ABAP/4 语言入门

郑州三全食品股份有限公司 信息部

内部培训教材

作者： 强晟

日期： 2006 年 8 月 2 日

1. ABAP/4 入门

简介

ABAP/4(Advanced Business Application Programming) 是 SAP/R3 目前唯一的系统开发工具，属 4GL(第四代语言)，语法比较近似 Visual Basic 或 JAVA，和传统的第三代语言，如C、PASCAL有很大不同，在程序模块(Program Structure Module)可分以下三个部分：

1. **过程块内的顺序编码**
   与一般语言语法近似，如IF，WHILE等，但并没有 GOTO 语法
2. **报表**
   调用一个独立的事件(Depending Event)，读取数据库产生数据列表
3. **对话框**
   屏幕参数输入的对话框，专门处理数据库读取或更改的事务过程

基本语言概览

1. **数据元素声明方式**，如数值，字符数据变量声明
2. **操作符使用**，如 + - * /
3. **控制元素使用**，如Boolean值
4. **特殊数据格式**，如日期与时间
5. **字符处理函数**，如部分字符串的截取
6. **子程序或自定函数的调用**
7. **SQL语法使用**
8. **数据结构的使用**，如过程内表的声明与使用

报表概览

1. **Reports Task**，如报表屏幕预览或打印机打印的选择
2. **Reports模块是一个独立的程序**
3. **数据库读取方式**，如可定义逻辑数据库(与磁盘的物理存储对应)
4. **报表数据的计算与产生**
5. **报表的输出**

对话框概览

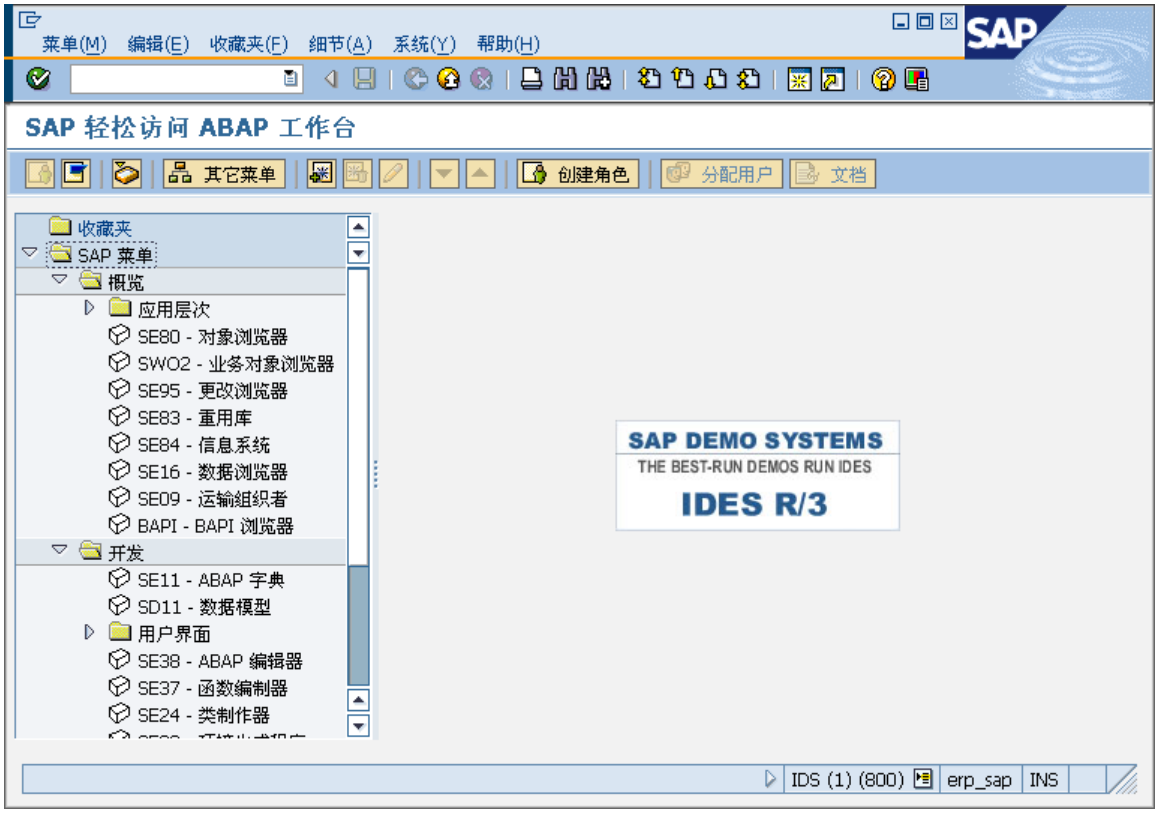
1. **专处理数据库的读取与更改**，如使用SQL命令
2. **对话框不是一个独立程序**，使用事务码来产生屏幕对话框
3. **由流程逻辑控制**，流程逻辑分成 PBO(Process Before Output，输出前过程)与PAI(Process After Input，输入后过程)

2.开始编程

2.1 ABAP/4 编辑器

创建ABAP/4 程序

使用ABAP工作台撰写程序（选择工具->ABAP/4工作台，事务码S001），屏幕如下：

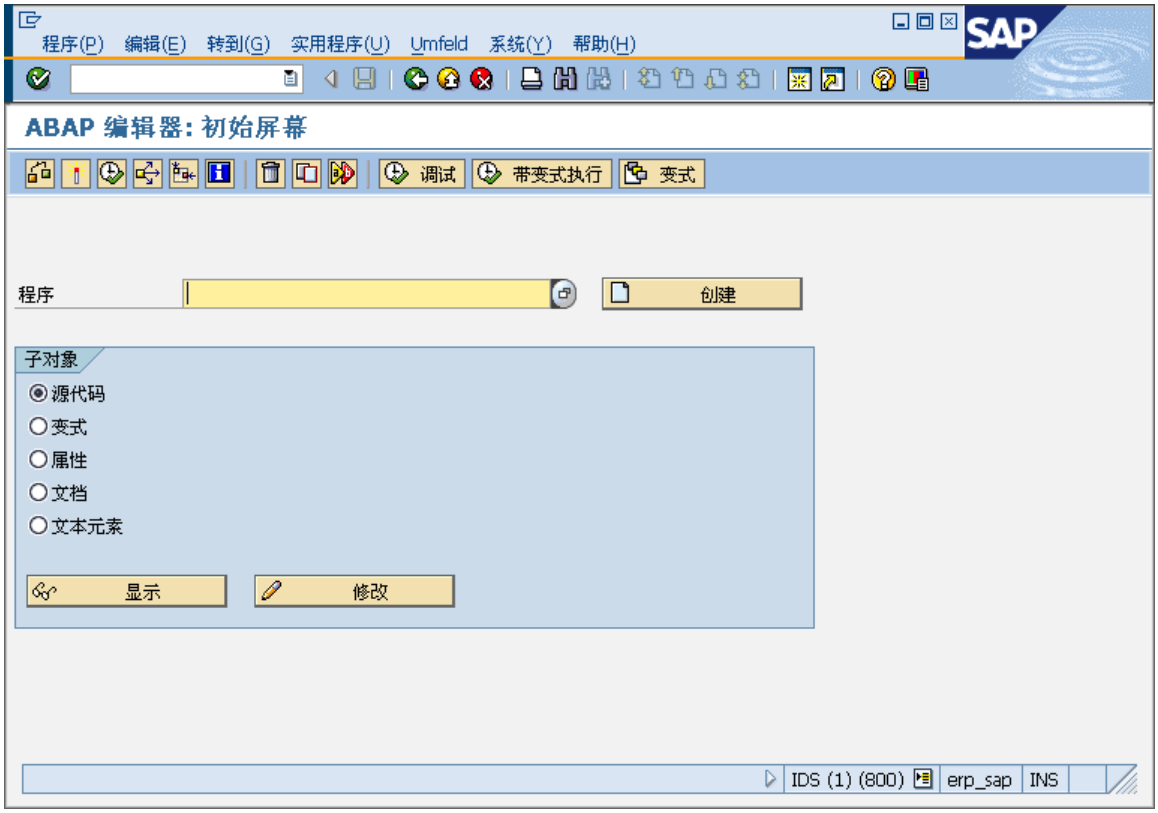


可分成：

1. **ABAP/4编辑器**：针对简单的报表或程序，仅使用几个组件或不使用
2. **对象浏览器**：针对复杂的报表或程序，如对话事务模块（Dialog Transaction Module）的编写

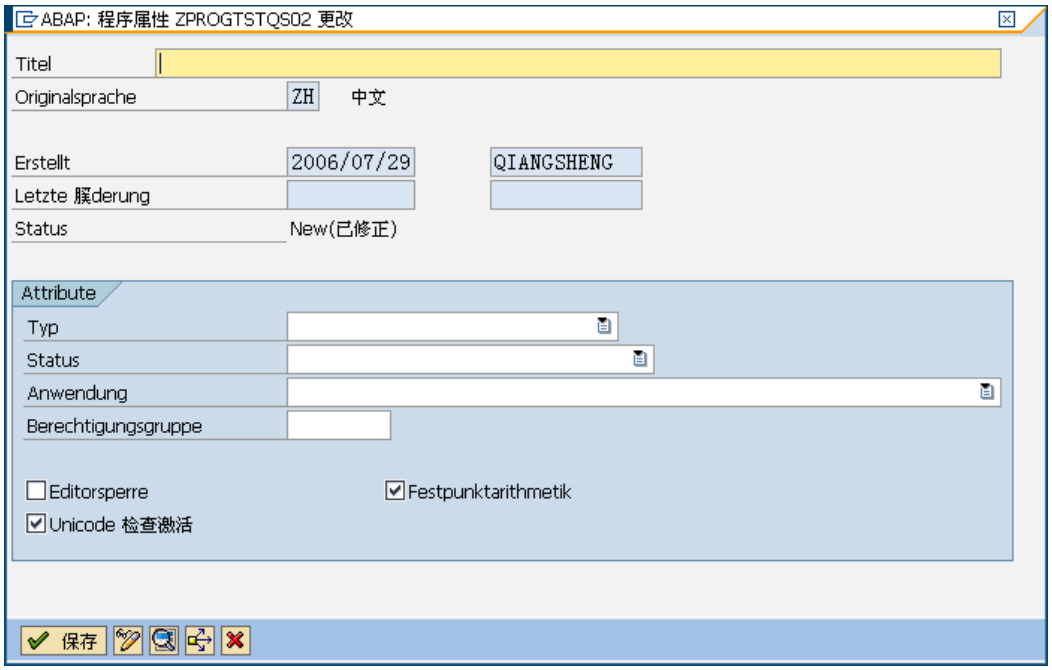
使用ABAP/4 编辑器撰写程序

1. ABAP/4编辑器界面如下（事务码SE38）：



2. 输入程序名称，如果是新程序，按下“创建”，如果修改已存在程序，则按下“修改”或F6键。在命名规则上，报表程序为 Yaxxxxxx 或 Zaxxxxxx，a表示 application module（应用程序模块）简称，如 s 表示 SD。对话框程序为 SAPMYxxx 或 SAPMZxxx

3. 输入程序属性



(1).Title：程序描述或功能说明

(2).Type：执行模式，包括：可执行程序，如报表；INCLUDE程序；模块池；函数组；子程序池

(3).Status：程序开发状态，包括：SAP标准生产程序；客户生产程序；系统程序；测试程序

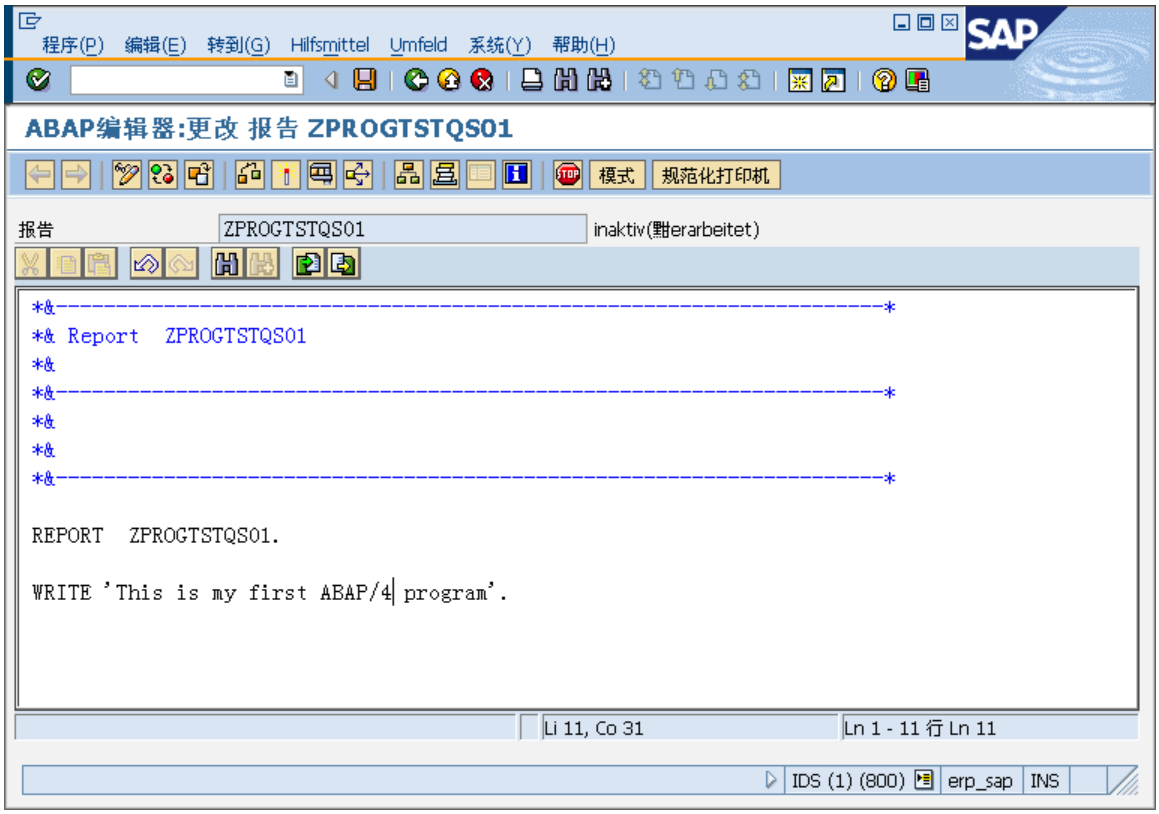
(4).Application：程序所属的应用模块，如：财务会计；物料管理；销售分销

注意：由于 SAP 翻译问题，上图界面语言混乱，操作时请以实际显示为准。

4. 选择开发类

开发类：用于同一系统中各个程序，如果不属任一类，可使用 $TMP

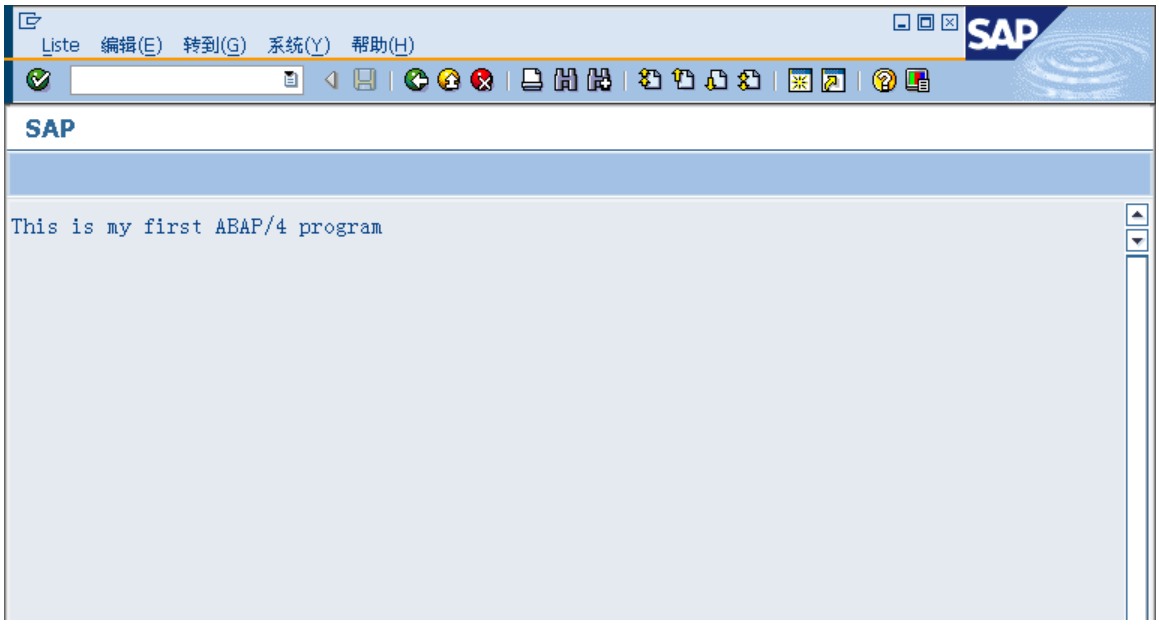
5. 撰写源代码



REPORT之后接的是程序名称，WRITE 是显示的意思，会将所接的字符串在屏幕上显示，注意每一行最后要有一个'.' (点)，表示语句的结束，储存后返回 ABAP/4 编辑器界面。

6. 执行程序

按“执行”或F8键，屏幕可见执行所得的结果



重要的编辑功能键

键	功能
F8	执行程序
F5	复制光标所在列的内容
F11	储存文件
CTRL+F11	删除光标所在列

2.2 ABAP/4 数据元素

数据类型

ABAP/4的数据类型可分成：

类型	长度	范围	初始值	说明
C	1	1-65535字节	空格	字符串数据，如'Program'
D	8	8字节	'00000000'	日期数据，格式为YYYYMMDD
F	8	8字节	0	浮点数
I	4	-2^31至2^31-1	0	整数
N	1	1-65535字节	'00...0'	数值所组成的字符串
P	8	1-16字节	0	Packed数，用在小数点数
T	6	6字节	'000000'	时间数据，格式为HHMMSS
X	1	1-65535字节	X'00'	16进制数

变量声明

变量声明包含 name, type, length 和 structure 四个部分，使用 DATA 命令，如

```
DATA: S1 TYPE I,
      SUM TYPE I.
```

常数声明

常数声明使用 CONSTANTS 命令，如声明 PI 是一个小数点 5 位的值 3.14159

```
CONSTANTS PI TYPE P DECIMALS 5 VALUE '3.14159'.
```

系统所定义数据

这是由系统所定义的专有名词，如：

```
SPACE "空格字符串
```

```
SY-SUBRC "系统执行返回值，0表示成功
SY-UNAME "登录帐号
SY-DATUM "系统日期
SY-UZEIT "系统时间
SY-TCODE "目前的事务码
```

TYPE声明

用来指定数据类型或声明自定数据类型

示例：

```
TYPES: BEGIN OF MYLIST,
         NAME(10) TYPE C,
         NUMBER TYPE I,
         END OF MYLIST.
DATA LIST TYPE MYLIST.
```

LIKE声明

跟TYPE声明使用格式相同，如

```
DATA TRANSCODE LIKE SY-TCODE.
```

不同的是LIKE用在已有值的数据项，如系统变量，而TYPE声明则是用在指定数据类型。

DATA声明

语法：

DATA <f> [<length>] <type> [<value>] [<decimals>]

<f>：变量名称，最长30个字符，不可含有 + . , : ( ) 等字符

<length><type>：数据类型及长度，如LINE(20) TYPE C. MYNAME LIKE SY-UNAME.

<value>：初值

<decimals>：小数位数

示例：

```
DATA: COUNTER TYPE P VALUE 1,
         FLAG TYPE C VALUE IS INITIAL,
         WEIGHT TYPE P DECIMALS 2 VALUE '1.25'.
```

字段变量的声明：

```
DATA: BEGIN OF ADDRESS,
         NAME(10) TYPE C ,
         NUMBER TYPE P,
```

```
END OF ADDRESS.
```

使用时用字段变量加上表名称，如 ADDRESS-NAME

📖 CONSTANTS声明

用来声明常数

语法：

```
CONSTANTS <c> [<length>] <type> [<value>] [<decimals>]
```

示例：

```
CONSTANTS: CNAME(10) VALUE '周庆日',
BIRTH_DAY TYPE D VALUE '19650201'.
```

📖 STATICS声明

声明的变量仅在目前的程序中使用，结束后会自动释放

语法：

```
STATICS <c> [<length>] <type> [<value>] [<decimals>]
```

📖 TABLES声明

用来声明表工作区的数据，对应至ABAP/4字典对象(Dictionary Object)，由SQL命令加载所需数据

语法：

```
TABLES <dbtab>
```

示例：

```
TABLES: SPFL.
SELECT * FROM SPFL.
WRITE: SPFL-MANDT, SPFL-CARRID, SPFL-CONNECTION.
ENDSELECT.
```

从ABAP/4字典的SPFL表载入MANDT,CARRID,CONNECTION三个字段至SPFL这个表工作区

2.3 向屏幕输出数据

📖 WRITE命令

ABAP/4用来在屏幕上输出数据的命令是 WRITE

语法：

```
WRITE[:] 数据项
```

数据项可以是常数或变量，如果同时输出多项，必须加冒号，如：

```
WRITE 'This is sample'.
WRITE: 'COMPANY:', STFL-CARRID.
```

指定屏幕位置显示

语法：

```
WRITE AT [/] [<pos>] [(<len>)] 资料项
/：先往下一列
pos：屏幕X 轴坐标
(len)：显示数据的长度
```

示例1：

```
WRITE 'First Line'.
WRITE /6 'Second Line'.
```

输出结果：

```
First Line
      Second Line
```

示例2：

```
DATA: NUMBER TYPE I VALUE '1234567890',
      TEXT(10) VALUE 'ABCDEFGHIJ'.
WRITE: (5) NUMBER, /(6) TEXT.
```

输出结果：

```
*7890
ABCDEF
```

指定显示格式

语法：

```
WRITE 数据项 <显示格式参数>
```

显示格式参数：

LEFT-JUSTIFIED 数据靠左显示
CENTERED 数据靠中间显示
RIGHT-JUSTIFIED 数据靠右显示
UNDER <g> 在数据项<g>的 X 轴开始坐标显示
NO-GAP 紧接着显示，不留空格
USING EDIT MASK <m> 使用内嵌字符显示，如 11:20:30
USING NO EDIT MASK 不使用内嵌字符
NO-ZERO 数字前面 0 的部分不显示

NO-SIGN 不显示正负号
DECIMALS <d> 显示d位小数字数
EXPONENT <e> F(浮点数) exponent 的值
ROUND <r> 四舍五入至小数位数下 r 位
CURRENCY <c> 币别显示
DD/MM/YY 日期显示格式
MM/DD/YY 日期显示格式
DD/MM/YYYY 日期显示格式
MM/DD/YYYY 日期显示格式
DDMMYY 日期显示格式
MMDDYY 日期显示格式
YYMMDD 日期显示格式

示例1：

```
DATA: X TYPE I VALUE '112030',
      A(5) VALUE 'ABCDE'.
WRITE X USING EDIT MASK '__:__:__'.
```

输出结果为：

```
11:20:30
```

示例2：

```
DATA: X TYPE I VALUE '112030',
      A(5) VALUE 'ABCDE'.
WRITE X USING EDIT MASK '$___,___'.
```

输出结果为：

```
$112,030
```

产生空白列

产生n个空白列

语法：

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

示例：

```
WRITE 'PASS1'.
SKIP.
WRITE 'PASS2'.
```

输出结果为：

```
PASS1


PASS2
```

显示图标

可以显示R/3系统所提供的符号或图标

语法：

```
WRITE <symbol-name> AS SYMBOL
WRITE <icon-name> AS ICON
```

示例：

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

执行结果：

要查看系统所提供有哪些符号及图标，可选择返回ABAP编辑器的初始页，执行程序SHOWSYMB和SHOWICON程序列出所有符号和图标

跳至指定列坐标

将坐标跳至指定的Y轴列坐标

语法：

```
SKIP TO LINE [<n>]
```

示例：

```
SKIP TO LINE 5.
WRITE 'PASS1'.
```

执行结果：

显示复选框数据

以字符串数据内容的第一个字符为复选框的输出，如果是空白，复选框显示为空白，相反则显示X，可用在逻辑判断检查。

语法：

```
WRITE <资料项> AS CHECKBOX.
```

示例：

```
DATA: FLAG1 VALUE ' ',
      FLAG2 VALUE 'X'.
WRITE: / 'CHECK FLAG 1:' , FLAG1 AS CHECKBOX.
WRITE: / 'CHECK FLAG 2:' , FLAG2 AS CHECKBOX.
```

执行结果：

2.4 处理数据

📖 赋值

语法：

```
MOVE <F1> TO <F2>.
```

将F1的值存至变量F2中，也可写成

```
F2 = F1.
```

示例：

```
M_NAME = 'CHER'.
```

📖 使用偏移量

语法：

```
MOVE <F1>[+<偏移量>(取位数)] TO <F2>[+<偏移量>].
```

示例1：

```
DATA: F1(10) VALUE 'ABCDEFGHIJ',
      F2(5).
F2 = F1+3(5). "自第4个位置开始取出5个字符
WRITE / F2. "F2的内容会变成DEFGH
```

执行结果：

```
DEFGH
```

示例2：

```
DATA: F1(10) VALUE 'ABCDEFGHIJ',
      F2(5) VALUE '12345'.
MOVE F1+3(5) TO F2+2. "自第4个位置开始取出5个字符，
                      "从F2的第三个位置开始替换
WRITE / F2. "F2的内容会变成12DEF
```

执行结果：

```
12DEF
```

字段变量组件的复制

语法：

```
MOVE–CORRESPONDING <Strings1> TO <String2>.
```

将Strings1中的字段组件的数据复制至String2中，仅复制相同名称的组件

示例：

```
DATA: BEGIN OF ADDRESS,
        FIRSTNAME(10) VALUE 'LULU',
        LASTNAME(10) VALUE 'CHOU',
        TEL(12) VALUE '4660570',
        END OF ADDRESS.
DATA: BEGIN OF NAME,
        FIRSTNAME(10),
        LASTNAME(10),
        E_MAIL(30),
        END OF NAME.
MOVE-CORRESPONDING ADDRESS TO NAME.
WRITE / ADDRESS.
WRITE / NAME
"NAME-FIRSTNAME 变成 'LULU'，NAME-LASTNAME 变成 'CHOU'，
"而 NAME-E_MAIL 则不变
```

执行结果：

```
LULU       CHOU       4660570
LULU       CHOU
```

变量以值调用的使用

在变量的使用上，可以使用类似Call By Value（以对象的值作为变量名）的方法

语法：

```
WRITE (<f>) TO <g>
```

示例：

```
DATA: NAME(20) VALUE 'SOURCE',
        SOURCE(10) VALUE 'LILY',
        TARGET(10).
WRITE (NAME) TO TARGET.
WRITE / TARGET. "屏幕可印出 LILY
                "（NAME的值是SOURCE，SOURCE的值是LILY）
```

执行结果：

```
LILY
```

清除变量内容

语法：

```
CLEAR <f>
```

清除变量现在内容，恢复成初值

示例：

```
DATA N TYPE I VALUE 100.
CLEAR N.
```

变量N的内容变成0

算术符号

**	乘幂
*	乘
/	除
+	加
-	减
DIV	整数除法
MOD	余数除法

数值函数

1. ABS(N): 返回数值N的绝对值

2. SIGN(N): 1 if N > 0
0 if N = 0
-1 if N < 0

3. CEIL(N): 返回大于数值N的最小整数

示例：

```
WRITE CEIL(-5.65)  显示 –5.00
WRITE CELL(4.54)  显示 5.00
```

4. FLOOR(N): 返回小于数值N 的最大整数

示例：

```
WRITE FLOOR(-5.65)  显示 –6.00
WRITE FLOOR(4.54)  显示 4.00
```

5. TRUNC(N): 返回数值N 的整数部分

示例：

```
WRITE TRUNC(5.65) 显示 5.00
```

6. FRAC(N): 返回数值N的小数部分

示例：

```
WRITE FRAC(5.65) 显示 0.65
```

7. COS(A),SIN(A),TAN(A): 返回三角函数 cos A, sin A, tan A 的值，A 为弧度量

8. EXP(N): 返回 e^N 值

8. LOG(N): 返回 log eN 值

9. LOG10(N): 返回 log N 值

10. SQRT(N): 返回 N 的平方根值

日期与时间运算

1. 日期数据的运算

日期数据可以直接运算，如加法与减法的运算

示例：

```
DATA: Mdate TYPE D.
Mdate = SY-DATUM. " 如返回 19971015，当前日期
Mdate+6(2) = '01' " Mdate 变成 19971001，当月的一号
Mdate = Mdate - 1 " Mdate 变成 19970931，上月最后一天
```

2. 时间数据的运算

时间格式为'hhmmss'，如 '212030' 表示 '21:20:30'

示例：

```
DATA: HOURS TYPE I,
      MINUTES TYPE I,
      T2 TYPE T VALUE '200000',
      T1 TYPE T VALUE '183000'.
HOURS = (T2 - T1) / 3600. "计算有几小时
MINUTES = (T2 – T1) / 60. "计算几分钟
```

字符串数据处理

1. 字符串移位

语法：

```
SHIFT <c> [BY <n> PLACES] [<modes>]
<modes> :  (1).空白，字符串往左移一位
           (2).LEFT，字符串往左移 n 位
```

(3).RIGHT，字符串往右移 n 位
(4).CIRCULAR，字符串以环状方式移位

示例：

```
DATA STRING(10) VALUE 'ABCDEFGHIJ'.
SHIFT STRING. "得到 BCDEFGHIJ
SHIFT STRING BY 2 PLACES RIGHT. "得到 ABCDEFGH
```

2. 替换字符串内容

语法：

```
REPLACE <string1> WITH <string2> INTO <c>
```

将字符串 <c> 中的 <string1> 以 <string2> 来取代

示例：

```
DATA: STRING(10) VALUE 'ABCDEFGHI',
STR1(3) VALUE 'DEF',
STR2(3) VALUE '123'.
REPLACE STR1 WITH STR2 INTO STRING.
WRITE / STRING. "得到 ABC123GHI
```

3. 大小写的转换

语法：

```
TRANSLATE <c> TO UPPER CASE. "转成大写
TRANSLATE <c> TO LOWER CASE. "转成小写
```

4. 在字符串中寻找部分字符串

语法：

```
SEARCH <c> FOR <str>
```

示例：

```
DATA STRING(10) VALUE 'ABCDEFGHIJ'.
会回存至两个变量，SY-SUBRC 和 SY-FDPOS，
若找到则 SY-SUBRC 为 0 ，SY-FDPOS 存开始位置，
若找不到则 SY-SUBRC 为 4 ，SY-FDPOS 为 0
```

5. 字符串长度

语法：

```
STRLEN(<c>)
```

示例：

```
INT = STRLEN('XYZABC'). "得到 6
INT = STRLEN('ABC'). "得到 3
```

6. 取部分字符串

语法：

```
<f>[+<o>][<l>]
从字符串 <f> 的 <o> 的下一个字符开始取 <l> 个字符
```

示例：

```
DATA T(10) VALUE 'ABCDEFGHIJ'.
WRITE / T+2(4). "得到 'CDEF'
```

2.5 流程控制

比较运算符

1. = 或 EQ : 等于
2. <> 或 >< 或 NE : 不等于
3. < 或 LT : 小于
4. <= 或 LE : 小于等于
5. > 或 GT : 大于
6. >= 或 GE : 大于等于
7. AND : 与
8. OR : 或
9. NOT : 非

条件语句

1. IF 语句

语法：

```
IF <条件1>.
<语句1>
ELSEIF <条件2>.
<语句2>
ELSEIF <条件3>.
<语句3>
.....
ELSE.
<else语句>
ENDIF.
```

(1).在每个判断语句之后要加上 .

(2).在巢状循环（循环嵌套）之中无法使用 ELSE 语句，ELSE 语句属 IF 语句（？）

示例：

```
IF 3 > 8.
  WRITE / '3 is less than 8'.
ENDIF.
```

2. CASE 语句

语法：

```
CASE <变量f>.
WHEN <值1>.
<语句1>
WHEN <值2>.
<语句2>
....
WHEN OTHERS.
<others语句>
ENDCASE.
```

示例：

```
DATA S(1) TYPE C
S = 'A'.
CASE S.
WHEN 'X'.
  WRITE / 'String is X'.
WHEN OTHERS.
  WRITE / 'String is not X'.
ENDCASE.
```

循环语句

1.计数循环

语法：

```
DO [n TIMES] [VARYING <f> FROM <start> NEXT <end> [RANGE <range>]].
<循环块>
ENDDO.
```

示例1：

```
DO 2 TIMES.
  WRITE / 'X'.
ENDDO.
```

执行结果：

```
X
```

```
X
```

示例2：

```
DO VARYING I FROM 1 TO 10.
  S = S + I.
ENDDO.
WRITE: / ,'1+2+3+...+10=',S
```

执行结果：

```
1+2+3+...+10=55
```

2.条件循环

语法：

```
WHILE <条件>.
<语句块>
ENDWHILE.
```

示例：

```
DATA: I TYPE I,
      S TYPE I.
I = 1.
S = 0.
WHILE I <= 10.
  S = S + I.
  I = I + 1.
ENDWHILE.
WRITE: / '1+2+3+...+10=',S.
```

执行结果为：

```
1+2+3+...+10=55
```

📖 循环控制语句

1. CONTINUE

跳至循环的下一次

示例：

```
DO 3 TIMES.
  IF SY-INDEX = 2.
    CONTINUE.
  ENDIF.
  WRITE / SY-INDEX.
ENDDO.
```

执行结果：

```
1
3
```

2. CHECK <条件>

CHECK 之后条件成立才继续往下执行循环

示例：

```
DO 5 TIMES.
  CHECK SY-INDEX BETWEEN 2 AND 4.
  WRITE / SY-INDEX.
ENDDO.
```

执行结果:

```
2
3
4
```

3. EXIT

跳离循环体

示例：

```
DO 10 TIMES.
  IF SY-INDEX = 4.
    EXIT.
  ENDIF.
  WRITE / SY-INDEX.
ENDDO.
```

执行结果:

```
1
2
3
```

📖 无穷循环

```
DO .
<Statement Block>
ENDDO.
```

无穷循环必须配合 EXIT 语句来执行

2.6 处理内表

内表的声明

ABAP/4 的内表（Internal Table）如同其它语言的数组结构，在操作上可以有复制，删除，新增插入等功能。

1. 使用 TYPE 语句

语法：

```
TYPES <t> <type> OCCURS <n>
```

声明一个数组 <t>，类型为 <type>，长度为 <n>

示例1：

```
TYPES A TYPE I OCCURS 10.
```

A 是个 10 行的整型数组

示例2：

```
TYPES: BEGIN OF LINE,
         COL1 TYPE I,
         COL3 TYPE I,
         END OF LINE.
TYPES ITAB TYPE LINE OCCURS 10.
```

声明一个内表 ITAB，总共有 10 个行，其工作区名称为 LINE

2. 使用 DATA 语句

若使用 DATA 语句来声明内表，可分成要不要有HEADER LINE，HEADER LINE 就是所谓的工作区，用在数据的存取上。

语法：

```
DATA <f> <type> OCCURS <n> [WITH HEADER LINE]
```

示例：

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

3. 直接声明，不使用工作区

语法：

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

示例：

```
DATA: BEGIN OF ITAB OCCURS 10,
        COL1 TYPE I,
        COL2 TYPE I,
        END OF ITAB.
```

如此产生的内表不会有工作区，也就是声明时不会引用其它的组件声明。

追加行

语法：

```
APPEND [<wa>] TO [Initial Line To] <itab>
```

[Initial Line To] 为增加一预设初值的行

示例1：使用工作区

```
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. "SY-INDEX 为循环的计数器
  LINE-COL2 = SY-INDEX ** 2.
  APPEND LINE TO ITAB. "新增至内表中
ENDDO.
LOOP AT ITAB INTO LINE. "ITAB 总共有两个行
  WRITE: / LINE-COL1, LINE-COL2.
ENDLOOP.
```

执行结果为：

```
1    1
2    4
```

示例2：不使用工作区

```
DATA: BEGIN OF ITAB OCCURS 10,
        COL1 TYPE I,
        COL2 TYPE I,
        END OF ITAB.
DO 2 TIMES.
  ITAB-COL1 = SY-INDEX.
  ITAB-COL2 = SY-INDEX ** 2.
  APPEND ITAB. "新增至内表中
ENDDO.
LOOP AT ITAB. "ITAB 总共有两个行
  WRITE: / ITAB-COL1, ITAB-COL2.
ENDLOOP.
```

执行结果为：

```
1 1
2 4
```

加入另一内表的行

语法：

```
APPEND LINES OF <itab1> [FROM <n1>] [TO <n2>] TO <itab2>
```

将<itab1>的行加入至<itab2>中，可选取自<n1>至<n2>的范围

示例：

```
APPEND LINES OF ITAB TO JTAB.
```

将 ITAB 所有行加入JTAB 中

聚集行

在加入新行时将有相同标准键（standard key，非数值字段）的数值字段汇总

语法：

```
COLLECT [<wa> INTO] <itab>
```

示例：

```
DATA: BEGIN OF ITAB OCCURS 3,
        COL1(3) TYPE C,
        COL2 TYPE I,
        END OF ITAB.
ITAB-COL1 = 'ABC'. ITAB-COL2 = 10.
COLLECT ITAB.
ITAB-COL1 = 'XYZ'. ITAB-COL2 = 20.
COLLECT ITAB.
ITAB-COL1 = 'ABC'. ITAB-COL2 = 30.
COLLECT ITAB. "汇总 COL2 至 COL1=ABC 的元素上
LOOP AT ITAB.
  WRITE: / ITAB-COL1, ITAB-COL2.
ENDLOOP.
```

执行结果:

```
ABC 40
XYZ 20
```

插入行

在指定的内表行位置之前插入行

语法：

```
INSERT [<wa> INTO] [INITIAL LINE INTO] <itab> [INDEX <idx>]
```

示例：

```
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 * 10.
  LINE-COL2 = SY-INDEX * 20.
  APPEND LINE TO ITAB.
ENDDO.
LINE-COL1 = 100.
LINE-COL2 = 200.
INSERT LINE INTO ITAB INDEX 2. "插入在位置2 之前
LOOP AT ITAB INTO LINE.
  WRITE: / SY-TABIX, LINE-COL1, LINE-COL2. "SY-TABIX 为Table 位置
ENDLOOP.
```

执行结果：

```
1 10 20
2 100 200 “插入的行
3 20 40
4 30 60
```

📖 插入另一内表行

语法：

```
INSERT LINES OF <itab1> [FROM <n1> TO <n2>] TO <itab2> INDEX <idx>
```

将<itab1>的行插入至<itab2>中，位置在<idx>之前，可选取自<n1>至<n2>的范围

示例：

```
INSERT LINES OF ITAB TO JTAB INDEX 3.
```

将 ITAB 所有行插入 JTAB 中，位置在第三个行之前

📖 内表行数据的读取

语法：

```
LOOP AT <itab> [INTO <wa>] [FROM <n1> TO <n2>] [WHERE <condition>]
<loop expression>
ENDLOOP.
```

根据设定的范围选取行记录，读完后自动移往下一行

示例：

```
LOOP AT ITAB INTO LINE WHERE COL1 >100.
  WRITE: / SY-TABIX, LINE-COL1.
ENDLOOP.
```

仅读取 COL1 > 100 的行

读取内表指定位置的行

语法：

```
READ TABLE <itab> [INTO <wa>] INDEX <idx>
```

自指定位置 <idx> 读取行数据

示例：

```
READ TABLE ITAB INTO LINE INDEX 5
```

读取 ITAB 的第 5 个行数据，放入 LINE 的字段中

根据字段内容寻找

语法：

```
READ TABLE <itab> INTO <wa>
```

示例：

```
ITAB-COL1 = 'ABC'.
READ TABLE ITAB INTO LINE.
```

找出 ITAB 中 COL1 字段内容是 ABC 的行，找到的值放入 LINE 中

若找到 SY-SUBRC 的值为 0，找不到则值为 4，<itab>必须声明有工作区

更改行内容

语法：

```
MODIFY <itab> [FROM <wa>] [INDEX <idx>] [TRANSPORTING <f1>...<f2>] [WHERE <condition>]
```

TRANSPORTING <f1> ..<f2>：指定更改的字段名称

示例：

```
LINE-COL1 = 4.
LINE-COL2 = 100.
MODIFY ITAB INDEX 2 FROM LINE.
```

将目前位置行以 LINE 的内容更改

示例：

```
LINE-COL1 = 10.
MODIFY ITAB FROM LINE INDEX 3 TRANSPORTING COL1 COL2.
```

将第三个行的COL1 字段更改为 10

删除行

删除内表的行

语法：

```
DELETE <itab> INDEX <idx>
```

示例：

```
DELETE ITAB INDEX 4
```

删除第四个行

加上删除条件：

```
DELETE <itab> [FROM <n1> TO <n2>] [WHERE <condition>]
```

示例：

```
DELETE ITAB FROM 3 TO 10.
```

删除第 3 至第 10 个行

内表排序

语法：

```
SORT <itab> [<order>] [BY <f1>] ....
```

[<order>]：可分成递减(DESCENDING)和递增(ASCENDING)，空白表示 ASCENDING

<f1>:为指定的字段

示例：

```
SORT ITAB DESCENDING BY COL2.
```

将 ITAB 根据 COL2 字段递减排序

计算数值字段总和

语法：

```
SUM
```

计算得总和存在工作区中，但只能存在 LOOP 语句中

示例：

```
LOOP AT ITAB INTO LINE.
  SUM.
ENDLOOP.
WRITE: / LINE-COL1, LINE-COL2.
```

LINE-COL1 和 LINE-COL2 存数值总和

初始化表

1. REFRESH <itab>

使用在没有 HEADER LINE 的内表中，清除所有行

示例：

```
REFRESH ITAB.
```

2. CLEAR <itab>[ ]

使用在有 HEADER LINE 的内表中，清除所有行

示例：

```
CLEAR ITAB[ ].
```

3. FREE <itab>

释放(Release)内表所占的内存空间，用在 REFRESH 和 CLEAR 命令之后

示例：

```
FREE ITAB.
```

2.7 处理字典表

R/3 对于存放在关系型数据库的数据可使用 SQL 命令读取或处理，命令种类可分成 DDL(Data Definition Language，数据定义语言)命令，如CREATE，及DML(Data Manipulation Language，数据处理语言)，如 SELECT 及INSERT 等，处理方式分成 OPEN SQL 及 NATIVE SQL，前者在处理时，数据库与命令解释器间有一个缓冲区，如SELECT * FROM...，后者则直接处理数据库，如 EXCE SQL SELECT...等，有两个重要的系统变量：

SY-SUBRC: 返回 0 表成功执行命令，4 表未找到符合条件数据

SY-DBCNT: 正处理的数据条数

TABLES命令

用于声明程序中所使用的表

语法：

```
TABLES table
```

SELECT命令

自数据库读取记录

语法：

```
SELECT <result> FROM <source> [INTO <target>] [WHERE <condition>] [ GROUP BY
```

```
<fields>] [ORDER BY <sort order>]
```

1.以循环方式读取所有记录

语法：

```
SELECT [DISTINCT] * ...
.....
ENDSELECT.
```

加上[DISTINCT]会自动去除重复的记录

示例：

```
TABLES SPFLI.
SELECT * FROM SPFLI WHERE COMPANY = 'DELTA'.
  WRITE: / PLANT, TEL.
ENDSELECT.
```

会以循环的方式逐条显示出符合条件的记录

2.读取单条记录

语法：

```
SELECT SINGLE * FROM .... WHERE....
```

示例：

```
TABLES SPFLI.
SELECT SINGLE * FROM SPFLI
        WHERE PLANT = 'CHUNGLI' AND TEL = '4526174'.
WRITE: / SPFLI-COMPANY, SPFLI-PLANT, SPFLI-TEL.
```

3.将读取的记录存放至工作区

语法：

```
SELECT ..... INTO <wa>
```

示例：

```
TABLES SPFLI.
DATA WA LIKE TABLES.
SELECT * FROM SPFLI INTO WA.
  WRITE: / WA-COMPANY, WA-PLANT.
ENDSELECT.
```

逐条写入 WA 工作区中

4.将读取的数据写入内表中

语法：

```
SELECT .. INTO TABLE <itab>
```

示例1：

```
TABLES SPFLI.
```

```
DATA ITAB LIKE SPFLI OCCURS 10 WITH HEADER LINE.
SELECT * FROM SPFLI INTO ITAB.
```

一次读10 条(内表的长度)记录存入 ITAB 中

语法：

SELECT .. INTO TABLE <itab> PACKAGE SIZE <n>

一次读取 <n> 条记录至 <itab>中

示例：

```
TABLES SPFLI.
DATA ITAB LIKE SPFLI OCCURS 10 WITH HEADER LINE.
SELECT * FROM SPFLI INTO ITAB PACKAGE SIZE 5.
```

一次读取 5 条记录

5.条件声明

语法：

WHERE <condition>

(1).BETWEEN <g1> AND <g2>

在 <g1> 至 <g2> 之间的条件范围

示例：

```
..WHERE YEAR BETWEEN 1995 AND 2000.
```

(2).LIKE <g>

表示条件包含的字符串

<1>._:表示一个字符

<2> %：表示一个字符串

示例：

```
..WHERE NAME LIKE 'LEE%'.
```

条件为 NAME 字段前3 个字符为 LEE

```
..WHERE MODEL LIKE '%SPS%'.
```

条件为 MODEL 字段包含 SPS

(3).IN (<g1>....<g2>)

包含在 <g1>...<g2>的条件

示例：

```
..WHERE PLANT IN ('TAOYUAN', 'CHUNGLI', 'LIUTU').
```

条件为 PLANT 是 TAOYUAN, CHUNGLI 或LIUTU 的记录

6.ORDER BY 声明

指定排序的字段或顺序

(1). ..ORDER BY PRIMARY KEY.

根据 PRIMARY KEY 递增排序

(2)...ORDER BY <f1> [DESCENDING] <f2> [DESCENDING]

示例：

```
SELECT * FROM IM ORDER BY PART .
```

INSERT命令

加入一条记录至数据库

1. 自工作区

语法：

```
INSERT INTO <database> VALUES <wa>
```

示例：

```
TABLES SPFLI.
DATA WA LIKE SPFLI.
WA-NO = '34051920'.
WA-COMPANY = 'DELTA'.
INSERT SPFLI VALUES WA.
```

将 ITAB 数据加入 SPFLI 中，也可写成 INSERT SPFLI FROM ITAB.

```
SPFLI-NO = '34299876'.
SPFLI-COMPANY = 'HP'.
INSERT SPFLI FROM SPFLI.
```

将工作区 SPFLI 中的数据加入数据库表 SPFLI 中

因工作区 SPFLI 的结构与数据库表 SPFLI 一样，所以也可写成 INSERT SPFLI.

2. 自内表

语法：

```
INSERT <database> FROM TABLE <itab> [ACCEPTING DUPLICATE KEY]
```

将 <itab>中非 NULL 的数据加入 <database>中，加上 [ACCEPTING DUPLICATE KEY] 能检查不加入有重复主键，若有重复则 SY-SUBRC 会返回 4

示例：

```
TABLES SPFLI.
DATA ITAB LIKE SPFLI OCCURS 10 WITH HEADER LINE.
ITAB-NO = '34051920'.
ITAB-COMPANY = 'DELTA'.
APPEND ITAB.
....
INSERT SPFLI FROM TABLE ITAB
ACCEPTING DUPLICATE KEY.
```

UPDATE命令

改变已存在的记录内容

1. 使用主键

语法：

```
UPDATE <database> FROM <wa>
```

示例：

```
TABLES SPFLI.
DATA WA LIKE SPFLI.
WA-NO = '34051920'.
WA-COMPANY = 'DELTA'.
UPDATE SPFLI FROM WA.
```

如 SPFLI 的主键是 NO，则会找到 NO = '34051920' 的记录，将其 COMPANY 字段改变为 DELTA

2. 使用条件表达式

语法：

```
UPDATE <database> SET < f1>=<values>... WHERE <condition>
```

根据条件表达式更改符合条件的记录

示例：

```
UPDATE SPFLI SET NO = '34051920',
COMPANY = 'DELTA'
WHERE TEL = '4526107'.
```

MODIFY命令

根据主键寻找数据表中符合的记录，若找到则更新，若找不到则新增记录

语法：

```
MODIFY <database> FROM <wa>
```

示例：

```
WA-NO = '34051920'.
WA-COMPANY = 'DELTA'.
MODIFY SPFLI FROM WA.
```

DELETE命令

删除数据文件的记录

1. 使用主键

语法：

```
DELETE <database> FROM <wa>
```

示例：

```
TABLES SPFLI.
DATA WA LIKE SPFLI.
WA-NO = '34051920'.
WA-COMPANY = 'DELTA'.
DELETE SPFLI FROM WA.
```

如 SPFLI 的主键是 NO，则会找到 NO = '34051920' 的记录，找到后将此行删除

2. 使用条件表达式

语法：

```
DELETE FROM <database> WHERE <condition>
```

根据条件式删除符合条件式的记录

示例：

```
DELETE FROM SPFLI WHERE AREA = 'AMERICAN'.
```

数据库游标

数据库游标（Database Cursor）是一个数据库暂存区，将经 SELECT 命令读取的记录存放至此暂存区，再由此暂存区放至工作区中，可减少数据库读取的次数。

1. 开启游标

语法：

```
OPEN CURSOR <c> FOR SELECT ... WHERE <condition>
```

示例：

```
TABLES SPFLI.
DATA: WA LIKE SPFLI,
      C1 TYPE CURSOR.
OPEN CURSOR C1 FOR SELECT * FROM SPFLI WHERE AREA = 'TAIWAN'.
```

2. 读取游标的数据存入工作区

语法：

```
FETCH NEXT CURSOR <c> INTO <wa>
```

示例：

```
FETCH NEXT CURSOR C1 INTO WA.
```

读取下一条游标位置的数据存入 WA，如果已无数据可读，SY-SUBRC 不返回 0

4. 关闭游标

语法：

```
CLOSE CURSOR <c>
```

示例：

```
CLOSE CURSOR C1.
```

📖 COMMIT WORK与ROLLBACK WORK

要确定数据成功写入数据库，可使用 COMMIT WORK 命令，如：

```
COMMIT WORK.
```

相反的，如果反悔要复原，可使用 ROLLBACK WORK，可复原在上个COMMIT WORK命令之后的数据，如：

```
ROLLBACK WORK.
```

📖 使用 NATIVE SQL 命令

语法：

```
EXEC SQL [PERFORMING <form>].
<statement block>] [;]
ENDEXEC.
```

示例：

```
DATA: BEGIN OF WA,
        NAME(8),
        AGE TYPE I,
        END OF WA.
DATA F1 TYPE I.
FI = 20.
EXEC SQL PERFORMING OUTPUT.
  SELECT NAME, AGE INTO :WA FROM NAME_TABLE WHERE AGE >= :F1.
ENDEXEC.
FORM OUTPUT.
  WRITE: / WA-NAME,WA-AGE.
ENDFORM.
```

2.8 ABAP/4 程序模块

ABAP/4 中所谓的模块在一般语言称之为子程序，其数据传递方式皆相似，如传值调用（CALL BY VALUE），参数调用（CALL BY REFERENCE）等，可分成以下几个部分：

1. 宏块（Macro Block）
2. 包含程序（Include Program）
3. 子程序（Subroutine Program）

4. 函数模块（Function Module）

宏块（Macro Block）

在程序中可以定义一段宏代码，并且可以传入参数，参数占位符(Placeholder)可自&1, &2 至 &9

1. 宏的定义

```
DEFINE <macro>.
<宏的内容>
END-OF-DEFINITION.
```

2. 宏的调用使用

```
<macro> [<p1> <p2>....]
```

<p1> 为传入宏的参数值，在参数间至少要给予一个空格

示例：利用宏计算N 次方

```
DATA RESULT TYPE I.
DEFINE MULTI.
        RESULT = &1 ** &2.
        WRITE: / '&1 ^ &2 = ', RESULT.
END-OF-DEFINITION.
MULTI 3 4.
```

执行结果为 3 ^ 4 = 81

包含程序（Include Program）

1. 在ABAP/4 中可以使用 Include 语句加载另一个程序的全部语句，通常用于共享数据项的声明，很类似 C 的 Include 头文件的做法。

语法：

```
INCLUDE <include program file>
```

示例：

```
程序YStart 的内容如下:
***INCLUDE YSTART.
WRITE: / 'User Name = ', SY-UNAME.
WRITE: / 'Host Server = ', SY-HOST.
另一程序如下：
PROGRAM YTEST1.
INCLUDE YSTART. "载入 YSstart 的所有内容
```

执行结果：

```
User Name = MIS-CHOU
Host Server = deidv01
```

2. 全局变量声明应用

语法：

```
DATA: BEGIN OF COMMON PART [<name>],
<data 声明>
END OF COMMON PART [<name>]
```

此常使用在 Include 的文件中，如

```
***INCLUDE INCOMMON.
DATA: BEGIN OF COMMON PART NUMBERS,
        MID(8),
        MNUM TYPE I,
        END OF COMMON PART NUMBERS.
```

📖 子程序（Subroutine Procedure）

在ABAP/4 子程序的调用可分成内部调用和外部调用，前者撰写在程序中，后者存在另一程序中，通常为专存放子程序的公用程序集，可提供给不同的程序调用。

1. 子程序的声明

```
FORM <subr> [<pass>].
<subroutine statement block>
ENDFORM.
```

2. 调用的方法

(1). 内部调用

语法：

```
PERFORM <subr> [<pass>]
```

示例：

```
NUM1 = 100. NUM2 = 200.
PROFORM ADD.
FORM ADD.
  SUM = NUM1 + NUM2.
  WRITE: / 'NUM1 + NUM2 = ', SUM.
ENDFORM.
```

执行结果：

```
NUM1 + NUM2 = 300
```

(2). 外部调用另一程序

语法：

```
PERFORM <subr>(<prog>) [<pass>] [IF FOUND]
<subr>:子程序名称
<prog>:存放子程序的程序名称
IF FOUND: 找到才执行
```

示例：

```
PROGRAM FORMPOOL.
FORM HEADER.
  WRITE: / 'USER NAME: ', SY-UNAME.
ENDFORM.
在程序中调用 HEADER 子程序
PROGRAM YTEST1.
PERFORM HEADER(FORMPOOL) IF FOUND.
```

(3). 外部调用另一专存放子程序的程序文件

语法：

```
PERFORM (<fsubr>) [IN PROGRAM (<fprog>) [<pass>] [IF FOUND]
```

示例：

存放子程序的程序文件

```
PROGRAM FORMPOOL.
FORM SUB1.
  WRITE: / 'USER NAME:', SY-UNAME.
ENDFORM.
FORM SUB2.
  WRITE: / 'HOST SERVER:', SY-HOST.
ENDFORM.
```

在程序中调用 FORMPOOL 中的 SUB2 子程序

```
SUBNAME = 'SUB2'.
PROGNAME = 'FORMPOOL'.
PERFORM (SUBNAME) IN PROGRAM (PROGNAME) IF FOUND
```

📖 参数值的传递

在 ABAP/4 中参数的传递可分成

1. 参数调用（Call By Reference）

传参数时将数据的存放地址(address)传至参数中，也就是子程序中的参数变量与外部实际变量共享地址内的值，又称为传地址调用（Call By Address），若在子程序中地址中的值改变了，外部实际变量的值也会跟着改变。

语法：

```
FORM <subr> [USING <f1> <f2>...] [CHANGING <f1>...]
```

```
PERFORM <subr> [USING <f1> <f2>...] [CHANGING <f1>...]
```

Using 之后接在子程序中不会改变的变量，CHANGING 接会改变值的变量但实际上 USING 之后的参数在子程序中也可将值改变

示例：

```
SUM = 0.
NUM1 = 100. NUM2=200.
PERFORM ADD USING NUM1 NUM2 CHANGING SUM.
WRITE: / NUM1, NUM2, SUM ' SUM 由 0 变成 300
FORM ADD USING NUM1 NUM2 CHANGING SUM.
  SUM = NUM1 + NUM2.
ENDFORM.
```

执行结果:

```
100 200 300
```

2. 传值调用（Call By Value）

传参数时将数据的值复制一份至另一地址中，所以在子程序中参数变量值改变，并不会影响外部实际变数的值。

语法：

```
FORM <subr> USING VALUE(<f1>...)
```

使用 VALUE(<f1>)表示 <f1>是 Call By Value 的传递

```
PERFORM <subr> USING <f1>
```

示例：

```
SUM = 0.
NUM1 = 5.
PERFORM MULTI USING NUM1 CHANGING SUM.
WRITE: / NUM1,SUM "NUM1 值还是5, SUM 由 0 变成 120
FORM MULTI USING VALUE(NUM1) CHANGING SUM.
  SUM = 1.
  WHILE NUM1 > 1
    SUM = SUM * NUM1.
    NUM1 = NUM1 – 1.
  ENDWHILE..
ENDFORM.
```

执行结果：

```
5 120
```

3. 传值调用并返回结果（Call By Value and Return Result）

传入参数值的方式同Call By Value，但在子程序结束执行时会将传入的参数值复制一份传回给外部实际变数。

语法：

```
FORM ..... CHANGING VALUE(<f1>)
PERFORM .... CHANGING .... <f1>
```

示例：

```
SUM = 0.
NUM1 = 100. NUM2=200.
PERFORM ADD USING NUM1 NUM2 CHANGING SUM.
WRITE: / NUM1, NUM2, SUM " SUM 由 0 变成 300
FORM ADD USING NUM1 NUM2 CHANGING VALUE(S).
  S = NUM1 + NUM2.
  WRITE: / NUM1, NUM2, SUM "得到结果为 100 200 0
ENDFORM.
```

执行结果：

```
100 200 0 "在子程序中 SUM 值尚未改变
100 200 300 "返回程序时，将变量 S 的值复制给 SUM，所以 SUM 值变成 300
```

子程序的控制

1. CHECK <Condition>

CHECK 之后条件成立才继续往下子程序语句

示例：

```
PERFORM SUB1.
  CHECK NUM1 < 10.
  WRITE / NUM1.
  NUM1 = NUM1 + 1.
ENDFORM.
```

2. EXIT

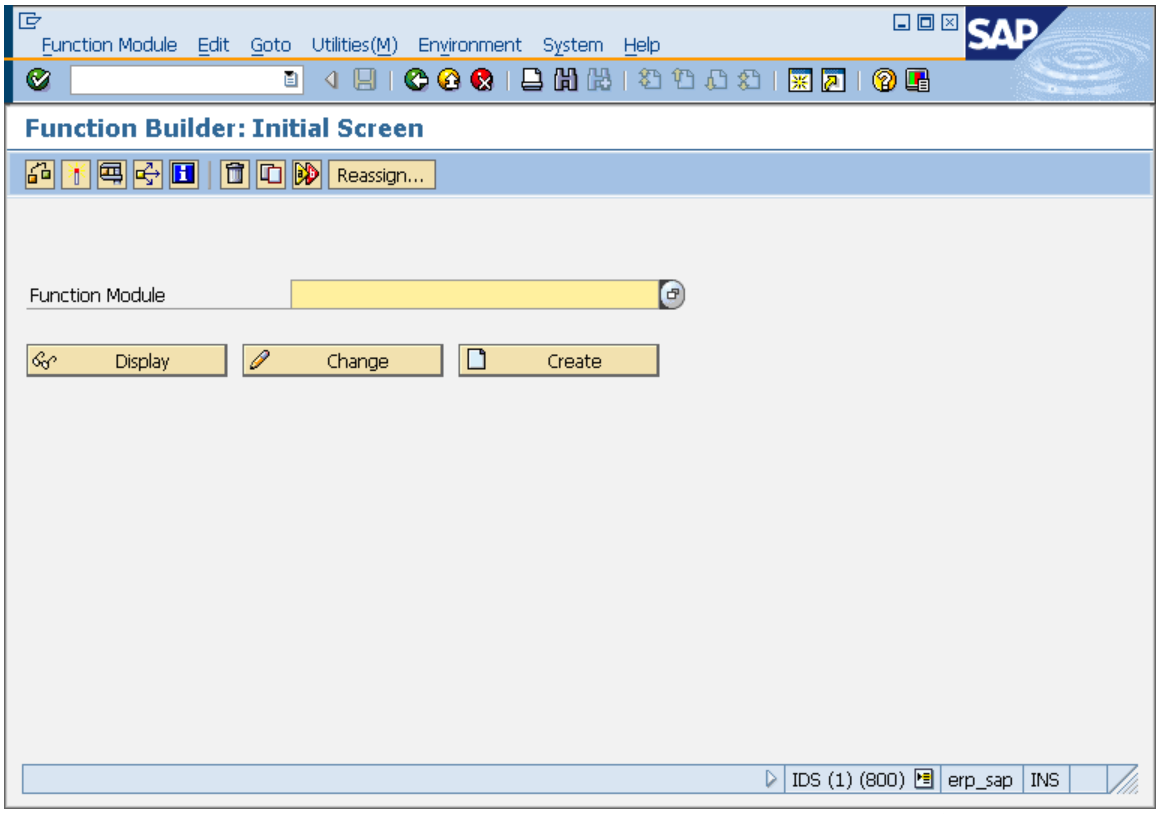
强迫结束子程序执行，返回上一层程序

函数模块（Function Module）

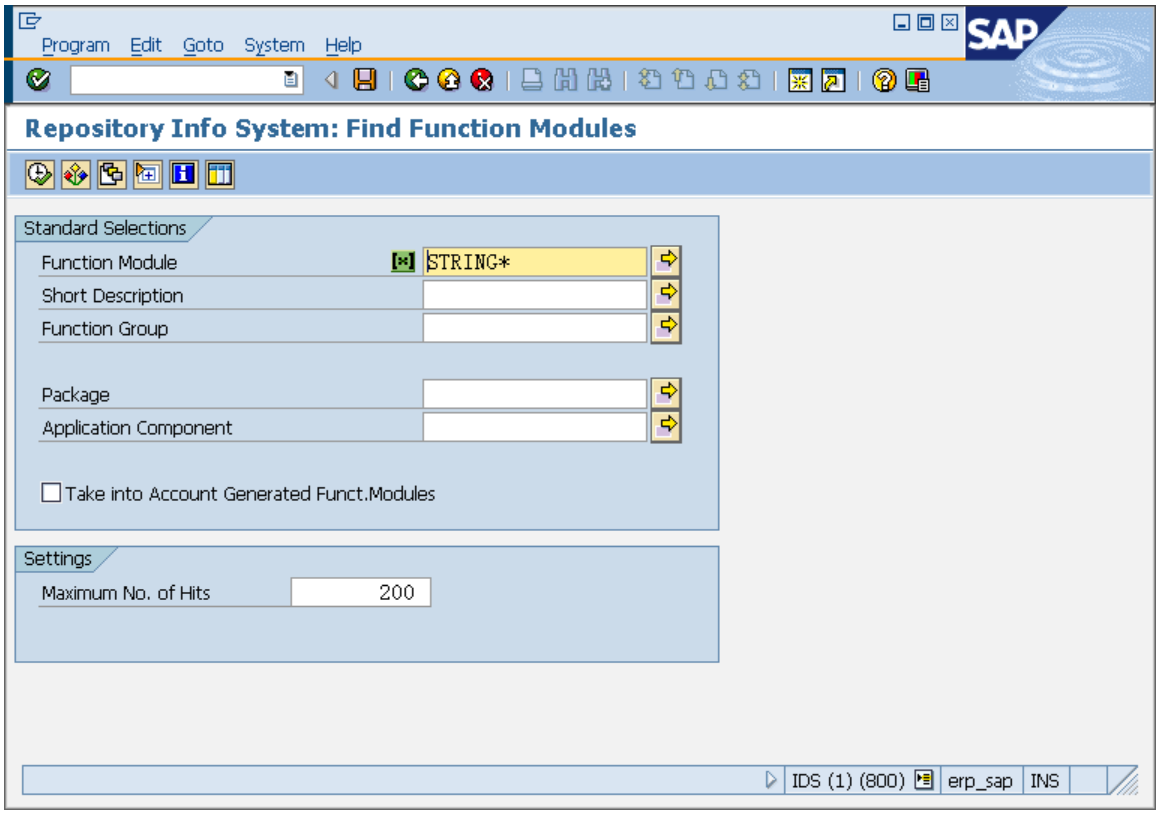
在ABAP/4 中的函数模块是储存在一个函数库中(library)，系统提供很多内设的函数模块供程序中调用也可以自行增加自己的函数模块。

1. 调用已存在的Function Module

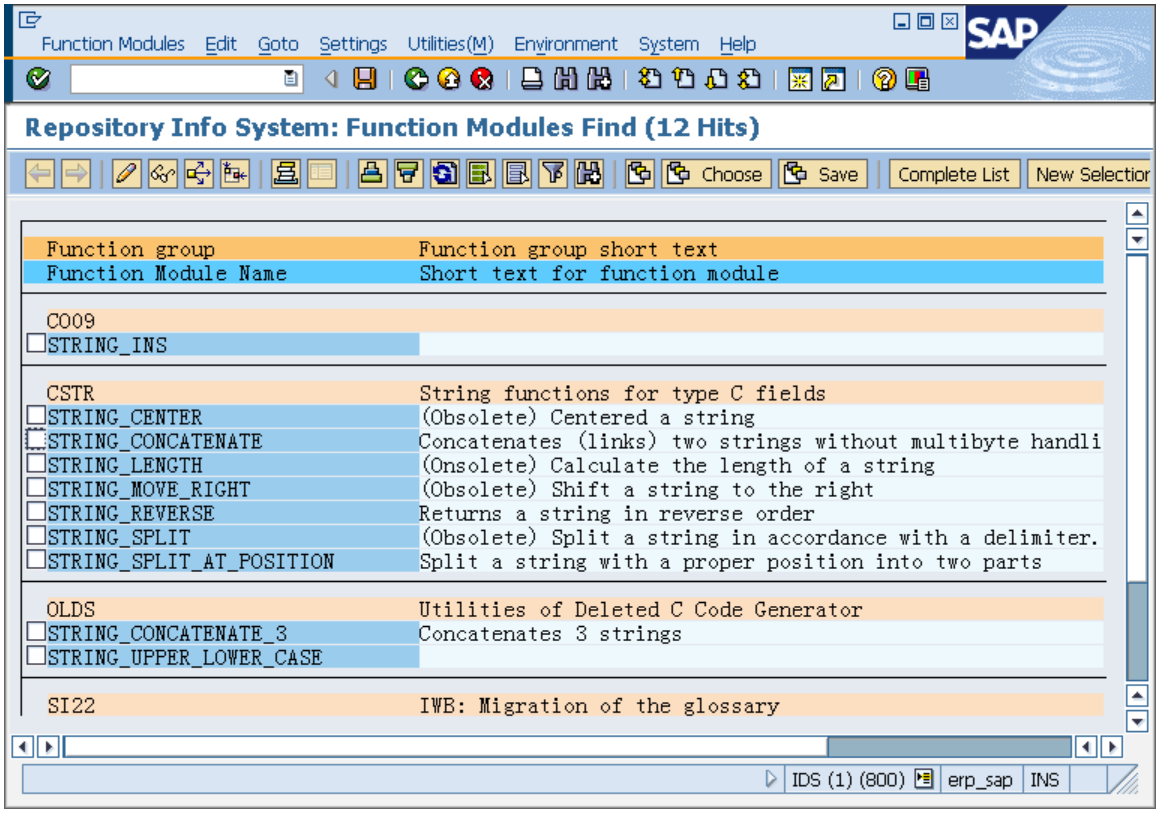
(1). 在ABAP/4 开发工作台界面中选择函数编制器（事务码SE37）可见以下界面：



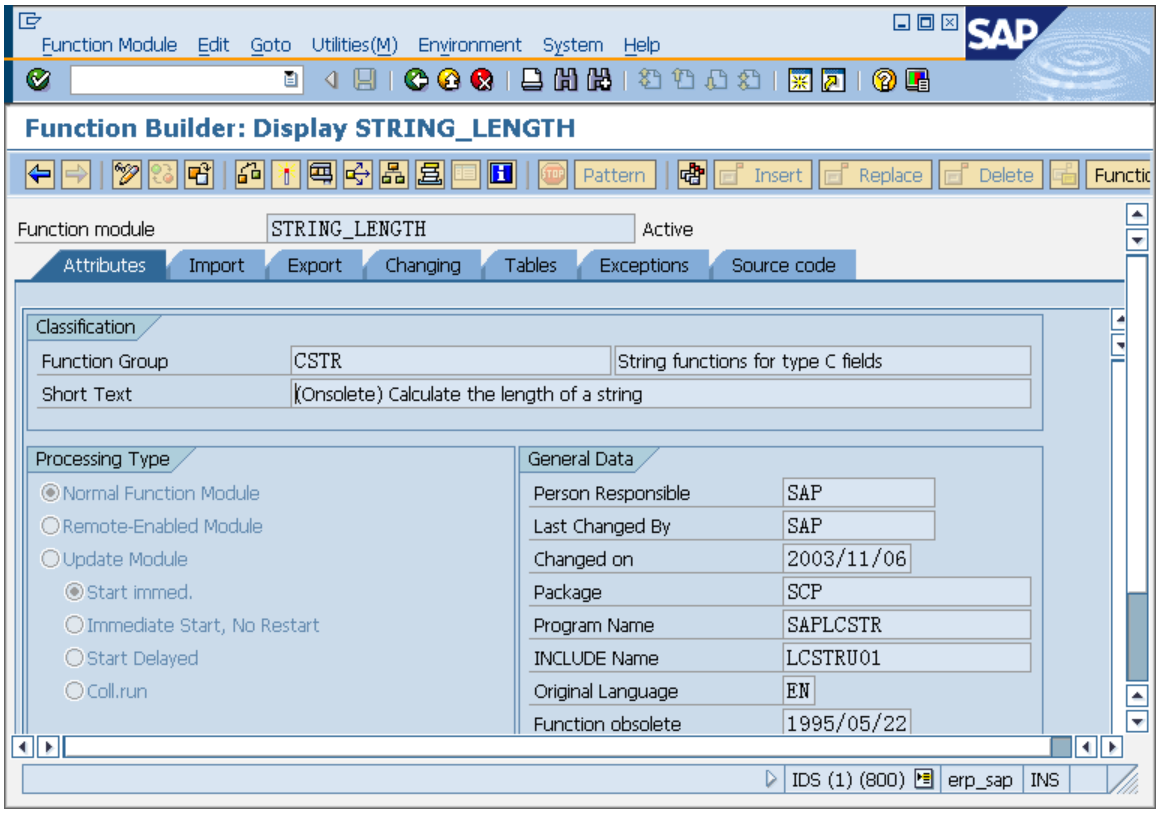
(2). 选择菜单“Utility”中的“Find”（快捷键Ctrl+F）中输入要寻找的函数模块名称，如输入STRING*，为找出前五个字符为 STRING 的函数模块：



(3). 输入后按下左上的“Execute”按钮  ，可见以下界面：



(4). 选择要查询或修改的函数名称，如选择 STRING_LENGTH，如要查询可选择“Display”按钮  ，可见函数模块之各项参数设定：



<1>.Import: 传入的参数名称，但实际在程序中使用时刚好与Export 相颠们

<2>.Export: 传回的参数名称，程序中变成 Import 的使用

<3>.Changing:使用Call By Value and Return Result 方法的参数

<4>.Tables: 使用的内表参数

<5>.Exceptions: 错误处理参数

如 STRING_LENGTH 为一传回字符串长度的函数，其设定的参数如下：

Import: String 传入一个字符串数据

Export: Length 传回的字符串长度值

(5). 要查看程序内容可按下“Source Code”，显示其程序内容如下：

```
function string_length.
length = strlen(string).
endfunction.
```

(6). 函数的调用

语法：

```
CALL FUNCTION <module>
IMPORTING F1=a1....
EXPORTING F1=a1....
CHANGING F1=a1...
TABLES F1=a1...
EXCEPTIONS F1=a1...
```

示例：

```
DATA: TEXT(20),
      LEN TYPE I.
TEXT = 'ABCDEFGHIJ'.
CALL FUNCTION 'STRING_LENGTH'
EXPORTING STRING = TEXT
IMPORTING LENGTH = LEN.
WRITE / LEN.
```

注意 EXPORTING 与IMPORTING 刚好颠倒，

执行所得结果为 10

3.屏幕输入命令

在 ABAP/4 中要自屏幕输入变量数据的内容，使用的命令是 PARAMETERS 及 SELECTION-OPTIONS：

1. PARAMETER：输入一个变量或字段内容
2. SELECT-OPTIONS：使用条件筛选界面来输入数据

📖 PARAMETERS命令

基本的输入命令，类似于 BASIC 的INPUT 命令，但无法使用 F 格式(浮点数)

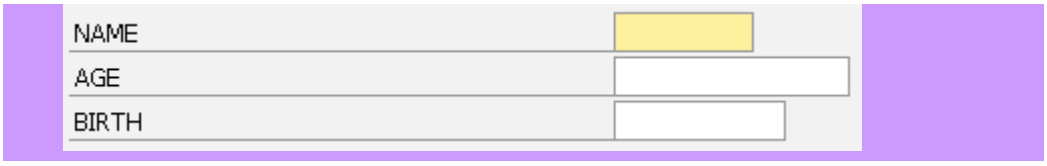
语法：

```
PARAMETERS <p> [DEFAULT <f>] [LOWER CASE] [OBLIGATORY] [AS CHECKBOX] [RADIOBUTTON GROUP <rad>]
```

示例：

```
PARAMETERS: NAME(8),
      AGE TYPE I,
      BIRTH TYPE D.
```

执行结果：



在日期的输入格式上为 MM/DD/YY，MM/DD/YYYY，MMDDYY或MMDDYYYY，如输入 020165 表示 1965 年 02 月 01 日，与 02/01/65 的输入是一样的，日期输入范围为公元 1950 年至 2049 年

1. DEFAULT

设定输入的默认值

示例：

```
PARAMETERS: COMPANY(20) DEFAULT 'DELTA',
            BIRTH TYPE D DEFAULT '19650201'.
```

2. LOWER CASE

ABAP/4 预设是将字符串输入值自动转换为大写，加上此参数会将输入的数据转成小写。

3. OBLIGATORY

强制要求输入，屏幕上会出现一个 ☑ ，使用者必须要输入才可以。

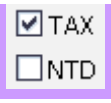
4. AS CHECKBOX

输入 CHECKBOX 的格式

示例：

```
PARAMETERS: TAX AS CHECKBOX DEFAULT 'X',
            NTD AS CHECKBOX.
```

执行结果:



5. RADIOBUTTON GROUP <rad>

输入 RADIO BUTTON GROUP 的方式

示例：

```
PARAMETERS: BOY RADIOBUTTON GROUP SEX DEFAULT 'X',
            GIRL RADIOBUTTON GROUP SEX.
```

执行结果：

📖 SELECT-OPTIONS

条件筛选检查条件输入界面命令，输入条件后可配合 SELECT 命令自 TABLE 读取符合条件的数据，直接执行或放入内表中，条件有四个参数：

1. SIGN:

I: 表筛选条件符合的资料

E: 表筛选条件不符合的资料

2. OPTION: 比较的条件符号

EQ(等于)，NE(不等于)，GT(大于)，LE(小于)，CP(包含)，NP(不包含)

3. LOW: 最小值

4. HIGH: 最大值

语法：

```
SELECT-OPTIONS <check-option> FOR <table-field>
```

示例：

```
TABLES SPFLI.
SELECT-OPTIONS AIRLINE FOR SPFLI-CONNID.
```

将条件的输入值存放入 AIRLINE，筛选选择为 SPFLI 表中的 CONNID 字段

执行结果：

可直接输入起始范围或按下选择界面，输入完后按下左上角的执行键  。

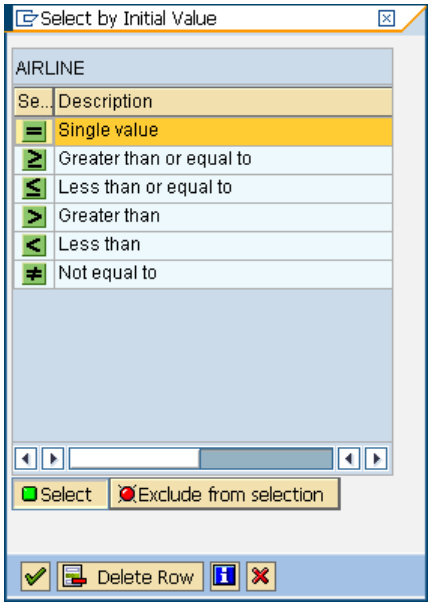
条件输入选择界面

1. 自表中选取

按下输入项的右边表选择按钮  ，调出表中数据项，选取开始和结束的范围。

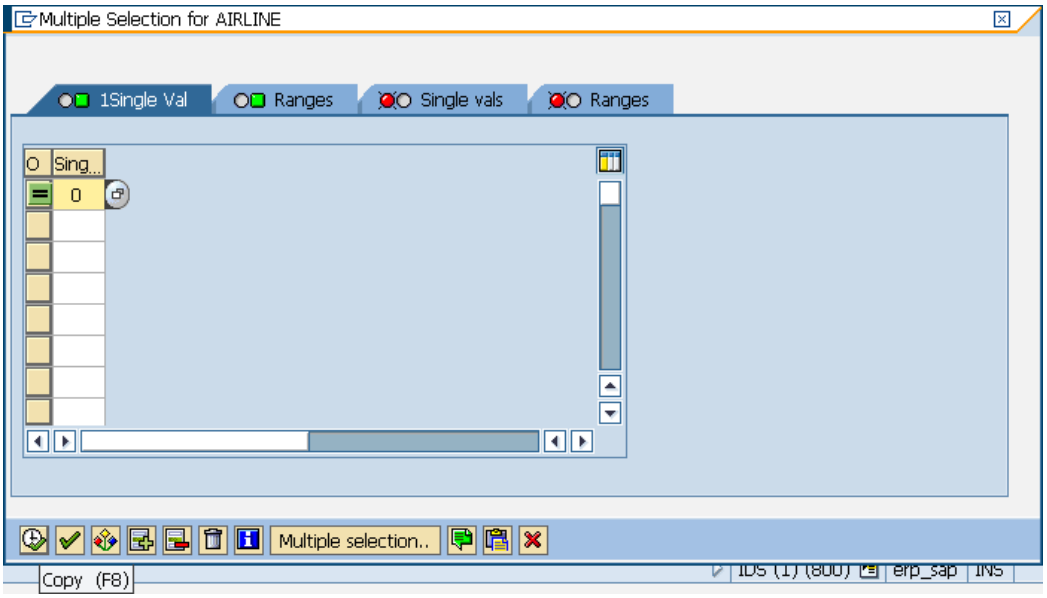
2. Selection Options

按下菜单“Edit”下的“Selection options”（快捷键F2），输入Selection Option参数内容，屏幕如下：



3. Multi-Options 输入

按下最右边的 Multi-Options 输入键  ，输入条件选取的范围，界面如下：



条件输入完后按下“Copy”按键。

改变条件输入格式

1. DEFAULT <begin> TO <end>

设定开始结束范围输入默认值

示例：

```
SELECT-OPTION AIRLINE FOR SPFLI-CONNID
DEFAULT '2042' TO '4555'.
```

2. NO-EXTENSION

设定不要Multi-Option 输入界面

3. NO INTERVALS

设定不要区间范围输入界面

4. LOWER CASE

输入转换成大写

5. OBLIGATORY

强制要求输入

配合SELECT命令

条件输入完后要将符合条件的数据筛选出来，可配合使用 SELECT 命令

1. 使用WHERE <条件式>

示例：

```
SELECT-OPTIONS AIRLINE FOR SPFLI-CONNID.
SELECT * FROM SPFLI WHERE CONNID IN AIRLINE.
  WRITE: / CONNID, FROMCITY, TOCITY.
ENDSELECT.
```

2. 使用 CHECK 参数

示例：

```
SELECT-OPTIONS AIRLINE FOR SPFLI-CONNID.
SELECT * FROM SPFLI.
  CHECK AIRLINE.
  WRITE: / CONNID, FROMCITY, TOCITY.
ENDSELECT.
```

3. 使用 IF ... IN 语句

示例：

```
SELECT-OPTIONS AIRLINE FOR SPFLI-CONNID.
SELECT * FROM SPFLI.
  IF SPFLI-CONNID IN AIRLINE.
    WRITE: / CONNID, FROMCITY, TOCITY.
  ENDIF
ENDSELECT.
```

📖 SELECTION-SCREEN

1. 产生空白列

语法：

```
SELECTION-SCREEN SKIP [<n>]
```

示例：

```
SELECTION-SCREEN SKIP 2.
```

产生两列空白列

2.产生底线

语法：

```
SELECTION-SCREEN ULINE / <pos>(length)
```

示例：

```
SELECTION-SCREEN ULINE /10(30).
```

自第10 格开始产生长度30 的底线

3.印出备注说明

语法:

```
SELECTION-SCREEN COMMENT / <pos>(length) <name>
```

示例：

```
REMARK = 'Pls enter your name'.
SELECTION-SCREEN COMMENT /10(30) REMARK.
```

4. 同一列中输入数个数据项

语法:

```
SELECTION-SCREEN BEGIN OF LINE.
.....
SELECTION-SCREEN END OF LINE.
```

示例：

```
SELECTION-SCREEN BEGIN OF LINE.
SELECTION-SCREEN POSITION 20.
PARAMETERS NAME(10).
SELECTION-SCREEN POSITION 40.
PARAMETERS BIRTH TYPE D.
SELECTION-SCREEN END OF LINE.
```

在20 格输入NAME 内容, 40 格输入 BIRTH 的内容

5. 绘出BLOCK PANEL

语法:

```
SELECTION-SCREEN BEGIN OF BLOCK <block>
[WITH FRAME [TITLE <title>].
.....
SELECTION-SCREEN END OF BLOCK <block>.
```

示例：

```
SELECTION-SCREEN BEGIN OF BLOCK RADIO
WITH FRAME .
PARAMETER R1 RADIOBUTTON GROUP GR1.
PARAMETER R2 RADIOBUTTON GROUP GR1.
PARAMETER R3 RADIOBUTTON GROUP GR1.
SELECTION-SCREEN END OF BLOCK RADIO.
```

4. 标准报表

一个典型的报表程序是由许多的程序区块(Code Block)所组成，在区块间最好能加上一些说明以利程序可读性，一个典型的报表程序格式如下：

```
* PROGRAM SOURCE HEADER : 说明程序名称及目的
* Program Name:
* Description:
* Date/Author:
```

```
* Table Update:
* Special Logic:
* Include:
*---------------------------------------------------------------
* MODIFICATION LOG : 程序修改更新记录
*---------------------------------------------------------------
* ChangeDate Programmer Request Description
* ========== ============= ============
============================
* NEW PROGRAM
*---------------------------------------------------------------
* REPORT NAME : 声明程序名称及报表格式,
*---------------------------------------------------------------
REPORT Z_____
NO STANDARD PAGE HEADING
MESSAGE-ID __ " 所使用的MESSAGE
LINE-COUNT ___ " 每页报表列数
LINE-SIZE ___. " 每页报表宽度
* TABLE DESCRIPTION : 声明程序会使用的TABLE
*---------------------------------------------------------------
TABLES:
* DATA : 声明程序所使用的变量及自定型态
*---------------------------------------------------------------
TYPES:
DATA:
* SELECTION SCREEN / OPTION / PARAMETER : 屏幕输入报表筛选条件
*---------------------------------------------------------------
SELECTION-SCREEN BEGIN OF BLOCK ____
SELECT-OPTIONS:
SELECTION-SCREEN END OF BLOCK ___
49
* INITIALIZATION : 启动程序开始执行, 如SELECT-OPTION 及
PARAMETER
*---------------------------------------------------------------
INITIALIZATION.
INCLUDE ____.
* AT START SELECTION : 输入结束后启动的区块, 如按下<F8>
*---------------------------------------------------------------
START-OF-SELECTION.
SET PF-STATUS ____. " 指定报表执行时所用的 GUI-STATUS 名称
PERFORM READ_DATA.
PERFORM PROCESS_DATA.
PERFORM PRINT_DATA.
PERFORM PRINT_SUMMARY.
* AT USER Commaand : 执行在GUI-STATUS 中自定的命令
*---------------------------------------------------------------
AT USER_COMMAND.
* AT LINE SELECTION : 由在报表中按下<F2>或Double-Click 启动
*---------------------------------------------------------------
AT LINE-SELECTION.
```

```
* TOP OF PGAE：每页开始打印时执行，用于定义报表表头
*---------------------------------------------------------------------
* END OF PAGE：报表打印完最后一页后启动
*---------------------------------------------------------------------
END-OF-PAGE
* END OF SELECTION：在结束打印数据后启动，如可用来印出USER 输入的条
件
*---------------------------------------------------------------------
END-OF-SELECTION.
INCLUDE _____
* FORM：撰写程序中所使用到的子程序
*---------------------------------------------------------------------
* Read Data：自TABLE 读取数据放入Internal Table
*---------------------------------------------------------------------
50
FORM READ_DATA.
SELECT * FROM _______
INTO ________
WHERE ________.
IF SY-SUBRC = 0.
ENDIF.
APPEND _____. " 增加Internal Table 元素
ENDSELECT.
ENDFORM.
* Process Data：处理Internal Table 的数据，如排序及汇总
*---------------------------------------------------------------------
FORM PROCESS_DATA.
ENDFORM.
* Print Data：依序输出 Internal Table 的数据
*---------------------------------------------------------------------
FORM PRINT_DATA.
ENDFORM.
* Print Summary：印出数值资料加总
*---------------------------------------------------------------------
FORM PRINT_SUMMARY.
ENDFORM.
* Include Program：列出所含入的其它程序source code，如子程序
*---------------------------------------------------------------------
INCLUDE _____
INCLUDE _____
```

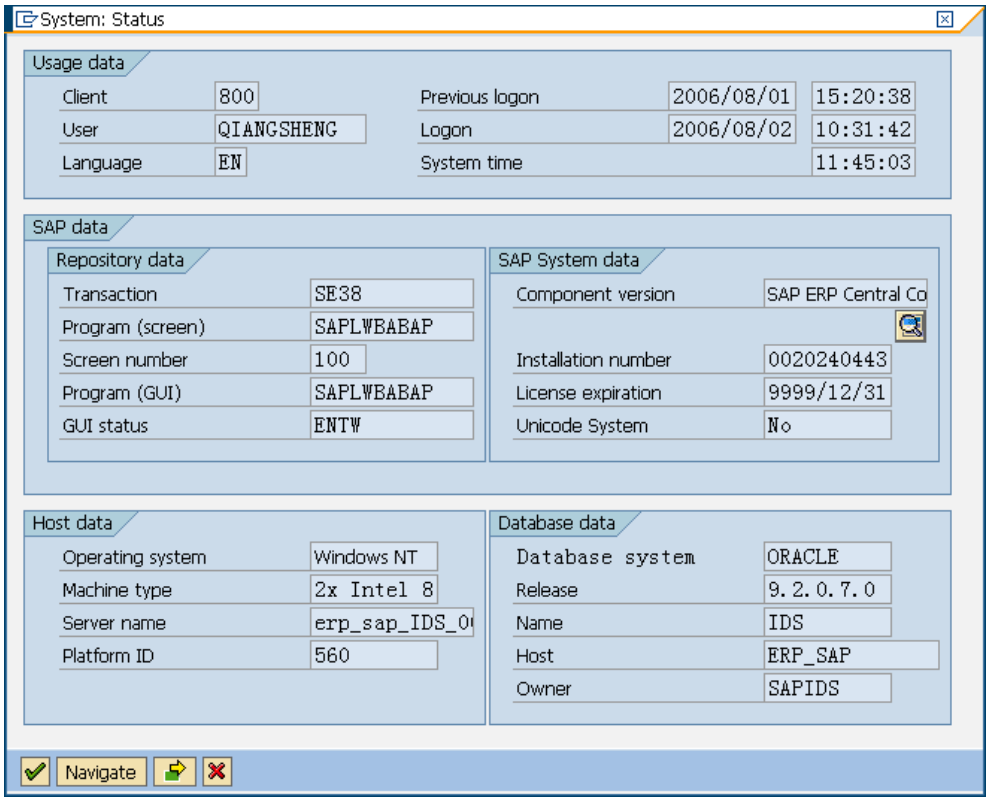
5. 写BDC程序

BDC程序(Batch Data Communication Program，批量数据通讯程序) 是ABAP/4 用来加载数据变更SAP数据库的方法，先将要输入的数据存在BDC表中，使用 CALL TRANSACTION 命令调用 R/3 输入界面，将输入所需数据项自BDC表中依序取入，最后送出按键句柄，如 /11 表按下<F11>存盘，此方法用在从不同系统转入 R/3 系统之数据迁移(Data Migration)，或者也

可使用在 Drill-Down 报表的撰写方式中。

5.1 事务状态屏幕

当用户使用 SAP R/3 来输入数据时，SAP R/3 使用事务来控制工作的执行，每个事务都会包含一些不同的屏幕，每个屏幕被定义为一个程序名称及一个屏幕号，在执行SAP R/3 时选择菜单“System”中的“Status”可看到如下的界面：



如图：事务码是 SE38，程序名称是 SAPLWBABAP，屏幕号是 100

5.2 BDC表

📖 BDC表结构

BDC表是一个结构，用来存放要放入输入界面的数据，包含有以下的字段：

Field Name Type Description
Program Char(8) Program name of Transaction
Dynpro Char(4) Screen number of Transaction
Dynbegin Char(1) Indicator for new Screen
Fnam Char(35) Name of Database Field from Screen
Fval Char(80) Value to Submit to Field

可在程序开始之初声明一个内表使用 BDCDATA 的结构：

```
DATA BEGIN OF INT_BDC OCCURS 0.
INCLUDE STRUCTURE BDCDATA.
DATA END OF INT_BDC.
```

如在以上的界面中要输入 VBAK-KUNNR 及 VBAK-NAME1 两个字段的内容，分别要填入'34051920' 及 '台达电子'，其 BDC表内容如下：

Program Dynpro Dynbegin Fnam Fval
SAPMV45A 0300 X
VBAK-KUNNR 34051920
VBAK-NAME1 台达电子
BDC_OKCODE /nn
BDC_OKCODE 字段的值就是结束输入时所要按下的键盘句柄.

常用的BDC Okcode（OK码）

OKCODE 声明

/nn 功能键nn，Fnn
/00 回车
/8 F8，继续或执行
/11 F11，Post提交
%EX 退出
BACK F3，返回上一屏
DLT 删除
PICK 双击
SAVE F11，保存

存入BDC表的资料

首先我们需建立两个子程序，BDC_SCREEN 是来存入Program，Dynpro 和 Dynbegin 三个字段，也就是输入界面的程序名称及屏幕号，BDC_FIELD 用来存入 Fnam 及 Fval 两个字段，也就是输入界面所需填入的各个字段内容，程序内容如：

*** Add BDC Screen Field Data**

```
FORM BDC_SCREEN TABLES P_BDC STRUCTURE BDCDATA
        USING P_PROGRAM P_SCREEN.
  CLEAR P_BDC.
  P_BDC-PROGRAM = P_PROGRAM.
  P_BDC-DYNPRO = P_SCREEN.
  P_BDC-DYNBEGIN = 'X'.
  APPEND P_BDC.
```

```
    ENDFORM.
```

*** Add BDC Field Data**

```
    FORM BDC_FIELD TABLES P_BDC STRUCTURE BDCDATA
            USING P_NAME P_VALUE.
      CLEAR P_BDC.
      CASE P_VALUE.
        WHEN ''.
        WHEN OTHERS.
          P_BDC-FNAM = P_NAME.
          P_BDC-FVAL = P_VALUE.
          APPEND P_BDC.
      ENDCASE.
    ENDFORM.
```

接下来在程序中去调用这两个子程序来填入数据，程序如下：

```
    PERFORM BDC_SCREEN TABLES INT_BDC:
    USING 'SAPMV45A' '0300'.
    PERFORM BDC_FIELD TABLES INT_BDC:
    USING 'VBAK-KUNNR' '34051920',
    USING 'VBAK-NAME1' '台达电子',
    USING 'BDC_OKCODE' '/00'. "  保存并结束
```

5.3 Call Transaction 命令

将数据依序填入 BDC 表后要写入 R/3 的数据库时要使用 CALL TRANSACTION 命令，命令格式如下：

```
CALL TRANSACTION WITH <tcode> USING <BDC Table> MODE <Display Mode>
<Display Mode> 可分成:
A Show all Screen
E Show only Screen with Error
N Show no Screen
```

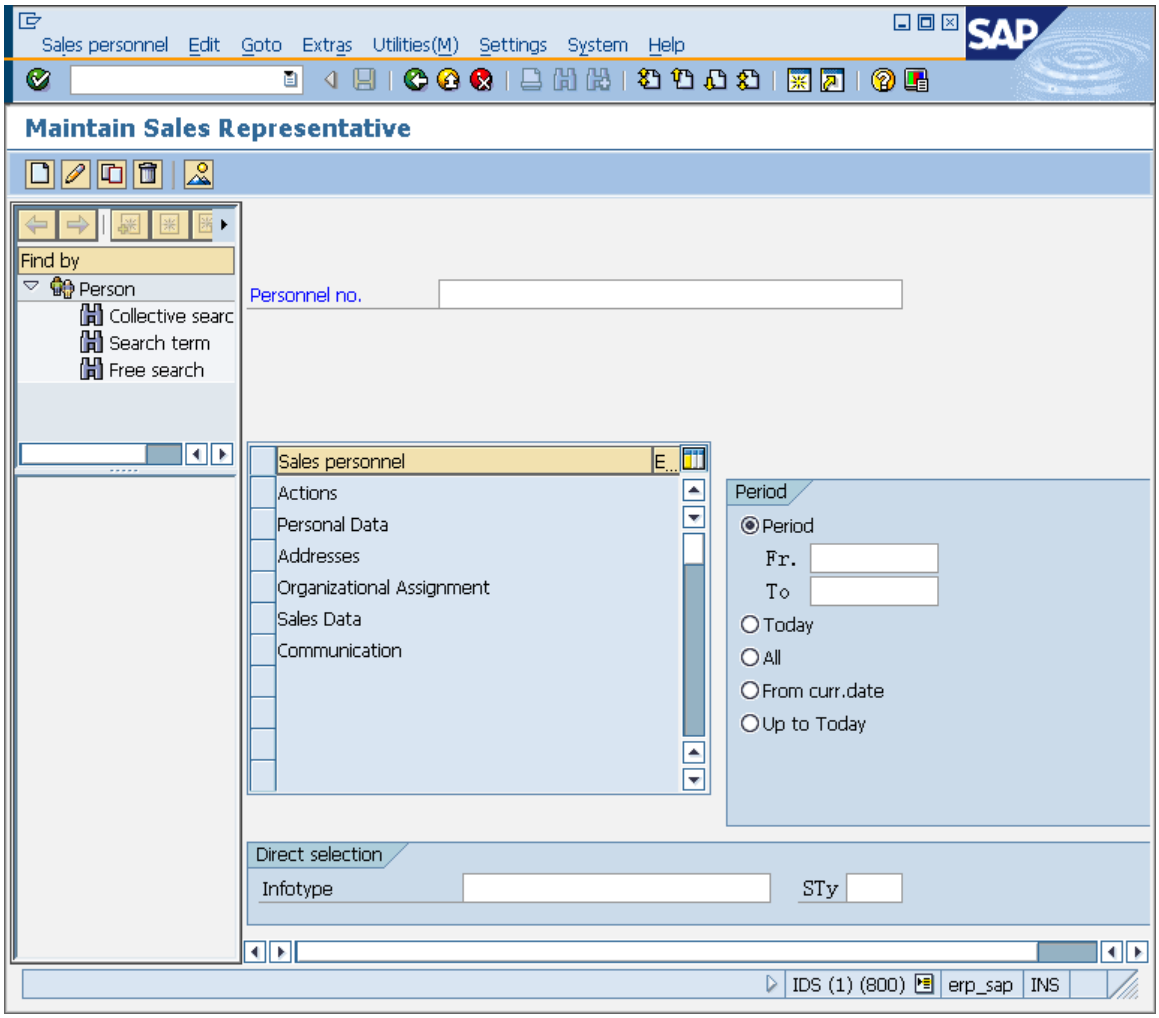
如：

```
CALL TRANSACTION 'VA03'
USING INT_BDC
MODE 'E'.
```

5.4 BDC程序示例

例如我们想经由 BDC 程序将 Sales Empolyee 的数据由 ASCII文本文件去更新 R/3 数

据库 Sales Empolyee 的 Persolnal Data，会有以下两个界面要输入：



Program Name 是 SAPMP50A，Screen Number 1000，Transaction Code 是 PAL3

Program Name 是 SAPMP50A，Screen Number 2042，Transaction Code 是 PAL3

在Screen 1000 要填入的资料是

说明	字段名	值
Personal Number	RP50G-PERNR	242147
Info Type	RP50G-CHOIL	Personal Data

在Screen 2042 要填入的资料是

说明	字段名	值
From Date	P002-BEGDA	01/13/1969
To Date	P002-ENDDA	12/31/9999
Last Name	P002-NACHN	Dora
First Name	P002-VORNA	Cheng

欲转入的ASCII文本文件名是 empoly.txt，其格式为

说明	长度	位置	值
Personal Number	10	1	242147
Info Type	20	11	Personal Data
From Date	10	31	01/13/1969
To Date	10	41	12/31/9999
Last Name	20	61	Dora
First Name	20	81	Cheng

程序如下：

```
* 程序头说明PROGRAM SOURCE HEADER
* Program Name: ZTBDC00
* Description: Change Sales Empolyee Data Using BDC Program
* Date/Author: 1998/06/10 周庆日
* Table Update:
* Special Logic:
* Include:
*---------------------------------------------------------------
* 变更日志MODIFICATION LOG
*---------------------------------------------------------------
* ChangeDate Programmer Request Description
* ==================================
* 1998/06/10 Chou NEW PROGRAM
*---------------------------------------------------------------------------
* 报表名REPORT NAME
*---------------------------------------------------------------
REPORT ZTBDC00
NO STANDARD PAGE HEADING
MESSAGE-ID ZZ
LINE-COUNT 60
LINE-SIZE 80.
* 定义数据表和变量DATA
*---------------------------------------------------------------
TABLES: RP50G,P002.
DATA: BEGIN OF INT_BDC OCCURS 0.
        INCLUDE STRUCTURE BDCDATA.
DATA: END OF INT_BDC.
* Work Internal Table
DATA: BEGIN OF IN_REC,
        PERNR LIKE RP50G-PERNR,
        CHOIL LIKE RP50G-CHOIL,
        BEGDA LIKE P002-BEGDA,
        ENDDA LIKE P002-ENDDA,
        NACHN LIKE P002-NACHN,
        VORNA LIKE P002-VORNA,
        END OF IN_REC.
DATA: P_FILE(30) VALUE 'empoly.txt'. "ASCII文本文件名
```

```
* 初始化INITIALIZATION
*---------------------------------------------------------------------
INITIALIZATION.
* AT START SELECTION
*---------------------------------------------------------------------
START-OF-SELECTION.
  PERFORM READ_DATA.
  PERFORM PROCESS_DATA.
* END OF SELECTION
*---------------------------------------------------------------------
END-OF-SELECTION.
PERFORM CLEAN_UP.
* FORM
*---------------------------------------------------------------------
* 打开ASCII文本文件读数据Read Data OPEN ASCII Text File
*---------------------------------------------------------------------
FORM READ_DATA.
        OPEN DATASET P_FILE FOR INPUT IN TEXT MODE.
        IF SY-SUBRC NE 0.
          MESSAGE E999 WITH '无法打开 ' P_FILE.
        ENDIF.
ENDFORM.
* 处理数据Process Data
*---------------------------------------------------------------------
FORM PROCESS_DATA.
  DO.
    READ DATASET P_FILE INTO IN_REC. "把数据添加到内表
    IF SY-SUBRC NE 0.
      EXIT.
    ENDIF.
    PERFORM BUILD_BDC. "添加到BDC表
    PERFORM SUBMIT_BDC. "调用事务码修改数据
  ENDDO.
ENDFORM.
*---------------------------------------------------------------------
* 把数据添加到BDC表Add Data to BDC Table
FORM BUILD_BDC.
  PERFORM BDC_SCREEN TABLES INT_BDC
        USING 'SAPMP50A' '1000'.
  PERFORM BDC_FIELD TABLES INT_BDC:
        USING 'RP50G-PERNR' IN_REC-PERNR,
        USING 'RP50G-CHOIL' IN_REC-CHOIL,
        USING 'BDC_OKCODE' '/00'.
  PERFORM BDC_SCREEN TABLES INT_BDC
        USING 'SAPMP50A' '2042'.
  PERFORM BDC_FIELD TABLES INT_BDC:
        USING 'P0002-BEGDA' IN_REC-BEGDA,
        USING 'P0002-ENDDA' IN_REC-ENDDA,
        USING 'P0002-NACHN' IN_REC-NACHN,
        USING 'P0002-VORNA' IN_REC-VORNA,
```

```
        USING 'BDC_OKCODE' '/11'.
ENDFORM.
*---------------------------------------------------------------------
* 调用事务码执行数据变更Call Transaction Code to Execute Change Data
FORM SUBMIT_BDC.
  CALL TRANSACTION 'PAL3' USING INT_BDC
  MODE 'N'.
  REFRESH INT_BDC. "将BDC Table 资料清除
ENDFORM.
*---------------------------------------------------------------------
* 关闭ASCII文本文件
FORM CLEAN_UP.
  CLOSE DATASET P_NAME.
  IF SY-SUBRC NE 0.
    MESSAGE E999 WITH '无法关闭 ' P_FILE.
  ENDIF.
ENDFORM.
*---------------------------------------------------------------------
* 增加BDC屏幕字段数据Add BDC Screen Field Data
*---------------------------------------------------------------------
FORM BDC_SCREEN TABLES P_BDC STRUCTURE BDCDATA
        USING P_PROGRAM P_SCREEN.
  CLEAR P_BDC.
  P_BDC-PROGRAM = P_PROGRAM.
  P_BDC-DYNPRO = P_SCREEN.
  P_BDC-DYNBEGIN = 'X'.
  APPEND P_BDC.
ENDFORM.
* Add BDC Field Data
*---------------------------------------------------------------------
FORM BDC_FIELD TABLES P_BDC STRUCTURE BDCDATA
        USING P_NAME P_VALUE.
  CLEAR P_BDC.
  CASE P_VALUE.
    WHEN ''.
    WHEN OTHERS.
      P_BDC-FNAM = P_NAME.
      P_BDC-FVAL = P_VALUE.
      APPEND P_BDC.
  ENDCASE.
ENDFORM.
```

6. OLE自动化

OLE(Object Linking and Embendding)是由 Microsoft 所制定的标准，可以让一个程序经由此去整合另一程序的对象(Object)，OLE Automation 是其中的一部分，经由既定的语法格式去产生连结的对象数据，可以分成Objects of Application、Call its Methods和Set and Get Object Properties。

在ABAP/4程序中使用OLE自动化

在 ABAP/4 中提供了许多的OLE 自动化接口，让 ABAP/4 程序可经由接口去整合其它应用软件(如WORD、EXCEL等)至 R/3 系统中，要知道有哪些的接口可使用，在工作台中选择Development->Programming Environ->OLE2->OLE2 Object Browser中去查询，至于更详尽的数据则需参考 Microsoft 发行的 OLE 2.0 参考或MSDN相关的资料。

应用程序对象

要使用 OLE自动化之前需在 ABAP/4 程序中先产生对象，命令是

```
CREATE OBJECT <对象名称> <对象函式库>
```

如要建立一个 EXCEL 5.0 的对象名称 application:

```
INCLUDE OLE2INCL.
DATA : APPLICATION TYPE OLE2_OBJECT.
CREATE OBJECT APPLICATION 'Excel.application'.
```

对象属性

对象产生后就可以设定其属性(Property)了，命令是

```
SET PROPERTY OF <对象名称> <属性> = <Value>
```

如设定对象APPLICATION 可在屏幕上显示：

```
SET PROPERTY OF APPLICATION 'Visible' = 1.
```

如果想获得目前对象属性的值，命令如下：

```
DATA: VISIBLE TYPE I.
GET PROPERTY OF APPLICATION 'Visible' = VISIBLE.
```

调用方法

命令格式：

```
CALL METHOD OF <对象名称> <Method> = <Value>
```

如要存放 R/3 的数据至 EXCEL 的单元格(CELL)中，在ABAP/4 的命令如下：

```
INCLUDE OLE2INCL.
DATA: APPLICATION TYPE OLE2_OBJECT,
      WORKBOOK TYPE OLE2_OBJECT,
      SHEET TYPE OLE2_OBJECT,
      CELLS TYPE OLE2_OBJECT.
CREATE OBJECT APPLICATION 'Excel.application'.
SET PROPERTY OF APPLICATION 'Visible' = 1.
```

```
* 声明一EXCEL 工作底稿档(WORKBOOK)
CALL METHOD OF APPLICATION 'Workbooks' = WORKBOOK.
* 增加一新的工作表(SHEET)，编号是 1 号
CALL METHOD OF WORKBOOK 'Add'.
CALL METHOD OF APPLICATION 'Worksheets' = SHEET
EXPORTING #1 = 1.
* 设定此工作表开启使用
CALL METHOD OF SHEET 'Activate'.
```

释放对象

对象使用完后必须自内存中释放，命令是：

```
FREE OBJECT <对象>
```

如：

```
FREE OBJECT APPLICATION.
```

将对象 APPLICATION 自内存释放

使用OLE增加数据

以要将KNA1(Customer General Data)中的资料存入 EXCEL 的单元格(CELL)为例：

```
FORM FILL_SHEET.
DATA: ROW_MAX TYPE I VALUE 256,
      ROWS TYPE I VALUE 1.
INDEX TYPE I.
FIELD-SYMBOLS: <NAME>.
SELECT * FROM KNA1.
  ROWS = ROWS + 1. "至第ROWS 列
  INDEX = ROW_MAX * ( ROWS - 1 ) + 1.
  DO 10 TIMES. "如要每一列放入10 个存格的数据
    ASSIGN COMPONENT SY-INDEX OF STRUCTURE KNA1 TO <NAME>.
    CALL METHOD OF SHEET 'Cells' = CELLS EXPORTING #1 = INDEX.
    SET PROPERTY OF CELLS 'Value' = <NAME>.
    ADD 1 TO INDEX.
  ENDDO.
ENDSELECT.
```

完整的程序

```
REPORT EXCEL.
```

```
INCLUDE OLE2INCL.
DATA: APPLICATION TYPE OLE2_OBJECT,
        WORKBOOK TYPE OLE2_OBJECT,
        SHEET TYPE OLE2_OBJECT,
        CELLS TYPE OLE2_OBJECT.
CREATE OBJECT APPLICATION 'Excel.application'.
SET PROPERTY OF APPLICATION 'Visible' = 1.
* 声明一EXCEL 工作簿文件(WORKBOOK)
CALL METHOD OF APPLICATION 'Workbooks' = WORKBOOK.
* 增加一新的工作表(SHEET)，编号是 1 号
CALL METHOD OF WORKBOOK 'Add'.
CALL METHOD OF APPLICATION 'Worksheets' = SHEET
EXPORTING #1 = 1.
* 设定此工作表开启使用
CALL METHOD OF SHEET 'Activate'.
PERFORM FILL_SHEET.
FREE APPLICATION.
*---------------------------------------------------------------------------------------
FORM FILL_SHEET.
DATA: ROW_MAX TYPE I VALUE 256,
        ROWS TYPE I VALUE 1.
INDEX TYPE I.
FIELD-SYMBOLS: <NAME>.
SELECT * FROM KNA1.
  ROWS = ROWS + 1. "至第ROWS 列
  INDEX = ROW_MAX * ( ROWS - 1 ) + 1.
  DO 10 TIMES. "如要每一列放入10 个存格的数据
    ASSIGN COMPONENT SY-INDEX OF STRUCTURE KNA1 TO <NAME>.
    CALL METHOD OF SHEET 'Cells' = CELLS
    EXPORTING #1 = INDEX.
    SET PROPERTY OF CELLS 'Value' = <NAME>.
    ADD 1 TO INDEX.
  ENDDO.
ENDSELECT.
```