



DSE 大型机实用指南系列

JCL 实用指南

V2.00

声明

本作品受知识共享许可协议保护，作者保留作品的版权，但是允许地球人以及外星人在保留作者的署名并且遵守在“[署名-非商业性使用-相同方式共享 2.5 中国大陆](#)”这里指定的条件的条件下，复制和发行本作品。

With a Creative Commons license, I keep my copyright but allow people to copy and distribute my work provided they give my credit — and only on the conditions I specify here - “[Creative Commons Attribution-Noncommercial-Share Alike 2.5 China Mainland License](#)”. 

本作品所述内容不代表任何公司或组织的立场和观点，如有雷同纯属抄袭。作者对本作品所显示的信息或资料的准确性、内容、完整性、合法性、可靠性、可操作性或可用性不承担任何责任。

关于

关于本作品

JCL 实用指南是 DSE 大型机实用指南系列作品的入门级作品，旨在介绍并说明 Job Control Language 在 MVS 上的用途、工作原理、用法以及使用技巧。本作品提供所有实例，均经过作者亲自上机反复实践得出。

任何建议、意见、不足、错误、遗漏，都欢迎您与作者联系，以保持本作品的科学性和严谨性。

目标读者

从事以及希望从事大型机（mainframe）相关工作的 IT 精英以及学生朋友们，为了能够清楚的理解本作品，在您阅读前，期望您掌握以下相关知识或经验并且拥有能够自由练习的 MVS 环境。

TSO/ISPF

MVS/zOS/OS390

SDSF/SYSVIEW

关于作者

Mainframe Senior System Programmer，80 后，“毕业”于素有 MF“西点军校”美誉的 IBM DL DSE 项目组，现任某海外金融数据中心 z/OS 首席技术顾问。

Email: mainframechina[at]gmail.com

本作品授权由 MIB(www.ibmmainframe.cn)唯一发行。

目录

JCL 实用指南.....	1
声明.....	2
关于.....	2
再版说明.....	4
1. 概要.....	5
1.1 Hello World.....	5
1.2 什么是 JCL?.....	6
1.3 为什么要使用 JCL?.....	7
1.4 JCL 在 MVS 上是怎样工作的?.....	7
1.5 JCL 都可以做什么?.....	8
2. 语法.....	9
2.1 规则.....	9
2.2 命名.....	9
2.3 语句.....	9
2.4 参数.....	10
2.5 续行.....	12
2.6 练习.....	13
3. 实例.....	14
3.1 Data Set 相关.....	14
3.1.1 百善孝为先——Allocate DS.....	14
3.1.2 万恶淫为首——Copy DS.....	16
3.1.3 勿以恶小而为之——Move DS.....	18
3.1.4 勿以善小而不为——Rename DS.....	19
3.1.5 己所不欲 勿施于人——Update DS.....	20
3.1.6 夫欲立 则先立人——Delete DS.....	21
3.1.7 穷则独善其身——Dump DS.....	22
3.1.8 达则兼济天下 ——Restore DS.....	23
3.1.9 修身 齐家 平天下——Compare DS.....	24
3.1.10 夫君子者 莫过于此——Compress DS.....	27
3.2 开发相关.....	29
3.2.1 编译链接.....	29
3.2.2 VSAM.....	31
3.2.3 GDG.....	33
3.2.4 SORT.....	35
3.3 系统相关.....	38
3.3.1 格式化磁盘.....	38
3.3.2 磁盘拷贝.....	38
3.3.3 更改磁盘卷标.....	40
3.3.4 定义/删除别名 (Alias).....	40

3.3.5 导入/备份 User Catalog.....	41
3.3.6 执行 TSO/RACF 命令.....	42
3.3.7 创建 RACF TSO ID.....	42
3.3.8 提交 REXX/CLIST 程序.....	43
3.3.9 DS 压缩与解压缩.....	43
3.4 SMPE 相关.....	45
3.4.1 Receive.....	45
3.4.2 Apply.....	46
3.4.3 Accept.....	47
3.4.4 Reject.....	48
3.4.5 Restore.....	48
3.4.6 Zone copy.....	49
3.4.7 Zone report.....	49
3.4.8 Zone list.....	50
4. 附录.....	51
4.1 参考文献.....	51
4.2 商标.....	51
4.3 致谢.....	52

再版说明

V2.00

1. Update - JCL5.1
2. Add - 3.3.7 创建 RACF TSO ID
3. Add - 3.4 SMPE 相关

1. 概要

1.1 Hello World

我们知道任何一门计算机语言都是由 Hello World 程序开始讲起的，但是现在你所看到的一定是第一本从 Hello World 开始讲起的 JCL 教材。在 TSO 上提交下面这段 JCL 代码，你将在 JOBLOG 里面得到图 1-1 的结果。

```
//HELLOW JOB , 'BILLRAIN',MSGCLASS=H,CLASS=A,NOTIFY=&SYSUID,
//
//          MSGLEVEL=(1,1)
//*=====
//*THIS JCL IS USED TO DISPLAY "HELLO WORLD" IN SYSOUT
//*=====
//STEP01 EXEC PGM=IEBGENER
//SYSIN DD DUMMY
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD *
*****
* HELLO, WORLD! *
* THIS IS YOUR 1ST JCL!! *
* WELCOME TO BILLRAIN.COM!!! *
*****
/*
//SYSUT2 DD SYSOUT=*
//
```

JCL1-1

```
DATA SET UTILITY - GENERATE
IEB352I WARNING: ONE OR MORE OF THE OUTPUT DCB PARMS COPIED FROM INPUT

PROCESSING ENDED AT EOD
*****
* HELLO, WORLD! *
* THIS IS YOUR 1ST JCL!! *
* WELCOME TO BILLRAIN.COM!!! *
*****
```

图 1-1

1.2 什么是 JCL?

如果你成功地跑出了 Hello World，恭喜你！那么如果你认为 JCL 是一门如此简单的编程语言，那你就大错特错了！这里必须要澄清的一点就是，JCL 根本不是一门编程语言，那么它究竟是什么呢？

Job Control Language，顾名思义，是控制 MVS 上 JOB 的语言，它与 COBOL 或 JAVA 这样的计算机编程语言不同，它没有编译、链接的过程，不会生成可执行文件。在大型机上，如果你想让 MVS 系统让你工作，你就需要告诉系统你需要完成什么工作，需要哪些资源，JCL 就可以为 MVS 提供这些信息。

简单地说，可以把 JCL 理解为饭馆里面的“菜单”，当你进入一家西餐厅打算享用一套正宗的法式大餐的时候，你并不需要走进厨房亲自烹调，侍者会为你提供一本精美的菜谱供你选择，你只要从中挑选中意的菜品，过不了多久，就会有人把开胃酒、沙拉、主菜和甜点依次送到你的面前。



图 1-2

类似地，当你酒足饭饱后回到办公室的 TSO 画面前，你希望 MVS 像餐厅一样为你提供优质的服务，那么首先你需要一本菜单，这就是 JCL，在 MVS 的菜单里面，记录的不是每道菜品的原料、做法以及价格，而是运行一个 JOB 所需要信

息和资源。当然你不会是餐厅内唯一的客人，你也不会是 MVS 上唯一的用户，同一时间会有无数用户通过 JCL 在执行不同的 JOB，访问不同或者相同的资源。当然你需要做的只是把写好的 JCL 提交给系统，系统会自动设定不同 JOB 的优先级以及它们访问资源的先后顺序，就好比后厨会安排不同客人的菜品的顺序一样，在等待菜品上来之前，你是可以自由支配这段时间的。

1.3 为什么要使用 JCL?

在大机上存在着两种 Workload 工作模式，以上我们讲到的通过 JCL 来提交工作的方法叫做 BATCH（类似于 DOS 的批处理）；另外一种好比你去使用 ATM 机，从插卡开始，你需要进行输入密码、金额、取现、取卡等一系列交互式操作，显然你是不能像去餐厅那样把卡片交给服务员然后等待他把现金提出来给你，这种模式我们称之为 ONLINE。

所以，如果要用一句话来解释 JCL 的功能——JCL 就是用来提交 BATCH JOB 的。当然通过 JCL 执行的 JOB 会产生一个 JOBLLOG，里面记录了 JOB 执行状况以及正常/异常终止信息，你可以通过这个 LOG 最直观地来了解 JOB 是否达到了预期的要求，或根据错误信息进行下一步调试。

1.4 JCL 在 MVS 上是怎样工作的？

现在各位应该都对 JCL 有一个简单的了解了，那么在 MVS 上 JCL 是怎么具体实现它的各种功能的呢？与怎样使用好 JCL 相比，这可是一个非常不好解释的问题。在这里，我们需要引入 JES（Job Entry Subsystem）系统的概念，就像你把点好的菜单交给侍者（JES）那刻起，侍者会把它传递到后厨，学徒们拿到菜单会为师傅准备原料，洗好切好给师傅下锅，而 MVS 真正所做的就像大厨一样，是入锅烹调的过程，当然做好后 JES 还会负责把 output 呈现到你的面前。

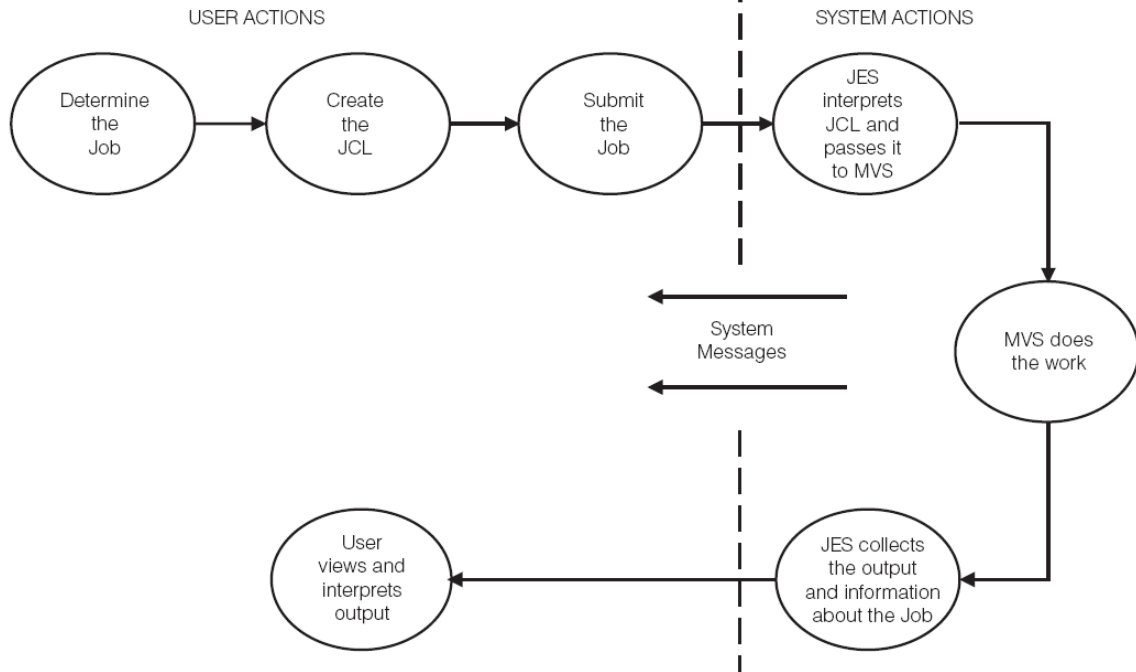


图 1-3

至于 JES 与 MVS 之间更详细的工作原理，我们将会在随后的系列指南中逐步向大家介绍。

1.5 JCL 都可以做什么？

作为一门非编程语言，显然 JCL 不是万能的，它的主要功能都是通过自带的 UTILITY 来实现的，一个 UTILITY 就相当于 JCL 提供的实现特定功能的一个组件，常用的也就十来种左右，他们可以实现诸如下面这些功能：

- 复制/移动/删除/重命名/更新 Data Set
- 执行 RACF 或 TSO 命令
- 更新数据库
- DASD/TAPE 操作
- 编译/链接/运行 COBOL/PLI 程序
- 等等.....

2. 语法

下面我们讲进入本教程最基础也是最枯燥的部分，如果你自认为对 JCL 的语法规则了然在胸的话可以跳过本章，新手们还是读一遍的好。

2.1 规则

JCL 一般是存放在长度为 80 格式为 FB 的 DS 里面，除去最右端 8 位行号之外，有效输入范围为 1-72 列（column 1-72），左端起始 2 列为 identifier field，只能允许下面这几种情况。

```
//jobname JOB , 'BILLRAIN', MSGCLASS=H, CLASS=A, NOTIFY=&SYSUID
//stepname EXEC PGM=IEBGENER
//ddname DD SYSOUT=*
-----+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7--
//SYSIN DD *
HELLO
/*
/* THIS IS COMMENT
//
```

JCL2-1

/* - 流内数据结束标识
 /* - 注释行
 // - 空语句（null statement），通常为结束行

2.2 命名

从第三列开始为 name field，每个名字最大长度为 8 个字符，有效字符包括大小写英文字母 a~z、A~Z、阿拉伯数字 0~9 以及 3 个通配符（\$、#、@），且必须以字母或通配符开始，名字后跟至少一个空格结束。

通常地，命名的时候不以 SYS、JOB 开头。

2.3 语句

在 name field 后面 operation field，JCL 的主要 3 个语句都在这里指定，其后面也必须跟随一个空格结束。

JOB statement – 用来标记一个 Job 的开始，告知系统如何处理这个 Job（必须是一个 Job 的第一个语句，且一个 Job 内只能有一个 JOB statement）

EXEC statement – 用来指定每个 Step 执行的程序或 PROC，并告知系统如何处理这个 Step（一个 Job 内最多有 255 个 Step）

DD statement – 用来描述和指定每个 Step 执行时需要用到的 Data Set 的位置、大小、状态等信息（一个 Step 内的所有 DD statement 必须写在 EXEC statement 下面，但是它们之间的顺序可以互换。

2.4 参数

在 operation field 后面称为 parameter field，在参数域中有 2 种类型的参数，位置参数（positional parameter）和关键字参数(keyword parameter)：位置参数是固定的，而关键字参数是可以省略的。

比如 JOB 语句后面的参数有：

```
//jobname JOB BMW,'BILLRAIN',MSGCLASS=H,CLASS=A,NOTIFY=&SYSUID
    ↑ 位置参数，记账信息
        ↑ 位置参数，程序员名
//jobname JOB , 'BILLRAIN'
    ↑ 没有内容位置也要留出来，用逗号省略
        ↑ 后面都是关键字参数，可以省略，当然一般没有省的这么干净
//jobname JOB BMW,'BILLRAIN',MSGCLASS=H,CLASS=A,NOTIFY=&SYSUID,
//          MSGLEVEL=(1,1)          写不下要续行喽，注意有逗号哦 ↑
```

JCL2-2

JOB 语句常用的关键字参数有：

Keyword Parameter	Value	Purpose
CLASS=n	A~Z, 0~9	Job 在 JES 队列中的优先级；
MSGCLASS=n	A~Z, 0~9	Joblog 在输出时的优先级
MSGLEVEL=(m,n)	m= 0 只有 JOB 语句 1 所有 JCL 和 PROC 语句 2 只有 JCL 语句 n= 0 只有 JCL 消息	指定在 Joblog 出打印的内容，一般为 (1, 1)

	<i>1 JCL、JES 和操作员消息</i>	
NOTIFY= <i>userid</i>	&SYSUID 默认为提交者 ID	将 Job 返回码返回给指定的 TSOID
RESTART= <i>stepname</i>	<i>stepname</i>	从指定 Step 重新执行该 Job
TYPRUN=	SCAN HOLD	只检查 JCL 语法而不实际运行结果； 将 JOB 提交以 HELD 状态提交，需要 RELEASE 才可执行
USER= <i>userid</i>	<i>userid</i>	以其他 ID 身份提交（需要特殊授权）
TIME=	(<i>mm,ss</i>) 1440	指定 JOB 执行最长占用 CPU 的时间； 1440 为无限期执行； 不要指定为 0
REGION=	8M 16M 32M 0M	指定 JOB 运行时最大占用的内存大小； 0M 为当前系统无限制

表 2-1

EXEC 语句的参数有：

```

//jobname JOB BMW,'BILLRAIN',MSGCLASS=H,CLASS=A,NOTIFY=&SYSUID
//stepname EXEC PGM=IEBGENER
    ↑ 位置参数 必须是这三种情况之一 ①. 程序名 PGM=
//stepname EXEC PROC=COPYPROC
    ↑ ②. PROC 名
//stepname EXEC COPYPROC
    ↑ ③. 直接写 PROC 名
//stepname EXEC PGM=LREC,PARM='CLPA,NOREQ',TIME=97
    ↑ 关键字参数

```

JCL2-3

DD 语句的参数有很多，我们将在第三章结合实例具体为大家讲解。

```
//DUMPTST1 JOB , 'BILLRAIN', NOTIFY=&SYSUID, CLASS=A, MSGCLASS=H
//STEP1 EXEC PGM=ADRDSSU
//SOURCE DD UNIT=3390, VOL=SER=ILY003, DISP=SHR
//TARGET DD DSN=BILLRAIN.TEST.DUMP2,
//          UNIT=3390, VOL=SER=COMYYB, DISP=(NEW, CATLG, DELETE),
//          SPACE=(TRK, (100, 100), RLSE)
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DUMP INDDNAME(SOURCE) OUTDDNAME(TARGET) -
DATASET(INCLUDE(SYS1.C1*.**)) ALLEXCP ALLDATA(*)
/*
//
```

JCL2-4

2.5 续行

上面我们说过 JCL 每行最多只能容纳 72 个字符，参数太多写不下该怎么办呢？

```
//ALLOCDS JOB CLASS=A,
// TIME=1440,
↑ 参数域续行最前可以从第四列开始
-----+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7--
//          NOTIFY=&SYSUID,
↑ 最后可以从在第十六列开始
//          REGION=0M,
↑ 从这开始续是非法的
//          MSGCLASS=H, TPYRUN=SCAN, THIS IS COMMENT
↑ 中间不能有空格，逗号后空格被视为注释
//          MSGLEVEL=(1,1) IF YOU WANT TO CONTINUE THE COMMENT, PUT      X
// A NONBLANK CHARACTER IN COLUMN 72.
续行注释的话需要在第 72 列放一个任意字符，然后从下一行第 4 列往后继续注释
内容 ↑
//*****IBMMAINFRAME.CN*****
↑ 华丽的分割线 (◉_◉)
//STEP1 EXEC          PGM=IEFBR14, PARM=(PARM1, '/DIR1/DIR2
括号内的续行比较特殊，最后一位
要卡在第 71 列上，非常严格哦 ↑
-----+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7--
```

```
//          /DIR3/DIR4/DIR5/DIR6/DIR7/DIR8/DIR9/DIR10/DIR11/DIR12/DI
          ↑ 接下来的话必须从第 16 列开始续
//          R13/FILENM' )
//STEP010 EXEC PGM=ADD2NUM2,PARM='123456789012345678901234567890123456
//          78901234567890123456789012345678901234567890923456789012
//          34567890' ← PARM 里面最多只可以写 100 个字符，吼，这是为什么呢~
//*****IBMMAINFRAME.CN*****
```

JCL2-5

2.6 练习

好了，如果你的汉语水平不是太烂的话应该能看懂我写的大部分内容了，下面我们一起来写一个 JCL 练习一下。

题目 1：程序员 SAM (ID : SAM008) 要为 SY2Z 公司用 JCL 创建名为 'EDU.TEST.JCL' 的 Data Set，优先级为 H，输出优先级为 K，显示所有相关信息，返回码显示给 SAM。请为山姆大叔写出这个 JCL。

3. 实例

3.1 Data Set 相关

3.1.1 百善孝为先——Allocate DS

我们接上面的练习来讲解大型机上最最最基本的工作。

```
//SAM00801 JOB SY2Z,'SAM',CLASS=H,MSGCLASS=K,MSGLEVEL=(1,1),
↑ 有时你会被要求只能提交自己 ID 开头的 Job
      ↑ 这个 Job 是记在这个公司帐下的
            ↑ 后面的你都明白了吧

//          NOTIFY=SAM008
//CRETDS   EXEC PGM=IEFBR14
            ↑ 很多人都教你用这个去新建 DS 吧

//NEWDS    DD DSN=EDU.TEST.JCL,DISP=(NEW,CATLG),
            ↑ DD 语句的参数将做重点介绍
//          SPACE=(TRK,(15,10,1)),UNIT=3390,
//          DCB=(RECFM=U,LRECL=0,BLKSIZE=32760),
//          VOL=SER=COMY10
//SYSTSPRT DD SYSOUT=*
//
```

JCL3-1-1

IEFBR14 是一个 JCL Utility，通常被描述为用来创建 DS，而实际上它是一个“空程序”，什么功能也没有，真正控制创建 DS 的是下面的 DD 语句，作为一段完整的 JCL，JOB、EXEC 和 DD 语句都是必不可少的，虽然 DD 本身就可以完成对 DS 的创建，我们仍然需要 IEFBR14 这样一个空杆司令来充充门面，以至于很多初学者都认为它具有创建 DS 的功能，实际上你把上面的 DD 跟到任何一个 PGM 下面都会创建 DS。

```
↓ 用 DSN 来指定 DS 的名字
//NEWDS    DD DSN=EDU.TEST.JCL,DISP=(NEW,CATLG),
            ↑ DS 当前的状态
            ↑ NEW 为新建
            ↑ CATLG 表示当前步成功则编目此 DS

//          SPACE=(TRK,(15,10,1)),UNIT=3390,
            ↑ 新建 DS 的大小          ↑ 分配磁盘类型为 3390
            ↑ 单位为 Track
            ↑ 第一次分配 15 个 Track
//          SPACE=(TRK,(15,10,1)),UNIT=3390,
//*
```

```

                                ↑ 第二次分配 10 个 Track
                                ↑ 表示新建的为 PDS
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=32760),
                                ↑ 记录的格式为 Fixed Block
                                ↑ 逻辑长度为 80 字节
                                ↑ BLOCK SIZE 应为长度的整数倍
//          VOL=SER=COMY10
                                ↑ 新建 DS 所处的卷

```

磁盘空间分配的参考方法：

我们知道在大机上，数据是 record-oriented 而不是 byte-oriented 的，那么把 PC 上一个文件上传到大机上的时候实际上是需要我们自己来估算其所占用的空间大小的，假设 3390-3 的磁盘，1 trk = 56664 bytes，1 cyl = 15 trks = 0.85MB，一共 3339 个 cylinders 约为 2.8GB。PDS 在分配 DS 空间时，除第一次分配外，最多追加分配 15 次，一共最多为 16 次分配，例如上面的情况，该 DS 最大容量为：

$$1 \times 15 + 15 \times 10 = 165 \text{ trks} = 140 \text{ MB}$$

另外，标识 PDS 的参数 Directory Block——即 SPACE=(TRK,(15,10,1))中的数值 1 的位置——表示这个 PDS 能容纳的 member 数，当然 1 不表示只能容纳一个 member，而是存在着一种倍数关系，达到上限则不能再其中新建 member 了。

再另外，如果需要批量新建 DS 的话，可以这样：

```

//SAM00801 JOB  ,'SAM',CLASS=H,MSGCLASS=K,MSGLEVEL=(1,1),NOTIFY=SAM008
//ALLOC  PROC CUSTNUM=
                                ↑ 这是一个 PROC
                                ↑ 这是一个变量
//STEP0  EXEC PGM=IEFBR14
//SYSPRINT DD SYSOUT=*
//DD1    DD UNIT=3390,VOL=SER=MIL001,DISP=(NEW,CATLG),
//        SPACE=(TRK,(30,30,10)),
//        DSN=EDU.TEST.&CUSTNUM,
                                ↑ 变量在这里
//        DCB=(LRECL=133,BLKSIZE=27930,RECFM=FBA,DSORG=PO)
//ALLOC  PEND
                                ↑ 表示 In-stream PROC 终结
//*
//S01    EXEC ALLOC,CUSTNUM=STUD1001
//S02    EXEC ALLOC,CUSTNUM=STUD1002

```

```

    ↑ 每个 STEP 都执行该 PROC, 即新建一个 DS
        ↑ 根据传递变量的不同, 新建的 DS 的名字均不同
//S03 EXEC ALLOC,CUSTNUM=STUD1003
        ↑ 这个建出来的就是 EDU.TEST.STUD1003
//S04 EXEC ALLOC,CUSTNUM=STUD1004
//S05 EXEC ALLOC,CUSTNUM=STUD1005
/*
//

```

JCL3-1-2

如果你们的存储是 SMS 自动管理的话, 那么在你创建 DS 的时候便不用指定 VOL 和 UNIT, 因为 SMS 会根据 HLQ 自动分配卷; 如果你在 SMS 管理的系统上想创建 NON-SMS 的 DS 的话, 可参考下面的 JCL:

```

//SAM00801 JOB , 'SAM', CLASS=H, MSGCLASS=K, MSGLEVEL=(1,1), NOTIFY=SAM008
//STEP0 EXEC PGM=IEFBR14
//SYSPRINT DD SYSOUT=*
//DD1 DD UNIT=3390, VOL=SER=MIL001, DISP=(NEW, CATLG),
// SPACE=(TRK, (30, 30, 10)),
// STORCLAS=NONSMS,
        ↑ 这个参数配合 UNIT 和 VOL 使用
// DSN=EDU.TEST,
// DCB=(LRECL=133, BLKSIZE=27930, RECFM=FBA, DSORG=PO)

```

JCL3-1-3

3.1.2 万恶淫为首——Copy DS

拷贝 DS 可以分为 PS 和 PDS 两种情况, 也将用到 IEBCOPY 和 IEBGENER 两个 Utility。

拷贝 PS 的例子:

```

//COPYPS JOB , 'BILLRAIN', CLASS=A, NOTIFY=&SYSUID
//STEP1 EXEC PGM=IEBGENER
        ↑ 用这个程序
//SYSPRINT DD SYSOUT=A
//SYSIN DD DUMMY
//SYSUT1 DD DSNAME=DSE003.Z19A.DATASETS, DISP=SHR,
        ↑ 这里指定要拷贝的目标 DS
// UNIT=SYSDA, VOL=SER=Z19SP2

```

```
//SYSUT2 DD DSN=DSE003.Z19B.DATASETS,DISP=(NEW,CATLG),
↑ 那么在这里就指定拷贝到的DS了          ↑ 表示这个B是新建的
//          UNIT=SYSDA,VOL=SER=COMUSR,
//          LIKE=DSE003.Z19A.DATASETS
↑ 这个关键字很方便,表示新建的B的参数是与A类似的,就不用具体指定大小了
```

JCL3-2-1

拷贝 PDS 的例子:

```
//COPYPDS JOB , 'BILLRAIN',CLASS=A,MSGCLASS=H,
//          MSGLEVEL=(1,1),NOTIFY=DSE003,USER=GOD
↑ 因为这次我们拷贝系统文件,而
↑ 这些文件通常是受保护的,需要用高权限的ID来提交,所以指定USER
↑ 因为是用GOD提交的,不指定自己的话会把RC返回给
GOD,阿门
/** USED TO COPY PDS MEMBER *****
//STEP      EXEC PGM=IEBCOPY
↑ 拷贝MEMBER要用这个
//IN1       DD DSN=SYS1.PROCLIB,DISP=SHR
↑ 这个DD名是可以自己改的
//OUT1      DD DSN=SYS1.TEST.PROCLIB,DISP=(NEW,CATLG),
↑ 这个也是
//          LIKE=SYS1.PROCLIB,VOL=SER=SYSLIB
//SYSIN     DD *
↑ SYSIN里面都是用来写控制语句的(control statement)
↑ *号为流内数据的标识
COPY INDD=IN1,OUTDD=OUT1
↑ 这里与上面的DD名是映射关系
SELECT MEMBER=(JES2,@ES2)
↑ 这里指定要拷贝的Member的名字,括号内写2个表示拷
贝后把JES2重命名为@ES2
/**          SELECT MEMBER=JES2
↑ 而如果直接这样写的话就没有重命名
/**          SELECT MEMBER=(MILAN,JUVENTUS)
↑ 拷贝多个Members的时候这样写
//SYSPRINT DD SYSOUT=*
//
```

JCL3-2-2

IEBGENER说自己很强大,拷贝PDS它也可以:

```
//COPYPDS2 JOB , 'BILLRAIN', CLASS=A, NOTIFY=&SYSUID, MSGCLASS=H
//STEP1 EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=A
//SYSIN DD DUMMY
//SYSUT1 DD DSNAME=DSE003.TEST.JCL(AAREADME), DISP=SHR
//SYSUT2 DD DSNAME=DSE003.PROD.JCL(AAREADME), DISP=SHR
//
```

JCL3-2-3

注意，当从一个 PDS 拷贝到同一个 PDS 的时候，应该选用 IEBGENER，而不是 IEBCOPY，因为 IEBCOPY 会执行压缩操作，比如：

```
//COMPRES JOB , 'BILLRAIN', CLASS=A, NOTIFY=&SYSUID, MSGCLASS=H
//STEP1 EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DSNAME=DSE003.TEST.JCL(AAREADME), DISP=OLD
//SYSIN DD *
        COPY INDD=SYSUT1, OUTDD=SYSUT1
/*
```

JCL3-2-4

3.1.3 勿以恶小而为之——Move DS

移动 DS 好像并不是很常用，因为从安全的角度来讲拷贝更常用一些，但是我们仍然可以把一个 DS 从一个卷转移到另一个卷上：

```
//MOVEDS JOB , 'BILLRAIN', CLASS=A, MSGCLASS=H, NOTIFY=&SYSUID
//STEP1 EXEC PGM=IEHMOVE
        ↑ 用这个
//SYSPRINT DD SYSOUT=*
//DD1 DD UNIT=SYSDA, VOL=SER=DSEDSY, DISP=OLD
//DD2 DD UNIT=SYSDA, VOL=SER=COMYYA, DISP=OLD
        ↑ 这里把所有涉及到的卷都指定一下，DD 名随便
//SYSUT1 DD UNIT=SYSDA, VOL=SER=COMYYA, DISP=OLD
        ↑ 这个用来临时存放，DD 名不能改
//SYSIN DD *
        MOVE DSNAME=DSE003.TEST.INPUT, TO=SYSDA=COMYYA, FROM=SYSDA=DSEDSY
        ↑ DS名          ↑ 未来位置          ↑ 现在位置
/*
```

JCL3-3-1

3.1.4 勿以善小而不为——Rename DS

通常我们从测试环境向生产环境更新程序的时候，可以把原有程序重命名以备不时之需，用 IEHPROGM：

```
//RENPD2 JOB , 'BILLRAIN', CLASS=A, MSGCLASS=H, NOTIFY=&SYSUID
//STEP1 EXEC PGM=IEHPROGM
//SYSPRINT DD SYSOUT=*
//DD1 DD UNIT=SYSDA, VOL=SER=COMYYB, DISP=OLD
    ↑ 名字随便起，但是这个 DD 语句是必须的，用来指定你下面要处理的 DS 所处的 VOL 位置
//SYSIN DD *
-----+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7--
RENAME MEMBER=IEASYSAA, DSNAME=AAAA.TEST.PARMLIB, X
        ↑ 要重命名的 Member
        VOL=3390=COMYYB, NEWNAME=@EASYSAA
                ↑ 新的名字
RENAME MEMBER=IEASYSBB, DSNAME=BBBB.TEST.PARMLIB, X
        VOL=3390=COMYYB, NEWNAME=@EASYSBB
        ↑ 这里的 VOL 应该是与上面的 DD 指定的一样的
RENAME DSNAME=TEST.OLD, VOL=3390=COMYYB, NEWNAME=TEST.NEW
    ↑ 重命名 PS 的话也很简单
/*
//
```

JCL3-4-1

如何 RENAME 一个 VSAM 呢？

```
//RENAME JOB , 'BILLRAIN', CLASS=A, MSGCLASS=H, NOTIFY=&SYSUID,
//
MSGLEVEL=(1,1)
//STEP1 EXEC PGM=IDcams
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
ALTER -
    DSE003.TEST.VSAM -
    NEWNAME (DSE003.TEST.NEW.VSAM)
ALTER -
    DSE003.TEST.VSAM.* -
    NEWNAME (DSE003.TEST.NEW.VSAM.*)
/*
```

JCL3-4-2

如果简单重命名 DS 的话，用 IDCAMS 也可以，不解释了：

```
//RENAME JOB , 'BILLRAIN', CLASS=A, MSGCLASS=H, NOTIFY=&SYSUID,
//
//          MSGLEVEL=(1,1)
//STEP1 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
ALTER -
      DSE003.TEST.OLD -
      NEWNAME (DSE003.TEST.NEW)
/*
```

JCL3-4-3

3.1.5 己所不欲 勿施于人——Update DS

用 IEBUPDTE 可以实现对 DS 里面记录的机械化批量修改，至于为什么不手动去修改里面的内容，有时候是出于安全权限方面的考虑，用 JCL 改的确比较麻烦：

```
//UPDATDS JOB , 'BILLRAIN', CLASS=A, MSGCLASS=H,
//
//          MSGLEVEL=(1,1), USER=GOD, NOTIFY=DSE003
//
//          ↑ 帝哥提交的
//STEP1B EXEC PGM=IEBUPDTE, REGION=30K, PARM=MOD
//SYSUT1 DD DISP=OLD, DSN=DSE003.TEST.REXX
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
./ CHANGE LIST=ALL, NAME=INPUT, UPDATE=INPLACE
./ NUMBER SEQ1=ALL, NEW1=1, INCR=1
//
//          ↑ 参数稍显复杂，推荐阅读手册 DFMSDfP Utilities 获得详细说明
BILLRAIN.COM IBM MAINFRAME
./ ENDUP
/*
```

JCL3-5-1

如何大批量修改若干个 DS 里面相同的字符串呢？我们可以用 IPOUPDTE 这个 IBM ServerPac 安装即带的程序，但是在使用这个程序之前，必须要往目标 DS 即你想要更新的 DS 里面插入一个名 \$\$\$COIBM 的 member，这个 member 内容可以为任何值，也可以为空，但必须保证每个你想要更新的 DS 里面都有这个 member 存在。之后才可以使用下面的 JCL 来进行大批量替换：

```
//IPOUPDTE EXEC PGM=IPOUPDTE, PARM=UPDATE
```

```

↑ 程序名
↑ 参数可为 CHECK 或 UPDATE，顾名思义，CHECK 并不做实际替换，而 UPDATE 则完成实际动作
//SYSPRINT DD SYSOUT=*
//@LIB01 DD DISP=SHR,DSN=DSE003.COBOL.SRCLIB
↑ 没跟你开玩笑，要更新的 DD 名字必须以@开始
//@LIB02 DD DISP=SHR,DSN=DSE004.COBOL.SRCLIB
//@LIB03 DD DISP=SHR,DSN=DSE005.COBOL.SRCLIB
//SYSIN DD *
COBOL.V3R0M0<COBOL.V4R1M0<
↑ 要替换的字符串
↑ 新的字符串
DB2.V8R2M0<DB2.V9R1M0<
↑ 可一次性替换无数 DS 里面的无数字符串
/*

```

JCL3-5-2

3.1.6 夫欲立 则先立人——Delete DS

删除一个 PS 或者 PDS 的 Member 跟重命名一样都用 IEHPROGM，只不过控制语言的写法不一样而已：

```

//DELETE01 JOB , 'BILLRAIN', CLASS=A, MSGCLASS=H, NOTIFY=&SYSUID
//STEP1 EXEC PGM=IEHPROGM
//SYSPRINT DD SYSOUT=*
//DD1 DD UNIT=SYSDA, VOL=SER=COMYYB, DISP=OLD
//DD2 DD UNIT=SYSDA, VOL=SER=COMYYA, DISP=OLD
↑ 名字随便起，但是这个 DD 语句是必须的，用来指定你下面要处理的 DS 所处的 VOL 位置
//SYSIN DD *
-----1-----2-----3-----4-----5-----6-----7--
SCRATCH MEMBER=@PXPRMBK, DSNAME=SYS1.TEST.PARMLIB, X
↑ 用这个关键字 ↑ 指定要删除的 Member
VOL=SYSDA=COMYYB
↑ 所处的卷
SCRATCH DSNAME=TEST.PROCLIB, VOL=SYSDA=COMYYA
↑ 删除一个 PS
UNCATLG DSNAME=TEST.PROCLIB
↑ 并且取消它的编目
/*

```

JCL3-6-1

与改名一样，删除整个 PS 或 PDS 的时候也可以选择强大的 IDCAMS：

```
//DELETE JOB , 'BILLRAIN', CLASS=A, MSGCLASS=H, NOTIFY=&SYSUID
//STEP1 EXEC PGM=IDCAMS
    ↑ 删除已编目的 DS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
    DELETE -
        DSE003.TEST.LIST.* -
        ↑ 注意通配符*的使用，我个人不建议使用除非是用来删除 GDG 的子集
    PURGE -
        CATALOG(UCAT.DSE000)
    DELETE -
        DSE003.TEST.*.PDSE -
    PURGE -
        CATALOG(UCAT.DSE000)
/*
/*
//STEP2 EXEC PGM=IDCAMS
    ↑ 删除未编目的 DS
//DELF DD VOL=SER=DSE000, UNIT=3390, DISP=OLD
    ↑ 用来指定要删除 DS 的位置
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
    DEL DSE003.RACF.TEMP FILE(DELF) NVR
    DEL DSE003.RACF.TEMP2 FILE(DELF) NVR
        ↑ DD 名
/*
```

JCL3-6-2

3.1.7 穷则独善其身——Dump DS

有的时候我们需要把很多甚至无数个 DS 从一个卷导出到另外一个卷上，这时候就可以用所谓 DUMP 这个概念，就像在 PC 上对文件夹打 ZIP 一样，其本质不过是把所选的若干 DS 通过某种转换打成一个包，然后放到另一个卷上，然后在通过下面的 RESTORE 在新卷上释放出来。这通常是批量备份和还原 DS 的过程：

```
//DUMPDS JOB , 'BILLRAIN', NOTIFY=&SYSUID, CLASS=A, MSGCLASS=H
```

```

//STEP1 EXEC PGM=ADRDSSU
           ↑ 这是 DFSMSdss 最常用的一个程序
//DASD1 DD UNIT=3390,VOL=SER=COMSPL,DISP=OLD
           ↑ 这里指定目标 DS 所处的卷
//DASD2 DD DSN=DSE003.TEST.DUMP,
           ↑ 那这个就是输出到的位置了
//          UNIT=3390,VOL=SER=COMBBK,DISP=(NEW,CATLG,DELETE),
//          SPACE=(TRK,(10,10),RLSE)
           ↑ 要自己根据 DS 的多少估算分配的大小,多余的自动释放掉
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DUMP -
  ↑ 控制语句开始了
INDDNAME(DASD1) OUTDDNAME(DASD2) -
  ↑ 固定的语法很好理解吧,刮弧内的都是对应的 DD 名
DATASET(INCLUDE(DSE003.**)) -
  ↑ 这里表示把所有 DSE003 开头的都 DUMP 了
COMPRESS
  ↑ 表示打包过程中压缩
/*
//

```

JCL3-7-1

3.1.8 达则兼济天下 ——Restore DS

在还原的时候:

```

//RESTDS JOB , 'BILLRAIN', NOTIFY=&SYSUID, CLASS=A, MSGCLASS=H
//STEP1 EXEC PGM=ADRDSSU
           ↑ 还原的话当然也是用这个的了
//SYSPRINT DD SYSOUT=*
//SOURCE DD UNIT=3390,DISP=OLD,DSN=DSE003.TEST.DUMP
           ↑ 我们上面打好的包包
//TARGET DD UNIT=3390,VOL=SER=COMSPL,DISP=OLD
           ↑ 这是我们的原盘
//SYSIN DD *
RESTORE INDDNAME(SOURCE) OUTDDNAME(TARGET) -
DATASET(INCLUDE(DSE003.**)) REPLACE
           ↑ 这样就原封不动的给还原回去了,当然是给覆盖掉了,很随吧

```

使用 **REPLACE** 请加倍注意，不要覆盖其他卷上的同名文件

```
/*
//
```

JCL3-8-1

我们说过通常这是备份与还原的一个过程，其实也可以用来从卷 1 拷贝到卷 3，中间需要用卷 2 做一下中转，因为 SOURCE 和 TARGET 应该是不能为同一个卷的。

3.1.9 修身 齐家 平天下——Compare DS

我们知道 ISPF 的 3.14 的 SUPERC 可以用来比较两个 DS 的内容，同样用 DFSMSfdp 的 Utility 也可以：

```
//COMPDS JOB , 'BILLRAIN', CLASS=A, MSGCLASS=H, NOTIFY=&SYSUID
/*=====
/*THIS JCL IS USED TO COMPARE TWO PDS MEMBERS
//STEP1 EXEC PGM=IEBCOMPR
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DSN=DSE003.TEST.INPUT, DISP=SHR
//SYSUT2 DD DSN=DSE003.TEST.INPUT2, DISP=SHR
    ↑ 上面分别给出了需要比较的两个 DS，可以是 PS 也可以是 PDS，但是 Member 不可以
//SYSIN DD *
    COMPARE TYPORG=PO
    ↑ 表示比较的是 PDS，默认的话是 PS
/*
//
```

JCL3-9-1

与 SUPERC 不同的是，IEB 这个一般只用来比较两个 DS 的逻辑长度，而不能提供详细的差异报告，所以通常放在 COPY 后面用来检测副本与目标是否一致，比如：

```
//CPCPDS JOB , 'BILLRAIN', CLASS=A, MSGCLASS=H, NOTIFY=&SYSUID
//STEP1 EXEC PGM=IEBCOPY
    ↑ 复制
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSNAME=MAINDB.LOG.OLDSET, DISP=SHR
//SYSUT2 DD DSNAME=MAINDB.LOG.NEWSET, UNIT=3390, DISP=(, PASS),
//          VOLUME=SER=111113, SPACE=(TRK, (5, 5, 5)),
//          DCB=(RECFM=FB, LRECL=80, BLKSIZE=640)
//SYSUT3 DD UNIT=SYSDA, SPACE=(TRK, (1))
```



```

1 ISRSUPC - MVS/PDF FILE/LINE/WORD/BYTE/SFOR COMPARE UTILITY- ISPF FOR
z/OS      2009/03/03 10.46 PAGE 1
NEW: SYS1.PARMLIB(IEASYSAC)          OLD:
SYS1TEST.PARMLIB(IEASYSAC)

LINE COMPARE SUMMARY AND STATISTICS

      48 NUMBER OF LINE MATCHES          0 TOTAL CHANGES
(PAIRE+NONPAIRE CHNG)
      0 REFORMATTED LINES                0 PAIRE CHANGES (REFM+PAIRE
INS/DEL)
      0 NEW FILE LINE INSERTIONS        0 NON-PAIRE INSERTS
      0 OLD FILE LINE DELETIONS         0 NON-PAIRE DELETES
      48 NEW FILE LINES PROCESSED
      48 OLD FILE LINES PROCESSED

LISTING-TYPE = DELTA      COMPARE-COLUMNS = 1:72      LONGEST-LINE = 80
PROCESS OPTIONS USED: SEQ(DEFAULT)

```

图 3-3

同时，用 ISRSUPC 还可以搜索 DS 中的字符串，比如：

```

//ISRSUPC JOB , 'BILLRAIN', CLASS=A, MSGCLASS=H, NOTIFY=&SYSUID
//SEARCH EXEC PGM=ISRSUPC, PARM= ('SRCHCMP, ANYC')
                                     ↑ 参数不同而已

//SYSPRINT DD SYSOUT=*
//NEWDD DD DISP=SHR, DSN=SYS1.PARMLIB
//OUTDD DD DSN=DSE.TEST.SUPCLOG, DISP=SHR
//SYSIN DD *
      SRCHFOR 'SOMETHING', W, 1:11
      ↑ 要搜索的字符串
      ↑ 起始位置和长度
/*
此 JOB 经常与 IPOUPDTE 配合使用，用于批量替换后的数据验证

```

JCL3-9-4

3.1.10 夫君子者 莫过于此——Compress DS

我们在 3.4 可以对 DS 进行 Compress，用 batch 同样可以：

```

//STEP1 EXEC PGM=IEBCOPY
                                     ↑ 这是它的一个应用

```

```
//SYSPRINT DD SYSOUT=*
//A DD DISP=OLD,DSN=MIB001.LOG1
//C DD DISP=OLD,DSN=MIB001.LOG2
//SYSIN DD *
COPY OUTDD=A,INDD=A
COPY OUTDD=C,INDD=C
    ↑ 指定同样的 IN 和 OUTDD 即为 COMPRESS
/*
//
//STEP2 EXEC PGM=IEBCOPY,PARM=COMPRESS
    ↑ 或者这样也行
//SYSPRINT DD SYSOUT=*
//SYSUT2 DD DISP=OLD,DSN=MIB001.LOG2
//SYSIN DD DUMMY
//
//
//STEP3 EXEC PGM=ADDRSSU
//SYSPRINT DD SYSOUT=*
//IN1 DD DISP=SHR,UNIT=3390,VOL=SER=MIBLOG
//SYSIN DD *
RELEASE DDN(IN1) -
INCLUDE(MIB001.LOG1, -
MIB001.LOG2)
    ↑ 在 COMPRESS 之后可以用上面的步骤彻底释放多余的空间，之后 DS 的空间使用率将变成
100%，一般用于存放 LOG 的 DS，请不要对存放 module 的 DS 乱加使用
/*
```

JCL3-10-1

好了，关于 DS 日常操作的就介绍到这里，相信已经能覆盖你一般的工作需求了，当然如果你觉得还有什么是很常用的也可以与我联系，我会补充进来。

3.2 开发相关

下面介绍一些常用的应用开发方面的 JCL，因为本人不具备丰富的开发经验，纯属业余爱好，所以也只能挑最基本的讲了（以 COBOL 为例），当然我希望你们最好了解源代码从出生到执行需要经过的过程。

3.2.1 编译链接

对于开发人员最常 SUB 的肯定就是下面这种 JCL 了，虽然可能干了好几年他们也不一定知道 IGYWCL 里面具体写的是是什么，因为真的无关紧要，只要我们知道系统库的名字，编译的 JCL 完全可以自己写出来：

```
//COBCPR JOB (DSE),BILLRAIN,CLASS=A,MSGCLASS=H,
//
//          NOTIFY=&SYSUID
//***** COMPLIE AND LINK COBOL PROGRAM
//IBMLIB  JCLLIB  ORDER='SYS1.ADMIN.PROCLIB'
//          ↑ 指定系统程序库，通常会先搜索默认的自带系统库，找不到才会到这里找
//  SET PGM=HELLO
//          ↑ 给变量赋值，即为 COB 程序名
//STEP01  EXEC IGYWCL,PGMLIB=DSE003.TEST.LOAD,GOPGM=&PGM
//          ↑ 生成的 Load Module 的位置
//COBOL.SYSIN DD DSN=DSE003.TEST.COBOL(&PGM),DISP=SHR
//          ↑ COB Source 的位置 ↑ 这是变量
//SYSPRINT DD  DSN=DSE003.TEST.JOBLOG,DISP=SHR
//          ↑ 这里把 LOG 输出到 DS 里面了，纯属吃饱了 o(∩_∩)o
//***** RUN COBOL PROGRAM
//STEP02  EXEC PGM=&PGM
//          ↑ 编译链接后直接执行，即 HELLO
//STEPLIB DD  DSN=DSE003.TEST.LOAD,DISP=SHR
//          ↑ Load 的位置
//SYSOUT  DD  DSN=DSE003.TEST.JOBLOG,DISP=MOD
//          ↑ 依然吃饱了
//SYSPRINT DD  SYSOUT=*
//
```

JCL4-1

不同环境或者不同版本的 COB 的 IGY 可能写法还不太一样，所以上面的 JCL 是针对下面这样的 ENCOBOL V4R1 的 IGYWCL 写的：

```
***** Top of Data *****
//IGYWCL PROC  LNGPRFX='COBOL.V4R1',SYSLBLK=3200,
//          LIBPRFX='CEE',
```

```

//          PGMLIB='&&GOSET',GOPGM=GO
// *
// *****
// *
// * Enterprise COBOL for z/OS
// *          Version 4 Release 1 Modification 0
// *
// * LICENSED MATERIALS - PROPERTY OF IBM.
// *
// * 5655-S71 © COPYRIGHT IBM CORP. 1991, 2007
// * ALL RIGHTS RESERVED
// *
// * US GOVERNMENT USERS RESTRICTED RIGHTS - USE,
// * DUPLICATION OR DISCLOSURE RESTRICTED BY GSA
// * ADP SCHEDULE CONTRACT WITH IBM CORP.
// *
// *****
// *
// * COMPILE AND LINK EDIT A COBOL PROGRAM
// *
// * PARAMETER  DEFAULT VALUE  USAGE
// *  LNGPRFX   COBOL.V4R1      PREFIX FOR LANGUAGE DATA SET NAMES
// *  SYSLBLK   3200             BLOCKSIZE FOR OBJECT DATA SET
// *  LIBPRFX   CEE              PREFIX FOR LIBRARY DATA SET NAMES
// *  PGMLIB    &&GOSET          DATA SET NAME FOR LOAD MODULE
// *  GOPGM     GO               MEMBER NAME FOR LOAD MODULE
// *
// * CALLER MUST SUPPLY //COBOL.SYSIN DD ...
// *          ↑ 已经告诉我们 JCL 怎么写了
// * CALLER MUST ALSO SUPPLY //COBOL.SYSLIB DD ... for COPY statements
// *          ↑ COPYBOOK 这样指定
//COBOL EXEC PGM=IGYCRCTL,REGION=0M
//STEPLIB DD DSNAME=&LNGPRFX..SIGYCOMP,
//          DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSLIN DD DSNAME=&&LOADSET,UNIT=SYSALLDA,
//          DISP=(MOD,PASS),SPACE=(TRK,(3,3)),
//          DCB=(BLKSIZE=&SYSLBLK)

```

```

//SYSUT1 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT2 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT3 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT4 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT5 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT6 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//SYSUT7 DD UNIT=SYSALLDA,SPACE=(CYL,(1,1))
//LKED EXEC PGM=HEWL,COND=(8,LT,COBOL),REGION=0M
//SYSLIB DD DSNAME=&LIBPRFX..SCEELKED,
// DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSLIN DD DSNAME=&&LOADSET,DISP=(OLD,DELETE)
// DD DDNAME=SYSIN
//SYSLMOD DD DSNAME=&PGMLIB(&GOPGM),
// SPACE=(TRK,(10,10,1)),
// UNIT=SYSALLDA,DISP=(MOD,PASS)
//SYSUT1 DD UNIT=SYSALLDA,SPACE=(TRK,(10,10))
***** Bottom of Data *****

```

JCL4-2

综上所述，只要你找到自己环境 COB 的安装路径，在 JCLLIB 那指定一下 IGY 系列的位置，用上面的 JCL 就可以实现编译链接，实在找不到的话把上面的 IGYCWL 考到你自己的 DS 里面，JCLLIB 指定成自己的也木问题的啦~

至于 DB2 的 COBOL 的 BIND、编译链接等等，我现在忘得差不多了，以后想起来再讲吧~

3.2.2 VSAM

Virtual Storage Access Method (VSAM) 是大机上另外一种数据存贮方法，与 PS 和 PDS 最大的差别就在于，VSAM 是支持索引 (index) 的，而 PS 和 PDS 都是顺序查找的，所以 VSAM 在存储方面有着其先天的优势，是 DB2 和 CICS 开发中必不可少的。

VSAM 分为四种：KSDS，ESDS，RRDS，LDS

对于 VSAM 只有下列四种操作需要你掌握：删除、创建、写入、查看，用一个 JCL 就能说的很明白：

```

//VSAMALL JOB (ACCT),AMIT,CLASS=A,MSGCLASS=X,
// NOTIFY=&SYSUID
//*****

```

```

/*          VASM DELETE
/*****
//STEP001 EXEC PGM=IDCAMS
           ↑ 非常强大的 Utility 华丽的出现了
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
    DELETE DSE003.MASTFL.VSAM PURGE
           ↑ 再新建 VSAM 之前我们通常先删除看看有没有以前残留下来的
/*****
/*****
/*          VASM DEFINE
/*****
//STEP002 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
           ↓ VSAM 的结构有 3 块，先来建一个 CLUSTER
    DEFINE CLUSTER          -
      ( NAME (DSE003.MASTFL.VSAM) -
        CYLINDERS (1,1)      -
           ↑ 大小
        RECORDSIZE (76 76)   -
           ↑ 记录长度
        KEYS (06 0)         -
           ↑ 索引的长度为 6 位，后面 0 表示从头开始算起
        FREESPACE (10,20)) -
    DATA (NAME (DSE003.MASTFL.VSAM.DATA) ) -
           ↑ 再建一个 DATA
    INDEX (NAME (DSE003.MASTFL.VSAM.INDEX) )
           ↑ 最后建一个 INDEX
/*
/*****
//*          REPRO DATA    下面来把数据导入到新建的 VSAM 里面
/*****
//STEP003 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//INPUT DD DSN=DSE003.ABCBANK.OUT, DISP=SHR
           ↑ 从这个里面拷
//SYSIN DD *

```

```

REPRO INFILE (INPUT) ODS (DSE003.MASTFL.VSAM)
    ↑ 用这个关键字，如此一般，输出到这里 ↑
/*
//*****
//*          VASM PRINT    因为VSAM文件在ISPF下是不能直接查看的
//*****
//STEP004 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//INPUT DD DSN=DSE003.MASTFL.VSAM,DISP=SHR
//SYSIN DD *
    PRINT INFILE (INPUT) CHAR
    ↑ 所以要这样把内容打印出来查看
//

```

JCL4-3

3.2.3 GDG

Generation Dataset Group (GDG) 是一组具有相同格式、用途和名称的 DS 的集合，因为我们知道大机上 DS 的名称是唯一的，而有的时候我们希望用具有同样名字的 DS 来存放具有相同意义的数据，比如每个月的工资或报告，这样我们就建立一个统一的 GDG，把每个月的数据都存放在一个 DS 里面，像这样的 DS 我们称之为世代，于是有人把 GDG 翻译叫做：世代数据集，我觉得还可以。

下面我们就来看看 GDG 是怎样创建的：

```

//GDGDEF JOB DSE,'BILLRAIN',CLASS=A,MSGCLASS=X,
//      NOTIFY=&SYSUID
//*****
//*          GDG DELETE
//*****
//STEP1 EXEC PGM=IDCAMS
    ↑ 同样用强大的 IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
    DELETE DSE003.TEST.GDG.* PURGE
    ↑ 跟 VSAM 一样我们先删除，先把所有的世代删除掉
    DELETE DSE003.TEST.GDG PURGE
    ↑ 再把整个 GDG 删除掉
//*****
//*          GDG DEFINE (GDG BASE)
//*****

```

```

//STEP2 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
    DEFINE GDG(NAME(DSE003.TEST.GDG) -
        ↑ 要先创建一个 BASE
            LIMIT(12) -
                ↑ 表示最多可以容纳 12 个世代
            NOEMPTY -
            SCRATCH)
//*****
//*          GDG DEFINE (GDG DSCB MODEL)
//*****
//STEP3 EXEC PGM=IEFBR14
        ↑ 接下来创建一个 MODEL 和一个世代
//GDGMODEL DD DSN=DSE003.GDG.MODEL,DISP(,PASS),
        ↑ 所谓的 MODEL 可以理解为, 为世代提供 DCB 模板的标准模型
//          DCB=(RECFM=FB,LRECL=101,BLKSIZE=1010),
        ↑ 也就是说每个世代的参数都是按照 MODEL 的定义来的
//          SPACE=(TRK,(0)),UNIT=3390
//MODEL1 DD DSN=DSE003.TEST.GDG(+01),
        ↑ 建立世代的方法比较特殊,+01 表示追加一个世代
//          DCB=*.GDGMODEL,UNIT=3390,
        ↑ 星星表示引用上面 MODEL 的 DCB 参数
//          SPACE=(TRK,(1)),DISP=(NEW,CATLG)
//SYSPRINT DD SYSOUT=*

```

JCL4-4

上面的提交成功后会得到下面这样的收获:

```

DSE003.TEST.GDG      <-- BASE
DSE003.TEST.GDG.G0001V00 <-- 世代

```

图 4-1

对于世代的操作:

```

//*****
//*          GDG REPRO DATA G0001
//*****
//STEP4 EXEC PGM=IDCAMS
        ↑ 往 GDG 里写入跟 VSAM 一样
//SYSPRINT DD SYSOUT=*

```

```

//INPUT DD DSN=DSE003.TEST.INPUT,DISP=SHR
//OUTPUT DD DSN=DSE003.TEST.GDG.G0001V00,DISP=SHR
//SYSIN DD *
    REPRO INFILE(INPUT) OUTFILE(OUTPUT)
/*
//*****
//*          GDG PRINT DATA TO SYSOUT
//*****
//STEP5 EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DSN=DSE003.TEST.GDG.G0001V00,DISP=SHR
        ↑ GDG 的世代是可以正常访问的
//SYSUT2 DD SYSOUT=*
//SYSIN DD DUMMY
//
//STEP6 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
    ALTER DSE003.TEST.GDG LIMIT(12)
        ↑ 另外, 可以这个命令来更新 GDG 的代数, 即修即改, 非常方便, 但改小时请自行考虑

```

JCL4-5

3.2.4 SORT

最后讲一下怎样用 JCL 对数据记录进行排序, 并对上面所有内容进行一下整合:

```

//DSESORT JOB (ACCT),BI11,CLASS=A,MSGCLASS=X,
//
//    NOTIFY=&SYSUID
//*****
//*          SORT INPUT FILE TO MAKE RECORDS SEQUENTIAL
//*****
//STEP01 EXEC PGM=ICEMAN
        ↑ 排序的程序, 很有个性吧, 有的用 DFSORT 也可以, 如果在未来你有机
        会用到一个叫 DSEMAN 的非常强大程序, 那一定是我写的~
//SYSOUT DD SYSOUT=*
//SORTIN DD DSN=DSE003.ABCBANK.IN,DISP=SHR
        ↑ 这里指定进行排序的输入文件, 这个 DD 名一般是不能乱改的
//SORTOUT DD DSN=&&DSE003,DISP=(MOD,PASS),
        ↑ 排序后的输出文件
        ↑ 两个&表示这是一个临时 DS, 这在当前的 JOB 内使用, 并不需要实际存放

```

```

                                ↑ PASS 表示留给下个 step 使用
//          UNIT=SYSDA,SPACE=(TRK,(2,2)),
           ↑ 虽然是临时的，参数还是要给人家配的，别拿豆包不当干粮
//          BLKSIZE=370,RECFM=FB,LRECL=37
//SORTWK01 DD SPACE=(TRK,(1,1)),UNIT=SYSDA
//SORTWK02 DD SPACE=(TRK,(1,1)),UNIT=SYSDA
//SORTWK03 DD SPACE=(TRK,(1,1)),UNIT=SYSDA
           ↑ 这几个是给 SORT 程序做缓冲区的，不占地方
//SYSIN DD *
          SORT FIELDS=(1,6,CH,A)
           ↑ 排序字段的起始位置
           ↑ 字段长度
           ↑ 表示按照字母顺序排序
           ↑ A 为升序，D 为降序
/*
//*****
//*          RUN PROGRAM ABCPGM
//*****
//STEP02   EXEC PGM=ABCPGM,PARM='20080101'
           ↑ 下面执行这个 COB 程序
//STEPLIB DD   DSN=DSE003.TEST.LOAD,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//MYIN     DD   DSN=&&DSE003,DISP=(OLD,DELETE)
           ↑ DD 名与 COB 里面对应 ↑ 这个临时 DS 作为输入 ↑ 过河拆桥，用完就删
//MYOUT    DD   DSN=DSE003.MASTFL.VSAM,DISP=SHR
           ↑ COB 程序把排好序的记录输出到 VSAM 里面
//MYERR    DD   DSN=DSE003.ABCBANK.ERR,DISP=SHR
//INFILE   DD   DSN=DSE003.MASTFL.VSAM,DISP=SHR
           ↑ 然后再把 VSAM 当做输入文件
//OUTFILE  DD   DSN=DSE003.TEST.GDG(0),DISP=SHR
           ↑ 处理后写到 GDG 里面，0 表示写到当前最新的世代
//*UTFILE  DD   DSN=DSE003.TEST.GDG(+1),
           ↑ +1 则表示新建一个世代，-1 则表示为上一个世代
//*          SPACE=(TRK,(1)),DISP=(NEW,CATLG)
//*****
//*          VASM PRINT
//*****

```

```
//STEP03 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//INPUT DD DSN=DSE003.MASTFL.VSAM,DISP=SHR
//SYSIN DD *
PRINT INFILE(INPUT) CHAR
    ↑ 显示一下 VSAM 的结果
//
```

JCL4-6

用 COBOL 也可以实现排序的过程，性能以及功能上的差异还是会有
的，COBOL 的书我是没兴趣写了，大家自己研究吧，不过我可以推荐一本书——
Murach's Structured COBOL，这本书看明白了 COBOL 基本精通了。

3.3 系统相关

哎讲到这终于到我的老本行了，泪奔啊~O(∩_∩)O~，当然我估计在国内有机会有条件有环境有能力有勇气有权利做下面这些操作的也够呛能凑够踢一场球的，97%的你们可以 bypass 这些内容，因为你们看也看不懂，系统知识我在这里是不会具体讲的（未来十一年内另有书目介绍），一切权当我自娱自乐了~o(′□′)o

3.3.1 格式化磁盘

DASD 初始化（Initialize）是指把磁盘上的内容全部删除并且重新分配一个卷标，这是一项危险性较高的工作，不小心把客户的数据弄没了你基本可以回家打酱油了，所以一定要仔细：

```
//DASDINIT JOB CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),NOTIFY=&SYSUID
/* VERIFY -> Present VOLSER
/* VOLID -> New VOLSER
/* Addr -> 9700-9703
//STEP01 EXEC PGM=ICKDSF,PARM='NOREPLY'
                                     ↑ 加此参数可以省去回复系统信息的操作
//CHKDS DD DISP=SHR,DSN=UCAT.BILLRAIN
  ↑ 验证是否在要求的系统上作业
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
  INIT UNIT(9700) VFY(COMYYB) VOLID(BR9700) PRG NVAL VTOC(0,1,30)
                                     ↑ 格式化磁盘物理地址
                                     ↑ 验证要格式化的磁盘的卷标，与地址是一一对应的
  INIT UNIT(9701) VFY(DB2SPL) VOLID(BR9701) PRG NVAL VTOC(0,1,30)
                                     ↑ 格式化后新的卷标
  INIT UNIT(9702) VFY(*NONE*) VOLID(BR9702) PRG NVAL VTOC(0,1,30)
                                     ↑ 如果这是一块处女盘，这样指定
  INIT UNIT(9703) VFY(*NONE*) VOLID(BR9703) PRG NVAL VTOC(0,1,30) SG
                                     如果这块卷是 SMS 的，请加 SG ↑
/*
```

JCL5-1

3.3.2 磁盘拷贝

一块磁盘在初始化之后才能进行拷贝：

```
//DASDCOPY JOB DSE,'DASD COPY',
// MSGCLASS=H,REGION=0M,NOTIFY=&SYSUID
//*=====
```

```

↓ 这是一个 PROC , In-stream PROC
//COPY   PROC INVOL=,OUTVOL=
           ↑ 这两个是变量, 从外面传进来
//DASDCP EXEC PGM=ADRDSSU
           ↑ Storage 方面最强程序
//CHKDS  DD DISP=SHR,DSN=UCAT.BILLRAIN
//SYSPRINT DD SYSOUT=*
//INDISK DD UNIT=SYSDA,DISP=SHR,VOL=SER=&INVOL
//OUTDISK DD UNIT=SYSDA,DISP=OLD,VOL=SER=&OUTVOL
           ↑ 拷贝的目标和终点
//SYSIN  DD DISP=SHR,DSN=DSE003.TEST.JCL(DDCPBK01)
           ↑ 这里把控制信息写到 DS 里面了, 因为 PROC 内是不能写 In-stream SYSIN 的
//
           ↑ 表示流内 PROC 结束
//*=====
//COPY01 EXEC COPY,INVOL=COMBK1,OUTVOL=BR9700
//COPY02 EXEC COPY,INVOL=COMBK2,OUTVOL=BR9701
//COPY03 EXEC COPY,INVOL=COMBK3,OUTVOL=BR9702
//COPY04 EXEC COPY,INVOL=COMBK4,OUTVOL=BR9703
           ↑ 执行这个 PROC, 把参数传进去, 一个是目标磁盘卷, 一个是新的磁盘卷

```

JCL5-2

在 copybook 里面放着这样的控制语句（此乃 physical DUMP）：

```

COPY -
FULL -
INDDNAME (INDISK) -
OUTDDNAME (OUTDISK) -
COPYVOLID -
ALLDATA (*) -
ALLEXCP -
CANCELError

```

JCL5-3

当磁盘拷贝成功后，新磁盘的卷标会自动更改为目标盘的卷标，因为在 MVS 系统上不能同时存在两块卷标一样的磁盘，所以新盘卷会自动 OFFLINE:

```

97 ADR369D AUTHORIZE FOR WRITE ACCESS A VTIOCIX DATA SET ON BR9700,
DASDCOPY, DASDCOPY, REPLY U OR T
R 97,U          ← 每一步你都需要回系统信息才能继续
IEE600I REPLY TO 97 IS;U

```

```

ADR320I (001)-SBRTN(01), VOLUME SERIAL BR9700 ON UNIT 9700 IS CHANGED
774
TO COMBK1
VARY 9700,OFFLINE DFDSS INTERNAL VARY
IEF524I 9700, VOLUME BR9700 PENDING OFFLINE
ADR344I (001)-SBRTN(01), VOLSER ON UCB 9700 IS A DUPLICATE. VOLUME MADE
777                                ↑ 重名了
UNAVAILABLE.
IEF281I 9700 NOW OFFLINE
                                ↑ 过一会就下线了, 并且 9700 的卷标已经由 BR9700 变成了拷贝盘的 COMBK1

```

图 5-1

3.3.3 更改磁盘卷标

有的时候我们只想改变一块磁盘的卷标而不格式化它的内容:

```

//DASDREFM JOB DSE,'BILLRAIN',CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1),
//          NOTIFY=&SYSUID
//*-----*
//*  CHANGE VOLSER NUMBER                               *
//*-----*
//* VERIFY -> PRESENT VOLSER
//* VOLID  -> NEW VOLSER(XXXXXX)
//STEP01   EXEC PGM=ICKDSF,PARM='NOREPLYU'
//CHKDS    DD DISP=SHR,DSN=UCAT.BILLRAIN
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
          REFORMAT UNITADDRESS(9700) VERIFY(COMBK1) VOLID(BR9700)
          ↑ 这个叫“格式化”   ↑ 物理地址
          REFORMAT UNITADDRESS(9701) VERIFY(COMBK2) VOLID(BR9701)
                                   ↑ 验证当前卷标       ↑ 改变后的新卷标
          REFORMAT UNITADDRESS(9702) VERIFY(COMBK3) VOLID(BR9702)
          REFORMAT UNITADDRESS(9703) VERIFY(COMBK4) VOLID(BR9703)
/*

```

JCL5-4

3.3.4 定义/删除别名 (Alias)

在处理 CATALOG 的时候, 我们有时要在 MACT 或 UCAT 里面为特定的字段定义别名:

```

//DEFALIS JOB CLASS=A,MSGCLASS=H,NOTIFY=&SYSUID,MSGLEVEL=(1,1)

```

```

//STEP1 EXEC PGM=IDCAMS
          ↑ CATALOG 实际上也是 VSAM，所以也用这个
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
    DEFINE ALIAS (NAME (DSE.BANKABC) -
                ↑ 你要新建的别名
    RELATE (UCAT.DSEVOL) CATALOG (MCAT.DSESYS)
                ↑ 别名与哪个 UCAT 关联          ↑ 在哪个 MCAT 存放
/*
//
//SYSIN DD *
    DELETE DSE.BANKABC ALIAS -
          ↑ 同样删除没用的别名也是类似的
    CATALOG (MCAT.DSESYS)
/*

```

JCL5-5

3.3.5 导入/备份 User Catalog

当我们在测试系统（或者叫 Driver System）上制作了一块产品卷的 User Catalog，当要把这个卷连接到本番系统（生产系统）上之后，要把这个卷上的编目信息导入到本番系统的 Master Catalog 里面才能识别：

```

//CATAIMPT JOB CLASS=A,MSGCLASS=H,NOTIFY=&SYSUID,MSGLEVEL=(1,1)
//STEP1 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
    IMPORT CONNECT OBJECTS ((UCAT.DSEPP1 -
                            ↑ 导入的 UCAT 名
    VOLUME (DSEPP1) DEVICETYPE (3390)) -
        ↑ UCAT 所处的卷
    CATALOG (MCAT.DSESYS)
        ↑ MCAT 名
/*
//

```

JCL5-6

对于生产系统 User Catalog 的日常备份，可以参考：

```

//STEP2 EXEC PGM=IDCAMS
//DD1 DD DISP=(,CATLG),DSN=DSE003.TEST.GDG(+1),
// UNIT=CART,RECFM=VBS,LRECL=32404,BLKSIZE=2048

```

```
↑ 一般都是备份到 TAPE 的 GDG 上
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
EXPORT UCAT.DSEPP1 OUTFILE(DD1) TEMPORARY
↑ 要有这个
/*
```

JCL5-7

3.3.6 执行 TSO/RACF 命令

我们可以在 ISPF=6 的 command line 执行 TSO 或者 RACF 的命令，JCL 也提供了接口可以在 batch 完成相同的功能，而且保留了 LOG：

```
//TEMP JOB , 'BILLRAIN', CLASS=A, MSGCLASS=H, NOTIFY=&SYSUID
//STEP01 EXEC PGM=IKJEFT01
↑ 这个就是接口程序，相当于 web 的 API
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSTSIN DD *
OUT DSE003(JOB33241) PR('DSE003.TEST.TEMP(MVSMAIL)') HOLD KEEP
↑ TSO 命令，用来把 SDSF 的 joblog 写入到 DS 中
OPUT 'DSE003.KOEIO6.PAX.ZBI' '/usr/loacl' binary
↑ TSO 命令，解压缩 UNIX 文件
LISTDSD ALL DATASET('DSE003.TEST.*') GENERIC
↑ RACF 命令，显示 DS profile 内容
//
```

JCL5-8

3.3.7 创建 RACF TSO ID

在上述基础上，我们可以执行一些 RACF 的命令来创建 TSO ID，GROUP，DS Profile 和一般资源，或赋予权限，比如：

```
//SYSTSIN DD *
ADDUSER MIB123 PASSWORD(BILLGATE) NAME('MIB ADMIN') +
DFLTGRP(SYS1) SPECIAL OPERATION NOGRPACC +
OWNER(SYS1) UACC(NONE) AUTHORITY(USE) +
TSO(ACCTNUM(ACCT#) PROC(IKJACCNT) SIZE(8192) UNIT(SYSDA))
↑ RACF 命令，用于创建拥有至高权力的管理员 ID
PERMIT IKJACCNT CLASS(TSOPROC) GENERIC ID(MIB123) ACCESS(READ)
↑ RACF 命令，用于赋予 ID MIB123 使用 LOGON PROC=IKJACCNT 的权限
```

```

SETROPTS REFRESH RACLIST(TSOPROC)
↑ 用来刷新 RACF 数据库在内存中的记录
ADDGROUP MIBSYS OWNER(IBMUSER) SUPGROUP(SYS1) OMVS(GID(001))
CONNECT MIB123 AUTHORITY(USE) GROUP(MIBSYS) GRPACC
↑ RACF GROUP 的概念很好, 要多加利用
RDEFINE FACILITY GIM.* UACC(NONE)
PERMIT GIM.* CLASS(FACILITY) ID(MIBSYS) ACCESS(READ)
↑ 创建一个一般资源保护一下 SMPE 的 PGM 吧, 其实不是我想保护, 没保护默认的是不让用啊
SETROPTS RACLIST(FACILITY) REFRESH
//

```

JCL5-9

特别强调, 此 JOB 执行无论成功与否, RC 永远为 00 (低级错误除外), 也就是说 IKJ 的程序是不会对调用的 RACF 命令的结果进行判断的, 所以, 切记一定要察看 **JOBLOG** 以确认结果!

3.3.8 提交 REXX/CLIST 程序

用上面同样的程序, 我们可以在 batch 提交 REXX 或 CLIST 程序:

```

//TSOBAT EXEC PGM=IKJEFT01,DYNAMNBR=1000
//*****
//*** TO EXEC REXX PGM IN BATCH ***
//*****
//SYSTSPRT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSEXEC DD DISP=SHR,DSN=DSE003.REXX.SRCLIB
↑ REXX 程序所在 DS 在这里指定
//*SYSPROC DD DISP=SHR,DSN=DSE003.CLIST.SRCLIB
↑ 如果是 CLIST 请用这个 DD 名
//*TEPLIB DD DISP=SHR,DSN=LOAD-LIB
//SYSTSIN DD *
SETPROFB
↑ 要执行的 REXX 或 CLIST 程序名
/*

```

JCL5-10

3.3.9 DS 压缩与解压缩

与 PC 上的 ZIP 类似, 大型机上也有把文件压缩成 BIN 的方法, 一般用于保

存系统 DUMP 文件或传输 Module Lib，压缩后的文件我们有特殊的叫法——TERSE 文件，其格式固定，FTP 上传下载时请用二进制 BIN 进行，在 PC 上保存的 TERSE 文件不用再 ZIP 压缩，已无压缩余地。

```
//STEP1 EXEC PGM=TRSMAIN, PARM='PACK'
      ↑ 用这个程序
      ↑ 压缩用 PACK，反之解压缩用 UNPACK 即可
//SYSPRINT DD SYSOUT=*
//INFILE DD DISP=SHR, DSN=MIB001.DMP00123
      ↑ 要压缩的 DD，解压缩是与下面呼唤 DSN 即可
//OUTFILE DD DISP=(NEW,CATLG), DSN=MIB001.DMP00123.TERSE,
//*          VOL=SER=SYS001, UNIT=3390,
//          SPACE=(CYL,(30,10),RLSE)
      ↑ 节省空间，释放多余
```

JCL5-11

系统相关方面暂且介绍这么多，更多的内容将在更专业的系统配置书籍中讲解。

3.4 SMPE 相关

本章仅限系统程序员阅读，SMPE 同 RACF/ACF2 一样作为 SP 行走将会必须掌握的两种技能之一，不解释，你懂的。

嗯，还是简单解释一下。SMPE 是在 MF 上安装操作系统、软件，打补丁必不可少的系统工具，前两种情况都会自带写好的 SMPE JOB，使用最多的情况就是用来对 OS 或产品（PP）打补丁。MF 上的补丁我们一般称作 PTF，APAR 或 FIX。关于 SMPE 的更多内容，请于 MIB 官网参考《[DSE102 SMPE 基本指南](#)》。

3.4.1 Receive

当你从网上把 FIX (PTF/APAR/USERMOD)或 HOLDDATA 下载后，上传到各个产品或 OS 的 PTF LIB 里面，然后开始 SMPE 的第一步。

```
//RECEIVE EXEC PGM=GIMSMP,REGION=6M,PARM='DATE=U'
    ↑ 用这个程序
//SMPCSI DD DISP=SHR,DSN=SMPE.ZOS.GLOBAL.CSI
    ↑ 指定特定 PP 的 GLOBAL CSI
//SMPPTFIN DD DISP=SHR,DSN=MIB1.V1R12.FIXES(RO23087)
    ↑ 用这个 DD 指定要 REC 的 FIX 源文件
// DD DISP=SHR,DSN=MIB1.V1R12.FIXES(SV9992)
    ↑ 如果有多个 FIX，继续 concatenate 就 OK
//*SMPPTFIN DD PATHDISP=KEEP,
//* PATH='/u/mib001/SMPPTFIN'
    ↑ 现在比较流行的方式是上传到 USS 然后在 REC
//SMPOUT DD SYSOUT=*
//SMPPUNCH DD SYSOUT=*
//SMRPRT DD SYSOUT=*
//SMPLIST DD SYSOUT=*
//SMPSNAP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYMDUMP DD DUMMY
//SYSUDUMP DD SYSOUT=*
//SMPWRK1 DD UNIT=SYSDA,SPACE=(CYL,(5,15,15)),DCB=BLKSIZE=3120
//SMPWRK2 DD UNIT=SYSDA,SPACE=(CYL,(5,15,15)),DCB=BLKSIZE=3120
//SMPWRK3 DD UNIT=SYSDA,SPACE=(CYL,(5,15,15)),DCB=BLKSIZE=3120
//SMPWRK4 DD UNIT=SYSDA,SPACE=(CYL,(5,15,15)),DCB=BLKSIZE=3120
//SMPWRK6 DD UNIT=SYSDA,SPACE=(CYL,(5,15,15)),DCB=BLKSIZE=3120
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(5,1))
```

```
//SYSUT2 DD UNIT=SYSDA,SPACE=(CYL,(5,1))
//SYSUT3 DD UNIT=SYSDA,SPACE=(CYL,(5,1))
//SYSUT4 DD UNIT=SYSDA,SPACE=(CYL,(5,1))
↑ 上面这些 SMP 的 DD 都是打酱油的.....
//SMPTLIB DD UNIT=SYSDA,VOL=SER=MIBPP1,STORCLAS=NONSMS,DISP=SHR
↑ 这个 DD 特别点,有时候需要特别指定一下
//SMPCNTL DD *
SET BOUNDARY(GLOBAL).
↑ 从这开始才是 SMPE 的 CMD,第一条总是 SET 一个 ZONE,REC 的话就是 GLOBAL
RECEIVE SYSMODS LIST HOLDDATA.
↑ 这个最简单不过了,自己跑跑看吧
/*
//
```

JCL6-1

3.4.2 Apply

Receive 成功后,需要进行 APPLY CHECK,顾名思义,就是在实际 APPLY 之前,先确认一下可能发生的问题,比如是否缺少 pre-requisite 的 PTF,也可以确认 HOLD 信息。

```
//APPLY EXEC PGM=GIMSMP,REGION=6M,PARM='DATE=U'
//SMPCSI DD DISP=SHR,DSN=SMPE.ZOS.GLOBAL.CSI
↑ 前面的都一样
//SMPOUT DD SYSOUT=*
//SMPPUNCH DD SYSOUT=*
//SMPRPT DD SYSOUT=*
//SMPLIST DD SYSOUT=*
//SMPSNAP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SMPWRK1 DD UNIT=SYSDA,SPACE=(CYL,(5,15,15)),DCB=BLKSIZE=3120
//SMPWRK2 DD UNIT=SYSDA,SPACE=(CYL,(5,15,15)),DCB=BLKSIZE=3120
//SMPWRK3 DD UNIT=SYSDA,SPACE=(CYL,(5,15,15)),DCB=BLKSIZE=3120
//SMPWRK4 DD UNIT=SYSDA,SPACE=(CYL,(5,15,15)),DCB=BLKSIZE=3120
//SMPWRK6 DD UNIT=SYSDA,SPACE=(CYL,(5,15,15)),DCB=BLKSIZE=3120
//SMPCNTL DD *
SET BOUNDARY(TSTTZN).
↑ 换成 TARGET ZONE 的名字
APPLY SELECT (
```

```

RO18014,
RO18242,
RO23087
↑ 这些都是要 APPLY 的 FIX 番号
)
BYPASS (HOLDSYSTEM (ACTION, ENH, DYNACT, RESTART, EXIT) )
↑ 要养成查看 HOLD 信息的好习惯
CHECK
↑ 而且一定要先做 CHECK
.
↑ 我是句号
/* 如果要做覆盖 APPLY, 可使用 REDO
// 另外比较有用的参数就是 GROUPEXTEND 和 FORFMID, 请自行阅读手册

```

JCL6-2

3.4.3 Accept

这个命令多用于 PP 安装，而不适用于 APPLY FIX，因为 ACC 后的 LMOD 不可恢复。

```

//APPLY EXEC PGM=GIMSMP,REGION=6M,PARM='DATE=U'
//SMPCSI DD DISP=SHR,DSN=SMPE.ZOS.GLOBAL.CSI
↑ 前面的依然一样
//SMPCNTL DD *
SET BOUNDARY (DISTZN) .
↑ 换成 TARGET ZONE 的名字
ACCEPT SELECT (
    H270370
    J270371
)
↑ ACC 通常用于产品安装, FIX 一般不需要 ACC, APP 即可
GROUPEXTEND /* Also all requisite PTFs */
CHECK /* Do not update libraries */
FORFMID (H270370) /* For base FMID */
BYPASS (HOLDSYS, HOLDUSER, /* Bypass options */
HOLDCLASS (UCLREL, ERREL) )
.
↑ 我是句号

```

```
/*
//
```

JCL6-3

3.4.4 Reject

REC 过后的 SYSMOD 如无特殊指定，都会保存在 SMPPTS Ds 里面，有的时候一个 FIX 可能需要重新 REC 最新版本，而 FIX 番号不变，SMPE 对于已经 REC 过的 SYSMOD 是不可以二次 REC 的，就需要把之前 REC 过的 SYSMOD REJECT 出去。

```
//SMPCNTL DD *
```

```
SET BOUNDARY (GLOBAL) .
```

↑ REC 和 REJECT 都是 GLOBAL

```
REJECT S(
```

```
    UA54359
```

```
    UA60137
```

```
)
```

```
BYPASS(APPLYCHECK) .
```

↑ 如果这些 PTF 已经 APPLY 过了，也不要紧，同样可以 REJECT；重新 REC 过后，你可以用 APPLY REDO 去覆盖这些 SYSMOD

```
/*
//
```

JCL6-4

3.4.5 Restore

对于已经 APP 而没有 ACC 的 SYSMOD，你还可以把它还原，比如在你不小心 APP 了一个不想要的 SYSMOD 或需要进行二次处理的时候；当然你也可以用上面提到的 APPLY REDO 功能，看你对这些命令的掌握程度了。

```
//SMPCNTL DD *
```

```
SET BOUNDARY (TSTTZN) .
```

↑ APPLY 和 RESTORE 都是 TARGET ZONE

```
RESTORE S(
```

```
    UA54359
```

```
    UA60137
```

```
)
```

```
.
```

↑ 注意要加句号

```
/*
//
```

JCL6-5

3.4.6 Zone copy

简单地，我们都有一个测试系统一个生产系统，所以一般会有 TST 和 PRD 两个 TARGET ZONE，当我们在 TST 上面安装成功后，可以直接做 SMPE 的 ZONE COPY，复制一个 PRD 的对应的 ZONE 出来，然后再更新相应的 DDDEF 中 DS 的名字，要比重新安装要省事的。

```
//SMPCNTL DD *
SET BDY(GLOBAL) .
UCLIN.
  ADD GLOBALZONE
    ZONEINDEX (
      (PRDTZN, SMPE.UE.R12.PRDTZN.CSI, TARGET)
      (PRDDZN, SMPE.UE.R12.PRDDZN.CSI, DLIB)
    ) .
  ↑ 首先要在 GLOBAL ZONE 里面登记对应的 PRD 的 T 和 D ZONE
ENDUCL.
SET BDY(PRDTZN) .
ZONECOPY (TSTTZN) INTO (PRDTZN) OPTIONS(UEOPT) RELATED(PRDDZN) .
  ↑ 从 TST 的 TARGET ZONE 拷贝到 PRD 的 TZN，下面同理
SET BDY(PRDDZN) .
ZONECOPY (TSTDZN) INTO (PRDDZN) OPTIONS(UEOPT) RELATED(PRDTZN) .
/*
```

JCL6-6

3.4.7 Zone report

这个命令也是基于在我们有 TST 和 PRD 两个 TARGET 或 DIST ZONE 的前提下，做过维护的人都知道，测试系统和生产系统的 SYSMOD 可能会存在差异，用 REPORT 命令便可以找出两个 ZONE 直接的不同，并自动生产要同步这些差异的 APPLY 命令。

```
//SMPCNTL DD *
SET BDY(GLOBAL) .
REPORT SYSMODS
  INZONE (TSTTZN)
  COMPAREDTO (PRDTZN)
```

↑ 指定两个不同的 ZONE，但他们的类型要一样，比如都是 TARGET

```
/*  
//
```

JCL6-7

3.4.8 Zone list

这个命令用来查看各个 ZONE 中的所有或个别的 SYSMOD 和 DDDEF 的情况，多用于调查或验证。

```
//SMPCNTL DD *  
SET BDY(GLOBAL) .  
LIST SYSMODS .  
LIST DDDEF .  
/*  
//
```

JCL6-8

4. 附录

4.1 参考文献

[z/OS V1R10.0 MVS JCL Reference](#)

[z/OS V1R6.0-V1R10.0 MVS JCL User's Guide](#)

[z/OS V1R10.0 DFSMSdfp Utilities](#)

[z/OS V1R10.0 DFSMS Managing Catalogs](#)

[Redbooks: VSAM Demystified](#)

Redbooks: Introduction to the New Mainframe z/OS Basics

z/OS Basic Skills Information Center: Reusable JCL collection

4.2 商标

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

Advanced Peer-to-Peer Networking® AD/Cycle® AIX® C/370™ CICS® CICSplex® Domino® DB2® DFS™ DFSMSdfp™ DFSMSdss™ DFSMSshsm™ DFSORT™ DRDA® Encina® Enterprise Storage Server® Enterprise Systems Architecture/390® ECKD™ ESCON® FlashCopy® FICON® Geographically Dispersed Parallel Sysplex™ GDDM® GDPS® HiperSockets™ IBM® IMS™ Language Environment® Lotus® Multiprise® MVS™ MVS/ESA™ MVS/XA™ NetRexx™ NetView® Open Class® OS/390® Parallel Sysplex® Processor Resource/Systems Manager™ PR/SM™ QMF™ Redbooks™ RACF® RMF™ S/360™ S/370™ S/390® Sysplex Timer® System z9™ System/360™ System/370™ System/390® SAA® Tivoli® VisualAge® VSE/ESA™ VTAM® WebSphere® z/Architecture™ z/OS® z/VM® z/VSE™ zSeries® z9™

The following terms are trademarks of other companies:
EJB, Java, JDBC, JMX, JSP, JVM, J2EE, RSM, Sun, Sun Java, Sun Microsystems, VSM, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Visual Basic, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

4.3 致谢

Dedicated to my beloved family.

And special thanks to my teammates at IBM DL and JP.

