第四章 作业控制语言

4.1 基本概念

在大型主机系统中，当用户需要使用计算机完成某项批处理任务时，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）。作业是用户在完成该任务时要求计算机所做工作的集合。

与 COBOL 等一般的编程语言不同，作业控制语言 JCL（Job Control Language）是批处理作业的用户与操作系统的接口。由于批处理作业的用户不能直接与他们作业交互，只能委托操作系统来对作业进行控制和干预，作业控制语言便是提供给用户，为实现所需作业控制功能委托系统代为控制的一种语言。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。总的来说，JCL 为用户提供了一种作业一级的接口，除了本章中介绍的 z/OS 环境下的 JCL，UNIX 环境下的 Shell 等语言都可以看作是一种作业控制语言。

一个作业中，每一段程序的执行称为一个作业步，一个作业可包含一个或多个作业步。一般的，作业由以下相对独立的三步组成：

（1）编译：把源程序语句（源模块）转换成目标模块；

（2）链接编辑：把目标模块同子程序库中的其他程序链接起来得到可执行模块；

（3）执行：运行可执行模块得到结果。

一个作业中的各个作业步是顺序执行的，因此一个作业步的输出可以作为下一个作业步的输入。

用户的作业可以由一个或多个作业步构成。只有一个作业步的作业叫做单步作业；由多个作业步构成的作业叫做多步作业。不论单步作业还是多步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。下面是一个多步作业的例子：

```
//JOB1     JOB  …
//STEP1    EXEC …
//DD1      DD …        作业步 1
//STEP2    EXEC …
//INDD1    DD …        作业步 2
//INDD2    DD …
//
```

除了上述一些基本概念，有关数据结构和存取方法的概念在 JCL 的使用中也是非常重要的，由于这一部分已在本书的第二章中详细讨论过，就本章不再重复了。

4．2　JCL 语句

4.2.1　JCL 语句的分类

作业控制语言 JCL 由九种语句组成，除了上一节中讨论过的三种基本语句外，还有以下六种附加语句：

（1）/* 语句：表示流内数据结束或调用 JES 控制语句；

（2）//*语句：注释语句，由第 4 到第 80 列写出注释内容；

（3）//语句：空语句，用以标记一个作业的结束；

（4）PROC 语句：流内过程（IN-STREAM　PROCEDURE）或编目过程（CATALOGED PROCEDURE）的起始标记。

（5）PEND 语句：标志一个流内过程的结束。

（6）Command 语句：操作员用这个语句在输入流中写入操作命令。

在这九种语句中，JOB、EXEC 和 DD 三种语句对于每个作业来说都是必要的。

下面给出一个单步作业的 JCL 实例：

```
//BACKUP JOB     ,'EXAMPLE JOB'
//*************************
//*    IT IS A EXAMPLE!     *
//*************************
//STEP1       EXEC    PGM=IEBGENER
//STEPLIB    DD       DSN=SYS1.LINKLIB,DISP=SHR
//SYSIN       DD       DUMMY
//SYSPRINT DD       SYSOUT=A
//SYSU1       DD       DSN=PR.MASTER,DISP=OLD
//SYSU2       DD       DSN=PR.MAILY.BACKUP,DISP=(NEW,CATLG),UNIT=TAPE,
//                     DCB=(RECFM=FB,LRECL=200,BLKSIZE=1000)
//
```

在上述例子中，我们给出了一个名为 BACKUP 的单步作业，在这个作业中我们通过调用实用程序 IEBGENER 完成了将 PR.MASTER 数据集备份到磁带上的工作。通过该例我们可以初步了解 JCL 中各语句的使用方式，下面我们将详细向大家介绍 JCL 的语法规则及语句的使用。

4.2.2 JCL 的语法规则

与其它计算机语言一样，JCL 有一套严格的语法规则。但与我们熟知的一些编程语言不同的是，JCL 还有其严格的语句格式规范。用户只有严格按照这些规则来编写作业控制程序，系统才能按照其意图正确完成用户的作业，否则系统就会给出错误信息，或产生不可预知的后果。

一、JCL 字符集

（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

使用十六进制值表示：X’40’ ~ X’FE’

在 JCL 语法中会用到一些特殊字符，其作用列表如下表 4-1：

表 4-1 JCL 中的特殊字符

字符	语法功能
,	分隔参数和子参数
=	分隔关键字参数(见 3.2.2 节)与它的值,例：CLASS=A
(b)	括起子参数列表或 PDS、PDSE 的成员名
&	标志一个符号参数(symbolic parameter)，例：&LIB
&&	标志一个临时数据集名（temporary data set name）例：&&TEMPDS 标志一个流内或系统输出（sysout）数据集名，例：&&PAYOUT
.	分隔受限数据集（qualified data set）名字的各部分，例：A.B.C 分隔一些特定参数与子参数的各部分，例：nodename.userid
*	提及一条先前的语句，例：OUTPUT=*.name ，或 在特定的语句内，标志特定的功能，例：//ddname DD *
‘	括起含有特殊字符的参数值
(空格)	划分域

二、一般语句格式规范

在 JCL 中，除/*语句外的所有语句均以第 1、2 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	名字区	操作符区	参数区	说明区
//	名字	操作符	参数	说明

- 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 1、2 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 3 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。这些决定该 JCL 语句如何被处理。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，它可以是任何所需的说明信息，注释区后必须跟一个空格。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，如果从 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**

三、参数规则

在 JCL 中，参数区内的参数的类型分为两类：

(1)位置参数（positional）：与其他参数保持相对位置的参数；

(2)关键字参数（keyword）：由一个关键字和等号后面的可变数据组成。

如果在一个语句内既有位置参数又有关键字参数时，所有的关键字参数必须位于位置参数之后。例：

//EXAMPLE　JOB　2000,CLASS=A

一个位置参数或关键字参数中的可变数字,也可能是一个子参数表。该表中同样可能含有位置和关键字这两种类型的参数，它们同样遵循位置参数和关键字参数的所有规则。当参数有子参数时，子参数必须顺序排列在圆括号括内。例：

//EXAMPLE　JOB　(2000,100,30),COND=(9,LT)

在了解参数类型的概念后，我们总结出参数的书写规则如下：

1．位置参数和关键字参数之间必须用逗号分开，不允许有空格。值得注意的是，在 JCL 语句中错写空格，经常导致非常难以查出的错误。

正确的	错误的
//EXAMPLE JOB 2000,CLASS=A	//EXAMPLE JOB 2000, CLASS=A
//EXP JOB (2000,9),CLASS=A	//EXP JOB (2000,9)CLASS=A

2．必须按规定的次序书写参数：所有的关键字参数必须位于位置参数之后，而所有位置参数也必须按规定排列。

正确的	错误的
//EXAMPLE JOB 2000,CLASS=A	//EXAMPLE JOB CLASS=A,2000

3．当缺省某个位置参数或某个子参数时，应以一个逗号指明所在位置。当缺省最后一个位置参数时，逗号可以省略。例如：

```
//EXP  JOB  (2000, ,9),CLASS=A
//SYSTEM  JOB  ,SYSTEM,CLASS=S,MSGLEVEL=(0,0)
```

4． 当没有任何位置参数时，则不必书写任何内容表示。例如：

```
//EXP  JOB  CLASS=A
```

5．关键字参数之间没有相对位置的规定，可以按任何次序排列。例如：

```
//EXP  JOB  2000,CLASS=A,MSGLEVEL=1
```

也可写作:

```
//EXP  JOB  2000, MSGLEVEL=1,CLASS=A
```

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非起某种特定的语法功能（见表 3.2.1)时，必须用撇号“’”替代括号将这些参数和子参数括起来，例：ACCT=’123+456’。而在这些参数与子参数中要用到撇号时，则需两个连续的撇号表示，例：O’NEIL 需写作’O’’NEIL’。有些语句中的某些参数或子参数含有一些特定的特殊字符时，将不需要用撇号括起来，详细的情况清参考表 3.2.2。在表 3.2.1 中我们可以知道，在 JCL 中用“&”来表识符号参数的开始。当参数中含有“&”且不用来表示符号参数时，则需使用连续的两个“&”来表示“&”。例：

```
//S1  EXEC  PGM=IEFBR14,ACCT=’&&ABC’
//DD1 DD    DSN=&&TEST,UNIT=SYSDA,SPACE=(TRK,(1,1))
```

MVS 系统中，系统将视连续的两个“&”为一个字符。所以建议用户将含有“&”的参数用撇号括起来以避免出错。

表 4-2 无需用撇号括起的特殊字符

语句、参数 或子参数	无需用撇号括起的 特殊字符	例子
JOB 语句中记账信息参数（accounting information）	连字符“-”	//JOBA JOB D58-D04

JOB 语句中程序员名参数（programmer’s-name）	连字符“-” 及 “.”（当“.”出现在字符前、字符间时无需撇号，但当其出现在字符串最后时则需用撇号）	//JOBB JOB ,S-M-T
		//JOBC JOB ,.ABC
		//JOBD JOB ,P.D.S
		//JOBE JOB ,’A.B.C.’

EXEC ACCT	连字符“-” 或 “+0”	//S1 EXEC … ACCT=D-L //S2 EXEC … ACCT=D+0
DD VOLUME=SER	连字符“-”	VOLUME=SER=PUB-RD
DD UNIT device-type	连字符“-”	UNIT=3330-1
DD DSNAME	连字符“-”	DSNAME=A-B-C
	分隔数据集名的“.”	DSNAME=A.B.C
	起语法功能的“&&” （见表 3.2.1）	DSNAME=&&TEMPDS DSNAME=&&PAYOUT
	“()”，其作用为表示（括起）： 1．PDS 及 PDSE 的成员名 2．索引顺序数据集（indexed sequential data set）的域名（area name） 3．PDS 及 PDSE 的生成数据集（generation data set）的代号（generation number） 4．生成数据集的代号	DSNAME=PDS1(MEMA) DSNAME=ISDS(PRIME) DSNAME=GDS(+1)
	标识生成数据集代号的加号“+”及减号“-”	DSNAME=GDS(-2)

7．JCL 的位置参数与关键字参数最多只能由两级子参数。也就是说用于括起子参数列表的括号最多只能有两层。

四、JCL 语句的位置

在下面各节中我们将详细讨论各语句的书写方法，为了便于编写 JCL，下面按照 JCL 语句的放置顺序来说明它们的位置：

1．JOB 语句。
2．JOBLIB 语句。
3．JOBCAT 及 SYSCHK 语句。
4．任何流内过程。
5．第一个 EXEC 语句
6．任何的 STEPCAT、STEPLIB，或一般的属于这一步的 DD 语句。
7．任何更多的 EXEC 语句及与他们相关联的 DD 语句。
8．任何空语句。

五、JCL 语法实例

```
作业语句        //EXPJOB  JOB  ,'USERNAME',MSGLEVEL=(1,1),        EXAMPLE
作业语句续行    //   MSGCLASS=Q,CLASS=A
注释语句        //**********************
                //*  IT IS A EXAMPLE  *
                //**********************
执行语句        //STEP1  EXEC  PGM=IEFBR14
DD 语句         //DD1  DD      DSN=MJSN.TEAM01.ONE,DISP=(,CATLG),
DD 语句续行     //   SPACE=(TRK,(5,2)), UNIT=SYSDA
DD 语句         // DD1  DD      DSN=MJSN.TEAM01.TWO,DISP=(,KEEP),
DD 语句续行     //   SPACE=(TRK,(1,1)), UNIT=SYSDA
```

值得注意的是：在本例中，采取了两种注释说明的方式，一种为作业语句中的“EXAMLE”，这是在说明区中说明的方式；另一种则是注释语句的方式。注释语句以第 1~3 列的“//*”开始，可以将它放在 JOB 语句后的任何 JCL 语句的前面或后面来说明 JCL。

4.2.3 JOB 语句

JOB 语句标志一个作业的开始、分配作业名并设置相关的位置参数及关键字参数，每个作业的第一个语句必须是 JOB 语句。

JOB 语句的格式如下：

//作业名 JOB 位置参数[，关键字参数][，关键字参数]。。。[注释说明]

一、作业名

作业名是用户给作业指定的名字。为使操作系统识别作业，必须选择确定的作业名字，由于系统不能同时运行具有相同名字得到作业，因此只能给作业一个唯一确定的名字。一般来说，建议用户采用“用户标识 USERID+数字或字符”的作业名，例如用户标识为 JACK，则作业名可用 JACKA。

二、位置参数

作业语句中的位置参数有两个：

1．记账信息（accounting information）：

记账信息位于操作符“JOB”后，它用于提供用户使用系统的合法性、机时及纸张的收费管理等。其格式为：

([account-number][,accounting-information]…)

account-number：用户账号；

accounting-information：附加的记账信息，如房间号、部门名等等。

记账信息参数及其子参数最多不可超过 143 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）。例：

```
//EXAMPLE1  JOB  (D548-8686,’12/8/98’,PGMBIN)
//EXAMPLE2  JOB  D548-8686
```

2．程序员名（programmer’s name）

程序员名用于标识作业的所有者（owner）信息，包括特殊字符在内，其长度不得超过 20 个字符。例：

```
//EXAMPLE1  JOB  2000,J.A.C.K
//EXAMPLE2  JOB  2001,JACK
//EXAMPLE3  JOB  2003,’O’’SUN’
```

下面是几个位置参数不同的书写格式的例子：

带有全部位置参数的作业语句：

```
//JOBA  JOB  (20008,60),A.B.C,CLASS=S,…
```

缺省记账信息的作业语句：

```
//JOBB  JOB  ,USER-NAME,CLASS=A,…
```

不带位置参数的作业语句：

```
//JOBC  JOB  CLASS=Q,…
```

二、关键字参数

JOB 语句中的关键字参数有如下几个：

1. ADDRSPC

指明作业所需之存储类型，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 表示作业请求实存存储空间。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 4-1），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**

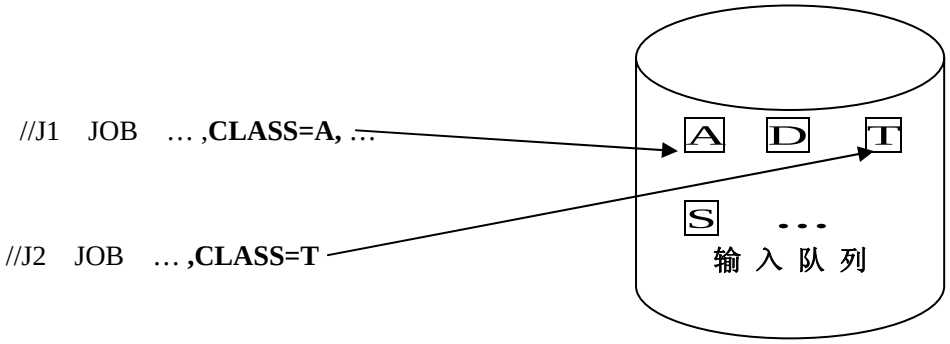

图 4-1 作业的输入队列

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。

4.2.4 EXEC 语句

EXEC 语句标明作业或过程中的每一作业步的开始，并告知系统如何执行该作业步。包括所有在 EXEC 语句中调用的过程中的所有作业步在内，一个作业最多可以有 **255** 个作业步。

EXEC 语句格式如下：

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

一、作业步名

作业步名是可以省略不写的，如需标明作业名时，该作业名必须在该作业内以及该作业调用的所有过程中是唯一的，它由 1~8 个字母或通配符开头的字符数字构成。

二、位置参数

EXEC 语句中的位置参数有两个：PGM 和 PROC。每条 EXEC 语句必须有且仅有一个位置参数或过程名。

1．　PGM

PGM 参数用于指明所要执行的程序名。该程序必须是一个分区数据集（PDS）的成员或用作系统库（system library）、私有库（private library）及临时库（temporary library）的扩充分区数据集（PDSE）的成员。程序名的调用方法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
```

在上例中，名为 STEP3 的 EXEC 语句采用程序间接调用方式，所调用的程序名由作业步 STEP2 中的名为 DDA 的 DD 语句决定，在该 DD 语句中定义了系统库 SYS1.LINKLIB，程序 P40 是该库的一个成员。“P40”即 STEP3 中要调用执行的程序名。关于 DD 语句的详细情况我们将在 4.2.5 中讨论。

2． PROC

指明作业步所要运行的过程名。

格式：

{PROC=procedure-name}

{procedure-name }

procedure-name：需要调用的过程名，过程名由 1~8 个字母或通配符开头的字符数字构成。所调用的过程名可以是：

• 编目过程的成员名或别名。

• 由 PROC 语句定义的流内过程的过程名，该流内过程必须在本作业内且本作业步前定义。

在设定该参数时，可直接写出过程名。

例：

```
//SP   EXEC   PROC=PAYWRKS
//BK   EXEC   OPERATE
```

三、关键字参数

EXEC 语句的关键字参数是可选的，这些参数制作用于本作业步。当 EXEC 语句的位置参数指定程序名时，关键字参数的写法同 JOB 语句；当 EXEC 语句的位置参数指定编目或流内过程时，EXEC 语句的关键字参数将覆盖所调用过程中各 EXEC 语句的关键字参数。因此如果想仅覆盖过程中的某个 EXEC 语句的关键字参数，则应在设置关键字参数时，同时指出所调用过程的相关过程步的名字。书写形式如下：

关键字参数.过程步名=值

下面将分别介绍EXEC语句中常用的关键字参数：

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。

4.2.5 DD 语句

数据定义语句（DD 语句）用于定义一个数据集以及该数据集所需的输入输出资源。DD 语句相对与前面介绍过的 JOB 语句和 EXEC 语句来说，其参数的定义、子参数的设置要复杂一些，在本小节内我们将仅讨论 DD 语句的一般规则以及部分位置参数，关于 DD 语句的一些常用参数以及特殊用法我们将用单独的一节讨论。

一、格式：

//[dd 名　　]　DD　[位置参数][,关键字参数]… [注释]

[过程步名.dd 名]

//[dd 名　　]　DD

[过程步名.dd 名]

二、dd 名

“dd 名”是为 DD 语句定义的名字，它由 1~8 个字母或通配符开头的字符数字构成。在一个作业步内可以有多个 DD 语句，但每个 DD 语句的 dd 名在本作业步中应该是唯一确定的。“dd 名”可以由系统定义也可以由用户自己定义，当用户需要调用公用程序时，需根据公用程序的具体要求选用系统定义的“dd 名”。用户自定义的“dd 名”不可与系统定义“dd 名”相重复。系统定义“dd”名有如下几个：

JOBCAT　　SYSCHK
JOBLIB　　SYSCKEOV
STEPCAT　　SYSIN
STEPLIB　　SYSMDUMP
SYSBEND　　SYSDUMP

JES2 子系统中：

JESJCLIN　　JESMSGLG
JESJCL　　JESYSMSG

JES3 子系统中：

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

用户子定义“dd 名”可以根据数据的用途，遵循“dd 名”的规则来命名，当为应用程序输入输出结果定义数据集时，“dd 名”的命名规则取决于程序所用语言的类型。汇编语言由 DCB 宏指令指定；COBOL 语言由 ASSIGN 子名指定；PL/1 语言由 DECLARE 语句指定；FORTRAN 语言由 READ 或 WRITE 语句中的通道号构成。

三、参数

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
```

```
          •
/*
```

例 2.

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

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。数据记录将以“/*”作为结束。

格式：

//dd 名　DD　DATA[,参数]… [注释]

例 1.

```
//GROUP1   DD   DATA
          •
          •
         data
          •
//GROUP2   DD   DATA
```

```
                •
                •
               data
                •
/*
```

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 BSAM（Basic Sequential Access Method）或 QSAM（Queued Sequential Access Method）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。

4.3 DD 语句的关键字参数

DD 语句的关键字参数及其相关内容相对 JOB 语句和 EXEC 语句来说比较复杂，所以在这里作为单独的一节讲述。DD 语句的关键字参数有很多，但总体上可分为两大类，一类与设备相关，另一类则与数据集或数据相关，与设备相关的参数有 UNIT、VOLUME、SPACE、LABEL 等，与数据集、数据相关的参数有 DSNAME、 DISP、 DCB、RECORG、EXPDT、RETPD、PROTECT、SYSOUT、HOLD 等。在实际应用中，这两类参数是配合使用的，没有一个绝对的分界线。DD 语句通过这些参数完成下述任务：

(1)定义顺序数据集（sequential data set）或分区数据集（partitioned data set）名；

(2)描述数据集状态、属性及保留期限；

(3)描述设备类型、数量；

(4)设置数据集的记录格式、占用空间；

(5)描述作业的处理方式。

DD 语句的关键字参数很多，在本节中我们仅讨论一些最常用的参数，并介绍一些基本概念及实例，以便于大家进一步的学习。

1. UNIT

UNIT 参数用于请求物理设备，用户通过设置设备地址或设备类型或设备组名等子参数确定设备；通过设置设备数或 P 等子参数确定设备数量。

格式：

```
{UNIT=([三位设备地址 ] [,设备数] [,DEFER])}
      [/三位设备地址] [,P     ]
      [/四位设备地址] [,      ]
      [设备类型     ]
      [设备组名     ]

{UNIT=AFF=DD 名}
```

设备地址：通过设备地址指定设备。设备地址是在系统安装时建立的，它由一个 3 位的十进制数或 4 位十六进制数构成。如用户请求的某设备其地址为 340 时，参数设置为 UNIT=340。

设备类型：通过设备类型名称指定设备，这个名称通常是数字的，如通过 3480、3422 指定磁带机，通过 3340、3375、3380、3390 指定磁盘机。如用户请求设备是 3380 磁盘机时，参数设置 UNIT=3380。

设备组名：通过设备组名请求一台或一组设备。被定义在一组中的设备可以是相同的，也可以可以是不同的。如一组设备中可以包含磁盘设备也可包含磁带设备。但通常都是将一类设备作为一个设备组，具体的设备组名在系统安装时定义。设备组名由 l—8 个字母符号

构成，常见的有 SYSDA、DASD、TAPE、CART 等。如需要直接访问的存储设备时，参数设置为：UNIT=DASD。

设备数：指定数据集所需的设备数量，取值范围为 1~59。

SER：SER 子参数的设置方式有两种：“SER=卷标号”和“SER=(卷标号[，卷标号]. . .)”，卷标号由 l~6 位的数字字母、通配符或特殊字符构成。用户通过 SER 定义数据集已占用或将占用的卷标号。在一个 DD 语句中最多可以有 255 个互不相同的卷标号。但需要注意的是：卷标号不能取作 SCRTCH、PRIVAT、 Lnnnnn(字母 L 带 5 个数)或 MIGAT。

REF：REF 子参数的设置方式有如下四种：“REF=数据集名”、“REF=*．DD 名”、“REF=*．作业步名.DD 名”及“REF= *．作业步名．过程步名. DD 名”。通过 REF 子参数可以从其它已知数据集或本语句前某个 DD 语句中获得所需的卷标号。

（1）“REF=数据集名”表示从其它已知数据集所在卷获得卷标号，定义中的数据集可以是编目数据集，也可以是由本语句前某个 DISP 参数传过来的数据集，但它不能是生成数据集(GDG)或 其成员。

（2）“REF=*.DD 名”表示从本作业步中的由“DD 名”指定的 DD 语句中获得卷标号。

（3）“REF=*.作业步名.DD 名”表示由指定的作业步中指定的 DD 语句获得所需卷标，其中作业步与 DD 语句分别由“作业步名”与“DD 名”指定。

（4）“REF=*.作业步名.过程步名.DD 名”表示从相关过程步中的相关 DD 语句中获得卷标，这个过程是由指定的作业步调用的。其中作业步、过程步、及 DD 语句分别由“作业步名”、“过程步名”及“DD 名”指定。

例 1.

```
//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
```

DD 语句 DDX 请求分配两个 3480 设备，DD 语句 DDZ 申请分配与 DDX 相同的两个设备。DD 语句 DDY 申请分配一个 3480 设备。

例 2.

```
//DD2    DD    DSNAME=X.Y.Z,DISP=OLD,UNIT=(,2)
```

本例中的 DD 语句定义了一个已编目的数据集，并且要求系统赋予两个设备给这个数据集，设备类型可以从相应的编目中获得。

例 3.

```
//DD3    DD    DSNAME=COLLECT,DISP=OLD,
//           VOLUME=SER=1095,UNIT=(3590,,DEFER)
```

在本例中定义了一个位于磁带卷上的已存在的数据集，并且请求系统分配一个 3590 磁带设备。由于指定了 DEFER 子参数，相应的磁带卷直到数据集被打开时才会装载。

例 4

```
//STEPA    DD    DSNAME=FALL,DISP=OLD,UNIT=237
```

对于这个数据集来说，系统将会从相应的编目中检索它的卷和设备类型。由于 UNIT 参

数被指定为设备 237，这将覆盖数据集在编目中的设备类型定义，因此要求设备 237 应该与编目中的定义相同。

2. VOLUME

通过 VOLUME 参数可以指定所引用的数据集所在的卷或卷组，也可以用来指定新建数据集所在的卷或卷组。在使用这个参数时，用户可以指定一个特定的卷、一组卷、具有特定序列号的卷或另外一个数据集所使用的卷。对于一个跨越多个卷的数据集来说，这个参数还可以用来指定首先被处理的卷。对于一个新建的数据集来说，可以通过不指定 VOLUME 参数或在 VOLUME 参数中不指定 SER 和 REF 子参数的方法在任何一个卷或卷组上创建该数据集，我们称这种方法为非特定卷。

格式:

```
{VOLUME} = ([PRIVATE] [,RETAIN] [,卷顺序号] [,卷数])
{VOL   }                [,      ] [,        ]

                [SER=序列号                              ]
                [SER=(序列号[,序列号]...)                ]
           [,]  [REF=数据集名                            ]
                [REF=*.DD 语句名                         ]
                [REF=*.作业步名.DD 语句名                ]
                [REF=*.作业步名.过程作业步名. DD 语句名]
```

PRIVATE：

申请一个私有的卷。这里的私有卷是指：

1. 除非使用 VOLUME=SER 子参数明确地请求这个卷，否则系统不会在这个卷上分配输出数据集。
2. 对于一个磁带卷来说,除非指定了 RETAIN 子参数或在 DISP 参数中指定 PASS，否则这个磁带卷将会在数据集关闭后被卸载。
3. 对于一个可卸载的直接访问卷来说，这个卷将在数据集关闭后被卸载。

RETAIN：

对于一个私有的磁带卷来说，指定 RETAIN 子参数表示在数据集关闭后或在作业步结束后，这个卷不会被卸载；对于一个公共的磁带卷来说，如果这个卷在作业中被卸载，它将保留在相应的设备上。

卷顺序号：

用来在一个多卷的数据集确定开始处理的卷。卷顺序号为 1~255 的十进制数，第一个卷的顺序号为 1，卷的顺序号必须小于等于数据集所占用的实际卷数，否在作业将会失败。如果不指定卷顺序号，则系统从 1 开始处理。

对于一个新数据集系统将忽略所指定的卷顺序号。

卷数：

用来确定一个输出数据集所申请的卷的最大数量。卷数为 1~255 的一个十进制数，在一个作业步中所有的 DD 语句中的卷数总和不能超过 4095。

SER=序列号

SER=(序列号[,序列号]...)

通过卷的序列号用来确定数据集占用或将占用那些卷。一个卷的序列号为 1~6 个字符可以包含字母、数字和$、#、@等特殊字符。不足 6 位的序列号将被空格填满。

在一条 DD 语句中最多可以指定 255 个卷序列号。不要在一个 SER 子参数中指定重复的序列号，无论是磁带卷还是磁盘卷，每个卷都应该有唯一的卷序列号。

不要将序列号指定为 SCRTCH、PRIVAT 或 Lnnnnn (L 后有五个数字)，这些名字已经被用在请求操作员装载卷的消息中；不要将序列号指定为 MIGRAT，这个名字被 DFHSM(Data Facility Hierarchical Storage Manager)用来做数据集的移植。

REF=数据集名

REF=*.DD 语句名

REF=*.作业步名.DD 语句名

REF=*.作业步名.过程作业步名.DD 语句名

用来表示系统将从其它的数据集或前面的 DD 语句中获得卷序列号的信息。

例 1

```
//DD1   DD   DSNAME=DATA3,UNIT=3340,DISP=OLD,
//           VOLUME=(PRIVATE,SER=548863)
```

在这个 DD 语句中指定了一个已存在的数据集，这个数据集位于一个直接访问的卷上，卷的序列号为 548863。由于指定了 PRIVATE，系统将不会将这个卷分配给另外一个申请非特定卷的数据集，在当前作业结束时系统将会释放这个卷。

例 2

```
//DD2   DD   DSNAME=QUET,DISP=(MOD,KEEP),UNIT=(3400-5,2),
//           VOLUME=( , , ,4,SER=(96341,96342))
```

这条 DD 语句中指定了一个已存在的数据集，这个数据集跨越两个卷，卷的序列号分别为 96341 和 96342。如果需要，可以在 VOLUME 参数中指定 4 个卷，当需要更多的空间时，系统会分配第三和第四个卷。

例 3

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

这个 DD 语句中定义了一个在作业步中创建并在同一作业步中被删除的数据集。通过不指定 VOLUME 参数表明在卷的分配上采用非特定卷的方式。

例 4

```
//DD4   DD   DSNAME=NEWDASD,DISP=(,CATLG,DELETE),UNIT=3390,
//           VOLUME=SER=335006,SPACE=(CYL,(10,5))
```

创建了一个新的数据集，这个数据集位于序列号为 335006 的卷上，这个卷是位于特定的 3390 设备上的永久卷。

例 5

```
//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)
```

在 DD 语句 OUTDD 中创建了一个多卷数据集并且对这个数据集进行编目，当然如果这个数据集不需要这么多卷的话，可以使用较少的卷。在 DD 语句 NEXT 中删除了这个数据集。

如果用户在多个卷上对数据集进行编目而实际上数据集仅使用了减少的卷的话，那么当系统删除这个数据集时下列信息将会被加入到作业日志中。

```
IEF285I    TEST.TWO                                    DELETED
IEF285I    VOL SER NOS=333001,333003.
IEF283I    TEST.TWO                                    NOT DELETED
IEF283I    VOL SER NOS=333002 1.
IEF283I    TEST.TWO                                     UNCATALOGED
IEF283I    VOL SER NOS=333001,333002,333003.
```

但如果数据集使用了所有分配给它的卷的话，当系统删除这个数据集时作业日志中将会包含下列信息。

```
IEF285I    TEST.TWO                                    DELETED
IEF285I    VOL SER NOS=333001,333002,333003.
```

例 6

```
//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
```

在作业步 STEP1 中的 DD 语句 DD1 标志了一个 SMS 数据集 OLD.SMS.DATASET，在作业步 STEP1 中的 DD 语句 DD2 和 STEP2 中的 DD 语句 DD3 分别创建了一个 SMS 数据集，数据集的属性引用在 DD1 中标志的数据集的属性。

3. SPACE

SPACE 参数用于为新建数据集分配磁盘空间，对于磁带卷不起作用。请求空间分配一般有两种方法：一是告知系统所需空间大小，由系统来分配合适的空间；二是请求系统分配某个特定的空间，如：从某个特定磁道到另一个特定磁道。

通常用第一种情况。用此种方法，用户告诉系统所要分配空间的存贮单位及存贮空间单位的数量。空间存贮单位可以是磁道(TRK)、柱面(CYL)、块长及记录长。不同类型的磁盘设备磁道、柱面容量也不同，所以为数据集分配空间时，要清楚用户所用的设备类型及磁道、柱面的容量。以 3380 为例，共有 885 个柱面，每个柱面有 15 个磁道，每个磁道的容量为 47476 字节。

格式：

由系统分配空间：

```
SPACE=({TRK,}(初次分配数量[,再次分配数量][,目录空间])[,RLSE][,CONTIG][,ROUND])
      ({CYL,}              [,             ][,索引    ] [,    ][,MXIG  ]
      ({块长度,  }                                        [,ALX   ]
      ({记录长度,}                                        [,      ]
```

请求特定的磁道：

```
SPACE= (ABSTR,(初次分配数量,地址[,目录空间])
                                [,索引    ]
```

仅请求目录空间：

SPACE=(,(,,目录空间))

由系统分配空间：

TRK：表示系统以磁道为单位分配空间。

CYL：表示系统以柱面为单位分配空间。

块长度：用来指定数据的平均块长度(字节)，块长度是 0~65535 的一个十进制数。这里指定的块长度用来作为空间分配的单位。(仅在 AVGREC 没有指定的情况下使用)

记录长度：在 SMS 环境下用来指定数据的平均记录长度(字节)，记录长度是 0~65535 的一个十进制数。这里指定的块长度用来作为空间分配的单位。(仅在指定 AVGREC 和 SMS 激活的情况下使用)

初次分配数量：初次为数据集分配的空间的大小，单位为磁道、柱面等。如果使用 TRK 或 CYL 作为单位为一个分区数据集分配空间，则初次分配的空间包含了目录空间；如果使用块长度或记录长度作为单位为一个分区数据集分配空间，则初次分配的空间不包含目录空间，系统另外分配目录空间。所要求的卷必须有足够的空间用于分配，否则作业将失败。

再次分配数量：当为数据集所分配的空间用完时，指定再次为数据集分配空间的数量。

目录空间：指定在一个分区数据集中用来作为目录的长度为 256 字节的记录的数量。

索引：对于一个索引顺序数据集的索引来说，用来指定所需的磁道或柱面，所需的磁道数应该等于一个或多个柱面。

RLSE：表示在数据集关闭时，那些分配给数据集但没有被使用的空间将会被释放。前提条件是数据集必须为了输出被打开并且最后一个操作为写操作。

CONTIG：指定分配给数据集的空间必须是连续的，这个子参数仅仅影响初次分配。

MXIG：要求为数据集分配的空间必须 1、是卷上最大的连续空间 2、大于或等于初次分配的空间大小。这个子参数仅仅影响初次分配。

ALX：作业在分配空间是将获得卷上最多 5 个最大的连续空间，并且每一个空间都应大于或等于初次分配的空间大小。这个子参数仅仅影响初次分配。

ROUND：只有在第一个子参数指定为块长度时表示分配的空间必须等于整数柱面，其它情况下忽略这个子参数。

申请特定的磁道：

ABSTR：表示将在卷上特定的位置为数据集分配空间。

初次分配数量：指定为数据集分配的磁道数，要求卷上必须有足够的空间。

地址：指定分配的第一个磁道的磁道号，第一个柱面上第一个磁道的磁道号为 0。

例 1

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

在这个 DD 语句中定义了一个临时数据集。UNIT 参数为数据集申请任何有效的磁带或直接访问设备卷，其中 MIXED 是一组磁带和直接访问设备的安装名。如果获得的是磁带卷的话，SPACE 参数被忽略；如果获得的是直接访问设备卷的话，SPACE 参数被用来为数据集分配空间。在本例中 SPACE 参数通过子参数指定了分配的单位和初次分配的数量：10 个柱面。

例 2

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

在 DD 语句中定义了一个新的分区数据集，系统将为这个数据集分配 10 个柱面，其中创建 10 个 256 字节的记录作为目录。由于指定了 CONTIG 子参数，系统将在卷上为数据集分配 10 个连续的柱面。

例 3

```
//REQUEST1 DD    DSNAME=EXM,DISP=NEW,UNIT=3330,VOLUME=SER=606674,
//              SPACE=(1024,75),DCB=KEYLEN=8
//REQUESTA DD    DSNAME=EXQ,DISP=NEW,UNIT=3380,
//              SPACE=(1024,75),DCB=KEYLEN=8
```

在本例的 DD 语句中根据块长分配空间。数据的平均块长为 1024 字节，需要申请 75 个数据块，每一个数据块前都需要有一个 8 个字节长的键，系统将会根据 UNIT 参数指定的设备计算需要多少个磁道。

例 4

```
//REQUEST2 DD    DSNAME=PET,DISP=NEW,UNIT=3330,VOLUME=SER=606674,
//              SPACE=(ABSTR,(5,1))
```

在本例中，SPACE 参数指定系统从卷上的第 2 个磁道起为数据集分配 5 个磁道。

例 5

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

这是一个在两个完整的卷上创建一个多卷数据集的例子，在这两个卷上不包含任何其它的数据集。一个 3350 设备上的卷包含 555 个柱面，未非配的柱面用来存放 VTOC。

4. DSNAME

DSNAME 参数被用来指定一个数据集的名字。对于一个新建的数据集来说 DSNAME 参数给定新数据集的名字；对于已存在的数据集来说，通过 DSNAME 参数来定位这个数据集。

格式：

```
{DSNAME} = 名字
{DSN    }
```

例 1.

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

在 DD 语句 DD1 中定义了一个名字为 ALPHA 的新数据集，随后的作业步或作业中的 DD 语句可以通过指定 DSNAME、UNIT 和 VOLUME 参数来引用这个数据集。

例 2

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

在 DD 语句 DDSMS1 中定义了一个名字为 ALPHA.PGM 的新 SMS 数据集，随后的作业步或作业中的 DD 语句可以通过指定 DSNAME 参数为 ALPHA.PGM 来引用这个 SMS 数据集。

例 3

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

DD 语句 DD2 中引用分区数据集 LIB1 中的数据集成员 PROG12。

例 4

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

在 DD 语句 DDIN 中指定 PAYIN1 作为系统为内部流数据集产生的数据集名的最后一个部分，这个数据集的名字将会是下面这种形式：

用户 ID.作业名.作业 ID.D 数据集号.PAYIN1

例 5

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

在 DD 语句 DDOUT 中指定 PAYOUT1 作为系统为系统输出数据集产生的数据集名的最后一个部分，这个数据集的名字将会是下面这种形式：

用户 ID.作业名.作业 ID.D 数据集号.PAYOUT1

例 6

```
//DD3     DD    DSNAME=&&WORK,UNIT=3420
```

在 DD 语句 DD3 中定义了一个临时数据集。一般来说由于临时数据集将在作业步结束时被删除，所有用户可以在 DD 语句中省略 DSNAME 参数。

例 7

```
//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)
```

在 STEP1 的 DD 语句 DD4 中定义了一个名为 ISDATA 的临时的索引顺序数据集，这条 DD 语句为这个索引顺序数据集定义了所有的区域。在 STEP2 中的 DD 语句 DD5 通过引用前面作业步中的 DD 语句的方式来引用这个数据集，因此这个临时数据集并不会在 STEP1 结束时被删除。

5. DISP

通过 DISP 参数可以向系统描述数据集的状态，并且可以设定系统在作业步或作业结束如何处理相应的数据集。你可以为作业或作业步的正常结束设定一个参数值同时为非正常结束也设定一个参数值。

.格式：

{DISP=状态}

{DISP=([状态][,正常结束参数][,非正常结束参数])}

```
DISP= ([NEW] [,DELETE ] [,DELETE ])
      [OLD] [,KEEP   ] [,KEEP   ]
      [SHR] [,PASS   ] [,CATLG  ]
      [MOD] [,CATLG  ] [,UNCATLG]
      [,  ] [,UNCATLG]
            [,       ]
```

状态子参数可以取下列值：

- NEW——表示在当前作业步中创建一个新的数据集。
- OLD——表示该数据集在当前作业步运行之前已经存在，并且当前作业步将以独占的方式使用这个数据集。
- SHR——表示该数据集在当前作业步运行之前已经存在，并且当前作业步将以共享的方式使用这个数据集，也就是说其它的作业也可以同时使用这个数据集。这个参数值也可以写成 SHARE。
- MOD——表示下列两种情况之一：1、数据集已经存在，记录将被添加到数据集的结尾，这个数据集必须是顺序的。2、一个新的数据集将被创建。在任何一种情况下数据集都将以独占的方式被使用。

正常结束参数可以取下列值：

- DELETE——表示在作业步正常结束后，该数据集将不再需要而被删除，所占用的空间将会被释放。
- KEEP——表示在作业步正常结束后，该数据集仍将继续保留在相应的卷上。
- PASS——表示该数据集将会被保留传递到同一作业的后续作业步中被使用。
- CATLG——在作业步正常结束后，系统将对数据集进行编目，在系统编目或用户编目中设置相应的入口指针指向该数据集。
- UNCATLG——在作业步正常结束后，系统解除对数据集的编目，在系统编目或用户编目中删除相应的入口指针和索引。

非正常结束参数可以取的值基本与正常结束参数可以取的值相同，但不能够取 PASS。这里就不再赘述了。

例 1.

```
//DD2   DD   DSNAME=FIX,UNIT=3420-1,VOLUME=SER=44889,
//             DISP=(OLD,,DELETE)
```

在本例的 DD 语句中定义了一个已存在的数据集，通过缺省的第二个子参数指定当作业步正常结束时数据集将会被保留，通过指定第三个子参数为 DELETE 指定当作业步非正常结束时系统将会删除这个数据集。

例 2

```
//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)
```

DD 语句 DD1 定义了一个新的数据集并指定这个数据集将被传递给后续的作业步。如果作业步 STEPA 非正常结束，这个新创建的数据集将会被删除。

在作业步 STEPB 中的 DD 语句 DD3 将会接收从上一个作业步中传递过来的数据集并将这个数据集传递给后续的作业步。如果 STEPB 非正常结束，根据 DISP 的第三个子参数这个数据集也将会被删除。

在作业步 STEPC 中的 DD 语句 DD4 将会接收从上一个作业步中传递过来的数据集并在作业步正常结束后将数据集编目，同样如果 STEPC 非正常结束则数据集被删除。

在 DD 语句 DD2 中指定了一个已存在的数据集 XTRA，根据 DISP 参数可以知道无论 STEPB 正常结束与否该数据集都将会被保存。

6. DCB

使用 DCB 参数可以完善数据集的数据控制块(DCB)中的信息。

格式：

```
[ DCB=(子参数[,子参数]...)]
```

**DCB 的子参数较多，这里不再一一赘述，用户可以参考实例进行初步的了解。如果需要获得详细的解释可以查阅相应的手册。*

```
[ DCB= ( {数据集名}[,子参数]...)]
[       ( {*.DD 语句名})]
[       ( {*.作业步名.DD 语句名})]
[       ( {*.作业步名.过程作业步名.DD 语句名})]
```

例 1

```
//DD1     DD      DSNAME=ALP,DISP=(,KEEP),VOLUME=SER=44321,
//                UNIT=3400-6,DCB=(RECFM=FB,LRECL=240,BLKSIZE=960,
//                DEN=1,TRTCH=C)
```

DD 语句 DD1 中定义了一个名为 ALP 的新的数据集。在 DCB 参数中包含了用以完成数据控制块的必要信息。

例 2

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

DD 语句 DD1A 在 3380 设备上定义了一个名为 EVER 的新的数据集。在 DCB 参数中包含了用以完成数据控制块的必要信息。

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

DD 语句 DD1B 与 DD1A 具有相同的功能，但 DD1B 使用了另一种语法格式。由于没有指定 BLKSIZE，系统将会为数据选择一个最合适的块的大小。

例 3

```
//DD2     DD      DSNAME=BAL,DISP=OLD,DCB=(RECFM=F,LRECL=80,
//                BLKSIZE=80)
```

```
//DD3     DD      DSNAME=CNANN,DISP=(,CATLG,DELETE),UNIT=3400-6,
//                LABEL=(,NL),VOLUME=SER=663488,DCB=*.DD2
```

DD 语句 DD3 中定义了一个名为 CNANN 的新的数据集，并且要求系统参照统一作业步中另外一个 DD 语句中的 DCB 参数来确定本语句中的 DCB 参数的值。

例 4

```
//DD4     DD      DSNAME=JST,DISP=(NEW,KEEP),UNIT=SYSDA,
//                SPACE=(CYL,(12,2)),DCB=(A.B.C,KEYLEN=8)
```

DD 语句 DD4 中定义了一个名为 JST 的新的数据集，并且要求系统参照一个已编目的数据集 A.B.C 的 DCB 信息来确定本语句中的 DCB 参数的值，通过指定子参数 KEYLEN 来将相应的定义覆盖。

7. SYSOUT

通过 SYSOUT 参数可以将相应的数据集标志为一个系统输出数据集。同时 SYSOUT 参数还可以完成以下的定义：

- 将这个系统输出数据集与一个输出类关联起来。
- 不通过 JES 而是要求一个外部的书写器程序来处理这个系统输出数据集。
- 指定这个数据集被打印输出的格式。
- 引用 JES2 的/*OUTPUT 语句。

系统输出数据集根据下面的输出定义顺序被处理：

- 在 SYSOUT DD 语句中指定的选项。
- 参考 JCL 的 OUTPUT 语句中指定的选项。
- 参考 JES2 的/*OUTPUT 语句中指定的选项或 JES3 的//*FORMAT 语句中指定的选项。
- 相关的输出类的缺省值。

格式：

```
SYSOUT= { 输出类                              }
        { *                                   }
        { ([输出类] [,书写器名] [,格式名]) }
                  [,INTRDR  ] [,代码名]

SYSOUT=(,)
```

输出类：

为数据集指定的输出类，输出类为一个字符：A~Z 或 0~9。每一个输出类的属性在 JES 初始化时被定义。在 JES2 中输出类的缺省值为 A。

*——表示输出类与在 JOB 语句中 MSGCLASS 参数的定义相同。

(,)——指定输出类为空值。当引用 JCL 的 OUTPUT 语句中 CLASS 参数的定义时必须指定输出类为空值。

书写器名：

确定一个系统书写器程序的名字(1~8 个字符)。一个外部书写器程序是系统中一个用来处理输出的已启动的任务，每一个外部书写器程序有一个用户标识符与其相关联。通过在 DD 语句中指定外部书写器的名字来使用该书写器程序处理输出，例如：

```
//MYOUTPUT    DD SYSOUT=(A,XTWTR)
```

不要将 STDWTR 作为一个书写器名，因为 STDWTR 是 JES 的一个保留字。出于同样的原因在 JES3 系统中不要使用 NJERDR 作为书写起名。

INTRDR：

通知 JES 将这个系统输出数据集作为输入作业流送到内部读卡机。

格式名：

确定打印输出的格式。格式名为 1~4 个字符，可以为字母、数字或特殊字符($、#、@)。

代码名：

用来确定 JES2 获得处理属性的 JES2 /*OUTPUT 语句，代码名必须与 JES2 的/*OUTPUT 语句中的 CODE 参数相同。代码名仅仅被 JES2 系统支持，当作业或作业步中包含了一个缺省的 JCL OUTPUT 语句时不要使用代码名。

例 1.

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

在本例中，通过 DD 语句指定 JES 将系统输出数据集写到处理 P 类输出的设备上。

例 2.

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

在本例的 DD 语句中定义 PAYOUT1 作为系统为系统输出数据集产生的名字的最后一个部分，这个数据集名类似于以下格式：用户 ID.作业名.作业 ID.D 数据集号.PAYOUT1。在 DD 语句中指定将系统输出数据集写到处理 P 类输出的设备上。

例 3.

```
//JOB50   JOB      ,'C. BROWN',MSGCLASS=C
//STEP1   EXEC     PGM=SET
//DDX      DD       SYSOUT=C
```

在本例的 DD 语句中指定将系统输出数据集写到处理 C 类输出的设备上。由于 SYSOUT 和 MSGCLASS 参数指定了同一个类，所以作业的消息和系统输出数据集将会被写到同一个设备上。

例 4.

```
//STEP1   EXEC     PGM=ANS
//OT1      OUTPUT   DEST=NYC
//OT2      OUTPUT   DEST=LAX
//OT3      OUTPUT   COPIES=5
//DSA      DD       SYSOUT=H,OUTPUT=(*.OT2,*.OT1,*.OT3)
```

本例中的 DD 语句通过结合三个 OUTPUT 语句将输出结果分成了三个部分：

1. DSA 与 OT1 相结合将系统输出数据集送到 NYC。
2. DSA 与 OT2 相结合将系统输出数据集送到 LAX。
3. DSA 与 OT3 相结合在处理 H 类输出的设备上将数据集打印五份。

例 5.

```
//DD5      DD       SYSOUT=(F, ,2PRT)
```

在本例的 DD 语句中指定 JES 将系统输出数据集写到处理 F 类输出的设备上，数据集将会按照名为 2PRT 的输出格式被打印。

4.4 特殊的 DD 语句

在 JCL 中有一些特殊的 DD 语句，通过这些语句用户可以完成指定私有编目、私有库、用于转存和检查点的数据集等特殊的功能。

1. JOBCAT

通过 DD 语句 JOBCAT 可以为作业定义一个私有的 VSAM 用户编目或完整的编目功能。系统可以在搜索主编目或搜索与数据集名的第一部分相关联的私有编目前先搜索本语句中定义的私有编目。

当作业中引用了一个 SMS 数据集时不要使用 JOBCAT 语句，因为 SMS 仅仅访问那些在系统编目中进行编目的 SMS 数据集。

格式：

```
//JOBCAT DD DISP={OLD},DSNAME=私有编目名[,参数]... [说明]
                 {SHR}
```

不要指定任何 UNIT 和 VOLUME 参数，系统将会从主编目中获取私有编目的位置。

可以通过在 JOBCAT 语句后立即跟有省略了语句名的 DD 语句的方法为作业指定多个这样的私有编目。

应当将 JOBCAT 语句放置在 JOB 语句之后，并且位于第一个 EXEC 语句之前。

如果作业中包含了 JOBLIB 语句，应当放置在 JOBCAT 语句之前。

例 1

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

在这个例子中，JOBCAT 语句指定了一个私有编目 LYLE，并且 JOBCAT 语句位于 JOBLIB 语句之后。

2. JOBLIB

通过 JOBLIB DD 语句用户可以创建一个私有库或为作业指定一个私有库。系统将会首先搜索所指定的私有库去查找那些在 EXEC 语句的 PGM 参数中使用的程序，只有在私有库中没发现相匹配的程序时系统才会去搜索系统库。

一个私有库实际上是一个位于一个直接访问设备上的分区数据集(PDS)或分区数据集扩展(PDSE)，其中的每一个成员都是一个用户的可执行程序。

格式：

```
//JOBLIB DD 参数[,参数]... [参数]
```

定义已编目的库：

- 指定 DSNAME 参数。
- 指定 DISP 参数，其中的状态子参数必须为 OLD 或 SHR。
- 不需要指定 VOLUME 或 UNIT 参数。

定义未编目的库：

- 指定 DSNAME 参数。

- 指定 DISP 参数，参数值必须为 DISP=(OLD,PASS)或 DISP=(SHR,PASS)。其中 SHR 表示这个数据集是已经存在的并允许其它作业使用这个库。
- 指定 UNIT 参数。
- 指定 VOLUME 参数。

创建一个库：

- 指定 DSNAME 参数，作为库的名字。
- 指定 UNIT 参数，注意一个库必须建立在一个直接访问设备上。
- 指定 VOLUME 参数，非特定卷的情况例外。
- 指定 SPACE 参数，为整个库分配足够的空间，并为 PDS 的目录分配空间。
- 指定 DISP 参数，其中的状态子参数必须为 NEW。

向库中添加成员：

- 在 DSNAME 参数包含相应的成员名，例如，DSNAME=LIBRARY(PROGRAM)
- 将 DISP 参数中的状态子参数指定为 MOD。如果在创建库时已经编目则不需要其它子参数，否则指定为 PASS 或 CATLG。
- 不要指定 SPACE 参数

其它参数：

如果在数据集标签中不包含数据控制块信息则需要指定相应的 DCB 参数，但不要指定 FREE=CLOSE。

例 1.

```
//PAYROLL JOB  JONES,CLASS=C
//JOBLIB  DD   DSNAME=PRIVATE.LIB4,DISP=(OLD,PASS)
//STEP1   EXEC PGM=SCAN
//STEP2   EXEC PGM=UPDATE
//DD1     DD   DSNAME=*.JOBLIB,DISP=(OLD,PASS)
```

在本例中 JOBLIB DD 语句中所指定的私有库已经被编目，所以无需指定 UNIT 和 VOLUME 参数。系统首先搜索私有库 PRIVATE.LIB4 去查找程序 SCAN 和 UPDATE，其次才查找系统库 SYS1.LINKLIB。在 DD1 语句中引用了 JOBLIB DD 语句中指定的私有库。

例 2.

```
//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
//DD1     DD   DSNAME=*.JOBLIB,VOLUME=REF=*.JOBLIB,DISP=(OLD,PASS)
```

因为在本例中 JOBLIB DD 语句中所指定的私有库没有被编目，所以必须指定 UNIT 和 VOLUME 参数。系统首先搜索私有库 PRIV.DEPT58 去查找程序 DAY 和 BENEFITS，其次才查找系统库 SYS1.LINKLIB。在 DD1 语句中引用了 JOBLIB DD 语句中指定的私有库。

例 3.

```
//TYPE    JOB  MSGLEVEL=(1,1)
//JOBLIB DD    DSNAME=GROUP8.LEVEL5,DISP=(NEW,CATLG),
//             UNIT=3350,VOLUME=SER=148562,SPACE=(CYL,(50,3,4))
```

```
//STEP1  EXEC PGM=DISC
//DDA    DD   DSNAME=GROUP8.LEVEL5(RATE),DISP=MOD,
//           VOLUME=REF=*.JOBLIB
//STEP2  EXEC PGM=RATE
```

由于在本例的 JOBLIB DD 语句中指定的私有库还不存在，因此必须在 JOBLIB DD 语句中包含所有用于定义这个库的必要的参数。这个库将在 STEP1 中被建立，并且由 DD 语句 DDA 为这个库添加了一个新的成员 RATE。明显可知，系统将在系统库 SYS1.LINKLIB 中查找程序 DISC，在作业步 STEP2 中系统将首先在新创建的私有库中查找程序 RATE。

例 4.

```
//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
```

通过三个 DD 语句为作业定义了三个相连接的私有库，系统将按照下面的顺序查找每一个程序：

KRG.LIB12
GROUP31.TEST
PGMSLIB
SYS1.LINKLIB

3．STEPCAT

通过 DD 语句 JOBCAT 可以为作业步定义一个私有的 VSAM 用户编目或完整的编目功能。系统可以在搜索主编目或搜索与数据集名的第一部分相关联的私有编目前先搜索本语句中定义的私有编目。

当作业步中引用了一个 SMS 数据集时不要使用 JOBCAT 语句，因为 SMS 仅仅访问那些在系统编目中进行编目的 SMS 数据集。

格式：

```
//STEPCAT DD DISP={OLD},DSNAME=私有编目名[,参数]... [说明]
                 {SHR}
```

不要指定任何 UNIT 和 VOLUME 参数，系统将会从主编目中获取私有编目的位置。

可以通过在 STEPCAT 语句后立即跟有省略了语句名的 DD 语句的方法为作业指定多个这样的私有编目。

通过下面的语句可以在一个特定的作业步中用主编目覆盖 JOBCAT 中定义的私有编目：
//STEPCAT DD DISP=OLD,DSNAME=主编目名。

在一个作业步中可以将 STEPCAT 语句放在 DD 语句中的任何一个位置。

例 1.

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

STEPCAT 语句为这个作业步定义了一个私有编目 BETTGER。

4. STEPLIB

STEPLIB 语句的作用与 JOBLIB 相似，主要区别在于作用的范围分别是作业步和作业。这里我们不再对 STEPLIB 进行详细的说明，仅仅给出一些例子共大家参考。同一个作业中后续作业步可以引用在 STEPLIB DD 语句中定义的私有库，同样，可以将一个 STEPLIB DD 语句放在内部流或编目过程中，但不能将 JOBLIB DD 语句放在内部流或编目过程中。

.格式：

//STEPLIB DD 参数[,参数]... [说明]

例 1

```
//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)
```

在本例中系统首先在私有库 PRIV.LIB5 中搜索程序 SPKCH，在私有库 PRIV.LIB12 中搜索程序 TIL80。

例 2

```
//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=3350,VOLUME=SER=55566
//STEP3   EXEC PGM=A1530
//STEP4   EXEC PGM=SNAP11
//STEPLIB DD   DSNAME=*.STEP2.STEPLIB,
//             DISP=(OLD,KEEP)
```

系统首先在私有库 LIBPARYP 中搜索程序 SNAP10；库 LIBRARYP 将被传递到后续的作业步。在作业步 STEP4 中的 STEPLIB DD 语句引用了 LIBRARYP 库，因此系统将会在 LIBRARYP 中搜索程序 SNAP11。由于在这个作业中包含了 JOBLIB DD 语句，系统将会首先在库 LIB5.GROUP4 中搜索程序 SNZ12 和 A1530。

例 3

```
//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
```

对于作业步 STEP2 和 STEP4 来说系统首先会在私有库 LIB5.GROUP4 中搜索程序 VARY 和 SHORE。但对于 STEP1 来说，系统或首先搜索库 SYS1.LINKLIB 来查找程序 SUM。

在 STEP3 中定义了一组私有库，系统按照下面的顺序搜索程序 CALC：PRIV.WORK、LIBRARYA、LIB.DEPT88、SYS1.LINKLIB。如果有后续的作业步引用在 STEP3 中定义的 STEPLIB 语句，所引用的仅仅是库 PRIV.WORK，其它的库将不会被引用。

5. SYSIN

通常，我们使用 SYSIN DD 语句作为一个内部流数据集的开始。内部流数据集以 DD * 或 DD DATA 语句开头，这样的 DD 语句可以有任何一个有效的名字，包括 SYSIN。如果在内部流数据之前省略这样的 DD 语句，系统会自动提供一个名为 SYSIN 的 DD *语句。

格式：

//SYSIN DD 参数[,参数]... ［说明］

第一个参数为*或 DATA，用以指出后面紧跟的时内部流数据。这条语句必须且只能够位于内部流数据前。

例 1

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

4.5 过程

和其它高级语言一样，作业控制语言中也允许定义过程。过程是一段预先编写好的 JCL 语句的集合，它可以被反复调用。作业控制语言中定义了两种过程，分别是编目过程（cataloged procedure）和流内过程（in-stream procedure）。

4.5.1 编目过程与流内过程

在指定的过程库中编目的过程称为编目过程。该过程库可以是分区数据集或扩展分区数据集,通常系统过程被编目在系统过程库中，用户过程被编目在用户库中。由于调用编目过程时，系统提供的是该过程的拷贝，因此一个编目过程可以同时被几个作业调用。

流内过程时放置在作业输入流中的过程。在一个作业中最多可以有 15 个流内过程，但

不能嵌套使用，也不能被其他作业调用。流内过程与编目过程的区别是流内过程随着一个作业放在输入流中，它紧跟在 JOB 语句后面而不是作为分区数据集的成员。在实际应用时，流内过程只用于测试阶段，一旦流内过程调试成功，即可对其进行编目，时期策划能够为编目过程。

4.5.2 过程的编写

一个过程可由几个作业步组成。在过程中可以包含除下列语句外的所有 JCL 语句：

- 调用过程的 EXEC 语句（一个过程不能调用另一个过程）
- JOB 语句、/*语句或//语句
- JOBLIB DD 语句或 JOBCAT DD 语句
- 任何的 JES 控制语句
- DD *语句或 DD DATA 语句

流内过程的开始和结束分别用 PROC 语句和 PEND 语句表示，对于编目过程，不能有 PEND 语句，而如果没有分配给符号参数默认值，PROC 语句是可选的。过程的结构如下：

```
//过程名    PROC   [符号参数]
//过程步 1   EXEC
//dd 名 1    DD
          •
          •
          •
//过程步 2   EXEC
//dd 名 2    DD
          •
          •
          •
//          PEND        （仅在流内过程中使用）
```

过程结构中的过程名、过程步名及 dd 名的书写规则与 JCL 中其它语句名的书写规则一样。 符号参数的功能类似于其它编程语言子程序中的形式参数。

例：流内过程

```
//MYJOB   JOB   （2216,82）,'TEST RUN',CLASS=A
//RUN     PROC                          ┐
//GO      EXEC   PGM=ONE                │
//SYSOUT  DD     SYSOUT=A               ├  流内过程
//          PEND                        ┘
//STEP1   EXEC   RUN
```

本例中，RUN 是一个流内过程，作业中名为 STEP1 的执行语句调用这个流内过程。该流内过程在去掉 PEND 语句后也可以作为标准的过程放在用户库中，作为编目过程。

4.5.3 过程的调用

过程的调用有如下两种方式：

```
//作业步名   EXEC   PROC=过程名   [符号参数]
```

或 //作业步名　EXEC　过程名　　[符号参数]

当调用一个过程时，系统会以输入流、用户库、系统库的顺序来检索所要调用的过程。如果所调用的过程是流内过程，则必须把流内过程放在调用它的 EXEC 语句之前。

如果调用的编目过程被编目在用户库中，系统从 JCLLIB 语句确定的用户库中进行检索。因此，若调用的过程是用户库中的过程时，要用 JCLLIB 语句来指明过程所在的用户库。如果调用的编目过程被编目在系统库中，系统从 JES2 中的 PROCLIB 参数指定的系统库进行检索。

4.5.4 过程的修改

由于不同用户的要求不同，所以当某一个作业调用标准过程时，系统应允许用户对过程进行修改，以满足自己的需要。过程修改的方式有如下三种：

（1）置换过程中的符号参数；

（2）对过程中的 EXEC 及 DD 语句参数进行覆盖和增加；

（3）增加新的 DD 语句。

1．　符号参数

符号参数由符号“&”和参数名组成，参数名可以是以字母或通配符开头的 1~8 位字母数字或通配符，关键字参数和关键字子参数不能作为符号参数的参数名。符号参数为修改过程提供了可选用的方法。

当过程中含有符号参数时，每一个符号参数必须指定一个值或赋空值，该值被称为符号参数的初值。

下面是一个使用符号参数的例子：

例 1：//RUN　PROC　PROGRAM=ONE,UNIT=SYSDA
　　//GO　EXEC　PGM=&PROGRAM
　　//A　DD　UNIT=&UNIT,SPACE=(TRK,20)

在本例中符号参数 PROGRAM、UNIT 都被赋了初值。当一个作业调用该过程时，这些符号参数的初值可以被调用该过程的 EXEC 语句中的符号参数值所取代，从而达到修改过程的目的。

下面是调用例 1 过程的例子：

例 2：//JOB1　JOB　…
　　//GO　EXEC　RUN,UNIT=TEMP
　　//　　PROGRAM=TWO

2．　EXEC 语句参数的覆盖和增加

用 EXEC 语句调用过程时，该语句的所有关键字参数都会影响过程的执行，它将覆盖过程中定义的参数，对于过程中没有定义的参数，系统会把它加到过程中去。用 EXEC 语句修改过程的方式如下：

//stepname　EXEC　过程名，参数.过程步=值

其中“参数.过程步=值”表示准备对过程中所希望的过程步的关键字参数进行修改。

例如有下述过程：

```
//RUN　PROC
//STEP1　EXEC　PGM=P1
　　•
```

```
        •
//STEP2    EXEC   PGM=P2
        •
        •
//STEP3   EXEC   PGM=P3,TIME=(2,30)
```

若要求对 STEP2 增加 COND 参数并指定条件测试为（8,GT），对 STEP3 改变其时间限制为 4 秒。那么调用该过程时，EXEC 语句为：

```
//GO   EXEC   RUN,
//        COND.STEP2=(8,GT),
//        TIME.STEP3=4
```

3. DD 语句参数的覆盖

对于过程中 DD 语句参数的修改可以通过下列方式：

//过程步名.DD 名　DD　…

“过程步名.DD 名”用来确定要修改的 DD 语句，其中“DD 名”指需要修改的 DD 语句，“过程步名”指过程重要修改的 DD 语句所在的过程步的名字。下面是对过程中 DD 语句进行修改的例子：

```
例：//RUN   PROC
        •••
//S1     EXEC …
        •••
//SYSUT2   DD   SYSOUT=*
        •••
//          PEND
        •••
//TEST2   EXEC   RUN,…
        •••
//S1.SYSUT2   DD   SYSOUT=S
```

4. 增加新的 DD 语句

对于调用的过程，可能不包含用户需要的 DD 语句，这是可通过下列方法来增加新的 DD 语句：

//过程步名.需增加的 DD 语句

其中过程步名是确定新加的 DD 语句在过程中的位置。

4.6 实用程序

4.6.1 实用程序的分类

在 z/OS 系统中，IBM 提供了种类繁多且十分有用的实用程序，来辅助用户对数据进行组织与维护。

实用程序分为三类：系统实用程序、数据集实用程序和独立实用程序。

系统实用程序通常以 IEH 打头，它的主要功能是维护和管理系统、用户数据集合整个

盘卷。系统实用程序及其功能如下：

IEHNITT：为磁带卷写标号。

IEHLIST：系统控制数据信息列表。

IEHMOVE：移动或拷贝若干组数据、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、建立时代数据组索引、重命名磁带卷、删除数据集等。

IEHDASDR：初始化一个直接存取卷。

数据集实用程序通常以 IEB 打头。它的主要功能是对数据集或数据集纪录进行组织、修改或比较。它可以作为单个作业来执行，也可以作为某个程序的子程序被调用。需要注意的是这些数据维护实用程序不能用于 VSAM 数据集。以下是数据集实用程序及其功能：

IEBCOMPR：比较顺序数据集、分区数据集或扩展分区数据集。

IEBCOPY：拷贝、压缩或合并分区数据集及扩展分区数据集。

IEBDG：创建含有模型数据的测试数据集。

IEBEDIT：有选择的拷贝作业步及其相关的作业语句。

IEBGENER：拷贝顺序数据集记录或将顺序数据集转换为分区数据集。

IEBIMAGE：修改、打印或连接模块。

IEBISAM：卸载、装载、拷贝或打印 ISAM 数据集。

IEBPTPCH：打印或穿卡输出一个数据集。

IEBUPDATE：对顺序、分区数据集或扩展分区数据集进行合并修改。

独立实用程序通常以 IBC 开头。它是一种特殊的实用程序，可独立于操作系统运行，通常被存放在磁带上。当系统出现重大故障而又无法恢复时，利用系统转储磁带，恢复系统盘卷。如：

IBCDASDI：用于初始化和分配一个直接存取卷上的可用道的实用程序。

IBCDUMPRS：是转储或再存储直接存取卷数据的实用程序。

4.6.2 实用程序的调用

实用程序的调用方法有两种，一种是在 ISPF 下用 TSO 的 CALL 命令调用，另一种是通过 JCL 语句调用，在本章中我们只讨论后一种调用方式。其格式为：

（1）//UTLFM JOB …
//STEP EXEC PGM=utility 调用公用程序
//SYSPRINT DD … 系统输出数据集（SYSOUT）
//SYSUT1 DD … 输入数据集
//SYSUT2 DD … 输出数据集
//SYSIN DD … 定义实用程序使用的控制数据

（2）//UTLFM JOB …
//STEP EXEC PGM=utility 调用实用程序
//SYSPRINT DD … 系统输出数据集（SYSOUT）
//ddname DD UNIT=…,VOL=…,DISP=OLD 存取所需数据集的有关信息
//ddname DD UNIT=…,VOL=…,DISP=OLD 所存储数据集的有关信息
//SYSIN DD … 定义实用程序实用的控制数据

4.6.3 控制语句的标准格式

标号	操作符	操作数	注释

16 **72**

标号用于表示控制语句,除实用程序 IEHNITT 外，其它实用程序都可以省略标号。标号必须放在控制语句开始的位置，后面留有一个以上的空格。标号是由 1~8 个字母或数字字符组成。

操作符用于标示控制语句的类型，其后至少跟有一个空格。操作数是由一个或多个关键字参数组成，参数之间以逗号相隔。操作数后至少跟一个空格。控制语句内可加注释，但它与操作数之间至少要有一个空格。

当控制语句有续行时，可在本行有逗号的地方断开，或在本行第 72 列处设置一个字符，或在下一行的 16 列开始。

4.6.4 常用实用程序简介

1. IEBCOMPR

IEBCOMPR 程序用于在两个数据集的逻辑记录间进行比较，这两个数据集可以是顺序数据集、分区数据集或扩展分区数据集。它能对数据集或数据集成员的定长、变长、组块、非组块或未定义记录进行比较。但它不能对加载模块进行比较。

两个顺序数据集比较相同，是指它们含有相同数量的记录且相关记录和关键字完全相同。而两个分区数据集或两个扩展分区数据集比较相同，则是指：

（1）相关成员含有相同的记录；

（2）注释列表在相关成员的位置相同；

（3）相关记录和关键字完全相同；

（4）相关目录和用户数据区完全相同。

对于相同的数据集必须同时满足这些条件，否则不能视其为相同数据集。

需要注意的是，对于分区数据集合扩展分区数据集，只有其中一个数据集的所有目录项名字在另一个数据集的目录中都能找到相同的目录项名时，才能进行比较。否则是不能比较的。

图 4-2 是一个可以比较的例子，而图 4-3 是一个不可比较的例子。

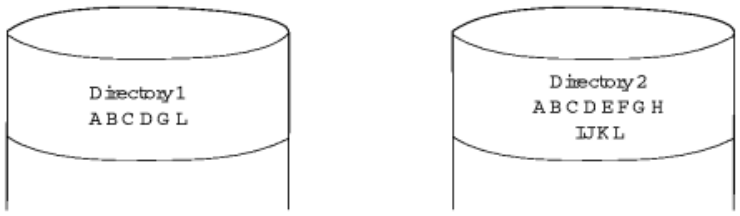

图 4-2 可以比较的 PDS

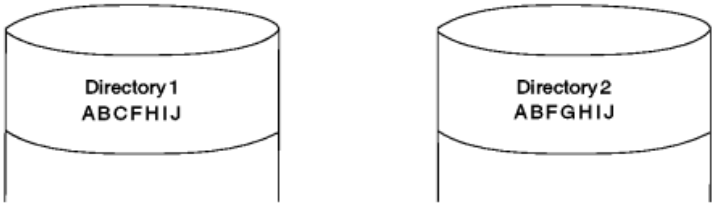

图 4-3 不可以比较的 PDS

下面是关于 IEBCOMPR 程序语句的列表：

语　　句	功　　能
JOB	作业开始
EXEC	定义程序名 PGM=IEBCOMPR
SYSPRINT　DD	指定系统输出数据集
SYSUT1　DD	定义需要比较的数据集
SYSUT2　DD	定义需要比较的数据集
SYSIN　DD	定义控制数据集或 DUMMY，控制语句可以是 COMPARE、EXITS、LABELS

控制语句说明：

COMPARE：定义数据集的组织结构，在 SYSIN　DD 中设置控制语句时，它必须是第一个控制语句，当输入数据集是分区数据集或扩展分区数据集时，必须设置这个语句。语句格式如下：

label　COMPARE　TYPROG={PS/PO}

其中 TYPROG={PS/PO}用于指定输入数据集的组织结构，PS 表示输入数据集为顺序数据集，为缺省值；PO 表示输入数据集是分区数据集或扩展分区数据集。

EXITS：定义用户所用的出口例程。当用户调用出口例程时，需要用该语句。当设置多个 EXITS 时，IEBCOMPR 将只用最后一个。EXITS 的语句格式为：

label　EXITS　INHDR=例程名
,INTLR=例程名
,ERROR=例程名
,PRECOMP=例程名

其中“INHDR=例程名”指定处理用户输入头标的例程名；“INTLR=例程名”指定处理用户输入尾标的例程名；“ERROR=例程名”指定出错处理接收控制的例程名；“PRECOMP=例程名”指定一个例程名，该例程在 IEBCOMPR 比较输入数据集之前对逻辑记录进行处理。

LABELS：指定是否将用户标号作为数据来处理，当设置多个 LABELS 语句时，IEBCOMPR 程序只用最后一个，LABELS 语句的格式为：

label　LABELS　DATA={YES | NO | ALL| ONLY}

其中 DATA= {YES | NO | ALL| ONLY}指明是否将用户标号作为数据处理。DATA 的取值如下：

YES：所有用户标号都作为数据处理，并依照返回码，将标号作为数据终止来处理，该值为缺省值。

NO：仅将用户标号作为数据处理。

ALL：所有用户标号作为数据处理，16 中返回码将使 IEBCOMPR 程序完成剩余用户标号组的处理并终止作业步。

ONLY：只用用户头标作为数据处理，处理时不管是否有返回码。

例1．比较两个分区数据集

```
//DISKDISK　JOB …
//STEP1　　EXEC　PGM=IEBCOMPR
//SYSPRINT　DD　SYSOUT=A
//SYSUT1　　DD　DSN=PDSSET1,UNIT=disk,DISP=SHR,
//　　　　　　DCB=(RECFM=FB,LRECL=80,BLKSIZE=2000),
```

```
//                    VOLUME=SER=111112
//SYSUT2      DD    DSN=PDSSET2,UNIT=disk,DISP=SHR
//                    DCB=(RECFM=FB,LRECL=80,BLKSIZE=2000)
//SYSIN       DD    *
    COMPARE    TYPROG=PO
/*
```

在上例中，SYSUT1 DD 语句定义输入数据集（PDSSET1），这是个组块数据集，它驻留在磁盘卷上。SYSUT2 DD 语句定义另一个输入数据集（PDSSET2），它也是个驻留在磁盘卷上的块组数据集。SYSIN DD 语句定义流内控制数据集，其中的控制语句表示两个输入数据集是分区数据集。

例 2. 比较磁带上的两个顺序数据集

```
//TAPETAPE JOB    ...
//              EXEC PGM=IEBCOMPR
//SYSPRINT DD    SYSOUT=A
//SYSUT1     DD    DSNAME=SET1,LABEL=(2,SUL),DISP=(OLD,KEEP),
//                   VOL=SER=001234,DCB=(DEN=2,RECFM=FB,LRECL=80,
//                   BLKSIZE=2000,TRTCH=C),UNIT=tape
//SYSUT2     DD    DSNAME=SET2,LABEL=(,SUL),DISP=(OLD,KEEP),
//                   VOL=SER=001235,DCB=(DEN=2,RECFM=FB,LRECL=80,
//                   BLKSIZE=2000,TRTCH=C),UNIT=tape
//SYSIN      DD    *
      COMPARE    TYPORG=PS
      LABELS    DATA=ONLY
/*
```

SYSUT1 DD 定义了一个输入数据集 SET1。该数据集在一个有标号（labeled）的、7 轨磁带卷上。

SYSUT2 DD 定义了一个输入数据集 SET2。该数据集在一个有标号（labeled）的、7 轨磁带卷上。它是该磁带上第一个或唯一一个数据集。

SYSINDD 后的控制语句 COMPARE TYPORG=PS 表明输入数据集是顺序数据集；LABELS DATA=ONLY 表明用户首标（header labels）被当作数据加以比较，而磁带上的其他标号则予以忽略。

2. IEBCOPY

IEBCOPY 程序用于拷贝或合并多个分区数据集或扩展分区数据集，其作用有：

（1）拷贝完整的数据集或部分数据集；

（2）建立分区数据集或扩展分区数据集的备份，备份是存放在磁带或磁盘上的顺序数据集；

（3）变更分区数据集或扩展分区数据集的成员、为选择的成员更换新名；

（4）对加载模块进行拷贝和重新组块。

下面是 IEBCOPY 的作业控制语句列表：

语 句	说　　明
JOB	作业初始
EXEC	定义程序名 PGM=IEBCOPY
SYSPRINT　DD	定义由 IEBCOPY 产生的系统输出信息
SYSUT1　DD	定义输入的分区数据集或扩展分区数据集
SYSUT2　DD	定义输出的分区数据集或扩展分区数据集
SYSUT3　DD	定义一个一处数据集，该语句在没有足够的虚拟空间存放输入的分区数据集或扩展分区数据集目录入口时使用
SYSUT4　DD	定义一个一处数据集，该语句在没有足够的虚拟空间存放输出的分区数据集或扩展分区数据集目录入口时使用
SYSIN　DD	定义控制语句，可在这里定义的语句有：COPY、ALTERMOD、COPYMOD、SELECT、EXECLUDE

下面对控制语句进行说明：

COPY：启动一个或多个拷贝、写在或加载操作。其格式为；

label　COPY　OUTDD=ddname

,INDD=({ddname | (ddname,R)},…)

,LIST={YES|NO}

操作数说明：

“OUTDD=ddname”：指定输出的分区数据集名，这里的 ddname 必须是本作业步中的一个 DD 语句名。

“INDD=({ddname | (ddname,R)},…)”：指定输入分区数据集或扩展分区数据集，对于卸载操作只能指定一个 ddname，这里的 R 表示从输入的数据集中选择所有的成员进行相应的操作，此时不需设置 SELECT 语句。

“LIST={YES|NO}”：指出是否将拷贝的成员名列在 SYSPRINT 数据集中，取值为 YES 时将成员名列表。

需要注意的是：

（1）如果为拷贝操作，输入数据集和输出数据集必须是分区数据集、扩展分区数据集或卸载模块结果的顺序数据集；

（2）如果是加载操作，输入数据集必须是分区数据集或顺序数据集，而输出数据集则必须是分区数据集；

（3）如果是卸载操作，输入数据集必须是分区数据集、扩展分区数据集或顺序分区数据集，输出数据集可以驻留在直接存取卷上，也可以驻留在磁带卷上。当驻留在磁带卷上时，它的组织结构必须是顺序数据集，且要指定省略目录或缩影值得 SPACE 参数。

ALTERMOD：指明加载模块变更的开始，其格式为：

label　ALTERMOD　OUTDD=ddname，LIST={YES|NO}

操作数说明：

“OUTDD=ddname”：指定要变更的数据集；

“LIST={YES|NO}”：指明是否将变更成员名列在 SYSPRINT 数据集中，取值为 YES 时，被变更的所有成员名列在 SYSPRINT 数据集中。

COPYMOD：指明对拷贝和加载模块从新组块的操作，其格式为：

label　COPYMOD　OUTDD=ddname

,INDD=({ddname | (ddname,R)},…)

,MAXBLK={nnnn|nnK}

,MINBLK={nnnn|nnk}

,LIST={YES|NO}

操作数说明：

“OUTDD=ddname”：指定加载模块要拷贝到的分区数据集；

“INDD=({ddname | (ddname,R)},…)”：指定输入数据集名，该数据集是一个加载模块库，并在本作业的一个名为 ddname 的 DD 语句中定义，这里的 R 表示从输入的数据集中选择所有的成员进行拷贝，并变更输出加载模块库中任何指定的成员名，此时不需设置 SELECT 语句；

“MAXBLK={nnnn|nnK}”：指定输出分区数据集记录的最大块值，通常设置的值要小于缺省值，以便数据记录能和其它系统或程序兼容，nnnn 是一个十进制数，nnK 则表示 nn 千字节，其缺省值是输出数据集的块大小；

“MINBLK={nnnn|nnk}”： 指定输出分区数据集记录的最小块值，缺省值为 1K；

“LIST={YES|NO}”：指明是否将变更成员名列在 SYSPRINT 数据集中，取值为 YES 时，被变更的所有成员名列在 SYSPRINT 数据集中。

SELECT：确定输入数据集中要拷贝的成员名，其格式为：

label SELECT MEMBER=（{name1|（name1，newname1,R）|（name1, ,R）}|
，（name2，newname2，R）|（name2，，R）}，。。。）

操作数说明：

“MEMBER=…”：name 确定拷贝的成员名；newname 确定拷贝输出的成员名，如果这个名字在输出分区数据集中已存在，则这个成员不被拷贝，除非同时设置 R 参数，newname 和 ALTERMOD 不能同时使用。

EXCLUDE：确定输入数据集中不被拷贝和不加载的成员，其格式为：

label EXCLUDE MEMBER=（name1，name2，…）

下面是几个实例：

例 1.

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

上面是一个拷贝整个数据集的例子。在该例中，SYSOUT1 DD 定义一个含有两个成员的分区数据集 DATASET5，SYSUT2 DD 定义一个新的分区数据集 DATASET4，并为它分配 5 个磁道，其中两个磁道分配给目录区。在这个例子中不需要 SYSIN DD 语句，它把数据集 DATASET5 的所有成员都拷贝到数据集 DATASET4 中。

例 2.

```
//COPY    JOB    …
//JOBSTEP    EXEC    PGM=IEBCOPY
//SYSPRINT    DD    SYSOUT=*
//OUT1          DD    DSN=DATESET1,UNIT=disk,VOL=SER=111112,
//                DISP=(OLD,KEEP)
```

```
//IN6          DD   DSN=DATASET6,UNIT=disk,VOL=SER=111115,
//                  DISP=OLD
//IN5          DD   DSN=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)
/*
```

上面是一个拷贝、置换被选择的数据集成员的例子。在该例中：

OUT1 DD 语句定义含有三个成员（A、B 和 F）的分区数据集 DATASET1；

IN6 DD 语句定义含有三个成员（B、C 和 F）的分区数据集 DATASET6；

IN5 DD 语句定义含有两个成员（A、C）的分区数据集 DATASET5；

SYSUT3 和 SYSUT4 DD 语句定义临时溢出数据集，并为它们分配一个磁道；

SYSIN DD 语句定义流内控制数据集，它包含一个 COPY 语句、一个 INDD 语句和一个 SELECT 语句。OUTDD 参数指定 DATASET1 作为输出数据集，INDD 指定 DATASET5 作为第一个处理的输入数据集，而 DATASET6 作为第二个处理的输入数据集，它的处理顺序为：

（1）在 DATASET5 中查询所选择成员。

（2）首先查到成员 A，但不拷贝到 DATASET1 中，因为 DATASET1 中已经含有成员 A 且没有设置 R 参数、。

（3）在 DATASET5 中未查到所需要的成员，继续在 DATASET6 中查找。

（4）找到成员 B 并拷贝到 DATASET1 中，虽然 DATASET1 中已经含有成员 B，但在这里设置了 R 参数。

SELECT 指定从输入数据集拷贝选择的成员到输入数据集。

例 3.

```
//CONVERT   JOB   …
//STEP1       EXEC   PGM=IEBCOPY
//SYSPRINT  DD      SYSOUT=*
//SYSUT1      DD      DSN=PDSSET,DISP=SHR,DSNTYPE=PDS
//SYSUT2      DD      DSN=PDSSET,LINK=PDSSET,DSNTYPE=LIBRARY,
//                DISP=(NEW,CATLG)
```

上面是一个分区数据集转换为扩展分区数据集的例子。在该例中，SYSUT1 DD 语句定义一个输入的分区数据集 PDSSET，SYSUT2 DD 语句定义输出的扩展分区数据集 PDSE，所有的扩展分区数据集都是由存储管理子系统（SMS）进行管理，其中 LINK 子参数为扩展分区数据集设置 DCB 和 SPACE 特性，DSNTYPE 子参数指出新建数据集是扩展分区数据集而不是分区数据集。这里不需要 SYSIN DD 语句。

例 4.

```
//UNLOAD   JOB   …
```

```
// EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=A
//SYSUT1   DD DSN=inpds,DISP=SHR
//SYSUT2   DD DSN=tape.dataset.name,UNIT=TAPE,
//    VOL=SER=tape#,LABEL=#,DISP=(NEW,PASS)
//SYSIN   DD DUMMY
/*
```

上面是将一个分区数据集转存为磁带上的顺序数据文件的例子。

例 5.

```
//UPLOAD JOB …
// EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=A
//SYSUT1   DD DSN=tape.dataset.name,DISP=(OLD,PASS),
//    UNIT=TAPE,VOL=SER=tape#,LABEL=#
//SYSUT2   DD   DSN=outpds,DISP=SHR
//SYSIN   DD   DUMMY
/*
```

上例是将一个转存为磁带上顺序文件的分区数据集按原有的 DCB 属性恢复到 DASD 上（假定该分区数据集已经存在）。

3. IEBGENER

IEBGENER 程序主要用于：

（1）建立顺序数据集、分区数据集或扩展分区数据集成员的备份。这个拷贝可以是磁带到磁带、磁盘到磁盘或磁盘到磁带。注意：如果需要将备份数据集放在原数据集所在的卷，它们两者不能同名。

（2）从顺序数据集产生分区数据集或扩展分区数据集。通过实用程序控制语句，将顺序数据集逻辑化分为若干个记录组并为其分配成员名，之后 IEBGENER 程序把这些新建的成员放到指定的分区数据集或扩展分区数据集中。注意：对于含有跨区记录的数据集不能产生分区数据集或扩展分区数据集。

（3）为分区数据集添加新成员。IEBGENER 程序将输入的顺序数据作为一个成员加到指定的分区数据集或扩展分区数据集中。

（4）产生一个编辑的顺序数据集、分区数据集或扩展分区数据集。通过使用实用程序控制语句，指定一个或一组记录或整个数据集的编辑信息。

（5）处理含有双字节字符数据集。用 IEBGENER 可以拷贝、编辑、重新组块或打印含有双字节字符（DBCS）的数据，也可以将含有 DBCS 数据的顺序数据集转换成为分区数据集。

（6）打印顺序数据集、分区数据集或扩展分区数据集的成员。

（7）对数据集的逻辑记录进行重新组块或改变其长度。

（8）为顺序输出数据集拷贝用户标号。

（9）为用户例程提供编辑设施及出口，该例程用于处理标号、受控输入数据及永久性输入输出错误。

下面是 IEBGENER 的作业控制语句列表：

语　句	说　　明
JOB	作业初始
EXEC	指定程序名 PGM=IEBGENER
SYSPRINT　DD	指定系统输出数据集
SYSUT1　DD	定义输入数据集
SYSUT2　DD	定义输出数据集
SYSIN　DD	定义控制数据集，控制语句可以是 GENERATE、EXITS、LABELS、MEMBER、RECORD

控制语句说明：

GENERATE：指明成员名和别名数、记录标识符、文字及控制数据集中的编辑信息。

EXITS：指明用户出口例程。

LABELS：特指用户标号处理。

MEMBER：指定新建分区数据集或扩展分区数据集的成员名或成员别名。

RECORD：定义将处理的记录组并提供编辑信息。

例 1. //PRINT　JOB …

```
//STEP1　EXEC　PGM=IEBGENER
//SYSPRINT　DD　SYSOUT=A
//SYSIN　　DD　DUMMY
//SYSUT1　DD　DSN=INPUT,UNIT=3380,DISP=SHR,
//　　DCB=(RECFM=F,LRECL=80,BLKSIZE=80),VOL=SER=111112
//SYSUT2　DD　SYSOUT=A,DCB= DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
```

上面是一个打印顺序数据集的例子。由于该作业不需要公用程序的控制语句，因此 SYSIN DD 语句定义了空数据集 DUMMY，SYSUT1 定义了输入顺序数据集 INPUT，SYSUT2 定义了输出数据集的输出设备，其中 DCB 参数用于改变数据输出格式。

4. IEBPTPCH

IEBPTPCH 可以用来打印输出部分或整个顺序或分区数据集。打印记录的格式可以采用标准格式也可以由用户指定。

标准的格式是：

- 一行只打印一个逻辑记录。
- 每行的输出格式为：每 8 个字符为一组，每组之间由 2 个空格隔开。
- 不能打印的字符显示为空格
- 如果输入是分块的，那么每个逻辑记录以“*”号间隔，而每个块之间以“**”间隔。
- 每页打印 16 行。

在输出设备的容量范围内，用户可以自定义输出记录的格式。IEBPTPCH 提供了可选的编辑功能来处理输入或输出记录。

IEBPTPCH 可以用来打印：

1. 整个顺序数据集或分区数据集（或扩展分区数据集）；
2. 一个分区数据集（或扩展分区数据集）的部分成员；
3. 一个顺序数据集或分区数据集（或扩展分区数据集）的部分记录；
4. 一个分区数据集（或扩展分区数据集）的目录；
5. 打印一个顺序数据集或或分区数据集（或扩展分区数据集）的修改版（Edited

version）。

下面是 IEBPTPCH 的作业控制语句列表：

语 句	说　　明
JOB	作业初始
EXEC	定义程序名 PGM=IEBPTPCH
SYSUT1　DD	定义输入的数据集
SYSUT2　DD	定义输出的数据集，如果定义为 SYSOUT=A，则为打印输出

控制语句说明：

PRINT：表示输入数据集将被打印。如果要进行打印操作，它必须是第一条操作语句。

MEMBER：说明一个分区数据集中需要打印的成员。

RECORD：说明用户设定的打印格式。

TITLE：指定一个标题。该标题将被打印在所有数据之前。每个打印作业可以包含两个 TITLE 语句，第一个 TITLE 语句指定标题，而第二个 TITLE 语句指定子标题。

实例：

一个一般的调用 IEBPTPCH 的作业格式如下：

//EXAMPLE　JOB …
// EXEC PGM=IEBPTPCH
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD …
//SYSUT2 DD …
//SYSIN DD *
在此处放置 IEBPTPCH 的实用程序控制语句
/*

1. 要打印一个顺序数据集或分区数据集，那么//SYSUT1 DD 语句应该定义为：
   - 磁带上的数据集：
     //SYSUT1 DD DSN=tape.dataset.name,UNIT=TAPE,
     //　VOL=SER=tape#,LABEL=#,DISP=OLD
   - 顺序的 DASD 上的数据集：
     //SYSUT1 DD DSN=seq.dataset.name,DISP=SHR
   - 分区数据集：
     //SYSUT1 DD DSN=*pds.name*,DISP=SHR

2. 要打印分区数据集中的部分成员，那么//SYSUT1 DD 语句应该定义为：
   //SYSUT1 DD DSN=*pds.name*,DISP=SHR
   并且， 实用程序控制语句应为：

```
PRINT MAXNAME=n
MEMBER NAME=member1
MEMBER NAME=member2
    .
    .
    .
MEMBER NAME=membern
```

其中，n 是所要打印的成员总数，而 member1,member2,...,membern 是所要打印的成员名字。

3．要打印顺序数据集或分区数据集的部分记录，那么//SYSUT1 DD 语句应该定义为：

//SYSUT1 DD DSN=*data.set.name*,DISP=SHR

并且， 实用程序控制语句应为：

```
PRINT MAXFLDS=f,MAXGPS=g,MAXLITS=l,STOPAFT=nnnnn
RECORD IDENT=(length,'name',inloc),FIELD=(length,inloc,,outloc)
```

其中：

f：表示 RECORD 语句中，FIELD 参数的个数

g：表示 RECORD 语句中，IDENT 参数的个数

l：表示 RECORD 语句中，IDENT 标示符中包含的字符数（最多 32,767 个）

nnnnn：表示要打印的逻辑记录的个数

length：输入记录中包含 identifying name 的区域长度（以字节为单位），该长度不能超过 8 个字节

'name'：该标识符用来精确标示记录组的最后一个记录

inloc：表示输入记录中包含 identifying name 的区域的开始位置

outloc：表示输出记录中该区域的开始位置

4．要打印一个分区数据集的目录，那么//SYSUT1 DD 语句应该定义为：

//SYSUT1 DD DSN=pds.name,DISP=SHR

并且， 实用程序控制语句应为：

```
PRINT    TYPORG=PS
TITLE    ITEM=('PRINT PARTITIONED DIRECTORY OF A PDS',outloc)
```

其中：

TYPORG=PS：表示分区数据集的目录是一个顺序的结构

outloc：在输出结构 ITEM 参数的内容的起始位置

5．要打印分区数据集或顺序数据集的修改版本，那么//SYSUT1 DD 语句应该定义为：

//SYSUT1 DD DSN=data.set.name,DISP=SHR

并且， 实用程序控制语句应为：

```
PRINT     MAXFLDS=f,CDSEQ=seqno1,CDINCR=incr
RECORD    FIELD=(72)
```

其中：

f：表示 RECORD 语句中，FIELD 参数的个数

seqno1：第一行的序号

incr：增加序号的数量

FIELD=(72)：表示输入记录的 1-72 列将被打印，73-80 行将被新增的序号替代

6．使用标准格式打印一个分区数据集的成员，但其中的数字将转换为 16 进制的格式打印，那么//SYSUT1 DD 语句应该定义为：

//SYSUT1 DD DSN=*data.set.name*,DISP=SHR

并且， 实用程序控制语句应为：

```
PRINT     TYPORG=PO,TOTCONV=XE,MAXNAME=n
MEMBER NAME=member1
MEMBER NAME=member2
    .
    .
```

.

MEMBER NAME=membern

其中，XE 表示将数字由十进制转换为十六进制输出。

需要注意的是，IEBPTPCH 的控制语句还有许多参数在本节中并没有办法一一介绍，读者需要参考有关手册进行使用。

5．IEHLIST

IEHLIST 程序用于系统信息列表，其中包括分区数据集目录列表、VTOC 列表以及编目列表等。下面是它的应用实例：

```
//LIST        EXEC   PGM=IEHLIST
//SYSPRINT    DD     SYSOUT=*
//D1          DD     UNIT=SYSDA,VOL=SER=PACK11,DISP=OLD
//D2          DD     UNIT=SYSDA,VOL=SER=PACK12,DISP=OLD
//D3          DD     UNIT=SYSDA,VOL=SER=PACK17,DISP=OLD
//SYSIN       DD     *
 LISTCTLG    VOL=SYSDA=PACK12
 LISTVTOC    VOL=SYSDA=PACK11,DSN=(USER.F1)
 LISTPDS     VOL=SYSDA=PACK17,DSN=U1.LIB
/*
```

注意：DD 语句中的 VOL 和 UNIT 参数的设置要与 SYSIN DD 中控制语句的相关参数一致。

6．IEFBR14

IEFBR14 是一个不含控制语句的实用程序，它可以用来创建或删除磁盘数据集。下面是它的应用实例：

例 1. 数据集的删除

```
//DELETE   JOB …
//            EXEC   PGM=IEFBR14
//DD1         DD   DSN=data.set.name,DISP=(OLD,DELETE)
/*
```

需要注意的是，如果要删除的数据集已编目，用户不能在指定 UNIT 或者 VOL=SER 参数。如果用户指定了以上参数，那么数据集虽然同样被删除，但不能被反编目。

7．DFSORT

DFSORT 程序用于数据排序，下面是它的应用实例：

```
//STEP1   EXEC   PGM=DFSORT
//SYSIN   DD   *
  SORT   FIELDS=(1,10,CH,A)
//SORTIN   DD   DSN=TEST.LOG,DISP=OLD
//SORTOUT   DD   DSN=SORT.LOG,DISP=(NEW,PASS),
//                      UNIT=SYSDA,SPACE=(CYL,1)
//SORTWK1   DD   UNIT=SYSDA, SPACE=(CYL,1)
```

STEP1 语句用于调用 DFSORT 程序；SYSIN DD 语句定义控制数据集，其中控制语句

SORT　FIELDS=(1,10,CH,A)指出要排序的内容始于输入数据的第一个位置，以递增顺序对前 10 字符进行排序；SORTIN DD 语句给出用于排序的输入数据集名和状态；SYSOUT DD 语句为排序的输出结果指定数据集；SORTWK1 DD 语句为排序操作分配工作空间。

z/OS 中的实用程序有很多，在本节中不能一一介绍，读者只要通过本节中介绍的几个常用的实用程序及其实例就可以掌握实用程序的一般使用方法，具体参数的说明及格式等，读者可以参考 IBM 有关的手册。

4.7 小结

JCL 是 z/OS 环境下用户进行批处理作业的基本手段。无论是大型主机的系统开发人员，熟练掌握 JCL 语言和常见的实用程序都是必需的。由于历史原因，主机的 JCL 对格式等方面的要求较为严格，语句也比较繁琐，掌握起来有一定的难度。希望读者能在掌握 JCL 的一般结构、编写规范的基础上参考有关手册多进行实践。本章的部分内容还涉及到一些 z/OS 及大型主机的基础知识，读者可以结合本书的其它章节以及其它资料进行学习。