入 值更改为大 写。
如果使用 LIKE 选项从 ABAP/4 词典中引用 字段，参数 将接受 ABAP/4 字段的所有 属性。不能 更改这些属 性，也不能 使用 LOWER CASE 选项。必须 在 ABAP/4 词典中定义 是否可以输 入大写或小 写值。

```
REPORT SAPMZTST.
```

```
PARAMETERS: FIELD1(10),
            FIELD2(10) LOWER CASE.
WRITE: FIELD1, FIELD2.
```

假设选择屏 幕上有下列 输入值：

输出如下所 示：

UPPER lower

FIELD1 的内容变为 大写。

制作需要的 输入字段的 参数

要制作需要 的输入字段 的参数，请 使用 PARAMETERS 语句的 OBLIGATORY 选项。语法 如下：

语法
PARAMETERS <p> ..... OBLIGATORY .....
使用该选项 时，在参数 <p> 的输入字段 上将出现一 个问号。用 户如果不在 选择屏幕的 这个字段上 输入值，程 序就无法继 续执行。

```
REPORT SAPMZTST.
PARAMETERS FIELD(10) OBLIGATORY.
```

最终的选择 屏幕如下所 示：

在选择屏幕 上创建复选 框

要为参数输 入定义复选 框，请使用 PARAMETERS 语句的 AS CHECKBOX 选项。语法 如下：

语法
PARAMETERS <p> ..... AS CHECKBOX .....
参数 <p> 按长度为 1 的类型 C 创建。在这 种情况下， 不允许使用 附加选项 TYPE 和 LIKE。
<p> 的有效值是 ‘ ’和‘X’ 。这些值在 用户单击选 择屏幕上的 复选框时赋 给参数。

```
PARAMETERS: A AS CHECKBOX,
            B AS CHECKBOX DEFAULT 'X'.
```

在该示例中 ，两个复选 框出现在选 择屏幕的左 侧，它们的 右侧出现字 段的名称。复选框 B 具有缺省值 ‘X’。

当用户单击 这些复选框 时，值‘X’ 和‘ ’将分配给 相应的参数 。

如果使用 PARAMETERS 语句的 LIKE 选项参阅 ABAP/4 词典的字段 ，该字段类 型为 CHAR， 长度为 1，允许的 值是‘X’ 和‘ ’（在字段 的域中定义 ，参见文档 *ABAP/4 词典* （页 **Error! Not a valid link.**））， 则该参 数将自动作 为复选框出 现在选择屏 幕上。

在选择屏幕 上创建单选 按钮组

要为参数输 入定义单选 按钮组，请 使用 PARAMETERS 语句的 RADIOBUTTON GROUP 选项。语法 如下：

语法
PARAMETERS <p> ..... RADIOBUTTON GROUP <radi>.....
参数 <p> 按类型 C，长度 1 创建，并分 配到组 <radi>。 字符串 <radi> 的最大长度 是 4。允许使 用附加的 LIKE 选项，但是 必须参阅类 型为 C、长 度为 1 的字段。
必须为每个 <radi> 组分配至少 两个参数。 每个组中只 有一个参数 可以用 DEFAULT 选项分配的 缺省值。该 值必须是 ‘ X’。
当用户单击 选择屏幕中 的单选按钮 时，相应的 参数被激活 （分配值 ‘ X’），同 时同组的其 它参数被设 为非活动的 （赋值 ‘ ’）。

```
PARAMETERS: R1 RADIOBUTTON GROUP RAD1,
            R2 RADIOBUTTON GROUP RAD1 DEFAULT 'X',
            R3 RADIOBUTTON GROUP RAD1,

            S1 RADIOBUTTON GROUP RAD2,
            S2 RADIOBUTTON GROUP RAD2,
            S3 RADIOBUTTON GROUP RAD2 DEFAULT 'X'.
```

在该示例中 ，R1、R2 和 R3 形成一个单 选按钮组， 而 S1、S2 和 S3 形成另一个 组。在选择 屏幕中，当 其它按钮非 活动时 R2 和 S3 被激活。

如果没有使 用 DEFAULT 选项，每个 组的第一个 参数（即 R1 和 S1 ）将被激活 并分配值 ‘ X’ 。

从 SAP 内存中使用 缺省值

PARAMETERS 语句的 MEMORY-ID 选项允许您 从全局 SAP 内存（ SPA/GPA 参数）使用 缺省值。语 法如下：

语法
PARAMETERS <p> ..... MEMORY ID <pid>.....
使用该选项 时，以名称 <pid> 存储在全局 用户相关的 SAP 内存中的值 将作为 <p> 的缺省值出 现在选择屏 幕上。
<pid> 最长 3 个字符，并 且不能用引 号封闭。
可以使用全 局 SAP 内存在程序 间传递保留 在事务限制 以外的值。 用户在整个 终端进程期 间都可利用 该内存，并 且所有并行 进程使用相 同的内存。 因此 SAP 内存包含的 内容要比局 限于事务的 ABAP/4 内存（参见 *ABAP/4 内存中的数 据簇*（页 **错误！链接无效。**））更广泛。
参数可以按 用户在名为 <pid> 的用户主记 录中的说明 进行设置。 关于 SAP 内存的详细 信息，参阅 SET/GET PARAMETER 的关键字文 档和 *在程序间传递数据*（页 **Error! Not a valid link.**）。

下列报表以 “HK”为 名称在全局 SAP 内存中存储 值：

```
REPORT SAPMZTS1.

SET PARAMETER ID 'HK' FIELD 'Test Parameter'.
```

该值在下列 报表中作为 参数 TEST 的缺省值使 用：

```
PROGRAM SAPMZTS2.

PARAMETERS TEST(16) MEMORY ID HK.
```

SAPMZTS2 的选择屏幕 如下所示：

给参数分配 匹配代码对 象

要给参数分 配匹配代码 对象，请使 用 PARAMETERS 语句的 MATCHCODE OBJECT 选项。语法 如下：

语法
PARAMETERS <p> ..... MATCHCODE OBJECT <obj> .....
匹配代码对 象 <obj> 的名称必须 是 4 字符的变量 名，并且不 能用引号封 闭。
如果使用该 选项，则可 能的条目按 钮将在参数 <p> 的输入字段 之后出现。 用户按下该 按钮时，其 结果是为执 行输入字段 的匹配代码 选择。关于 匹配代码的 详细信息， 参阅文档 *ABAP/4 词典* （页 Error! Not a valid link.）。

```
PARAMETERS CONN(10) MATCHCODE OBJECT SPFL.
```

单击可能的 条目按钮后 ，在选择屏 幕上将出现 下列对话框 ：

匹配代码对 象 SPFL 用于查找航 班号。通过 在字段上安 置光标，输 入值并选择 “返回”， 用户将获得 与输入值相 匹配的航班 号列表，例 如，特定承 运方的所有 航班。

给修改组分 配参数

要将参数分 配给修改组 （该参数可 用于随后的 屏幕修改） ，请使用 PARAMTERS 语句的 MODIF ID 选项，如下 所示：

语法
PARAMETERS <p> ..... MODIF ID <key> .....
修改组 <key> 的名称必须 是不带引号 的 3 字符变量名 。
MODIF ID 选项总是把 <key> 分配到内表 SCREEN 的 SCREEN-GROUP1 列。
在 AT SELECTION-SCREEN OUTPUT 事件（参见*选择屏幕的 PBO* （页 Error! Not a valid link.））中，分配 给修改组的 参数可以用 LOOP AT SCREEN/MODIFY SCREEN 语句按整个 组进行处理 。
有关修改组 和内表 SCREEN 的详细信息 ，参见 LOOP AT SCREEN 的关键字文 档或文档*工作台工具* （页 Error! Not a valid link.）. 中有关屏幕 修改的部分 。

```
PARAMETERS: TEST1(10) MODIF ID SC1,
            TEST2(10) MODIF ID SC2,
            TEST3(10) MODIF ID SC1,
            TEST4(10) MODIF ID SC2.

AT SELECTION-SCREEN OUTPUT.

LOOP AT SCREEN.
   IF SCREEN-GROUP1 = 'SC1'.
        SCREEN-INTENSIFIED = '1'.
        MODIFY SCREEN.
        CONTINUE.
   ENDIF.
   IF SCREEN-GROUP1 = 'SC2'.
        SCREEN-INTENSIFIED = '0'.
        MODIFY SCREEN.
   ENDIF.
ENDLOOP.
```

在 PARAMETERS 语句中，参 数 TEST1 和 TEST3 被分配给组 SC1，而 TEST2 和 TEST4 被分配给组 SC2。在 AT SELECTION-SCREEN OUTPUT 事件中，按 照 GROUP1 字段的内容 ，内表 SCREEN 的字段 INTENSIFIED 被设置为 1 或 0。在选择 屏幕中，TEST1 和 TEST3 所在行被设 置为高亮度 ，而 TEST2 和 TEST4 则不是，如 下所示：

使用选择标 准

在 ABAP/4 中 ，选择标准 存储在特殊 内表中，以 方便设置数 据库访问条 件。每个选 择标准通常 分配到数据 库表的特定 列。可以用 SELECT-OPTIONS 语句定义选 择标准。
如果在报表 程序中用 SELECT-OPTIONS 语句定义了 选择标准， 系统将在报 表选择屏幕 中自动创建 输入字段。 这样，用户 可以为访问 相关的数据 库表输入条 件。

选择标准是 什么？

要使报表用 户能够在选 择屏幕中输 入对数据库 访问的限制 ，您可以使 用选择标准 。它们提供 方便的方式 以限制程序 从数据库表 到某个子集 对数据进行 访问。
如果您想编 程实现复合 选择，保存 在指定内表 中的 PARAMETERS 选择标准是 很有用的， 因为它们使 您不用在长 的 WHERE 条件中进行 编码。
使用 SELECT-OPTIONS 语句定义选 择标准。在 定义期间， 通常把选择 标准连接到 必须在程序 中说明的数 据库表的特 定列上。但 是,您也可 以把选择标 准连接到报 表的内部字 段。多数情 况下每个选 择标准能连 接到一个数 据库表，而 数据库表可 以连接到几 个选择标准 （例如一个 标准对应每 一列）。
如果您使用 SELECT-OPTIONS 语句，报表 用户可以在 选择屏幕中 输入选择标 准。
保存选择标 准的内表称 为选择表。
如果逻辑数 据库被连接 到报表程序 ，您必须注 意一些特殊 规则。

选择表

系统为每个 SELECT-OPTIONS 语句创建选 择表。选择 表的目的是 按标准化的 方式保存复 合选择限制 。它们可按 多种方式使 用。它们的 主要目的是 使用 Open SQL 语句的 WHERE 子句（参见 *使用报表的 选择标准* （页 207） ）把选择标 准直接传输 到数据库表 。
选择表是一 个带表头行 的内表。它 的行结构是 字段字符串 ，由四个组 件构成，即 SIGN、OPTION、LOW 和 HIGH。 每个选择表 行表示数据 选择的条件 ：

SIGN
SIGN 的数据类型 是 C，长度为 1。SIGN 是标志，表 示保存在 OPTION 中的运算符 是否需要翻 转。允许值 是 I 和 E。
- I 表示“包含 ”（包含标 准－运算符 不翻转）
- E 表示“排除 ”（排除标 准－运算符 翻转）

OPTION
OPTION 的数据类型 是 C，长度为 2。OPTION 包含选择运 算符。如果 SIGN 包含 E，运算符 的作用就象 它的前面有 NOT（有 关 NOT 的详细信息 ，参见 *组合几个逻 辑表达式* （页 **错误！链接无效。**） ）一样。下 列运算符是 可用的：
- 如果 HIGH 是空的，您 可以使用 EQ 、 NE 、 GT 、 LE 、 LT 、 CP 和 NP。这些 运算符在 *编程逻辑表达式* （页 **错误！链接无效。**）中描述。运 算符 CP 和 NP 没有它们在 通常逻辑表 达式中所具 有的范围。 只有当在输 入字段中使 用了通配符 （“ * ”或“ + ”）时它们 才是有效的 。没有定义 退出符号。
- 如果 HIGH 已被填充， 则可以使用 BT（位于 ）和 NB（不位 于）。这些 运算符的功 能与 BETWEEN 和 NOT BETWEEN 一样(参见 *检查字符是 否属于某一 范围* （页 **错误！链接无效。**））。

LOW
LOW 的数据类型 与数据库表 的列类型相 同，该表与 选择标准相 连接。
- 如果 HIGH 为空，LOW 的内容定义 单值选择。 它与 OPTION 中的运算符 相结合，为 数据库选择 指定了条件 。
- 如果 HIGH 已填充，LOW 和 HIGH 中的内容为 间隔选择指 定上界和下 界。与 OPTION 中的运算符 相结合，该 间隔为数据 库选择指定 了条件。

HIGH
HIGH 的数据类型 与数据库表 的列类型相 同，该表与 选择标准相 连接。HIGH 中的内容为 间隔选择指 定了上界。 与 OPTION 中的运算符 相结合，该 间隔为数据 库选择指定 了条件。

如果选择表 包含多行， 系统将按下 列规则执行 数据选择：
1. 组成 在 SIGN 字段值为 I（包含） 的行上定义 的集合联合 。
2. 去掉 在 SIGN 字段值为 E（排除） 的行上定义 的集合联合 。
3. 选择 结果集合。

> 如果选择表 只具有 SIGN 字段等于 E 的行，系统 将选择这些 行所指定的 集合之外的所有数据。

除了用 SELECT-OPTIONS 语句创建选 择表，您可 以用在 *创建和处理内表* （页 **错误！链接无效。**）中描述的语 句或 RANGES 语句，按相 同的结构创 建内表。这 些表可以用 与 SELECT-OPTIONS 语句创建的 真正的选择 表相似方法 使用，但是 有限制。有 关 RANGES 语句的详细 信息，参见

RANGES 语句

要用与选择 表相同的结 构创建内表 ，可使用 RANGES 语句，如下 所示：

语法
RANGES <seltab> FOR <f>.
该语句创建 选择表 <seltab>， 该表参考数 据库表的列 <f> 或内部字段 <f>。选 择表 <seltab> 必须在程序 中填充。不 必在程序中 再用 TABLES 语句声明数 据库表。
RANGES 语句是下列 语句的短格 式：

```
DATA: BEGIN OF <seltab> OCCURS 10,
         SIGN(1),
         OPTION(2)
         LOW  LIKE <f>,
         HIGH LIKE <f>,
       END OF <seltab>.
```

用 RANGES 创建的内表 与选择表结 构相同，但 功能不同。

用 RANGES 语句创建的 选择表

不是选择 屏幕的一部 分：它们在 程序 <prog> 中不能用于 数据传递， 该程序由下 列语句启动

SUBMIT <prog> WITH <seltab> IN <table>.

请注意，表 <table> 可以在调用 程序中用 RANGES 创建（参见*调用报表* (页 Error! Not a valid link.)）。

不与数据 库表相链接 。这意味着

- 它们不 被传递给连 接数据库（ 参见 *自定义选择 标准与逻辑 数据库* (页 197)）。
- 它们不 能与逻辑表 达式的短格 式一起使用 （参见 *在逻辑表达 式中使用选 择表* (页 170)）
- 它们不 能与 *在 GET 事件中与 CHECK 语句一起使 用选择表* (页 144) 中描述的变 体 CHECK SELECT-OPTIONS 一起使用。

您可以象使 用真正的选 择表一样在 开放式 SQL 语句的 WHERE 子句中和在 带 IN 参数（参见 *使用报表的 选择标准* (页 207)）的逻辑表 达式中使用 这些内表。

```
RANGES S_CARRID FOR SPFLI-CARRID.

S_CARRID-SIGN   = 'I'.
S_CARRID-OPTION = 'EQ'.
S_CARRID-LOW    = 'LH'.

APPEND S_CARRID.
```

该示例中， 内表 S_CARRID 按选择表的 结构创建， 并参考数据 库表 SPFLI 的列 CARRID。 字段 S_CARRID-LOW 和 S_CARRID-HIGH 具有相同的 类型 CARRID。 内表 S_CARRID 的表头行被 填充并添加 到表中。表 中定义的选 择条件与下 列逻辑表达 式的功能相 同：

```
SPFLI-CARRID EQ 'LH'
```

自定义选择 标准与逻辑 数据库

如果逻辑数 据库与报表 程序相连接 ，并且定义 了与连接数 据库的数据 库表相连接 的选择标准 ，那么必须 区别两种情 况：

如果您为 数据库表列 定义了选择 标准，该数 据库不提供 动态选择（ 参见 *选择屏幕与 逻辑数据库* (页 234)），您的自 定义选择标 准不会影响 逻辑数据库 读入的数据 总数。只有 在读入后， 您才可以在 GET 事件（参见 *使用报表的 选择标准* (页 207)）中在报表 程序中进行检查。

如果您为 具有动态选 择（参见 *选择屏幕与 逻辑数据库* (页 234)）的数据库 表的列定义 了选择标准 ，系统将把 用户在选择 屏幕的输入 字段中输入 的值传递给 逻辑数据库 。在那里，它们被用作 动态选择。 这种选择比 不具有动态 选择的数据 库表效率更 高。

通过用 SELECT-OPTIONS 语句为预知 动态选择的 数据库表的 行定义选择 标准，可以 使系统直接 在选择屏幕 上显示动态 选择。要让 它出现，用 户就不能单 击“用户选 择”。其结 果就象用户 在屏幕上为动态选择输 入选择限制 一样。

也可以避免 系统把选择 标准传递到 逻辑数据库 （参见 *避免将选择 标准传递到 逻辑数据库* (页 191)）。

用 RANGES 语句创建的 内表中的内 容没有传递 到逻辑数据 库。

定义选择标 准

可以用 SELECT-OPTIONS 语句定义选 择标准。SELECT-OPTIONS 具有几个对 报表来说很 重要的变体 。下列主题 说明
用户在逻辑 数据库程序 中还可以使 用 SELECT-OPTIONS 语句的其它 变体（参见 关键字文档 和 *逻辑数据库 的特征和 护*（页 **Error! Not a valid link.**））。

SELECT-OPTIONS 语句的基本 格式

要创建报表 用户可以在 选择屏幕中 填充的选择 标准，可以 使用 SELECT-OPTIONS 语句，如下 所示：

语法
SELECT-OPTIONS <seltab> FOR <f>.
该语句创建 选择表 <seltab>， 该表与数据 库表的列 <f> 或内部字段 <f> 相连接。该 数据库表必 须在程序中 用 TABLES 语句声明。 名称 <seltab> 最多可以包 含 8 个字符。
选择表 <seltab> 由报表用户 在选择屏幕 中填充。在 程序中，可 以修改选择 表的行或往 里面添加更多的行。
可以用在 *选择文本*（页 **错误！链接无效。**）中描述的文 字摘要来更 改选择屏幕 上输入字段 左面的文本 字段。
下列示例说 明选择表是 如何用选择 屏幕中的用 户输入填充 的：

```
REPORT SAPMZTST.

TABLES SPFLI.

SELECT-OPTIONS AIRLINE FOR SPFLI-CARRID.

LOOP AT AIRLINE.
   WRITE: / 'SIGN:',   AIRLINE-SIGN,
            'OPTION:', AIRLINE-OPTION,
            'LOW:',    AIRLINE-LOW,
            'HIGH:',   AIRLINE-HIGH.
ENDLOOP.
```

启动 SAPMZTST 后，将出现 如下选择屏 幕：

显示选择标 准的名称、 两个输入字 段“从”和 “到”以及 箭头图标。 报表用户输 入到第一个 输入字段（ “到”字段 ）的值被写 入选择屏幕 的 AIRLINE-LOW 组件。报表 用户输入到 第二个输入 字段(“到 ”字段)的 值被写入选 择屏幕的 AIRLINE-HIGH 组件。

例如，假设 报表用户按 下图所示输 入值：

SAPMZTST 的输出将如 下所示：

```
SIGN: I OPTION: EQ LOW: AA HIGH:
```

如果用户让 “到”字段 为空（单值 选择），则 SIGN 和 OPTION 的缺省设置 为 I 和 EQ。

例如，假设 报表用户按 下图所示输 入值：

SAPMZTST 的输出将如 下所示：

```
SIGN: I OPTION: BT LOW: AA  HIGH: LH
```

如果用户在 “到”字段 输入值（间 隔选择）， SIGN 和 OPTION 的缺省设置 将是 I 和 BT。

要设置更复 杂的选择模 式，用户可 以单击选择 屏幕右侧的 箭头图标。将出现新 的“复杂选 择”窗口。 假设用户填 充各字段。 如下所示：

在用户通过 单击“复制 ”保存这种 扩展选择模 式后，选择 屏幕上的箭 头图标变成 绿色以表示 选择比“到 ”和“从” 字段中显示 的更复杂。

“复杂选择 ”窗口的目 的是填充选 择表中的多 个行。

SAPMZTST 的输出如下 所示：

SIGN: I OPTION: EQ LOW: AA HIGH:

SIGN: I OPTION: EQ LOW: AF HIGH:

SIGN: I OPTION: BT LOW: DL HIGH: LH

SIGN: I OPTION: BT LOW: SA HIGH: UA

通过单击“ 复杂选择” 窗口中的“ 扩充”图标， 用户可以输 入要从结果 集合中排除 的选择标准 ：

如果用户如 上所示输入 值，SAPMZTST 的输出将如 下所示：

SIGN: E OPTION: EQ LOW: FC HIGH:

SIGN: E OPTION: BT LOW: SQ HIGH: SR

在选择表中 对应行的 SIGN 字段包含 E。

要设置选择 表各行的 SIGN 和 OPTION 值，用户必 须双击选择 屏幕或“复 杂选择”窗 口的输入字 段。将出现 “维护选择 选项”窗口 ，该窗口寻 求单值选择 ，如下所示 ：

用户可以通 过“选择选 项”挑选选 择运算符。 OPTION 字段中符号 与允许的运 算符之间的 关系与 *在程序中为 行选择指 条件* (页 **错误！链接无效。**) 的表中的相 同。

用户可以通 过“包括/排 除”挑选在 选择表的 SIGN 字段中是否 包含 I（“选择 ”）或 E（“从选 择中排除” ）。

当用户保存 该选择模式 时，选择屏 幕或“复杂 选择”窗口 中的相应字 段如下所示 ：

OPTION 字段中运算 符的符号被 显示出来， 红色表示 SIGN 字段包含 E。如果 SIGN 字段包含 I，它将是 绿色的。

用该选择， SAPMZTST 的输出将如 下所示：

SIGN: E OPTION: GE LOW: AA HIGH:

间隔选择标 准的“维护 选择选项” 窗口具有相 似的功能。

给选择标准 分配缺省值

要给将显示 在选择屏幕 上的选择标 准分配缺省 值，请使用 SELECT-OPTIONS 语句的 DEFAULT 选项。语法 如下：

语法
SELECT-OPTIONS <seltab> FOR <f> DEFAULT <g> [TO <h>] ....
<g> 和 <h> 的缺省值可 以是实际值 （在单引号 中）或是字 段名，该字 段的值将用 做缺省值。
系统在时间 事件 INITIALIZATION （参见 *INITIALIZATION* (页 **Error! Not a valid link.**)）之前把缺省值传递给 选择标准。 因此，只有 那些在用户 启动程序时 已被填充的 字段才可以 用字段名而 不用文字。 例如，这样 的字段可以 是系统字段 （参见 *系统定义数据对象* (页 **错误！链接无效。**)）。
对于每个 SELECT-OPTIONS 语句，可以 指定 DEFAULT 附加部分。 这意味着只 能用缺省值 填充选择表 <seltab> 的第一行。 选择表行的 第一行的所 有组件可以 用 DEFAULT 选项预先设 置：

要只设置 LOW 字段（单值 选择），请 使用
.....DEFAULT <g>.
要设置 LOW 和 HIGH 字段（间隔 选择），请 增加 TO，如下 所示：
.....DEFAULT <g> TO <h>.
要设置 OPTION 字段（选择 运算符）， 请增加 OPTION <op>， 如下所示：
.....DEFAULT <g> [to <h>] OPTION <op>.

- 对于单 值选择，<op> 可以是 EQ 、 NE 、 GE 、 GT 、 LE 、 LT 、 CP 或 NP。缺省 值是 EQ。
- 对于期 间选择， <op> 可以是 BT 或 NB。缺省 值是 BT 。

要设置 SIGN 域（包括/排 除），请增 加 SIGN <s> ，如下所示 ：

.....DEFAULT <g> [to <h>] [OPTION <op>] SIGN <s>.
SIGN <s> 可以是 I （包括） 和 E （排除）。 缺省值是 I。

```
REPORT SAPMZTST.

TABLES SPFLI.

SELECT-OPTIONS AIRLINE FOR SPFLI-CARRID
               DEFAULT 'AA'
                    TO 'LH'
                OPTION  NB
                  SIGN  I.
```

在开始 SAPMZTST 之后，选择 屏幕如下所 示：

“到”字段 前面的符号 表明 AIRLINE-OPTION 字段包含运 算符 NB（不位 于）。该符号是绿色的 ，表明 AIRLINE-SIGN 字段包含 I。右边的 箭头不是绿 色的，因为 选择表中只 有一行被填 充。

把选择表限 制在一行

要把用户对 选择表的访 问限制在第 一行，请使 用 SELECT-OPTIONS 语句的 NO-EXTENSION 选项，语法 如下所示：

语法
SELECT-OPTIONS <seltab> FOR <f> ..... NO-EXTENSION .....
如果指定该 选项，则选 择屏幕上不 出现右箭头 ，并且用户 不能访问“ 复杂选择” 窗口。

```
REPORT SAPMZTST.

TABLES SPFLI.

SELECT-OPTIONS AIRLINE FOR SPFLI-CARRID NO-EXTENSION.
```

选择屏幕如 下所示：

把选择表限 制为单值选 择

要将选择屏 幕上选择标 准的外观限 制为单值选 择，请使用 SELECT-OPTIONS 语句的 NO INTERVALS 选项。语法 如下所示：

语法：
SELECT-OPTIONS <seltab> FOR <f> ..... NO INTERVALS .....
如果指定该 选项，则“ 到”字段不 出现在选择 屏幕上。选 择屏幕上的 输入将限制 为单值选择 。但是，用 户可以在“ 复杂选择” 屏幕中输入 间隔选择。

```
REPORT SAPMZTST.

TABLES SPFLI.

SELECT-OPTIONS AIRLINE FOR SPFLI-CARRID NO INTERVALS.
```

选择屏幕如 下所示：

用户只能直 接输入单值 选择，但是 ，用户通过 单击屏幕右 边的箭头可 以输入进一步选择。

如果增加 NO-EXTENSION 选项，如下 所示：

```
REPORT SAPMZTST.

TABLES SPFLI.

SELECT-OPTIONS AIRLINE FOR SPFLI-CARRID NO INTERVALS
                                        NO-EXTENSION.
```

则选择屏幕 如下所示：

现在，用户 只能输入单 值选择。

避免将选择 标准传递到 逻辑数据库

如果逻辑数 据库被连接 到报表程序 ，就能把选 择标准连接 到作为逻辑 数据库一部 分的数据库 表。如果数 据库表允许 动态选择（ 参见 *选择屏幕与 逻辑数据库* (页 234) ），则系统 将相应的选 择标准传递 到逻辑数据 库（参见 *自定义选择 标准与逻辑 数据库* (页 241))。逻辑数 据库将不从 不满足这些 选择标准的 数据库表中 读取行。
如果您希望 逻辑数据库 读取这些行 ，例如，当 您将选择标 准用于其它 目的而不是 限制数据库 访问时（参 见 *使用报表的 选择标准* (页 207)），则请使 用 SELECT-OPTIONS 语句的 NO DATABASE SELECTION 选项，如下 所示：

语法
```
SELECT-OPTIONS <seltab> FOR <f> ..... NO DATABASE SELECTION .....
```

逻辑数据库 F1S 被连接到下 列报表程序 。数据库表 SPFLI 的列 CONNID 具有动态选择（参见 *选择屏幕与 逻辑数据库* (页 234) ）。

```
REPORT SAPMZTST.

TABLES SPFLI.

SELECT-OPTIONS CONN FOR SPFLI-CONNID NO DATABASE SELECTION.

GET SPFLI.
   IF SPFLI-CONNID IN CONN.
     WRITE: SPFLI-CARRID, SPFLI-CONNID, 'meets criterion'.
   ELSE.
     WRITE: SPFLI-CARRID, SPFLI-CONNID,
                          'does not meet criterion'.
   ENDIF.
```

选择屏幕的 第一部分在 逻辑数据库 中定义。最 后一行 (CONN) 在报表程序 中定义。

如果用户输 入上述选择 标准，输出 将如下所示 ：

```
LH  2402 does not meet criterion

LH  2436 meets criterion

LH  2462 does not meet criterion
```

下列报表不 使用 NO DATABASE SELECTION OPTION：

```
REPORT SAPMZTST.

TABLES SPFLI.

SELECT-OPTIONS CONN FOR SPFLI-CONNID.

GET SPFLI.
   IF SPFLI-CONNID IN CONN.
     WRITE: SPFLI-CARRID, SPFLI-CONNID, 'meets criterion'.
   ELSE.
WRITE: SPFLI-CARRID, SPFLI-CONNID,
                     'does not meet criterion'.
   ENDIF.
```

如果具有与 上面相同的 选择，输出 将如下所示 ：

```
LH  2436 meets criterion
```

结果就好象 您没有使用 SELECT-OPTIONS 语句，但是 用户已在选 择屏幕的应 用工具条中 选择了“用 户选择”， 然后在那里 输入了 2436。

选择标准的 其它选项

有很多选项 可以和 SELECT-OPTIONS 语句一起使 用。每个选 项能完成的 任务和每个 选项的语法 如下所示：

要在选择 屏幕上禁止 显示选择标 准，请使用

```
SELECT-OPTIONS <seltab> FOR <f> ... NO-DISPLAY .............
```

要使选择 标准能够接 受大写和小 写字母，请 使用

```
SELECT-OPTIONS <seltab> FOR <f> ... LOWER CASE .............
```

要进行选 择屏幕上的 “到”字段 所必需的选 择，请使用

```
SELECT-OPTIONS <seltab> FOR <f> ... OBLIGATORY .............
```

要为“到 ”字段从 SAP 内存中使用 缺省值，请 使用

```
SELECT-OPTIONS <seltab> FOR <f> ... MEMORY ID <pid>..........
```

要将选择 标准的字段 分配给修改 组，请使用

```
SELECT-OPTIONS <seltab> FOR <f> ... MODIF ID <key>...........
```

要将匹配 码对象分配 给选择标准 的“从”和 “到”字段 ，请使用

```
SELECT-OPTIONS <seltab> FOR <f> ... MATCHCODE OBJECT <obj>...
```

这些选项具 有与 PARAMETERS 语句的选项 相同的语法 和功能。有 关选项的解 释，参见*为变量定义 输入字段* (页 235) 。

使用报表的 选择标准

可以使用选 择表中输入 的选择标准 ，以便在报 表程序中完 成三个不同 的任务：

在 WHERE 子句中使用 选择表

要限制开放 式 SQL 语句 SELECT、 UPDATE 和 DELETE 的数据库访 问，请使用 WHERE 子句。
要在 WHERE 条件中使用 选择表，请 写：

语法

```
.......... WHERE <f> IN <seltab>.
```

<f> 是数据库字 段（数据库 表的列）的 不带前缀的 名称，而 <seltab> 是与该字段 相连接的选 择表。如果 您用开放式 SQL 语句（参见 *在程序中为行选择指定条件* (页 **错误！链接无效。**）） 使用 WHERE 条件，则系 统只访问指 定数据库表 的某些行， 其字段 <f> 中的内容满 足 <seltab> 中存储的选 择标准。

```
REPORT SAPMZTST.

TABLES SPFLI.

SELECT-OPTIONS AIRLINE FOR SPFLI-CARRID.

SELECT * FROM SPFLI WHERE CARRID IN AIRLINE.
   WRITE SPFLI-CARRID.
ENDSELECT.
```

在该示例的 SELECT-OPTIONS 语句中，选 择表AIRLINE 与数据库表 SPFLI 的 CARRID 列相连接。 SELECT 语句的WHERE 子句使系统 检查 CARRID 列中的内容 是否满足 AIRLINE 中存储的选 择标准。

假定报表用 户在选择表 中输入两行 ，即间隔选 择和单值选 择，如下所 示：

SIGN	OPTION	LOW	HIGH
I	BT	DL	UA
E	EQ	LH	

则 报表输出如 下所示：

DL DL SQ UA UA UA

所有在“DL” 和“UA” 之间的航空 公司，除了 “LH”外 都被选中。

在逻辑表达 式中使用选 择表

要控制程序 的内部流程 ，必须用逻 辑表达式编 写条件。可 以用选择表 （参见 *检查选择标准*（页 **错误！链接无效。**））编写特殊 逻辑表达式 。语法如下 所示：

语法

```
... <f> IN <seltab> ....
```

如果字段 <f> 中的内容满 足存储在选 择表 <seltab> 中的选择限 制，则逻辑 表达式为真 。 <f> 可以是任何 内部字段或 选择表的列 。

如果用 SELECT-OPTIONS 语句将选择 表 <seltab> 连接到 <f>，则 可以使用下 列逻辑表达 式的短格式 ：

语法

```
... <seltab> ....
```

该短格式不 能用于由 RANGES 语句定义的 选择表。

```
REPORT SAPMZTST.

TABLES SPFLI.

SELECT-OPTIONS AIRLINE FOR SPFLI-CARRID.

WRITE: 'Inside', 'Outside'.

SELECT * FROM SPFLI.
   IF SPFLI-CARRID IN AIRLINE.
      WRITE: / SPFLI-CARRID UNDER 'Inside'.
   ELSE.
      WRITE: / SPFLI-CARRID UNDER 'Outside'.
   ENDIF.
ENDSELECT.
```

假 设报表用户 在选择表中 输入了两行 ，即间隔选 择和单值选 择，如下所 示：

SIGN	OPTION	LOW	HIGH
I	BT	DL	UA
E	EQ	LH	

则 报表输出如 下所示：

在 SELECT 循环中，所 有的行都从 数据库表 SPFLI 中读取。使 用 IF 语句，程序 流将按照逻 辑表达式分 枝为两个语 句块。短格 式 IF AIRLINE 也可用于该 程序。

```
REPORT SAPMZTST.

TABLES SPFLI.
```

```
SELECT-OPTIONS: S_CARRID FOR SPFLI-CARRID,
                S_CITYFR FOR SPFLI-CITYFROM,
                S_CITYTO FOR SPFLI-CITYTO,
                S_CONNID FOR SPFLI-CONNID.

SELECT * FROM SPFLI.
   CHECK: S_CARRID,
          S_CITYFR,
          S_CITYTO,
          S_CONNID.
   WRITE: / SPFLI-CARRID, SPFLI-CONNID,
          SPFLI-CITYFROM, SPFLI-CITYTO.
ENDSELECT.
```

启动程序后 ，出现选择 屏幕，用户 可以在其中 填充输入字 段，如下所 示：

这时，输出 如下所示：

在 SELECT 循环中，系 统从数据库 表 SPFLI 中读取所有 的行。在这 些行中，系 统只把满足 选择表中的 条件的行写 到输出屏幕 。否则，CHECK 语句之后， 系统将离开循环传递。 CHECK 语句使用了 逻辑表达式 的短格式。 长格式是：

```
CHECK: SPFLI-CARRID IN S_CARRID,
       SPFLI-CITYFR IN S_CITYFR,
       SPFLI-CITYTO IN S_CITYTO,
       SPFLI-CONNID IN S_CONNID.
```

在 GET 事件中与 CHECK 语句一起使 用选择表

可以使用 CHECK 语句离开
循环（参 见*有条件终止 循环过程*（页 **错误！链接无效。**））
子程序（ 参见 *无条件中 子程序*（页 **错误！链接无效。**））
处理块（ 参见 *有条件地离 开过程块*（页 Error! Not a valid link.））
除了允许在 逻辑表达式 中与 CHECK 语句（参见 *在逻辑表达 式中使用选 择表*（页 247）中的示例） 一起使用选 择表，ABAP/4 还提供 CHECK 语句的变体 ，该变体只 能在用逻辑 数据库读入 数据库表的 一行之后使 用。
要检查该行 的内容是否 满足所有与 该数据库相 连接的选择 表中存储的 选择标准， 请使用 CHECK 语句，如下 所示：

语法
```
CHECK SELECT-OPTIONS.
```
使用该语句 ，可以通过 所有选择表 检查实际数 据库表（由 GET 给出地址） 的内容，数 据库表通过 使用不同的 SELECT-OPTIONS 语句连接到 选择表。CHECK 语句的该变 体
只能与通 过 SELECT-OPTIONS 语句连接到 数据库表的 选择标准一 起使用。
应该只用 于 GET 事件（参见 *GET〈表格〉*（页 Error! Not a valid link.））的处理块 。

由于 CHECK 语句只有在 从逻辑数据 库读入行后 才能使用， 只有当逻辑 数据库提供的选择不能 满足要求， 并且相关表 不具有动态 选择时才使 用该变体。

逻辑数据库 F1S 与下列报表 相连接：

```
REPORT SAPMZTST.

TABLES: SPFLI,SFLIGHT.

SELECT-OPTIONS: MAX    FOR SFLIGHT-SEATSMAX,
                OCC    FOR SFLIGHT-SEATSOCC.
```

```
GET SFLIGHT.
   WRITE: / SPFLI-CARRID, SPFLI-CONNID.
   CHECK SELECT-OPTIONS.
   WRITE: SFLIGHT-SEATSMAX, SFLIGHT-SEATSOCC.
```

例如，报表 用户填充选 择屏幕，如 下所示：

则输出如下 所示

系统从 SFLIGHT 中读入所有 满足逻辑数 据库选择标 准的行。如 果这些内容 不满足自定 义选择标准 MAX 和 OCC，则 系统在把 SEATSMAX 和 SEATSOCC 的内容写到 屏幕上之前 就离开 GET 事件。

格式化选择 屏幕

在程序中使 用 PARAMETERS 或 SELECT-OPTIONS 语句时出现 的选择屏幕 具有标准格 式，其中参 数按行显示 在上面。
该标准输出 并非总是足 够的。例如 ，当您定义 一组单选按 钮时，该按 钮组必须与 其它输入字 段区别开。
要格式化选 择屏幕，请 使用 SELECTION-SCREEN 语句，该语 句使用户能 指定在选择 屏幕上参数 和选择标准 格式。
使用 SELECTION-SCREEN 语句的各种 选项，可以 做下列工作 ：

SELECTION-SCREEN 只能用于选 择屏幕。它 在没有选择 屏幕的报表 中是无效的 。例如，不 执行 PARAMETERS 或 SELECT-OPTIONS 语句就无法 创建按钮。

指定空行、 下划线和注 释

在下列主题 中说明的 SELECTION-SCREEN 语句的选项 创建
示例参见

空行

要在选择屏 幕上产生空 行，请使用 SELECTION-SCREEN 语句的 SKIP 选项。语法 如下：

语法
```
SELECTION-SCREEN SKIP [<n>].
```
该语句产生 <n> 个空行，其 中 <n> 的值可以是 1 到 9。要产生 单个空行， 可以省略 <n>。

下划线

要在选择屏 幕中给一行 或行中的一 部分加下划 线，请使用 SELECTION-SCREEN 语句的 ULINE 选项。语法 如下：

语法
```
SELECTION-SCREEN ULINE [[/]<pos(len)>] [MODIF ID <key>].
```
该语句创建 下划线。
如果不使用 格式选项 <pos(len)>， 则在当前行 下面创建新 行。如果使 用格式选项 <pos(len)>， 下划线将从 当前行的位 置 <pos> 处开始，连 续 <len> 个字符。通 过一行中的 几个元素， 也可以不通 过 <pos> 来指定 (<len>)。
可以使用可 选的斜杠 (/) 请求换行 (请与*将数据输出 到屏幕* (页 **错误！链接无效。**) 中的 ULINE 和 WRITE 语句相比较)。
对于 <pos>，您可以指 定一个数， POS_LOW 或 POS_HIGH。 POS_LOW 和 POS_HIGH 是使用 SELECT-OPTIONS 语句时“从 ”和“到” 字段在选择 屏幕上的显 示位置。

和 PARAMETERS 语句一样， 可以使用 MODIF ID <key> 选项把下划 线分配给修 改组 <key>， 该修改组可 以用在 AT SELECTION-SCREEN OUTPUT 事件中以修 改屏幕（参 见 *给修改组分 配参数*（页 239） ）。

注释

要在选择屏 幕中书写文 本，请使用 SELECTION-SCREEN 语句的 COMMENT 选项。语法 如下：

语法
```
SELECTION-SCREEN COMMENT [/]<pos(len)> <name> [FOR FIELD <f>]
                                              [MODIF ID <key>].
```
使用该选项 时必须定义 格式（开始 位置和长度 － 只有一行上 具有几个元 素时才可以 省略 <pos>）。对于 <name>，可以指定 文本符号（ 参见 *文本符号*（页 **错误！链接无效。**）），或指定 最大长度为 8 个字符的字 段名。该字 符字段不能 用（例如） DATA 语句声明， 而是通常自 动地按长度 <len> 生成。必须 在 INITIALIZATION 事件（参见 *INITIALIZATION*（页 **Error! Not a valid link.**）） 中动态地 填充该字符 字段。
将显示文本 <name>， 从列 <pos> 开始，长度 为 <len>。 如果不使用 斜杠 (/)，注 释将写到当 前行，否则 将创建新行 。
要把文本标 签分配给参 数或选择选 项，请使用 FOR FIELD <f> 选项。<f> 可以是参数 或选择标准 的名称。因 此，如果用 户在选择屏 幕的注释上 请求帮助， 则显示分配 给字段 <f> 的帮助文本 。
选择屏幕注 释的 MODIF ID <key> 选项与为 PARAMETERS 语句（参见 *给修改组分 配参数*（页 239） ）所描述的 相同。

空行、下划 线和注释示 例

这是在选择 屏幕中如何 使用空行、 下划线和注 释的示例：

```
SELECTION-SCREEN COMMENT /2(50) TEXT-001 MODIF ID SC1.

SELECTION-SCREEN SKIP 2.
SELECTION-SCREEN COMMENT /10(30) COMM1.
SELECTION-SCREEN ULINE.

PARAMETERS: R1 RADIOBUTTON GROUP RAD1,
            R2 RADIOBUTTON GROUP RAD1,
            R3 RADIOBUTTON GROUP RAD1.

SELECTION-SCREEN ULINE /1(50).
SELECTION-SCREEN COMMENT /10(30) COMM2.
SELECTION-SCREEN ULINE.

PARAMETERS: S1 RADIOBUTTON GROUP RAD2,
            S2 RADIOBUTTON GROUP RAD2,
            S3 RADIOBUTTON GROUP RAD2.

SELECTION-SCREEN ULINE /1(50).

INITIALIZATION.

COMM1 ='Radio Button Group 1'.
COMM2 ='Radio Button Group 2'.

LOOP AT SCREEN.
   IF SCREEN-GROUP1 = 'SC1'.
        SCREEN-INTENSIFIED = '1'.
        MODIFY SCREEN.
   ENDIF.
ENDLOOP.
```
选择屏幕如 下所示：

在文本符号 001 中指定的文 本将以高亮 度出现在选 择屏幕的顶 部。在文本 下面显示两 组单选按钮 ，由下划线 分隔开并由 注释进行描 述。如果语 句中没有斜 杠(/)，即，使用了 格式化的 ULINE 选项，则下 划线将改写 每个单选按 钮组的最后 一行。

将几个元素 放在一行上

要将参数或 注释集合定 位在选择屏 幕的一行上 ，必须在由 下列两条语 句封闭的块 中声明元素 ：

语法
```
SELECTION-SCREEN BEGIN OF LINE.
  ...
SELECTION-SCREEN END OF LINE.
```
请注意，使 用该选项时 不会显示选 择文本（参 数名或文本 元素）。要 显示选择文 本，必须通 过与 COMMENT 选项一起使 用 SELECTION-SCREEN 语句提供描 述。
另外，请不 要把斜杠与 格式选项 〈pos(len)〉一起使用。 在该格式选 项中，可以 在上述语句 中省略位置 说明 〈pos〉。 随后对象将 定位在行中 的当前位置 。

```
SELECTION-SCREEN BEGIN OF LINE.
   SELECTION-SCREEN COMMENT 1(10) TEXT-001.
   PARAMETERS: P1(3), P2(5), P3(1).
SELECTION-SCREEN END OF LINE.
```

该示例使下 列行出现在 选择屏幕上 ：

该行以文本 符号 001 开头，后跟 参数 P1 、 P2 和 P3 的输入字段 。

定位元素

要在选择屏 幕上定位下 一个参数或 注释，请和 POSITION 选项一起使 用 SELECTION-SCREEN 语句。语法 如下：

语法
```
SELECTION-SCREEN POSITION <pos>.
```
对于 〈pos〉，可以指定 数字 POS_LOW 或 POS_HIGH。 POS_LOW 和 POS_HIGH 是使用 SELECT-OPTIONS 语句时“从 ”和“到” 字段在选择 屏幕上出现 的位置。

请只在 BEGIN OF LINE 和 END OF LINE 选项之间使 用位置选项 。

```
REPORT SAPMZTST.

TABLES SPFLI.

SELECT-OPTIONS AIRLINE FOR SPFLI-CARRID.

SELECTION-SCREEN BEGIN OF LINE.
   SELECTION-SCREEN POSITION POS_HIGH.
   PARAMETERS FIELD(5).
SELECTION-SCREEN END OF LINE.
```

选择屏幕如 下所示：

参数 FIELD 的输入字段 出现在选择 标准 AIRLINE 的“到”字 段之下。对 于 FIELD，不显示任何 选择文本。

创建元素块

要在选择屏 幕上创建元 素逻辑块， 请用 SELECTION-SCREEN 语句的 BEGIN OF BLOCK 选项选择块 的开始，然 后定义各个 元素并用 END OF BLOCK 选项选择块 的结束，如 下所示：

语法
```
SELECTION-SCREEN BEGIN OF BLOCK <block>
                              [WITH FRAME [TITLE <title>]]
                              [NO INTERVALS].
  ...
SELECTION-SCREEN END OF BLOCK <block>.
```
必须为每个 块定义名称 <block> 。可以嵌套 块。
如果增加 WITH FRAME 选项，在块 的周围将画 外框。最多 可以嵌套 5 层带外框的 不同块。
可以使用 TITLE 选项给每个 外框增加标 题。就象 COMMENT 选项中的 <name>， <title>可以是文本 符号或字符 文字。该字 符字段的长 度是外框的 宽度，该宽 度自动按照 外框的嵌套 深度进行设 置。
如果使用 NO INTERVALS 选项，则系 统处理块中 的所有 SELECT-OPTIONS 语句，就好 象它们具有 该选项（参 见 *把选择表限 制为单值选 择* (页 244))。如果块 有外框，外 框的宽度被 画得更小， 并且嵌套的 块将自动继 承 NO INTERVALS 选项。

```
SELECTION-SCREEN BEGIN OF BLOCK RAD1
                                WITH FRAME TITLE TEXT-002.
    PARAMETERS R1 RADIOBUTTON GROUP GR1.
    PARAMETERS R2 RADIOBUTTON GROUP GR1.
    PARAMETERS R3 RADIOBUTTON GROUP GR1.
SELECTION-SCREEN END OF BLOCK RAD1.
```
在选择屏幕 上，三个单 选按钮 R1 、 R2 、 R3 组成块，该 块被外框包 围并且具有 在文本符号 002 中指定的标 题，如下所 示：

在应用工具 条中创建按 钮

在选择屏幕 的应用工具 条中最多可 以创建 5 个按钮。这 些按钮自动 与功能键相 连接。语法 如下所示：

语法
```
SELECTION-SCREEN FUNCTION KEY <i>.
```
<i> 必须在 1 到 5 之间。必须 在 ABAP/4 词典字段 SSCRFIELDS-FUNCTXT_0<i> 中指定要在 运行时间出现在按钮上 的文本。

必须用 TABLES 语句声明 SSCRFIELDS。

当用户单击 该按钮时， FC0<i> 输入到字段 SSCRFIELDS-UCOMM 中，可以在 事件 AT SELECTION-SCREEN（参见 *AT SELECTION-SCREEN* (页 **Error! Not a valid link.**) ）中检查该 字段。

```
TABLES SSCRFIELDS.

DATA FLAG.

PARAMETERS TEST.

SELECTION-SCREEN FUNCTION KEY 1.
SELECTION-SCREEN FUNCTION KEY 2.

INITIALIZATION.
   SSCRFIELDS-FUNCTXT_01 = 'Button 1'.
   SSCRFIELDS-FUNCTXT_02 = 'Button 2'.

AT SELECTION-SCREEN.
   IF SSCRFIELDS-UCOMM = 'FC01'.
      FLAG = '1'.
   ELSEIF SSCRFIELDS-UCOMM = 'FC02'.
```

```
      FLAG = '2'.
   ENDIF.

START-OF-SELECTION.
   IF FLAG = '1'.
      WRITE / 'Button 1 was clicked'.
   ELSEIF FLAG = '2'.
      WRITE / 'Button 2 was clicked'.
   ENDIF.
```

该示例使具 有文本“Button 1”和“Button 2”的按钮 出现在选择 屏幕的应用 工具条上， 如下所示：

当用户单击 按钮中的一 个时，内部 字段 FLAG 在时间事件 AT SELECTION-SCREEN 中被设置。 FLAG 字段可以在 程序的后续 流程中进行 进一步处理 （参见 *通过事件控制 ABAP/4 程序流* （页 **Error! Not a valid link.**））。

在选择屏幕 上创建按钮

要在选择屏 幕上创建按 钮，可以与 PUSHBUTTON 参数一起使 用 SELECTION-SCREEN 语句。语法 如下：

语法

```
SELECTION SCREEN PUSHBUTTON [/]<pos(len)> <name>
                            USER-COMMAND <ucom> [MODIF ID <key>].
```

参数 [/]<pos(len)>、 <name> 和 MODIF ID 选项与在*注释* (页 250). 中对 COMMENT 选项进行的 描述相同。
在 <name> 中指定的文 本是按钮文 本。
对于 <ucom>， 必须指定最 多为 4 字符的代码 。当用户在 选择屏幕上 单击按钮时 ，<ucom> 被输入词典 字段 SSCRFIELDS-UCOMM。

必须使用 TABLES 语句声明 SSCRFIELDS。

可以在事件 AT SELECTION-SCREEN （参见 *AT SELECTION-SCREEN* (页 **Error! Not a valid link.**)）中检查 SSCRFIELDS-UCOMM 字段的内容 。
下列示例与 展示应用工 具条按钮的 示例（参见 *在应用工具 条中创建按 钮* (页 252)）的效果相 同。但是， 按照“ SAP 风格指南” ，建议尽可 能把按钮放 在应用工具 条中。

```
TABLES SSCRFIELDS.

DATA FLAG.

PARAMETERS TEST.

SELECTION-SCREEN PUSHBUTTON /20(10) BUT1
                                    USER-COMMAND CLI1.
SELECTION-SCREEN PUSHBUTTON /20(10) TEXT-020
                                    USER-COMMAND CLI2.

INITIALIZATION.
   BUT1 = 'Button 1'.

AT SELECTION-SCREEN.
   IF SSCRFIELDS-UCOMM = 'CLI1'.
      FLAG = '1'.
   ELSEIF SSCRFIELDS-UCOMM = 'CLI2'.
      FLAG = '2'.
ENDIF.

START-OF-SELECTION.
   IF FLAG = '1'.
      WRITE / 'Button 1 was clicked'.
   ELSEIF FLAG = '2'.
```

```
    WRITE / 'Button 2 was clicked'.
  ENDIF.
```

如果文本符 号 TEXT-020 被定义为“ Button 2”（参见 *文本符号*（页 **错误！链接无效。**）），该示例 使具有文本 “Button 1”和“Button 2”的两个 按钮出现在 选择屏幕上 ，如下所示 ：

CLI1 和 CLI2 用于 <ucom>。 当用户单击 按钮时，内 部标志 FLAG 按照时间事 件 AT SELECTION-SCREEN 中的定义进 行设置。FLAG 字段可以在 程序的后续 流程中进行 进一步的处 理（参见 *通过事件控制 ABAP/4 程序流*（页 **Error! Not a valid link.**））。

第三章 使用变式预 设置选择

概览

内容

本节描述：

什么是变式 ？

启动报表程 序时，ABAP/4 在选择屏幕 上提供特定 数据库和特 定报表选择 的输入字段 。要选择某 种数据集， 必须键入相 应的值（参 见 *使用选择屏幕*（页 **错误！链接无效。**））。
如果想在固 定间隔时间 运行具有相 同选择的同 一报表程序 （例如，每 月的销售额 统计），则 肯定不想每 次键入相同 的值。因此 ，ABAP/4 可以在一个 选择设置中 组合全部选 择所需的值 。可以对每 个报表程序 创建任意多 个不同的选 择设置，而 它们仍只分 配给值得怀 疑的报表程 序。这种选 择设置称为 变式。
变式是选择 屏幕的界面 。联机使用 的变式与在 后台处理中 使用的变式 可能具有不 同功能。

使用联机变 式
联机情况下 ，由于不必 在每次显示 选择屏幕时 都输入同一 种选择设置 ，所以通过 变式启动报 表会保存用 户工作。另 外，使用变 式可使输入 错误减到最 少。通过为 报表程序的 每个应用程 序创建具有 最优值的变 式，可保证 最终列表尽 可能地既精 确又迅速。 在选择屏幕 上输入准确 值也可缩短 报表程序的 运行时间。

在后台处理 中使用变式
在后台处理 中，变式是 为选择传递 值的唯一可 行的方法。 因此在后台 执行的报表 程序必须通 过变式启动 （例外：SUBMIT… VIA JOB）。 为避免每次 值变化时都 创建新的变 式，ABAP/4 提供了以变 量值为变式 赋值的可能 （参见 *变式属性*（页 235））。
为确保报表 程序总是通 过变式启动 ，可以在程 序属性中设 定。

创建和更改 变式

要创建或更 改变式，请 从“ABAP/4 编辑器初始 屏幕”中启 动。输入要 维护其变式 的程序名， 标记“变式 ”并选择“ 更改”。将 显示“ABAP/4： 变式 -初始屏幕” 。现在可以 显示程序现 有变式的列表、创建新 变式或修改 现有变式。

下列主题描 述：

显示所有变 式报表

创建新变式 前，应检查 是否存在您 可能使用或 轻微修改的 特定报表程 序的变式。 要继续，请 单击可能的 变式名字段 的输入按钮 。

或在初始屏 幕上选择“ 变式 -> 目录”。

在该屏幕中 ，可以显示 、删除或打 印变式。显 示变式示例 如下：

要显示属性 ，请单击“ 显示属性” 按钮。
要从目录屏 幕更改变式 ，请选择“ 更改值”或 “更改属性 ”。然后可 以看到在*创建变式*（页 154）说明的屏幕之 一。

创建变式

要创建新变 式，请进行 下列操作：

1. 在“ ABAP/4 编辑器初始 屏幕”上， 输入要为其 创建变式的 报表程序名 ，标记“变 式”并选择 “更改”。
2. 在“ ABAP/4： 变式 - 初始屏幕” 上，输入变 式名。
   名称可由最 多 14 个字母数字 字符组成。 不要在名称 中使用 '&' 或 '%'，因 为它们会导 致错误消息 ；也不要在 名称中以 SAP_ 打头，这是 因为更新系 统时这些变 式可能会被 覆盖掉。
3. 选择 “创建”。
   出现选择屏 幕。
4. 输入 所需的选择 值，包括多 个选择和动 态选择。
5. 选择 “继续”。
   然后您会看 到概述屏幕 ，在该屏幕 上，您可以 指定并保存 变式属性。 （参见*变式属性* (页 235)）。
   请注意，创 建新变式时 ，总须填写 属性和值。 因此，只能 在第二屏上 保存新变式 。

变式属性

在下列屏幕 上维护变式 的属性：

创建变式时 ，可以输入 下列属性：

描述
输入变式的 简短而有意 义的描述。 最多 30 个字符。
仅后台
指定只在后 台处理中使 用该变式， 还是在联机 环境中也使 用。
受保护变 式
如果想保护 该变式不被 其他用户更 改，请标记 该字段。
不显示变 式
如果只想在 分类中显示 变式名，而 不在 **F4** 值列表中显 示，请标记 该字段。

对包含在变 式中的选择 ，可以输入 下列属性：

类型
系统显示该 字段是参数 还是选项。
受保护
在选择屏幕 上标记要保 护而不被覆 盖字段。以 该方式标记 的值可以显 示给用户， 但用户不能 更改，也就 是说它们不 接受输入。
不可见
如果标记该 栏，则启动 报表程序时 ，系统不会 在用户见到 的选择屏幕 上显示相应 字段。
变量
如果要运行 期间设置值 ，请标记该 。可按下列 三种方式设 置，详情请 见*使用变式变 量* (页 230) 。

- 变量日 期计算
- 特定用 户值
- 表格 TVARV 中的值

输入所有属 性后，保存 该设置。创 建新变式时 ，必须填写 值和属性屏 幕上的所有 条目，而且 只能在属性 屏幕上保存 新变式。但 是，如果要 更改现有变 式的值或属 性，则可在 相应屏幕上 保存变式。

更改变式

要更改现有 变式，请调 用*创建变式* (页 255) 中描述的变 式。但是， 不使用“创 建”，而选 择现有变式 名并单击“ 更改”。现 在可以更改 值或属性（ 参见 *变式属性* (页 256) ）。
输入更改后 ，在相应屏 幕上保存值 或属性。

删除变式

要删除变式 ，请在变式 初始屏幕上 输入报表名 和变式名， 然后选择“ 变式 -> 删除”。
系统首先显 示窗口，可 以在该窗口 中确认或取 消决定。
然后确认带 有适当消息 的活动。
要删除一个 活动中的几 个变式，请 选择“变式 -> 目录”。标 记要删除的 变式，并选 择“删除” 。

打印变式

要打印变式 ，请在变式 初始屏幕上 输入变式名 ，选择要显 示的值（从 “ABAP/4： 变式 -初始屏幕”上或从“变 式”菜单中 ），单击“ 打印”。请 注意，如果 在更改模式 中，则不能 打印变式值 。
出现“打印 屏幕列表” 屏幕。
如果打印参 数与屏幕上 显示的默认 值不一致， 则可以设置 适于您的部 门的参数。 从系统管理 员那里可以 得到正确值 。标记“立 即打印”字 段。
要打印变式 ，请选择“ 打印”。

使用变式变 量

如果使用变 量值，则不 必在每次发 生值变化时 都创建新变 式。
要给选择提 供变量值， 请在属性屏 幕上标记所 需选择中的 “变量”栏 ，并单击“ 选择变量” 。出现下列 屏幕：

使用变式变 量包括三种 不同选项：

使用变量 日期计算（ 参见*使用变量日 期计算* (页 206) ）
在变式中， 如果想使用 某一天或者 最后一月的 最后一天的 日期，则可 以使用变量 日期计算。
使用特定 用户值（参 见*使用特定用 户值* (页 249)）
要在某些选 择上填写因 用户而异的 值，可以使 用具有特定 用户值的变 式。
使用在表 TVARV 中定义的变 式（参见 *使用表格 TVARV* (页 246) ）。
要在某些选 择上填写根 据应用程序 而变化的值 ，可以使用 从表 TVARV 获取变量值 的变式。
要 避免创建新 的变式和在 每个后台处 理（在每个 选择数据的 微小变化后 运行）中指 定它，可以 只在表 TVARV 中为变式分 配变量，然 后在表格中 更改相关值 。这对于设 置了输入保 护的选择屏 幕上的值尤 其重要。

使用变量日 期计算

要使用变式 中的变量日 期计算，必 须在报表程 序中将日期 字段定义为 选择标准或 参数。

```
PARAMETERS DATE LIKE SY-DATUM.
```

然后可以将 变量分配给 该日期字段 。
要给日期字 段分配变量 ，请在选择 变量屏幕上 单击“变式 类型”字段 上可能的输 入按钮。在 下列对话框 中选择“日 期变量”。

可以返回选 择变量屏幕 。单击“变 式名”上可 能的输入按 钮以得到日 期计算的建 议值列表。

如果需要， 请键入所需 值。
注意：不能 在列表中维 护建议值或 增加新建议 值。

使用特定用 户值

通过使用特 定用户值， 可以将任何 特定用户条 目包括进带 变式的选择 。对于每个 授权的用户 ，这些值都 保存在特定 的表格中， 并在变式执 行时逐一填 写到选择中 。
用该方法， 用户不必每 次启动报表 程序时都输 入从未更改 过的值，诸 如用户号或 公司代码等 。在选择屏 幕上，用户 只需填写在 不同程序执 行时其值发 生变化的字 段。几个用 户可以使用 同一变式。

创建特定用 户值
要用特定用 户值填写选 择，该用户 的主记录必 须包含带有 标识 <pid> 的相应用户 参数。
在报表程序 中，必须用 相应参数标 识的选项 ...MEMORY ID <pid> 定义参数或 选择标准。（参见 *从 SAP 内存中使用 缺省值*（页 **错误！链接无效。**））。

```
DATA: COMPANY_CODE(6).
...
SELECT-OPTIONS: CC LIKE COMPANY_CODE MEMORY ID BUK.
...
```

创建变式时 ，在属性屏 幕上为相应 选择标记“ 变量”并单 击“选择变 量”。在显 示的屏幕上 ，单击下列 对话框中的 “变量类型 ”的输入按 钮并选择“ 特定用户变 量”。

可以返回选 择变量屏幕 。系统自动 将参数的合 适长文本设 置为“变量 名”。
在运行期间 ，特定用户 变量接收当 前用户的值 为当前参数 标识。
为更改现有 特定用户变 量的值，ABAP/4 提供了两种 方法：

- 程序员可 以使用报表 程序中的功 能模块 VARI_USER_VARS_* 更改任何用 户的变量（ 参见 *在程序中更 改特定用户 变量值*（页 242））。
- 用户可以 在选择屏幕 上更改其变 量值（参见 *交互式更改 特定用户变 量值*（页 235））。

在程序中更 改特定用户 变量值
要在报表程 序更改特定 用户变量的 当前值，可 以使用下列 功能模块集 。

功能模块	功 能
VARI_USER_VARS_GET	读 取现有变量 值
VARI_USER_VARS_SET	更 改现有变量 值
VARI_USER_VARS_COPY	复 制现有变量 值
VARI_USER_VARS_DELETE	删 除变量值
VARI_USER_VARS_RENAME	重 命名变量值
VARI_USER_VARS_DIALOG	允 许交互式输 入变量值

要 在程序中插 入这些功能 模块，可以 使用 ABAP/4 编辑器中的 “编辑 -> 插入语句 -> CALL FUNCTION” 。

交互式更改 特定用户变 量值
用户可以在 报表选择屏 幕上更改其 特定用户变 量值。为此 ，请选择“ 转向 -> 用户变量” 。
随后显示包 含特定用户 选择标准和 参数的窗口 。现在可以 显示或更改 相关的值。

如果选择“ 更改”，则 系统显示另 一个窗口。 这里，可选 择是否通过 变量或选择 屏幕上的输 入更改建议 值。

在这两种情 况下，系统 都会显示可 在其中输入 并保存所需 值的窗口。

要当心，这 些更改会影 响使用相应 特定用户变 量的所有变 式。

使用表格 TVARV

使用表格 TVARV 中的值对后 台处理尤其 有用。表格 TVARV 使您不必每 次更改值或 更改现有变 式时都要创 建新变式， 因为您只需 更改存储在 表格 TVARV 中的值。然 而，要当心 表格 TVARV 中值的更改 将影响所有 使用该特殊 变量的变式 。
下列主题描 述：

在表格 TVARV 中创建新条 目

要将选择与 表格 TVARV 的条目连接 起来，请在 属性屏幕上 标记“变量 ”并选择“ 维护变量” 。在后续屏 幕上，选择 “变量选择 ”作为“变 量类型”。

然后，输入 变量名：

如果想使 用现有变量 并且知道变 量名，可直 接在“变量 名”字段中 输入变量名 。
如果想使 用现有变量 但是不知道 变量名，请 单击“名称 ”字段中可 能的输入按 钮，从显示的列表中选 择所需值。
要查看列表 中的变量值 ，请标记变 量并单击“ 值”，系统 随后转到表 格 TVARV 中的条目。
如果想创 建新变量， 请输入名称 并选择“输 入”。
系统将告诉 您该变量不 存在并询问 是否转到维 护屏幕。如 果确认，则 显示初始表 格屏幕，在 该屏幕上已 预先选定表 格 TVARV。
单击“维护 ”按钮。
出现表格 TVARV 的表格维护 屏幕。输入 变量的名称 和类型并选 择“创建” 。
如果想创建 选择选项的 新变量，则 会出现空白 屏幕，可以 在该屏幕上 输入上、下 限和选项以 及包含和不 包含的标准 。
如果想创建 参数的新变 量，则会显 示对话框， 可以在该对 话框上输入 参数值：

保存新变量 并返回变量 维护屏幕。 现在可以输 入下一个变 量名或保存 相关的变量 ，并返回属 性屏幕。

更改表格 TVARV 中的条目

要当心，更 改表格 TVARV 中的值会影 响所有使用 这些特定变 量的变式。
要更改表格 TVARV 中的值，请 选择“系统 -> 服务 -> 表格”。
然后，在“ 表格维护” 屏幕上输入 名称 TVARV 并选择“维 护”。
出现表格 TVARV 的维护屏幕 ，要求输入 变量的名称 和类型。在 该屏幕上， 可以显示和 更改现有变 量值、创建 新变量、复 制变量值或 删除它们。 “分类”按 钮提供现有 变量的概况 。

显示变量值
要显示变量 值，请进行 下列操作：

如果知道 变量的名称 和类型，可 以直接调用 。输入名称 和类型，然 后选择“显 示”。
如果想查 看现有变量 的列表，请 选择“分类 ”。
该操作将转 到选择屏幕 ，在该屏幕 上可以指定 想在列表中 查看的现有 变量集的变 量。如果没 有任何输入 ，则系统列 出所有现有 变量。
现在选择“ 执行”以获 得所需的变 量列表。
双击所需变 量以显示它 。

如果选择参 数，系统将 显示包含当 前参数值的 窗口。
如果选定选 择标准，系 统将显示包 含该值的新 屏幕。

更改变量值
要更改变量 值，按进行 下列操作：

1. 如果 知道变量名 称和类型， 则可以直接 调用该变量 。输入名称 和类型，然 后选择“更 改”。
   要查看现有 变量的列表 ，请选择“ 分类”。
   该操作后将 转到选择屏 幕，在该屏 幕上，可以 指定想在列 表中查看的 现有变量中 的变量。
   如 果没有任何 输入，则系 统会列出所 有现有的变 量。
   双击所需变 量以选择它 。
   如果选择参 数，则系统 将显示包含 当前参数值 的窗口。
   如果选定选 择标准，则 系统将显示 包含该值的 新屏幕。
2. 通过 覆盖旧值来 更改值。
3. 保存 新值。

更改的值显 示在表格 TVARV 中。运行期 间，系统在 相应的选择 中显示这些 值。

创建变量值
要在表格中 直接创建带 值的新变量 ，请指定变 量名称和类 型，然后选 择“创建” 。
如果想创建 新的选择标 准，则可以 看到空白屏 幕，可以在 该屏幕上输 入上、下限 和选项以及 包含和不包 含的标准。
如果想创建 参数，则显 示可在其中 输入参数值 的窗口。
保存新变量 。
不要忘记将 新变量名输 入到变式中 以便使用。

复制变量值
要复制变量 ，请输入原 变量的名称 和类型并选 择“复制” 。
随后将显示 窗口，在该 窗口中，可 以指定副本 名称。通过 选择“复制 ”保存。
现在可以随 意更改新变 量。但是， 不要忘记将 新变量名输 入到变式中 以便使用。
也可以通过 使用“分类 ”选项从显 示的列表中 复制变量。

删除变量
要删除变量 ，请键入变 量的名称和 类型，然后 选择“删除 ”。

在系统随后 显示的窗口 中确认或取 消决定。
也可以通过 使用“分类 ”选项从显 示的列表中 删除变量。

使用变式执 行报表程序

要用变式执 行报表程序 ,请在 ABAP/4 编辑器的初 始屏幕上输 入报表名称 ,并选择“ 使用变式执 行”。必须在随后 出现的对话 窗中输入所 需的变式名 。

如果不知道 变式名，则 单击可能的 输入按钮。 系统会显示 现有变式的 列表。标记 所需的变式 并将它复制 下来。在对 话窗口中选 择“执行” 以便用选择 的变式启动 报表程序。

显示报表的 选择屏幕。 那些由变式 覆盖的字段 已包含值。

如果需要， 请填写其它 字段并选择 “执行”。 系统会显示 所需的列表 。

第四章 通过事件控 制 ABAP/4 程序流

概览

内容

ABAP/4 是事件驱动 语言。这意 味着通过外 部事件控制 一般的 ABAP/4 程序流。程 序的一部分 组成过程块 ，并将该过 程块分配到 特定的事件 。
事件关键字 提供了事件 和程序之间 的接口。系 统总是在相 应事件出现 时启动过程 块（参见 *ABAP/4 中 流控制的概 念*（页 **错误！链接无效。**）） 。
在*控制 ABAP/4 程序流*（页 **错误！链接无效。**）中介绍了过 程块中的流 控制（内部 控制）。本 节介绍 ABAP/4 程序的外部 控制和可能 涉及到的事 件。
可以学到

定义过程块

可以在 ABAP/4 中通过使用 事件关键字 定义过程块 。在 *事件及其事 件关键字*（页 255）中对可能的 事件关键字 进行了说明 。

> 两个事件关 键字之间或 事件关键字 与 FORM 语句之间的 所有语句（ 参见 *定义子程序*（页 **错误！链接无效。**））构成过程 块。出现事 件时，系统 将在相应事 件关键字之 后处理过程 块。
>
> ABAP/4 报表程序中 的所有语句 都是过程块 或子程序的 一部分。

没有紧跟着 事件关键字 或 FORM-ENDFORM 块的语句自 动成为默认 事件 START-OF-SELECTION 过程块的一 部分（关于 此默认事件 的详细信息 ，参见 *START-OF-SELECTION*（页 255））。这样将 有下列影响 ：

如果在 REPORT 或 PROGRAM 语句与第一 个事件关键 字或 FORM 语句之间写 语句，则将 把这些语句 包括在 START-OF-SELECTION 过程块中。

- 如果没 有把 START-OF-SELECTION 关键字包括 在报表中， 则这些语句 构成整个 START-OF-SELECTION 过程块。
- 如果将 START-OF-SELECTION 关键字包括 在报表中， 则将把这些 语句插入到 此块的开始部分。

如果没有 在程序中指 定事件关键 字，则该程 序 FORM 语句之前的 所有语句构 成 START-OF-SELECTION 过程块。注 意，这种情 况下，永远 不会执行 FORM - ENDFORM 块之后的所 有语句。

因为通过事 件唯一切换 执行顺序， 所以在程序 中过程块的 出现顺序是 没有任何关 系的。但是 ，为了保证 可读性，在 程序中应该 以过程块的 执行顺序排 列它们。
与操作或控 制语句不一 样，系统在 生成程序时 而不是在运 行时间中执 行说明性语 句（关于说 明性语句的 详细信息， 参见 *关键字*（页 **错误！链接无效。**））。对它们 的执行与它 们在程序代 码中的位置 无关，并且 可以属于任 何过程块。 为清楚起见 ，应该把所 有说明性的 关键字放到 程序或子程 序的开始处 。

```
REPORT SAPMZTST.
```

```
WRITE / 'Statement 1'.

FORM ROUTINE.
   WRITE / 'Subroutine'.
ENDFORM.

WRITE / 'Statement 2'.
PERFORM ROUTINE.
WRITE / 'Statement 3'.
```

输出如下所 示：

```
Statement 1
```

在此程序中 ，只执行了 START-OF-SELECTION 过程块。此 块包含第一 个 WRITE 语句。

现在，按照 下列方法把 START-OF-SELECTION 语句插入程 序中：

```
REPORT SAPMZTST.

WRITE / 'Statement 1'.

FORM ROUTINE.
   WRITE / 'Subroutine'.
ENDFORM.

START-OF-SELECTION.
   WRITE / 'Statement 2'.
   PERFORM ROUTINE.
   WRITE / 'Statement 3'.
```

输出如下所 示：

```
Statement 1

Statement 2

Subroutine

Statement 3
```

在此程序中 ，START-OF-SELECTION 过程块包含 除 FORM-ENDFORM 块之外的所 有语句。下 列是此程序 更易读的形 式：

```
REPORT SAPMZTST.

START-OF-SELECTION.
   WRITE / 'Statement 1'.
   WRITE / 'Statement 2'.
   PERFORM ROUTINE.
   WRITE / 'Statement 3'.

FORM ROUTINE.
   WRITE / 'Subroutine'.
ENDFORM.
```

在程序的这 种形式中， 也可以忽略 START-OF-SELECTION 语句。

ABAP/4 处理程序

报表程序是 过程块的集 合，执行该 程序以对特 定事件作出 反应，特别 是使用逻辑 数据库时。 可以把这些 块视为可调 用模块。在 程序代码中 ，它们不需 要以任何特 殊的顺序出 现。一旦调 用报表程序 ，系统就启 动另一个进 程（控制程 序），该进 程调用这些 模块，并控 制外部程序 流。

此控制程序 是 ABAP/4 处理程序。 它监视 ABAP/4 报表程序、 逻辑数据库 程序和其他 程序模块（ 例如对话屏 幕）之间的 交互作用。 它也解释 ABAP/4 程序的运行 时间对象。 必须生成每 个 ABAP/4 程序，在 ABAP/4 编辑器中编 写这些程序 ，以创建其 运行时间对 象。这既适 用于报表程 序，也适用 于逻辑数据 库程序。即 可以在 ABAP/4 编辑器中通 过选择“程 序 -> 生成”执行 该生成，也 可以在“ABAP/4 编辑器：初 始屏幕”上 通过选择“ 生成”执行 。第一次启 动报表程序 时，自动生 成该报表程 序。

通过在程序 中使用的事 件关键字控 制 ABAP/4 处理程序和 不同程序之 间的交互作 用。例如

在报表程 序 START-OF-SELECTION、 GET 中，等等，

参见 *事件及其事 件关键字*（页 255），

在逻辑数 据库程序 PUT 中，等等，

参见 *逻辑数据库 的特征和维 护*（页 **Error! Not a valid link.**）。

根据由事件 关键字定义 的流，ABAP/4 处理程序解 释相应程序 单元，并开 始处理它们 。

事件及其事 件关键字

这里有几组 事件关键字 ，这些事件 关键字在特 定环境下控 制 ABAP/4 程序流。

逻辑数据库 是典型报表 程序的外部 流控制的中 心点（参见 *用逻辑数据 库访问数据*（页 **错误！链接无效。**））。如果将 逻辑数据库 链接到报表 程序，将导 致显示选择 屏幕，并决 定系统如何 从数据库表 中

读取数据 。这引起在 下列表中说 明的事件序 列。关于事 件出现顺序 的详细信息 ，参见 *逻辑数据库和 ABAP/4 报表* (页 **Error! Not a valid link.**) 。
下列事件出 现在典型报 表程序的运 行时间处， 该报表程序 使用逻辑数 据库：

事件关键字	事件
INITIALIZATION (页 255)	在显示选择 屏幕之前的 点
AT SELECTION-SCREEN (页 246)	选择屏幕仍 然活动时， 处理用户在 选择屏幕上 输入之后的 点
START-OF-SELECTION (页 255)	处理选择屏 幕之后的点
GET <table> (页 256)	点位于逻辑 数据库提供 数据库表 <table> 的行处。
GET <table> LATE (页 255)	处理所有表 之后的点， 在逻辑数据 库的结构中 ，使这些表 位于数据库 表 <table> 的下层。
END-OF-SELECTION (页 244)	处理完逻辑 数据库提供 的所有行之 后的点。

下 列主题说明 这些事件的 过程块。
处理和显示 报表程序输 出列表时， 出现其他没 有连接到逻 辑数据库的 事件。可以 用这些事件 格式化输出 列表，并使 报表程序成 为交互式的 。在适当的 节中将继续 说明这些事 件。
处理报表程 序输出列表 时，出现下 列事件：

事件关键字	事件
TOP-OF-PAGE	启 动新页时， 列表处理中 的点
END-OF-PAGE	结 束页时，列 表处理中的 点

可以用这 些关键字改 善输出列表 的格式。说 明参见*创建列表* (页 **Error! Not a valid link.**) 。
下列事件出 现在报表程 序输出列表 的显示中：

事件关键字	事件
AT LINE-SELECTION	用 户在该点处 选择行
AT USER-COMMAND	用 户在该点处 按下功能键 或在命令字 段中输入命 令
AT PF<nn>	用 户在该点处 按下有功能 代码 PF<n> 的功能键

可 以使用这些 关键字编写 交互式报告 的程序。详 情参见 *交互式列表* (页 **Error! Not a valid link.**) 。

关于所有事 件关键字的 详细说明， 参见事件关 键字文档。

INITIALIZATION

启动已定义 选择屏幕的 程序（在程 序自身或在 被链接的逻 辑数据库程 序中）时， 系统通常首 先处理此选 择屏幕。如 果希望在处 理选择屏幕 之前执行过 程块，可以 把它分配到 事件关键字 INITIALIZATION。
在此块中， 指定初始化 选择屏幕的 语句，例如 通过更改参 数或选择标 准的默认值 。这只对在 逻辑数据库 中定义的参 数或选择标 准才有效。 对于选择标 准，应该通 过更改选择 表 <seltab> 的表头行并 把它附加到 表中，至少 定义选择表 <seltab> 的组件 <seltab>-SIGN、 <seltab >-OPTION、<seltab>-LOW (参见 *选择表* (页 **错误！链接无效。**))。否则， 可能未定义 部分选择标 准。
如果希望通 过检查逻辑 数据库 SAPDB<ldb> 自己（使用 事务 SLDB 或通过选择 “工具 -> ABAP/4 开发工作台 -> 开发 -> 编程环境 -> 逻辑数据库 ”），或通 过检索字段 的技术信息 而更改内部 字段，可以找到该字段 的名称。为 此，在选择 屏幕上选择 输入字段， 并按下 **F1**。然后 ，选择对话 框中的“技 术信息”。 在下列窗口 的字段“屏 幕字段”中 ，将会看到 程序中使用 的字段名称 。

假定有链接 到逻辑数据 库 F1S 的报表程序 ：

```
REPORT SAPMZTST.

PARAMETERS FIRSTDAY LIKE SY-DATUM DEFAULT SY-DATUM.

TABLES SPFLI.
```

启动此程序 时，将自动 出现下列选 择屏幕：

在逻辑数据 库 F1S 中定义有选 择文本“承 运方 ID”的选 择标准，以 及有选择文 本“从”和 “到”的参 数（参见 *选择文本* (页 **错误！链接无效。**) ）。在程序 自身中定义 参数 FIRSTDAY。

例如，现在 选择“承运 方 ID”的第 一个输入字 段，按下 **F1**，然后 选择“技术 信息”，以 找到选择表 的名称：

在字段“屏 幕字段”中 ，将看到名 称 CARRID-LOW， 它是对应于 选择输入的 选择表的组 件。从这里 看到选择标 准的名称是 CARRID。 在上面所述 的相同过程 中，可以发 现把输入字 段“从”和 “到”命名 为 CITY_FR 和 CITY_TO。

现在，可以 按照下列方 法更改报表 程序：

```
REPORT SAPMZTST.

PARAMETERS FIRSTDAY LIKE SY-DATUM DEFAULT SY-DATUM.

TABLES SPFLI.

INITIALIZATION.
   CITY_FR = 'NEW YORK'.
   CITY_TO = 'FRANKFURT'.
   CARRID-SIGN   = 'I'.
   CARRID-OPTION = 'EQ'.
   CARRID-LOW    = 'AA'.
   APPEND CARRID.
   FIRSTDAY+6(2) = '01'.
```

启动 SAPMZTST 之后，选择 屏幕如下：

更改了选择 标准的默认 值和所有参 数。

AT SELECTION-SCREEN

事件关键字 AT SELECTION-SCREEN 提供了好几 种在系统处 理选择屏幕 时执行过程 块的方法。 为了对处理 选择屏幕时 出现的不同 事件都有效 ，关键字 AT SELECTION-SCREEN 有不同选项 。
与 PARAMETERS 和 SELCECT-OPTIONS 语句的特殊 选项一起， 在 *使用选择屏幕* (页 **错误！链接无效。**) 中显示了关 键字 AT-SELECTION 屏幕的某些 应用。
如果指定字 段的关键字 没有任何选 项，则在系 统处理完选 择屏幕之后 启动相应过 程块。如果 从此过程块 中发送 ERROR MESSAGE， 则系统将又 显示选择屏 幕，并且可 以更改所有 输入字段。 必须提供适 当的 ERROR MESSAGE。
例如，此方 法可以输入 字段命令， 尽管没有在 逻辑数据库 程序中使用 PARAMETERS 或 SELECT-OPTIONS 语句的 OBLIGATORY 选项定义它 们。

在表 T100 中存储和维 护 MESSAGE。 通过语言、 两个字符的 ID 和三位数字 对它们进行 分组，可以 从程序中以 不同条件发 送 MESSAGE：

A：Abend， 终止当前事 务

E：错误 ，系统等待 新输入数据

I：信息 ，处理 ENTER 之后，系统 继续处理

S：确认 ，消息出现 在下一屏上

W：警告 ，可以更改 输入数据或 通过按 ENTER 继续

必须在程序 的 REPORT 或 PROGRAM 语句之后指 定 MESSAGE-ID。 通过 ABAP/4 编辑器，选 择“编辑 -> 插入语句...” ，可以很容 易地将 MESSAGE 包括到程序 中。也可以 从这里更改 MESSAGE。 关于报表中 MESSAGE 处理的详细 信息，参见 *列表中的消 息* (页 Error! Not a valid link.) 。

将逻辑数据 库 F1S 附加到下列 报表程序：

```
REPORT SAPMZTST MESSAGE-ID HB.

TABLES SPFLI.

AT SELECTION-SCREEN.
   IF CARRID-LOW IS INITIAL
      OR CITY_FR IS INITIAL
      OR CITY_TO IS INITIAL.
      MESSAGE E000.
   ENDIF.
```

此报表使用 有 ID HB 的 MESSAGE。 启动 SAPMZTST 之后，选择 屏幕显示在 逻辑数据库 F1S 中定义的内 容。一旦用 户不把值输 入到每个输 入字段中， 下列错误信 息就将出现 在屏幕的状 态栏中。

在表 T100 中，以 ID HB 为此例子编 写了 MESSAGE 000 代码。

事件关键字 AT SELECTION-SCREEN 的选项使您 能够在处理 选择屏幕时 为特定事件 创建过程块 。在下列主 题中说明这 些事件：

在下列主题 中即将介绍 这些选项。 详细信息， 参见关于 AT SELECTION-SCREEN 的关键字文 档。

处理特殊输 入字段
要在处理选 择屏幕的特 殊输入字段 之后启动过 程块，请按 照下列方法 使用关键字 AT SELECTION 屏幕：

语法：
AT SELECTION-SCREEN ON <field>.
系统处理了 变量〈field〉的输入字段 之后，启动 相应过程块 。如果从此 过程块中发 送 ERROR MESSAGE， 则系统再次 显示选择屏 幕，用户仅 必须更改变 量〈field〉的输入字段 。

将逻辑数据 库 F1S 附加到下列 报表程序。

```
REPORT SAPMZTST MESSAGE-ID HB.

TABLES SPFLI.

AT SELECTION-SCREEN ON CITY_FR.
   IF CITY_FR NE 'NEW YORK'.
      MESSAGE E010.
   ENDIF.
```

如果用户没 有在选择屏 幕中的字段“从”中插 入“NEW YORK”，则下列 ERROR MESSAGE 会出现在屏 幕的状态栏 中，

，

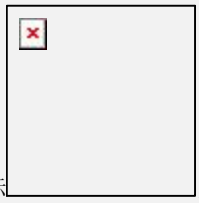
直到用户作 出正确输入 。在表 T100 中，以此 ID HB 为例编写了 MESSAGE 010 代码。

处理多重选 择
将特殊选择 标准的复杂 选择输入到 选择屏幕的 “多重选择 ”窗口，并 处理此窗口 之后（参见 *SELECT-OPTIONS 语句的基本格式*（页 **错误！链接无效。**）中的例子） ，可以调用 过程块，为 此，请按照 下列方法使 用 AT SELECTION-SCREEN 语句：

语法
AT SELECTION-SCREEN ON END OF <seltab>.
在处理选择 标准〈seltab〉的“复杂选 择”窗口结 束处启动相 应过程块。 可以使用这 些选项检查 内表〈seltab〉中的条目。

将逻辑数据 库 F1S 附加到下列 报表程序中 。

```
REPORT SAPMZTST MESSAGE-ID HB.

TABLES SPFLI.

AT SELECTION-SCREEN ON END OF CARRID.
   LOOP AT CARRID.
      IF CARRID-HIGH NE '   '.
         IF CARRID-LOW IS INITIAL.
            MESSAGE W020.
         ENDIF.
      ENDIF.
   ENDLOOP.
```

启 动 SAPMZTST 之后，如果 用户在选择 屏幕上单击 箭头图标  ， 然

后在“多 重选择”窗 口中输入范 围选择的上 限，而不输 入下限，则 出现下列对 话框，并发 出警告：

在表 T100 中，以此 ID HB 为例编写了 MESSAGE 020 代码。

创建输入值 列表
You 通过按照下 列方法使用 AT SELECTION-SCREEN 语句，可以 为选择屏幕 上的输入字 段创建可能 输入值的列 表：

语法
AT SELECTION-SCREEN ON VALUE-REQUEST FOR <field>.
如果使用此 语句，在选 择屏幕上选 定参数或选 择标准 <field> 的输入字段 时，会自动 紧跟着该字 段出现可能 的登录按钮 。

只可以在报 表程序中使 用此语句。 在逻辑数据 库程序中， 可以使用 PARAMETERS 和 SELECT-OPTIONS 语句的 VALUE-REQUEST 选项(参见 关键字文档)。

必须在 AT SELECTION-SCREEN ON VALUE REQUEST 语句的过程 块内为 <field> 编写建议值 列表程序。 用户单击可 能的登录按 钮或按下 **F4** 时，将显示 此列表。如 何编写这种 列表的代码 是对话编程 的范畴，并 在*编写字 - 和值 - 帮助程序* (页 **Error! Not a valid link.**) 中作了介 绍。

PARAMETERS FIELD(10).

AT SELECTION-SCREEN ON VALUE-REQUEST FOR FIELD.

参数如下：

如果为 FIELD 编写了建议 值列表，并 且用户单击 了可能的登 录按钮，则 将显示它。

创建输入字 段的帮助
通过按照下 列方法使用 AT SELECTION-SCREEN 语句，可以 为选择屏幕 上的输入字 段创建自己 的帮助：

语法
AT SELECTION-SCREEN ON HELP-REQUEST FOR <field>.
如果使用此 语句，用户 在选择屏幕 上选择 <field> 的输入字段 ，并按下 **F1** 键时，将显 示帮助文本 。

只可以在报 表程序中使 用此语句。 在逻辑数据 库程序中， 使用 PARAMETERS 和 SELECT-OPTIONS 语句的 HELP-REQUEST 选项（参见 关键字文档 ）。

必须在 AT SELECTION-SCREEN ON HELP REQUEST 语句的过程 块内编写帮 助文本的程 序。如何编 写这种帮助 的代码是对 话编程的范 畴，并在 *编写字 - 和值 - 帮助程序* (页 **Error! Not a valid link.**) 中作了介绍 。

处理单选按 钮组
在选择屏幕 上处理完单 选按钮组之 后(参见 *在选择屏幕 上创建单选 按钮组* (页 **错误!链接无效。**))，要启动 过程块，请 按照下列方 法使用关键 字 AT SELECTION 屏幕：

语法：
AT SELECTION-SCREEN ON RADIOBUTTON GROUP <radi>.
系统处理单 选按钮组 <radi> 后启动相应 过程块。如 果从过程块 中发送 ERROR MESSAGE， 则系统又显示选择屏幕 ，用户只必 须更改单选 按钮 <radi> 的输入字段 。

```
REPORT SAPMZTST MESSAGE-ID HB.

PARAMETERS: R1 RADIOBUTTON GROUP RAD1 DEFAULT 'X',
            R2 RADIOBUTTON GROUP RAD1,
            R3 RADIOBUTTON GROUP RAD1.

AT SELECTION-SCREEN ON RADIOBUTTON GROUP RAD1.
   IF R1 = 'X'.
      MESSAGE I030.
   ENDIF.
```

如果用户没 有更改选择 屏幕上的单 选按钮，将 出现下列 INFORMATION MESSAGE：

在表 T100 中，以此 ID HB 为例编写了 MESSAGE 030 代码。

处理输入字 段块

在选择屏幕 上处理完元 素块（参见*创建元素块*（页 **错误！链接无效。**））之后启动 过程块，请 按照下列方 法使用关键 字 AT SELECTION 屏幕：

语法：

```
AT SELECTION-SCREEN ON BLOCK <block>.
```

系统处理完 元素块〈block〉时启动相应 过程块。如 果从此过程 块中发送 ERROR MESSAGE， 则系统再次 显示选择屏 幕，用户只 必须更改块〈block〉的输入字段 。

```
REPORT SAPMZTST MESSAGE-ID HB.

SELECTION-SCREEN BEGIN OF BLOCK PART1
                              WITH FRAME TITLE TEXT-001.
   PARAMETERS: NUMBER1 TYPE I,
               NUMBER2 TYPE I,
               NUMBER3 TYPE I.
SELECTION-SCREEN END OF BLOCK PART1.

AT SELECTION-SCREEN ON BLOCK PART1.
   IF NUMBER3 LT NUMBER2 OR
      NUMBER3 LT NUMBER1 OR
      NUMBER2 LT NUMBER1.
      MESSAGE E040.
   ENDIF.
```

启动 SAPMZTST 后，用户可 以把号码输 入到块 PART1 的输入字段 中。如果他 没有按升序 输入号码，

则下列 ERROR MESSAGE 将出现在状 态栏中：

在表 T100 中，以此 ID HB 为例编写了 MESSAGE 040 代码。

选择屏幕的 PBO

要在每个 **ENTER** 的选择屏幕 PBO 中启动过程 块，请使用 下列 AT SELECTION-SCREEN 语句：

语法

```
AT SELECTION-SCREEN OUTPUT.
```

例如，在这 些事件中， 可以申请对 选择屏幕上 的字段作出 修改。详情 和例子，参 见 *给修改组分 配参数*（页 **错误！链接无效。**）。

START-OF-SELECTION

过程块的可 能。例如， 可以使用这 些过程块设 置内表的值 ，或将信息 语句写到输 出屏幕上。
在 START-OF-SELECTION 事件处，也 处理没有附 加到事件关 键字的所有 语句，或在 FORM-ENDFORM 块后写这些 语句（例子 参见 *定义过程块*（页 260））。

GET <table>

对于有附加 逻辑数据库 的报表程序 ，最重要的 事件是逻辑 数据库程序 从数据库表 中读取了行 的时刻（参 见*用逻辑数据 库访问数据*（页 **错误！链接无效。**））。要在此 事件处启动 过程块，请 按照下列方 法使用 GET 语句：

语法
GET <table> [FIELDS <list>].
此语句之后 ，可以使用 数据库表〈table〉的当前行。 在表工作区〈table〉中提供了数 据。
逻辑数据库 从**所有**数据 库表中读取 **所有**列，没 有为逻辑数 据库中的字 段指定这些 数据库表， 并且这些数 据库表优于 逻辑数据库 访问路径上 的〈table〉(参见 *从报表程序 中控制数据 库访问*（页 **错误！链接无效。**））。不管是 否为这些表 指定了 GET 语句，这都 与事实无关 。只有在程 序中使用 TABLES 语句声明表 之后，才可 以访问这些 表的数据。
对于为逻辑 数据库中的 选择字段指 定的表，性 能可以更好 一些（参见 *编辑选择*（页 Error! Not a valid link.））。如果这 样的表优于 逻辑数据库 访问路径上 的〈table〉，并且没有为 它们指定 GET 语句，则系 统只从在程 序中用 TABLES 语句声明的 表中读取所 有列的数据 。从在程序 中没有用 TABLES 语句声明的 表中，系统 只读取关键 字段（逻辑 数据库需要 关键字段生 成访问路径 ）。
使用 FIELDS 选项，可以 外在地指定 应该由逻辑 数据库读取 的数据库表 的列。在 *外在地指定 数据库表的 字段*（页 240）中对 FIELDS 选项作了介 绍。

将逻辑数据 库 F1S 附加到了下 列报表程序 。F1S 的结构为：

报表程序 SAPMZTST 的代码为：

```
REPORT SAPMZTST.

TABLES: SPFLI, SFLIGHT, SBOOK.

START-OF-SELECTION.
   WRITE 'Test Program for GET'.

GET SPFLI.
   SKIP.
   WRITE: / 'From:', SPFLI-CITYFROM,
            'To:',   SPFLI-CITYTO.

GET SFLIGHT.
   SKIP.
   WRITE: / 'Carrid:', SFLIGHT-CARRID,
            'Connid:', SFLIGHT-CONNID.
   ULINE.

GET SBOOK.
   WRITE: / 'Fldate:',     SFLIGHT-FLDATE,
            'Bookid:',     SBOOK-BOOKID,
            'Luggweight', SBOOK-LUGGWEIGHT.
   ULINE.
```

启动 SAPMZTST 之后，假定 用户按照下 列方法填写 选择屏幕：

这样，输出 列表的第一 部分如下：

注意，在 GET SBOOK 后的过程块 中使用表工 作区 SPFLI。

外在地指定 数据库表的 字段
要指定在 GET 事件处使用 数据库表的 哪一个字段 ，请按照下 列方法使用 GET 语句的 FIELDS 选项：

语法
GET <table> [LATE] FIELDS <F1> <F2> ...
使用 FIELDS 选项，从数 据库表〈table〉中，逻辑数 据库程序只 读取字段 〈**F1**〉〈**F2**〉... 和关键字段 。FIELDS 选项的使用 可以引起**相 应性能改善** 。

链接到报表 程序的逻辑 数据库必须 为字段选择 指定数据库 <table>(参见 *编辑选择* (页 Error! Not a valid link.))。

逻辑数据库 不读取所有 非关键字段 的字段和所 有没有在 FIELDS 后列出的字 段。在 GET 事件中,没 有使用 FIELDS 选项定义表 工作区 <table> 的相应组件 内容。也没 有在数据库 表的 GET 事件中定义 它们,这些 数据库表次 于逻辑数据 库分层中的 <table>。 不应该定位 这些未定义 的字段,并 且不调用外 部子程序, 程序使用这 些字段。

假定逻辑数 据库 F1S 为字段选择 指定了数据 库表 SFLIGHT 和 SBOOK。这样,可以 写下列程序 :

```
TABLES: SFLIGHT, SBOOK.

GET SFLIGHT FIELDS CARRID CONNID.

GET SBOOK   FIELDS CUSTOMID.

GET SFLIGHT LATE FIELDS PLANETYPE.
```

在此例中, 系统

从 SFLIGHT 中读取字段 MANDT、 CARRID、 CONNID、 FLDATE 和 PLANETYPE

从 SBOOK 中读取字段 MANDT、 CARRID、 CONNID、 FLDATE 和 BOOKID

注意, 系统 从 SFLIGHT 中读取字段 MANDT 和 FLDATE from SFLIGHT, 这是因为 MANDT 和 FLDATE 是此表的关 键字段。

在此例子中 , 从 SBOOK 中只读取关 键字段。

GET <table> LATE

要在系统处 理完逻辑数 据库(它们 在层次上低 于特定数据 库表)的所 有数据库表 之后启动过 程块,请按 照下列方法 使用事件关 键字 GET:

语法

```
GET <table> LATE [FIELDS <list>].
```

与只使用 SELECT 语句的报表 程序(参见 *访问方法比 较* (页 **错误! 链接无效。**) 中的表) 相 似, 在数据 库表 <table> 的 SELECT 循环中, GET <table> LATE 语句的过程 块直接出现 在 ENDSELECT 语句的前面 。

FIELDS 选项的作用 与 GET <table> 事件相同, 在 *外在地指定 数据库表的 字段* (页 267) 中对它作了解释。

将逻辑数据 库 F1S 连接到下列 报表程序。

```
REPORT SAPMZTST.

TABLES: SPFLI, SFLIGHT, SBOOK.

DATA WEIGHT TYPE I VALUE 0.

START-OF-SELECTION.
   WRITE 'Test Program for GET <table> LATE'.

GET SPFLI.
   SKIP.
   WRITE: / 'From:',    SPFLI-CITYFROM,
            'To:',      SPFLI-CITYTO,
            'Connid:', SPFLI-CONNID.
   ULINE.

GET SFLIGHT.
   SKIP.
   WRITE: / 'Date:', SFLIGHT-FLDATE.

GET SBOOK.
   WEIGHT = WEIGHT + SBOOK-LUGGWEIGHT.

GET SFLIGHT LATE.
   WRITE: / 'Total luggage weight =', WEIGHT.
```

```
   ULINE.
   WEIGHT = 0.
```

对于 *GET 〈table〉*（页 267）例子中的相 同选择，输 出屏幕的第 一部分如下 ：

在事件 GET SBOOK 处为每个航 班计算行李 总重量。将 它写到屏幕 上，并在事 件 GET SFLIGHT LATE 处重新设置 它。

END-OF-SELECTION

要在系统读 取和处理完 逻辑数据库 的所有数据 库表之后定 义过程块， 请使用关键 字 END-OF-SELECTION。

将逻辑数据 库 F1S 连接到下列 报表程序。

```
REPORT SAPMZTST.

TABLES SBOOK.

DATA NUMBER TYPE I VALUE 0.

START-OF-SELECTION.
   WRITE 'Test Program for END-OF-SELECTION'.
   SKIP.
   WRITE: / CITY_FR, CITY_TO.

GET SBOOK.
   NUMBER = NUMBER + 1.

END-OF-SELECTION.
   WRITE: / 'Total number of bookings:', NUMBER.
```

对于 *GET 〈table〉*（页 267）例子中的相 同选择，输 出如下：

在事件 GET SBOOK 处计算从法 兰克福到柏 林的预订号 并编写到事 件 END-OF-SELECTION 的屏幕中。

终止过程块

ABAP/4 提供了好几 种离开过程 块的语句。 取决于当前 事件，不同 语句可以将 程序流转向 不同过程块 。可以无条 件地或有条 件地全部离 开过程块的 语句。
下列主题介 绍
如果使用逻 辑数据库访 问数据库表 ，GET 语句后的过 程块终止有 某些特殊性 质。另外两 个主题介绍

无条件地离 开过程块

可以通过下 列方法无条 件地离开过 程块：

转向 END-OF-SELECTION
可以立即离 开任何过程 块，并通过 按照下列方 法使用 STOP 语句转向 END-OF-SELECTION 过程块：

语法
```
STOP.
```
运行 STOP 语句后，系 统执行 END-OF-SELECTION 过程块，并 停止运行程 序。

将逻辑数据 库 F1S 附加到下列 报表程序：

```
REPORT SAPMZTST.

TABLES: SPFLI, SFLIGHT, SBOOK.

START-OF-SELECTION.
   WRITE 'Test Program for STOP'.
```

```
GET SBOOK.
   WRITE: 'Bookid', SBOOK-BOOKID.
   STOP.

END-OF-SELECTION.
   WRITE: / 'End of selection'.
```

输出如下：

```
Test Program for STOP
Bookid 00010001
End of selection
```

从 SBOOK 中读取第一 行之后，系 统直接执行 END-OF-SELECTION 语句块。

转向输出屏 幕

可以立即离 开任何过程 块（除在 AT 事件以外） ，并通过按 照下列方法 使用 EXIT 语句转向输 出屏幕：

语法

```
EXIT.
```

EXIT 语句后，系 统显示输出 列表，并停 止运行程序 。它不执行 END-OF-SELECTION 过程块。

EXIT 语句在 AT 事件的过程 块中的功能 不同（参见 *离开 AT 事件* (页 252) ）。

关于 EXIT 语句在循环 中有何种功 能的详细信 息，参见 *中止循环* (页 **错误！链接无效。**) 。

将逻辑数据 库 F1S 附加到下列 报表程序：

```
REPORT SAPMZTST.

TABLES: SPFLI, SFLIGHT, SBOOK.

START-OF-SELECTION.
   WRITE 'Test Program for EXIT'.

GET SBOOK.
   WRITE: 'Bookid', SBOOK-BOOKID.
   EXIT.

END-OF-SELECTION.
   WRITE: / 'End of selection'.
```

输出如下：

```
Test Program for EXIT
Bookid 00010001
```

读取 SBOOK 的第一行之 后，系统直 接处理输出 列表，而不 是 END-OF-SELECTION 语句块。

离开 AT 事件

如果在 AT 事件的过程 块中使用 EXIT 语句（有以 AT 开头的事件 关键字的所 有事件，参 见 *事件及其事 件关键字* (页 261) ），则系统 离开此过程 块，并转向 下一个出现 事件的过程 块。

在所有其他 事件的过程 块中，EXIT 转向输出屏 幕。

关于 EXIT 语句在循环 中如何起作 用的详细信 息，参见 *中止循环* (页 **错误！链接无效。**) 。

将逻辑数据 库 F1S 附加到下列 报表程序：

```
REPORT SAPMZTST.

TABLES: SPFLI, SFLIGHT, SBOOK.

DATA FLAG.

AT SELECTION-SCREEN.
   IF CARRID-LOW IS INITIAL.
      FLAG = 'X'.
      EXIT.
   ENDIF.
   ........

START-OF-SELECTION.
   IF FLAG = 'X'.
      WRITE / 'No selection for CARRID made'.
      EXIT.
   ENDIF.

GET SPFLI.
   .........

GET SFLIGHT.
   .........

GET SBOOK.
   .........

END-OF-SELECTION.
   WRITE / 'End of Selection'.
```

如果用户没 有在“承运 方 ID”字段 中输入值， 则输出如下 ：

```
No selection for CARRID made
```

注意，EXIT 语句是与上 下文紧密相 关的。第一 个 EXIT 语句后，也 就是执行 START-OF-SELECTION 块。第二个 EXIT 语句后，将 显示输出列 表。

有条件地离 开过程块

通过按照下 列方法使用 CHECK 语句的两个 变式，可以 有条件地离 开任何过程 块：

语法
CHECK <condition>.
如果 CHECK 语句中的条 件为**假**，则 系统离开过 程块并转向 下一个出现 事件的过程 块。对于 <condition>， 可以使用 *编程逻辑表达式* (页 **错误！链接无效。**) 中介绍的任 何逻辑表达 式。

语法
CHECK <seltab>.
如果数据库 表的表工作 区的内容与 选择表 <seltab> 中的条件不 匹配，并且 将选择表 <seltab> 附加到该数 据库表（参 见*在 GET 事件中与 CHECK 语句一起使 用选择表* (页 **错误！链接无效。**))，则系统离开过程块 。

CHECK 语句是与上 下文紧密相 关的。关于 GET 事件中下一 个出现事件 的定义，参 见 *有条件地离 开 GET 事件* (页 240) 。

关于 CHECK 语句在循环 中如何起作 用的详细信 息，参见 *中止循环* (页 **错误！链接无效。**) 。

将逻辑数据 库 F1S 附加到下列 报表程序：

```
REPORT SAPMZTST.

TABLES: SPFLI, SFLIGHT, SBOOK.

START-OF-SELECTION.
   CHECK CITY_FR NE ' '.
   WRITE: / 'Selected City-From:', CITY_FR.
   ULINE.
   CHECK CITY_TO NE ' '.
   WRITE: / 'Selected City-To:', CITY_TO.
   ULINE.

GET SFLIGHT.
   WRITE: / 'Connid:', SFLIGHT-CONNID,
            'Carrid:', SFLIGHT-CARRID,
            'Fldate:', SFLIGHT-FLDATE.
```

假定下列选 择：

输出屏幕的 第一部分如 下：

第二个 CHECK 语句后，系 统离开 START-OF-SELECTION 过程块，并 转向 GET SFLIGHT 事件。

无条件地离 开 GET 事件

要无条件地 离开 GET 事件的过程 块，有四种 可能：

转向当前数 据库表的下 一行

要离开 GET 语句的过程 块，并转向 逻辑数据库 同一层上的 下一 GET 事件，请按 照下列方法 使用 REJECT：

语法

REJECT.

此语句后， 系统立即处 理相同数据 库表的下一 GET 事件。这意 味着它从当 前表中启动 相同过程块 的新行。

REJECT 语句与上下 文无关。在 循环和子程 序中它也有 相同效果。

将逻辑数据 库 F1S 连接到下列 报表程序。

```
REPORT SAPMZTST.

TABLES: SPFLI, SFLIGHT, SBOOK.

GET SFLIGHT.
   SKIP.
   WRITE: / 'Connid:', SFLIGHT-CONNID,
            'Carrid:', SFLIGHT-CARRID,
            'Fldate:', SFLIGHT-FLDATE.
   ULINE.

GET SBOOK.
   IF SBOOK-BOOKID GT 00010002.
      REJECT.
   ENDIF.
   WRITE: / 'Bookid:', SBOOK-BOOKID.
```

对于 *GET 〈table〉*（页 267）例子中的相 同选择，输 出屏幕的第 一部分如下 ：

对每个航班 ，只显示头 两个预订。 但是，请注 意系统读取 每个预订。 如何改善性 能的例子， 参见 *转向上级数 据库表的下 一行* (页 233) 中的例子。

转向上级数 据库表的下 一行
要离开 GET 语句的过程 块，并执行 逻辑数据库 高层的下一 GET 事件，请按 照下列方法 使用 REJECT 语句：

语法
REJECT <dbtab>.
执行此语句 后，系统将 立即处理数 据库表 <dbtab> 的下一 GET 事件。在链 接到当前数 据库表的分 层结构中， 相比之下， 数据库表 <dbtab> 必须位于高 层。

将逻辑数据 库 F1S 连接到下列 报表程序。

```
REPORT SAPMZTST.

TABLES: SPFLI, SFLIGHT, SBOOK.

GET SFLIGHT.
   SKIP.
   WRITE: / 'Connid:', SFLIGHT-CONNID,
            'Carrid:', SFLIGHT-CARRID,
            'Fldate:', SFLIGHT-FLDATE.
   ULINE.

GET SBOOK.
   IF SBOOK-BOOKID GT 00010002.
      REJECT 'SFLIGHT'.
   ENDIF.
   WRITE: / 'Bookid:', SBOOK-BOOKID.
```

对于 *GET <table>* (页 267) 例子中的相 同选择，输 出与 *转向当前数 据库表的下 一行* (页 Error! Not a valid link.) 中例子的输 出相同。但 是，请注意 ，因为处理完 REJECT 语句之后， 系统转向 GET SFLIGHT， 所以在现在 的例子中， 从 SBOOK 中只读取显 示行。

有条件地离 开 GET 事件

要有条件地 离开 GET 事件，可以 使用 CHECK 语句。用 CHECK 语句离开 GET 事件之后， 系统将执行 逻辑数据库 相同层上的 下一 GET 事件。它读 取当前表的 下一行。没 有处理在逻 辑数据库的 分层顺序中 较靠后的数 据库表。以 CHECK 语句离开 GET 事件所用的 方法与 *转向当前数据库表的下一行* (页 272) 中介绍的方 法相同。
在 GET 事件中，可 以使用 CHECK 语句的两种 变式：

语法
CHECK <condition>.
如果 CHECK 语句中的条 件为**假**，则 系统离开过 程块（参见 *有条件地离 开过程块* (页 271) ）。

语法
CHECK SELECT-OPTIONS.
如果从当前 数据库表中 读取的行与 所有选择表 中的条件不 匹配，并且 将这些选择 表连接到该 数据库表（ 参见 *在 GET 事件中与 CHECK 语句一起使 用选择表* (页 **错误！链接无效。**))，则系统 离开过程块 。

将逻辑数据 库 F1S 连接到下列 报表程序。

```
REPORT SAPMZTST.

TABLES: SPFLI, SFLIGHT, SBOOK.

GET SFLIGHT.
   CHECK SFLIGHT-CARRID EQ 'LH'.
   WRITE: / 'Connid:', SFLIGHT-CONNID,
            'Carrid:', SFLIGHT-CARRID,
            'Fldate:', SFLIGHT-FLDATE.
```

```
GET SBOOK.
   CHECK SBOOK-BOOKID LT 00010003.
   WRITE: / 'Bookid:', SBOOK-BOOKID.

GET SFLIGHT LATE.
   ULINE.
```

如果没有将 任何值填写 到选择屏幕 的字段中， 则输出屏幕 的第一部分 如下：

在上面的例 子中，必须 读取表 SFLIGHT 的所有行， 并且，如果 SFLIGHT-CARRID 是 “LH” ，还必须读 取表 SBOOK 的所有行。 为了提高性 能，请不要 象例子中那 样编写选择 程序，而是 尽可能地使 用逻辑数据 库的选择。

第五章 提炼数据

概览

内容

其它任务中 的报表必须 提炼数据。 即，除了检 索数据，报 表还必须给 数据排序、 计算合计以 及统计列表 中的条目等 等。用户可 以从数据库 表或顺序文 件中读取需 要提炼的数 据，或者在 报表中创建 一般数据。
下面主题包 含了提炼数 据的示例：
本节将讨论 在创建数据 集之后提炼 数据集的数 据。提炼过 程独立于检 索数据的过 程。首先要 创建数据集 ，然后再提 炼该数据集 。
该规则也有 例外，就是 用户一般性 地创建自己 的数据集的 过程，或使 用 SQL 语句直接访 问数据库的 过程。有时 ，在检索过 程中就能对 数据进行充 分的提炼。 例如，参见 检索过程中 提炼数据的 SQL 报表。
使用逻辑数 据库访问数 据库表时， 或从顺序文 件中读取数 据时，或者 如果开放式 SQL 的选 项不够全面 ，那么检索 的数据将会 按需要提炼 的顺序和结 构显示。要 想以后再提 炼这些数据 ，请在检索 过程中，将 它们按压缩 格式保存在 临时数据集 中。

创建并提炼 数据集
用户使用临 时数据集为 检索之后的 提炼工作提 供选定的数 据。ABAP/4 提供了两种 在存储中创 建数据集的 方法：内表 和摘录数据 集。用户的 选择取决于 要完成的任 务。

内表
如果希望数 据集尽可能 密切地映射 下面的的数 据结构，或 者希望直接 访问单个数 据，请使用 内表。关于 如何使用内 表提炼数据 的示例，参 见：

摘录数据集
摘录是用户 可用报表创 建的顺序数 据集。如果 需要多次将 大量数据作 为一个整体 处理，请使 用摘录。下 面的主题说 明如何创建 摘录数据集 、如何填充 数据以及最 后如何提炼 这些数据。

已提炼数据 的示例

对于许多报 表评估来说 ，处理数据 时的顺序与 数据存储的 顺序可能不 同。由于读 取操作的结 果反映数据 存储的顺序 ，所以必须 对选定的整 个数据材料 按所需的顺 序重新排序 。
在航班预订 应用环境中 提炼数据的 典型结果是 创建一个列 表，用来包 含每个航班 号的预订信 息。航班连 接将按起飞 城市排序， 航班按日期 排序，乘客 按等级和是 否吸烟排序 。对于每次 航班，都要 确定乘客总 数和行李总 重量。
结果列表中 的一段如下 所示：

用户可从下 列内容中找 到四个不同 的创建该列 表的程序：

在读取数据 过程中提炼 数据

提炼数据最 直接的方法 是使用 SELECT 语句中相应 的选项（参 见 *从数据库表提炼数据* （页 **错误！链接无效。**））。
下面的示例 程序用来从 表 SPFLI、 SFLIGHT 和 SBOOK 中提炼数据 ，如 *已提炼数据 的示例* （页 275）中说明的那 样。

```
REPORT SAPMZTST.
```

```
DATA: SUM TYPE I, CNT TYPE I.

TABLES: SPFLI, SFLIGHT, SBOOK.

SELECT * FROM SPFLI ORDER BY CITYFROM CITYTO CONNID.
  SKIP.
  WRITE: / SPFLI-CARRID,
           SPFLI-CONNID,
           'from', (15) SPFLI-CITYFROM,
           'to',   (15) SPFLI-CITYTO.
  ULINE.
  SELECT * FROM SFLIGHT WHERE CARRID = SPFLI-CARRID
                          AND CONNID = SPFLI-CONNID
                          ORDER BY FLDATE.
    SKIP.
    WRITE: / 'Date:', SFLIGHT-FLDATE.
    WRITE: 20 'Book-ID', 40 'Smoker', 50 'Class'.
    ULINE.
    SUM = 0.
    CNT = 0.
    SELECT * FROM SBOOK WHERE CARRID = SFLIGHT-CARRID
                          AND CONNID = SFLIGHT-CONNID
                          AND FLDATE = SFLIGHT-FLDATE
                          ORDER BY CLASS SMOKER BOOKID.
      WRITE: / SBOOK-BOOKID UNDER 'Book-ID',
               SBOOK-SMOKER UNDER 'Smoker',
               SBOOK-CLASS  UNDER 'Class'.
    SUM = SUM + SBOOK-LUGGWEIGHT.
    CNT = CNT + 1.
    ENDSELECT.
    ULINE.
    WRITE: 'Number of bookings:  ', (3) CNT,
         / 'Total luggage weight:', (3) SUM, SBOOK-WUNIT.
  ENDSELECT.
ENDSELECT.
```

该程序使用 了 SELECT 语句的 ORDER BY 子句，它们 已在 *指令行的顺 序*（页 **错误！链接无效。**）中说明。关 于 SELECT 语句可用于 提炼数据的 其它选项， 参见*选择并处理 指令列中数 据*（页 **错误！链接无效。**）。

用这种方法 提炼数据， 必须自己编 写数据库访 问程序。另 外，还必须 编写选择屏 幕程序，这 使用户用该 屏幕限制数 据集被读取 成为可能（ 参见 *使用选择屏幕*（页 **错误！链接无效。**））。
用逻辑数据 库读取数据 时，无需自 己编写数据 库访问程序 。系统将自 动创建选择 屏幕。但是 ，不会影响 读取数据的 顺序。下面 内容将说明 如何用临时 数据集给这 些数据记录 排序。

用内表提炼 数据

将数据保存 在内表中时 ，对于读取 的每个数据 库都需要一 个内表。该 内表包含数 据库表的部 分或所有列 。无论是使 用平铺结构 为每个数据 库表创建内 表，还是使 用嵌套结构 创建内表， 包含多少栏 都由用户决 定。对于使 用平铺结构 的表，必须 使用关键字 。在嵌套结 构的表中， 可以根据逻 辑数据库中 的层次从数 据库表保存 数据。这里 不需要关键 字。
如果使用大 量数据，则 将数据保存 在内表中并 在其中提炼 数据有以下 缺点：
由于系统一 个接一个地 处理所有内 表，所以在 几个内表中 划分大量数 据时，必须 考虑处理内 表所需的运 行时间。并 且系统以未 压缩形式存 储内表，这 意味着大量 的数据需要 大量的存储 空间。甚至 系统还必须 将数据集展 开成已分页 的区域，这 样就降低了 处理速度。
创建和处理内表（页 **错误！链接无效。**）详细说明了 如何使用内 表。
下面的主题 包含了两个 使用内表提 炼数据的示 例：

用平铺内表 提炼数据

下面的示例 说明如何使 用平铺内表 提炼数据：

```
该程序被连 接到逻辑数 据库 F1S。

REPORT SAPMZTST.

DATA: SUM TYPE I, CNT TYPE I.

TABLES: SPFLI, SFLIGHT, SBOOK.

DATA: SORT_SPFLI   LIKE SPFLI   OCCURS 100 WITH HEADER LINE,
      SORT_SFLIGHT LIKE SFLIGHT OCCURS 100 WITH HEADER LINE,
      SORT_SBOOK   LIKE SBOOK   OCCURS 100 WITH HEADER LINE.
```

```
START-OF-SELECTION.

GET SPFLI.
  APPEND SPFLI TO SORT_SPFLI.

GET SFLIGHT.
  APPEND SFLIGHT TO SORT_SFLIGHT.

GET SBOOK.
  APPEND SBOOK TO SORT_SBOOK.

END-OF-SELECTION.
  SORT: SORT_SPFLI   BY CITYFROM CITYTO CONNID,
        SORT_SFLIGHT BY FLDATE,
        SORT_SBOOK   BY CLASS SMOKER BOOKID.

  LOOP AT SORT_SPFLI.
    SKIP.
    WRITE: / SORT_SPFLI-CARRID,
             SORT_SPFLI-CONNID,
             'from', (15) SORT_SPFLI-CITYFROM,
             'to',   (15) SORT_SPFLI-CITYTO.
    ULINE.
    LOOP AT SORT_SFLIGHT WHERE CARRID = SORT_SPFLI-CARRID
                           AND CONNID = SORT_SPFLI-CONNID.
      SKIP.
      WRITE: / 'Date:', SORT_SFLIGHT-FLDATE.
      WRITE: 20 'Book-ID', 40 'Smoker', 50 'Class'.
      ULINE.
      SUM = 0.
      CNT = 0.
      LOOP AT SORT_SBOOK WHERE CARRID = SORT_SFLIGHT-CARRID
                           AND CONNID = SORT_SFLIGHT-CONNID
                           AND FLDATE = SORT_SFLIGHT-FLDATE.
        WRITE: / SORT_SBOOK-BOOKID UNDER 'Book-ID',
                 SORT_SBOOK-SMOKER UNDER 'Smoker',
                 SORT_SBOOK-CLASS  UNDER 'Class'.
                 SUM = SUM + SORT_SBOOK-LUGGWEIGHT.
                 CNT = CNT + 1.
      ENDLOOP.
      ULINE.
      WRITE: 'Number of bookings:  ', (3) CNT,
           / 'Total luggage weight:',
             (3) SUM, SORT_SBOOK-WUNIT.
    ENDLOOP.
  ENDLOOP.
```

该程序创建 *已提炼数据 的示例*（页 275）中显示的列 表。

检索数据的 GET 事件明显独 立于排序过 程。数据库 表的结构和 内容完全由 三个内表确 定。用 SORT 语句排序过 程之后（参 见*内表排序*（页 **错误！链接无效。**））， 系统将 显示表的内 容。该循环 结构与 *在读取数据 过程中提炼 数据*（页 275）中的示例中 的 SELECT 循环结构完 全对应。

使用嵌套内 表提炼数据

下面的示例 说明如何使 用嵌套内表 给数据排序 。

该程序被连 接到逻辑数 据库 F1S。

```
REPORT SAPMZTST.

DATA: SUM TYPE I, CNT TYPE I.

TABLES: SPFLI, SFLIGHT, SBOOK.

DATA: BEGIN OF SORT_SBOOK,
        BOOKID     LIKE SBOOK-BOOKID,
        SMOKER     LIKE SBOOK-SMOKER,
        CLASS      LIKE SBOOK-CLASS,
        LUGGWEIGHT LIKE SBOOK-LUGGWEIGHT,
```

```
        WUNIT      LIKE SBOOK-WUNIT,
      END OF SORT_SBOOK.

DATA: BEGIN OF SORT_SFLIGHT,
        FLDATE LIKE SFLIGHT-FLDATE,
        SBOOK  LIKE SORT_SBOOK OCCURS 100,
      END OF SORT_SFLIGHT.

DATA: BEGIN OF SORT_SPFLI OCCURS 100,
        CARRID   LIKE SPFLI-CARRID,
        CONNID   LIKE SPFLI-CONNID,
        CITYFROM LIKE SPFLI-CITYFROM,
        CITYTO   LIKE SPFLI-CITYTO,
        SFLIGHT  LIKE SORT_SFLIGHT OCCURS 100,
      END OF SORT_SPFLI.

START-OF-SELECTION.

GET SPFLI.
  REFRESH SORT_SPFLI-SFLIGHT.

GET SFLIGHT.
  REFRESH SORT_SFLIGHT-SBOOK.

GET SBOOK.
  MOVE-CORRESPONDING SBOOK TO SORT_SBOOK.
  APPEND SORT_SBOOK TO SORT_SFLIGHT-SBOOK.

GET SFLIGHT LATE.
  MOVE-CORRESPONDING SFLIGHT TO SORT_SFLIGHT.
  APPEND SORT_SFLIGHT TO SORT_SPFLI-SFLIGHT.

GET SPFLI LATE.
  MOVE-CORRESPONDING SPFLI TO SORT_SPFLI.
  APPEND SORT_SPFLI.

END-OF-SELECTION.

  SORT SORT_SPFLI BY CITYFROM CITYTO CONNID.
  LOOP AT SORT_SPFLI.
    SKIP.
    WRITE: / SORT_SPFLI-CARRID,
             SORT_SPFLI-CONNID,
             'from', (15) SORT_SPFLI-CITYFROM,
             'to',   (15) SORT_SPFLI-CITYTO.
    ULINE.
    SORT SORT_SPFLI-SFLIGHT BY FLDATE.
    LOOP AT SORT_SPFLI-SFLIGHT INTO SORT_SFLIGHT.
      SKIP.
      WRITE: / 'Date:', SORT_SFLIGHT-FLDATE.
      WRITE: 20 'Book-ID', 40 'Smoker', 50 'Class'.
      ULINE.
      SORT SORT_SFLIGHT-SBOOK BY CLASS SMOKER BOOKID.
      SUM = 0.
      CNT = 0.
      LOOP AT SORT_SFLIGHT-SBOOK INTO SORT_SBOOK.
        WRITE: / SORT_SBOOK-BOOKID UNDER 'Book-ID',
                 SORT_SBOOK-SMOKER UNDER 'Smoker',
                 SORT_SBOOK-CLASS  UNDER 'Class'.
        SUM = SUM + SORT_SBOOK-LUGGWEIGHT.
        CNT = CNT + 1.
      ENDLOOP.
      ULINE.
      WRITE: 'Number of bookings:  ', (3) CNT,
           / 'Total luggage weight:',
             (3) SUM, SORT_SBOOK-WUNIT.
    ENDLOOP.
  ENDLOOP.
```

该程序创建 *已提炼数据 的示例* (页 275) 中显示的列 表。

在 GET 事件中，系 统将数据读 取到三级表 SORT_SPFLI 中，该表包 含了子结构 SFLIGHT 和子子结构 SBOOK。 排序过程在 单个嵌套层 中进行。

该编程方法 不要求内表 之间的主关 系（无 WHERE 条件），但 它比平铺内 表复杂。并 且增加的内 部维护工作 对要求的存 储空间和运 行时间行为 将起到消极 影响。

使用摘录数 据集提炼数 据

由于内表具 有固定的行 结构，所以 它不适于处 理具有变化 结构的数据 集。出于这 种考虑，ABAP/4 提供了创建 所谓的摘录 数据集的可 能性。摘录 数据集是报 表存储区中 的顺序数据 集。这意味 着只能在循 环中访问它 的数据。不 能象对内表 操作那样通 过索引访问 其单个行。在报表运行 过程中，系 统可以正确 地创建一个 摘录数据集 。至于内表 ，因为系统 可根据需要 展开它，所 以原则上摘 录数据集的 大小是没有 限制的。
摘录数据集 由一系列预 定义的结构 的记录组成 。但是，不 是所有记录 的结构都必 须相同。在 一个摘录数 据集中，可 以一个接一 个地存储不 同长度和结 构的记录。 不必为要存 储的不同结 构创建单个 的数据集。 这实际上大 大减少了维 护工作。
与内表不同 ，系统在存 储摘录数据 集时，将部 分压缩摘录 数据集。这 减少了所需 的存储空间 。另外，不 必在报表的 开始处指定 摘录数据集 的结构，而 可以在程序 流过程中动 态决定其结 构。
下面主题解 释
关于如何使 用摘录数据 集提炼数据 的示例，参 见

创建并填充 摘录数据集

要在报表中 创建并填充 摘录数据集 ，请执行下 列三个步骤 ：

1. 将要 在摘录数据 集中使用的 记录类型定 义为字段组 。
2. 通过 分配字段定 义每个记录 类型的结构 。
3. 将所 需的数据摘 录到数据集 中。

将摘录记录 定义为字段 组

摘录数据集 由一系列记 录组成。这 些记录可以 有不同的结 构。所有具 有相同结构 的记录形成 一个记录类 型。必须使 用 FIELD-GROUPS 语句将摘录 数据集的每 个记录类型 定义为字段 组。

语法
```
FIELD-GROUPS <fg>.
```
该语句定义 了字段组 <fg>。 字段组将几 个字段组合 到一个名称 下。关于如 何定义字段 组的单个字段的信息， 参见 *给字段组分 配字段* (页 268)。出于可读 性的原因， 最好在报表 的开始处， 即声明段之 后定义字段 组。
字段组不为 字段保留存 储空间，但 它包含现有 字段的指针 。用记录填 充摘录数据 集时，这些 指针将决定 存储记录的 内容。
可以定义特 殊字段组 HEADER：

语法
```
FIELD-GROUPS HEADER.
```
填充摘录数 据集时，系 统将自动用 该字段组给 所有其它字 段组加上前 缀。这意味 着在摘录数 据集中，字 段组 <fg> 的记录总是 首先包含 HEADER 字段组的字 段。在给摘 录数据集排 序时，系统 将使用这些 字段作为缺 省的排序关 键字。

```
FIELD-GROUPS: HEADER, FLIGHT_INFO, FLIGHT_DATE.
```
该语句链创 建了三个字 段组。

给字段组分 配字段

要确定将哪 个字段包含 进字段组中 ，请使用 INSERT 语句：

语法
```
INSERT <F1> ... <fn> INTO <fg>.
```
该语句定义 字段组 <fg> 的字段。在 给字段组分 配字段以前 ，必须用 FIELD-GROUPS 语句定义字 段组 <fg>。 而对于字段 $<f_i>$， 就只能使用 全局数据对 象。不能将 在子程序或 功能模块中 定义的局部 数据对象分 配给字段组 。
INSERT 语句与 FIELD-GROUPS 语句一样， 既不保留存 储空间，也 不转换值。 您使用 INSERT 语句创建指 向字段组 <fg> 中的字段 $<f_i>$ 的指针，因 而定义了摘 录记录的结 构。要填充 摘录记录， 参见 *摘录数据集* (页 266)。
执行报表时 ，可给字段 组分配字段 ，直到首次 使用该字段 组填充摘录 记录。从这 时起，记录 的结构就已 固定，不再 改变。简而 言之，如果 还没有使用 字段组，就 可以动态地 给它分配任 意值。

由于特殊字 段组 HEADER 是每个摘录 记录的一部 分，所以在 填充了第一 个摘录记录后，就不能 再更改该字 段组。

某一字段可 出现在多个 字段组中； 但是，这将 导致摘录数 据集中有不 必要的数据 冗余。
您不必用字 段填充字段 组。如果定 义 HEADER 字段组，则 只用 HEADER 中 的字段填充 摘录记录。

```
TABLES: SPFLI, SFLIGHT.

FIELD-GROUPS: HEADER, FLIGHT_INFO, FLIGHT_DATE.

INSERT: SPFLI-CARRID SPFLI-CONNID SFLIGHT-FLDATE
        INTO HEADER,
        SPFLI-CITYFROM SPFLI-CITYTO
        INTO FLIGHT_INFO.
```

该示例创建 了三个字段 组。INSERT 语句用字段 填充了两个 字段组。

摘录数据集

要创建实际 的摘录数据 集，请使用 EXTRACT 语句：

语法
EXTRACT <fg>.
系统用报表 的第一个 EXTRACT 语句创建摘 录数据集并 添加第一个 摘录记录。 并且每使用 一个后续的 EXTRACT 语句，就向 摘录数据集 填充另一个 摘取记录。
如果已经指 定，则每个 摘录记录都 包含字段组 HEADER 中的字段， 并用它作为 排序关键字 ，其后紧跟 那些包含在 字段组 <fg> 中的字段。 在摘录过程 中，系统用 相应字段中 的当前内容 填充摘录记录。
一旦系统为 字段组 <fg> 处理了第一 条 EXTRACT 语句，摘录 数据集中的 相应摘录记 录的结构就 已固定。不 能再在字段 组 <fg> 和 HEADER 中插入新的 字段。如果 试图在后面 修改其中一 个字段组并 在另一条 EXTRACT 语句中使用 它，将会发 生运行时错 误。
通过使用不 同的字段组 处理几次 EXTRACT 语句，就可 以用不同长 度和结构的 记录填充摘 录数据集。由于在 EXTRACT 语句中首次 使用字段组 之前，可动 态地修改该 字段组，所 以摘录数据 集的优点是 不必在程序 一开始就确 定字段组的 结构。

下列程序被 连接到逻辑 数据库 F1S。

```
REPORT SAPMZTST.

TABLES: SPFLI, SFLIGHT.

FIELD-GROUPS: HEADER, FLIGHT_INFO, FLIGHT_DATE.

INSERT: SPFLI-CARRID SPFLI-CONNID SFLIGHT-FLDATE
        INTO HEADER,
        SPFLI-CITYFROM SPFLI-CITYTO
        INTO FLIGHT_INFO.

START-OF-SELECTION.

GET SPFLI.
  EXTRACT FLIGHT_INFO.

GET SFLIGHT.
  EXTRACT FLIGHT_DATE.
```

该示例创建 三个字段组 。INSERT 语句给其中 的两个字段 组分配字段 。在 GET 事件中，系 统用两个不 同的记录类 型填充摘录 数据集。字 段组 FLIGHT_INFO 的记录由五 个字段组成 ：SPFLI-CARRID、 SPFLI-CONNID、 SFLIGHT-FLDATE、 SPFLI-CITYFROM 和 SPFLI-CITYTO。 前三个字段 属于加前缀 的字段组 HEADER。 字段组 FLIGHT_DATE 的记录只由 字段组 HEADER 的三个字段 组成。下图 显示摘录数 据集的结构 ：

处理摘录数 据集

下列主题说 明如何处理 已填充的摘 录数据集。

在开始处理 报表的摘录 数据集之前 ，必须先用 所有所需的 数据填充该 数据集。在进行首次处 理操作之后 ，就不能再 摘录任何记 录到数据集 中。

如何处理摘 录数据集

读取摘录数 据集

处理内表时 ，也可以使 用循环读取 摘录数据集 的数据：

语法

```
LOOP.
  ...
  [AT FIRST | AT <fgi> [WITH <fgj>] | AT LAST.
    ...
   ENDAT.]
  ...
ENDLOOP.
```

语句 LOOP-ENDLOOP 终止创建报 表的摘录数 据集并在数 据集的所有 记录上执行 循环。在每 个循环过程 中，系统将 读取一个摘 录记录并将 其数据值直 接放回到源 字段中。可 以连续执行 多个循环。

与内表不同 ，对于摘录 数据集不必 使用特殊的 工作区作为 接口（参见 *访问内表* （页 **错误！链接无效。**））。对于每 个记录的读 取，都可以 使用它们原 始字段名在 循环的语句 块中处理读 取的数据。

与内表上的 LOOP AT-ENDLOOP 循环不同， 在摘录数据 集上不能使 用嵌套的 LOOP-ENDLOOP 循环，否则 将产生运行 时错误。

在循环的语 句块中及处 理循环后， 不允许再使 用 EXTRACT 语句。否则 将产生运行时错误。

循环控制
如果只需对 数据集的某 些记录执行 一些语句， 则可使用控 制语句 AT 和 ENDAT。
系统将针对 AT 不同的选项 处理控制语 句之间的语 句块，如下 所示 ：

AT FIRST
系统将针对 数据集的第 一条记录执 行一次该语 句块。
AT <fgi> [WITH <fgj>]
如果当前读 取的摘录记 录的记录类 型是用字段 组 <fgi> 定义的，系 统就处理该 语句块。使 用 WITH <fgj> 选项时，在 摘录数据集 中，字段组 <fgi> 当前读取的 记录后面必 须紧跟字段 组 <fgj> 的记录。
AT LAST
系统将针对 数据集的最 后一条记录 执行一次该 语句块。

关于如何使 用控制语句 AT 和 ENDAT 处理控制级 的信息，参 见 *处理控制级* （页 260）。
关于在循环 结束前如何 终止该循环 的信息，参 见 *中止循环* （页 **错误！链接无效。**）。

下面的程序 被连接到逻 辑数据库 F1S。

```
REPORT SAPMZTST.

TABLES: SPFLI, SFLIGHT.

FIELD-GROUPS: HEADER, FLIGHT_INFO, FLIGHT_DATE.

INSERT: SPFLI-CARRID SPFLI-CONNID SFLIGHT-FLDATE
        INTO HEADER,
        SPFLI-CITYFROM SPFLI-CITYTO
        INTO FLIGHT_INFO.

START-OF-SELECTION.

GET SPFLI.
  EXTRACT FLIGHT_INFO.

GET SFLIGHT.
  EXTRACT FLIGHT_DATE.
```

```
END-OF-SELECTION.

  LOOP.
    AT FIRST.
      WRITE / 'Start of LOOP'.
      ULINE.
    ENDAT.
    AT FLIGHT_INFO WITH FLIGHT_DATE.
      WRITE: / 'Info:',
               SPFLI-CARRID , SPFLI-CONNID, SFLIGHT-FLDATE,
               SPFLI-CITYFROM, SPFLI-CITYTO.
    ENDAT.
    AT FLIGHT_DATE.
      WRITE: / 'Date:',
                SPFLI-CARRID , SPFLI-CONNID, SFLIGHT-FLDATE.
    ENDAT.
    AT LAST.
      ULINE.
      WRITE / 'End of LOOP'.
    ENDAT.
  ENDLOOP.
```

创建并填充 摘录数据集 对应于 *创建并填充 摘录数据集* (页 279) 中的示例。 在事件 END-OF-SELECTION 中，数据检 索已经结束 。系统在 LOOP-ENDLOOP 循环中读取 一次数据集 。

控制语句 AT FIRST 和 AT LAST 指导系统在 列表中各写 入一行及一 带下划线的 行，一次在 循环的开始 ，一次在循 环的结束。

控制语句 AT <fg_i> 通知系统输 出与两种记 录类型中的 每一个相对 应的字段。 由于有选项 WITH FLIGHT_DATE，如果后面至 少跟了一条 字段组 FLIGHT_DATE 的记录，即 ，逻辑数据 库至少传送 一次航班的 日期，系统 就只输出字 段组 FLIGHT_INFO 的记录。

输出列表的 开始如下所 示：

没有定义 FLIGHT_INFO 记录类型的 HEADER 区中的 SFLIGHT-FLDATE 字段的内容 ，这是由于 逻辑数据库 在每个层次 结构的最后 都用空值填 充该层中的 所有字段。该特征对排 序和处理摘 录数据集的 控制级很重 要。

给摘录数据 集排序

和处理内表 一样，也可 以使用 SORT 语句给摘录 数据集排序 ：

语法

```
SORT [<order>][AS TEXT]
     [BY <F1> [<order>][AS TEXT] ... <fn> [<order>][AS TEXT]].
```

SORT 语句终止创 建报表的摘 录数据集， 同时给数据 集记录排序 。如果没有 BY 选项，系统 将根据在 HEADER 字段组中指 定的关键字 给数据集排 序。
要定义不同 的排序关键 字，请使用 BY 选项。然后 系统将根据 指定组件 <**F1**>...<f_n> 给数据集排 序。这些组 件必须是 HEADER 字段组的字 段，或者是 只包含 HEADER 字段组中的 字段的字段 组。关键字 段的数量限 制为 50。如果 指定了多个 字段，那么 系统将首先 根据 <**F1**> 给记录排序 ，然后根据 <**F2**> 排序，等等 。系统将在 BY 之前指定的 选项作为 BY 之后指定的 所有字段的 缺省值。对 于单个字段 ，在这些字 段之后指定 的选项将覆 盖在 BY 之前指定的 选项。
可以在 <order> 选项中使用 DESCENDING 或 ASCENDING 指定排序顺 序。缺省值 是升序排列 的。对于文本字段，请 使用 AS TEXT 选项指定排 序方法。该 选项与给内 表排序一样 ，将按字母 顺序排序 (参见 *内表排序* (页 **Error! Not a valid link.**))。如果需 要多次按字 母顺序给摘 录数据集排 序，则由于 性能方面的 原因，必须 在排序关键 字中包含按 字母排序的 字段来替代 文本字段。要填充该字 段，请使用 CONVERT 语句 (参见*转换为可排 序格式* (页 **错误！链接无效。**))。
如果用 AS TEXT 给 BY 加上前缀， 则该选项只 适用于排序 关键字中类 型为 C 的字段。如 果将 AS TEXT 置 于一个字段 之后，那么 该字段的类 型必须为 C。如果将 AS TEXT 置 于一个字段 组之后，则 该选项只适 用于该组中 类型为 C 的字段。
该排序过程 是不稳定的 ，也就是说 ，带相同排 序关键字的 记录的顺序 是毫无必要 保存。
如果可利用 的主存储空 间对排序数 据不够用， 那么在排序 过程中系统 会将数据写 到一个外部 帮助文件中 。该帮助文 件由 SAP 参数文件参 数 DIR_SORTTMP 确定。

SORT 语句将摘录 过程中未定 义的排序关 键字中的所 有字段按空 值对待。系 统不管指定 的排序顺序 如何，总是 将空值排在 具有定义值 的字段的前 面。

在处理 SORT 语句以后， 不能再执行 EXTRACT 语句。否则 将产生运行 时错误。

在程序中， 可使用不同 的关键字随 时给摘录数 据集排序。 唯一的前提 是作为排序 依据的所有 字段在摘录 过程中要包 含在 HEADER 中。不能在 LOOP-ENDLOOP 循环中使用 SORT 语句。但是 ，可以按照 任何顺序一 个接一个地 排序并读取 摘录数据集 。
每一个使用 SORT 语句、在摘 录数据集上 执行的排序 过程都定义 一个控制级 结构，该过 程对于后续控制级处理 是必需的（ 参见 *处理控制级* (页 260) ）。

下面的程序 被连接到逻 辑数据库 F1S。

```
REPORT SAPMZTST.

TABLES: SPFLI, SFLIGHT.

FIELD-GROUPS: HEADER, FLIGHT_INFO, FLIGHT_DATE.

INSERT: SPFLI-CARRID SPFLI-CONNID SFLIGHT-FLDATE
        INTO HEADER,
        SPFLI-CITYFROM SPFLI-CITYTO
        INTO FLIGHT_INFO.

START-OF-SELECTION.

GET SPFLI.
  EXTRACT FLIGHT_INFO.

GET SFLIGHT.
  EXTRACT FLIGHT_DATE.

END-OF-SELECTION.

  SORT DESCENDING.

  LOOP.
    AT FIRST.
      WRITE / 'Start of LOOP'.
      ULINE.
    ENDAT.
    AT FLIGHT_INFO WITH FLIGHT_DATE.
      WRITE: / 'Info:',
               SPFLI-CARRID , SPFLI-CONNID, SFLIGHT-FLDATE,
               SPFLI-CITYFROM, SPFLI-CITYTO.
    ENDAT.
    AT FLIGHT_DATE.
      WRITE: / 'Date:',
                SPFLI-CARRID , SPFLI-CONNID, SFLIGHT-FLDATE.
    ENDAT.
    AT LAST.
      ULINE.
      WRITE / 'End of LOOP'.
    ENDAT.
  ENDLOOP.
```

除了 SORT DESCENDING 语句以外， 该示例与 *读取摘录数 据集* (页 281) 中的示例相同。SORT 语句通知系 统在使用 LOOP-ENDLOOP 循环读取摘 录数据集之 前，根据 HEADER 字段组的三 个字段按降 序给摘录数 据集排序。 该列表的最 后部分如下 所示：

请注意，在 排序过程中 ，系统将 SFLIGHT-FLDATE 字段中包含 未定义字符 的记录放在 其它记录的 前面。这样 做将保护逻 辑数据库的 数据层次， 而不考虑排 序顺序。

处理控制级

通过使用 SORT 语句给摘录 数据集排序 ，可定义一 个控制级结 构。摘录数 据集的控制 级结构对应 于 HEADER 字段组中的 字段的顺序 。排序后， 可在 LOOP-ENDLOOP 循环中使用 AT 语句编写语 句块，使系统只在控制 中断时（即 ，控制级更 改时）才处 理它。

语法

```
AT NEW <f> | AT END OF <f>.
  ...
ENDAT.
```

如果字段〈f〉或当前摘录 记录中的排 序关键字的 较高层字段 中包含其它 值，并且该 值与摘录数 据集前面的 记录（对于 AT NEW）或 后续记录（ 对于 AT END）中 的值不同， 将产生控制 中断，并且 系统将处理 AT-ENDAT 中的语句块 。字段〈f〉必须是 HEADER 字段组的一 部分。

如果摘录数 据集没有排 序，则系统 将忽略 AT-ENDAT 中的语句块 。

在决定控制 中断时，系 统将忽略未 在摘录过程 中定义的〈f〉的字段内容 （空值）。

系统将一个 接一个地处 理循环中的 所有 AT...ENDAT 处理块。所 以要当心它 们的顺序。 在控制级逻辑内，请保 持排序顺序 。该顺序不 一定是 HEADER 字段组中的 字段的顺序 ，但可以是 在 SORT 语句中确定 的顺序。

```
REPORT SAPMZTST.

DATA: T1(4), T2 TYPE I.

FIELD-GROUPS: HEADER, TEST.

INSERT T2 T1 INTO HEADER.

T1 ='AABB'. T2 = 1. EXTRACT TEST.
T1 ='BBCC'. T2 = 2. EXTRACT TEST.
T1 ='AAAA'. T2 = 2. EXTRACT TEST.
T1 ='AABB'. T2 = 1. EXTRACT TEST.
T1 ='BBBB'. T2 = 2. EXTRACT TEST.
T1 ='BBCC'. T2 = 2. EXTRACT TEST.
T1 ='AAAA'. T2 = 1. EXTRACT TEST.
T1 ='BBBB'. T2 = 1. EXTRACT TEST.
T1 ='AAAA'. T2 = 3. EXTRACT TEST.
T1 ='AABB'. T2 = 1. EXTRACT TEST.

SORT BY T1 T2.

LOOP.

  AT FIRST.
    WRITE 'Start of LOOP'.
    ULINE.
  ENDAT.

  AT NEW T1.
    WRITE / '   New T1:'.
  ENDAT.

  AT NEW T2.
    WRITE / '   New T2:'.
  ENDAT.

  WRITE: /14 T1, T2.

  AT END OF T2.
    WRITE / 'End of T2'.
  ENDAT.

  AT END OF T1.
    WRITE / 'End of T1'.
  ENDAT.

  AT LAST.
    ULINE.
  ENDAT.

ENDLOOP.
```

该程序创建 了一个只包 含 HEADER 字段组中字 段的样本摘 录。在排序 过程之后， 摘录数据集 有几个对于 控制级 T1 和 T2 的控制中断 ，这些在下 面的图中说 明：

在 LOOP-ENDLOOP 循环中，系 统输出数据 集的内容， 并将遇到下 列控制中断 ：

计算序号和 合计

在使用 LOOP-ENDLOOP 读取已排序 的摘录数据 集时，可以 访问两个自 动生成的字 段 CNT(<f>) 和 SUM(<g>)， 这些字段提 供了不同值 的序号或数 字字段的合 计。系统将 在控制级的 最后（在控 制中断之前 ，参见 *处理控制级* （页 Error! Not a valid link.）） 以及读取 数据集的最 后一个记录 后填充这些 字段。如下 所示：

CNT(<f>)
如果 <f> 是 HEADER 字段组的非 数字字段， 并且系统是 根据 <f> 给摘录数据 集排序的， 则 CNT(<f>) 包含了在控 制级中或在 整个数据集 中分别设定 的不同值 <f> 的序号。

SUM(<g>)
如果 <g> 是摘录数据 集的数字字 段，SUM (<g>) 将包含控制 级或整个数 据集中 <g> 的值的合计 。

可以在控制 中断产生之 前，在 AT END OF 后的过程块 中访问这些 字段（参见 *处理控制级* （页 283））；或在读 取整个数据 集后，在 AT LAST 后的过程块 中访问这些 字段（参见 *读取摘录数 据集* （页 281））。

只能在给数 据集排序后 才能访问 CNT(<f>) 和 SUM(<g>) 字段。否则 会产生运行 时错误。

```
REPORT SAPMZTST.

DATA: T1(4), T2 TYPE I.

FIELD-GROUPS: HEADER, TEST.

INSERT T2 T1 INTO HEADER.

T1 ='AABB'. T2 = 1. EXTRACT TEST.
T1 ='BBCC'. T2 = 2. EXTRACT TEST.
T1 ='AAAA'. T2 = 2. EXTRACT TEST.
T1 ='AABB'. T2 = 1. EXTRACT TEST.
T1 ='BBBB'. T2 = 2. EXTRACT TEST.
T1 ='BBCC'. T2 = 2. EXTRACT TEST.
T1 ='AAAA'. T2 = 1. EXTRACT TEST.
T1 ='BBBB'. T2 = 1. EXTRACT TEST.
T1 ='AAAA'. T2 = 3. EXTRACT TEST.
T1 ='AABB'. T2 = 1. EXTRACT TEST.

SORT BY T1 T2.

LOOP.

  WRITE: /20 T1, T2.

  AT END OF T2.
    ULINE.
    WRITE: 'Sum:', 20 SUM(T2).
    ULINE.
  ENDAT.

  AT END OF T1.
    WRITE: 'Different values:', (6) CNT(T1).
  ULINE.
  ENDAT.

  AT LAST.
    ULINE.
    WRITE: 'Sum:', 20 SUM(T2),
           / 'Different values:', (6) CNT(T1).
  ENDAT.

ENDLOOP.
```

该程序创建 只包含 HEADER 字段组中字 段的样本摘 录。排序后 ，系统将在 每个控制级 的最后和循 环的最后输 出数据集的 内容，不同 的 T1 字段的序号 以及 T2 字段的合计 ：

使用摘录数 据集提炼数 据示例

下面的示例 提炼表 SPFLI、 SFLIGHT 和 SBOOK 中的数据， 象 *已提炼数据 的示例* (页 275) 中说明的那 样。

下面的报表 被连接到逻 辑数据库 F1S。

```
REPORT SAPMZTST.

TABLES: SPFLI, SFLIGHT, SBOOK.

FIELD-GROUPS: HEADER, FLIGHT_INFO, FLIGHT_BOOKING.

INSERT:
  SPFLI-CITYFROM SPFLI-CITYTO
  SPFLI-CONNID SFLIGHT-FLDATE
  SBOOK-CLASS SBOOK-SMOKER SBOOK-BOOKID INTO HEADER,
  SPFLI-CARRID                          INTO FLIGHT_INFO,
  SBOOK-LUGGWEIGHT SBOOK-WUNIT          INTO FLIGHT_BOOKING.

START-OF-SELECTION.

GET SPFLI.
  EXTRACT FLIGHT_INFO.

GET SFLIGHT.

GET SBOOK.
  EXTRACT FLIGHT_BOOKING.

END-OF-SELECTION.

  SORT.

  LOOP.
    AT FLIGHT_INFO.
      SKIP.
      WRITE: / SPFLI-CARRID,
               SPFLI-CONNID,
               'from', (15) SPFLI-CITYFROM,
               'to',   (15) SPFLI-CITYTO.
      ULINE.
    ENDAT.
    AT NEW SFLIGHT-FLDATE.
        SKIP.
        WRITE: / 'Date:', SFLIGHT-FLDATE.
        WRITE: 20 'Book-ID', 40 'Smoker', 50 'Class'.
        ULINE.
    ENDAT.
    AT FLIGHT_BOOKING.
        WRITE: / SBOOK-BOOKID UNDER 'Book-ID',
                 SBOOK-SMOKER UNDER 'Smoker',
                 SBOOK-CLASS  UNDER 'Class'.
    ENDAT.
    AT END OF SFLIGHT-FLDATE.
      ULINE.
      WRITE: 'Number of bookings:  ', (3) CNT(SBOOK-BOOKID),
           / 'Total luggage weight:',
             SUM(SBOOK-LUGGWEIGHT), SBOOK-WUNIT.
    ENDAT.
  ENDLOOP.
```

该系统创建 三个字段组 ，并用几个 字段填充。 在 GET 事件中，由 于有 EXTRACT 语句，所以 将填充摘录 数据集。请 注意，对于 GET SFLIGHT 事件，没有 EXTRACT 语句，这是 由于所需的 字段 SFLIGHT_FLDATE 是 HEADER 字段组的一 部分，所以 将自动地为 每个子事件 GET SBOOK 摘录这些数 据。

在检索数据 之后，由于 使用了 SORT 语句，系统 将终止创建 数据集，并 且按 HEADER 排序关键字 给数据集排 序。在 LOOP-ENDLOOP 循环中，通 过执行几个 AT-ENDAT 块并使用字 段 CNT(...) 和 SUM(...)， 系统将排序 后的摘录数 据集写入输 出列表。

概览

内容

列表是 ABAP/4 报表程序数 据的输出媒 介。每个 ABAP/4 报表程序将 其输出数据 传递到直接 与该程序连 接的列表中 。每个程序 最多生成 21 个列表： 1 个基本列表 和 20 个辅助列表 。本节对创 建列表作一 般说明。也 就是说，此 处描述的大 部分语句适 用于基本列 表和辅助列 表。缺省情 况下，系统 将报表的输 出传递到基 本列表。大 多数情况下 ，报表仅有 基本列表。 因此，本节 中示例主要 处理基本列 表。关于如 何编程辅助 列表的详细 信息，参见 *交互式列表* （页 **Error! Not a valid link.**）。
从 ABAP/4 程序之中， 既可以将列 表输出到屏 幕也可以输 出到 SAP 假脱机系统 。缺省情况 下，在屏幕 上显示列表 。本节中的 所有示例都 使用该缺省 值。关于如 何打印列表 的详细信息 ，参见 *打印列表* （页 **Error! Not a valid link.**）。
将数据写入 列表的基本 ABAP/4 语句是 WRITE 语句。其他 输出语句是 ULINE 和 SKIP。 关于这三条 语句的详细 信息，参见 *将数据输出到屏幕* （页 **错误！链接无效。**）。
下列主题说 明列表的结 构和创建列 表时定义列 表格式的选 项：

标准列表

如果 ABAP/4 报表只使用 WRITE、 SKIP 和 ULINE 输出语句并 且不包含本 节后面所说 明的编辑语 句，则系统 将输出传递 到标准列表 。完成数据 选择之后在 该屏幕上显 示列表。
下列主题说 明

标准列表示 例

下面的输出 屏幕显示标 准列表：

要创建该标 准列表，请 使用下面的 样本报表。

```
REPORT SAPMZTST.

TABLES SPFLI.

SKIP.
ULINE AT /(62).

SELECT * FROM SPFLI WHERE CONNID GE 0017
                      AND CONNID LE 0400.
    WRITE: / SY-VLINE, SPFLI-CONNID, SY-VLINE,
           (15) SPFLI-CITYFROM, 26 SY-VLINE,
```

```
            31 SPFLI-CITYTO, 51 SY-VLINE, 62 SY-VLINE,
            / SY-VLINE, 8 SY-VLINE,
            SPFLI-DEPTIME UNDER SPFLI-CITYFROM, 26 SY-VLINE,
            SPFLI-ARRTIME UNDER SPFLI-CITYTO,   51 SY-VLINE,
            SPFLI-FLTIME, SY-VLINE.
    ULINE AT /(62).
  ENDSELECT.

WRITE: /10 'SAP *** SAP *** SAP *** SAP *** SAP *** SAP',
        /19(43) 'Flight Information System',
        /19(43) 'International Connections'.
```

SELECT 语句从数据 库表格 SPFLI 读取选定行 。在 SELECT 循环内， WRITE、 SKIP 和 ULINE 语句将工作 区 SPFLI 的字段以及 水平和垂直 行输出到该 列表。关于 所使用的所 有 WRITE 选项的详细 信息，参见 *将数据输出到屏幕* (页 **错误！链接无效。**) 。 *创建和更改 列表及列表 头* (页 **错误！链接无效。**) 对如何创建 列表和列标 题进行解释 。

关于该标准 列表结构的 详细信息， 参见 *标准列表结 构* (页 282) 。

标准列表结 构

下列主题提 供标准列表 结构的信息 。
标准列表包 括
有关标准列 表宽度的信 息，可在下 列主题中找 到：

标准页眉

标准页眉至 少包括两行 标准标题。 标准标题的 首行包含列 表表头和页 号。第二行 由水平线组 成。执行程 序时，列表 表头存储在 系统字段 SY-TITLE 中。如果需 要，可以在 标准标题中 包括最多四 行列头和另 一条水平线 。

创建和更改 列表及列表 头 (页 **错误！链接无效。**) 说明如何维 护列表和列 头。而且， 显示列表之 后，可以在 标准列表的 用户界面中 调整这些表 头（参见 *修改列表和 列标题* (页 280) ）。
标准页眉的 宽度自动调 整到窗口宽 度。
如果用户垂 直滚动列表 ，则标准页 眉保持可见 ，只滚动表 头之下的列 表。
如果用户水 平滚动列表 ，则列表表 头和页号保 持可见。

标准页

输出数据显 示在页眉之 下。标准列 表包含一个 动态长度的 单页（内部 限制： 60,000 行）。输出 长度由当前 列表大小确 定。

输出屏幕包 括一个垂直 滚动条，允 许用户滚动 页长超过窗 口的列表。

标准列表的 宽度

标准列表的 宽度取决于 执行报表时 的窗口宽度 。如果用户 窗口小于或 等于标准窗 口大小，则 标准页宽应 符合标准窗 口宽度。用 户可能必须 滚动列表才 能查看列表 的所有部件 。如果用户 窗口超过标 准窗口宽度 ，则标准列 表的宽度符 合选定标准 窗口的宽度 。总之，标 准列表的宽 度至少与标 准窗口一样 宽。标准窗 口的宽度取 决于操作系 统。
输出屏幕包 括允许用户 滚动超过窗 口宽度部分 的水平滚动 条。

标准列表的 用户界面

标准列表的 输出屏幕包 含 R/3 系统的标准 菜单栏和标 准工具栏。

要滚动标准 列表，系统 提供滚动条 和功能“首 页”、“上 一页”、“ 下一页”和 “末页”。 要查找列表中某一模式 ，用户可以 选择“编辑 -> 查找...” 。
用户可以使 用下列特定 列表功能：

打印输出列 表

要打印屏幕 上显示的列 表，用户可 以选择“列 表 -> 打印”。
打印的标准 页眉与显示 的标准页眉 不同，它还 包含当前日 期：

打印 *标准列表示 例* (页 288) 中创建的标 准列表，其 结果为：

```
12.01.1996              Example for Standard List                       1
------------------------------------------------------------------------------
   ID    Departure from       Arrival at           Time of
         Departure Time       Arrival Time         Flight
------------------------------------------------------------------------------

+------+----------------+----------------------+----------+
| 0017 | NEW YORK       |    SAN FRANCISCO     |          |
|      | 13:30:00       |    16:31:00          | 06:01:00 |
+------+----------------+----------------------+----------+
| 0064 | SAN FRANCISCO  |    NEW YORK          |          |
|      | 09:00:00       |    17:21:00          | 05:21:00 |
+------+----------------+----------------------+----------+
| 0400 | FRANKFURT      |    NEW YORK          |          |
|      | 10:10:00       |    11:34:00          | 08:24:00 |
+------+----------------+----------------------+----------+
| 0026 | FRANKFURT      |    NEW YORK          |          |
|      | 08:30:00       |    09:50:00          | 08:20:00 |
+------+----------------+----------------------+----------+
         SAP *** SAP *** SAP *** SAP *** SAP *** SAP
                 Flight Information System
                 International Connections
```

只有由于测 试原因需要 屏幕列表的 硬副本时， 才能使用本 打印方法。 关于如何打 印列表的详 细信息，参 见 *打印列表* (页 **Error! Not a valid link.**) 。

保存列表

要保存显示 的列表，用 户可以选择 “列表 -> 保存”。出 现下列主题 ：

将列表保存 在 SAPoffice 中

当选择“列 表 -> 保存 -> Office” 时，出现一 个对话框， 询问用户是 将显示的列 表存储在用 户的 Office 文件夹中还 是发送给另 一用户。

将列表保存 在报告树中

当选择“列 表 -> 保存 -> 报告树”时 ，出现一个 对话框，询 问用户是否 将显示的列 表保存在报 表树的适当 分支中。

将列表作为 本地文件保 存在演示服 务器中

选择“列表 -> 保存 -> 文件”时， 出现一个对 话框，询问 用户是否将 显示的列表 作为本地文 件保存，并 提供几个格 式选项。

格式选项为 ：

不转换: 系统将文件 存储为文本 文件。
表格统计 : 系统在列之 间插入 tab 键。
RTF 格式: 系统存储格 式化为文本 处理的数据 。

如果用户以 RTF 格式存储 *标准列表示 例* (页 288) 中创建的标 准列表并且 使用能读取 该格式的文 本处理程序 （如 MS WORD） 重新显示， 则列表显示 如下：

修改列表和 列标题

通常,可以 列表和列标 题创建为文 本元素(参 见 *创建和更改 列表及列表 头* (页 **错误!链接无效。**))。然而， 也可以在屏 幕上显示列 表时修改这 些表头。为 此，请选择 “系统 -> 列表 -> 列表头”。 页眉行现在 接受输入：

例如，使用 该功能将列 标题放置在 显示列表的 列之上。
保存更改。 系统将修改 后的列标题 另存为当前 登录语言文 本池中程序 的文本元素 。关于文本 元素的详细 信息，参见 *处理文本摘 要* (页 **错误！链接无效。**) 。

自定义列表

可以修改标 准列表的结 构并创建单 个结构的列 表。使用 REPORT 语句选项以 及事件 TOP-OF-PAGE 和 END-OF-PAGE。 PROGRAM 语句与 REPORT 语句等价并 有相同的选 项。
可以作下列 修改：
如果列表包 括几页，则 可以分别定 义每页的结 构。关于如 何进行这一 操作的详细 信息，参见 *多页列表* (页 280) 。

单个页眉

要分别定义 页眉的格式 ，必须在紧 随事件关键 字 TOP-OF-PAGE 的处理块中 定义：

语法
```
TOP-OF-PAGE.
  WRITE: ....
```
TOP-OF-PAGE 事件在系统 开始处理列 表新页时出 现。系统在 输出新页首 行之前处理 TOP-OF-PAGE 后面的语句 。关于事件 和处理块的 详细信息，参见 *通过事件控制 ABAP/4 程序流* (页 **错误!链接无效。**)。

> 如果以后要 启动实际列 表处理，记 住要通过使 用合适的事 件关键字， 例如 START-OF-SELECTION， 结束处理块 (参见 *定义处理块* (页 **错误！链接无效。**)) 。

自定义页眉 出现在标准 页眉之下。 如果要取消 标准页眉， 请使用 REPORT 语句的 NO STANDARD PAGE HEADING 选项：

语法
```
REPORT <rep> NO STANDARD PAGE HEADING.
```
使用该语句 时，系统不 在报表 <rep> 列表页上显 示标准页眉 。如果使用 TOP-OF-PAGE 定义单个页 眉，则系统 就将其显示 出来。
垂直滚动时 ，自定义页 眉与标准页 眉一样保持 可见。但是 ，自定义页 眉包括正常 列表行，因 此不能自动 符合窗口宽 度。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING.

TOP-OF-PAGE.

WRITE: SY-TITLE, 40 'Page', SY-PAGNO.
ULINE.
WRITE: / 'SAP AG', 29 'Walldorf, ',SY-DATUM,
       / 'Neurottstr. 16', / '69190 Walldorf/Baden'.
ULINE.

START-OF-SELECTION.

DO 5 TIMES.
  WRITE / SY-INDEX.
ENDDO.
```

报表程序不 使用标准页 眉，而是使 用 TOP-OF-PAGE 之后的自定 义页眉。要 显式结束 TOP-OF-PAGE 处理块，必 须指定事件 关键字 START-OF-SELECTION 。输出如下 所示：

自定义页眉 包括六行。 程序标题来 自 SY-TITLE 系统字段， 页号来自 SY-PAGNO。 自定义页眉 与列表宽度 不等。

确定列表宽 度

要确定输出 列表的宽度 ，请使用 REPORT 语句的 LINE-SIZE 选项。.

语法
REPORT <rep> LINE-SIZE <width>.
该语句根据 <width> 字符确定报 表 <rep> 输出列表的 宽度。如果 将 <width> 设置为 0，则系统 使用标准列 表的宽度 (参见 *标准列表的 宽度* (页 289))。
一行最多可 包含 255 个字符。但 是，如果要 打印列表， 请记住大多 数打印机不 能打印超过 132 个字符的列 表。如果要 在创建时直 接打印列表 ，则页宽必 须符合某种 现有打印格 式。否则， 系统就不能 打印该列表 （参见*打印参数* (页 **Error! Not a valid link.**))。 一定不要选 择超过 132 字符的列表 宽度，除非 创建只用于 显示的列表 。
创建列表时 ，系统字段 SY-LINSZ 包含当前行 宽。要将列 表宽度调整 为当前窗口 宽度，参见 *多页列表* (页 280) 。
使用 ULINE 语句（不带 AT 选项）创建 的水平线自 动适配自定 义列表宽度 。

```
REPORT SAPMZTST LINE-SIZE 40.
WRITE: 'SY-LINSZ:', SY-LINSZ.
ULINE.
DO 20 TIMES.
  WRITE SY-INDEX.
ENDDO.
```

程序创建下 列输出：

示例使用标 准页眉。如 果将 LINE-SIZE 值从 **40** 替换为 **60** ，则输出如 下所示：

标准页眉和 下划线自动 适配列表宽 度。

确定页长

要确定输出 列表的页长 ，请使用 REPORT 语句的 LINE-COUNT 选项。

语法
REPORT <rep> LINE-COUNT <length>[(<n>)].
该语句用 <length> 行确定报表 <rep> 输出列表的 页长。如果 指定选项号 <n>,则 系统为页脚 保留 <n> 行页长。并 不将这些页 脚行填写到 END-OF-PAGE 事件中，而 是作为空行 显示（参见 *定义页脚* (页 238))。
如果将 <length> 设置为 0 ，则系统使 用标准页长 （参见 *标准页* (页 289))。要根据 当前窗口大 小调整页长 ，参见 *多页列表* (页 280) 。创建列表 时，系统字 段 SY-LINCT 包含每页的 当前行数(即 <n> 行或 0 代表标准页 长度)。

记住页眉长 度是 <length> 的部件。因 此，对于列 表本身，只 能使用 <length> 减去页眉长 度减去 <n> 行。如果 <length> 小于页长度 ，则产生运 行错误。

如果在列表 处理期间系 统到达为实 际列表提供 的区域的结 尾，则输出 页脚，如果 有，插入一 些空格并且 启动新页。 插入的空格 属于列表背 景并且不是 列表行。SY-PAGNO 系统字段总 包含当前页 号。

确定页长时 ，应记住下 列各项：

对于屏幕 输出，请使 用标准页长 以避免在屏 幕中间分页 。

对于打印 列表，则根 据打印机要 求设置页长 。应采用对 任何页长都 能产生合理 输出的方式 书写程序。 如果选择现 有打印格式 之外的页长 ，则不能在 创建时直接 打印。关于 打印格式的 详细信息， 参见 *打印列表* (页 **Error! Not a valid link.**) 。

对于仅指 定页面设置 的表格形式 的列表使用 固定长度规 格。在为此 类列表编写 程序代码之 前，请检查 能否使用预 定义的 SAP*script* 表格。关于 表格的详细 信息，参见 文档*样式和页面设置维护*（页 Error! Not a valid link.）。

下列程序旨 在说明 LINE-COUNT 选项的用法 。因此在一 个屏上显示 不同的列表 页。

```
REPORT SAPMZTST LINE-SIZE 40 LINE-COUNT 4.

WRITE:  'SY-LINCT:', SY-LINCT.
SKIP.

DO 6 TIMES.
 WRITE / SY-INDEX.
ENDDO.
```

该程序将页 长确定为四 行。使用标 准页眉。假 定标准页眉 包括两行列 表表头，则 输出如下所 示：

列表包括四 页，每页四 行。每页由 页眉和两行 实际列表组 成。应注意 每页结尾的 空格。

定义页脚

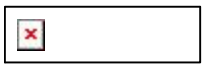
要定义页脚 ，请使用 END-OF-PAGE 事件。在处 理列表页时 ，如果系统 到达页脚保 留行，或者 如果 RESERVE 语句触发分 页则该事件 发生（参见 *条件分页 - 定义行块*（页 242））。在紧接 着 END-OF-PAGE 事件关键字 的处理块中 填充页脚的 行：

语法

```
END-OF-PAGE.
  WRITE: ....
```

如果为 REPORT 语句的 LINE-COUNT 选项中的脚 注保留行， 则系统只处 理紧接着 END-OF-PAGE 的处理块（ 参见 *确定页长*（页 292））。

如果要以后 开始处理实 际列表，记 住要通过使 用合适的事 件关键字， 例如 START-OF-SELECTION， 结束紧接着 END-OF-PAGE 的处理块（ 参见 *定义处理块*（页 **错误！链接无效。**））。



```
REPORT SAPMZTST LINE-SIZE 40 LINE-COUNT 6(2)
                NO STANDARD PAGE HEADING.

TOP-OF-PAGE.

WRITE: 'Page with Header and Footer'.
ULINE AT /(27).

END-OF-PAGE.

ULINE.
WRITE: /30 'Page', SY-PAGNO.

START-OF-SELECTION.

DO 6 TIMES.
  WRITE / SY-INDEX.
ENDDO.
```

该程序包括 三个处理块 。关闭标准 页眉。将页 长设置为六 行，其中两 行留作页脚 。

列表包括三 页,每页六 行。每页由 自定义的两 行页眉、两 行实际列表 和两行页脚 组成。页脚 中显示的当 前页号来自 系统字段 SY-PAGNO 。

多页列表

如果在报表 中写入列表 输出页的行 数超过在 REPORT 语句 LINE-COUNT 选项中定义 的行数，则 系统自动创 建新页(参 见 *确定页长* (页 292))。每个新 页都包含为 报表定义的 页眉和页脚（如果有的 话)。除自动分页 外，可以使 用 NEW-PAGE 和 RESERVE 语句显式地 编码分页。 NEW-PAGE 语句的选项 允许分别设 置每页格式 。也需要 NEW-PAGE 语句在程序 内打印列表 （参见 *打印列表* (页 Error! Not a valid link.))。
下列主题说 明

编程分页

要编程无条 件的分页， 请使用 NEW-PAGE 语句。
要编程取决 于页左边空 行数的分页 ，请使用 RESERVE 语句。

无条件分页

要在页处理 期间触发分 页，请使用 NEW-PAGE 语句的基本 格式：

语法
```
NEW-PAGE.
```
该语句
结束当前 页。在新页 上显示所有 其他输出。
如果输出 写入到当前 页和 NEW-PAGE 之后的新页 ，则只开始 新页。系统 然后将 SY-PAGNO 系统字段家 1。不能产 生空页。
不能触发 END-OF-PAGE 事件。这意 味着即使定 义了页脚， 系统也不输 出。

```
REPORT SAPMZTST LINE-SIZE 40.

TOP-OF-PAGE.

WRITE: 'TOP-OF-PAGE', SY-PAGNO.
ULINE AT /(17).

START-OF-SELECTION.

DO 2 TIMES.
  WRITE / 'Loop:'.
  DO 3 TIMES.
    WRITE / SY-INDEX.
  ENDDO.
  NEW-PAGE.
ENDDO.
```

该样本程序 使用列表表 头“标准页 眉”定义为 文本元素的 标准页眉和 自定义页眉 。每页都出 现两个页眉 。

DO 循环两次遇 到 NEW-PAGE 语句，但是 只执行一次 分页。在第 二条 NEW-PAGE 语句之后， 没有输出。

条件分页 - 定义行块

要在少于某 一页剩余行 数的条件下 执行分页， 请使用 RESERVE 语句：

语法
```
RESERVE <n> LINES.
```

如果当前列 表页的最后 输出和页脚 之间剩余的 自由行数少 于 <n>，则 该语句触发 分页。 <n> 可以是变量 。在开始新 页之前，系 统处理 END-OF-PAGE 事件。只有 输出写入到 后续页时， RESERVE 才生效。不 创建空页。
RESERVE 语句就这样 定义必须整 个输出的行 块。要查找 行块可能有 的附加实际 效果，参见 *将输出定位 在行块的首 行* (页 250) 。

```
REPORT SAPMZTST LINE-SIZE 40 LINE-COUNT 8(2).

END-OF-PAGE.

ULINE.

START-OF-SELECTION.

DO 4 TIMES.
  WRITE / SY-INDEX.
ENDDO.

DO 2 TIMES.
  WRITE / SY-INDEX.
ENDDO.

RESERVE 3 LINES.
WRITE: / 'LINE 1',
       / 'LINE 2',
       / 'LINE 3'.
```

该样本程序 标准页眉的 列表头定义 为“标准页 眉”。REPORT 语句将页长 确定为八行 ，两行用于 标准页眉， 另外两行为 页脚保留。 页脚包括水 平行和空行 。因此，输 出的实际列 表，每页留 下四行。第 一个 DO 循环填充这 四行，然后 发生 END-OF-PAGE 事件，其后 系统自动开 始新页。在 第二个 DO 循环之后， 因为页上剩 余的自由行 数少于三， 因此 RESERVE 语句触发 END-OF-PAGE 事件和分页 。输出如下 ：

第 3 页的三行组 成行块。

单页的标准 页眉

标准页眉包 括列表和列 标题（参见*标准页眉* (页 289) .)。要影 响标准页眉 这些组件的 表现形式， 请使用 NEW-PAGE 语句的下列 选项：

语法
NEW-PAGE [NO-TITLE|WITH-TITLE] [NO-HEADING|WITH-HEADING].
使用 NO-TITLE 或 WITH-TITLE 选项取消或 者在以后所 有页上显示 标准表头。 基本列表的 缺省值为 WITH-TITLE， 辅助列表为 NO-TITLE。
使用 NO-HEADING 或 WITH-HEADING 选项取消或 在以后所有 页上显示列 标题。基本 列表的缺省 值为 WITH-HEADING， 辅助列表为 NO-HEADING。
关于基本和 辅助列表的 详细信息， 参见 *交互式列表* (页 **Error! Not a valid link.**) 。

即使使用 REPORT 语句的 NO STANDARD PAGE HEADING 选项取消标 准页眉，也 能使用 WITH-TITLE 和 WITH-HEADING 激活单个组 件的显示。

因为 TOP-OF-PAGE 事件在新页 上处理，所 以 NEW-PAGE 语句不能影 响在该事件 中定义的页 眉显示（参见 *单个页眉* (页 291))。

```
REPORT SAPMZTST LINE-SIZE 40.
WRITE: 'Page', SY-PAGNO.

NEW-PAGE NO-TITLE.
WRITE: 'Page', SY-PAGNO.

NEW-PAGE NO-HEADING.
WRITE: 'Page', SY-PAGNO.

NEW-PAGE WITH-TITLE.
WRITE: 'Page', SY-PAGNO.
```

```
NEW-PAGE WITH-HEADING.
WRITE: 'Page', SY-PAGNO.
```

该程序创建 五页，分别 带有不同页 眉。将列表 表头标题文 本元素定义 为“标准页眉”，列标 题定义为“ 列”。

第 1 和 5 页包含完整 的标准页眉 。第 2 页没有列表 表头。在第 3 页上，取消 整个页眉。 在第 4 页上，省略 列标题。

单页页长

要分别确定 每页的页长 ，请使用 NEW-PAGE 语句：

语法
```
NEW-PAGE LINE-COUNT <length>.
```
该语句将后 续页的页长 确定为 <length>。 <length> 可以是变量 。如果将 <length> 设置为 0 ，则系统使 用标准页长 （参见 *标准页* (页 289))。页眉是 页的部分因 此也是页长 的部分。

不能使用 NEW-PAGE 创建或更改 页脚。将 REPORT 语句中定义 的页脚（参 见 *确定页长* (页 292) ）保存为与 NEW-PAGE 语句无关。

对于实际列 表输出，<length> 减去页眉长 度即可。

使用 NEW-PAGE 语句的 LINE-COUNT 选项时，请 参阅 *确定页长* (页 292) 中的注意。

要将页长度 适配当前窗 口长度，请 将 <length> 设置为 SY-SROWS。 SY-SROWS 系统字段包 含当前窗口的行数。

```
REPORT SAPMZTST LINE-SIZE 40 LINE-COUNT 0(1).

END-OF-PAGE.

ULINE.

START-OF-SELECTION.

NEW-PAGE LINE-COUNT 5.
DO 4 TIMES.
  WRITE / SY-INDEX.
ENDDO.

WRITE: / 'Next Loop:'.

NEW-PAGE LINE-COUNT 6.
DO 6 TIMES.
  WRITE / SY-INDEX.
ENDDO.
```

程序创建 5 页，长度各 不相同。将 列表表头文 本元素定义 为“标准页 眉”。REPORT 语句为页脚 每页保留一 行。在 END-OF-PAGE 事件中将页 脚定义为水 平行。第一 条 NEW-PAGE 语句将页长 设置为 5，第二条 设置为 6。

因为以前没 有输出写入 列表，因此 第一条 NEW-PAGE 语句不开始 新页。标准 页眉每页使 用两行作为 列表表头。 页脚使用一 行。对于第 一个 DO 循环，每页 两行可用于 WRITE 输出。DO 循环内的所 有分页在列 表处理到达 页眉时自动 发生。第二 条 NEW-PAGE 从第 3 页到第 4 页创建分页 。这里不处 理 END-OF-PAGE 事件。对于 第二个 DO 循环，每页 三行可用于 WRITE 输出。自动 再次分页。 出现页脚。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING
                LINE-SIZE 40 LINE-COUNT 0(2).

TOP-OF-PAGE.

WRITE: 'Top of Page', SY-PAGNO,
       'SY-SROWS:', SY-SROWS.
ULINE.

END-OF-PAGE.

ULINE.
WRITE: 'End of Page', SY-PAGNO.

START-OF-SELECTION.

* NEW-PAGE LINE-COUNT SY-SROWS.

DO 100 TIMES.
  WRITE / SY-INDEX.
ENDDO.
```

因为 NEW-PAGE 被标记为注 释，所以该 程序创建没 有结尾的单 页：

系统在当前 窗口中显示 尽可能多的 行，即 12 行。在上面 的数字中， 12 行包含两行 自定义表头 行和 10 行实际列表 。在垂直滚 动时，页眉 保持可见。 如果删除 NEW-PAGE 语句前面的 星号并且保 持当前窗口 长度，则输 出如下：

根据 SY-SROWS ，列表分为 几页，每页 12 行。12 行中两行留 作页眉，两 行留作页脚 。在该列表 中，用户可 以使用“下 一页”显示 地滚动（例 如，到第 11页）：

列表级的页 宽

不能更改列 表级内单个 页宽。只能 更改新列表 级的所有页 宽。为此， 请使用 NEW-PAGE 语句：

语法
NEW-PAGE LINE-SIZE <width>.
从新页开始 的所有列表 级宽度都为 <width>，而不是 REPORT 语句中定义 的宽度。如 果将 <width> 设置为 0 ，则系统使 用标准列表 的宽度（参 见 *标准列表的 宽度* (页 289) ）。
如果将 <width> 设置为 SY-SCOLS， 则可以使新 列表级的宽 度适配窗口 宽度，即使 该窗口小于 标准窗口。 SY-SCOLS 系统字段包 含当前窗口 一行的字符 数。

在列表级之 内，即下页 不是新列表 级的开始， 则系统忽略 LINE-SIZE 选项。

关于如何创 建新列表级 的详细信息 ，参见 *交互式列表* (页 **Error! Not a valid link.**) 。

在程序之内 滚动

在程序之内 ，可以垂直 和水平滚动 列表。使用 SCROLL 关键字。例 如，如果要 滚动到某页 以响应用户 输入，则从 程序内滚动 就很有意义 。
SCROLL 语句只对完 成的列表生 效。如果在 列表的第一 条输出语句 之前使用该 语句，则不 影响该列表 。如果在列 表的第一条 输出语句之 后使用 SCROLL， 则影响整个 列表，包括 以后的所有 输出语句。在每条 SCROLL 语句之后， 可以查询 SY-SUBRC 以查看系统 是否成功。 如果系统成 功滚动则 SY-SUBRC 为 0 ；如果滚动 不可能则为 4 ，因为其超 过列表边界 。如果正在 使用几个列 表级，则 SY-SUBRC 也可能是 8，表明指 定的列表级 不存在（参 见 *滚 换ナ 列表* (页 **Error! Not a valid link.**))。
SCROLL 语句允许

垂直滚动

水平滚动

通过窗口滚 动窗口

要以当前窗 口大小垂直 滚动列表并 且与页长无 关，请使用 这条语句：

语法
```
SCROLL LIST FORWARD|BACKWARD [INDEX <idx>].
```
不带 INDEX 选项时，该 语句以当前 窗口大小向 前或向后滚 动当前列表 。使用 INDEX 选项时，系 统以列表级 <idx> 滚动列表。 关于在列表 级中滚动的 详细信息， 参见 *滚动交互式列表* (页 Error! Not a valid link.) 。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING LINE-SIZE 40.

TOP-OF-PAGE.

WRITE: 'Top of Page', SY-PAGNO, 'SY-SROWS:', SY-SROWS.
ULINE.

START-OF-SELECTION.

DO 100 TIMES.
  WRITE / SY-INDEX.
ENDDO.

DO 3 TIMES.
  SCROLL LIST FORWARD.
ENDDO.
```

该报表创建 没有结尾页 的列表。在 DO 循环之内， 系统三次 SCROLL 语句。如果 当前窗口为 12 行（存储在 SY-SROWS 中），则程 序输出如下 ：

应注意，实 际列表是以 SY-SROWS 减去表头行 数进行滚动 。用户可以 继续朝两个 方向滚动。

按页滚动

要按页滚动 列表，即按 页长垂直滚 动列表，SCROLL 语句提供下 列选项：

滚动到某页
要滚动到某 页，请使用 SCROLL 语句的 TO 选项：

语法
```
SCROLL LIST TO FIRST PAGE | LAST PAGE | PAGE <pag>
              [INDEX <idx>] [LINE <lin>].
```
不用 INDEX 选项，语句 将当前列表 滚动到首页 、尾页或 <pag> 页。使用 INDEX 选项，则系 统滚动列表 级 <idx> 的列表。关 于列表级的 详细信息， 参见 *交互式列表* (页 Error! Not a valid link.) 。使用 LINE 选项时，系 统显示从实 际列表的 <lin> 行开始滚动 的页。不包 括页眉行。

按页数滚动
要按页数滚 动列表，请 使用 SCROLL 语句的下列 选项：

语法
```
SCROLL LIST FORWARD | BACKWARD <n> PAGES [INDEX <idx>].
```
不用 INDEX 选项时，语 句向前或向 后滚动 <n> 页。如上所 述，INDEX 选项参考某 一列表级。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING
                LINE-SIZE 40 LINE-COUNT 8(2).

DATA: PAG TYPE I VALUE 15,
      LIN TYPE I VALUE 4.

TOP-OF-PAGE.

WRITE: 'Top of Page', SY-PAGNO.
ULINE.

END-OF-PAGE.

ULINE.
WRITE: 'End of Page', SY-PAGNO.
```

```
START-OF-SELECTION.

DO 100 TIMES.
  DO 4 TIMES.
    WRITE / SY-INDEX.
  ENDDO.
ENDDO.

SCROLL LIST TO PAGE PAG LINE LIN.
```

该程序创建 100 页的列表， 每页 8 行。每页四 行用于页眉 和页脚。因 为 SCROLL 语句，该程 序的输出如 下：

列表从第 15 页开始显示 。由于 LINE 选项，实际 列表的前三 行滚动到页 眉以下。

滚动到列表 的页边缘

要水平滚到 列表的左或 右页边缘， 请使用 SCROLL 语句的下列 选项：

语法
SCROLL LIST LEFT | RIGHT [INDEX <idx>].
不用 INDEX 选项时，语 句滚动到当 前列表的左 或右页边缘 。使用 INDEX 选项时，系 统滚动列表 级 <idx> 的列表。关 于列表级的 详细信息， 参见 *交互式列表* (页 **Error! Not a valid link.**) 。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING LINE-SIZE 200.

TOP-OF-PAGE.

WRITE: AT 161 'Top of Page', SY-PAGNO,
                'SY-SCOLS:', SY-SCOLS.
ULINE.

START-OF-SELECTION.

DO 200 TIMES.
  WRITE SY-INDEX.
ENDDO.

SCROLL LIST RIGHT.
```

该程序宽度 为 200 的一页列表 。如果当前 窗口宽度(存储在 SY-SCOLS 中)等于 40，则程序输 出如下：

列表显示滚 动到右页边 边缘。用户 现在可以使 用滚动条滚 动到左边。

按列滚动

要按列水平 滚动列表， SCROLL 语句提供几 个选项。在 此情况下， 一列意味着 列表行一个 字符。

滚动到某列
要滚动到某 列，请使用 SCROLL 语句的 TO COLUMN 选项：

语法
SCROLL LIST TO COLUMN <col> [INDEX <idx>].
不用 INDEX 选项时，系 统显示从列 <col> 开始的当前 列。使用 INDEX 选项时，系 统滚动列表 级 <idx> 的列表。关 于列表级的 详细信息， 参见 *交互式列表* (页 **Error! Not a valid link.**) 。

按列数滚动
要按某一列 数滚动列表 ，请使用 SCROLL 语句的下列 选项：

语法
SCROLL LIST LEFT | RIGHT BY <n> PLACES [INDEX <idx>].
不用 INDEX 选项时，系 统按 <n> 列将当前列 表滚动到左 边或右边。 如上所述， INDEX 选项参考特 定的列表级 。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING LINE-SIZE 200.

TOP-OF-PAGE.

WRITE: AT 161 'Top of Page', SY-PAGNO,
                'SY-SCOLS:', SY-SCOLS.
ULINE.

START-OF-SELECTION.

DO 200 TIMES.
  WRITE SY-INDEX.
ENDDO.

SCROLL LIST TO COLUMN 178.
```

该程序创建 宽度为 200 的一页列表 。如果当前 窗口宽度（ 存储在 SY-SCOLS 中）等于 40 ，则程序输 出如下：

从 178 列开始显示 列表。用户 可以滚动到 列表左边。

列表页面设 置

列表页面设 置确定如何 组织列表显 示以便容易 读取。页面 信息量大小 并不重要， 重要的是信 息的表现方 式。人眼更 易于处理小 块信息。因 此，将包含 新信息块的 列或行与前 面的块从视 觉上分开也 同等重要。 设置列表页 面时，应该 使用几个空 行或垂直行 以将各栏分 开。在输出 包含新信息 项的行之前 ，应画一空 行或下划线 。
下列主题说 明 ABAP/4 为设置列表 页面提供多 种功能。

定位输出

可以在当前 页的任意位 置定位 WRITE 和 ULINE 语句的输出 。紧接着位 置规格的 WRITE、 SKIP、 或 ULINE 语句可以覆 盖现有输出 。对于当前 输出位置， 参阅系统字 段
SY-COLNO （ 用于当前列 ）
SY-LINNO （ 用于当前行 ）
可以使用系 统字段在页 上引导。
ABAP/4 提供一些关 键字以更改 绝对和相对 输出位置。 参见下列主 题：

SAP 允许只读取 系统字段 SY-COLNO 和 SY-LINNO。 因此，要定 位输出，**只** 能使用这些 主题中说明 的语句。**不 要**通过直接 给系统字段 赋值来定位 输出。在那 种情况下，SAP 不能保证系 统字段的内 容，因为这 样赋值不触 发合理性检 查。即使现 在，也可将 列号赋给页 外的 SY-COLNO， 但这样做毫 无意义。

绝对定位

指定绝对位 置之后，将 后续输出写 入在固定行 和列开始的 屏幕。

水平定位

要指定水平 输出位置， ABAP/4 提供两种方 法：
WRITE 和 ULINE 语句的 AT 选项(参见 *在屏幕上 位 WRITE 输出*（页 **错误！链接无效。**））以及 POSITION 语句。POSITION 语句的语法 为：

语句
POSITION <col>.
该语句将水 平输出位置 和 SY-COLNO 系统字段设 置为 <col>。 如果 <col> 超出页面之 外，则忽略 后续输出语 句。

系统将 POSITION 语句或使用 AT 格式化的 WRITE 语句的输出 写入指定位 置，不管是 否有足够空 间。超出该 行的输出部 分被截断。 其他 WRITE 输出在下一 行开始。

垂直定位

如下指定垂 直输出位置 ：

语法
SKIP TO LINE <n>.
该语句将垂 直输出位置 和 SY-LINNO 系统字段设 置为 <lin> 。如果 <lin> 超出 1 和页长范围 ，则系统忽 略该语句。

> 使用 LINE 时，系统也 计算页眉和 页脚行。请 确认没有无 意改写页眉 和页脚行。

在页眉之下 定位输出

要将输出定 位到整个页 眉之后的第 一行，请使 用 BACK 语句：

语法
BACK.
如果该语句 不是紧跟 RESERVE 语句，则后 续输出出现 在页眉之下 。系统将 SY-COLNO 设置为 1 并根据页眉 的长度设置 SY-LINNO。 与 RESERVE 语句组合时 ，其他规则 也适用（参 见 *将输出定位 在行块的首 行* (页 250) ）。
如果在 TOP-OF-PAGE 事件中指定 BACK，则系统不将 输出位置设 置到整个页 眉之下，而 只在标准页 眉之下。写 入的任何输 出现在都覆 盖 TOP-OF-PAGE 中指定的自 定义页眉。

绝对定位示 例

```
REPORT SAPMZTST NO STANDARD PAGE HEADING LINE-SIZE 60.

DATA: X(3), Y(3).

X = SY-COLNO. Y = SY-LINNO.

TOP-OF-PAGE.

WRITE: 'Position of Header: ', X, Y.
ULINE.

START-OF-SELECTION.

SKIP TO LINE 10.
POSITION 20.
X = SY-COLNO. Y = SY-LINNO.
WRITE: '* <- Position', X, Y.

SKIP TO LINE 12.
ULINE AT 20(20).

BACK.
X = SY-COLNO. Y = SY-LINNO.
WRITE: 'Position after BACK:', X, Y.
```

该程序创建 下列列表页 ：

> 系统将 SY-COLNO 和 SY-LINNO 的初值分配 给字段 X 和 Y。应注意 ，该赋值实 际发生在 START-OF-SELECTION 事件中（参 见*定义处理块* (页 **错误！链接无效。**) ）。初始 输出位置就 是第一各表 头行的位置 。输出写到 此处。SKIP TO LINE 和 POSITION 在列 20、行 10 中放一个“ *”。SKIP TO LINE 和 AT 产生下横线 。最后，BACK 将输出位置 设置到两行 页眉下的列 1、行 3。

相对定位

相对定位参 考以前写入 列表的输出 。某些相对 定位自动发 生。使用无 定位的 WRITE 时，在上一 输出之后空 一列出现输 出。如果在 当前行没有 足够空间， 则换行。不 使用定位的 ULINE 和 SKIP 语句产生换 行。

要编程相对 定位，请将 SY-COLNO 和 SY-LINNO 系统字段与 *绝对定位* (页 300) 中说明的语 句组合使用或者使用下 述相对定位 语句。
产生换行 (页 241)
将输出定位 到其它输出 之下 (页 249)
将输出定位 在行块的首 行 (页 250)
相对定位的 示例 (页 249)

产生换行

要产生换行 ，请在 WRITE、 ULINE 或 NEW-LINE 语句的 AT 选项中使用 反斜杠。

语法
```
NEW-LINE.
```
该语句将输 出定位在新 行中，将 SY-COLNO 设置为 1 并且将 SY-LINNO 加 1。如果自 最后换行之 后将输出写 入屏幕，则 系统只执行 该语句。NEW-LINE 不创建空行 。要创建空 行，请使用 SKIP 语句（参见 *创建空行* (页 183))。
在 NEW-PAGE 语句和事件 开始时发生 自动换行。

将输出定位 到其它输出 之下

可以将 WRITE 输出定位到 上一 WRITE 输出的列中 。请使用 WRITE 语句的格式 化选项 UNDER：

语法
```
WRITE <f> UNDER <g>.
```
系统从开始 输出字段 <g> 的同一列中 开始输出 <f>。该 语句不限于 当前页，即 <g> 不必出现在 相同页上。

> 务必调整垂 直位置以免 改写上一输 出。

参考字段 <g> 必须与相应 WRITE 语句中一样 ，包括所有 说明，如偏 移量等（参 见 *用指夕 量写入值* (页 **错误！链接无效。**))。如果 <g> 是文本符号 （参见 *文本符号* (页 **错误！链接无效。**))，则系统 从该文本符 号的号码确 定参考字段 。
有关示例， 参见 *相对定位的 示例* (页 249) 。

将输出定位 在行块的首 行

要将下一输 出行定位到 通过 RESERVE 语句定义的 行块的首行 （参见 *条件分页 - 定义行块* (页 294))， 请按如下格 式使用 BACK 语句：

语法
```
RESERVE.
  .....
BACK.
```
如果 BACK 紧随 RESERVE， 则后续输出 出现在 RESERVE 之后的第一 行中。例如 ，可以使用 该语句在某 循环之内的 输出之后跳 转回某行。
有关事例， 参见 *相对定位的 示例* (页 249) 。

相对定位的 示例

第一个示例 显示如何通 过自定义页 眉创建纵向 列表。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING
                LINE-SIZE 80 LINE-COUNT 7.

DATA: H1(10) VALUE '    Number',
      H2(10) VALUE '    Square',
      H3(10) VALUE '      Cube',
      N1 TYPE I, N2 TYPE I, N3 TYPE I,
      X TYPE I.

TOP-OF-PAGE.

       X = SY-COLNO + 8.  POSITION X. WRITE H1.
       X = SY-COLNO + 8.  POSITION X. WRITE H2.
       X = SY-COLNO + 8.  POSITION X. WRITE H3.
```

```
        X = SY-COLNO + 16. POSITION X. WRITE SY-PAGNO.
ULINE.

START-OF-SELECTION.

DO 10 TIMES.
  N1 = SY-INDEX. N2 = SY-INDEX ** 2. N3 = SY-INDEX ** 3.
  NEW-LINE.
  WRITE: N1 UNDER H1,
         N2 UNDER H2,
         N3 UNDER H3.
ENDDO.
```

本程序创建 两页列表。 在自定义页 眉中，使用 SY-COLNO 系统字段和 POSITION 语句相对定 位列标题。 通过 WRITE 语句的 UNDER 选项将实际 列表输出定 位在页眉行 字段之下。 输出如下：

ABAP/4 缺省的写字 符串的左对 齐和数字字 段的右对齐 导致单个字 段的不同输 出位置。要 影响对齐方 式，请使用 WRITE 语句的格式 化选项 LEFT-JUSTIFIED、RIGHT-JUSTIFIED 和 CENTERED（ 参见 *格式化选项*（页 **错误！链接无效。**））。

第二个示例 显示紧随 RESERVE 之后的 BACK 语句的作用 。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING LINE-SIZE 40.

DATA X TYPE I.

WRITE 'Some numbers:' NO-GAP.
X = SY-COLNO.
ULINE AT /(X).

RESERVE 5 LINES.

DO 5 TIMES.
  WRITE / SY-INDEX.
ENDDO.

X = SY-COLNO.

BACK.
WRITE AT X '   <- Start of Loop'.
```

本程序创建 如下输出：

输出前两行 之后，RESERVE 语句用于将 后续五行定 义为块。紧 随 BACK 之后的输出 写入到块的 第一行。请 注意，如何 使用 SY-COLNO 系统字段给 第一行加下 划线以及如何定位最后 的 WRITE 输出。

格式输出

要格式化列 表输出，ABAP/4 提供几个格 式化选项。
在 *格式化选项*（页 **错误！链接无效。**）中对 WRITE 语句的格式 化选项加以 说明。
其他重要的 格式化选项 ，例如，确 定输出颜色 以及使列表 字段接受输 入的选项， 都是 FORMAT 语句的格式 化选项。这 些将在后面 的主题中加 以说明。FORMAT 语句的全部 选项都可作 为 WRITE 语句的格式 化选项。

FORMAT 语句

要在程序中 静态设置格 式化选项， 请按如下格 式使用 FORMAT 语句：

语法
FORMAT <option$_1$> [ON|OFF] <option$_2$> [ON|OFF] ....
FORMAT 语句中设置 的格式化选 项 <option$_i$> 适用于所有 后续输出， 直到使用 OFF 选项关闭。 打开格式化 选项的 ON 选项可选， 也就是说， 可以忽略。
要在运行时 动态设置格 式化选项， 请按如下格 式使用 FORMAT 语句：

语法
FORMAT <option₁> = <var₁> <option₂> = <var₂> ....
系统将变量 <var₁> 解释为数字 。因此，它 们应该是数 据类型 I 。如果 <var₁> 的内容是零 ，则该变量 与 OFF 选项等效。 如果 <var₁> 不等于零， 则变量或者 与 ON 选项等效， 或者在与 COLOR 选项同时使 用时与相应 颜色号等效 （参见 *列表中的颜 色* (页 196) ）。
如果对于紧 随 FORMAT 语句之后的 WRITE 语句使用相 同的格式化 选项，则 WRITE 语句的设置 将覆盖当前 输出的 FORMAT 语句的相应 选项。

> 对于每个新 事件，系统 将所有格式 化选项复位 到缺省值。 关于事件列 表，参见 *事件及其事 件关键字* (页 **错误！链接无效。**) 。所有格式 化选项都有 缺省值 OFF，INTENSIFIED 选项除外（ 参见 *列表中的颜 色* (页 196) ）。

要一次性将 所有格式化 选项设置为 OFF，请 使用：

语法
FORMAT RESET.
下列主题说 明可用的格 式化选项。

列表中的颜 色

FORMAT 语句的选项 COLOR、 INTENSIFIED 和 INVERSE 影响输出列 表的颜色。
要在程序中 设置颜色， 请使用：

语法
FORMAT COLOR <n> [ON] INTENSIFIED [ON|OFF] INVERSE [ON|OFF].
要在运行时 设置颜色， 请使用：

语法
FORMAT COLOR = <c> INTENSIFIED = <int> INVERSE = <inv>.
这些格式化 选项不适用 于由 ULINE 创建的水平 行。其功能 如下所示：
COLOR 设置行背景 的颜色。而 且，如果设 置 INVERSE ON ，则系统将 更改前景颜 色而不是背景颜色。
对于 <n>，可 以设置颜色 号或者颜色 规范。但是 ，要代替颜 色号 0，必须使 用 OFF。如 果在运行时 设置颜色号 码，则小于 0 或大于 7 的所有 <c> 值都将导致 未定义的结 果。下表汇 总各种不同 的选项：

	<n>	<c>	颜 色	用于
OFF	或 COL_BACKGROUND	0	取 决于 GUI	背景
1	或 COL_HEADING	1	灰 蓝	标题
2	或 COL_NORMAL	2	淡 灰	列表正文
3	或 COL_TOTAL	3	黄	总计
4	或 COL_KEY	4	蓝 绿	关键字列
5	或 COL_POSITIVE	5	绿	正门限值
6	或 COL_NEGATIVE	6	红	负门限值
7	或 COL_GROUP	7	紫	组级别

缺省 值为 COLOR OFF。
INTENSIFIED 确定行背景 的颜色调色 板。
上面指定的 行背景的颜 色调色板可 以很浓或正 常，有一个 例外 (COLOR OFF)。 缺省设置为 INTENSIFIED ON。对于 COLOR OFF，系 统将更改前 景颜色而不 是背景颜色 。而且，如 果设置 INVERSE ON，则 INTENSIFIED OFF 无效（同样 ，COLOR OFF 例外）。
INVERSE 影响前景颜 色。
系统从背景 颜色调色板 获取指定 COLOR 并将其用作 前景颜色， 有一个例外 （COLOR OFF）。 背景颜色保 持不变。对 于 COLOR OFF,INVERSE 无效,因为 这将前景和 背景颜色设 置为相同颜 色。

> 下列语句等 效：
> **FORMAT INTENSIFIED ON.** 和 **SUMMARY.**
> **FORMAT INTENSIFIED OFF.** 和 **DETAIL.**
> 为了便于阅 读，SAP 建议一直使 用 FORMAT 语句。

下列示例显 示列表中可 能的颜色和 如何使用它 们。
关于列表中 颜色的其他 演示，请在 任何系统中 调用 SHOWCOLO 报表。

演示列表中 可用的颜色

下例显示各 种颜色格式 化选项的不 同组合情况 ：

```
REPORT  SAPMZTST.

DATA I TYPE I VALUE 0.
DATA COL(15).

WHILE I < 8.

  CASE I.
    WHEN 0. COL = 'COL_BACKGROUND '.
    WHEN 1. COL = 'COL_HEADING    '.
    WHEN 2. COL = 'COL_NORMAL     '.
    WHEN 3. COL = 'COL_TOTAL      '.
    WHEN 4. COL = 'COL_KEY        '.
    WHEN 5. COL = 'COL_POSITIVE   '.
    WHEN 6. COL = 'COL_NEGATIVE   '.
    WHEN 7. COL = 'COL_GROUP      '.
  ENDCASE.

  FORMAT INTENSIFIED COLOR = I.
  WRITE: /(4) I, AT 7            SY-VLINE,
            COL,                 SY-VLINE,
            COL INTENSIFIED OFF, SY-VLINE,
            COL INVERSE.

  I = I + 1.

ENDWHILE.
```

在 FORMAT 语句中，在 运行时设置 后续 WRITE 语句的 COLOR 选项。在程 序中单独设置每条 WRTIE 语句的其他 选项。

输出显示如 下表格：

标准页眉创 建为文本元 素。在联机 帮助中，由 于技术原因 ，该列表的 颜色与 R/3 系统的颜色 稍有不同。

在列表中使 用颜色的示 例

本例显示如 何在列表中 使用颜色以 强调输出。

下列报表可 以连接到逻 辑数据库 F1S。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING LINE-SIZE 70.

TABLES: SPFLI, SFLIGHT.
DATA SUM TYPE I.

TOP-OF-PAGE.

WRITE 'List of Flights' COLOR COL_HEADING.
ULINE.

GET SPFLI.

FORMAT COLOR COL_HEADING.
WRITE: 'CARRID', 10 'CONNID', 20 'FROM', 40 'TO'.

FORMAT COLOR COL_KEY.
WRITE: / SPFLI-CARRID   UNDER 'CARRID',
         SPFLI-CONNID   UNDER 'CONNID',
         SPFLI-CITYFROM UNDER 'FROM',
         SPFLI-CITYTO   UNDER 'TO'.
ULINE.
```

```
FORMAT COLOR COL_HEADING.
WRITE: 'Date', 20 'Seats Occupied', 50 'Seats Available'.
ULINE.

SUM = 0.

GET SFLIGHT.

IF SFLIGHT-SEATSOCC LE 10.
  FORMAT COLOR COL_NEGATIVE.
ELSE.
  FORMAT COLOR COL_NORMAL.
ENDIF.

WRITE: SFLIGHT-FLDATE   UNDER 'Date',
       SFLIGHT-SEATSOCC UNDER 'Seats Occupied',
       SFLIGHT-SEATSMAX UNDER 'Seats Available'.

SUM = SUM + SFLIGHT-SEATSOCC.

GET SPFLI LATE.

ULINE.
WRITE: 'Total Bookings:' INTENSIFIED OFF,
       SUM UNDER SFLIGHT-SEATSOCC COLOR COL_TOTAL.
ULINE
SKIP.
```

该报表创建 下列输出列 表：

所有标题以 背景颜色 COL_HEADING 显示。表格 SPFLI 的关键字字 段使用 COL_KEY 作为背景颜 色。事件 GET SFLIGHT 的列表正文 行背景颜色 (COL_NORMAL)不 同于列表背景 (COL_BACKGROUND) 。而且，预 定数量在某 一最小数量 以下的航班 ，其背景为 红色。每个 航班的总预 定数量背景 为黄色。

请注意，对 于每个新事 件，系统都 将格式化选 项复位为缺 省设置（COLOR OFF、INTENSIFIED ON）。因 此，在上面 程序中，GET LATE 事件时输出 “总计预定 ”的行背景 还是 COL_BACKGROUND。 将 INTENSIFIED 设置为 OFF 以获取与其 他输出相同 的前景颜色 。

在联机帮助 中，由于技 术原因，该 列表的颜色 与 R/3 系统的颜色 稍有不同。

使字段可接 受输入

可以使列表 中的输出字 段作好接受 输入的准备 。用户可以 在屏幕上更 改这些字段 ，然后打印 这些更改， 或者使用交 互列表处理 中的 READ LINE 语句处理这 些更改（参 见 *交互式列表* (页 Error! Not a valid link.))。如果使 变量内容在 输出列表中 接受输入， 则用户输入 的更改不影 响变量自身 。要在程序中 将输出字段 设置为“准 备输入”， 请按如下格 式使用 FORMAT 语句：

语法
```
FORMAT INPUT [ON|OFF].
```
要在运行时 将输出字段 设置为“准 备输入”， 请使用：

语法
```
FORMAT INPUT = <i>.
```
使用 ON 选项（或者 <i> 不等于 0 ）将后续输 出格式化为 输入字段。 输入字段背 景和前景颜 色与列表其 余字段不同 。对于输入 字段，选项 COLOR、 INVERSE 和 HOTSPOT 无效。INTENSIFIED 选项更改输入字段的前 景颜色。
通过将水平 行格式化为 输入字段， 可以使其接 受输入。但 是，使用 SKIP 创建的空行 不能接受输 入。

```
REPORT SAPMZTST.

WRITE   'Please fill in your name before printing:'.

WRITE / '   Enter name here    ' INPUT ON.
ULINE.

WRITE 'You can overwrite the following line:'.
```

```
FORMAT INPUT ON INTENSIFIED OFF.
ULINE.
FORMAT INPUT OFF INTENSIFIED ON.
```

在本程序中 ，WRITE 语句直接接 受 INPUT ON 格式以及使 用 FORMAT 语句格式化 的 ULINE 行。将标题 定义为文本 元素。输出 显示如下：

由于 INTENSIFIED OFF，第 二个输入字 段的前景颜 色不同于第 一个。用户 可以在屏幕 上输入这些 输入字段， 例如：

将字段输出 为热点

热点是输出 列表的特殊 区域。如果 用户单击热 点字段，则 触发事件（ 例如，AT LINE-SELECTION） 。对于未定 义为热点的 字段，要触 发某一事件 ，必须双击 鼠标或按某 一功能键。 关于列表处 理期间的事 件的详细信 息，参见 *交互式列表* （页 **Error! Not a valid link.**）。
要将某一区 域输出为热 点，请使用 FORMAT 语句的下列 选项：

语法
```
FORMAT HOTSPOT [ON|OFF].
```
要在运行时 将字段设置 为热点，请 使用：
```
FORMAT HOTSPOT = <h>.
```
ON 选项（或 <h> 不等于 0 ）将后续输 出格式化为 热点。如果 用户将鼠标 移动到该字 段，则鼠标 指针变为带 有指向手指 的手形。只 要该手形可 见，单击即 可触发事件 。除更改鼠 标指针外， 也可能需要 使用不同颜 色强调热点 。
如果设置 INOPUT ON，则不 能使用 HOTSPOT 选项，因为 使用 HOTSPOT ON 之后，无法 将光标定位 在输入字段 上。而且不 能将使用 ULINE 创建的水平 行和使用 SKIP 创建的空行 格式化为热 点。

```
REPORT SAPMZTST.
INCLUDE <LIST>.

START-OF-SELECTION.

WRITE 'Now comes a'.

FORMAT HOTSPOT ON COLOR 5 INVERSE ON.
WRITE 'HOTSPOT'.
FORMAT HOTSPOT OFF COLOR OFF.

AT LINE-SELECTION.

WRITE / 'New list AT-LINE-SELECTION'.
SKIP.
WRITE 'This is also a hotspot:'.

WRITE ICON_LIST AS ICON HOTSPOT.
```

在本程序中 ，首行的 START-OF-SELECTION 事件部分格 式化为热点 。标准页眉 定义为文本 元素。如果 用户将鼠标 移动到输出 的 HOTSPOT 单词上，则 鼠标指针更 改为手形：

单击触发事 件 AT-LINE-SELECTION。 在该事件中 ，程序创建 包含其他热 点的辅助列表。辅助列 表中的热点 为图标：

关于 AT-LINE-SELECTION 事件和辅助 列表的详细 信息，参见 *交互式列表* （页 **Error! Not a valid link.**）。

特殊输出格 式

关于 WRITE 语句所有格 式化选项的 摘要，参见 *格式化选项* (页 **错误！链接无效。**)。 当前主题说 明一些特殊 格式化选项 。这些选项 按照在特殊 数据库表格 中的某条目 格式化输出 。通常，当 定制应用程 序时，客户 维护这些表 格（关于定 制的详细信 息，参见文 档 *定制* (页 Error! Not a valid link.)。
可以使用下 列输出格式 ：

国家和用户 特有的输出 格式

。使用下列 语句可以在 程序中更改 这些设置：

语法
SET COUNTRY <c>.
对于 <c>，可 设置为表格 T005X 中定义的国 家关键字或 设置为 SPACE。
如果 <c> 不为 SPACE， 则系统关闭 用户主记录 的设置并在 表格 T005X 中搜索国家 关键字。如 果关键字存 在，则系统 将 SY-SUBRC 设置为 0 并根据 T005X 中定义的设 置格式化所 有后续 WRITE 语句的输出 。如果指定 的国家关键 字不存在， 则系统将 SY-SUBRC 设置为 4 并将所有后 续 WRITE 语句的小数 字符格式化 为句号“。 ”，将日期 规格格式化 为 MM/DD/YY。
如果 <c> 为 SPACE， 则系统不读 取表格 T005X， 而使用用户 主记录中的 设置。在这 种情况下，SY-SUBRC 总为 0 。
维护表格 T005X 是定制工作 的一部分。 但是，可以 使用“系统 -> 服务 -> 表格维护” 显示或更改 条目。

```
REPORT SAPMZTST LINE-SIZE 40.

DATA: NUM TYPE P DECIMALS 3 VALUE '123456.789'.

ULINE.
WRITE: / 'INITIAL:'.
WRITE: / NUM, SY-DATUM.
ULINE.

SET COUNTRY 'US'.
WRITE: / 'US,      SY-SUBRC:', SY-SUBRC.
WRITE: / NUM, SY-DATUM.
ULINE.

SET COUNTRY 'GB'.
WRITE: / 'GB,      SY-SUBRC:', SY-SUBRC.
WRITE: / NUM, SY-DATUM.
ULINE.

SET COUNTRY 'DE'.
WRITE: / 'DE,      SY-SUBRC:', SY-SUBRC.
WRITE: / NUM, SY-DATUM.
ULINE.

SET COUNTRY 'XYZ'.
WRITE: / 'XYZ,     SY-SUBRC:', SY-SUBRC.
WRITE: / NUM, SY-DATUM.
ULINE.

SET COUNTRY SPACE.
WRITE: / 'SPACE,   SY-SUBRC:', SY-SUBRC.
WRITE: / NUM, SY-DATUM.
ULINE.
```

本程序使用 不同的格式 选项输出压 缩数字 NUM 和系统字段 SY-DATUM。

第一和最后 的输出是用 户特有的。对于所有的 其他输出，系统读取表 格 T005X。它不查找条 目“XYZ” 而是自己设 置输出格式 。T005X 中的条目是 客户特有的 。

货币特有的 输出格式

要根据货币 格式化数字 字段的输出 ，请使用 WRITE 语句的 CURRENCY 选项：

语法
WRITE <f> CURRENCY <c>.
该语句根据 货币 <c> 确定输出中 的小数位。 如果 <c> 的内容在表 格 TCURX 中作为货币 关键字 CURRKEY 存在，则系 统根据 TCURX 中的 CURRDEC 项设置小数 位。否则， 它使用两个 小数位的缺 省设置。这 意味着表格 TCURX 必须只包含 数字中小数 位不为 2 的例外。

货币的输出 格式不依赖 于程序中可 能存在的数 字的小数位 数。系统只 使用数字顺 序。该数字 顺序代表以 所使用的最 小货币单位 指定的数额 ，例如美国 美圆(USD) 中的分和比 利时法郎(BEF)中 的法郎。由 于 ABAP/4 程序处理货 币量，因此 建议使用不 带小数位的 数据类型 P。

```
REPORT SAPMZTST LINE-SIZE 40.

DATA: NUM1 TYPE P  DECIMALS 4 VALUE '12.3456',
      NUM2 TYPE P             VALUE '123456'.

SET COUNTRY 'US'.

WRITE: 'USD', NUM1 CURRENCY 'USD', NUM2 CURRENCY 'USD',
     / 'BEF', NUM1 CURRENCY 'BEF', NUM2 CURRENCY 'BEF',
     / 'KUD', NUM1 CURRENCY 'KUD', NUM2 CURRENCY 'KUD'.
```

本程序定义 两个压缩数 字 NUM1 和 NUM2， 包含相同数 字顺序，但 小数位不同 。这些数字 出现在几种 货币的输出 中：

对于每种货 币，因为只 参考数字顺 序，因此 NUM1 和 NUM2 的输出格式 相同。因为 最小单位是 分即百分之 一美圆，因 此货币 US Dollar (USD) 显示为两个 小数点的缺 省设置。对 于比利时法 郎 (BEF)， TCURX 中的 CURRDEC 设置为 0，因为比 利时法郎无最小单位。 科威特第纳 尔 (KUD) 有千分之一 单位因此有 三个小数位 （CURRDEC 为 3）。

单位特有的 输出格式

可以根据某 一单位格式 化类型 P 字段。例如 ，数量应该 没有小数位 ，重量规格 应该有三个 小数位，等 等。为此， 请使用 WRITE 语句的 UNIT 选项：

语法
WRITE <f> UNIT <u>.
该语句根据 单位 <u> 设置小数位 。<u> 的内容必须 是列 MSEHI 中数据库表 格 T006 中的条目。 列 DECAN 中的条目然 后确定要显 示的字段 <f> 中的小数位 。如果系统 在表格 T006 中没有发现 条目 <u>，则 忽略该选项 。
下列限制适 用于该操作 ：
<f> 必须为压缩 数字（类型 P）。
如果 <f> 比单位 <u> 的小数位少 ，则系统忽 略该选项。
如果 <f> 比单位 <u> 的小数位多 ，并且该操 作不截断任 何不等于 0 的数字，则 系统只使用该选项。

```
REPORT SAPMZTST LINE-SIZE 40.

DATA: NUM1 TYPE P DECIMALS 1 VALUE 1,
      NUM2 TYPE P DECIMALS 4 VALUE '2.5'.

SET COUNTRY 'US'.

WRITE: 'KG', NUM1 UNIT 'KG', NUM2 UNIT 'KG',
     / 'PC', NUM1 UNIT 'PC', NUM2 UNIT 'PC'.
```

本程序定义 两个压缩数 字，一个小 数位的 NUM1 和四个小数 位的 NUM2。 如果单位 “ KG”（千 克）在表格 T006 中有三个小 数位并且“ PC”(件 数)在表格 T006 中有 0 个小数位， 则输出显示 如下：

因为 NUM1 少于三个小 数位，所以 系统忽略 NUM1 的选项 UNIT ‘KG’ 。UNIT 'PC' 选项缩短 NUM1 的输出到 0 个小数位。 对于 NUM2， 系统忽略选 项 UNIT 'PC'， 因为小数位 不等于 0，否则将 被截断。

创建空行

要创建空行 ，请按如下 格式使用 SKIP 语句：

语法
SKIP [<n>].

系统从当前 行开始，向 当前行中写 入〈n〉个空行。如 果省略〈n〉选项，则系 统创建一个 空行。
SKIP 有下列限制 ：
如果当前 页上剩余行 数太小，则 SKIP 语句产生分 页，如果存 在页脚，则 显示出来。 系统将下一 输出定位到 新页页眉下 的第一行。
在页开始 处，如果该 页是列表级 的第一页或 者该页由 NEW-PAGE 语句创建的 ，则系统只 执行上述语 句。对于其 他页， 系统 在页开始处 忽略该语句 。
如果上述 语句是最后 列表页的最 后的输出语 句（即再没 有其他 WRITE 或 ULINE 语句），则系统将其忽 略。
在缺省设置 中，系统不 输出使用 WRITE 带 AT 选项创建的 任何空行。 空行是只包 含字符串的 行且每个字 段只包括空 字符。但是 ，如果想在 输出字符串 时要输出 WRITE 语句创建的 空行，则请 使用该语句 ：

语法
```
SET BLANK LINES ON|OFF.
```
使用 ON 选项，系统 在输出中不 再取消使用 WRITE 语句创建的 空行。要复 位缺省设置 ，请使用 OFF 选项。

例如，使用 该语句在列 表中提供空 表格条目。 注意，除非 事先指定 SET BLANK LINES ON，否则 系统显示不 包含任何内 容的空行， 例如，空输 入字段或者 空复选框。

下列程序创 建 5 个空行。在 6 行中显示输 出 '*****' 。

```
REPORT SAPMZTST.

SKIP 5.

WRITE '*****'.
```

下列程序不 创建任何空 行。在首行 显示输出 '*****'。 SET BLANK LINES OFF 语句只用于 强调缺省设 置。

```
REPORT SAPMZTST.

SET BLANK LINES OFF.

DO 5 TIMES
  WRITE / ' '.
ENDDO:

WRITE '*****'.
```

因为使用 SET BLANK LINES ON 语句，所以 下面的程序 创建 5 空行。在第 6 行显示输出 '*****' 。

```
REPORT SAPMZTST.

SET BLANK LINES ON.
DO 5 TIMES
  WRITE / ' '.
ENDDO.
SET BLANK LINES OFF.

WRITE  / '*****'.
```

画线、框架 和网格

ABAP/4 提供创建水 平和垂直线 的几种可能 性。关于相 应语句的列 表，参见 *输出屏幕上 的线和空行* (页 **错误！链接无效。**) 。
系统自动连 接在连接处 和框架相交 的线。如果 在该方向上 至少一条线 没有空字符 或者空行分 隔该线，则 线相交。根 据某点处相 交线的类型 和数目，可 以画下列线 段。
关于自动连 线的示例， 请在任何系 统中调用 SHOWLINE 报表。
要防止系统 意外的连线 ，可以使用 特殊线：

注意在该上 下文中，页 眉和页脚占 页的数行。 如果忘记创 建空行可能 导致不需要的连线。例 如，在标准 页眉后的首 行中输出垂 直线将自动 连接这些线 到页眉的下划线。

可以结合不 同的可用线 段以定义列 表的格式。 下列示例显 示

直线

下例显示如 何创建水平 和垂直线。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING.

SKIP TO LINE 3.
ULINE AT 2(1).
WRITE 4 '-'.
WRITE 6 '--'.
WRITE 9 '---'.
ULINE AT 12(4).

SKIP TO LINE 1.
POSITION 18.
WRITE '|'.

SKIP TO LINE 3.
DO 4 TIMES.
  NEW-LINE.
  POSITION 18.
  WRITE '|'.
ENDDO.
```

输出显示如 下：

第一条 ULINE 语句创建一 列的水平线 。第一条 WRITE 语句的连字 符显示为正 常输出字段 。第二条 WRITE 语句的连字 符创建两列 宽的直线。 下面三个连 字符与 ULINE 语句一起创 建七列宽的 直线。

输出的第一 个'|'字符在第一 行中创建垂 直线。其他 四个''字符创建四 条线段长的 直线，从第 3 行开始。

角

下面的示例 显示如何创 建不同的角 。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING.

WRITE   '--'.
WRITE / '|'.

SKIP TO LINE 1.
ULINE AT 5(6).
NEW-LINE.
WRITE 10 '|'.

SKIP TO LINE 4.
WRITE: '|        |',
     / '----------'.
```

输出如下：

垂直线的终 点与水平线 的终点会合 处，都出现 角。

T 型交叉

下面示例显 示如何创建 不同的 T 交叉。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING.

WRITE '---'.
WRITE /2 '|     |'.
ULINE AT /5(8).

SKIP TO LINE 4.
DO 3 TIMES.
  WRITE '|'.
  NEW-LINE.
ENDDO.
SKIP TO LINE 5.
WRITE '---------'.
SKIP TO LINE 4.

ULINE AT 6(10).
WRITE 15 '|' .
```

输出如下：

直线终点与 直线垂直相 交处，都出 现 T 型交叉。

十字交叉

下例显示如 何创建十字 相交。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING.

WRITE   '    |'.
WRITE  /'-------'.
WRITE  /'    |'.

SKIP TO LINE 1.
DO 3 TIMES.
  WRITE 12 SY-VLINE.
  NEW-LINE.
ENDDO.
SKIP TO LINE 2.
ULINE AT 12(1).
```

输出如下：

如果两直线 交叉，则出 现十字相交 。

使用特殊线

如果使用封 闭嵌合框架 或者封闭层 次结构表示 式，则可能 要将某些线 段隔开，即 使没有插入 空字符或空 行的空间。
在这种情况 下，可以使 用在 INCLUDE 程序 <LINE> 中定义为系 统定义常数 的特殊线：

语法
WRITE <lin> AS LINE.
要在程序中 使用特殊线 ，必须在程 序中包括 INCLUDE 程序 〈LINE〉或者更复杂 的 INCLUDE 程序 〈LIST〉。 INCLUDE 程序 〈LINE〉包含特殊线 的简短说明 。
系统在输出 列表中以定 义的方法显 示特殊线。 直线仅在实 际相交的地 方连接。系 统并不能自 动延长特殊 线以使其相 交。
输出特殊线 的最简单办 法就是使用 现有的关键 字结构（参 见 *通过语句结 构使用 WRITE*（页 **错误！链接无效。**））。从屏幕 “集成 WRITE 语句”选择 单选按钮“ 线”，然后 选择“显示 ”。出现下 列对话框：

它包含所有 可用的特别 线，非常容 易就能将其 包括在程序 代码中。

以下程序一 方面显示如 何使用特殊 线创建封闭 图案。另一 方面，也显 示如何在列表中动态编 程线。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING LINE-SIZE 60.

INCLUDE <LINE>.

DATA: X0 TYPE I VALUE 10,
      Y0 TYPE I VALUE 10,
      N TYPE I VALUE 16,
      I TYPE I VALUE 0,
      X TYPE I, Y TYPE I.

X = X0. Y = Y0. PERFORM POS.

WHILE I LE N.
  WRITE LINE_BOTTOM_LEFT_CORNER AS LINE.
  X = X + 1. PERFORM POS.
  ULINE AT X(I).
  X = X + I. PERFORM POS.
  WRITE LINE_BOTTOM_RIGHT_CORNER AS LINE.
  Y = Y - 1. PERFORM POS.
  DO I TIMES.
    WRITE '|'.
    Y = Y - 1. PERFORM POS.
  ENDDO.
  WRITE LINE_TOP_RIGHT_CORNER AS LINE.
  I = I + 1.
  X = X - I. PERFORM POS.
  ULINE AT X(I).
  X = X - 1. PERFORM POS.
  WRITE LINE_TOP_LEFT_CORNER AS LINE.
  Y = Y + 1. PERFORM POS.
  DO I TIMES.
    WRITE '|'.
    Y = Y + 1. PERFORM POS.
  ENDDO.
  I = I + 1.
ENDWHILE.

FORM POS.
  SKIP TO LINE Y.
  POSITION X.
ENDFORM.
```

在该程序中 ，使用子程 序 POS 为每个输出 设置位置 X、Y。输 出显示如下 ：

程序创建封 闭的螺旋结 构，如果不 使用特殊线 将不可能建 立该结构。 可以使用变量 N 设置螺旋的 数目。

编程框架

可以使用 ABAP/4 中可用的线 类型编程框 架。下面的 样本程序定 义宏 WRITE_FRAME，可以取代 WRITE <f> 语句。系统 在 WRITE_FRAME 语句指定的 字段 <f> 周围画框架 ，该框架字 段动态适配 该字段的长度。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING LINE-SIZE 60.

DATA: X TYPE I, Y TYPE I, L TYPE I.

DEFINE WRITE_FRAME.
  X = SY-COLNO. Y = SY-LINNO.
  WRITE: '|' NO-GAP, &1 NO-GAP, '|' NO-GAP.
  L = SY-COLNO - X.
  Y = Y - 1. SKIP TO LINE Y. POSITION X.
  ULINE AT X(L).
  Y = Y + 2. SKIP TO LINE Y. POSITION X.
  ULINE AT X(L).
  Y = Y - 1. X = SY-COLNO. SKIP TO LINE Y. POSITION X.
END-OF-DEFINITION.

SKIP.
WRITE       'Demonstrating'.
WRITE_FRAME 'dynamic frames'.
WRITE       'in'.
WRITE_FRAME 'ABAP/4'.
WRITE       'output lists.'.
```

上面程序中 定义的宏 WRITE_FRAME 将在下面的 输出中介绍 。关于宏的 详细信息， 参见 *定义和调用宏*（页 **错误！链接无效。**）。

编程网格

可以使用 ABAP/4 中可用的行 类型为表格 类型的列表 编程网格。 下面的样本 程序定义两 个宏 NEW_GRID 和 WRITE_GRID。 NEW_GRID 用于初始化 网格和网格 内部的换行 。可以使用 WRITE_GRID 取代 WRITE <f> 语句。对于 每个使用 WRITE_GRID 的字段输出 ，系统画垂 直线到字段 右边并在下 面画水平网 格线。水平 线动态的适 配该字段长 度。所有输 出字段的线 组成网格。

```
REPORT SAPMZTST LINE-SIZE 60 NO STANDARD PAGE HEADING.

TABLES SPFLI.
DATA:  X TYPE I, Y TYPE I, L TYPE I.

TOP-OF-PAGE.
WRITE 3 'List of Flights in a Dynamic Grid'
        COLOR COL_HEADING.
ULINE.

START-OF-SELECTION.

DEFINE NEW_GRID.
  Y = SY-LINNO. Y = Y + 2. SKIP TO LINE Y.
  X = SY-COLNO. POSITION X. WRITE '|'.
END-OF-DEFINITION.

DEFINE WRITE_GRID.
  X = SY-COLNO. Y = SY-LINNO. POSITION X.
  WRITE:  &1, '|'.
  L = SY-COLNO - X + 1.
  X = X - 2. Y = Y + 1. SKIP TO LINE Y. POSITION X.
  ULINE AT X(L).
  Y = Y - 1. X = SY-COLNO. SKIP TO LINE Y. POSITION X.
END-OF-DEFINITION.

GET SPFLI.

NEW_GRID.
WRITE_GRID: SPFLI-CARRID,
            SPFLI-CONNID,
```

```
                SPFLI-CITYFROM,
                SPFLI-CITYTO.
```

在该程序中 定义的宏 NEW_GRID 和 WRITE_GRID 的功能将在 下面的输出 中介绍。关 于宏的详细 信息，参见 *定义和调用宏* （页 **错误！链接无效。**）。报表连接 到逻辑数据 库 F1S。在 选定屏幕输 入相应值后 ，输出显示 如下：

注意最上端 网格线来自 TOP-OF-PAGE 后语句块中 的 ULINE 语句。系统 自动将列表 正文的垂直 线连接到该 线。

确定页的哪 一部分可以 水平滚动

可以限制可 由用户或从 程序中水平 滚动的页面 部分（参见 *在程序之内 滚动* （页 297））。
下列主题说 明：
可以将这两 种情况结合 起来。

从水平滚动 中排除行

要从水平滚 动中排除某 行（例如， 页眉或注释 行），请采 用如下格式 为该行定义 换行：

语法
```
NEW-LINE NO-SCROLLING.
```
紧随该语句 之后的行不 能水平滚动 。但是，它 可以垂直滚 动。
要撤消上述 语句，请使 用：

语法
```
NEW-LINE SCROLLING.
```
该语句只有 在 NEW-LINE NO-SCROLLING 之后无输出 行的情况下 才有意义。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING
                LINE-COUNT 3 LINE-SIZE 140.

START-OF-SELECTION.

DO 3 TIMES.
  WRITE: / 'SY-INDEX:'.
  DO 10 TIMES.
    WRITE SY-INDEX.
  ENDDO.
ENDDO.

NEW-LINE NO-SCROLLING.
ULINE AT 20(20).
NEW-LINE NO-SCROLLING.
WRITE AT 20 '| **** Fixed! **** |'.
NEW-LINE NO-SCROLLING.
ULINE AT 20(20).

DO 3 TIMES.
WRITE: / 'SY-INDEX:'.
  DO 10 TIMES.
    WRITE SY-INDEX.
  ENDDO.
ENDDO.
```

本程序创建 不带页眉和 页脚的三页 ，每页三行 。因为 NEW-LINE NO-SCROLLING， 第二页的三 行不能滚动 。程序输出 如下：

如果用户滚 动到右边， 输出如下：

第一和第三 页的行滚动 固定行。

水平滚动的 左边界

要确定可水 平滚动的区 域的左边界 ，请使用：

语法
SET LEFT SCROLL-BOUNDARY [COLUMN <col>].
不带 COLUMN 选项时，当 前页可滚动 区的左边界 设置为输出 位置；带 COLUMN 选项时，则 设置为位置 <col>。 现在，只有 该区域的右 边部分可以 水平滚动。

上述语句适 用于整个当 前页并且只 适用于当前 页。必须对 每个新页重 复该语句，否则系统使 用缺省值（ 左列表页边 ）。

要为列表的 所有页设置 相同的可滚 动区，可以 执行该语句 ，例如在 TOP-OF-PAGE 事件中。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING
                LINE-COUNT 3 LINE-SIZE 140.

START-OF-SELECTION.

DO 3 TIMES.
  WRITE: /10 'SY-INDEX:'.
  DO 10 TIMES.
    WRITE SY-INDEX.
  ENDDO.
ENDDO.

SET LEFT SCROLL-BOUNDARY COLUMN 20.

DO 3 TIMES.
WRITE: / 'SY-INDEX:'.
  DO 10 TIMES.
    WRITE SY-INDEX.
  ENDDO.
ENDDO.

SET LEFT SCROLL-BOUNDARY COLUMN 10.
```

本程序创建 三行的两页 ，每页没有 页眉和页脚 。第一条 SET 语句影响第 一页，因为 在第二个 DO 循环中的第 一条 WRITE 语句之前不 自动发生分 页。第二条 SET 语句影响第 二页。程序 输出如下：

如果用户滚 动到右边， 则输出如下 ：

首页的可滚 动区不同于 第二页的可 滚动区，它 在左页边的 固定区域下 面的不同位置消失。

第七章 交互式列表

概览

内容

ABAP/4 允许创建交 互式列表。 从屏幕上的 交互式列表 中，用户可 以选择行、 键入输入以 及使用功能 键、菜单条 或按钮输入 命令。交互 式列表用对 话框功能增 强了输出列 表的传统类 型，因此更 接近对话式 编程。交互 式列表为用 户提供所谓 的“交互式 报表”功能 。
后面的主题 说明

什么是交互 式报表？

传统的、非 交互式报表 由创建单个 列表的一个 程序组成。 这意味着启 动报表之后 ，它创建的 列表必须包 含所有要求 的数据，而 不管用户想 查看的细节 的数目。该 过程可能导 致扩展列表 ，用户必须 从该列表中 挑选相关的 数据。对于 后台处理， 这是唯一可 能的方法。 启动后台作 业之后，没 有任何方法 影响程序。 必须预先做 出所要求的 选择，并且 报表必须提 供详细信息 。
对于对话框 会话，没有 这种限制。 执行时用户 在场，可以 直接控制和 操作程序流 。为了能够 利用联机环 境的所有优 点，传统报 表已发展为 交互式报表 。
交互式报表 允许用户在 会话期间积 极参与检索 和提出数据 。交互式报 表创建的不 是一个扩展 的和详细的 列表，而是 一个精简的 基本列表， 通过定位光 标并输入命 令，用户可 以从该列表 中调用详细 信息。因此 ，交互式报 表减少了实 际需要检索 的数据信息 。
在次列表中 显示详细信 息。次列表 完全覆盖基 本列表，或 者出现在屏 幕上的附加 对话窗口中 。次列表本 身也可以是 交互式的。
除了创建次 列表之外， 交互式报表 还允许从列 表中调用事 务和其它报 表。然后， 这些程序将 显示在列表 中的值用作 输入值。例 如，用户可 以从列表中 调用事务以 更改数据库 表格，该数 据库表的数 据显示在列 表中。

本节说明如 何编程对话 列表并解释 可以用来调 用报表和事 务的 ABAP/4 语句，并在 不同组件之 间传递数据 。
关于说明交 互式报表主 要特征的示 例，参见 *HIDE 技术* （页 299） 。

交互式列表 的事件控制

ABAP/4 程序由事件 关键字控制 （参见 *ABAP/4 中 流控制的概 念*（页 **错误！链接无效。**））。 事件关键字 是交互地使 用列表所必 需的。

交互式列表 事件
下列事件是 交互式列表 环境特有的 ：

```
AT LINE-SELECTION
AT PF<nn>
AT USER-COMMAND
```

如果在程序 中为这些事 件之一定义 处理块，则 程序可以对 某些用户动 作做出反应 。如果用户 随后执行显 示列表中的 已定义的行 为，则系统 将触发相应 事件。系统 将在上述事 件之一的处 理块中编写 的所有输出 语句的输出 写入到所谓 的次列表中 。关于次列 表的详细信 息，参见*基本列表和 次列表*（页 288）。

交互式列表 上的动作
必须在列表 的界面定义中确 定用户可以 在列表上执 行的，并触 发特定事件 的动作。可 以为每个列 表定义单独 的界面。默认情 况下，事件 发生在列表 的下列动作 之后：

```
AT LINE-SELECTION 出现在用户 双击某行之 后，单击热 点或者选择 “编辑 -> 选择”。
AT PF<nn> 出现在用户 按下相应功 能键之后。
AT USER-COMMAND 出现在用户 选择自定义 行为之后。
```

关于用户动 作和用户界 面的详细信 息，参见 *交互式列表 的用户界面*（页 291）。

事件控制带 来的结果
使用事件关 键字编写交 互式列表的 事实带来下 列重要结果 ：如 *通过事件控 制 ABAP/4 程序流*（页 **错误！链接无效。**）中所述那样 ，不能嵌套 处理块，这 是由于每个 新事件关键 字终止前面 的处理块。因此，无法 在交互式列 表的处理块 内处理其它 事件。
尤其不能

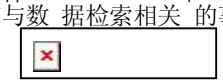
```
使用 GET 和 GET LATE 之类的事件 检索次列表 的数据，但 必须使用 SELECT 语句。只有 对基本列表 才能使用分 配给报表的 逻辑数据库 。如果要在 交互式事件 期间使用逻 辑数据库， 必须调用使 用 SUBMIT 的独立报表 （参见 *调用程序*（页 289））。
使用事件 TOP-OF-PAGE 和 END-OF-PAGE 影响次列表 结构。为了 设置次列表 的页眉，必 须使用事件 TOP-OF-PAGE DURING LINE-SELECTION（ 参见 *次列表的页 眉*（页 292））。
使用独立 的处理块处 理其它交互 式事件。一 定的用户动 作总是触发 程序中同一 个处理块。必须使用处 理块内的控 制语句（IF 和 CASE） ，以确保系 统处理要求 的语句。在 该环境中有 几个系统字 段帮助完成 该操作（参 见 *次列表系统 字段*（页 293））。
```

基本列表和 次列表

本主题概述 基本列表和 次列表。

创建基本列 表

ABAP/4 程序将处理 与数据检索 （START-OF-SELECTION、 GET，等 等）相关的 事件时创建 的输出数据 放置到所谓 的基本列表 中。
默认情况下 ，基本列表 具有标准页 眉（参见 *标准页眉*（页 **错误！链接无效。**））。创建基 本列表时，如果发生事 件 TOP-OF-PAGE 和 END-OF-PAGE， 则系统将后 续所有输出 写入基本列 表的页眉或 页脚。处理 完所有与数 据检索相关 的事件之后 ，系统在输 出屏幕上显 示基本列表 。

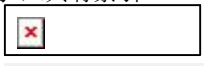


SY-LSIND 系统字段包 含当前创建 的列表的索 引。创建基 本列表时， SY-LSIND 等于 0。

基本列表的 示例，参见*创建列表*（页 **错误！链接无效。**）。

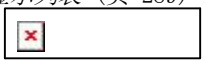
创建次列表

如果用户界 面允许触发 事件的动作 ，并且相应 的交互式事 件的关键字 出现在报表 中，则该列 表就是交互 式列表（参 见 *交互式列表 的事件控制*（页 318））。在交互 式列表事件 中执行的所 有输出语句 将其数据写 入具有索引 SY-LSIND 的新列表中 （列表级别 ）。



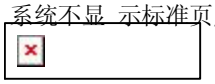
每次发生交 互式列表事 件，系统自 动使 SY-LSIND 加 1。

处理完事件 关键字的整 个处理块之 后，或者由 于 EXIT 或 CHECK 而离开处理 块之后，系 统显示该列表。默认情 况下，新列 表完全覆盖 前面的列表 。如果要编 写只覆盖部 分前面列表 的窗口，参 见 *在对话窗口 中显示列表* (页 289) 。



在交互式列 表事件中创 建的所有列 表都是次列 表。

每个交互式 列表事件创 建新次列表 。使用一个 ABAP/4 报表，可以 维护一个基 本列表和最 多 20 个次列表。 如果用户在 下一级上创 建列表（即 ，SY-LSIND 增加），则 系统将存储 前面的列表 ，并显示新 列表。只有 一个列表是 活动的，并 且它总是最 近创建的列 表。要删除 现有列表， 参见 *维护列表* (页 292) 。
对于次列表 ，系统不显 示标准页眉 。要为次列 表创建页眉 ，参见 *次列表的页 眉* (页 292) 。



```
REPORT SAPMZTST.
WRITE: 'Basic List, SY-LSIND =', SY-LSIND.
AT LINE-SELECTION.
WRITE: 'Secondary List, SY-LSIND =', SY-LSIND.
```

执行程序之 后，系统将 显示下列基 本列表：

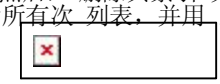
如果将工具 栏与 *标准列表的 用户界面* (页 **错误！链接无效。**) 的标准工具 栏进行比较 ，发现没有 任何区别。 然而，在行 上定位光标 之后，事件 关键字 AT LINE-SELECTION 将导致这种 区别，通过 选择“编辑 -> 选择”、双 击鼠标或按 功能键 F2 触发该事件 （参见 *允许行选择* (页 300)）。因此， 该列表是交 互式的，并 且在用户完 成动作之后 ，输出屏幕 发生变化：

第一个次列 表覆盖基本 列表。该列 表没有标准 页眉。它又 是交互式列 表。通过选 择“选择” ，用户现在 最多可以创 建 20 个这样的列 表。试图生 成多于 20 个列表将导 致运行错误 。

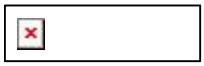
使用“返回 ”， 用户可以返 回到前面列 表。关于返 回到前面列 表的详细信 息，参见*维护列表* (页 292) 。

维护列表

要从高列表 级返回到较 低级 (SY-LSIND)， 用户选择次 列表上的“ 返回”。然 后，系统释 放当前显示的列表，并 激活前一步 创建的列表 。系统删除 已释放列表 的内容。
要显式地指 定在其中放 置输出的列 表级，请设 置 SY-LSIND 字段。系统 只接受与现 有列表级相 对应的索引 值。然后， 删除其索引 大于或等于 指定索引所 有现有列表 级。例如， 将 SY-LSIND 设置为 0，系统将 删除所有次 列表，并用 当前次列表 覆盖基本列 表。



系统只在事 件结束处， 直接在显示 次列表之前 对 SY-LSIND 的操作做出 反应。所以 ，如果在处 理块中使用 语句，该语 句的 INDEX 选项使用 SY-LSIND 索引（如 SCROLL） 访问列表， 则应确保只 在处理这些 报表之后使 用 SY-LSIND 字段。避免 无意混淆的 最好方法是 ，总是将使 用 SY-LSIND 的语句作为 处理块的最 后一条语句输入。



```
REPORT SAPMZTST.
WRITE: 'Basic List, SY-LSIND =', SY-LSIND.
AT LINE-SELECTION.
IF SY-LSIND = 3.
  SY-LSIND = 0.
ENDIF.
```

```
WRITE: 'Secondary List, SY-LSIND =', SY-LSIND.
```

执行程序之 后，系统显 示基本列表 ，包含以下 行：

```
Basic List, SY-LSIND =    0
```

由于 AT LINE-SELECTION 语句的缘故 ，该列表是 交互式的， 并使用户动 作“选择”可用(参见 *允许行选择* (页 300))。如果用 户将光标定 位在列表行 上，并选择 “选择”以 触发 AT LINE-SELECTION 事件，系统 将显示次列 表，包含以 下行：

```
Secondary List, SY-LSIND =    1
```

再次选择“ 选择”将产 生：

```
Secondary List, SY-LSIND =    2
```

“转向 -> 后退”将转 到前一列表 级。第三次 选择“选择 ”将产生次 列表，由于 IF 条件，该次 列表包含以 下行：

```
Secondary List, SY-LSIND =    0
```

系统删除列 表级 1 和 2。如果现 在选择“转 向 -> 后退”，将 返回到启动 报表的屏幕 。如果选择 “选择”， 系统将创建 次列表，其 索引为 1。但是， 0 级上的列表 不再是基本 列表（无页 眉），但还 是次列表。

次列表系统 字段

用每个交互 式事件，系 统自动设置 下列系统字 段：

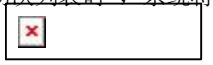
系统字段	信 息
SY-LSIND	当 前事件中创 建的列表索 引（基本列 表 = 0）
SY-LISTI	触 发事件的列 表级索引
SY-LILLI	触 发事件的行 的绝对号
SY-LISEL	触 发事件的行 的内容
SY-CUROW	触 发事件的行 在窗口中的 位置（计数 开始于 1）
SY-CUCOL	触 发事件的列 在窗口中的 位置（计数 开始于 2）
SY-CPAGE	触 发事件的列 表显示的第 一页的页号
SY-STARO	触 发事件的列 表显示的第 一页第一行 的行号（计 数开始于 1）。该行 可能是页眉
SY-STACO	触 发事件的列 表显示的第 一列的列号 （计数开始 于 1）
SY-UCOMM	触 发事件的功 能代码
SY-PFKEY	显 示列表的状 态

如果使用 自定义的列 表界面，则 系统字段 SY-UCOMM 和 SY-PFKEY 很重要（参 见 *定义单独的 用户界面* (页 288))。
可以使用包 含在上面所 列出的系统 字段中的信 息构造次列 表。详细信 息及示例， 参见 *自动传递数据* (页 297) 。

次列表的页 眉

系统在次列 表上不显示 标准页眉， 并且不触发 事件 TOP-OF-PAGE。 要创建次列 表的页眉， 必须增强 TOP-OF-PAGE：

语法
```
TOP-OF-PAGE DURING LINE-SELECTION.
```
对每个次列 表，系统都 触发该事件 。如果要为 不同列表级 创建不同的 页眉，必须 相应地编写 该事件的处 理块，例如 ，通过使用 系统字段， 如控制语句 （IF 和 CASE） 中的 SY-LSIND 或 SY-PFKEY。用户垂直滚 动次列表时 ，系统将保 留页眉而只 滚动页眉下 面的列表行 。



```
REPORT SAPMZTST.
WRITE  'Basic List'.
AT LINE-SELECTION.
WRITE  'Secondary List'.
```

```
TOP-OF-PAGE DURING LINE-SELECTION.

CASE SY-LSIND.
  WHEN 1.
    WRITE 'First Secondary List'.
  WHEN 2.
    WRITE 'Second Secondary List'.
  WHEN OTHERS.
    WRITE: 'Secondary List, Level:', SY-LSIND.
ENDCASE.

ULINE.
```

用户执行上 面的程序之 后，系统将 显示基本列 表，与 *创建次列表* (页 318) 中的示例列 表相似。用 户可以选择 “选择”以 创建次列表 。列表级 1 的次列表如 下所示：

列表级 3 的次列表如 下所示：

由于 TOP-OF-PAGE DURING LINE-SELECTION 处理块内的 CASE 控制语句的 缘故，系统 为每个次列 表创建不同 页眉。

列表中的消 息

ABAP/4 允许您根据 错误的严重 程度通过显 示影响程序 流的消息对 错误或有疑 问的用户输 入做出反应 。处理消息 是对话编程 中详细描述 的主题（参 见 *处理错误和 消息* (页 **Error! Not a valid link.**))。该主题 只解释信息 对交互式列 表处理的影 响。
在表 T100 中存储和维 护消息。消 息是通过语 言、双字符 标识和三个 数字的号码 存储的。可 以为输出的 各种消息分 配不同消息 类型。消息 对程序流的 影响取决于 消息类型。
在程序中使 用 MESSAGE 语句静态或 动态地输出 信息，并确 定信息类型 。
如果要静态 地使用程序 中特定标识 的消息，请 使用 REPORT 或 PROGRAM 语句的 MESSAGE-ID 选项：

语法
```
REPORT <rep> MESSAGE-ID <id>.
```
由于该语句 的作用，报 表 <rep> 可以使用存 储标识 <id> 下表格 T100 中的所有消 息。如果动 态地指定消 息标识，则 可以忽略此 选项。
要静态地指 定消息号， 请使用：

语法
```
MESSAGE <c><num> [WITH <F1> ... <F4>].
```
该语句输出 存储在号码 <num> 下表格 T100 中的消息， 它的标识与 REPORT 语句中的标 识相同，与 消息类型 <c> 一样。
要在运行时 动态地指定 消息标识、 类型和号码 ，请使用：

语法
```
MESSAGE ID <id> TYPE <c> NUMBER <num> [WITH <F1> ... <F4>].
```
该语句输出 其标识、号 码和类型存 储在字段 <id>、<num> 和 <c> 中的消息。对该语句，不需要 REPORT 语句中的 MESSAGE-ID 选项。
消息可以有 五种不同类 型。列表处 理期间，这 些消息类型 有下列作用 ：

A (=终止)：
系统在对话 窗口中显示 该消息类型 的消息。用 户使用 **ENTER** 确认信息之 后，系统终 止整个事务 （例如，SE38） 。

E (=错 误) 或 W (=警告) ：
系统在状态 行中显示该 消息类型的 消息。用户 选择 **ENTER** 之后，系统 将按下列情 况运作：
- 创建基 本列表时， 系统终止报 表。
- 创建次 列表时，系 统终止相应 处理块，并 继续显示前 面的列表级 。

I (=信 息) ：
系统在对话 窗口中显示 该消息类型 的消息。用 户选择 **ENTER** 之后，系统 在当前程序 位置继续处 理。

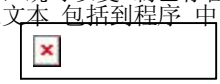
S (=成 功) ：
在输出屏幕 上，系统在 当前创建的 列表的状态 行中显示该 消息类型的 消息。

字符‘&’ 在消息中起 占位符的作 用。如果使 用 WITH 选项，系统 用字段 $\langle f_i \rangle$ 的内容替代 消息中的占 位符‘&’ ，并根据号 码替代数字 化的占位符 ‘&I’。 要在信息中 输出‘&’ ，必须写 ‘ &&’。

要创建、显 示或更改信 息，只需简 单地在 ABAP/4 编辑器中双 击标识 <id> 或信息号。 如果该对象 还不存在， 则系统将询 问是否创建 它。

通过从 ABAP/4 编辑器中选 择“编辑 -> 插入语句...” ，可以很容 易地将静态 MESSAGE 语句包括到 程序中。选 择 MESSAGE 作为语句：

使用此方法 ，既可以复 制已存在的 信息，也可 以创建新对 象（标识和 号码）。另 外，系统以 注释的方式 将信息文本 包括到程序 中。



假定下列信 息存储在标 识 HB 下的 T100 表格中：

下列程序使 用这些信息 ：

```
REPORT SAPMZTST MESSAGE-ID HB NO STANDARD PAGE HEADING.

WRITE 'Basic List'.
MESSAGE S100.

AT LINE-SELECTION.
  IF SY-LSIND = 1.
    MESSAGE ID 'HB' TYPE 'I' NUMBER 100.
  ENDIF.
  IF SY-LSIND = 2.
    MESSAGE E200 WITH SY-LSIND.
  ENDIF.
  WRITE: 'Secondary List, SY-LSIND:', SY-LSIND.
```

执行程序之 后，系统将 显示基本列 表，并在状 态行中显示 成功信息 100。通 过双击选择 行之后，将 出现 AT LINE-SELECTION 事件。系统 创建第一个 次列表时， 它将显示信 息为 100 的对话窗口 。因为信息 200 的信息类型 为 E，所以不 可以创建第 二个次列表 ：

交互式列表 的用户界面

如果希望用 户在列表显 示时与系统 通讯，列表 必须是交互 式的。可以 在列表的用 户界面 (GUI) 的状态中定 义特定交互 式可能性。 要在 R/3 系统中定义 界面状态， 请使用菜单 绘制器工具 。在菜单绘 制器中为特 定交互式可 能性分配功 能代码。用 户行为出现 在整个界面 上之后，ABAP/4 处理器将检 查功能代码 ，并且，如 果有效，将 触发相应事 件。可以从 程序中使用 SY-UCOMM 系统字段访 问功能代码 。
可以为报表 定义单独的 界面，并在 报表中将它 们分配到任 何列表级。 如果在报表 中不指定自 定义的界面 ，而是在程 序中使用三 个事件关键 字 AT LINE-SELECTION、 AT PF<nn> 或 AT USER-COMMAND 中的至少一 个，则系统 自动使用适 当的预定义 标准界面。 这些标准界 面提供的功 能与 *标准列表* (页 **错误！链接无效。**) 中所描述的 标准列表相 同。按照事 件关键词， 它们为用户 提供与系统 交互作用的 附加可能性 。
下面两个主 题说明与预 定义的标准 界面连接的 事件 AT LINE-SELECTION 和 AT PF<nn>。
AT USER-COMMAND 事件主要处 理自己的功 能代码。这 种情况下， 应该使用菜 单绘制器创 建单独界面 ，并定义这 样的功能代 码。下列主 题解释如何 定义单独的 界面和如何 使用 AT USER-COMMAND 事件。
也可以在对 话窗口中显 示列表，而 不是全屏显 示：
不但可以从 界面中触发 交互式事件 ，而且可以 从程序中触 发交互式事 件：

这部分介绍 的用户界面 只适用于类 型为 1 的程序（联 机程序）。 系统相应地 设置界面， 并使用列表 处理器。注 意，创建事 务的程序使 用不同的环 境。

允许行选择

要允许用户 从列表中选 择行，请在 程序中定义 并写入 AT LINE-SELECTION 事件的处理 块：

语法
```
AT LINE-SELECTION.
  <statements>.
```
如果没有为 列表定义单 独的界面， 则系统自动 使用预定义 的交互式界 面。此界面 的外观与标 准界面相同 （参见 *标准列表* (页 **错误！链接无效。**))。
但是，系统 对下列用户 行为做出反 应：
选择菜单 条目“编辑 -> 选择”。
按下功能 键 F2。
双击列表 行或单击热 点（参见*将字问涞 为热点* (页 **错误！链接无效。**)) 。
在列表行上 定位光标之 后（包括页 眉或页脚行 ），选择上 面所描述的 四个行为中 的某一个， 将出现 AT LINE-SELECTION 事件。
功能代码 PICK 内部触发 AT LINE-SELECTION 事件。在预 定义界面中 ，将“编辑 -> 选择”和 F2 分配到 PICK。 设置功能键 F2 自动激活鼠 标功能。详 细信息，参 见 *定义单独的 用户界面* (页 288) 。

```
REPORT SAPMZTST.
WRITE  'Basic List'.
AT LINE-SELECTION.
WRITE: 'Secondary List by Line-Selection',
     / 'SY-UCOMM =', SY-UCOMM.
```

执行该程序 之后，系统 将显示此界 面：

用户可以在 上面所描述 的行为之间 选择。通过 程序创建的 所有次列表 显示都象这样：

对每个交互 式用户行为 ，SY-UCOMM 返回值‘PICK’ 。

允许功能键 选择

要允许用户 通过按下功 能键选择， 请在程序中 定义并编写 AT PF<nn> 事件处理块 ：

语法
```
AT PF<nn>.
  <statements>.
```
如果没有为 列表定义单 独的界面， 则系统使用 预定义的交 互式界面。 标准界面和 此预定义界 面之间的区 别是，把预 定义系统功 能没有使用 的键盘 F<NN> 的所有功能 键都设置为 功能代码 PF<nn>，这里 <nn> 是 01 和 24 之间的数。 如果用户按 下这些键中 的某一个， 系统将触发 相应事件。 这种情况下 ，与光标位 置无关。
要查看用系 统功能预定 义的键列表 ，请将 AT PF<nn> 语句包括到 程序中，显 示输出列表 ，并在列表上的单击鼠 标右键：

使用在左边 有文本的所 有键预定义 系统功能。 这些功能比 自定义事件 有更高的优 先权。系统 不触发预定 义功能代码 键的 AT PF<nn> 事件。

AT PF<nn> 事件只用作 测试目的。 在最终程序 版本中，使 用 AT USER-COMMAND 事件，以及 自定义界面 和单独的、 有意义的功 能键。在自 定义用户界 面上，可以 通过为菜单 条目或按钮 选择的文本 告诉用户行 为的功能。只使用功能 键将失去友 好的用户特 征。

```
REPORT SAPMZTST.
WRITE 'Basic List'.
```

```
AT PF5.
PERFORM OUT.

AT PF6.
PERFORM OUT.

AT PF7.
PERFORM OUT.

AT PF8.
PERFORM OUT.

FORM OUT.
  WRITE: 'Secondary List by PF-Key Selection',
       / 'SY-LSIND =', SY-LSIND,
       / 'SY-UCOMM =', SY-UCOMM.
ENDFORM.
```

执行该程序 之后，系统 将显示基本 列表。用户 可以按下功 能键 F5、F6、 F7 和 F8，以创 建次列表。 例如，如果 用户按下的 第 14 个键是 F6，显示 次列表上的 输出如下：

```
Secondary List by PF-Key Selection

SY-LSIND =  14

SY-UCOMM = PF06

SY-UCOMM 返回功能代 码 PF06。
```

定义单独的 用户界面

可以使用交 互式列表的 单独用户界 面。可以定 义自己的功 能代码，并 通过在菜单 条、按钮、 图标和 PF 代码中提供 适当条目向 用户提供通 常的行为以 激活这种功 能。调用分 配到某个功 能代码的功能，触发 AT USER-COMMAND 事件。在该 事件的处理 块中使用控 制语句（IF 与 CASE） 和 SY-UCOMM 系统字段为 每个功能代 码编写需要 的语句。
用户界面包 含状态和标 题。状态包 含菜单条、 功能键、标 准工具栏和 应用程序工 具栏。可以 将功能代码 分配给所有 这些元素。
后面的主题 说明

定义交互式 列表的状态

要定义用户 界面的状态 ，请使用菜 单绘制器。 菜单绘制器 是 ABAP/4 开发工作台 中的工具之 一。关于如 何使用菜单 绘制器的完 整说明，参 见文档 *ABAP/4 开发工作台* （页 Error! Not a valid link.）。另外，可以 使用 ABAP/4 开发工作台 的对象浏览 器显示、创 建或复制状 态。关于对 象浏览器的 说明，参见 文档 *ABAP/4 开发工作台* （页 Error! Not a valid link.） 。
下面两个主 题涉及到使 用菜单绘制 器创建交互 式列表的单 独界面必须 知道的内容 ：

启动交互式 列表的菜单 绘制器工具

要启动菜单 绘制器工具 ，可以直接 从 ABAP/4 开发工作台 的初始屏幕 上选择“菜 单绘制器” ，或者按下 述步骤操作 ，该方法对 于连接中的 交互式列表 更合适：

1. 在程 序中设置当 前列表的状 态。
   要设置菜单 ，请使用语 句：
   语法
   SET PF-STATUS <stat>.
   此语句设置 用户界面的 状态。直到 设置另一状 态之前该状 态一直是活 动的。<stat> 的名称最多 可以有八个 字符。选择 有意义的名 称说明状态 的用途（例 如 ET PF-STATUS 'MAIN'） 。关于如何 设置状态的 详细信息， 参见 *设置状态* （页 296）。
   要查看程序 现有状态的 列表（如果 有），请使 用 ABAP/4 开发工作台 的对象浏览 器（参见文档 *ABAP/4 开发工作台* （页 Error! Not a valid link.）） 。
2. 在 ABAP/4 编辑器中， 使用鼠标双 击 <stat>。
   如果状态 <stat> 存在，系统 将离开 ABAP/4 编辑器（允 许保存程序 ），并直接 转到状态 <stat> 的菜单绘制 器。继续进 行第 4 点。
   如果状态 <stat> 不存在，则 系统显示下 列对话窗口 ：

3. 选择 “是”。
   系统离开 ABAP/4 编辑器（允 许保存程序 ），并显示 对话窗口“ 创建状态” ，可以按下 列方法填写 该窗口：

   对交互式列 表的“状态 类型”，选 择“列表” 或“对话框 中的列表” 。然后，系 统自动将为 列表处理预 定义的功能 代码加载到 菜单绘制器 中。系统使 用选择的状 态类型，该 状态类型支 持菜单绘制 器中要求的 条目。这些 预定义的条 目符合 SAP 风格指南的 标准（参见 文档 *SAP 样式指南* (页 **Error! Not a valid link.**)），并帮助 您与这些标 准保持一致 。
   对具有完整 用户界面的 全屏显示列 表，使用“ 列表”状态 类型。对于 要显示在对 话窗口中的 、没有菜单 条和标准工 具栏的列表 ，使用状态 类型“对话 框中的列表 ”（参见 *在对话窗口 中显示列表* (页 289) ）。
   选择“输入 ”。
   系统显示状 态的菜单绘 制器。
4. 按照 在 *使用交互式 列表的菜单 绘制器工具* (页 294) 中所描述的 那样定义状 态。

使用交互式 列表的菜单 绘制器工具

使用菜单绘 制器创建自 己的功能代 码，并因此 生成状态中 的条目。为 “列表”创 建的状态已 经包含一些 列表处理的 标准功能代 码。激活“ 显示标准” 。交互式列 表的菜单绘 制器如下所 示：

选择的开发 环境（这里 是 WINDOWS） 不影响菜单 绘制器的功 能或用它定 义的界面。 不需要为每 种环境创建 界面。因为 不同环境中 功能键的设 置也不相同 （例如，WINDOWS 中的 CTRL-F12 是 Motif 中的 ALT-SHIFT-F12），所以需要 做的唯一事 情就是显示 功能键。
建议的菜单 条包含下列 特定于列表 的条目：

下列功能代 码是预定义 的，并已分 配给状态的 不同元素。 请注意，此 设置只针对 列表，如果 创建屏幕的 新状态，则 不是这样。

代码	菜单	状态元素 标准工具栏	功能键	说明
%PC	列表		SHIFT-F8	将 列表写入文 件
PRI	列 表		CTRL-P	打 印显示列表
%EX	列表		SHIFT-F3	退 出处理
PICK	编 辑		F2	事 件 AT LINE-SELECTION
RW	编 辑		F12, ESC	取消 处理
%SC	编 辑		CTRL-F	搜 索式样
%SC+	编 辑		CTRL-G	继 续搜索
BACK	转 向		F3	返 回一级
P--			F21	滚 动到第一窗 口页
P-			F22	滚 动到上一窗 口页
P+			F23	滚 动到下一窗 口页
P++			F24	滚 动到最后窗 口页

另外， 下列功能代 码是预定义 的，但没有 设置为状态 功能，可以 随意分配到 任何空的状 态元素：

代码	说明
PF<nn>	事 件 AT PF<nn>
PP<n>	滚 动到列表页 顶部 <n>
PP-[<n>]	各 自向后滚动 一或 <n> 列表页

PP+[<n>]	各 自向前滚动 一或 <n> 列表页
PS<n>	滚 动到列 <n>
PS--	滚 动到列表的 第一列
PS-[<n>]	各 自向左滚动 一或 <n> 列
PS+[<n>]	各 自向右滚动 一或 <n> 列
PS++	滚 动到列表的 最后一列
PZ<n>	滚 动到行 <n>
PL-[<n>]	向 后滚动到页 的第一行或 滚动 <n> 行
PL+[<n>]	向 前滚动到页 的第一行或 滚动 <n> 行
/....	其 他系统命令

系统（ABAP/4 处理器）直 接查询和处 理上表中列 出的所有功 能代码（除 PICK 和 PF<nn> 之外）。这些功能代码 不触发事件 ，不能用于 AT USER-COMMAND 事件。无论 何时光标位 于列表行， 功能代码 PICK 都触发 AT LINE-SELECTION 事件，功能 代码 PF<nn> 总是触发 AT PF<nn> 事件。因此 ，二者都不 能用于 AT USER-COMMAND 事件中。

上述功能代 码是固定的 。对于其他 任何功能， 可以定义任 意单独的四 字符的功能 代码。使用 有意义的短 格式，例如 SORT， 触发排序处 理。由于许 多系统定义 功能以 P 开头，所以 不应该使用 P 作为自己功 能代码的第 一个字母。

至于相关的 功能键，请 注意下列两 种特殊情况 ：

功能键 **F2**：

双击鼠标总 是等于按下 功能键 **F2**。因此 ，通过双击 激活分配到 **F2** 的每个功能 代码。只有 当功能代码 PICK 分配到 **F2** 之后，双击 才触发 AT LINE-SELECTION 事件。如果 将自己的功 能代码分配 到 **F2**，则双 击将触发 AT USER-COMMAND 事件。如果 将预定义的 功能代码分 配到 **F2**，则双 击将触发相 应系统行为 。

功能键 **F10**：

功能键 **F10** 总是将光标 放置于菜单 条中选择菜 单功能。不 可以将任何 其他（自己 的或预定义 的）功能代 码分配到 **F10**。

可以按照需 要修改菜单 绘制器中建 议的特定列 表的预定义 。可以

用自己的 功能代码替 代功能代码 PICK， 以避免在报 表中触发 AT LINE-SELECTION 事件。然后 可以在单个 处理块中编 码对用户行 为的反应 (AT USER-COMMAND)。

删除不想 支持其功能 的预定义功 能代码。例 如，可能不 希望用户直 接打印列表 或将列表保存到表示服 务器上的文 件中。

修改标准 键设置。例 如，可以将 自己的功能 代码分配到 **F3**，以根 据需要在列 表中导航， 而不是返回 到上一列表 级（“返回 ”）。如果 在相同逻辑 级上保留好 几个列表， 并因此不希望按照标准 **F3** 设置删除显 示列表，则 这一点就显 得很重要。 或者希望在 离开列表级 之前显示警 告（参见 *列表中的消 息* (页 321) ）。

```
REPORT SAPMZTST.

SET PF-STATUS 'TEST'.

WRITE: 'Basic list, SY-LSIND =', SY-LSIND.

AT LINE-SELECTION.

  WRITE: 'LINE-SELECTION, SY-LSIND =', SY-LSIND.

AT USER-COMMAND.

CASE SY-UCOMM.
  WHEN 'TEST'.
  WRITE: 'TEST, SY-LSIND =', SY-LSIND.
ENDCASE.
```

对于该报表 ，在菜单绘 制器中定义 状态 TEST：

1. 将功 能代码 TEST 分配到功能 键 **F5**。

2. 将功 能代码 TEST 输入到菜单 “列表”中 。

3. 将功 能代码 PICK 和 TEST 定义为按钮 。

保存并生成 状态之后， 报表输出屏 幕如下所示 ：

用户既可以 通过按下 F5，也可 以通过选择 “列表 -> 手册测试” ，还可以通 过单击按钮 “手册测试 ”触发 AT USER-COMMAND 事件。用户 可以通过选 择行触发 AT LINE-SELECTION 事件。关于 AT USER-COMMAND 事件的详细 信息，参见 *在程序中使 用自己的功 能代码* (页 294) 。

定义交互式 列表的标题

默认情况下 ，系统使用 程序标题作 为报表输出 屏幕的标题 。要为屏幕 选择另一个 标题，请使 用语句：

语法
```
SET TITLEBAR <tit> [WITH <g1> ... <g9>].
```
该语句设置 输出列表用 户界面的标 题。对所有 输出屏幕， 在使用 SET TITLEBAR 指定另一个 标题之前， 该标题是一 直都是活动 的。<tit> 是标题名， 最多可以有 三个字符。 标题作为自 定义状态连 接到程序。 要维护标题 ，可以使用 ABAP/4 开发工作台 的对象浏览 器或菜单绘 制器工具的 初始屏幕(参见文档*开发工作台* (页 Error! Not a valid link.)) 。
也可以从 ABAP/4-编 辑器中维护 报表标题。双击上面语 句中的 <tit>， 如果标题 <tit> 不存在，则 系统将显示 下列对话窗 口：

如果选择“ 是”，系统 将显示另一 个对话窗口 ，可以在该 窗口中输入 需要的标题 。可以指定 最多九个不 同占位符 &1.... &9。然后 在运行时， 使用 SET TITLEBAR 语句的 WITH 选项，用相 应字段 <g1>.... <g9> 的内容替代 这些标题中 的占位符。 系统也连续 的用相应 <gi> 参数的内容 替代占位符 ‘&’。要 输出通配符 ‘&’ 本身 ，请输入 ‘ &&’。

标题和替代 字符一起最 多可以有 70 个字符长。 系统忽略超 出的字符， 并以空格代 替缺失的变 量 <gi>。

```
REPORT SAPMZTST.
WRITE 'Click me!' HOTSPOT COLOR 5 INVERSE ON.
AT LINE-SELECTION.
SET TITLEBAR 'TST' WITH SY-LSIND.
WRITE 'Click again!' HOTSPOT COLOR 5 INVERSE ON.
```

此程序为每 个次列表设 置新标题。 例如，如果 将标题 TST 定义为上图 的形式，则 第五个列表 级将有下列 标题栏：

关于热点的 详细信息， 参见 *将字段输出为热点* (页 **错误！链接无效。**) 。

在程序中使 用自己的功 能代码

要在程序中 使用自己的 功能代码， 必须设置自 定义界面的 状态，并为 AT USER-COMMAND 事件编码处 理块。

设置状态

如在 *启动交互式 列表的菜单 绘制器工具* (页 324) 中提到的那 样，使用 SET TITLEBAR 语句设置状 态。完整语 法是：

语法
```
SET PF-STATUS <stat> [EXCLUDING <f>|<itab>] [IMMEDIATELY].
```

此语句设置 当前输出列 表的 <stat> 状态。必须 为程序定义 状态 <stat>。 通过设置该 状态，使得 用户可以选 择状态中定 义的功能。 对所有后面 的列表级， 设置另一状 态之前该状 态一直是活 动的。SY-PFKEY 系统字段总 是包含当前 列表的状态 。
使用 SET PF-STATUS 可以显示不 同列表级的 不同用户界 面，并按照 各自需求向 用户提供不 同功能。要 对应于程序 设置系统定 义的列表状 态，请使用 SET PF-STATUS SPACE。
从程序中使 用 EXCLUDING 选项影响状 态的功能。 如果不同列 表级的各个 用户界面区 别不大，则 可以定义单 个全包括状 态，并使用 EXCLUDING 使每个列表 级不需要的 功能代码处 于非活动状 态。指定 <f> 使存储于字 段 <f> 中的功能代 码处于非活 动状态。指 定 <itab> 使存储于内 部表格 <itab> 中的所有功 能代码处于 非活动状态 。字段 <f> 和表格 <itab> 的行的类型 应该是 C，长度应 该为 4。
在交互式事 件的处理块 中使用 IMMEDIATELY 选项更改当 前显示列表 (SY-LISTI) 的状态。如 果没有该选 项，系统将 更改当前次 列表 (SY-LSIND) 的状态，只 在处理块的 末尾处显示 该列表。

```
REPORT SAPMZTST.

DATA FCODE(4) OCCURS 10 WITH HEADER LINE.
FCODE = 'FC1 '. APPEND FCODE.
FCODE = 'FC2 '. APPEND FCODE.
FCODE = 'FC3 '. APPEND FCODE.
FCODE = 'FC4 '. APPEND FCODE.
FCODE = 'FC5 '. APPEND FCODE.
FCODE = 'PICK'. APPEND FCODE.

SET PF-STATUS 'TEST'.
WRITE: 'PF-Status:', SY-PFKEY.

AT LINE-SELECTION.

IF SY-LSIND = 20.
  SET PF-STATUS 'TEST' EXCLUDING FCODE.
ENDIF.

WRITE: 'Line-Selection, SY-LSIND:', SY-LSIND,
     / '                SY-PFKEY:', SY-PFKEY.

AT USER-COMMAND.

........
```

假定在状态 TEST 中定义从 FC1 到 FC5 的功能代码 ，并将它们 分配到按钮 ：

执行程序之 后，例如， 用户可以通 过选择行创 建次列表。对最多到 20 级的次列表 ，用户界面 TEST 与基本列表 的界面相同 ：

在列表级 20 上，EXCLUDING ITAB 使创建次列 表的所有功 能代码处于 非活动状态 。

这避免用户 通过创建次 列表号 21 而导致程序 异常终止。

```
REPORT SAPMZTST.

WRITE: 'SY-LSIND:', SY-LSIND.

AT LINE-SELECTION.

  SET PF-STATUS 'TEST' IMMEDIATELY.
```

执行该程序 之后，输出 屏幕显示的 基本列表和 为行选择预 定义的用户 界面（参见 *允许行选择* (页 322)）：

如果用户选 择“选择” ，则用户界 面将发生变 化。但是， 由于 AT LINE-SELECTION 处理块不包 含输出语句 ，所以系统 不创建次列 表：

用前面示例 中使用的方 法定义状态 TEST。

T USER-COMMAND 事件

要允许程序 对触发自定 义功能代码 的用户行为 作出反应， 请定义并编 码 AT USER-COMMAND 事件的处理 块。

语法
```
AT USER-COMMAND.
  <statements>.
```
无论何时用 户从自定义 用户界面中 选择自定义 功能代码， 都将发生 AT USER-COMMAND 事件。如果 用户选择为 系统功能预 定义的功能 代码，或者 选择总是触 发 AT LINE-SELECTION 事件的功能 代码 PICK， 则不发生该 事件。关于 这些功能代 码的详细信 息，参见 *使用交互式 列表的菜单 绘制器工具* (页 325) 中的表格。
使用 AT USER-COMMAND 事件处理块 中的 SY-UCOMM 系统字段区 别不同的功 能代码。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING.

WRITE: 'Basic List',
     / 'SY-LSIND:', SY-LSIND.

TOP-OF-PAGE.

WRITE 'Top-of-Page'.
ULINE.

TOP-OF-PAGE DURING LINE-SELECTION.

CASE SY-PFKEY.
  WHEN 'TEST'.
    WRITE 'Self-defined GUI for Function Codes'.
    ULINE.
ENDCASE.

AT LINE-SELECTION.

SET PF-STATUS 'TEST' EXCLUDING 'PICK'.
PERFORM OUT.
SY-LSIND = SY-LSIND - 1.

AT USER-COMMAND.

CASE SY-UCOMM.
  WHEN 'FC1'.
    PERFORM OUT.
    WRITE / 'Button FUN 1 was pressed'.
  WHEN 'FC2'.
    PERFORM OUT.
    WRITE / 'Button FUN 2 was pressed'.
  WHEN 'FC3'.
    PERFORM OUT.
    WRITE / 'Button FUN 3 was pressed'.
  WHEN 'FC4'.
    PERFORM OUT.
    WRITE / 'Button FUN 4 was pressed'.
  WHEN 'FC5'.
    PERFORM OUT.
    WRITE / 'Button FUN 5 was pressed'.
ENDCASE.

SY-LSIND = SY-LSIND - 1.

FORM OUT.
  WRITE: 'Secondary List',
       / 'SY-LSIND:', SY-LSIND,
       / 'SY-PFKEY:', SY-PFKEY.
ENDFORM.
```

执行该程序 之后，系统 将显示下列 基本列表， 并带有自定 义页眉：

用户可以通 过双击行触 发 AT LINE-SELECTION 事件。系统 设置状态 TEST 并使功能代 码 PICK 处于非活动 状态。状态 TEST 与在示例 *设置状态* (页 327) 中定义的状 态相同。次 列表的页眉 取决于状态 。通过双击 创建的次列 表如下所示 ：

这里，双击 行不再触发 事件。在另 一方面，用 户现在可以 使用工具栏 中五个自定 义按钮。它 们触发 AT USER-COMMAND 事件。由于 CASE 语句的缘故 ，系统对不 同按钮作出 不同反应。 例如，单击 “功能 3”之后， 屏幕如下所 示：

对每个交互 式事件，系 统使 SY-LSIND 系统字段减 一，因此抵 消了自动的 增加。现在 ，所有次列 表的级与基 本列表相同 。它们将覆 盖基本列表 。注意，创 建次列表时 ，SY-LSIND 仍包含 1。

在对话窗口 中显示列表

要在对话窗 口中而不是 全屏幕显示 次列表，请 使用 WINDOW 语句：

语法
WINDOW STARTING AT <left> <upper> [ENDING AT <right> <lower>].
此语句使当 前列表（索 引 SY-LSIND） 在对话窗口 中显示。使 用 <left> 和 <upper> 指定左上角 的列和行， 借此指定基 本列表的输 出屏幕。如 果 <upper> 等于 0，则列表 全屏幕显示 。右下角的 坐标取决于 次列表需要 的空间。可 以指定 ENDING 选项，使用 <right> 和 <lower> 设置右下角 的列和行。 窗口不会超 出这些值。 默认情况下 ，系统使用 发生事件的 窗口右下角 的值。
WINDOW 语句只在交 互式事件处 理块中有效 ，也就是只 影响次列表 。对话窗口 中的列表的 列表功能与 全屏幕中的 列表的列表 功能相同。
对话窗口的 用户界面与 全屏幕显示 的不同。对 话窗口没有 菜单条和标 准工具栏。 应用程序工 具栏出现在 窗口下边。 这是 R/3 系统中的所 有对话窗口 的普遍特征 。
如果没有设 置自定义状 态，但设置 了报表中 AT LINE-SELECTION 或 AT PF<nn> 的程序处理 块（从该处 理块中将列 表写入对话 窗口），则 系统使用这 些窗口的预 定义界面。
要亲自定义 对话窗口的 状态，请按 *启动交互式 列表的菜单 绘制器工具* (页 324) 中所述的步 骤操作。但 是，在第 3 点下将“对 话框中的列 表”标记为 “状态类型 ”。出现菜 单绘制器， 使预定义条 目与该状态 类型一致：

系统不提供 菜单条或标 准工具栏。 在应用程序 工具栏中， 预设置了功 能代码 PRI、%SC、 %SC+ 和 RW 以允许用户 打印列表、搜索式样和 离开窗口。对于任何其 他功能，请 使用 *使用交互式 列表的菜单 绘制器工具* (页 325) 中所述的菜 单绘制器。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING.

SET PF-STATUS 'BASIC'.

WRITE 'Select line for a demonstration of windows'.

AT USER-COMMAND.
  CASE SY-UCOMM.
    WHEN 'SELE'.
      IF SY-LSIND = 1.
        SET PF-STATUS 'DIALOG'.
        SET TITLEBAR 'WI1'.
        WINDOW STARTING AT 5 3 ENDING AT 40 10.
        WRITE 'Select line for a second window'.
      ELSEIF SY-LSIND = 2.
        SET PF-STATUS 'DIALOG' EXCLUDING 'SELE'.
        SET TITLEBAR 'WI2'.
        WINDOW STARTING AT 45 10 ENDING AT 60 12.
        WRITE 'Last window'.
      ENDIF.
  ENDCASE.
```

此程序设置 基本列表的 BASIC 状态。在状 态 BASIC 中，建议将 功能代码 PICK 分配到 **F2**，这是 由于用自定 义功能代码 SELE（ 文本 SELECT） 替代状态类 型“列表” ，该自定义 功能代码也 分配到了按 钮：

因此，SELECT、 **F2** 和双击鼠标 都触发事件 AT USER-COMMAND。 在相应处理 块中，列表 级 1 和 2 的状态为 DIALOG， 并出现在对 话窗口中。 在状态 DIALOG 中，完全与状态 BASIC 相同，由于 用功能代码 SELE 替代状态类 型“列表” ，建议的功 能键 **F2**

的功能代码 分配到 RW 之后的应用 程序工具栏 中。执行程 序并两次选 择行之后， 输出屏幕如 下所示：

对话窗口的 标题栏 WI1 和 WI2 是自定义的 。在第二个 对话窗口中 ，SET PF-STATUS 语句的 EXCLUDING 选项使功能 代码 SELE 处于非活动 状态。

从程序中触 发事件

不是使用户 通过输出屏 幕上的行为 触发交互式 事件，而是 您自己可以 从程序中触 发事件。使 用语句：

语法
SET USER-COMMAND <fc>.
此语句在完 成当前列表 后生效。系 统显示列表 之前，将触 发与存储在 <fc> 中的功能代 码对应的事件，而独立 于应用的用 户界面。这 意味着，对 于自定义的 功能代码， 将发生 AT USER-COMMAND 事件，对于 功能代码 PICK 和 PF<nn>， 将发生事件 AT LINE-SELECTION 和 AT PF<nn>。

只有把光标 置于列表行 上时，功能 代码 PICK 才触发事件 （参见示例 ）。

对于为系统 功能定义的 功能代码， 系统不触发 事件，而是 执行相应行 为。例如， 要一起显示 列表和“搜 索”对话窗 口，请在 <fc> 中指定 ‘%SC’ 。通过指定 适当的滚动 功能代码， 可以在显示 列表之前将 它滚动到一 定位置。关 于系统定义 功能代码的 概要，参见 *使用交互式 列表的菜单 绘制器 工具* (页 325) 。

如果创建列 表时使用好 几个 SET USER-COMMAND 语句，系统 只执行最后 一个语句。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING.

SET USER-COMMAND 'MYCO'.
WRITE 'Basic List'.

AT USER-COMMAND.
  CASE SY-UCOMM.
    WHEN 'MYCO'.
      WRITE 'Secondary List from USER-COMMAND,'.
      WRITE: 'SY-LSIND', SY-LSIND.
      SET USER-COMMAND 'PF05'.
  ENDCASE.

AT PF05.
  WRITE 'Secondary List from PF05,'.
  WRITE: 'SY-LSIND', SY-LSIND.
  SET CURSOR LINE 1.
  SET USER-COMMAND 'PICK'.

AT LINE-SELECTION.
  WRITE 'Secondary List from LINE-SELECTION,'.
  WRITE: 'SY-LSIND', SY-LSIND.
  SET USER-COMMAND 'PS+10'.
```

此程序创建 一个基本列 表和三个次 列表。执行 该程序之后 ，系统立即 显示第三个次列表，并 向右滚动十 列：

```
List from LINE-SELECTION, SY-LSIND    3
```

要滚动，该 程序使用系 统定义的功 能代码 PS+10。 要查看其他 列表，用户 选择“返回 ”。

```
级 2:

Secondary List from PF05, SY-LSIND    2

级 1:
```

```
Secondary List from USER-COMMAND, SY-LSIND    2
级 0:
Basic List
```

请注意，在 事件 AT PF05 中，使用 SET CURSOR 语句在列表 行上定位光 标，以支持 功能代码 PICK。 关于如何在 程序中设置 光标的详细 信息，参见 *从程序中设 置光标* (页 300) 。

将数据从列 表传递到报 表

要有效地使 用交互式报 表的交互式 列表，程序 只对用户在 输出列表上 触发的事件 作出反应还 不够。还必 须能够解释 用户选择的 行及其内容 。
为此，使用 交互式列表 传递到程序 的信息。ABAP/4 提供三种传 递数据的方 法：

使用系统 字段自动传 递数据
使用自动数 据主要传送 辅助数据， 可以使用这 些数据更好 地定位用户 行为。
在程序中 使用语句获 取数据
使用程序控 制的数据传 送将单个输 出字段的内 容转移到交 互式事件的 处理块中， 并继续处理 这些值。
传递列表 属性
如果创建列 表时没有保 存列表级的 某些属性， 例如每页的 页号和行号 ，则可以稍 后使用 DESCRIBE LIST 语句检索这 些数据。

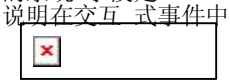
下列主题说 明这些可能 性：

自动传递数 据

自动数据传 送依靠每个 交互式事件 系统填写的 系统字段发 生。关于相 关系统字段 的概述，参 见 *次列表系统 字段* (页 320) 。
后面的主题 描述

交互式列表 系统字段中 的数据

主要从系统 字段中检索 下列信息： 列表的索引 、输出窗口 中列表的位 置和光标的 位置。唯一 包含选定行 内容的系统 字段是 SY-LISEL。
下面的示例 说明在交互 式事件中， 系统如何填 写这些系统 字段。



```
REPORT SAPMZTST NO STANDARD PAGE HEADING
                LINE-COUNT 12 LINE-SIZE 40.

DATA: L TYPE I, T TYPE C.

DO 100 TIMES.
 WRITE: / 'Loop Pass:', SY-INDEX.
ENDDO.

TOP-OF-PAGE.

WRITE: 'Basic List, Page', SY-PAGNO.
ULINE.

TOP-OF-PAGE DURING LINE-SELECTION.

WRITE 'Secondary List'.
ULINE.

AT LINE-SELECTION.

DESCRIBE FIELD SY-LISEL LENGTH L TYPE T.

WRITE: 'SY-LSIND:', SY-LSIND,
     / 'SY-LISTI:', SY-LISTI,
     / 'SY-LILLI:', SY-LILLI,
     / 'SY-CUROW:', SY-CUROW,
     / 'SY-CUCOL:', SY-CUCOL,
     / 'SY-CPAGE:', SY-CPAGE,
     / 'SY-STARO:', SY-STARO,
     / 'SY-LISEL:', 'Length =', L, 'Type =', T,
     /  SY-LISEL.
```

此程序创建 有十页的列 表。执行该 程序之后，用户应该滚 动列表，并 按下列显示 定位光标：

选择“选择 ”或 **F2 之后**，次列 表如下所示 ：

SY-LSIND 是当前列表 的索引，SY-LISTI 是上一个列 表的索引。 SY-LILLI 是绝对列表 中选定行的 行号（前面 页的九个十 二行加上当 前页的九行 ），SY-CUROW 是选定行在 屏幕上的位 置。SY-CUCOL 是光标在窗 口中的位置 。该位置比 相应未滚动 列表列多一 列。SY-CPAGE 是列表的当 前显示页。 SY-STARO 是显示在当 前页的最顶 端的实际列 表行的行号 。这不包括 页眉。SY-CPAGE 和 SY-STARO 不取决于光 标位置。对于 SY-LISEL， 程序将显示 长度、数据 类型和内容 。SY-LISEL 的长度总是 255，而 与列表的宽 度无关。

使用 SY-LISEL

SY-LISEL 系统字段是 C 类型的字段 ，长度为 255 个字符。它 包含选定的 行，并将其 当作一简单 字符串，因 此使得检索 单个字段的 值变得很困 难。要处理 SY-LISEL 的某些部分 ，必须指定 相应偏移量 （参见 *指女 流 象的偏移量* (页 **错误！链接无效。**))。
由于上面提 到的原因， 从字段中传 送值时，请 不要使用 SY-LISEL。*通过程序语 句传递数据* (页 297) 中描述的方 法将更加合 适。但是， 要格式化次 列表的表头 行或检查选 定的行是否 既不为空， 也没有用底 线填充，SY-LISEL 将提供解决 方法。

```
PROGRAM SAPMZTST NO STANDARD PAGE HEADING.

DATA NUM TYPE I.

SKIP.
WRITE 'List of Quadratic Numbers between One and Hundred'.
SKIP.
WRITE 'List of Cubic Numbers between One and Hundred'.

TOP-OF-PAGE.

  WRITE 'Choose a line!'.
  ULINE.

TOP-OF-PAGE DURING LINE-SELECTION.

  WRITE SY-LISEL.
  ULINE.

AT LINE-SELECTION.

  IF SY-LISEL(4) = 'List'.
    CASE SY-LILLI.
      WHEN 4.
        DO 100 TIMES.
          NUM = SY-INDEX ** 2.
          WRITE: / SY-INDEX, NUM.
        ENDDO.
      WHEN 6.
        DO 100 TIMES.
          NUM = SY-INDEX ** 3.
          WRITE: / SY-INDEX, NUM.
        ENDDO.
    ENDCASE.
  ENDIF.
```

执行该程序 之后，系统 显示下列输 出屏幕：

例如，如果 用户选择较 高的行，则 将出现下列 次列表。

选择基本列 表上的较低 行时，则将 出现立方数 的列表。

选定行的内 容成为次列 表的表头。 IF 语句使用 SY-LISEL 的头四个字 符，以保证 只选择有效 行。由于 AT LINE-SELECTION 后面的处理 块逻辑条件 的缘故，次 列表上的行 选择不会生 成任何进一 步的输出。

该示例还强 调使用基本 列表中的压 缩信息显示 次列表中的 相关详细信 息的概念。

通过程序语 句传递数据

要在交互式 事件中将单 个输出字段 或附加信息 从行传递到 相应处理块 ，请使用这 些语句：

HIDE
HIDE 语句是交互 式报表的基 本语句之一 。使用 HIDE 技术，可以 在创建列表 级时定义， 哪些信息将 随后传递到 后面的次列 表。

READ LINE
使用语句 READ LINE 和 READ CURRENT LINE 从现有的列 表级的行中 显式地读取 数据。这些 语句与 HIDE 是紧密相连 的。

GET CURSOR
使用语句 GET CURSOR FIELD 和 GET CURSOR LINE 将输出字段 或交互式事 件期间光标 所在的输出 行传递到处 理块。

下列主题描 述这些语句 ：

HIDE 技术

创建列表级 时使用 HIDE 技术存储特 定行的信息 ，以备后用 。

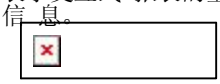
语法
HIDE <f>.
该语句在所 谓的 HIDE 区域中内部 地存储与当 前输出行（ 系统字段 SY-LINNO） 相关的变量〈f〉的内容。变 量〈f〉不必出现在 当前行上。

要提高程序 的可读性， 请始终将 HIDE 语句直接放 在变量〈f〉的输出语句 之后，或放 在当前行的 最后一个输 出语句之后 。要隐藏好 几个变量， 请将这几个 HIDE 语句连接起 来（参见*两酉嗨朴 句*（页 **错误！链接无效。**））。

一旦用户选 择了存储 HIDE 字段的行， 系统立即用 存储的值填 充程序中的 变量。可以 通过下列方 式选择行：

通过交互 式事件。
对每个交互 式事件，在 事件中用存 储的值填充 光标所在行 上的 HIDE 字段。
通过 READ LINE 语句。
参见 *从列表中读 取行*（页 275）。

可以将 HIDE 区域看作表 格，系统在 该表中为每 个列表和行 号存储所有 HIDE 字段的名称 和值。一旦需要它们， 系统就从表 格中读取值 。
下面的示例 表示交互式 报表的基本 特征。基本 列表包含总 结信息。通 过 HIDE 技术，每个 次列表包含 更详细的信 息。



下列报表被 连接到了逻 辑数据库 F1S。

```
PROGRAM SAPMZTST NO STANDARD PAGE HEADING.

TABLES: SPFLI, SBOOK.

DATA: NUM TYPE I,
      DAT TYPE D.

START-OF-SELECTION.

  NUM = 0.
  SET PF-STATUS 'FLIGHT'.

GET SPFLI.

  NUM = NUM + 1.
  WRITE: / SPFLI-CARRID, SPFLI-CONNID,
           SPFLI-CITYFROM, SPFLI-CITYTO.
  HIDE:    SPFLI-CARRID, SPFLI-CONNID, NUM.

END-OF-SELECTION.

  CLEAR NUM.
```

```
TOP-OF-PAGE.

  WRITE 'List of Flights'.
  ULINE.
  WRITE 'CA  CONN FROM                TO'.
  ULINE.

TOP-OF-PAGE DURING LINE-SELECTION.

  CASE SY-PFKEY.
    WHEN 'BOOKING'.
      WRITE SY-LISEL.
      ULINE.
    WHEN 'WIND'.
      WRITE:  'Booking', SBOOK-BOOKID,
           /  'Date   ', SBOOK-FLDATE.
      ULINE.
  ENDCASE.

AT USER-COMMAND.

  CASE SY-UCOMM.
    WHEN 'SELE'.
      IF NUM NE 0.
        SET PF-STATUS 'BOOKING'.
        CLEAR DAT.
        SELECT * FROM SBOOK WHERE CARRID = SPFLI-CARRID
                            AND   CONNID = SPFLI-CONNID.
          IF SBOOK-FLDATE NE DAT.
            DAT = SBOOK-FLDATE.
            SKIP.
            WRITE / SBOOK-FLDATE.
            POSITION 16.
          ELSE.
            NEW-LINE.
            POSITION 16.
          ENDIF.
          WRITE SBOOK-BOOKID.
          HIDE: SBOOK-BOOKID, SBOOK-FLDATE, SBOOK-CUSTTYPE,
                SBOOK-SMOKER, SBOOK-LUGGWEIGHT, SBOOK-CLASS.
        ENDSELECT.
        IF SY-SUBRC NE 0.
          WRITE / 'No bookings for this flight'.
        ENDIF.
        NUM = 0.
        CLEAR SBOOK-BOOKID.
      ENDIF.
    WHEN 'INFO'.
      IF NOT SBOOK-BOOKID IS INITIAL.
        SET PF-STATUS 'WIND'.
        SET TITLEBAR 'BKI'.
        WINDOW STARTING AT 30 5 ENDING AT  60 10.
        WRITE: 'Customer type   :', SBOOK-CUSTTYPE,
             / 'Smoker          :', SBOOK-SMOKER,
             / 'Luggage weigtht :', SBOOK-LUGGWEIGHT,
             / 'Class           :', SBOOK-CLASS.
      ENDIF.
  ENDCASE.
```

在事件 START-OF-SELECTION 中，系统为 基本列表设 置状态 FLIGHT。 在状态 FLIGHT 中，功能代 码 SELE（ 文本 SELECT） 分配给功能 键 **F2**，并与 按钮。因此 ，双击鼠标 、按下 **F2** 和单击按钮 “SELECT” 等用户事件 都将触发 AT USER-COMMAND 事件。

基本列表如 下所示：

创建基本列 表时，将 SPFLI-CARRID、 SPFLI-CONNID 和 NUM 三个字段存 储在 HIDE 区域中。选 择行后，系 统将显示在 AT USER-COMMAND 事件中为功 能代码 SELE 定义的次列 表。在 AT USER-COMMAND 事件中，系 统重新填写 存储在 HIDE 区域内的选 定行的所有 字段。使用 NUM 检查用户是 否从实际列 表中选择了 行。次列表 具有状态 BOOKING， 该状态中将 **F2** 分配到了功 能代码 INFO（ 文本：帐面 信息）。次 列表表示程 序通过基本 列表的 HIDE 字段选择的 数据。对每 一个显示的 列表行，系 统在 HIDE 区域内存储 附加信息。

如果用户选 择次列表的 行，系统将 在对话窗口 中使用状态 WIND 显示“隐藏 ”信息。对 状态 WIND， 接受了状态 类型“对话 框中的列表 ”的建议值 。程序使用 SBOOK-BOOKID 检查用户是 否选择了有 效行。例如 ，从那里创 建的第一个 次列表和另 一个次列表 显示如下：

程序本身设 置对话窗口 的所有页眉 和标题栏。

从列表中读 取行

系统内在地 存储一个报 表程序连续 创建的所有 列表。因此 ，可以从由 当前会话创 建的程序中 访问任何列 表，并且没 有通过返回 到前一列表 级而删除它 （参见 *维护列表* (页 319))。要读取 行，请使用 语句 READ LINE 和 READ CURRENT LINE。
要在出现交 互式列表事 件之后读取 行，请使用 READ LINE 语句：

语法
```
READ LINE <lin> [INDEX <idx>]
                [FIELD VALUE <F1> [INTO <g1>] ... <fn> [INTO <gn>]]
                [OF CURRENT PAGE|OF PAGE <p>].
```
没有任何选 项的语句存 储列表中行 <lin> 的内容，在 该列表上用 SY-LISEL 系统字段触 发了事件 (索引 SY-LILLI) ，并把为此 行存储的 HIDE 信息填写回 相应字段中 （参见 *HIDE 技术* (页 334))。至于 SY-LISEL 和 HIDE 区域，READ LINE 与交互式行 选择具有相 同的效果。
如果选定的 行 <lin> 存在，则系 统将 SY-SUBRC 设置为 0，否则， 将它设置为 4。
选项有下列 作用：

INDEX <idx>
系统从级 <idx> 的列表中读 取行的信息 。

FIELD VALUE <F1> [INTO <g_1>] ... <f_n> [INTO <g_n>]
系统将行 <lin> 中变量 <f_i> 的输出值解 释为字符串 ，并将它们 放到同一字 段 <f_i> 中，或者使 用 INTO 时，将它们 放到字段 <g_i> 中。重新填 写字段时， 系统将应用 转换规则 (参见*源类型字符* （页 **错误！链接无效。**)) 。
未出现在行 中的字段不 影响目标字 段。如果字 段在行中出 现了好几次 ，则系统只 使用第一个 字段。
系统使用输 出格式，即 包括所有格 式化字符， 传输字段内 容。这可能 导致无法交 换，例如将 编辑字符转 换为十进制 字符或其它 字符。

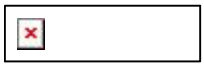
由于这种情 况下无法使 用 HIDE 技术，所以 主要使用该 选项处理接 受输入的列 表字段中的 用户输入。

OF CURRENT PAGE
对于此选项 ，<lin> 不是整个列 表的行号， 而是定址列 表当前显示 页的行号。 系统字段 SY-CPAGE 存储相应页 号。

OF PAGE <p>
对于此选项 ，<lin> 不是整个列 表的行号， 而是定址列 表 <p> 页上的行号 。

取决于读取 行的内容， 可能希望从 同一行中重 新填写字段 。为此，请 在程序中使 用 READ CURRENT LINE 语句。此语 句连续两次 读行：

语法
```
READ CURRENT LINE [FIELD VALUE <F1> [INTO <g1>] ...].
```
此语句通过 交互式事件 (F2) 或通过 READ LINE 读取前面读 取过的行。 FIELD VALUE 选项与 READ LINE 的相同。



下列报表连 接到逻辑数 据库 F1S。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING.

TABLES: SFLIGHT.

DATA: BOX, LINES TYPE I, FREE TYPE I.

START-OF-SELECTION.

  SET PF-STATUS 'CHECK'.

GET SFLIGHT.

WRITE: BOX AS CHECKBOX, SFLIGHT-FLDATE.
HIDE: SFLIGHT-FLDATE, SFLIGHT-CARRID, SFLIGHT-CONNID,
      SFLIGHT-SEATSMAX, SFLIGHT-SEATSOCC.
```

```
END-OF-SELECTION.

  LINES = SY-LINNO - 1.

TOP-OF-PAGE.

  WRITE: 'List of Flights from',
         (12) CITY_FR, 'to', CITY_TO.
  ULINE.

TOP-OF-PAGE DURING LINE-SELECTION.

  WRITE:  'Date:', SFLIGHT-FLDATE.
  ULINE.

AT USER-COMMAND.

  CASE SY-UCOMM.
    WHEN 'READ'.
      BOX = SPACE.
      SET PF-STATUS 'CHECK' EXCLUDING 'READ'.
      DO LINES TIMES.
        READ LINE SY-INDEX FIELD VALUE BOX.
        IF BOX = 'X'.
          FREE = SFLIGHT-SEATSMAX - SFLIGHT-SEATSOCC.
          IF FREE > 0.
            NEW-PAGE.
            WRITE: 'Company:', SFLIGHT-CARRID,
                   'Connection: ',SFLIGHT-CONNID,
                 / 'Number of free seats:', FREE.
          ENDIF.
        ENDIF.
      ENDDO.
  ENDCASE.
```

如果在执行 报表之后， 用户在如下 选择屏幕上 填写离开和 到达机场

则系统将显 示输出屏幕 ，用户可以 在该屏幕上 如下所示填 写复选框：

自定义页眉 使用参数 CITY_FR 和 CITY_TO。 它们是在逻 辑数据库 F1S 中定义的。 在主题 *INITIALIZATION* (页 **错误！链接无效。**).的示例中 介绍了如何 使用 F1 和“技术信 息”查找参 数名。

程序使用状 态 CHECK， 这里，将自 定义功能代 码 READ（ 文本“读取 行）分配到 功能键 F5 和应用程序 工具栏中的 按钮。相应 的用户行为 触发 AT USER-COMMAND 事件。现在 ，系统使用 DO 循环中的 READ LINE 读取基本列 表的所有行 。对每行， 它使用前面 存储在 HIDE 区域内的所 有值填写相 应字段。通 过 FIELD VALUE 选项，系统 还读取复选 框 BOX。如 果仍有空座 ，则系统将 公司、连接 和空闲座位 号写入在复 选框中做出 标记的每行 的次列表。

系统为每个 输出启动具 有单独页眉 的新页。在 次列表上， 由于 SET PF-STATUS 语句的 EXCLUDING 选项使功能 代码 READ 处于非活动 状态，所以 不支持任何 自定义用户 行为。

从次列表返 回之后，复 选框仍然是 选中的。*修改字段格 式* (页 301) 中的示例显 示如何清除 它们。

在光标位置 处读列表

要在交互式 事件中检索 光标位置处 的信息，请 使用 GET CURSOR 语句引用字 段或行。
关于该字段 的信息，请 使用语法：

语法

```
GET CURSOR FIELD <f> [OFFSET <off>] [LINE <lin>]
                    [VALUE <val>] [LENGTH <len>].
```

在用户行为 中，该语句 将光标位置 处的字段名 传送到变量 <f> 中。如果光 标在字段上 ，则系统将 SY-SUBRC 设置为 0，否则， 将其设置为 4。
系统传送子 程序的全局 变量、常量 、字段符号 或引用参数 的名称。对 子程序的文 字、局部字 段和 VALUE 参数，系统 将 SY-SUBRC 设置为 0，但将 SPACE 作为名称传 输。

选项有下列 作用：
OFFSET <off>
字段 <off> 包含光标在 字段中的位 置。如果光 标在第一列 上，则 <off> = 0。
LINE <lin>
字段 <lin> 包含光标所 在列表行的 行号 (SY-LILLI)。
VALUE <val>
字段 <val> 包含光标所 在字段的字 符串输出表 示法。表示 法包括格式 化字符。
LENGTH <len>
字段 <len> 包含光标所 在字段的输 出长度。
关于行的信 息，请使用 语法：

语法
GET CURSOR LINE <lin> [OFFSET <off>] [VALUE <val>] [LENGTH <len>].
此语句将用 户行为期间 光标所在行 的行号传输 到变量 <lin> 中。如果光 标在列表行 上，则系统 将 SY-SUBRC 设置为 0，否则， 将其设置为 4。可以使 用此语句避 免用户选择 无效行。
选项有下列 作用：
OFFSET <off>
字段 <off> 包含光标在 列表行中的 位置。如果 光标在第一 列上，则 <off> = 0。
VALUE <val>
字段 <val> 包含光标所 在行的字符 串输出表示 法。表示法 包括格式化 字符。
LENGTH <len>
字段 <len> 包含光标所 在行的输出 长度。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING LINE-SIZE 40.

DATA: HOTSPOT(10) VALUE 'Click me!',
      F(10), OFF TYPE I, LIN TYPE I, VAL(40), LEN TYPE I.

FIELD-SYMBOLS <FS>.
ASSIGN HOTSPOT TO <FS>.
WRITE 'Demonstration of GET CURSOR statement'.
SKIP TO LINE 4.
POSITION 20.
WRITE <FS> HOTSPOT COLOR 5 INVERSE ON.

AT LINE-SELECTION.

  WINDOW STARTING AT 5 6 ENDING AT 45 20.
  GET CURSOR FIELD F OFFSET OFF
             LINE LIN VALUE VAL LENGTH LEN.
  WRITE: 'Result of GET CURSOR FIELD: '.
  ULINE AT /(28).
  WRITE: / 'Field: ', F,
         / 'Offset:', OFF,
         / 'Line:  ', LIN,
         / 'Value: ', (10) VAL,
         / 'Length:', LEN.
  SKIP.
  GET CURSOR LINE LIN OFFSET OFF VALUE VAL LENGTH LEN.
  WRITE: 'Result of GET CURSOR LINE: '.
  ULINE AT /(27).
  WRITE: / 'Offset:', OFF,
         / 'Value: ', VAL,
         / 'Length:', LEN.
```

在此程序中 ，将 HOTSPOT 字段分配到 字段符号 <FS>， 并在输出屏 幕上显示为 热点。如果 用户将光标 定位在热点 的字母 ‘C’ 上，并选择 该行，则将 出现下列对 话窗口：

系统在 AT LINE-SELECTION 事件中输出 GET CURSOR 语句的结果 。注意，在 GET CURSOR FIELD 之后，把分 配到字段符 号 <FS> 的字段的名 称存储在 F 中，而不是 字段符号的 名称。

传递列表属 性

处理列表级 时，如果需 要知道创建 列表时忘记 存储在变量 中的属性， 或者如果使 用另一个报 表创建的列 表级，则请 使用 DESCRIBE LIST 语句。
要检索列表 的行数或列 表的页数， 请使用：

语法
DESCRIBE LIST NUMBER OF LINES|PAGES <n> [INDEX <idx>].

此语句将列 表级 <idx> 的行数或页 数写入变量 <n> 中。如果索 引为 <idx> 的列表不存 在，则系统 将 SY-SUBRC 设置为非 0，否则， 将其设置为 0。
要检索特定 行号的页码 ，请使用：

语法
DESCRIBE LIST LINE <lin> PAGE <pag> [INDEX <idx>].
此语句将列 表行号 <lin> 所在的列表 级 <idx> 的页号写入 变量 <pag> 中。系统按 照下列情况 设置 SY-SUBRC： 如果索引为 <idx> 的列表不存 在，则设置 为 8；如果行 号为 <lin> 的行不存在 ，则设置为 4；否则， 设置为 0。
要检索特定 页的属性， 请使用：

语法
DESCRIBE LIST PAGE <pag> [INDEX <idx>] [<options>]
此语句检索 列表级 <idx> 的属性，该 属性是在页 <pag> 的 <options> 中指定的。 系统按照下 列情况设置 SY-SUBRC： 如果索引为 <idx> 的列表不存 在，则设置 为 8；如果页 号为 <pag> 的页不存在 ，则设置为 4；否则， 设置为 0。
该语句的 <options> 是：

LINE-SIZE <col>
将页宽度写 入变量 <col> 中。
LINE-COUNT <len>
将页长度写 入变量 <len> 中。
LINES <lin>
将显示行的 行号写入变 量 <lin> 中。
FIRST-LINE <lin1>
将第一行的 绝对行号写 入变量 <lin1> 中。
TOP-LINES <top>
将页眉行的 行号写入变 量 <top> 中。
TITLE-LINES <tit>
将标准页眉 表头的行号 写入变量 <tit> 中。
HEAD-LINES <head>
将标准页眉 表头的列号 写入变量 <head> 中。
END-LINES <end>
将页脚行行 号写入变量 <end> 中。

只对完成的 列表才使用 DESCRIBE LIST， 这是因为， 对正创建的 列表（索引 是 SY-LSIND） ，某些属性 不是最新的 。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING
                LINE-SIZE 40 LINE-COUNT 5(1).

DATA: LIN TYPE I, PAG TYPE I,
      COL TYPE I, LEN TYPE I, LIN1 TYPE I,
      TOP TYPE I, TIT TYPE I, HEAD TYPE I, END TYPE I.

DO 4 TIMES.
  WRITE / SY-INDEX.
ENDDO.

TOP-OF-PAGE.
  WRITE 'Demonstration of DESCRIBE LIST statement'.
  ULINE.

END-OF-PAGE.
  ULINE.

AT LINE-SELECTION.
  NEW-PAGE LINE-COUNT 0.
  WINDOW STARTING AT 1 13 ENDING AT 40 25.
  DESCRIBE LIST: NUMBER OF LINES LIN INDEX 0,
                 NUMBER OF PAGES PAG INDEX 0.
  WRITE: 'Results of DESCRIBE LIST: '.
  ULINE AT /(25).
  WRITE: / 'Lines: ', LIN,
         / 'Pages: ', PAG.
  SKIP.
  DESCRIBE LIST LINE SY-LILLI PAGE PAG INDEX 0.
  WRITE: / 'Line', (1) SY-LILLI, 'is on page', (1) PAG.
  SKIP.
  DESCRIBE LIST PAGE PAG INDEX 0 LINE-SIZE   COL
                                 LINE-COUNT  LEN
```

```
                                LINES       LIN
                                FIRST-LINE  LIN1
                                TOP-LINES   TOP
                                TITLE-LINES TIT
                                HEAD-LINES  HEAD
                                END-LINES   END.
  WRITE: 'Properties of Page', (1) PAG, ':',
       / 'Width:       ',  COL,
       / 'Length:      ',  LEN,
       / 'Lines:       ',  LIN,
       / 'First Line:  ',  LIN1,
       / 'Page Header: ',  TOP,
       / 'Title Lines: ',  TIT,
       / 'Header Lines:',  HEAD,
       / 'Footer Lines:',  END.
```

此程序创建 两页列表，每页有五行 。两行用作 自定义页眉 ，一行用作 页脚。如果 用户在第二 页上选择数 字 3，则出现 下列对话窗 口：

创建次列表 时，DESCRIBE LIST 的所有变体 适用于基本 列表。系统 在对话窗口 中显示结果 。注意，使 用 SY-LILLI 交互定义要 说明的行和 页。

使用交互式 列表

此主题说明 如何使用交 互式列表的 表示法和属 性。您将了 解到

滚动交互式 列表

要从程序中 滚动到交互 式列表，请 使用 SCROLL 语句。在 *在程序之内 滚* （页 **错误！链接无效。**）中详细说明 如何在基本 列表中使用 此语句。

要在交互式 列表中使用 SCROLL 语句，应清 楚下列项：

只有对完 成的列表才 能使用 SCROLL 语句。如果 在列表的第 一个输出语 句之前使用 SCROLL，则它不影响 此列表。如 果在列表的 第一个输出 语句之后使 用 SCROLL， 则它将影响 整个列表， 即包括后面 的所有输出 语句。

创建次列 表时，没有 INDEX 选项的 SCROLL 语句总是引 用前面显示 的交互式事 件发生的列 表（索引 SY-LISTI） 。

只有创建 基本列表时 ，SCROLL 语句才引用 当前创建的 列表。

可以使用 INDEX 选项显式地 滚动现有的 列表级。为 此，不需要 显示列表。 用户再次显 示该列表时 ，将把它滚 动到指定位 置。如果指 定的列表级 不存在，则 系统把 SY-SUBRC 设置为 8。

在交互式 事件中，如 果要滚动当 前创建的列 表，请在 SCROLL 语句中将 SY-LSIND 用作索引。

注意，SY-LSIND 只在已创建 列表的事件 的最后才有 效，而与语 句在处理块 中的位置无 关。

如果希 望显式地设 置列表级， 必须在处理 块的最后语 句中使用 SY-LSIND， 以确保处理 块内的 SCROLL 语句访问正 确的列表。

从程序中滚 动交互式列 表的另一种 方法是使用 与相应的系 统定义功能 代码 (P...) 连接的 SET USER-COMMAND 语句。详细 信息，参见 *从程序中触 发事件* （页 331）。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING LINE-SIZE 50.

SET PF-STATUS 'SELECT'.

WRITE 'Create a secondary list by choosing SELECT'.

AT USER-COMMAND.

  NEW-PAGE LINE-SIZE 200.
  CASE SY-UCOMM.
    WHEN 'SELE'.
      SET PF-STATUS 'SCROLLING'.
      DO 200 TIMES. WRITE SY-INDEX. ENDDO.
      SCROLL LIST RIGHT BY 48 PLACES INDEX SY-LSIND.
      SY-LSIND = SY-LSIND - 1.
    WHEN 'LEFT'.
      SCROLL LIST LEFT BY 12 PLACES.
    WHEN 'RGHT'.
      SCROLL LIST RIGHT BY 12 PLACES.
  ENDCASE.
```

此程序创建 一行的基本 列表，其状 态为 SELECT。 在状态 SELECT 中，将功能 代码 SELE（ 文本 SELECT） 分配到功能 键 **F2** 和应用程序 工具栏的按 钮。

选择“选择 ”之后，系 统将触发 AT USER-COMMAND 事件并创建 状态为 SCROLLING 的次列表。在状态 SCROLLING 中,将功能 代码 LEFT(文本 LEFT）和 RGTH（文本 RIGHT）分配到功能 键 F5 和 F6 以及应用程 序工具栏。 次列表的宽 度为 200 字符。创建 次列表之后 (SY-LSIND = 1)，SCROLL 语句把次列 表向右滚动 48 列。然后， SY-LSIND 减一，并用 滚动后的列 表替代基本 列表。

通过单击“ 左”和“右 ”，用户可 以在显示的 列表中向左 或向右滚动 。在 USER-COMMAND 事件中，为 相应功能代 码编写 SCROLL 语句。

从程序中设 置光标

可以从程序 中动态地设 置当前列表 上的光标。 这样做可以 支持用户将 值输入到输 入字段中， 或者选择字 段或行。如 果输入字段 出现在列表 上，默认情 况下，系统 将光标放置 于第一个输 入字段中。要设置光标 ，请使用 SET CURSOR 语句。该语 句在最近创 建的列表中 设置光标， 创建基本列 表时，总是 基本列表本 身。创建次 列表时，就 是前一个列 表。
可以使用 SET CURSOR 将光标设置 到某绝对位 置、字段或 行。

显式地设置 光标

要在输出窗 口中将光标 设置到特定 位置，请使 用：

语法
SET CURSOR <col> <lin>.
此语句将光 标设置到输 出窗口的 <col> 列和 <lin> 行。
系统只将光 标设置到显 示中可见的 位置。对显 示区域之外 的位置，系 统将忽略该 语句。要将 光标设置到 当前不可见 的列表中， 必须首先滚 动该列表（ 参见 *滚动交互式 列表* (页 340))。
可以只将光 标设置到包 含列表输出 的行,包括 用 SKIP 语句跳过的 行,但不包 括下划线。如果 <lin> 在底部列表 行的下面， 则系统将光 标设置到底 行。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING LINE-SIZE 80.

SET PF-STATUS 'SCROLLING'.

NEW-LINE NO-SCROLLING.
WRITE 'Input Fields:'.
NEW-LINE NO-SCROLLING.
WRITE '-------------'.

NEW-LINE.
DO 5 TIMES.
  WRITE:  ' Input', (1) SY-INDEX, '   ' INPUT ON NO-GAP.
ENDDO.

FORMAT INPUT OFF.
SET CURSOR 11 3.

AT USER-COMMAND.

CASE SY-UCOMM.
    WHEN 'LEFT'.
      IF SY-STACO > 1.
        SCROLL LIST LEFT BY 12 PLACES.
      ENDIF.
    WHEN 'RGHT'.
      IF SY-STACO < 40.
        SCROLL LIST RIGHT BY 12 PLACES.
      ENDIF.
  ENDCASE.
  SET CURSOR 11 3.
```

此程序创建 包含五个输 入字段的基 本列表。状 态 SCROLLING 与在 *滚动交互式 列表* (页 340) 中的示例中 定义的状态 相同。光标 设置到第一 个输入字段 。用户可以 使用按钮“ 左”和“右 ”水平地滚 动列表。每 次滚动之后 ，又将光标 设置到输入 字段。例如 ，单击“右 ”两次之后 ，输出屏幕 显示如下：

将光标设置 到字段

要将光标设 置到显示列 表行的特定 字段上，请 使用：

语法
```
SET CURSOR FIELD <f> LINE <lin> [OFFSET <off>].
```
此语句将 <lin> 行上的光标 设置到名称 存储在 <f> 中的字段上 。如果字段 在行上不止 出现一次， 则系统将光 标设置到第 一个字段上 。如果字段 没有出现在 行上，或者 如果它在显 示区域之外 ，则系统将 忽略此语句 。在最后这 种情况中， 使用 SCROLL 语句将该行 滚动到窗口 的可见区域 （参见 *滚动交互式 列表* (页 340) ）。
使用 OFFSET 选项将光标 设置到存储 在 <f> 中的字段的 <off> 位置。<off> = 0 表示第一位 置。

```
REPORT SAPMZTST LINE-SIZE 40.

DATA: INPUT1(5) VALUE '*****',
      INPUT2(5) VALUE '*****',
      INPUT3(5) VALUE '*****'.

SET PF-STATUS 'INPUT'.

WRITE   'Input Fields:'.
WRITE / '-------------'.
SKIP.

WRITE: 'Input 1', INPUT1 INPUT ON,
     / 'Input 2', INPUT2 INPUT ON,
     / 'Input 3', INPUT3 INPUT ON.

AT USER-COMMAND.

  CASE SY-UCOMM.
    WHEN 'INP1'.
      SET CURSOR FIELD 'INPUT1' LINE 6 OFFSET 4.
    WHEN 'INP2'.
      SET CURSOR FIELD 'INPUT2' LINE 7 OFFSET 4.
    WHEN 'INP3'.
      SET CURSOR FIELD 'INPUT3' LINE 8 OFFSET 4.
  ENDCASE.
```

此程序创建 包含三个输 入字段的基 本列表。在 状态 INPUT 中，将功能 代码 INP1、INP2 和 INP3 分配到功能 键 **F5**、**F6** 和 **F7** 以及应用程 序工具栏。 选择这些功 能之一后， 系统将触发 AT USER-COMMAND 事件。它将 光标置于相 应输入字段 上。如果用户选择“输 入 2”，则输 出屏幕显示 如下：

注意，定位 光标时，系 统包括表头 行。

将光标设置 到行

要将光标设 置到输出窗 口中列表的 特定行，请 使用：

语法
```
SET CURSOR LINE <lin> [OFFSET <off>].
```

此语句将光 标设置到行 <lin>。如果该行不 出现在列表 中或窗口的 可见区域内 ，则系统忽 略该语句。在后一种情 况中，请首 先使用 SCROLL 语句将该行 滚动到窗口 的可见区域 （参见 *滚动交互式 列表* (页 340) ）。
使用 OFFSET 选项将光标 设置到 <off> 列和 <lin> 行。<off> = 0 表示第一列 。如果该位 置在列表的可见区域之 外，则系统 忽略该语句 。

```
REPORT SAPMZTST LINE-SIZE 30 NO STANDARD PAGE HEADING.

DATA: INP(2), FLD(3), TOP(2).

SET PF-STATUS 'LINE_NUMBER'.

FLD = 'INP'.

DO 50 TIMES.
  WRITE: / 'Line ', (2) SY-INDEX.
ENDDO.

TOP-OF-PAGE.

  WRITE: 'Line number:', INP INPUT ON.
  ULINE.
  SKIP.

AT USER-COMMAND.
  DESCRIBE LIST PAGE 1 TOP-LINES TOP.
  CASE SY-UCOMM.
    WHEN 'LINE'.
      READ LINE 1 FIELD VALUE INP.
      SCROLL LIST TO PAGE 1 LINE INP.
      INP = INP + TOP.
      SET CURSOR LINE INP OFFSET 6.
  ENDCASE.
```

此程序创建 下列基本列 表：

在状态 LINE_NUMBER 中，将功能 代码 LINE（ 文本 LINE NUMBER） 分配到功能 键 F5 和应用程序 工具栏的按 钮。如果用 户将数字输 入到字段中 并选择“行 号”，则系 统将滚动指 定行号，并 将光标设置 到这个位置 ：

系统使用 READ LINE 读取输入字 段。为设置 光标，系统 使用 DESCRIBE LIST 考虑页眉的 大小。注意 ，此示例特 别使用了自 动类型转换 。

修改列表行

要从程序中 修改已完成 列表的行， 请使用 MODIFY LINE 语句。有两 种方法可以 指定要修改 的行：
可以显式 地指定要修 改的行：
语法

```
MODIFY LINE <n> [INDEX <idx>|OF CURRENT PAGE|OF PAGE <p>]
                [<modifications>].
```

如果没有第 一行的选项 ，此语句将 修改发生交 互式事件的 列表的 <n> 行（索引 SY-LISTI） 。
如果有第 一行的选项 ，则可以如 下指定修改 的行：
- 使用 INDEX <idx> 指定索引为 <idx> 的列表级的 行 <n>。
- 使用 OF CURRENT PAGE 指定当前显 示页的行 <n>（页 码 SY-CPAGE） 。
- 使用 OF PAGE <p> 指定页 <p> 的行 <n>。

可以引用 最近读取的 行：
语法

```
MODIFY CURRENT LINE [<modifications>].
```

此语句通过 行选择 (F2) 或 READ LINE 语句修改最 近读取的行 。
如果没有选 项 <modifications>， 则上面的语 句将 SY-LISEL 系统字段的 当前内容填 写入指定行 中。用相应 字段的当前 值改写行的 HIDE 区域。但是 ，这不影响 显示值。
如果系统成 功地修改了 指定的行， 则它把 SY-SUBRC 设置为 0，否则将 其设置为非 0 值。
除上所述， 选项 <modifications> 还包含其它 几种修改行 的可能性。 下列主题说 明这些可能 性：

修改行格式

要修改需要 更改格式的 行，请如下 使用 MODIFY 语句的 LINE FORMAT 选项：

语法
MODIFY ... LINE FORMAT <option$_1$> <option$_2$> ... .
此语句按照 用 <option$_i$> 指定的格式 设置整个已 修改行的输 出格式。可 以指定与 FORMAT 语句相同的 格式选项（ 参见 *FORMAT 语句*（页 **错误！链接无效。**））。

```
REPORT SAPMZTST LINE-SIZE 40 NO STANDARD PAGE HEADING.

DATA C TYPE I VALUE 1.

WRITE 'Select line to modify the background'.

AT LINE-SELECTION.

  IF C = 8.
     C = 0.
  ENDIF.

  MODIFY CURRENT LINE LINE FORMAT COLOR = C.
  ADD 1 TO C.
```

此程序创建 输出行，用 户可以通过 反复选择该 行而改变其 背景色：

修改字段内 容

对于要更改 的行中的字 段，要显式 地修改其内 容，请使用 MODIFY 语句的 FIELD VALUE 选项：

语法
MODIFY ... FIELD VALUE <F1> [FROM <g$_1$>] <F2> [FROM <g$_2$>] ... .
此语句用字 段 <f$_i$> 或 <g$_i$> 的当前内容 改写列表行 中字段 <f$_i$> 的内容。系 统没有在程 序中更改字段。如有必 要，系统将 字段类型转 换为类型 C。
如果字段 <f$_i$> 在列表行上 不止出现一 次，则系统 只在第一次 出现该字段 时修改它。 如果根本就 没有出现字 段 <f$_i$>， 则系统忽略 该选项。
系统修改现 有的字段 <f$_i$> 时，将不考 虑从 SY-LISEL 写入行中的 当前内容。 如果使用 SY-LISEL 在字段 <f$_i$> 的输出位置 处对行进行 修改，则 FIELD VALUE 选项将改写 它们。

```
REPORT SAPMZTST LINE-SIZE 40 NO STANDARD PAGE HEADING.

DATA C TYPE I.

WRITE: '   Number of selections:', (2) C.

AT LINE-SELECTION.

  ADD 1 TO C.
  SY-LISEL(2) = '**'.
  MODIFY CURRENT LINE FIELD VALUE C.
```

此程序创建 输出行，用 户可以通过 选择该行而 修改字段 C。同时， 由于 SY-LISEL 的修改，系 统将用两个 星号‘**’ 改写行的头 两个字符。 例如，四次 选择该行之 后，输出如 下所示：

修改字段格 式

对要更改的 行中的字段 ，要修改其 格式，请如 下使用 MODIFY 语句的 FIELD 选项：

语法
MODIFY ... FIELD FORMAT <F1> <options$_1$> <F2> <options$_2$> ... .
此语句按照 <options$_i$> 中指定的格 式设置出现 在行中的字 段 <f_i> 的输出格式 。与 <options$_i$> 一样，可以 指定 FORMAT 语句的一个 或多个格式 选项（参见 *FORMAT 语句* (页 **错误！链接无效。**))。
选项 FIELD FORMAT 改写相应字 段 LINE FORMAT 选项的规格 。如果字段 <f_i> 在行中多次 出现，则系 统只对第一 次做出修改 。如果字段 <f_i> 没有出现在 行中，则系 统将忽略此 选项。

此示例显示 评估复选框 后如何继续 处理。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING.

DATA: BOX, LINES TYPE I, NUM(1).

SET PF-STATUS 'CHECK'.

DO 5 TIMES.
  NUM = SY-INDEX.
  WRITE: / BOX AS CHECKBOX, 'Line', NUM.
  HIDE: BOX, NUM.
ENDDO.
LINES = SY-LINNO.

TOP-OF-PAGE.
  WRITE  'Select some checkboxes'.
  ULINE.

AT USER-COMMAND.
  CASE SY-UCOMM.
    WHEN 'READ'.
      SET PF-STATUS 'CHECK' EXCLUDING 'READ'.
      BOX = SPACE.
      DO LINES TIMES.
        READ LINE SY-INDEX FIELD VALUE BOX.
        IF BOX = 'X'.
          WRITE: / 'Line', NUM, 'was selected'.
          BOX = SPACE.
          MODIFY LINE SY-INDEX
                      FIELD VALUE  BOX
                      FIELD FORMAT BOX INPUT OFF
                                   NUM COLOR 6 INVERSE ON.
        ENDIF.
      ENDDO.
  ENDCASE.
```

此程序创建 状态为 CHECK 的基本列表 。在状态 CHECK 中，将功能 代码 READ（ 文本 Read Lines） 分配到功能 键 **F5** 和按钮。用 户可以标记 复选框，然 后选择“读 取行”。

在 AT USER-COMMAND 事件中，系 统使用 READ LINE 读取列表的 行。它继续 在次列表上 处理选定的 行。返回到 基本列表后 ，系统使用 MODIFY LINE 删除选定行 的复选框标 记，并将格 式 INPUT OFF 设置到复选 框。另外， 它还更改字 段 NUM 的格式。

现在，用户 只能标记还 没有更改的 行。

仅稍做改动 ，可以将此 程序中执行 行修改的代 码行包括到 *从列表中读 取行* (页 336) 中的示例程 序中。

调用程序

如果需要编 写扩展应用 程序，则一 个简单的报 表将变得很 复杂。要使 报表易读， 通常需要在 好几个程序 中分别实现 需要的功能 。
ABAP/4 允许使用这 些语句调用 报表及事务 ：

	报表	事务
无 返回调用	SUBMIT	LEAVE TO TRANSACTION
调 用并返回	SUBMIT AND RETURN	CALL TRANSACTION

可 以在任何 ABAP/4 程序中使用 这些语句。 要在对话编 程的环境中 调用其它程 序，参见 *调用外部程序组件* (页 **Error! Not a valid link.**) 。

此主题显示 如何从报表 中调用其它 程序。主要 在为交互式 报表处理交 互式列表时 使用这些调 用。用户可 以通过选择 列表行或提 供的功能显 式地调用其 它程序。可 以按照用户 行为编写传 输到调用程 序的数据。
该主题分为

调用报表

要从其它报 表中调用报 表，请使用 SUBMIT 语句。
要在程序编 码中静态地 设置调用程 序的名称， 请写：

语法
SUBMIT <rep> [AND RETURN] [<options>].
要在运行中 动态地设置 调用报表的 名称，请写 ：

语法
SUBMIT (<rep>) [AND RETURN] [<options>].
第一个语句 启动报表 <rep>， 第二个语句 启动名称存 储在字段 <rep> 中的报表。
执行程序时 ，如果系统 无法找到指 定的报表， 则出现运行 错误。如果 静态地指定 报表，可以 双击 ABAP/4 编辑器中的 <rep>， 以创建、显 示或修改调 用的报表。
如果忽略 AND RETURN 选项，则系 统将删除调 用报表的所 有数据和列 表级。终止 被调用报表 之后，它将 返回到启动 调用报表的 级别。
如果使用 AND RETURN， 系统将存储 调用报表的 数据，并在 处理完调用 的报表后返 回到调用报 表。系统用 紧接着调用 的语句恢复 执行调用报 表。要从程 序中通过 SUBMIT ... AND RETURN 退出被调用 的报表，参 见
任何使用 SUBMIT 的被调用报 表，都与 AND RETURN 选项无关， 都可以访问 调用报表的 ABAP/4 内存，该调 用报表包括 存储在 ABAP/4 内存中的数 据簇。关于 簇的详细信 息，参见 *ABAP/4 内存中的数 据簇*（页 **错误！链接无效。**）。
SUBMIT 语句为传递 数据和处理 调用的报表 提供了许多 其它 <options>。 下面描述了 部分选项。

从程序中退 出被调用的 报表

通常，用户 通过选择被 调用报表列 表级 0 上的 **F3** 或 **F15**，使 用 SUBMIT ... AND RETURN 退出调用的 报表。
但是，如果 想执行其它 语句，如将 数据输出到 ABAP/4 内存，则必 须在返回调 用的报表之 前，为被调 用报表创建 自定义用户 界面。在该 界面上，为 “返回”定 义自己的功 能代码，并 在 AT USER-COMMAND 事件中处理 它。在执行 所需的语句 后，使用此 语句离开被 调用报表：

语法
LEAVE.
此语句使用 SUBMIT ... AND RETURN 离开被调用 报表，并返 回到调用报 表中该调用 的语句之后 的语句。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING.

DATA: ITAB TYPE I OCCURS 10,
      NUM TYPE I.

SUBMIT SAPMZTS1 AND RETURN.

IMPORT ITAB FROM MEMORY ID 'HK'.

LOOP AT ITAB INTO NUM.
  WRITE / NUM.
ENDLOOP.

TOP-OF-PAGE.
WRITE 'Report 1'.
ULINE.
```

此程序调用 下列报表：

```
REPORT SAPMZTS1 NO STANDARD PAGE HEADING.

DATA: NUMBER TYPE I,
      ITAB TYPE I OCCURS 10.

SET PF-STATUS 'MYBACK'.

DO 5 TIMES.
  NUMBER = SY-INDEX.
  APPEND NUMBER TO ITAB.
  WRITE / NUMBER.
ENDDO.
```

```
TOP-OF-PAGE.
WRITE 'Report 2'.
ULINE.

AT USER-COMMAND.
  CASE SY-UCOMM.
    WHEN 'MBCK'.
      EXPORT ITAB TO MEMORY ID 'HK'.
      LEAVE.
  ENDCASE.
```

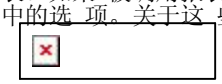
在自定义状 态 MYBACK 中，将功能 代码 MBCK 分配到功能 键 F3 和 F15：

如果用户在 界面 MYBACK 上选择“返 回”，则系 统将把表格 ITAB 传送到 ABAP/4 内存中，然 后离开 SAPMZTS1。 在 SAPMZTST 中，它又读 取表格 ITAB。

使用被调用 报表的列表 结构

要使用由 SUBMIT 调用的报表 的列表结构 ，请写：

语法
SUBMIT ... [LINE-SIZE <width>] [LINE-COUNT <length>].
如果被调用 报表不包含 REPORT 语句中这样 的选项，则 系统将按照 SUBMIT 语句中的选 项格式化被 调用报表的 列表。如果 被调用报表 的 REPORT 语句包含相 应的选项， 则系统将使 用这些选项 ，并忽略 SUBMIT 语句中的选 项。关于这 些选项的详 细信息，参 见 *自定义列表*（页 **错误！链接无效。**）。



```
REPORT SAPMZTST NO STANDARD PAGE HEADING.

DATA: NAME(8) VALUE 'SAPMZTS1',
              WID TYPE I VALUE 80,
              LEN TYPE I VALUE 0.

SET PF-STATUS 'SELECT'.

WRITE: 'Select a report and its list format:';
     / '-----------------------------------'.
SKIP.

WRITE: 'Report     ', NAME INPUT ON,
     / 'Line size  ', WID  INPUT ON,
     / 'Page length', LEN  INPUT ON.

AT USER-COMMAND.

  CASE SY-UCOMM.
    WHEN 'SELE'.
      READ LINE: 4 FIELD VALUE NAME,
                 5 FIELD VALUE WID,
                 6 FIELD VALUE LEN.
      SUBMIT (NAME) LINE-SIZE WID LINE-COUNT LEN AND RETURN.
  ENDCASE.
```

可以使用此 程序启动允 许用户定义 的列表格式 的报表。用 户可以在基 本列表上通过改写默认 值输入报表 名称和所需 的列表宽度 及长度：

在 AT USER-COMMAND 事件中，系 统读取这些 值并使用 SUBMIT 启动指定的 报表。如果 被调用报表 的 REPORT 语句不包含 LINE-SIZE 或 LINE-COUNT 规范，则系 统使用值 WID 和 LEN 以创建列表 。执行被调 用报表之后 ，用户可以 在基本列表 上更改输入 值，并调用 新报表。

填写被调用 报表的选择 屏幕

启动报表时 ，系统通常 显示选择屏 幕，用户在 该屏幕上输 入选择标准 和被连接逻 辑数据库和 报表本身的 参数（参见 *使用选择屏幕* （页 **错误！链接无效。**））。从另一 个报表中调 用报表时， 有好几种可 能的方法填 写选择标准 和被调用报 表的参数。
使用 SUBMIT 语句的下列 选项：

语法

```
SUBMIT ... [VIA SELECTION-SCREEN]
           [USING SELECTION-SET <var>]
           [WITH <sel> <criterion>]
           [WITH FREE SELECTIONS <freesel>]
           [WITH SELECTION-TABLE <rspar>].
```

这些选项有 下列作用：

VIA SELECTION-SCREEN
显示被调用 报表的选择 屏幕。如果 使用一个或 多个其他选 项将值传输 到报表中， 则填写选择 屏幕中的相 应输入字段 。用户可以 更改这些值 。默认情况 下，SUBMIT 之后系统不 显示选择屏 幕。

USING SELECTION-SET <var>
此选项告诉 系统使用变 体 <var> 启动被调用 程序（参见 *使用变式预设置选择* （页 **错误！链接无效。**））。

WITH <sel> <criterion>
使用此选项 填写选择屏 幕的单个元 素 <sel>（ 选择表和参 数）。使用 元素 <criterion> 之一：

- <op> <f> [SIGN <s>]， 用于单值选 择
  如果 <sel> 是选择标准 ，请用 <op> 填写被调用 的报表选择 表格 <sel> 的 OPTION 字段，用 <f> 填写 LOW 字段，用 <s> 填写 SIGN 字段（参见 *选择表* （页 **错误！链接无效。**））。
  如果 <sel> 是参数，则 可以使用 <op> 任何操作符 。总是用 <f> 填写参数 <sel>。
- [NOT] BETWEEN <**F1**> AND <**F2**> [SIGN <s>]， 用于间隔选 择
  将 <**F1**> 传送到被调 用报表中选 择表 <sel> 的 LOW 字段中，将 <**F2**> 传送到 HIGH 字段中，将 <s> 传送到 SIGN 字段中。如 果忽略 NOT 选项，则系 统将把值 BT 填写到 OPTION 字段中；如 果使用 NOT，则 系统将把值 NB 填写到 OPTION 字段中（参 见 *选择表* （页 **错误！链接无效。**））。
- IN <seltab>， 传送选择表
  用调用报表 中表格 <seltab> 的值填写被 调用报表中 的选择表 <sel>。 表格 <seltab> 必须有选择 表的结构。 可以使用 RANGES 语句创建这 样的表格（ 参见 *RANGES 语句* （页 **错误！链接无效。**））。

WITH FREE SELECTION <freesel>， 动态选择的 用户对话
要使用此选 项，必须将 调用报表和 被调用报表 都连接到支 持动态选择 的逻辑数据 库。在调用 报表中，使 用功能模块 FREE-SELECTIONS_INIT 和 FREE_SELECTIONS_DIALOG，它们允许用 户在选择屏 幕上输入动 态选择。这 些功能块的 一个输出参 数有 RSDS 类型组中的 RSDS_TEXPR 结构。通过 相同结构的 内部表 <freesel> 将此输出参 数的值传送 到被调用报 表。

WITH SELECTION-TABLE <rspar>， 值的动态传 送
首先，使用 词典结构 RSPARAMS 创建内部表 <rspar>。 该表格包含 下列六个字 段：

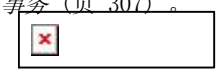
- SELNAME（ 类型 C，长度 8），选择 标准或参数 的名称
- KIND（ 类型 C，长度 1），选择 类型（S 是选择标准 ，P 是参数）
- SIGN、 OPTION、 LOW、HIGH， 作为常规选 择表（参见 *选择表* （页 **错误！链接无效。**）），除 LOW 和 HIGH 外，类型都 为 C、长度都 为 45。

在调用报表 内，可以动 态地用被调 用报表选择 屏幕所需的 任何值填写 此表。如果 选择标准的 名称不止出 现一次，则 在被调用报 表中，该标 准将占用选 择表的好几 行。如果参 数名不止出 现一次，则 系统使用最 后一个值。 注意，LOW 和 HIGH 的类型为 C，以便使 系统对被调 用报表的标 准执行类型 转换。例如 ，这对日期 字段是重要 的。因此， 请使用 VIA SELECTION-SCREEN 选项对程序 进行检查。

除 WITH SELECTION-TABLE 之外，可以 在 SUBMIT 语句内任意 地使用和组 合上述任何 选项。特别 是 WITH <sel> 选项，可以 在单个标准 <sel> 中多次使用 。在被调用 报表中，系 统将相应行 附加到使用的选择表中 。对于参数 ，它使用最 后指定的值 。WITH SELECTION-TABLE 选项唯一可 能的组合是 USING SELECTION-SET。

> 在交互式事 件中，可以 使用 SUBMIT 语句的上述 选项，用选 定行的 HIDE 区域中的数 据填写被调 用报表的选 择屏幕。这 允许绕过交 互式事件中 不可以使用 逻辑数据库 的限制。在 不同报表之 间分别使用 GET 语句，并且 在用户选定 行之后，使 用 SUBMIT 调用这些报 表并传递相 应值。

如果将选择 屏幕的输入 字段连接到 SPA/GPA 参数（参见 *从 SAP 内存中使用 缺省值* （页 **错误！链接无效。**），可以使 用 SPA/GPA 技术将数据 传送到选择 屏幕。关于 该技术的详 细信息，参 见 *将 SPA/GPA 参数传送到 事务* （页 307）。



> 下列报表创 建包含参数 PARAMET 和选择标准 SELECTO 的选择屏幕 ：

```
REPORT SAPMZTS1.
DATA NUMBER TYPE I.
PARAMETERS      PARAMET(14).
SELECT-OPTIONS  SELECTO FOR NUMBER.
```

下列报表使 用不同的选 择标准调用 报表 SAPMZTS1：

```
REPORT SAPMZTST NO STANDARD PAGE HEADING.

DATA: INT TYPE I,
      RSPAR LIKE RSPARAMS OCCURS 10 WITH HEADER LINE.

RANGES SELTAB FOR INT.

WRITE: 'Select a Selection!';
     / '--------------------'.
SKIP.

FORMAT HOTSPOT COLOR 5 INVERSE ON.
WRITE: 'Selection 1',
     / 'Selection 2'.

AT LINE-SELECTION.
  CASE SY-LILLI.
    WHEN 4.
      SELTAB-SIGN = 'I'. SELTAB-OPTION = 'BT'.
      SELTAB-LOW  = 1.   SELTAB-HIGH   = 5.
      APPEND SELTAB.
      SUBMIT SAPMZTS1 VIA SELECTION-SCREEN
                      WITH PARAMET EQ 'Selection 1'
                      WITH SELECTO IN SELTAB
                      WITH SELECTO NE 3
                      AND RETURN.
    WHEN 5.
      RSPAR-SELNAME = 'SELECTO'. RSPAR-KIND = 'S'.
      RSPAR-SIGN = 'E'. RSPAR-OPTION = 'BT'.
      RSPAR-LOW  = 14.  RSPAR-HIGH = 17.
      APPEND RSPAR.
      RSPAR-SELNAME = 'PARAMET'. RSPAR-KIND = 'P'.
      RSPAR-LOW  = 'Selection 2'.
      APPEND RSPAR.
      RSPAR-SELNAME = 'SELECTO'. RSPAR-KIND = 'S'.
      RSPAR-SIGN = 'I'. RSPAR-OPTION = 'GT'.
      RSPAR-LOW  = 10.
      APPEND RSPAR.
      SUBMIT SAPMZTS1 VIA SELECTION-SCREEN
                      WITH SELECTION-TABLE RSPAR
                      AND RETURN.
  ENDCASE.
```

执行该报表 之后，显示 下列基本列 表：

单击第一个 热点后，SAPMZTS1 的选择屏幕 显示如下：

单击第二个 热点后，SAPMZTS1 的选择屏幕 显示如下：

系统为 SAPMZTS1 的两个调用 传送导致两 行选择表 SELECTO 的值。第二 行出现在各自的对话窗 口“SELECTO 的多重选择 ”中。如果 没有 SUBMIT 语句的 VIA SELECTION-SCREEN 选项，系统 将相应地在 SAPMZTS1 中填写 PARAMET 和 SELECTO， 但不显示它 们。

调用事务

要从报表中 调用事务， ABAP/4 提供两种可 能的方法。
如果不想在 终止事务后 返回调用报 表，请使用 ：

语法
```
LEAVE TO TRANSACTION <tcod> [AND SKIP FIRST SCREEN].
```
此语句结束 报表，并启 动事务 <tcod>。

如果要在终 止事务后返 回调用报表 ，请使用：

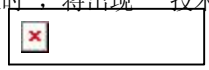
语法
CALL TRANSACTION <tcod> [AND SKIP FIRST SCREEN] [USING <itab>].
此语句保存 报表的数据 ，并启动事 务〈tcod〉。 事务的结束 后，系统返 回到调用报 表中调用语 句后面的语 句。
可以使用变 量指定事务〈tcod〉。 这允许静态 地调用事务 ，也允许动 态地调用事 务。
使用 AND SKIP FIRST SCREEN 选项禁止事 务初始屏幕 的显示。但 是，只有当 用 SPA/GPA 参数中的输 入值完全并 正确地填写 了事务初始 屏幕上的所 有命令字段 之后，该选 项才有效。 关于 SPA/GPA 技术的详细 信息，参见 ：
使用 CALL TRANSACTION 语句的 USING ITAB 选项将内部 表〈itab〉传送到被调 用事务，该 事务具有批输入表格的 格式。关于 批输入的详 细信息，参 见文档 *基础编程界面* (页 **Error! Not a valid link.**)。

将 SPA/GPA 参数传送到 事务

要用报表中 的数据填写 被调用事务 的输入字段 ，可以使用 SPA/GPA 技术。SPA/GPA 参数是系统 存储在全局 的、用户相 关的 SAP 内存中的值 。使用 SAP 内存在超过 事务边界的 程序之间传 送值。对于 所有并行使 用的模式， 用户可以在 终端会话中 访问存储在 SAP 内存中的值 。
通常，将事 务初始屏幕 上的输入字 段连接到 SPA/GPA 参数。如果 在调用事务 之前从程序 中填写这些字段，系统 将使用相应 值填写输入 字段。
要填写 SPA/GPA 参数，请使 用：

语法
SET PARAMETER ID <pid> FIELD <f>.
此语句保存 SAP 内存中标识〈pid〉下面的字段〈f〉的内容。标 识〈pid〉使用三个字 符。如果存 在标识〈pid〉， 此语句将改 写先前存储 在那里的值 。
如果不存在 标识〈pid〉， 请双击 ABAP/4 编辑器中的〈pid〉以创建新参 数对象。
要将 SPA/GPA 参数读入 ABAP/4 程序中，请 使用：

语法
GET PARAMETER ID <pid> FIELD <f>.
此语句将存 储在标识〈pid〉下面的值填 写入变量〈f〉中。如果系 统在 SAP 内存中没有 找到〈pid〉的值，则它 将 SY-SUBRC 设置为 4，否则设 置为 0。要将参 数传送到被 调用程序， 则不需要该 语句。
调用事务时 ，必须知道 将哪一个 SPA/GPA 参数连接到 事务初始屏 幕上的输入 字段。要查 明该情况，请启动事务 ，并将光标 定位在单个 输入字段上 ，选择“帮 助”，在随 后出现的对 话窗口中选 择“技术信 息”。这时 ，将出现“ 技术信息” 对话窗口， 显示字段“ 参数标识” 中的适当 SPA/GPA 参数。



预订事务 TCG2 的第一个输 入字段的技 术信息显示 如下：

输入字段“ 公司”的 SPA/GPA 参数有 ID CAR。使 用该方法查 找其它输入 字段的 ID CON、DAY 和 BOK。

下列报表连 接到逻辑数 据库 F1S，并 调用事务 TCG2：

```
REPORT SAPMZTST NO STANDARD PAGE HEADING.

TABLES SBOOK.

START-OF-SELECTION.
  WRITE: 'Select a booking',
       / '----------------'.
  SKIP.

GET SBOOK.
  WRITE: SBOOK-CARRID, SBOOK-CONNID,
         SBOOK-FLDATE, SBOOK-BOOKID.
  HIDE:  SBOOK-CARRID, SBOOK-CONNID,
         SBOOK-FLDATE, SBOOK-BOOKID.

AT LINE-SELECTION.
  SET PARAMETER ID: 'CAR' FIELD SBOOK-CARRID,
                    'CON' FIELD SBOOK-CONNID,
                    'DAY' FIELD SBOOK-FLDATE,
                    'BOK' FIELD SBOOK-BOOKID.
  CALL TRANSACTION 'TCG2'.
```

报表的基本 列表按照用 户在选择屏 幕上的输入 显示数据库 表 SBOOK 中的字段。 这些数据也 存储在每行 的 HIDE 区域中。

如果用户通 过双击选择 预订数据的 行，则系统 将触发 AT LINE-SELECTION 事件，并获 取存储在 HIDE 区域中的数 据，以将它 们填写到事 务 TGC2 初始屏幕的 SPA/GPA

参数中。然 后调用该事 务。由于没 有使用 AND SKIP FIRST SCREEN 禁止初始屏 幕，所以该 初始屏幕可 能显示如下 ：

如果在 CALL TRANSACTION 语句中使用 AND SKIP FIRST SCREEN 选项，因为 填写了第一 个屏幕的所 有要求字段 ，所以将立 即出现第二 个屏幕。

第八章 打印列表

概览

内容

ABAP/4 报表的输出 结果显示在 列表中。默 认情况下， 系统在创建 列表之后将 它（基本列 表和辅助列 表）发送到 输出屏幕上 。本节讨论 如何将列表 发送到 SAP 假脱机系统 而不发送到 输出屏幕。
在 ABAP/4 中，将列表 发送到 SAP 假脱机系统 通常称为 ‘ 打印列表’ 。但是这并 不一定意味 着在打印机 上实际打印 列表。也可 以使用假脱 机系统暂时 存储列表， 并且可以使 用该系统将 列表存档而不打印。关 于 SAP 假脱机系统 的详细信息 ，请参见文 档*打印指南* (页 **Error! Not a valid link.**)，关于存 档的详细信 息，参见文 档 *ArchiveLink 编程界面* (页 **Error! Not a valid link.**).
ABAP/4 提供了两种 打印列表的 可能：
可以在列表 创建之时或 创建之后打 印。

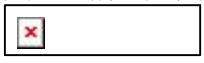
在创建列表 之后打印

在创建列表 之后打印时 ，请勿使用 下面主题中 所描述的专 用打印语句 来将列表从 程序内发送 到 SAP 假脱机系统 。
默认情况下 ，系统将完 整的列表发 送到输出屏 幕。如果列 表用户界面 的状态中已 激活了“打 印”功能（ 功能码为 PRI)， 那么，用户 可以选定“ 打印”以将 所显示的列 表发送到 SAP 假脱机系统 （参见*打印输出列表* (页 **错误！链接无效。**))。 在“打印屏 幕列表”对 话窗口中， 系统需要输 入打印参数 （参见*打印参数* (页 322))。 要修改该屏 的预设置， 请参见*打印参数 - 预设置值* (页 324) 。
创建列表之 后打印可能 产生如下几 个问题：

为屏幕显 示而不是打 印输出格式 化了显示在 输出屏幕上 的列表。由 于下述原因 ，不能始终将显示格式 用于打印：

- 输出屏 幕上的列表 通常只包含 一个单页（ 参见 *确定页长* (页 **错误！链接无效。**)中的注意事 项)。打印 时，系统将 逻辑页‘分 成’若干物 理页(其格 式取决于指 定的打印参 数)。系统 在每个打印 页中设置页 眉。如果页 眉包含页码 ，则所有页 中（SY-PAGNO） 的页码都相 同。这样， 就禁止了对 打印页的连 续编号。
- 如果列 表含有使用 NEW-PAGE（ 参见 *无条件分页* (页 **错误！链接无效。**)）编排的分 页符，则这 些分页符不 适用于打印 页的格式， 因为它可能 导致进一步 自动设置分 页符。由于只有 NEW-PAGE 增加 SY-PAGNO 系统字段， 所以对于由 自动分页符 创建的打印 页，系统将使用与前一 页相同的页 眉。
- 如果由 于 REPORT 或 NEW-PAGE 语句中的 LINE-COUNT 选项而使得 列表包含若 干页（参见 *多页列表* (页 **错误！链接无效。**)），那么， 可以根本不 打印列表（ 要是指定的 页长度超出 了打印页的 最大页长度 ），也可以 不充分使用 此物理打印 页。
- 可以将 输出屏幕上 列表的宽度 设置成 1 ~ 255 之间的任何 值（参见*确定列表宽度* (页 **错误！链接无效。**)） 。该列表宽 度不适用于 打印格式。 标准打印机 不能打印超 过 130 列的列表， 如果超出此 范围，打印 机将截去行 的超出部分 。

创建屏幕 输出列表时 ，列表中不 能包含打印 控制语句（ 参见*打印控制* (页 323) ）。
在每一打 印页的末尾 ，不能输出 程序中所定 义的页脚行 。相反，只 在“打印屏 幕列表”对话窗口中标 记“页脚” 。系统将在 每页为系统 定义的页脚 行保留一空 行。
输出屏幕上 完整列表的 打印输出是 屏幕的硬拷 贝而不是真 正的程序控 制打印输出 。这种方法 只能用于测 试或打印机 可以接受其 格式的列表 。对于复杂 列表（例如 ，包含并不 在每个打印 页出现的扩 展页眉的列 表)，请在 程序中使用 打印语句（ 参见*在创建列表 时打印* (页 322))。



如果要允许 用户从输出 屏幕启动程 序控制打印 进程，请使 用交互式报 告方法（参见 *交互式列表* (页 **错误！链接无效。**) ）。例如， 第一次为输 出屏幕创建 列表时，

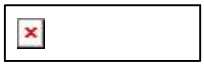
请 使用自定义 用户界面， 在该界面中 可以用自定 义功能码代 替功能码 PRI 。在 AT USER-COMMAND 事件中，请 为假脱机系 统重新创建 列表（参见 *在创建列表 时打印* （页 322） ）。

在创建列表 时打印

如果创建列 表时打印， 由于系统可 根据打印机 的要求格式 化列表，因 此将有最好 的打印输出 。系统将根 据打印格式 设置列表宽 度和页长。 这样可避免 行的宽度超 出所使用的 打印格式的 范围。分页 符将出现在 物理打印页 的末尾。
报表程序在 开始创建列 表之前必须 识别此打印 格式。打印 格式是打印 参数的一部 分。打印参 数由用户交 互地设置， 也可从程序 中设置。
ABAP/4 可能提供下 列方法以便 在创建列表 时打印：

- 如果报表 程序显示选 择屏幕，那 么，用户可 以在选择屏 幕上选择“ 执行 + 打印”。
- 可以使用 NEW-PAGE PRINT ON 语句，从报 表中启动打 印输出。
- 可以使用 SUBMIT ... TO SAP-SPOOL 语句调用报 表。
- 可以使用 功能模块 JOB_SUBMIT 将报表包含 到后台作业 。关于后台 作业和功能 模块 JOB_SUBMIT 的详细信息 ，请参见文 档*基本编程界面* （页 **Error! Not a valid link.**）。

创建列表时 打印，可以 控制打印格 式。



创建列表时 打印,系统 将每个完整 页发送到假 脱机系统， 然后删除该 页。因此已 打印列表的 长度仅受假 脱机系统容 量的限制。 与显示列表 相反，打印 时系统不存 储列表级别 。由于打印 的总列表从 不存在，所 以您无法查 阅前面页内 容。

打印参数

打印进程开 始之前，必 须设置打印 参数。
创建列表之 后再打印时 ，系统使用 打印参数中 指定的打印 格式将完整 列表分割以 符合打印页 面，必要时 还会作相应 的截断。
在创建列表 时打印，系 统使用打印 格式实际格 式化程序中 的列表。
打印参数由 用户交互地 设置，也可 从程序中得 到。
下述主题包 含：

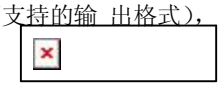
打印参数 - 概述
对于每种打 印进程，假 脱机系统都 需要一组完 整一致的打 印参数。在 ABAP/4 中，结构 PRI_PARAMS （ABAP/4 词典）的字 段串表示一 组打印参数 。
交互地传递 打印参数时 ，在启动报 表之后系统 将显示下列 对话窗口， 并要求输入 最重要的打 印参数。

下表说明“ 打印参数” 对话窗口的 输入字段与 PRI_PARAMS 组件的对应 关系。

输入字段	组 件	说明
输出 设备	PDEST	打 印机或传真 机名称 （从 用户缺省值 中预设）
份 数	PRCOP	打 印的份数。 （预设值： 1）
名称	PLIST	假 脱机请求名 称。只在您 不想立即打 印时设置此 项。（用包 括用户名（ SY-UNAME ）前三个字 符的报表名 称作预设值 ）。
标题	PRTXT	假 脱机请求的 说明文本， 该文本出现 在封面上。
权限	PRBER	对 假脱机请求 的权限。仅 有权限的用 户才可以查 看请求内容 。
立即打印	PRIMM	如 果标记该字 段，那么， 系统在完成 标记之后立 即向“输出 设备”发送 假脱机请求 。 （从用户 缺省值中预 设）
打印后 删除	PRREL	如 果标记该字 段，系统在 将假脱机请 求输出到“ 输出设备” 之后立即将 其删除。否 则系统将在 “保留期”到期后删除 假脱机请求 。

		（从用户 缺省值中预 设）
新假脱 机请求	PRNEW	如 果标记该字 段，系统将 创建新假脱 机请求。否 则系统尽量 将该假脱机 请求附加到 尚未完成的 请求中。在 这种情况下 ，“名称” 、“输出设 备”、“份 数”以及“ 格式”必须 相同。
保留 期	PEXPI	系 统在删除假 脱机请求之 前的保留天 数。 （预设 值：8）
存 档模式	ARMOD	指 定存档模式 。要选择“ 打印”、“ 存档”和“ 打印及存档 ”（ARMOD 值是 1、2 或 3），请单 击可能的条 目按钮。 （ 预设值：打 印）
SAP 封面	PRSAP	如 果该字段的 值为‘X’ ，系统将创 建包含若干 数据的封面 。如果该字 段的值为‘ D’，系统 将根据输出 设备的设置 来决定封面 打印与否。 如果该字段 为空，系统 将不创建封 面。 （预设 值：‘D’ ）
接受者	PRREC	为 “SAP 封面”指定 接受者名称 。 （预设值 ：用户名）
部门	PRABT	为 “SAP 封面”指定 部门名称。 （其值从用 户地址中预 设）
行数	LINCT	列 表行数。该 字段与 REPORT 语句中的 LINE-COUNT 选项有相同 效果。不能 将打印行数 指定为 0（不限制行数）。该字 段中的最大 数值取决于 “格式”字 段的内容。 （值由内部 预设）
列数	LINSZ	列 表每行的字 符数。该字 段与 REPORT 语句中的 LINE-SIZE 选项有相同 效果。对于 标准打印机 ，所指定的 列数不能超 过 130。该 字段中的最 大数取决于“格式”字 段的内容。 （值由内部 预设）
格式	PAART	该 字段实际确 定了输出的 页面格式。 对应于所连 接的打印机 ,用户可以 在该字段中 设置具有不 同最大页长 和行宽值的 打印格式。 （值由内部 预设）

“打 印参数”对 话窗口将检 查输入值的 一致性和完 整性。如果 打印参数不 一致(例如 ,使用了指 定输出设备 不支持的输 出格式)， 就无法打印 。



打印参数 LINCT 和 LINSZ 不能覆盖 REPORT 或 NEW-PAGE 语句中的LINE-COUNT 和 LINE-SIZE 选项。如果 您在程序中 使用这些选 项，则此处 所指定的值 将填充 LINCT 和 LINSZ 组件。“打 印参数”对 话窗口中相 应的输入字 段将不再接 受输入。如 果所指定的 值超出了在 “格式”字 段中确定的 最大值，则 无法打印列 表。

除了打印参 数，还有存 档参数。然 而，只要打 开光学存档 （存档模式 为“存档” 或“打印及 存档”），就必须指定 这些参数。 在 ABAP/4 中，与 ARC_PARAMS（ ABAP/4 字典）具有 相同结构的 字段串代表 一组存档参 数。如果已 打开了光学 存档并且交 互设置了打 印参数，则 将显示另一 个对话窗口 ，即“存档 参数”窗口 ，在该窗口 上用户必须 设置最重要 的存档参数 ：

下表说明“ 存档参数” 对话窗口的 输入字段与 “ ARC_PARAMS ”组件的对 应关系。

输入字段	组 件	说明
对象 类型	SAP_OBJECT	SAP 对象的对象 类型
文档类 型	AR_OBJECT	存 档对象的文 档类型
信息	INFO	存 档请求的简 捷信息
文本	ARCTEXT	存 档请求的说 明文本

除非 在对话窗口 中所作的输 入一致并且 完整，否则 无法将列表 存档。

要从程序中 设置打印和 存档参数， 必须使用功 能模块GET_PRINT_PARAMETERS （参见 *在程序中设 置打印参数* （页 345））。 系统不接受 直接赋给打 印和存档参 数集的任何 值。如果直 接赋值并稍 后使用这些 值，将回出 现运行时间 错误。

打印参数 - 预设置值

用户选择下 列选项之后 将始终显示 “打印参数 ”对话窗口
选择屏幕 上的“执行 + 打印”选项 。
列表界面 上的“打印 ”选项。
您不能禁止 用户操作对 话窗口。
但可以在程 序中对“打 印参数”对 话窗口进行 预设置。请 使用功能模 块 SET_PRINT_PARAMETERS。 该功能模块 没有输出参 数，只对由 上述用户操 作之一触发 的列表打印 起作用。
对于选择屏 幕上的“执 行 + 打印”，必 须在 AT SELECTION-SCREEN 事件期间调 用参数集 SET_PRINT_PARAMETERS。 对于列表界 面的“打印 ”，您必须 在列表发送 到输出屏幕 之前（或更 早）调用功 能模块。
下表说明 SET_PRINT_PARAMETERS 的输入参数 与打印及存 档参数之间 的对应关系 ：

输入参数	参 数	说明
IN_PARAMETERS	PRI_PARAMS	整 个集合
IN_ARCHIVE_PARAMETERS	ARC_PARAMS	整 个集合
ARCHIVE_MODE	PRI_PARAMS-ARMOD	存 档模式
AUTHORITY	PRI_PARAMS- PRBER	授 权
COPIES	PRI_PARAMS- PRCOP	份 数
COVER_PAGE	PRI_PARAMS-PRBIG	选 择封面页
DATA_SET	PRI_PARAMS-PRDSN	假 脱机文件
DEPARTMENT	PRI_PARAMS-PRABT	部 门名称
DESTINATION	PRI_PARAMS-PDEST	输 出设备
EXPIRATION	PRI_PARAMS-PEXPI	假 脱机保留期
IMMEDIATELY	PRI_PARAMS-PRIMM	立 即打印
LAYOUT	PRI_PARAMS-PAART	页 面设置
LINE_COUNT	PRI_PARAMS-LINCT	每 页的行数
LINE_SIZE	PRI_PARAMS-LINSZ	每 行的列数
LIST_NAME	PRI_PARAMS-PLIST	假 脱机请求的 名称
LIST_TEXT	PRI_PARAMS-PRTXT	说 明文本
NEW_LIST_ID	PRI_PARAMS-PRNEW	新 假脱机请求
RECEIVER	PRI_PARAMS-PRREC	接 受者
RELEASE	PRI_PARAMS-PRREL	输 出后删除
SAP_COVER_PAGE	PRI_PARAMS-PRSAP	SAP 封面页
TYPE	PRI_PARAMS-PTYPE	假 脱机请求类 型
FOOT_LINE	PRI_PARAMS-FOOTL	输 出页脚行
ARCHIVE_ID	ARC_PARAMS-ARCHIV_ID	目 标存档
ARCHIVE_INFO	ARC_PARAMS-INFO	信 息
ARCHIVE_TEXT	ARC_PARAMS-ARCTEXT	说 明文本
AR_OBJECT	ARC_PARAMS-AR_OBJECT	文 档类型
SAP_OBJECT	ARC_PARAMS-SAP_OBJECT	对 象类型

要了 解“打印参 数”和“文 档参数”对 话窗口中的 哪些字段与 这些参数对 应，请参见 *打印参数 - 概述* （页 353） 中的表格。

对于参数 IN_PARAMETERS 和 IN_ARCHIVE_PARAMETERS， 必须分别使 用结构 PRI_PARAMS 和 ARC_PARAMS 来给字段串 赋值。这些 字段串必须 已初始化或 包含功能模 块 GET_PRINT_PARAMETERS 的结果（参 见 *在程序中设 置打印参数* (页 345))。
如果用户在 列表界面上 选择了“打 印”，则系 统只使用 FOOT_LINE 参数。如果 该参数等于 ‘X’，则 系统在每一 页都输出一 个系统定义 的页脚行。
要在程序中 包括功能模 块，请在 ABAP/4 编辑器中使 用“编辑 -> 插入语句...CALL FUNCTION” 。
关于如何使 用 SET_PRINT_PARAMETERS 的示例，请 参见 *执行和打印* (页 330) 。

在程序中设 置打印参数

如果使用打 印语句

```
NEW-PAGE PRINT ON
SUBMIT ... TO SAP-SPOOL
CALL FUNCTION 'JOB-SUBMIT'
```

来打印，可 以使用打印 语句中的相 应选项，在 程序中设置 打印参数。 您可以通过 “打印参数 ”对话窗口 选择允许或 禁止用户对 话。
为确保正确 完整地将参 数发送到假 脱机系统， 应始终使用 打印语句传 输完整的参 数集。要创 建参数集， 请使用功能 模块 GET_PRINT_PARAMETERS。 下列主题将 讨论该功能 模块：
GET_PRINT_PARAMETERS - 概述 (页 290)
GET_PRINT_PARAMETERS 的输入参数 (页 332)
GET_PRINT_PARAMETERS 的输出参数 (页 318)
GET_PRINT_PARAMETERS 的例外参数 (页 318)
如何使用 GET_PRINT_PARAMETERS (页 318)

GET_PRINT_PARAMETERS - 概述

功能模块 GET_PRINT_PARAMETERS 要完成下列 任务：

- 创建一组 打印和存档 参数。
- 各个打印和 存档参数紧 密相连而且 必须完整。 例如对每台 输出设备， 您必须指定 页面设置的 格式，而页 面设置格式 又需要设置 行数和列数 。否则设置 存档模式“ 存档”或“ 打印和存档 ”时，必须 设置存档参 数。
- 从实际打 印语句中分 解用户对话 。
- 打印语句（ NEW-PAGE PRINT ON, SUBMIT <rep> TO SAP-SPOOL） 支持用户对 话，但是有 不提供“返 回”功能的 不足。在使 用打印语句 启动打印进 程后，系统 不能返回到 打印语句之 前。用户只 有通过“结 束”（终止 整个程序） 来结束该进 程。

GET_PRINT_PARAMETERS 执行下列功 能：

- 您可以使 用输入参数 设置打印和 存档参数。 功能模块可 从系统中接 收尚未通过 输入参数设置的任何所 需值。这些 值对应于“ 打印参数” 对话窗口中 的预设值， 其中部分值 在用户的主 记录中设置 。
- 默认情况 下，功能模 块显示用户 对话框的“ 打印参数” 对话窗口。 在此，用户 可以改写用输入参数或 预设值填充 的字段。
- 功能模块 将自动设置 相关值。如 果设置输入 参数（例如 ，为某种页 面设置参数 ），则系统自动设置诸 如行数和列 数等相关参 数，而不是 象输入参数 一样请求输 入。
- 功能模块 提供完整的 打印和存档 参数集作为 输出参数。 您可以使用 打印语句中 的选项将这些输出参数 传输到假脱 机系统。参 数集可以全 部填充，也 可以全部为 空。

GET_PRINT_PARAMETERS 的输入参数

功能模块 GET_PRINT_PARAMETERS 与功能模块 SET_PRINT_PARAMETERS 具有相同的 输入参数（ 参见 *打印参数 - 预设置值* (页 355))，但是下 列参数例外 ：
GET_PRINT_PARAMETERS 没有输入参 数 FOOT_LINE， 这是由于只 有用户在列 表输出屏幕 上选择“打 印”时，才 需要该参数 。
GET_PRINT_PARAMETERS 有下列附加 输入参数：

MODE

下列值影响 模块功能：

MODE	效 果
PARAMS	该 值为默认值 。用户可以 在对话窗口 中选择“打 印”或“取 消”。
PARAMSEL	对 话窗口包含 附加的“选 择封面页” 复选框。如 果用户填写 了该字段（ 打印参数 PRBIG ），则系统 在包含选择 屏幕选项的 输出中包括 封面页。
DISPLAY	对 话窗口中的 打印参数为 只显示。
CURRENT	在 当前打印进 程中（在打 印语句之后 ），使用功 能模块确定 打印参数。这些值相当 于打印参数 集。如果没 有打印进程 ， 系统将使 用预设值。
BATCH	使 用功能模块 确定后台作 业的打印参 数。必须在 输入参数 REPORT 中指定要启 动的报表程 序。如果报 表程序的 REPORT 语句中包含

	LINE-COUNT 和 LINE-SIZE 选项，则系 统在对话窗 口中将它们 作为预设值 。在对话窗 口中系统提 供“保存” 按钮而不是 “打印”按 钮。

REPORT
REPORT 中包含的值 总是影响假 脱机请求（ PLIST 组件）中的 名称预设值 ，否则该预 设值由 SY-REPID 系统字段确 定。该值本 身可由输入 参数 LIST_NAME（ 如果使用） 改写。
如果将 MODE 设置成‘BATCH’ ，REPORT 中的值将指 定想作为后 台作业予以 启动的报表 程序名称。 GET_PRINT_PARAMETERS 为后台报表 程序而不是 为当前报表 程序确定打 印参数。
NO_DIALOG
确定是否显 示对话窗口 。如果 NO_DIALOG 的值为‘X’ ，则系统禁 用该对话。

GET_PRINT_PARAMETERS 的输出参数

功能模块 GET_PRINT_PARAMETERS 包括下列输 出参数：
OUT_PARAMETERS
该参数要么 包含一组完 整的打印参 数，要么为 空（参见 VALID ）。
OUT_ARCHIVE_PARAMETERS
该参数要么 包含一组完 整的存档参 数，要么为 空（参见 VALID ）。
VALID
该参数显示 参数集 OUT_PARAMETERS 和 OUT_ARCHIVE_PARAMETERS 是已完全填 充还是为空 。如果 VALID 的值为‘X’，则参数集 是完整的。这时就可以 将它们传输 到假脱机系 统。如果 VALID 的值为 SPACE， 则字段集为 空。如果用 户取消用户 对话，就可 将 VALID 值设置成 SPACE。 因此，用户 对话之后始 终检查 VALID 。如果没发 生用户对话 ，那么，VALID 的值将为‘ X’。

GET_PRINT_PARAMETERS 的例外参数

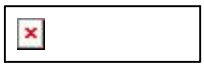
功能模块 GET_PRINT_PARAMETERS 包含下列例 外参数：
ARCHIVE_INFO_NOT_FOUND
指定的存档 数据不一致 或指定的档 案文件在系 统中不存在 。
INVALID_PRINT_PARAMS, INVALID_ARCHIVE_PARAMS
打印或存档 参数集无效 。通过直接 给参数字段 串的各个组 件赋值或使 用这些结构 填写输入参 数 IN_PARAMETERS 或 IN_ARCHIVE_PARAMETERS， 您可创建无 效的参数集 。参数字段 串必须是前 一次调用 GET_PRINT_PARAMETERS 的结果。无 效的输入参 数(如,被 设为 0 的行数或列 数)也会创 建无效的参 数集。

如何使用 GET_PRINT_PARAMETERS

功能模块 GET_PRINT_PARAMETERS 是 ABAP/4 允许的给打 印和存档参 数集赋值的 唯一途径。 使用 GET_PRINT_PARAMETERS 向假脱机系 统传输所填 写的参数集 可以避免程 序异常终止 ，这对后台 处理特别重 要。但是， 必须确保输 出参数 VALID 不等于 SPACE 并且没有例 外发生。
注意：对于 GET_PRINT_PARAMETERS， 参数集完整 因而系统能 执行打印请 求是最重要 的。 GET_PRINT_PARAMETERS 不象“打印 参数”对话 窗口那样执 行完整一致 性检查。只 有在执行打 印请求时它 才提供一致 性检查。对 于不一致的 条目，可部 分忽略、部 分替换。例 如，您可以
使用与 LAYOUT 参数不相配 的输入参数 LINE_SIZE、 LINE_COUNT 设置值。在 用户对话中 ，系统可发 现这种不一 致性。如果 没有用户对 话，这些值 可能导致截 断打印输出 。
在输入参 数 DESTINATION 中设置无效 值，同时将 IMMEDIATELY 设置成‘X’ 。在这种情 况下，功能 模块将用默 认值（LP01） 替换输出设 备，并将组 件 PRIMM 设置成 SPACE。 这将导致假脱机系统使 用与默认打 印机相配的 设置来存储 请求。
要在程序中 包括功能模 块，请在 ABAP/4 编辑器中使 用“编辑 -> 插入语句...CALL FUNCTION” 。
连续多次调 用功能模块 GET_PRINT_PARAMETERS 是合理的。 例如，您可 以在程序开 始处使用 GET_PRINT_PARAMETERS 触发用户对 话，并提示 用户进行基 本设置。然 后可以将输 出参数 OUT_PARAMETERS 和 OUT_ARCHIVE_PARAMETERS 用作输入参 数以进一步 调用功能模 块（已从此 功能模块的 程序中修改 了某些参数 ，例如用横 向格式打印 宽列表以及 用纵向格式 打印窄列表 时）。
关于使用 GET_PRINT_PARAMETERS 的示例，请 参见*从程序中打 印* (页 336) 和*调用报表的 打印列表* (页 289).

执行和打印

在创建时打 印列表的最 简单方法是 用户在报表 的选择屏幕 上选择“执 行 + 打印”。用 户可以选择 在屏幕上显 示列表（选 择“执行” ）或无需显 示直接打印 列表（选择 “执行 + 打印”）。
如果用户在 报表程序的 选择屏幕上 选择了“执 行 + 打印”，那 么，系统在 创建列表之 前将显示“ 打印参数” 对话窗口。 用户可输入 打印参数。 您可以使用 功能模块 SET_PRINT_PARAMETERS 给该对话窗口预设值（ 参见*打印参数 - 预设置值* (页 355) ）。
这样，您必 须按既可显 示又可打印 的方式来编 写此列表。 因此，在 REPORT 语句中不要 指定宽于 132 个字符（LINE-SIZE 选项）的页 面宽度并且 最好忽略页 长的设置（ LINE-COUNT 选项）。
使用“执行 + 打印”，用 户可以只打 印报表的基 本列表。要 打印在所显 示的列表的 交互事件期 间创建的辅 助列表，请 使用 NEW-PAGE PRINT ON（参见*从程序中打 印* (页 336) ）。



```
REPORT SAPMZTST NO STANDARD PAGE HEADING LINE-COUNT 0(2).

PARAMETERS P TYPE I.

INITIALIZATION.

  CALL FUNCTION 'SET_PRINT_PARAMETERS'
       EXPORTING
            ARCHIVE_MODE   = '3'
            COPIES         = '5'
            DEPARTMENT     = 'BASIS'
            DESTINATION    = 'LT50'
            EXPIRATION     = ' '
            IMMEDIATELY    = 'X'
            LAYOUT         = 'X_65_132'
            LINE_COUNT     = 54
            LINE_SIZE      = 20
            LIST_NAME      = 'Test'
            LIST_TEXT      = 'Test for User''s Guide'
            NEW_LIST_ID    = 'X'
            RECEIVER       = 'KELLERH'
            RELEASE        = ' '
            SAP_COVER_PAGE = 'X'


START-OF-SELECTION.

  DO P TIMES.
    WRITE / SY-INDEX.
  ENDDO.

TOP-OF-PAGE.

  WRITE: 'Page', SY-PAGNO.
  ULINE.

END-OF-PAGE.
  ULINE.
  WRITE: 'End of', SY-PAGNO.
```

在执行该程 序后，用户 可以在选择 屏幕上输入 参数 P 的值（例如 100 ）并选择“ 执行 + 打印”。然 后系统显示 该对话窗口 ：

功能模块 SET_PRINT_PARAMETERS 用预设值填 写输入字段 。由于调用 功能模块， 即使 REPORT 语句中包含 LINE-COUNT 选项，字段 “行”也准 备接受输入 。在这种情 况下，需要 该选项为两 页脚行保留 空间。

在“打印参 数”对话窗 口中选择“ 打印”之后 ，由于输入 参数 ARCHIV_MODE 将存档模式 设置成“打 印和存档” ，因此，系 统将显示“ 存档参数” 对话窗口。

如果用户在 选择屏幕上 输入 100 作为参数 P 的值，那么 系统将创建 SAP 封面页和两 张打印页， 具体形式如 下所示。

第一页：

```
Page     1
--------------------
         1
         2
         3

   .....

        49
        50
--------------------
End of     1
```

第二页：

```
Page     2
--------------------
```

```
         58
         59
         60

   .....

         99
        100
--------------------
End of      2
```

每页可输出 的行数多达 54 行（包括页 眉和页脚） 。请注意， 系统将按*创建列表*（页 **错误！链接无效。**）中 所说明的那 样，启用分 页符并创建 页眉和页脚 。

如果用户在 选择屏幕上 选择了“执 行”而不是 “执行 + 打印”，则 系统将列表 作为一页予 以显示，并 且在输出屏 幕上不显示 页脚。

从程序中打 印

创建列表的 同时要从程 序中启动打 印进程，请 使用带有 PRINT ON 选项的 NEW-PAGE 语句：

语法

```
NEW-PAGE PRINT ON [NEW-SECTION]
                  [<params> | PARAMETERS <pripar>]
                  [ARCHIVE PARAMETERS <arcpar>]
                  [NO DIALOG].
```

该语句的作 用是将所有 后续输出放 置在新页上 （参见 *无条件分页*（页 **错误！链接无效。**）），并且系统将 NEW-PAGE PRINT ON 之后的输出 语句解释为 打印语句。 换句话说， 从 NEW-PAGE PRINT ON 启动后，系 统不再创建 用于显示的 列表而创建 用于假脱机 系统的列表 。

已经创建假 脱机系统的 列表之后， 如果使用的 NEW-PAGE PRINT ON 语句无 NEW-SECTION 选项，则语 句无效。

如果使用 NEW-SECTION 选项，请重 新将页数（ SY-PAGNO 系统字段） 设为“1” 。如果系统 已为假脱机 系统创建了 列表，那么 NEW-SECTION 可能有两种 效果结果：

- 如果指定 的打印参数 与当前创建 的列表参数 匹配，并且 打印参数 PRNEW 等于 SPACE， 那么，系统 不创建新的 假脱机请求 。
- 如果指定 的打印参数 与当前创建 的列表参数 不匹配，或 者打印参数 PRNEW 不等于 SPACE，那么系统将 关闭当前假 脱机请求并 创建新的假 脱机请求。

其它选项决 定打印参数 （如下所述 ）。

在处理块（ 数据恢复期 间的事件或 交互事件） 的结尾将自 动结束打印 进程。要直 接结束为假 脱机系统创 建列表，请 使用 NEW-PAGE 语句中的 PRINT OFF 选项：

语法

```
NEW-PAGE PRINT OFF.
```

该语句将创 建分页符并 将最后一页 发送到假脱 机系统。该 语句之后的 任何输出语 句都将显示 在输出屏幕 的列表中。

确定打印参 数

要确定 NEW- PAGE PRINT 语句之后用 于打印输出 的打印参数 ，请使用语 句的下列选 项：

可以使用若 干选项 <params> 以指定每个 打印参数（ 例如，DESTINATION <dest>） 。关键字文 档对每个选 项都有说明 。使用 NO DIALOG 选项可以告 诉系统是显 示还是禁止 显示“打印 参数”对话 窗口。由于 系统不能检 查指定参数 是否完整，所以这种设 置打印参数 的方法是不 方便的。只 有在使用“ 打印参数” 对话窗口时 ，才检测不 完整的打印 参数。但是 这对后台作 业行不通。 如果打印参 数不完全并 且您使用了 NO DIALOG 选项，那么 ，系统在语 法检查之后 发送警告消 息，但并不 终止处理。 执行程序时 ，这可能导 致无法预见 的后果。

因此，SAP 建议不要使 用 <params> 选项，取而 代之，使用 PARAMETERS 选项，必要 时还可使用 ARCHIVE PARAMETERS 选项。要创 建相关参数 <pripar> 和 <arcpar>， 请使用功能 模块 GET_PRINT_PARAMETERS 的输出参数 （参见*在程序中设 置打印参数*（页 356））。 这是确保设 置完整的参 数以及可执 行打印请求的唯一方法 。由于功能 模块 GET_PRINT_PARAMETERS 具有自己的 用户对话， 所以总是使 用 NEW-PAGE PRINT ON 语句中的 NO DIALOG 选项。

```
REPORT SAPMZTST NO STANDARD PAGE HEADING.
```

```
DATA: VAL,
      PRIPAR LIKE PRI_PARAMS,
      ARCPAR LIKE ARC_PARAMS,
      LAY(16), LINES TYPE I, ROWS TYPE I.

CALL FUNCTION 'GET_PRINT_PARAMETERS'
     IMPORTING
          OUT_PARAMETERS         = PRIPAR
          OUT_ARCHIVE_PARAMETERS = ARCPAR
          VALID                  = VAL
     EXCEPTIONS
          ARCHIVE_INFO_NOT_FOUND = 1
          INVALID_PRINT_PARAMS   = 2
          INVALID_ARCHIVE_PARAMS = 3
          OTHERS                 = 4.

IF VAL <> SPACE AND SY-SUBRC = 0.
  SET PF-STATUS 'PRINT'.
  WRITE '   Select a format!'.
ENDIF.

TOP-OF-PAGE DURING LINE-SELECTION.
  WRITE: 'Page', SY-PAGNO.
  ULINE.

AT USER-COMMAND.
  CASE SY-UCOMM.
    WHEN 'PORT'.
      LAY = 'X_65_80'.
      LINES = 60.
      ROWS  = 55.
      PERFORM FORMAT.
    WHEN 'LAND'.
      LAY = 'X_65_132'.
      LINES = 60.
      ROWS  = 110.
      PERFORM FORMAT.
  ENDCASE.

FORM FORMAT.
  CALL FUNCTION 'GET_PRINT_PARAMETERS'
       EXPORTING
            IN_ARCHIVE_PARAMETERS  = ARCPAR
            IN_PARAMETERS          = PRIPAR
            LAYOUT                 = LAY
            LINE_COUNT             = LINES
            LINE_SIZE              = ROWS
            NO_DIALOG              = 'X'
       IMPORTING
            OUT_ARCHIVE_PARAMETERS = ARCPAR
            OUT_PARAMETERS         = PRIPAR
            VALID                  = VAL
       EXCEPTIONS
            ARCHIVE_INFO_NOT_FOUND = 1
            INVALID_PRINT_PARAMS   = 2
            INVALID_ARCHIVE_PARAMS = 3
            OTHERS                 = 4.
  IF VAL <> SPACE AND SY-SUBRC = 0.
    PERFORM LIST.
  ENDIF.
ENDFORM.

FORM LIST.
  NEW-PAGE PRINT ON
    NEW-SECTION
    PARAMETERS PRIPAR
    ARCHIVE PARAMETERS ARCPAR
    NO DIALOG.
  DO 440 TIMES.
    WRITE (3) SY-INDEX.
```

```
  ENDDO.
ENDFORM.
```

该程序还未 传递输入参 数就立即调 用功能模块 GET_PRINT_PARAMETERS。 在“打印参数”选择屏 幕上,用户 可以输入该 程序的打印 和存档参数 。使用功能 模块的输出 参数,系统 将这些参数 传递到字段 串 PRIPAR 和 ARCPAR。 为保证参数 的完整一致 性,程序通 过对话窗口 执行用户对 话并检查 VALID 的返回值。

完成对话之 后,系统显 示下列基本 列表:

在基本列表 的状态 PRINT 中,功能码 PORT 和 LAND 分别被分配 给功能键 **F5** 和 **F6**,**以及 应用程序工 具栏的两个 按钮(参见** *丌宓夕的 用户界面* (页 **错误!链接无效。**))**。如果用 户选择此中 的某一功能 ,则可能发 生** AT USER-COMMAND 事件,同时 将用于纵向 或横向输出 格式的值赋 给变量 LAY、LINES 及 ROWS 并调用子程 序 FORMAT。

子程序 FORMAT 调用功能模 块 GET_PRINT_PARAMETERS, 传递前步确 定的参数 PRIPAR 和 ARCPAR 并将它们作 为输入参数 。程序将存 储在 LAY、LINES 及 ROWS 中的值赋给输入参数 LAYOUT、LINE_COUNT 及 LINE_SIZE。但不显示用 户对话。系 统将参数返 回到字段串 PRIPAR 和 ARCPAR。 子程序调用 的功能是给 结构 PRIPAR 的组件 PAART、LINCT 和 LINSZ 设置新值。

在检查参数 的完整性和 一致性后, 程序将调用 子程序 LIST。 该子程序使 用 NEW-PAGE PRINT ON 将列表发送 到假脱机系 统,由此使 用 PRIPAR 和 ARCPAR 确定决定打 印和存档参 数。由于所 有需要的设 置都由 GET_PRINT_PARAMETERS 设置,所以无需用户对 话。

要查看存储 的假脱机请 求,用户可 选择“系统 -> 服务 -> 打印请求” 。选择“纵 向”之后, 假脱机请求 如下所示:

而选择“横 向”之后, 假脱机请求 如下所示:

调用报表的 打印列表

要将用 SUBMIT 从程序中调 用的报表输 出发送到假 脱机系统, 必须在 SUBMIT 语句中包括 TO SAP-SPOOL 选项:

语法

```
SUBMIT <rep> TO SAP-SPOOL
            [<params>|SPOOL PARAMETERS <pripar>]
            [ARCHIVE PARAMETERS <arcpar>]
            [WITHOUT SPOOL DYNPRO].
```

关于 SUBMIT 语句的说明 , 参见*调用列表* (页 **错误!链接无效。**)。在 创建打印输 出的调用报 表的列表并 将它们发送 到假脱机系 统时,使用 TO SAP-SPOOL 选项将导致 系统格式化 这些列表。 使用其它选 项确定打印 参数。

确定打印参 数

要确定打印 参数,请按 NEW-PAGE PRINT ON 语句中所说 明的那样进 行操作(参 见 *从程序中打 印* (页 359)):

可以使用某 一 <params> 选项(参见 关键字文档)单独设置 打印参数或 使用 SUBMIT 语句执行用 户对话。但 是,要确定 打印参数, 请只使用功 能模块 GET_PRINT_PARAMETERS(参见*在程序中设 置打印参数* (页 356))。功 能模块 GET_PRINT_PARAMETERS 将从 SUBMIT 语句中分解 用户对话并 且在不执行用户对话的 情况下保证 设置完整的 参数集。要 确定该参数 ,只能使用 选项 SPOOL PARAMETERS 和 ARCHIVE PARAMETERS, 要禁止 SUBMIT 语句的用户 对话,请使 用 WITHOUT SPOOL DYNPRO 选项。

下列报表与 逻辑数据库 F1S 相连接:

```
REPORT SAPMZTS1.

TABLES SPFLI.

GET SPFLI.
```

```
  NEW-LINE.
  WRITE: SPFLI-MANDT, SPFLI-CARRID, SPFLI-CONNID,
  SPFLI-CITYFROM, SPFLI-AIRPFROM, SPFLI-CITYTO,
  SPFLI-AIRPTO, SPFLI-FLTIME, SPFLI-DEPTIME, SPFLI-ARRTIME,
  SPFLI-DISTANCE, SPFLI-DISTID, SPFLI-FLTYPE.
```

下列程序调 用 SAPMZTS1 并将输出发 送到假脱机 系统：

```
REPORT SAPMZTST NO STANDARD PAGE HEADING.

DATA: VAL,
      PRIPAR LIKE PRI_PARAMS,
      ARCPAR LIKE ARC_PARAMS.

CALL FUNCTION 'GET_PRINT_PARAMETERS'
      EXPORTING
          LAYOUT                 = 'X_65_132'
          LINE_COUNT             = 65
          LINE_SIZE              = 132
     IMPORTING
          OUT_PARAMETERS         = PRIPAR
          OUT_ARCHIVE_PARAMETERS = ARCPAR
          VALID                  = VAL
     EXCEPTIONS
          ARCHIVE_INFO_NOT_FOUND = 1
          INVALID_PRINT_PARAMS   = 2
          INVALID_ARCHIVE_PARAMS = 3
          OTHERS                 = 4.

IF VAL <> SPACE AND SY-SUBRC = 0.
  SUBMIT SAPMZTS1 TO SAP-SPOOL
    SPOOL   PARAMETERS PRIPAR
    ARCHIVE PARAMETERS ARCPAR
    WITHOUT SPOOL DYNPRO.
ENDIF.
```

启动程序后 ，功能模块 GET_PRINT_PARAMETERS 触发用户对 话，同时显 示用输入参 数值填充的 “打印参数 ”对话窗口 的“输出格 式”区：

用户输入并 确认打印参 数后，程序 调用 SAPMZTS1， 同时传递 GET_PRINT_PARAMETERS 的输出参数 并将它们作 为打印和存 档参数。SAPMZTS1 既不创建屏 幕显示也不 创建用户对 话。它将已 创建的列表 直接发送到 假脱机系统 。选择“系 统 -> 服务 -> 打印请求” ，用户可以 查看所存储 的假脱机请 求。使用以 上指定的输 出格式，假 脱机请求如 下所示：

打印控制

在打印进程 中您可以从 报表中控制 列表输出。 只有当直接 将列表发送 到假脱机时 ，语句 SET MARGIN 和 PRINT-CONTROL（ 在下列主题 中说明）才 有效。语句 不影响在屏 幕上显示并 使用“列表 -> 打印”从屏 幕上打印的 列表。

确定上边距 和下边距

要确定打印 页的左边距 和上边距大 小，请使用 该语句：

语法
SET MARGIN <x> [<y>].
该语句将当 前打印页发 送到假脱机 系统，并将 打印页左边 距设置成 <x> 列，如果指 定 <y>，则 将上边距设 置成 <y> 行。
该语句将系 统字段 SY-MACOL 和 SY-MAROW 的内容设置 给 <x> 和 <y> 。对于打印 输出，这些 系统字段确 定左边距的 列数和上边 距的行数。
所设置的值 适用于所有 后续页面， 直到使用另 一 SET MARGIN 语句为止。 如果您在一 个页上指定 了多个 SET MARGIN 语句，则系 统总是使用 最后一个。

下列报表与 逻辑数据库 F1S 相连接。

```
REPORT SAPMZTST LINE-SIZE 60.

TABLES SPFLI.

SET MARGIN 5 3.

GET SPFLI.
  WRITE: / SPFLI-CARRID,    SPFLI-CONNID,   SPFLI-CITYFROM,
           SPFLI-AIRPFROM, SPFLI-CITYTO,   SPFLI-AIRPTO.
```

启动此报表 程序之后， 如果用户在 选择屏幕上 选择“执行 ”，那么， 在输出屏幕 上将出现下 列列表：

SET MARGIN 语句对屏幕 显示无效。

启动报表程 序之后，如 果用户在选 择屏幕上选 择“执行 + 打印”，那 么，在创建 列表的同时 也将打印此 列表。用户 可以使用“ 系统 -> 服务 -> 打印请求” 查看存储的 假脱机请求 ：

将列表转换 成右边距五 列及下边距 三行。

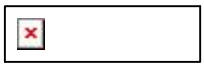
确定打印格 式

要确定打印 格式，请使 用 PRINT-CONTROL 语句：

语法
PRINT-CONTROL <formats> [LINE <lin>] [POSITION <col>].
在没有使用 选项 LINE 和 POSITION 的情况下， 该语句将按 <formats> 为从当前输 出位置（系 统字段 SY-COLNO 和 SY-LINNO ）开始打印 的所有字符 设置指定打 印格式。LINE 选项从 <lin> 行开始设置 打印格式。 POSITION 选项从 <pos> 列开始设置 打印格式。
在 <formats> 中，您可以 指定几种不 同的打印格 式。系统将 参数值转换 成打印机的 专用代码， 即所谓的打 印控制代码 。打印时系 统将打印控 制代码转换 成选定打印 机的打印机 专用控制字 符。
下表列出了 有效的 <formats> 选项以及对 应的打印控 制代码：

<formats>	代 码	说明
CPI <cpi>	CI<cpi>	每 英寸的字符 数
LPI <lpi>	LI<lpi>	每 英寸的行数
COLOR BLACK	CO001	黑 色
COLOR RED	CO002	红 色
COLOR BLUE	CO003	蓝 色
COLOR GREEN	CO004	绿 色
COLOR YELLOW	CO005	黄 色
COLOR PINK	CO006	粉 红色
LEFT MARGIN <lfm>	LM<lfm>	左 边距
FONT <fnt>	FO<fnt>	字 体
FUNCTION <code>	<code>	用 于直接指定 代码

由于打 印控制代码 比 <formats> 选项多，所 以您可以使 用 FUNCTION 选项直接指 定任何打印 控制代码。



只能使用打 印格式设置 格式（指在 格式化输出 屏幕的输出 时既不可能 也不合理的格式，例如 大小规定或 字体）。对 于其它任何 格式，请使 用在 *格式化选项*（页 **错误！链接无效。**）或*格式化输出*（页 **错误！链接无效。**）. 中说明的方 法。这些格 式自动适用 于打印和显 示（只要指 定的打印机 支持）。

要查找某种 打印机支持 的代码，请 选择“工具 -> 管理 -> 池 -> 池管理”， 这样，您就 可以进入“ 假脱机管理 ”屏幕（事 务 SPAD ）。

要获得下列 信息，请选 择单个组件 ：

选择	以获得
输出设备	所 安装打印机 的列表，包 括设备类型 。
设备类型	设备类型说 明
标准打印 控制	打印控 制代码列表
打印控制	为 每种设备类 型的打印控 制代码分配 特殊打印机 控制字符
页 面格式	页面 格式列表
纸 张类型	有效 格式列表
设 备的初始化	每种设备类 型的格式赋 值

查看“打 印控制”以 查找要使用 的打印机的 打印控制代 码。从显示 的列表中， 选择所需设 备类型。用于 Post Script 打印机的表 格（T02DD） 部分如下所 示：

在表格右栏 ，您可以查 看是否为打 印控制代码 维护了打印 机专用控制 字符。要在 对话窗口显 示单个代码 的附加信息 ，请选择“ 信息”。
关于假脱机 管理的详细 信息，参见 文档*打印指南*（页 Error! Not a valid link.）。

下列报表与 逻辑数据库 F1S 连接。

```
REPORT SAPMZTST LINE-SIZE 60.

TABLES SPFLI.

PRINT-CONTROL FUNCTION: 'SABLD' LINE 1,
                        'SAOFF' LINE 2,
                        'SAULN' LINE 3.

GET SPFLI.
  WRITE: / SPFLI-CARRID,   SPFLI-CONNID,  SPFLI-CITYFROM,
           SPFLI-AIRPFROM, SPFLI-CITYTO,  SPFLI-AIRPTO.
```

在表 T02DD 中，如果打 印控制代码 SABLD、 SAOFF 以及 SAULN 的打印机控 制字符如上 图所定义， 则系统格式 化输出，如 下所示：

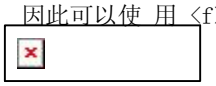
```
1996/03/13                  SPFLI                                1
------------------------------------------------------------
AA  0017 NEW YORK            JFK SAN FRANCISCO        SFO
AA  0064 SAN FRANCISCO       SFO NEW YORK             JFK
DL  1699 NEW YORK            JFK SAN FRANCISCO        SFO
DL  1984 SAN FRANCISCO       SFO NEW YORK             JFK
LH  0400 FRANKFURT           FRA NEW YORK             JFK

....
```

使用打印控 制代码 SABLD 可以将第一 行的打印格 式设置成粗 体。打印控 制代码 SAOFF 从第二行开 始关掉粗体 格式。使用 控制代码 SAULN，系统将从 第三行开始 将所有行标 上下划线。

输出存档信 息

要在列表中 包括存档信 息，请按如 下格式使用 PRINT-CONTROL 语句：

语法
PRINT-CONTROL INDEX-LINE <f>.
该语句在完 成当前打印 行之后将字 段〈f〉的内容写入 索引行。系 统将索引行 存储在假脱 机文件中 但 并不打印。 因此可以使 用〈f〉将管理假脱 机文件和档 案的信息包 含在列表中 。



下列报表与 逻辑数据库 F1S 相连接。

```
REPORT SAPMZTST LINE-SIZE 60.

TABLES SPFLI.

PRINT-CONTROL INDEX-LINE: 'Test of INDEX-LINE',
                          '------------------'.

GET SPFLI.
  WRITE: / SPFLI-CARRID,   SPFLI-CONNID,  SPFLI-CITYFROM,
           SPFLI-AIRPFROM, SPFLI-CITYTO,  SPFLI-AIRPTO.
```

在创建列表 之前，定义 了两个索引 行。如果用 户在选择屏 幕上选择“ 执行 +打印”并在 假脱机系统 中存储打印 请求，则用 户可以使用 “系统 -> 服务 -> 打印请求”查看该请求 ：

在列表的上 方将显示两 个索引行， 但并不予以 打印。

第十章 逻辑数据库 的特征和维 护

概览

内容

逻辑数据库 是 ABAP/4 报表读取和 处理数据的 方法。**每个** ABAP/4 报表都链接 到报表属性 指定的逻辑数据库中。
逻辑数据库 有个三字符 的名称（例 如，KDF） ，其最后字 母表示应用 。定义报表 属性时如果 不指定逻辑 数据库的名 称，则系统 使用控制选 择屏幕格式 的标准数据 库，但不读 取任何数据 。
在本节中， 还可以了解

有关逻辑数 据库的其它 节：
通过逻辑数 据库访问数 据的一般介 绍，参见
使用逻辑数据库访问数据库表（页 **Error! Not a valid link.**）。
关于如何使 用报表中连 接到选定屏 幕的逻辑数 据库的信息 ，参见
选择屏幕与 逻辑数据库（页 **错误！链接无效。**）
关于在报表 中如何分析 逻辑数据库 读取的数据 的信息，参 见
通过事件控制 ABAP/4 程序流（页 **错误！链接无效。**）

逻辑数据库 的特征

逻辑数据库 能提高数据 库访问的效 率并且提供 易于使用和 生成的用户 界面。
在下列主题 中，可以了 解
关于逻辑数 据库的示例 ，参见

逻辑数据库 的任务

逻辑数据库 允许集中编 程几个不同 的任务。例 如，在逻辑 数据库中集 中编码用户 界面格式和 数据库访问 ，以避免报 表的应用逻 辑去处理技 术细节。逻 辑数据库可 以执行下列 任务：

如果几个 报表读取相 同数据，则 可以在单个 逻辑数据库 中编码读取 访问。对于 单个报表，不再需要知 道所涉及的 数据库表的 确切结构（ 特别是外来 关键字相关 性）。但是 ，可以肯定 当执行 GET 事件时将以 正确的顺序 检索该项。

如果要为 几个报表使 用相同用户 界面，则可 以使用逻辑 数据库的选 择屏幕轻易 地达到目的 。要达到必 要的适应性 ，可以生成 自己的选择 屏幕版本。

在逻辑数 据库中集中 编码重要（ 以及敏感的 ）数据的授 权检查，以 使它们不受 单个报表的影响。

如果要提 高响应时间 ，则逻辑数 据库允许采 取多种措施 达到目的（ 例如使用视 图替代嵌入的 SELECT 语句）。这 些在所有相 关报表中立 即生效并且 避免修改源 代码。

逻辑数据库 的基本特征

下列定义解 释了逻辑数 据库的基本 特征：

定义

ABAP/4 报表使用逻 辑数据库读 取和处理数 据。报表可 用的数据顺 序取决于相 关逻辑数据 库的层次结 构。逻辑数 据库也提供 用户对话框 （即选择平 幕）的界面 并且将用户 输入检查和 错误对话组 合起来。可 以在报表中 修改并扩展 该界面。

该定义由逻 辑数据库格 式实现：

格式

逻辑数据库 包括至少下 列三个组件 ：

结构

结构是逻辑 数据库的基 本组件。它 决定其它组 件的结构以 及运行时逻 辑数据库的行为。

逻辑数据库 结构 (页 306)

选择

该组件决定 每个报表的 用户界面。 其格式通常 由结构决定 。可以调整 和扩展选择以适应需要 。

逻辑数据库 选择 (页 305)

数据库程 序

数据库程序 是选择数据 并将其传递 到报表的子 程序集合。 数据库程序 的格式由结构和选择共 同决定。可 以调整和扩 展数据库程 序以适应需 要。

逻辑数据库 的数据库程 序 (页 305)

其它组件诸 如文档、特 定语言文本 和用户定义 选择屏幕将 进一步扩展 功能。

逻辑数据库 允许模块化 报表中使用 的应用程序 ：

逻辑数据库 和报表

逻辑数据库 程序中的子 程序和报表 的处理块， 其中将逻辑 数据库指定 为属性，组成执行数据 库访问的**模 块化**系统。 与逻辑数据 库的结构和 选择屏幕分 开，报表中的 GET 语句决定运 行时数据库 的行为。

逻辑数据库 和 ABAP/4 报表 (页 362)

逻辑数据库 结构

通常，逻辑 数据库反映 SAP 系统中层次 表格的外来 关键字相关 性。(参见 *使用逻辑数据库访问数据* (页 **错误！链接无效。**))。

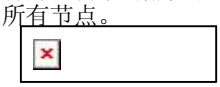
逻辑数据库 有定义如下 的层次结构 ：

- 最高层只 有一个节点 ，称为根节 点。
- 每个节点 可以有一个 或几个分支 。
- 每个节点 从其它节点 派生。

必须在 ABAP/4 字典中定义 节点结构。 一般地，这 些结构都是 逻辑数据库 为进一步评 估而读取并 传递到 ABAP/4 报表的数据 库表格结构 。但是，使 用不带下面 数据库的 ABAP/4 字典有时也 是可能并有用的。

由于技术原 因，在逻辑 数据库结构 中节点数目 有上限（MAX） 。上限如下 计算：

LEN = 结构中名称 的最大长度 （例如 7）。
MAX = 1200 / LEN （例如 1200 / 7 = 171）。
ABAP/4 报表可以在 逻辑数据库 的结构中为 每个节点包 含 GET 语句。运行 时按层次结 构中所定义 的顺序执行 处理块。
如果报表没 有为逻辑数 据库的每个 节点包含 GET 语句，则处 理块传递给 位于从根到 GET 语句指定节点路径上的 所有节点。



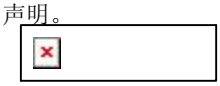
假定 LFA1 是根节点， LFBK 和 LFB1 是 LFA1 的分支，并 且 LFC1 是 LFB1 的分支。

如果报表为 所有节点都 包含 GET 语句，则以 LFA1、 LFBK、 LFB1、 LFC1 的顺序执行 GET 事件。

如果报表只 对 LFB1 包含 GET 语句，则处 理只传递到 LFA1 和 LFB1。

关于结构如 何影响选择 和数据库程 序的详细信 息，参见 *创建和维护 逻辑数据库* (页 311) 。

逻辑数据库 选择
在逻辑数据 库中，可以 使用 SELECT-OPTIONS 和 PARAMETERS 语句在选择 屏幕上定义 输入字段。 通过称为选 择包含程序 的特定包含 程序的帮助 可达此目的 。在每个 ABAP/4 报表中，可 以通过定义 报表特定选 择扩展逻辑 数据库选择 。所有报表 特定选择都 显示在特定 数据库选择 之后。
生成报表的 选择平幕时 ，系统只考 虑数据库的 特定选择标 准和参数， 其相应表格 （由选择包 含程序中的 SELECT-OPTIONS 和 PARAMETERS 语句的 FOR 选项定义， 参见 *编辑选择* (页 312) ）由报表中 的 TABLES 语句声明。



假定逻辑数 据库程序包 含下列行：

```
SELECT-OPTIONS SLIFNR FOR LFA1-LIFNR.
PARAMETERS     PBUKRS LIKE LFB1-BUKRS FOR TABLE LFB1.
```

选择标准 SLIFNR 链接到表格 LFA1， 参数 PBUKRS 链接到表格 LFB1。

如果报表中 的 TABLES 语句声明 LFA1 但未声明 LFB1， 则在选择屏 幕上显示 SLIFNR，但不显示 PBUKRS 。

可以通过逻 辑数据库包 含程序中的 SELECTION-SCREEN 语句格式化 选择屏幕（ 例如，通过 定义框、按钮、单选按 钮和空行， 或者通过在 一行中写入 几个 PARAMETER） （参见 *格式化选择标准* (页 **错误！链接无效。**))。
通过使用带 SELECT-OPTIONS 和 PARAMETERS 语句的附加 VALUE-REQUEST 和 HELP-REQUEST， 可以显示选择屏幕字段 的可能输入 值和字段文 档（参见相 应关键字文 档)。

动态选择
动态选择除 了允许用户 在逻辑数据 库选择包含 程序中已定 义的选择标 准之外，另 外还允许定 义进一步的 选择。由于 性能原因， 应该只为不 是特定表格 的选择，在 报表中的 GET 事件过程中 ，而使用带 CHENK 语句的选择 标准（参见 *有条件地离 开 GET 事件* (页 **错误！链接无效。**))。否则， 直到数据库 访问之后才 执行选择。
另一方面， 动态选择在 逻辑数据库 的数据库访 问期间已经 生效。要支 持数据库表 格 <dbtab> 的动态选择 ，必须在逻 辑数据库的 选择包含程 序中指定下 列语句：
SELECTION-SCREEN DYNAMIC SELECTIONS FOR TABLE <dbtab>.
这种情况下 ，如果在报 表中使用表 格 <dbtab> ，则“动态 选择”按钮 显示在选择 屏幕上（参 见 SELECTION-SCREEN 的关键字文 档)。按该 按钮允许用 户为由逻辑 数据库定义 的字段输入 动态选择（ 参见*选择标准与逻辑数据库* (页 **错误！链接无效。**))。为 事件关键字 GET <dbtab> 选择数据时 ，逻辑数据 库在动态 WHERE 条件下使用 它们。
在 ABAP/4 开发工作台 中，用户可 以以逻辑数 据库选择视 图的格式定 义动态选择 的字段列表 。这些视图 由其来源（ ‘CUStomer’ 的‘SAP’ 或者‘CUS’ ）标识，逻 辑数据库名 称和带有此 处描述的选 择屏幕的功 能名称必须 总是‘STANDARD’ 。只有未创 建带源‘CUS’ 的选择视图 时，系统才 使用带源‘ SAP’的 选择视图。 这样，用户 可以为其所 需定义最好 的逻辑数据 库选择视图 。

定义选择屏 幕版本
逻辑数据库 选择屏幕（ 屏幕号 1000 ）有标准的 格式，其中 选择标准和 参数以声明 的顺序显示 在各行中。 系统为每个 没有在属性 中指定选择 屏幕的报表 自动生成该 屏幕。

如果要为报 表消除某个 逻辑数据库 选择屏幕的 输入字段， 则可以在选 择包含程序 中定义选择 屏幕版本(屏幕号小于 或等于 999)并且将其 输入报表属 性。通过按 **F4**，可以 得到相关逻 辑数据库中 定义的选择 屏幕版本概 述。使用 SELECTION-SCREEN BEGIN|END OF VERSION 和 SELECTION-SCREEN EXCLUDE 语句定义选 择屏幕版本 。后一语句 允许指定想 要从选择屏 幕版本中排 除的对象(参见 *编辑选择* (页 312))。使用 SELECTION-SCREEN 语句，可以 定义其它格 式。
如果报表属 性包含选择 屏幕版本号 ，则系统在 生成选择屏 幕时将该版 本号用作模 型。自动为 选择屏幕版 本生成屏幕 流逻辑并因 而不能修改 。特别不允 许删除特定 数据库选择 。
选择屏幕版 本替代 3.0 版本之前所 使用的特定 用户选择屏 幕。
关于选择屏 幕和 PARAMETERS、 SELECT-OPTIONS 以及 SELECTION-SCREEN 语句的详细 信息，参见 *使用选择屏幕* (页 **错误！链接无效。**) 。

逻辑数据库 的数据库程 序
逻辑数据库 <dba> 的数据库程 序的名称符 合命名规则 SAPDB<dba>。 重要的是， 它包括系统 在运行 ABAP/4 报表时调用 的子程序集 合。
报表中带事 件关键字的 子程序之间 的相互作用 ，在 *逻辑数据库 和 ABAP/4 报表* (页 362) 中进行了描 述。
逻辑数据库 的结构决定 PUT 语句的行为 ，该语句在 子程序 PUT_<table> 中很重要(参见下面) 。
逻辑数据库 程序通常包 含 *定义子程序* (页 **错误！链接无效。**). 中描述的下 列子程序， 并且都使用 FORM 语句定义。

FORM INIT
在显示选择 屏幕之前调 用一次。
FORM PBO
每次刷新选 择屏幕之前 调用。
FORM PAI
用户每次在 选择屏幕上 按 **ENTER** 时调用。
系统将参数 FNAME 和 MARK 传递给子程 序，这些参 数自动定义 和填充。
- FNAME 包含选择屏 幕上选择标 准或参数的 名称。
- MARK 说明用户所 做的选择：
  MARK = SPACE 意味着用户 已经输入简 单的单值或 范围选择。
  MARK = '*' 意味着用户 已经在“多 重选择”屏 幕上制作条 目。

FORM PUT_<table>
以逻辑数据 库结构决定 的顺序调用 子程序。使 用 SELECT 语句读取节 点 <table> 的数据，并且 PUT 语句将程序 流定向到报 表中合适的 GET 语句中。PUT 语句是该子 程序的主语 句：

语法
PUT < table >.
只能在逻辑 数据库子程 序中使用 PUT <table> 语句，该逻 辑数据库包 含以 PUT_< table > 开头的名称 的节点 <table>。

PUT 语句根据逻 辑数据库的 结构定向程 序流。读取 深度由有关 报表中的 GET 语句决定。

首先，为根 节点执行子 程序 PUT_<root>。 PUT 语句然后如 下定向程序 流：

1. 如果 数据库程序 包含子程序 AUTHORITY_CHECK_< table >，则首先 执行该子程序。
2 PUT 语句试图触 发报表中的 GET 事件，即如 果存在相关 GET <table> 语句，则执 行合适的代 码块。
3. PUT 语句将程序 流定向到
- 下一节 点的子程序 上，前提是 节点的 GET 语句在报表 中相关分支 的较低级别上。
- 相同级 别节点的子 程序上，前 提是先前节 点分支到此 节点并且在 报表中存在该节点的 GET 语句。

步骤 1 时子程序中 的 PUT 再次起动。 如果它到达 有 GET 语句的报表 中分支的最 低层节点的 子程序，则 不再分支， 但继续处理 当前子程序 。当处理完 整个 PUT_<table> 子程序时， 程序流返回 将其分支到 子程序 PUT_<table> 处的 PUT 语句。

4. 从下 级子程序 PUT_< table > 返回后，如 果存在的话 ，PUT 语句分支到 报表的 GET <table> LATE 语句。

在逻辑数据 库结构中， LFB1 是 LFA1 的分支。

假定在选择 包含程序中 定义如下选 择标准：

```
SELECT-OPTIONS: SLIFNR FOR LFA1-LIFNR,
                SBUKRS FOR LFB1-BUKRS.
```

将读取数据 库程序的某 节：

```
FORM PUT_LFA1.
  SELECT * FROM LFA1
           WHERE LIFNR IN SLIFNR.
    PUT LFA1.
  ENDSELECT.
ENDFORM.

FORM PUT_LFB1.
  SELECT * FROM LFB1
           WHERE LIFNR =  LFA1-LIFNR.
           AND   BUKRS IN SBUKRS.
    PUT LFB1.
  ENDSELECT.
ENDFORM.
```

链接到逻辑 数据库的报 表将包含：

```
GET LFA1.
  WRITE LFA1-LIFNR.

GET LFB1.
  WRITE LFB1-BUKRS.
```

在该示例中 ，系统在选 择处理的开 始时调用过 程 PUT_LFA1。 PUT LFA1 语句将程序流定向到报 表中的GET LFA1 语句处理块 上。当执行 该块时， PUT LFA1 转到子程序 PUT_LFB1， 该子程序将 程序流定向 到报表中的 GET LFB1 语句。如果 LFB1 是读取的最 后节点，则 用 PUT_LFB1 中的 SELECT 循环恢复处 理。否则， 程序流移到 下一节点的 PUT_< table > 子程序上。 在最后节点 的 SELECT 循环末尾、 下一级开始 处节点的 SELECT 循环中恢复 处理。

下列图象显 示程序流：

在该示例中 ，PUT 语句不分支 到授权检查 子程序。

FORM AUTHORITY_CHECK_< table >
由 PUT <table> 语句自动调 用。在该子 程序中，可 以从逻辑数 据库结构中 为合适的节 点 <table> 指定授权检 查。
FORM PUT_<dba>_MATCHCODE
在带有选定 匹配码记录 的匹配码选 择的情况下 调用。<dba> 是逻辑数据 库的名称。 从该子程序 ，可以使用 匹配码记录 从根节点 <root> 读取相关项 。可以用 PUT <root> 调用报表中 的处理。
FORM BEFORE_EVENT, AFTER_EVENT
在事件之前 或之后调用 ，其名称在 参数 EVENT 中传递。

用如下方式 给字段 EVENT 赋值：

```
EVENT = 'START-OF-SELECTION'.
```

然后可以在 BEFORE_EVENT 和 AFTER_EVENT 子程序的列 表中使用它 。

FORM <par>_VAL, <selop>_VAL, <selop>-LOW_VAL, <selop>-HIGH_VAL
当用户按 **F4** 以获取参数 <par> 输入字段的 可能条目列 表或者为了 获得选择屏 幕上的选择 标准 <selop> （都是特定 数据库）时 调用。
FORM <par>_HLP, <selop>_HLP, <selop>-LOW_HLP, <selop>-HIGH_HLP
当用户按 **F1** 以获取参数 <par> 输入字段的 可能项的帮 助或者为了 要获得选择 屏幕上的选 择标准 <selop> （都是特定 数据库）时 调用。

关于该主题 的详细信息 ，参见:

创建和维护 逻辑数据库 (页 311)
带附加 VALUE-REQUEST 和 HELP-REQUEST的 SELECT-OPTIONS 关键字文档
带附加AS MATCHCODE STRUCTURE、 VALUE-REQUEST 和 HELP-REQUEST的 PARAMETERS 关键字文档

逻辑数据库 和 ABAP/4 报表

生成 ABAP/4 报表时的逻 辑数据库
每个 ABAP/4 报表都链接 到报表属性 中指定的逻 辑数据库上 。该逻辑数 据库将影响 报表的生成 ：
生成的选 择屏幕包含 逻辑数据库 选择（选择 标准和参数 ）**和**报表。
在选择屏 幕上，**只**显 示报表中与 数据评估相 关的特定数 据库选择。

链接到逻辑 数据库的报 表的运行行 为
当执行链接 到逻辑数据 库的报表时 ，系统以特 定顺序调用 一系列处理 块（参见 *ABAP/4 处理程序* (页 **错误！链接无效。**）)。某些处 理在报表中 编码而某些 则在逻辑数 据库程序中 编码。

在数据库程 序 SAPDB<dba> 中执行特定 数据库子程 序（参见 *逻辑数据库 的数据库程序* (页 369) ）。

在 ABAP/4 报表中执行 事件的处理 块（关于事 件和示例的 详细信息， 参见 *事件及其事件关键字* (页 **错误！链接无效。**） ）。

下列列表包 含系统为链 接到逻辑数 据库 <dba> 的 ABAP/4 报表而执行 的处理步骤 。在每种情 况下，ABAP/4 程序代码行 指定属于这 些步骤的处 理块（子程 序和事件） 。

1. 在显 示选择屏幕 之前初始化 （例如，关 键数据的默 认值） PBO。
   - 子程序 ：

     FORM INIT
     在第一次显 示选择屏幕 之前调用一 次该子程序 。
     FORM PBO.
     每次刷新选 择屏幕时调 用该子程序 （在用户按 ENTER 之后）。
   - 事件：

     INITIALIZATION.
     在第一次显 示选择屏幕 之前发生该 事件（参见 *INITIALIZATION* (页 **错误！链接无效。**） ）。
     AT SELECTION-SCREEN OUTPUT.
     该事件在每 次刷新选择 屏幕时发生 （参见 *选择屏幕的 PBO* (页 **错误！链接无效。**） ）。
2. 系统 显示选择屏 幕，用户在 输入字段中 输入数据。
3. 当用 户在选择屏 幕上按 **F4** 或 **F1** 时，显示可 能的条目和 帮助。
   - 子程序 ：

     FORM <par>_VAL.
     FORM <selop>_VAL.
     FORM <selop>-LOW_VAL.
     FORM <selop>-HIGH_VAL.
     如果用户请 求特定数据 库参数 <par> 的可能条目 (**F4**) 列表或者选 择标准 <selop> ，则根据需 要调用子程 序。
     如果用户请 求这些参数 的帮助 (**F1**) ，则调用以 _HLP 而不是 _VAL 结尾的子程序。
   - 事件：

     AT SELECTION-SCREEN ON VALUE-REQUEST FOR <par>.
     AT SELECTION-SCREEN ON VALUE-REQUEST FOR <selop>-LOW.
     AT SELECTION-SCREEN ON VALUE-REQUEST FOR <selop>-HIGH.
     如果用户请 求特定数据 库参数 <par> 的可能条目 (**F4**) 列表或者选 择标准 <selop>， 则事件发生 （参见 *创建输入值 列表* (页 **错误！链接无效。**） ）。

如果用户请 求这些参数 的帮助 (F1)， 则带附加 ON HELP-REQUEST 的事件发生而不是 ON VALUE-REQUEST（参见 *创建输入字段的帮助*（页 **错误！链接无效。**））。

4. PAI， 系统检查用 户输入是否 正确、完整 和可行，还 检查用户授 权。如果检 测到错误， 则导出个与 用户间的对 话并要求再 次输入某些 条目，以便 矫正错误。
   - 子程序 ：

     FORM PAI USING FNAME MARK.

     系统决定并 填充字段 FNAME 和 MARK。

     FNAME 包含选择屏 幕上的选择 标准或参数 的名称。

     如果 MARK = SPACE， 则用户已经 输入简单单 值或范围选 择。

     如果 MARK = '*'，则 用户已经在 “多重选择 ”屏幕上输 入选择。

     联合使用 FNAME = '*' 和 MARK = 'ANY'， 则可以在用 户选择“确 定”后立即 检查所有条 目。
   - 事件：

     AT SELECTION-SCREEN ON <fname>.

     在处理特定 输入字段之 后的事件。 必须在报表 中指定字段 <fname>（ 参见 *处理特殊输 入字* （页 **错误！链接无效。**））。

     AT SELECTION-SCREEN ON END OF <fname>.

     在处理多重 选择之后的 事件。必须 在报表中指 定字段 <fname> （参见 *处理多重选择*（页 **错误！链接无效。**））。

     AT SELECTION-SCREEN.

     用户通过选 择“确定” 显示整个选 择屏幕之后 的事件。参 见 *AT SELECTION-SCREEN*（页 **错误！链接无效。**）.。
5. 逻辑 数据库中的 数据选择和 ABAP/4 报表中的处 理
   - 子程序 ：

     FORM PUT_<table>.

     逻辑数据库 读取节点 <table> 的选择。
   - 事件：

     START-OF-SELECTION.

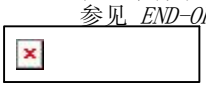
     在该事件中 ，ABAP/4 报表执行准 备工作（例 如从文件输 入数据）。 参见 *START-OF-SELECTION*（页 **错误！链接无效。**）。

     GET <table> [LATE].

     报表处理以 逻辑数据库 结构决定的 顺序读自 <table> 的数据（参 见 *GET <table>*（页 **错误！链接无效。**）and *GET <table> LATE*（页 **错误！链接无效。**））。

     END-OF-SELECTION.

     在该事件中 ，ABAP/4 报表执行结 束操作（例 如，计算总 计，向文件 输出数据）。参见 *END-OF-SELECTION*（页 **错误！链接无效。**）。



假定 TABLE1 是根节点并 且 TABLE2 是其在逻辑 数据库中仅 有的子程序 。这种情况下，数据选 择的处理步 骤嵌套和处 理如下：

1. **START-OF-SELECTION.**

报表中的准 备步骤。

2. **FORM PUT_TABLE1.**

在数据库程 序中循环读 取 TABLE1

3. **GET TABLE1.**

报表中 TABLE1 的数据处理

4. **FORM PUT_TABLE2.**

在数据库程 序中循环读 取 TABLE2

5. **GET TABLE2.**

报表中 TABLE2 的数据处理

6. **GET TABLE1 LATE.**

结束 TABLE1 循环，报表 中的数据处 理

7. **END-OF-SELECTION.**

结束报表中 的步骤。

子程序
PUT_<dba>_MATCHCODE
BEFORE_EVENT
AFTER_EVENT
在程序流中 由系统在合 适点处调用 。
根据要作的 授权检查， 可以将子程 序 AUTHORITY_CHECK_<table>放置在报表 中（参见 *逻辑数据库 的授权检查*（页 309））。
关于数据库 程序的详细 信息，参见 *编辑数据库 程序*（页 312）。

逻辑数据库 的授权检查

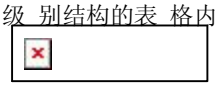
通常，可以 在下列数据 库程序的子 程序或者报 表的处理块 中包括授权 检查：
数据库程 序中的子程 序：
- PAI
- AUTHORITY_CHECK_<table>

报表中的 事件关键字 ：
- AT SELECTION-SCREEN
- AT SELECTION-SCREEN ON <fname>
- AT SELECTION-SCREEN ON END OF <fname>
- GET <table>

将授权检查 放置在数据 库程序还是 放在报表中 取决于：
逻辑数据 库的结构； 例如，如果 在运行时读 取包含公司 代码字段的 行，则应该 只检查公司代码授权。
性能；例 如在 SELELCT 循环中不执 行重复检查 。
在任何情况 下，数据库 访问和应用 逻辑的分离 允许在逻辑 数据库程序 中集中地编 码**所有**授权 。这使维护 大的编码系 统更加容易 。

逻辑数据库 的性能

因为在所有 有关的 ABAP/4 报表中更改 逻辑数据库 将立即生效 ，所以通过 集中优化可 以提高程序 库中的不同 对象的响应 时间。
通过允许用 户精确指定 系统从数据 库中读取哪 个表格条目 可以获得最 大的性能提 高。为此， 可以在数据 库程序中使 用下列技术 ：
选择标准 和参数（参 见 *使用选择屏幕*（页 **错误！链接无效。**）），可能带 默认值和值 列表。
动态选择 （参见 *逻辑数据库 选择*（页 368））。
匹配码选 择（参见 *编辑匹配码 选择*（页 314））。
查看从数 据库读取的 条目或者将 其存储在内 表中。
另外，应该 在早期进行 授权检查， 即尽可能在 选择屏幕处 理期间而不 是等到数据 选择处理期 间。
因为它们依 赖于读取的 数据，因此 没有优化的 步骤规则。 试图优化响 应时间时应 该知道以下 各点：
在不同级 别结构的表 格内容之间 的数字关系 十分重要。



如果某一级 别结构的数 据库表格的 某行包括下 一级别数据 库表格的某 行（情况 A），则其它 优化可能对 于比例 1:100 或者 1:1000 更有意义（ 情况 B ）。

在情况 A 中，通过使 用数据库视 图可以提高 响应时间（ 关于视图的 详细信息，参见文档 *ABAP/4 词典*（页 **Error! Not a valid link.**））。

在情况 B 中，可以使 用内表。首 先从数据库 中将数据读 到内表中（ 参见*将数据恋　工作区中*（页 **错误！链接无效。**）），然后在 逻辑数据库 中通过 LOOP/ENDLOOP 处理内表。

在情况 B 中，使用光 标处理选择 行也很有用 （参见*使用光标从 数据库表中 寥へ*（页 **错误！链接无效。**））。

一些 ABAP/4 报表只参阅 带 GET 语句的层次 结构的一部 分，而其它 报表访问结 构中的所有节点。在这 种情况下， 提高单个报 表性能有下 列选择：
- 在逻辑 数据库程序 中，使用表 格 GET_EVENTS。 使用逻辑数 据库生成报 表之后，对 于结构的每 个节点，该 报表表示报 表中每条 GET 语句是否发 生（参见 *编辑数据库 程序*（页 312））。
- 通过在 选择 INCLUDE 中使用

```
SELECTION-SCREEN FIELD SELECTION FOR TABLE <table>.
```

语句，可以 为字段选择 指定逻辑数 据库中的数 据库表格 <table> （参见 SELECTION-SCREEN 的关键字文 档）。在报 表中，可以 使用合适的 GET 语句（参见 *外在地指　数据库表的 字* （页 **错误！链接无效。**））。

逻辑数据库 示例

假定逻辑数 据库 HKS 有下列**结构** ：

假定在选择 包含程序中 定义下列选 择标准：

```
SELECT-OPTIONS: SLIFNR   FOR LFA1-LIFNR,
                SBUKRS   FOR LFB1-BUKRS,
                SGJAHR   FOR LFC1-GJAHR,
                SBELNR   FOR BKPF-BELNR.
```

下面是完整 的**数据库程 序**：

```
*---------------------------------------------------------------*
* DATABASE PROGRAM OF THE LOGICAL DATABASE HKS
*---------------------------------------------------------------*
PROGRAM SAPDBHKS DEFINING DATABASE HKS.
TABLES: LFA1,
        LFB1,
        LFC1,
        BKPF.

*---------------------------------------------------------------*
* Initialize selection screen (process before PBO)
*---------------------------------------------------------------*
FORM INIT.
  ....
ENDFORM.                                "INIT

*---------------------------------------------------------------*
* PBO of selection screen (process always after ENTER)
*---------------------------------------------------------------*
FORM PBO.
  ....
ENDFORM.                                "PBO

*---------------------------------------------------------------*
* PAI of selection screen (process always after ENTER)
*---------------------------------------------------------------*
FORM PAI USING FNAME MARK.
  CASE FNAME.
    WHEN 'SLIFNR'.
      ....
    WHEN 'SBUKRS'.
      ....
    WHEN 'SGJAHR'.
      ....
    WHEN 'SBELNR'.
      ....
  ENDCASE.
ENDFORM.                                "PAI

*---------------------------------------------------------------*
* Call event GET LFA1
*---------------------------------------------------------------*
FORM PUT_LFA1.
  SELECT * FROM LFA1
           WHERE LIFNR      IN SLIFNR.
    PUT LFA1.
  ENDSELECT.
ENDFORM.                                "PUT_LFA1
```

```
*---------------------------------------------------------------*
* Call event GET LFB1
*---------------------------------------------------------------*
FORM PUT_LFB1.
  SELECT * FROM LFB1
           WHERE LIFNR      =  LFA1-LIFNR
             AND BUKRS      IN SBULRS.
    PUT LFB1.
  ENDSELECT.
ENDFORM.                                "PUT_LFB1

*---------------------------------------------------------------*
* Call event GET LFC1
*---------------------------------------------------------------*
FORM PUT_LFC1.
  SELECT * FROM LFC1
           WHERE LIFNR      =  LFA1-LIFNR
             AND BUKRS      =  LFB1-BUKRS
             AND GJAHR      IN SGJAHR.
    PUT LFC1.
  ENDSELECT.
ENDFORM.                                "PUT_LFC1

*---------------------------------------------------------------*
* Call event GET BKPF
*---------------------------------------------------------------*
FORM PUT_BKPF.
  SELECT * FROM BKPF
           WHERE BUKRS      =  LFB1-BUKRS
             AND BELNR      IN SBELNR
             AND GJAHR      IN SGJAHR.
    PUT BKPF.
  ENDSELECT.
ENDFORM.                                "PUT_BKPF
```

PROGRAM 语句包含附 加 DEFINING DATABASE HKS，它 将数据库程 序定义为属 于逻辑数据 库 HKS。

用生成适当 表格工作区 的 TABLES 语句声明结 构的节点。 因为这些表 格工作区由数据库程序 和相关的报 表**共享**，因 此它们变为 在逻辑数据 库和报表之 间数据传输的界面。

子程序 INIT 和 PBO 初始化选择 屏幕。

子程序 PAI 对选择屏幕 上的用户输 入进行授权 检查。也可 能进行可行 性和值范围检查。如果 检查产生负 输出，则出 现适当的错 误对话框并 且相关字段 再次准备接收输入。

根据由用户 输入的选择 标准 PUT_<table> 子程序读取 数据库表格 并且在报表 中调用相关 处理块。调 用子程序的 顺序由逻辑 数据库的结 构决定。

下图显示数 据库结构决 定的程序流 ：

创建和维护 逻辑数据库

创建或维护 逻辑数据库 的事务是 SE36 或者 SLDB。 要进行该进 程，请选择 “工具 -> ABAP/4 工作台 -> 开发 -> 编程环境 -> 逻辑数据库 ”。
随后出现如 下显示的初 始屏幕：

在“逻辑数 据库”字段 中，输入逻 辑数据库的 名称。
要通过“显 示”或“更 改”显示或 更改逻辑数 据库，请选 择逻辑数据 库子对象。 要创建逻辑 数据库，请 选择“创建 ”
下列主题介 绍如何创建 和维护逻辑 数据库：

创建逻辑数 据库

创建逻辑数 据库时，系 统将承担大 部分工作：

通过在图 形编辑器中 定义其结构 ，可以定义 逻辑数据库 的最重要的 特征。
定义结构 后，系统自 动建议选择 包含程序。
最后，系 统使用结构 和选择生成 数据库程序 。

要全部自动 生成 ABAP/4 语句，应该 以下列次序 处理子组件 ：
要创建新逻 辑数据库， 在初始屏幕 上选择“创 建”。

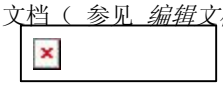
1. 在随 后的对话框 中，输入短 文本，用“ 创建”确认 并指定开发 级别（关于 开发级别的 详细信息， 参见*指ü 绦蜘 性*（页 **错误！链接无效。**））。

   如果要更改 短文本或者 以其它语言 维护，可以 选择“细节 -> 短文本”以 后再完成此 目的。
2. 指定 结构的根节 点，例如

   用“创建” 确认。
3. 逻辑 数据库现在 具有单个节 点结构。

   可以按 *更改结构*（页 106）中的描述扩 展该结构。
4. 保存 该结构。系 统在此基础 上自动建议 选择包含程 序。
5. 选择 “转向 -> 选择 ”并且按 *编辑选择*（页 312）中的描述维 护包含程序 。
6. 保存 选择并且选 择“转向 -> 数据库程序 ”。确认下 列对话框：

   然后系统根 据结构和选 择条件生成 数据库程序 。已经定义 了所有必须 的子程序并 且也生成了 SELECT 语句的大部 分 WHERE 条件。按 *编辑数据库 程序*（页 312）. 中的描述维 护该数据库程序。
7. 最后 ，可以维护 下列可选子 对象：
   - 选择文 本（参见 *编辑选择文 本*（页 313））
   - 匹配码 选择（参见 *编辑匹配码 选择*（页 314））
   - 文档（ 参见 *编辑文档*（页 307））



虽然从系统 获得的支持 可以帮助您 快速创建可 执行的逻辑 数据库，但 必须自己注意诸如提高 性能等细节 。

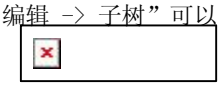
处理结构

要显示或更 改逻辑数据 库结构，选 择初始屏幕 上的“结构 ”。
可以执行下 列操作

显示结构
要显示结构 ，请在初始 屏幕上选择 “结构”和 “显示”。
系统在左上 角最高级别 上（结构的 根节点）显 示节点名称 。每个下层 节点显示在 前一节点的 下方，向右 缩排，相同 级别的节点 显示在相同 列中。

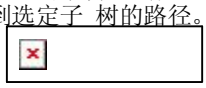
逻辑数据库 KDF 有如下结构 ：

通过单击可 以展开或者 折叠某个节 点的次级层 次结构。折 叠节点时， 只显示子树 的根并加前 缀 +。
通过选 择“编辑 -> 子树”可以 达此目的。



折叠 KDF 中节点 LFB1 的所有下层 节点：

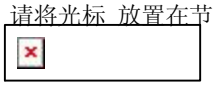
要显示特定 节点的子树 ，请通过光 标选择节点 并且选择“ 编辑 -> 子树 -> 显示”。结 果屏幕的首 行显示了从 根到选定子 树的路径。 也可以单击 该路径的节 点以显示相 应子树。



要显示节点 的单个字段 ，将光标放 置在节点上 并且双击或 者选定“显 示表格字段 ”。

KDF 的 LFAS 节点的字段 为：

更改结构
要更改现有 结构，请在 初始屏幕上 选定“结构 ”和“更改 ”或者在结 构的显示屏 幕上选定“ 数据库 -> 显示 <-> 更改”。
要更改现有 节点，请相 应放置光标 并且选定“ 编辑 -> 节点 -> 更改”。
要在光标位 置次级级别 或者相同级 别上创建新 节点，请选 定“编辑 -> 节点 -> 创建”。
要选定/取 消子树，请 选定“编辑 -> 子树 -> 选择/撤消 选择”。
要将结构中 的子树移到 光标指定的 位置，请选 定“编辑 -> 子树 -> 重分配”。
要删除子树 ，请将光标 放置在节点 上或者选定 它并选择“ 编辑 -> 子树 -> 删除”。

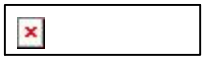


在下列屏幕 上，用户首 先将光标放 在节点 BKPF 上并且选择 “编辑 -> 子树 -> 选择”。然 后将光标放 置在 LFB5 上并选择“ 编辑 -> 节点 -> 创建”。

如同所看到 的，选定了 BKPF 的子树。现 在可以对其 重新赋值。 然后，出现 名为“创建 节点”的对 话框，可以 在其中创建 LFB5 以下的和相 同级别的节 点。

编辑选择

要编辑逻辑 数据库的选 择屏幕，请 在初始屏幕 上选择“选 择”和“更 改”。这将 进入包含程 序 DB<dba>SEL 的编辑器。 <dba> 是逻辑数据 库的名称。



因为对于数 据库程序和 其它相关的 程序，系统 **不能**自动将 此包含程序 组合进数据库程序中， 所以**不能**使 用 INCLUDE 语句达到此 目的。

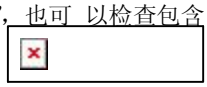
如果以前未 做选择，则 系统自动为 所有结构中 的数据库表 格生成关联 SELECT-OPTIONS 语句并且为所有关键字 字段建议选 择标准（根 据 ABAP/4 字典）。然 后必须为这 些选择标准 赋名。为此 ，必须为包 含程序中的 每个“?” 输入最多 8 个字符的名 称，并且删 除语句前的 注释符“*” 。
也建议匹配 码选择的 PARAMETERS 语句。
除建议的选 择标准之外 ，可以根据 需要通过下 列元素扩展 选择屏幕：

通过 PARAMETERS 语句及其附 加项，可以 参看可能用 到的附加参 数，例如， 要控制程序 流（参见 *为变量了 输入字* （页 **错误！链接无效。**））。

在包含程序 DB<dba>SEL 中，**必须**使 用 PARAMETERS 语句的附加 项 FOR TABLE。 当生成选择 屏幕时，该 项保证系统 只考虑与报 表的 TABLES 语句声明的 表格相链接 的参数。

通过 SELECTION-SCREEN 语句可以设 计选择屏幕 的格式和在 *格式化选择屏幕* (页 **错误！链接无效。**).中指定的 附加项。
语句
SELECTION-SCREEN DYNAMIC SELECTIONS FOR TABLE <table>.

为动态选择 定义数据库 表格（参见 *选择屏幕语逻辑数据库*（页 **错误！链接无效。**））。
语句
SELECTION-SCREEN FIELD SELECTION FOR TABLE <table>.
为字段选择 定义数据库 表格（参见 *外在地指　数据库表的 字*　（页 **错误！链接无效。**））。
语句
SELECTION-SCREEN BEGIN|END OF VERSION
和
SELECTION-SCREEN EXCLUDE
允许创建选 择屏幕的不 同版本（参 见 SELECTION-SCREEN 的关键字文 档）。
如果逻辑数 据库已经存 在带选择的 包含程序， 则可以通过 选择“细节 -> 生成 -> 选择”用定 义的系统程 序将其覆盖 。然后必须 在对话框中 确认它。
要检查包含 程序 DB<dba>SEL 的语法错误 ，请在初始 屏幕上选择 “检查”。 如果在编辑 数据库程序 时选择“检 查”，也可 以检查包含 程序的语法 错误。



假定逻辑数 据库 HKS 有下列结构 ：

建议的包含 程序如下：

```
*---------------------------------------------------------------------*
* INCLUDE DBHKSSEL
* 自动包括在 数据库程序 中。
*---------------------------------------------------------------------*
*
* If the source code is automatically generated,
* please perform the following steps:
* 1. Replace ? by suitable names (at most 8 characters).
* 2. Activate SELECT-OPTIONS and PARAMETERS (delete
*    asterisks).
* 3. Save source code.
* 4. Edit database program
*
* Hint: Syntax check is not possible in this include
* 因为在数据 库程序的语 法检查期间 对它进行了 检查
*
*---------------------------------------------------------------------*

* SELECT-OPTIONS: ?        FOR LFA1-LIFNR.
* Parameter for matchcode selection (ABAP/4 Dictionary
* structure MCPARAMS):
* PARAMETERS p_mc AS MATCHCODE STRUCTURE FOR TABLE LFA1.

* SELECT-OPTIONS:
*                 ?        FOR LFB1-LIFNR,
*                 ?        FOR LFB1-BUKRS.

* SELECT-OPTIONS:
*                 ?        FOR BKPF-BUKRS,
*                 ?        FOR BKPF-BELNR,
*                 ?        FOR BKPF-GJAHR.

* SELECT-OPTIONS:
*                 ?        FOR LFC1-LIFNR,
*                 ?        FOR LFC1-BUKRS,
*                 ?        FOR LFC1-GJAHR.
```

例如，可以 按下述方法 修改自动生 成的包含程 序：

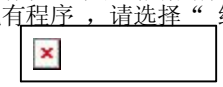
```
* Selection criteria:

  SELECT-OPTIONS: SLIFNR   FOR LFA1-LIFNR.

  SELECT-OPTIONS: SBUKRS   FOR LFB1-BUKRS.

  SELECT-OPTIONS: SGJAHR   FOR LFC1-GJAHR.

  SELECT-OPTIONS: SBELNR   FOR BKPF-BELNR.

* Self-defined parameters:

  PARAMETERS PDATE LIKE SY-DATUM FOR TABLE BKPF.
```

```
* Dynamic selections for LFA1 and LFB1:
  SELECTION-SCREEN DYNAMIC SELECTIONS FOR TABLE: LFA1, LFB1.
* Field selection for LFC1:
  SELECTION-SCREEN FIELD SELECTION FOR TABLE: LFC1.
```

此处，从可 用的选择标 准中进行选 择并且给定 其名称。声 明附加参数 PDATE 并链接到表 格 BKPF。 为表格 LFA1 和 LFB1 定义动态选 择。为表格 LFC1 定义字段选 择。

编辑数据库 程序

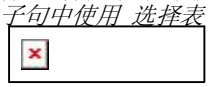
要编辑逻辑 数据库访问 程序，请在 初始屏幕上 选择“数据 库程序”和 “更改”。 进入程序 SAPDB<dba> 的编辑器。 <dba> 是逻辑数据 库的名称。
如果程序不 存在，系统 根据结构和 选择包含程 序中的信息 自动建议生 成的程序。 如果要用建 议的程序覆 盖现有程序 ，请选择“ 细节 -> 生成 -> 程序”。然 后必须在对 话框中确认 该操作。



不能更改预 定义的 TABLES 语句和自动 生成子程序 的预定义名 称。但是， 可以为数据 库访问定义 其它子程序 或更改 ABAP/4 语句。

自动生成的 SELECT 语句的 WHERE 从句包含有 关表格中的 **所有关键字 字段**。对于 比较字段， 有下列选项 ：

如果选择 包含程序为 字段定义选 择标准，则 将字段和相 关选择表格 与 IN 相比较，正 如 *在 WHERE 子句中使用 选择表* (页 **错误！链接无效。**) 中所述。



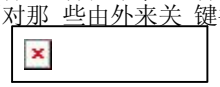
如果选择包 含程序包括

```
SELECT-OPTIONS SLIFNR FOR LFA1-LIFNR.
```

那么下列子 程序自动出 现在数据库 程序中：

```
FORM PUT_LFA1.
  SELECT * FROM LFA1
           WHERE LIFNR IN SLIFNR.
    PUT LFA1.
  ENDSELECT.
ENDFORM.
```

如果选择 包含程序**不** 包含字段的 选择标准， 但该字段的 上级表格的 关键字字段 是外来关键字，则对那 些由外来关 键字相关性 链接的字段 进行比较。



在逻辑数据 库结构中， LFB1 是 LFA1 的下级节点 并且 LFA1 中的关键字 字段 LIFNR 是 LFB1 的外来关键 字。

如果选择包 含程序包括

```
SELECT-OPTIONS SBUKRS FOR LFB1-BUKRS.
```

在数据库程 序中自动出 现下列子程 序：

```
FORM PUT_LFB1.
  SELECT * FROM LFB1
           WHERE LIFNR =  LFA1-LIFNR.
           AND   BUKRS IN SBUKRS.
    PUT LFB1.
  ENDSELECT.
ENDFORM.
```

此处，为 LFB1 中的字段 BUKRS 定义选择标 准 SBUKRS。 WHERE 条件 BUKRS IN SBUKRS 如上述显示 。不定义字 段 LIFNR 的选择标准 ，但是 LIFNR 在 LFA1 中也是关键 字字段。因 此，通过使 用 AND 的条件LIFNR = LFA1-LIFNR 扩展 WHERE 从句。

下面是自动 生成 ABAP/4 程序的完整 示例：

假定逻辑数 据库 HKS 有下列结构 ：

让包含程序 DBHKSSEL 包括下列编 码选择：

```
  SELECT-OPTIONS: SLIFNR   FOR LFA1-LIFNR.
  SELECT-OPTIONS: SBUKRS   FOR LFB1-BUKRS.
  SELECT-OPTIONS: SGJAHR   FOR LFC1-GJAHR.
  SELECT-OPTIONS: SBELNR   FOR BKPF-BELNR.
```

自动生成数 据库程序的 最重要行列 在下面。程 序也包含某 些用户和性 能提示，如注释行，但 此处不包括 这些。

```
*---------------------------------------------------------------------*
* DATABASE PROGRAM OF LOGICAL DATABASE HKS
*---------------------------------------------------------------------*

PROGRAM SAPDBHKS DEFINING DATABASE HKS.

TABLES: LFA1,
        LFB1,
        BKPF,
        LFC1.

*---------------------------------------------------------------------*
* BEFORE_EVENT will be called before event EVENT
* Possible values for EVENT: 'START-OF-SELECTION'
*---------------------------------------------------------------------*
* FORM BEFORE_EVENT USING EVENT.
*   CASE EVENT.
*     WHEN 'START-OF-SELECTION'
*
*   ENDCASE.
* ENDFORM.                              "BEFORE_EVENT

*---------------------------------------------------------------------*
* AFTER_EVENT will be called after event EVENT
* Possible values for EVENT: 'END-OF-SELECTION'
*---------------------------------------------------------------------*
* FORM AFTER_EVENT USING EVENT.
*   CASE EVENT.
*     WHEN 'END-OF-SELECTION'
*
*   ENDCASE.
* ENDFORM.                              "AFTER_EVENT

*---------------------------------------------------------------------*
* Initialize selection screen (processed before PBO)
*---------------------------------------------------------------------*
FORM INIT.

ENDFORM.                              "INIT.

*---------------------------------------------------------------------*
* PBO of selection screen (always processed after ENTER)
*---------------------------------------------------------------------*
FORM PBO.

ENDFORM.                              "PBO.
```

```
*---------------------------------------------------------------------*
* PAI of selection screen (always processed after ENTER)
*---------------------------------------------------------------------*
FORM PAI USING FNAME MARK.
* CASE FNAME.
*   WHEN 'SLIFNR  '.
*   WHEN 'SBUKRS  '.
*   WHEN 'SBELNR  '.
*   WHEN 'SGJAHR  '.
*   WHEN '*'.
* ENDCASE.
ENDFORM.                                "PAI

*---------------------------------------------------------------------*
* Call event GET LFA1
*---------------------------------------------------------------------*
FORM PUT_LFA1.
* SELECT * FROM LFA1
*          INTO TABLE ?
*          WHERE LIFNR      IN SLIFNR.
    PUT LFA1.
* ENDSELECT.
ENDFORM.                                "PUT_LFA1

*---------------------------------------------------------------------*
* Call event GET LFB1
*---------------------------------------------------------------------*
FORM PUT_LFB1.
* SELECT * FROM LFB1
*          INTO TABLE ?
*          WHERE LIFNR      =  LFA1-LIFNR
*            AND BUKRS      IN SBUKRS.
    PUT LFB1.
* ENDSELECT.
ENDFORM.                                "PUT_LFB1

*---------------------------------------------------------------------*
* Call event GET BKPF
*---------------------------------------------------------------------*
FORM PUT_BKPF.
* SELECT * FROM BKPF
*          INTO TABLE ?
*          WHERE BUKRS      =  LFB1-BUKRS
*            AND BELNR      IN SBELNR
*            AND GJAHR      =  ?.
    PUT BKPF.
* ENDSELECT.
ENDFORM.                                "PUT_BKPF

*---------------------------------------------------------------------*
* Call event GET LFC1
*---------------------------------------------------------------------*
FORM PUT_LFC1.
* SELECT * FROM LFC1
*          INTO TABLE ?
*          WHERE LIFNR      =  LFB1-LIFNR
*            AND BUKRS      =  LFB1-BUKRS
*            AND GJAHR      IN SGJAHR.
    PUT LFC1.
* ENDSELECT.
ENDFORM.                                "PUT_LFC1

*---------------------------------------------------------------------*
* Authority check for table LFA1
*---------------------------------------------------------------------*
* FORM AUTHORITYCHECK_LFA1.
*   AUTHORITY-CHECK ...
* ENDFORM.                              "AUTHORITYCHECK_LFA1
```

```
*---------------------------------------------------------------------*
* Authority check for table LFB1
*---------------------------------------------------------------------*
* FORM AUTHORITYCHECK_LFB1.
*   AUTHORITY-CHECK ...
* ENDFORM.                                "AUTHORITYCHECK_LFB1

*---------------------------------------------------------------------*
* Authority check for table BKPF
*---------------------------------------------------------------------*
* FORM AUTHORITYCHECK_BKPF.
*   AUTHORITY-CHECK ...
* ENDFORM.                                "AUTHORITYCHECK_BKPF

*---------------------------------------------------------------------*
* Authority check for table LFC1
*---------------------------------------------------------------------*
* FORM AUTHORITYCHECK_LFC1.
*   AUTHORITY-CHECK ...
* ENDFORM.                                "AUTHORITYCHECK_LFC1

*---------------------------------------------------------------------*
* PUT_HKS_MATCHCODE.
* Processed when matchcode selection is used,
* 即用户输进 PARAMETERS p_mc AS MATCHCODE STRUCTURE.
*---------------------------------------------------------------------*
* FORM PUT_HKS_MATCHCODE.
* ENDFORM.                                " PUT_HKS_MATCHCODE
```

子程序 BEFORE_EVENT、 AFTER_EVENT 和 PUT_<dba>_MATCHCODE 在后面讲述 。关于其余 子程序的说 明，参见 *逻辑数据库 的数据库程 序* (页 369) 。

必须删除强 制语句之外 的 ABAP/4 语句之前的 注释星号“ *”并且用 有效的 ABAP/4 语法组件取 代问号（?） 。然后保存 并检查程序 。同时自动 检查选择包 含程序。
某些表格和 其它子程序 在运行时总 是可用的并 且愿意的话 就可以使用 ：

内表 GET_EVENT 说明报表中 使用哪个逻 辑数据库节 点。生成报 表时生成它 们的操作如 下：

```
DATA: BEGIN OF GET_EVENTS OCCURS 10,
        NODE(10),
        KIND,
      END OF GET_EVENTS.
```

表格包含字 段 NODE 中逻辑数据 库(每行一 个)的所有 节点的名称 。字段 KIND 指定在报表 中是否和如 何使用该节 点（即在 GET 或 GET-LATE 中）：

- KIND = 'X': 在 GET 和 GET LATE 中定址表格 。
- KIND = 'G': 只在 GET 中定址表格 。
- KIND = 'L': 只在 GET LATE 中定址表格 。
- KIND = 'P': 既不在 GET 也不在 GET LATE 中定址表格 。但是，在 GET 或 GET LATE 中定址子程 序表格。
- KIND = ' ': 既不在 GET 也不在 GET LATE 中定址表格 。也不定址 子程序表格 。

逻辑数据 库中的子程 序 BEFORE_EVENT 作为注释生 成（见上例 ）。可以修 改并且通过 删除星号（ *）激活它 。
在参数 EVENT 中指定的事 件处理之前 调用 BEFORE_EVENT。
逻辑数据 库中的子程 序 AFTER_EVENT 作为注释生 成（见上例 ）。可以修 改并可通过 删除星号（ *）激活它 。
在参数 EVENT 指定的事件 处理之前调 用 AFTER_EVENT。
逻辑数据 库中的子程 序 PUT_<dba>_MATCHCODE 作为注释生 成。可以修 改并且通过 删除星号（ *）激活它 。<dba> 是逻辑数据 库的名称（ 见上例）。
详细信息， 参见 *编辑匹配码 选择* (页 314) 。

编辑选择文 本

选择文本， 即在选择屏 幕上带输入 字段显示的 文本，通常 是选择标准 的名称。可 以在逻辑数 据库程序之 外(或在选 择的包含程 序之外)维 护这些文本 。这使的逻 辑数据库程 序独立于语 言。参见 *处理文本摘要* (页 **错误！链接无效。**) 。
要用每种登 录语言编辑 选择文本， 请在初始屏 幕上选择“ 选择文本” 和“更改” 。如果登录 语言与初始 语言（即生 成逻辑数据 库的登录语 言）不同， 则出现对话 框提示是否 想更改初始 语言中的文 本摘要或者 是否要更改 初始语言。 通过更改初 始语言，可 以用任何语 言维护选择 文本。

假定逻辑数 据库 HKS 有下列结构 ：

让包含程序 DBHKSSEL 包含下列编 码选择：

```
SELECT-OPTIONS: SLIFNR    FOR LFA1-LIFNR.
  SELECT-OPTIONS: SBUKRS    FOR LFB1-BUKRS.
  SELECT-OPTIONS: SGJAHR    FOR LFC1-GJAHR.
  SELECT-OPTIONS: SBELNR    FOR BKPF-BELNR.
```

当选择“选 择文本”时 ，出现下列 屏幕：

可以如下完 成文本字段 ：

假定下列报 表链接到逻 辑数据库 HKS：

```
REPORT SAPMZTST.
TABLES: LFA1, LFB1, LFC1, BKPF.
GET LFA1.
  ........
GET LFB1.
  ........
GET LFC1.
  ........
GET BKPF.
  ........
```

结果选择屏 幕如下：

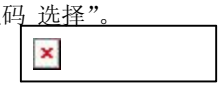
如果随后用 登录语言德 语“D”登 录到 R/3 系统并且调 用报表，则 在选择屏幕 上显示选择 标准名称。 因此，通过 逻辑数据库 的初始屏幕 进入选择文 本的维护， 在对话框中 将初始语言 更改为德语 并且输入下 列文本：

当再次以德 语登录时， 报表的选择 屏幕显示如 下，但继续 对英语用户 显示英语选择文本：

编辑匹配码 选择

要为逻辑数 据库显示、 更改或创建 匹配码，请 在初始屏幕 上选择“匹 配码选择” 。这将进入 更改匹配码 对象的合适 屏幕。可以 从值列表中 选择匹配码 对象（用 **F4** ）或者删除 现有对象：

要找到需要 那个匹配码 对象做为逻 辑数据库的 匹配代码， 必须知道逻 辑数据库结 构的哪个表 格在匹配码 对象视图中 发生。关于 编辑匹配码 的详细信息 ，参见文档 *ABAP/4 词典*（页 **Error! Not a valid link.**）。
当为逻辑数 据库选定匹 配码对象后 ，可以通过 在选择包含 程序的说明 中包括附加 项 AS MATCHCODE STRUCTURE 的参数向用 户提供匹配 码（参见 PARAMETERS 关键字文档 ）。在选择 屏幕上，这 导致显示组 框“匹配码 选择”。



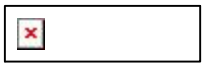
逻辑数据库 KDF 包含匹配码 选择 KRED。 选择包含程 序包括下列 行：

```
PARAMETERS KD_INDEX AS MATCHCODE STRUCTURE FOR TABLE LFA1.
```

如果调用链 接到 KDF 的报表，将 在选择屏幕 上显示下列 框：

选择屏幕上 组框“匹配 码选择”的 输入字段是 “匹配码 ID”（输 入示例“D” ）和“搜索 字符串”（ 输入示例“ ...EDW..” ）。
系统评估用 户输入并选 择带关键字 字段的合适 的匹配码记 录。然后， 这使这些记 录可用于内 表<dba>_MC 中的数据库 程序。<dba> 是逻辑数据 库的名称。 使用 <dba>_MC 的关键字， 该子程序必 须选择数据 并且通过 PUT <root> 语句触发事 件 GET <root>。 <root> 是根节点的 名称。由于 PUT 语句的属性 ，必须为该 任务使用适 当的子程序 PUT_<root>....。

内表 <dba>_MC 和其它自动 生成表格的 结构显示为 数据库程序 的自动生成 源代码中的 注释。在源 代码中也说 明这些表格 的使用方法 。



假定逻辑数 据库 HZS 有根节点 KNA1。

让包含程序 DBHZSSEL 包括下列行 ：

```
SELECT-OPTIONS: SKUNNR FOR KNA1-KUNNR.

PARAMETERS P_MC AS MATCHCODE STRUCTURE FOR TABLE KNA1.
```

DEBI 指定为匹配 码对象。

数据库出现 的源代码现 在包括更多 的注释行， 这些注释行 表示除 *编辑数据库 程序* (页 Error! Not a valid link.) *编辑数据库 程序* (页 379) 下所列的之 外，创建了 下列表格和 字段：

内表 HZS_MC：

在 START-OF-SELECTION 之后，该表 格包含匹配 码记录的关 键字字段， 这些匹配代码记录与选 择屏幕上的 搜索字符串 相匹配。它 有如下结构 ：

```
DATA: BEGIN OF HZS_MC OCCURS 1000,
            KNA1_MANDT              LIKE KNA1-MANDT,
            KNA1_KUNNR              LIKE KNA1-KUNNR,
          END   OF HZS_MC.
```

内表 MC_FIELDS：

该表格包含 在匹配码选 择期间赋值 的字段（只 有在匹配码 选择期间所 有字段不赋值才是重要 的）。其结 构为：

```
DATA: BEGIN OF MC_FIELDS OCCURS 10.
            INCLUDE STRUCTURE RSMCFIELDS.
    DATA: END   OF MC_FIELDS.
```

如果通过匹 配码界面将 值赋予 MC_FIELDS-FIELDNAME 中的字段， 则字段 MC_FIELDS-SUPPLIED 不等于 SPACE。

内表 MC_TABLES：

在匹配码选 择期间给这 些表格值赋 （只有当匹 配码对象包 含不同表格 的字段时才是重要的） 。其结构如 下：

```
DATA: BEGIN OF MC_TABLES OCCURS 10.
            INCLUDE STRUCTURE RSMCTABS.
    DATA: END   OF MC_TABLES.
```

如果通过匹 配码界面将 值赋给 MC_FIELDS-FIELDNAME 中的字段， 则字段 MC_FIELDS-SUPPLIED 不等于 SPACE。

字段 MC_EVENTS：

该字段为 200 字节长。MC_EVENTS 中的每个字 节都代表逻 辑数据库结 构中的一个 表格（例如 ，首字符代 表根节点） 。对于表格 单个位置的 内容有下列 意义：

- 'X': 报表中通过 GET 语句定址表 格并且通过 匹配码给关 键字字段赋 值。
- 'R': 报表中通过 GET 语句定址表 格但不通过 匹配码给关 键字字段赋 值。
- 'M': 报表中不通 过 GET 语句定址表 格，但通过 匹配码给关 键字字段赋 值。
- ' ':报表中 不通过 GET 语句定址表 格，也不通 过匹配码给 关键字字段 赋值。

例如，注释 出的子程序 PUT_HZS_MC （参见 *编辑数据库程序*（页 379）），为了使 用内表 HZS_MC 中的匹配码 记录，可以 如下修改和 激活：

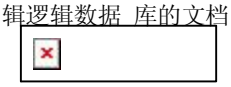
```
FORM PUT_HZS_MC.
  IF MC_EVENTS(1) NE SPACE.
    READ TABLE GET_EVENTS WITH KEY 'KNA1'.
    IF SY-SUBRC = 0 AND GET_EVENTS-KIND NE SPACE.
      SELECT * FROM KNA1 FOR ALL ENTRIES IN HZS_MC.
                        WHERE KUNNR = HZS_MC-KNA1_KUNNR
        PERFORM PUT_KNA1_MC.
      ENDSELECT.
    ENDIF.
  ENDIF.
ENDFORM.

FORM PUT_KNA1_MC.
  PUT KNA1.
ENDFORM.
```

使用表格 GET_EVENTS （参见 *编辑数据库程序*（页 379））检查链接 报表是否包 含 KNA1 的 GET 语句或下级 节点。根据 结果，在 KNA1 （包含满足 表格 HZS_MC 中条件的行 ）上执行 SELECT 循环。（关 于 WHERE 条件的详细 信息，参见 *运行时指 行选择的条 件*（页 **错误!链接无效。**））。因为 PUT 语句只可以 这样使用，所以该 SELECT 循环调用执 行 PUT KNA1 语句的 PUT_KNA1_MC 子程序。

也可以使用 优化性能的 匹配码选择 ：
根据使用 和填充的表 格和字段， 内表 GET_EVENTS、 MC_FIELDS 和 MC_TABLES 以及字段字 符串 MC_EVENTS 允许您在数 据库程序中 编码不同的 的数据库访 问。例如， 可以在内表 中使用视图 并且收集所 读取的记录 。然后可以 通过 LOOP/ENDLOOP 处理这些内 表并且触发 适当的 GET 事件。

编辑文档

要显示或编 辑逻辑数据 库的文档， 请选择初始 屏幕上的“ 文档”和“ 显示”或者 “更改”。



假定 HKS 是新的逻辑 数据库。当 创建结构、 选择包含程 序和数据库 程序时，可 以创建文档 。如果选择 初始屏幕上 的“更改”，则出现 SAP 编辑器。请 进行下列操 作：

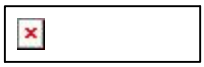
如果随后选 择“显示” ，则在初始 屏幕上出现 HKS 的下列文档 ：

其它编辑选 项

初始屏幕也 提供下列编 辑选项：

编辑数据模 型

要选定属于 逻辑数据库 表格的视图 和实体，请 选择初始屏 幕上的“转 向 -> 数据模型 -> 视图和实体 ”。对于结 构中的每个 表格，在随 后出现的屏 幕上显示所 有视图，该 视图至少部 分指向该表 格。通过选择适 当的复选框 选择视图。 这样做时， 也自动选择 了显示在相 同行的实体 。
要显示带选 定项之间所 有相关性的 图形，请选 择“图形” 。



假定逻辑数 据库包含三 个节点：LFA1、 LFB1 和 LFC1。
如果选择“ 转向 -> 数据模型 -> 视图和实体 ”，则出现 下面的屏幕 ：

如果按此处 所示选定复 选框，然后 选择“图形 ”，则 SAP 网络编辑器 显示如下：

检查逻辑数 据库

要检查逻辑 数据库是否 正确和完整 ，请选择初 始屏幕上的 “检查”。
在随后出现 的屏幕上显 示下列检查 ：

可使用该检 查决定哪个 子对象可用 或正确。

复制逻辑数 据库

要复制逻辑 数据库，请 选择初始屏 幕上的“复 制”。
在后续对话 框中通过覆 盖总是源数 据库名称的 标准值输入 目标数据库 的名称。

删除逻辑数 据库

要删除逻辑 数据库，请 选择初始屏 幕上的“删 除”。
如果逻辑数 据库链接到 报表，即程 序属性中指 定的，则得 到有关程序 名称的信息 。这种情况 下，因为受 影响的程序 会不正确， 因此不能删 除该数据库 ：

如果没有报 表使用该逻 辑数据库， 则可以删除 它。

第三部分　编写 ABAP/4 事务

目　　录

®

第一章 对话编程简 介

概览

内容

本节介绍对 话编程。将 描述下列主 题：
事务 (页 357)
事务示例 (页 317)
动态程序 (页 356)
ABAP/4 模块池 (页 357)
动态程序和 ABAP/4 模块池之间 的交互作用 (页 357)

事务

事务是实施 与用户对话 的程序。在 典型的对话 中，用户可 以在系统显 示的屏幕上 输入或请求 信息。作为 对用户输入 或请求的响 应，程序执 行适当的动 作：转到下 一个屏幕， 显示输出或 更改数据库 。

某个旅行社 想预定航班 。旅行社在 屏幕上输入 相应的数据 。系统或者 确认期望的请求，也就 是说，旅行 社可以预定 航班，并且 旅客可以在 期望的日期 、坐预定的 座位到选定 的目的地， 或者系统显 示航班已经 预定的信息 。

要满足这样 的请求，对 话程序必须 提供：
友好的用 户界面
对用户输 入的数据格 式化并作一 致性检查
输入错误 的简易纠正 法
通过将数 据存储在数 据库中来访 问数据。
ABAP/4 提供不同的 工具和语言 要素以满足 对话程序中 上述要求。

对话程序的 结构

对话程序包 含下面的基 本组件：

屏幕（动 态程序）
SAP 系统中的每 个对话都是 由动态程序 控制的。动 态程序包含 一个屏幕和 它的流逻辑 并且精确控 制一个对话 步骤。流逻 辑决定在显 示屏幕前(输出前的 PBO 处理)和接 收到用户在 屏幕上制作 的条目后（ 输入后的 PAI 处理）进行 的处理。
在屏幕绘制 器中固定的 屏幕格式决 定输入/输 出字段、文 本字段和诸 如单选按钮 以及复选框之类的图形 元素的位置 。另外，菜 单绘制器允 许在一个或 多个 GUI 状态下存储 菜单、图表 、按钮及功 能键。动 态程序和 GUI 状态都指向 控制动态程 序和 GUI 状态的运行 顺序的 ABAP/4 程序。
ABAP/4 模块池
每个动态程 序都精确指 向一个 ABAP/4 对话程序。 这样的对话 程序也称为 模块池，原 因是它包含 交互模块。 动态程序的 流逻辑包含 从相应模块 池中调用模 块。在 PBO 事件中调用 的交互模块 依照上下文 进行屏幕模 板的准备，例如通过设 置字段内容 或通过禁止 显示不需要 的字段等方 式准备屏幕 模板。在 PAI 事件中调用 的交互模块 用于检查用 户的输入并 触发合适的 对话步骤， 例如更新任 务。
从一个事务 中调用的所 有动态程序 都指向一个 公用模块池 。模块池的 动态程序是 编号的。默 认情况下， 系统为每个 动态程序存 储下一个要 显示的动态 程序。该动 态程序序列 或链可以是 线性的也可 以是环型的 。在动态程 序链中甚至 可以调用另 一个动态程 序链，并且 在处理它之 后返回原始 链。

传输字段数 据

如何在屏幕 上显示 ABAP/4 模块中已知 的字段？如 何将屏幕上 的用户条目 传输给模块 ？与报表编程相反，**不 能**用 WRITE 语句将字段 数据写入屏 幕。系统通 过屏幕字段 名和 ABAP/4 变量名的比 较来代替数 据传输。如 果两个名称 相同，它将 屏幕字段值 传输给 ABAP/4 程序字段， 反之亦然。 在显示屏幕 之前和之后 立即发生上 述过程。

字段属性
在屏幕制作 器中定义所 有动态程序 屏幕字段的 属性。如果 屏幕中的字 段名对应于 某个 ABAP/4 词典字段的 名称，则系 统自动建立 这两个字段 之间的参考 。因此自动 从 ABAP/4 词典中复制 动态程序中的大量字段 属性。字段 属性和数据 元素以及分 配的词典字 段形成了动 态程序在对 话（屏幕字 段的自动格 式检查，自 动值范围检 查，联机帮 助，等等） 中执行的标 准函数的基 础。

错误对话
动态程序处 理器的另一 个任务是管 理错误对话 。使用 ABAP/4 词典的检查 表格自动检 查输入数据 或者通过 ABAP/4 程序本身检 查。动态程 序处理器包 括接收屏幕 中的错误消 息并将屏幕 返回用户。 消息可以是 上下文敏感 的，也就是 说，系统用 当前字段内 容替换消息 文本中的占 位符。另外 ，只有内容 与错误有关 并且可以纠 正错误的字 段才可以接 收输入。关 于错误处理 的详细信息 ，请参见 *处理错误和消息* (页 **Error! Not a valid link.**) 。

数据一致性
要在复杂的 应用程序中 保持数据一 致性，ABAP/4 提供优化数 据库更新的 技术，对它 的操作独立 于下面的数 据库并符合 对话编程的 特殊请求。关于数据库 更新的详细 信息,参见 *编程数据库 更新* (页 **Error! Not a valid link.**) 。
为了说明事 务的概念和 用法，提供 下面的事务 示例。

事务示例

事务 TZ10（ 开发级别 SDWA） 是随系统传 递的。该事 务仅包含一 个动态程序 。用户可以 输入一个航空公司的标 识和航班号 以请求航班 信息：

如果用户选 择“显示” ，则系统从 数据库中检 索请求的数 据并显示它 ：

事务 TZ10 的结构在下 面的主题中 描述：

动态程序

每个屏幕都 包含用于显 示或请求信 息的字段。 字段可以是 文本串、输 入或输出字 段、单选按 钮、复选框 或按钮。事 务 TZ10 的屏幕仅包 含文本和输 入/输出字 段。
SAP 动态程序包 含几个组件 ：

流逻辑： 为屏幕调用 ABAP/4 模块。
屏幕格式 ： 屏幕上文 本、字段、 按钮等的位 置。
屏幕属性 ： 屏幕号、 后续屏幕号 及其它。
字段属性 ： 屏幕上单 独字段的属 性定义。

用户在屏幕 制作器中创 建或编辑动 态程序的所 有组件。要 调用屏幕制 作器，在对 象浏览器中 创建一个动 态程序或双 击现有的动 态程序，然 后对象浏览 器调用屏幕 制作器。在 那里可以输 入新动态程 序的流逻辑 。按相应的 按钮可以维 护“屏幕属 性”、转到 全屏幕编辑 器或者选择 按钮“字段 列表”更改 字段属性。关于屏幕制 作器的详细 信息,参见 文档 *ABAP/4 开发工作台：工具* (页 **Error! Not a valid link.**) 。

屏幕属性
从用户的观 点看，事务 是一系列一 个接一个显 示的屏幕。 如何决定该 顺序呢？事 务的属性决 定了要显示 的第一个屏 幕。单个动 态程序的属 性决定了在 当前屏幕之 后要显示的 屏幕。也可 以在 ABAP/4 程序中动态 设置后续屏 幕号。
因为没有调 用后续屏幕 ，所以本示 例不需要更 改屏幕属性 。

布局
选择“全屏 ”转到屏幕 编辑器。在 此可以决定 屏幕的布局 。对于事务 TZ10， 可以从 ABAP/4 词典的表格 SPFLI 中复制所需 的字段。关 于全屏幕编 辑器的详细 信息，参见 *ABAP/4 开发工作台：工具* (页 **Error! Not a valid link.**)

字段属性
要显示和修 改单独字段 的属性（输 入/输出字 段、请求的 输入、可能 条目按钮、 不可见等等 ），请使用 “字段列表 ”。
字段“ 公司”(SPFLI-CARRID) 和“航班号 ” (SPFLI-CONNID)定 义为输入/输 出字段。所 有其它字段仅用于航班 数据的输出 。

流逻辑
动态程序的 流控制代码 包括句法上 与 ABAP/4 相似的一些 语句。但不 能在 ABAP/4 中使用流控 制关键字， 反之亦然。 在屏幕制作 器中输入流 控制代码作 为动态程序 的一个组件 。
事务 TZ10 的动态程序 的流控制如 下所示：

```
PROCESS BEFORE OUTPUT.
    MODULE SET_STATUS_0100.
*
PROCESS AFTER INPUT
    MODULE USER_COMMAND_0100.
```

PROCESS 语句为动态 程序命名事 件类型，MODULE 语句告诉系 统为该事件 调用的 ABAP/4 例程。在该 示例中每个 事件 PBO 和 PAI 只有一个 MODULE。 但一个事件 可以包含带 有多个关键 字的语句。 （流控制语 言仅包含几 种语句类型 。最重要的 是 MODULE、 FIELD、 CHAIN、 LOOP、 CALL SUBSCREEN） 。

要显示流逻 辑中语句的 语法信息， 请选择流逻 辑编辑器中 的“实用程序 -> 帮助...” 。在后面的 对话窗口中 ， 标记“流 逻辑关键字 ”，输入所 需关键字的 名称，然后 按 ENTER。

ABAP/4 模块池

在对象浏览 器中，模块 池代码属于 下列类别之 一：

全局字段 ：模块池中 所有模块都 可使用的数 据声明
PBO 模块：显示 屏幕前调用 的模块
PAI 模块：响应 用户输入而 调用的模块
子程序： 可以在模块 池中任何位 置调用的子 程序

默认情况下 ，系统将模 块池分成一 个或多个包 含程序。一 个包含程序 可以包含相 同类型的多 个模块（仅 PBO 模块或 PAI 模块）。然 后主程序包 含一系列将 模块链接到 模块池的 INCLUDE 语句：

```
*&---------------------------------------------------------------------*
*& Module pool      SAPMTZ10                                           *
*&                                                                     *
*&---------------------------------------------------------------------*
*&                                                                     *
*&     Display data of Table SPFLI                                     *
*&                                                                     *
*&---------------------------------------------------------------------*

* Global data
INCLUDE MTZ10TOP.

* PAI modules
INCLUDE MTZ10I01.

* PBO modules
INCLUDE MTZ10001.
```

在 ABAP/4 编辑器中， 选择“编辑 -> 其它功能 -> EXPAND 包含程序” 可以显示隐 藏在 INCLUDE 语句后的代 码。带有所 有扩展的 INCLUDE 语句的模块 池显示如下 ：

```
*&---------------------------------------------------------------------*
*& Module pool      SAPMTZ10                                           *
*&            FUNCTION: Display data from Table SPFLI                  *
*&                                                                     *
*&---------------------------------------------------------------------*
*---------------------------------------------------------------------*
*   INCLUDE MTZ10TOP (This is the TOP include:                        *
*           the TOP module contains global data declarations)         *
*---------------------------------------------------------------------*
PROGRAM  SAPMTZ10.
  TABLES: SPFLI.

  DATA OK_CODE(4).
*---------------------------------------------------------------------*
*   INCLUDE MTZ10I01  (This is a PAI include.)                        *
*---------------------------------------------------------------------*
*&---------------------------------------------------------------------*
*&      Module   USER_COMMAND_0100  INPUT
*&---------------------------------------------------------------------*
*       Retrieve data from SPFLI or leave transaction                 *
*---------------------------------------------------------------------*
MODULE USER_COMMAND_0100 INPUT.
  CASE OK_CODE.
     WHEN 'SHOW'.
       CLEAR OK_CODE.
       SELECT SINGLE * FROM SPFLI WHERE CARRID = SPFLI-CARRID
                                  AND   CONNID = SPFLI-CONNID.
     WHEN SPACE.
     WHEN OTHERS.
       CLEAR OK_CODE.
       SET SCREEN 0. LEAVE SCREEN.
  ENDCASE.
ENDMODULE.
*---------------------------------------------------------------------*
*   INCLUDE MTZ10001  (This is a PBO include.)                        *
*---------------------------------------------------------------------*
*&---------------------------------------------------------------------*
*&      Module STATUS_0100
*&---------------------------------------------------------------------*
*    Specify GUI status and title for screen 100                      *
*---------------------------------------------------------------------*
MODULE STATUS_0100.
 SET PF-STATUS 'TZ0100'.
 SET TITLEBAR '100'.
ENDMODULE.
```

使用 ABAP/4 词典集中存 储频繁使用 的数据声明 。在词典中 定义的对象 在整个系统 中是已知的 。可以通过 任何应用程 序访问活动 词典定义。 在词典中定 义的数据可 以包括在屏 幕中或由 ABAP/4 程序使用。 用户使用 TABLES、 STRUCTURE、 LIKE 语句和其它 语句，在事 务的 TOP 模块中声明 全局数据。事务 TZ10 访问表格 SPFLI 的词典定义 ，提供所需 的航班数据 显示。如果 TOP 包含程序包 括 TABLES: SPFLI 声明，那么 模块池中的 所有模块都 可以访问表 格 SPFLI 的表格字段 。
PAI 模块 USER_COMMAND_0100 检查用户激 活了哪个按 钮 (**CASE OK_CODE**)。 事务 TZ10 中的“显示 ”按钮有函 数代码‘SHOW’ 。(关于处 理函数代码 的详细信息 ，参见 *处理用户请求* (页 **Error! Not a valid link.**))。然后程 序在 SPFLI 数据库中选 定对应于用 户输入数据 的记录。WHERE 条件通过比 较字段 **SPFLI-CARRID** 及 **SPFLI-CONNID** 与数据库关 键字段 **CARRID** 及 **CONNID 来决定匹配 记录。**一旦 找到了匹配 记录，数据 库将所有伴 随 SPFLI 的字段都传 输给程序表 格。
再次显示屏 幕时，在屏 幕的输出字 段中出现完 整的信息。 系统自动显 示这些字段 ，原因是 ABAP/4 字段名 **SPFLI-CARRID** 及 **SPFLI-CONNID 和屏幕字段 的相同。**
在事务 TZ10 的 PBO 模块 STATUS_0100 中，屏幕 100 接收 GUI 状态（使用 SET PF-STATUS） 和 GUI 标题（使用 SET TITLEBAR）：

```
SET PF-STATUS 'TZ0100'.
SET TITLEBAR '100'.
```

“GUI 状态”是用 于某屏幕的 界面元素的 子集。状态 包括当前事 务所需的元 素。事务的 GUI 状态可能由 下列元素组 成：

GUI 标题是显示 在窗口标题 栏中的屏幕 标题。与 GUI 状态相反， GUI 状态可用于 多个屏幕， 而 GUI 标题只属于 一个屏幕。
要创建并编 辑 GUI 状态和 GUI 标题，请使 用菜单制作 器。要启动 菜单制作器 ，请在对象 浏览器的对 象列表中创 建 GUI 状态或 GUI 标题（或双 击现有的状 态或标题） 。
关于菜单制 作器的详细 信息，参见 文档 *ABAP/4 开发工作台：工具* (页 **Error! Not a valid link.**)。

动态程序和 ABAP/4 模块池之间 的交互作用

最简单格式 的事务是屏 幕和 ABAP/4 例程的集合 ，由对话处 理器控制和 执行。对话 处理器一个 接一个地处 理屏幕，由 此为每个屏 幕触发适当 的 ABAP/4 处理。
对于每个屏 幕，系统执 行包含相应 ABAP/4 处理的的流 逻辑。控制 从屏幕流逻 辑传给 ABAP/4 代 码，然后再 返回。
例如，事务 TZ10 的事件序列 如下所示：

1. 在 PBO 事件中，语 句 **MODULE STATUS_0100** 将控制传给 相应的 ABAP/4 模块。
   在 ABAP/4 模块池中， 要显示的屏 幕接收一个 菜单接口。
2. 在处 理模块 STATUS_0100 后，控制返 回流逻辑。
   对于 PBO 事件，不需 要进一步处 理。系统显 示屏幕并从 用户接收条 目。条目是 ：
   - 字段“ 公司”和“ 航班号”的 值。
   - 四个字 符的功能代 码可以判断 用户激活的 是哪一个按 钮。
3. 用户 输入触发 PAI 事件。第一 个 PAI 语句将控制 传给 ABAP/4 模块 USER_COMMAND_0100。
   模块 USER_COMMAND_0100 处理用户的 请求。在本 示例中只可 能有一种请 求：显示指 定航班的航 班数据。ABAP/4 语句 SELECT 从数据库中 检索数据并 显示它。
4. 处理 **MODULE USER_COMMAND_0100** 之后，控制 返回 PAI，并 终止对话。

第二章 处理用户请 求

概览

在事务中， 用户通过选 定屏幕按钮 、菜单功能 、功能键、 工具栏按钮 、图标或 ENTER 键进行请求 。

内容

对话程序如 何处理用户 请求？
执行动作时 ，系统触发 PROCESS AFTER INPUT 事件。传送 的数据包括 由用户输入 的字段屏幕 数据和功能 代码。功能 代码是个技 术名称，在 屏幕制作器 或菜单制作 器中将它分 配给菜单项 、按钮、ENTER 键或屏幕的 功能键。PAI 模块中的内 部工作区（ ok-code） 根据功能代 码执行相应 的动作。
下面的章节 用 TZ20 事务（开发 类 SDWA） 显示如何用 按钮控制事 务执行的方 式。

本章最后两 节介绍如何 使用单选按 钮和复选框 。
下列主题介 绍如何在事 务中包括用 户功能：
用功能代码 编程 (页 392)
用单选按钮 编程 (页 464)
用复选框编 程 (页 413)

用功能代码 编程

下列主题介 绍如何使用 功能代码：
设置功能代 码 (页 463)
处理功能代 码 (页 424)
处理字段选 择 (页 412)
共享 GUI-状 态 (页 421)

设置功能代 码

要在对话程 序中处理用 户请求，必 须将功能代 码分配给屏 幕制作器或 菜单制作器 中相关的屏 幕和窗口元 素。

在屏幕制 作器中：
屏幕按钮
在菜单制 作器中：
菜单功能
功能键
工具栏按钮 和图标
ENTER 键

屏幕制作器
在屏幕制作 器中，可以 通过设置按 钮字段的“ 功能代码” 属性来分配 功能代码。
在 TZ20 事务中为“ 显示”按钮 在屏幕制作 器中设置了 功能代码（ FctCode） FTCH。

菜单制作器
在 SAP 系统中，程 序的用户界 面包括在菜 单制作器中 定义的一个 或多个 GUI 状态和一个 GUI 标题。GUI 状态是一组 事务在给定 时间所需的 动态元素。 GUI 标题是显示 在窗口会话 顶部的标题 栏文本。为了处理用 户在菜单制 作器中定义 的屏幕元素 的功能代码 ，程序必须 指定将出现 在 PBO 处的屏幕 GUI 状态。
在 TZ20 事务中将“ 其它航班” 按钮定义为 TD0100 GUI 状态中菜单 栏的一部分 。“其它航 班”按钮在 菜单制作器 中有功能代 码 NEW。

可以用关键 字 SET PF-STATUS 在对话程序 中设置 GUI 状态：

```
SET PF-STATUS <GUI_status>.
```

<GUI_status> 是可以有 8 个 字符的字符 串，它可以 是文字（在 单引号内） 或变量。
按如下方式 用关键字 SET TITLEBAR 设置屏幕或 对话框的标 题：

```
SET TITLEBAR <title> WITH <p1> <p2> <p3> <p4>.
```

该语句可以 接收多达四 个参数值。 用户在标题 定义中输入 相应的占位 符（用 &）。例如 ，在“维护 表格”屏幕 中，可能这 样定义标题 ：

```
SET TITLEBAR 'ABC' WITH 'Customer'.
```

如果标题“ ABC”为 ：

```
Maintain Table &
```

那么窗口标 题将如下所 示：

```
Maintain Table Customer
```

系统在运行 时用指定的 表格名称填 充 &。

> 如果不为屏 幕设置用户 界面，则系 统用已经为 以前的屏幕 设置的相同 GUI 元素显示该 屏幕。如果 事务没有以 前的屏幕， 或者如果还 没有设置任 何 GUI 状态，则屏幕将没有用 户界面。

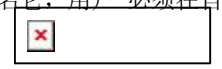
要得到如何 在相应屏幕 的流逻辑中 编写自己的 PBO 模块的示例 ，参见事务 TZ20。 单个 PBO 模块设置 GUI 界面和事务 的标题。TD0100 GUI 状态包含“ 其它航班” 按钮的定义 。

```
MODULE STATUS_0100 OUTPUT.
    SET PF-STATUS 'TD0100'.
    SET TITLEBAR '100'.
ENDMODULE.
```

关于定义 GUI 状态和标题 的详细信息 ，参见 *ABAP/4 开发工作台：工具* （页 **Error! Not a valid link.**）

处理功能代 码

当用户在事 务中选定功 能时，系统 将功能代码 复制到称为 OK_CODE 的特殊设计 的工作字段 。该字段在 ABAP/4 模块池中是 全局的。然 后可以在相 应的 PAI 模块中给 OK_CODE 赋值。
不管功能代 码是来自屏 幕的按钮、 菜单选项、 功能键或其 它 GUI 元素，它通 常是以实际 相同的方式 传输的。
在屏幕制作 器中，如果 显示屏幕的 字段列表， 则 OK_CODE 字段通常是 列表的最后 一个字段。 该字段开始 是无名的， 并且字段类 型为“OK” 。用户可以 给该字段任 何名称，但 它通常称为 OK_CODE。
不管如何命 名它，用户 必须在自己 的模块池中 包括一个将 它作为全局 字段的定义 ：

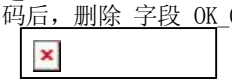


```
PROGRAM SAPMTZ20.

  ...
  DATA: OK_CODE(4).
```

OK_CODE 的程序声明 是“四个” 字符长。
要得到如何 处理功能代 码的示例， 参看事务 TZ20。

```
*---------------------------------------------------------------------*
*   PAI Modules MTZ20I01                                              *
*&--------------------------- ----------------------------*
*&      Module  USER_COMMAND_0100  INPUT
*&---------------------------------------------------------------------*
MODULE USER_COMMAND_0100 INPUT.
            CASE OK_CODE.
                 WHEN 'FTCH'.
                           SELECT SINGLE * FROM SPFLI WHERE CARRID = SPFLI-CARRID
                                                      AND CONNID = SPFLI-CONNID.
                           IF SY-SUBRC NE 0. CLEAR SPFLI. ENDIF.
                           CLEAR OK_CODE.
                 WHEN 'NEW'.
                           CLEAR: SPFLI, OK_CODE.
                 WHEN 'CANC'.
                           CLEAR OK_CODE.
                           SET SCREEN 0. LEAVE SCREEN.
                 WHEN 'EXIT'.
                           CLEAR OK_CODE.
                           SET SCREEN 0. LEAVE SCREEN.
                 WHEN 'BACK'.
                           CLEAR OK_CODE.
                           SET SCREEN 0. LEAVE SCREEN.
            ENDCASE.
ENDMODULE.
```

例如，如果 用户选定“ 显示”按钮 ，则将 FTCH 功能代码复 制到 OK_CODE 内部工作字 段中。然后 在 PAI 模块中检查 OK_CODE， 如果它包含 FTCH， 则为了获得 要显示的数 据执行 SELECT 。
SET SCREEN 和 LEAVE SCREEN 语句控制屏 幕流。规范 SET SCREEN 0 告诉系统回 到整个调用 级别。在这 个小程序中 ，这意味着 一起从事务 中退出。调 用级别在 *控制屏幕流 程* (页 **Error! Not a valid link.**) 中描述。
处理功能代 码后，删除 字段 OK_CODE 的内容以避 免任何意外 的功能选择 。



也可以从 SY-UCOMM 变量中确认 当前在程序 中活动的功 能代码。

处理字段选 择

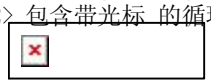
事务经常让 用户选定字 段来请求功 能。在某些 情况下，仅 字段选择就 可以触发功 能。在其它 情况下，用 户单击一个 字段，然后 选定菜单选 项、功能键 或其它按钮 。
要获得程序 的通知或字 段选择：
仅字段选 择
用户提供有 功能代码的 **F2** 键时，系统 在用户选定 字段时将代 码传送给程 序。在这方 面，双击字 段和用 **F2** 单击它是相 同的。系统 通过为 **F2** 功能代码触 发 PAI 来响应两种 选择。
字段选择 加上功能请 求
和通常一样 提供有功能 代码的屏幕 元素。当用 户选定或按 元素时，将 功能代码传 送给程序。当程序接收 相关的功能 代码时，它 首先需要知 道选定的字 段。使用 GET CURSOR 命令：

```
GET CURSOR FIELD <field name>.
```

因此用户可 以找到光标 所在的字段 。系统将变 量 <field name> 设置为当前 光标所在位 置的屏幕字段名。
注意如果用 户在非字段 区域上双击 ，则 GET CURSOR 返回空的字 段名参数。
当处理单步 循环时，使 用 LINE 参数找出包 含光标的循 环块行。

```
GET CURSOR FIELD <field name> LINE <field2>.
```

变量 <field2> 包含带光标 的循环块行 号。



字段选择加 上功能请求 ：

```
GET CURSOR FIELD selfield.
IF SELFIELD NOT SPACE.
  CASE OK_CODE.
   WHEN 'SELE'. PERFORM DISPLAY_FIELD_INFO USING SELFIELD.
   WHEN 'CHNG'. PERFORM MODIFY_FIELD USING SELFIELD.
   WHEN 'DELE'. PERFORM DELETE_FIELD USING SELFIELD.
  ENDCASE.
  CLEAR OK_CODE.
ENDIF.
```

共享 GUI-状 态

在 ABAP/4 开发工作台 上，屏幕和 用户界面是 相互独立的 。例如，用 户可以为多 个屏幕使用 相同的界面 。但是，一 些屏幕的活 动功能可以 比其它屏幕 少。可以使 用相同的状 态并取消一 个或多个屏 幕功能来取 代为这些屏 幕创建新的 GUI-状 态。
系统提供附 加的关键字 撤消 GUI-状 态中的特定 菜单功能。 该语句的格 式是：

```
SET PF-STATUS <GUI status> EXCLUDING <function codes>.
```

如果想撤消 某个功能， 则 **<function codes>** 参数是一种 类型 C 字段。在引 号中输入适 当功能的名称。要撤消 多个功能， 请将这些功 能代码放在 内部表中。 内部表必须 有如下的结 构：

```
DATA: BEGIN OF INTTAB OCCURS 20,
        FUNCTION (4),
      END OF INTTAB.
```

用 EXCLUDING 指定功能撤 消所有分配 给给定代码 的相关 GUI-项 （菜单项、 按钮和功能 代码）。撤 消的按钮是 不显示的。 撤消的菜单 功能（与用 鼠标右键显 示在列表中 的功能键分 配相同）也 会显示，但 仅以变灰的 形式显示。 如果用户选 定它们，什 么都不会发 生。

用单选按钮 编程

单选按钮是 简单输入字 段。它们没 有关联的功 能代码，因 此它们自己 不能触发 PAI 事件。但是 ，用它们解 释用户请求 ，还需要了 解一些特殊 的情况。

设置单选按 钮

将单选按钮 添加到屏幕 上时，屏幕 制作器自动 为按钮创建 一个字符的 屏幕字段。 用户应该在 自己的 ABAP/4 模块中为单 选按钮声明 一个相应的 字符变量。

```
DATA: RADIO1, RADIO2, RADIO3.
```

默认情况下 ，系统将组 中的第一个 单选按钮设 置为打开。

检查单选按 钮选择

单选按钮是 属于逻辑组 的唯一选择 按钮。如果 用户单击一 个，则系统 自动取消选 定组中的所 有其它按钮 。如果设置 了单选按钮 ，则它的值 为‘X’。 用户不必处 理单选按钮 的取消选定 ：屏幕接口 自动完成这 项工作。

在相关 PAI 事件处，用 户检查可以 假定一次只 打开一个按 钮。

```
IF RADIO1 NE SPACE:
    PERFORM PROCESS_RADIO1.
ELSEIF RADIO2 NE SPACE:
    PERFORM PROCESS_RADIO2.
ELSE.
    PERFORM PROCESS_RADIO3.
ENDIF.
```

在程序中设 置单选按钮 的值

用户单击单 选按钮时， 屏幕接口关 闭组中所有 其它按钮。

但是，如果 用户想自己 （从模块池 中）设置单 选按钮，屏 幕接口对取 消选定剩余 按钮无效。 必须自己取 消选定。

```
RADIO1 = SPACE.
RADIO2 = 'X'.
RADIO3 = SPACE.
```

用复选框编 程

复选框不是 唯一的选择 按钮。用户 可以一次打 开多个。

将复选框添 加到屏幕时 ，屏幕制作 器自动为每 个框创建一 个字符的屏 幕字段。用 户应该在 ABAP/4 模块中为框 声明一个相 应的字符变 量。

```
DATA: CHECK1, CHECK2.
```

默认情况下 ，所有复选 框初始化为 关闭状态。 如果想把它 们初始化为 打开，则给 它们赋值：

```
CHECK1 = 'X'.
CHECK2 = 'X'.
```

要查看用户 选定的框， 可以查询其 值不等于空 的框。

第三章 处理错误和 消息

概览

内容

当用户键入 屏幕输入时 ，事务在使 用此输入之 前必须检查 其是否合法 。SAP 系统提供了 错误处理的特征以尽可 能地简化字 段检查过程 。这些特征 包括用于编 辑错误处理 程序的关键 字，以及对 话处理运行 时间环境方 面的因素：

自动字段 检查（由系 统执行）
某些字段检 查以存储在 ABAP/4 字典上的信 息为基础， 由系统自动 执行。
FIELD 和 CHAIN 语句（在流 逻辑语言中 ）
FIELD 和 CHAIN 流逻辑语句 允许用户设 计自己的字 段检查。FIELD和 CHAIN 告诉系统正 在检查哪些 字段，以及 系统是在流 逻辑中进行 检查还是调 用ABAP/4模 块。如果发 现错误，则 系统为用户 输入一错误 对话。
MESSAGE 语句（在 ABAP/4中 ）
MESSAGE语 句（在 ABAP/4中 ）允许用户 从 ABAP/4 程序中输出 消息。ABAP/4 程序通过输 出错误消息 或警告将错 误通知系统 。相应地， 系统为用户 输入错误对 话。
错误对话 （由系统执 行）
系统或 ABAP/4 模块都可以 检测错误。 无论是哪种 情况，只要 发现错误， 系统就自动 重新显示屏 幕并输出消 息。
错误通常是 由于特定字 段。重新显 示时，引起 错误的字段 允许输入， 而其它所有 字段则禁止 输入。系统 将光标定位 于错误字段 中，并且要 求用户重新 输入。然后 重复字段检 查的过程。

有关信息由 下列主题提 供：
错误处理简 介 (页 395)
检查屏幕字 段的有效性 (页 394)
发布消息 (页 368)
示例事务： 检查字段输 入 (页 465)

错误处理简 介

在正常的对 话处理过程 中，事务逐 屏予以处理 。然而，如 果出现错误 ，则系统重 新显示出错 的屏幕。同 时显示一则 消息，并且 ，如果错误 涉及到字段 输入，那么 该字段允许 输入。(所 有其它字段保持固定值 。) 这对用 户怎样？程 序如何告诉 ABAP/4 处理器有必 要重新显示 ？
请看错误处 理示例事务 TZ31。 TZ31 (开发类 SDWA) 是用于显示 和更新航班 信息的小事 务。该事务 允许系统自 动进行字段 检查，而且 也包含指导 其它错误检 查的逻辑。
正常情况下 ，使用事务 时，总是输 入航空公司 和航班标识 符并按下 ENTER 。然后系统 以更新模式 显示所有字 段细节。要 作更改，请 键入新信息 并保存。

当输入错误 时会发生什 么情况？假 定未键入所 需的信息而 只按下 **ENTER**。（其中带“ ？ ”的字段 是所需的输 入字段。） 系统自动检 查，并向您 发送消息：

TZ31 还检查系统 忽略的事情 。例如，更 新显示时， 输入了不存 在的机场代 码会发生什 么情况？程 序向您发送 消息：

在该屏幕上 ，只能更改 机场字段。 所有其它输 入都是固定 的。当更正 机场并重新 输入时，事 务继续其它 处理。

通过用 TZ31 测试，将看 到进行几种 字段检查。 一些由系统 自动处理， 而一些由程 序处理：

要求输入 的字段有输 入吗？（**自 动**）
“航空公司 “和“航班 号”字段在 屏幕制作器 中具有要求 输入的属性 。系统自动 检查这些字 段是否从用 户获得输入 。

输入的航 空公司存在 吗？ （**自动**）
在屏幕制作 器中，“航 空公司”字 段声明为表 字段 SPFLI-CARRID。 在字典中， CARRID 字段与检查 表 SCARR 有外部关键 字关系。结 果，系统自 动检查 SCARR 中是否包含 所有 SPFLI-CARRID 的输入。

该航空公 司的航班号 存在吗？（ **ABAP/4**）
事务 TZ31 中的 ABAP/4 模块（CHECK_FLIGHT） 检查为给定 航线输入的 航班号是否 存在。

起飞/到 达城市：它 们存在吗？ （**自动**）
“起飞城市 ”字段（SPFLI-CITYFROM） 是检查表 SGEOCITY 的外部关键 字。系统自 动检查在 SGEOCITY 表中是否找 到该字段的 输入。

起飞/到 达机场：它 们存在吗？ （**ABAP/4**）
ABAP/4 模块检查输 入的机场是 否存在。

该章其余部 分讲述如何 设计处理错 误的程序:

检查屏幕字 段的有效性 (页 394)
发布消息 (页 368)

关于如何实 现事务 TZ31 的讨论，参 见：

示例事务： 检查字段输 入 (页 465)

> 如果想在系 统中使用事 务 TZ31，请记住，可 以使用数据 浏览器查找 给定航空公 司代码的有 效航班号。 要访问数据 浏览器，请 进入对象浏 览器（在工 作台中）， 然后选择“ 环境 -> 数据库浏览 器 -> 表格目录” 。

检查屏幕字 段的有效性

R/3 系统提供检 查屏幕字段 的各种方法 ：

由系统执 行的自动字 段检查
当适合进行 自动字段检 查时，使用 它最容易。 系统在存储 于 ABAP/4 字典的字段 信息的基础上进行检查 。

在屏幕流 逻辑中进行 的检查
可以在屏幕 的流逻辑中 指定某些字 段检查。使 用流逻辑语 句 FIELD...VALUES... 来完成此操作，该语句 指定屏幕字 段的可能值 列表。系统 甚至不输入 ABAP/4 模块，而检 查这些值的 屏幕字段输 入。结果， 没有任何可 用的 ABAP/4 代码就可以 设计并测试 屏幕。

在 ABAP/4 中进行检查
当自动字段 检查和 FIELD...VALUES... 流逻辑语句 不够灵活时 ，可以编码 ABAP/4 模块以执行字段检查。 要触发该模 块的调用， 则可以在屏 幕流逻辑中 编码 FIELD...MODULE... 流逻辑语句 。

详细信息， 参见下列主 题：

理解自动字 段检查 (页 423)
检查屏幕流 逻辑中的字 段 (页 398)
在 ABAP/4 中检查字段 (页 390)

理解自动字 段检查

系统在屏幕 字段中自动 执行某些有 效性检查。 在用户输入 之后和 PAI 处理开始之 前执行这些 检查。执行 的自动检查 类型如下：

要求的输 入
在屏幕制作 器中，可以 设置字段的 要求输入（ “要求输入 ”）属性。 完成此操作 之后，系统 就要求用户 在输入 PAI 处理之前对 该字段作相 应的输入。

正确数据 格式
在屏幕制作 器中，每个 字段都有数 据格式。该 格式限制了 有效输入的 种类。例如 ，DATS 字段（日期 字段）是格 式为 YYYYMMDD 的 8 字符的字符 串。所有字 符都必须是 数字。子字 符串 MM 和 DD 必须分别小 于或等于 12 和 31。对于 输入的给定 月值，系统 还检查日值 是否有效。
当用户输入 不符合数据 格式要求的 值时，系统 重新显示屏 幕直到更正 输入为止。

字段的有 效值
在词典中， 有两种方式 将字段值限 制到允许的 集合中。字 段可以与其 它表存在**外 部关键字关 系，**或者它 的域可为该 字段指定**固 定值列表**。 对外部关键 字字段，系 统检查在相 关检查表中 能否找到用 户的输入值 。对具有已 定义固定值 列表的字段 ，系统确保 用户的输入 值在列表中 。

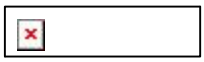
对外部关键 字，可以用 该字段存储 预定义的错 误消息。

如果字段的 有效性测试 失败，那么 ，系统重新 显示屏幕并 要求用户更 改输入。一 旦重新输入 ，则将重复 自动检查用 户在其中输 入新值的字 段。

要扩展自动 字段检查， 可以定义自 己的字段检 查。具体操 作，参见：

检查单个字 段 (页 408)

检查多个字 段 (页 454)



有时想要编 码处理(应 在系统执行 任何自动检 查之前进行)。该过程 通常处理退 出请求。（ 例如，当用 户要从事务 中返回时， 则无须在所 需字段中输 入值。）要 编码应在系 统执行自动 字段检查之 前执行的逻 辑，请使用 流逻辑命令 MODULE ... AT EXIT-COMMAND。 详细信息， 参见 *避免自动字 段检查* (页 401) 。

检查屏幕流 逻辑中的字 段

用 FIELD...VALUES 流逻辑语句 检查屏幕流 逻辑中的字 段值。使用 该语句，指 定了可能值 列表，而系 统则将字段 输入与该列 表进行对比 。该语句的 语法如下：

```
FIELD <screen field> VALUES (<value-list>)
```

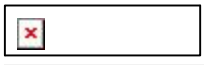
<value-list> 可以包括单 个值或值区 间。所有值 用单引号封 闭。例如：

```
FIELD SBOOK-CARRID VALUES ('AA', 'LH', 'US').
FIELD SBOOK-CONNID VALUES (BETWEEN '1' AND '10', NOT '3').
```

如果用户输 入的值不等 于列表中的 值（或不在 指定区间内 ），则系统 重新显示屏 幕要求新输 入。

下表显示 <value-list> 的所有可能 格式。

比较	语法
单 值	('<value>')
单 值的补集	(NOT '<value>')
几 个单值或补 集	('<value1>', '<value2>', NOT '<value3>')
值 区间	(BETWEEN '<value>' AND '<value>')
某 区间之外的 值	(NOT BETWEEN '<value>' AND '<value>')



要使用 FIELD...VALUES 选项,用于 比较的字段 必须具有数 据类型 CHAR 或 NUMC。还必须记住 在单引号中 给定的所有 值应大写。

在 ABAP/4 中检查字段

要在 ABAP/4 中进行字段 检查，请使 用 FIELD 和 CHAIN 流逻辑语言 语句。FIELD 语句的以下 形式允许调 用进行字段 检查的 ABAP/4 模块：

```
FIELD <field> MODULE <module>.
```

FIELD 语句可能包 含多个 MODULE 调用：

```
FIELD <field>: MODULE <module1>,
               MODULE <module2>.
```

也可以在 FIELD 语句中指定 多个字段。 当用 CHAIN 语句链接多 个字段检查 时，它特别 有用。例如 ，FIELD 语句的以下 两种形式都 是允许的：

```
CHAIN.
    FIELD <field1>.
    FIELD <field2>.
    FIELD:  <field3>, <field4>, ... <fieldn>.
    MODULE <module1>.
    MODULE <module2>.
ENDCHAIN.
```

ABAP/4 模块发现错 误时，就输 出错误消息 或警告以通 知用户。发 布这些消息 以提醒系统 需要错误对 话框。系统 重新显示屏 幕，要求用 户为出错的 字段输入新 值。所有其 它字段都不 允许输入。 如果找到错 误，那么， 屏幕就使用 CHAIN 语句重新显 示链中允许 输入的所有 字段。

有关信息在 下列主题中 提供：

检查单个字 段 (页 408)
检查多个字 段 (页 454)
有条件地调 用模块 (页 433)
避免自动字 段检查 (页 401)

检查单个字 段

FIELD...MODULE 流逻辑语句 使您将有效 性检查与特 定屏幕字段 联系起来。 FIELD...MODULE 触发对检查 特定字段的 ABAP/4 模块的调用 ，并且一发 现错误就发 送错误消息 。在这种情 况下，系统 重新显示屏 幕，并禁止 输入除指定 字段以外的 其他所有字 段。

要通知系统 已发现错误 ，ABAP/4 模块必须输 出错误消息 （类型 E）或警告 （类型 W)。例如 ：

```
**** Screen flow logic: ****
PROCESS AFTER INPUT.
    FIELD SPFLI-AIRPFROM
      MODULE CHECK_FR_AIRPORT.

**** ABAP/4 module: ****
MODULE CHECK_FR_AIRPORT INPUT.
    SELECT SINGLE * FROM SAIRPORT
                  WHERE ID = SPFLI-AIRPFROM.
    IF SY-SUBRC NE 0. MESSAGE E003 WITH SPFLI-AIRPFROM. ENDIF.
ENDMODULE
```

错误消息导 致系统重新 显示屏幕并 要求用户输 入新值。重 新显示时， 只有 SPFLI-AIRPFROM 字段可接纳 新的输入。 而其它所有 字段则不允 许输入。

有关错误消 息的详细信 息，参见*发布消息* (页 368) 。

FIELD 语句如何传 送数据

系统将数据 从屏幕传送 到 ABAP/4 程序时，对 每个屏幕显 示只传送一 次。一般都 是在系统已 经执行完自 动字段检查 之后 PAI 处理开始时 才传送。

但是也有例 外。FIELD 语句中所提 及字段的数 据传送被延 迟，直到实 际执行 FIELD 语句。如果 屏幕字段出 现在多条 FIELD 语句中，则 只在使用该 字段的第一 条 FIELD 语句中的传 送字段值。

因为在数据 从屏幕传送 之前不应在 ABAP/4 模块中使用 字段，所以 传送时间非 常重要。若 此，则 ABAP/4 字段将包含 在用户的屏 幕输入之前 就已存在的 值。该值可 能是上一屏 幕保留的值 ，也可能是 无效的值。

在多条 FIELD 语句中使用 某字段

有时需要在 多条 FIELD 语句中指定 相同字段。 这使得错误 处理更复杂 。当某模块 发现错误时 ，系统重新 显示屏幕并 用更正的输 入重新启动 PAI 处理。然而 ，系统无法 用包含该模 块的 FIELD 语句简单地 重新启动。 此出错的字 段也可能在 某条更早的 FIELD 或 CHAIN 语句中出现 过。在这种 情况

下，系 统有确定应 在何处重新 启动处理的 专门程序。 详细信息， 参见 *在错误对话 框之后重新 启动 PAI* (页 455) 。

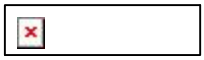
检查多个字 段

有时想作为 一组检查几 个字段。为 此，请在 FIELD 语句中包含 这些字段， 并将它们放 入 CHAIN-ENDCHAIN 块中。在示 例事务 TZ31 中使用了 CHAIN 语句：

```
**** Screen flow logic: ****
CHAIN.
     FIELD: SPFLI-CARRID, SPFLI-CONNID.
       MODULE CHECK_FLIGHT.
     ENDCHAIN.

**** ABAP/4 module: ****
MODULE CHECK_FLIGHT INPUT.
     SELECT SINGLE * FROM  SPFLI
                       WHERE     CARRID    = SPFLI-CARRID
                       AND  CONNID    = SPFLI-CONNID.
     IF SY-SUBRC NE 0.
       MESSAGE E005 WITH SPFLI-CARRID SPFLI-CONNID.
     ENDIF.
*  .....
ENDMODULE
```

在链块中， 所有字段相 互依赖。链 内发现错误 时，会重新 显示屏幕， 并允许输出 链中任何地 方发现的字 段，但仍不 允许输入任 何非链字段 。在用户重 新输入值（ 输入到链字 段之一）之 后，重新启 动 PAI 并将链中的 所有语句作 为一个单元 重新执行。
链可以包括 其它任何允 许的流逻辑 语言语句。 而且，链可 以包含多条 FIELD 语句。一般 情况下，所有 FIELD 语 句应出现在 CHAIN 块的开始处 。

```
     CHAIN.
         FIELD: A, B, C.
         FIELD: D, E, F.
         MODULE X.
         MODULE Y.
     ENDCHAIN.
```



允许将 MODULE 语句添加到 包含在 CHAIN 块中的 FIELD 语句中，但 这样实际上 并没有意义 ：

```
   CHAIN.
       FIELD F1.
       FIELD: F2, F3  MODULE m1.    "（ 在 F3后无周 期）
       MODULE m2.
   ENDCHAIN.
```

如 果模块 **m** 发现错误， 则在重新显 示时它打开 所有要输入 的链字段， 而不只是 **F2** 和 **F3**。 只有当使用 AT- 或 ON- 条件之一时 ，用该方式 使用 FIELD...MODULE 才意义。详 细信息，参 见 *条件 CHAIN 语句* (页 433) 。

有条件地调 用模块

可以在模块 调用（从屏 幕流逻辑） 中设置条件 。例如，可 以指定某模 块只在给定 字段具有值 （非初始值 ）时才被调 用：

```
     FIELD X MODULE CHECK_FIELDX ON INPUT.
```

使用 FIELD 语句的条件 形式，可以 防止不必要 的模块调用 。尤其当更 新表输入时 ，条件调用 可以极大地 提高性能。 有关信息在 下列主题中 提供：
条件 FIELD 语句 (页 393)
条件 CHAIN 语句 (页 433)

条件 FIELD 语句

添加 ON- 和 AT- 条件之后， FIELD...MODULE 流逻辑语句 就变成了条 件语句。请 使用以下条 件指定何时 应调用模块 ：

```
       ON INPUT
```

只有当字段 包含初始值 以外的值时 ，才调用 ABAP/4 模块。此初 始值由该字 段的数据类 型决定：空 格对应于字 符字段，零 对应于数字 字段。如果 用户将字段 值更改回初 始值，则 ON INPUT 不触发调用 。（与 ON REQUEST 调用相比， 此时它不触 发调用。）

ON REQUEST

只有自上一 屏幕显示以 来用户已输 入字段值时 ，才调用 ABAP/4 模块。即使 用户键入已 存在的值， 值也象已变 化了一样予 以记数。

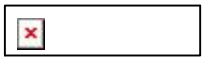
一般地，通 过任何形式 的“手工输 入”都可以 触发 ON REQUEST 条件。系统 将设置字段 的下列方式 作为手工输 入：

- 实际用 户输入
- SET PARAMETER 字段输入（ 手工和自动 两种）
- HOLD DATA 字段输入
- 用于参 数事务的参 数输入（CALL TRANSACTION...USING）
- 用于定 制系统的全 局字段（这 些为某些字 段指定自动 设置）

所有这些符 合 ON REQUEST 条件并将触 发模块调用 。

ON *-INPUT

如果用户在 字段的首字 符中已经输 入“*”，并且该字段 在屏幕制作 器中具有属 性 *-entry*，则调用 ABAP/4 模块。可以 在想只检查 某些输入类 型的例外情 况下使用该 选项。



有些条件只 适用于 FIELD 语句，而其 它条件则适 用于 CHAIN 块中的 FIELD 语句。尤其 是，ON- 和 AT- 条件在包含 多字段但又 未包含在 CHAIN 块中的 FIELD 语句中有特 殊意义，。 详细信息， 参见 *检查多个字 段* (页 400) 。

条件 CHAIN 语句

要在条件 CHAIN 中调用模块 ，有两个选 项：

ON CHAIN-INPUT

类似于 ON INPUT。 如果链中的 任一字段包 含初始值（ 空值或零） 以外的值， 则调用 ABAP/4 模块。

ON CHAIN-REQUEST

该条件功能 类似于 ON REQUEST， 但是如果链 中的任一字 段的值发生 更改，则调 用 ABAP/4 模 块。

例如：

```
CHAIN.
  FIELD: A, B, C.
  FIELD: D, E, F.
  MODULE X ON CHAIN-INPUT.
  MODULE Y.
ENDCHAIN.
```

在此，如果 字段 A、B、C、 D、E 和 F 中任何一个 具有不同于 初始值的值 ，则调用模 块 X。而始终 调用模块 Y。 如果 Y 发现错误， 则在错误对 话期间，重 新打开这六 个字段并等 待输入。

要将条件限 制到特定字 段，请将 MODULE 语句连接到 相关 FIELD 语句。

```
CHAIN.
  FIELD: A, B, C  MODULE X ON INPUT.
ENDCHAIN.
```

在该示例子 中，只有当 列表（C） 中的最后字 段包含非初 始值的值时 ，才调用模 块 X。但是 ，如果 X 发现错误， 则在错误对 话框中重新 打开这三个 字段（A， B，C）并 等待输入。

有时想要的 调用只取决 于几个字段 ，而不是其 它字段。为 明确起见， 将正在使用 的链断开并 为独立字段 组合创建独 立链是最简 单的。在每 种情况下， 都使用 ON CHAIN-INPUT 或 ON CHAIN-REQUEST。

例如：

```
CHAIN.
  FIELD: A, B, C  MODULE X ON CHAIN-REQUEST.
ENDCHAIN.
CHAIN.
  FIELD: A, B, D, E  MODULE Y ON CHAIN-REQUEST.
ENDCHAIN.
```

避免自动字 段检查

在进行自动 字段检查之 前，有时希 望系统执行 某种处理逻 辑。例如， 如果用户想 从屏幕退出 ，则无须在 需要输入的 字段中输入 数据。

流逻辑关键 字 AT EXIT-COMMAND 在流逻辑中 是 MODULE 语句的特殊 附加部分。 AT EXIT-COMMAND 允许在系统 执行自动字 段检查之前 调用模块：

```
**** Screen flow logic: ****
PROCESS AFTER INPUT.
     MODULE EXIT AT EXIT-COMMAND.
```

要使用 AT EXIT-COMMAND， 则必须将功 能类型 E 赋给菜单制 作器或屏幕 制作器中的 相关功能。 在屏幕制作 器中，调用 用于所需按 钮的属性， 并将属性“ FctType” 设置为“E” 。在菜单制 作器中，选 择“转向 -> 功能列表” ，然后在“ 类型”列中 为每个应作 为退出命令 的功能代码 输入 E。
一旦已经将 功能定义为 类型 E， 则在执行任 何字段检查 之前，可以 用 AT EXIT-COMMAND 选项告诉系统处理所有 与该功能相 关的 ABAP/4 模块。只有 当用户激活 定义为类型 E 的功能时， 才触发 AT EXIT-COMMAND 事件。

```
**** ABAP/4 module: ****
MODULE EXIT INPUT.
     CASE OK_CODE.
       WHEN 'NEW'.
              CLEAR: SPFLI, OK_CODE.
              LEAVE SCREEN.
       WHEN 'CANC'.
              CLEAR OK_CODE.
              SET SCREEN 0. LEAVE SCREEN.
     ENDCASE.
ENDMODULE.
```

正常情况下 ，MODULE...AT EXIT-COMMAND 语句倾向于 处理退出命 令 BACK、 EXIT 和 CANCEL。 为处理这些 命令而编码 的 ABAP/4 模块应包含 从屏幕或事 务退出的语 句（例如， LEAVE TO SCREEN 0）。
如果未在 AT EXIT-COMMAND 模块中终止 屏幕或事务 ，则系统象 平常一样继 续流逻辑处 理：首先执 行自动字段 检查，然后 按顺序处理 PAI 语句。

发布消息

ABAP/4 模块通过发 布错误或警 告消息来通 知系统已经 出错。这些 消息促使系 统进入同用 户的错误对 话。（错误 对话重新显 示屏幕，如 果是错误消 息则要求输 入新值。）
消息是系统 中存储为开 发对象的预 定义文本。 该文本可能 包含运行时 可用实际文 本替代的变 量子字符串 。每条消息 属于一种消 息类。
要发送模块 存储中的消 息，必须：

- 发送消息 （用 MESSAGE） 并在 PROGRAM 语句中指定 消息类。
- 如果必要 ，请在系统 中创建消息 类和消息。

以下部分提 供使用 ABAP/4 中消息的有 关信息：
发送消息 (页 424)
创建消息类 (页 392)
创建消息 (页 392)

发送消息

通常使用 ABAP/4 语句 MESSAGE 发送消息， 使用消息类 型表示错误 类型。例如 ，在以下语 句中：

```
     IF SY-SUBRC NE 0.
       MESSAGE E001.
     ENDIF.
```

消息号是 001， E 是消息类型 （错误）。
可以在消息 号前面添上 五种不同的 消息类型（ E、W、I、 A、S）。 例如，对于 消息号 001，可 以指定：

- **E001**发 送错误消息 001
- **W001**发 送警告消息 001
- **I001**发 送信息消息 001
- **A001**发 送异常终止 消息 001（A=异 常终止）
- **S001**发 送成功消息 001

输出消息时 ，产生的错 误处理取决 于消息类型 和环境。详 细信息，参 见 *对话模式的 消息处理* (页 454) 。

指定消息类
所有消息都 属于某消息 类。因此如 果在程序中 发送消息， 还必须指定 消息类。可 以在 TOP- 模块的 PROGRAM 语句中完成 此操作：

```
     PROGRAM SAPMTZ31 MESSAGE-ID A&.
```

MESSAGE-ID 参数指定用 于消息类的 两字符代码 （字母的或 数字的）。 该类保持在 模块存储中 输出的所有 消息。如果 还需要从在 PROGRAM 语句中指定 的类之外的 其它类中发 送消息，请 在消息号之 后给出不同 类：

```
MESSAGE E123(ZZ).
```

请记住，尽 管如此，用 户定义的开 发对象（如 消息类）也 应以 Y 或 Z 开始。

添加到消息 文本

消息文本可 以包含多达 四个变量文 本。创建消 息时，在文 本中用 & 代表变量， 如：

航班 &1 &2 不存在。

运行时，系 统用 MESSAGE 语句中所提 供的文本替 换串 &1 和 &2。要指 定替换文本 ，请使用 WITH 参数：

```
MESSAGE E005 WITH SPFLI-CARRID SPFLI-CONNID.
```

如果以上两 个变量包含 航班号和航 空公司，则 用户得到消 息：

航班 AA 177 不存在。

对话模式的 消息处理

正常情况下 的对话处理 中，事务逐 屏工作，并 为每个屏幕 处理 PBO 和 PAI 事件。

发布消息时 ，屏幕处理 可以采取不 同的方式， 这取决于消 息的类型。 我们已经看 到对于错误 消息，系统 重新显示屏 幕而不重复 PBO 处理。重新 显示时，消 息显示在状 态行上（屏 幕底部）并 且除那些引 起错误的字 段之外的所 有字段都不 允许输入。

一般情况下 ，消息类型 决定了消息 显示的屏幕 ，以及对话 的进行方式 。不同点在 于：

消息类型	消 息显示于	意 义
错误	当前 屏幕	所有在 FIELD 语句中提到 的屏幕字段 都允许输入 。用户必须 输入新值。 然后系统用 新值重新启 动 PAI 处理。
警告	当前屏 幕	所有在 FIELD 语句中提到 的屏幕字段 都允许输入 。但是，用 户不必输入 新值。如果 输入新值， 则系统用新 值重新启动 对该屏幕的 PAI 处理。如果 未输入新值 （用户只按下ENTER） ，则系统在 MESSAGE 语句之后立 即继续 PAI 处理。
信息	弹出式屏幕	中断当前屏 幕，并且系 统在弹出式 屏幕中自动 显示信息消 息。
成功	后 续屏幕	在下 列屏幕的 PBO 处理之后， 显示成功消 息。这样成 功消息对于 生成此屏幕 的处理无影 响。
取消	当 前屏幕	用户 按下 ENTER 时，中断当 前处理。系 统将用户返 回到 SAP 主菜单。

由于错误消 息和警告， 系统在重新 显示屏幕之 后，重新启 动屏幕的 PAI 处理。然而 ，可能并不 重复所有 PAI 处理。详细 信息，参见*在错误对话 框之后重新 启动 PAI* （页 455）。

在错误对话 框之后重新 启动 PAI

当用 FIELD 或 CHAIN 调用的 ABAP/4 模块发布警 告或错误消 息时，系统 重新显示屏 幕，从用户 获得新的输 入，然后重 新启动 PAI 处理。重新 启动的位置 （即，PAI 事件中的位 置）可能并 不会立即明了。如果与 发现错误的 FIELD 或 CHAIN 语句共享字 段，则必须 重复所有 FIELD 或 CHAIN 语句。该系统用下列步 骤确定重新 启动的位置 ：

1. 在流 逻辑中，确 定：
   - 在引起 错误的 CHAIN 或 FIELD 语句中已指 定了哪些字 段以及
   - 在最后 的屏幕显示 期间用户已 更新了哪些 字段
2. 在 PAI 事件中搜索 涉及步骤 1 中的任何字 段的第一条 FIELD 语句。
3. 在此 FIELD 语句中重新 启动处理， 否则，如果 出现在 CHAIN 中，则在 CHAIN 语句开始处 重新启动处 理。

例如，在下 列 PAI 事件中：

```
PROCESS AFTER INPUT.

    FIELD A  MODULE M.
    FIELD B MODULE N.

    CHAIN.
```

```
        FIELD: A, B, C.
        FIELD: D, E, A.
        MODULE P.
        MODULE Q.
    ENDCHAIN.

    CHAIN.
        FIELD: F.
        MODULE R.
    ENDCHAIN.

    CHAIN.
        FIELD: D.
        MODULE S.      ==> ERROR !!!
    ENDCHAIN.
```

模块 S 引起错误。 在重新显示 屏幕之后， 系统用模块 P 重新启动处 理。

创建消息类

在系统中可 以从两个地 方创建消息 类：

从“对象 类”对象列 表创建（在 对象浏览器 中）

从 ABAP/4 模块中创建 （在 ABAP/4 编辑器中）

最常用的方 法是从包含 PROGRAM 语句的 ABAP/4 模块。

1. 请输 入 MESSAGE-ID 参数值：

   PROGRAM SAPMTZ31 MESSAGE-ID A&。

2. 双击消息 标识符名称 （在此为“ A&”）。

系统相应 地生成对话 窗口以询问 是否想要创 建消息类。 请按“是” 。跳到消息 类屏幕。

3. 输入 “短文本” 说明并按“ 保存”图标 。

   这就创建了 消息类。

4. 按下 “消息”以 输入实际消 息文本。

关于输入消 息文本的方 法，参见 *创建消息* (页 392) 。

创建消息

有几种方式 跳转到消息 类以输入消 息文本。最 常用的方法 是：

1. 在程 序中使用所 需的消息号 编写 MESSAGE 语句。

   MESSAGE E001.

   如果不知道 下一个可用 消息号，请 输入任意号 码。

2. 双击消息 号。

   系统跳转到 消息类的消 息列表中。 因为消息类 可由多个用 户使用，所 以最好一次 只维护一个 消息。因此 ，只有您请 求的消息号 处于更新模 式。

   如果输入的 消息号已经 存在于系统 中，请以显 示模式将它 分支。在这 种情况下：

   a. 请按 下“下一个 可用号码” 跳转到下一 个可用消息 号。

   b. 请按 下“单个维 护”编辑此 消息号。

3. 键入 消息文本并 按下“保存 ”图标。

4. 返回 到代码段并 在必要时更 正消息号。

如果想一次 输入几条消 息文本，请 使用“维护 全部”按钮 将所有消息 转换到更新 模式。而且 不必分别保 存每条消息 文本。但是 应记住，这 样操作时， 会阻止其他 用户更新消 息类。

示例事务： 检查字段输 入

作为用户设 计的字段检 查的示例， 我们已经使 用过事务 TZ31， 该事务是每 个系统都随 带的示例事务之一。快 速查看程序 的流逻辑和 ABAP/4 代码以了解 如何将各种 检查加入程 序。

屏幕流逻辑

事务 TZ31 的屏幕流逻 辑（只用于 PAI）如 下所示：

```
*---------------------------------------------------------------------*
*   Screen 100: Flow Logic                                            *
*&---------------------------------------------------------------------*
PROCESS AFTER INPUT.
```

```
     MODULE EXIT AT EXIT-COMMAND.
*
     CHAIN.
       FIELD: SPFLI-CARRID, SPFLI-CONNID.
              MODULE CHECK_FLIGHT ON CHAIN-REQUEST.
     ENDCHAIN.
*
     FIELD SPFLI-AIRPFROM
              MODULE CHECK_FR_AIRPORT ON REQUEST.
     FIELD SPFLI-AIRPTO
              MODULE CHECK_TO_AIRPORT ON REQUEST.
*
     MODULE USER_COMMAND_0100.
```

组织流逻辑 以便：

语句 MODULE...AT EXIT-COMMAND 处理“退出 命令”并在 PAI 开始处使用 。退出命令 包括必须在 系统执行自 动字段检查 之前进行的 处理。

所有 FIELD 和 CHAIN 语句位于处 理用户命令 的逻辑之前 。此处语句 说明只有在 符合 ON REQUEST 条件时，才 执行用于检 查字段值的 ABAP/4 模块。

ABAP/4 代码

以下是与字 段检查有关 的 TZ31 模块。首先 ，EXIT 模块处理无 需用户屏幕 输入的退出 命令。它包 括 NEW 和 CANC 功能：SET SCREEN 和 LEAVE SCREEN 语句导致屏 幕或事务的 终止。这些 语句描述在*控制屏幕流程*（页 **Error! Not a valid link.**）中。

```
*&---------------------------------------------------------------------*
*&      Module  EXIT  INPUT
*&---------------------------------------------------------------------*
MODULE EXIT INPUT.
     CASE OK_CODE.
       WHEN 'NEW'.
              CLEAR: SPFLI, OK_CODE.
              LEAVE SCREEN.
       WHEN 'CANC'.
              CLEAR OK_CODE.
              SET SCREEN 0. LEAVE SCREEN.
     ENDCASE.
ENDMODULE.
```

CHECK_FLIGHT 模块确保给 定航线的航 班号存在。 否则，程序 输出消息。 请注意，由 于“规则” 和“特权” 屏幕字段在 数据库中是 用单字段表 示的，所以 ，只使用 SELECT 语句无法填 充它们。因 此模块必须 直接填充这 些屏幕字段 。

```
*&---------------------------------------------------------------------*
*&      Module  CHECK_FLIGHT  INPUT
*&---------------------------------------------------------------------*
MODULE CHECK_FLIGHT INPUT.
     SELECT SINGLE * FROM  SPFLI
                   WHERE    CARRID    = SPFLI-CARRID
                   AND  CONNID    = SPFLI-CONNID.
     IF SY-SUBRC NE 0.
       MESSAGE E005 WITH SPFLI-CARRID SPFLI-CONNID.
     ENDIF.
     CLEAR: CHARTER, REGULAR.
     IF SPFLI-FLTYPE = SPACE. REGULAR = 'X'.
     ELSE.                    CHARTER = 'X'.
     ENDIF.
ENDMODULE
```

用于机场字 段的字段检 查如下所示 ：

```
*&---------------------------------------------------------------------*
*&      Module  CHECK_AIRPORT  INPUT
*&---------------------------------------------------------------------*
MODULE CHECK_FR_AIRPORT INPUT.
     SELECT SINGLE * FROM  SAIRPORT
```

```
                        WHERE     ID    = SPFLI-AIRPFROM.
    IF SY-SUBRC NE 0. MESSAGE E003 WITH SPFLI-AIRPFROM. ENDIF.
ENDMODULE.
```

在所有字段 检查已经执 行之后，程 序可以安全 地使用屏幕 输入以处理 用户命令。 模块 USER_COMMAND_0100（ 在此未显示 ）更新数据 库并输出消 息（取消或 成功），这 取决于结果 。这些消息 类型都不会 导致屏幕处 理器进入错 误对话。

第四章 控制屏幕流 程

概览

内容

对于用户来 说，事务就 是逐个出现 的一系列屏 幕。在事务 程序中，屏 幕由一系列 的“下一个 屏幕”号链 接在一起。 定义事务时 ，要指定第 一个屏幕的 号码。然后 ，对事务中 的每个屏幕 ，可以静态 地或动态地 指定“下一 个屏幕”

静态屏幕 指示器

定义屏幕时 ，要为它指 定“下一个 屏幕”属性 。该属性给 出紧随着当 前屏幕的默 认屏幕名称 。然而，无 论何时动态 设置“下一 个屏幕”， 都会覆盖静 态属性。

动态屏幕 序列

任何屏幕都 可以设置它 自己的“下 一个屏幕”作为屏幕处 理的一部分 。做到这一 点的 ABAP/4 命令是 SET SCREEN 和 CALL SCREEN。 动态设置屏 幕时，可以 逐个地串在 一起（象在 链中一样） ，或将它们 的组插入到 当前链中。

下列主题提 供关于在事 务中处理屏 幕的信息：

屏幕流程控 制介绍 (页 402)

设置下一个 屏幕 (页 399)

调用新的屏 幕序列 (页 400)

退出当前屏 幕 (页 403)

示例事务： 设置和调用 屏幕 (页 394)

后台处理屏 幕 (页 400)

屏幕流程控 制介绍

关于在事务 中控制屏幕 流程的示例 ，请参见事 务 TZ40。 （该事务在 开发类 SDWA 中，同系统 一起传送。 ）TZ40 让用户显示 航班信息并 将更新信息 输入到显示 中。

TZ40 使用两个屏 幕和对话框 （弹出窗口 ）用于获得 用户的更新 。事务总是 显示开始的 两个屏幕（ 号码为 100 和 200 ）。第三个 （210） 只是在一定 的条件下才 显示。可能 的屏幕流程 如下：

在实践中， 用户见到如 下序列：

屏幕 100：用 户输入航班 信息，并按 **ENTER** 请求显示航 班详情。

屏幕 200：系 统在更新模 式中显示关 于航班的全 部细节。用 户通过键入 全部显示输 入所作的更 改。

屏幕 210

屏幕 210 只在用户试 图不保存而 退出屏幕 200 时才出现。 弹出内容提 醒用户保存 所作的更改 或取消（通 过指定“是 ”或“否” ）

要使该屏 幕序列成为 可能，事务 TZ40 必须能够有 条件地调用 对话框屏幕 。
ABAP/4 模块可以“ 转入”或“ 调用”下一 个屏幕。两 者的差别在 于处理完下 一个屏幕之 后用户要让 控制转到哪 里。相关的 ABAP/4 命令是：

```
SET SCREEN <screen-number>.
CALL SCREEN <screen-number>.
LEAVE SCREEN.
LEAVE TO SCREEN <screen-number>.
```

通过 SET SCREEN， 在链中当前 屏幕简单地 指定下一个 屏幕。当前 屏幕处理完 毕后，控制 转入该下一 个屏幕。从 下一个屏幕 返回到当前 屏幕不是自 动的。
使用 CALL SCREEN， 挂起当前（ 正调用的） 链，调入下 一个屏幕（ 或屏幕链） 。通过语句 LEAVE SCREEN TO SCREEN 0 调用的屏幕 就能够返回 到挂起的链 上。
详细信息， 参见：
设置下一个 屏幕 (页 399)
调用新的屏 幕序列 (页 400)
退出当前屏 幕 (页 403)

设置下一个 屏幕

每一个屏幕 都有静态的“下一个屏 幕”属性，该属性指定 跟随当前屏 幕的下一个 屏幕。可以 使用 SET SCREEN 语句覆盖该 指定：

```
SET SCREEN <screen number>.
```

SET SCREEN 告诉系统忽 略静态定义 的“下一个 屏幕”，而 使用〈屏幕号〉作为下一个 屏幕。

该覆盖是暂 时的并且对 储存在屏幕 制作器中的 属性值没有 影响。
SET SCREEN 语句只指定 下一个屏幕 ：它不中断 当前屏幕的 处理。如果 未完成当前 屏幕就要转 入下一个屏 幕，请使用 LEAVE SCREEN 语句。

注意可以使 用变量指定 下一个屏幕 号：

```
DATA: REQSCRN LIKE SY-DYNNR VALUE '100'.
MODULE SET_NEXT_SCREEN.
   SET SCREEN REQSCRN.
ENDMODULE.
```

系统字段 SY-DYNNR 总是包含当 前屏幕的号 码。

调用新的屏 幕序列

有时要将屏 幕或整个屏 幕序列插入 到事务过程 中。例如， 可能要让某 个用户从主 应用程序屏 幕中调用弹 出屏幕，以 输入辅助信 息。完成输 入以后，用 户应当能够 关闭弹出屏 幕并直接返 回到主屏幕 中先前所离 开的地方。 要做到这一 点，有两种 方法：

使用 CALL SCREEN 语句
CALL SCREEN 可以向当前 序列中插入 这样的序列 。在此对该 语句的使用 作了说明。
调用对话 模块
对话模块是 能够调用的 、不属于特 定事务的屏 幕序列。对 话模块有自 己的模块池 ，能被任何 事务调用。 关于使用对 话模块的详 细信息，参 见*调用对话模块* (页 **Error! Not a valid link.**) 。

调用新屏幕 序列的语法 是：

```
CALL SCREEN <screen number>.
```

既然该语句 实际上是挂 起当前序列 并启动新序 列，因此可 以将 CALL SCREEN 看作“堆积 ”序列。系统继续进行 新序列，直 到它完成才 恢复先前挂 起的序列。 （CALL SCREEN 之后直接恢 复处理该命 令）

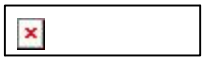
要调用屏幕 作为对话框 （弹出）， 请使用带选 项 STARTING AT、 ENDING AT 的 CALL SCREEN：

```
CALL SCREEN <screen number>
     STARTING AT <start column> <start line>
     ENDING AT   <end column>   <end line>
```

STARTING AT 和 ENDING AT 选项告诉系 统在何处定 位弹出屏幕 。屏幕本身 必须比常规 屏幕小。
在 ABAP/4 中，屏幕的 每个可堆积 序列都是“ 调用模式” 。从给定当 前序列返回 的方法使得 这一点很重 要。要终止 调用模式并 返回到挂起 链，请将“ 下一个屏幕 ”设置为 0 并退出：

```
LEAVE TO SCREEN 0.      或   SET SCREEN 0.
                             LEAVE SCREEN.
```

返回到挂起 链时，用直 接跟随初始 CALL SCREEN 语句的语句 来恢复执行 。
事务中的初 始屏幕序列 本身即是调 用模式。如 果在该序列 中 LEAVE TO SCREEN 0 （也就是说 ，没有堆积 任何其它的 调用模式） ，则从事务 中一起返回 。



一次可以堆 积最多 9 个调用模式 。

退出当前屏 幕

要终止处理 当前屏幕， 请使用：

```
LEAVE TO SCREEN <screen number>。
```

或

```
SET SCREEN <number>。
LEAVE SCREEN。
```

这两个命令 都终止处理 当前屏幕， 并直接跳到 <屏幕号>。 如果使用 SET SCREEN 而没有使用 LEAVE SCREEN， 则程序在转 入〈屏幕号〉之前完成当 前屏幕的处 理。
如果使用 LEAVE SCREEN 而之前没有 使用 SET SCREEN， 则终止当前 屏幕，并直 接转入屏幕 属性中指定 为默认的下 一个屏幕
在“调用模 式”中，特 殊屏幕号 0 （LEAVE TO SCREEN 0）导致系 统跳回到先 前的调用级 别。也就是说，如果使 用 CALL SCREEN 调用屏幕序 列，则 LEAVE TO SCREEN 0 将终止该序 列并返回调 用屏幕。如 果没有调用 过屏幕序列 ，LEAVE TO SCREEN 0 将终止事务 。关于 CALL SCREEN 的详细信息 ，参见 *调用新的屏 幕序列* (页 408) 。

示例事务： 设置和调用 屏幕

要了解屏幕 流程控制的 完整实施， 查看事务 TZ40（ 开发类 SDWA） 是如何组织 的很有用。

屏幕流程逻 辑
要显示事务 如何转入或 调用屏幕， 请看屏幕 200 的处理。退 出命令的处 理（功能代 码为 BACK 和 EXIT） 显示了该过 程。处理 BACK 或 EXIT 功能代码时 ，PAI 模块必须检 查在屏幕显 示或最后保 存之后航班 细节是否更 改。如果是 ，屏幕 200 必须调出弹 出屏幕 210 以提示保存 。屏幕 200 流程逻辑的 相关部分是 ：

```
*---------------------------------------------------------------*
*   Screen 200: Flow Logic                                      *
*&--------------------------------------------------------------*
PROCESS AFTER INPUT.
     MODULE EXIT_0200 AT EXIT-COMMAND.
*      (...<Field checks here>...)
     MODULE USER_COMMAND_0200.
```

ABAP/4 代码
屏幕 200 的 PAI 模块如下。 事务 TZ40 提供所有返 回功能（“ 后退”、“ 退出”和“ 取消”）作 为退出命令 。然而，在 屏幕 200 中，只有“ 取消”功能 允许立即从 屏幕退出。 要使取消生 效，使用标 准退出逻辑 告诉系统跳 回到屏幕 100：

```
*&---------------------------------------------------------------*
*&      Module  EXIT_0200  INPUT
*&---------------------------------------------------------------*
MODULE EXIT_0200 INPUT.
     CASE OK_CODE.
       WHEN 'CANC'.
```

```
              CLEAR OK_CODE.
              SET SCREEN 100.
              LEAVE SCREEN.
    ENDCASE.
ENDMODULE.
```

屏幕 200 的所有其它 的功能代码 处理如下：
“保存” 功能触发数 据库的更新
“退出” 和“后退” 功能触发调 用 SAFETY_CHECK 例程。该例 程检查屏幕 上未保存的 数据，必要 时提醒用户 保存。
注意返回方 法。对于“ 退出”功能 ，控制从事 务中一起返 回（SET SCREEN 0）。对于 “后退”功 能，将先前 屏幕设置为 后继屏幕（ SET SCREEN 100）。

```
*&---------------------------------------------------------------------*
*&      Module  USER_COMMAND_0200  INPUT
*&---------------------------------------------------------------------*
MODULE USER_COMMAND_0200 INPUT.
    CASE OK_CODE.
      WHEN 'SAVE'.
              UPDATE SPFLI.
              IF SY-SUBRC = 0.
                MESSAGE S001 WITH SPFLI-CARRID SPFLI-CONNID.
              ELSE.
                MESSAGE A002 WITH SPFLI-CARRID SPFLI-CONNID.
              ENDIF.
              CLEAR OK_CODE.
      WHEN 'EXIT'.
              CLEAR OK_CODE.
              PERFORM SAFETY_CHECK USING RCODE.
              IF RCODE = 'EXIT'. SET SCREEN 0. LEAVE SCREEN. ENDIF.
      WHEN 'BACK'.
              CLEAR OK_CODE.
              PERFORM SAFETY_CHECK USING RCODE.
              IF RCODE = 'EXIT'. SET SCREEN 100. LEAVE SCREEN. ENDIF.
    ENDCASE.
ENDMODULE.
```

SAFETY_CHECK 例程代码如 下。CHECK 语句将当前 屏幕值与保 存屏幕值作 比较。如果 值匹配，则 不需要保存 并且终止例 程。
如果值不同 ，则 SAFETY_CHECK 调用弹出屏 幕 210。弹 出屏幕询问 用户是否要 保存，并在 OK_CODE 中返回答案 （“保存” 、“退出” 和“取消” ）。

```
*---------------------------------------------------------------------*
*   Subroutine SAFETY_CHECK                                           *
*---------------------------------------------------------------------*
FORM SAFETY_CHECK USING RCODE.
    LOCAL OK_CODE.
    RCODE = 'EXIT'.
    CHECK SPFLI NE OLD_SPFLI.
    CLEAR OK_CODE.
    CALL SCREEN 210 STARTING AT 10 5.
    CASE OK_CODE.
      WHEN 'SAVE'. UPDATE SPFLI.
      WHEN 'EXIT'.
      WHEN 'CANC'. CLEAR SPFLI.
    ENDCASE.
ENDFORM.
```

后台处理屏 幕

可以使用 SUPPRESS DIALOG 取消所有屏 幕。该命令 允许“在后 台”进行屏 幕处理。系 统执行所有 PBO 和 PAI 逻辑，但并 不把屏幕显 示给用户。
从事务对话 步骤转到列 表模式时取 消屏幕很有 用。
在从屏幕的 PBO 逻辑调用的 第一个模块 中请使用 SUPPRESS DIALOG 命令。例如 ：

```
**** ABAP/4 module processing for Screen 100
     CALL SCREEN 110 STARTING AT 10 5

**** Screen 110 flow logic
PROCESS BEFORE OUTPUT.
     MODULE DIALOG_WINDOW.

**** ABAP/4 module processing
MODULE DIALOG_WINDOW OUTPUT.
     SUPPRESS DIALOG.
     LEAVE TO LIST-PROCESSING AND RETURN TO SCREEN 0.
     WRITE: /
     WRITE: /
ENDMODULE.
```

SUPPRESS DIALOG 语句将屏幕 110 处理上下文 作为显示标 准列表输出 的框架使用 。如果不使 用 SUPPRESS DIALOG， 则显示屏幕 100 并且显示为 空。当用户 按 ENTER 时，显示标 准列表输出 。

第五章 修改屏幕

概览

运行时可以 用许多不同 的方式修改 屏幕：

内容

在动态程 序中设置字 段属性
可以在对话 程序中临时 更改字段属 性（例如输 入/输出字 段，强制字 段）。也可 以临时禁用字段。使用 该技术动态 修改屏幕通 常意味着可 以避免定义 附加屏幕。
可以在功 能字段选择 的帮助下更 改字段属性 。
功能字段选 择支持用户 动态修改屏 幕的属性。
在运行时 显示子屏幕
为了在运行 时增强现有 的屏幕，可 以显示子屏 幕。子屏幕 用于有选择 地显示某些 字段。例如 ，可以有两 个子屏幕， 一个包含“ 物料名称” 和“物料数 目”字段， 另一个包含 “客户名称 ”和“客户 数目”字段 。根据用户 在上一个屏 幕中的输入 选定两个子 屏幕之一。
选择光标 位置
根据用户的 输入，可以 在对话程序 中将光标放 在屏幕上的 特定字段中 。

在下面的章 节中可以找 到详细信息 ：

设置屏幕字 段属性 (页 404)
用功能“字 段选择”更 改屏幕字段 属性 (页 396)
使用子屏幕 (页 397)
使用光标 (页 341)

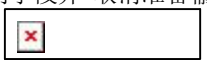
设置屏幕字 段属性

每个屏幕字 段都有用户 在定义屏幕 时在屏幕制 作器中设置 的属性。在 运行时，根 据用户在上 一个屏幕中 请求的功能 ，可能需要 更改这些属 性。在运行 时，每个屏 幕字段的属 性存储在名 为 SCREEN 的内存表格 中。不必在 程序中声明 该表格，系 统内部维护 该表格并且 在每次屏幕 更改时更新 它。

内存表格 SCREEN 包括下列字 段：

名称	长度	说 明
NAME	30	屏 幕字段的名 称
GROUP1	3	属 于字段组 1 的 字段
GROUP2	3	属 于字段组 2 的 字段
GROUP3	3	属 于字段组 3 的 字段
GROUP4	3	属 于字段组 4 的 字段
ACTIVE	1	可 见并准备输 入的字段
REQUIRED	1	字 段输入是强 制的
INPUT	1	字 段准备输入
OUTPUT	1	字 段仅用于显 示
INTENSIFIED	1	高 亮显示字段
INVISIBLE	1	禁 用字段
LENGTH	1	减 少字段输出 长度
DISPLAY_3D	1	以 三维框架显 示字段
VALUE_HELP	1	显 示有值帮助 的字段

要激 活字段属性 ，则将它的 值设置为 1。 要使它无效 ，则将它设 置为 0。将 ACTIVE 属性设置为 0 时，系统 禁用字段并 取消准备输 入属性。用 户不能查看 字段也不能 对它输入值 。



用户可以在 屏幕制作器 字段列表中 的“1 个字段的属 性”部分中 为每个属性 定义值。如 果需要关于 属性含义的 详细信息， 参见 *ABAP/4 开发工作台：工具*（页 **Error! Not a valid link.**）。

动态修改屏 幕的示例， 从事务 tz50 开始（开发 级别 SDWA） 。
事务包含两 个屏幕。在 第一个屏幕 中，用户可 以输入航班 标识符以及 请求航班详 情（按“显 示”按钮） 或按“更改 ”按钮来更 改屏幕 200 的数据。

现在根据选 定了“显示 ”按钮还是 “更改”按 钮，动态设 置字段属性 。在两种情 况下都调用 同一屏幕， 但有不同的 字段属性。
如果需要同 时为多个字 段更改相同 的属性，则 可以将这些 字段组合在 一起。例如 ，为了在屏 幕 200 中动态更改 字段，我们 将这些在屏 幕制作器中 的字段分配 给组 MOD。用 户可以为每 个字段最多 指定四个修 改组。“组 ”字段的内 容存储在 SCREEN 表格中。

可以在 PBO 模块中实现 对该组中字 段属性的更 改：

```
MODULE MODIFY_SCREEN OUTPUT.
          CHECK MODE = CON_SHOW.
          LOOP AT SCREEN.
               CHECK SCREEN-GROUP1 = 'MOD'.
               SCREEN-INPUT = '0'.
               MODIFY SCREEN.
          ENDLOOP.
ENDMODULE.
```

内存表格 SCREEN 包含当前屏 幕的每个字 段和它的属 性。
LOOP AT SCREEN 语句将该信 息放在该系 统表格的表 头行中。
在事务 tz50 的该示例中 ，如果用户 选择“显示 ”，则将 SCREEN-INPUT 设置为‘0’ ，并且所有 属于 MOD 组的字段因 此而成为只 显示字段。
由于属性已 更改，MODIFY SCREEN 语句用于将 表头行写回 表格。

用功能“字 段选择”更 改屏幕字段 属性

本主题描述 特殊功能“ 字段选择” （事务 SFAW 和一些功能 模块）如何 支持动态更 改屏幕字段 属性。

字段选择 - 概述

功能“字段 选择”允许 用户在运行 时动态更改 屏幕字段的 属性。但只 有在由于技 术原因需要 经常将不同 字段属性分 配给相同屏 幕时才使用 该选项。在 这种情况下 ，所有字段 使用相同的 规则，因此 任何字段修 改都是清楚 的。
基本规则应 用如下：

- 字段选择 进程中涉及 的所有字段 在字段选择 表格中组合 在一起并用 “字段选择 ”功能维护 。
- 维护通常 是通过模块 池和屏幕组 进行的。
- 在属于屏 幕组“空格 ”（‘_’ ）的屏幕上 没有动态字 段选择。
- 由于屏幕 字段属性 SCREEN-GROUP1 是为中心字 段选择而保 留的，因此 不能同时将 它用于其它 目的。
- 如果正在 使用特殊的 预定义规则 使任何更改 等同于程序 更改，则不 要使用该功 能，而在程序本身中更 改 。

用字段选择 可以在运行 时激活或撤 消下列属性 ：

- 输入
- 输出
- 强制
- 激活
- 高亮显示
- 不可见

用户也可以 确定有关更 改的条件和 类型。在事 件 PROCESS BEFORE OUTPUT 中可以调用 功能模块来 检查条件和 更改属性（ 如果必要的 话）。
字段选择区 分有影响的 字段和修改 的字段。当 然修改的字 段必须是屏 幕字段。所 有字段应该 在数据词典 中定义，并 且应该用 TABLES 语句在模块 池中全局声 明相应的表 格。在运行 时，功能模 块分析影响 字段的内容 ，然后相应 设置修改字 段的属性。

在屏幕组中 组合屏幕
用户可以将 逻辑上相关 的屏幕组合 在屏幕组中 ，而不是为 程序的每个 屏幕单独维 护字段选择 。要将屏幕 分配给屏幕 组，请在屏 幕制作器属 性屏幕上的 字段“屏幕 组”中输入 组。

调用字段选 择

要调用字段 选择，选定 “工具 -> ABAP/4 工作台 -> 开发 -> 其它功能 -> 字段选择” 。维护是通 过程序和屏 幕组进行的 。

首先必须声 名有关字段 的表格名。 选择“将表 格分配给屏 幕组”并输 入表格，例 如：

保存条目并 选择“有影 响的字段” 将所需的影 响字段输入 到列表中， 并有选择地 指定 NOT 条件、默认 值以及字段 “客户”， 例如：

NOT 条件理解为 预选择。如 果字段之一 满足 NOT 条件,则它 与下列屏幕 的修改是不 相关的。使 用 NOT 条件可以提 高性能。

> 有影响的字 段： SPFLI-CARRID
>
> NOT 条件： NE LH
>
> 仅当 SPFLI-CARRID 的内容在运 行时与 LH 相同时它与 字段选择才 是相关的。

在运行时， 如果系统在 维护值的列 表中找不到 影响字段的 当前值，则 使用默认值 。用户必须 自己定义默 认值。该选 项允许用户 维护影响字 段的所有格 式，这些格 式有相同的 影响，带有 单个条目。用字段“客 户”，用户 可以决定是 否允许客户 使用字段选 择的相应字 段。如果标 记为“客户 ”，那么客 户也可以使 用该字段来 影响其它字 段。
然后选择“ 修改的字段 ”将所有可 修改的字段 输入到列表 中，例如：

如果想允许 客户修改字 段，则再次 标记字段“ 客户”。
每个有影响 的和可修改 的字段都有 每个程序唯 一的内部号 。当按 *F16* 生成时，内 部号将自动 放入适当屏 幕的 SCREEN-GROUP1 中并且不能 在屏幕制作 器中更改它 。这就允许 系统在字段 名和 SCREEN-GROUP1 之间建立一 对一的关系 。
最后创建两 个列表的有 影响的和可 修改字段之 间的链接： 指定有影响 的字段中影 响可修改字 段的内容和 方式。
要链接字段 ，请用“选 择”从列表 中选定字段 或双击它。 如果选定一 个有影响的 字段，则出 现可修改字 段的列表， 反之亦然。 从该列表中 选定所需的 链接。出现 可以在其中 输入相关条 件的列表，例如：

上面的条目 导致禁止在 这些屏幕上 显示字段 SPFLI-AIRPFROM， 在这些屏幕 的 PBO 中调用相应 的字段选择 功能模块，并且如果调 用 SPFLI-CARRID 则包含‘LH’（参见 *字段选择的 功能模块* (页 416))。“激活”属 性的功能在 *显示属性 ‘ 激活’* (页 419) 中说明。

属性的组合 规则

如果多个有 影响字段影 响同一个修 改的字段， 则必须有组 合规则来确 定如何链接 这些影响。 如果由不同 的影响激活 或解除激活 ，则使用下 面的表格了 解如何设置 单个字段属 性。屏幕处 理器控制多 个属性的组 合。

输入			
		字段1	
		’_’	’X’
字 段2	’_’	’_’	’_’
	’X’	’_’	’X’

输出			
		字段1	
		’_’	’X’
字 段2	’_’	’_’	’_’
	’X’	’_’	’X’

激活			
		字段1	
		' ' _	'X'
字 段2	' ' _	' ' _	' ' _
	'X'	' ' _	'X'

强制			
		字段1	
		' ' _	'X'
字 段2	' ' _	' ' _	'X'
	'X'	'X'	'X'

高亮显示			
		字 段1	
		' ' _	'X'
字 段2	' ' _	' ' _	'X'
	'X'	'X'	'X'

不可见			
		字段 1	
		' ' _	'X'
字 段2	' ' _	' ' _	'X'
	'X'	'X'	'X'

字符说明：
_ = 关闭 (off)， X = 打开 (on)

如果字段 1 使某个屏幕 字段不可见 （X），则 字段2不能 更改。

屏幕制作器 属性

用屏幕制作 器，系统不 仅考虑在字 段选择期间 制作的条目 ，而且考虑 在屏幕制作 器中制作的 任何条目。 意思是根据 在 *属性的组合 规则* (页 414) 中所述的相 同链接规则 将上面组合 的结果链接 到屏幕字段 属性。
要利用全部 的动态修改 范围，应该 在屏幕制作 器中使用下 列属性：

```
Input = 'X'
Output = 'X'
Mandatory = '_'
Invisible = '_'
Highlighted = '_'.
```

相反地，用 户不能以下 列方式更改 在屏幕上定 义的值：

```
Input = '_'
Output = '_'
```

```
Mandatory = 'X'
Invisible= 'X'
Highlighted = 'X'
```

假设输入了 影响的下列 组合，但因 为组合规则 规定其它的 有影响字段 （或者屏幕 ）不能更改 指定的显示 属性，所以 它实际上不 是有效的组 合。

```
Input = 'X'
Output = 'X'
Active = 'X'
Mandatory = '_'
Highlighted = '_'
Invisible = '_'
```

再次进入字 段选择时不 显示这样的 无效影响， 除非已经为 影响字段定 义了默认值 ；此处，显 示和维护这 样的影响是 有用的。

生成字段选 择

如果确实更 改了已修改 字段的列表 ，则必须生 成字段选择 。这将在相 关模块池的 屏幕中为修 改的 SCREEN-GROUP1 字段产生连 续号。
为此，在事 务 SFAW 中选择“生 成”。

字段选择的 功能模块

要激活屏幕 的字段选择 ，可以在 PROCESS BEFORE OUTPUT 事件中调用 FIELD_SELECTION_MODIFY_ALL 或 FIELD_SELECTION_MODIFY_SINGLE。 两个功能模 块都决定有 影响字段的 内容，如果 需要的话， 参见组合规 则并执行屏 幕修改。FIELD_SELECTION_MODIFY_ALL 自己执行 LOOP AT SCREEN 语句。但是 如果用 FIELD_SELECTION_MODIFY_SINGLE， 用户必须自 己编码并且 在该循环中 调用功能模 块。因此用 户可以在 LOOP 中执行自己 的附加屏幕 修改。
在事件 PBO 中调用功能 模块示例：

```
CALL FUNCTION 'FIELD_SELECTION_MODIFY_ALL'
EXPORTING MODULEPOOL   = MODULEPOOL
          SCREENGROUP  = SCRGRP.
```

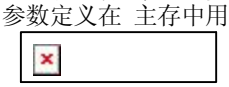
或

```
LOOP AT SCREEN.
  IF SCREEN_GROUP1 NE SPACE AND
     SCREEN-GROUP1 NE '000'.
    CALL FUNCTION 'FIELD_SELECTION_MODIFY_SINGLE'
      EXPORTING MODULEPOOL   = MODULEPOOL
                SCREENGROUP  = SCRGRP.
* Separate special rules
    MODIFY SCREEN.
  ENDIF.
ENDLOOP.
```

或

```
与 a) 一样，但为 特殊规则包 括自己的 LOOP AT SCREEN。
```

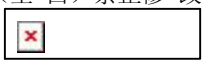
在每个单独 的情况下， 用户必须决 定哪个选项 b) 或 c) 将获得最佳 性能。
由于“模块 池”和“屏 幕组”参数 决定字段选 择，因此必 须为它们维 护影响。
“模块池” 参数定义在 主存中用于 搜索影响字 段当前值的 已加载模块 池。



```
调用功能模 块时，必须 直接包括系 统字段 SY-REPID 和 SY-DYNGR。 将它们的内 容直接传输 到其它字段 的适当代码 位置，例如 ：
```

```
MODULEPOOL = SY-REPID.
SCRGRP     = SY-DYNGR
```

有些时候， “模块池” 的值与当前 SY-REPID 值可能不同 。
如果“屏幕 组”参数为 空，则系统 使用 SY-DYNGR 的当前内容 。这对于“ 模块池”参 数是不可能 的，因为值 “_”（空 白）禁止修 改任何字段 。



注意在开发 级别 SDWA 中与 TZ50 类似的事务 （参见 *设置屏幕字 段属性* （页 412））。

假设第二个 屏幕的动态 程序包含 PBO 事件中的下 列模块调用 ：

```
PROCESS BEFORE OUTPUT.

  ...

  MODULE MODIFY_SCREEN.
```

假设模块 MODIFY_SCREEN 包含下列函 数调用：

```
MODULE MODIFY_SCREEN OUTPUT.

   CALL FUNCTION 'FIELD_SELECTION_MODIFY_ALL'
        EXPORTING
             SCREENGROUP =  'SCREEN'
             MODULEPOOL    =  'SAPMTXXX'
        EXCEPTIONS
             OTHERS        = 1.
```

假设对于屏 幕组 SCREEN 和模块池 SAPMTXXX， 按 *调用字段选 择* （页 414）中图表所示 维护事务 SFAW 中的影响。

调用事务后 ，假设已经 制作了这些 条目：

选择“更改 ”后，出现 下列屏幕：

但是，如果 输入‘AA’ 航空公司代 替‘LH’ ，则显示下 列屏幕：

当输入‘LH’ 时，字段 SPFLI-AIRPFROM 是不可见的 。当输入 ‘ AA’时， 它显示为 “ 机场部门” 。

链接字段

每个有影响 字段可以影 响那些忽略 其它字段而 进行修改的 字段。在某 些情况下需 要有影响字 段的链接， 但只有通过 帮助字段的 定义才有可 能实现链接 ，必须在调 用功能模块 之前在应用 程序中设置 帮助字段（ 参见 *属性的组合 规则* （页 414））。
该限制有助 于字段选择 的透明度。

链接示例
假设下列字 段：

```
有影响字段 ：          F4711, F4712
可以修改的 字段：      M4711
```

下列情况只 能通过迂回 的方式实现 ：

OR 条件和“输 入准备”

```
如果          F4711 = 'A' OR F4712 = 'B',
那 么         M4711 准备输入。
```

解决

```
在 SFAW 中将 H4711 定义为有影 响字段,
在 SFAW 中定义下列 条件:
   如 果 H4711 = 'AB'
   那么 M4711 可以输入 （也就是说 , 输入 = 'X'）
```

在应用程序 中，必须在 调用功能模 块前进行下 列编程：

```
IF F4711 = 'A' OR F4712 = 'B'.
   H4711 = 'AB'.
ENDIF.
```

AND 条件和 “强制”

```
如果          F4711 = 'A' AND F4712 = 'B',
那 么         必须 M4711。
```

解决：

```
在字段选择 中维护:
如 果 H4711 = 'AB',
那么 M4711 是所需的条 目字段
（H4711 = 'AB' 仅在上面的 AND 条件下才精 确）。
```

在应用程序 中，编程：

```
IF F4711 = 'A' AND F4712 = 'B'
   H4711 = 'AB'
ELSE.
  H4711 = ....
ENDIF.
```

另一方面， 可以直接表 现下列情况 ：

AND 条件和“输 入准备”

```
如果          F4711 = 'A' AND F4712 = 'B',
那么          M4711 准备输入。
 因此:        如果 F4711 <> 'A' OR F4712 <> 'B'
```

解决：

```
屏幕: M4711 准备输入

字段选择:

有影响字段   F4711     值 'A'    输 入 = 'X'
                  值 'A1'   输 入= ' '
                  值 'AX'   输 入 = ' '
有影响字段   F4712     值 'B'    Input = 'X'
```

```
                    值 'B1'   输 入 = ' '
                    值 'BX'   输 入 = ' '
```

OR 条件和“强 制”

```
如果          F4711 = 'A' OR F4712 = 'B',
那 么         M4711 是所需的条 目字段。
```

解决：

```
屏幕：关闭 了强制
有影响字段   F4711     值 'A',
             强制 = 'X'
有影响字段   F4712     值 'B',
             强制 = 'X'
```

```
为有影响字 段而定义 NOT 条件的可能 性提供了字 段选择定义 的更多变体 。
```

显示属性 ‘ 激活’

目前该显示 属性只有一 种结果。如 果在 ABAP/4 程序中 SCREEN-ACTIVE = '0'，则 用 MODIFY SCREEN 语句在运行 时发生下列 结果：

```
SCREEN-INPUT = '0'
SCREEN-OUTPUT = '0'
SCREEN-INVISIBLE = '1'
```

如果 SCREEN-ACTIVE = '1'，则 不发生任何 事情。
另一方面， 如果 SCREEN-INPUT = '0'、SCREEN-OUTPUT = '0' 并且 SCREEN-INVISIBLE = '1'，则 内部结果将 是 SCREEN-ACTIVE = '0'（无 后继步骤） 。
甚至当 SCREEN-ACTIVE = '0' 时，在屏幕 流逻辑中用 相关 FIELD 语句指定的 模块总在运 行。但是不 运行有撤消 激活字段的 模块。如果 不想这么做 ，可以用句 号分隔 FIELD 和 MODULE 语句。
如果非活动 字段有附加 的规范，如 ON INPUT、 ON REQUEST ...，则 屏幕模块不 运行，通常 处理没有附 加规范的模 块。

字段选择授 权

“字段选择 ”的授权对 象是“中心 字段选择” (S-FIELDSEL)。 该对象包含 一个活动和 一个程序授 权组。后者 是从程序授 权中获得的 。
允许下列活 动：

```
'02' = 更改
'03' = 显示
'14' = 在屏幕上生 成字段选择 信息
'15' = 将相关表格 分配给字段 选择
```

使用子屏幕

子屏幕是显 示在另一个 （“主”） 屏幕区域中 的独立屏幕 。用户可能 想使用子屏 幕来改变主 屏幕中的某 些字段。例 如，根据用 户在上一个 屏幕中的输 入，可以在 出现补充字 段的主屏幕 中定义一个 区域。
按下列方式 创建子屏幕 ：

1. 在“ 主”屏幕中 调整子屏幕 的框架直到 它的大小符 合需要。在 “字段名” 字段中命名 子屏幕。
2. 用屏 幕类型“子 屏幕”创建 屏幕。

3. 在子 屏幕中排列 字段使它们 显示在主屏 幕中所要求 的地方。
   如果定义 的子屏幕比 主屏幕中的 可用区域大 ，则只有与 可用区域相 当大小的部 分子屏幕可见（从左上 角计）。

要使用子屏 幕，必须在 主屏幕的流 逻辑（PBO 和 PAI）中 调用它。CALL SUBSCREEN 语句告诉系 统将子屏幕 的 PBO 和 PAI 事件作为主 屏幕的 PBO 和 PAI 事件的一部 分执行。就 象为主屏幕 编程一样为子屏幕编码 ABAP/4 模块。
主程序的流 逻辑如下所 示：

```
PROCESS BEFORE OUPTPUT.
    ...
    CALL SUBSCREEN <area> INCLUDING <program> <screen>.
    ...
PROCESS AFTER INPUT.
    ...
    CALL SUBSCREEN <area>.
    ...
```

“区”是在 主屏幕中定 义的子屏幕 区域的名称 。该名称最 多可有十个 字符。“程 序”是子屏 幕所属程序 的名称，“ 屏幕”是子 屏幕的编号 。
在单个屏幕 中可能有多 个子屏幕。 也可以在运 行时动态指 定子屏幕。 子屏幕有几 个限制。它 们不能：

设置它们 自己的 GUI 状态
有已命名 的“确定代 码”
调用其它 屏幕
包含 AT EXIT-COMMAND 模块

支持光标 位置。

使用光标

在 PBO 事件中，可 以告诉系统 将光标放在 某些屏幕字 段上。光标 的放置是一 种使事务更 友好的方式 。

在屏幕上放 置光标
显示屏幕时 ，系统自动 将光标放在 准备输入的 第一个字段 中。
但是用户可 以自己指定 要显示光标 的位置。有 两个选项。

1. 在屏 幕制作器中 定义屏幕时 输入合适的 字段名作为 “光标位置 ”。“光标 位置”字段 显示在“屏 幕属性”中 。
2. 也可 以使用 ABAP/4 命令 SET CURSOR FIELD 设置光标。 按如下方式 使用 SET CURSOR 语句：
   SET CURSOR FIELD <field name>.
   <field name> 可以是引用 中的显式字 段名，或是 包含字段名 的变量。要 将光标放在 字段中的特 定位置上， 请使用 OFFSET 参数，在 <position> 中为光标输 入字符位置 ：
   SET CURSOR FIELD <field name> OFFSET <position>.
   系统从行的 开头根据给 定的位置偏 置光标。
   如果在屏幕 中有单步循 环，则可以 将光标放在 单步循环块 中的特定元 素上。使用 LINE 参数，在想 放置光标的 地点输入循 环块的行：
   SET CURSOR FIELD <fieldname> LINE <line>.
   如果需要， 可以同时使 用 OFFSET 和 LINE 参数。

第六章 在屏幕中使 用表格

概览

ABAP/4 提供了两种 机理以便在 屏幕中显示 和使用表格 数据。这两 种机理分别 为“表格控 制”和“步循环”。表 格控制和步 循环均为屏 幕表格，您 可以在屏幕 制作器中将 此类屏幕表 格添加到屏 幕里。例如 ，下面的屏 幕在底部就 含有表格控 制。

内容

本章讲述如 何编制允许 您使用屏幕 表格的屏幕 流逻辑和 ABAP/4 代码。有关 使用屏幕表 格的信息，参见：

简介 (页 409)
使用 LOOP 语句 (页 451)
使用表格控 制 (页 403)
使用步循环 (页 408)
示例事务： 表格控制 (页 451)

简介

本节讲述如 何在屏幕中 使用表格显 示，以便用 户很快就能 大略了解其 概况。

要想获取此 处所概列的 原理示例， 请参阅事务 TZ60 和 TZ61， 它们分别描 述了表格控 制和步循环 的使用。(TZ60 和 TZ61 是同系统一 起发布的开 发级别 SDWA 中的样本事 务。)

表格控制和 步循环

表格控制和 步循环是用 于屏幕表格 显示的对象 ，在屏幕制 作器中用户 可将其添加 到屏幕。从 编程的角度 来看，表格 控制和步循 环几乎完全 一样。表格 控制只是增 强了的步循 环，此步循 环可以使用 桌面应用程 序中表格工 具的“Look” 和“Feel” 来显示数据 。使用表格 控制，用户 可以：

- 沿着表格 水平和竖直 地滚动
- 重新调整 某一栏的宽 度
- 在字段内 滚动（当字 段内容比该 字段宽时）
- 选择表格 的行数或列 数
- 重新规定 栏的顺序
- 保存当前 的显示设置 以备后用

表格控制还 提供使表格 易于查看和 使用的专用 格式化功能 （有些是自 动的，有些 是可选的） 。例如：

- 当用户调 整窗口时， 自动进行表 格调整（水 平或竖直）
- 行与行之 间和列与列 之间的分隔 线（水平和 竖直）
- 所有列的 列表头字段 。

步循环的一 个特点是它 们的表行能 够在屏幕上 跨越多行。 相反，表格 控制中的行 总是单行， 但可以很长 。(表格控 制的行能够 滚动。)

通常，表格 控制所提供 的许多特征 由系统的 SAPGUI 前端操纵， 因此，不必 在 ABAP/4 事务中编制 任何特征 （ 除了竖直滚 动）。

屏幕表格处 理

屏幕表格是 屏幕中重复 的系列表格 行。每一条 目都包含一 个或多个字 段，并且所 有的行都具 有相同的字 段结构。屏 幕表格要么 是表格控制 要么是步循 环。显示飞 行数据的表 格控制如下 所示：

屏幕表格和 LOOP 动态语句

如同在内表 中循环来处 理内表，用 户也可以在 屏幕表格中 循环以处理 屏幕表格。 。为此，就 必须在屏幕 流逻辑中加 入一条 LOOP...ENDLOOP 动态语句。 该循环通常 要调用一次 ABAP/4 模块，但也 允许使用其 它流逻辑命 令。系统每 次通过循环 时就运行此 模块。

LOOP 动态语句有 多种格式。 两个最重要 的格式的功 能如下：

只在屏幕 表格中循环

同时在屏 幕表格和内 表中循环

屏幕表格和 程序字段

可以将屏幕 表格字段声 明为数据库 字段、内表 字段、结构 字段或其它 程序字段。 屏幕表格字 段有时出现 在屏幕字段 列表中，有 时出现在程 序中。因此 ，屏幕表格 中所有的行 在程序中共 用同一个字 段集（类似 “表头区域 ”）。在流 逻辑的 LOOP 过程中，系 统将屏幕表 格行的所有 字段复制到 相关的程序 字段中或从 中复制回来 。

LOOP 语句的任务

LOOP 语句负责读 取往返传递 于屏幕和 ABAP/4 程序之间的 屏幕表格值 。因此必须 同时在 PBO 和 PAI 事件中为屏 幕中的每一 个表格编制 LOOP 语句。至少 在此应有一 个空 LOOP...ENDLOOP 语句。

LOOP 语句还用于 驱动滚动。 在 PBO 事件中，LOOP 使用一个参 数告诉从表 格的何处开 始循环。该 参数因此导 致下一个屏 幕表格显示 的更新（对 于表格控制 ，该参数是 表格控制结 构中的 TOP_LINE 字段；对于 步循环，该 参数为用于 LOOP 语句的 CURSOR 参数。）。 ABAP/4 程序和系统 都可以设置 该参数。

注意，屏幕 表格中所显 示的行数可 以改变。当 屏幕表格可 调整并且用 户更改窗口 的高度时就 会出现这种 情况。在这 种情况下， PAI 中的下一个 LOOP 就更改 PAI 中传到 ABAP/4 程序的表格 行数。

使用 LOOP 语句

LOOP...ENDLOOP 动态命令可 在流逻辑中 执行循环操 作。可以使 用该语句在 表格控制和 步循环中进行循环。在 LOOP 和 ENDLOOP 之间，可以 使用 FIELD、 MODULE、 SELECT、 VALUES 和 CHAIN 动态关键字 。最为常见 的是使用 MODULE 语句来调用 ABAP/4 模块。

必须同时在 PBO 和 PAI 事件中为屏 幕的每个表 格编制 LOOP 语句。由于 LOOP 语句导致在 ABAP/4 程序和屏幕 字段之间来 回复制屏幕 字段。因此 ，此处至少 必须有一个 空 LOOP...ENDLOOP 语句。

LOOP 语句有两种 重要的格式 ：

LOOP。

该语句在屏 幕表格行中 循环，同时 在每个块和 程序的对应 ABAP/4 字段之间往 复传送数据 。可以在 ABAP/4 中将屏幕表 格字段声明 为任何类型 （数据库表 格、结构或 单个字段） ，内表字段 除外。

对于步循环 ，如果正在 执行本身的 滚动（例如 ，使用 F21－F24） ，则必须使 用该语句。

LOOP AT <internal table>.

该语句可同 时在内表和 屏幕表格行 中循环。常 常将此屏幕 表格字段声 明为内表字 段，但不是 非得如此。

对于该 LOOP， 步循环显示 都带有滚动 条。该滚动 由系统自动 操纵。

有关不同 LOOP 语句的详细 信息，参见 ：

在屏幕表格 中直接循环 (页 422)
在内表中循 环 (页 397)

在屏幕表格 中直接循环

使用 LOOP 语句的简单 格式

```
LOOP.
...<actions>...
ENDLOOP.
```

在当前所显 示的屏幕表 格行中循环 。如果正在 使用表格控 制，就必须 包括附加的 WITH CONTROL 参数：

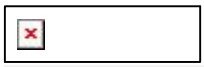
```
LOOP WITH CONTROL <table-control>.
...<actions>...
ENDLOOP.
```

该简单的 LOOP 是 LOOP 语句最一般 的格式。如 果使用该 LOOP，就能够将屏 幕表格字段 声明为任何 类型（内表 、数据库表 格、结构或 单个字段） 。此简单的 LOOP 将屏幕表格 字段往复地 复制到相关 的 ABAP/4 字段。如果 要在不同的 结构中使用 屏幕值，就 必须直接将 它们移到所 要的地方。

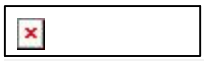
每经过一次 循环，都将 下一表格行 置于 ABAP/4 字段中，并 为其执行 LOOP<actions>（ 通常为 ABAP/4 模块调用） 。在 PBO 事件中，LOOP 语句可将程 序中的循环 字段逐行复 制到屏幕。 在 PAI 事件中，字 段被逐行复 制到相关的 程序字段上 。

在 ABAP/4 模块中，使 用系统变量 SY-STEPL 以查找当前 正在处理的 屏幕表格行 的索引。每 循环一次， 系统就设置 一次该变量 。SY-STEPL 的值总是介 于 1 与当前所显 示的行数之 间。当在屏 幕表格和内 表之间来回 地传送字段 值时，这很 有用。您可 以在程序中 声明表格偏 移量（常称 为 BASE， 并通常用 SY-LOOPC 初始化）并 用它同 SY-STEPL 一起获得与 当前屏幕表 格行相对应 的内表行。

（下例中， 将屏幕字段 声明为内表 。程序读取 或修改内表 以读取往返 传送到屏幕 的表格字段 。）



```
***SCREEN FLOW LOGIC***
PROCESS BEFORE OUTPUT.
   LOOP.
       MODULE READ_INTTAB.
   ENDLOOP.

PROCESS AFTER INPUT.
   LOOP.
       MODULE MODIFY_INTTAB.
   ENDLOOP.


***ABAP/4 MODULES***
MODULE READ_INTTAB.
   IND = BASE + SY-STEPL - 1.
   READ TABLE INTTAB INDEX IND.
ENDMODULE.

MODULE MODIFY_INTTAB.
   IND = BASE + SY-STEPL - 1.
   MODIFY INTTAB INDEX IND.
ENDMODULE.
```



请记住，系 统变量 SY-STEPL 只在 LOOP...ENDLOOP 处理的范围 之内才有意 义。循环之 外，它没有 合法值。

在内表中循 环

语句

```
LOOP AT <internal table>.
```

表示同时在 内表和屏幕 表格中循环 。特别地， LOOP AT 在屏幕中当 前可见的内 表部分中循 环。对于表 格控制和步 循环都可使 用此 LOOP 语句格式。

该 LOOP 语句的完整 语法格式为 ：

```
LOOP AT <internal table> CURSOR <scroll-var>
                         WITH CONTROL <table-control>
                         FROM <line1>      TO <line2>  .
```

```
...<actions>...
ENDLOOP.
```

该形式的 LOOP 语 句在内表中 循环，对每 行都执行 <actions>。 对于每个内 表行，系统 将相应程序 字段传送到 对应的屏幕 表格行，或 将相应的屏 幕表格行传 送回对应的 程序字段。
使用步循环 时，在 PAI 事件中省略 CURSOR 参数。FROM 和 TO 参数只能用 于步循环（ 参见 *使用步循环* （页 408） 。）。WITH CONTROL 参数只能用 于表格控制 。
下列标题提 供详细信息 ：
系统如何传 送数据值 （页 400）
滚动和滚动 变量 （页 407）

系统如何传 送数据值

对于 LOOP AT <internal table> 语句，不必 将屏幕表格 声明为内表 。屏幕表格 也可以是数 据库表格、结构或其它 程序字段。 如果没有将 屏幕表格定 义为内表， 则必须确认 在循环中移 到内部表头 中或从内表 头中移出的 字段是正确 的，
还要注意， 在屏幕表格 中所使用的 字段可能只 是相应词典 或内表字段 的一个子集 。系统在屏 幕表格中传 送需要的字 段。详细处 理过程如下 ：

PBO 处理

1. 内表 的一行被放 入表头区。
2. 为该 行执行所有 的循环语句

   循环语句最 通常是调用 ABAP/4 模块。只要 需要，这些 模块就将内 表字段移到相关的程序 字段中（字 典表格或其 它字段）。

   示例事务 TZ60 在例程 DISPLAY_FLIGHTS 中执行下列 命令：

```
MODULE DISPLAY_FLIGHTS OUTPUT.
  SFLIGHT-FLDATE    = INT_FLIGHTS-FLDATE.
  SFLIGHT-PRICE     = INT_FLIGHTS-PRICE.
  SFLIGHT-CURRENCY  = INT_FLIGHTS-CURRENCY.
  SFLIGHT-PLANETYPE = INT_FLIGHTS-PLANETYPE.
  SFLIGHT-SEATSMAX  = INT_FLIGHTS-SEATSMAX.
  SFLIGHT-SEATSOCC  = INT_FLIGHTS-SEATSOCC.
ENDMODULE.
```

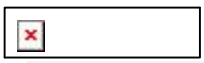
3. 系统 将值从程序 字段传送到 具有相同名 称的屏幕字 段中。

PAI 处理

1. 内表 的行被放入 内表表头区 中。
2. 所有 的屏幕表格 字段被传送 到具有相同 名称的 ABAP/4 程序字段中 。
3. 为当 前内表行执 行所有的循 环 <actions>。

   同样<actions> 也通常是对 ABAP/4 模块的调用 。只要需要 ，该模块应 当首先将程 序字段值移 到该内表的 表头区。本 步骤用屏幕 数据值覆盖 了内表数据值。步骤 1 对于屏幕表 格字段只是 为内表所定 义的字段的 子集的情况 是必须的。 如果想使用 新的屏幕值 更新内表， 那么，在表 头区必须有 作为基础的 原表格行。

   记住：如果 要使用表头 区中的内容 来更新内表 ，就请使用 ABAP/4 模块中的 MODIFY 语句。系统 不会自动进 行这种更新 。



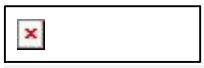
注意，现有 动态语言 MODIFY 语句和 ABAP/4 MODIFY 语句。请不 要使用动态 语言 MODIFY 语句来更新 内表。

滚动和滚动 变量

使用 LOOP AT <internal table> 语句可以自 动进行滚动 ，也可由编 程人员的操 作来执行滚 动。用滚动 条的滚动是 自动的，并 由系统来管 理。使用功 能键或其它 按钮的滚动 必须由编程 人员来进行 操作。（示 例事务 TZ60 给出了两种 方式的例子 。）
对于表格控 制和步循环 ，竖直滚动 都由 <scroll-var> 参数控制。 本节说明如 何使用该变 量。水平滚 动（只用于 表格控制） 由 SAPGUI 前端局部执 行。ABAP/4 代码不需要 支持任何水 平滚动。

第一次处理 PBO 事件时，程 序应当设置 <scroll-var> 以告诉系统 从何处开始 显示。对于 表格控制，<scroll-var> 是 TABLEVIEW 结构中的 TOP_LINE 字段。对于 步循环，请 声明局部程 序变量以用 作 CURSOR 参数。
之后，系统 在两种情形 下设置 <scroll-var> ：
只要用户 使用滚动条 滚动就更新 <scroll-var>。
每次进入 LOOP AT 块时增加 <scroll-var>。
这是因为每 当用户使用 滚动条滚动 时，系统执 行如下动作 ：

1. 为当 前屏幕显示 触发 PAI。
   滚动触发 PAI 事件就象用 户已经执行 了一个传送 功能代码的 动作。在使 用滚动条滚 动的 PAI 事件中，OK_CODE 字段没有值 。（如果用 功能键来进 行滚动，则 OK_CODE 的值就是该 功能键的功 能代码。）
   a. 将当 前显示的屏 幕值传送到 ABAP/4 程序中。
   b. 使用 <scroll-var> 的当前值， 为当前屏幕 值执行 PAI 处理。每经 过一次 LOOP， 系统就将 <scroll-var> 加一，因此 程序能够自 动访问相应 的内表条目 。
      （和平常一 样，只要程 序在此检测 出任何错误 ，都将发送 一条警告或 错误信息。 PAI 过程将重复 执行直到不 再发生错误 为止。）
2. 为新 屏幕显示触 发 PBO （由用户滚 动请求）。
   对于该 PBO，系 统将 <scroll-var> 更新为新值 （由用户滚 动设置）并 反复对屏幕 进行 LOOP AT 处理。结果 ，内表的新 程序段被传 送到屏幕上 。



```
***SCREEN FLOW LOGIC***
PROCESS BEFORE OUTPUT:
   LOOP AT INTTAB CURSOR TABCNTL-TOP_LINE.
   ENDLOOP.

PROCESS AFTER INPUT.
   LOOP AT INTTAB.
        MODULE MODIFY_INTTAB.
   ENDLOOP.


***ABAP/4 MODULE***
MODULE MODIFY_INTTAB.
   MODIFY INTTAB INDEX COUNT.
ENDMODULE.
```

如果想提供 除滚动条之 外的其它滚 动选项（例 如，标准工 具栏中的 **F21 F24** 滚动按钮） ，就必须自 己编程。为 此，可以在 必要的地方 自己设置 <scroll-var>。 于是 PBO 中的 LOOP 语句相应地 在下一个屏 幕显示中更 新屏幕表格 。

使用表格控 制

表格控制的 结构不同于 步循环。作 为屏幕对象 的步循环只 是一系列字 段行，它们 表现为一个 重复的块。 而作为屏幕 对象的表格 控制却由以 下几部分组 成：
表格字段 （在屏幕中 显示）
控制结构 ，支配着表 格显示以及 用户可以用 它来做什么 。
屏幕 200 的字段列表 （在事务 TZ60 中）显示这 两项：

这里可看到 控制结构（ 带有 *Ftyp = Table* 的 FLIGHTS 字段）和实 际表字段（ 带有 *Ftyp = Text* 或 *I/O* 的 SFLIGHT 字段）。FLIGHTS 字段的属性 （当您按 *Attribs. for 1 Field* 时可用）显 示可以静态 设置的控制 结构字段。 其它控制结 构字段可以 从 ABAP/4 程序中动态 地设置。
下列标题提 供详细信息 ：
在 ABAP/4 中声明表格 控制 (页 432)
设 置表格控制 属性 (页 401)

在 ABAP/4 中声明表格 控制

在屏幕中使 用表格控制 时，必须在 ABAP/4 程序中同时 声明表格控 制结构和表 格控制字段 。例如，事 务 TZ60 有：

```
TABLES:    SFLIGHT.
CONTROLS: FLIGHTS TYPE TABLEVIEW USING SCREEN 200.
```

CONTROLS 语句定义 TABLEVIEW 类型的控制 结构。系统 从给定屏幕 的屏幕制作 器属性中获 得结构的初始值。

TABLEVIEW 结构包含如 下字段：

字段名	类型	用途
FIXED_COLS	integer	表 格左端不可 移动列数。 固定列后面 的所有列都 可以移动， 并能在表格 中重新排序 。
LINES	integer	表 格中的可显 示的行数。 如果使用表 格控制来显 示内表，那 么，LINES 将给出能在 屏幕上显示 的总行数。（如果内表 不包含行， 则屏幕表格 将在结尾包 含空行）。 系统还使用 LINES 来设置滚动 条的显示以 示意用户在 表格中向下 滚动了多远 。
TOP_LINE	integer	屏 幕开始显示 的表格行。
CURRENT_LINE	integer	循 环内当前正 被处理的行 。该字段是 绝对（非相 对）指标， 其值为 TOP_LINE + SY_STEPL.-1
LEFT_COL	integer	最 左边非固定 的列。因为 用户可以滚 动显示的非 固定部分， 所以该字段 控制着出现 在固定列后 面的列号。 LEFT_COL 给出列的绝 对（非相对 ）值，不管 用户是否重 新排列了列 的顺序。
LINE_SEL_MODE	integer	允 许使用行选 择。取值： 0＝不选， 1＝只选一 行，2＝允 许选多行。
COL_SEL_MODE	integer	允 许使用列选 择。取值： 0＝不选， 1＝只选一 列，2＝允 许选多列。
LINE_SELECTOR	char 1	指示器： 显示行选择 栏，这是一 个能在 ABAP/4 程序中进行 检查的一般 复选框。用 户单击某复 选框时，系统就将它设 置为 X。
H_GRID	char 1	指示器： 显示水平网 格线
V_GRID	char 1	指示器： 显示垂直网 格线
COLS (OCCURS 10)	TAB_COLUMN	嵌 入内表：表 格中每个列 的一个表格 条目。

TAB_COLUMN 结构中的字 段说明了屏 幕表格中的 单个字段及 其列：

字段名	类型	用途
SCREEN	SCREEN	嵌 入 SCREEN 结构：所有 的字段出自 SCREEN 系统表格的 单个行。
INDEX	integer	显 示中列的当 前位置（用 于用户重新 排列了列顺 序的情况） 。
SELECTED	char 1	用户单击 该列时，(系统）将它 设为 X。
VISLENGTH	int1	字 段的可见长 度（字符数 ）。最大的 允许长度为 255 个字符。

“ 屏幕”类型 指的是 SCREEN 表格，系统 表格说明了 屏幕中的所 有字段。关 于使用 SCREEN 表格的完整 信息，参见 ： *修改屏幕*（页 **错误！链接无效。**）.

设 置表格控制 属性

使用 TABLEVIEW 控制结构中 的字段在运 行时设置屏 幕表格属性 。例如，设 置滚动变量 TOP_LINE， 事务 TZ60 使用语句：

```
FLIGHTS-TOP_LINE = 1。
```

起初，表格 控制字段从 屏幕制作器 中为控制所 指定的属性 中获得它们 的值。其后 ，大部分控 制字段可用 多种方法来 指定：由系 统指定、由 ABAP/4 程序指定或 由用户定制 显示。下表 确切地指定 每个字段的 设置方式：

字段名	初始 化方式	后来 设置值的方 式
FIXED_COLS	在 ABAP/4 中或默认 ＝ 0。	在 ABAP/4 中。
LINES	在 ABAP/4 中或默认 ＝ 0。	在 ABAP/4 中。如果使 用 LOOP AT <int-table>， 也可由系统 自动设置。
TOP_LINE	在 ABAP/4 中或默认 ＝ 1。	在 ABAP/4.中 ，如果用户 移动垂直滚 动条上的滑 块时，也可 由系统设置 。
CURRENT_LINE	默 认＝0。	在 ABAP/4 中不允许修 改。在循环 时由系统设 置。

		(CURRENT_LINE=TOP_LINE + SY_STEPL-1)
LEFT_COL	在 ABAP/4 中或默认 = 0。	在 ABAP/4 中，当用户 在水平滚动 条上移动滑 块时，还可 由系统设置 。
LINE_SEL_MODE	屏 幕 制 作 器 或 ABAP/4 程序。	在 ABAP/4 中。
COL_SEL_MODE	屏 幕 制 作 器 或 ABAP/4 程序。	在 ABAP/4 中。
LINE_SELECTOR	屏 幕 制 作 器 或 ABAP/4 程序。	在 ABAP/4 中。
H_GRID	屏 幕 制 作 器 或 ABAP/4 程序。	在 ABAP/4 中。
V_GRID	屏 幕 制 作 器 或 ABAP/4 程序。	在 ABAP/4 中。
COLS_SCREEN	屏幕制作器 、ABAP/4 程序或用户 定制。	在 ABAP/4 中。
COLS_INDEX	屏 幕 制 作 器 、 ABAP/4 程序或用户 定制。	在 ABAP/4 中。
COL_SELECTED	在 ABAP/4 中。	在 ABAP/4 中。

如果必要， 可以始终将 表格控制重 置为屏幕制 作器中所指 定的初始值 。为此，请 使用语句 REFRESH CONTROL <table-control> FROM SCREEN <screen>。 详细信息参 见 ABAP/4 联机关键字 帮助。

示例事务： 表格控制

诸如对表格 控制的编程 示例，参见 事务 TZ60 的代码。TZ60（ 开发级别 SDWA） 使用两个屏 幕显示航班 信息。在第 一屏，用户 指定航班的 航线并请求 显示。在第 二屏（屏幕 200）， 事务显示所 有计划航线 的航班。

查看代码时 ，要注意的 主要问题是 ：

表格数据 是如何获取 并向各处传 送的

在程序 ABAP/4 中将表格控 制中的字段 声明为数据 库字段（表 格 SFLIGHTS） 。要同时存 储几个 SFLIGHTS 行，程序将 使用内表 INT_FLIGHTS。

1. 在屏 幕 100 的 PAI 中，程序选 择给定的航 线的所有航 班并将它们 存储到内表 INT_FLIGHTS 中。
2. 在屏 幕 200 的 PBO 中，LOOP 语句在 INT_FLIGHTS 中循环，并 为每行调用 模块 DISPLAY_FLIGHTS。 DISPLAY_FLIGHTS 将 INT_FLIGHTS 行传送到 SFLIGHTS 工作区。SFLIGHTS 字段匹配屏 幕表格名， 因此每次循 环的结尾， 系统将 ABAP/4 值传送到屏 幕表格字段 中。

使用表格 控制字段直 接滚动

表格控制包 含了管理滚 动的信息。 这包括告诉 已填充的表 格行数以及 应当首先在 屏幕上显示 哪一表格行 的字段。对 于步循环和 表格控制， 系统都自动 使用滚动条 来管理滚动 。这包括滚 动实际屏幕 显示，以及 重置滚动变 量。而示例 程序提供了 使用 F21－F24 键滚动。在 这种情况下 ，程序必须 直接执行这 些功能。

1. 在屏 幕 100 的 PAI 中，程序将 “first-table-row” 变量（字段 TOP_LINE） 初始化为 1。
2. 在屏 幕 200 的 PAI 中，如果用 户已经使用 F21－F24 功能键做了 滚动，那么 ，程序将重新设置滚动 变量，尤其 是 TOP_LINE 字段。

屏幕流逻辑

事务 TZ60 中屏幕 200 的流逻辑（ 仅对于 PAI）如 下：

```
*---------------------------------------------------------------------*
*   Screen 200: Flow Logic                                            *
*&---------------------------------------------------------------------*
PROCESS BEFORE OUTPUT.
     MODULE STATUS_0200.
     LOOP AT INT_FLIGHTS WITH CONTROL FLIGHTS
                         CURSOR FLIGHTS-TOP_LINE.
     MODULE DISPLAY_FLIGHTS.
     ENDLOOP.
*
```

```
*
PROCESS AFTER INPUT.
     LOOP AT INT_FLIGHTS.
       MODULE SET_LINE_COUNT.
     ENDLOOP.
     MODULE USER_COMMAND_0200.
```

ABAP/4 代码

TOP 模块代码后 面的程序段 显示了相应 于后面代码 的定义。

```
*---------------------------------------------------------------------*
*   INCLUDE MTD40TOP                                                  *
*---------------------------------------------------------------------*
PROGRAM  SAPMTZ60 MESSAGE-ID A&.
     TABLES: SFLIGHT.
     <....Other declarations....>
     DATA INT_FLIGHTS LIKE SFLIGHT OCCURS 1 WITH HEADER LINE.

     DATA: LINE_COUNT TYPE I.

     CONTROLS: FLIGHTS TYPE TABLEVIEW USING SCREEN 200.
```

下面的 PBO 模块将内表 字段传送给 PBO 中相应的屏 幕表格字段 。

```
*&---------------------------------------------------------------------*
*&      Module  DISPLAY_FLIGHTS  OUTPUT
*&---------------------------------------------------------------------*
MODULE DISPLAY_FLIGHTS OUTPUT.
     SFLIGHT-FLDATE    = INT_FLIGHTS-FLDATE.
     SFLIGHT-PRICE     = INT_FLIGHTS-PRICE.
     SFLIGHT-CURRENCY  = INT_FLIGHTS-CURRENCY.
     SFLIGHT-PLANETYPE = INT_FLIGHTS-PLANETYPE.
     SFLIGHT-SEATSMAX  = INT_FLIGHTS-SEATSMAX.
     SFLIGHT-SEATSOCC  = INT_FLIGHTS-SEATSOCC.
ENDMODULE.
```

在 PAI 中，程序必 须再次循环 ，以便将屏 幕表格字段 传送回程序 中。在该循 环中，程序 可以使用 SY_LOOPC 来查找所传 送的行数。 SY_LOOPC 是一个系统 变量，它告 诉当前显示 中的总行数 。

```
*&---------------------------------------------------------------------*
*&      Module  SET_LINE_COUNT  INPUT
*&---------------------------------------------------------------------*
MODULE SET_LINE_COUNT INPUT.
     LINE-COUNT = SY-LOOPC.
ENDMODULE
```

事务 TZ60 允 许用户按功 能键（F21— F24）来 滚动此显示 。系统使用 滚动条自动 处理滚动， 但不用功能 键来滚动。 因此，屏幕 200 的 PAI 必须直接执 行功能键滚 动：

```
MODULE USER_COMMAND_0200 INPUT.
     CASE OK_CODE.
*      WHEN 'CANC'...
*      WHEN 'EXIT'...
*      WHEN 'BACK'...
*      WHEN 'NEW'...
       WHEN 'P--'.
              CLEAR OK_CODE.
              PERFORM PAGING USING 'P--'.
       WHEN 'P-'.
              CLEAR OK_CODE.
              PERFORM PAGING USING 'P-'.
```

```
      WHEN 'P+'.
             CLEAR OK_CODE.
             PERFORM PAGING USING 'P+'.
      WHEN 'P++'.
             CLEAR OK_CODE.
             PERFORM PAGING USING 'P++'.
    ENDCASE.
ENDMODULE.
```

通过将表格 控制字段重 置为新值， PAGING 例程使系统 可以滚动此 显示。TOP_LINE 字段告诉 LOOP 语句 PBO 中开始循环 的位置。

```
*&---------------------------------------------------------------------*
*&      Form  PAGING
*&---------------------------------------------------------------------*
FORM PAGING USING CODE.
    DATA: I TYPE I.
             J TYPE I.

    CASE CODE.
      WHEN 'P--'. FLIGHTS-TOP_LINE = 1.
      WHEN 'P-'.  FLIGHTS-TOP_LINE = FLIGHTS-TOP_LINE - LINE_COUNT.
                  IF FLIGHTS-TOP_LINE LE 0.
                     FLIGHTS-TOP_LINE = 1. ENDIF.
      WHEN 'P+'.  I = FLIGHTS-TOP_LINE + LINE_COUNT.
                  J = FLIGHTS-LINES - LINE_COUNT + 1.
                  IF J LE 0.
                     J = 1. ENDIF.
                  IF I LE J.
                     FLIGHTS-TOP_LINE = I.
                  ELSE. FLIGHTS-TOP_LINE = J.
                  ENDIF.
      WHEN 'P++'.
                  FLIGHTS-TOP_LINE = FLIGHTS-LINES - LINE_COUNT + 1.
                  IF FLIGHTS-TOP_LINE LE 0.
                     FLIGHTS-TOP_LINE = 1. ENDIF.
    ENDCASE.
ENDFORM.
```

使用步循环

步循环是屏 幕中的一系 列重复的字 段块。每一 个块可以包 含一个或多 个字段，并 能在屏幕上 扩展为多行 。

与屏幕中的 结构相同， 步循环没有 独立的名称 。屏幕可以 包含多个步 循环，然而 ，要是屏幕 确有多个步 循环，那么 ，用户必须 相应地在流 逻辑中编制 LOOP...ENDLOOP 语句。LOOP...ENDLOOP 的顺序必须 完全同屏幕 中步循环的 顺序相平行 。该顺序告 诉系统哪个 循环处理应 用于哪个循 环。屏幕中 的步循环首 先由屏幕的 行来排列顺 序，其次才 是屏幕的列 。

事务 TZ61（开发级别 SDWA） 执行您在事 务 TZ60 中看到的表 格的步循环 的版本。 TZ61 的屏幕 200，显 示下列步循 环：

关于如何使 用步循环的 说明，参见 TZ61 代码。

静态和动态 步循环

步循环分为 两类：静态 的和动态的 。静态步循 环有固定的 大小，运行 时无法更改 。动态步循 环大小可变 。如果用户 重置窗口的 大小，系统 就会自动地 增加或减小 所显示的步 循环块的个 数。在任何 给定的屏幕 中，您可以 定义任意数 量的静态步 循环，但动 态步循环只 能有一个。

可以在屏幕 制作器中指 定步循环的 类型。屏幕 中的每一个 循环都有“ 循环类型” （*fixed*= 静态，*variable*= 动态）和“ 循环次数” 属性。如果 循环是固定 的，“循环 次数”告诉 系统用于显 示的循环块 的数量。该 数值可以永 久不变。

编制动态和 静态步循环 的方法完全 一样。两种 类型都可以 使用 LOOP 和 LOOP AT 语句。

在步循环中 循环

当把 LOOP AT <internal-table> 用于步循环 时，系统使 用垂直滚动 条自动地显 示步循环。 滚动条以及 已更新（已 滚动）的表 格显示由系 统来进行管 理。

如果需要， 使用如下附 加参数：

FROM <line1> and TO <line2>

这些参数限 制了在步循 环中能够显 示或处理的 内表部分。 如果您使用 其中的一个 或两个，请 在 ABAP/4 中使用 LIKE SY-TABIX 声明 <line1> 和 <line2>。 如果您不使 用它们，则 系统仅使用 内表的开始 /结束部分 来限制。

CURSOR <scroll-var>

CURSOR 参数告诉哪 一个内表行 是屏幕显示 中的第一行 。<Scroll-var> 是一个局部 程序变量，它能够由程 序设置，也 可以由系统 自动设置。 <scroll-var> 值是对应于 内表的绝对 值（即与 FROM 或 TO 值无关）。

在 PBO 事件中需要 CURSOR <scroll-var> 参数以告诉 系统开始显 示的位置。

第七章 转到列表过 程

概览

如果经常想 从您的事务 中生成一个 列表。可使 用下面两种 方法。

内容

提交一个 独立的报告
使用 SUBMIT 语句直接从 事务中启动 一个独立的 报表。SUBMIT 语句的用法 在*提交报表*（页 **Error! Not a valid link.**）中详细说 明。
从模块存 储中使用 LEAVE TO LIST-PROCESSING 生成列表。

LEAVE TO LIST-PROCESSING 语句是从模 块存储中生 成列表时所 使用的语句 。该语句允 许您在对话 程序中从对 话模式切换 到列表模式 。可在模块 存储中对所 需的列表处 理逻辑进行 编码。
当运行 LEAVE TO LIST-PROCESSING 语句时，模 块存储仍旧 保持执行的 控制权。事 务的数据区 域对报表处 理编码来说 完全可用， 因此没必要 来回传送参 数。
当您进入列 表模式时， 可以使用交 互式报表可 用的所有 ABAP/4 工具来生成 报表。当您 想定制字段帮助或可能 取值的显示 时，这对 PROCESS ON VALUE-REQUEST 或 PROCESS ON HELP-REQUEST 过程尤其有用。
有关信息参 见下列资料 ：
使用 LEAVE TO LIST-PROCESSING（页 422）
在列表模式 中使用 GUI 状态（页 432）
返回对话模 式（页 426）
关于如何进 入列表模式 的具体示例 ，参见事务 TZ70（ 在同系统一 起发表的开 发类 SDWA 中）和本章中的讨论。
事务示例： 转到列表模 式（页 426）

使用 LEAVE TO LIST-PROCESSING

要使用 LEAVE TO LIST-PROCESSING， 请把该语句 放在开始列 表模式处理 的代码中。 该语句要做 两件事：

切换到列 表模式处理
从此处开始 ，可以编码 发布和控制 报表所需的 报表语句。 所有标准的 同报表有关 的事件和特 征都可用： AT LINE-SELECTION、 功能键、基 本的详细列 表层、窗口 等等。
把标准列 表输出设置 为当前屏幕 的“下一屏 ”
系统提示标 准列表输出 应当在当前 屏幕显示之 后。根据应 用程序需要 ，可以让两 者都显示，或者禁止当 前屏幕显示 而替换为列 表输出。

由清除器封 装所有单个 子程序中的 报表代码是 很普遍的事 。事务 TZ70 的示例如下 ：

```
**** ABAP/4 模块和格式 ： ****
MODULE PREPARE_LIST OUTPUT.
    LEAVE TO LIST-PROCESSING AND RETURN TO SCREEN 0.
    PERFORM EDIT_LIST.
    LEAVE SCREEN.
ENDMODULE.
FORM EDIT_LIST.
    SET PF-STATUS 'LIST'.
    SET TITLEBAR  'LST' WITH SFLIGHT-CONNID SFLIGHT-CARRID.

    NEW-PAGE LINE-SIZE 72.
    SELECT * FROM SFLIGHT WHERE  CARRID = SFLIGHT-CARRID
                          AND   CONNID = SFLIGHT-CONNID.
      WRITE: /    SY-VLINE NO-GAP,
                  SFLIGHT-FLDATE    COLOR 4 INTENSIFIED OFF NO-GAP,
                  SY-VLINE NO-GAP,
                  SFLIGHT-PRICE     COLOR 2 INTENSIFIED OFF NO-GAP,
                  .......
ENDFORM.
```

列表模式在 对话模式中 如何工作

运行时，模 块存储保持 执行控制。 可以用 PBO 或 PAI 为当前屏幕 编制列表模 式逻辑代码 。选择哪一 个取决于您 希望列表在 当前屏幕之 后输出，还 是取代当前 屏幕。在两 种情况下， 当前屏幕终 止时，列表 都会出现。 （当控制遇 到 LEAVE SCREEN 语句或者到 PAI 结尾时屏幕 处理都会终 止。）

要在显示 当前屏幕之 后显示列表 输出：

把 LEAVE TO LIST-PROCESSING 逻辑放在 PAI 的结尾。以 这种方式编 码，程序就 会在当前 PAI 处理中响应 列表输出请 求。从列表 显示中返回 时，系统就 会从 PBO 的头部开始 ，重复当前 屏幕过程。

要显示列 表输出而不 显示当前屏 幕：

在 PBO 中编制 LEAVE TO LIST-PROCESSING 逻辑代码， 后面加上 LEAVE SCREEN。 这样就告诉 系统显示列 表而不显示 当前屏幕。 当前屏幕的 PAI 过程将不执 行。

详细信息， 参见：

后台处理屏 幕 (页 **错误！链接无效。**)

事务示例： 转到列表模 式 (页 426)

在列表模式 中使用 GUI 状态

可以使用列 表的标准 GUI 状态，或者 定义自己的 界面。如果 使用标准列 表状态，系 统将自动替 您执行许多 GUI 功能。例如 ：

两个返回 功能 (BACK, CANCEL)

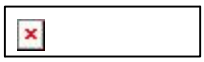
“打印” 和“搜索” 功能

标准滚动 键 (**P+, P++, P-, P--**)

您不需要在 自己的代码 中安排这些 功能。

如果要定义 自己的状态 ，您必须做 两件事。首 先，在 GUI 状态中显式 地激活要求 的功能（通 过在相应的 按钮和功能 码中输入它 们的功能码)。其次， 在 ABAP/4 模块中，系 统没有自动 处理编码逻 辑的功能。 包括 **F21－F24** 键，尽管 P-、P-、 P+ 和 P++ 功能已自动 定义。

事务示例 TZ70 定 义了自己的 状态（命名 为 LIST） 。因为该状 态有“对话 框中的列表 ”类型，所 以只能定义 按钮（不需 要菜单条） 。因为系统 自己执行了 退出功能， 所以“返回 ”和“取消 ”功能（BACK und RW）自动 显示为按钮 。



要想得到列 表模式下系 统自动处理 的所有功能 列表，请执 行下列操作 ：

1. 在“ 菜单画笔” 工作簿中， 选定“实用程序 -> 展开 -> 常规/建 议”。
2. 在弹 出结果中， 双击“列出 功能”元素 层次结构 。

关于交互式 报表技术的 信息，参见 *交互式列表* (页 **错误！链接无效。**) 。

返回对话模 式

有两种方法 可以从列表 模式返回到 对话模式。 在这两种情 况下，程序 返回的地方 都默认为进 行列表处理 的屏幕。(包含 LEAVE TO LIST-PROCESSING 语句的屏幕 。)

如果想返回 其它屏幕， 参见*返 回其它屏幕* (页 410) 。

您的程序将 继续在列表 模式下运行 ，除非遇到 下列情况：

系统在代 码中遇到一 条 LEAVE LIST-PROCESSING 语句。

LEAVE LIST-PROCESSING 语句将控制 返回到对话 屏幕。返回 时，系统在 PBO 的开始处重 新启动过程 。

用户从报 表的基本列 表层中请求 “返回”或 “取消”。

如果用户使 用“返回” 或“**取消**” 图标退出列 表，您不必 安排显式的 LEAVE LIST-PROCESSING 语句。“返 回”或“取 消”是系统 自动运行的 标准返回功 能。当用户 按下了其中 的一个，系 统就返回到 含有 LEAVE TO LIST-PROCESSING 的屏幕中。 这里系统重 新启动 PBO 处理屏幕。（记住，“ 退出”功能 不同于“返 回”或“取 消”，它不 是由系统运 行的。如果 您想要提供 “退出”功 能，就必须 自己实现。 在这种情况 下，请使用 一条 LEAVE LIST-PROCESSING 语句来实现 。）

事务示例 TZ70 使用第二种 方法从列表 模式中返回 。因为系统 自动提供了 “返回”和 “取消”功 能，当事务 从屏幕 200 进入列表模 式时，并没 有编制退出 功能代码。而对于没有 列表过程的 屏幕 100 来说，程序 必须执行自 己的退出功 能（“返回 ”、“退出 ”和“取消 ”）：

```
MODULE EXIT_0100 INPUT.
    CASE OK_CODE.
      WHEN 'CANC'.
            CLEAR OK_CODE.
            SET SCREEN 0. LEAVE SCREEN.
      WHEN 'EXIT'.
            CLEAR OK_CODE.
            SET SCREEN 0. LEAVE SCREEN.
      WHEN 'BACK'.
```

```
            CLEAR OK_CODE.
            SET SCREEN 0. LEAVE SCREEN.
    ENDCASE.
ENDMODULE.
```

返 回其它屏幕

当返回对话 模式时，程 序还能重新 指定用户路 径以便到达 与启动列表 屏幕不同的 屏幕。为此 ，请在首次 转到列表模 式时使用关 键字 AND RETURN TO SCREEN:

```
    LEAVE TO LIST-PROCESSING AND RETURN TO SCREEN 100.
```

使用该语句 ，无论程序 何时返回到 对话模式（ 因为用户从 列表中退出 ，或执行了 命令 LEAVE LIST-PROCESSING） ，系统都恢 复屏幕请求 的 PBO 过程（这里 是屏幕 100)。

事务 TZ70 使 用模块 PREPARE_LIST 中的 AND RETURN TO SCREEN 0:

```
      ** PROCESSING FOR SCREEN 200**
      MODULE PREPARE_LIST OUTPUT.
        LEAVE TO LIST-PROCESSING AND RETURN TO SCREEN 0.
        PERFORM EDIT_LIST.
        LEAVE SCREEN.
      ENDMODULE.
```

这里，返回 到屏幕 0 将 关闭 CALL 模式，该模 式是 CALL SCREEN 触发屏幕 200 时启动的。 系统将控制 返回到先前 最后调用模 式的屏幕（ 那就是，屏 幕 100 的 PBO）

事务示例： 转到列表模 式

事务 TZ70 介绍了在事 务内部进行 列表处理的 一种方法：

1. 屏幕 100 PAI： CALL SCREEN 200.
2. 屏幕 200 PBO： 转到列表处 理，生成列 表并离开屏 幕 200
   - 屏幕 200 无显示（由 LEAVE SCREEN 命令禁止）
3. 屏幕 200 PAI：没 有需要的编 码过程（由 LEAVE SCREEN 禁止）
   - 进行列 表显示输出 ：
   - 返回到 屏幕 100 的 PBO（因 为用户按了 “取消”或 “结束”）
4. 屏幕 100 PBO：重 新执行 PBO...

屏幕流逻辑

两种屏幕流 逻辑的相关 部分是：

```
*---------------------------------------------------------------------*
*   Screen 100: Flow Logic  (PAI only)                                *
*&---------------------------------------------------------------------*
PROCESS AFTER INPUT.
     MODULE EXIT_0100 AT EXIT-COMMAND.
     MODULE USER_COMMAND_0100.

*---------------------------------------------------------------------*
*   Screen 200: Flow Logic                                            *
*&---------------------------------------------------------------------*
PROCESS BEFORE OUTPUT.
     MODULE PREPARE_LIST.
*
PROCESS AFTER INPUT.
```

ABAP/4 代码

屏幕 100 的主 PAI 模块调用屏 幕 200，该 屏幕将作为 单独的对话 框出现（弹 出）：

```
*&---------------------------------------------------------------------*
*&      Module  USER_COMMAND_0100  INPUT
*&---------------------------------------------------------------------*
MODULE USER_COMMAND_0100 INPUT.
     CASE OK_CODE.
       WHEN SPACE.
            <...Select flight info for the requested flight..>
            CLEAR OK_CODE.
            CALL SCREEN 200 STARTING AT 10 5 ENDING AT 80 15.
     ENDCASE.
ENDMODULE.
```

屏幕 200 的 PBO 模块将设置 列表模式分 支：

```
*&---------------------------------------------------------------------*
*&      Module  PREPARE_LIST  OUTPUT
*&---------------------------------------------------------------------*
MODULE PREPARE_LIST OUTPUT.
     LEAVE TO LIST-PROCESSING AND RETURN TO SCREEN 0.
     PERFORM EDIT_LIST.
     LEAVE SCREEN.
ENDMODULE.
```

产生列表的 子例程包含 有规律的报 表过程（部 分显示）：

```
*&---------------------------------------------------------------------*
*&      Form  EDIT_LIST
*&---------------------------------------------------------------------*
FORM EDIT_LIST.
     SET PF-STATUS 'LIST'.
     SET TITLEBAR  'LST' WITH SFLIGHT-CONNID SFLIGHT-CARRID.

     NEW-PAGE LINE-SIZE 72.
     SELECT * FROM SFLIGHT WHERE CARRID = SFLIGHT-CARRID
                           AND  CONNID = SFLIGHT-CONNID.
     WRITE: / SY-VLINE NO-GAP,
                    SFLIGHT-FLDATE    COLOR 4 INTENSIFIED OFF NO-GAP,
*                   .......
     ENDSELECT.
     IF SY-SUBRC = 0. ULINE. ENDIF.
ENDFORM.
```

列表模式事 件 TOP_OF_PAGE 是列表显示 编制的：

```
TOP-OF-PAGE.
     ULINE.
     WRITE: / SY-VLINE NO-GAP,
                    ......
```

第九章　编程数据库 更新

概览

要有效地进 行编程数据 库更新，程 序员必须主 要关注于：

内容

保持数据 库的正确性
优化用户 的反应时间

SAP 系统提供了 许多更新数 据库的方法 。每种方法 都利用了 SAP 体系结构的 不同特征。 本章用以描述这些方法 并提供选择 最适合用户 应用程序的 方法指南。 下列主题提 供信息：

SAP 数据库环境 (页 348)
更新编程技 术 (页 346)
SAP 系统中的锁 定 (页 344)

SAP 数据库环境

如果正在事 务中更新数 据库，那么 SAP 数据库环境 提供可想而 知的一些专 用特征。关 于编程更新 之前的快速 入门，参见 ：

SAP 系统中的事 务 (页 376)
更新绑定介 绍 (页 367)
SAP 锁定介绍 (页 350)

R/3 体系结构： 概述

为了了解 R/3 系统如何执 行数据库更 新，应该熟 悉 R/3 体系结构。

客户机/服 务器配置

SAP 的三级客户 机/服务器 配置包括演 示（用户前 端），应用 逻辑（用户 处理对话） 和数据存储 （执行数据 库请求）。 演示服务器 使用 SAPGUI 程序提供 R/3 用户界面。 如果用户启 动事务，则 SAPGUI 传递条目到 调度程序。 调度程序将 挂起的处理 任务分配给 多个工作进 程。具体的 工作进程数 取决于配置 。工作进程 可以直接访 问数据库， 该数据库使 用共享服务 在不同的计 算机上存储 。

多用户系统

该对话工作 进程交替控 制当前用户 会话。如果 用户输入动 态处理请求 系统处理， 则调度程序 将其放入队 列。一旦对 话工作进程 为空，则队 列顶部的作 业将被分配 给该进程以 便进行处理 。然后，对 话工作进程 执行一对话 步骤。每次 更改屏幕之 后，都将触 发数据库的 提交。这样 将清空屏幕 间的对话进 程，以允许 其处理由调 度程序所分 配的新对话 步骤。

切换工作进 程和更新数 据库

由于已将任 务分配给了 空的工作进 程，因此必 须优化系统 资源的使用 并优化负载 的承担。但 是，设计事 务流时，事 务开发者必 须记住在事 务处理期间 工作进程必 须切换多次 。如果事务 更改了数

据 库，那么， 此事实十分 重要，因为 一旦工作进 程清空，则 系统释放数 据库锁定并 关闭数据库 光标。要保 持更改正确 ，必须知道 该点何时到 达以及应采 取的行动。 本章将说明 用于编程的 校正和有效 数据库更改 的技术。ABAP/4 提供的优化 数据库更新 的技术独立 于基础数据 库，并且符 合对话编程 的特别要求 。

更新和排队 服务
除对话进程 外，每个 R/3 系统包含一 个或多个更 新服务和一 个排队服务 。使用这些 服务更新 R/3 中的数据库
对话事务可 以直接或间 接更改数据 库。对于直 接更改，由 对话工作进 程执行更新 程序。强制 对话用户等 待直到作任 何输入之前 更新操作已 经完成为止 。在异步更 新中，事务 的对话部分 与数据库的 实际更新分 离（例如， 由于性能的 原因）。专 门的更新工 作进程执行 数据库更新 。
排队服务管 理 R/3 系统中的内 部锁定。按 照规则，关 系型数据库 系统的锁定 机制不满足 R/S 的要求。由 不同工作进 程处理对话 步骤的事务 甚至在切换 时都必须保 留赋予的锁 定。每个锁 定不仅适用 执行锁定事 务的应用服 务器，而且 适用其它任 何客户机/服 务器配置。 因此，每个 R/3 系统只包含 一个排队服 务。（关于 逻辑锁定的 详细信息， 参见 *SAP 锁定介绍* （页 350） ）。

SAP 系统中的事 务

在通常意义 下，*事务*是 让用户更改 数据库的操 作。该操作 必须以“全 部或没有” 的方式执行 。如果事务 运行成功， 则应该执行 所有的更改 。如果事务 遇到错误， 则不执行任 何更改。
当在事务的 中途发生错 误时，应该 取消该点前 的任何数据 库更改。这 将使数据库 保留事务开 始前的状态 。
在 SAP 系统中，数 据库级事务 和作为程序 员设计的事 务之间有重 要区别。此 区别（本节 中说明）是 用于执行数 据库更新的 绑定技术的 动机。

事务和 LUW
在 SAP 系统中，单 词事务有几 种意义：
数据库事 务（“LUW” 或“数据库 LUW”）
在数据库世 界中，“全 部或没有” 事务称为 LUW（工 作逻辑单元 ）。LUW 是必须作为 单元执行的 所有更新请 求的时间段 。在 LUW 结尾，系统 或者提交对 数据库的更 新或者丢弃 这些更新（ 丢弃更改称 为“反转” ）。每次提 交或反转既 是一个 LUW 的结束又标 志着下一个 LUW 的开始。
S 系统在每次 更改屏幕时 自动触发数 据库的提交 操作。这意 味着数据库 LUW 最长维持从 一个屏幕到 下一个屏幕 的更改。

更新事务 （“SAP LUW”）
事务是组成 逻辑单元的 业务相关任 务。该任务 的执行需要 几个 SAP 屏幕来实施 ，但是可以组成单个功 能（例如， 显示数据库 、更改数据 库或者发送 消息）。不 是所有事务 都涉及更新 数据库。但 是在本章中 ，我们只讨 论执行更新 的事务（“ 更新事务” ）。
作为逻辑单 元，更新事 务应该全部 执行或根本 不执行。因 此有时称它 们为 SAP LUW，以 使其同数据 库 LUW 区分开来。 一般地，更 新事务通常 跨越几个数 据库 LUW，并 且在 ABAP/4 级使用 COMMIT WORK 命令关闭。 （该命令执 行几个任务 ，其间触发 数据库执行 ）。

ABAP/4 事务（“SAP 事务”）
ABAP/4 事务是在单 个事务代码 下结合的业 务相关任务 集。ABAP/4 事务可能包 括几个“更 新事务”（ 如同以前所 作的说明） 。作为程序 ，ABAP/4 事务是包括 模块存储、 屏幕、菜单 界面、事务 代码等等的 复杂对象。

LUW 和更新事务
“LUW” （工作逻辑 单元）是数 据库更新必 须以“全部 或没有”方 式执行的时 间段。或者 全部执行（ 提交），或 者丢弃（返 回）。在 ABAP/4 系统中，LUW 和事务可以 有几个意义 ：
LUW（ 或“数据库 LUW”或 “数据库事 务”）
这是由数据 库提交终止 的更新集。 LUW 最长从一个 屏幕更改维 持到另一个 屏幕更改（ 因为 SAP 系统在每次 屏幕更改时 自动触发数 据库的提交 ）。
更新事务 （或“SAP LUW”）
这是由 ABAP/4 提交终止的 更新集。SAP LUW 的维持可以 比数据库 LUW 的维持长得 多，原因是大多数更新 处理超过多 重事务屏幕 。程序员通 过发出 COMMIT WORK 语句来终止 更新事务。

更新绑定介 绍

用于绑定更 新的 ABAP/4 技术使您可 以编程超过 几个屏幕的 全部或没有 的事务（逻 辑 LUW）。 只能在结尾 时提交对这 些事务的更 改。该事务 有两个重要 特征：
可以避免 每次屏幕更 改时提交更 新。
可以锁定 即将跨过多 重屏幕予以 更新的对象 。
介绍主题是 ：
为什么更新 绑定？ （页 382）

绑定技术概 要 (页 402)
不同工作进 程中的更新 (页 373)
同步和异步 更新 (页 369)
关于绑定更 新的详细信 息，参见：
更新编程技 术 (页 346)

为什么更新 绑定？
SAP 系统在每次 屏幕更新和 其它时间（ 每个 CALL SCREEN、 CALL DIALOG 或 CALL TRANSACTION 语句，每个 MESSAGE 语句和每个 远程函数调 用）自动触 发“数据库 提交”。
这些数据库 提交允许系 统在屏幕间 释放对话任 务。提交在 系统内部发 生，并且不 受程序员影 响。无论何 时出现运行 时间错误或 者程序发出 错误消息（ MESSAGE 类型“A” ），都会触 发数据库反 转。一般地 ，可以将数 据库 LUW 看作是跨越 单个屏幕的 所有处理。
但是编程的 更新事务， 通常跨越几 个屏幕。因 此，不能依 靠数据库提 交结束更新 。如果已经 这样做，那 么，只要允 许（在每次 屏幕更改时 ）系统将提 交更新，并 且不能反转 。如果稍后 发生错误，或者用户要 取消此操作 ，那么，由 于太迟将无 法反转以前 屏幕上所作 的更新。
数据库锁定 涉及相关问 题。系统自 动创建并释 放数据库锁 定。为每条 更新语句创 建数据库锁 定并在每个 数据库提交 时予以释放 。然后在每 次屏幕更改 时释放数据 库锁定，这 意味着不提 供对要锁定 保护长于单 个屏幕的对 象的额外的 访问。
要定址这些 主题，SAP 系统提供：

在 ABAP/4 中的更新绑 定技术
ABAP/4 提供在特别 更新例程中 的绑定更新 命令。推迟 执行这些例 程直到程序 发出显式的 “SAP 提交”。SAP 提交是触发 数据库提交 的 ABAP/4 语句（COMMIT WORK） ，但也执行 其它功能。 通过这些技 术，可以在 更新事务结 束处执行更 新，而不是 在每次屏幕 更新时。
SAP 锁定
系统允许在 数据库对象 上定义 SAP 锁定。SAP 锁定是 SAP 系统中的逻 辑锁定，而 不是自动应用的数据库 锁定。通过 SAP 锁定，可以 锁定即将跨 越多重屏幕 予以更新的 对象。

绑定技术概 要
通过更新绑 定，可以将 更新包装到 只有当程序 发出 ABAP/4 提交时才运 行的特殊例 程。（或者 ，如果程序 发出 ABAP/4 反转，则取 消此执行） 。为此，请 使用：
PERFORM ON COMMIT
CALL FUNCTION IN UPDATE TASK
CALL FUNCTION IN BACKGROUND TASK
这些语句指 定不立即执 行给定的 FORM 例程或者功 能模块，而 是在下次 ABAP/4 提交时执行 。ABAP/4 提交是触发 数据库提交 的 ABAP/4 语句，同时 也执行其它 功能。执行 这些提交和 反转的 ABAP/4 语句是：
COMMIT WORK
ROLLBACK WORK
COMMIT WORK 和 ROLLBACK WORK 语句执行与 任务同步执 行相关的许 多功能。其 中之一用于 关闭更新事 务，可以触 发数据库提 交或数据库 反转。如同 LUW 中的数据库 提交（或反 转），COMMIT WORK 和 ROLLBACK WORK 将“操作边 界”（逻辑 开始和结束 ）定义为全 部或没有的 更新集。因 此，在*更新事务* (页 436) 最后的 LUW 中使用这些 语句。
使用 ABAP/4 绑定技术， 可以根据需 要定制更新 过程。介绍 信息，参见 ：
不同工作进 程中的更新 (页 373)
同步和异步 更新 (页 369)

不同工作进 程中的更新
ABAP/4 绑定技术可 以将更新分 配到不同的 工作进程。 使用每个可 能的技术， 用 COMMIT WORK 语句触发实 际执行：

在对话任 务中的更新
PERFORM ON COMMIT 语句在对话 任务中调用 表单例程， 但是推迟其 执行直到系 统遇到下一 条 COMMIT WORK 语句为止。 由于 COMMIT WORK 语句出现在 更新事务的 逻辑结束处 ，所以任何 通过 PERFORM ON COMMIT 调用的表单 例程中的更 新语句都在 *更新事务* (页 436) 中的最后 LUW 中运行。

详细信息， 参见 *在对话任务 中的绑定更 新* (页 329) 。

更新任务 中的更新
CALL FUNCTION IN UPDATE TASK 语句记录在 更新任务中 执行的功能 模块。后续 的 COMMIT WORK 语句触发实 际执行。

在更新任务 中运行的功 能模块，有 指定执行功 能如何运行 和何时运行 的属性（“ 进程类型” 字段）。一 些更新功能 （称为“V1” 功能）和相 同进程类型 的其它功能 一起在单个*更新事务* (页 436) 中运行，并 按请求顺序 立即执行。 其它更新功 能（称为“ V2”功能 ）总在其单 个更新事务 中运行。这 些功能有推 迟起点的“ 进程类型” ，并且用于 较少的关键 请求。
详细信息， 参见：*更新任务中 的绑定更新* (页 332) 。

后台任务 中的更新（ 远程主机）
CALL FUNCTION IN BACKGROUND TASK 语句记录在 后台任务中 运行的功能 模块。通常 ，该语句用 于执行远程 主机上的函 数（通过指 定附加参数 DESTINATION） 。后面的 COMMIT WORK 语句触发实 际远程函数 调用。
后台任务函 数作为低优 先级请求处 理,但是相 同目的地的 所有请求在 同一更新事 务中运行。

详细信息， 参见：*后台任务中 的绑定更新*（页 334）。
关于 RFC 编程的详细 信息，参见 *远程通讯*（页 **Error! Not a valid link.**）。

同步和异步 更新
“同步”和 “异步”概 念是等待概 念的关键。 程序请求系 统执行某一 任务，然后 或者等待或 者不等待完 成任务。在 同步处理中 ，程序等待 ：只有任务 完成之后控 制才返回到 程序。而在 异步处理中 ，程序不等 待：在记录 执行请求之 后系统返回 控制。
在 SAP 数据库系统 中，等待不 能在请求语 句（CALL FUNCTION 或 PERFORM） 处发生，但 可在 COMMIT WORK 触发执行时 发生。同步 更新在 COMMIT WORK 语句完成时 保证完成。 异步执行不 保证完成。
SAP 绑定技术允 许两种处理 ：

同步更新 ：
- 通过 PERFORM ON COMMIT 调用 FORM 例程
- 通过 COMMIT WORK AND WAIT 触发更新任 务功能模块

异步更新 ：
- 通过 COMMIT WORK 触发更新任 务功能模块
- 通过 COMMIT WORK 触发后台任 务功能模块

详细信息， 参见 *优化事务性 能*（页 367）。

SAP 锁定介绍

要支持更新 绑定计划， SAP 系统提供了 完全不同于 数据库锁定 的锁定机制 。SAP 锁定的优点 在于可以在 多重屏幕间 保留它们， 如同 *更新事务*（页 436）所需要的。
数据库锁定 是数据库系 统中的物理 锁定。在程 序中使用更 新语句（SELECT SINGLE FOR UPDATE、INSERT、 UPDATE、 MODIFY、 DELETE） 时，系统自 动创建数据 库锁定。数 据库锁定在 每次数据库 提交（即每 次屏幕更改 ）时自动释 放。因此， 数据库锁定 对长于一个 屏幕的锁定 无效，并且 作为程序员 也无法控制 。
SAP 锁定是 SAP 系统定义的 逻辑锁定。 要使用它们 ，首先定义 指定要锁定 数据库对象 的锁定对象 。激活锁定 对象时，系 统生成所定 义的每个锁 定对象的锁 定和解锁例 程（称为 **ENQUEUE-<object>** 和 **DEQUEUE-<object>** 的功能模块 ）。调用这 些功能模块 从 ABAP/4 程序中直接 设置和释放 锁定。
关于使用 SAP 锁定的详细 信息，参见 *SAP 系统中的锁 定*（页 344）。

更新编程技 术

要在事务中 实施数据库 更新，参见 下列主题：

维护数据库 完整性（页 327）
优化事务性 能（页 367）
未绑定的更 新（页 332）
在对话任务 中的绑定更 新（页 329）
更新任务中 的绑定更新（页 332）
后台任务中 的绑定更新（页 334）
COMMIT WORK 处理（页 333）
ROLLBACK WORK 处理（页 334）
后台处理考 虑（页 375）
绑定更新的 错误处理（页 373）

维护数据库 完整性

本节说明通 过 R/3 系统保证数 据完整性的 形式，和必 须在程序中 保证的东西 。

语义完整性
语义完整性 描述数据的 正确性和完 整性。尤其 适用于包括 几个步骤的 更新操作。 必须执行所 有相关步骤 以保证语义 的完整性。 一些数据库 系统使用约 束和触发技 术来保证语 义的完整性 ，但 SAP 数据库却并 不如此。用 户必须：

编程事务 以便对操作 作所有必须 的检查
发出 COMMIT WORK 语句以使更 改结束

关系完整性
关系完整性 意味着坚持 用于设计数 据库的关系 模型的规则 。所有 SAP 数据库都是 关系型数据 库。通常， 有三种类型 的关系完整 性：

主码完整 性
数据库中的 每个对象必 须由其主码 唯一标识。 在 ABAP/4 词典中不可 能创建没有 主码的表格 。因此，整 个 R/3 系统保证主 码的完整性 。不必检查 数据库数据 的唯一性。

值设置完 整性和**外来 关键字完整 性**

在关系数据 库中，没有 字段能包含 非该字段的 允许值的值 。同样，表 中的每个外 来关键字 必 须参照关联 检查表格中 的主码。许 多数据库系 统自动检查 维护的这些 条件。但是 SAP 系统不这样 做。在编程 事务时，必 须牢记两级 保护：

- 从屏幕 数据设置数 据库值

  可以在 ABAP/4 词典中指定 字段的允许 值集或外来 关键字。该 词典使用这 些信息请求 联机用户输 入有效屏幕 值。如果通 过使用屏幕 字段输入， 程序更新数 据库，则数 据库保证有 效。
- 从非屏 幕数据设置 数据库值

  使用非屏幕 源中的数据 ，任何 ABAP/4 程序可以更 新数据库。 这些数据不 保证有效。 在这种情况 下，编程所 有必须的检 查以保证数 据库值的正 确性是用户 的责任。

操作完整性

操作完整性 指避免多个 用户同时更 新相同的数 据库对象。 这个问题的 标准解决方 案是每个用 户锁定自己 当前正使用 的对象，以 避免其它人 访问。

SAP 系统提供了 SAP 锁定，一种 定义和将逻 辑锁定应用 于数据库对 象的机制。 该机制比数 据库系统提 供的物理锁 定更复杂。

要保证操作 完整性，必 须在事务中 使用 SAP 锁定机制。 详细信息， 参见 *SAP 系统中的锁 定* (页 344) 。

优化事务性 能

访问数据库 可能很费时 。事务访问 数据库花的 时间越多， 用户的响应 时间越长。 要优化性能 ，可以将部 分事务工作 分散给更新 任务（或后 台任务）处 理。

选择异步处 理

异步处理意 味着可将更 新分发给其 它工作进程 或应用服务 器。这通常 可降低用户 的响应时间 。应该将更 新异步运行 吗？这实际 上是折衷选 择：

任务结果

通过同步处 理，当控制 返回到请求 程序时，保 证完成处理 请求。程序 需要完成更 新以便继续 处理吗？如 果是，则使 用同步处理 。

响应时间

同步处理响 应时间长： 用户必须等 待已完成的 更新。如果 不急于要求 立即更新数 据库，则使 用异步更新 。

但是，异步 更新确实涉 及某些开销 。记录更新 任务请求时 ，系统进行 两次额外数 据库访问： 一次输入更 新到日志文 件，一次读 出它们。这 些访问甚至 不包括更新 任务执行的 实际更新。 一次输入更 新到日志文 件，一次读 出它们。这 些访问甚至 不包括更新 任务执行的 实际更新。 一般地，如 果下列条件 适用则异步 更新有效：

对话任务 中的用户响 应时间非常 重要。

响应时间 有些重要， 并且要求的 更新如此复 杂以至于超 过记录请求 的开销。

例如，在批 处理工作进 程中运行事 务时，异步 处理没有优 点。详细信 息，参见 *后台处理考 虑* (页 375) 。

LUW 处理：哪些 任务互相依 赖？

一些更新请 求相互依赖 ：必须一起 执行它们（ 或反转）。 其它请求可 以完全独立 运行。必须 决定更新是 否需要作为 逻辑单元一 起运行。如 果需要,则 应该或者在 对话任务中 一起运行，或者在更新 任务中作为 共同 LUW 请求运行。

对于更新任 务更新，在 设置更新任 务功能模块 的“进程类 型”属性时 指定如何处 理请求。“ 进程类型” 决定是在共 同的 LUW 中还是在单 个 LUW 中运行更新 请求。详细 信息，参见 *更新任务处 理如何工作* (页 **Error! Not a valid link.**) 。

更新优先级 ：如何执行 卸载任务？

异步更新请 求可以立即 运行或延迟 。立即开始 更新是按其 提交的顺序 执行的共同 LUW 请求。在独 立的 LUW 中，延迟开 始请求以后 以任何顺序 运行。它们 可能分发到 不同工作进 程。

进程类型属 性（用于更 新任务功能 模块）决定 是否立即或 延迟运行更 新任务请求 。其计时如 何关键？必 须按请求的 顺序运行吗 ？详细信息 ，参见 *更新任务处理如何工作* (页 324) 。

未绑定的更 新

在 ABAP/4 中不绑定也 可以更新数 据库。但是 在具体的更 新时必须考 虑 SAP 数据库环境 的特殊方面 。下面两节 列出一些选 项。但一般 还是应该使 用下面说明 的绑定更新 方法：

在对话任务 中的绑定更 新 (页 329)

更新任务中 的绑定更新 (页 332)

后台任务中 的绑定更新 (页 334)

行内更新

可以将更新 语句（INSERT、 UPDATE、 MODIFY、 DELETE） 直接放置在 代码中。这 些是“行内 ”更新（也 就是非绑定 的），无需 使用任何 ABAP/4 绑定技术就 可以执行。 即使没有编 码 COMMIT WORK 语句，下一 屏幕更改的 数据库提交 也将更新提 交给数据库 。

此行内更新 方法只适用 于单个屏幕 事务。通过 多屏幕事务 ，如果错误 发生在以后 的屏幕中， 则不能反转 早期屏幕提 交的数据。
如果使用该 方法，则不 依靠屏幕更 改时执行的 自动提交。 SAP 建议在屏幕 处理之前显 式地使用 COMMIT WORK。

缓冲行内更 新
可能缓冲更 改以便在更 新事务结束 时更新。因 为在提交处 理期间未绑 定执行，因 此这些仍然 是重要的行 内缓冲。但 是，缓冲允 许通过多屏 幕收集更新 ，并且在完 成时将其写 入数据库。
例如，许多 事务需要用 户采取一系 列步骤，在 每个屏幕中 一步，并且 每步涉及分 隔的表格更 新。当整个 顺序成功运 行之后则完 成整个操作 。
要执行简单 行内更新， 可以在给定 屏幕上更新 每个表格。 结果当然是 ：如果在以 后的屏幕中 发生错误， 则以前屏幕 的更新不能 反转。
要执行缓冲 行内更新， 可以在内部 表（或其它 存储）中保 存每个屏幕 的更改，直 到该操作结 束。如果成 功运行，则 使用缓冲信 息可以为每 个表更新。 该技术类似 于 ABAP/4 绑定，为以 后执行要将更新保存在 其中。但是 ，一旦到达 UPDATE 语句（或类 似语句）则 更新在代码 中执行。它 们不需要 COMMIT WORK 语句触发。

优点和缺点
何时应该使 用缓冲行内 更新，何时 应该通过 PERFORM ON COMMIT 绑定？这取 决于应用程 序，可能与 下列几点有 关：

- 行内更新 可能更简单 和自然。
- 因为所有 访问一次进 行，所以 PERFORM ON COMMIT 可以更有效 地使用数据 库资源。
- 到处使用 PERFORM ON COMMIT 保证实施的 一致性。不 可能无意提 交或太早提 交。

另一个重要 的考虑是代 码和应用的 其它部分的 相互依赖性 。如果代码 生成数据库 更改，并且 将要由相同 更新事务中 运行的其它 程序或模块 调用，则需 要清楚地设 计谁提交、 何时提交。
如果调用代 码生成更改 并且发出 COMMIT WORK， 则它可能结 束在尚未准 备提交的调 用者中生成 的更改。另 一方面，如 果调用代码 生成更改并 且不发出 COMMIT WORK， 则调用程序 也可能忽略 提交。在这 种情况下， 可能丢失生 成于调用代 码中的更改 。
一个好的通 用规则是决 定应该在何 处进行应用 提交。如果 希望的话， 主要或调用 代码可以提 供使用缓冲 行内更新。 另一方面， 如果生成更 新但将在提 交时放弃控 制，则调用 代码应该使 用 PERFORM ON COMMIT 计划绑定更 新。

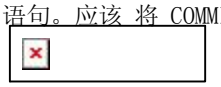
在对话任务 中的绑定更 新

对话任务中 执行的更新 总是同步的 。对话任务 更新可以绑 定或不绑定 。不绑定的 更新在 *未绑定的更 新* (页 439) 中说明。

使用 PERFORM ON COMMIT
可以为对话 任务处理绑 定更新，并 且为该事务 在最后的 LUW 中执行它们 。使用 PERFORM ON COMMIT 命令进行操 作。为此， 请：

1. 将所 有更新语句 （INSERT、 UPDATE、 MODIFY、 DELETE， ...）放 在 FORM 例程中。
2. 通过 PERFORM <form> ON COMMIT 调用例程。

使用 PERFORM ON COMMIT 替代简单的 PERFORM 的作用，是 系统延迟该 表单例程的 执行，直到 遇到 COMMIT WORK 语句。应该 将 COMMIT WORK 放在事务结 尾，或者用 户可以选择 “保存”功 能的地方。



不要在表单 例程中使用 COMMIT WORK 或 ROLLBACK WORK。

给表单例程 分配运行优 先级
通过将 LEVEL 参数添加到 PERFORM ON COMMIT 语句，可以 将运行优先 级分配给每 个 FORM 例程。例如 ：

```
PERFORM update_table1 ON COMMIT LEVEL 2.
...
PERFORM update_table2 ON COMMIT LEVEL 3.
...
PERFORM update_table3 ON COMMIT LEVEL 1.
```

考虑到提交 期间其它例 程也在运行 ，LEVEL 优先级指定 每个例程何 时运行。级 别数越低， 例程的优先 级越高（即 越早执行） 。在示例中 ，到达 COMMIT WORK 语句时，FORM 例程以下列 次序运行：

```
First:        update_table3
Second:   update_table1
Third:        update_table2
```

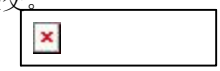
联合同步和 异步执行
通过 PERFORM ON COMMIT 调用的 FORM 例程可能包 含下列语句 ：

```
INSERT, UPDATE, MODIFY, DELETE...
CALL FUNCTION
```

```
CALL FUNCTION IN UPDATE TASK
CALL FUNCTION IN BACKGROUND TASK DESTINATION
```
在事务最后 的 LUW 中，所有四 条语句将在 对话任务中 执行。在前 两条语句涉 及的更新：
```
INSERT, UPDATE, MODIFY, DELETE...
CALL FUNCTION "Updates inside the function module
```
将严格地同 步运行。这 些更新由 COMMIT WORK 触发的数据 库提交提交 执行。但是 后两条语句 ：
```
CALL FUNCTION IN UPDATE TASK
CALL FUNCTION IN BACKGROUND TASK DESTINATION
```
自己计划功 能模块以便 在其它工作 进程中运行 。当这些功 能包含更新 语句时，就 进行同步更 新。请求的 工作进程（ 更新或后台 任务）在已 经进行的提 交处理结束 时触发。（ 即不需要编 码其它 COMMIT WORK 语句以触发 它们。）
后两个调用 的主要原因 是因为要该 功能带当前 COMMIT WORK 语句的参数 值运行。详 细信息，参 见 *将更新任务 调用添加到 子程序中* (页 331) 。

更新任务中 的绑定更新

SAP 系统提供专 用于执行数 据库更新的 更新任务。 可以将所有 更新放在一 个功能模块 中，并且使 用语句 CALL FUNCTION IN UPDATE TASK 在该更新任 务中执行此 更新。
对于该条语 句，系统将 请求（功能 模块和参数 信息）写入 日志表中。 在下一个 SAP 提交中，系 统将请求传 递到更新任 务以便执行 。可以通过 COMMIT WORK（ 异步）或 COMMIT WORK AND WAIT（ 同步）发出 SAP 提交。



记住更新任 务由 SAP 提交在调用 代码中触发 。不要在更 新任务中使 用 COMMIT WORK。

下列主题提 供编写和使 用更新任务 功能模块的 指南：
更新任务处 理如何工作 (页 324)
创建更新任务功能模块 (页 420)
调用更新任 务功能模块 (页 343)
特别 LUW 考虑 (页 342)
访问更新任 务日志 (页 340)

更新任务处 理如何工作

要有效地编 程异步更新 ，必须了解 R/3 系统中更新 任务如何工 作。典型的 SAP 系统配置包 括对话任务 和更新任务 。对话任务 处理所有与 其它联机用 户的交互会 话。更新任 务专用于执 行数据库更 新。可以在对话 任务和更新 任务中执行 更新。对话 任务更新是 同步更新。 更新任务更 新是异步更 新。（例外 ：用 COMMIT WORK AND WAIT 触发更新任 务功能时， 它是同步更 新。）
在事务内部 ，对于每条 COMMIT WORK 语句可以调 用一个或多 个更新任务 功能模块。

请求更新任 务处理
要请求更新 任务处理， 应在功能模 块中编程全 部更新，并 且通过语句 CALL FUNCTION...IN UPDATE TASK 调用它。在 更新任务中 调用的功能 模块必须是 更新任务功 能模块。即 必须为功能 设置更新任务属性。这 些属性指定 功能的优先 级，并且说 明是在公共 *更新事务* (页 436) (SAP LUW) 中运行，还是自身运行 。
（详细信息 ，参见 *创建更新任 务功能模块* (页 420)
)。

异步处理中 的计时
为更新任务 功能模块选 择的优先级 决定它通过 相同的 COMMIT WORK 触发如何相 对于其它功 能运行。高 优先级功能 （“V1” ）全部都在 公共更新事 务中运行。 低优先级功 能（“V2” ）总在单个 更新事务中 运行。
V1 和 V2 的区别在于 时间关键和 非时间关键 数据库更改 。一旦可能 必须处理时 间关键更改 （在 V1 功能中）， 并且更新反 映实时的情 况（例如， 航班预定和 仓库移动） 。无须立即 进行非关键 更改（在 V2 功能中）， （例如，显 示利润率分 析的周期性 统计更新） 。
系统在处理 V2 功能前为单 个 COMMIT WORK 处理所有 V1 功能。如果 V1 功能无错误 运行，则系 统释放所有 相关 ENQUEUE 锁定，并且 从日志表中 删除该功能 。最后，系 统开始在日 志表中处理 V2 功能。为单个 COMMIT WORK 请求的 V1 功能模块总 是以调用顺 序处理。V2 功能模块可 以按任何顺 序处理，甚 至平行于不 同的更新任 务。但是， V2 功能模块不 应该只依靠 为其它 V2 模块创建的 ENQUEUE 锁定。
为 COMMIT WORK 请求的 V1 功能模块也 可以平行于 不同的更新 进程处理。

异步错误处 理
与发生在对 话任务中的 错误不同， 更新任务中 的错误不能 由用户校正 。相反，系 统停止处理 当前的更新 功能，并且 反转任何在 该模块中已 经进行的数 据库更改。 取决于是否 为相同*更新事务*（页 436）（SAP LUW）的 一部分，也 反转在以前 功能模块中 所作的其它 更改。

由相同 COMMIT WORK 触发的所有 V1 功能视为单 个更新事务 的一部分。
因此，在一 个 V1 功能模块中 的错误导致 为该 COMMIT WORK 记录的其它 V1 功能的所有 更新的反转 。所有功能 模块依旧在 日志表中， 并且错误模 块接收到标 记 ERR。
给定 COMMIT WORK 的每个 V2 功能模块总 在自己（分 开）的更新 事务中运行 。
在一个 V2 功能模块中 的错误不导 致其它 V2 模块的反转 。只反转错 误功能的更 改,并且功 能留在日志 表中标记为 错误。执行 其它 V2 功能。

关于运行时 错误处理的 详细信息， 参见 *绑定更新的 错误处理*（页 373）。
系统将反转 功能模块标 记为更新任 务日志中的 错误功能。 然后可校正 该错误并稍 后重新启动 该功能。要 访问更新任 务日志，请 选定“工具 -> 管理 -> 监控 -> 更新”。
关于该事务 的详细信息 ，参见 *访问更新任 务日志*（页 340）。

创建更新任 务功能模块
编写更新任 务功能模块 与编写其它 功能模块基 本相似。但 是，这里提 供一些特别 的考虑。
要创建功能 模块，必须 首先进入函 数库。(选 定“工具 -> ABAP/4 工作台”， 然后按“函 数库”按钮)。
关于创 建和编写功 能模块的详 细信息,参 见 *ABAP/4 开发工作台工具*（页 **Error! Not a valid link.**）。

注册更新任 务功能模块
在更新任务 中运行的功 能模块必须 在函数库中 注册。创建 功能模块时 ，将“进程 类型”属性 设置为下列 之一：

立即开始 更新
为在共享*更新事务*（页 436）（SAP LUW）中 运行的高优 先级（“V1” ）功能设置 该选项。出 现错误时更 新任务可以 重新启动这 些功能。
立即开始 更新，不能 重启
为在共享更 新事务中运 行的高优先 级（“V1” ）功能设置 该选项。不 能由更新任 务重新启动 这些功能。
延迟启动 更新
为在自身更 新事务中运 行的低优先 级（“V2” ）功能设置 该选项。出 现错误时更 新任务可以 重新启动这 些功能。

要访问函数 库内的属性 值屏幕，请 选定“转向 -> 管理” 。

定义接口
在更新任务 中运行的功 能模块具有 有限的接口 ：

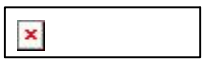
因为更新 任务功能模 块不能报告 其结果，因 此不允许结 果参数或例 外。
必须指定 输入参数和 带有 ABAP/4 词典中所定 义的参考字 段和参考结 构的表格。

调用更新任 务功能模块

同步或异步 处理?
在更新任务 中运行的功 能模块可以 同步或异步 运行。所使 用的提交语 句的格式决 定了其运行 的具体方式 ：

COMMIT WORK
这是执行异 步处理的标 准格式。程 序并不等待 所请求的功 能完成处理 。
COMMIT WORK AND WAIT
该格式指定 同步处理。 该提交语句 等待所请求 的功能完成 处理。在高 优先级（V1） 功能模块成 功运行之后 控制权返回 给程序。
AND WAIT 格式适合将 旧程序切换 到同步处理 而不必重新 编写代码。 将 AND WAIT 用于更新任务更新与使 用 PERFORM ON COMMIT 的对话任务 更新在功能 方面正好相 同。

执行的参数 值
在 ABAP/4 中，可以以 两种不同的 方法调用更 新任务功能 模块。所选 定的方法决 定了实际执 行功能模块 时所应使用 的参数值。 可以在 CALL FUNCTION 语句时设置 参数，也可 以在 COMMIT WORK 语句时设置 参数。下列 章节将做详 细说明。



这些章节中 的示例显示 使用 COMMIT WORK 的异步提交 。

直接调用更 新功能
要直接调用 功能模块， 请在代码中 直接使用 CALL FUNCTION IN UPDATE TASK。

```
CALL FUNCTION 'FUNCTMOD' IN UPDATE TASK EXPORTING...
```
然后系统登 录请求，并 且在到达下 一 CIMMIT WORK 语句时执行 此功能模块 。执行功能 模块所使用 的参数值是 调用时的当 前值。例如 ：
```
a = 1.
CALL FUNCTION 'UPD_FM' IN UPDATE TASK EXPORTING PAR = A...
a = 2.
CALL FUNCTION 'UPD_FM' IN UPDATE TASK EXPORTING PAR = A...
a = 3.
COMMIT WORK.
```
此处在更新 任务中功能 模块 UPD_FM 执行两次： 第一次 PAR 中的值为 1 ；第二次 PAR 中的值为 2。

将更新任务 调用添加到 子程序中

也可以将 CALL FUNCTION IN UPDATE TASK 放入子程序 ，并且用下 面的方法调 用该子程序 ：
```
PERFORM SUBROUT ON COMMIT.
```
如果选择这 种方法，则 在提交时执 行子程序。 因而在提交 处理期间也 登录要在更 新任务中运 行功能的请 求。所以， 该请求登录 的参数值是 提交时的当 前值。例如 ：
```
a = 1.
PERFORM F ON COMMIT.
a = 2.
PERFORM F ON COMMIT.
a = 3.
COMMIT WORK.

FORM f.
  CALL FUNCTION 'UPD_FM' IN UPDATE TASK EXPORTING PAR = A.
ENDFORM.
```
在该示例中 ，以 PAR 中的值为 3 来执行功能 模块 UPD_FM。 即使有两条 PERFORM ON COMMIT 语句，更新 任务也只执 行一次功能 模块。这是 因为用相同 参数值登录 的给定功能 模块不能在 更新任务中 执行多次。 包含功能模 块调用的子 程序本身不 能有参数。

这里说明的 方法不适合 在对话模块 代码中使用 。但是，如 果确实需要 在对话模块中使用这种 方法，参见 *调用更新任 务功能的对 话模块* (页 315) 。

特别 LUW 考虑

在更新任务 队列中，系 统通过分配 公共更新关 键字以识别 属于同一*更新事务* (页 436) (SAP LUW) 的 所有功能模 块。在下一 COMMIT WORK，更新任务读 取该队列， 并且处理带 预定义更新 关键字的所有请求。

如果程序调 用更新任务 功能模块， 则向执行模 块（或调用 它的子程序 ）的请求提 供当前 LUW 的更新关键 字并将其放 入队列。

当在由其它 程序调用的 模块（事务 或对话模块 ）中嵌入更 新任务功能 调用时，将 发生什么？ 下列章节将 详细说明。

调用更新任 务功能的事 务

如果程序调 用本身就调 用更新任务 功能模块的 事务（或报 表），则应 该注意下列 事项：

每个被调用 的程序或报 表都启动一 新的*更新事务* (页 436) (SAP LUW)， 并赋予该事 务一新的更 新关键字。 在调用事务 （或报表） 期间，该关 键字用于识 别所请求的 所有更新任 务操作。

从事务（或 报表）返回 时，调用程 序的 LUW 同时恢复为 旧的更新关 键字。

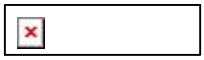
如果所调用 的事务（或 报表）不包 含其自身的 COMMIT WORK， 则既不执行 所请求的数 据库更改， 也不执行对 更新任务功 能模块的调 用。例如， 在下列代码 中，**F1**、 **F2** 和 **F3** 是更新任务 功能模块：

```
CALL FUNCTION 'F1' IN UPDATE TASK.
CALL FUNCTION 'F2' IN UPDATE TASK.
CALL FUNCTION 'F3' IN UPDATE TASK.
COMMIT WORK.
```

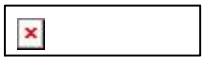
由于主程序 中的 COMMIT WORK 触发 **F1** 和 **F3** 的执行，因 此此处在更 新任务中 **F1** 和 **F3** 被执行。然 而，事务 ZABC 中并不包含 COMMIT WORK 语句，因此 更新任务不 执行功能 **F2**。

调用更新任 务功能的对 话模块

与事务和报 表不同，对 话模块并不 启动新的*更新事务* (页 436) 。 从对话模块 中调用更新 任务功能模 块就使用与 调用程序中 相同的更新 关键字。结 果是只要调 用程序中出 现 COMMIT WORK 语句，就执 行从对话模 块对更新任 务功能模块 的调用。



如果在对话 模块中放置 COMMIT WORK， 则不提交对 数据库（例 如，通过 UPDATE） 的更改。但 是，它不启 动更新任务 。如果对话 模块调用更 新任务功能 模块，则只 有遇到调用 程序中的 COMMIT WORK 之时才触发 此功能模块 。



如果正在使 用对话模块 ，那么请尽 量避免将更 新任务功能 模块的调用 包括在用 PERFORM ON COMMIT 调用的子程 序中。一般 地，对话模 块中只要存 在 PERFORM ON COMMIT （无论是否 有更新功能 的调用）， 就可能有问 题。

原因是对话 模块在其自 身滚动区域 中运行，而 当模块完成 之后该滚动 区域就消失了。这意味 着一旦到达 主程序中的 提交,则所 有本地数据（包括调用 更新任务功 能模块时作 为参数使用 的数据）将 会消失。

如果必须在 对话模块中 使用这种方 法（即在子 程序中包括 对更新任务 功能的调用 ），则必须 保证实际参 数值在更新 任务功能实 际运行时仍 然存在。为 此，可以通 过 EXPORT TO MEMORY 存储所需的 值，并且在 COMMIT WORK 语句前将其 输入到主程 序中（IMPORT FROM MEMORY） 。

访问更新任 务日志

要访问更新 日志，请选 定“工具 -> 管理 -> 监控 -> 更新”。在 主屏幕中，

1. 在“ 用户”字段 中输入用户 名称。
2. 单击 用于要查看 的更新记录 的正确“状 态”选项。
3. 按“ 回车”。

日志中的更 新记录或者 还没有写入 到数据库中 （状态=将 执行），或 者以错误终 止（状态=终 止）。出现错误时 ，可以在日 志中选定重 新运行，然 后选择“管 理 -> 调试”以便 在调试模式 下重新运行该更新。

后台任务中 的绑定更新

可以在远程 主机上执行 更新。如果 在两个数据 库中维护重 复的数据， 此功能可能 是所希望的 。在本地系 统的更新任 务中作主更 新时，也将 在远程数据 库中执行相 同更新。系 统使用 RFC（远 程函数调用 ，协议）将 请求发送到 远程系统。

要在远程主 机上执行更 新，请使用 语句：

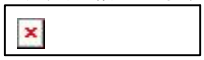
```
CALL FUNCTION <fctmod> IN BACKGROUND TASK DESTINATION <dest>...
```

该语句请求 在外部系统 中执行更新 功能。DESTINATION 参数指定远 程系统。一 旦遇到该语 句，系统将 请求写入日 志表，然后 在下一 COMMIT WORK 时执行此请 求（即，将 其转发到远 程系统）。

也可以使用 不带参数 DESTINATION 的 CALL FUNCTION IN BACKGROUND TASK 语句。在这 种情况下， 该功能在本 地主机上的 独立工作进 程中运行。 但是，以这 种方法使用 该语句没有 好处，因为 它和更新任 务中运行功 能的结果相 同。

通过相同 COMMIT WORK 触发的所有 远程主机后 台任务功能 都在公共 LUW 中运行。指 定 DESTINATION 参数时，为 相同目标系 统请求的所 有功能（由 相同 COMMIT WORK 触发）在公 共 LUW 中运行。

关于 RFC 编程的详细 信息，参见*远程通讯* （页 **Error! Not a valid link.**） 。



对于远程系 统的更新， 后台任务处 理只执行单 阶段提交。 不执行双阶 段提交。

COMMIT WORK 处理

COMMIT WORK 语句执行许 多与任务的 同步执行有 关的功能。 这些任务通 常涉及数据 库更新，但 却并不需要 数据库的更 新。分布式 处理环境中 需要的许多 操作依赖于 同步功能的 执行。

要执行更新 ，COMMIT WORK 终止*更新事务* （页 436），并且提交 数据库更改 。因为 COMMIT WORK 关闭事务， 所以通常用 于执行“保 存”功能。

一般地，COMMIT WORK 语句进行下 列处理：

执行所有 由 PERFORM ON COMMIT 请求的 FORM 例程。
按优先级的 升序执行这 些例程，顺 序由 PERFORM 语句的 LEVEL 参数指定。 关于该参数 的详细信息 ，参见 *在对话任务 中的绑定更 新* （页 440） 。
如果请求 ，触发所有 更新任务功 能模块。
如果请求 ，触发所有 后台任务功 能模块。
触发数据 库提交（依 次释放数据 库锁定）。
清空反转 日志。

反转日志包 含应用更改 前的表格快 照。执行反 转时，该快 照用来将表 格复位到原 值。
关闭所有 打开的数据 库光标。
将所有 TEMSE 对象写入永 久性文件或 数据库。
TEMSE 文件是由于 性能原因在 执行事务期 间缓存的临 时序列文件 。TEMSE 文件的示例 有假脱机对 象或作业日 志。
将时间片 计数器（用 于访问工作 进程）复位 到 0。
系统中的时 间片计数器 限制工作进 程中程序运 行的时间量 。如果程序 常常超过时 间片限制，则可以使用 COMMIT WORK 为程序获得 更多时间。
但是，要达 此目的，必 须很容易将 处理分为更 小单元（全 部或没有操 作）。然后 可以在每个 单元之后插 入 COMMIT WORK 语句。当然 处理单元逻 辑上必须独 立，因为在 发生错误时 ，不能取消 前面的单元 中所提交的 更新。

不要在用 PERFORM ON COMMIT 调用的任何 FORM 例程中或者 在更新任务 中运行的功 能模块中使 用 COMMIT WORK。 这样做将导 致运行时间 错误。相反 ，可以在后 台任务功能 模块中使用 COMMIT WORK。 但是，应该 保证提交（ 和反转）正 常工作。

ROLLBACK WORK 处理

ROLLBACK WORK 语句“取消 ”任务同步 执行的所有 请求。特别 是，当任务 涉及数据库 更新时，ROLLBACK WORK“ 抛弃”当前 事务的所有 更新：
抛弃所有 以前用 PERFORM ON COMMIT 登录的 FORM 例程
在更新任 务队列中将 所有以前请 求的更新任 务功能标记 为错误
抛弃所有 以前请求的 后台任务功 能
从缓冲存 储中删除所 有 TEMSE 对象（临时 连续文件， 如假脱机对 象和作业日 志）
触发数据 库反转操作 （依次释放 所有数据库 锁定）
关闭所有 打开的数据 库光标

注意，无论 发生下列哪 种情况，都 自动执行所 有上述操作 ：
运行时间 错误
程序发出 "A" MESSAGE（ 异常终止） 。
在这种情况 下，程序终 止。不同之 处在于，ROLLBACK WORK 语句使程序 在错误之后 继续处理。 例如，无论 用户是要取 消操作还是 早点退出事 务，都可以 使用 ROLLBACK WORK。

不要在用 PERFORM ON COMMIT 调用的 FORM 例程中或者 在更新任务 中运行的功 能模块中使 用 ROLLBACK WORK。 这样做将导 致运行时间 错误。相反 ，可以在后 台任务功能 模块中使用 ROLLBACK WORK。 但是，应该 保证反转逻 辑（反转） 正确工作。

后台处理考 虑

SAP 系统提供在 后台运行作 业的工作进 程。在后台 进程中运行 的程序可以 连续运行直 到结束。即 使其它程序 要访问工作 进程也不能 中断它们。 因此，当在 后台进程中 需要屏幕处 理时，系统 不触发联机 执行时发生 的自动数据 库提交。
在后台处理 中，用户不 能直接调用 事务。但是 ，任何报表 程序都可以 使用 CALL TRANSACTION USING 关键字来启 动事务。因 此，许多事 务既可以在 对话任务运 行，也能在 后台运行。
由于为联机 事务所请求 的异步更新 ，当事务在 后台运行时 性能可能降 低，所以这 一点很重要 。要在更新 任务中执行 更新，系统 必须登录请 求及其数据 （在数据库 中），然后 当更新任务 运行时再次获取它。在 后台执行事 务时，这些 登录活动无 效。更新可 以正好在后 台进程中执 行（即同步 ）。

请求本地更 新

在 ABAP/4 中，可以告 知系统要在 本地，而不 是在更新任 务中执行更 新。为此， 只要程序在 背景中运行 ，就请使用 语句 SET UPDATE TASK LOCAL。 通过查询 SY-BATCH 系统变量， 代码可以检 查当前更新 是否正在后 台运行。
SET UPDATE TASK LOCAL 语句为程序 设置“本地 更新”开关 。设置了该 开关之后， 系统将 CALL FUNCTION IN UPDATE TASK 解释为本地 的更新请求 。每个函数 及其数据登 录在 SAP 内存中（而 不是数据库 ），然后作 为 COMMIT WORK 处理的一部 分在本地执 行（在调用 者的相同任 务中）。在 继续之前事 务等待更新 的完成。
如同在正常 的更新任务 中，如果程 序发出任何 消息（类型 S 除外），则 更新处理终 止并且导致 反转。但是 与正常更新 任务不同，如果需要反 转，则事务 的对话和更 新任务部分 反转。（这 是因为本地 更新与事务 的对话部分 共享 LUW。） 不能通过本 地更新使用 V2 功能（延迟 启动）。
（关于 V1 和 V2 功能的详细 信息，参见 *更新任务处 理如何工作* （页 441） 。）

何处使用 SET UPDATE TASK LOCAL
缺省情况下 本地更新开 关为关闭， 并且在每条 COMMIT WORK 或 ROLLBACK WORK 语句之后复 位到关闭状 态。因此在 每个更新事 务的开始处 需要 SET UPDATE TASK LOCAL 语句。(更 新事务和 LUW 定义在 *SAP 系统中的事 务* (页 436) 中。)

```
IF SY-BATCH NE SPACE.
    SET UPDATE TASK LOCAL.
ENDIF.
...
CALL FUNCTION UPDATE_TABLE1 IN UPDATE TASK.
...
CALL FUNCTION UPDATE_TABLE2 IN UPDATE TASK.
...
COMMIT WORK.
```

如果调用任 何外部事务 或提交报表 ，这些程序 中也应该有 SET UPDATE TASK LOCAL 语句（因为 它们有自己 的更新事务)。但是对 话模块不需 要 SET 语句。
SET 语句必须在 事务中的 CALL FUNCTION IN UPDATE TASK 语句之前执 行。如果在 SET 语句到达前 运行了 CALL FUNCTION ，则 SET 失败（返回 SY-SUBRC = 1)。在这 种情况下， 事务的所有 功能调用（ 在 SET 前或后）都 将作为正常 更新任务请 求予以对待 。

检查更新结 果
通过正常异 步处理，可 以查找更新 任务队列以 检查结果。 出现错误时 ，请求保留 在队列中并 标记为错误 。
如果使用本 地更新，则 保存请求的 SAP 内存到达事 务结尾。因 此，不能在 此检查更新 如何运行。 但是，可以 在批日志中 检查后台处 理步骤的记 录。要访问 批日志，请 在任何屏幕 中选定“系 统 -> 服务 -> 作业 -> 作业概述” 。
注意：如果 未在后台运 行时使用本 地更新，则 系统不能提 供结果报表 。也不能获 取更新是否 成功的反馈 信息。

绑定更新的 错误处理

在执行绑定 更新期间发 生了运行时 间错误。如 何处理它们 ？通常，按 下列顺序执 行 COMMIT WORK 处理：

1. 执行 所有通过 PERFORM ON COMMIT 登录的对话 任务 FORM 例程。
2. 执行 所有高优先 级（V1） 更新任务功 能模块。
   V1 更新处理的 结尾就标记 *着更新事务* (页 436) 的结束。如 果使用 COMMIT WORK AND WAIT 触发提交处 理，那么， 控制权就返 回给对话任 务程序。
3. 触发 所有低优先 级（V2） 更新任务功 能模块。
   触 发所有后台 任务功能模 块。

运行时间错 误的发生可 能在系统中 ，也可能因 为程序发出 异常终止消 息(消息类 型‘A’) 。ROLLBACK WORK 语句也自动 发出运行时 间错误的信 号。系统根 据错误发生 的位置来处 理错误：

在 FORM 例程中（通 过 PERFORM ON COMMIT 调用）
- 反转已 经为当前更 新事务执行 的更新。
- 不启动 其它 FORM 例程。
- 不启动 进一步更新 任务或后台 任务功能。
- 屏幕上 出现错误消 息。

在 V1 更新任务功 能模块中（ 在更新任务 中请求）
- 反转已 经为 V1 功能执行的 更新。
- 丢弃所 有进一步的 更新任务请 求（V1 或 V2）。
- 丢弃所 有后台任务 请求。
- 不反转 已经为通过 PERFORM ON COMMIT 调用的 FORM 例程执行的 更新。
- 如果已 将系统设置 为发送错误 消息，则消 息出现在屏 幕上。

在 V2 更新任务功 能模块中（ 在更新任务 中请求）
- 反转已 经为当前 V2 功能执行的 更新。
- 执行所 有要执行的 更新任务请 求（V2） 。
- 执行所 有要执行的 后台任务请 求。
- 不反转 以前执行的 V1 或 V2 功能的更新 。
- 不反转 以前为（通 过 PERFORM ON COMMIT 调用的）FORM 例程执行的 更新。
- 如果已 将系统设置 为发送错误 消息，则消 息出现在屏 幕上。

在后台任 务功能模块 中（在后台 任务目的地 中请求）
- 反转已 经为当前 DESTINATION 执行的后台 任务更新。
- 丢弃所 有相同 DESTINATION 的进一步后 台任务请求 （V1 或 V2）。
- 不反转 其它以前执 行的更新。
- 不在屏 幕上出现错 误消息。

如果程序检 测到远程处 理发生错误 ，则它可以 决定以后是 否重新提交 此请求。
关于 RFC 编程的详细 信息，参见*远程通讯* (页 **Error! Not a valid link.**) 。

SAP 系统中的锁 定

SAP 锁定机制允 许程序员在 整个*更新事务* (页 436) 中设置和释 放锁定。该 机制包括两 个基本步骤 ：

1. 在 ABAP/4 词典中定义 锁定对象

锁定对象是 参照于一个 或多个表的 逻辑锁定。 激活这些锁 定对象将导 致系统为锁 定和解锁相 关表行生成 特殊功能模 块。这些功 能模块称为 ENQUEUE-<object> 和 DEQUEUE-<object>。
参见 *定义锁定对 象* (页 344) 。

2. 在程 序中调用 ENQUEUE-<lock-object> 和 DEQUEUE-<lock-object>
要使用 SAP 锁定，在访 问数据库对 象的前后程 序须调用 ENQUEUE-<lock-object> 和 DEQUEUE-<lock-object>。
参见 *调用 ENQUEUE/DEQUEUE 功能模块* (页 344) 。

定义锁定对 象

锁定对象表 示要锁定的 数据库对象 。数据库对 象可以是特 定表格或表 格集。
定义锁定对 象时，指定 相关表格以 及锁定参数 。锁定参数 包括指定表 格的所有主 码字段。例 如，在运行时间 中（通过 ENQUEUE-<lock-object>） 锁定对象时 ，将锁定参 数字段用作 输入参数。 通过指定参 数字段的值 ，可以锁定 单个或一类 表格行。指 定锁定参数 值的 WHERE 条件可以只 包含 AND 条件（没有 OR 条件）。

设置锁定模 式

锁定对象也 指定表格可 能的锁定种 类：

共享锁定
该模式允许 多个用户访 问指定表行 ，但只能读 访问。任何 时候都不允 许写访问。
排他锁定
该模式允许 单个用户对 指定表行进 行读和写访 问。其它用 户不能访问 该行。
扩展排他 锁定
该模式避免 具有读写访 问权限的单 个用户获得 对相同表行 集的进一步 锁定。当使 用递归例程 更新时，该 模式很有用 。

在数据词典 中定义该锁 定对象，然 后激活它。 关于这些任 务的详细信 息，参见 *ABAP/4 词典* (页 **Error! Not a valid link.**) 。

调用 ENQUEUE/DEQUEUE 功能模块

激活锁定对 象导致系统 生成用于锁 定和解锁对 象的特殊功 能模块。这 些功能模块 称为：
ENQUEUE_<lock-object-name> ’用于锁定 对象 DEQUEUE_<lock-object-name> ’用于解锁 对象
运行时，在 试图读或写 之前可以锁 定该数据库 对象。要锁 定对象，请 在第一屏幕 的 PAI 事件中调用 功能模块 ENQUEUE_<lock-object-name>。 要解锁此对 象，请调用 DEQUEUE_<lock-object-name>。

ENQUEUE/DEQUEUE 参数

ENQUEUE/DEQUEUE 功能模块有 下列参数集 ：

arg 和 *x_arg* （ ENQUEUE 和 DEQUEUE）
这两个 EXPORTING 参数，存在 于锁定参数 的每个字段 *arg* 中。将 *arg* 设置为所需 的关键字字段值。如果 该字段不需 要特殊值， 则忽略 *arg* 参数，或者 将其设置为 字段的初始 值。如果要 将字段的初 始值作为实 际选择值， 请将 *x_arg* 设置为‘X’ 。

_SCOPE (ENQUEUE)
如果事务不 调用更新任 务功能，则 只在对话任 务中更新。 应该使用相 应的 DEQUEUE 功能直接释 放锁定。
如果调用任 意的 V1 更新任务功 能，请设置 参数 _SCOPE 以告知系统 应该如何释 放 SAP 锁定。可能 的值为：

1 该值用 于创建更新 任务中不需 要的锁定。 在整个对话 任务处理中 保持使用 _SCOPE=1 设置的锁定 ，但该锁定 并不能用于 任意的更新 任务请求。 要保证不将 锁定保持超 过必要的时 间，在事务 结束时应该 直接释放它 （通过相应 的DEQUEUE 功能）。
在ROLLBACK WORK 时释放：系 统不释放使 用 _SCOPE=1 设置的锁定 。在编制反 转程序时，请使用 DEQUEUE 功能。

2 该值用 于创建下列 锁定：
- 在更新 任务触发之 前在对话任 务中使用
- 一旦已 经触发了更 新任务，则 只在更新任 务中使用。

这意味着在 COMMIT WORK 已经触发更 新任务之后 ，如果该任 务继续运行 ，则锁定不 再可用于对 话任务事务 。系统在 V1 更新任务处 理结束之后 （或者在下 一 ROLLBACK WORK） 释放锁定。
如果不指定 _SCOPE，该值就是缺 省值。不需 要将DEQUEUE 用于 使用 _SCOPE 值创建的锁 定。但是， 如果 _SCOPE = 2，并且不 调用更新任 务功能，则 不触发更新 任务而且系 统不释放锁 定。

在 ROLLBACK WORK 释放：如果 在提交前发 生反转，则 系统释放使 用 _SCOPE=2 设置的锁定 。在提交之 后 ，锁定保持 到更新任务 处理结束。

3 使用该 值创建以下 锁定：
- 在触发 更新任务之 前在对话任 务中使用
- 在触发 更新任务之 后，由对话 任务和更新 任务使用。

也就是说， 对话任务程 序继续使用 该锁定，甚 至该更新任 务功能正在 运行时也是 如此。在这 种情况下， 锁定由对话 事务和更新 任务功能所 “共有”。
在 V1 更新任务处 理之后的某 个时刻系统 释放锁定。但是，应该 直接释放锁 定（通过相 应的 DEQUEUE 函数）以确 保尽可能早 地释放。

在 ROLLBACK WORK 释放：一旦 已经触发更 新任务，则 使用 _SCOPE=3 设置的锁定 有两个独立 的所有者。要删除锁定 ,必须在对 话任务和更 新任务两方 同时删除此 锁定。如果 在提交前发 生反转，则 在更新任务 方释放锁定 ，但是对话 任务方仍然 保留它。如 果在提交后 发生反转， 则系统在双 方都不释放 该锁定。在 这种情况下 ，必须使用 DEQUEUE 功能在对话 方直接释放 该锁定。更 新任务方将 自动释放锁 定。

如果用户在 事务完成前 已经退出了 该事务（例 如通过“/n” ）或者程序 异常终止， 则系统释放 所有锁定。 V2 功能也不能 继承对话任 务所创建的 锁定。

_WAIT （ 只 ENQUEUE）

此 EXPORTING 参数告知： 如果即将锁 定的对象已 经由其它用 户锁定，ENQUEUE 是否应该等 待。如果要 等待则将 _WAIT 设置为 'X'。在 这种情况下 ，系统试图 以指定时间 长度的重复 间隔锁定该 对象。如果 这些尝试失 败，则 ENQUEUE 导致 FOREIGN_LOCK 例外。

如果程序不 想等待，则 将 _WAIT 设置为其它 任何值。在 这种情况下 ，ENQUEUE 导致 FOREIGN_LOCK 例外，并且 将系统字段 SY-MSGV1 设置为已经 拥有该锁定 的用户名。

ENQUEUE 例外

在调用 ENQUEUE 功能模块之 后，应该检 查在程序中 该对象是否 已经锁定。 在功能模块 中定义了下 列例外：

FOREIGN_LOCK

其它用户已 经锁定了对 象。系统字 段 SY_MSGV1 包含该用户 名。

SYSTEM_FAILURE

一般系统错 误。

示例事务： SAP 锁定

关于如何使 用锁定对象 锁定表的示 范，参见事 务 TZ90。 该事务允许 用户请求给 定的航班（ 在屏幕 100 中），并可 显示和更新 该航班（在 屏幕 200 中）。更新 航班的请求 锁定该表， 而显示的请 求则不锁定 该表。

屏幕流逻辑

事务 TZ90 中屏幕 200 的屏幕流逻 辑（只有 PAI）如 下：

```
*---------------------------------------------------------------------*
*   Screen 200: Flow Logic                                            *
*&---------------------------------------------------------------------*
PROCESS BEFORE OUTPUT.
     MODULE STATUS_0200.
     MODULE MODIFY_SCREEN.
*
PROCESS AFTER INPUT.
     MODULE EXIT_0200 AT EXIT-COMMAND
     '<...Relevant field checks...>
     MODULE USER_COMMAND_0200.
```

ABAP/4 代码

屏幕 100 的主 PAI 处理允许用 户输入并设 置以执行所 请求的操作 （“更改” 或“显示” ）。如果请 求“更改” ，则程序调 用相关的 ENQUEU 功能以锁定 相关数据库 对象。

```
*---------------------------------------------------------------------*
*      Module  USER_COMMAND_0100  INPUT                               *
*---------------------------------------------------------------------*
MODULE USER_COMMAND_0100 INPUT.
  CASE OK_CODE.
    WHEN 'SHOW'....
    WHEN 'CHNG'.
*     <...Authority-check and other code...>
      CALL FUNCTION 'ENQUEUE_ESFLIGHT'
           EXPORTING
                MANDT          = SY-MANDT
                CARRID         = SPFLI-CARRID
                CONNID         = SPFLI-CONNID
           EXCEPTIONS
                FOREIGN_LOCK   = 1
                SYSTEM_FAILURE = 2
                OTHERS         = 3.
      IF SY-SUBRC NE 0.
         MESSAGE ID     SY-MSGID
                 TYPE   'E'
                 NUMBER SY-MSGNO
                 WITH   SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
      ENDIF.
```

```
      MODE = CON_CHANGE.
      CLEAR OK_CODE.
      SET SCREEN 200.
      OLD_SPFLI = SPFLI.
  ENDCASE.
ENDMODULE.
```

只要用户试 图退出“更 改”操作，屏幕 200 的 PAI 处理就再次 将对象解锁 。程序使用 模块 EXIT_0200 以捕获退出 功能代码 (EXIT、 BACK、 CANC)。 对于“更改 ”，只有一 个相关的退 出功能 (CANC)， 并且它调用 用于释放航 班锁定的 FORM 例程。

```
*&---------------------------------------------------------------------*
*&      Module  EXIT  INPUT
*&---------------------------------------------------------------------*
MODULE EXIT_0200 INPUT.
  IF MODE = CON_CHANGE.
     CASE OK_CODE.
        WHEN 'CANC'.
           CLEAR OK_CODE.
           PERFORM UNLOCK_FLIGHT.
           LEAVE TO SCREEN 100.
     ENDCASE.
  ELSE.    "mode = con_show
     CASE OK_CODE.
       WHEN 'CANC'...
       WHEN 'EXIT'...
       WHEN 'BACK'...
       ENDCASE.
     CLEAR OK_CODE.
  ENDIF.
ENDMODULE.
```

FORM 例程 UNLOCK_FLIGHT 为给定的锁 定对象调用 DEQUEUE 功能。

```
*&---------------------------------------------------------------------*
*&      Form  UNLOCK_FLIGHT
*&---------------------------------------------------------------------*
FORM UNLOCK_FLIGHT.
  CALL FUNCTION 'DEQUEUE_ESFLIGHT'
       EXPORTING
            MANDT     = SY-MANDT
            CARRID    = SPFLI-CARRID
            CONNID    = SPFLI-CONNID
       EXCEPTIONS
            OTHERS    = 1.
  SET SCREEN 100.
ENDFORM.
```

屏幕 200 的其它退出 功能（BACK 和 EXIT） 由屏幕 200 的主 PAI 模块处理。 这允许程序 检查用户是否未保存数 据就退出。 如果这样， 则 PROMPT_AND_SAVE 例程提醒用 户并在必要 时执行保存 操作。然后 ，调用 UNLOCK_FLIGHT 就可以对该 航班解锁。

```
*---------------------------------------------------------------------*
*      Module  USER_COMMAND_0200  INPUT
*---------------------------------------------------------------------*
MODULE USER_COMMAND_0200 INPUT.
  CASE OK_CODE.
    WHEN 'SAVE'....
    WHEN 'EXIT'.
      CLEAR OK_CODE.
      IF OLD_SPFLI NE SPFLI.
         PERFORM PROMPT_AND_SAVE.
      ENDIF.
      PERFORM UNLOCK_FLIGHT.
      LEAVE TO SCREEN 0.
    WHEN 'BACK'.
      CLEAR OK_CODE.
      IF OLD_SPFLI NE SPFLI.
        PERFORM PROMPT_AND_SAVE.
      ENDIF.
      PERFORM UNLOCK_FLIGHT.
      LEAVE TO SCREEN 100.
```

```
    ENDCASE.
ENDMODULE.
```

第十章 编写字段 - 和值 - 帮助程序

概览

F1 或 为用户提供 当前光标所 在字段的帮 助文本。F4 或组合框 （ 如果可用的 话）提供可 在该字段中 输入的值的 列表。用户 可将列表中 的光标放到 所选值上并 按下 F2 或双击鼠标 ，将该值复 制到字段中 。

内容

用这种方法 输出的信息 保存在 ABAP/4 库中。在对 话程序中， 屏幕处理器 自动显示各 字段的数据 元素的帮助 文本。可能 值的 F4 列 表通常会引 用某个域的 有效值的固 定设置或相 关值表中的 数据。标准 列表对于用 户自己开发 的应用程序 通常也是够 用的。但是 ，根据具体 的对话程序 不同，也可 以建立更详 细或稍有不 同的文档和 可能值列表 。

可以用 PROCESS ON HELP-REQUEST (POH) 和 PROCESS ON VALUE-REQUEST (POV) 事件来编写 帮助文本和 可能值列表 程序。

这些事件可 以在屏幕处 理逻辑中与 PROCESS BEFORE OUTPUT 和 PROCESS AFTER INPUT 事件一起执 行。

PROCESS ON HELP-REQUEST：

语法

```
PROCESS ON HELP-REQUEST.
       FIELD <field> MODULE <module>.
       FIELD <field> MODULE <module> ...
```

通过 FIELD 语句将模块 调用分配给 屏幕字段。 如果按下 F1 或 ， 就会执行光 标所在字段 的 POH 模块。

PROCESS ON VALUE-REQUEST：

语法

```
PROCESS ON VALUE-REQUEST.
       FIELD <field> MODULE <module>.
       FIELD <field> MODULE <module> ...
```

用户按下 F4 或激活屏幕 字段中的组 合框之后， 会发生 POV 事件。如果 该字段包含 FIELD 语句，则调 用指定的 POV 模块而不是 帮助处理器 。

在调用屏幕 时，将执行 PROCESS BEFORE OUTPUT 事件。然后 就会出现自 动传输给屏 幕的字段。 如果在屏幕 中选定 F1 帮助或 F4 值列表，则 系统会再次 发送 动态程序， 但不执行 PBO 事件，也不 执行字段转 移以将所有 字段转移给 动态程序 。在 POV 后，只为光 标所在字段 和通过 F2 键或双击而 选定的字段 的内容进行 传输。

如果 POV 处理需要附 加字段传输 ，则可以使 用 DYNP_VALUES_READ 和 DYNP_VALUES_UPDATE 功能模块。其它功能模 块可简化特 定上下文的 帮助文本和 输入值的编 程。

匹配代码对 于可能值帮 助也会有所 帮助。本章 最后一节包 含一个示例 。

以下主题提 供详细信息 ：

定制 F4-值-请 求 (页 431)

定制 F1-帮助 (页 425)

字段 - 帮助的功能 模块 (页 429)

具有匹配代 码的字段-帮 助 (页 409)

定制 F4-值-请 求

在为字段建 立自己的 F4 值帮助时， 可以使用固 定值或相关 域的检查表 。如果标准 的 F4 帮助不充分 或不适合特 定的上下文 ，则可以使 用 PROCESS ON VALUE-REQUEST 事件编写自 己的 F4 值帮助程序 。示例事务 TZBO 包含编程的 “出发机场 ”和“目的 机场”字段 的可能值帮 助。通过 F2 或双击鼠标 可在列表中 选定出发机 场和目的机 场。

这两个字段 都引用 S_AIRPID 域。建立标 准 F4 帮助的最简 单方法就是 在 ABAP/4 词典中为该 域分配一个 检查表。但 是，如果结 果值列表不 符合要求， 则必须自己 编写值帮助 程序。下面 以事务 TZBO 为例，讲述 一种编程方 法。
首先,事务 TZBO 的屏幕 200 的处理逻辑 必须指定要 为其提供值 帮助的字段 。在其中某 个字段上使 用 F4 键会导致执 行该字段的 POV 模块。

```
PROCESS ON VALUE_REQUEST.
     FIELD SPFLI-AIRPFROM
               MODULE VAL_REQ_AIRPORT.
     FIELD SPFLI-AIRPTO
               MODULE VAL_REQ_AIRPORT.
```

PAI 模块 VAL_REQ_AIRPORT 包含对同名 子程序的调 用。

```
MODULE VAL_REQ_AIRPORT INPUT.
     PERFORM VAL_REQ_AIRPORT.
ENDMODULE.
```

VAL_REQ_AIRPORT 子程序由以 下三部分组 成：

- 使用 DYNP_VALUES_READ 功能模块从 屏幕上读取 字段内容。
- 为编程 的值帮助调 用 VALUE_REQUEST_AIRPORT 功能模块。
- 使用 DYNP_VALUES_UPDATE 功能模块将 字段内容写 回到屏幕上 。

> 完全处于演 示目的才将 DYNP_VALUES_READ 和 DYNP_VALUES_UPDATE 功能模块集 成到该示例 事务中。如 果在 POV 处理中需要 多个字段传 输，则可以 使用它们。 如果在调用 VALUE_REQUEST_AIRPORT 功能模块后 ，打算使用 MOVE 命令将选定 字段值复制 到 SPFLI-AIRPFROM 和 SPFLI-AIRPTO 字段中，则 可以将这些 功能模块从 该特殊示例 事务中省略 掉。

在 VAL_REQ_AIRPORT 子程序中， DYNP_VALUES_READ 功能模块获 取请求值帮 助的字段内 容。然后将 该值传送给 帮助处理器 。
在输出参数 DYNAME 和 DYNUMB 中指定程序 名和屏幕号 。 DYNPFIELDS 表参数需要 一个具有 ABAP/4 词典结构 DYNPREAD 的表。本例 中该表命名 为 SCR_FIELDS。 在 DYNP_VALUES_READ 功能模块的 该表中指定 要读取的屏 幕字段。该 表应包含值 帮助所需值 的字段。用 GET CURSOR 命令找到字 段名然后将 其复制到 SCR_FIELDS 表中。

```
FORM VAL_REQ_AIRPORT.
     DATA:     CURSORFIELD(30),
               SCR_FIELDS LIKE DYNPREAD OCCURS 1 WITH HEADER LINE,
               AIRPORT LIKE SPFLI-AIRPFROM.

GET CURSOR FIELD CURSORFIELD.
SCR_FIELDS-FIELDNAME = CURSORFIELD.
APPEND SCR_FIELDS.
CALL FUNCTION 'DYNP_VALUES_READ'
               EXPORTING
                    DYNAME    = 'SAPMTZBO'
                    DYNUMB    = SY-DYNNR
               TABLES
                    DYNPFIELDS     = SCR_FIELDS
               EXCEPTIONS
                    UNDEFIND_ERROR= 08
                    OTHERS         = 04.
IF SY-SUBRC NE 0. MESSAGE I007. EXIT. ENDIF.
```

VAL_REQ_AIRPORT 子程序从 SCR_FIELDS 表中顺序读 取机场名并 用 AIRPORT 更改参数调 用 VALUE_REQUEST_AIRPORT 功能模块。 该功能模块 负责输出带 有机场列表 的弹出窗口 。用户选定 机场之后， 该值通过 AIRPORT 变量传回调 用程序。

```
READ TABLE SCR_FIELDS INDEX 1.
AIRPORT = SCR_FIELDS-FIELDVALUE.
CALL FUNCTION 'VALUE_REQUEST_AIRPORT'
               CHANGING
                    AIRPORT = AIRPORT
               EXCEPTIONS
                    CANCELLED = 1
                    NO_VALUES = 2
                    OTHERS    = 3.
IF SY-SUBRC = 2. MESSAGE I008. ENDIF.
CHECK SY-SUBRC = 0.
```

使用 DYNP_VALUES_UPDATE 功能模块将 AIRPORT 参数的内容 读入 SCR_FIELDS 表并将当前 字段内容放到屏幕上。

```
READ TABLE SCR_FIELDS INDEX 1.
SCR_FIELDS-FIELDVALUE = AIRPORT.
MODIFY SCR_FIELDS INDEX 1.
CALL FUNCTION 'DYNP_VALUES_UPDATE'
              EXPORTING
                   DYNAME    = 'SAPMTZBO'
                   DYNUMB    = SY-DYNNR
              TABLES
                   DYNPFIELDS     = SCR_FIELDS
              EXCEPTIONS
                   OTHERS    = 04.
ENDFORM.                              " VAL_REQ_AIRPORT
```

VALUE_REQUEST_AIRPORT 功能模块负 责该程序中 值列表的输 出。首先检 查是否输入 了一般机场 名。如果是 ，则对特殊 字符进行转 换，以便后 面的 SELECT 语句能使用 该一般输入 。

```
*"----------------------------------------------------------------------
*"*"Local interface:
*"             CHANGING
*"                  VALUE(AIRPORT) LIKE SPFLI-AIRPFROM
*"             EXCEPTIONS
*"                  CANCELLED
*"                  NO_VALUES
*"----------------------------------------------------------------------
            IF AIRPORT CA '+*'. SELECTION = AIRPORT.
                                       TRANSLATE SELECTION USING '*%+_'.
            ELSE.                   SELECTION = '%'.
            ENDIF.

     RETURN = 'BEGIN'.
     CALL SCREEN 100 STARTING AT 10 5
                     ENDING AT 41 15.

     CASE RETURN.
              WHEN 'SELE'.  AIRPORT = SELECTION.
              WHEN 'BREAK'. RAISE CANCELLED.
              WHEN 'NOVAL'. RAISE NO_VALUES.
     ENDCASE.
ENDFUNCTION.
```

屏幕 100（用 CALL SCREEN 调用）用作 值帮助的弹 出屏幕。在 屏幕制作器 中将该屏幕 创建为空屏 蔽。用 SELECT 获取数据， 然后将其写 入屏幕 100 的 PBO 模块 LIST_PROCESSING 的弹出屏幕 中。

```
MODULE LIST_PROCESSING OUTPUT.
LEAVE TO LIST-PROCESSING AND RETURN TO SCREEN 0.
                  NEW-PAGE LINE-SIZE 31.
                  SET PF-STATUS 'POP_PICK'.
                  SET TITLEBAR 'TIT'.
                  SELECT * FROM SAIRPORT WHERE ID LIKE SELECTION.
                             WRITE: /    SY-VLINE NO-GAP,
                                SAIRPORT-ID COLOR 4 INTENSIFIED OFF NO-GAP,
                                SY-VLINE NO-GAP,
                                SAIRPORT-NAME COLOR 2 INTENSIFIED OFF NO-GAP,
                                SY-VLINE NO-GAP.
                             HIDE SAIRPORT.
                  ENDSELECT.
                  IF SY-SUBRC NE 0. RETURN = 'NOVAL'. ENDIF.
                  ULINE.
                  LEAVE SCREEN.
ENDMODULE.                                " LIST_PROCESSING  OUTPUT
```

在编写 PROCESS ON VALUE-REQUEST 程序时，要 记住必须能 选定值并能 将其复制到 所属的屏幕 中。在执行 HIDE SAIRPORT 时，用 SELECT 获取的所有 值都被记录 下来。然后 ，用光标选 定一行并按 下菜单制作 器中被赋予 PICK 功能的功能 键（通常为 F2）之后 ，对 AT LINE-SELECTION 进行处理， 即自动用存 储值对该行 进行填充。

```
AT LINE-SELECTION.
     CHECK SAIRPORT-ID NE SPACE.
     RETURN = 'SELE'.
     SELECTION = SAIRPORT-ID.
     LEAVE LIST-PROCESSING.

AT USER-COMMAND.
            CASE SY-UCOMM.
              WHEN 'EXIT'. RETURN = 'BREAK'. LEAVE LIST-PROCESSING.
```

```
WHEN 'RTRN'. RETURN = 'BREAK'. LEAVE LIST-PROCESSING.
ENDCASE.
```

选定机场之 后，在 LSTZ1E01Include 中调用 AT LINE-SELECTION。 RETURN 变量包含值 'SELE' 。结果,SELECTION (SAIRPORT-ID) 之值被赋给 VALUE_REQUEST_AIRPORT 功能模块中 的 AIRPORT 变量。AIRPORT 的内容将传 回调用程序 ，所选机场 的名字出现 在为其调用 值帮助的字 段中。
如果按下值 列表中的功 能键（"确 定"或"取 消"），则 对 AT USER-COMMAND 进行处理。 在其中任何 情况下,都 将值'BREAK' 赋给 RETURN 变量,然后 停止对列表 进行处理。如果 RETURN 变量包含' BREAK'，则在功能 模块中产生 CANCELLED 例外。

定制 F1-帮助

ABAP/4 开发环境提 供许多上下 文敏感的 **F1** 帮助的设计 方法：

数据元素 文档。
通过为 ABAP/4 词典中的数 据元素添加 额外描述性 文本可以增 强 **F1** 帮助。在屏 幕制作器中 ，将屏幕字 段串中的光 标放在要引 证的字段上 ，并选定菜 单功能"转 向 -> 文档数据元 素文档"。 现在即可扩 展现有字段 帮助。扩展 帮助将在系 统中的任何 地方显示。
屏幕制作 器中的数据 元素附加文 本。
如果希望只 在特定事务 中扩展特定 字段的字段 帮助，则为 屏幕字段串 中的字段选 定"转向 -> 文档 -> 数据元素细 节"。将出 现一个弹出 窗口，不仅 包含数据元 素，也包含 当前屏幕号 ，用作数据 元素附加文 本的详细标 识符。除 ABAP/4 词典包含的 帮助文本外 ，还可以使 用编辑器输 入有关字段 使用方法的 信息，包括 示例和任何 影响该字段 的相互依赖 关系。
使用 PROCESS ON HELP-REQUEST 事件。
如果屏幕的 处理逻辑中 没有 PROCESS ON HELP-REQUEST 事件，则将 以 ABAP/4 词典中的字 段文档作为 基础。如果 存在 POH 事件，则只 要为指定的 字段按下 **F1** 就会执行。 例如，可以 按以下方式 实现特定字 段的数据元 素附加文本 ：

```
PROCESS ON HELP-REQUEST.
      FIELD XY WITH 'Number'.
```

通过使用以 下变量，也 可以将数据 元素附加文 本合并 **F1** 帮助中：

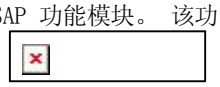
```
PROCESS ON HELP-REQUEST.
      FIELD XY WITH <var>.
```

为 XY 字段显示 <var> 变量中包含 的数据元素 附加文本。
如果按下 **F1** 键后才知道 数据元素附 加文本，则 可以在模块 中获取要显 示的附加文 本，然后将 其写入变量 中：

```
PROCESS ON HELP-REQUEST.
      FIELD XY MODULE XYZ WITH <variable>.
```

字段 - 帮助的功能 模块

如果要自己 编写 **F1** 和 **F4** 用户支持程 序，另一种 可能的方法 （上述方法 除外）就是 调用 POH 和 POV 事件的相关 SAP 功能模块。 该功能模块 提供标准 **F4** 帮助的全显 示功能。



> 调用示例事 务"*shlp*" ，这有助于 理解所述的 模块功能。 相应的程序 RSHELP01 包含该事务 的文档。也 可以在功能 库中查找该 功能模块文 档。

可以按如下 方法使用这 些功能模块 ：

1. 首先 通过 DYNP_VALUES_READ 获取屏幕的 当前字段值 （如果需要 的话）。
2. 在功 能组 SHL3 中选定工作 所需的功能 模块。

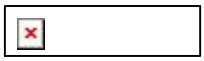
功能模块 的使用：

HELP_DOCU_SHOW_FOR_FIELD	数 据库字段的 **F1** 帮助。
HELP_VALUES_GET_EXTEND	根 据屏幕上现 有的字段值 显示 **F4** 值帮助。通 过 **F4** 选定并且返 回选定字段 值之后，也 可以将依赖 于该字段的 其它字段值 传送到屏幕 上。该功能 模块包括 DYNP_VALUES_READ 和 DYNP_VALUES_UPDATE 的功能。
HELP_VALUES_GET_NO_DD_NAME	显 示没有词典 参考的内部 表；字段在 ABAP/4 词典中必须 处于活动状 态。
HELP_VALUES_GET_WITH_DD_NAME	显 示具有词典 参考的内部 表。
HELP_VALUES_GET_WITH_TABLE_EXT	功 能模块允许 显示具有或 没有词典参 考的内部表 。表或者所 有字段在 ABAP/4 词典中都必 须处于活动 状态。

3. 调用 **F4** 之后，可以 用 DYNP_VALUES_UPDATE 将选定的表 行值返回到 屏幕。

具有匹配代 码的字段-帮 助

匹配代码是 SAP 系统中支持 数据库记录 搜索的一个 部件。可以 指定表和字 段，然后创 建基于这些 表和字段的 SAP 表上的视图 。
如果想启用 屏幕字段的 匹配代码搜 索，则应在 屏幕属性定 义中的“匹 配代码”字 段中指定匹 配代码对象 名。
要查看有关 示例，可以 链接 SPFL 匹配代码和 航班数据模 型中任何事 务中的 SPFLI-CONNID 字段。无论 何时需要输 入航班号码 ，这都能给 您提供帮助 。



要查找有关 如何定义匹 配代码的信 息，参见文 档 *ABAP/4 词典* (页 **Error! Not a valid link.**) 。

第十一章　调用外部程 序组件

概览

在 ABAP/4 中，有多种 使事务模块 化的选项可 供选择。这 些选项包括 所有可以调 用程序外部 代码组件的 方法。这些 外部组件可 以是功能模 块、其它事 务、对话模 块或报表。

内容

详细信息， 参见：
嵌入程序调 用 (页 353)
调用功能模 块 (页 352)
调用其它事 务 (页 353)
调用对话模 块 (页 371)
提交报表 (页 362)
在程序间传 送数据 (页 357)

嵌入程序调 用

外部程序组 件由系统进 行维护，对 所有程序都 可用。可在 事务中按任 意组合调用 这些组件。

外部程序和 滚动区

滚动区包含 程序的运行 时间上下文 。除运行时 间堆栈和其 它结构外， 所有局部变 量和程序可 以识别的任 何数据都存 储在这里。 系统是如何 处理外部程 序组件的滚 动区的呢？

事务在各 自的滚动区 中运行
报表在各 自的滚动区 中运行
对话模块 在各自的滚 动区中运行
功能模块 在其调用程 序的滚动区 中运行

调用运行自 己的滚动区 的外部程序 时，可以最 多嵌入 9 层调用。所 调用的功能 模块不单独 增加一层。

外部程序和 LUW 处理

运行时，事 务必须以“ 全都或全都 不”方式对 数据库进行 更新。或者 全部执行， 或者全部丢 弃。“LUW” （逻辑工作 单元）是某 一时间段， 在此期间所 需的任何更 新都属于“ 全都或全都 不”单元。“SAP LUW”是 指 ABAP/4 事务进行单 式组更新的 时间段。(SAP-LUW， 也称为*更新事务* (页 **错误！链接无效。**) ，与 数据库 LUW 不同)。
在调用外部 程序时，了 解相关的被 调程序或调 用程序如何 进行更新非 常重要。外 部程序与调 用程序在同 一 SAP LUW 中运行，还 是在单独的 SAP LUW 运行？

事务用**单 独的 SAP LUW** 运行
报表用**单 独的 SAP LUW** 运行

对话模块 与调用程序 在**同一 SAP LUW** 中运行
功能模块 与调用程序 在**同一 SAP LUW** 中运行
上述规则的 唯一例外是 用 IN UPDATE TASK （只是 V2 功能） 或 IN BACKGROUND TASK （ALE 应用程序） 调用的功能 模块。它们 总是在自己 的（单独的 ）更新事务 中运行。
如果程序调 用更新任务 功能或使用 COMMIT WORK，则外部程序 与其调用程 序共享(或 不共享）SAP LUW 这一事实会 产生特殊效 果。详细信 息，参见*编程数据库 更新*（页 **错误！链接无效。**）。

调用功能模 块

功能模块是 通用库例程 ，在整个系 统内都能使 用。有多种 用途，如操 作串、进行 特殊计算、 调用远程系 统上的程序 或发行标准 的屏幕序列 。
每个功能模 块都属于某 个“功能组 ”。功能组 是逻辑相关 的模块集合 ，相互共享 全局数据。 组内的所有 模块都包括 在同一主程 序中。当 ABAP/4 程序包含 CALL FUNCTION 语句时，系 统会在运行 时与程序代 码一起将整 个功能组装 载进去。

访问功能库

系统在功能 库中管理功 能模块。从 中可查阅现 有功能、其 调用接口和 文档，以及 创建新功能 等。要访问 功能库，请 按工作台中 的“功能库 ”。
在 ABAP/4 编辑器中， 可以双击代 码中的功能 模块名，或 使用“编辑 -> 插入语句” 功能。“插 入语句”会 查找功能模 块接口并在 程序中插入 功能调用模 板。所插入 的调用包含 预先格式化 的参数。

进行调用

用 CALL FUNCTION 语句调用功 能模块。例 如，假定要 在用户未保 存就退出事 务时给出要 求进行的提示。有一个 功能模块可 以完成这一 提示任务：

```
CALL FUNCTION 'POPUP_TO_CONFIRM_LOSS_OF_DATA'
     EXPORTING
          TEXTLINE1     = 'Do you want to save?'
          TEXTLINE2     = '????'
          TITEL         = 'REMINDER'
     IMPORTING
          ANSWER        = REPLY.
```

POPUP_TO_CONFIRM_LOSS_OF_DATA 使用 TEXTLINE 参数产生一 个弹出窗口 ：

从调用返回 后，变量 REPLY 包含用户的 回答：是（ ‘J’）或 否（‘N’ ）。
本节讲述如 何调用和编 写功能模块 。关于使用 功能库工具 的信息，参 见 *ABAP/4 工作台工具*（页 Error! Not a valid link.）。

使用功能模 块接口

程序只能使 用在功能模 块接口中说 明的参数向 功能模块传 送数据。在 CALL FUNCTION 语句中，参 数赋值的形 式通常为： 〈形参〉=〈实参〉。其中，形 参是接口中 指定的名字 。〈实参〉可以是变量 或常数。
在对 POPUP_TO_CONFIRM_LOSS_OF_DATA 的调用中， 调用程序使 用形参 TEXTLINE1、 TEXTLINE2 和 TITEL 将文本串 ‘ 想保存吗？ ’、‘????’ 和‘REMINDER’ 发送给功能 。用户的回 答放在变量 REPLY 中。
功能模块通 常可以包含 四种类型的 参数：

EXPORTING： 用于向被调 功能传送数 据。相应的〈形参〉在功能模块 接口中被指 定为输入参数。如果 EXPORTING 参数在功能 模块接口屏 幕中标记为 “可选”， 则可将其忽 略。
IMPORTING： 用于接受从 功能模块返 回的数据。〈形参〉在 功能模块接 口中被指定 为输出参数 。如果不需 要，可忽略 任何 IMPORTING 参数：这些 都是可选的 。
TABLES： 只用于通过 引用（即通 过地址）传 送内部表。 TABLES 参数在功能 模块接口屏 幕中未标记 为“可选” ，则不能将 其忽略。
CHANGING： 用于与功能 相互传送参 数（可能更 改为调用程 序版本）。 如果 CHANGING 参数在功能 模块接口屏 幕中标记为 “可选”， 则可将其忽 略。

> 也可以使用 “编辑 -> 插入语句” 功能（位于 ABAP/4 编辑器中） 指明哪些参 数可选。在 功能插入 CALL FUNCTION 语句之后， 就将可选参 数标注出来 。

关于处理功 能模块中例 外情况的详 细信息，参 见：

处理例外情 况

功能模块允 许程序员决 定出现例外 之后是由调 用程序进行 处理还是由 系统进行处 理。要告知 系统由调用 程序处理例 外，则应在 CALL FUNCTION 语句中指定 EXCEPTIONS：

```
CALL FUNCTION 'CONVERT_TO_FOREIGN_CURRENCY'
     EXPORTING
          DATE             = TRANS_DATE
          FOREIGN_CURRENCY = FCURRKEY
          LOCAL_AMOUNT     = AMOUNT
          LOCAL_CURRENCY   = LCURRKEY
```

```
      IMPORTING
           EXCHANGE_RATE     = RATE_USED
           FOREIGN_AMOUNT    = CONVD_AMT
           FOREIGN_FACTOR    = FCURR_FACTOR
      EXCEPTIONS
           NO_RATE_FOUND     = 1
           OVERFLOW          = 2.
```

例外类型在 功能模块接 口中进行定 义。对于 CALL FUNCTION 中提到的各 种例外类型 ，系统都假 定调用程序 自己处理该 错误类型。在上述语句 中，程序员 指明程序处 理两种错误 类型（NO_RATE_FOUND 和 OVERFLOW） ，其他则由 系统进行处 理。
要处理所有 的例外类型 ，请使用：

```
CALL FUNCTION 'CONVERT_TO_FOREIGN_CURRENCY'
      EXPORTING ...
      IMPORTING ...
      EXCEPTIONS
           NO_RATE_FOUND     = 1
           OVERFLOW          = 2
           NO_FACTORS_FOUND  = 3
           OTHERS            = 4.
```

OTHERS 关键字包含 CALL FUNCTION 语句中未列 出的所有例 外。使用 OTHERS 可包含打算 用同一方法 处理的所有 例外类型。 例如，如果 编写以下代 码：

```
CALL FUNCTION 'CONVERT_TO_FOREIGN_CURRENCY'
      EXPORTING ...
      IMPORTING ...
      EXCEPTIONS
           OTHERS            = 4.
```

则程序仍将 得到所有相 同的例外通 知，但这次 都使用单独 例外代码。

调用程序处 理例外时

如果出现了 程序应处理 的例外，功 能模块就将 SY-SUBRC 设置为相应 的数字，并 将控制直接 返回给程序 。只有引用 所调用的参 数才包含返 回值。如果 例外是由 MESSAGE ..... RAISING 产生的，则 系统用以下 系统字段中 将消息传回 去：

SY-MSGID （消息标识 符）
SY-MSGTY （消息类型 ）
SY-MSGNO （消息号）
SY-MSGV1 到 SY-MSGV4（ 消息中包括 字段 <**F1**> 到 <**F4**>的 内容）。

这些字段有 助于使用具 有特殊参数 的 MESSAGE 语句：

```
MESSAGE SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
                  WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
```

如果字段 SY-MSGV1 到 SY-MSGV4 中有任何一 个为空，系 统就将其全 部忽略。关 于发布消息 的详细信息 ，参见*发布消息*（页 **错误！链接无效。**） 。

系统处理例 外时

如果让系统 处理例外， 对于用户则 会发生以下 两种情况之 一：

程序终止
系统显示 消息并根据 消息类型继 续进行处理 。

如果既要编 写功能模块 又要编写调 用程序，参 见 *编程功能模块*（页 **错误！链接无效。**）以获取有关 例外处理的 详细信息。

调用其它事 务

可以从事务 内部转到或 调用其它事 务。“转到 ”新事务将 完全终止原 事务：用户 无法返回。 “调用”新 的事务则可 在被调事务 结束后返回 原事务。返 回后，将恢 复执行紧跟 在调用后面 的指令。

转到事务

要转到另一 事务并结束 当前事务， 请使用 LEAVE TO TRANSACTION 语句：

```
LEAVE TO TRANSACTION '<TRAN>'.
```

系统将显示 要转到的事 务初始屏幕 。开始新事 务之后，用 户就无法通 过按“退出 ”图标返回 上一事务的 上下文。在 上一事务中 用户没有保 存的所有数 据都将丢失 。

调用事务

如果希望用 户在处理完 中间事务后 能够返回初 始事务，请 使用 ABAP/4 语句：

```
CALL TRANSACTION '<TRAN>'.
```

与 LEAVE TO TRANSACTION 语句不同， CALL TRANSACTION 语句使系统 开始新的 SAP LUW（或*更新事务*（页 **错误！链接无效。**））。第二个 SAP LUW 与调用事务 的 SAP LUW 并行运行。 通过在被调 事务中使用 关键字 LEAVE， 可以让用户 返回调用程 序的程序上 下文。
在调用事务 时，可以告 知系统取消 事务的初始 屏幕而直接 到序列中的 下一个屏幕 ：

```
CALL TRANSACTION '<TRAN>' AND SKIP FIRST SCREEN.
```

对初始屏幕 进行处理但 不显示。只 有当初始屏 幕中所有必 需字段都已 赋值时取消 初始屏幕才 有意义。否 则，调用事 务时程序必 须传送数据 值。有关参 数传送技术 的信息，参 见 *在程序间传 送数据* (页 357) 。

调用与调用 程序共享 SAP LUW 的事务

有时需要调 用独立的事 务，但又希 望其与调用 程序在同一 SAP LUW 中运行。其 技巧在于将 现有事务转 换成对话模 块。为此， 只需创建一 个新的对话 模块，使其 主程序和初 始屏幕与现 有事务相同 。然后用 CALL DIALOG 调用新的对 话模块。
必须按照一 定规则对同 时用作事务 和对话模块 的事务进行 编程。详细 信息，参见 *用事务作为 对话模块* (页 356) 。

调用对话模 块

对话模块是 不属于特定 事务的可调 用屏幕序列 。对话模块 有自己的模 块池，可由 任何事务调 用。对话模 块与调用事 务在同一 SAP LUW （或*更新事务* (页 **错误！链接无效。**) ）中运 行。因此， 系统忽略对 话模块发送 给更新任务 的任何更新 例程。这就 使得在编写 使用异步更 新的应用程 序时对话模 块特别有用 。因为对话 模块在自己 的滚动区中 运行（功能 模块共享调 用程序的滚 动区），所 以对话模块 比功能模块 运行速度慢 的多。
系统采用类 似于功能库 的工具管理 对话模块。可以使用该 工具查阅现 有对话、其 调用接口和 文档，以及 创建新对话 等。
要在 Workbench 中使用对话 模块工具， 请使用菜单 选项“开发 -> 编程环境 -> 对话模块” 。在“对象 浏览器” 中，使用“ 环境 -> 程序开发 -> 对话模块” 。

运行时执行 对话模块

使用 CALL DIALOG 语句调用对 话模块。例 如：

```
CALL DIALOG 'SWO_OBJTYPE_GENERATE'
                    EXPORTING
                              DIALOG_MODE          FROM MODE
                              DIALOG_OBJTYPE       FROM OBJTYPE
                    IMPORTING
                              DIALOG_RETURN TO RESULT.
```

用事务作为 对话模块

必须将用作 对话模块的 事务编程为 要作为事务 和对话模块 运行。以下 各节讲述事 务代码必须 适合两种用 途的两个区 域。

编写两用 SAP LUW
事务必须既 能够在自己 的 SAP LUW （或*更新事务* (页 **错误！链接无效。**) ）中运 行，也能够 在调用程序 的 SAP LUW 中运行。
对话模块运 行时，(在 其调用程序 的 SAP LUW 中)，要有 特殊的条件 ：

- 继承调用 程序用 ENQUEUE 创建的锁
  事务作为对 话模块运行 时，可以假 定给定对象 的锁已存在 。事务作为 事务运行时 ，则必须排 列自己的锁 。可以使用 系统变量 SY-CALLD 来确定在运 行时程序是 否在调用模 式下运行。关于锁的详 细信息，参 见 *SAP 系统中的锁* (页 **错误！链接无效。**) 。
- 继承调用 程序的更新 任务关键字
  关于更新任 务处理的信 息，参见 *更新任务中的绑定更新* (页 **错误！链接无效。**) 。
- 忽略 COMMIT WORK 语句
  可以在两用 事务中包含 COMMIT WORK， 但在程序作 为对话模块 运行时则被 将忽略。对 话模块中所 要求的全部 更新将在调 用程序的下 一 COMMIT WORK 中进行处理 。详细信息 ，参见 *COMMIT WORK 处理* (页 **错误！链接无效。**) 。
- 调用 IN UPDATE TASK 的功能模块 不在对话模 块中触发。
  必须确保用 ON UPDATE TASK 调用的任何 功能模块都 可以延迟到 调用程序中 的下一 COMMIT WORK 之后。
- PERFORM ON COMMIT 例程不在对 话模块中执 行。
  必须确保用 PERFORM ON COMMIT 调用的任何 FORM 例程都可以 延迟到调用 程序中的下 一 COMMIT WORK 之后。特殊 情况下，在 控制返回调 用程序时， 对话模块滚 动区中的局 部数据会消 失。用这种 方式调用的 FORM 例程不应依 赖该局部数 据。详细信 息，参见 *调用更新任 务功能的对话模块* (页 **错误！链接无效。**) 。

在编写“保 存”功能（ F12）或 在其它允许 用户确认其 操作的地方 ，通常需要 上述特征。 通常情况下 ，在编写两 用事务的代 码时，必须 使所有托付 处理所需的 更新既可以 在被调程序 中发生也可 以在调用程 序中发生， 而不失去其 正确性。

编写两用退 出
从两用事务 中退出必须 符合两种情 况：即从调 用模式返回 和从离开模 式返回。如 果程序由 CALL TRANSACTION、 CALL DIALOG 或 SUBMIT REPORT （ 及其它）等 激活，则以 调用模式执 行。在这种 情况下，系 统变量 SY_CALLD 设置为‘X’ 。如果当前 没有活动的 嵌入调用， 则以离开模 式执行。
LEAVE 语句用于以 下情况：

- LEAVE 语句导致从 调用模式返 回。
- LEAVE TO SCREEN NNN 导致从离开 模式返回。

由于两用事 务既能在调 用模式下， 又能在离开 模式下，因 此，应该同 时使用两个 语句，其顺 序如下：

```
AT USER-COMMAND.
....
  LEAVE.
  LEAVE TO SCREEN NNN.
```

如果当前是 作为事务运 行的，则忽 略第一个 LEAVE， 而执行 LEAVE TO SCREEN NNN。如 果是作为对 话模块运行 的，则执行 第一个 LEAVE ，而始终不 会执行第二 个 LEAVE。

提交报表

有两种从事 务生成报表 的选项：LEAVE TO LIST-PROCESSING 语句或 SUBMIT 语句。
在模块池中 使用 LEAVE TO LIST-PROCESSING 来编写列表 处理权。当 事务收集到 大量报表数 据时，用 LEAVE TO LIST-PROCESSING 生成报表是 最好的方法 。详细信息 ，参见 *转到列表过 程* (页 **错误！链接无效。**) 。
使用 SUBMIT 语句开始独 立于事务的 报表。该报 表在自己的 滚动区中运 行，不与调 用程序共享 公共数据区 。因此，当 事务和报表 很少使用共 同数据时， SUBMIT 比较合适。 使用 SUBMIT 的语法为：

```
SUBMIT RSBBB013.
```

要执行该语 句，系统将 离开当前程 序并启动报 表。
返回调用程 序： 要允许用户 返回调用事 务，请使用 关键字 AND RETURN。

```
SUBMIT RSFLFIND AND RETURN.
```

在这种情况 下，系统将 为报表打开 内部会话。 当用户从列 表显示返回 时，系统将 返回启动报 表的事务屏 幕。(直接 从紧跟 SUBMIT 后的指令继 续执行。)
显示选择屏 幕： 缺省情况下 ，在使用简 单 SUBMIT 语句时，不 会出现报表 的选择屏幕 。要显示报 表的选择屏 幕，请使用 VIA SELECTION-SCREEN 关键字：

```
SUBMIT RSFLFIND VIA SELECTION-SCREEN.
```

系统显示选 择屏幕，用 户可指定自 己的选择条 件。在从列 表显示返回 后，再次出 现选择屏幕 （以便用户 请求另一列 表)。关于 使用 SUBMIT 的详细信息 ，参见：

向报表传送 数据

有三种向报 表传送选择 和参数数据 的选项。分 别是：

- 使用 SUBMIT...WITH
- 使用报表 变量
- 使用 RANGE 表

下面将讲述 这些选项。

使用 WITH 关键字

可以使用 WITH 关键字指定 参数或选择 项所需的全 部值：

```
SUBMIT RSBBB013 WITH CARRID = SPFLI-CARRID.
```

在上述语句 中，“航空 公司”是报 表 RSBBB013 的选择项（ 航空公司） 。报表只显 示与要求的 航空公司相 关的记录。
可以通过以 下方法找到 相应选择屏 幕字段的名 字：

- 使用“编 辑 -> 插入语句 ”功能（位 于 ABAP/4 编辑器中） 将 SUBMIT 语句添加到 代码中
- 用 F1 键请求字段 帮助（在选 择屏幕中）
- 使用“实 用程序 -> 有关帮助” 请求逻辑数 据库信息

指定值的范 围
可以用 WITH 指定值的范 围。例如：

```
SUBMIT RSFLFIND WITH DATE BETWEEN '19950301' AND '19951003'.
```

其中，*date* 是 RSFLFIND 的选择项， 要求程序带 指定边界值 之间的日期 显示所有记 录。WITH 规范有多种 合法的格式 ：

```
WITH <P> <OPERATION> <Q> SIGN <S>
WITH <P> BETWEEN     <F1> AND <F2> SIGN <S>
WITH <P> NOT BETWEEN <F1> AND <F2> SIGN <S>
WITH <P> IN <RANGES-TABLE>
WITH SELECTION-TABLE <SELTAB>
WITH FREE SELECTIONS <EXPRESSION-TABLE>
```

<operation> 可以是 EQ、NE、 CP、NP、 GE、LT、 LE、GT 中的任何一 个。符号<S>可 选。如果使 用，则 <S> 必须是‘I’ （包含）或 ‘E’(排 斥)。所有 操作符 <Q>、 <F1> 和 <F2> 都将按内部 （非显示） 格式传送给 程序。
当 <P> 是选择项时 ，系统将在 相关选择项 表中创建条 目，按照指 定情况对 LOW、HIGH、OPTION 和 SIGN 字段进行。 如果 <P> 是参数，则 将 <operation> 的所有值都 解释为 EQ。
关于使用 WITH 的详细信息 ，参见 SUBMIT 的关键字文 档。
关于使用 RANGES 表的信息， 参见 *RANGES 语句* (页 **错误！链接无效。**) 。
关于选择项 表的详细信 息，参见 *选择表* (页 **错误！链接无效。**) 。

在报表中使 用变量

可以使用报 表变量来指 定参数和选 择项值。为 此，请使用 USING SELECTION-SET 关键字：

```
SUBMIT RSFLFIND USING SELECTION-SET VARIANT1.
```

关于报表变 量的详细信 息，参见 *使用变式预定义选择* (页 **错误！链接无效。**) 。

使用 RANGES 指定选择项

可以通过填 写 RANGES 表指定选择 项值。例如 ：

```
TABLES SPFLI.
RANGES S_CARRID FOR SPFLI-CARRID.
S_CARRID-SIGN   = 'I'.
S_CARRID-OPTION = 'EQ'.
S_CARRID-LOW    = 'LH'.
APPEND S_CARRID.
SUBMIT RSFLFIND WITH CARRID IN S_CARRID.
```

在此例中， S_CARRID 是结构与选 择表相同的 内部表。通 过引用列 CARRID （数据库表 SPFLI） ，字段 S_CARRID-LOW 和 S_CARRID-HIGH 获得与 CARRID 相同的数据 类型。内部 表 S_CARRID 的表头行将 被填写并被 附加到表中 。该表功能 中定义的选 择条件类似 于逻辑表达 式 SPFLI-CARRID EQ 'LH'。

关于使用 RANGES 表的详细信 息，参见 *RANGES 语句* (页 **错误！链接无效。**) 。

保存或打印 报表

不必为用户 显示列表。 可以将其打 印出来，或 保存到其它 存储设备中 。

将报表发送 给打印机

可以将报表 发送给打印 机而不显示 在屏幕上。 为此，请使 用关键字 TO SAP-SPOOL：

```
SUBMIT RSFLFIND ... TO SAP-SPOOL DESTINATION 'LT50'.
```

在使用该特 征时，有多 种选项可指 定打印所需 的参数。详 细信息，请 参见 SUBMIT 联机文档（ ABAP/4 编辑器中的 “工具 -> 有关帮助...”)。

将报表保存 在内存中

可以使用 SUBMIT 生成报表并 将其保存在 ABAP/4 内存中，而 不显示在屏 幕上。为此 ，请使用关 键字 EXPORTING LIST TO MEMORY：

```
SUBMIT RSFLFIND ... AND RETURN
                    EXPORTING LIST TO MEMORY.
```

该特征将生 成的报表放 在 ABAP/4 内存中，从 SUBMIT 调用返回之 时，调用程 序可从中进 行访问。功能组 SLST 提供访问已 保存的报表 的功能模块 ，其中包括 ：

```
LIST_FROM_MEMORY
WRITE_LIST
DISPLAY_LIST
```

请注意，AND RETURN 关键字是该 特征所必需 的，但不允 许使用 TO SAP-SPOOL 关键字。详 细信息，参见 SUBMIT 联机文档（ ABAP/4 编辑器中的 “工具 -> 有关帮助...”)。

在程序间传 送数据

事务、对话 模块和报表 都在自己的 滚动区中运 行。在从事 务中调用其 中任何一个 时，都必须 给它们传送 运行所需的 数据。向外 部程序传送 数据的选项 有：

使用 SPA/GPA 参数（SAP 内存）

这是在外部 程序之间传 送数据的最 常用方法。 详细信息， 参见：*用 SPA/GPA 参数传送数 据* (页 359) 。

使用 EXPORT/IMPORT 数据（ABAP/4 内存）

任何程序都 可以使用 EXPORT 语句在 ABAP/4 内存中存储 数据字段簇 。因此，该 数据就全局 有效（使用 IMPORT） ，在程序本 身中以及任 何被调事务 、报表或其 它模块中都 有效。使用 EXPORT：

```
EXPORT <OBJECT1> <OBJECT2> ... <OBJECTN> TO MEMORY ID <ID-NAME>.
```

然后调用程 序就会检索 数据：

```
IMPORT <OBJECT1> <OBJECT2> ... <OBJECTN> FROM MEMORY ID <ID-NAME>.
```

ID 参数标识唯 一的数据簇 。如果将同 一对象多次 输出到同一 ID，则会 改写内存中 该簇的第一 个版本。如 果第二次输 出对象的子 集，则仍会 改写该组的 第一个版本 中的“所有 ”对象（不 仅是子集） 。

只有调用程 序和被调用 程序经常一 起使用时， 才用 EXPORT/IMPORT 实现参数传 送。对于外 部应用程序 可用的调用 程序不推荐 EXPORT/IMPORT，因为这些应 用程序将根 本无法找到 调用所需的 接口。

关于输出和 输入数据簇 的详细信息 ，参见 *ABAP/4 内存中的数 据簇* (页 **错误！链接无效。**) 。

用 SPA/GPA 参数传送数 据

可使用 SPA/GPA 参数向被调 用的程序传 送数据。SPA/GPA 参数是全局 保存在内存 中的字段值 。每个参数 都用三个字 符代码标识 ：通过选择 在第一个屏 幕上的“其 他对象”可 以在对象浏 览器中定义 这些参数。 SPA/GPA 存储器是用 户指定的并 在用户整个 会话期中都 有效。

有两种使用 SPA/GPA 参数的方法 ：

通过在“ 屏幕制作器 ” 中设置字段 属性

“SET 参数”、“ GET 参数”和“ 参数 ID”属性 告知系统是 向“参数 ID”存储 值还是从中 检索值。系 统使用这些 值自动初始 化屏幕字段 值。

对调用屏幕 中给定字段 的“SET 参数”属性 以及被调用 屏幕中相应 字段的“GET 参数”属性 进行标记。 系统会自动 将字段内容 从调用事务 传送给它所 触发的事务 中。

通过使用 SET PARAMETER 或 GET PARAMETER 语句

用这些语句 可以存储和 检索来自 ABAP/4 程序的 SPA/GPA 值。如果两 个事务的选 择屏幕没有共享同一必 需的字段， 则请使用这 些语句按名 称显式存储 屏幕字段。

在从 PAI 模块调用新 事务之前， 用一个名称 之下存储调 用程序事务 的字段：

```
SET PARAMETER ID 'RID' FIELD <FIELD NAME1>.
```

系统将值存 储在 SPA 参数‘RID’ 中的<字段 1>中。三 个字符的标 识符‘RID’ 必须在 SAP 表 TPARA 中定义。如 果 SPA 参数‘RID’ 已经包含值 ，则 SET PARAMETER 语句会将其 改写掉（用<FIELD NAME1>的 内容）。

在被调事务 的 PBO 模块中，在 其他名称下 检索字段：

```
GET PARAMTER ID 'RID' FIELD <FIELD NAME2>.
```

系统读取‘ RID’的 内容并将其 传送给<FIELD NAME2>。

例如，假定 要将屏幕字 段和其它数 据从调用事 务传送给被 调用事务。 调用事务可 以将某些值 存储在 SPA 参数中：

```
SET PARAMETER ID 'RID' FIELD REPORTID.
CALL TRANSACTION 'SE38'.
```

然后，被调 事务即可在 PBO 获取信息， 以便将其显 示到屏幕上 。此处将出 现事务 SE38 的初始屏幕 ，其报表 ID 已填好。这 在使用 CALL TRANSACTION AND SKIP FIRST SCREEN 时非常有用 。除非所需 的字段值由 内存提供， 否则不能取 消第一个屏 幕。

第十二章　定制事务

概览

记住需要的 用户和工作 环境的同时 ，应设计事 务使之尽可 能简单和对 用户友好。 根据事务目 的，可以预 先给字段分 配值，更改 其输入准备 情况以及取 消字段甚至 整屏。完成 这些操作可 用的技术包 括：

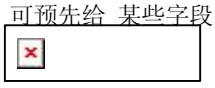
内容

1. 定义 全域字段值
   全域字段 值允许您为 每个用户组 在系统范围 内给字段预 分配值。可 以禁止这些 字段或使它们只显示。
2. 使用 事务变量
   可以为同一 事务创建不 同变量。这 些变量的不 同之处在于 相关事务的 输入准备可 以包含不同 值或字段。 可能已禁止 屏幕或者已 更改字段的 输入准备。

详细信息， 参见下列章 节：
全域字段值 (页 412)
事务变量 (页 414)

全域字段值

要简化事务 ，可预先给 某些字段分 配值。还可 以禁止这些 预分配的字 段，或使它 们只显示。

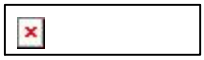


只具有一个 公司代码的 客户可以将 相关公司代 码预分配给 那个字段， 然后禁止显示或阻止用 户更改该字 段

特别是复杂 屏幕，可以 通过禁止字 段进行简化 。带受禁止 字段的屏幕 通常都是被 压缩的。
全域值参考 字段类。字 段类将志向 相同域的字 段组合在一 起。有关的 域必须具有 值表。由于 不同的域不 能有相同的 值表，所以 相同的值表 属于字段类 中的所有字 段。
如果全域值 是为特定字 段维护的并 且是活动的 ，那么运行 时，该值是 为字段类的 所有字段设 置的，而不 管事务如何 。
要维护全域 值，请进行 下列操作：

1. 选择 “工具 -> 管理 -> 维护用户 -> 全域值”。
2. 在初 始屏幕上， 输入要为其 维护全域值 的用户组。 默认值为您 自己的用户 组。
3. 如果 选择“更改 ”，可以看 到要为其输 入全域值的 字段列表。
   如果还没有 为您的用户 组创建字段 类，请选择 “生成字段 类”。然后 输入可以从 中生成字段 类的域。
4. 选定 字段类后， 在“值”列 中输入所需 的全域字段 值。
5. 下一 列“活动” 特别重要。 只要选定该 列，相关字 段类的字段 都将被预分 配的值填充 （或被禁止 等）。
6. 如果 想禁止该字 段，请选择 “不可见” 列。

7. 如果 希望字段不 能接收输入 ，请选择“ 拒绝输入” 列。



不能更改带 初始值的字 段的输入属 性。如果要 设置的值与 事务内部设 置的值或手工输入的值 中的某一个 不匹配，则 全域值不是 为该事务设 置的，并且 字段的显示属性保持不 变。选择屏 幕和列表中 的字段没有 改变。分步 循环字段被 分配相同的值。

全域值优先 于用参数 ID 设置的值。
由事务变量 设置的值（ 和显示属性 ）优先于全 域值。
全域字段值 是特定于客 户和用户的 。

事务变量

事务变量允 许用户通过 以下步骤简 化事务流：

预先给字 段分配值
禁止和更 改字段的输 入准备
禁止整屏

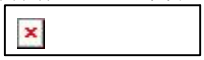
禁止与与屏 幕压缩有关 的字段以及 禁止屏幕都 可提高可读 性和简化事 务。
事务变量分 配给一个事 务。一个事 务可以有多 个变量。
详细信息， 参见下列章 节：
维护事务变 量 (页 419)
用变量启动 事务 (页 415)
传输事务变 量 (页 413)

维护事务变 量

可在 R/3 系统中用事 务 shd0 维护事务变 量。
在初始屏幕 上，输入事 务和变量名 称。然后选 择“创建” 或“更改” 进入下一屏 。在此输入 一个短文本 。如果想为 变量指定字 段值，则必 须保存该文 本。
“屏幕输入 ”功能允许 您为变量输 入值。
这时您要为 其维护变量 的事务开始 启动，您可 以输入值。 屏幕每更改 一次，在带 有输入值的 当前屏幕上 您都可看到 一个包含字 段列表的对 话框。带有 相同技术名 称的字段（ 如文本和输 入字段）一 起处理。
在这里，可 选择不同选 项。前两个 选项参考整 个屏幕，而 其它选项可 适用于单个 字段。

复制字段值	采用系统设 置的默认值 。
不显示屏 幕	事务期间 禁止屏幕。
带内容	在变 量中使用值 。
只输出	字 段只读。
不 可见	不显示 字段。

通过取消该 屏幕的“复 制字段值” 选项，可以 从变量中删 除预先为整 个屏幕分配 的值。
要继续事务 ，请选择“ 继续”。您 可以为后续 屏幕输入值 。
如果选择“ 取消”，则 系统再次显 示当前屏幕 ，您可以纠 正任何错误 条目。屏幕 改变时，对 话框也改变 。
选择“退出 并保存”退 出事务。然 后就会得到 要处理其字 段值的事务 中的所有屏 幕列表。



变量中每个 屏幕只能被 处理一次， 例如，如果 多次处理一 个屏幕，则 每次都使用 相同的值。

属性与字段 如何匹配？
对于输入字 段，可以使 用字段值和 更改输出属 性。如果为 输入字段更 改输出属性 ，则在变量 中自动使用 字段内容（ 即使没选定 相关的列） 。
对于具有用 户特有格式 的字段（例 如，日期字 段和带小数 位的字段） ，不使用任 何值。但是 ，可以更改 输出属性。 这同样适用 于初始值。
变量中不能 使用功能条 目。
对于单选按 钮和按钮， 不使用任何 值，但可以 更改按钮的 输出属性。
所有其他字 段都可禁止 。
特殊规则运 用于：

分步循环 字段
对于分步循 环中的每个 输入字段， 可以设置自 己的值。只 能为每个分 步循环（对 于第一个字 段）设置一 次输出属性 （不可见和 无条目）； 这些涉及分 步循环中的 所有字段。
子屏幕

对于每个屏 幕，都有一 个独立的对 话框，通过 该对话框您 可以处理子 屏幕的字段 值。在这些 对话框中， 由于技术原 因，“取消 ”功能无效 。

用变量启动 事务

可以按以下 任何方式， 用变量启动 事务：

用维护事 务中的“测 试”功能。
用自己的 事务代码。 然后将该事 务定义为“ 变量事务” 。
通过调用 功能模块 RS_HDSYS_CALL_TC_VARIANT 从程序中启 动。

变量事务

对于带变量 的事务，可 以创建您自 己的事务代 码。该代码 总是用特殊 变量调用预 定义的事务 。
定义事务时 ，首先输入 事务名称（ 在 tz11 情况下)， 然后再选择 事务类型“ 变量事务” 。在下一屏 中，输入要 执行的事务 的名称和变 量名

变量和参数 事务

变量事务与 参数事务不 同之处在于 ：

变量事务 是客户特有 的
变量事务 允许您给多 屏预分配值 以及禁止字 段；参数事 务只允许在 初始屏幕执 行这些操作 。

维护事务代 码时，可以 选择“编辑 -> 更改事务类 型”以将参 数事务转换 为变量事务 。其中，系 统创建包含 参数事务值 的变量 CV_P_<变 量事务名称 >。该变量 是为被调用 的事务创建 的。维护事 务变量时可 以更改它。 变量只能在 当前客户创 建。

事务变量 STANDARD

变量 STANDARD 有特殊用途 。每次启动 事务（和变 量事务）时 ，系统自动 搜索该变量 。如果变量 存在，并且 对相关事务 是活动的， 则事务用该 变量启动。

传输事务变 量

事务变量是 客户特有的 。变量维护 中的“客户 复制”功能 允许您将变 量复制到另 一个客户。
为此，请在 变量维护的 初始屏幕输 入事务名称 和变量，并 选择“客户 复制”。
维护事务时 ，通过选择 “开发对象 -> 传输”，可 以创建传输 请求并用其 变量将事务 传输给其它 系统。可以 选择要按常 规方式传输 的变量。