第 4 章　ABAP 语言基本语法

一、单元概述

本章主要介绍了 ABAP 语言的基本语法。通过本章学习，理解 ABAP 数据对象与数据类型的含义与关系；理解并掌握多语言支持程序的含义与编写方法；掌握数据对象的声明与赋值方法、ABAP 常用的字符串的处理方法、ABAP 语言流程控制的类型与方法、发送对话消息的方法、ABAP 程序调试的发方法。

二、教学重点与难点

重点：

(1)多语言支持程序的含义与编写方法；

(2)数据对象的声明与赋值方法；

(3)发送对话消息的方法；

(4)ABAP 程序调试的发方法。

难点：

(1)ABAP 数据对象与数据类型的含义与关系；

(2)ABAP 常用的字符串的处理方法；

(3)ABAP 语言流程控制的类型与方法。

解决方案：

通过教授教学法讲解理论知识；演示教学法课堂演示程序编程过程；练习教学法要求学生完成课后作业。

【案例引入】

经过上一阶段的准备工作，已经完成了 AIRFLY 旅行公司机票预定系统的系统工程创建工作，接下来的任务就是具体的功能开发，想要成功的开发出这个系统，应掌握 ABAP 开发的哪些技术要点呢？如何正确的编写 ABAP 程序呢？

【知识正文】

4.1 ABAP 语言概述

SAP R/3 系统是通用的企业管理 ERP 平台，可以在很大程度上适应各种行业的业务需求，但是这种通用性不能完全解决所有企业具体的业务需求。高级商务应用程序设计语言 ABAP 是 SAP R/3 系统专门的开发语言，SAP 客户或合作伙伴可以根据自己的业务需要，使用 ABAP 进行二次开发，这样的开发对 SAP R/3 标准解决方案适应企业的特殊问题非常重要。ABAP 是第四代支持结构化的程序设计语言，它包含了所有通常的程序控制结构和模块化概念，同时，ABAP 语言又发展成了一种面向对象的编程语言。

SAP 最初开发 ABAP(高级商务应用程序设计)语言仅为 SAP 系统内部使用，为应用程序员提供了一个优化的工作环境，经过持续的改进以满足商业领域二次开发的需要。SAP 系统的绝大多数商务应用程序都是通过 ABAP 开发的，现在仍然是 SAP R/3 系统中客户开发自己的业务应用程序的强大工具。

SAP 客户按照自己的需求和业务，用 ABAP 进行其自身的开发；ABAP 开发工作平台包含所有用于创建和维护 ABAP 程序的工具。ABAP 的可解释特性使其易于生成、测试并运行程序的中间版本，以便将来生成最终版本，此过程也叫做早期的原型处理。ABAP 的基本特性包括：

(1)ABAP 语言元素包括：

①具有各种类型和结构数据声明的声明元素；

②用于数据操作的操作元素；

③控制程序流程的控制元素；

④反应外部事件的事件元素。

(2)ABAP 支持多种语言。文本摘要(如：标题、页眉和其他文本)将根据文本编码和语言种类分别存储。程序员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，这种技术使同一个 ABAP 程序在不同语言的用户登录界面中将按照不同的语言显示为不同版本的文字。

(3)ABAP 支持商业数据类型和操作，程序员可以用特殊日期和时间字段或者货币等计量单位进行计算，系统会自动执行必需的类型转换。

(4)ABAP 提供一系列强大的处理字符串的功能。

(5)ABAP 语言包含一个叫 OpenSQL 的 SQL 子集。使用 OpenSQL，程序员可以读取和访问数据库表，并且与所用的数据库系统无关，该技术使 ABAP 应用程序自动获得了跨数据库平台运行的特性。

(6)ABAP 允许程序员定义和处理内表(Internal Table)，该内表数据对象是在程序的运行期间存在于内存中的变量。ABAP 内表的机制可以和用于数据库操作的 OpenSQL 相结合使用，使得程序员的数据库操作变得更加简单和容易，并且基于内表与其他 ABAP 数据类

型的组合可以进一步在程序中构造出更加复杂的内存数据结构。

(7)ABAP 允许程序员定义并且调用子程序,也可以调用其他程序的子程序。外部数据可以通过子程序的接口(参数列表),以各种方式从子程序传递或传递到子程序。

(8)ABAP 包含一种特殊类型的子程序,叫做功能模块(Function Module),相当于可以被全局调用的子程序。程序员可以在 ABAP 数据仓库(Repository)中创建和维护这些功能模块。在调用程序和功能模块之间有一个明确定义的数据接口。

(9)ABAP 已经发展成为一个面向对象(Object Oriented)的语言,支持所有的面向对象的程序设计与开发方法。

(10)ABAP 程序类型主要有两种:

①报表程序:主要用于提取并分析数据库表中的数据。这种分析结果可以显示在屏幕上或发送到打印机上。报表程序也可以基于逻辑数据库,逻辑数据库是特殊的 ABAP 程序,使开发者不必编写所有的数据库访问代码。

②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 对话程序。屏幕的流逻辑包含对 ABAP 对话模块程序的调用。

4.2　ABAP 数据类型

数据类型和数据对象(内存变量、常量和文字)是 ABAP 变量定义的根本,二者均可由程序员来声明和维护,我们用数据类型来定义数据对象。在 ABAP 中,可以通过标准数据类型来定义出新的数据类型,新的数据类型可以声明在一个 ABAP 程序内部并被该程序局部使用;也可以将新的数据类型声明成全局的,放在 ABAP 字典库中供所有程序调用。

4.2.1　数据类型和数据对象

ABAP 数据类型和数据对象的主要联系和区别是:

(1)数据类型:是纯粹的类型说明,不与任何实际的内存相关联,但可以表明由一个数据类型定义出来的数据对象占有多少内存。数据类型描述了数据对象的技术属性,用于定义数据对象。

(2)数据对象(如文本、变量、常量):主要指内存变量,由数据类型定义得到,是程序在运行期使用的用于存储临时数据的物理单元。每个数据对象都有分配给它的特定的数据类型,且每个数据对象占有一定的内存空间。不同数据类型对应的数据对象可以有不同的操作方法,ABAP 根据数据对象的数据类型来处理数据对象。

在 ABAP 程序中,全部数据对象必须要先声明再使用。声明数据对象的过程中,必须给数据对象指定数据类型。ABAP 中的数据类型的范围可以从基本类型(如:整型、字符串等)到非常复杂的结构(如:复杂结构体和内表等)。在 ABAP 中,数据类型可以应用于很多场合。如图 4-1 所示。

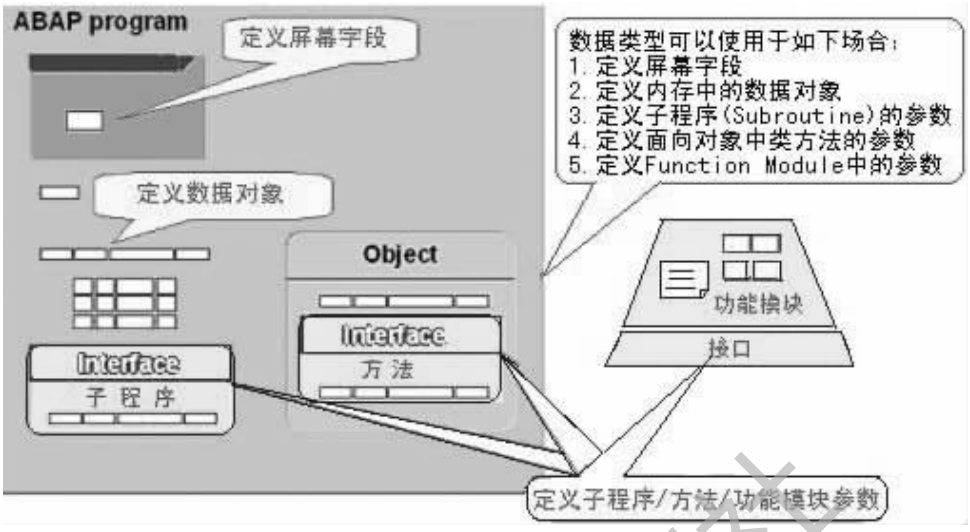

图 4-1 ABAP 数据类型的使用场合

(1)用于定义内存数据对象(Data Object)。

(2)用于定义屏幕上的输入/输出域(Input/Output fields)。

(3)用于定义子程序的输入输出参数(Interface parameters)。

(4)用于定义 ABAP 对象中方法(Method)的输入输出参数(Interface parameters)。

(5)用于定义 ABAP 全局功能模块(Function Module)的输入输出参数(Interface parameters)。

标准的数据类型是 ABAP 语言中预定义的、基本的、不可再分的类型。ABAP 标准数据类型按照其定义数据对象时的不同方式又可以分成两种,即完整的 ABAP 标准数据类型和不完整的 ABAP 标准数据类型。

4.2.2 完整的 ABAP 标准数据类型(Complete ABAP standard types)

完整的数据类型其长度是固定的,因此在用这种类型声明数据对象的时候,不需要指明数据对象的长度。完整的数据类型见表 4-1(有效大小以字节为单位)。

表 4-1 完整的 ABAP 标准数据类型

类型	长度	初始值	说明	定义数据对象例子
D	8	'00000000'	日期型 格式:YYYYMMDD	DATA mydate TYPE d VALUE '20080106'
T	6	'000000'	时间型 格式:HHMMSS	DATA mytime TYPE t VALUE '193027'
I	4	0	整型	DATA myint TYPE i VALUE 10
F	8	0	浮点型	DATA myfloat TYPE f.
STRING	变长	''	可变长字符串	DATA mystr TYPE string
XSTRING	变长	''	可变长十六进制字符串	DATA myxstr TYPE xstring

ABAP 提供了很多函数用于浮点型的计算,如:三角函数 cos,sin,tan 等;双曲函数 tanh,sinh,cosh 等;指数函数 exp;对数函数 log 与 log10;平方根函数 sqrt。

4.2.3　不完整的 ABAP 标准类型(Incomplete ABAP standard types)

不完整的数据类型其长度是不固定的,因此在用这种类型声明数据对象的时候,需要指明数据对象的长度。不完整的数据类型见表 4-2(有效大小以字节为单位)。

表 4-2　　不完整的 ABAP 标准数据类型

类型	默认大小	有效大小	初始值	说明	定义数据对象例子
C	1	1～65536	SPACE	文本、字符	DATA s(10) TYPE c VALUE 'hello'
N	1	1～65536	'00...0'	数字文本	DATA n1(4) TYPE n VALUE '3.14'
X	1	1～65536	X'00'	十六进制	DATA x1(4) TYPE x VALUE '3D6F'
P	8	1～16	0	压缩数字型	DATA p1(6) TYPE p DECIMALS 2 VALUE '67.56'

其中的 C 类型与完整标准类型中的 STRING 相对应,两者都可以用于声明一个字符串数据对象,但前者是定长的,而后者是可变长的。X 与 XSTRING 的情况与此相似。

定义一个内存数据对象要使用"DATA…TYPE"关键字,定义的同时,如果要为该数据对象赋初始值,则在后面使用 value 可选项,见表 4-2。如果定义数据对象时省略了 TYPE,缺省的类型是 C,若定义时省略了长度信息,则采用该类型的缺省长度。如"DATA c1(10)"是定义一个定长字符串 c1,最大长度是 10,而"DATA c2"则是定义一个长度为 1 的字符串变量 c2。

压缩型数字型 P 允许在小数点后有数字,其数值范围取决于大小和小数点后的位数,有效大小可以是 1～16 字节的任何值,可用于如距离、重量和钱数等商业数据的精确计算,与浮点型 F 类型相比较,压缩型数字型 P 具有更高的算术计算精度,因此更加适合于商业运算。P 类型的存储机制是将两个十进制数字压缩到一个字节,即每半个字节存储一个数字字符,而最后一个字节则包含一个数字和一个符号(+或-),因此如果 P 类型的变量声明长度是 n,则其最多能表示 2n-1 个字符,若小数部分超过长度,则自动按四舍五入将多余的小数位除掉;若整数部分超过长度,则系统运行出错。如定义:

```
DATA   p1(3)   TYPE   p   DECIMALS  4.
```

若设置 p1='67.567676767676876887'.,运行不会出错,p1 的值为'67.57';但如果要设置 p1='6756.76',就会导致程序运行错误。

4.2.4　局部数据类型

局部数据类型是由用户在程序中采用 ABAP 标准数据类型或字典库数据类型自定义的数据类型,用 TYPES 关键字声明局部数据类型,且只在被声明的程序中可见,因此称作局部类型。如:

(1)声明一个中国的身份证类型:TYPES id_card_type(18) TYPE c.

(2)声明一个 6 位的数字型作为人员编号:TYPES person_id_type(6) TYPE n.

(3)声明一个自定义结构体类型:

```
TYPES: BEGIN OF student,
       s_num(10)   TYPE p,
```

```
        s_name(10)   TYPE c,
        s_birth_date TYPE d,
        s_speci TYPE string,
END OF student.
```

使用自定义的数据类型定义数据对象的用法与使用 ABAP 语言标准数据类型定义数据对象的用法是一样的。如对于上述自定义局部类型可以如下声明数据对象：

```
DATA   myid   TYPE   id_card_type. "定义一个长度为 18 的定长字符串
DATA   wa   TYPE   student. "定义一个 student 类型的数据对象
wa-s_num = '0122358976'.
wa-s_name = '李明'.
wa-s_birth_date  = '19870812'.
wa-s_speci = '05 级电子商务专业'.
```

4.2.5 ABAP 数据类型分类

在 ABAP 程序中，定义一个数据对象可以采用三种数据类型中的一种，如图 4-2 所示。要在我们的 ABAP 程序中定义一个内存变量“do_name”，可以采用的数据类型有三个来源：

(1)ABAP 语言中预定义的标准数据类型，如图 4-2 中的“①”。

(2)数据字典中的全局数据类型，如图 4-2 中的“②”。

(3)用户在程序中自定义的数据类型，如图 4-2 中的“③”。

在 ABAP 程序中，还可以通过“DATA <变量 1> LIKE <变量 2>”的方式来定义一个新的<变量 1>，如图 4-2 中的符号“④”。图中的符号“⑤”是指采用 ABAP 语言的标准数据类型定义新的用户自定义数据类型。

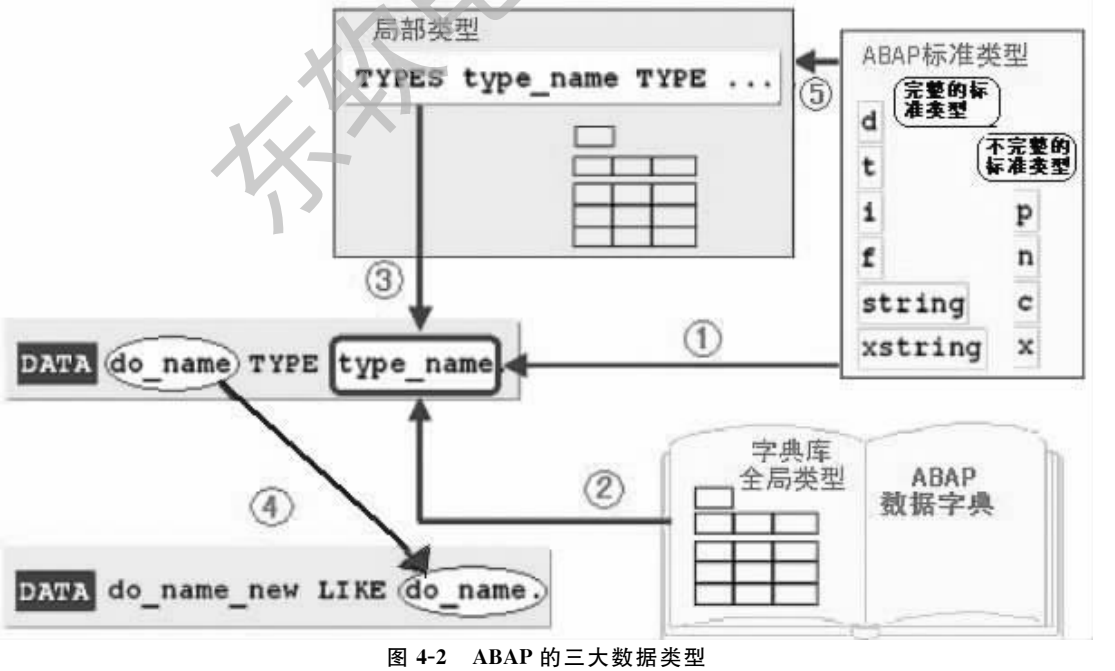

图 4-2 ABAP 的三大数据类型

其中，全局的数据类型(Global Data Types)存储在 ABAP 字典中主要有下几种类型：

(1)数据元素(Data Eelement):用于定义一个字段类型的变量。

(2)结构体(Structure):用于定义一个内存中的结构体变量,又称为工作区(Work Area)。

(3)透明表(Transparent Table):对于定义内存变量来说,其功能与结构体类型是一致的,即用于定义一个内存中的工作区(Work Area),但其定义的工作区一定是扁平的。

(4)内表(Internal Table):用于定义一个内存的多维表格。如果定义采用的行类型是基本的数据类型,则相当于一个一维数组;若其行类型为扁平结构体或透明表,则定义出来的是一个二维数组;若其行类型为复杂结构体,则定义出来的是一个多维数组。

采用数据类型定义 ABAP 程序中的数据对象的一个例子如图 4-3 所示。

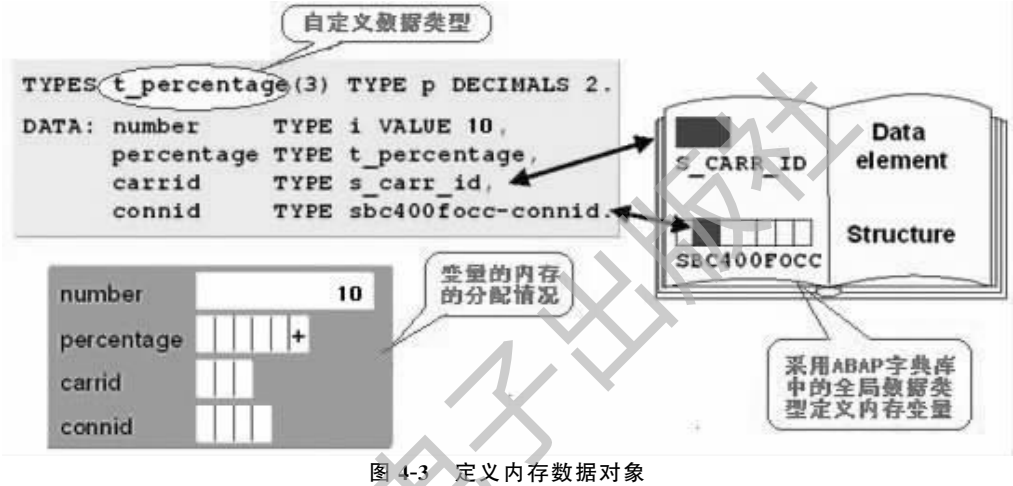

图 4-3　定义内存数据对象

4.2.6　ABAP 数据对象分类

在 ABAP 中,主要的数据对象有变量、常量和文字。

1. 变量

变量在内存保存临时数据,可以改变变量的值。变量的主要属性有名称、类型、长度和结构。可以采用 DATA 语句在程序中声明变量:

```
DATA:  i1  TYPE  i,
       i2  TYPE  i,
       i3  TYPE  i.
i3 =  i1 + i4.
...
```

2. 常量

常量是包含固定值的数据对象,该值是在初始化时确定的,并且在程序执行期间不能更改常量的值。在程序中采用 CONSTANTS 语句来声明常量。如果试图在程序中改变常量的值,则在语法检查或运行期间,系统将报出错误消息。若需要在程序中频繁地使用特殊数值,请使用常量。如果需要修改该值,只须更改在声明该常量时的初始值。如:

```
CONSTANTS  PI(10)  TYPE  p  DECIMALS  5  VALUE  '3.14159'.
```

因为定义的是常量,因此其后面的 VALUE 项不能省。定义常量的同时要给该常量赋

缺省值，否则将不能通过语法检查。如：

```
CONSTANTS c1(10)  TYPE  c.
```

就是错误的用法。

3. 文字

ABAP 中称固定值为文字(Literals)，ABAP 语言区别于文本型文字和数字型文字。

(1)文本文字是放在单引号内的字符序列，如：

```
'HELLO, SAP ABAP'
'Walldorf  678912'
```

如果某文本文字包含引号，则必须重复使用引号，这样使系统可以将引号内的内容识别为文本文字而不是字符串的标识，如：

```
WRITE: / 'That is Mary"s textbook'.
```

该语句生成下列输出：

```
That is Mary's textbook
```

(2)数字文字。是一个数字序列，前面可以包含“+”或“-”符号，如：

```
356       -38          +789
```

如果需要非整型数值或较长的数字，则应该使用文本文字。在运算时，该文本文字将自动被转换到正确的类型。如：

```
'+0.732'        '-8765.76'        '123456789012345678901'
```

跟其他编程语言一样，要使 ABAP 程序更加易于维护，尽量不要在程序中大量显式地使用文字，而应该尽量将它们定义为常量后再使用。

4.2.7 文本符号的创建与使用方法

1. 文本符号的使用方法

为了使编写出的 ABAP 程序能够获得多语言支持的特点，应该尽量少地在程序中直接使用文本型文字，而应将其定义为文本符号(Text Symbol)。文本符号是 ABAP 文字基本概念的一部分。文本符号的使用方法如图 4-4 所示。

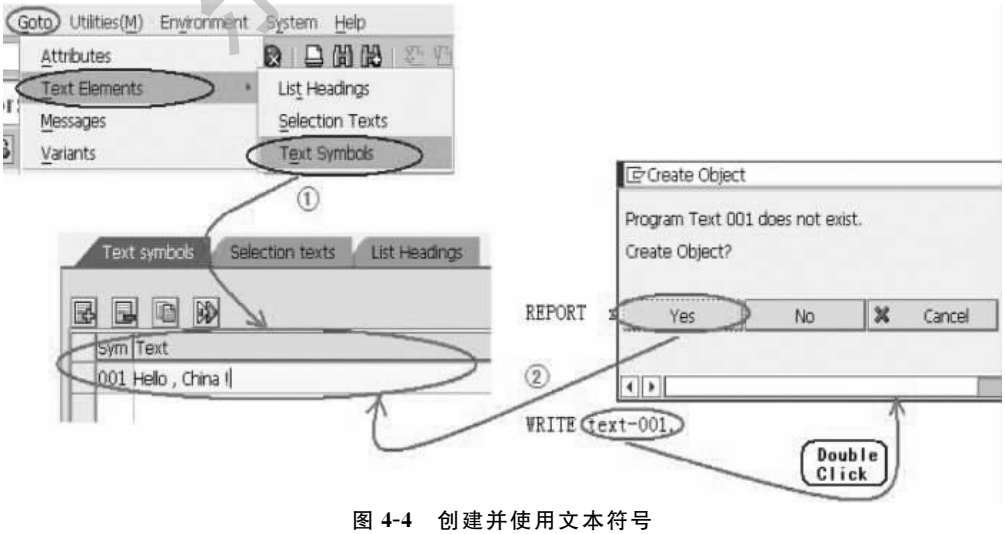

图 4-4 创建并使用文本符号

在 ABAP Editor(SE38)的程序编辑界面，选中系统菜单“Goto→Text Elements→Text Symbols”，如图中“①”所示。或者先在程序中直接使用一个不存在的文本符号，如：WRITE text-001，其中 001 是对文本符号的三位编号，text-001 是对该文本符号的引用。在程序编辑窗口的“text-001”上双击鼠标键，弹出确认创建的对话框，选“YES”，如图中“②”所示。“①”和“②”这两种方式都将进入一个文本符号的编辑窗口。在 Text 这一列中输入该文本符号所代表的字符串，然后在系统标准工具条上选择激活按钮，再选择“保存”按钮即可。

2. 文本符号的翻译方法

文本符号创建和使用的主要目的是使 ABAP 程序具有跨语言运行的能力，是编写出一个“国际化”的 ABAP 程序所必须掌握的技术。因此要为同一个文本符号创建多个语言的翻译版本，先在 ABAP Editor(SE38)的程序编辑界面，选中系统菜单“Goto→Translation”，在语言选择窗口选择目标语言为中文(代码 ZH)，点确定后弹出程序选择窗口，在对应的程序名称上双击鼠标。如图 4-5 所示。

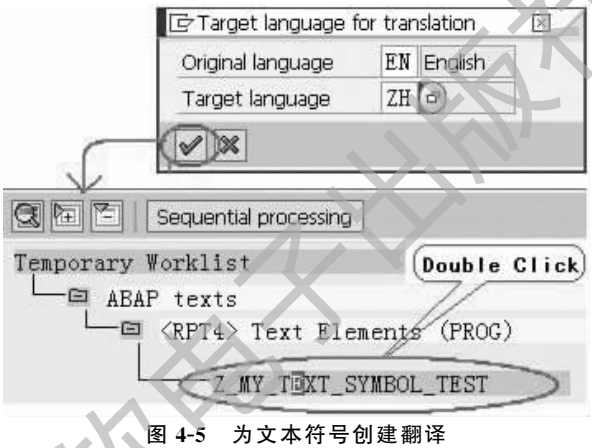

图 4-5　为文本符号创建翻译

弹出文本翻译的编辑窗口，在其中的文本输入域输入对应的中文翻译“你好，中国！”。如图 4-6 所示。

图 4-6　为文本符号输入翻译的文字

选择“保存”按钮保存录入的信息。这里需要注意的是，在使用该文本符号之前一定要将其激活。分别采用英文(EN)和中文(ZH)登录系统并运行该程序。界面表现如图 4-7 所示。

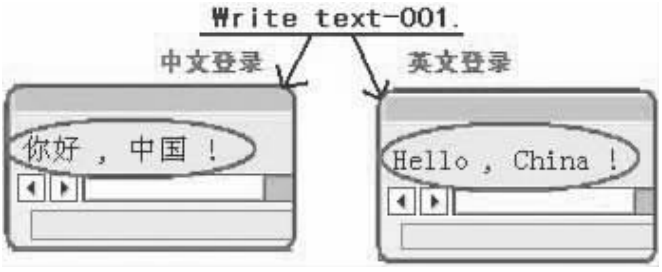

图 4-7 使用文本符号在不同语言环境下程序的运行效果

可见,使用文本符号确实可以创建出跨语言运行的 ABAP 程序。

4.2.8 系统定义的数据对象

系统定义的数据对象是系统预先定义好的数据对象,无需声明,可在程序中直接使用。其中系统全局结构体变量中的很多系统字段在编程中非常重要。所有系统字段都有格式为 SY-<fieldname>的名称,<fieldname>是该结构体中的单个字段。下面是较常用的系统字段的例子:

(1)SY-SUBRC:返回代码值(0 代表操作成功)。

(2)SY-UNAME:用户的登录名。

(3)SY-TCode:当前事务。

(4)SY-DATUM:当前日期。

(5)SY-UZEIT:当前时间。

在编程中一般不建议改变系统字段的值。

4.3 ABAP 语法元素

4.3.1 ABAP 语句

ABAP 程序由多条单个的 ABAP 语句构成,每条语句以关键字开头,以句号“.”结束。如:

```
PROGRAM  ZMY_TEST.
WRITE  'Hello World ! '.
```

该示例包含两条语句,每行一条。位于行首的 PROGRAM 和 WRITE 分别是对应语句的关键字。该程序运行后会生成一个列表屏幕,屏幕上会输出一行“Hello World !”。

4.3.2 ABAP 关键字分类

关键字是 ABAP 语句的第一个单词,它决定了整个语句的含义。ABAP 中有四种不同类型的关键字:

1. 声明型关键字

这些关键字用于定义数据类型,或定义程序可以访问的数据对象(即内存变量)。示例

如下：

```
TYPES,  DATA,  TABLES
```

声明型关键字在程序被编译时处理，而不是代码在运行时被执行。为清楚起见，应该在程序开头的“说明部分”指定所有声明型关键字。

2. 事件关键字

这些关键字用于在 ABAP 程序中定义处理块，事件处理块是当特定事件发生时进行处理的语句块。示例如下：

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

3. 程序流程控制关键字

这些关键字根据特定条件控制 ABAP 程序的流程走向。示例如下：

```
IF, WHILE, CASE
```

4. 操作关键字

操作关键字用于处理数据(由声明型关键字定义)。示例如下：

```
WRITE, MOVE, ADD
```

4.3.3　数据对象赋值

在 ABAP 中，可以通过声明语句或操作语句给数据对象赋值。

1. 通过声明性语句赋初始值

在声明语句中，在定义一个数据对象的同时将初始值赋给该数据对象，可以在 DATA 和 CONSTANTS 等语句中使用 VALUE 参数，在声明选择屏幕字段变量时，使用 default 为选择屏幕字段变量赋初始值。如：

```
CONSTANTS MY_STR(8) TYPE  c  VALUE 'HELLO'. "定义一个常量 MY_STR
DATA myid(10) TYPE c VALUE '100'.  "定义一个变量 myid.
PARAMETER mycarrid  TYPE  s_carr_id  DEFAULT'AA'. "定义选择屏幕变量 mycarrid
```

2. 通过操作性语句给数据对象赋值

要在操作语句中给数据对象赋值，可以使用 MOVE 语句，或使用对应的赋值运算符(=)，其语法为：

```
MOVE source TO destination.
```

如：

```
DATA: var1 TYPE i,
      var2 TYPE i.
MOVE 100 TO var1. "等价于 var1 = 100.
var2 = var1 + 10. "但不能使用 MOVE var1 + 10 TO var2.
```

其中的“=”两边要有空格。可见，对于基本的数据对象的赋值，采用“=”号更加灵活方便，而 MOVE 更多地用于复合类型的数据对象间的赋值，还有一个与 MOVE 类似的 MOVE-CORRESPONDING，功能是将两个结构体变量中的同名字段进行赋值。如：

```
TYPES: BEGIN OF airline,
       carrid   TYPE SCARR-carrid,
       carrname   TYPE SCARR-carrname,
END OF airline.
```

```
DATA: var1 TYPE SCARR,
      var2 LIKE var1,
      var3 TYPE airline.
var1-carrid =  'LH'.
var1-carrname =  'LH AirLine'.
var1-url =  'http://www.lhairling.com'.
MOVE var1 TO var2. "var1 与 var2 结构相同.
MOVE-CORRESPONDING var2 TO var3. "var2 与 var3 结构不完全相同.
```

3. 通过 CLEAR 操作使数据对象具有缺省值

将一个数据对象的值变成该对象类型的缺省值，如下列整形变量 var1：

```
DATA var1 TYPE i VALUE 10.
CLEAR var1. " var1 的值为缺省值 0.
```

4.3.4 数据对象运算

要处理数值型数据对象并将结果值赋给另外的数据对象，可以使用 COMPUTE 语句或直接用赋值运算符(=)。COMPUTE 语句的语法如下所示：

```
COMPUTE  <n>  =  <expression>.
```

其中关键字 COMPUTE 是可选的并且可以省略，通常都写成<n> = <expression>的形式。两条语句的效果相同。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中所述。如果运算结果与<n>的数据类型不同，则系统将自动进行类型转换。在算术表达式中，可用括号按任何排列组合进行嵌套，有效的算术表达式运算符有(假设 var1 与 var2 为整型，且 var1=10)：

(1)+(加法运算符)：var2 = var1 + 10. "var2 的值为 20。

(2)-(减法运算符)：var2 = var1 - 5. "var2 的值为 5。

(3)*(乘法运算符)：var2 = var1 * 10. "var2 的值为 100。

(4)**(乘方运算符)：var2 = var1 ** 3. "var2 是 10 的三次方，值为 1000。

(5)/(除法运算符)：其运算结果带四舍五入。如“9 / 2”的值为 5，“10 / 3”的值为 3。

(6)DIV(整除运算符)：直接取被除数被除以后的整数部分。如“9 DIV 2”的值为 4，“10 DIV 3”的值为 3。

(7)MOD(除余运算符)：直接取被除数被除以后的余数部分。如“7 MOD 2”的值为 1，“10 MOD 4”的值为 2。

上述所有操作符以及小括号都是 ABAP 语言的关键字，它们的前面和后面都应该有空格，否则会出错。这是 ABAP 语言不同于其他语言的地方。

ABAP 提供多个处理字符串类型数据对象的关键字，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。

4.3.5 处理字符串

字符串处理功能在任何编程语言中都是非常重要的内容。ABAP 语言中处理字符串的

方法有：

1. 连接字符串

在 ABAP 中，使用 CONCATENATE 语句将几个字符串连接成一个字符串，其语法如下：

```
CONCATENATE <s1>... <sn> INTO <s> [SEPARATED BY <c>].
```

该语句连接字符串<s1> ...与<sn>并将连接的结果赋给字符串<s>，该操作忽略尾部空格。子语句 SEPARATED BY <c>将指定的字符串<c>放置在将要被连接的相邻两个字符串之间，并且将按照字符串<c>的声明长度放置，不足的部分以空格补充。连接的结果如果能被字符串<s>定义的长度所容纳，则将 SY-SUBRC 的值设置为 0，如果<s>的长度不足，则连接的结果必须被截断，则 SY-SUBRC 的值被设置为 4。字符串连接的示例代码见程序 4-1。

程序 4-1　连接字符串

```
DATA: s1(5) TYPE c  VALUE  '2008',
      s2(8)  TYPE c VALUE  'BeiJing',
      s3(8) TYPE c  VALUE  'Olympic',
      s4(6) TYPE c VALUE  'Games',
      s5(35) TYPE c,
      s6(18) TYPE c,
      s7(35) TYPE c,
      sep(1) VALUE '-'.
CONCATENATE s1 s2 s3 s4 INTO s5.
WRITE: / 's5 = ', s5,
         'sy-subrc = ', sy-subrc.
CONCATENATE s1 s2 s3 s4 INTO s6.
WRITE: / 's6 = ', s6,
         'sy-subrc = ', sy-subrc.
CONCATENATE s1 s2 s3 s4 INTO s7 SEPARATED BY sep.
WRITE: / 's7 = ', s7,
        'sy-subrc = ', sy-subrc.
```

该程序的输出结果为三行数据，分别代表三种字符串的连接结果：

```
s5 = 2008BeiJingOlympicGames      sy-subrc = 0
s6 = 2008BeiJingOlympic           sy-subrc = 4
s7 = 2008-BeiJing-Olympic-Games   sy-subrc = 0
```

其中第一行输出结果的 sy-subrc = 0，表示目标字符串 s5 的长度足够容纳所有被连接字符串；而第二行的输出结果的 sy-subrc = 4，正好相反，表示目标字符串 s6 的长度不足以容纳所有被连接的字符串；第三行中的 s7 中间被加上了“-”进行分割。

2. 拆分字符串

ABAP 中使用 SPLIT 语句将一个母字符串拆分成多个子串，其语法如下：

```
SPLIT  <s>  AT  <sep>  INTO  <s1>... <sn>.
```

该语句在字符字段<s>中搜索分隔符字符串<sep>，并将分界符之前和之后的部分

分别放到目标字段<s1>... <sn>中。要将所有子部分都成功地放到不同的目标字段中，需要指定足够多的目标字符串，并且每个目标字符串的长度必须足够长；如果目标字符串的数量不够，则会将母串<s>的剩余部分填充在最后的目标字符串中，并且可能剩余部分仍然还包含了分隔符。

如果所有目标字段足够长而不必被截断任何部分，则将 SY-SUBRC 的值设置为 0；若有字段因为目标字段不够长而被截断，则将 SY-SUBRC 的值设置为 4。见程序 4-2。

程序 4-2 拆分字符串

```
DATA: p_str(60) VALUE  '2008-BeiJing-Olympic-Games',
s1(10),s2(15),s3(10),s4(10),sep(1) VALUE '-'.
WRITE:  'p_str = ',p_str.
SPLIT p_str AT sep INTO s1 s2 s3 s4.
WRITE: /  's1 = ',s1,
       /  's2 = ',s2,
       /  's3 = ',s3,
       /  's4 = ',s4,
       /  'sy-subrc = ',sy-subrc.
SPLIT p_str AT sep INTO s1 s2.
WRITE: /  's1 = ',s1,
       /  's2 = ',s2,
      /  'sy-subrc = ',sy-subrc.
```

该程序的输出结果如下：

```
p_str = 2008-BeiJing-Olympic-Games
s1 = 2008
s2 = BeiJing
s3 = Olympic
s4 = Games
sy-subrc = 0 "不存在任何被截断的子串，操作成功
s1 = 2008
s2 = BeiJing-Olympic
sy-subrc = 4  "由于存在被截断的子串，即 s2 的长度不足够容纳“2008-”后面剩余部分
```

也可以将母串被分割后的各个子串放到一个内表中(此处相当于一维数组)，如：

```
SPLIT  <s>  AT  <sep>  INTO  <itab>.
```

对于字符串的每一部分，系统将在内表中添加新行。示例代码见程序 4-3。

程序 4-3 拆分字符串到内表

```
DATA: s1 TYPE string,
      itab TYPE TABLE OF string, "相当于定义了一个一维数组
      text TYPE string VALUE `I'm learning ABAP programming`.
SPLIT text AT space INTO TABLE itab.
LOOP AT itab INTO s1.
      WRITE: / 'loop', sy-tabix, '=', s1. "sy-tabix 中的值表示当前内表行的索引
```

```
ENDLOOP.
```

运行结果是：

```
loop1 = I'm
loop2 = learning
loop3 = ABAP
loop4 = programming
```

将字符串分割后的各子串放到一个内表中，比放在多个字符串构成的列表中具有更加方便的优点。主要是因为内表中的行是动态扩展的，不需要定义大量的字符串，也不需要考虑定义的目标字符串数量不够的问题。

3. 搜索字符串

ABAP 中使用 SEARCH 语句按照特定模式搜索的子字符串，其语法如下：

```
SEARCH <s> FOR <sub>.
```

该语句在字段<s>中搜索<sub>中的字符串，如果搜索成功，则将 SY-SUBRC 的返回代码值设置为 0 并将 SY-FDPOS 的值设置为字段<s>中该字符串的偏移量。如不能找到子串，则将 SY-SUBRC 设置为 4。搜索子串 <sub> 可为下列格式之一：

(1)<pattern>　　搜索 <pattern>(任何字符顺序)，忽略尾部空格。

(2).<pattern>.　搜索 <pattern> ，但是不忽略尾部空格。

(3) * <pattern>　搜索以 <pattern> 结尾的单词。

(4)<pattern> *　搜索以 <pattern> 开头的单词。

搜索字符串的示例代码见程序 4-4。

程序 4-4　查找字符串

```
DATA str(30) VALUE 'SAP R3 is a management information system.'.
SEARCH str FOR 'BC'.
WRITE: / 'Search BC:', 'sy-subrc = ',sy-subrc,
                'sy-fdpos = ',sy-fdpos .
SEARCH str FOR 'R3'.
WRITE: / 'Search R3:', 'sy-subrc = ',sy-subrc,
                'sy-fdpos = ',sy-fdpos .
SEARCH str FOR '* mana'.
WRITE: / 'Search * mana :', 'sy-subrc = ',sy-subrc,
                'sy-fdpos = ',sy-fdpos .
SEARCH str FOR 'mana *'.
WRITE: / 'Search mana * :', 'sy-subrc = ',sy-subrc,
                'sy-fdpos = ',sy-fdpos .
```

该程序的输出如下：

```
Search BC:        sy-subrc = 4    sy-fdpos = 0
Search R3:        sy-subrc = 0    sy-fdpos = 4
Search * mana :   sy-subrc = 4    sy-fdpos = 0
Search mana * :   sy-subrc = 0    sy-fdpos = 12
```

4. 获得字符串长度

使用内部函数 STRLEN 获得一个字符串的长度，该长度从字符串的第一个字符开始，一直到最后一个非 SPACE 的字符，取得的长度相当于除掉字符串右边所有的空格后，剩余的字符数。其语法如下：

```
[COMPUTE] <n> = STRLEN( <s> ).
```

STRLEN 将操作数<s>作为字符数据类型处理。关键字 COMPUTE 可选。计算字符串长度的示例代码见程序 4-5。

程序 4-5 获取字符串的长度

```
DATA: i1 TYPE i,
      str1(10) VALUE 'abc def',
      str2(10),
      str3(20) VALUE ' 1 '.
i1 = STRLEN( str1 ).
  WRITE  i1.
i1 = STRLEN( str2 ).
  WRITE / i1.
i1 = STRLEN( str3 ).
  WRITE / i1.
```

该程序的运行结果分别是 7、0、2，其中在求 str3 的长度时，右边的空格不计，而左边的空格计算在内。

对于十六进制的字符串，使用 XSTRLEN 函数来求其长度，但是求出的长度是以字节为单位的。见程序 4-6。

程序 4-6 获取十六进制字符串的长度

```
DATA: i1 TYPE i,
     str1 TYPE string,
     xstr2 TYPE xstring.
  str1  = '123456ABCDEF'.
  xstr2 = '123456ABCDEF'.
  i1 = STRLEN( str1 ).
WRITE: 'Lenth of str1 is : ', i1.
  i1 = XSTRLEN( xstr2 ).
WRITE: / 'Lenth of xstr2 is : ', i1.
```

该程序的运行结果为：

```
Lenth of str1 is :     12
Lenth of xstr2 is :     6
```

可见，以十六进制求字符串“123456ABCDEF”的长度是 6(字节)，而不是 12 个字符。

5. 替换字符串内容

要用另外的字符串替换字符串的某些部分，应使用 REPLACE 语句。

```
REPLACE <s1> WITH <s2> INTO <src> [LENGTH <l>].
```

如果指定了长度<l>，则 ABAP 首先在<src>中搜索子串<s1>的前<l>个字符；

如果未指定长度＜1＞，则按＜s1＞的全长进行搜索；当第一次发现＜s1＞或＜s1＞的前＜1＞个字符，则＜s1＞或＜s1＞的前＜1＞个字符在字符串＜src＞中第一次出现的位置用字符串＜s2＞替换。如果指定了长度＜1＞，则只替换＜src＞中由长度＜1＞指定的长度。如果系统字段 SY-SUBRC 的返回代码为 0，则说明在＜src＞中找到了＜s1＞并且已用＜s2＞成功替换；SY-SUBRC 非 0 的返回值意味着没有子串被替换。字符串替换的代码例子见程序 4-7。

程序 4-7　替换字符串

```
DATA: tmp(30) VALUE 'Hello world !   ',
      s LIKE tmp,  s1(5) VALUE 'world',  s2(10) VALUE 'dear kitty'.
  s = tmp.
WRITE s.
REPLACE s1 WITH s2 INTO s.
WRITE / s.
  s = tmp.
REPLACE s1 WITH s2 INTO s LENGTH 3. "只替换“Hello world !”中“wor”这部分
WRITE / s.
```

该程序的运行结果为：

```
Hello world !
Hello dear kitty !
Hello dear kittyld !
```

REPLACE 语句只替换目标字符串中第一次出现的子串，要想将目标字符串中所有重复的部分全部都用另外一个字符串替换掉，可以采用 WHILE 循环进行多次替换。

6. 字符串的大小写转换

ABAP 中使用 TRANSLATE 语句进行字符串的大小写转换。语法如下：

(1)TRANSLATE ＜s＞ TO UPPER CASE. ：将字符串转换成大写。

(2)TRANSLATE ＜s＞ TO LOWER CASE. ：将字符串转换成小写。

另外还可以使用替换规则将字符串 s 中的字母转换，使用替换规则的语法如下：

```
TRANSLATE <s> USING <r>.
```

该语句根据＜r＞中的规则将字符串＜s＞中相应字符替换。规则＜r＞中包含成对的字母，对于其中的每一对字母中的第一个字母，应该用其后的第二个字母在目标字符串＜s＞中进行替换。字符串大小写转换以及规则替换的代码示例见程序 4-8。

程序 4-8　字符串大小写转换与规则替换

```
DATA: t(16) VALUE 'I loVE aBaP',  s LIKE t,  r1(20) VALUE 'AbBcEfIjLmOpPqVw',
      r2(20) VALUE 'bAcBfEjImLpOqPwV'.
  s=t.
TRANSLATE s TO LOWER CASE.
WRITE / s.
  s=t.
TRANSLATE s TO UPPER CASE.
WRITE / s.
```

```
TRANSLATE s USING r1.
WRITE / s.
TRANSLATE s USING r2.
WRITE / s.
```

该程序的运行结果为：

```
i love abap
I LOVE ABAP
j mpwf bcbq
I LOVE ABAP
```

该程序中使用了 4 个 TRANSLATE 语句。其中第一个将“I loVE aBaP”转换成小写，第二个 TRANSLATE 语句将“I loVE aBaP”转换成大写；第三个 TRANSLATE 语句将“I LOVE ABAP”中的每个字母都转换成与该字母的下一个字母所对应的小写字母，即“j mpwf bcbq”，使字面意义不容易被人看懂，具有最简单的加密功能；而第四个 TRANSLATE 语句则又将“j mpwf bcbq”重新还原回原来的内容，其中第三和第四个 TRANSLATE 语句都使用了替换规则。

4.4 ABAP 程序的流程控制

如要根据一定条件执行 ABAP 程序块，或要重复执行某语句块，应该使用 ABAP 语言提供的用于程序流程控制的相关标准关键字。在进行程序流程控制的过程中，要使用逻辑表达式作为判断的条件，根据逻辑表达式的结果为真还是为假来决定程序的具体走向。

4.4.1 逻辑表达式

ABAP 逻辑表达式的一般语法为：

```
... <v1>  <operator>  <v2> ...
```

该表达式比较两个数据对象 v1 和 v2，比较的结果不为真就为假。在带关键字 IF，CHECK 和 WHILE 的条件语句中可使用逻辑表达式。根据操作数<v1>和<v2>的数据类型，可以使用不同的逻辑运算符。

1. 比较所有基本类型的数据对象

要比较所有的数据对象，可以在逻辑表达式中使用表 4-3 中的运算符。

表 4-3　　逻辑表达式中的比较运算符

运算符	含义
EQ 或 =	等于
NE 或 <> 或 ><	不等于
LT 或 <	小于
LE 或 <=	小于等于
GT 或 >	大于
GE 或 >=	大于等于

表 4-3 中操作符所对应的操作数可以是基本类型的变量、常量或文字。

2. 比较字符串

使用表 4-4 中所示的字符型逻辑表达式来判断两个字符串之间的包含关系，其语法为：

```
... s1 <operator> s2 ...
```

被比较的两个操作数 s1 和 s2 只能是字符串型(类型 C)或数字文本(类型 N)。见表 4-4。

表 4-4　　逻辑表达式中的比较运算符

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

其中 CO、CN、CA 和 NA 在比较时要区分大小写，并且比较时要包含字符串尾部的空格；而 CS、NS、CP 和 NP 在比较时不区分大小写，且比较时忽略字符串尾部的空格。这些字符串运算符的功能如下。

(1)s1 CO s2

如果 s1 仅包含 s2 中的字符，则逻辑表达式为真。该比较运算符区分字符的大小写，并且包含尾部空格。如果比较结果为真，则系统字段 SY-FDPOS 包括 s1 的长度。如果为假，则 SY-FDPOS 包含 s1 中第一个未在 s2 内出现的字符的偏移量。代码示例见程序 4-9。

程序 4-9　字符串仅包含

```
DATA : s1 TYPE  string,
       s2 TYPE  string.
s1 = 'I Love ABAP! '.
s2 = 'ABcDILvXeZoP ! '.
IF s1 CO s2 .
  WRITE : / 'all chars in s1 can be found in s2! '.
ELSE.
  WRITE : sy-fdpos.
ENDIF.
```

在程序 4-9 中，字符串 s1 中的所有字符都能在 s2 中找到，因此比较结果为真。输出结果为：

```
all chars in s1 can be found in s2!
```

(2)s1 CN s2

如果 s1 中还包含 s2 之外的其他字符，则逻辑表达式为真。该比较运算符也区分大小

写,并且尾部空格也在比较的范围内。如果比较结果为真,则系统字段 SY-FDPOS 的值是 s1 中第一个未在 s2 中出现的字符的偏移量;如果为假,SY-FDPOS 包含 s1 的长度。该表达式的代码示例请参照程序 4-9 并稍做修改即可。

(3)s1 CA s2

如果 s1 中至少包含 s2 中的一个字符,则逻辑表达式 s1 CA s2 的值为真。该比较运算符区分大小写。如果比较的结果为真,则系统字段 SY-FDPOS 的值是 s1 中第一个也在 s2 中出现的字符的偏移量;如果为假,则 SY-FDPOS 的值是 s1 的长度。该表达式的代码示例请参照程序 4-9 并稍做修改即可。

(4)s1 NA s2

如果 s1 中不包含 s2 中的任何字符,则逻辑表达式 s1 NA s2 的值为真。该比较运算符区分大小写。如果比较的结果为真,则系统字段 SY-FDPOS 的值为 s1 的长度;如果为假,则 SY-FDPOS 的值为 s1 中在 s2 内出现的第一个字符的位置。该表达式的代码示例请参照程序 4-9 并稍做修改即可。

(5)s1 CS s2

如果 s1 中包含字符串 s2,则逻辑表达式 s1 CS s2 为真。忽略尾部空格并且该比较运算符不区分大小写。如果比较结果为真,则系统字段 SY-FDPOS 包含 s2 在 s1 中的偏移量;如果为假,SY-FDPOS 包含 s1 的长度。代码见程序 4-10。

程序 4-10 字符串包含

```
DATA : s1 TYPE  string,
       s2 TYPE  string.
s1 = 'I Love ABAP! '.
s2 = 'aBaP '.
IF s1 CS s2 .
  WRITE : / 's2 can be found in s1!, the position = ', sy-fdpos.
ELSE.
  WRITE : / 'can not find s2 in s1'.
ENDIF.
```

该程序中,字符串 s1 中包含了“ABAP”,字符串 s2 中也包含了“aBaP”,但是两者的大小写不一致,并且 s2 的尾部还包含空格。而 s1 CS s2 的结果仍然为真,表明 s1 包含了 s2,并且比较的过程中忽略了字母的大小写以及尾部空格。该程序的运行结果为:

```
s2 can be found in s1!, the position =  7
```

(6)s1 NS s2

如果 s1 中不包含字符串 s2,则逻辑表达式 s1 NS s2 为真,该比较忽略尾部空格且比较不区分大小写。如果比较为真,则系统字段 SY-FDPOS 的值为 s1 的长度,如果为假,则系统字段 SY-FDPOS 包含 s2 在 s1 中的偏移量。代码示例请参考程序 4-9。

(7)s1 CP s2

如果 s1 中包含模式 s2,则逻辑表达式 s1 CP s2 为真;如果 s2 属于类型 C,则可以在 s2 中使用如下通配符:

* 用于替代任何字符串

＋ 用于替代任何单个字符

该比较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 s2 在 s1 中的偏移量。如果为假，SY-FDPOS 包含 s1 的长度。代码示例见程序 4-11。

程序 4-11　字符串包含——使用通配符

```
DATA : s1  TYPE  string,
       s2(4) TYPE  c.
s1 = 'I Love ABAP! '.
s2 = '*bA*'.
IF s1 CP s2 .
  WRITE : / 's2 can be found in s1!, the position = ', sy-fdpos.
ELSE.
  WRITE : / 'can not find s2 in s1'.
ENDIF.
```

上述代码中“s2 = '*bA*'”中的 * 代表任何字符，而 s1 中包含这个模式，因此 s1 CP s2 比较的结果为真，该程序的输出结果为：

```
s2 can be found in s1!, the position =  8
```

(8)s1 NP s2

如果 s1 中不包含模式 s2，则逻辑表达式 s1 NP s2 为真。在 s2 中，可以使用与 CP 相同的通配符(*或＋)，该比较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 的值为 s1 的长度，如果为假，则 SY-FDPOS 中的值为 s2 在 s1 中的偏移量。

3. 检查字段是否属于某一范围

ABAP 使用带有 BETWEEN 参数的逻辑表达式来检查某个值是否属于一个特定范围。其语法为：

```
... <v1>  BETWEEN  <v2>  AND  <v3> ...
```

如果<v1>的值在<v2>和<v3>之间(含<v1>和<v2>)，则表达式为真。它是下列逻辑表达式的短格式：

```
IF  <v1>  GE  <v2>  AND  <v1> LE  <v3> .
```

代码示例见程序 4-12。

程序 4-12　检查一个值是否在一个区间中

```
DATA: num TYPE  i  VALUE 6.
IF  num  BETWEEN 2  AND  8.
  WRITE : 'num ', num , 'between 2 AND 8 is true'.
ELSE.
  WRITE : 'num ', num , 'between 2  AND 8 is false'.
ENDIF.
```

在该程序中，如果 num 的值在 2 和 8 之间，则表达式“num BETWEEN 2 AND 8”的值为真，该程序的输出结果是：

```
num  6  between  2  AND  8  is  true
```

4. 检查字段的初始值

要检查字段是否设置为初始值，应使用带有 IS INITIAL 参数的逻辑表达式，其语法为：

```
... <v> IS INITIAL ...
```

如果<v>的值是其数据类型对应的初始值，则表达式为真。一般情况下，任何字段，包括基本的或结构化的字符串(或内表)，在“CLEAR <v>”语句执行后，<v>中都包含其初始值。见程序 4-13。

程序 4-13 检查一个数据对象否为初始值

```
DATA str(10) VALUE 'Hello'.
IF str IS INITIAL.
  WRITE / 'str is initial'.
ELSE.
  WRITE / 'str is not initial'.
ENDIF.
CLEAR str.
IF str IS INITIAL.
  WRITE / 'str is initial'.
ELSE.
  WRITE / 'str is not initial'.
ENDIF.
```

该程序将产生如下输出：

```
str is not initial
str is initial.
```

当字符串 CLEAR str 以后，str 的值变成字符类型的初始值。

5. 组合逻辑表达式

可以使用逻辑连接运算符 AND、OR 和 NOT，将几个逻辑表达式组合为一个表达式。

(1)用 AND 连接：要将几个逻辑表达式组合为一个表达式，且该表达式中仅当其所有的子表达式为真时总结果才为真，则表达式之间要用 AND 连接。

(2)用 OR 连接：要将几个逻辑表达式组合为一个表达式，且只要其中的某一个子表达式为真时，该表达式即为真，则表达式之间要用 OR 连接。

(3)用 NOT 取反：要将逻辑表达式的结果在真和假之间转换，则应在该表达式前面加 NOT。

它们之间的优先级为：NOT 优先于 AND，AND 优先于 OR。建议使用任何小括号组合指定的处理顺序，且小括号的前面和后面都必须要有空格。

ABAP 从左到右处理逻辑表达式。如果确定组合表达式之一是真或者假，就不再执行该组合条件中其余的比较，因此应该尽量将为假的子表达式放在 AND 链的开头，将为真的子表达式放在 OR 链的开头，而将很费时的表达式，如字符串查找或比较等操作放到最后。组合逻辑表达式的代码示例见程序 4-14。

程序 4-14 组合逻辑表达式

```
DATA: i1 TYPE i VALUE  100,
      i2  TYPE i VALUE 200,
```

```
      i3  TYPE i VALUE 300,
      result(6).
IF( i1 <= i2 )  AND  ( i3 > i1 )  AND  ( i3 > i2 ).
  result = 'TRUE'.
ELSE.
  result = 'FALSE'.
ENDIF.
WRITE : / 'The following logical expression is :', result.
```

该程序运行将产生如下输出：

```
The following logical expression is TRUE:
```

在本例中，IF 语句使用了用 AND 连接起来的三个子逻辑表达式，若最后总结果为真，则这三个子表达式都被判断而且结果都为真。

4.4.2　使用条件分支语句

ABAP 语言提供了流程控制语句，可以在程序中通过流程控制关键字和条件表达式来决定程序的流程。主要有条件分支语句和循环语句。

1. 使用 IF 的条件分支

IF 语句允许根据指定的条件将程序流程跳转到特定的语句块中。该语句包括 IF 语句以及其后面的 ELSEIF，ELSE 和 ENDIF。IF 的语法为：

```
IF <condition1>.
   <statement block1>
ELSEIF <condition2>.
   <statement block2>
ELSEIF <condition3>.
   <statement block3>
...
ELSE.
   <statement blockn>
ENDIF.
```

由 IF…ELSEIF…ELSE…ENDIF. 形成的多个流程分支中，有且仅有一个分支得到执行，即各分支之间是互相排斥的，只有前面分支的条件是假的时候，后面的分支才有可能得到判断并且执行。如果 IF 或 ELSEIF 所有的条件都为假，则执行 ELSE 开始的语句块。最后的语句块必须用 ENDIF 结束。可以使用任何逻辑表达式作为 IF 和 ELSEIF 语句中的条件。ABAP 语言允许 IF…ENDIF 语句块之间进行嵌套，其嵌套层次无限制，但是所有的 IF-ENDIF 语句必须在一个 ABAP 事件中完成，不可以跨事件。使用 IF 进行程序流程分支控制的代码示例见程序 4-15。

程序 4-15　使用 IF 进行程序流程的分支控制

```
PARAMETERS  week_num  TYPE  i  DEFAULT  1.
    DATA  week_word  TYPE  string.
    IF week_num EQ 1.
```

```
    week_word = 'Monday'.
  ELSEIF  week_num EQ 2.
    week_word = 'Tuesday'.
  ELSEIF  week_num EQ 3.
    week_word = 'Wednesday'.
  ELSEIF  week_num EQ 4.
    week_word = 'Thursday'.
  ELSEIF  week_num EQ 5.
    week_word = 'Friday'.
  ELSEIF  week_num EQ 6.
    week_word = 'Saturday'.
  ELSEIF  week_num EQ 7.
    week_word = 'Sunday'.
  ELSE.
    week_word = 'an invalid week number! '.
  ENDIF.
  WRITE : 'The week you input is [',week_word, ']'.
```

该程序的功能是根据用户在选择屏幕上输入的星期号码(week_num),产生该星期号码对应的是星期几的英文单词,如用户分别在选择屏幕界面上输入数字 7 和 8,则程序两次执行的结果分别是:

```
The week you input is [Sunday]
The week you input is [an invalid week number!]
```

在这样的具有 ELSE 从句的 IF 语句中,有且仅有一个程序分支得到了执行。

2. 使用 CASE 的条件分支

ABAP 语言的 CASE 语句用于根据一个指定的变量的内容而执行不同的语句块。其语法如下:

```
CASE  <v>.
  WHEN <v1>.
       <statement block1>
  WHEN <v2>.
       <statement block2>
  WHEN <v3>.
       <statement block3>
  WHEN...
...
  WHEN OTHERS.
       <statement blockn>
ENDCASE.
```

如果 CASE 后面的变量<v>的值与后面多个 CASE 中的某一个<vi>的值相等,则系统执行对应的 WHEN 语句之后的语句块,执行完一个 WHEN 分支后,程序流程直接跳出 CASE…ENDCASE 范围,继续处理 ENDCASE 语句后面的语句;如果变量<v>中的值不

和下面多个＜vi＞中的任何一个相等，则执行 WHEN OTHERS 分支中的语句块，CASE 语句的最后必须以 ENDCASE 结束。ABAP 语言的多分支处理语句 CASE 其实就是上述的 IF…ELSEIF…ELSE…ENDIF. 的一种简短写法，与下面的写法等价：

```
IF ＜v＞ = ＜v1＞.
   ＜statement block1＞
ELSEIF ＜v＞ = ＜v2＞.
   ＜statement block1＞
ELSEIF ＜v＞ = ＜v3＞.
   ＜statement block1＞
ELSEIF ＜v＞ =...
...
ELSE.
   ＜statement blockn＞
ENDIF.
```

在 ABAP 语言中，CASE…ENDCASE 语句之间、CASE…ENDCASE 语句与 IF-ENDIF 语句之间相互可以嵌套，但不能跨事件块，即必须在同一程序块中处理完成。见程序 4-16。

程序 4-16　使用 CASE 语句进行 ABAP 程序流程的多分支控制

```
PARAMETERS week_num TYPE i DEFAULT 1.
DATA week_word TYPE string.
CASE week_num .
  WHEN 1.
    week_word = 'Monday'.
  WHEN 2.
    week_word = 'Tuesday'.
  WHEN 3.
    week_word = 'Wednesday'.
  WHEN 4.
    week_word = 'Thursday'.
  WHEN 5.
    week_word = 'Friday'.
  WHEN 6.
    week_word = 'Saturday'.
  WHEN 7.
    week_word = 'Sunday'.
  WHEN OTHERS.
    week_word = 'an invalid week number! '.
  ENDCASE.
  WRITE : 'The week you input is [',week_word, ']'.
```

程序 4-16 与程序 4-15 完成的功能完全一致，但是在写法上却简化了很多。若用户分别在界面上输入数字 1 和 7，则程序两次执行的结果分别是：

```
The week you input is [Monday]
```

```
The week you input is [Sunday]
```

4.4.3 使用循环语句

1. 使用 DO 的无条件循环

如果想多次重复地执行某 ABAP 语句块,可以使用如下的 DO 循环语句,其语法为:

```
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 DO…ENDDO 之间的语句块。其中<n> TIMES 用于限制循环的最大次数,可以是变量或常量。如果<n>的值小于等于 0,则系统不执行该循环。系统字段 SY-INDEX 中包含已处理过的循环次数。使用 DO 循环语句时要注意避免产生死循环,若没有使用 TIMES 选项,则在语句块中至少应包含一个可以被执行到的 EXIT、STOP 或 REJECT 语句,以便系统能够退出循环。程序 4-17 显示 DO 循环的基本格式。

程序 4-17 使用 DO 循环语句求整数的累加和

```
PARAMETERS max TYPE i DEFAULT 100.
DATA sum TYPE i.
DO.
  IF sy-index > max.
    EXIT.
  ENDIF.
  sum = sum + sy-index.
ENDDO.
WRITE : 'sum = ',sum.
```

该程序的功能是根据用户在选择屏幕上输入的整数变量 max,然后求从 1 到 max 之间所有正整数的和并且输出。若用户在屏幕的 max 字段中输入 100,则该程序的输出结果为:

```
sum = 5050
```

由于 DO 型循环本身不带循环结束的条件,因此在循环体的内部加入了语句:

```
IF sy-index > max.
    EXIT.
  ENDIF.
```

来结束循环。

DO 循环之间以及 DO 循环与其他循环之间可以任意嵌套,还可以使用循环次数限制 TIMES 来使用 DO 循环。见程序 4-18。

程序 4-18 使用 DO n TIMES 循环语句求整数的累加和

```
PARAMETERS max TYPE i DEFAULT 100.
DATA sum TYPE i.
DO  max  TIMES.
  sum = sum + sy-index.
ENDDO.
```

```
WRITE : 'sum = ',sum.
```

程序 4-18 的功能与程序 4-17 完全一致，但是此处的 DO 循环中使用了“DO　max TIMES.”语句，限制了循环的最大次数，因此在循环体中就不一定要用 EXIT 这样的退出语句来结束循环。由于 ABAP 语言中不提供其他语言中基本都有的 for 循环，因此 DO n TIMES 循环其实就相当于其他语言中的 for 循环。若用户在屏幕的 max 字段中输入 10，则该程序的输出结果为：

```
sum = 55
```

还可以使用 VARYING 选项在每次循环中给变量<f>重新赋值。<f1>,<f2>,<f3>等必须是内存中类型和长度都相同的等距字段序列。第一次循环中，先将<f1>赋给<f>，第二次循环中，将<f2>分配给<f>，依此类推。可以在一个 DO 语句中使用多个 VARYING 选项。应保证循环次数不超过涉及的变量<f1>,<f2>和<f3>的数量。程序 4-19 说明如何在 DO 循环中使用 VARYING 选项。

程序 4-19　使用 DO　n　TIMES　VARYING 循环语句

```
DATA: BEGIN OF str_struc,
  s1(8) VALUE 'We',
  s2(8) VALUE 'are',
  s3(8) VALUE 'learning',
  s4(8) VALUE 'ABAP! ',
END OF str_struc.
DATA: s_tmp(8).
DO 4 TIMES VARYING s_tmp FROM str_struc-s1 NEXT str_struc-s2.
  WRITE s_tmp.
ENDDO.
```

该程序的输出为：

```
We are learning ABAP!
```

字段串 str_struc 代表内存中的四个等距的、同类型的字段序列。当执行第 n 次 DO 循环时，则会将 str_struc 中第 n 个字段的值赋给 s_tmp。

2. 使用带条件的 WHILE 循环

ABAP 中，WHILE 循环语句用于只要条件为真，就一直执行循环体中的语句。其语法如下：

```
WHILE <condition> [VARY <f> FROM <f1> NEXT <f2> ].
            <statement block>
ENDWHILE.
```

只要<condition>逻辑表达式的值为真，系统将一直执行在 WHILE…ENDWHILE 之间的语句块，直到<condition>逻辑表达式的值为假或者在循环体中执行到 EXIT、STOP 或 REJECT 语句循环才会结束。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，也可与其他循环嵌套使用。WHILE 语句的 VARY 选项与 DO 循环的 VARYING 选项工作方式一样，具体用法请参考程序 4-19。使用 WHILE 语句也应当注意避免编程不当出现死循环，在执行的某次循环上，一定要使 WHILE 语句的循环条件变为假，或者在循环体中能够执行到 EXIT、STOP 或 REJECT 语句，从而结束循环。见程序

4-20。

程序 4-20 使用 WHILE 条件循环语句

```
PARAMETERS max TYPE i DEFAULT 100.
DATA sum TYPE i.
WHILE  sy-index <= max .
  sum = sum + sy-index.
ENDWHILE.
WRITE : 'sum = ',sum.
```

该程序的功能还是求 1 到指定的 max 中的所有整数的和，如用户指定了 20，则输出结果为：

```
sum  =  210.
```

3. 终止循环

ABAP 中可以使用下列语句来终止循环：

(1)CONTINUE：无条件终止本次循环过程；

(2)CHECK：有条件终止本次循环过程；

(3)EXIT：完全终止循环。

其中的 CONTINUE 语句用于终止本次循环，继续下一次循环，只能在循环中使用。但关键字 CHECK 和 EXIT 还可以在循环外使用，如，可以终止子程序或整个程序块。EXIT 结束循环的用法上文中已经讲述，下面重点介绍如何使用 CONTINUE 和 CHECK 来终止循环。

CONTINUE 语句用于无条件终止本次循环，继续进行下一次循环。见程序 4-21。

程序 4-21 使用 CONTINUE 结束本次循环

```
DATA: sum TYPE i,
      max LIKE sum VALUE 100.
WHILE  sy-index <= max.
DATA itmp TYPE i.
  itmp = sy-index MOD 2 .
    IF itmp <> 0.
       CONTINUE.
    ENDIF.
  sum = sum + sy-index.
ENDWHILE.
WRITE : 'Even number from 1 to 100 , sum = ',sum.
```

该程序的功能是求从 1 到 100 之间的所有偶数的和，在程序中，做了从 1 到 100 之间的循环，在每次遇到奇数次循环的时候，则调用 CONTINUE 语句中断本次循环，进入下一次偶数次循环。该程序的运行结果是：

```
Even number from 1 to 100 , sum = 2550
```

CHECK 语句用于有条件地终止本次循环，继续进行下一次循环。其语法为：

```
CHECK <condition>.
```

如果<condition>逻辑表达式的值是假，则跳过当前语句块中所有剩余语句块，继续后

面的循环过程。见程序 4-22。

程序 4-22　使用 CHECK 结束本次循环

```
DATA: sum TYPE i,
      max LIKE sum VALUE 100.
WHILE  sy-index <= max.
  DATA itmp TYPE i.
  itmp = sy-index MOD 2 .
  CHECK itmp = 0.
  sum = sum + sy-index.
ENDWHILE.
WRITE : 'Even number from 1 to 100 , sum = ',sum.
```

程序 4-22 实现的功能与程序 4-21 完全一致。

4. ABAP 循环类型总结

在 ABAP 语言所支持 DO…ENDDO,DO n TIMES…ENDDO,WHILE…ENDWHILE 三种循环中,可以采用系统变量 SY-INDEX 来作为到当前循环为止已经发生的循环次数,而且即使是在嵌套的循环中,SY-INDEX 照样会在各个层次的循环中正确地表示出当前循环的次数。另外,在 ABAP 中,还有两种用于专门用途的循环语句,即:

(1)LOOP…ENDLOOP 循环:用于依次处理内表(Internal Table)的行;

(2)SELECT…ENDSELECT 循环:用于从数据表中循环地读取数据。

这两种循环的用法将在本书其他章节中具体讲述。

4.5　发送对话消息(Dialog Messages)

在 ABAP 中,使用 MESSAGE 语句向程序的用户发送对话消息,其语法如下:

```
MESSAGE  tnnn(message_class)  [ WITH  v1  [ v2 ]  [ v3 ]  [ v4 ]  ].
```

使用 MESSAGE 语句发送消息,必须在程序中指定一个三位的消息编号 nnn 以及消息的类别 message_class,用于确定具体的消息;如果消息的文本中具有占位符(palceholders),并且在 MESSAGE 语句中指定了 WITH 子句,则可以将 WITH 子句后面的数据对象"v1,v2…"的值从左到右依次替换到消息文本的占位符中,同时也必须指定一个消息的类型 t,用来表明消息的发送方式。消息的类型 t 的取值见表 4-5。

表 4-5　　**Dialog Message 的几种类型**

类型 t	类型描述	对话行为	消息显示位置
I	消息提示	程序执行被对话框中断,用户选择后程序继续	模态对话框
S	消息提示	程序执行不中断	下一个屏幕的状态条上
W	警告	与程序的上下文有关	模态对话框或状态条
E	错误	与程序的上下文有关	模态对话框或状态条
A	终止	程序被取消	模态对话框
X	异常堆栈	运行期错误	屏幕显示异常的堆栈信息

对话消息的演示代码见程序 4-23。

程序 4-23 发送对话消息

```
PARAMETERS msg_type TYPE c.
WRITE : / 'This code example is to show how to use the Message clause'.
CASE msg_type.
  WHEN 'i' OR 'I'.
MESSAGE i790(bc400) WITH 'i'  'Modal dialog box'.
WHEN 's' OR 'S'.
    MESSAGE s790(bc400) WITH 's'  'Status bar next screen'.
  WHEN 'w' OR 'W'.
    MESSAGE w790(bc400) WITH 'w'  'Waring,Status bar '.
  WHEN 'e' OR 'E'.
    MESSAGE e790(bc400) WITH 'e'  'Error,Status bar '.
  WHEN 'a' OR 'A'.
MESSAGE a790(bc400) WITH 'a'  'Termination,Modal dialog box'.
  WHEN 'x' OR 'X'.
    MESSAGE x790(bc400) WITH 'x'  'Short dump'.
  WHEN OTHERS.
    WRITE : / 'You input an invalid message type! '.
ENDCASE.
```

在上述程序中,首先通过 PARAMETERS 声明了一个屏幕输入字段 msg_type,用户在其中输入了一个合法的 Message Type 后点击“确定”,程序将会根据用户输入的值进行不同类型的消息演示。其中用到的消息语句如:

```
MESSAGE i790(bc400) WITH 'i'  'Modal dialog box'.
```

是采用一个模态对话框,将消息类别为 bc400,消息号码为 790 的消息取出来显示,其中 WITH 后面的两个文字分别用于替换消息中的两个占位符 &1 和 &2。若用户输入了“i”,则对应的显示结果如图 4-8 所示。

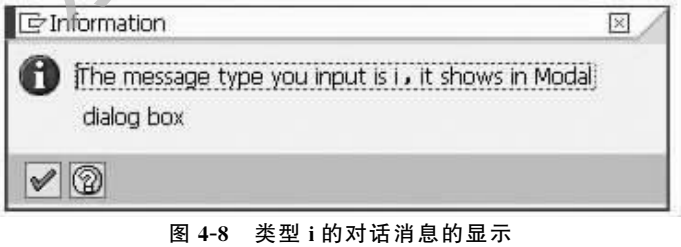

图 4-8 类型 i 的对话消息的显示

若 bc400 中的 790 号消息不存在,则可以按照如下的顺序创建。

(1)在 ABAP Editor(SE38)的代码编辑窗口,在一个不存在的消息号上双击鼠标左键,弹出是否创建新对象的提示对话框,选择“Yes”按钮。如图 4-9 所示。

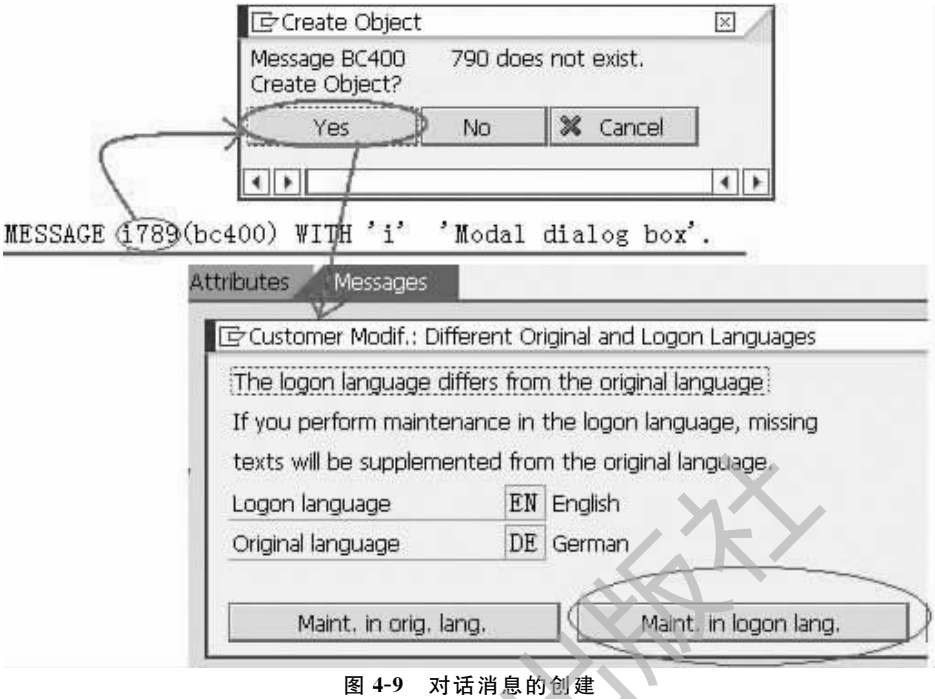

图 4-9　对话消息的创建

(2)在弹出的窗体中,选择“Maint. In logon lang.”,如图 4-9 所示,即采用系统登录所采用的语言来编写消息的文本。如图 4-10 所示。

图 4-10　对话消息文本的创建

在录入文本的过程中如果需使用占位符,要采用 &1,&2 的形式,录入完消息文本后,选择保存按钮将消息保存。然后运行程序,在用户选择屏幕上输入相应的各种消息类型观察该程序的运行效果。另外,还可以不用配置 MESSAGE 所属于的类别,直接在程序中编程向界面弹出一个对话框显示信息。其语法为:

```
MESSAGE　message_text　TYPE　<msg_type>.
```

如:

```
MESSAGE'Hello Kitty！' TYPE 'I'.
```

显示结果如图 4-11 所示。

图 4-11　程序直接发送的消息对话框

4.6 调试 ABAP 程序的方法

4.6.1 进入调试模式

任何语言的程序员在编程的时候都离不开编程工具提供的调试(Debug)功能,代码调试功能使程序员可以跟踪内存变量状态的变化以及程序流程的实际走向,为程序员做出正确的判断从而为解决程序中的问题起到非常关键的作用。一般特定编程语言的工具中都集成了调试的工具。在 ABAP 编程中,Object Navigator(SE80)中提供了几种方法使程序可以进入调试模式下运行(Debugging Mode),如图 4-12 所示。

(1)可以在 Object Navigator 左边的导航区域,选中一个程序,然后点击鼠标右键,在其弹出的上下文菜单上选择"Exeute→Debugging"。

(2)将一个要被调试的 ABAP 程序激活,在代码的编辑区域选中一行代码,然后在 Object Navigator 应用工具条上点击设置"断点"按钮,再直接通过 F8 或者在该程序的上下文菜单上选择"Execute→Direct"直接启动该程序就可以了。

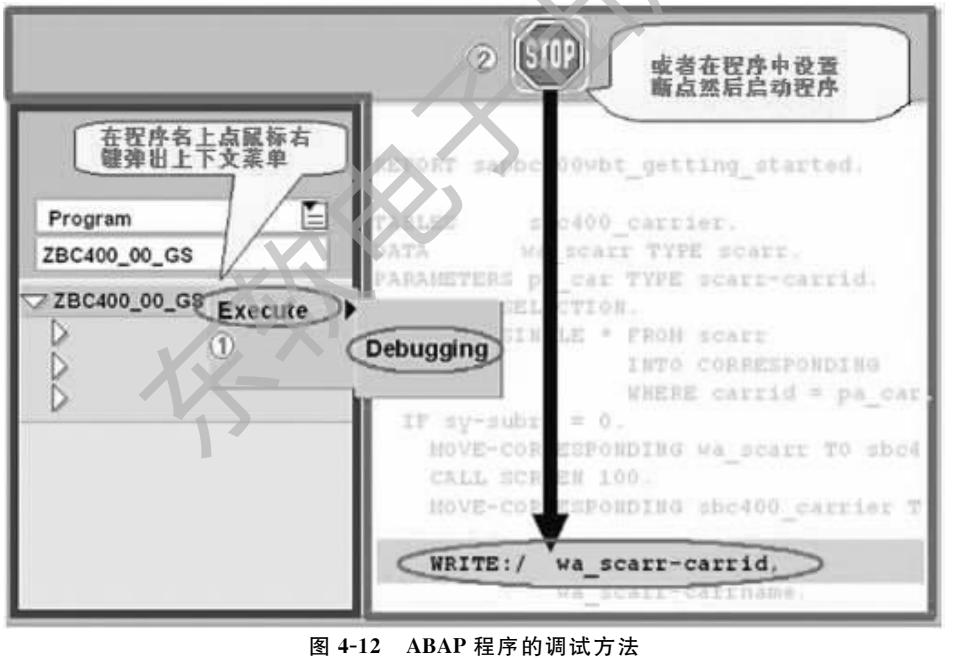

图 4-12 ABAP 程序的调试方法

4.6.2 在程序运行过程中进入调试模式

如果想调试程序中的一个特定的功能(如执行一个功能按钮或菜单项等),可以在程序运行期进行切换进入调试状态:先直接运行该程序,然后在程序中断等待用户界面交互的时刻,切入调试模式。有两种方法可以在运行期间进入调试模式,如图 4-13 所示。

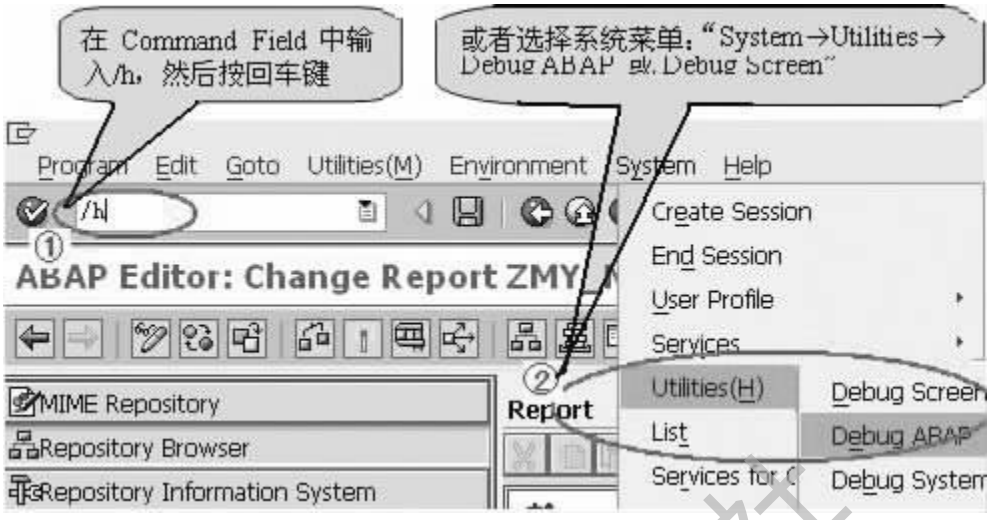

图 4-13　程序运行期进入调试模式的两种方法

在传统的 ABAP 调试器中，进入调试模式以后，可以在工具条上选择"单步执行(Single Step)"按钮，进行代码逐行跟踪。如图 4-14 所示。

图 4-14　单步跟踪程序执行与内存变量的状态变化

在屏幕下方的变量查看窗口输入要跟踪的变量名(最多可以输入 8 个)，或者直接在代码的编辑窗口某一行代码的某一个变量上双击鼠标，可以进一步查看并且跟踪该变量的值的变化，这个操作对定位和解决程序中存在的 Bug 是至关重要的。在传统的 ABAP 调试器中，可以在一行代码的前面双击鼠标来设置一个断点，或者通过选择系统的菜单项"Breakpoint→Breakpoint at→Statement"来为一行代码设置一个断点。如图 4-15 所示。

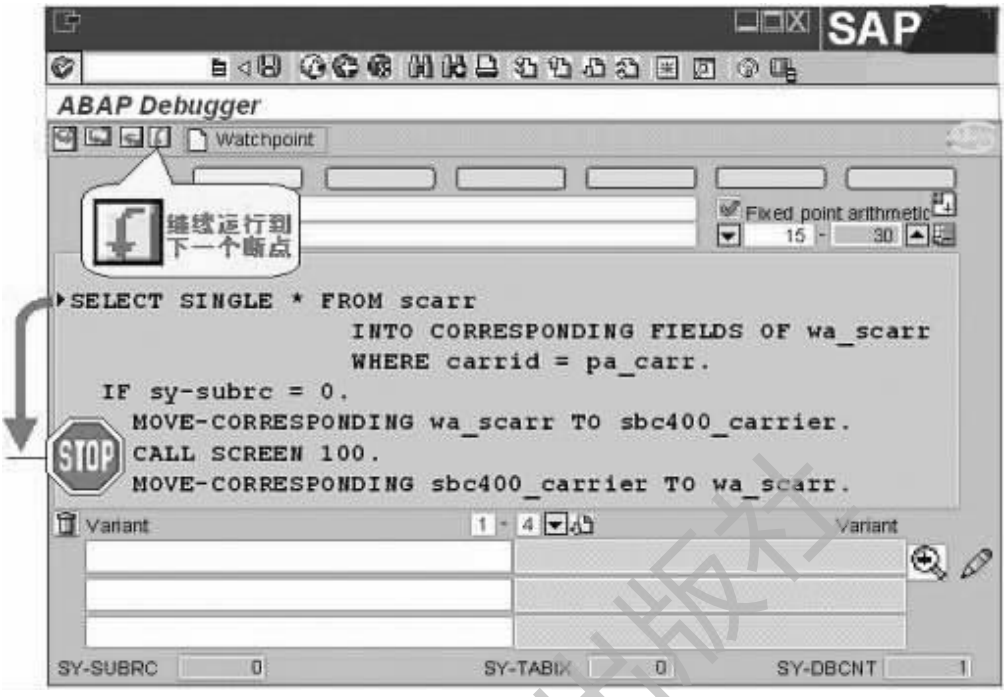

图 4-15 为代码行设置断点

当点击应用工具条上的“Continue”按钮时，程序将运行到下一个断点所在的代码行停止，可以在该行上观察内存变量的状态变化。

【能力训练】

运用 ABAP 语言的条件分支语句和循环语句进行程序设计。

小 结

本章重点向读者介绍了 ABAP 语言的基本语法，介绍了完整的 ABAP 标准数据类型与不完整的 ABAP 标准数据类型以及它们之间的区别，还介绍了 ABAP 局部数据类型与 ABAP 数据对象的定义方法。在 ABAP 程序中，采用数据类型定义数据对象，数据对象必须要先声明再使用。要掌握常用的数据类型之间的运算方法，重点掌握字符串常用的操作方法。本章第二部分介绍了 ABAP 语句的基本结构，如 ABAP 关键字分类，数据对象赋值等。要重点掌握 ABAP 程序流程控制和向用户发送对话消息的方法。然后介绍了调试 ABAP 程序的方法。应掌握断点的作用与设置断点的方法，当程序出现问题的时候，可以运用跟踪调试的方法对代码和内存变量的状态进行跟踪，从而快速定位问题的原因和所在位置。

【学习效果评估】

(一)思考题

1. 在 ABAP 程序中定义一个数据对象,可能的数据类型来源有哪几种?

2. ABAP 中,数据类型可以应用于哪些场合?

3. ABAP 标准数据类型分哪几种? 分别都包含哪些具体的类型?

4. 如果在 ABAP 程序中需要显示一个固定的字符串常量,且需要该显示支持多种语言,该如何做?

5. 本章介绍的 ABAP 字符串的操作有哪些? 具体使用方法是什么?

6. ABAP 中的多分支控制语句 CASE…ENDCASE 能够使用其他语法替换吗? 如何替换?

7. ABAP 中的循环语句都包括哪几种?

8. 如何退出循环? 如何退出本次循环?

9. 调试一个 ABAP 程序有什么重要意义? 如何使 ABAP 程序进入调试状态?

10. 如何为一行 ABAP 代码设置断点?

(二)练习题

1. 编写程序,求使整数数列"1,2,3,…,n"的和不大于 6000 的最大的正整数 n。

2. 求(100,1000)之间所有的能被 3 整除的偶数的和,并且在列表屏幕上输出结果。

3. 用 PARAMETERS 语句接受用户从选择屏幕上输入的 1 个正整数 iWeek,当 iWeek 在[1,7]之间时,向用户发送对话消息对应的星期的英文提示(消息类型 I),否则采用对话消息提示输入的星期错误(消息类型 E)。

4. 任意从一个英文网站上下载一段 30 个单词以内的英文,作为 ABAP 程序中的字符串变量 str1 的初始值。

(1)将该字符串按照空格拆分成一个个英文单词,并将拆分结果保存到一个内表中。将该内表的内容按多行显示。

(2)将内表中的多行重新合并为一个新的字符串 str2,并以空格分割单词,显示合并结果。

(3)比较合并前的字符串 str1 与合并后的字符串 str2 之间是否完全一致,显示比较结果。