大型计算机

应用技术培训

JCL 语言与实用程序

目录

课程介绍

JCL 语言与实用程序

目的：

通过本课程的学习，学员能够对 JCL 语言有深入的了解，能够读懂和独立编写各种 JCL 语言，掌握常用实用程序的功能与使用，能根据需要编写进行处理的 JCL。

主要内容：

学员主要完成以下主要内容的学习：

- ✓ JCL 基本概念
- ✓ JCL 语句的语法
- ✓ JOB 语句的参数定义
- ✓ EXEC 语句参数定义
- ✓ DD 语句的参数定义
- ✓ 特殊的 DD 语句
- ✓ JCL 过程的使用
- ✓ 常用实用程序的使用

预修课程：

IBM 大型计算机基本操作

长度：

2 天

相关课程：

教程作者：温洪涛 venn@sina.com

第一章　JCL 语言介绍

- JCL 语言基本概念
- JCL 语言的一个简单例子
- JCL 语言的使用
- 查看 JCL 执行结果

1-1 JCL 语言基本概念

在 S/390 系统中 ,当用户需要使用计算机完成某项任务时，用户必须准备一个作业流(Job Stream)。作业流中包含一个或多个作业 (Job) ,作业是用 JCL(job control language)书写的.

与 COBOL 等一般的编程语言不同,作业控制语言 JCL(Job Control Language)是用户与操作系统的接口。用户通过 JCL 的相应语句来与操作系统通讯，获得作业所需的资源等，按自己的意图来控制作业的执行。

ＪＣＬ由几个语句组成，对于一个作业，JCL 为被执行的任务引导操作系统，并说明所需要的全部Ｉ／Ｏ设备,在一个作业中，每一次程序的执行称为一个作业步，一个作业可包含几个作业步。一个作业中的各步是顺序执行的，因此一个作业步的输出可以作为下一个作业步的输入。

作业都必须包含三个 JCL 基本语句（JCL Statement)。它们分别是：

（1）作业语句（JOB)：标识一个作业的开始，提供必要的运行参数。

（2）执行语句（EXEC)：标识一个作业步的开始，定义本作业步所要执行的程序或过程。

（3）数据定义语句（DD)：用于描述应用程序所需要的数据文件。

系统规定这三种语句行必须以“ // ”开头,/和/分别占据一行 JCL 语句的第一列和第二列

1-2 JCL 语言的一个简单例子

```
//COPYDATA  JOB (),'ERIC',
//          TIME=1440,
//          NOTIFY=&SYSUID,
//          REGION=0M,
//          CLASS=A,
//          MSGCLASS=X,
//          MSGLEVEL=(1,1)
//*  COPY DATASET
//CPYLOAD  EXEC PGM=IEBCOPY,PARM='SIZE=1M'
//SYSPRINT DD SYSOUT=*
//IN1      DD DISP=SHR,DSN=DEVP124.JCL
//OUT1     DD DISP=SHR,DSN=DEVP124.SRC
//SYSIN    DD *
  COPY I=IN1,O=OUT1
     S MEMBER=((JCLSMP1,,R))
/*
```

上面就是一个 JCL 的例子，它的功能是把“ JCLSMP1 ”从一个分区数据集拷贝到另一个分区数据集。

1．为 JCL 分配数据集

JCL 必须作为一个成员存储在分区数据集（PDS）中。使用 ISPF 3.2 的功能为 JCL 分配一个分区数据集，其格式必须是定长(RECFM=FB)并且记录长度 80 字节 (LRECL=80).

2．编辑 JCL

在上面建立的 PDS 里面创建一个 MEMBER，使用 ISPF 的编辑功能编辑 JCL

3．提交 JCL

编辑好后就可以提交 JCL 给 JES2 去运行，有三种常用的提交方式：

第一种是在 JCL 的编辑界面里，下面的命令行上输入“ SUB ”命令：

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
EDIT       DEVP124.JCL(JCLSMP1) - 01.10                  Columns 00001 00072
****** *************************** Top of Data ****************************
000001 //COPYDATA  JOB (),'ERIC',
000002 //         TIME=1440,
000003 //         NOTIFY=&SYSUID,
000004 //         REGION=0M,
000005 //         CLASS=A,
000006 //         MSGCLASS=X,
000007 //         MSGLEVEL=(1,1)
000008 //*  COPY NUMBER BETWEEN PDS
000009 //CPYLOAD  EXEC PGM=IEBCOPY,PARM='SIZE=1M'
000010 //SYSPRINT DD SYSOUT=*
000011 //IN1      DD DISP=SHR,DSN=DEVP124.JCL
000012 //OUT1     DD DISP=SHR,DSN=DEVP124.SRC
000013 //SYSIN    DD *
000014   COPY I=IN1,O=OUT1
Command ===> SUB                                          Scroll ===> CSR
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

第二种是在 ISPF 3.4 的成员列表中，在 JCL 前直接写 SUB 命令：

```
  Menu  Functions  Confirm  Utilities  Help
sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
BROWSE            DEVP124.JCL                          Row 00001 of 00012
           Name     Prompt       Size   Created          Changed          ID
_________ CDB2BAT                 15  2002/09/05  2004/08/28 14:33:12  DEVP124
_________ CFBM999                217  2004/06/22  2004/07/13 14:04:57  DEVP82
_________ CICSCOMP                 6  2004/06/15  2004/07/27 17:01:00  DEVP124
_________ DB2CBAT                  4  2004/06/15  2004/08/17 19:13:39  DEVP124
_________ DB2COBB                  8  2004/06/15  2004/08/27 10:40:40  DEVP124
_________ DB2CPROC                96  2004/08/17  2004/08/17 19:29:53  DEVP124
_________ DEFVSAM                 17  2004/07/27  2004/08/04 11:21:23  DEVP124
_________ DFHSIP                  51  2002/09/18  2004/05/20 10:21:35  SYSPRGG
_________ DSNTIJMV              1201  2004/08/17  2004/08/17 18:56:23  DEVP124
__sub____ JCLSMP1                 16  2004/07/27  2004/09/06 19:27:45  DEVP124
_________ MAPCOMP                  4  2004/07/27  2004/07/28 10:19:43  DEVP124
_________ RUNBAT                  15  2002/09/05  2004/08/28 14:33:12  DEVP124
         **End**
Command ===>                                                Scroll ===> CSR
  F1=Help    F2=Split   F3=Exit    F5=Rfind   F7=Up      F8=Down    F9=Swap
 F10=Left   F11=Right  F12=Cancel
```

第三种方法是使用直接使用 TSO 命令 SUBMIT 提交，可以在 ISPF 中任何地点发出命令：

```
 ----------------------  ISPF/PDF PRIMARY OPTION MENU  ----------------
OPTION  ===> TSO SUBMIT 'DEVP124.JCL(JCLSMP1)'
                                                            USERID   -
   0  ISPF PARMS  - Specify terminal and user parameters    TIME     -
   1  BROWSE      - Display source data or output listings  TERMINAL -
   2  EDIT        - Create or change source data            PF KEYS  -
   3  UTILITIES   - Perform utility functions
   4  FOREGROUND  - Invoke language processors in foreground
   5  BATCH       - Submit job for language processing
   6  COMMAND     - Enter TSO command or CLIST
   7  DIALOG TEST - Perform dialog testing
   8  DB2         - Perform DATABASE 2 interactive functions
   C  CHANGES     - Display summary of changes for this release
   T  TUTORIAL    - Display information about ISPF/PDF
   X  EXIT        - Terminate ISPF using log and list defaults

Enter END command to terminate ISPF.

 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

1-4 查看 JCL 执行结果

JCL 提交时屏幕会返回提交程序的进程号，进入 SDSF 可以根据进程号或 JOB 名去查看 JCL 执行结果，选择 SD.ST，找到提交的进程，在需要查看的 JCL 前输入“ ？ ”查看详细信息，如下例：

```
  Display  Filter  View  Print  Options  Help
------------------------------------------------------------------------------
SDSF STATUS DISPLAY ALL CLASSES                        LINE 1-3 (3)
NP   JOBNAME  JobID    Owner    Prty Queue      C  Pos  SAff  ASys Status
     DEVP124  TSU01249 DEVP124    15 EXECUTION          ZDVP  ZDVP
     DEVP124  TSU00904 DEVP124     1 PRINT         1153
?    COPYDATA JOB01280 DEVP124     1 PRINT      A  1245



COMMAND INPUT ===>                                          SCROLL ===> CSR
  F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=IFIND     F6=BOOK
  F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

回车进入如下界面：

```
   Display  Filter  View  Print  Options  Help
------------------------------------------------------------------------------
SDSF JOB DATA SET DISPLAY - JOB COPYDATA (JOB01280)    LINE 1-4 (4)
NP   DDNAME   StepName ProcStep DSID Owner    C Dest               Rec-Cnt Page
     JESMSGLG JES2                 2 DEVP124  X LOCAL                   20
     JESJCL   JES2                 3 DEVP124  X LOCAL                   15
     JESYSMSG JES2                 4 DEVP124  X LOCAL                   15
     SYSPRINT CPYLOAD            102 DEVP124  X LOCAL                   13

COMMAND INPUT ===>                                          SCROLL ===> CSR
 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=IFIND     F6=BOOK
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

在结果中 JESMSGLG、JESJCL 和 JESYSMSG 三个 DDNAME 是所有 JCL 结果里都会有的，JESMSGLG 里存放各步的返回值和统计信息，JESJCL 里存放用户提交的 JCL 语句，JESYSMSG 里存放的是 JES 运行 JCL 时发生的详细信息。其他的 DDNAME 则根据程序不同而或多或少，都是程序输出的数据。

JESMSGLG 的一个例子：

```
********************************* TOP OF DATA ************************************* TOP OF DATA *********************************
1                      J E S 2  J O B  L O G  --  S Y S T E M  Z D V P  --  N O D E  N 1       <- JES 节点名 N1
0
 09.12.22 JOB01465 ---- TUESDAY,   07 SEP 2004 ----                                          <- 执行日期
 09.12.22 JOB01465  IRR010I  USERID DEVP124  IS ASSIGNED TO THIS JOB.                        <- 提交 JCL 的用户
 09.12.22 JOB01465  ICH70001I DEVP124  LAST ACCESS AT 09:11:30 ON TUESDAY, SEPTEMBER 7, 2004
 09.12.22 JOB01465    HASP373 COPYDATA STARTED - INIT 1    - CLASS A - SYS ZDVP
 09.12.22 JOB01465  IEF403I COPYDATA - STARTED - TIME=09.12.22                                <- 执行时间
 09.12.22 JOB01465  -                                                --TIMINGS (MINS.)--            ----PAGING COUNTS---
 09.12.22 JOB01465  -JOBNAME  STEPNAME PROCSTEP    RC   EXCP   CONN    TCB    SRB  CLOCK   SERV  PG  PAGE  SWAP   VIO SWAPS  <- 作业步信息
 09.12.22 JOB01465  -COPYDATA CPYLOAD              00     45     38    .00    .00     .0    182   0     0     0     0     0
 09.12.22 JOB01465  IEF404I COPYDATA - ENDED - TIME=09.12.22
 09.12.22 JOB01465  -COPYDATA ENDED.  NAME-ERIC                TOTAL TCB CPU TIME=   .00  TOTAL ELAPSED TIME=    .0     <- 时间统计
 09.12.22 JOB01465    HASP395 COPYDATA ENDED
0------ JES2 JOB STATISTICS ------                   <- JOB 统计
-  07 SEP 2004 JOB EXECUTION DATE
-           16 CARDS READ
-           63 SYSOUT PRINT RECORDS
-            0 SYSOUT PUNCH RECORDS
-            4 SYSOUT SPOOL KBYTES
-         0.00 MINUTES EXECUTION TIME
******************************** BOTTOM OF DATA ********************************
```

1-4　查看 JCL 执行结果（续页 2）

JESJCL 例子：

```
****************************** TOP OF DATA ********************************************************************** TOP OF DATA ********************************
       1 //COPYDATA  JOB (),'ERIC',                                                                                JOB01465
         //         TIME=1440,
         //         NOTIFY=&SYSUID,
         //         REGION=0M,
         //         CLASS=A,
         //         MSGCLASS=X,
         //         MSGLEVEL=(1,1)
         //*  COPY NUMBER BETWEEN PDS
         IEFC653I SUBSTITUTION JCL - (),'ERIC',TIME=1440,NOTIFY=DEVP124,REGION=1440,NOTIFY=DEVP124,REGION=0M,CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
       2 //CPYLOAD  EXEC PGM=IEBCOPY,PARM='SIZE=1M'
       3 //SYSPRINT DD SYSOUT=*
       4 //IN1      DD DISP=SHR,DSN=DEVP124.JCL
       5 //OUT1     DD DISP=SHR,DSN=DEVP124.SRC
       6 //SYSIN    DD *
         //*
****************************** BOTTOM OF DATA ******************************************************************* BOTTOM OF DATA ********************************
```

JESYSMSG 例子 ：

```
******************************** TOP OF DATA ********************************************* TOP OF DATA ********************************
ICH70001I DEVP124  LAST ACCESS AT 09:11:30 ON TUESDAY, SEPTEMBER 7, 2004
IEF236I ALLOC. FOR COPYDATA CPYLOAD                                          <- 数据集的分配
IEF237I JES2 ALLOCATED TO SYSPRINT
IGD103I SMS ALLOCATED TO DDNAME IN1
IGD103I SMS ALLOCATED TO DDNAME OUT1
IEF237I JES2 ALLOCATED TO SYSIN
IEF142I COPYDATA CPYLOAD - STEP WAS EXECUTED - COND CODE 0000                <- 各步的返回码
IEF285I   DEVP124.COPYDATA.JOB01465.D0000102.?         SYSOUT
IGD104I DEVP124.JCL                                 RETAINED,  DDNAME=IN1    <- 数据集的释放
IGD104I DEVP124.SRC                                 RETAINED,  DDNAME=OUT1
IEF285I   DEVP124.COPYDATA.JOB01465.D0000101.?         SYSIN
IEF373I STEP/CPYLOAD /START 2004251.0912                                     <- 各步及整个作业的时间统计
IEF374I STEP/CPYLOAD /STOP  2004251.0912 CPU    0MIN 00.01SEC SRB    0MIN 00.00SEC VIRT  1164K SYS   264K EXT       4K SYS   10544K
IEF375I  JOB/COPYDATA/START 2004251.0912
IEF376I  JOB/COPYDATA/STOP  2004251.0912 CPU    0MIN 00.01SEC SRB    0MIN 00.00SEC
******************************** BOTTOM OF DATA ***************************************** BOTTOM OF DATA ********************************
```

第二章　JCL 语句

- JCL 语法规则
- JOB 语句
- EXEC 语句
- DD 语句

2-1 JCL 语法规则

- JCL 语句类型
- JCL 字符集
- 一般语句格式规范
- 参数规则

2-1-1 JCL 语句类型

JCL 例子：

```
//COPYDATA  JOB  NOTIFY=&SYSUID
//*  COPY NUMBER BETWEEN PDS
//MYPROC PROC INDD=,OUTDD=
//CPY      EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=&INDD.,DISP=SHR
//SYSUT2   DD DSN=&OUTDD.,DISP=(NEW,CATLG,DELETE),
//            SPACE=(TRK,(2,1)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=240,DSORG=PS)
//SYSIN    DD DUMMY
//MYEND    PEND
//CPYLOAD1 EXEC PGM=IEBCOPY,PARM='SIZE=1M'
//SYSPRINT DD SYSOUT=*
//IN1      DD DISP=SHR,DSN=DEVP124.JCL
//OUT1     DD DISP=SHR,DSN=DEVP124.SRC
//SYSIN    DD *
  COPY I=IN1,O=OUT1
     S MEMBER=((JCLSMP1,,R))
/*
//CPYLOAD2 EXEC MYPROC,INDD='DEVP124.DATA',
//  OUTDD='DEVP124.DATA1'
//
//CPYLOAD3  EXEC PGM=IEBCOPY,PARM='SIZE=1M'
//SYSPRINT DD SYSOUT=*
NOT USE
```

JCL 语句分为以下几种：

- JOB 语句
- EXEC 语句
- DD 语句
- /*　　流内数据结束
- //*　　注释
- //　　标记作业结束
- PROC　　过程起始标记
- PEND　　过程结束标记
- Command　　输入流中的操作命令

JCL 字符集

JCL 语言可以使用的字符集包括：

(1) 字母（共 26 个）A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
(2) 数字（共 10 个）0 1 2 3 4 5 6 7 8 9
(3) 特殊字符（共 10 个）, . / ‘ ( ) * & + - =
(4) 通配符（共 6 个）@ $ # （也可分别用 X’7C’ X’5B’ 和 X’7B’表示）
(5) EBCDIC 可打印字符集

JCL 中，除/*语句外的所有语句均以第一、二列的//符号作为开始标志，系统规定这些语句的长度为 80 列。这 80 列在逻辑上被划分为五个区域，分别是标识符区、名字区、操作符区、参数区和说明区，即:

·标识符区

一般情况下，标识符区的符号为“ // ”，该符号表明该条语句为 JCL 语句。标识符区位于每条语句的第一、二列。在特殊情况下，标识符区的符号将有所变化。如 3.2.1 中所讨论过的“ /* ”语句和“ //*语句，则分别在标识符区中使用的符号“ /* ”和“ //* ”表示。

·名字区

名字区指明一个语句,便于系统控制块或其他语句引用它。名字可以由 1~8 个字母数字或通配符组成，但第一个字符必须是字母或通配符，且必须从第三列开始。名字区后必须跟一个或多个空格，可以选择名字表达出这个 JCL 语句的作用。下面给出几个正确与错误的名字区的例子：

正确的	错误的
//Z	//9Z
//BACKUP#1	//TAPEBACKUP
//#99	//TEST*9
//$EXAM	//EXAM(0)

·操作符区

操作符区位于名字区之后，规定了语句的类型：JOB、EXEC、DD、PROC、PEND，或操作员命令。名字区后必须跟一个或多个空格。例如：

```
//EXAMPLE JOB
//STEP1 EXEC
//INDD1 DD
```

·参数区

参数区在操作符区之后，其中包括被逗号分隔的参数，参数由事先规定好的关键字组成，对于这些参数其数值必须是可被代换的变值。参数区没有固定的长度及列的要求。例如：

```
//EXAMPLE JOB 2000，CLASS=A
//STEP1 EXEC PGM=IEYFORT
//PRINT DD SYSOUT=A
```

·说明区

说明区位于参数区后，用于对相应语句进行注释说明，它可以是任何需要的说明信息，注释区后必须跟一空格。需要注意的是，仅当参数出现时才能书写说明信息，不然容易与参数混淆。下面是一个说明区的例子：

```
//EXAMPLE JOB ，CLASS=A    IT IS A COMMENT
```

JCL 只允许在参数区和说明区有续行，当需要续行时，在当前行的第 71 列前必须将某个参数或某个子参数以及参数后的逗号写完整，且下一行第 1、2 列为“ // ”，第 3 列为空格，续行的内容只能从 4~16 列开始，如从 16 列后开始，将被认为是注释语句。下面是一个续行的例子：

```
//DATA DD DSN=SYS1.FORTLIB,
//   DISP=OLD
```

2-1-4 参数规则

在 JCL 中，参数区内的参数分为两种类型：

1）位置参数：靠与其他参数的相对位置而定的参数。

2）键字参数：以“ 键字名=值 ”的形式给出的参数。

如果一个语句中既有位置参数又有位置参数，又有键字参数时，位置参数要按其位置，排在键字参数之前，如：//JOBA JOB 1000,CLASS=A

JCL 中参数的书写规则如下：

1）位置参数和键字参数之间必须用逗号分开，不允许有空格，这点要非常注意，否则会造成比较难于查找的错误。

正确的情况：

```
//COPYDATA  JOB 1000,'ERIC',NOTIFY=&SYSUID
//JOBB  JOB (1000,9),NOTIFY=&SYSUID
```

错误的情况：

```
//COPYDATA  JOB 1000,'ERIC',  NOTIFY=&SYSUID
//JOBB  JOB (1000,9)NOTIFY=&SYSUID
```

2）参数必须按规定的次序书写，位置参数要依照位置次序书写，键字参数放在所有位置参数之后，键字参数可以随意顺序书写。

正确的例子：

```
//COPYDATA  JOB 1000,NOTIFY=&SYSUID
```

错误的例子：

```
//COPYDATA  JOB NOTIFY=&SYSUID,1000
```

3）有几个位置参数时，如果前面的参数不写使用省缺，必须用逗号留出位置，如：

```
//JOBA JOB ,SYSTEM,CLASS=A
```

4）如果没有指定任何位置参数时则可不为他们留位置，如：

```
//JOBB JOB NOTIFY=&SYSUID
```

5）键字参数只靠键字的名字区分，前后次序无关，以下两句是等效的：

```
//JOBA JOB NOTIFY=&SYSUID,REGION=0M
//JOBA JOB REGION=0M,NOTIFY=&SYSUID
```

6）允许含有特殊字符的参数或子参数，且其中的特殊字符并不起某种特定的语法功能，只当符号用事，要用单引号扩起来。如 ACCOUNT= ‘ 123+345 ’

7）JCL 参数可以有子参数，即以括号扩起来，逗号间隔的多个子参数，子参数最多有 2 级，也就是括号最多有 2 层。

2-2 JOB 语句

- JOB 语句的作用
- JOB 语句的位置参数
- JOB 语句的键字参数

JOB 语句的作用

- 提供了记帐信息；
- 定义了执行特征
- 指定系统信息和 JCL 语句输出级别
- 保持（HOLD）一个作业
- 指定作业的优先级别
- 限制使用系统资源（如 CPU 时间和主存空间要求）

JOB 语句的通常模式为：

作业名　　JOB　　帐号（记帐信息）　　程序员名　　键字参数
位置参数

作业名由 1-8 个字符组成，标志一个作业，在作业提交时系统还会为作业产生一个作业号，这个作业号在整个系统里是唯一的。由于系统不能同时运行两个作业名相同的作业，所以希望同时运行的作业不要起相同的名字。

2-2-2 JOB 语句的位置参数

JOB 语句的位置参数有两个，依次是：

1）记帐信息：记帐信息是关键字 JOB 后面第一个参数，用于提供用户使用系统的合法性和机时记帐，其格式为：

```
([account-number][,accounting-information]...)
account-number：         帐号
accounting-information：是可选的附加记帐信息。
```

记帐信息及起子参数加起来不能超过 143 个字符，以下都是合法的记帐信息参数：

```
//JOB43 JOB D548-8686
//JOB44 JOB (D548-8686,'12/8/85',PGMBIN)
//JOB45 JOB (CFH1,2G14,15,,,,2)
```

2）程序员名：用来标志编写作业的 JCL，程序员名信息总长度不能超过 20 个字符，若名字中间有空格或其他特殊字符,则用单引号。以下是合法的程序员名参数的 例子：

```
//APP  JOB ,ERIC.VENN
//JOBB JOB ,'WEN HONG TAO'
//JOBA JOB (846349,GROUP12),MATTHEW
```

JOB 语句中的关键字参数有如下几个：

1． ADDRSPC

指明作业所需之存贮类型，它有两个子参数：VIRT 及 REAL。VIRT 表示作业请求虚拟页式存贮，而 REAL 表示作业请求实存空间。缺省值为 VIRT。其格式为：

ADDRSPC={VIRT}
{REAL}

例：

```
//PEH    JOB    ,BAKER,ADDRSPC=VIRT
//DEB    JOB    ,ERIC,ADDRSPC=REAL,REGION=100K
```

2． BYTES

指明打印作业的系统输出数据集的最大千字节数，同时该参数还指出当超过所给出的最大字节数时，系统对作业的处理方式。这些方式包括：取消作业（转储（dump）或不转储）或继续作业并向操作员发出超过最大字节数的警告信息。其格式为：

BYTE={nnnnn}
{([nnnnnn][,CANCEL])}
{([nnnnnn][,DUMP])}
{([nnnnnn][,WARNING])}

nnnnnn：指明打印输出的最大千字节数，例：nnnnnn 取值 500，则表示 500,000 字节。nnnnnn 取值范围为：0 ~ 999999。

CANCEL：当作业输出字节数超过 nnnnnn 时，系统将不转储而直接取消该作业。

DUMP：当作业输出字节数超过 nnnnnn 时，系统在取消该作业前将发出转储请求。

WARNING：当作业输出字节数超过 nnnnnn 时，作业继续执行，系统将按照安装时规定的时间间隔不断向操作员发送警告信息。

当 BYTE 参数或其子参数省略不写时，系统将采用安装时定义的默认值。

例：

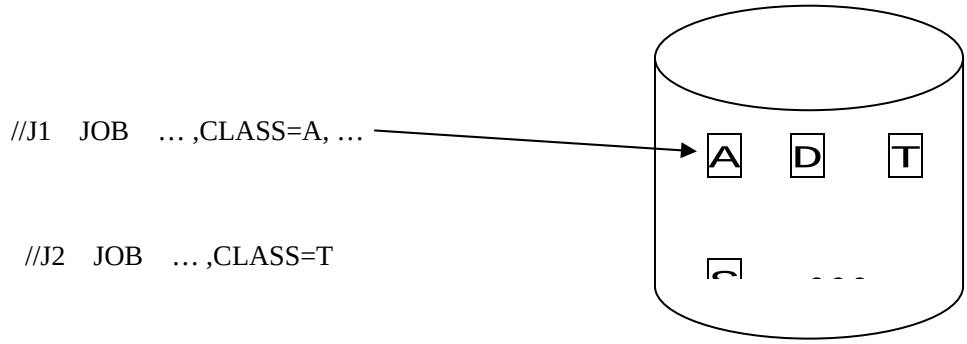
```
//JOB1    JOB    (123456),'R F B',BYTES=(5000,CANCEL)
// JOB1    JOB    (123456),'R F B',BYTES=40
```

除了 BYTES 参数外，JOB 语句中还有另三个参数可以限制作业输出的最大值，其格式及子参数的意义也与 BYTES 类似，它们是：CARDS、LINES 及 PAGES。上述三个参数与 BYTES 不同之处在于子参数 nnnnnn 的单位不同，分别是：卡数、行数及页数，读者可以类推使用。

3． CLASS

CLASS 参数规定了作业的类别，JCL 中可选用的作业类别有 36 个，用字母 A~Z 及数字 0~9 表示。相同类别的作业处于同一输入队列等待执行（如图 3.2.1），并具有相同的处理属性。作业类别的属性定义在 JES 中。当 CLASS 参数缺省时，JES 将会根据安装时的缺省值赋予该作业一个缺省的 CLASS 值。

格式：CLASS=jobclass



4．MSGCLASS

用于为作业日志（job log）设置输出类别。作业日志是为程序员提供的与作业相关信息的记录。当该参数省略时，系统将会采用默认值。

格式：

MSGCLASS=class

class：定义作业日志的类别。与输入队列相似，class 是一个 A~Z 的字母或一个 0~9 的数字。

例：

//EXMP1　JOB　,GEORGE,MSGCLASS=F

5．MSGLEVEL

用于控制 JCL 作业输出清单的内容，用户可以要求系统打印出如下内容：

· JCL 语句；

· 输入流中的所有控制语句，即：所有的 JCL 语句及 JES2 或 JES3 语句；

· 任何作业步调用的流内过程和编目过程语句；

· 作业控制语句的信息；

· JES 及操作员对作业的处理信息：设备和卷的分配、作业步及作业的执行和终止、数据集的处理等。

格式：

MSGLEVEL=([statements][,messages])

statements：指明在 JCL 作业输出清单中应打印出的作业控制语句的类型，取值范围为：0 ~ 2。

取值 0：仅打印出作业的 JOB 语句；

取值 1：打印出作业中包括过程语句在内的所有 JCL 语句；

取值 2：输入流中的所有控制语句。

messages：指明在 JCL 作业输出清单中应打印出信息的类型，取值范围为：0 ~ 1。

取值 0：只有在作业异常终止时，打印出有关 JCL、JES、操作员及 SMS 的处理信息；

取值 1：无论作业是否异常终止，都打印出有关 JCL、JES、操作员及 SMS 的处理信息。

例：

```
//EXMP3    JOB    ,MSGLEVEL=(2,1)
//EXMP4    JOB    ,MENTLE,MSGLEVEL=0
//EXMP5    JOB    ,MIKE,MSGLEVEL=(,0)
```

6．NOTIFY

用于请求系统在后台作业处理完毕时给指定用户发送信息。如果作业完成时，该用户未在系统登录，则系统所发送的信息将会保留到此用户下次登录。

格式：

NOTIFY={userid}

userid：必须以字母或通配符开头的 1~7 个字母、数字或通配符组成，其值必须是一个存在的 TSO 用户标识。

例：

```
//SIGN    JOB    ，TLOMP，NOTIFY=TSOUSER
```

7．PRTY

用于为相应的输入队列中的作业分配优先级。系统根据作业优先级的高低来选择来选择作业执行，对于同一级的作业的选择将采取“ 先进先出 ”的原则。

格式：

PRTY=priority

priority：用数字量来表示优先级，数字越大表示优先级越高。根据作业进入子系统的类型，其取值范围是 JES2：0~15；JES3：0~14。

例：

```
//JOBA    JOB    1,’JIM WEBSTER’,PRTY=12
```

8．REGION

用于指定作业所需的实存或虚存空间的大小，系统将在该作业中的每一作业步使用该值。所需空间大小必须包含以下内容：

·运行所有程序所需的空间

·在运行期间，程序中宏指令 GETMAIN 所需的所有附加空间

·任务初始化和终止时需要的自由空间

如果 JOB 语句中的 REGION 参数省略不写的话，系统将采用每条 EXEC 语句中所定义的 REGION 参数，当 EXEC 语句中的 REGION 参数省略不写时，系统将采用安装缺省值。

格式：

REGION={valueK}

={valueM}

valueK：以千字节（Kb）为单位指出所需空间大小，value 可取 1~7 位的十进制数，其取值范围为 1~2096128。系统以每 4k 为一存储单位分配空间，所以 value 值应取 4 的倍数，如 REGION=68K。当 value 值不是 4 的倍数时，系统会将其增至一最为接近的 4 的倍数的值。

valueM：以兆字节（Mb）为单位指出所需空间大小，value 可取 1~4 位的十进制数，其取值范围为 1~2047

注：REGION 值必须是有效的存储空间，如果取值为 0 或任何大于系统极限的值时都有可能会引起存储问题。当系统未定义极限值时，value 值不能超过 16384K 或 16M。

例：

```
//ACCT1    JOB    A23,SMITH,REGION=100K,ADDRSPC=REAL
//ACCT2    JOB    175,FRED,REGION=250K
```

9．TIME

用于指定作业占用处理器的最长时间并可通过一些信息得知该作业占用处理器的时间。当作业占用处理器时间超过指定值时，系统将终止该作业。通常情况下，此参数不用设置。当作业所需处理器时间长于系统缺省值时，或出于某种测试目的才设置此参数。

格式：

TIME={([minutes][,seconds])}

={1440　　}

={NOLIMIT　　}

={MAXIMUM　　}

minutes：指定作业可占用处理器最长时间的分钟数，取值范围为 0~357912（248.55 天）。不可以将 TIME 参数写作 TIME=0，这样将导致不可预知的后果。

Seconds：作为 minutes 的补充，定义指定作业可占用处理其最长时间的秒钟数，取值范围为 0~59。

NOLIMIT：表明作业的运行无时间限制,等同于 TIME=1440。

1440：表明表明作业的运行无时间限制，即 24 小时。

MAXIMUM：表示作业的运行时间为 357912 分钟。

当 JOB 语句中的 TIME 参数没有指明时，每作业步的运行时间限制由以下值决定：

·在 EXEC 语句中 TIME 参数的值。

·当 EXE 语句中没有设置 TIME 参数时，采用默认的时间限制值（也就是 JES 默认作业步时间限制值）。

例 1：

```
//STD1    JOB    ACCT271,TIME=(12,10)
```

例 2：

```
//STD2    JOB    ,GOR,TIME=(,30)
```

例 3：

```
//FIRST    JOB    ,SMITH,TIME=2
//STEP1    EXEC    PGM=READER,TIME=1
                 . . .
//STEP2 EXEC    PGM=WRITER,TIME=1
                 . . .
```

在上例中，JOB 语句中规定了 2 分钟的作业运行时间限制，每个作业步允许 1 分钟，如果任何一个作业步的执行时间超过 1 分钟，作业将会异常终止。

例 4：

```
//SECOND    JOB    ,JONES,TIME=3
//STEP1    EXEC    PGM=ADDER,TIME=2
                  . . .
//STEP2    EXEC    PGM=PRINT,TIME=2
                  .
```

上例中，JOB 语句中规定了 3 分钟的作业运行时间限制，每个作业步允许 2 分钟，如果任何一个作业步的执行时间超过 2 分钟，作业将会异常终止。但两个作业步的总共运行时间不得超过作业运行时间限制——3 分钟，也即：如果作业步 1 的运行时间为 1.56 分钟,则作业步 2 的运行时间不得超过 1.44 分，否则作业也会异常终止。

10．TYPRUN

用于请求特殊的作业处理。TYPRUN 可以告知系统如下要求：

·在 JES2 中，将输入作业流直接拷贝到系统输出数据集并对其进行输出处理。

·在 JES2 或 JES3 中，挂起一个作业，直至某特定事件发生。当该特定事件发生时，操作员根据用户的要求释放该作业，并允许系统选择该作业执行。使用 JES2 中的/*MASSAGE 语句或 JES3 中的//*OPERATOR 语句通知操作员释放该作业。

·在 JES2 或 JES3 中，对作业的 JCL 进行语法检查。

值得注意的是：不能对已经开始的任务（task）设置该参数，否则该作业将会出错。

格式：

```
TYPRUN={COPY      }
       {HOLD      }
       {JCLHOLD }
       {SCAN      }
```

子参数说明：

COPY（仅支持 JES2）： 请求 JES2 将输入作业流直接拷贝到系统输出数据集并对其进行输出处理。系统并不执行该作业。系统输出数据集的类别与该作业 JOB 语句中 MSGCLASS 参数定义的信息类别（massage class）相同 。

HOLD：请求系统在执行作业之前将其挂起，等待某特定事件发生后，请求操作员将其释放。如果在作业的输入过程中出现错误，JES 将不会挂起该作业。

JCLHOLD（仅支持 JES2）：请求 JES2 在 JCL 执行前将其挂起，直到操作员将其释放。

SCAN：请求系统只对作业的 JCL 进行语法检查，不执行也不为其分配设备。

例：

```
//UPDTAE    JOB    ,HUBBARD
//STEP1       EXEC    PGM=LIBUTIL
                  .
                  .
                  .
//LIST         JOB    ,HUBBARD,TYPRUN=HOLD
//STEPA      EXEC    PGM=LIBLIST
                  .
                  .
                  .
```

上例中，作业 UPDATE 与 LIST 在同一个作业流中被提交执行。作业 UPDATE 的功能是在库中增加一个成员再删除一个成员；作业 LIST 则列出该库的成员目录。显然，LIST 应在 UPDATE 之后在执行。作业 LIST 的 JOB 语句中设置的 TYPRUN=HOLD 使得保证了这一执行顺序。

如果输入流中或操作员已执行了 MONITOR JOBNAMES 的命令，当 UPDATE 执行完后，系统会通知控制台操作员。操作员释放作业后，系统可以选择该作业执行。

11．其他参数

在 JCL 的 JOB 语句中的关键字参数还有：

COND、GROUP、PASSWORD、PERFORM、RD、RESTART、SECLABEL、USER，由于本书篇幅有限，在这里就不再一一介绍了，详细的使用方法读者可以参考《MVS JCL Reference》一书。

2-3 EXEC 语句

- EXEC 语句的作用
- EXEC 语句的位置参数
- EXEC 语句的键字参数

2-3-1　EXEC 语句的作用

EXEC 语句标识要执行的程序和过程；提供作业步记帐信息；给出执行该作业步的条件；指定该作业步使用处理器的时间。包括所有在 EXEC 语句中调用的过程所包含的作业步在内，一个 JOB 最多可以有 255 个 EXEC 语句。

EXEC 语句格式如下：

//[作业步名]　EXEC 位置参数 [，键字参数] 。。。[符号参数=值] 。。。 [注释]

作业步名由 1-8 个字符组成，可以省略不写，如果需要写出，则在一个作业里所有作业步名不能重复。

EXEC 语句有两个位置参数：PGM 和 PROC。每条 EXEC 语句必须有且仅有一个 PGM 或 PROC 参数

1）PGM

PGM 参数用于指名所要执行的程序名，程序以二进制可执行代码的形式存放在一个分区数据集（PDS）的成员里，系统到作业中的 JOBLIB DD 语句指定的 PDS 或本作业步中的 STEPLIB DD 定义的 PDS 或默认的 PDS 中去查找 PGM 所指的成员，装载执行。程序名的调用方式分为直接调用和间接调用。

格式：

PGM={program-name}

{*.stepname.ddname}

{*.stepname.procstepname.ddname}

program-name：program-name（程序名）指明要执行程序的成员名或别名。程序名由由 1~8 个字母或通配符开头的字符数字构成。

***.stepname.ddname**：表示要执行的程序名由本作业步前名为 “ stepname ” 的作业步内名为 “ ddname ” 的 DD 语句的 DSN 参数决定。

***.stepname.procstepname.ddname**：表示要执行的程序名由本作业步前名为 “ stepname ” 的作业步里所调用过程内名为“ procstepname ”的过程步中相应名为“ ddname ”DD 语句的 DSN 参数决定。

在上述三种程序调用方法中，第一种为直接调用，而后两种为间接调用。间接调用采用向后参考的方法，这里的 “ 后 ” 指在本作业步读入之前，已先读入系统的本作业其它 JCL 语句。当需调用的程序在系统库（如 SYS1.LINKLIB）或私有库（由作业中的 JOBLIB DD 语句或本作业步中的 STEPLIB DD 定义）中时使用第一种调用方法；而当需调用的程序在本作业步前的某一作业步创建的临时库中时采用后两种调用方法。例：

```
//JOBC     JOB    ,JOHN,MSGCLASS=H
//STEP2    EXEC    PGM=UPDT
//DDA      DD    DSNAME=SYS1.LINKLIB(P40),DISP=OLD
//STEP3    EXEC    PGM=*.STEP2.DDA
```

2）PROC：指名作业步所要运行的过程名

格式：

{PROC=procedure-name}

{procedure-name }

procedure-name：要调用的过程名称，过程名由 1-8 个字符组成，所调用的过程名可以是：

①编目过程的成员名或别名。编目过程是把过程语句专门写到一个数据集中，这个数据集在系统中编目。

②本 JCL 内定义的流内 PROC。流内 PROC 是在 JCL 中由 PROC 和 PEND 之间部分定义的一个过程。

一些例子：

```
//STEP1 EXEC PROC=MYPROC
//SETP2 EXEC MYPORC2
```

1. ACCT

指明作业步所需的一个或多个记账信息子参数。记账信息参子参数最多不可超过 142 个字符（包括分隔子参数的逗号，但不包括括起子参数列表的括号）。

格式：

ACCT[.过程步名]=(记账信息)

例：

```
//STP3  EXEC  PROC=LOOKUP,ACCT=(‘/83468’)
```

2. ADDRSPC

指明作业步所需之存贮类型，它有两个子参数：VIRT 及 REAL。VIRT 表示作业步请求虚拟页式存贮，而 REAL 表示作业步请求实存空间，不能进行页式处理。缺省值为 VIRT。EXEC 语句中的 ADDRSPC 参数仅在本作业步中起作用，JOB 语句中的 ADDRSPC 参数会覆盖该作业中的所有 EXEC 语句中的 ADDRSPC 参数。

格式：

ADDRSPC[.过程步名]={VIRT}
{REAL}

例：

```
//CAC1  EXEC  PGM=A,ADDRSPC=VIRT
//CAC2  EXEC  PROC=B,ADDRSPC=REAL,REGION=100K
```

3. REGION

用于指定作业步所需的实存或虚存空间的大小，系统仅在本作业步中使用该值。

格式：

REGION[.过程步名]={valueK}
={valueM}

EXEC 语句中 REGION 的子参数定义与 JOB 语句中相同。

例：

```
//MKBOYLE  EXEC  PROC=A,REGION=100K,ADDRSPC=REAL
//STEP6   EXEC  PGM=CONT,REGION=250K
```

4. TIME

用于指定作业步占用处理器的最长时间，并可通过作业输出清单得知该作业步占用处理器的时间。当作业步占用处理器时间超过指定值时，系统将终止该作业。

格式：

TIME[.过程步名]={([minutes][,seconds])}
={1440 }
={NOLIMIT }
={MAXIMUM }

EXEC 语句与 JOB 语句中的 TIME 参数的子参数的设置方法基本相同。值得注意的是：在 JOB 语句中不可设置 TIME=0，而在 EXEC 语句中则可以设置 TIME=0，当 TIME=0 时表示本作业步的执行时间由前面作业步的剩余执行时间决定。

例 1：

```
//STP1  EXEC  PGM=ACCT,TIME=(12,10)
```

例 2：

```
//STP2  EXEC  PGM=PAY,TIME=(,30)
```

例 3：

```
//FIRST  JOB  ,SMITH  MSGLEVEL=(1,1)
//STEP1  EXEC  PGM=READER,TIME=1
           .
           .
           .
//STEP2 EXEC  PGM=WRITER
           .
```

在上例中，STEP1 规定了 1 分钟的执行时间，STEP2 的运行时间将由 STEP1 决定，也即 STEP2 的执行时间为：（1 分钟 － STEP2 实际运行时间）。

5． COND

用于对先前作业步执行的返回码（return code）进行测试，以决定是否执行本作业步。用户可以对特定作业步的返回码进行测试也可以对每一执行完毕的的返回码都进行测试。**如果测试条件不满足，系统执行本作业步；如果测试条件满足系统则不执行该作业步**。作业中的第一个 EXEC 语句中的 COND 参数将被系统忽略。**注意，当测试条件满足时，系统并非不正常终止该作业步，而只是跳过该作业步，该作业仍将正常执行。**

格式：

（1）COND[.过程步名]=(code,operator)

（2）COND[.过程步名]=((code,operator[,作业步名][,过程步名])
[,(code,operator[,作业步名][,过程步名])]…[,EVEN])
[,ONLY]

（3）COND=EVEN
COND=ONLY

利用 COND 参数最多可以有 8 个返回码测试，如果有 EVEN 或 ONLY 时，最多有 7 个测试。格式（1）只有在先前作业步没有非正常终止时，才能进行该测试。格式（2）、（3）测试决定于 EVEN 和 ONLY 的设置。

code：系统使用 code（测试码）与先前作业步或某特定作业步的返回码进行比较。Code 的取值范围为：0~4095。

operater：表示 code 与返回码的比较类型，这些比较的操作符是：GT（大于）、GE（大于等于）、EQ（等于）、NE（不等于）、LT（小于）、LE（小于等于）。

作业步名：指定先前某一作业步，并用该作业步的返回码与本作业步的测试码进行比较。当省略作业步名时，表示本作业步的测试码将与先前所有作业定额的返回码进行比较测试。

作业步名.过程步名：指定先前某一作业步调用过程的过程步。系统将用该过程步的返回码与给定的测试码进行比较。其中该作业步由“ 作业步名 ”指定，而过程步由“ 过程步名 ”指定。

EVEN：表示无论**即使**先前作业步异常终止，本作业步都要执行。当 EVEN 子参数设定时：

· 不测试先前任何的异常终止作业步的返回码。

· 测试那些正常完成的作业步的返回码，如果测试条件全部不满足的话，本作业步将执行。

ONLY：表示**只有**先前作业步异常终止，本作业步才执行。当 ONLY 子参数设定时：

· 不测试先前任何的异常终止作业步的返回码。

· 测试那些正常完成的作业步的返回码，如果测试条件全部不满足的话，本作业步将执行。

EVEN 与 ONLY 的具体情况见下表：

EVEN/ ONLY	先前作业步是否异常终止？	测试条件是否满足？	本作业步是否执行？
EVEN	否	否	是
EVEN	否	是	否
EVEN	是	否	是
EVEN	是	是	否
ONLY	否	否	否
ONLY	否	是	否
ONLY	是	否	是
ONLY	是	是	否

例 1.

```
//STEP6    EXEC    PGM=DISKUTIL,COND=(4,GT,STEP3)
```

在本例中如果 STEP3 的返回码小于 4，系统将不执行 STEP6。由于没有设置 EVEN 或 ONLY，如果先前的作业步异常终止，系统将不会执行本作业步。

例 2.

```
//TEST2    EXEC    PGM=DUMPINT,COND=((16,GE),(90,LE,STEP1),ONLY)
```

由于设置了 ONLY 子参数，系统只在以下两种情况满足时执行本作业步：

（1）先前作业步异常终止；

（2）返回值的测试条件都不满足。

那么对于本例来说，系统将会在以下三种情况都满足的情况下执行本作业步：

· 一个先前作业步异常终止。

· 所有先前作业步的返回码大于等于 17。

· STEP1 的返回码小于等于 89。

例 3．

```
//STEP1   EXEC   PGM=CINDY
          .
          .
          .
//STEP2   EXEC   PGM=NEXT,COND=(4,EQ,STEP1)
          .
          .
          .
//STEP3   EXEC   PGM=LAST ,COND=((8,LT,STEP1),(8,GT,STEP2))
          .
```

在本例中，如果 STEP1 的返回码为 4，STEP2 将不被执行。在 STEP3 执行前，系统将执行第一个返回码测试。而由于 STEP2 并未被执行，所以将不会进行第二个返回码的测试。由于 8 大于 4 所以 STEP3 被执行。

例 4．

```
//STP4   EXEC   PROC=BILLING,COND.PAID=((20,LT),EVEN),
//       COND.LATE=(60,GT,FIND),
//       COND.BILL=((20,GE),(30,LT,CHGE))
```

在本例中的 EXEC 语句调用了一个名叫 BILLING 的过程。这条语句中定义了几个不同的分别对过程步 PAID、LATE、BILL 的返回码的测试。由于设置了 EVEN 子参数，除非相应的返回值测试满足条件，那么即使先前作业步异常终止，过程步 PAID 都将被执行。

6． PARM

用于向本作业步执行的程序传递变量信息。该程序必须有相应的指令接收这些信息，并使用它们。

格式：

PARM[.过程步名]= 子参数

PARM[.过程步名]=(子参数，子参数)

PARM[.过程步名]=(‘子参数’，子参数)

PARM[.过程步名]=’子参数，子参数’

包括所有的逗号、撇号以及括号在内，所有子参数的总长度不得超过 100 个字符。当某子参数中含有特殊字符或空格时，可以将该子参数用撇号括起来，在其它子参数一起用括号括起来，或将所有在参数用撇号括起来。

子参数：包含传递给程序的变量信息。

例 1．

```
//RUN3   EXEC   PGM=APG22,PARM=’P1,123,P2=5’
```

在本例中，系统将参数 P1、123 及 P2=5 传递给程序 APG22。

例 2．

```
// STP6   EXEC   PROC=ASFCLG,PARM.LKED=(MAP,LET)
```

在本例中系统将 MAP、LET 传递到过程 ASFCLG 中名为 LKED 的过程步。

2-4 DD 语句

- EXEC 语句的作用

2-4-1　DD 语句作用

数据定义语句（DD 语句）用来定义一个数据集以及该数据集所需的输入输出资源。DD 语句的参数较多而且比前面介绍的 JOB 和 EXEC 语句更加复杂，我们将在下章在详细讨论。

DD 语句格式：

```
//[DD 名]  DD [位置参数][，键字参数]。。。[注释]
```

DD 名：

DD 名是为 DD 语句定义的名字，由 1-8 个字符组成,一个作业步可以有多个 DD 语句,每个 DD 语句指向一种系统数据资源,每个作业步里的 DD 名不能重复,但不同 DD 步里可以使用相同的 DD 名.

每个 DD 名定义一种数据资源，JCL 所运行的程序中，通过这些 DD 名来引用这些资源，不同语言中使用 DD 名的方式不同，例如对于以下 DD 语句

```
//LOCCODE DD DSN=USER01.DATA(MYCODE),DISP=SHR
```

COBOL 语言中：

```
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
         FILE-CONTROL.
           SELECT LOCATION-FILE
               ASSIGN LOCCODE
               FILE STATUS LOCCODE-FILE-STATUS.
```

C 语言中：

```
FILE * fp;
...
fp=fopen(“DD:LOCCODE”,”r”);
```

还有一些有固定用途的 DD 名是由系统定义的，系统定义的 DD 名主要有：

```
JOBCAT      SYSCHK      JOBLIB      SYSCKEOV
STEPCAT     SYSIN       STEPLIB     SYSMDUMP
SYSABEND    SYSUDUMP
```

JES2 系统中还有：

```
JESJCLIN    JESMSGLG    JESJCL  JESYSMSG
```

JES3 中有：

```
JCBIN       JESJCL      JS3CATLG
JCBLOCK     JESMSGLG    J3JBINFO
JCBTAB      JOURNAL     J3SCINFO
JESJCLIN    JST         J3STINFO
JESInnnn    JESYSMSG    STCINRDR
TSOINRDR
```

DD 语句的参数也分为位置参数及关键字参数，这些参数都是可选的。每个 DD 语句只能有一个位置参数，但根据需要可以有多个关键字参数。位置参数有“ * ”、“ DATA ”和“ DUMMY ”。在本小节中将只介绍位置参数的使用，关键字参数将在下一节中介绍。

1．参数“ * ”

参数“ * ”用于开始一个流内数据集。数据记录跟在“ DD ”语句之后，其第一、二列不能是“ // ”或“ /* ”；该记录可以是任何编码，如 EDCBIC。下列符号表明流内数据记录的结束：

- 输入流中的“ /* ”。
- 表示另一个 JCL 语句开始的“ // ”。

当数据记录中需以“ // ”开始时，就必须使用 DATA 参数来代替“ * ”参数。

格式：

//dd 名　DD　*[,参数]… [注释]

例 1.

```
//INPUT1  DD  *
          ...
         data
          ...
//INPUT2  DD  *
          ...
         data
          ...
/*
```

上例中定义了两个 DD 数据，第一个以第一二列的“ // ”作结束，另第二个以“ /* ”作结束。

例 2.

```
//INPUT3  DD  *, DSNAME=&&INP3
          ...
         data
          ...
/*
```

A name such as userid.jobname.jobid.Ddsnumber.INP3 is generated.

例 3.

```
//STEP2          EXEC  PROC=FRESH
//SETUP.WORK   DD   UNIT=3400-6,LABEL=(,NSL)
//SETUP.INPUT1  DD   *
                 ...
                data
                 ...
/*
//PRINT.FRM     DD   UNIT=180
//PRINT.INP      DD   *
                ...
               data
                .
/*
```

例 3 在输入流中定义了两组数据。DD 语句“ SETUP.INPUT1 ”定义的输入数据将被编目过程中名为“ SETUP ”的过程步使用。而 DD 语句“ PRINT.INP ”定义的输入数据将被编目过程中名为“ PRINT ”的过程步使用。

2. DATA

用作一个流内数据集的开始，该流内数据集里含有以“ // ”开头的语句。数据记录紧跟在“ DD DATA ”语句之后；该数据记录可以是 BCD 或 EDCBIC 编码。与“ * ”的区别是数据记录只能以“ /* ”作为结束。

格式：

//dd 名　DD　DATA[,参数]… [注释]

例 1.

```
//GROUP1  DD  DATA
            ...
           data
            ...
/*
//GROUP2  DD  DATA
            ...
           data
            ...
/*
```

上例也定义了两个 DD 数据，每个都以“ /* ”结束。

例 2.

```
//GROUP3  DD  DATA,DSNAME=&&GRP3
            ...
           data
            ...
/*
```

例 3.

```
//STEP2      EXEC  PROC=UPDATE
//PREP.DD4   DD   DSNAME=A.B.C,UNIT=3350,VOLUME=SER=D88230
//            SPACE=(TRK,(10,5)),DISP=(,CATLG,DELETE)
//PREP.IN1   DD   DATA
             .
             .
            data
             .
/*
//ADD.IN2   DD   *
             .
             .
            data
             .
/*
```

3． DUMMY

DUMMY 参数用于标明：

（1）　没有设备或外存空间分配给该数据集。

（2）　对该数据集不进行状态处理。

（3）　对 BASM 或 QSAM 来说，不对该数据集作输入输出操作。

用户使用 DUMMY 参数对程序进行测试。当测试完成时，如果用户希望恢复对数据集的输入输出操作时，只需将 DD　DUMMY 参数替换成完整的数据集定义 DD 语句。DUMMY 的另一个用途是在编目或流内过程中，这将会在本章后续节中讨论。

格式：

//dd 名　DD　DUMMY[,参数]…

所有在 DUMMY 语句中的参数必须在语法上是正确的。系统将对他们进行语法检查。

例 1.

```
//OUTDD1  DD  DUMMY,DSNAME=X.X.Z,UNIT=3380,
//               SPACE=(TRK,(10,2)),DISP=(,CATLG)
```

本例中 DD 语句“ OUTDD1 ”定义了一个空数据集。该语句中除 DUMMY 以外的参数将接受系统语法检查但并不起作用。

第三章　DD 语句的键字参数

- DD 语句功能
- DSNAME 参数
- DISP 参数
- VOLUME 参数
- SPACE 参数
- DCB 参数
- SYSOUT 参数

3-1　DD 语句功能

DD 语句的键字参数比 JOB 语句和 EXEC 语句的参数相对复杂，不但数目更多，而且其功能需要更多的介绍，因此专门分出一章来进行说明。DD 语句的键字参数主要分为两类，一类与设备相关，如 UNIT、VOLUME、SPACE、LABLE 等，令一类与数据集或数据相关，如 DSNAME、DISP、DCB、RECORG、EXPDT、RETPD、REPOTECT、SYSOUT、HOLD 等。DD 语句通过这些参数主要完成以下功能：

1）定义顺序数据集或分区数据集名
2）描述设备类型、数量
3）描述数据集状态、属性以及保留期限
4）设置数据集的纪录格式、占用空间
5）描述作业的 处理方式

DD 语句的键字参数很多，全部参数的了解可以参照《MVS JCL Reference》下面我们来对一些常用参数的含义与用法进行详细介绍：

3-2 DSNAME 参数

DSNAME:

使用 DSNAME 参数来指定使用的数据集的名字。如果是要新建数据集，这个名字将指派给新建的数据集，如果是使用旧数据集，系统用这个名字去查找。

参数格式：

{DSNAME} =name

{DSN }

name: 数据集的名字。数据集的名字可以是一般数据集，直接把数据集名字写在这里；还可以是系统自动生成的临时数据集，使用“ &&dsname ”的格式，还可以是间接地使用其它地方已经指定的 DD 名，使用“ *.ddname ”、“ *.stepname.ddname ”或“ *.stepname.procstepname.ddname ”的格式。

例一：

```
//DD1 DD DSNAME=USER01.ALPHA,DISP=(,KEEP),
//       UNIT=3420,VOLUME=SER=389984
```

DD 名 DD1 定义了一个新文件，并给它起名为“ USER01.ALPHA ”。后面的作业步或其它作业可以用“ USER01.ALPHA ”这个名字和指定的 UNIT、VOLUME 参数去使用它

例二:

```
//DDSMS1 DD DSNAME=ALPHA.PGM,DISP=(NEW,KEEP),DATACLAS=DCLAS1,
//          MGMTCLAS=MCLAS1,STORCLAS=SCLAS1
```

DD 语句新建了一个在 SMS 系统管理下的数据集,名字叫"ALPHA.PGM"，后面的作业步只要使用“ ALPHA.PGM ”这个名字就可以使用它

例三：

```
//DD2 DD DSNAME=LIB1(PROG12),DISP=(OLD,KEEP),UNIT=3350
//       VOLUME=SER=882234
```

DD 语句指定分区数据集 LIB1 中的成员 PROG12 作为输入。

例四：

```
//DDIN DD DATA,DSNAME=&&PAYIN1
.
data
.
/*
```

以上 DD 语句为流内数据定义了一个系统生成的临时数据集，这个数据集名字的最后部分是 PAYIN1,这样系统会自动生成名字叫“ 用户名.作业名.作业号.DD 序号.PAYIN1 ”的数据集。

例五：

```
//DDOUT DD DSNAME=&&PAYOUT1,SYSOUT=P
```

DD 语句 DDOUT 定义了一个数据集存放 SYSOUT 数据,这个数据集是系统生成的临时数据集,它的名字的最后部分是 "PAYOUT1", 系统实际生成的数据集名是 "userid.jobname.jobid.Ddsnumber.PAYOUT1"

例六:

```
//DD3 DD DSNAME=&&WORK,UNIT=342E
```

DD 名 DD3 定义了一个临时数据集，本作业步结束后这个数据集将被自动删除。下面这个例子可以看到临时数据如何传给后面的作业步用：

例七：

```
//STEP1 EXEC PGM=CREATE
//DD4 DD DSNAME=&&ISDATA(PRIME),DISP=(,PASS),UNIT=(3350,2),
//       VOLUME=SER=334859,SPACE=(CYL,(10,,2),,CONTIG),DCB=DSORG=IS
//STEP2 EXEC PGM=OPER
//DD5 DD DSNAME=*.STEP1.DD4,DISP=(OLD,DELETE)
```

DD4 里面定义了临时数据集&&ISDATA(PRIME)，由于使用了 DISP=(,PASS)参数，指定作业步结束时临时数据集不被清除，而能传给下个作业步使用，所以在 DD5 里可以用“*.STEP1.DD4”的形式间接引用它。

DISP：

DISP 参数用来告诉系统要使用的数据集的状态是什么并且告诉系统当本作业步执行完毕后如何处理这个数据集。你可以为作业步正常结束和非正常结束时分别指定不同的处理方法。

语法：

```
{DISP=status }
{DISP=([status][,normal-termination-disp][,abnormal-termination-disp])}
DISP= ( [NEW] [,DELETE ] [,DELETE ] )
        [OLD] [,KEEP ]   [,KEEP ]
        [SHR] [,PASS ]   [,CATLG ]
        [MOD] [,CATLG ]  [,UNCATLG]
        [, ]  [,UNCATLG]
        [, ]
```

状态说明：

数据集状态描述：

NEW：

指明数据集要在这个作业步中生成。

OLD：

指明数据集在本作业步前已经存在，这里用排他方式使用它

SHR：

又可以写作“ SHARE ”，指明指明数据集在本作业步前已经存在，这里用共享方式使用它。

MOD

指明是以下两种情况之一：

. 如果数据集已经存在，在本作业步里的新数据会加到原文件末尾，要求数据集必须是顺序文件。

. 如果数据集不存在,在本步里创建它.

以上两种情况下,数据集都是排他性使用的，文件打开后，读写指针都会移到文件末尾。

正常结束时的处理参数：

DELETE：

指明本作业步结束后系统删除此数据集。

KEEP：

指明作业步结束时数据集仍然保留

PASS：

指明数据集要保留给后面的作业步使用,一般对临时数据集使用.

注意以下两点：

1. 数据集只能在一个作业内部传递，如果是临时数据集，作业结束时仍然会被删除。

2. 被传递下去的数据集会占用一个 JOB 中最大允许 DD 数里的一个名额。

CATLG：
　　指明作业步结束后自动对数据集编目
UNCATLG：
　　指明作业步结束后自动清除数据集编目

非正常结束时的处理参数：
DELETE：
　　指明本作业步结束后系统删除此数据集。
KEEP：
　　指明非正常结束时数据集仍然保留
CATLG：
　　指明作业步结束后自动对数据集编目
UNCATLG：
　　指明作业步结束后自动清除数据集编目

DISP 参数的默认值:

. 如果省略状态参数,默认值是 NEW.
. 如果省略正常结束处理参数,对新分配的数据集是 DELETE,对旧有数据时 KEEP
. 如果省略非正常结束时的处理参数,会使用与正常结束时处理参数一样的选项.但是如果正常结束处理选项是 PASS 默认的非正常结束处理选项会是对新分配的数据集是 DELETE,对旧有数据时 KEEP
. 如果没有给出 DISP 选项,默认的是 DISP=(NEW,DELETE,DELETE)

例一:

```
//DD2 DD DSNAME=FIX,UNIT=342E-1,VOLUME=SER=44889,
//       DISP=(OLD,,DELETE)
```

DD 语句定义了一个已存在的数据集,DISP 参数省略了第二个参数,根据默认规则,它将是 KEEP,如果作业步非正常结束,数据集将被删除.

例二:

```
//STEPA  EXEC PGM=FILL
//DD1 DD DSNAME=SWITCH.LEVEL18.GROUP12,UNIT=3350,
//       VOLUME=SER=LOCAT3,SPACE=(TRK,(80,15)),DISP=(,PASS)
//STEPB  EXEC PGM=CHAR
//DD2 DD DSNAME=XTRA,DISP=OLD
//DD3 DD DSNAME=*.STEPA.DD1,DISP=(OLD,PASS,DELETE)
//STEPC  EXEC PGM=TERM
//DD4 DD DSNAME=*.STEPB.DD3,DISP=(OLD,CATLG,DELETE)
```

DD1 定义了一个新数据集,并且指明作业步正常结束时数据集可以传给后面使用,如果 STEPA 非正常结束,这个数据集将被删除,因为它是一个新分配的数据集且没指明非正常处理选项.在 DD3 里重复使用上面这个数据集,正常结束时仍然传下去,非正常结束时把它删掉.DD4 里仍然使用这个数据集,正常结束后为它编目,非正常结束时把它删掉.

UNIT:
使用 UNIT 参数来指明数据集在:
. 某一特定设备上,或
. 某种类型的设备或某组设备上,或
. 与另一个数据集在相同的设备上.

当系统中有 SMS 时对于 SMS 管理的数据集不需要使用 UNIT 参数,使用 STORCLAS 参数或让系统自动指定一个 storage class 来存放数据集

语法:

```
{UNIT=( [ddd ]      [,unit-count]   [,DEFER]) }
        [/ddd ]     [,P ]
        [/dddd ]    [, ]
        [device-type]
        [group-name]
{UNIT=AFF=ddname }
```

子参数定义:

device-number:

通过一个设备号来指定一个特定的设备.设备号是系统安装时为连到系统中的所有设备如磁盘机、磁带机、读卡器等都分配一个唯一的设备号，设备号由 3-4 位字符构成。比如当某个磁带机的设备号是 0181 时，如果指定 UNIT=1801，则数据集在这个读带机上。

注意:只在必要时直接指定设备号，如果指定的设备忙，作业只能等待。另外像固定磁盘机这样的设备，通过指定设备类型(如 UNIT=3390)和卷名(VOLUME=SER)能起到相同的效果。

device-type：

指定设备类型名。IBM 为每种类型的设备规定一个名字，通常是由数字组成。比如 UNIT=3390，代表磁盘设备。当设备类型名中有连字符时，可以不用引号括起来，如 UNIT=3400-5。常见的设备类型有 3480、3422、3430 是不同型号的磁盘机，3340、3350、3375、3380、3390 是不同型号的磁盘设备。

group-name：

指定一个设备组名。设备组由一个或多个设备组成,一个设备组里的设备类型可以相同也可以不同,如一个设备组可以既包括磁盘又包括磁带设备,但更一般的情况是包括相同类型的设备.设备组名在系统安装时分配,由 1-8 个字符组成.一般系统中常见的定义有 SYSDA、DASD、TAPE 等名字。

unit-count：

指明这个数据集使用的设备个数，是一个从 1 到 59 的十进制数.

DEFER：

指明数据集在指定的设备上，但系统并不立即安装这个设备，直到程序打开这个文件时才安装。

AFF=ddname

指定数据集使用和另一个 DD 语句所使用相同的设备。

例一：

```
//STEP1 EXEC PGM=MYPGM
//DDA   DD DSNAME=USER01.DATA1,DISP=SHR,UNIT=SYSDA,VOLUME=SER=(WORK01)
```

DDA 在 SYSDA 设备组上建一个数据集

例二：

```
//STEP2 EXEC PGM=POINT
//DDX DD DSNAME=EST,DISP=MOD,VOLUME=SER=(42569,42570),
//       UNIT=(3480,2)
//DDY DD DSNAME=ERAS,DISP=OLD,UNIT=3480
//DDZ DD DSNAME=RECK,DISP=OLD,
//       VOLUME=SER=(40653,13262),UNIT=AFF=DDX
```

DDX 语句使用了 2 个 3480 磁带机设备，磁带的卷标分别是 42569 和 4257E，DDZ 使用与 DDX 相同的两个读带机，但使用不同卷标的磁带，作业步运行时操作员要去更换磁带。

3-5 VOLUME 参数

VOLUME：

通过 VOLUME 参数可以指定所引用的数据集所在的卷或卷组，也用来指定新建的数据集所在的卷或卷组。使用这个参数时，用户可以指定某个特定的卷、或卷组或另与一个 DD 相同的卷。对跨多个卷的数据集，如果作为输入数据，可以指定第一个卷，如果作为输出数据，这个参数用来指定依次使用的数据卷。建立新数据集时可以不指定 VOLUME，这时系统自动分配卷。

语法：

```
{VOLUME} = ([PRIVATE]   [,RETAIN] [,volume-sequence-number] [,volume-count]
{VOL }                  [,      ] [,                       ]
            [SER=serial-number ]
            [SER=(serial-number[,serial-number]...)]
        [,] [REF=dsname ]
            [REF=*.ddname ]
            [REF=*.stepname.ddname ]
            [REF=*.stepname.procstepname.ddname ]
```

子参数说明：

PRIVATE：

请求一个私有卷，私有的含义是：

. 系统不在这个卷上分配其他数据集，除非有 JCL 使用 VOLUME=SER 来申请这个卷

. 如果数据集在磁带上，当数据集关闭时，磁带将被卸载，除非同时使用了 RETAIN 参数或 DISP 指定了 PASS.

. 如果数据集在可卸载直接存取卷上，当数据集关闭时，卷将被卸载 If a demountable direct access volume, the volume is to be demounted

RETAIN：

对于一个私有磁带卷来说，RETAIN 指明当数据集关闭或作业步结束时不卸载卷，对公共磁带卷来说，如果这个卷在作业中被卸载，它将保留在这个设备上。

volume-count：

卷数量，用来指定一个输出数据集所申请的卷的最大数量，是一个从 1 到 255 的十进制数。

SER=serial-number

SER=(serial-number[,serial-number]...)

指明数据集存放或将要存放的卷的序列号.卷序列号是一个 1-6 字符的名字,它是在卷的初始化时给定的.一个 DD 语句里最多可以罗列 255 个卷号,也就是一个顺序数据集最多跨 255 个卷.

3-5　VOLUME 参数（续页）

REF=dsname
REF=*.ddname
REF=*.stepname.ddname
REF=*.stepname.procstepname.ddname

用来指定使用和某一个 DD 语句相同的卷.

例一:

```
//DD1 DD DSNAME=USER01.DATA3,UNIT=SYSDA,DISP=OLD,
//       VOLUME=(PRIVATE,SER=548863)
```

DD 语句使用一个已经存在的数据集,它在 SYSDA 组上,卷号是 548863。由于使用了 PRIVATE 子参数，除非另有 JCL 指定这个卷，系统不会自动向这个卷上分配数据集。

例二：

```
//DD3 DD DSNAME=QOUT,UNIT=3400-5
```

这个 DD 语句定义了一个数据集在作业步里生成，作业步结束时删除（如果没有给出 DISP 选项,默认的是 DISP=(NEW,DELETE,DELETE)）由于没有指定卷名，系统会直接使用 3400-5 设备上的一个卷。

例三:

```
//DD4 DD DSNAME=NEWDASD,DISP=(,CATLG,DELETE),UNIT=3350,
//       VOLUME=SER=335006,SPACE=(CYL,(10,5))
```

DD4 定义了一个数据集在 335006 卷上,这个卷是安装在某个 3350 磁盘机上的永久卷。有另一种指定方式和它有相同的效果，就是直接使用 UNIT=设备号的方式直接指定那个磁盘机的设备号。

例四：

```
//OUTDD DD DSNAME=TEST.TWO,DISP=(NEW,CATLG),
//         VOLUME=(,,,3,SER=(333001,333002,333003)),
//         SPACE=(TRK,(9,10)),UNIT=(3330,P)
//NEXT  DD DSNAME=TEST.TWO,DISP=(OLD,DELETE)
```

OUTDD 创建了一个跨卷数据集，数据集最多可用 333001,333002,333003 三个卷，正常结束时编目它，如果实际数据没有那么多，也可能使用不到那么多卷。NEXT 语句里指定当作业步正常结束时删除这个数据集。

例五：

```
//STEP1 EXEC PGM=....
//DD1   DD DSN=OLD.SMS.DATASET,DISP=SHR
//DD2   DD DSN=FIRST,DISP=(NEW,CATLG,DELETE),VOL=REF=*.DD1
//STEP2 EXEC PGM=...
//DD3   DD DSN=SECOND,DISP=(NEW,CATLG,DELETE),VOL=REF=*.STEP1.DD1
```

STEP1 中的 DD1 定义了一个数据集 OLD.SMS.DATASET。STEP1 种的 DD2 创建一个数据集，使用与 DD1 相同的卷，STEP2 的 DD3 又使用这个卷创建了一个数据集

SPACE:

使用 SPACE 参数指定在磁盘上创建新数据集时，为新数据集分配空间的大小,可以使用两种方式指定空间：

. 告诉系统需要多少空间，由系统去分配。

. 直接指定系统要为数据集分配哪几个磁道。

一般常使用的是让系统去分配具体的磁道，JCL 只提供需要空间的大小。对磁带而言，SPACE 参数没有意义。

语法：

由系统指定磁道：

```
SPACE= ({TRK,}(primary-qty[,second-qty][,directory])[,RLSE][,CONTIG][,ROUND])
       ({CYL,}            [,          ][,index ]    [, ]   [,MXIG ]
       ({blklgth,}                                         [,ALX ]
       ({reclgth,}                                          [, ]
```

指定特定磁道:

```
SPACE= (ABSTR,(primary-qty,address [,directory])
[,index ]
```

只指定目录空间：

```
SPACE=(,(,,directory))
```

子参数说明：

由系统指定磁道时：

TRK：

以磁道为单位分配空间。不同的磁盘设备每个磁道的容量不同，如常用的 3390 设备使用的磁盘每个磁道有 56832 字节。

CYL：

以柱面为单位分配空间。以 3390 设备为例，一个柱面有 15 个磁道。

blklgth ：

数据的平均块长度（字节数），以指定的块长度为单位分配空间，块长度的范围是 0-65535

reclgth：

记录平均长度（字节数），以记录的平均长度为单位分配空间，记录长度范围是 0-65535

3-6　SPACE 参数（续页 1）

primary-qty

首次分配数量。分配数据空间时，首次分配时以前面指定的单位分配多少数目的磁道、柱面、块数或记录数。如果分配的是分区数据集，以 TRK 和 CYL 为单位时，这部分空间要包括 PDS 的索引，如果以记录长度或块长为单位时，分配的空间里不包括索引，索引另外分配。数据集所在卷里的空闲空间必须能够容纳首次分配的空间，否则 JCL 会报错停止。

second-qty：

次级分配数量。向数据集写入数据时，如果已分配的空间使用完了系统会自动按照次级分配数量再分配出一定量的空间给数据集使用。次级空间的分配次数是有限的，比如 15 次，如果已经分配了 15 个次级空间，数据集仍然写满，再写时会出错停止。

directory：

指明目录空间要分配的大小，只在创建分区数据集（PDS）时需要指定，每个目录是一个 256 字节的记录。

index：

为索引顺序数据集指定索引的大小。

RLSE ：

指明当数据集关闭时，没有用到的空间被释放掉。数据集必须是为写入打开，而且最后一个操作是写入时才起作用。

CONTIG：

指明为数据集分配的空间必须是物理连续的，这个参数只对初次分配有效。

MXIG：

指明为数据集分配的空间必须是：(1)卷上最大的连续空间，或者(2)等于或大于初次空间大小。这个参数只对初次分配空间有效。

ALX：

指明系统分配空间时将使用最多 5 个最大的连续空间，每都大于或等于首次分配空间。只对首次分配空间有效。

ROUND：

若使用块长作为分配单元，实际分配空间会取整为若干柱面。以 TRK 或 CYL 为单元时忽略此选项.

例一:

```
//DD1 DD DSNAME=&&TEMP,UNIT=MIXED,SPACE=(CYL,10)
```

以上 DD 语句分配了一个临时数据集,UNIT 参数指定的是磁带机和磁盘机的混合存储设备。如果系统把它分配到了一个磁带上，SPACE 参数将被忽略，如果被分配到了磁盘上，会一个 10 个柱面的空间。

例二：

```
//DD2 DD DSNAME=PDS12,DISP=(,KEEP),UNIT=3350,
//       VOLUME=SER=25143,SPACE=(CYL,(10,,10),,CONTIG)
```

以上 DD 定义了一个新的分区数据集，系统为它分配 10 个柱面，其中有 10 个 256 字节的记录用来记载 PDS 的索引信息。由于使用了 CONTIG 子参数，系统将为它分配连续空间

例三：

```
//DD3 DD DSNAME=MULTIVOL,UNIT=3350,DISP=(,CATLG),
//       VOLUME=SER=(223344,223345),SPACE=(CYL,(554,554))
```

这个例子中建立了一个跨两个 3350 卷的数据集，由于每个 3350 卷总共有 555 个柱面，除去一个用来存放 VTOC 数据外，其余的柱面只有 554 个，所以这个数据集占满了两卷。

DCB:

用 DCB 参数来给定新建数据集的 Data Control Block (DCB)，在 DCB 里指定数据集的类型、记录长度等信息

命令语法：

```
[ DCB=(subparameter[,subparameter]...)                      ]
[ DCB= ( {dsname                       }[,subparameter]...) ]
[      ({*.ddname                      }                  ) ]
[      ({*.stepname.ddname             }                  ) ]
[      ({*.stepname.procstepname.ddname}                  ) ]
```

主要 DCB 参数：

DSORG：

DSORG 指明数据集的组织形式，可选值有：

PS：physical sequential ,

IS：indexed sequential

PO: partitioned

DA: direct

KEYLEN:

KEYLEN 指明记录的健字长。KEYLEN 的值是包括在 BLKSIZE 或 LRECL 之外的,具体使用方法请参照 PDSE 数据集的参考.

LRECL:

LRECL 指明记录长度字节数,如果数据集是变长纪录,要给出记录的最大长度.

RECFM:

RECFM 指定记录格式,可选值:

F: fixed-length records

V: variable-length records

D: ASCII variable-length records

U: undefined-length records

FB: fixed-length blocked records

VB: variable-length blocked records

DB: ASCII variable-length blocked records

UB: undefined-length blocked records

OPTCD=W

指明写入校验选项打开。系统向这样的文件写入数据后，要再自动读出校验。

例一：

```
//DD1A DD DSNAME=EVER,DISP=(NEW,KEEP),UNIT=3380,
//        DCB=(RECFM=FB,LRECL=326,BLKSIZE=23472),
//        SPACE=(23472,(200,40))
```

DD1A 定义了一个新数据集，存放在 3380 设备上，数据集格式是定长块文件，记录长度 325 字节，块大小 23472，空间分配以块为单位，首次分配 200 块，次级分配 40 块。

```
//DD1B DD DSNAME=EVER,DISP=(NEW,KEEP),UNIT=3380,
//        RECFM=FB,LRECL=326,
//        SPACE=(23472,(200,40))
```

上面的 DD 语句与 DD1A 的定义效果一样，但使用了不同的语法。

例二：

```
//DD2 DD DSNAME=JST,DISP=(NEW,KEEP),UNIT=SYSDA,
//       SPACE=(CYL,(12,2)),DCB=(A.B.C,KEYLEN=8)
```

DD4 指定的数据集 DCB 格式使用与数据集“ A.B.C “ 一致,但 KEYLEN 子参数的值用 8 覆盖。

例三：

```
//DD3 DD DSNAME=SMAE,DISP=OLD,
//       DCB=(*.STEP1.PROCSTP5.DD8)
```

以上 DD 语句定义了一个数据集，它的 DCB 参数与 STEP1 作业步中调用的 PROCSTP5 过程的叫 DD8 的 DD 语句中定义的数据集一致。

SYSOUT：

SYSOUT 指定这个数据集是一个系统输出数据集。用 SYSOUT 参数可以：

. 可选性地把 SYSOUT 数据指定到某个输出类（output class）里，输出类是在 JES 安装时定义的

. 可选性地要求由外部程序来处理输出，而不是 JES2

. 可选性地给输出数据指定一个格式

. 可选性地引用 JES2 的/*OUTPUT 语句参数来处理

语法:

```
SYSOUT= { class                                  }
        { *                                      }
        { ([class]  [,writer-name]  [,form-name])}
                    [,INTRDR ]      [,code-name]
SYSOUT=(,)
```

子参数说明:

class:

指定 SYSOUT 的输出类,输出类是一个从 A 到 Z 或从 0 到 9 的字母,输出类是 JES 安装时定义的,每个输出类有不同属性.

"*":

指明使用 JOB 语句里 MSGCLASS 参数指定的输出类

",":

指定空输出类,当引用 JCL 的 OUTPUT 中的 CLASS 参数的值时,必须定义空类.

writer-name:

指明一个 1-8 字节的书写器名字,输出将由这个书写器处理.书写器要预先在系统中定义.

INTRDR:

指明要把系统输出转接到 JES 的内部读卡器中,其效果是本作业步的输出将作为另一个 JCL 提交执行.

form-name:

指定一个输出格式名.

例一：

```
//DD1 DD SYSOUT=P
```

上例中 DD 语句指明把 SYSOUT 数据写到处理输出类 P 的设备上。

例二:

```
//DD2 DD DSNAME=&&PAYOUT1,SYSOUT=P
```

上例中,DD 语句定义了一个尾名是 PAYOUT1 的系统生成数据集,它的实际名字会是 "userid.jobname.jobid.Ddsnumber.PAYOUT1 格式,DD 语句将使 JES 把这个数据集输出到输出类 P 里面.

例三:

```
//JOB3 JOB ,'ERIC.VENN',MSGCLASS=C
//STEP1 EXEC PGM=SET
//DDX DD SYSOUT=*
```

上例中指定 DDX 作为 SYSOUT,由于使用 SYSOUT=*,本语句将使用 JOB 里 MSGCLASS=C 所指定的输出类 C

第四章　特殊的 DD 语句

- 系统定义的 DD 语句
- JOBCAT DD 语句
- JOBLIB DD 语句
- STEPCAT DD 语句
- STEPLIB DD 语句
- SYSABEND, SYSMDUMP, 和 SYSUDUMP DD 语句
- SYSIN DD 语句

- JOBCAT
- JOBLIB
- STEPCAT
- STEPLIB
- SYSABEND
- SYSCHK
- SYSCKEOV
- SYSIN
- SYSMDUMP
- SYSUDUMP

上一章介绍了用户自己使用的 DD 语句的格式,每个 DD 语句的 DD 名由用户根据需要自己编写.在系统中有若干特定的 DD 名字,系统预先为它赋予了特定的含义,这些 DD 名只能按照系统对它的规定使用,这些 DD 名就是特殊 DD 语句.用户可以根据这些特殊 DD 语句的功能来确定是否需要在自己的 JCL 里面使用它们.

JOBCAT 语句

功能:

使用 JOBCAT 语句可以为本 JOB 指定一个专用的目录,系统按照名字去查找数据集时,先到专用目录里去找,找不到时再到主目录或根据数据集前缀到相应的目录里去查找.

语法：

```
//JOBCAT DD DISP={OLD},DSNAME=private-catalog-name[,parameter]... [comments]
                 {SHR}
```

JOBCAT DD 的参数：

不要为专用目录指定任何 unit 或 volume 参数，系统会从主目录里找到专用目录数据集的信息

与 STEPCAT DD 的关系：

JOBCAT DD 只对没有指定 STEPCAT 的作业步起作用。

在 JCL 中的位置：

. 把 JOBCAT DD 放在 JOB 语句之后，第一个 EXEC 语句之前。

. 如果使用 JOBLIB DD 语句，要把它放在 JOBCAT DD 语句之前。

例子：

```
//EXAMPLE JOB  WILLIAMS,MSGLEVEL=1
//JOBLIB  DD   DSNAME=USER.LIB,DISP=SHR
//JOBCAT  DD   DSNAME=LYLE,DISP=SHR
//STEP1   EXEC PGM=SCAN
```

例子中使用了一个专用目录

JOBLIB DD 语句

功能：

使用 JOBLIB 语句可以:

. 创建一个私有库

. 指定一个私有库，系统到里面去找 EXEC 语句 PGM 参数指定的程序名，只有在私有库中找不到要执行的程序时才会去到系统数据集里查找。

指定的私有库必须是一个 PDS 数据集，里面存放可执行程序。

语法：

```
//JOBLIB DD parameter[,parameter]... [comments]
```

JOBLIB DD 语句的参数：

当引用已编目 PDS：

. 要编写 DSNAME 参数

. 编写 DISP 参数，状态必须是 OLD 或 SHR

. 不要编写 VOLUME 或 UNIT 参数.

当引用未编目 PDS：

. 编写 DSNAME 参数

. 编写 DISP 参数，并且必须是 DISP=(OLD,PASS)或 DISP=(SHR,PASS)

. 编写 UNIT 参数

. 编写 VOLUME 参数

当创建一个 PDS 时：

. 编写 DSNAME 参数指定一个名字

. 编写 UNIT 参数，必须指定一个磁盘设备。

. 编写 VOLUME 参数指定卷

. 编写 SPACE 参数分配足够空间

. 编写 DISP

如果需要在作业里指定一个以上私有库,可以:

1) 编写一个 JOBLIB DD 语句,定义一个库

2) 紧跟着编写另一个 DD 语句定义另一库,这个语句不要写 DD 名,如果需要,可以再跟若干个这样的语句.

系统会根据各个库定义时的顺序去找可执行代码.例如

```
//JOBLIB  DD  DSN=CEE.SCEERUN,DISP=SHR
//        DD  DSN=SYS2.DB2.SDSNLOAD,DISP=SHR
```

在 JCL 中的位置:

. JOBLIB DD 语句必须紧跟在 JOB 语句之后,两者之间不能再有其他语句.

. 如果 JOBLIB 指定一个以上私有库,其他私有库定义必须紧跟在第一个之后

. JOBLIB DD 不能放在 JCL 过程里面

与 STEPLIB DD 语句的关系：

STEPLIB 用来定义某个作业步的私有库.如果既定义了 JOBLIB,又为某个作业步定义了 STEPLIB,这个作业步中会先到 STEPLIB 中查找,再到系统库中查找,JOBLIB 被忽略.

例一：

```
//PAYROLL JOB  JONES,CLASS=C
//JOBLIB  DD   DSNAME=PRIVATE.LIB4,DISP=(OLD,PASS)
//STEP1   EXEC PGM=SCAN
//STEP2   EXEC PGM=UPDATE
```

上例中指定了一个私有库 PRIVATE.LIB4,这个私有库是编目过的.系统先到 PRIVATE.LIB4 里去查找程序 SCAN 和 UPDATE,找不到时再到系统库 SYS1.LINKLIB 里查找.

例二：

```
//PAYROLL JOB FOWLER,CLASS=L
//JOBLIB  DD DSNAME=PRIV.DEPT58,DISP=(OLD,PASS),
//          UNIT=3350,VOLUME=SER=D58PVL
//STEP    EXEC PGM=DAY
//STEP2   EXEC PGM=BENEFITS
```

上例中指定了一个私有库 PRIV.DEPT58,这个私有库是未编目过的,所以要给出它的 UNIT 和 VOLUME 参数.系统先到 PRIV.DEPT58 里去查找程序 DAY 和 BENEFITS,找不到时再到系统库 SYS1.LINKLIB 里查找.

例三：

```
//PAYROLL JOB BIRDSALL,TIME=1440
//JOBLIB  DD DSNAME=KRG.LIB12,DISP=(OLD,PASS)
//         DD DSNAME=GROUP31.TEST,DISP=(OLD,PASS)
//         DD DSNAME=PGMSLIB,UNIT=3350,
//            DISP=(OLD,PASS),VOLUME=SER=34568
```

上例定义了多个私有库,系统会按照如下顺序去查找程序:

```
KRG.LIB12
GROUP31.TEST
PGMSLIB
SYS1.LINKLIB
```

STEPCAT DD 语句

功能:

为某作业步定义一个私有目录,系统根据名字查找本作业步用到的数据集时先使用私有目录,查不到时再使用主目录

语法：

```
//STEPCAT DD DISP={OLD},DSNAME=private-catalog-name[,parameter]... [comments]
                  {SHR}
```

STEPCAT DD 语句参数：

不必为 STEPCAT 指定 unit 或 volume 参数，系统会到主目录中查到。

与其他控制语句关系：

覆盖 JOBCAT 语句

JCL 中的位置：

STEPCAT DD 语句可以放在一个作业步中任何位置。

例子：

```
//STEP1   EXEC PROC=SNZ12
//STEPCAT DD DSNAME=BETTGER,DISP=SHR
```

STEPLIB DD 语句

功能：

使用 STEPLIB DD 语句可以:

. 创建一个私有库

. 指定一个私有库，系统到里面去找本步 EXEC 语句 PGM 参数指定的程序名，只有在私有库中找不到要执行的程序时才会去到系统数据集里查找。

语法：

```
//STEPLIB DD parameter[,parameter]... [comments]
```

JOBLIB DD 语句的参数：

当引用已编目 PDS：

. 要编写 DSNAME 参数

. 编写 DISP 参数，状态必须是 OLD 或 SHR

. 不要编写 VOLUME 或 UNIT 参数.

当引用未编目 PDS：

. 编写 DSNAME 参数

. 编写 DISP 参数，并且必须是 DISP=(OLD,PASS)或 DISP=(SHR,PASS)

. 编写 UNIT 参数

. 编写 VOLUME 参数

当创建一个 PDS 时：

. 编写 DSNAME 参数指定一个名字

. 编写 UNIT 参数，必须指定一个磁盘设备。

. 编写 VOLUME 参数指定卷

. 编写 SPACE 参数分配足够空间

. 编写 DISP

与其他控制语句关系：

STEPLIB 用来定义某个作业步的私有库.如果既定义了 JOBLIB,又为某个作业步定义了 STEPLIB,这个作业步中会先到 STEPLIB 中查找,再到系统库中查找,JOBLIB 被忽略.

JCL 中的位置:

可以放在作业步的 DD 语句的任何一行

例一：

```
//PAYROLL JOB  BROWN,MSGLEVEL=1
//STEP1   EXEC PROC=LAB14
//STEP2   EXEC PGM=SPKCH
//STEPLIB DD   DSNAME=PRIV.LIB5,DISP=(OLD,KEEP)
//STEP3   EXEC PGM=TIL80
//STEPLIB DD   DSNAME=PRIV.LIB12,DISP=(OLD,KEEP)
```

系统会到 PRIV.LIB5 里查找 SPKCH,到 PRIV.LIB12 里查找 TIL80.这两个库都是已编目过的.

例二：

```
//PAYROLL JOB  BAKER,MSGLEVEL=1
//JOBLIB  DD   DSNAME=LIB5.GROUP4,DISP=(OLD,PASS)
//STEP1   EXEC PROC=SNZ12
//STEP2   EXEC PGM=SNAP10
//STEPLIB DD   DSNAME=LIBRARYP,DISP=(OLD,PASS),
//          UNIT=335e,VOLUME=SER=55566
//STEP3   EXEC PGM=A1530
//STEP4   EXEC PGM=SNAP11
//STEPLIB DD   DSNAME=*.STEP2.STEPLIB,
//          DISP=(OLD,KEEP)
```

系统到 LIBRARYP 里去找 SNAP10;到 LIBRARYP 里去找 SNAP11;由于有 JOBLIB DD 语句系统到 LIB5.GROUP4 里找 SNZ12 和 A1530。

例三：

```
//PAYROLL JOB  THORNTON,MSGLEVEL=1
//JOBLIB  DD   DSNAME=LIB5.GROUP4,DISP=(OLD,PASS)
//STEP1   EXEC PGM=SUM
//STEPLIB DD   DSNAME=SYS1.LINKLIB,DISP=OLD
//STEP2   EXEC PGM=VARY
//STEP3   EXEC PGM=CALC
//STEPLIB DD   DSNAME=PRIV.WORK,DISP=(OLD,PASS)
//        DD   DSNAME=LIBRARYA,DISP=(OLD,KEEP),
//           UNIT=3350,VOLUME=SER=44455
//        DD   DSNAME=LIB.DEPT88,DISP=(OLD,KEEP)
//STEP4   EXEC PGM=SHORE
```

STEP2 和 STEP4 系统到 LIB5.GROUP4 里查找程序；STEP1 到 SYS1.LINKLIB 里查找程序；STEP3 依次到 PRIV.WORK, LIBRARYA, LIB.DEPT88,和系统库 SYS1.LINKLIB 查找.

SYSABEND, SYSMDUMP, 和 SYSUDUMP DD 语句

功能:

在作业步里使用 SYSABEND, SYSMDUMP, 或 SYSUDUMP DD 语句指定系统向哪里输出 DUMP, 以下情况下会生成 DUMP

. 作业步非正常结束

. 作业程序非正常结束,但系统恢复功能使作业步正常结束.

SYSABEND DD 语句

产生一个用户和系统区的 DUMP,包括在 SYSUDUMP 中的所有数据和:

. 本地系统队列区(LSQA),包括 229, 230, 249 子池

. 失败任务的 IO 系统控制块

DUMP 是格式化过的,可直接打印

SYSMDUMP DD 语句

产生系统区和程序区的 DUMP,数据没有格式化,是程序才能识别的二进制数据.

SYSUDUMP DD 语句

产生用户区 DUMP,内容是格式化过的,可直接打印.

语法:

```
//SYSABEND DD parameter[,parameter]... [comments]
//SYSMDUMP DD parameter[,parameter]... [comments]
//SYSUDUMP DD parameter[,parameter]... [comments]
```

例一:

```
//STEP2    EXEC PGM=A
//SYSUDUMP DD   SYSOUT=A
```

SYSUDUMP DD 语句指定把所要的 DUMP 输出到系统输出的 A 类.

例二:

```
//STEP1    EXEC PGM=PROGRAM1
//SYSABEND DD DSNAME=USER1.DUMP,UNIT=3390,DISP=(,PASS,KEEP),
//            VOLUME=SER=1234,SPACE=(TRK,(40,20))
//STEP2    EXEC PGM=PROGRAM2
//SYSABEND DD DSNAME=*.STEP1.SYSABEND,DISP=(OLD,DELETE,KEEP)
```

两个作业步中都把 DUMP 输出到 USER1.DUMP 数据集中

SYSIN DD 语句

功能:

SYSIN DD 语句用来定义一个流内数据集,数据以"DD *"开始,"/*"结束.SYSIN 定义的数据,在程序中可以从标准输入设备得到,如 COBOL 语言中可以使用"ACCEPT"语句,C 语言里可以用"scanf( )"、"gets( )"等函数从标准输入得到。如果一个 DD 语句没有起 DD 名字，系统会为它自动命名为 SYSIN.

语法:

```
//SYSIN DD parameter[,parameter]... [comments]
```

例子:

```
//STEP1 EXEC PGM=READ
//SYSIN DD *
.
.
data
.
//OUT1  DD SYSOUT=A
//STEP2 EXEC PGM=WRITE
//SYSIN DD DATA,DLM=17
.
.
.
17
```

第五章　JCL 过程

- 编目过程与流内过程
- 过程的参数与调用
- JCLLIB 语句
- 调用过程时语句的覆盖
- 一个复杂的例子

JCL 语言中也像其他语言一样，可以定义过程，一个过程是一段预先编好的 JCL 语句，能够被其他 JCL 语句调用。JCL 语言里有两种过程，一种是编目过程（Cataloged Procedure），另一种是流内过程（in-stream procedure）。

编目过程是把过程的 JCL 语句写在一个编目的数据集里，这种编目数据集必须是分区数据集（PDS）或扩展分区数据集（PDSE），每个编目 JCL 是它里面的一个成员。其他的 JCL 语言使用过程的成员名来调用它，系统把过程内容复制并替换进调用 JCL。由于过程是放在单独的数据集里，所以能够同时被多个 JCL 调用。流内过程是放在 JCL 作业当中的过程,在 JCL 过程中以 PROC 语句开始 PEND 结束的一段语句.一个 JCL 的流内过程只能被它自己使用,别的 JCL 不能使用这个过程.一个作业中最多有 15 个流内过程.

过程的编写

一个过程可以由若干作业步组成,在过程中可以包含除以下语句之外的各种 JCL 语句:

。JOB 语句

。流内数据集定义语句 “ DD * ” 到 “ /* ”

。JCL 结束语句：一行空的“ // ”

。JOBLIB 和 JOBCAT 语句

流内过程：

流内过程嵌在 JCL 里，以 PROC 语句开始，PEND 语句结束，例如：

```
//COPYDATA  JOB  NOTIFY=&SYSUID
//********* COPY DATASET ***********
//MYPROC PROC INDD=,OUTDD=
//CPY      EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=&INDD.,DISP=SHR
//SYSUT2   DD DSN=&OUTDD.,DISP=(NEW,CATLG,DELETE),
//            SPACE=(TRK,(2,1)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=240,DSORG=PS)
//SYSIN    DD DUMMY
//MYEND    PEND
//CPYLOAD EXEC MYPROC,INDD=DEVP124.DATA,
//  OUTDD=DEVP124.DATA1
```

编目过程：

编目过程要存放在一个分区数据集的成员里，语法与流内过程一样，但不需要使用 PEND 语句，如：

```
DEVP124.PROC(MYPROC)
//MYPROC PROC INDD=,OUTDD=
//CPY      EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD DSN=&INDD.,DISP=SHR
//SYSUT2   DD DSN=&OUTDD.,DISP=(NEW,CATLG,DELETE),
//            SPACE=(TRK,(2,1)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=240,DSORG=PS)
//SYSIN    DD DUMMY
```

过程使用的参数：

JCL 语句通过参数来向过程传送数据，使同一个过程能完成不同的处理，过程在 PROC 语句后面列出过程所需参数的名称和默认值。其他语句调用过程时，在 EXEC 过程名后面给出所使用的参数的具体值。过程代码中以 “ &参数名. ” 的形式引用参数，在不引起异议的情况下，后面的句点可以省略，执行时系统会使用参数的实际值来替代。例如上面的流内过程例子中，过程里有"//SYSUT1　DD DSN=&INDD.,DISP=SHR"的语句,调用时给定参数值"//CPYLOAD EXEC MYPROC,INDD=DEVP124.DATA,"　，在实际执行时会使用 "//SYSUT1　　DD DSN=DEVP124.DATA,DISP=SHR"

过程的调用:

过程的调用使用以下格式:

//作业步名 EXEC PROC=过程名,[参数名=参数值[,参数名=参数值]]

//作业步名 EXEC 过程名,[参数名=参数值[,参数名=参数值]]

调用一个过程时,系统会依流内、私有库、系统库的次序去查找过程，如果是流内过程，过程代码必须写在调用代码前面。

JCLLIB 语句

功能：

使用 JCLLIB 语句可以指定一个或多个本作业用的私有库，系统依照次序到这些库里去查找作业中调用的编目过程.

语法:

```
//[name] JCLLIB ORDER=(library[,library]...) [comments]
```

参数说明:

name:

名字是可选的,用来为 JCLLIB 语句指定一个名字,这个名字是 1-8 个字节长,如果不写名字,第三列必须是空格.

ORDER=(library[,library...])

用来给出一个库的列表,系统依照次序到这些库里去查找作业中调用的编目过程,列表中最多给出 15 个数据集

comments:

注释域是可选的,与前面的参数以一个及以上的空格间隔.

例一:

```
//MYJOB1  JOB ...
//MYLIBS1 JCLLIB ORDER=CAMPBEL.PROCS.JCL
//S1      EXEC   PROC=MYPROC1
```

上例中系统先到"CAMPBEL.PROCS.JCL"里去找编目过程"MYPORC1",如果找不到,再到系统过程库"SYS1.PROCLIB"里去找.

例二:

```
//MYJOB2  JOB ...
//MYLIBS2 JCLLIB ORDER=(CAMPBEL.PROCS.JCL,PUCHKOF.PROCS.JCL,
//           YUILL.PROCS.JCL,GARY.PROCS.JCL)
//S2 EXEC PROC=MYPROC2
```

上例中,系统会按以下顺序去寻找编目过程 MYPROC2:

1. CAMPBEL.PROCS.JCL
2. PUCHKOF.PROCS.JCL
3. YUILL.PROCS.JCL
4. GARY.PROCS.JCL
5. SYS1.PROCLIB

EXEC 语句的覆盖

使用 EXEC 语句调用过程时，该语句的所有键字参数都会影响到过程的执行。对于过程中定义的参数，将用调用语句的覆盖，过程中没有的参数，将使用调用语句的参数。调用语句可以选择专对过程中的某一个作业步的参数进行覆盖，这时使用如下格式的调用语句：

//作业步名 EXEC 过程名，参数.过程步名=值

例如有如下过程：

```
//PROCA PROC
//STEP1 EXEC PGM=PROG1
...
//STEP2 EXEC PGM=PROG2
...
//STEP3 EXEC PGM=PROG3,TIME=(1,30)
```

调用语句可以使用如下方式修改过程作业步里的参数：

```
//STEPN EXEC PROCA,COND.STEP2=(8,GT),TIME.STEP3=1440
```

使过程的 STEP2 在前面返回码大于 8 时停止执行，STEP3 可以不限时间地执行。

DD 语句的覆盖

调用过程的 JCL 中可以对过程中 DD 语句规定的参数进行修改，调用过程的作业步里，用“过程作业步名.过程 DD 名 DD”的格式定义一个 DD 语句，系统会用这个 DD 语句来覆盖过程中的 DD 语句参数。

例如有如下过程定义：

```
//PROCA PROC
//STEP1 EXEC PGM=PROG1
//SYSUT1 DD DSN=USER01.DATA1,DISP=SHR
...
```

调用时使用：

```
//STEPN EXEC PROCA
//STEP1.SYSUT1 DD DSN=MYDATA.DATA1,DISP=SHR
...
```

实际执行时，过程中 STEP1 作业步的 SYSUT1 将使用“MYDATA.DATA1”数据集

DD 语句的增加

与 DD 语句的覆盖类似，在调用过程时可以为过程中的某个作业步定义新的 DD，运行时这个 DD 语句将加到过程语句中去。

5-5 一个复杂的例子

下面我们来看一个复杂的 JCL 例子，这个例子由一个 JCL 和一个过程组成，它的功能是编译连接并 BIND 一个 COBOL 语言写的 DB2 应用程序：

JCL 内容：

```
//COMP  JOB NOTIFY=&SYSUID
//* DB2 COBOL BATCH PROCESS
//MYPROC  JCLLIB ORDER='USER01.PROC'
//CICSCBL  EXEC DB2BCOMP,MEM=SAMP1
//PC.SYSIN  DD DSN=USER01.SRC(SAMP1),DISP=SHR
//BND.SYSTSIN  DD *
DSN SYSTEM(DSND)
BIND PACKAGE(DCBPCKG) MEMBER(SAMP1) ACT(REP) ISO(CS)  VAL(BIN)
/*
```

它调用的过程如下：

```
//*********************************************************
//DB2BCOMP PROC WSPC=500,MEM=,
//             LOADLIB='MY.LOAD',
//             PRFX='USER01'
//*********************************************************
//*        PRECOMPILE THE IBM COBOL PROGRAM                          *
//*********************************************************
//PC     EXEC PGM=DSNHPC,REGION=4096K,
//            PARM='HOST(IBMCOB),SOURCE,APOST'
//STEPLIB  DD DSN=DSN710.SDSNLOAD,DISP=SHR
//         DD DSN=DSN710.SDSNEXIT,DISP=SHR
//DBRMLIB  DD DSN=&PRFX..DBRM(&MEM),DISP=SHR
//SYSCIN   DD DSN=&&DSNHOUT,DISP=(MOD,PASS),UNIT=SYSDA,
//            SPACE=(800,(&WSPC,&WSPC))
//SYSLIB   DD DSN=&PRFX..COPY,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSTERM  DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSUT1   DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=SYSDA
//SYSUT2   DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=SYSDA
```

5-5 一个复杂的例子（续页 1）

```
//*********************************************************
//*        COMPILE THE IBM COBOL PROGRAM IF THE PRECOMPILE          *
//*        RETURN CODE IS 4 OR LESS.                                *
//*********************************************************
//COB      EXEC PGM=IGYCRCTL,REGION=2048K,COND=(4,LT,PC),
// PARM='DATA(31),DYNAM,SIZE(MAX),APOST,RENT,LIB,XREF,OFFSET,DBCS,
//             SOURCE,SSRANGE,TEST'
//STEPLIB  DD DSN=IGY.SIGYCOMP,DISP=SHR
//SYSLIB   DD DSN=&PRFX..COPY,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSTERM  DD SYSOUT=*
//SYSLIN   DD DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,
//            SPACE=(800,(&WSPC,&WSPC))
//SYSIN    DD DSN=&&DSNHOUT,DISP=(OLD,DELETE)
//SYSUT1   DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=SYSDA
//SYSUT2   DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=SYSDA
//SYSUT3   DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=SYSDA
//SYSUT4   DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=SYSDA
//SYSUT5   DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=SYSDA
//SYSUT6   DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=SYSDA
//SYSUT7   DD SPACE=(800,(&WSPC,&WSPC),,,ROUND),UNIT=SYSDA
//*********************************************************
//*  PRELINK STEP.                                          *
//*********************************************************
//PLKED   EXEC PGM=EDCPRLK,REGION=2048K,COND=((4,LT,PC),(4,LT,COB))
//STEPLIB  DD  DSN=CEE.SCEERUN,DISP=SHR
//SYSMSGS  DD  DSN=CEE.SCEEMSGP(EDCPMSGE),DISP=SHR
//SYSIN    DD  DSN=&&LOADSET,DISP=(OLD,DELETE)
//SYSMOD   DD  DSN=&&PLKSET,UNIT=SYSDA,DISP=(MOD,PASS),
//             SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSLIB   DD  DUMMY
//SYSDEFSD DD  DUMMY
//SYSOUT   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSTERM  DD  SYSOUT=*
```

5-5　一个复杂的例子（续页 2）

```
//*****************************************************
//*        LINKEDIT IF THE PRECOMPILE AND COMPILE                    *
//*        RETURN CODES ARE 4 OR LESS.                               *
//*****************************************************
//LKED     EXEC PGM=IEWL,PARM='MAP',
//         COND=((4,LT,PC),(4,LT,COB),(4,LT,PLKED))
//SYSLIB   DD DSN=CEE.SCEELKED,DISP=SHR
//         DD DSN=DSN710.SDSNLOAD,DISP=SHR
//         DD DSN=ISP.SISPLOAD,DISP=SHR
//         DD DSN=&LOADLIB,DISP=SHR
//SYSLIN   DD DSN=&&PLKSET,DISP=(OLD,DELETE)
//         DD DDNAME=SYSIN
//SYSLMOD DD DSN=&LOADLIB(&MEM),DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSUT1   DD SPACE=(1024,(50,50)),UNIT=SYSDA
//*
//******* DB2 BIND (PACKAGE AND PLAN)
//*
//BND      EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//STEPLIB  DD  DSN=DSN710.SDSNLOAD,DISP=SHR
//         DD  DSN=DSN710.SDSNEXIT,DISP=SHR
//DBRMLIB  DD DSN=&PRFX..DBRM,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSTSIN  DD DUMMY
```

第六章　常用实用程序

- 实用程序介绍
- IEFBR14
- IEBCOMPR
- IEBCOPY
- IEBGENER
- DFSORT

定义：

实用程序（utility）是指由 IBM 提供的一些程序，这些程序在 OS/390 环境中完成一些常用的任务（task），比如对 DASD，磁带，的各种操作以及打印和对 DATA SETS 的 allocate，update，delete,catalog,uncatalog....

实用程序主要可以分为以下几类：

系统实用程序

以 IEH 开始的是系统实用程序，主要功能是对系统进行维护和管理、对卷和数据集进行维护和管理，主要包括：

IEHNITT:磁带写卷标

IEHLIST:系统控制信息列表

IEHMOVE:移动或拷贝若干组数据集、移动或拷贝整个卷、移动或拷贝编目目录等

IEHPROGM：建立及维护系统控制数据、建立世代数据集组索引、重命名磁带卷、删除数据集等

IDHDASDR：初始化一个直接存取卷。

数据集实用程序

以 IEB 开头的一般是数据集实用程序，其主要功能是对数据集进行操作，主要有：

IEBCOMPR：对顺序数据集、分区数据集或扩展分区数据集进行比较

IEBCOPY：对分区数据集进行拷贝、压缩或合并

IEBDG:创建含有模型数据的测试数据集

IEBEDIT: 有选择地拷贝作业步及其相关的作业语句.

IEBGENER:拷贝顺序数据集,或将顺序数据集转为分区数据集

ECBIMAGE:修改、连接或打印模块

IEBISAM：对 ISAM 数据集进行装载、卸载拷贝和打印。

IEBPTPCH：对一个数据集进行打印或穿孔卡输出

IEBUPDATE：对顺序数据集、分区数据集和扩展分区数据集进行合并和修改

独立实用程序

以 IBC 开头的程序是独立实用程序，它们是一种特殊的实用程序，可独立于操作系统运行，通常用来进行恢复或建立系统的操作，主要有：

IBCDASDI：格式化一个直接存取卷

IBCDUMPRS：转储一个直接存取卷

IEFBR14 是一个比较常用的实用程序，它实际是一个空程序，什么也不做，只是返回返回码 0，但是使用它可以进行 JCL 语言的各种处理，如 DD 语句中创建数据集等。

使用 IEFBR14 来创建数据集：

例一：

```
//DEFFILE  JOB  CLASS=A,REGION=4096K,
//         MSGLEVEL=(1,1),MSGCLASS=A,NOTIFY=&SYSUID
//DELETE   EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  DELETE TEST.FILE
/*
//ALLOC    EXEC PGM=IEFBR14
//SYSPRINT DD SYSOUT=*
//FILE1    DD DISP=(NEW,CATLG),DSN=TEST.FILE,
//            UNIT=3380,VOL=SER=MVSDL3,
//            SPACE=(3200,(360,180)),
//            DCB=(RECFM=VB,BLKSIZE=3200)
```

以上 JCL 创建了一个新的顺序数据集 TEST.FILE

例二：

```
//SAMPJOB  JOB  CLASS=A,MSGCLASS=A,MSGLEVEL=(1,1)
//DELETE   EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *
 DELETE  TEST.FILE1
/*
//DEFINE   EXEC PGM=IEFBR14
//FILE1    DD DSN=TEST.FILE1,
//         DISP=(NEW,CATLG),
//         STORCLAS=OSBASE,MGMTCLAS=OSMGMT,
//         SPACE=(6160,(500,100,200)),
//         DCB=(LRECL=80,BLKSIZE=6160,DSORG=PO,RECFM=FB)
```

上例中创建了一个分区数据集 TEST.FILE1

删除文件：

```
//IEFBR14  JOB  CLASS=A,REGION=4096K,MSGLEVEL=(1,1),
//              MSGCLASS=A,NOTIFY=&SYSUID
//STEP1    EXEC PGM=IEFBR14
//DATASET  DD DSN=TEST.BACKUP,DISP=(OLD,DELETE)
```

IEBCOMPR 程序用来对两个数据集的纪录进行比较,这两个数据集可以使顺序数据集、分区数据集或扩展分区数据集。它能对数据集或数据集的成员的定长、变长、分块或不分块纪录进行比较。

作业控制语句：:

JOB:作业语句

EXEC:指定程序名 PGM=IEBCOMPR

SYSPRINT DD:定义一个顺序数据集，用来存放比较输出信息，一般可以把它写到 SYSOUT 上 my (DUMMY DD).

SYSUT1 DD:定义输入数据集

SYSUT2 DD:定义另一个要比较的数据集

SYSIN DD :定义控制信息，当要比较的是顺序文件时，可以为 DUMMY。一般使用流内数据集，也可以使用一个 PDS 里的成员。

控制选项：

控制选项是在 SYSIN 中输入的控制程序执行的语句，主要有：

COMPAREP:

用来指名数据集的类型,如果出现,它必须是第一个控制语句,COMPAREP 的语法: COMPARE TYPORG={PS|PO},PS 指名比较的是顺序数据集,PO 指名比较的是分区数据集，默认值是 PS

EXITS :

EXITS 语法：

```
[ label] EXITS [INHDR= routinename]
               [,INTLR= routinename]
               [,ERROR= routinename]
               [,PRECOMP= routinename]
```

INHDR= routinename:

指定处理用户输入头标的例程名

INTLR= routinename:

指定处理用户输入尾标的例程名

ERROR= routinename

指定错误处理例程名

PRECOMP= routinename

指定预处理例程名,预处理例程在进行记录比较前可以对数据进行预先加工

LABELS:

用 LALES 语句指定用户标号是否也作为数据进行比较.

LABLES 语法:

```
[ label] LABELS [DATA={YES|NO|ALL|ONLY}]
```

```
DATA={YES|NO|ALL|ONLY}
```

YES:所有没有被处理例程清除的用户标号都作为数据处理;NO:用户标号不作为数据;ALL:所有用户标号都作为数据处理;ONLY:只有用户头标被认为是数据.

例一:

比较磁带上的两顺序数据集：

```
//TAPETAPE JOB ...
//         EXEC PGM=IEBCOMPR
//SYSPRINT DD SYSOUT=A
//SYSUT1   DD DSNAME=SET1,UNIT=tape,LABEL=(,NL),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=2000),
//            DISP=(OLD,KEEP),VOLUME=SER=001234
//SYSUT2   DD DSNAME=SET2,UNIT=tape,LABEL=(,NL),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=1040),
//            DISP=(OLD,KEEP),VOLUME=SER=001235
//SYSIN DD DUMMY
/*
```

以上 JCL 中 SYSUT1 定义了一个数据数据集，它存放在磁带上，SYSUT2 定义了另一个在磁带上的数据集。SYSIN 定义了一个空数据集，所有的控制命令都使用默认值。 比较结果输出到系统输出里面。

例二：

比较两分区数据集

```
//DISKDISK JOB ...
//STEP1    EXEC PGM=IEBCOMPR
//SYSPRINT DD SYSOUT=A
//SYSUT1   DD DSNAME=PDSSET1,UNIT=disk,DISP=SHR,
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=2000),
//            VOLUME=SER=111112
//SYSUT2   DD DSNAME=PDSSET2,UNIT=disk,DISP=SHR,
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=2000),
//            VOLUME=SER=111113
//SYSIN DD *
   COMPARE TYPORG=PO
/*
```

以上 JCL 中，SYSUT1 定义了一个分区数据集，SYSUT2 定义了另一个要比较的分区数据集，SYSIN 里定义的控制信息，比较结果放到系统输出里。

IEBCOPY 可以拷贝或者合并分区数据集和扩展分区数据集，详细功能有：

- ✓ 复制一个分区数据集或扩展分区数据集。
- ✓ 合并分区数据集
- ✓ 把分区数据集或扩展分区数据集拷贝成一个顺序数据集，用来备份或传输
- ✓ 把备份分区数据集的一个活动多个成员复制到其他分区数据集或扩展分区数据集中
- ✓ 选择性地复制、备份或恢复分区数据集或扩展分区数据集中的一个或多个成员。
- ✓ 替换 PDS 或 PDSE 数据集中的一个成员
- ✓ 在拷贝时对 PDS 中的成员改名。
- ✓ 压缩 PDS 数据集的空间

作业控制语句：

JOB： JOB 定义
EXEC： 定义作业步
SYSPRINT DD : 定义一个顺序数据集,来输出控制语句和结果信息.
SYSUT1 or anyname1 DD: 定义一个顺序数据集或备份的数据集作为程序的输入.
SYSUT2 or anyname2 DD： 定义一个顺序数据集或备份数据集作为输出。
SYSUT3 DD： 定义作为输入的溢出数据集。
SYSUT4 DD： 定义一个输出用的溢出数据集
SYSIN DD： 定义控制选项数据集

控制选项：

EXEC 语句控制选项：

EXEC 语句格式：

```
//[stepname] EXEC [,PGM=IEBCOPY]
                  [,REGION={ n| nK| nM}]
                  [,PARM= <parms>]
```

在 EXEC 语句的 PARM=参数中，可以以任意次序使用如下参数：

COMPRESS： 指示 IEBCOPY 对数据集进行压缩。
COPY： 指示 IEBCOPY 拷贝数据集
COPYGRP 指示 IEBCOPY 进行组拷贝。
COPYMOD 指示 IEBCOPY 进行拷贝和加载模块重组修改.
LC= n
LPP= n
LINECOUNT= n 以上三种写法等效,指定打印输出时每页的行数,程序根据这个值来打页头,默认是 60 行.
LIST=NO/YES：给出 COPY, COPYMOD, 或 ALTERMOD 命令的 LIST=参数在没给定时的默认值
RC4NOREP： 指定当因为没有指定 REPLACE 参数而引起数据集不能拷贝的情况下,程序返回返回码 04
REPLACE: 指定当存在同名输出数据集时,覆盖原文件.

SIZE={ n| nK| nM}： 指定 IEBCOPY 可以使用的虚拟内存最大值.

SYSIN 控制选项:

在 SYSIN 中编码这些选项,这里只介绍一些常用选项,主要有:

COPY 语句:

使用 COPY 语句来开始一个复制、卸载或加载操作，一个 COPY 语句中可以完成多个操作，一个作业步里可以有多个 COPY 命令。

COPY 语法：

```
[ label]    COPY    OUTDD= DDname
            ,INDD=[(]{ DDname|( DDname,R) }[,...][)] [,LIST={YES|NO}]
```

这里:

OUTDD= DDname

指名作为输出数据集的 DD 语句的 DD 名

INDD=[(]{ DDname| (DDname,R)}[,...][)]

INDD 指名作为输入数据集的 DD 语句的 DD 名，可以有多个输入数据集，R 代表当这个数据集里的成员名在输出数据集中已经存在，用输入数据集的内容覆盖输出数据集中的成员。当一个 INDD=语句单独出现（没有和一个 COPY 关键字同时出现）时它会使用当前的参数启动一个新的 COPY 操作，也就是不写 COPY 只写 INDD=时，对没有指定的参数，会用前一个 COPY 操作的参数作默认值，发起一个新的拷贝操作。

LIST={YES|NO}

指名拷贝完成的数据集名是否在输出信息中写出。

EXCLUDE 语句

EXCLUDE 语句指定若干成员名，拷贝、卸载或加载时会排出这些成员。语法是：

```
[ label] EXCLUDE MEMBER=[(] name1[, name2][,...][)]
```

SELECT 语句

SELECT 语句指定处理时要涉及的成员名。语法:

```
[ label]  SELECT  MEMBER=({ name1|
                            ( name1, newname1[,R])|
                            ( name1,,R)}
                          [,{ name2|
                            ( name2, newname2[,R])|
                             ( name2,,R)}][,...])
```

其中 name1, newname1 表示拷贝时更换成员名,"R"表示覆盖同名成员.

6-4 IEBCOPY（续页 2）

例一:

拷贝整个数据集

```
//COPY     JOB ...
//JOBSTEP  EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=A
//SYSUT1   DD DSNAME=DATASET5,UNIT=disk,VOL=SER=111113,
//            DISP=SHR
//SYSUT2   DD DSNAME=DATASET4,UNIT=disk,VOL=SER=111112,
//            DISP=(NEW,KEEP),SPACE=(TRK,(5,1,2))
```

SYSUT1 定义了一个分区数据集 DATASET5,它里面有两个成员 A 和 C;SYSUT2 定义了一个新的分区数据集 DATASET4,IEBCOPY 把 SYSUT1 的数据拷进去.

例二:合并数据集

这个例子中 PDS 的所有成员将从(DATASET1, DATASET5, 和 DATASET6)中拷贝进一个已经存在的分区数据集(DATASET2)中.

```
//COPY     JOB ...
//JOBSTEP  EXEC PGM=IEBCOPY
//SYSPRINT DD  SYSOUT=A
//IN1      DD  DSNAME=DATASET1,UNIT=disk,VOL=SER=111112,
//             DISP=SHR
//IN5      DD  DSNAME=DATASET5,UNIT=disk,VOL=SER=111114,
//             DISP=OLD
//OUT2     DD  DSNAME=DATASET2,UNIT=disk,VOL=SER=111115,
//             DISP=(OLD,KEEP)
//IN6      DD  DSNAME=DATASET6,UNIT=disk,VOL=SER=111117,
//             DISP=(OLD,DELETE)
//SYSUT3   DD UNIT=SYSDA,SPACE=(TRK,(1))
//SYSIN    DD *
COPYOPER COPY OUTDD=OUT2
         INDD=IN1
         INDD=IN6
         INDD=IN5
/*
```

例三：拷贝并覆盖指定 PDS 成员

```
//COPY     JOB  ...
//JOBSTEP  EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=A
```

```
//OUT1     DD DSNAME=DATASET1,UNIT=disk,VOL=SER=111112,
//            DISP=(OLD,KEEP)
//IN6      DD DSNAME=DATASET6,UNIT=disk,VOL=SER=111115,
//            DISP=OLD
//IN5      DD DSNAME=DATASET5,UNIT=disk,VOL=SER=111116,
//            DISP=(OLD,KEEP)
//SYSUT3   DD UNIT=SYSDA,SPACE=(TRK,(1))
//SYSUT4   DD UNIT=SYSDA,SPACE=(TRK,(1))
//SYSIN    DD *
COPYOPER COPY OUTDD=OUT1
              INDD=IN5,IN6
         SELECT MEMBER=((B,,R),A)
/*
```

这个例子中，DATASET5 有成员 A 和 C，DATASET6 有成员 B、C、D，DATASET1 原有成员 A、B、F。成员 A 从 DATASET5 中考到 DATASET1，成员 B 从 DATASET6 中考到 DATASET1，并覆盖原来 DATASET1 中的同名成员。

例四：备份并压缩一个 PDS

```
//SAVE     JOB ...
//STEP1    EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=A
//SYSUT1   DD DSNAME=PARTPDS,UNIT=disk,VOL=SER=PCP001,
//            DISP=OLD
//SYSUT2   DD DSNAME=SAVDATA,UNIT=tape,VOL=SER=TAPE03,
//            DISP=(NEW,KEEP),LABEL=(,SL)
//SYSUT3   DD DSNAME=TEMP1,UNIT=disk,VOL=SER=111111,
//            DISP=(NEW,DELETE),SPACE=(80,(60,45))
//SYSIN    DD DUMMY
//STEP2    EXEC PGM=IEBCOPY,COND=(0,NE),PARM=.SIZE=500K.
//SYSPRINT DD SYSOUT=A
//COMPDS   DD DSNAME=PARTPDS,UNIT=disk,DISP=OLD,
//            VOL=SER=PCP001
//SYSUT3   DD DSNAME=TEMPA,UNIT=disk,VOL=SER=111111,
//            DISP=(NEW,DELETE),SPACE=(80,(60,45))
//SYSIN DD *
        COPY OUTDD=COMPDS,INDD=COMPDS
/*
```

STEP1 中把分区数据集 PARTPDS 备份到 SYSUT2 定义的顺序数据集里，保存在磁带上。STEP2 种的控制语句指定输入和输出数据集都是 COMPDS 指定的数据集，实际做的是对 PDS 进行了压缩。

例五：复杂的拷贝

```
//COPY     JOB ...
//JOBSTEP  EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=A
//OUTA     DD DSNAME=DATASETA,UNIT=disk,VOL=SER=111113,
//         DISP=OLD
//INOUTB   DD DSNAME=DATASETB,VOL=SER=111115,UNIT=disk,
//         DISP=(OLD,KEEP)
//INOUTC   DD DSNAME=DATASETC,VOL=SER=111114,UNIT=disk,
//            DISP=(OLD,KEEP)
//INOUTD   DD DSNAME=DATASETD,VOL=SER=111116,DISP=OLD,
//            UNIT=disk
//INE      DD DSNAME=DATASETE,VOL=SER=111117,DISP=OLD,
//            UNIT=disk
//SYSUT3   DD UNIT=SYSDA,SPACE=(TRK,(1))
//SYSUT4   DD UNIT=SYSDA,SPACE=(TRK,(1))
//SYSIN   DD *
      COPY OUTDD=OUTA
           INDD=INE
    SELECT MEMBER=(MA,MJ)
           INDD=INOUTC
   EXCLUDE MEMBER=(MM,MN)
      COPY OUTDD=INOUTB,INDD=INOUTD
           INDD=((INOUTC,R),INOUTB)
      COPY OUTDD=INOUTD,INDD=((INOUTB,R))
     SELECT MEMBER=MM
/*
```

DATASETA(MA, MB, MD)；DATASETB(MA , MJ)；DATASETC(MF, ML, MM, MN);DATASETD(MM , MP);DATASETE(MA, MJ, MK)

SYSUT3 和 SYSUT4 定义的是临时溢出数据集。SYSIN 里定义的控制选项.第一个 COPY 语句开始一个 COPY 操作，OUTDD 指名输出数据集是 DATASETA，第一个 INDD 指明 DATASETE 是第一个要拷贝的数据集，接下来的 SELECT 语句指定 MA 和 MJ 两个成员要从 DATASETE 考到 DATASETA，处理结果将如下:

1．成员 MA 被找到但不被拷贝，因为没有指定 REPLACE 参数.

2．成员 MJ 被找到，并且被拷贝到 DATASETA 中

6-4 IEBCOPY（续页 5）

第二个 INDD 语句结束前一个操作，并开始一个新的 COPY 操作，它指定 DATASETC 作为输入，并抑制其中的 MM 和 MN 成员，结果只有 MF 和 ML 被拷贝到 DATASETA 中。

第二个 COPY 语句开始第三个拷贝操作，OUTDD 指定 DATASETB 作输出，第一个 INDD 指定 DATASETD 作为第一个输入，成员 MP 和 MM 会拷贝到 DATASETB，第二个 INDD 指定 DATASETC 作第二个输入，DATASETB 作第三个输入。DATASETC 的成员 MF、ML 和 MN 会拷贝过来，由于指定了 REPLACE 操作,MM 也将被拷过来.然后就是对 DATASETB 进行压缩.

第三个 COPY 语句中,OUTDD 指向 DATASETD,INDD 指向 DATASETB,SELECT 指定只对成员 MM 进行操作,由于指定了 REPLACE 参数,这个成员将被拷贝.

6-5 IEBGENER

IEBGENER

IEBGENER 可以用来:

- ✓ 建立顺序数据集、PDS 或 PDSE 的成员的备份，备份是一个顺序数据集。备份操作可以使从磁盘到磁盘、从磁盘到磁带和从磁带到磁带。
- ✓ 从顺序数据集产生 PDS 或 PDSE,通过控制语句,把顺序数据集划分为若干记录组,并为每个记录组分配成员名,从而把一个顺序数据集创建成含有若干成员的 PDS 或 PDSE
- ✓ 从一个顺序数据集或 HFS 中读取数据，创建成 PDS 或 PDSE 中一个成员。
- ✓ 产生一个编辑的顺序数据集、PDS 或 PDSE。通过控制语句，指定一个或一组记录或整个数据集的编辑信息。
- ✓ IEBGENER 可以处理双字节字符（DBCS）数据。
- ✓ 打印顺序数据集或 PDS/PDSE 的成员。
- ✓ 对数据集的逻辑记录重组，改变块或记录长度。
- ✓ 为顺序输出数据集拷贝用户编号。
- ✓ 为用户例程提供编辑设施及出口，该例程用于处理标号、受控输入数据及输入输出错误。

作业控制语句:

JOB: 指定 JOB 参数
EXEC: 指定程序名(PGM=IEBGENER)以及其他作业步参数.
SYSPRINT DD:指定输出信息用的顺序数据集,可以指定在 SYSOUT 上、磁盘上或磁带上
SYSUT1 DD： 指定输入数据集，可以是一个顺序数据集或者 PDS/PDSE 的一个成员。
SYSUT2 DD： 指定输出数据集，可以是一个顺序数据集或者 PDS/PDSE 的一个成员。
SYSIN DD： 指定控制语句，当输出是顺序数据集且不需要特殊处理时可以指定为 DUMMY.

控制选项:

SYSIN 中可以使用的控制选项主要有：
IEBCOPY： 主要有以下控制选项:
GENERATE： 知名成员名或别名、记录标志符文字及控制数据中的编辑信息
EXITS： 指定用户出口例程
LABELS： 指定用户标号处理例程
MEMBER： 指定要处理的 PDS/PDSE 成员名或别名
RECORD： 指定一个记录组并提供编辑信息。

例一：拷贝顺序数据集

```
//COPYDATA  JOB  NOTIFY=&SYSUID
//*  COPY DATASET
//COPY EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DSN=DEVP124.DATA,DISP=SHR
//SYSUT2 DD DSN=DEVP124.DATA2,DISP=(NEW,CATLG,DELETE),
// SPACE=(TRK,(2,1)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=240,DSORG=PS)
//SYSIN DD DUMMY
```

例二：从顺序数据集拷贝为分区数据集成员

```
//COPYDATA  JOB  NOTIFY=&SYSUID
//*  COPY DATASET
//COPY EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DSN=DEVP124.DATA,DISP=SHR
//SYSUT2 DD DSN=DEVP124.COPY(DATA1),DISP=SHR
//SYSIN DD DUMMY
```

例三：从分区数据集成员拷贝成顺序数据集

```
//COPYDATA  JOB  NOTIFY=&SYSUID
//*  COPY DATASET
//COPY EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DSN=DEVP124.COPY(DATA1),DISP=SHR
//SYSUT2 DD DSN=DEVP124.DATA3,DISP=(NEW,CATLG,DELETE),
// SPACE=(TRK,(2,1)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=240,DSORG=PS)
//SYSIN DD DUMMY
```

DFSORT 是 IBM 的 Data Facility 家族的一员,用来对数据集进行排序、拷贝和合并。DFSORT 有丰富的功能和控制语句，这里只针对常用功能作简单介绍。

作业控制语句：

JOB： 指定作业参数
EXEC： 指定作业步参数
SYSOUT DD： 定义输出信息的顺序数据集
SORTIN DD: 定义输入数据集
SORTWKdd DD: 定义一系列的排序工作数据集
SORTOUT DD: 定义排序完的数据的输出数据集
SYSIN DD: 定义控制信息

控制选项:

SORT:指定对数据集进行排序操作
MERGE:指定对数据集进行合并操作
COPY:指定对数据集进行拷贝操作
FIELDS:指定排序键值的信息,格式为
FIELDS=(起始位置,长度,升降序[,起始位置,长度,升降序]),FORMAT=数据类型
(起始位置,长度,数据格式,升降序[,起始位置,长度,数据格式,升降序])
其中位置和长度都是以字节为单位,顺序可选的有 A:升序,D:降序,E:使用用户出口例程定序;FORMAT=指定键字的类型,可选项有 CH:字符类型,PD:压缩十进制型,BI:二进制型,AC:ASCII 码型

例一:

对单个数据集排序

```
//EXAMP    JOB A492,PROGRAMMER
//SORT     EXEC PGM=SORT
//STEPLIB  DD DSN=A492.SM,DISP=SHR
//SYSOUT   DD SYSOUT=A
//SORTIN   DD DSN=A123456.SORT.SAMPIN,DISP=SHR
//SORTWK01 DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SORTOUT  DD DSN=A123456.SORT.SAMPOUT,DISP=OLD
//SYSIN    DD *
   SORT FIELDS=(110,10,A,145,17,A,
        1,75,A),FORMAT=CH
/*
```

例二：

对两数据集合并排序

```
//EXAMP    JOB A492,PROGRAMMER
//SORT     EXEC PGM=SORT
//STEPLIB  DD DSN=A492.SM,DISP=SHR
//SYSOUT   DD SYSOUT=A
//SORTIN01 DD DSN=A123456.MASTER,DISP=OLD
//SORTIN02 DD DSN=A123456.NEW,DISP=OLD
//SORTOUT  DD DSN=A123456.SORT.SAMPOUT,DISP=OLD
//SYSIN    DD *
     MERGE FIELDS=(110,5,A,1,75,A),FORMAT=CH
/*
```

例三：

拷贝数据集

```
//EXAMP   JOB A492,PROGRAMMER
//SORT    EXEC PGM=SORT
//STEPLIB DD DSN=A492.SM,DISP=SHR
//SYSOUT  DD SYSOUT=A
//SORTIN  DD DSN=A123456.SORT.SAMPIN,DISP=SHR
//SORTOUT DD DSN=A123456.SAMP.SORTOUT,DISP=OLD
//SYSIN DD *
   OPTION COPY
/*
```