**笔 记**

# 难点

1. 进程间IPC通信:管道，FIFO，消息队列，共享内存，信号量
2. 同步：互斥量，条件变量，读写锁，文件，写记录锁，信号量
3. socket通信中高并发模型epoll,select ,poll，如何设置超时，检测连接是否断开。

# 网络编程

## 1、什么是Socket？

上面我们已经知道网络中的进程是通过socket来通信的，那什么是socket呢？socket起源于Unix，而Unix/Linux基本哲学之一就是“一切皆文件”，都可以用“打开open –> 读写write/read –> 关闭close”模式来操作。我的理解就是Socket就是该模式的一个实现，socket即是一种特殊的文件，一些socket函数就是对其进行的操作（读/写IO、打开、关闭），这些函数我们在后面进行介绍。

### socket一词的起源

在组网领域的首次使用是在1970年2月12日发布的文献[IETF RFC33](http://datatracker.ietf.org/doc/rfc33/)中发现的，撰写者为Stephen Carr、Steve Crocker和Vint Cerf。根据美国计算机历史博物馆的记载，Croker写道：“命名空间的元素都可称为套接字接口。一个套接字接口构成一个连接的一端，而一个连接可完全由一对套接字接口规定。”计算机历史博物馆补充道：“这比BSD的套接字接口定义早了大约12年。”

## 2、socket的基本操作

既然socket是“open—write/read—close”模式的一种实现，那么socket就提供了这些操作对应的函数接口。下面以TCP为例，介绍几个基本的socket接口函数。

### 2.1、socket()函数

int **socket**(int domain, int type, int protocol);

socket函数对应于普通文件的打开操作。普通文件的打开操作返回一个文件描述字，而**socket()**用于创建一个socket描述符（socket descriptor），它唯一标识一个socket。这个socket描述字跟文件描述字一样，后续的操作都有用到它，把它作为参数，通过它来进行一些读写操作。

正如可以给fopen的传入不同参数值，以打开不同的文件。创建socket的时候，也可以指定不同的参数创建不同的socket描述符，socket函数的三个参数分别为：

* domain：即协议域，又称为协议族（family）。常用的协议族有，AF\_INET、AF\_INET6、AF\_LOCAL（或称AF\_UNIX，Unix域socket）、AF\_ROUTE等等。协议族决定了socket的地址类型，在通信中必须采用对应的地址，如AF\_INET决定了要用ipv4地址（32位的）与端口号（16位的）的组合、AF\_UNIX决定了要用一个绝对路径名作为地址。
* type：指定socket类型。常用的socket类型有，SOCK\_STREAM、SOCK\_DGRAM、SOCK\_RAW、SOCK\_PACKET、SOCK\_SEQPACKET等等（socket的类型有哪些？）。
* protocol：故名思意，就是指定协议。常用的协议有，IPPROTO\_TCP、IPPTOTO\_UDP、IPPROTO\_SCTP、IPPROTO\_TIPC等，它们分别对应TCP传输协议、UDP传输协议、STCP传输协议、TIPC传输协议（这个协议我将会单独开篇讨论！）。

注意：并不是上面的type和protocol可以随意组合的，如SOCK\_STREAM不可以跟IPPROTO\_UDP组合。当protocol为0时，会自动选择type类型对应的默认协议。

当我们调用**socket**创建一个socket时，返回的socket描述字它存在于协议族（address family，AF\_XXX）空间中，但没有一个具体的地址。如果想要给它赋值一个地址，就必须调用bind()函数，否则就当调用connect()、listen()时系统会自动随机分配一个端口。

### 2.2、bind()函数

正如上面所说bind()函数把一个地址族中的特定地址赋给socket。例如对应AF\_INET、AF\_INET6就是把一个ipv4或ipv6地址和端口号组合赋给socket。

int bind(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);

函数的三个参数分别为：

* sockfd：即socket描述字，它是通过socket()函数创建了，唯一标识一个socket。bind()函数就是将给这个描述字绑定一个名字。
* addr：一个const struct sockaddr \*指针，指向要绑定给sockfd的协议地址。这个地址结构根据地址创建socket时的地址协议族的不同而不同，如ipv4对应的是：
* struct sockaddr\_in {
* sa\_family\_t sin\_family; /\* address family: AF\_INET \*/
* in\_port\_t sin\_port; /\* port in network byte order \*/
* struct in\_addr sin\_addr; /\* internet address \*/
* };
* /\* Internet address. \*/
* struct in\_addr {
* uint32\_t s\_addr; /\* address in network byte order \*/

};

ipv6对应的是：

struct sockaddr\_in6 {

sa\_family\_t sin6\_family; /\* AF\_INET6 \*/

in\_port\_t sin6\_port; /\* port number \*/

uint32\_t sin6\_flowinfo; /\* IPv6 flow information \*/

struct in6\_addr sin6\_addr; /\* IPv6 address \*/

uint32\_t sin6\_scope\_id; /\* Scope ID (new in 2.4) \*/

};

struct in6\_addr {

unsigned char s6\_addr[16]; /\* IPv6 address \*/

};

Unix域对应的是：

#define UNIX\_PATH\_MAX 108

struct sockaddr\_un {

sa\_family\_t sun\_family; /\* AF\_UNIX \*/

char sun\_path[UNIX\_PATH\_MAX]; /\* pathname \*/

};

* addrlen：对应的是地址的长度。

通常服务器在启动的时候都会绑定一个众所周知的地址（如ip地址+端口号），用于提供服务，客户就可以通过它来接连服务器；而客户端就不用指定，有系统自动分配一个端口号和自身的ip地址组合。这就是为什么通常服务器端在listen之前会调用bind()，而客户端就不会调用，而是在connect()时由系统随机生成一个。

#### 网络字节序与主机字节序

**主机字节序:**就是我们平常说的大端和小端模式：不同的CPU有不同的字节序类型，这些字节序是指整数在内存中保存的顺序，这个叫做主机序。引用标准的Big-Endian和Little-Endian的定义如下：

　　a) Little-Endian就是低位字节排放在内存的低地址端，高位字节排放在内存的高地址端。

　　b) Big-Endian就是高位字节排放在内存的低地址端，低位字节排放在内存的高地址端。

**网络字节序**：4个字节的32 bit值以下面的次序传输：首先是0～7bit，其次8～15bit，然后16～23bit，最后是24~31bit。这种传输次序称作大端字节序。**由于TCP/IP首部中所有的二进制整数在网络中传输时都要求以这种次序，因此它又称作网络字节序。**字节序，顾名思义字节的顺序，就是大于一个字节类型的数据在内存中的存放顺序，一个字节的数据没有顺序的问题了。

所以：在将一个地址绑定到socket的时候，请先将主机字节序转换成为网络字节序，而不要假定主机字节序跟网络字节序一样使用的是Big-Endian。由于这个问题曾引发过血案！公司项目代码中由于存在这个问题，导致了很多莫名其妙的问题，所以请谨记对主机字节序不要做任何假定，务必将其转化为网络字节序再赋给socket。

### 2.3、listen()、connect()函数

如果作为一个服务器，在调用socket()、bind()之后就会调用listen()来监听这个socket，如果客户端这时调用connect()发出连接请求，服务器端就会接收到这个请求。

int listen(int sockfd, int backlog);

int connect(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);

listen函数的第一个参数即为要监听的socket描述字，第二个参数为相应socket可以排队的最大连接个数。socket()函数创建的socket默认是一个主动类型的，listen函数将socket变为被动类型的，等待客户的连接请求。

connect函数的第一个参数即为客户端的socket描述字，第二参数为服务器的socket地址，第三个参数为socket地址的长度。客户端通过调用connect函数来建立与TCP服务器的连接。

### 2.4、accept()函数

TCP服务器端依次调用socket()、bind()、listen()之后，就会监听指定的socket地址了。TCP客户端依次调用socket()、connect()之后就向TCP服务器发送了一个连接请求。TCP服务器监听到这个请求之后，就会调用accept()函数取接收请求，这样连接就建立好了。之后就可以开始网络I/O操作了，即类同于普通文件的读写I/O操作。

int accept(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen);

accept函数的第一个参数为服务器的socket描述字，第二个参数为指向struct sockaddr \*的指针，用于返回客户端的协议地址，第三个参数为协议地址的长度。如果accpet成功，那么其返回值是由内核自动生成的一个全新的描述字，代表与返回客户的TCP连接。

注意：accept的第一个参数为服务器的socket描述字，是服务器开始调用socket()函数生成的，称为监听socket描述字；而accept函数返回的是已连接的socket描述字。一个服务器通常通常仅仅只创建一个监听socket描述字，它在该服务器的生命周期内一直存在。内核为每个由服务器进程接受的客户连接创建了一个已连接socket描述字，当服务器完成了对某个客户的服务，相应的已连接socket描述字就被关闭。

### 2.5、read()、write()等函数

万事具备只欠东风，至此服务器与客户已经建立好连接了。可以调用网络I/O进行读写操作了，即实现了网咯中不同进程之间的通信！网络I/O操作有下面几组：

* read()/write()
* recv()/send()
* readv()/writev()
* recvmsg()/sendmsg()
* recvfrom()/sendto()

我推荐使用recvmsg()/sendmsg()函数，这两个函数是最通用的I/O函数，实际上可以把上面的其它函数都替换成这两个函数。它们的声明如下：

#include <unistd.h>

ssize\_t read(int fd, void \*buf, size\_t count);

ssize\_t write(int fd, const void \*buf, size\_t count);

#include <sys/types.h>

#include <sys/socket.h>

ssize\_t send(int sockfd, const void \*buf, size\_t len, int flags);

ssize\_t recv(int sockfd, void \*buf, size\_t len, int flags);

ssize\_t sendto(int sockfd, const void \*buf, size\_t len, int flags,

const struct sockaddr \*dest\_addr, socklen\_t addrlen);

ssize\_t recvfrom(int sockfd, void \*buf, size\_t len, int flags,

struct sockaddr \*src\_addr, socklen\_t \*addrlen);

ssize\_t sendmsg(int sockfd, const struct msghdr \*msg, int flags);

ssize\_t recvmsg(int sockfd, struct msghdr \*msg, int flags);

read函数是负责从fd中读取内容.当读成功时，read返回实际所读的字节数，如果返回的值是0表示已经读到文件的结束了，小于0表示出现了错误。如果错误为EINTR说明读是由中断引起的，如果是ECONNREST表示网络连接出了问题。

write函数将buf中的nbytes字节内容写入文件描述符fd.成功时返回写的字节数。失败时返回-1，并设置errno变量。 在网络程序中，当我们向套接字文件描述符写时有俩种可能。1)write的返回值大于0，表示写了部分或者是全部的数据。2)返回的值小于0，此时出现了错误。我们要根据错误类型来处理。如果错误为EINTR表示在写的时候出现了中断错误。如果为EPIPE表示网络连接出现了问题(对方已经关闭了连接)。

其它的我就不一一介绍这几对I/O函数了，具体参见man文档或者baidu、Google，下面的例子中将使用到send/recv。

### 2.6、close()函数

在服务器与客户端建立连接之后，会进行一些读写操作，完成了读写操作就要关闭相应的socket描述字，好比操作完打开的文件要调用fclose关闭打开的文件。

#include <unistd.h>

int close(int fd);

close一个TCP socket的缺省行为时把该socket标记为以关闭，然后立即返回到调用进程。该描述字不能再由调用进程使用，也就是说不能再作为read或write的第一个参数。

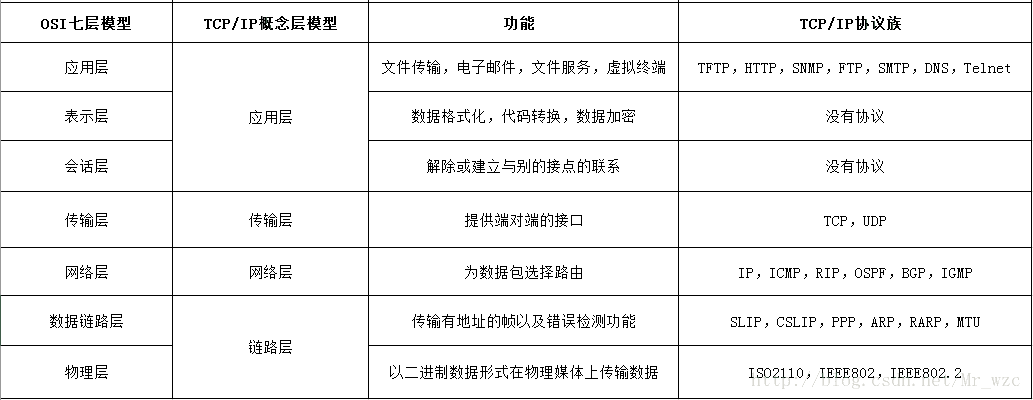
注意：close操作只是使相应socket描述字的引用计数-1，只有当引用计数为0的时候，才会触发TCP客户端向服务器发送终止连接请求。

IP协议、HTTP协议、TCP协议、TCP/IP 协议我们经常遇到，而且初学的时候容易弄混。那么他们之间有什么关系呢？重点内容

## 3、什么是TCP/IP 协议

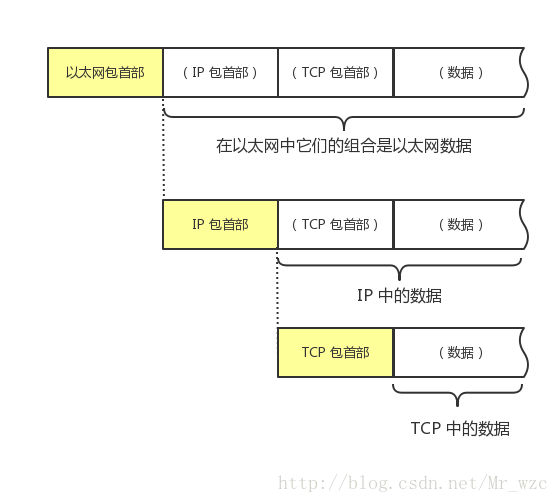
TCP/IP不是一个协议，而是一个协议族的统称。里面包括IP协议、HTTP协议、TCP协议等。而三次握手四次挥手就发生在TCP协议中。

TCP/IP协议栈主要分为四层：应用层、传输层、网络层、数据链路层。每层都有相应的协议，具体如下图：



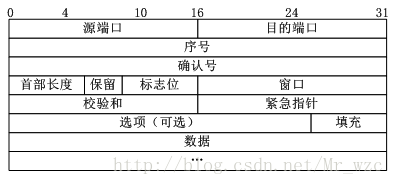
下面是从网络上找到的TCP/IP通信数据流





## 4、TCP协议

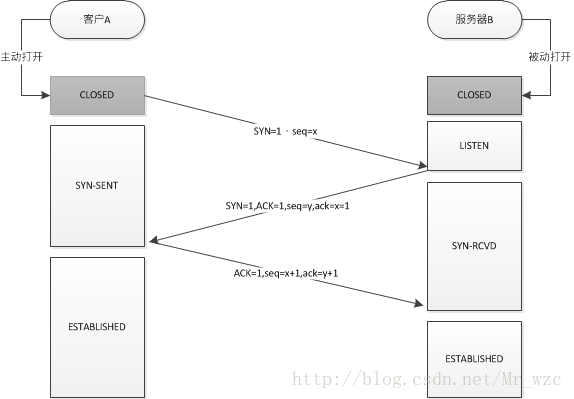
TCP是一个属于传输层的面向连接的、可靠的、基于字节流的传输层通信协议。 为了确保信息能够确保准确无误的到达，TCP采用了著名的三次握手策略。

TCP报文格式图：  


上图中有几个字段需要重点介绍下：  
（1）序号：Seq序号，占32位，用来标识从TCP源端向目的端发送的字节流，发起方发送数据时对此进行标记。  
（2）确认号：Ack序号，占32位，只有ACK标志位为1时，确认序号字段才有效，Ack=Seq+1。  
（3）标志位：共6个，即URG、ACK、PSH、RST、SYN、FIN等，在三次握手中建立连接时出现。  
具体含义如下：  
（A）URG：紧急指针（urgent pointer）有效。  
（B）ACK：确认序号有效。  
（C）PSH：接收方应该尽快将这个报文交给应用层。  
（D）RST：重置连接。  
（E）SYN：发起一个新连接。  
（F）FIN：释放一个连接。  
需要注意的是：  
- 不要将确认序号Ack与标志位中的ACK搞混了，注意大小写。  
- 确认方Ack=发起方Req+1，两端配对。

## 5、三次握手

所谓三次握手（Three-Way Handshake）即建立TCP连接，是指建立一个TCP连接时，需要客户端和服务端总共发送3个包以确认连接的建立。在socket编程中，这一过程由客户端执行connect来触发，整个流程如下图所示：



- SYN(synchronous建立联机)  
- ACK(acknowledgement 确认)  
- PSH(push传送)  
- FIN(finish结束)  
- RST(reset重置)  
- URG(urgent紧急)  
- seq(Sequence number 顺序号码)  
- Ack(Acknowledge number 确认号码 )  
- ESTABLISHED 建立，创建

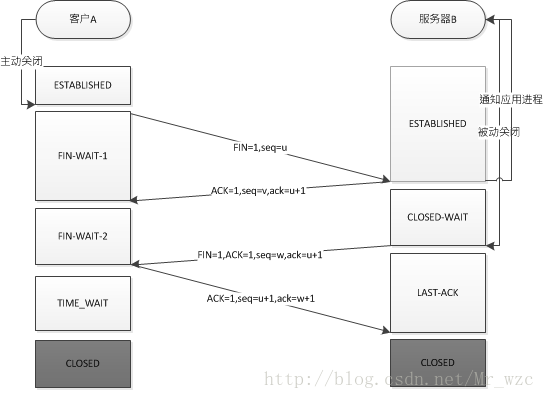
（1）第一次握手：Client将标志位SYN置为1，随机产生一个值seq=J，并将该数据包发送给Server，Client进入SYN\_SENT状态，等待Server确认。

（2）第二次握手：Server收到数据包后由标志位SYN=1知道Client请求建立连接，Server将标志位SYN和ACK都置为1，ack (number )=J+1，随机产生一个值seq=K，并将该数据包发送给Client以确认连接请求，Server进入SYN\_RCVD状态。

（3）第三次握手：Client收到确认后，检查ack是否为J+1，ACK是否为1，如果正确则将标志位ACK置为1，ack=K+1，并将该数据包发送给Server，Server检查ack是否为K+1，ACK是否为1，如果正确则连接建立成功，Client和Server进入ESTABLISHED状态，完成三次握手，随后Client与Server之间可以开始传输数据了。

SYN攻击：  
在三次握手过程中，Server发送SYN-ACK之后，收到Client的ACK之前的TCP连接称为半连接（half-open connect），此时Server处于SYN\_RCVD状态，当收到ACK后，Server转入ESTABLISHED状态。SYN攻击就是Client在短时间内伪造大量不存在的IP地址，并向Server不断地发送SYN包，Server回复确认包，并等待Client的确认，由于源地址是不存在的，因此，Server需要不断重发直至超时，这些伪造的SYN包将长时间占用未连接队列，导致正常的SYN请求因为队列满而被丢弃，从而引起网络堵塞甚至系统瘫痪。SYN攻击时一种典型的DDOS攻击，检测SYN攻击的方式非常简单，即当Server上有大量半连接状态且源IP地址是随机的，则可以断定遭到SYN攻击了，使用如下命令可以让之现行：  
#netstat -nap | grep SYN\_RECV

## 6、四次挥手



由于TCP连接时全双工的，因此，每个方向都必须要单独进行关闭，这一原则是当一方完成数据发送任务后，发送一个FIN来终止这一方向的连接，收到一个FIN只是意味着这一方向上没有数据流动了，即不会再收到数据了，但是在这个TCP连接上仍然能够发送数据，直到这一方向也发送了FIN。首先进行关闭的一方将执行主动关闭，而另一方则执行被动关闭，上图描述的即是如此。  
（1）第一次挥手：Client发送一个FIN，用来关闭Client到Server的数据传送，Client进入FIN\_WAIT\_1状态。  
（2）第二次挥手：Server收到FIN后，发送一个ACK给Client，确认序号为收到序号+1（与SYN相同，一个FIN占用一个序号），Server进入CLOSE\_WAIT状态。  
（3）第三次挥手：Server发送一个FIN，用来关闭Server到Client的数据传送，Server进入LAST\_ACK状态。  
（4）第四次挥手：Client收到FIN后，Client进入TIME\_WAIT状态，接着发送一个ACK给Server，确认序号为收到序号+1，Server进入CLOSED状态，完成四次挥手。

## 6、附注

关于三次握手与四次挥手通常都会有典型的面试题，在此提出供有需求的XDJM们参考：  
（1）三次握手是什么或者流程？四次握手呢？答案前面分析就是。  
（2）为什么建立连接是三次握手，而关闭连接却是四次挥手呢？

这是因为服务端在LISTEN状态下，收到建立连接请求的SYN报文后，把ACK和SYN放在一个报文里发送给客户端。而关闭连接时，当收到对方的FIN报文时，仅仅表示对方不再发送数据了但是还能接收数据，己方也未必全部数据都发送给对方了，所以己方可以立即close，也可以发送一些数据给对方后，再发送FIN报文给对方来表示同意现在关闭连接，因此，己方ACK和FIN一般都会分开发送。

# C/S模型中的并发服务器

Linux中实现并发服务器的方法很简单：每当客户端对服务器有网络请求时，服务器程序就fork一个子服务器进程来处理这个客户的请求，而主服务器程序仍处于等待其他客户端网络请求的状态。基于这个原理，并发服务器的基本模型如下：

if ((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) {

my\_error("socket", errno, \_\_LINE\_\_);

}

if (bind(sockfd, (struct sockaddr \*)&my\_addr,

sizeof(struct sockaddr\_in)) == -1) {

my\_error("bind", errno, \_\_LINE\_\_);

}

if (listen(sockfd, BACKLOG) == -1) {

my\_error("listen", errno, \_\_LINE\_\_);

}

while (1) {

sin\_size = sizeof(struct sockaddr\_in);

if ((client\_fd = accept(sockfd, (struct sockaddr \*)&remote\_addr, &sin\_size)) == -1) {

my\_error("accept", errno, \_\_LINE\_\_);

continue;

}

if ((pid = fork()) == 0) {

close(sockfd);

process\_client\_request(client\_fd);

close(client\_fd);

exit(0);

} else if (pid > 0)

close(client\_fd);

else

my\_error("fork", errno, \_\_LINE\_\_);

}

每当accept()接收到一个TCP连接时，主服务器进程就fork一个子服务器进程。子服务器进程调用相应的函数，通过client\_fd（连接套接字）对客户端发来的网络请求进程处理；由于客户端的请求已被子服务进程处理，那么主服务器进程就什么也不做，通过sockfd（监听套接字）继续循环等待新的网络请求。

这里我们需要特别强调的是，在父子进程中需要关闭相应的套接字描述符。从上述代码中可以看到，子服务进程在处理客户端的网络请求之前关闭了监听套接字，主服务进程则关闭了连接套接字，子服务进程在处理完客户端请求后关闭了连接套接字。最后一种关闭套接字的情形比较合乎常理，这里我们重点关注前两种关闭套接字的情况。

我们的问题是：子服务进程关闭了监听套接字，服务器是否还能监听其他连接请求；父服务进程关闭了连接套接字，服务器是否还能够处理客户端请求。

如果理解了文件的的引用计数，这个问题就会迎刃而解。每个文件都有一个引用计数，该引用计数表示当前系统内的所有进程打开该文件描述符的个数。套接字是一种特殊的文件，当然也有引用计数。

在并发服务器这个例子当中，在accept函数之前，sockfd的引用计数为1；在fork函数执行之前，sockfd和client\_fd的引用计数分别为1；当fork执行后，由于子进程复制了父进程的资源，所以子进程也拥有这两个套接字描述符，则此时sockfd和client\_fd的引用计数都为2。只有当子进程处理完客户请求时，client\_fd的引用计数才由于close函数而变为0。由于父服务器进程的主要任务是监听客户请求，则它关闭了连接套接字client\_fd；而子进程的主要任务是处理客户请求，它不必监听其他客户请求，因此子进程关闭了sockfd。由上可知，父子进程关闭相应的套接字并不会影响其所负责的通信功能。

这就是监听套接字和连接套接字两者的最终状态，子进程处理客户请求，父进程监听其他客户请求。这样的并发服务器看似完美，但是会产生许多僵死进程，这是为什么？下文将会分析。

# C/S模型的健壮性-为僵死进程收尸

并发服务器可以同时处理多个客户端的请求，服务器对客户端网络请求的处理是通过fork子服务进程来完成的。当有多个客户请求时，主服务器进程就会产生多个子服务器进程。当这些子服务器进程处理完客户请求时，就终止运行了。由于主服务器进程（也就是这些子服务器进程的父亲）仍然在运行，因此这些终止的子服务器进程在主服务器退出之前就成为了僵死进程。通常服务器进程都长时间处于运行状态，所以这些僵死进程就会一直存在于系统内。如果不及时清除这些僵死进程，他们将占用内核的空间，最终可能会耗尽系统资源。本文将描述如何清理这些僵死进程。

### 1. 僵死进程的产生

下面将通过一个简单的C/S模型回射程序来演示僵死进程的产生。我们首先将服务器置于后台运行，然后再查看当前终端（pts/3）下的进程状态：

edsionte@edsionte-desktop:~/echoCS$ ps -o pid,ppid,args,stat,wchan

PID PPID COMMAND STAT WCHAN

5071 2305 bash Ss wait

6146 5071 ./server S inet\_csk\_wait\_for\_connect

6149 5071 ps -o pid,ppid,args,stat,wc R+ -

由上可以看出，服务器进程处于等待连接的状态，此刻它正在监听客户请求。接着运行客户端程序，连接成功后输入任意的字符串，可以看到服务器和客户端通信正常。

edsionte@edsionte-desktop:~/echoCS$ ./client 127.0.0.1

received a connection from:127.0.0.1

ps

ps

我们在另一个终端下，查看位于终端pts/3下客户端和服务器进程的状态。

edsionte@edsionte-desktop:~/echoCS$ ps -t pts/3 -o pid,ppid,args,stat,wchan

PID PPID COMMAND STAT WCHAN

5071 2305 bash Ss wait

6146 5071 ./server S inet\_csk\_wait\_for\_connect

6150 5071 ./client 127.0.0.1 S+ n\_tty\_read

6151 6146 ./server S sk\_wait\_data

由于客户进程发来连接请求，因此服务器进程fork了一个子服务器进程，从pid和ppid可以看到两个服务器进程之间的父子关系。父子进程此刻都处于等待状态，不过他们等待的目标不同：父进程监听客户请求，子进程则阻塞于read函数等待套接字数据的来临。

接下来通过发信号SIGINT（通过键盘上的ctrl+c可发出此信号）使客户进程终止。

edsionte@edsionte-desktop:~/echoCS$ ./client 127.0.0.1

received a connection from:127.0.0.1

ps

ps

^C

我们在另一个终端查看pty/3终端当前进程的状态：

edsionte@edsionte-desktop:~/echoCS$ ps -t pts/3 -o pid,ppid,args,stat,wchan

PID PPID COMMAND STAT WCHAN

5071 2305 bash Ss+ n\_tty\_read

6146 5071 ./server S inet\_csk\_wait\_for\_connect

6151 6146 [server] Z exit

可以看到，子服务器进程现在处于僵死状态。目前我们只有运行了一次客户端程序，如果一个服务器的网络请求繁忙，则会产生很多僵死进程。

### 2.处理僵死进程

当子进程终止时会给父进程发送SIGCHLD信号，因此我们可以利用信号处理函数捕获这个信号并对僵死进程进行处理。我们知道在父进程中调用wait函数可以防止先于父进程终止的子进程编程僵死进程，因此我们的信号捕获函数如下所示：

void sig\_zchild(int signo)

{

pid\_t pid;

int stat;

pid = wait(&stat);

printf("child %d terminated\n", pid);

return;

}

并且我们需要适当的修改服务器程序，在accept函数调用之前调用signal函数：

if(listen(sockfd, BACKLOG) == -1) {

printf("listen error!\n");

exit(1);

}

if (signal(SIGCHLD, sig\_zchild) == SIG\_ERR) {

printf("signal error!\n");

exit(1);

}

while (1) {

sin\_size = sizeof(struct sockaddr\_in);

if ((client\_fd = accept(sockfd, (struct sockaddr \*)&remote\_addr,

&sin\_size)) == -1) {

printf("accept error!\n");

continue;

}

…… ……

}//while

我们下面进行测试，我们首先在终端后台运行服务器程序，然后同时在其他多个终端分别运行客户端程序。当前终端的进程状态如下：

edsionte@edsionte-desktop:~/echoCS$ ps

PID TTY TIME CMD

6699 pts/0 00:00:00 bash

6926 pts/0 00:00:00 server

6947 pts/0 00:00:00 server

6966 pts/0 00:00:00 server

6985 pts/0 00:00:00 server

6986 pts/0 00:00:00 ps

然后我们分别在运行客户程序的终端杀死客户端进程。再次查看当前终端下的进程状态，可以发现并没有僵死的子服务进程。因为每个子进程在终止时发送SIGCHLD信号都会被父进程中的信号处理函数捕获。

### 3.僵死进程处理的加强版

上述的客户端进程每次向服务器端只发出一个连接请求，当客户端终止时子服务器进程将终止；如果一个客户端向服务器发出多个连接请求，当该客户端终止时多个子服务器进程将同时终止，也就是说父服务器进程将面临同时处理多个SIGCHLD信号的情况。

按照上述的特殊客户端，我们将回射C/S模型中的客户端适当修改，使运行一次客户端就连接服务器5次，这样就会产生相应的5个子服务器进程。

int i;

for (i = 0; i < 5; i++) {

if((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) {

printf("socket error!\n");

exit(1);

}

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(SERV\_PORT);

serv\_addr.sin\_addr = \*((struct in\_addr \*)host->h\_addr);

if(connect(sockfd, (struct sockaddr \*)&serv\_addr,

sizeof(struct sockaddr)) == -1) {

printf("connect error!\n");

exit(1);

}

}

str\_cli(stdin, sockfd);

运行服务器程序和刚刚修改过的客户端程序，我们作如下测试：

edsionte@edsionte-desktop:~/echoCS$ ps -t pts/0 -o pid,ppid,args,stat

PID PPID COMMAND STAT

2651 2649 bash Ss

2861 2651 ./server S

2865 2651 ./mclient 127.0.0.1 S+

2866 2861 ./server S

2867 2861 ./server S

2868 2861 ./server S

2869 2861 ./server S

2870 2861 ./server S

edsionte@edsionte-desktop:~/echoCS$ kill 2865

edsionte@edsionte-desktop:~/echoCS$ ps -t pts/0 -o pid,ppid,args,stat

PID PPID COMMAND STAT

2651 2649 bash Ss+

2861 2651 ./server S

2867 2861 [server] Z

2868 2861 [server] Z

2869 2861 [server] Z

2870 2861 [server] Z

当杀死客户端进程后，只有一个子服务器进程被SIGCHLD信号处理函数捕获处理，其他四个子服务器进程仍然处于僵死状态。出现这种现象的原因是，主服务器进程中的信号处理函数使用了wait函数，该函数会一直阻塞到出现第一个终止的子进程。也就是说，该函数只等待第一个终止的子进程并返回。

本客户端程序终止后，将同时产生5个SIGCHLD信号，而wait函数只处理其中一个，解决这个问题的办法是使用waitpid函数。该函数将周期性的检查是否有子进程终止，也就是说可以等待所有的终止子进程。修改后的服务器信号处理函数如下：

void sig\_zchild(int signo)

{

pid\_t pid;

int stat;

while ((pid = waitpid(-1, &stat, WNOHANG)) > 0)

printf("child %d terminated\n", pid);

return;

}

按此修改后SIGCHLD信号处理函数后，就可以避免留下僵死进程。

## epoll说明

epoll主要涉及到三个函数,这个要比select和poll复杂,从本质上来说是因为epoll把通知内核我想要观察的描述符和内核通知我哪些描述符发生了变化分成了两个独立的部分,而前述的select和poll都是合二为一的,这也是为什么epoll能做到性能优势。

#### 1.通知内核我想要观察哪些描述符

##### epoll\_create

epollcreate用于创建一个文件描述符,以后所有的操作都与此文件描述符有关。实际上你可以认为这个相当于一个文件，存放了进程要观察的所有文件描述符集合。  
原型是

int epoll\_create(int size)

调用方式也很简单:  
int epfd=epoll\_create(10);  
size 根据需要调整.

#### epoll\_ctl 用于通知内核添加删除待观察文件描述符

epoll\_ctl的使用比较复杂,主要是涉及到epoll\_event 的操作.  
epoll\_event的定义如下:

typedef union epoll\_data {

void \*ptr;

int fd;

\_\_uint32\_t u32;

\_\_uint64\_t u64;

} epoll\_data\_t;

//感兴趣的事件和被触发的事件

struct epoll\_event {

\_\_uint32\_t events; /\* Epoll events \*/

epoll\_data\_t data; /\* User data variable \*/

};

epoll\_ctl的原型如下:

int epoll\_ctl(int epfd, int op, int fd, struct epoll\_event \*event);

* epfd 就是刚刚用epoll\_create返回的描述符
* op 是对文件描述符集合的操作,主要有增,删,改,没有查是因为这个是使用者自己维护的
  + EPOLL\_CTL\_ADD 增
  + EPOLL\_CTL\_MOD 改
  + EPOLL\_CTL\_DEL 删
* fd 待增删改的文件描述符
* event 这个是最复杂的数据结构也是我们关注的核心.这里我们只使用到了文件描述符的操作,其他说真的,我也不晓得用来做什么.
  + events是要观察文件描述符的哪些事件,主要有
    - EPOLLIN 可读
    - EPOLLOUT 可写
    - EPOLLERR 发生错误
    - EPOLLET 设置为边沿出发Edge Triggered, 类似于数电中的概念 ,如果不指定默认是水平触发Level Triggered
    - EPOLLLT——水平触发  
      EPOLLET——边缘触发  
        
      epoll有EPOLLLT和EPOLLET两种触发模式，LT是默认的模式，ET是“高速”模式。LT模式下，只要这个fd还有数据可读，每次 epoll\_wait都会返回它的事件，提醒用户程序去操作，而在ET（边缘触发）模式中，它只会提示一次，直到下次再有数据流入之前都不会再提示了，无 论fd中是否还有数据可读。所以在ET模式下，read一个fd的时候一定要把它的buffer读光，也就是说一直读到read的返回值小于请求值，或者 遇到EAGAIN错误。
    - EPOLLONESHOT 只观察一次事件，一旦发生,对应的fd就会被内核从集合中删除
    - EPOLLPRI tcp 带外数据,用的很少
    - EPOLLHUP 对应的文件描述符被挂断,终端关闭事件.
  + data中的fd 设置为和epoll\_ctl参数列表中的fd一致即可.

经过以上处理,内核已经知道了和epfd关联的待观察文件描述符集以及上面的事件.

#### 等待关心事件的发生

epoll\_wait就是用来等待上述事件发生的,原型:int epoll\_wait(int epfd, struct epoll\_event \* events, int maxevents, int timeout), epfd和epoll\_ctl是一样的,events是内核返回给进程哪个文件描述符发生了什么事情,具体用法参考**epoll\_ctl**。maxevents指明了events数组 大小；timeout可以指定超时，单位为毫秒（0会立即返回，-1是永久阻塞）。

### unix网络编程中的str\_cli函数

这个函数同时监听标准输入和一个连接到服务器的socket,如果用户输入了什么内容就会发送到服务器,如果从服务器收到了任何内容就直接输出到终端.

#include "../lib/unp.h"

#include <sys/epoll.h>

void str\_cli(FILE \*fp, int sockfd)

{

int efd;

struct epoll\_event event;

struct epoll\_event events[20];

int i;

int nfds;

char buf[MAXLINE];

int n;

efd = epoll\_create (10);

if(efd<0){

err\_sys("epoll create failed");

}

event.data.fd=fileno(fp);

event.events=EPOLLIN;

epoll\_ctl(efd,EPOLL\_CTL\_ADD,fileno(fp),&event);

event.data.fd=sockfd;

event.events=EPOLLIN;

epoll\_ctl(efd,EPOLL\_CTL\_ADD,sockfd,&event);

for ( ; ; ) {

nfds=epoll\_wait(efd,events,sizeof(events)/sizeof(struct epoll\_event),-1);

if(nfds<0){

err\_sys("epoll wait error %d ",errno);

}

for(i=0;i<nfds;i++){

if(events[i].data.fd==sockfd){

if ( (n = Read(sockfd, buf, MAXLINE)) == 0) {

err\_quit("str\_cli: server terminated prematurely");

}

Write(fileno(stdout), buf, n);

}

if(events[i].data.fd==fileno(fp)){

if ( (n = Read(fileno(fp), buf, MAXLINE)) == 0) {

close(sockfd);

break;

}

Writen(sockfd, buf, n);

}

}

}

close(efd);

}

## server bind失败的原因

**当我们先启动server端，后启动client端，然后Ctrl-C使server端终止，这时马上再运行server后就会出现：bind error；这是因为虽然server的应用程序终止了，**

**但是TCP协议层的连接并没有完全断开，因此不能再次监听同样的server端口；当我们把client也终止掉以后，client会自动关闭自己的socket描述符，**

**server的TCP连接也收到了client发送的FIN段后处于TIME\_WAIT状态。TCP协议规定，主动关闭连接的一方要处于TIME\_WAIT状态，等待2MSL的时间后才能回到CLOSED状态，**

**因为我们先终止的是server，所以server是主动关闭连接的一方，在TIME\_WAIT期间仍不能再次监听同样的server端口。**

**其实，在server的TCP连接没有完全断开之前不允许重新监听是不合理的，因为，TCP的连接没有完全断开指的是connfd(127.0.0.1:8080)没有完全断开，**

**而我们重新监听的lesten(0.0.0.0:8080)，虽然端口号一样，但是IP不同，connfd对应的某个客户端通讯的一个具体的IP地址，**

**而listenfd对应的是wildcard address。解决这个问题的方法是使用setsockopt()设置socket描述符的选项SO\_REUSEADDR为1，**

**表示允许创建端口号相同但IP地址不同的多个sock描述符，在server代码的socket()和bind()调用之间插入**

**int opt = 1;**

**setsockopt(listenfd, SOL\_SOCCKET, SO\_REUSEADDR, &opt, sizeof(opt));**

## 线程池

### 1.线程池基本原理

**在传统服务器结构中, 常是 有一个总的 监听线程监听有没有新的用户连接服务器, 每当有一个新的 用户进入, 服务器就开启一个新的线程用户处理这 个用户的数据包。这个线程只服务于这个用户 , 当 用户与服务器端关闭连接以后, 服务器端销毁这个线程。然而频繁地开辟与销毁线程极大地占用了系统的资源。而且在大量用户的情况下, 系统为了开辟和销毁线程将浪费大量的时间和资源。线程池提供了一个解决外部大量用户与服务器有限资源的矛盾, 线程池和传统的一个用户对应一 个线程的处理方法不同, 它的基本思想就是在程序 开始时就在内存中开辟一些线程, 线程的数目是 固定的,他们独自形成一个类, 屏蔽了对外的操作, 而服务器只需要将数据包交给线程池就可以了。当有新的客户请求到达时 , 不是新创建一个线程为其服务 , 而是从“池子”中选择一个空闲的线程为新的客户请求服务 ,服务完毕后 , 线程进入空闲线程池中。如果没有线程空闲 的 话, 就 将 数 据 包 暂 时 积 累 , 等 待 线 程 池 内 有 线 程空闲以后再进行处理。通过对多个任务重用已经存在的线程对象 , 降低了对线程对象创建和销毁的开销。当客户请求 时 , 线程对象 已 经 存 在 , 可 以 提 高 请 求 的响应时间 , 从而整体地提高了系统服务的表现。**

### 2. 线程池的实现

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <unistd.h>**

**#include <sys/types.h>**

**#include <pthread.h>**

**#include <assert.h>**

**/\***

**\*线程池里所有运行和等待的任务都是一个CThread\_worker**

**\*由于所有任务都在链表里，所以是一个链表结构**

**\*/**

**typedef struct worker**

**{**

**/\*回调函数，任务运行时会调用此函数，注意也可声明成其它形式\*/**

**void \*(\*process) (void \*arg);**

**void \*arg;/\*回调函数的参数\*/**

**struct worker \*next;**

**} CThread\_worker;**

**/\*线程池结构\*/**

**typedef struct**

**{**

**pthread\_mutex\_t queue\_lock;**

**pthread\_cond\_t queue\_ready;**

**/\*链表结构，线程池中所有等待任务\*/**

**CThread\_worker \*queue\_head;**

**/\*是否销毁线程池\*/**

**int shutdown;**

**pthread\_t \*threadid;**

**/\*线程池中允许的活动线程数目\*/**

**int max\_thread\_num;**

**/\*当前等待队列的任务数目\*/**

**int cur\_queue\_size;**

**} CThread\_pool;**

**int pool\_add\_worker (void \*(\*process) (void \*arg), void \*arg);**

**void \*thread\_routine (void \*arg);**

**static CThread\_pool \*pool = NULL;**

**void pool\_init (int max\_thread\_num)**

**{**

**pool = (CThread\_pool \*) malloc (sizeof (CThread\_pool));**

**pthread\_mutex\_init (&(pool->queue\_lock), NULL);**

**pthread\_cond\_init (&(pool->queue\_ready), NULL);**

**pool->queue\_head = NULL;**

**pool->max\_thread\_num = max\_thread\_num;**

**pool->cur\_queue\_size = 0;**

**pool->shutdown = 0;**

**pool->threadid =(pthread\_t \*) malloc (max\_thread\_num \* sizeof (pthread\_t));**

**int i = 0;**

**for (i = 0; i < max\_thread\_num; i++)**

**{**

**pthread\_create (&(pool->threadid[i]), NULL, thread\_routine,NULL);**

**}**

**}**

**/\*向线程池中加入任务\*/**

**int pool\_add\_worker (void \*(\*process) (void \*arg), void \*arg)**

**{**

**/\*构造一个新任务\*/**

**CThread\_worker \*newworker =**

**(CThread\_worker \*) malloc (sizeof (CThread\_worker));**

**newworker->process = process;**

**newworker->arg = arg;**

**newworker->next = NULL;/\*别忘置空\*/**

**pthread\_mutex\_lock (&(pool->queue\_lock));**

**/\*将任务加入到等待队列中\*/**

**CThread\_worker \*member = pool->queue\_head;**

**if (member != NULL)**

**{**

**while (member->next != NULL)**

**member = member->next;**

**member->next = newworker;**

**}**

**else**

**{**

**pool->queue\_head = newworker;**

**}**

**assert (pool->queue\_head != NULL);**

**pool->cur\_queue\_size++;**

**pthread\_mutex\_unlock (&(pool->queue\_lock));**

**/\*好了，等待队列中有任务了，唤醒一个等待线程；**

**注意如果所有线程都在忙碌，这句没有任何作用\*/**

**pthread\_cond\_signal (&(pool->queue\_ready));**

**return 0;**

**}**

**/\*销毁线程池，等待队列中的任务不会再被执行，但是正在运行的线程会一直**

**把任务运行完后再退出\*/**

**int pool\_destroy ()**

**{**

**if (pool->shutdown)**

**return -1;/\*防止两次调用\*/**

**pool->shutdown = 1;**

**/\*唤醒所有等待线程，线程池要销毁了\*/**

**pthread\_cond\_broadcast (&(pool->queue\_ready));**

**/\*阻塞等待线程退出，否则就成僵尸了\*/**

**int i;**

**for (i = 0; i < pool->max\_thread\_num; i++)**

**pthread\_join (pool->threadid[i], NULL);**

**free (pool->threadid);**

**/\*销毁等待队列\*/**

**CThread\_worker \*head = NULL;**

**while (pool->queue\_head != NULL)**

**{**

**head = pool->queue\_head;**

**pool->queue\_head = pool->queue\_head->next;**

**free (head);**

**}**

**/\*条件变量和互斥量也别忘了销毁\*/**

**pthread\_mutex\_destroy(&(pool->queue\_lock));**

**pthread\_cond\_destroy(&(pool->queue\_ready));**

**free (pool);**

**/\*销毁后指针置空是个好习惯\*/**

**pool=NULL;**

**return 0;**

**}**

**void\* thread\_routine (void \*arg)**

**{**

**printf ("starting thread 0x%x\n", pthread\_self ());**

**while (1)**

**{**

**pthread\_mutex\_lock (&(pool->queue\_lock));**

**/\*如果等待队列为0并且不销毁线程池，则处于阻塞状态; 注意**

**pthread\_cond\_wait是一个原子操作，等待前会解锁，唤醒后会加锁\*/**

**while (pool->cur\_queue\_size == 0 && !pool->shutdown)**

**{**

**printf ("thread 0x%x is waiting\n", pthread\_self ());**

**pthread\_cond\_wait (&(pool->queue\_ready), &(pool->queue\_lock));**

**}**

**/\*线程池要销毁了\*/**

**if (pool->shutdown)**

**{**

**/\*遇到break,continue,return等跳转语句，千万不要忘记先解锁\*/**

**pthread\_mutex\_unlock (&(pool->queue\_lock));**

**printf ("thread 0x%x will exit\n", pthread\_self ());**

**pthread\_exit (NULL);**

**}**

**printf ("thread 0x%x is starting to work\n", pthread\_self ());**

**/\*assert是调试的好帮手\*/**

**assert (pool->cur\_queue\_size != 0);**

**assert (pool->queue\_head != NULL);**

**/\*等待队列长度减去1，并取出链表中的头元素\*/**

**pool->cur\_queue\_size--;**

**CThread\_worker \*worker = pool->queue\_head;**

**pool->queue\_head = worker->next;**

**pthread\_mutex\_unlock (&(pool->queue\_lock));**

**/\*调用回调函数，执行任务\*/**

**(\*(worker->process)) (worker->arg);**

**free (worker);**

**worker = NULL;**

**}**

**/\*这一句应该是不可达的\*/**

**pthread\_exit (NULL);**

**}**

**void\* myprocess (void \*arg)**

**{**

**printf ("threadid is 0x%x, working on task %d\n", pthread\_self (),\*(int \*) arg);**

**sleep (1);/\*休息一秒，延长任务的执行时间\*/**

**return NULL;**

**}**

**int main ()**

**{**

**pool\_init (3);/\*线程池中最多三个活动线程\*/**

**/\*连续向池中投入10个任务\*/**

**int \*workingnum = (int \*) malloc (sizeof (int) \* 10);**

**int i;**

**for (i = 0; i < 10; i++)**

**{**

**workingnum[i] = i;**

**pool\_add\_worker (myprocess, &workingnum[i]);**

**}**

**/\*等待所有任务完成\*/**

**sleep (5);**

**/\*销毁线程池\*/**

**pool\_destroy ();**

**free (workingnum);**

**return 0;**

**}**

## 读写锁

读写锁实际是一种特殊的[自旋锁](https://baike.baidu.com/item/%E8%87%AA%E6%97%8B%E9%94%81)，它把对共享资源的访问者划分成读者和写者，读者只对共享资源进行读访问，写者则需要对共享资源进行写操作。这种锁相对于自旋锁而言，能提高并发性，因为在[多处理器系统](https://baike.baidu.com/item/%E5%A4%9A%E5%A4%84%E7%90%86%E5%99%A8%E7%B3%BB%E7%BB%9F)中，它允许同时有多个读者来访问共享资源，最大可能的读者数为实际的逻辑CPU数。写者是排他性的，一个读写锁同时只能有一个写者或多个读者（与CPU数相关），但不能同时既有读者又有写者。

在读写锁保持期间也是抢占失效的。

如果读写锁当前没有读者，也没有写者，那么写者可以立刻获得读写锁，否则它必须自旋在那里，直到没有任何写者或读者。如果读写锁没有写者，那么读者可以立即获得该读写锁，否则读者必须自旋在那里，直到写者释放该读写锁。

一次只有一个[线程](https://baike.baidu.com/item/%E7%BA%BF%E7%A8%8B)可以占有写模式的读写锁, 但是可以有多个线程同时占有读模式的读写锁. 正是因为这个特性,

当读写锁是写加锁状态时, 在这个锁被解锁之前, 所有试图对这个锁加锁的线程都会被阻塞.

当读写锁在读加锁状态时, 所有试图以读模式对它进行加锁的线程都可以得到访问权, 但是如果线程希望以写模式对此锁进行加锁, 它必须直到所有的线程释放锁.

通常, 当读写锁处于读模式锁住状态时, 如果有另外线程试图以写模式加锁, 读写锁通常会阻塞随后的读模式锁请求, 这样可以避免读模式锁长期占用, 而等待的写模式锁请求长期阻塞.

读写锁适合于对[数据结构](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84/1450)的读次数比写次数多得多的情况. 因为, 读模式锁定时可以共享, 以写模式锁住时意味着独占, 所以读写锁又叫共享-独占锁.

#include <pthread.h>

int pthread\_rwlock\_init(pthread\_rwlock\_t \*restrict rwlock, const pthread\_rwlockattr\_t \*restrict attr);

int pthread\_rwlock\_destroy(pthread\_rwlock\_t \*rwlock);

成功则返回0, 出错则返回错误编号.

同[互斥量](https://baike.baidu.com/item/%E4%BA%92%E6%96%A5%E9%87%8F)以上, 在释放读写锁占用的内存之前, 需要先通过pthread\_rwlock\_destroy对读写锁进行清理工作, 释放由init分配的资源.

#include <pthread.h>

int pthread\_rwlock\_rdlock(pthread\_rwlock\_t \*rwlock);

int pthread\_rwlock\_wrlock(pthread\_rwlock\_t \*rwlock);

int pthread\_rwlock\_unlock(pthread\_rwlock\_t \*rwlock);

成功则返回0, 出错则返回错误编号.

这3个函数分别实现获取读锁, 获取写锁和释放锁的操作. 获取锁的两个函数是阻塞操作, 同样, 非阻塞的函数为:

#include <pthread.h>

int pthread\_rwlock\_tryrdlock(pthread\_rwlock\_t \*rwlock);

int pthread\_rwlock\_trywrlock(pthread\_rwlock\_t \*rwlock);

成功则返回0, 出错则返回错误编号.

非阻塞的获取锁操作, 如果可以获取则返回0, 否则返回错误的EBUSY.

# C 语言

## [结构体在内存中的对齐规则](https://www.cnblogs.com/zhoujiayi/p/7872262.html)

一个结构体变量定义完之后，其在内存中的存储并不等于其所包含元素的宽度之和

**1：元素为基本数据类型的结构体**

例一：

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif 结构体定义

例一中的结构体变量S1定义之后，经测试，会发现：

sizeof(S1）= 16，sizeof(S1.a) = 1，sizeof(S1.b) = 4， sizeof(S1.c) = 8

原则一：结构体中元素是按照定义顺序一个一个放到内存中去的，但并不是紧密排列的。

从结构体存储的首地址开始，每一个元素放置到内存中时，它都会认为内存是以它自己的大小

来划分的，因此元素放置的位置一定会在自己宽度的整数倍上开始（以结构体变量首地址为0计算）

比如此例，首先系统会将字符型变量a存入第0个字节（相对地址，指内存开辟的首地址）；

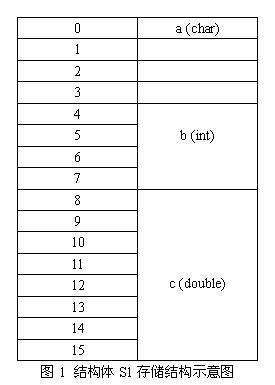
然后在存放整形变量b时，会以4个字节为单位进行存储，由于第一个四字节模块已有数据，

因此它会存入第二个四字节模块，也就是存入到4~8字节；

同理，存放双精度实型变量c时，由于其宽度为8，其存放时会以8个字节为单位存储，

也就是会找到第一个空的且是8的整数倍的位置开始存储。

此例中，由于头一个8字节模块已被占用，所以将c存入第二个8字节模块。整体存储示意图如图1所示：



考虑另外一个实例，例二：

struct X

{

char a;

double b;

int c;

}S2;

将double型的变量和int型的变量互换位置。测试程序不变，测试结果却截然不同，sizeof(S2)=24，

不同于我们按照原则一计算出的8+8+4=20，这就引出了我们的第二原则。

原则二：在经过第一原则分析后，检查计算出的存储单元是否为所有元素中最宽的元素的长度的整数倍，

是，则结束；若不是，则补齐为它的整数倍。

例二中，我们分析完后的存储长度为20字节，不是最宽元素长度8的整数倍，因此将它补齐到8的整数倍，

也就是24。这样就没问题了。其存储示意图如图2所示。



**2：元素包含指针类型的的结构体**

原则：只要记住指针本身所占的存储空间是4个字节就行了，而不必看它是指向什么类型的指针

struct X            struct Y            struct Z  
{                  {                  {  
  char \*a;            int \*b;            double \*c;  
};                 };                 };

经测试，可知sizeof(X)、sizeof(Y)和sizeof(Z)的值都为4。

**3：元素包含其他结构体类型的的结构体**

struct X

{

char a;

int b;

double c;

};

struct Y

{

char a;

X b;

};

经测试，可知sizeof(X)=16，sizeof(Y)=24。

即计算Y的存储长度时，在存放第二个元素b时的初始位置是在double型的长度8的整数倍处，而非16的整数倍处，

即系统为b所分配的存储空间是第8~23个字节。

如果将Y的两个元素char型的a和X型的b调换定义顺序，则系统为b分配的存储位置是第0~15个字节，

为a分配的是第16个字节，加起来一共17个字节，不是最长基本类型double所占宽度8的整数倍，

因此要补齐到8的整数倍，即24。测试后可得sizeof(Y)的值为24。

由于结构体所占空间与其内部元素的类型有关，而且与不同类型元素的排列有关，因此在定义结构体时，

在元素类型及数量确定之后，我们还应该注意一下其内部元素的定义顺序。

## FD\_CLOEXEC标志作用

通过fcntl设置FD\_CLOEXEC标志有什么用？  
close on exec, not on-fork, 意为如果对描述符设置了FD\_CLOEXEC，使用execl执行的程序里，此描述符被关闭，不能再使用它，但是在使用fork调用的子进程中，此描述符并不关闭，仍可使用。  
eg:  
jamie@jamie-laptop:~$ cat test.c  
#include <fcntl.h>  
#include <unistd.h>  
#include <stdio.h>  
#include <string.h>  
  
int main(void)  
{  
        int fd,pid;  
        char buffer[20];  
        fd=open("wo.txt",O\_RDONLY);  
        printf("%d/n",fd);  
        int val=fcntl(fd,F\_GETFD);  
        val|=FD\_CLOEXEC;  
        fcntl(fd,F\_SETFD,val);  
  
        pid=fork();  
        if(pid==0)  
        {  
                //子进程中，此描述符并不关闭，仍可使用  
                char child\_buf[2];  
                memset(child\_buf,0,sizeof(child\_buf) );  
                ssize\_t bytes = read(fd,child\_buf,sizeof(child\_buf)-1 );  
                printf("child, bytes:%d,%s/n/n",bytes,child\_buf);  
  
                //execl执行的程序里，此描述符被关闭，不能再使用它  
                char fd\_str[5];  
                memset(fd\_str,0,sizeof(fd\_str));  
                sprintf(fd\_str,"%d",fd);  
                int ret = execl("./exe1","exe1",fd\_str,NULL);  
                if(-1 == ret)  
                        perror("ececl fail:");  
        }          
  
        waitpid(pid,NULL,0);  
        memset(buffer,0,sizeof(buffer) );  
        ssize\_t bytes = read(fd,buffer,sizeof(buffer)-1 );  
        printf("parent, bytes:%d,%s/n/n",bytes,buffer);  
}  
  
jamie@jamie-laptop:~$ cat exe1.c  
#include <fcntl.h>  
#include <stdio.h>  
#include <assert.h>  
#include <string.h>  
  
int main(int argc, char \*\*args)  
{  
        char buffer[20];  
        int fd = atoi(args[1]);  
        memset(buffer,0,sizeof(buffer) );  
        ssize\_t bytes = read(fd,buffer,sizeof(buffer)-1);  
        if(bytes < 0)  
        {  
                perror("exe1: read fail:");  
                return -1;  
        }  
        else  
        {  
                printf("exe1: read %d,%s/n/n",bytes,buffer);  
        }  
        return 0;  
}  
  
jamie@jamie-laptop:~$ gcc -o exe1 exe1.c  
jamie@jamie-laptop:~$ gcc -o test test.c  
jamie@jamie-laptop:~$ cat wo.txt  
this is a test  
jamie@jamie-laptop:~$ ./test  
3  
child, bytes:1,t                               //子进程中可使用fd  
  
exe1: read fail:: Bad file descriptor  //execl调用的程序中不能使用fd  
parent, bytes:14,his is a test          //父进程中当然能使用fd

## 声明与定义的区别

数组：保存数据；直接访问数据；用于存储数目固定且类型相同的数据；由编译器自动分配和删除；自身即为数据名。  
   指针：保存地址；间接访问数据（先取得指针的内容，然后以它为地址，取得数据）；通常用于动态数据结构；动态的分配和删除；通常指向隐式数据。

首先呢我们必须明白一个概念在C语言中，一个变量的声明和定义有什么区别。

我们知道定义只是一个特殊的声明。

定义：只能出现在一个地方，创建新对象，同时确定对象的类型并分配内存。(注意这里所说的对象跟C++中所说的对象没有任何的关系。)

声明：可以出现多次。 描述对象的类型，用于指代其他地方定义的对象。它所说明的并非本身。

extern 对象声明告诉编译器对象的类型和名字，对象的内存分配则在别处进行，由于并未在声明中为数组分配内存，所以并不需要提供关于数组长度的信息。

所以下来我们看一个例子：

文件1：int mango[100];

文件2：exitern int \*mango;

在我们运行时总是会出错，为什么呢？

首先我们在看一下内存对数组和指针的引用：

比如说：char a[9]="abcdefgh"; c=a[i];

编译器符号表具有一个地址 9980

运行时第一步：取i的值，将它与9980相加

运行时第二步：取地址（9980+i）的值。

但是当:char \*a="abcdefgh";     c=a[i];时

编译器符号表具有一个a，地址为4624

运行时第一步：取地址 4624的内容，如：5081(注意这里：因为p是一个指针，指针我们知道它存的是地址（也就是4624单元内的内容仍然是一个地址），而相对于本题来说它所存的地址就是数组的地址。)

运行时第二步：去i的值，并将它与5081相加。

运行时第三步：取地址【5081+1】的值。

我们很容易看到上面对数组内存是直接访问的，而对于指针是间接访问的。(虽然最终都可以取得所要的元素，但是访问途径是不一样的！)

当extern char \*a;然后用a[i]来引用其中的一个元素时，就是对:char \*a="abcdefgh";     c=a[i];的访问步骤。

所以既然把p声明为指针，那么不管p原先是定义为指针还是数组，都会按照指针形式访问元素的，但是只有当p原来定义为指针时这个方法才是有效的。

然后我们分析开始的那个为什么运行时总是出错？mango 被声明为extern char \*mango,而它原先的定义却是char mango[100],这种情形，当用mango[i]这种形式提取这个声明的内容时，实际上得到的是一个字符。但按照上面的说法，编译器却把它当成是一个指针，把ACSII码解释为地址，显然是牛头不对马嘴么。

解决方案很容易：即声明为:extern int mango[];

什么时候数组和指针可以相同呢？

所有作为函数参数的数组名总是可以通过编译器转换为指针。在其他所有情况下（最有趣的情况就是“在一个文件中定义为数组，在另一个文件中声明为指针”），数组的声明就是数组，指针的声明就是指针，两者不能混淆。但在是用数组（在语句或表达式中引用）时，数组总是可以写成指针的形式，两者可以互换。

对编译器而言，一个数组就是一个地址，一个指针就是一个地址的地址。

总结一下相同点的地方和不同点的地方：

声明方面：1、extern 如extern char a[];不能写成指针形式。

    2、定义，如 char a[10];不能该写成指针形式。

      3、函数的参数，如fun(char a[])或fun(char \*a)随便写，因为当数组作为函数

参数时，编译都将把它转化为指针形式处理。

在表达式中使用：如c=a[i]也可以随便写，数组形式或指针形式都可以。

标准C规定了三条规则：数组与指针相同

1、表达式中的数组名（与声明不同）白编译器当作是一个指向该数组的第一个元素的指针。

2、下标总是与指针的偏移量相同。

3、在函数参数的声明中，数组名被编译器当作指向该数组第一个元素的指针。

在表达式中，指针和数组是可以互换的，因为他们在编译器里的最终形式都是指针，并且都可以进行取下标运算。当然也有几个极少见的例外：在下列情况下，对数组的引用不能用指向该数组第一个元素的指针来代替：

（1）、数组作为sizeof（）的操作数显然此时需要的是整个数组的长度，而不是指向第一个元素的指针。

（2）、使用&取数组的地址，它所取的是整体数组的一个地址。

（3）、数组是一个字符串常量初始值时。

C语言把数组小标该写成指针偏移量的根本原因就是指针和偏移量是底层硬件所使用的基本模型。

标准规定作为“类型的数组”的形参的声明应该调整为“类型的指针”。在函数形惨定义这个特殊情况下，编译器必须把数组形式改写成指向数组第一个元素的指针形式，编译器只向函数传递数组的地址，而不是整个数组的拷贝。

把作为形惨的数组和指针等同起来是处于效率原因的考虑。

在C语言中，所有非数组形式的数据实参均以传值形式（对实参作一份拷贝传递给调用的函数，函数不能修改作为实参的实际变量的值，而是只能修改传递给他的那份拷贝）调用。然而，如果要拷贝整个数组，无论是在时间上还是空间上的开销都是很大的，而且在大多数情况下你并不需要拷贝整个数组，只要告诉函数那个地址就可以了。

## asset宏

　assert宏的原型定义在assert.h中，其作用是如果它的条件返回错误，则终止程序执行. 　　  
assert的作用是现计算表达式 expression ，如果其值为假（即为0），那么它先向stderr打印一条出错信息,然后通过调用 abort 来终止程序运行。

 使用assert的缺点是，频繁的调用会极大的影响程序的性能，增加额外的开销。   
　　在调试结束后，可以通过在包含#include 的语句之前插入 #define NDEBUG 来禁用assert调用，示例代码如下：

1 #include   
2 #define NDEBUG   
3 #include

## const与define的区别

const修饰的变量会分配内存空间的

define只是编译预处理，符号替换

const 比define安全

例如

#define PI 3,14 + 3.14

int a = PI \* PI

展开后变成 int a = 3.14 + 3.14 \* 3.14 + 3.14

如果是const int PI=3.14 + 3.14

展开后是：int a =( 3.14+3.14) \* ( 3.14+3.14)

## VSS，RSS的含义

ps aux|head -1中VSS，RSS字段的含义

VSS - Virtual Set Size 虚拟耗用内存（包含共享库占用的内存）

**RSS** - Resident Set Size 实际使用物理内存（包含共享库占用的内存）

**PSS** - Proportional Set Size 实际使用的物理内存（比例分配共享库占用的内存）

**USS** - Unique Set Size 进程独自占用的物理内存（不包含共享库占用的内存）

## ubuntu系统单用户模式下重置root密码

1. 启动时候一直按住shift键

以前的Ubuntu版本，启动grub的时候，有一个grub loading，press ESC to get the menu的提示，所以按ESC就可以看到grub界面

现在的Ubuntu没有了，现在的做法是，启动的时候按住shift键，就可以看到grub界面了。

（接下来就修改启动参数，加入single，然后按ctrl+x启动，就能看到一个ncurse的界面，选择drop to root prompt，就可以得到一个root的terminal然后就可以重新设置root口令了）

2. 修改grub文件来完成

点击(此处)折叠或打开

1.sudo gedit /etc/default/grub

修改如下：

GRUB\_HIDDEN\_TIMEOUT=0

#GRUB\_HIDDEN\_TIMEOUT\_QUIET=true

GRUB\_TIMEOUT=0

2.sudo update-grub

下面展开说明：

首先修改 /etc/default/grub.cfg文件，参考以下config文件注释掉GRUB\_HIDDEN\_TIMEOUT=0

点击(此处)折叠或打开

# If you change this file, run 'update-grub' afterwards to update

# /boot/grub/grub.cfg.

# For full documentation of the options in this file, see:

# info -f grub -n 'Simple configuration'

GRUB\_DEFAULT=0

#GRUB\_HIDDEN\_TIMEOUT=0

GRUB\_HIDDEN\_TIMEOUT\_QUIET=true

GRUB\_TIMEOUT=10

GRUB\_DISTRIBUTOR=`lsb\_release -i -s 2> /dev/null || echo Debian`

GRUB\_CMDLINE\_LINUX\_DEFAULT="quiet splash"

GRUB\_CMDLINE\_LINUX=""

# Uncomment to enable BadRAM filtering, modify to suit your needs

# This works with Linux (no patch required) and with any kernel that obtains

# the memory map information from GRUB (GNU Mach, kernel of FreeBSD ...)

#GRUB\_BADRAM="0x01234567,0xfefefefe,0x89abcdef,0xefefefef"

# Uncomment to disable graphical terminal (grub-pc only)

#GRUB\_TERMINAL=console

# The resolution used on graphical terminal

# note that you can use only modes which your graphic card supports via VBE

# you can see them in real GRUB with the command `vbeinfo'

#GRUB\_GFXMODE=640x480

# Uncomment if you don't want GRUB to pass "root=UUID=xxx" parameter to Linux

#GRUB\_DISABLE\_LINUX\_UUID=true

# Uncomment to disable generation of recovery mode menu entries

#GRUB\_DISABLE\_RECOVERY="true"

# Uncomment to get a beep at grub start

#GRUB\_INIT\_TUNE="480 440 1"

然后执行"sudo update-grub"。这样以后在grub.cfg中会多出以下代码

点击(此处)折叠或打开

### BEGIN /etc/grub.d/30\_os-prober ###

if [ "x${timeout}" != "x-1" ]; then

if keystatus; then

if keystatus --shift; then

set timeout=-1

else

set timeout=0

fi

else

if sleep --interruptible 3 ; then

set timeout=0

fi

fi

fi

### END /etc/grub.d/30\_os-prober ###

最后简单介绍下其中的几个参数

点击(此处)折叠或打开

1.GRUB\_HIDDEN\_TIMEOUT=0

此配置将影响菜单显示。若设置此选项,将在此时间内隐藏菜单而显示引导画面。

菜单将会被隐藏,除非在此行开头加上一个 # 符号。(# GRUB\_HIDDEN\_TIMEOUT=0)。

GRUB 2 第一次执行时将会寻找其他操作系统。若没有其他操作系统被检测到,菜单将会配置为隐藏。若辨认出其他操作系统,菜单将会显示。

若是大于 0 的整数,系统将会依此配置的秒数暂停,但不会显示菜单。

0 则菜单不会显示,也不会有延迟。

使用者可以在启动时按住 SHIFT 键不放以强制显示菜单。

启动过程中,系统将会检查 SHIFT 键状态。若无法辨识按键状态,会有一个短时间的延迟让使用者可通过按下 ESC 键来显示菜单。

2.GRUB\_HIDDEN\_TIMEOUT\_QUIET=true

true 不显示倒计时。屏幕将会是空白的。

false 在 GRUB\_HIDDEN\_TIMEOUT 中配置的时间,空白屏幕上会有一个倒数计时器。

3.GRUB\_TIMEOUT=10

此命令将顺从 GRUB\_HIDDEN\_TIMEOUT 配置,除非 GRUB\_HIDDEN\_TIMEOUT 被注释掉(#)。若 GRUB\_HIDDEN\_TIMEOUT 启用,则当菜单显示时,GRUB\_TIMEOUT 将会只执行一次。

配置此值为 -1 将会导致菜单一直显示,直到用户选择。

GRUB 2 菜单默认为隐藏,除非其他操作系统被系统检测到。若没有其他操作系统,此行将会被注释掉,除非使用者修改它。为了在每次启动时显示菜单,去掉此行的注释并使用 1 或更大的值。

## [Linux写时拷贝技术(copy-on-write)](http://www.cnblogs.com/biyeymyhjob/archive/2012/07/20/2601655.html)

**COW技术初窥：**

在Linux程序中，fork（）会产生一个和父进程完全相同的子进程，但子进程在此后多会exec系统调用，出于效率考虑，linux中引入了“写时复制“技术，也就是只有进程空间的各段的内容要发生变化时，才会将父进程的内容复制一份给子进程。

那么子进程的物理空间没有代码，怎么去取指令执行exec系统调用呢？

在fork之后exec之前两个进程用的是相同的物理空间（内存区），子进程的代码段、数据段、堆栈都是指向父进程的物理空间，也就是说，两者的虚拟空间不同，但其对应的物理空间是同一个。当父子进程中有更改相应段的行为发生时，再为子进程相应的段分配物理空间，如果不是因为exec，内核会给子进程的数据段、堆栈段分配相应的物理空间（至此两者有各自的进程空间，互不影响），而代码段继续共享父进程的物理空间（两者的代码完全相同）。而如果是因为exec，由于两者执行的代码不同，子进程的代码段也会分配单独的物理空间。

 在网上看到还有个细节问题就是，fork之后内核会通过将子进程放在队列的前面，以让子进程先执行，以免父进程执行导致写时复制，而后子进程执行exec系统调用，因无意义的复制而造成效率的下降。

**COW详述：**

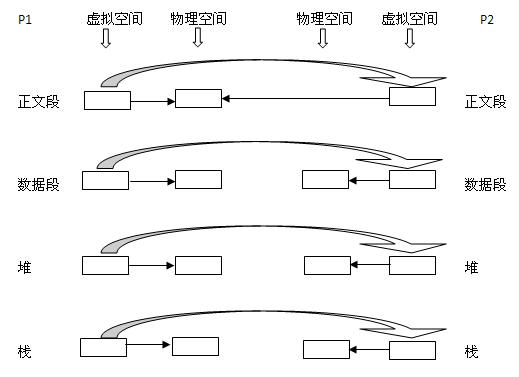
现在有一个父进程P1，这是一个主体，那么它是有灵魂也就身体的。现在在其虚拟地址空间（有相应的数据结构表示）上有：正文段，数据段，堆，栈这四个部分，相应的，内核要为这四个部分分配各自的物理块。即：正文段块，数据段块，堆块，栈块。至于如何分配，这是内核去做的事，在此不详述。

1.现在P1用fork()函数为进程创建一个子进程P2，

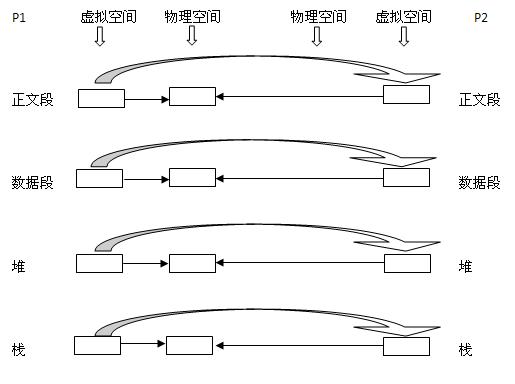
内核：

（1）复制P1的正文段，数据段，堆，栈这四个部分，注意是其内容相同。

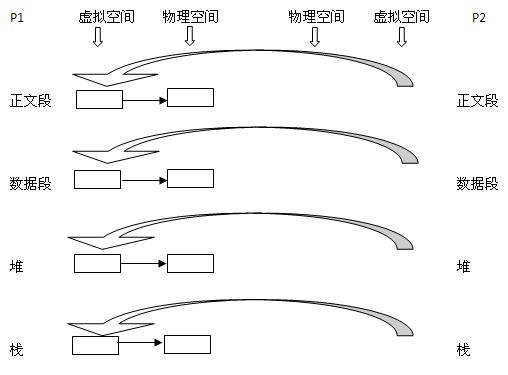
（2）为这四个部分分配物理块，P2的：正文段－＞PI的正文段的物理块，其实就是不为P2分配正文段块，让P2的正文段指向P1的正文段块，数据段－＞P2自己的数据段块（为其分配对应的块），堆－＞P2自己的堆块，栈－＞P2自己的栈块。如下图所示：同左到右大的方向箭头表示复制内容。



2.写时复制技术：内核只为新生成的子进程创建虚拟空间结构，它们来复制于父进程的虚拟空间结构，但是不为这些段分配物理内存，它们共享父进程的物理空间，当父子进程中有更改相应段的行为发生时，再为子进程相应的段分配物理空间。



3.vfork()：这个做法更加火爆，内核连子进程的虚拟地址空间结构也不创建了，直接共享了父进程的虚拟空间，当然了，这种做法就顺水推舟的共享了父进程的物理空间



通过以上的分析，相信大家对进程有个深入的认识，它是怎么一层层体现出自己来的，进程是一个主体，那么它就有灵魂与身体，系统必须为实现它创建相应的实体， 灵魂实体与物理实体。这两者在系统中都有相应的数据结构表示，物理实体更是体现了它的物理意义。以下援引LKD

传统的fork()系统调用直接把所有的资源复制给新创建的进程。这种实现过于简单并且效率低下，因为它拷贝的数据也许并不共享，更糟的情况是，如果新进程打算立即执行一个新的映像，那么所有的拷贝都将前功尽弃。Linux的fork()使用写时拷贝（copy-on-write）页实现。写时拷贝是一种可以推迟甚至免除拷贝数据的技术。内核此时并不复制整个进程地址空间，而是让父进程和子进程共享同一个拷贝。只有在需要写入的时候，数据才会被复制，从而使各个进程拥有各自的拷贝。也就是说，资源的复制只有在需要写入的时候才进行，在此之前，只是以只读方式共享。这种技术使地址空间上的页的拷贝被推迟到实际发生写入的时候。在页根本不会被写入的情况下—举例来说，fork()后立即调用exec()—它们就无需复制了。fork()的实际开销就是复制父进程的页表以及给子进程创建惟一的进程描述符。在一般情况下，进程创建后都会马上运行一个可执行的文件，这种优化可以避免拷贝大量根本就不会被使用的数据（地址空间里常常包含数十兆的数据）。由于Unix强调进程快速执行的能力，所以这个优化是很重要的。这里补充一点：**Linux COW与exec没有必然联系**

PS：实际上COW技术不仅仅在Linux进程上有应用，其他例如C++的String在有的IDE环境下也支持COW技术，即例如：

string str1 = "hello world";

string str2 = str1;

之后执行代码:

str1[1]='q';

str2[1]='w';

在开始的两个语句后，str1和str2存放数据的地址是一样的，而在修改内容后，str1的地址发生了变化，而str2的地址还是原来的,这就是C++中的COW技术的应用，不过VS2005似乎已经不支持COW。

## freopen输入输出重定向函数

freopen是被包含于C标准库头文件<stdio.h>中的一个函数，用于重定向输入输出流。该函数可以在不改变代码原貌的情况下改变输入输出环境，但使用时应当保证流是可靠的。

FILE \*freopen(const char \* restrict filename, const char \* restrict mode, FILE \* restrict stream);

形参说明：

filename：需要重定向到的文件名或文件路径。

mode：代表文件访问权限的字符串。例如，"r"表示“只读访问”、"w"表示“只写访问”、"a"表示“追加写入”。

stream：需要被重定向的文件流。

返回值：如果成功，则返回该指向该输出流的文件指针，否则返回为NULL。

### 举例

这个例子实现了从stdout到一个文本文件的重定向。即，把输出到屏幕的文本输出到一个文本文件中。

#include <stdio.h>

int main(void)

{

int i;

if (freopen ("D:\\output.txt", "w", stdout) == NULL)

fprintf(stderr, "error redirecting stdout\n");

for (i = 0; i < 10; i++)

printf("%3d", i);

printf("\n");

fclose(stdout);

return 0;

}

编译运行一下，你会发现，十个数输出到了D盘根目录下文本文件output.txt中。

### 恢复文件流

当标准输出stdout被重定向到指定文件后，如何把它重定向回原来“默认”的输出设备（即显示器）呢？

C标准库的回复是：不支持。没有任何方法可以恢复原来的输出流。

那是否存在依赖具体平台的实现呢？存在。

在操作系统中，命令行控制台（即键盘或者显示器）被视为一个文件，既然是文件，那么就有“文件名”。由于历史原因，命令行控制台文件在DOS操作系统和Windows操作系统中的文件名为"CON"，在其它的操作系统（例如Unix、Linux、Mac OS X、Android等等）中的文件名为"/dev/tty"。

因此，在Windows中可以使用

freopen( "CON", "w", stdout );

其它操作系统中使用：

freopen( "/dev/tty", "w", stdout );

#### Windows代码举例

#include <stdio.h>

#include <stdlib.h>

int main(void)

{

FILE \*stream;

if ((stream = freopen("file.txt", "w", stdout)) == NULL)

exit(-1);

printf("this is stdout output\n");

stream = freopen("CON","w",stdout);

/\*stdout是向程序的末尾的控制台重定向\*/

printf("And now back to the console once again\n");

return 0;

}

#### 类Unix代码举例

#include <stdio.h>

#include <stdlib.h>

int main(void)

{

FILE \*stream;

if ((stream = freopen("file.txt", "w", stdout)) == NULL)

exit(-1);

printf("this is stdout output\n");

stream = freopen("/dev/tty","w",stdout);

/\*stdout是向程序的末尾的控制台重定向\*/

printf("And now back to the console once again\n");

return 0;

}

警告：在使用上诉方法在输入输出流间进行反复的重定向时，极有可能导致流指针得到不被期待的结果，是输入输出发生异常，所以如果需要在文件的输入输出和标准输入输出流之间进行切换建议使用fopen或者是C++标准的ifstream及ofstream

## [Linux 内存映射函数 mmap（）函数详解](http://blog.csdn.net/yangle4695/article/details/52139585)

mmap将一个文件或者其它对象映射进内存。文件被映射到多个页上，如果文件的大小不是所有页的大小之和，最后一个页不被使用的空间将会清零。mmap在用户空间映射调用系统中作用很大。  
头文件 <sys/mman.h>  
函数原型  
void\* mmap(void\* start,size\_t length,int prot,int flags,int fd,off\_t offset);

int munmap(void\* start,size\_t length);

mmap()[1] 必须以PAGE\_SIZE为单位进行映射，而内存也只能以页为单位进行映射，若要映射非PAGE\_SIZE整数倍的地址范围，要先进行内存对齐，强行以PAGE\_SIZE的倍数大小进行映射。

用法:

下面说一下内存映射的步骤:  
用open系统调用打开文件, 并返回描述符fd.  
用mmap建立内存映射, 并返回映射首地址指针start.  
对映射(文件)进行各种操作, 显示(printf), 修改(sprintf).  
用munmap(void \*start, size\_t lenght)关闭内存映射.  
用close系统调用关闭文件fd.

UNIX网络编程第二卷进程间通信对mmap函数进行了说明。该函数主要用途有三个：  
1、将一个普通文件映射到内存中，通常在需要对文件进行频繁读写时使用，这样用内存读写取代I/O读写，以获得较高的性能；  
2、将特殊文件进行匿名内存映射，可以为关联进程提供共享内存空间；  
3、为无关联的进程提供共享内存空间，一般也是将一个普通文件映射到内存中。  
  
函数：void \*mmap(void \*start,size\_t length,int prot,int flags,int fd,off\_t offsize);   
参数start：指向欲映射的内存起始地址，通常设为 NULL，代表让系统自动选定地址，映射成功后返回该地址。  
  
参数length：代表将文件中多大的部分映射到内存。  
  
参数prot：映射区域的保护方式。可以为以下几种方式的组合：  
PROT\_EXEC 映射区域可被执行  
PROT\_READ 映射区域可被读取  
PROT\_WRITE 映射区域可被写入  
PROT\_NONE 映射区域不能存取  
  
参数flags：影响映射区域的各种特性。在调用mmap()时必须要指定MAP\_SHARED 或MAP\_PRIVATE。  
MAP\_FIXED 如果参数start所指的地址无法成功建立映射时，则放弃映射，不对地址做修正。通常不鼓励用此旗标。  
MAP\_SHARED对映射区域的写入数据会复制回文件内，而且允许其他映射该文件的进程共享。  
MAP\_PRIVATE 对映射区域的写入操作会产生一个映射文件的复制，即私人的“写入时复制”（copy on write）对此区域作的任何修改都不会写回原来的文件内容。  
MAP\_ANONYMOUS建立匿名映射。此时会忽略参数fd，不涉及文件，而且映射区域无法和其他进程共享。  
MAP\_DENYWRITE只允许对映射区域的写入操作，其他对文件直接写入的操作将会被拒绝。  
MAP\_LOCKED 将映射区域锁定住，这表示该区域不会被置换（swap）。  
  
参数fd：要映射到内存中的文件描述符。如果使用匿名内存映射时，即flags中设置了MAP\_ANONYMOUS，fd设为-1。有些系统不支持匿名内存映射，则可以使用fopen打开/dev/zero文件，然后对该文件进行映射，可以同样达到匿名内存映射的效果。  
  
参数offset：文件映射的偏移量，通常设置为0，代表从文件最前方开始对应，offset必须是分页大小的整数倍。  
  
返回值：  
  
若映射成功则返回映射区的内存起始地址，否则返回MAP\_FAILED(－1)，错误原因存于errno 中。  
  
错误代码：  
  
EBADF 参数fd 不是有效的文件描述词  
EACCES 存取权限有误。如果是MAP\_PRIVATE 情况下文件必须可读，使用MAP\_SHARED则要有PROT\_WRITE以及该文件要能写入。  
EINVAL 参数start、length 或offset有一个不合法。  
EAGAIN 文件被锁住，或是有太多内存被锁住。  
ENOMEM 内存不足。  
  
系统调用mmap()用于共享内存的两种方式：  
（1）使用普通文件提供的内存映射：  
  
适用于任何进程之间。此时，需要打开或创建一个文件，然后再调用mmap()  
  
典型调用代码如下：  
  
fd=open(name, flag, mode); if(fd<0) ...  
  
ptr=mmap(NULL, len , PROT\_READ|PROT\_WRITE, MAP\_SHARED , fd , 0);   
  
通过mmap()实现共享内存的通信方式有许多特点和要注意的地方，可以参看UNIX网络编程第二卷。  
  
（2）使用特殊文件提供匿名内存映射：  
  
适用于具有亲缘关系的进程之间。由于父子进程特殊的亲缘关系，在父进程中先调用mmap()，然后调用 fork()。那么在调用fork()之后，子进程继承父进程匿名映射后的地址空间，同样也继承mmap()返回的地址，这样，父子进程就可以通过映射区 域进行通信了。注意，这里不是一般的继承关系。一般来说，子进程单独维护从父进程继承下来的一些变量。而mmap()返回的地址，却由父子进程共同维护。 对于具有亲缘关系的进程实现共享内存最好的方式应该是采用匿名内存映射的方式。此时，不必指定具体的文件，只要设置相应的标志即可。

## strcspn函数

### 简介

原型：size\_t strcspn(const char \*s, const char \* reject);

相关头文件：string.h

strcspn是由**str**ing **c**omplementary **sp**a**n**的缩写组成，表示获取字符串s起始位置起字符不在reject的跨度（长度）

### 功能

函数说明：strcspn()从参数s 字符串的开头计算连续的字符，而这些字符都完全不在参数reject 所指的字符串中。简单地说， 若strcspn()返回的数值为n，则代表字符串s 开头连续有n 个字符都不含字符串reject 内的字符。

返回值：返回字符串s 开头连续不含字符串reject 内的字符数目。

### 举例

#include <string.h>

main()

{

char \*str = "Linux was first developed for 386/486-based pcs. ";

printf("%d\n", strcspn(str, " "));

printf("%d\n", strcspn(str, "/-"));

printf("%d\n", strcspn(str, "1234567890"));

}

执行结果：

5 //只计算到" "的出现, 所以返回"Linux"的长度

33 //计算到出现"/"或"－", 所以返回到"6"的长度

30 // 计算到出现数字字符为止, 所以返回"3"出现前的长度

## [linux下core文件调试方法](http://www.cnblogs.com/li-hao/archive/2011/09/25/2190278.html)

在程序不寻常退出时，内核会在当前工作目录下生成一个core文件（是一个内存映像，同时加上调试信息）。使用gdb来查看core文件，可以指示出导致程序出错的代码所在文件和行数。

1.core文件的生成开关和大小限制  
 （1）使用ulimit -c命令可查看core文件的生成开关。若结果为0，则表示关闭了此功能，不会生成core文件。  
 （2）使用ulimit -c filesize命令，可以限制core文件的大小（filesize的单位为kbyte）。若ulimit -c unlimited，则表示core文件的大小不受限制。如果生成的信息超过此大小，将会被裁剪，最终生成一个不完整的core文件。在调试此core文件的时候，gdb会提示错误。

2.core文件的名称和生成路径  
core文件生成路径:  
输入可执行文件运行命令的同一路径下。  
若系统生成的core文件不带其它任何扩展名称，则全部命名为core。新的core文件生成将覆盖原来的core文件。

（1）/proc/sys/kernel/core\_uses\_pid可以控制core文件的文件名中是否添加pid作为扩展。文件内容为1，表示添加pid作为扩展名，生成的core文件格式为core.xxxx；为0则表示生成的core文件同一命名为core。  
可通过以下命令修改此文件：  
echo "1" > /proc/sys/kernel/core\_uses\_pid

（2）proc/sys/kernel/core\_pattern可以控制core文件保存位置和文件名格式。  
可通过以下命令修改此文件：  
echo "/corefile/core-%e-%p-%t" > core\_pattern，可以将core文件统一生成到/corefile目录下，产生的文件名为core-命令名-pid-时间戳  
以下是参数列表:  
    %p - insert pid into filename 添加pid  
    %u - insert current uid into filename 添加当前uid  
    %g - insert current gid into filename 添加当前gid  
    %s - insert signal that caused the coredump into the filename 添加导致产生core的信号  
    %t - insert UNIX time that the coredump occurred into filename 添加core文件生成时的unix时间  
    %h - insert hostname where the coredump happened into filename 添加主机名  
    %e - insert coredumping executable name into filename 添加命令名

 3.core文件的查看  
 core文件需要使用gdb来查看。  
 gdb ./a.out  
 core-file core.xxxx  
 使用bt命令即可看到程序出错的地方。

 4.开发板上使用core文件调试  
如果开发板的操作系统也是linux，core调试方法依然适用。如果开发板上不支持gdb，可将开发板的环境（依赖库）、可执行文件和core文件拷贝到PC的linux下。  
在 PC上调试开发板上产生的core文件，需要使用交叉编译器自带的gdb，并且需要在gdb中指定solib-absolute-prefix和 solib-search-path两个变量以保证gdb能够找到可执行程序的依赖库路径。有一种建立配置文件的方法，不需要每次启动gdb都配置以上变量，即：在待运行gdb的路径下建立.gdbinit。  
配置文件内容：  
set solib-absolute-prefix YOUR\_CROSS\_COMPILE\_PATH  
set solib-search-path YOUR\_CROSS\_COMPILE\_PATH  
set solib-search-path YOUR\_DEVELOPER\_TOOLS\_LIB\_PATH  
handle SIG32 nostop noprint pass

注意：待调试的可执行文件，在编译的时候需要加-g，core文件才能正常显示出错信息！有时候core信息很大，超出了开发板的空间限制，生成的core信息会残缺不全而无法使用，可以通过挂载到PC的方式来规避这一点。

## 全局变量宏定义方法

假如：

a.c是含有main函数的c文件其中需要定义全局变量int g\_iOpen

b.c、和c.c程序文件中需要引用该全局变量g\_iOpen

通用的做法是

1、定义个头文件g.h

在头文件中定义如下内容

#ifdef \_MODULE\_MIAN\_

int g\_iOpen = 0;//定义全局变量

#else

extern g\_iOpen = 0;//声明外部变量

#endif

2、在含有main函数的程序文件a.c中定义宏

\_MODULE\_MIAN\_

#define \_MODULE\_MIAN\_

3、在需要用到全局变量的b.c,c.c文件中包含头文件g.h

#include “g.h”

## 结构体位域用法

有些信息在存储时，并不需要占用一个完整的字节， 而只需占几个或一个二进制位。例如在存放一个开关量时，只有0和1 两种状态，用一位二进位即可。为了节省存储空间，并使处理简便，Ｃ语言又提供了一种数据结构，称为“位域”或“位段”。所谓“位域”是把一个字节中的二进位划分为几个不同的区域，并说明每个区域的位数。每个域有一个域名，允许在程序中按域名进行操作。 这样就可以把几个不同的对象用一个字节的二进制位域来表示。  
一、位域的定义和位域变量的说明  
位域定义与结构定义相仿，其形式为：  
struct 位域结构名  
{ 位域列表 };  
其中位域列表的形式为： 类型说明符 位域名：位域长度  
例如：  
struct bs  
{  
    int a:8;  
    int b:2;  
    int c:6;  
};  
位域变量的说明与结构变量说明的方式相同。 可采用先定义后说明，同时定义说明或者直接说明这三种方式。例如：  
struct bs  
{  
    int a:8;  
    int b:2;  
    int c:6;  
}data;  
说明data为bs变量，共占两个字节。其中位域a占8位，位域b占2位，位域c占6位。对于位域的定义尚有以下几点说明：  
1. 一个位域必须存储在同一个字节中，不能跨两个字节。如一个字节所剩空间不够存放另一位域时，应从下一单元起存放该位域。也可以有意使某位域从下一单元开始。  
例如：  
struct bs  
{  
    unsigned a:4;  
    unsigned :0;  
    unsigned b:4;  
    unsigned c:4;  
}  
在这个位域定义中，a占第一字节的4位，后4位填0表示不使用，b从第二字节开始，占用  
4位，c占用4位。  
2. 由于位域不允许跨两个字节，因此位域的长度不能大于一个字节的长度，也就是说不能超过8位二进位。  
3. 位域可以无位域名，这时它只用来作填充或调整位置。无名的位域是不能使用的。例如：  
struct k  
{  
    int a:1;  
    int :2;  
    int b:3;  
    int c:2;  
};  
从以上分析可以看出，位域在本质上就是一种结构类型， 不过其成员是按二进位分配的。  
  
二、位域的使用  
位域的使用和结构成员的使用相同，其一般形式为：  
位域变量名·位域名  
位域允许用各种格式输出。  
main(){  
    struct bs  
        {  
             unsigned a:1;  
             unsigned b:3;  
             unsigned c:4;  
        } bit,\*pbit;  
    bit.a=1;  
    bit.b=7;  
    bit.c=15;  
    printf("%d,%d,%d\n",bit.a,bit.b,bit.c);  
    pbit=&bit;  
    pbit->a=0;  
    pbit->b&=3;  
    pbit->c|=1;  
    printf("%d,%d,%d\n",pbit->a,pbit->b,pbit->c);  
}  
上例程序中定义了位域结构bs，三个位域为a,b,c。说明了bs类型的变量bit和指向bs类型的指针变量pbit。这表示位域也是可以使用指针的。程序的9、10、11三行分别给三个位域赋值。( 应注意赋值不能超过该位域的允许范围)程序第12行以整型量格式输出三个域的内容。第13行把位域变量bit的地址送给指针变量pbit。第14行用指针方式给位域a重新赋值，赋为0。第15行使用了复合的位运算符"&="，该行相当于：pbit->b=pbit->b&3位域b中原有值为7，与3作按位与运算的结果为3(111&011=011,十进制值为3)。同样，程序第16行中使用了复合位运算"|=".之所以要有透析基础知识这么个分栏,就是告诉大家重 在细节的道理,粗略的东西谁都懂,修炼内功为高手的必经之路.  
前面的内容存在缺陷，具体还要参考如下文章：  
C99规定int、unsigned int和bool可以作为位域类型，但编译器几乎都对此作了扩展，允许其它类型类型的存在。

使用位域的主要目的是压缩存储，其大致规则为：  
1) 如果相邻位域字段的类型相同，且其位宽之和小于类型的sizeof大小，则后面的字段将紧邻前一个字段存储，直到不能容纳为止；  
2) 如果相邻位域字段的类型相同，但其位宽之和大于类型的sizeof大小，则后面的字段将从新的存储单元开始，其偏移量为其类型大小的整数倍；  
3) 如果相邻的位域字段的类型不同，则各编译器的具体实现有差异，VC6采取不压缩方  
式，Dev-C++采取压缩方式；  
4) 如果位域字段之间穿插着非位域字段，则不进行压缩；  
5) 整个结构体的总大小为最宽基本类型成员大小的整数倍。

typedef struct  AA  
{  
       unsigned char b1:5;  
       unsigned char b2:5;  
       unsigned char b3:5;  
       unsigned char b4:5;  
       unsigned char b5:5;  
}AA;  
  
sizeof(AA) = 5;　但实际上只用了25位，即4个字节，

（1）typedef struct  AA  
{  
       unsigned int b1:5;  
       unsigned int b2:5;  
       unsigned int b3:5;  
       unsigned int b4:5;  
       unsigned int b5:5;  
}AA;

（2）typedef struct  AA  
{  
       unsigned int b1:5;  
       unsigned int b2:5;  
       unsigned int b3:5;  
       unsigned int b4:5;  
       unsigned int b5:5;  
       unsigned int b6:5;  
       unsigned int b7:5;  
}AA;  
  
（1）是5个成员,按第一条规则,共占25位,按第五条规则,即sizeof(AA)=4  
现把成员加到7个，参考（2）,按第一条规则,共占35位,按第五条规则,即sizeof(AA)=8,

再看一个例子：

 struct test1

{

char a:1;

char :2;

long b:3;

char c:2;

};

int len = sizeof(test1);

对于上述例子，len的值应该是12.解释如下：

首先以最长的类型位宽做为偏移量，最长的是long型，占4位，所以不同类型之间应该是4个字节的偏移，即test1应该是4字节的整数倍。

char a：1；　//用一个字节去存储

char ：2；　 //空域。因为与前面的a的类型相同，而两个位域的位宽相加仍然少于8位，所以依然用1个字节表示

long b：3； //long类型的位宽是4个字节，与前面的char类型不同，所以b与a之间偏移4个字节，它们之间自动补充3个字节

char c：2； //因为c与b又不同型，以test1中的最长的long类型的位宽进行偏移，所以虽然char只用1个字节就够了

//但依然要占4个字节。

总共是12字节。

///////////////////////

struct s1

　　{

　　int i: 8;

　　int j: 4;

　　int a: 3;

　　double b;

　　};

　　struct s2

　　{

　　int i: 8;

　　int j: 4;

　　double b;

　　int a:3;

　　};

　　printf("sizeof(s1)= %d\n", sizeof(s1));

　　printf("sizeof(s2)= %d\n", sizeof(s2));

　　result: 16, 24

　　第一个struct s1

　　{

　　int i: 8;

　　int j: 4;

　　int a: 3;

　　double b;

　　};

　　理论上是这样的，首先是i在相对0的位置，占8位一个字节，然后，j就在相对一个字节的位置，由于一个位置的字节数是4位的倍数，因此不用对齐，就放 在那里了，然后是a，要在3位的倍数关系的位置上，因此要移一位，在15位的位置上放下，目前总共是18位，折算过来是2字节2位的样子，由于 double是8 字节的，因此要在相对0要是8个字节的位置上放下，因此从18位开始到8个字节之间的位置被忽略，直接放在8字节的位置了，因此，总共是16字节。

## 共享内存加锁写法LockShm(int key)

int LockShm(int Key)

{

int fd;

static struct sembuf op\_lock[2]=

{

0, 0, 0,

0, 1, SEM\_UNDO

};

fd = semget(Key,1,0);

if (fd < 0)

{

return 报错

}

if（semop(fd, &op\_lock[0], 2) < 0）

{

return 报错;

}

return 0;

}

## 共享内存开锁写法UnLockShm(int key)

int UnLockShm(int Key)

{

int fd;

static struct sembuf op\_ulock[1]=

{

0, -1, (IPC\_NOWAIT|SEM\_UNDO)

};

fd = semget(Key,1,0);

if (fd < 0)

{

return 报错

}

if（semop(fd, &op\_ulock[0], 1) < 0）

{

return 报错;

}

return 0;

}

## errno对应的意义

strerror(errno)：获取errno对应的错误

#include <string.h>  
#include <errno.h> /\* for strerror \*/  
#include <stdio.h>  
  
int main(int argc, char \*\* argv)  
{  
 int i = 0;  
 for(i = 0; i < 256; i++)  
 printf("errno.%02d is: %s/n", i, strerror(i));  
 return 0;  
}

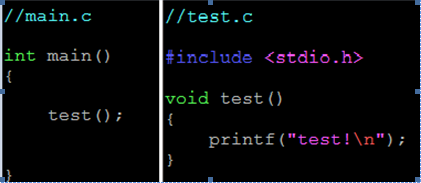
## [Linux中link，unlink，close，fclose详解](http://blog.csdn.net/cyberrusher/article/details/7423843)

每一个文件，都可以通过一个struct stat的结构体来获得文件信息，其中一个成员st\_nlink代表文件的链接数。  
struct stat {  
               dev\_t     st\_dev;     /\* ID of device containing file \*/  
               ino\_t     st\_ino;     /\* inode number \*/  
               mode\_t    st\_mode;    /\* protection \*/  
               nlink\_t   st\_nlink;   /\* number of hard links \*/  
               uid\_t     st\_uid;     /\* user ID of owner \*/  
               gid\_t     st\_gid;     /\* group ID of owner \*/  
               dev\_t     st\_rdev;    /\* device ID (if special file) \*/  
               off\_t     st\_size;    /\* total size, in bytes \*/  
               blksize\_t st\_blksize; /\* blocksize for file system I/O \*/  
               blkcnt\_t  st\_blocks;  /\* number of 512B blocks allocated \*/  
               time\_t    st\_atime;   /\* time of last access \*/  
               time\_t    st\_mtime;   /\* time of last modification \*/  
               time\_t    st\_ctime;   /\* time of last status change \*/  
           };  
      当通过shell的touch命令或者在程序中open一个带有O\_CREAT的不存在的文件时，文件的链接数为1。  
  
      通常open一个已存在的文件不会影响文件的链接数。open的作用只是使调用进程与文件之间建立一种访问关系，即open之后返回fd，调用进程可以通过fd来read、write 、 ftruncate等等一系列对文件的操作。  
      close()就是消除这种调用进程与文件之间的访问关系。自然，不会影响文件的链接数。在调用close时，内核会检查打开该文件的进程数，如果此数为0，进一步检查文件的链接数，如果这个数也为0，那么就删除文件内容。  
  
      link函数创建一个新目录项，并且增加一个链接数。  
      unlink函数删除目录项，并且减少一个链接数。如果链接数达到0并且没有任何进程打开该文件，该文件内容才被真正删除。如果在unlilnk之前没有close，那么依旧可以访问文件内容。  
    
      综上所诉，真正影响链接数的操作是link、unlink以及open的创建。  
      删除文件内容的真正含义是文件的链接数为0，而这个操作的本质完成者是unlink。close能够实施删除文件内容的操作，必定是因为在close之前有一个unlink操作。  
  
     举个例子简单说明：通过shell   touchtest.txt  
 1、stat("test.txt",&buf);  
     printf("1.link=%d\n",buf.st\_nlink);//未打开文件之前测试链接数  
  
 2、fd=open("test.txt",O\_RDONLY);//打开已存在文件test.txt  
     stat("test.txt",&buf);  
     printf("2.link=%d\n",buf.st\_nlink);//测试链接数  
  
 3、close(fd);//关闭文件test.txt  
     stat("test.txt",&buf);  
     printf("3.link=%d\n",buf.st\_nlink);//测试链接数  
  
 4、link("test.txt","test2.txt");//创建硬链接test2.txt  
      stat("test.txt",&buf);  
      printf("4.link=%d\n",buf.st\_nlink);//测试链接数  
  
 5、unlink("test2.txt");//删除test2.txt  
     stat("test.txt",&buf);  
     printf("5.link=%d\n",buf.st\_nlink);//测试链接数  
  
6、重复步骤2  //重新打开test.txt  
  
7、unlink("test.txt");//删除test.txt  
    fstat(fd,&buf);  
    printf("7.link=%d\n",buf.st\_nlink);//测试链接数  
  
8、close(fd);//此步骤可以不显示写出，因为进程结束时，打开的文件自动被关闭。  
  
   顺次执行以上8个步骤，结果如下：  
   1.link=1  
   2.link=1   //open不影响链接数  
   3.link=1   //close不影响链接数  
   4.link=2   //link之后链接数加1  
   5.link=1   //unlink后链接数减1  
   2.link=1   //重新打开  链接数不变  
   7.link=0   //unlink之后再减1，此处我们改用fstat函数而非stat，因为unlilnk已经删除文件名，所以不可以通过  文件名访问，但是fd仍然是打开着的，文件内容还没有被真正删除，依旧可以使用fd获得文件信息。  
   执行步骤8，文件内容被删除。。。。

## 编译时["undefined reference to" 问题解决方法](http://blog.csdn.net/aiwoziji13/article/details/7330333)

### ****1.链接时缺失了相关目标文件（.o）****

[测试](http://lib.csdn.net/base/softwaretest)代码如下：

[](http://img1.51cto.com/attachment/201011/133144914.png)

    然后编译。

1. gcc -c test.c
2. gcc –c main.c

 得到两个 .o 文件，一个是 main.o，一个是 test.o ，然后我们链接 .o 得到可执行程序：

gcc -o main main.o

这时，你会发现，报错了：

1. main.o: In function `main':
2. main.c:(.text+0x7): undefined reference to `test'
3. collect2: ld returned 1 exit status

 这就是最典型的undefined reference错误，因为在链接时发现找不到某个函数的实现文件，本例中test.o文件中包含了test()函数的实现，所以如果按下面这种方式链接就没事了。

gcc -o main main.o test.o

【扩展】：其实上面为了让大家更加清楚底层原因，我把编译链接分开了，下面这样编译也会报undefined reference错，其实底层原因与上面是一样的。

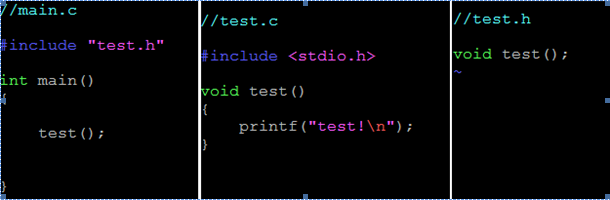
gcc -o main main.c //缺少test()的实现文件

需要改成如下形式才能成功，将test()函数的实现文件一起编译。

gcc -o main main.c test.c //ok,没问题了

### ****2.链接时缺少相关的库文件（.a/.so）****

在此，只举个静态库的例子，假设源码如下。

[](http://img1.51cto.com/attachment/201011/133734483.png)

 先把test.c编译成静态库(.a)文件

1. gcc -c test.c
2. ar -rc test.a test.o

 至此，我们得到了test.a文件。我们开始编译main.c

gcc -c main.c

这时，则生成了main.o文件，然后我们再通过如下命令进行链接希望得到可执行程序。

gcc -o main main.o

你会发现，编译器报错了：

1. /tmp/ccCPA13l.o: In function `main':
2. main.c:(.text+0x7): undefined reference to `test'
3. collect2: ld returned 1 exit status

  其根本原因也是找不到test()函数的实现文件，由于该test()函数的实现在test.a这个静态库中的，故在链接的时候需要在其后加入test.a这个库，链接命令修改为如下形式即可。

gcc -o main main.o ./test.a  //注：./ 是给出了test.a的路径

   【扩展】：同样，为了把问题说清楚，上面我们把代码的编译链接分开了，如果希望一次性生成可执行程序，则可以对main.c和test.a执行如下命令。

gcc -o main main.c ./test.a  //同样，如果不加test.a也会报错

### ****3.链接的库文件中又使用了另一个库文件****

  这种问题比较隐蔽，也是我最近遇到的与网上大家讨论的不同的问题，举例说明如下，首先，还是看看测试代码。

[](http://img1.51cto.com/attachment/201011/134157339.png)

 从上图可以看出，main.c调用了test.c的函数，test.c中又调用了fun.c的函数。  
 首先，我们先对fun.c，test.c，main.c进行编译，生成 .o文件。

1. gcc -c func.c
2. gcc -c test.c
3. gcc -c main.c

  然后，将test.c和func.c各自打包成为静态库文件。

1. ar –rc func.a func.o
2. ar –rc test.a test.o

  这时，我们准备将main.o链接为可执行程序，由于我们的main.c中包含了对test()的调用，因此，应该在链接时将test.a作为我们的库文件，链接命令如下。

1. gcc -o main main.o test.a

    这时，编译器仍然会报错，如下：

1. test.a(test.o): In function `test':
2. test.c:(.text+0x13): undefined reference to `func'
3. collect2: ld returned 1 exit status

  就是说，链接的时候，发现我们的test.a调用了func()函数，找不到对应的实现。由此我们发现，原来我们还需要将test.a所引用到的库文件也加进来才能成功链接，因此命令如下。

1. gcc -o main main.o test.a func.a

  ok，这样就可以成功得到最终的程序了。同样，如果我们的库或者程序中引用了第三方库（如pthread.a）则同样在链接的时候需要给出第三方库的路径和库文件，否则就会得到undefined reference的错误。

### ****4.多个库文件链接顺序问题****

 这种问题也非常的隐蔽，不仔细研究你可能会感到非常地莫名其妙。我们依然回到第3小节所讨论的问题中，在最后，如果我们把链接的库的顺序换一下，看看会发生什么结果？

1. gcc -o main main.o func.a test.a

    我们会得到如下报错.

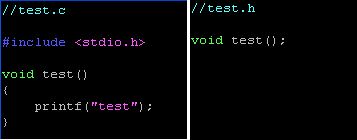
1. test.a(test.o): In function `test':
2. test.c:(.text+0x13): undefined reference to `func'
3. collect2: ld returned 1 exit status

  因此，我们需要注意，在链接命令中给出所依赖的库时，需要注意库之间的依赖顺序，依赖其他库的库一定要放到被依赖库的前面，这样才能真正避免undefined reference的错误，完成编译链接。

### ****5.在c++代码中链接****[C语言](http://lib.csdn.net/base/c)****的库****

 如果你的库文件由c代码生成的，则在c++代码中链接库中的函数时，也会碰到undefined reference的问题。下面举例说明。

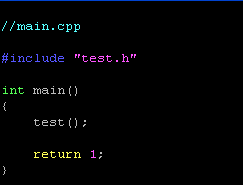
 首先，编写c语言版库文件：

[](http://img1.51cto.com/attachment/201109/150358466.png)

 编译，打包为静态库：test.a

1. gcc -c test.c
2. ar -rc test.a test.o

 至此，我们得到了test.a文件。下面我们开始编写c++文件main.cpp

[](http://img1.51cto.com/attachment/201109/150553287.png)

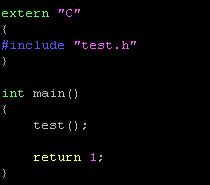
    然后编译main.cpp生成可执行程序：

1. g++ -o main main.cpp test.a

    会发现报错：

1. /tmp/ccJjiCoS.o: In function `main':
2. main.cpp:(.text+0x7): undefined reference to `test()'
3. collect2: ld returned 1 exit status

  原因就是main.cpp为c++代码，调用了c语言库的函数，因此链接的时候找不到，解决方法：即在main.cpp中，把与c语言库test.a相关的头文件包含添加一个extern "C"的声明即可。例如，修改后的main.cpp如下：

[](http://img1.51cto.com/attachment/201109/150959476.png)

1. g++ -o main main.cpp test.a

   再编译会发现，问题已经成功解决。

## pthread\_key\_setspecific和pthread\_key\_getspecific

在单线程程序中，我们经常要用到"全局变量"以实现多个函数间共享数据。在多线程环境下，由于数据空间是共享的，因此全局变量也为所有线程所共享。但有时应用程序设计中有必要提供线程私有的全局变量，仅在某个线程中有效，但却可以跨多个函数访问，比如程序可能需要每个线程维护一个链表，而使用相同的函数操作，最简单的办法就是使用同名而不同变量地址的线程相关数据结构。这样的数据结构可以由posix线程库维护，称为线程私有数据（Thread- specific Data，或TSD）。

下面说一下线程存储的具体用法。

1. 创建一个类型为pthread\_key\_t类型的变量。
2. 调用pthread\_key\_create()来创建该变量。该函数有两个参数，第一个参数就是上面声明的pthread\_key\_t变量，第二个参数是一个清理函数，用来在线程释放该线程存储的时候被调用。该函数指针可以设成 NULL，这样系统将调用默认的清理函数。该函数成功返回0.其他任何返回值都表示出现了错误。
3. 当线程中需要存储特殊值的时候，可以调用 pthread\_setspcific() 。该函数有两个参数，第一个为前面声明的pthread\_key\_t变量，第二个为void\*变量，这样你可以存储任何类型的值。
4. 如果需要取出所存储的值，调用pthread\_getspecific()。该函数的参数为前面提到的pthread\_key\_t变量，该函数返回void \*类型的值。下面是前面提到的函数的原型：

int pthread\_setspecific(pthread\_key\_t key, const void \*value);

void \*pthread\_getspecific(pthread\_key\_t key);

int pthread\_key\_create(pthread\_key\_t \*key, void (\*destructor)(void\*));

posix定义了两个API分别用来创建和注销TSD：

int pthread\_key\_create(pthread\_key\_t \*key, void (\*destr\_function) (void \*))

注销一个TSD采用如下API：

int pthread\_key\_delete(pthread\_key\_t key)

int pthread\_setspecific(pthread\_key\_t key, const void \*pointer)

void \* pthread\_getspecific(pthread\_key\_t key)

下面是一个如何使用线程存储的例子：

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

pthread\_key\_t key; // 好吧，这个玩意究竟是干什么的呢？

struct test\_struct { // 用于测试的结构

int i;

float k;

};

void \*child1(void \*arg)

{

struct test\_struct struct\_data; // 首先构建一个新的结构

struct\_data.i = 10;

struct\_data.k = 3.1415;

pthread\_setspecific(key, &struct\_data); // 设置对应的东西吗？

printf("child1--address of struct\_data is --> 0x%p\n", &(struct\_data));

printf("child1--from pthread\_getspecific(key) get the pointer and it points to --> 0x%p\n", (struct test\_struct \*)pthread\_getspecific(key));

printf("child1--from pthread\_getspecific(key) get the pointer and print it's content:\nstruct\_data.i:%d\nstruct\_data.k: %f\n",

((struct test\_struct \*)pthread\_getspecific(key))->i, ((struct test\_struct \*)pthread\_getspecific(key))->k);

printf("------------------------------------------------------\n");

}

void \*child2(void \*arg)

{

int temp = 20;

sleep(2);

printf("child2--temp's address is 0x%p\n", &temp);

pthread\_setspecific(key, &temp); // 好吧，原来这个函数这么简单

printf("child2--from pthread\_getspecific(key) get the pointer and it points to --> 0x%p\n", (int \*)pthread\_getspecific(key));

printf("child2--from pthread\_getspecific(key) get the pointer and print it's content --> temp:%d\n", \*((int \*)pthread\_getspecific(key)));

}

int main(void)

{

pthread\_t tid1, tid2;

pthread\_key\_create(&key, NULL); // 这里是构建一个pthread\_key\_t类型，确实是相当于一个key

pthread\_create(&tid1, NULL, child1, NULL);

pthread\_create(&tid2, NULL, child2, NULL);

pthread\_join(tid1, NULL);

pthread\_join(tid2, NULL);

pthread\_key\_delete(key);

return (0);

}

ld1--address of struct\_data is --> 0x0x7ffff77eff40 child1--from pthread\_getspecific(key) get the pointer and it points to --> 0x0x7ffff77eff40 child1--from pthread\_getspecific(key) get the pointer and print it's content: struct\_data.i:10 struct\_data.k: 3.141500 ------------------------------------------------------ child2--temp's address is 0x0x7ffff6feef44 child2--from pthread\_getspecific(key) get the pointer and it points to --> 0x0x7ffff6feef44 child2--from pthread\_getspecific(key) get the pointer and print it's content --> temp:20

#include <stdio.h>

#include <pthread.h>

pthread\_key\_t key;

void pp();

void\* start\_routine(void\* id)

{

int t\_id = (int)id;

pthread\_setspecific(key ,(void \*)t\_id);

sleep(2);

printf("start\_routine=%d\n",pthread\_getspecific(key));

return ;

}

int main()

{

pthread\_t id;

int i=4;

pthread\_key\_create(&key,NULL);

pthread\_create(&id , NULL, start\_routine ,i);

sleep(1);

pthread\_setspecific(key ,(void \*)i);

pp();

sleep(3);

pthread\_key\_delete(key);

return 0;

}

void pp()

{

int t=pthread\_getspecific(key);

printf("pp\_fun key=%d\n" ,t);

}

注意：pthread\_setspecfic和pthread\_getspecfic要成对出现，不然一个set的值，第一个get取的值是正确的，第二个就不正确的，

如果要2个get就需要2个set,才能取到对应的值。

## Linux系统调用--getrusage函数详解

**功能描述：**

获得进程的相关资源信息。如：用户开销时间，系统开销时间，接收的信号量等等;

用法：

#include <sys/types.h>

#include <sys/time.h>

#include <sys/resource.h>

#define RUSAGE\_SELF 0

#define RUSAGE\_CHILDREN -1

int getrusage(int who, struct rusage \*usage);

当调用成功后，返回0，否则-1；

**参数：**

who：可能选择有

RUSAGE\_SELF：获取当前进程的资源使用信息。以当前进程的相关信息来填充rusage(数据)结构

RUSAGE\_CHILDREN：获取子进程的资源使用信息。rusage结构中的数据都将是当前进程的子进程的信息

usage：指向存放资源使用信息的结构指针。

rusage(数据)结构定义在/usr/include/sys/resource.h中，下面是rusage的结构：

struct rusage {

struct timeval ru\_utime; // user time used

struct timeval ru\_stime; // system time used

long ru\_maxrss; // maximum resident set size

long ru\_ixrss; // integral shared memory size

long ru\_idrss; // integral unshared data size

long ru\_isrss; // integral unshared stack size

long ru\_minflt; // page reclaims

long ru\_majflt; // page faults

long ru\_nswap;// swaps

long ru\_inblock; // block input operations

long ru\_oublock; // block output operations

long ru\_msgsnd; // messages sent

long ru\_msgrcv; //messages received

long ru\_nsignals; // signals received

long ru\_nvcsw; // voluntary context switches

long ru\_nivcsw; // involuntary context switches

};

**举个例子：**

int who = RUSAGE\_SELF;

struct rusage usage;

int ret;

ret = getrusage(who, &usage);

这样你就可以用usage获取你想要的东西了；

同样wait3()和wait4()两个函数也可以得到进程资源信息.

得到CPU使用时间的用法：

struct rusage rup;

getrusage(RUSAGE\_SELF, &rup);

long sec = rup.ru\_utime.tv\_sec + rup.ru\_stime.tv\_sec;

long usec = rup.ru\_utime.tv\_usec + rup.ru\_stime.tv\_usec;

sec += usec/1000000;

usec %= 1000000;

## getpass()函数

getpass()函数用于从控制台输入一行字符串，关闭了回显（输入时不显示输入的字符串），适用于用密码的输入。

语法 char \* getpass (const char \* prompt);

参数prompt为提示字符串地址。

getpass()函数返回值：输入字符串地址。

[cpp] view plain copy

在CODE上查看代码片派生到我的代码片

#include

#include

int main(void)

{

char \*password;

password = getpass("Input a password:"); /\*输入密码\*/

printf("The password is: %s/r/n", password); /\*显示密码\*/

return 0;

}

getpass函数通常会与crypt加密函数一同使用，

[cpp] view plain copy

在CODE上查看代码片派生到我的代码片

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <crypt.h>

int main()

{

char passwd[13];

char \*key;

char slat[2];

key = getpass("Input first passward:");

slat[0] = key[0];

slat[1] = key[1];

strcpy(passwd,crypt(key,slat));

key = getpass("Input second passward:");

slat[0] = passwd[0];

slat[1] = passwd[1];

printf("After crypt(),1st passwd:%s\n",passwd);

printf("After crypt(),2st passwd:%s\n",passwd);

return 0;

}

gcc -lcrypt xxx.c

[root@iZ28o6dlzs9Z process]# ./a.out

Input first passward: //test

Input second passward: //test

After crypt(),1st passwd:teH0wLIpW0gyQ

After crypt(),2st passwd:teH0wLIpW0gyQ

getpass函数man翻译

函数说明：getpass()会显示参数prompt所指的字符串，然后从/dev/tty中读取所输入的密码，若无法从/dev/tty中读取则会转从标准输入设备中读取密码。所输入的密码长度限制在128个字符，包含结束字符NULL， 超过长度的字符及换行字符/n将会被忽略。在输入密码时getpass()会关闭字符回应，并忽略一些信号如CTRL-C 或 CTRL-Z所产生的信号

返回值 ：返回一个指向以NULL结尾的密码字符串

附加说明：为了系统安全考虑，在般在使用getpass()输入密码后，该密码最好尽快处理完毕，然后将该密码字符串清除

## socket AF\_UNIX通信

family参数代表地址家族，比较常用的为AF\_INET或AF\_UNIX。AF\_UNIX用于同一台机器上的进程间通信，AF\_INET对于IPV4协议的TCP和UDP 。

type参数代表套接字类型，SOCK\_STREAM（流套接字）或者SOCK\_DGRAM（数据报文套接字）。

其中AF\_INET类型有关的例子非常多，但AF\_UNIX的少见，所以给出一个AF\_UNIX简单的例子，并通过select实现非阻塞socket。

服务器端代码：

#include <sys/types.h>

#include <sys/socket.h>

#include <stdio.h>

#include <sys/un.h>

#include <unistd.h>

#include <stdlib.h>

int main (int argc, char \*argv[])

{

int server\_sockfd, client\_sockfd;

int server\_len, client\_len;

struct sockaddr\_un server\_address; /\*声明一个UNIX域套接字结构\*/

struct sockaddr\_un client\_address;

int i, bytes;

char ch\_send, ch\_recv;

unlink ("server\_socket"); /\*删除原有server\_socket对象\*/

/\*创建 socket, 通信协议为AF\_UNIX, SCK\_STREAM 数据方式\*/

server\_sockfd = socket (AF\_UNIX, SOCK\_STREAM, 0);

/\*配置服务器信息(通信协议)\*/

server\_address.sun\_family = AF\_UNIX;

/\*配置服务器信息(socket 对象)\*/

strcpy (server\_address.sun\_path, "server\_socket");

/\*配置服务器信息(服务器地址长度)\*/

server\_len = sizeof (server\_address);

/\*绑定 socket 对象\*/

bind (server\_sockfd, (struct sockaddr \*)&server\_address, server\_len);

/\*监听网络,队列数为5\*/

listen (server\_sockfd, 5);

printf ("Server is waiting for client connect...\n");

client\_len = sizeof (client\_address);

/\*接受客户端请求; 第2个参数用来存储客户端地址; 第3个参数用来存储客户端地址的大小\*/

/\*建立(返回)一个到客户端的文件描述符,用以对客户端的读写操作\*/

client\_sockfd = accept (server\_sockfd, (struct sockaddr \*)&server\_address, (socklen\_t \*)&client\_len);

if (client\_sockfd == -1) {

perror ("accept");

exit (EXIT\_FAILURE);

}

printf ("The server is waiting for client data...\n");

for (i = 0, ch\_send = '1'; i < 5; i++, ch\_send++) {

if ((bytes = read (client\_sockfd, &ch\_recv, 1)) == -1) {

perror ("read");

exit (EXIT\_FAILURE);

}

printf ("The character receiver from client is %c\n", ch\_recv);

sleep (1);

if ((bytes = write (client\_sockfd, &ch\_send, 1)) == -1) {

perror ("read");

exit (EXIT\_FAILURE);

}

}

close (client\_sockfd);

unlink ("server socket");

}

客户端代码：

#include <sys/types.h>

#include <sys/socket.h>

#include <stdio.h>

#include <sys/un.h>

#include <unistd.h>

#include <stdlib.h>

int main (int argc, char \*argv[])

{

struct sockaddr\_un address;

int sockfd;

int len;

int i, bytes;

int result;

char ch\_recv, ch\_send;

/\*创建socket,AF\_UNIX通信协议,SOCK\_STREAM数据方式\*/

if ((sockfd = socket(AF\_UNIX, SOCK\_STREAM, 0)) == -1) {

perror ("socket");

exit (EXIT\_FAILURE);

}

address.sun\_family = AF\_UNIX;

strcpy (address.sun\_path, "server\_socket");

len = sizeof (address);

/\*向服务器发送连接请求\*/

result = connect (sockfd, (struct sockaddr \*)&address, len);

if (result == -1) {

printf ("ensure the server is up\n");

perror ("connect");

exit (EXIT\_FAILURE);

}

for (i = 0, ch\_send = 'A'; i < 5; i++, ch\_send++) {

if ((bytes = write(sockfd, &ch\_send, 1)) == -1) { /\*发消息给服务器\*/

perror ("write");

exit (EXIT\_FAILURE);

}

sleep (2); /\*休息二秒钟再发一次\*/

if ((bytes = read (sockfd, &ch\_recv, 1)) == -1) { /\*接收消息\*/

perror ("read");

exit (EXIT\_FAILURE);

}

printf ("receive from server data is %c\n", ch\_recv);

}

close (sockfd);

return (0);

}

程序说明：

先运行服务器端，然后再运行客户端可以在两边同时看到输出。服务器端先运行后会出现如下提示：

./sock\_local\_server

Server is waiting for client connect...

这表示，服务器端已经被阻塞到到 accept() 这里，服务器就在此等候客户端的连接。

如果不是先运行服务器端，而直接运行客户端，那么客户端会提示：

./sock\_local\_client

ensure the server is up

connect: Connection refused

提示服务器没有准备好，连接被拒绝，从而直接退出程序。

如果服务器和客户端依次运行，可以在两边看到输出：

服务器端：

./sock\_local\_server

Server is waiting for client connect...

The server is waiting for client data...

The character receiver from client is A

The character receiver from client is B

The character receiver from client is C

The character receiver from client is D

The character receiver from client is E

客户端：

./sock\_local\_client

receive from server data is 1

receive from server data is 2

receive from server data is 3

receive from server data is 4

receive from server data is 5

## static变量在函数中只初始化一次

请写出下列程序的运行结果

int getNuber(int x)

{

static y = 1;

y = y + 1;

return (x+y);

}

int main()

{

Int a=1;

Int b=2;

printf(“getNuber(a)=[%d]\n”,getNuber(a));

printf(“getNuber(b)=[%d]\n”,getNuber(b));

return 0;

}

程序运行结果是 (3) (5)，而不是（3）（4），原因是static y=1；在函数中只初始化一次，如果把static y=1换成static y;y=1则结果才是（3）（4）

## extern和static关键字总结

### extern 关键字：

1、在一个文件中声明的外部变量

     作用域：如果在变量定义之前要使用该变量，则在用之前加

     extern，作用域扩展到从声明开始，到本文件结束。

2、在多个文件中声明的外部变量

在一个文件中想引用另外一个文件中已经

定义的外部变量时，则只需在引用变量的文件中用

extern关键字加以声明即可。可见，

其作用域从一个文件扩展到多个文件了。

例如：

文件a.c的内容：

#include <stdio.h>

int BASE=2;

//

变量定义

int exe(int x);

//

外部函数提前声明

int main(int argc, char \*agrv[])

{

    int a=10;

    printf("%d^%d = %d\n",BASE,a,exe(a));

    return 0;

}

文件b.c的内容：

#include <stdio.h>

extern BASE;

//

外部变量声明

int exe(int x)

{

    int i;

    int ret=1;

    for(i=0;i<x;i++)

    {

        ret\*=BASE;

    }

    return ret;

}

3、多个文件中声明的外部结构体变量

     在声明文件中必须包含有结构体类型的实现

      例如：

       typedef  struct \_student{

                    int    stuno;

                    char  name[50+1];

                    char  classNo[20+1];

       }T\_Student;

       extern    T\_Student   st;

声明是必须包含Student的实现部分，而且必须是 extern  T\_Student st，这样声明，不能extern st；这样声明

4、extern声明的外部函数可以被其他文件调用，函数默认的声明就是外部函数

### static关键字：

1、static关键字修饰的变量只能在本文件中引用

2、static变量程序执行期间一直存在，不会释放内存

3、static修饰的函数只能在本文件中引用，

      不能在其他文件中引用

### sizeof关键字

sizeof是关键字不是函数

int I = 100；

sizeof i 是正确的，

但不能写成sizeof int 必须sizeof(int)

## 不用中间变量交换变量

int a = 1 ,b = 2;

a = a + b;

b = a –b;

a = a –b;

## 指针与数组

“指针和数组可以交换！”

说出这句话并不是毫无根据的，因为在下面的两个举例中使用数组形式和指针形式都可以达到相同的结果。

举例1：

01

#include < stdio.h >

02

03

int main()

04

{

05

char \*p = "edsionte";

06

char str[] = "edsionte";

07

08

printf("p[1]=%c \*(p+1)=%c\n",p[1],\*(p+1));

09

printf("str[1]=%c \*(str+1)=%c\n",str[1],\*(str+1));

10

11

return 0;

12

}

13

14

/\* 编译并运行程序 \*/

15

edsionte@edsionte-laptop:~/code/expertC$ gcc tmp.c -o tmp

16

edsionte@edsionte-laptop:~/code/expertC$ ./tmp

17

p[1]=d \*(p+1)=d

18

str[1]=d \*(str+1)=d

在举例1中，指针p指向一个匿名的字符串“edsionte”，这个匿名字符串的占用的内存空间为9个字节；与p指向一个匿名字符串不同，数组str内存储着字符串“edsionte”，占用了9个字节的空间。

现在分别要访问’d’，则方法如下。对于指针p，分别可以通过指针形式\*(p+1)和数组形式p[1]来访问其所指的数据；对于数组str，分别可以通过指针形式\*(str+1)和数组形式str[1]来访问数组内的元素。

我们已经知道指针和数组在内存构造和访问方式上都不同，但为什么它们都分别可以通过指针的方式和数组的方式进行访问？

举例2：

#include < stdio.h >

void getStr\_pointer(char \*str)

{

printf("%s\n",str);

printf("getStr\_pointer(): sizeof(str)=%d\n",sizeof(str));

}

void getStr\_array(char str[100])

{

printf("%s\n",str);

printf("getStr\_array(): sizeof(str)=%d\n",sizeof(str));

}

int main()

{

char str[] = "I am edsionte!";

getStr\_pointer(str);

getStr\_array(str);

printf("main(): sizeof(str)=%d\n",sizeof(str));

}

/\* 编译并运行程序 \*/

edsionte@edsionte-laptop:~/code/expertC$ gcc tmp2.c -o tmp2

edsionte@edsionte-laptop:~/code/expertC$ ./tmp2

I am edsionte!

getStr\_pointer(): sizeof(str)=4

I am edsionte!

getStr\_array(): sizeof(str)=4

main(): sizeof(str)=15

在举例2中，getStr\_pointer函数和getStr\_array函数的功能都是显示一条字符串。但不同的是，前者传入的参数是一个指针，后者传入的参数是一个数组。在主函数中分别调用这两个函数，传入的参数都是数组str。

既然数组和指针不同，但为什么作为函数的形参，char str[ ]和char \*str相同？

上述举例所引出的这两个问题正是本文讨论的重点，它们分别对应着“指针和数组是相同”的两种情况。下面将分别进行讨论。

### 1.表达式中的数组名就是指针

表达式中的数组名其实就是数组首元素的首地址。对于编译器而言，a[i]其实就是\*(a+i)的形式，因此以数组形式访问数组元素总是可以写成“数组首元素首地址加上偏移量”的形式。取下标符号[ ]其实可以看成一种运算规则，即指向T类型的指针和一个整数相加，最终产生的结果类型为T。这里的指针就为数组首元素首地址，而整数即为数组的偏移量。

这里必须说明一下偏移量，它是指针每次移动的步长。对于数组而言，偏移量即数组元素的大小；对于指针而言，它的偏移量即为指针所指类型的大小。在对指针进行移动时，编译器负责计算每次指针移动的步长。

因此，str[i]和\*(str+i)两种形式其实是等价的。因为编译器总是将数组形式的访问自动转换成指针形式的访问。上面的分析都是针对数组而言，其实对指针以数组和指针形式访问的原理也是如此。只不过此时的访问是对指针所指向数据的访问。

结合数组和指针访问方式的不同，下面对举例1的代码做详细分析：

#### 1.1.以指针的形式和以数组的形式访问数组

从符号表中得到符号str的地址即为数组首元素的首地址。

以数组的形式：str[1]。从符号表中得到str符号的地址，即数组首元素的首地址；编译器将数组形式转化为\*(str+1)，在首元素首地址上加一个偏移量得到新地址；从这个新地址中读取数据，即为’d’；

以指针的形式：\*(str+1)。从符号表中得到str的地址，即数组首元素的首地址；在此地址上加一个偏移量得到新地址；从这个新地址中读取数据，即为’d’；

#### 1.2.以指针的形式和以数组的形式访问指针

不管以何种方式访问，我们应该清楚p始终是一个指针。从编译器符号表中得到符号p的地址为指针p的地址。

以指针的形式：\*(p+1)。首先从符号表中得到p的地址；从该地址中得到指针p；对指针p加上1个偏移量得到新地址；从这个新地址中读取数据，即为’d’；

以数组的形式：p[1]。首先从符号表中得到p的地址；从该地址中得到指针p；编译器将数组形式转化成\*(p+1)，对p加一个偏移量得到新地址；从这个新地址中读取新数据，即为’d’；

分析至此，你应该了解到以数组形式和以指针形式访问只是写法上的不同而已，其本质对内存的访问过程是一样的。

### 2.作为函数参数的数组名等同于指针

当作为函数形参时，编译器会将数组改成指向数组首元素的指针。此时的数组就等价于指针。之所以将传递给函数的数组形参转化为指针是处于效率的考虑。

在C语言中，所有非数组的实参数据都是以传值形式传递给函数的，即将实参的一份拷贝传递给调用函数中的形参，调用函数对这份拷贝（也就是形参）的修改不影响实参本身的值。如果按照这样的道理，传递数组时就必须拷贝整个数组空间，这样必然会产生很大的开销。并且，大部分时候并不会访问到数组中所有的元素而只是其中的几个。考虑到上述的原因，数组作为实参传递给调用函数时，只需将数组名传递给函数即可；而形参会被编译器该成指针的形式。因此，作为形参的数组既可以写成数组也可以写成指针。

现在再回到举例2中的代码，对于形参中的char str[]和char \*str也就感到不再奇怪了。事实上，即便将形参写成char str[]或char str[100]，编译器仍然会将它们改成char \*str的形式。

既然任何数组作为形参时候都等价于一个指针，那么在函数内对“数组”的一切操作都等价于对指针的操作。验证这一点的很好例证就是举例2中对数组str求长度。在主函数中，sizeof(str)的值为15，这个结果毫无争议，它就是数组str的长度。而在getStr\_pointer()和getStr\_array()中，sizeof(str)的值都为4，也就验证了作为形参的数组str在调用函数中就是一个指针！在上述情况1中，虽然表达式中数组名也被认为是指针，但是数组仍然是数组（main函数中sizeof的结果就是很好的验证），而此部分数组就是指针。这也是数组等价于指针的唯一情况。

换句话说，虽然在将数组作为形参的函数中，你可以继续以数组的形式使用这个参数，但实际上你跟不可能找到数组的踪影！

总结

关于指针和数组之间的异同需要反复的思考和总结，才能搞清关系。下面对指针和数组之间的可交换性再作义简单的总结。

1.在表达式中以a[i]这样的形式对数组进行访问时，编译器总将其解释为\*(a+i)的形式；

2.在数组作为函数的形参时，编译器将数组改写成指针，这个指针即为数组首元素的首地址。这也是数组等价指针的唯一情形；

3.由于2的原因，一个数组作为函数的形参时，既可以将数组定义成数组，也可以将数组定义成指针；

4.指针和数组永远是两码事，因此在不同文件中的声明和定义必须匹配，但却始终都能写成指针的形式和数组的形式（这完全是写法的不同）。

## [指针和数组的访问方式](http://edsionte.com/techblog/archives/2475)

前文从内存结构的角度说明了指针和数组的不同，本文将以访问方式的角度再次说明指针和数组的不同。先看下面的代码：

|  |  |  |
| --- | --- | --- |
| 1 | char str[] = "edsionte"; | |
| 2 | char \*p = "edsionte"; |

当编译完程序后，程序中的标示符都有一个地址，所有标示符的地址形成一个符号表。

### 数组的访问方式

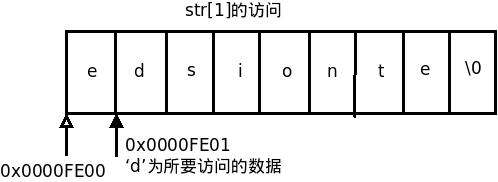
以数组str为例，如果要访问str[1]，即数组str的第二个元素，则它的访问步骤如下：

1.从编译器的符号表中得到str的地址，比如0x0000FE00。这个地址即为数组str首元素的首地址；

2.在这个地址上加一个偏移量得到新的地址0x0000FE01；

3.从这个新地址中读取数据；

通过下图可以加深对数组访问方式的理解：



### 指针的访问方式

以上述代码中的指针p为例，如果要访问\*(p+1)，即指针p所指向的匿名字符串的第二个字符，则它的访问步骤如下：

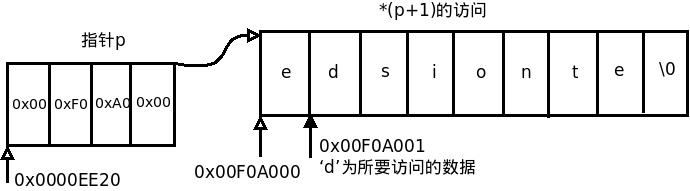
1.从编译器符号表中得到指针p的地址，比如0x0000EE20。这个地址即为&p，也就是指向指针p的指针；

2.从地址0x0000EE20中读取它内容即为指针p，比如0x00F0A000；

3.在0x00F0A000的基础上加一个偏移量，得到新地址0x00F0A001；

4.读取0x00F0A001中的内容，即为指针p所指的数据；

通过下图可以近一步理解指针的访问方式



通过分析得知，在符号表中得到的是指针的地址而不是我们所要访问的指针；而在符号表中可以直接得到数组首元素的首地址。因此，访问指针时必须先通过符号表中指针的地址得到所要访问的指针，再接着进行指针所指内容的访问；而数组则直接可以通过符号表中的地址进行元素访问。也就是说指针的访问比数组的访问多了一次对内存地址的读取。

### 一不小心就引发的错误

现在看下面的两段代码：

|  |  |  |
| --- | --- | --- |
| 01 | /\* 代码段1 \*/ | |
| 02 | file1: |

|  |  |  |
| --- | --- | --- |
| 03 | char str[] = "edsionte"; | |
| 04 | file2: |

|  |  |  |
| --- | --- | --- |
| 05 | extern char \*str; | |
| 06 | /\* 代码段2 \*/ |

|  |  |
| --- | --- |
| 07 | file1: |
| 08 | char \*p == "edsionte"; | |

|  |  |
| --- | --- |
| 09 | file2: |
| 10 | extern char str[]; | |

对于代码段1，在文件1中定义了数组str，而在文件2中将str声明为指针；对于代码段2，在文件1中定义了指针p，而在文件2中将p声明为数组。这里的声明指的是外部引用型声明，定义指的是定义型声明。

不管是上述那种情形，编译的时候都会出现错误。从上述对指针和数组访问方式的分析中可以得知，一个标示符被声明成指针还是数组对其访问方式影响巨大。下面我们对这两种错误作详细分析。

##### ****定义为数组，声明为指针****

在文件2中，既然str被声明成指针，那么就应当按照指针的方式进行访问。首先从符号表中得到指针str的地址；从该地址中读取4个字节的数据即为指针str；接下来根据指针str访问其所指向的数据。这个过程好像很顺利，不过对于文件1中的数组str，其访问过程又是怎样的？文件1和文件2的访问结果是否一致？下图可帮助你理解。

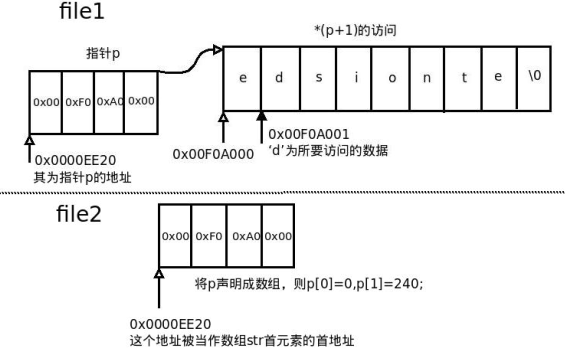


从上图可以看到，str在文件2中被声明成指针，那么就符号表中str的地址0x0000FE00会被当作指针str的地址。根据指针的访问方式，必须从这个地址中取出指针p。虽然以0x0000FE00为首的四个字节中存储的是“edsi”，但是它们一律会被当成地址，按十六进制表示即为0x65647379。即便可以访问到这个地址，但是从这个地址中按照char型取出的数据并不是我们想要的。

由于编译器对每个文件进行单独编译，文件2并不知道str在文件1中被定义成什么类型。str在文件1中被定义成数组，那么就应该按照数组的方式访问数据；str在文件2中被声明成指针，那么就应该按照数组的方式访问数据。因此，在两个不同的文件中分别将str定义成数组而声明成指针会出现对str访问不一致的现象，所以编译器会产生错误。

##### ****定义为指针，声明为数组****

此时，对于这种情况的理解也就简单多了。由于在文件而中p被声明成数组，因此就应该按照数组的方式对其进行访问。编译器会将原本指针str的地址当作str数组首元素的首地址，再对其加相应偏移量进行访问。这显然也是不合理的，因此编译器产生错误。具体可参见下图：



对上述的错误进行分析后，我们应该清楚将一个标示符声明成数组，编译器就会按照数组的访问方式去访问它；指针也是如此。因此，应该在多个文件中保持声明和定义相匹配。

### 指针和数组的其他区别

指针和数组除了在内存构造和访问方式上不同外，还有一些其他的区别。

1.指针通常用于指向一个动态的数据结构，而数组则用于存储固定大小和数据类型相同的数据；

2.指针所指向的数据通过malloc()分配，并且需要free()释放；而数组本身的内存空间则是隐士分配和释放，也就是在定义数组的时候进行；

3.指针所指向的数据通常是匿名的，而数组名则是数组所占内存空间的名字；

在本系列的最后一篇文章中，我们将分析指针和数组易被混淆的根源——也可将其称为指针和数组的可交换性。

### 数组的声明与定义

|  |  |
| --- | --- |
| 1 | char str[10]; |
| 2 | extern char str[]; | |

第一条声明语句定义了一个char型的数组，并为其分配了10字节大小的内存空间。而第二条声明语句则是为了告诉编译器这个数组的类型以及大小。由于在外部引用型声明中并不会数组分配内存空间，因此这种声明并不需要指定数组的大小。对于多维数组也并不需要指定第一维的大小。

### 指针的内存布局

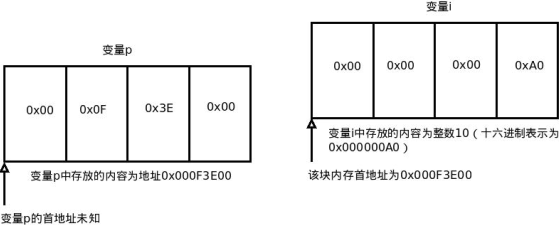
指针本质上就是一个内存地址，为了方便使用这个内存地址将它和一个标示符绑定在一起。比如：

|  |  |
| --- | --- |
| 1 | int i = 10; |
| 2 | int \*p = &i; /\* 假设变量i的内存地址为0x000F3E00 \*/ | |

上述语句将地址0x000F3E00和p绑定在一起，并且一旦绑定就不能再修改，p此时也被称为指针变量，简称指针。“指针变量“中的“变量”并不是说明p可以再和其他地址绑定，而是强调与p绑定的这个地址中的内容可变，即i的值可以变化。

既然指针p是一个内存地址，那么在32位的系统中指针p所占的内存大小就始终为4字节。虽然整型变量i也占4字节，但是这两个同大小的内存空间却有着本质区别。指针p只能存放内存地址，并且这个地址只能是整型数据的首地址。即使在p内存放了其他数据，也会一律被当作内存地址来处理。

通过下图可以近一步了解指针和其所指数据的关系：



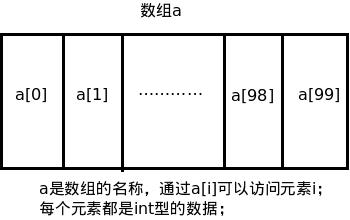
从图中可得知，不管指针所指数据占多大的内存空间，指针本身只占用4字节的大小。由于指针p本身占用4字节的内存空间，因此这部分内存空间也必然会有首地址。通过&p操作就可以得到指针p的首地址，也就是存储指针p的内存空间的首地址。从上图中可以看到指针p中即为整型变量i的首地址，因此我们也称p是一个指向整型变量i的指针。

### 数组的内存布局

数组是一块连续的内存空间，这块内存空间的名称即为数组名。比如：

|  |  |
| --- | --- |
|  | int a[100]; |

当定义了一个具体的数组a时，编译器就根据数据类型和大小为其分配100\*sizeof(int)大小的内存空间，并将这块连续的内存空间命名为a。虽然我们可以通过a[i]这种方式来访问元素i，但这并不代表a[i]就是这个元素的名称。因此每个数组元素实际上是没有名字的，编译器只为这块内存提供了唯一的名称a。同时数组名a也代表数组首元素的首地址。数组的内存结构如下：



通过对指针和数组内存布局的分析，我们可以得知这两者完全是不相同的。指针不管指向什么数据，它本身的大小就是4个字节（32位系统）；而数组则是一块连续的内存空间。在下文中，将会从访问方式的角度分析指针和数组的不同。

## 数组名与数组的地址

Int a[100];

sizeof (a)是400，

sizeof(&a),sizeof(&a[0])和sizeof(&a[100])是相等的都是4，在32位寻址机器中,地址都是占用4个字节

数组名是常量，是数组元素的首地址即a和&a[0]是一样的，所以\*(a+1)和a[1]是相同的

&a是数组的地址

&a+1和a+1是不一样的，&a+1=[a的地址 + sizeof(a)],

a+1=[a的地址 + sizeof(int)]

需要注意的是不同的编译器，sizeof(&a)的结果可能不一样

在visual C++中sizeof(&a)的大小竟然是400,这个是错误的结果，

在gcc和cc编译器中sizeof(&a)的大小都是4

经典例子：

Int a[5] = {1,2,3,4,5};

Int \*p = (int \*)(&a + 1)

printf(“\*(a+1) = [%d], \*(p-1) = [%d]”,\*(a + 1)，\*(p-1) );

打印结果如下：

\*(a+1) = [2], \*(p-1) = [5]

## 数组长度

int a[MAX];

由于MAX是宏定义,他的值是可以变的,所以数组的长度不好确认

可以采用以下方法确定数组的长度

sizeof(a) / sizeof(a[0]);

## 常量指针和指向常量的指针

char a1=’c’

char a2=’b’

常量指针：

char const\* p1=&a1或const char \* p1=&a1 正确

p1=&a1 编译会报错，常量必须在定义时就赋值

指向常量的指针：

char \* const p2

p2=&a1;

p2=&a2;

指向常量的指针p2，指向地址可以改变，但是不能通过\*p2=‘’操作去改变地址里面的内容。

如 \*p2=‘D’编译会报错

## 指向数组的指针

int \*p[n];-----指针数组，每个元素均为指向整型数据的指针。

int (\*p)[n];------p为指向一维数组的指针，这个一维数组有n个整型数据。

int \*p();----------函数返回指针，指针指向返回的值。

int (\*p)();------p为指向函数的指针。

## 指向函数的指针

int (\*p)();------p为指向函数的指针。

## 数组用在函数参数中是和指针一样的

在编译时，编译器始终会把函数的数组参数，作为指向第一个元素的指针，这样不需要对数组和指针这两种情况做区分

所以一定不能用sizeof(形参数组)来获取数组的长度,

## 函数指针数组

Int (\*p[10])();

定义了一个具有10函数指针元素的数组

## void \* 指针

如果函数的参数可以是任意类型指针，那么应该申明其参数为void \*

典型的内存操作函数memcpy和memset的函数原型分别为：

void \* memcpy(void \* dest, const void \* src,size\_t len);

void \* memset(void \* buffer,int c, size\_t num);

## 二级指针用于函数传参数

例子：

void GetMemory(char \*p,int num)

{

p = (char \*) malloc(sizeof(char) \* num);

}

Int main()

{

char \* str = NULL;

GetMemory(str,10);

Strcpy(str,”hello”);

free(str);//free幷没有起作用，内存泄露

return 0;

}

在运行strcpy(str,”hello”)语句的时候发生错误。这时候观察str的值，发现任然为NULL，也就是说str本身幷没有改变，我们malloc的内存的地址并没有赋值给str，而是赋值给p而这个p编译器自动分配和回收的。我们根本无法使用，解决这一问题有两个办法：

第一：用return

char \* GetMemory(char \*p,int num)

{

p = (char \*) malloc(sizeof(char) \* num);

return p;//return不能返回栈指针，但可以返回堆指针

}

第二：二级指针

void GetMemory(char \*\* p,int num)

{

\*p = (char \*) malloc(sizeof(char) \* num);

}

## 二维数组参数与二维指针参数

1、void fun(char a[3][4])

可以把a[3][4]理解成为a[3],其每个元素都是一个包含4个char类型数据的数组。C语言中，当一维数组作为函数参数的时候，编译器总是把它解成一个指向其首元素地址的指针，因此可以把这个函数的参数声明改写成：char fun(char (\*p)[4])

同样作为参数时，一维组“[]”号内的数字完全可以省略即：char fun (char [][4]),但第二维数不能省略。

2、void fun(char \* p[4])可以改写成void fun(char \*\*p)

对于p来说，它是一个包含4个指针的一维数组，同样把这个一维数组也可以改写成指针的形式，那就得到上面的写法

需要注意的是，C语言中，当一维数组作为函数参数的时候，编译器总是把它解析成指向其首元素地址的指针。这个规则并不是递归的，也就是说只有一维数组才是如此，当 数组超过一维时，将第一维改写成指向数组元素首地址的指针后，后面的维再也不可以改写。比如：a[3][4][5]作为参数时可以被改写成为（\*p）[4][5]

## 野指针

调用free(p)或delete（p）函数后指针p只是解除了和堆内存的联系，但是p的值幷不是NULL所以，调用完这两个函数后一定要对p赋值为NULL即p = NULL,防止p成为野指针，因为当p的值不为NULL的时候对于，if (NULL ! = p)这样的逻辑判断就会产生致命的逻辑判断错

对于调用malloc和new函数分配内存时，也可能由于堆内存不足等于原因导致分配内存失败，所以对于p = NULL,p = （char \*）malloc(sizeof(char))或p = new char[10]表达式，一定要异常判断处理if ( NULL == NULL)或者if（NULL != p）时才表示分配内存成功。

## strlen与sizeof的区别

1. char\* str=”hello world”; sizeof(str)=12,strlen(str)=11.因为strlen函数求字符串长度时不包含字符串末尾的’\0’。
2. char a[10];int len=strlen(a); len其实不等于10，应该注意strlen求的是从字符串第一个元素到‘\0’之间字符总数（不包含‘\0’），而此数组未初始化。尽管语法上是正确的，但是用法错误。如果换作sizeof(a)，结果便是10。

## bug – sizeof关键字

#define ARRAY\_LEN(AR) sizeof(AR) / sizeof(AR[0])

int array = {1,2,3,4,5,6,7,8,9}

int main(void)

{

int d = -1;

if (d <= ARRAY\_LEN(array))

{

printf(“hello world\n”);

}

return 0;

}

程序执行并不会打印出hello world ，因为此处有个bug，

sizeof关键在返回的是unsigned int 类型的数字if语句在signed int 和unsigned int之间测试相等性，所以导致d被升级为unisigned int 类型，-1转换成unsigned int 的结果将是一个非常巨大的正整数导致表达式结果为假

修复这个bug，应该在ARRAY\_LEN(array)前加上强制类型转换即（int）ARRAY\_LEN(array)

总结：在ANSI C中：当执行算术运算时，操作数的类型如果不同，就会发生转换。数据类型一般朝着浮点精度更高，长度更长的方向呢转换，整型如果转换为signed不会丢失信息，就转换成signed否则就转换成unsigned，所以把几个整型操作数像上面一样混合使用时结果类型有可能是有符号数，也有可能是无符号数，取决于操作数的类型的相对大小

## 字符串常量与数组变量

如下语句：

char\* dest=”hello,world”;

char\* src=”hello”;

puts(strcpy(dest,src));

这样做是错误的，原始是因为dest指向字符串常量，不可修改其某个字符值。其实正确的写法应该是const char\* dest=”hello,world”;但是一般情况下不加const编译的时候也不会出错。但是如果dest[2]=’a’;那么就会出错，即便这条语句合乎语法。如果 char a[100]; char\* dest=a;因为数组中的元素是变量，可以修改，也就可成功打印。如果有下面语句:char str[100]=”hello,world”，这样写完全正确，因为只是用hello world初始化这个数组而已。

## scanf与gets的区别

想要输入下面一段字符串：edsionte is a goog boy。用scanf函数是不行的。因为scanf函数以回车键和空格键作为字符串输入结束的标志。而gets函数则仅以回车键作为字符串输入的结束符。因此可以使用gets函数来接收含有空格键的字符串

## 宏定义和预编译

gcc -E hello.c -o hello.i

宏定义和预编译不属于c语言语句，因此不已；结尾。而结构体定义属于c语言语句，应该以分号结尾

## y = x/\*p

y = x/\*p,表示y等于x除以指针p指向的内容，但是该语句是编译不过的因为

只要/和\*之间没有空格，编译器就认为是注释的开始/\*，写成y = x / \*p 就不会报错了，所以运算符和操作数之间最好要加空格，养成良好的编程习惯很重要。

## pow() –幂函数

pow() 函数用来求 x 的 y 次幂（次方），其原型为：

double pow(double x, double y);

pow()用来计算以x 为底的 y 次方值，然后将结果返回。设返回值为 ret，则 ret = xy。

可能导致错误的情况：

如果底数 x 为负数并且指数 y 不是整数，将会导致 domain error 错误。

如果底数 x 和指数 y 都是 0，可能会导致 domain error 错误，也可能没有；这跟库的实现有关。

如果底数 x 是 0，指数 y 是负数，可能会导致 domain error 或 pole error 错误，也可能没有；这跟库的实现有关。

如果返回值 ret 太大或者太小，将会导致 range error 错误。

错误代码：

如果发生 domain error 错误，那么全局变量 errno 将被设置为 EDOM；

如果发生 pole error 或 range error 错误，那么全局变量 errno 将被设置为 ERANGE。

注意，使用 GCC 编译时请加入-lm。

【实例】请看下面的代码。

#include <stdio.h>

#include <math.h>

int main ()

{

printf ("7 ^ 3 = %f\n", pow (7.0, 3.0) );

printf ("4.73 ^ 12 = %f\n", pow (4.73, 12.0) );

printf ("32.01 ^ 1.54 = %f\n", pow (32.01, 1.54) );

return 0;

}

输出结果：

7 ^ 3 = 343.000000

4.73 ^ 12 = 125410439.217423

32.01 ^ 1.54 = 208.036691

## ceil函数

double ceil(double x) 返回不小于x的最小整数值

## floor函数

double round(double x)返回不大于x的最大整数值

## round函数

double round(double x)返回四舍五入的整数值

## 函数不定参数va\_list

C中传递函数的参数时的用法和原理：

1.在C中，当我们无法列出传递函数的所有实参的类型和数目时,可以用省略号指定参数表

void foo(...);

void foo(parm\_list,...);

这种方式和我们以前认识的不大一样，但我们要记住这是C中一种传参的形式，在后面我们就会用到它。

2.函数参数的传递原理

　　函数参数是以数据结构:栈的形式存取,从右至左入栈。

　　首先是参数的内存存放格式：参数存放在内存的堆栈段中，在执行函数的时候，从最后一个开始入栈。因此栈底高地址，栈顶低地址，举个例子如下：

void func(int x, float y, char z);

　　那么，调用函数的时候，实参 char z 先进栈，然后是 float y，最后是 int x，因此在内存中变量的存放次序是 x->y->z，因此，从理论上说，我们只要探测到任意一个变量的地址，并且知道其他变量的类型，通过指针移位运算，则总可以顺藤摸瓜找到其他的输入变量。

　　下面是 <stdarg.h> 里面重要的几个宏定义如下：

typedef char\* va\_list;

void va\_start( va\_list ap, prev\_param ); /\* ANSI version \*/

type va\_arg ( va\_list ap, type );

void va\_end ( va\_list ap );

va\_list 是一个字符指针，可以理解为指向当前参数的一个指针，取参必须通过这个指针进行。

<Step 1> 在调用参数表之前，定义一个 va\_list 类型的变量，(假设va\_list 类型变量被定义为ap)；

<Step 2> 然后应该对ap 进行初始化，让它指向可变参数表里面的第一个参数，这是通过 va\_start 来实现的，第一个参数是 ap 本身，第二个参数是在变参表前面紧挨着的一个变量,即“...”之前的那个参数；

<Step 3> 然后是获取参数，调用va\_arg，它的第一个参数是ap，第二个参数是要获取的参数的指定类型，然后返回这个指定类型的值，并且把 ap 的位置指向变参表的下一个变量位置；

<Step 4> 获取所有的参数之后，我们有必要将这个 ap 指针关掉，以免发生危险，方法是调用 va\_end，他是输入的参数 ap 置为 NULL，应该养成获取完参数表之后关闭指针的习惯。说白了，就是让我们的程序具有健壮性。通常va\_start和va\_end是成对出现。

例如 int max(int n, ...); 其函数内部应该如此实现：

#include <iostream.h>

void fun(int a, ...)

{

　　int \*temp = &a;

　　temp++;

　　for (int i = 0; i < a; ++i)

　　{

　　　　cout << \*temp << endl;

　　　　temp++;

　　}

}

int main()

{

　　int a = 1;

　　int b = 2;

　　int c = 3;

　　int d = 4;

　　fun(4, a, b, c, d);

　　system("pause");

　　return 0;

}

Output::

1

2

3

4

3:获取省略号指定的参数

　　在函数体中声明一个va\_list，然后用va\_start函数来获取参数列表中的参数，使用完毕后调用va\_end()结束。像这段代码：

void TestFun(char\* pszDest, int DestLen, const char\* pszFormat, ...)

{

va\_list args;

va\_start(args, pszFormat); //一定要“...”之前的那个参数

\_vsnprintf(pszDest, DestLen, pszFormat, args);

va\_end(args);

}

4.演示如何使用参数个数可变的函数，采用ANSI标准形式

#include 〈stdio.h〉

#include 〈string.h〉

#include 〈stdarg.h〉

/\*函数原型声明，至少需要一个确定的参数，注意括号内的省略号\*/

int demo( char, ... );

void main( void )

{

   demo("DEMO", "This", "is", "a", "demo!", "");

}

/\*ANSI标准形式的声明方式，括号内的省略号表示可选参数\*/

int demo( char msg, ... )

{

       /\*定义保存函数参数的结构\*/

   va\_list argp;

   int argno = 0;

   char para;

　　   /\*argp指向传入的第一个可选参数，msg是最后一个确定的参数\*/

   va\_start( argp, msg );

   while (1)

       {

        para = va\_arg( argp, char);

           if ( strcmp( para, "") == 0 )

               break;

           printf("Parameter #%d is: %s\n", argno, para);

           argno++;

}

va\_end( argp );

/\*将argp置为NULL\*/

return 0;

}

 以上是对va\_start和va\_end的介绍。

## 管道

管道是进程间通信的主要手段之一。一个管道实际上就是个只存在于内存中的文件，对这个文件的操作要通过两个已经打开文件进行，它们分别代表管道的两端。管道是一种特殊的文件，它不属于某一种文件系统，而是一种独立的文件系统，有其自己的数据结构。根据管道的适用范围将其分为：无名管道和命名管道。

### 无名管道

主要用于父进程与子进程之间，或者两个兄弟进程之间。在linux系统中可以通过系统调用建立起一个单向的通信管道，且这种关系只能由父进程来建立。因此，每个管道都是单向的，当需要双向通信时就需要建立起两个管道。管道两端的进程均将该管道看做一个文件，一个进程负责往管道中写内容，而另一个从管道中读取。这种传输遵循“先入先出”（FIFO）的规则。

系统调用pipe是用来建立管道的。该调用的声明格式如下：

　　　　　int pipe(int filedes[2]);

一个管道拥有两个文件描述符用来通信，它们指向一个管道的索引节点，该调用将这两文件描述符放在参数filedes中返回。现在的许多系统中管道允许数据双向流动，但一般习惯上，文件描述符filedes[0]用来读数据，filedes[1]用来写数据。如果要求程序的可移植性好，就按照习惯的用法来编程。调用成功时，返回值为0；错误时，返回-1，并设置错误代码errno：

　　EMFILE：进程使用了过多的文件描述符。

　　ENFILE：系统文件表满。

　　EFAULT：参数filedes无效。

对于写管道：

写入管道的数据按到达次序排列。如果管道满，则对管道的写被阻塞，直到管道的数据被读操作读取。对于写操作，如果一次write调用写的数据量小于管道容量，则写必须一次完成，即如果管道所剩余的容量不够，write被阻塞直到管道的剩余容量可以一次写完为止。如果write调用写的数据量大于管道容量，则写操作分多次完成。如果用fcntl设置管道写端口为非阻塞方式，则管道满不会阻塞写，而只是对写返回0。

Example:

#include<unistd.h>

void main()

{

int pid;

int fileds[2];

char buffer[80;

pipe(fileds);

pid = fork();

if (pid >0)

{

char s[]=”Hello\n”

write(fileds[1],s,sizeof(s));

}

else if(pid == 0)

{

read(fileds[0],buffer,80);

printf(“buffer =[%s]”);

}

else

{

printf(“call fork() failed\n”);

return ;

}

}

### 有名管道

命名管道是为了解决无名管道只能用于近亲进程之间通信的缺陷而设计的。命名管道是建立在实际的磁盘介质或文件系统（而不是只存在于内存中）上有自己名字的文件，任何进程可以在任何时间通过文件名或路径名与该文件建立联系。为了实现命名管道，引入了一种新的文件类型——FIFO文件（遵循先进先出的原则）。实现一个命名管道实际上就是实现一个FIFO文件。命名管道一旦建立，之后它的读、写以及关闭操作都与普通管道完全相同。虽然FIFO文件的inode节点在磁盘上，但是仅是一个节点而已，文件的数据还是存在于内存缓冲页面中，和普通管道相同。

有名管道又称为FIFO，是进程间通信的一种方式。FIFO具有以下特点：

1.全双工的通信模式，数据先进先出；

2.可以用于任意的进程之间，通过指定相同的管道文件进行通信；

3.文件名存在文件系统中，而管道中的内容存在于内存中。可通过open、read、write对其操作；

使用FIFO的步骤如下：

#### 一、创建/打开一个FIFO

FIFO是一种文件类型，在Linux系统中FIFO的类型用p表示。如下所示：

-rwxr-xr-x 1 root root 7368 2008-10-29 09:05 create\_fifo

-rw-r--r-- 1 root root 380 2008-10-29 09:05 create\_fifo.c

prw-r--r-- 1 root root 0 2009-06-12 14:18 myfifo

-rwxr-xr-x 1 root root 8178 2008-10-29 08:58 read\_fifo

-rw-r--r-- 1 root root 1185 2008-10-29 09:00 read\_fifo.c

-rwxr-xr-x 1 root root 8333 2009-06-12 14:20 write\_fifo

-rw-r--r-- 1 root root 1139 2009-06-12 14:19 write\_fifo.c

可以看到，虽然FIFO文件存在于文件系统中(可供不同的进程打开)，但FIFO中的内容都存放在内存中，所以文件大小始终为0。

由于FIFO不是普通文件，所以只能用文件ＩＯ来访问。

#include <sys/stat.h>

int mkfifo(const char \*path, mode\_t mode);

函数mkfifo用于创建一个有名管道，参数path指定要创建的FIFO的路径，mode为该管道文件的访问权限，一般用八进制数表示。

#include <sys/stat.h>

#include <fcntl.h>

int open(const char \*path, int oflag, ... );

函数open通过指定路径打开一个文件，不同的进程可以调用open打开同一个FIFO进行通信。参考下面的代码(相关头文件省略)

#define BUF\_SIZE 51

int main(int argc, char \*argv[])

{

int fd;

ssize\_t n;

char buf[BUF\_SIZE];

if ( argc <２)

{

fprintf(stdout, “Usage: %s <fifo\_path>\n”, argv[0]);

exit(1);

}

if ( mkfifo(argv[1], 0666) < 0 ) // 创建FIFO失败

{

if (errno != EEXIST ) // 出错原因不是因为管道已存在

{

fprintf(stderr, “mkfifo() failed %s\n”, strerror(errno));

exit(-1);

}

}

if ( (fd = open(argv[1], O\_RDWR)) < 0 ) // 打开FIFO出错

{ // 注：< 优先级要高于 =

fprintf(stderr, “open() failed %s\n”, strerror(errno));

exit(-1);

}

…

return 0;

}

#### 二、读/写FIFO

进程打开FIFO后，就可以根据open时指定的选项对其进行相应的读/写操作(请参考open的帮助文档中关于选项的说明)。

#include <unistd.h>

ssize\_t read(int fildes, void \*buf, size\_t nbyte);

ssize\_t write(int fildes, const void \*buf, size\_t nbyte);

……

if ((n = read(fd, buf, BUF\_SIZE)) < 0 )

{

fprintf(stderr, “read() failed %s\n”, strerror(errno));

exit(-1);

}

else if ( n = = 0 )

{

fprintf(stdout, “all write sides are closed…\n”);

exit(-1);

}

else

{

fprintf(stdout, “read %d bytes from FIFO : %s\n”, n, buf);

}

……

对FIFO的写操作，大家可以仿照上面的代码。

最后总结一下在使用FIFO时要注意的问题：

1. 在用open打开FIFO时有可能会阻塞，原因就是当前只有读端或写端存在。换句话说，如果程序在打开FIFO时指定了只读方式/只写方式，那么该进程对于打开的FIFO来说就是一个读端/写端。如果指定的是读写方式，那么进程既是读端又是写端。

2. 从FIFO中读数据时(用read函数)，如果没有数据，默认是阻塞等待，直到有数据被写入FIFO。如果read函数返回0，说明该FIFO所有的写端都已关闭，程序要做相应的处理。 向FIFO写入数据时(使用write函数)，如果FIFO有足够空间，write函数会返回写入的字节数；如果空间不够，write函数会阻塞，直到写完为止。当所

有的读端都关闭时，再向FIFO写数据会出错。内核会向写进程发管道断裂的信号(SIGPIPE), 从而终止该进程。处理的办法有两种：程序以读写方式打开,

FIFO或是在程序中捕捉SIGPIPE信号，由用户自行处理。

## java中的命名规则

1、类名命令：

A、如果类名使用拉丁字母，那么类名的首字母使用大写字母，如Hello，Time

B、如果类名由几个单词组成，每个单词的首字母使用大写字母，如BeijingTime，AmericaTime

2、变量命令：

A、如果变量名使用拉丁字母，那么变量名的首字母使用小写字母，如hello，time

B、如果变量名由几个单词组成，从第二个单词开始的首字母使用大写字母，其他单词的首字母使用小写，如beijingTime，americaTime

3、方法命令：

A、如果变量名使用拉丁字母，那么变量名的首字母使用小写字母，如hello，time

B、如果变量名由几个单词组成，从第二个单词开始的首字母使用大写字母，其他单词的首字母使用小写，如beijingTime，americaTime

## **逗号表达式**

1、 下面这个程序的输出结果是什么？  
int main()  
{ int a = （1, 2）;  
printf(“a = %d\n”, a);  
return 0;  
}

**解答：**  
**a = 2**  
**这里利用了逗号表达式。**

## 多维数组

我们可以将多维数组抽象的看作是具有某种类型的一维数组。当“某种类型”为基本的数据类型时，多维数组就退化成普通的一维数组；当“某种类型”仍然为数组时，那么就形成了多维数组。也就是说任何一个多维数组都可以分解成几个一维数组。

下面通过示例程序来深入了解多维数组ma[2][3]的构成。

|  |  |  |
| --- | --- | --- |
| 01 | #include < stdio.h > | |
| 02 |  |

|  |  |  |
| --- | --- | --- |
| 03 | int main() | |
| 04 | { |

|  |  |
| --- | --- |
| 05 | int ma[2][3]; |
| 06 | int (\*r)[2][3]; | |

|  |  |  |
| --- | --- | --- |
| 07 | int (\*p)[3]; | |
| 08 | int \*t; |

|  |  |
| --- | --- |
| 09 |  |
| 10 | /\*代码段1\*/ | |

|  |  |
| --- | --- |
| 11 | p = ma; |
| 12 | printf("sizeof(ma[0])=%d\n",sizeof(ma[0])); | |

|  |  |  |
| --- | --- | --- |
| 13 | printf("ma      =%p\tp   =%p\n",ma,p); | |
| 14 | printf("p+1 =%p\n",p+1); |

|  |  |  |
| --- | --- | --- |
| 15 | /\*代码段2\*/ | |
| 16 | r = &ma; |

|  |  |
| --- | --- |
| 17 | printf("sizeof(ma)=%d\n",sizeof(ma)); |
| 18 | printf("&ma     =%p\tr  =%p\n",&ma,r); | |

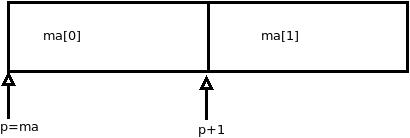
|  |  |  |
| --- | --- | --- |
| 19 | printf("&ma+1   =%p\tr+1=%p\n",&ma+1,r+1); | |
| 20 | /\*代码段3\*/ |

|  |  |
| --- | --- |
| 21 | t = ma[0]; |
| 22 | printf("sizeof(ma[0][0])=%d\n",sizeof(ma[0][0])); | |

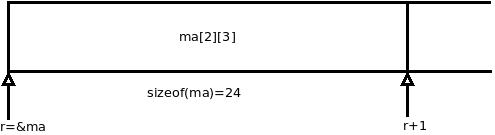
|  |  |
| --- | --- |
| 23 | printf("ma[0]   =%p\tt   =%p\n",ma[0],t); |
| 24 | printf("ma[0]+1 =%p\tt+1 =%p\n",ma[0]+1,t+1); |

|  |  |  |
| --- | --- | --- |
| 25 | return 0; | |
| 26 | } |

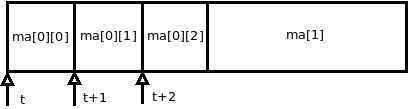
由多维数组ma最左维的长度2可知，ma数组包含两个元素ma[0]和ma[1]。数组名ma在表达式中是数组ma首元素的首地址。在代码段1中将ma赋值给数组指针p，则p指向多维数组ma的首元素ma[0]，则p+1指向第二个元素ma[1]。其中p是一个数组指针，它指向一个长度为3的数组，则指针p每次移动的偏移量为12。可参考下图：



在代码2中对ma取地址并将其赋值给指针r。r现在指向一个“第一维的大小为2，第二维的大小为3的数组”，则r+1将指向下一个这样的数组（尽管这样的数组并不存在）。由此也可得知r每次的偏移量为24



ma[0]和ma[1]都是一个长度为3的整型数组，现在以ma[0]为例进行说明。ma[0]中包含三个元素ma[0][0]，ma[0][1]和ma[0][2]。在代码段3中将ma[0]赋值给t，则t指向数组ma[0]的第一个元素a[0][0]，则t+1和t+2则依次指向第二个元素和第三个元素。



对多维数组ma的结构有了一定了解后，现在再看上述程序的运行结果：

|  |  |  |
| --- | --- | --- |
| 01 | edsionte@edsionte-laptop:~/code/expertC$ gcc array.c -o array | |
| 02 | edsionte@edsionte-laptop:~/code/expertC$ ./array |

|  |  |
| --- | --- |
| 03 | sizeof(ma[0])=12 |
| 04 | ma   =0xbfdfaa6c    p=0xbfdfaa6c | |

|  |  |  |
| --- | --- | --- |
| 05 | p+1  =0xbfdfaa78 | |
| 06 | sizeof(ma)=24 |

|  |  |  |
| --- | --- | --- |
| 07 | &ma  =0xbfdfaa6c    r=0xbfdfaa6c | |
| 08 | r+1  =0xbfdfaa84 |

|  |  |
| --- | --- |
| 09 | sizeof(ma[0][0])=4 |
| 10 | ma[0]=0xbfdfaa6c    t=0xbfdfaa6c | |

|  |  |
| --- | --- |
| 11 | t+1  =0xbfdfaa70 |

注意在结果中，p，r和t的值均相同，但是所指向的数据却不同。更具体的说，这三个指针每次移动时的偏移量不同。

## 多维数组的初始化

数组的初始化只能在对数组进行声明（具体为定义型声明）时进行。一维数组的初始化很简单，只要将所有初始值放在一个大括号中即可。如果声明数组时未指定数组的长度，则编译器会根据初始值的个数来确定数组的长度。

|  |  |  |
| --- | --- | --- |
| 01 | #include < stdio.h > | |
| 02 |  |

|  |  |  |
| --- | --- | --- |
| 03 | int main() | |
| 04 | { |

|  |  |
| --- | --- |
| 05 | int m[] = {1,2,3}; |
| 06 | int n[] = {1,2,3,}; | |

|  |  |
| --- | --- |
| 07 |  |
| 08 | printf("length(m)=%d\n",sizeof(m)/sizeof(m[0])); | |

|  |  |  |
| --- | --- | --- |
| 09 | printf("length(n)=%d\n",sizeof(n)/sizeof(n[0])); | |
| 10 | return 0; |

|  |  |
| --- | --- |
| 11 | } |
| 12 |  | |

|  |  |
| --- | --- |
| 13 | /\* 编译并运行 \*/ |
| 14 | edsionte@edsionte-laptop:~/code/expertC$ gcc init\_array.c -o init\_array | |

|  |  |  |
| --- | --- | --- |
| 15 | edsionte@edsionte-laptop:~/code/expertC$ ./init\_array | |
| 16 | length(m)=3 |

|  |  |
| --- | --- |
| 17 | length(n)=3 |

注意，在最后一个初始值后面可以继续加一个逗号也可以省略，这并不影响数组的长度。

对于多维数组而言，通常使用嵌套的大括号进行多维数组的初始化。由于多维的数组其实是有若干个一维数组构成的，则每个大括号都代表一个一维数组。对于多维数组而言只能省略最左边 下标的长度。

|  |  |  |
| --- | --- | --- |
| 01 | #include < stdio.h > | |
| 02 |  |

|  |  |  |
| --- | --- | --- |
| 03 | int main() | |
| 04 | { |

|  |  |
| --- | --- |
| 05 | int b[][3] = {1,2,1,1}; |
| 06 | int c[][3] = {{1,2,1},{1,2,3},}; | |

|  |  |
| --- | --- |
| 07 |  |
| 08 | printf("length(b)=%d\n",sizeof(b)/sizeof(b[0])); | |

|  |  |  |
| --- | --- | --- |
| 09 | printf("length(c)=%d\n",sizeof(c)/sizeof(c[0])); | |
| 10 | return 0; |

|  |  |
| --- | --- |
| 11 | } |
| 12 |  | |

|  |  |
| --- | --- |
| 13 | /\* 编译并运行 \*/ |
| 14 | edsionte@edsionte-laptop:~/code/expertC$ gcc init\_array.c -o init\_array | |

|  |  |  |
| --- | --- | --- |
| 15 | edsionte@edsionte-laptop:~/code/expertC$ ./init\_array | |
| 16 | length(b)=2 |

|  |  |
| --- | --- |
| 17 | length(c)=2 |

可以看到，不使用大括号也可以对多维数组进行初始化，只不过代码可读性较差

## 多维数组与逗号表达式

一旦涉及到多维数组，总有些让你迷惑的地方。比如：

|  |  |
| --- | --- |
| 1 | char ma[2][3][2]={ |
| 2 | {{1,2},{2,3},{3,4}}, | |

|  |  |  |
| --- | --- | --- |
| 3 | {{3,5},{4,5},{3,3}} | |
| 4 | }; |

|  |  |
| --- | --- |
| 5 |  |
| 6 | sizeof(ma[0,1,1])=？ | |

对于上面的代码，我们最后的迷惑点都可能落在ma[0,1,1]上。难道多维数组可以这样使用吗？如果ma[0,1,1]和ma[0][1][1]等价，那么sizeof(ma[0,1,1])的值就是1。很可惜这样的猜测是不正确的，正确答案为6。再比如下面的代码：

|  |  |
| --- | --- |
| 1 | char ma[3][2] = { |
| 2 | (1,2),(3,4),(5,3) |

|  |  |  |
| --- | --- | --- |
| 3 | }; | |
| 4 |  |

|  |  |
| --- | --- |
| 5 | ma[0][0]=？ |

上述代码是为数组ma进行初始化，那么ma[0][0]的值是多少？恐怕很多人都会认为是1。不过正确答案是2。

这两个问题都涉及到了逗号表达式。如果你对逗号表达式有基本的了解，那么也就没有上述那种莫名其妙的迷惑了。根据逗号表达式的运算，对于举例1中的ma[0,1,1]实际上等价于ma[1]；对于举例2中的初始化其实等价为char ma[3][2] = {2,4,3}。

## **字节对齐**

2、下面这个程序的输出结果是什么？  
struct value {  
char a[3];  
short b;  
};  
struct value tmp;  
printf(“sizeof(tmp) is %d\n”, sizeof(tmp));

**解答：**  
**sizeof(tmp) is 6**

## C语言中的名字空间

在说明名字空间之前，请先阅读下面的代码：

|  |  |
| --- | --- |
| 01 | /\* |
| 02 | \*Author: edsionte | |

|  |  |  |
| --- | --- | --- |
| 03 | \*Email:  edsionte@gmail.com | |
| 04 | \*Time:   2011/02/03 |

|  |  |  |
| --- | --- | --- |
| 05 | \*/ | |
| 06 |  |

|  |  |
| --- | --- |
| 07 | #include < stdio.h > |
| 08 | #include < string.h > | |

|  |  |
| --- | --- |
| 09 |  |
| 10 | struct id | |

|  |  |
| --- | --- |
| 11 | { |
| 12 | int id; | |

|  |  |  |
| --- | --- | --- |
| 13 | }id; | |
| 14 |  |

|  |  |  |
| --- | --- | --- |
| 15 | typedef struct name | |
| 16 | { |

|  |  |  |
| --- | --- | --- |
| 17 | char name[20]; | |
| 18 | }name; |

|  |  |
| --- | --- |
| 19 |  |
| 20 | struct name name1; | |

|  |  |  |
| --- | --- | --- |
| 21 | name name2; | |
| 22 |  |

|  |  |  |
| --- | --- | --- |
| 23 | int main() | |
| 24 | { |

|  |  |
| --- | --- |
| 25 | id.id = 1; |
| 26 | strcpy(name1.name,"hello,"); | |

|  |  |
| --- | --- |
| 27 | strcpy(name2.name,"edsionte!"); |
| 28 | printf("id.id = %d, %s%s\n",id.id,name1.name,name2.name); | |

|  |  |  |
| --- | --- | --- |
| 29 | return 0; | |
| 30 | } |

|  |  |
| --- | --- |
| 31 | /\* 运行结果 \*/ |
| 32 | edsionte@edsionte-laptop:~/code/expertC$ gcc tpdef.c -o tpdef | |

|  |  |  |
| --- | --- | --- |
| 33 | edsionte@edsionte-laptop:~/code/expertC$ ./tpdef | |
| 34 | id.id = 1, hello,edsionte! |

你可能已经发现在上述代码中出现了多个id和name，并且这样的代码可以成功的编译。这些相同的名字标签为何可以同时出现？每个标签代表什么含义？这些问题将是下面分析的重点。

以上述代码中10到13行的代码为例，这条语句中包含三个id标签。它们分别对应C语言中三种常见的名字空间：

* 结构标签：这种标签用于结构体、联合体和枚举类型；struct后的id即为此类型的名字空间；
* 成员名：每个结构体或联合体内部都与属于自己的名字空间；struct内部的成员id即为此类型；
* 标签名：声明中的标示符；比如最后一个id，他是struct id类型的变量；

由于这三种标签所处的名字空间不同，因此它们可以同时存在。但是在同一个名字空间中不能出现多个同名的标签。常见的例子就是一个结构体内不可能出现同名的变量。

根据上面对名字空间的划分，15到18行的代码的解释为：struct后的name属于结构体标签；结构体内部的name属于成员名；而最后一个name属于声明的标示符；整条语句的含义是通过typedef声明将name结构体重命名为name。

通过上面对typedef的分析，你应该对于struct name和name均可以声明一个变量不再陌生。此处我们用名字空间的来理解他们的区别，20句中的name属于结构标签，21句中的name属于一种类型的名称。

上述同名的情况在日常的代码中实属罕见，这里只是为了说明名字空间而特别的举例。一般为了提高代码的可阅读性，最好对容易产生混淆的标签加上特别标记。比如VFS中inode和dentry结构体，两者内部均有flag一字段。尽管不同的结构体内有各自的名字空间，但是实际命名时仍然采用i\_flag和d\_flag。

## typedef int x和#define x int是不一样的

typedef和宏定义看似都是文本替换，但其实质不同。typedef表面上是对已有数据类型引入新名称，实则是对数据类型的严格封装。这种封装体现在下述两个方面。

首先，经过宏定义后的类型名可以进行再次扩展，但是经过typedef引入的类型名则不能进行扩充。比如：

|  |  |
| --- | --- |
| 1 | #define myint1 int |
| 2 | unsigned myint1 x; /\* 正确 \*/ | |

|  |  |
| --- | --- |
| 3 |  |
| 4 | typedef int myint2; | |

|  |  |
| --- | --- |
| 5 | unsigned myint2 x; /\* 语法错误！ \*/ |

由于typedef是一种严格的数据封装，它只引入了myint2类型而没有引入unsigned myint2类型。也就是说，通过typedef的声明，编译器只能识别myint2类型。而unsigned myint2既不是基本类型也不是经过typedef声明过的类型，编译器就无法识别。

其次，在连续的几个变量声明中，使用typedef定义的类型能够保证所有变量均为相同类型，而用宏定义的变量则无法保证统一性。比如：

|  |  |
| --- | --- |
| 1 | #define myint1 int \* |
| 2 | myint1 x,y; /\* 经过宏替换后为: int \*x,y; \*/ | |

|  |  |
| --- | --- |
| 3 |  |
| 4 | typedef int \* myint2; | |

|  |  |
| --- | --- |
| 5 | myint2 x,y; |

由于宏定义只是直接的文本替换，因此只能保证x是整型的指针变量而y为整型变量。而typedef定义过的类型myint2则是对int \*的完全封装，所以x和y均为整型的指针变量。

## 联合体变量union

共用体内可以定义多个不同类型的成员变量 ，但是每次只能使用一个变量。即这些变量共同使用一段内存空间，这段内存空间的大小由这些变量中长度最长的变量决定。

## 枚举类型变量enum

定义一个枚举类型的变量：

|  |  |  |
| --- | --- | --- |
| 1 | enum return\_value | |
| 2 | { |

|  |  |
| --- | --- |
| 3 | RET\_TURE; |
| 4 | RET\_FALSE; | |

|  |  |  |
| --- | --- | --- |
| 5 | RET\_RANDOM; | |
| 6 | } |

枚举类型中每一个枚举常量都是用标示符表示的整形常量。上述枚举类型中的各个枚举常量并未赋值，但是默认值依次为0，1，2。

## [const和enum](http://edsionte.com/techblog/archives/1756)

#### const

我们可以将const关键字所修饰的变量看作是只读变量，也就是说这个变量不可被修改，只能读取其值。但它决不是常量。在标准C语言中，下面的语句是错误的：

|  |  |
| --- | --- |
| 1 | const int i=5; |
| 2 | int a[i]={1,2,3,4,5}; | |

诸如5，“hello”这样的常量以及宏定义过的常量之所以不能被修改是因为在内存中没有为他们分配空间，它们存储在运算单元的寄存器中。而上述所言的const修改过的变量说到底还是一个变量，内存中有存储那个变量的地方。只不过，这个变量有些特殊：只能读取。现在再看上面的代码，由于i为变量，它不能用来初始化数组的大小。

接着看下面的代码：

|  |  |
| --- | --- |
| 1 | typedef char \* charp; |
| 2 | char string[4] = "abc"; | |

|  |  |
| --- | --- |
| 3 | const char \*p1 = string; |
| 4 | const charp p2 = string; |

|  |  |
| --- | --- |
| 5 | p1++; |
| 6 | p2++; |

最后一句会报错，为什么？

在第三句中，p1是可变的，而\*p是const的；第四句中，p2是const的，因此不能进行++；也许你此时对p和\*p哪一个为const很迷惑。但是没有关系，这里有一个小技巧：**将上述语句中的数据类型去掉，离const最近的就是只读的**。比如const char \*p1 = string去掉char后，\*p离const最近，那么\*p是const的；再如const charp p2 = string去掉数据类型charp后，p2离const最近，那么p2就是不可改变的。

不过，对于类似const (char \*)p1 = string;这样的语句，上述办法有点阻碍。不过你可以将（char\*）看成一个整体（你也可以认为是用typedef定义了一个新类型），那么就可以继续用上面的方法。

#### enum

关于中断处理函数的返回值类型，有下面的定义：

//linux/include/linux/interrupt.h

typedef irqreturn\_t (\*irq\_handler\_t)(int, void \*);

//linux/include/linux/irqreturn.h

enum irqreturn {

IRQ\_NONE,

IRQ\_HANDLED,

IRQ\_WAKE\_THREAD,

};

typedef enum irqreturn irqreturn\_t;

有一些书上使用的是int型，并且使用int也可以通过。这是为什么？

其实，了解了枚举类型的本质，这个问题就一目了然了。枚举类型的定义形式如下：

enum  
{ 枚举常量1，枚举常量2，……，枚举常量n};

而枚举常量是用标示符表示的整形常量。并且C语言规定枚举常量的默认值依次为：0，1，……，n-1。对应到上面的定义代码，enum irqreturn中三个常量的值即为0，1，2。

## 大端模式小端模式

大端模式：数据的高字节存储在低地址中，低字节存储在高地址中

小端模式：数据的高字节存储在高字节地址中，低字节存储在低字节地址中

验证系统采用哪种字节顺序测试程序：

int GetMachineType()

{

int x = 1 /\*大端:[0000 0001],小端:[1000,0000]\*/;

if (\*(unsigned char \*)&x == 1)

{

return 0 /\*小端\*/

}

else

{

return 1 /\*大端\*/

}

}

void   main()

{

     int            num =  0x12345678;

     printf("num[0]=[%x]\n"  ,  \*(((char\*)&num+0)));

     printf("num[1]=[%x]\n"  ,  \*(((char\*)&num+1)));

     printf("num[2]=[%x]\n"  ,  \*(((char\*)&num+2)));

     printf("num[3]=[%x]\n"  ,  \*(((char\*)&num+3)));

}

    如果输出是：

    num[0]=12

    num[1]=34

    num[2]=56

    num[3]=78

则说明系统是采用的大端字节顺序

    如果输出是：

    num[0]=78

    num[1]=56

    num[2]=34

    num[3]=12

则说明系统是采用小端字节顺序

### 判断机器是大端还是小端的函数

#### 试题1

请写一个C函数，若处理器是Big\_endian的，则返回0；若是Little\_endian的，则返回1

思路：（数字确定后，根据大小端规则，判断内存值）

解答：

int checkCPU()

{

　{

　　union w

　　{

　　　int a;

　　　char b;

　　} c;

　　c.a = 1;

　　return (c.b == 1);

　}

}

#### 试题2

思路（内存值确定后，根据大小端规则和数据类型，写出内存中代表具体的数）

请写出程序的输出

#include <stdio.h>

#include <stdlib.h>

int main(void)

{

union

{

int i;

char x[5];

}a;

a.x[0] = 5;

a.x[1] = 4;

a.x[2] = 3;

a.x[3] = 2;

a.x[4] = 1;

printf("\n a.i = %x", a.i); /\* 这里i 只有四个字节，所以没有最低位的a.x[4] \*/

printf("\n a.i = %p, a.x=%p", &a.i, &a.x);

printf("\n a.x[3] = %p, a.x[4] = %p, sizeof(a)= %d", &a.x[3], &a.x[4], sizeof(a));

return 0;

}

## java中char和C语言中char的区别

C中char类型的变量占用一个字节的内存，因为c语言中字符使用ASCII（ American Standard Code for Information Interchange的）编码（0-127）

java中char类型的变量占用两个字节（没有符号位），因为java中字符采用Unicode编码（范围0-65535，其中0到127和ASCII一样）

## TLV标签用法总结

tag标签的属性为bit，由16进制表示，占1～2个字节长度。

例如，“9F33”为一个占用两个字节的tag标签。

而“95”为一个占用一个字节的tag标签。若tag标签的第一个字节（注：字节排序方向为从左往右数，第一个字节即为最左边的字节。bit排序规则同理。）的后五个bit为“11111”，则说明该tag占两个字节，例如“9F33”；否则占一个字节，例如“95”。

子域长度（即L本身）的属性也为bit，占1～3个字节长度。具体编码规则如下：

a) 当L字段**最左边字节**的最左bit位（即bit8）为0，表示该L字段占一个字节，

它的后续7个bit位（即bit7～bit1）表示子域取值的长度，采用二进制数表示子域取值长度的十进制数。

例如，某个域取值占3个字节，那么其子域取值长度表示为“00000011”。

所以，若子域取值的长度在1～127字节之间，那么该L字段本身仅占一个字节。

b) 当L字段**最左边字节**的最左bit位（即bit8）为1，

表示该L字段不止占一个字节，那么它到底占几个字节由该最左字节的后续7个bit位（即bit7～bit1）的十进制取值表示。

例如，若最左字节为10000010，表示L字段除该字节外，后面还有两个字节。其后续字节的十进制取值表示子域取值的长度。

例如，若L字段为“1000 0001 1111 1111”，表示该子域取值占255个字节。

所以，若子域取值的长度在128～255字节之间，那么该L字段本身需占两个字节。

## strtok函数注意事项

char \*strtok (char  \*  s,  const char \*delim);

strtok函数会在参数s中发现分隔符时会将该字符替换成‘\0’,因此是破坏原字符串的结构, 所以为了不破坏原字符串，应将原字符串的拷贝作为要分隔的参数

第一次调用时必须给参数 s赋值，以后每次调用参数s赋值为NULL，

每次调用成功会返回下一个分割后的字符串指针

## UTC时间与GMT时间

整个地球分为24时区，每个时区都有自己的本地时间，在国际无线电通信场合，为了统一起见，使用一个统一的时间，称为UTC(Universal Time Coordinated)通用协调时,UTC时间与格林尼治平均时GMT(Greenwich Mean Time)一样，都与英国伦敦的本地时间相同

## 常用时间函数

[**linux**](http://lib.csdn.net/base/linux)下存储时间常见的有两种存储方式，一个是从1970年到现在经过了多少秒，一个是用一个结构来分别存储年月日时分秒的。

struct timeval  
{  
   long tv\_sec; /\*秒\*/  
   long tv\_usec; /\*微秒\*/  
};

而直接存储年月日的是一个结构：

struct tm  
{  
   int tm\_sec; /\*秒，正常范围0-59， 但允许至61\*/  
   int tm\_min; /\*分钟，0-59\*/  
   int tm\_hour; /\*小时， 0-23\*/  
   int tm\_mday; /\*日，即一个月中的第几天，1-31\*/  
   int tm\_mon; /\*月， 从一月算起，0-11\*/  
   int tm\_year; /\*年， 从1900至今已经多少年\*/  
   int tm\_wday; /\*星期，一周中的第几天， 从星期日算起，0-6\*/  
   int tm\_yday; /\*从今年1月1日到目前的天数，范围0-365\*/  
   int tm\_isdst; /\*日光节约时间的旗标\*/  
};

1、ftime(struct  timeb \* tmb)

struct  timeb {

time\_t time;//1970年1月1日至今的秒数

unsigned short millitm;//千分之一秒

short timezone;*//和greenwich时间差了多少分钟*

short dstflag:*//日光节约时间的状态*

} ;

2、int getimeofday ( struct timeval \*tv, struct timezone \*tz ):

struct timeval {

long tv\_sec; *//秒*

long tv\_usec*;//微秒*

};

**struct timezone** {

int tz\_minuteswest: *//和greenwich时间差了多少分钟*

int tz\_dsttime*;//日光节约时间的状态*

};

3、struct tm    \* gmtime (const time\_t \*timep);此函数返回的时间日期未经过时区转换的UTC时间

4、struct tm   \*  localtime (const time\_t \*timep),此函数返回的时间日期已经当地的时间

struct tm

(

int rm\_sec;  //代表秒数,正常范围0-59，但允许至61秒

int tm\_min;   //代表分数,范围0-59

int tm\_hour; //从午夜算起的时数，范围:0-23

int tm\_mday; //代表月份的日数,范围0-31

int tm\_mon;  //代表月份,从一月算起，范围0-11

int tm\_year;  //1900年算起至今的年数

int tm\_wduy;//一星期中的日数，从星期日算起，范围0-6

int tm\_yday;//从1月1日起至今的天数，范围：0-365

int tm\_isdst;//日光借节约时间旗标

):

5、strftime (char \*s, size\_t max. const char \*format. const struct tm \*tm) ;,环境变量TZ和TC\_TIME会影响此函数的结果

format格式如下：

%a 星期几的简写

%A 星期几的全称

%b 月份的简写

%B 月份的全称

%c 标准的日期的时间串

%C 年份的前两位数字

%d 十进制表示的每月的第几天

%D 月/天/年

%e 在两字符域中，十进制表示的每月的第几天

%F 年-月-日

%g 年份的后两位数字，使用基于周的年

%G 年份，使用基于周的年

%h 简写的月份名

%H 24小时制的小时

%I 12小时制的小时

%j 十进制表示的每年的第几天

%m 十进制表示的月份

%M 十时制表示的分钟数

%n 新行符

%p 本地的AM或PM的等价显示

%r 12小时的时间

%R 显示小时和分钟：hh:mm

%S 十进制的秒数

%t 水平制表符

%T 显示时分秒：hh:mm:ss

%u 每周的第几天，星期一为第一天 （值从1到7，星期一为1）

%U 第年的第几周，把星期日作为第一天（值从0到53）

%V 每年的第几周，使用基于周的年

%w 十进制表示的星期几（值从0到6，星期天为0）

%W 每年的第几周，把星期一做为第一天（值从0到53）

%x 标准的日期串

%X 标准的时间串

%y 不带世纪的十进制年份（值从0到99）

%Y 带世纪部分的十制年份

%z，%Z 时区名称，如果不能得到时区名称则返回空字符。

%% 百分号

6、time\_t  time (time\_t\*t);

返回从公元1970年1月1日的UTC时间从0点0分0秒算起至今所经过的秒数，

若t非空的话，此函数也会将返回值保存到t指针所指的内存中

7、time\_t mktime (struct tm \*timeptr ):用来将timeptr所指的tm结构数据转换成

     从公元1970年1月1日0时0分0秒算起至今的UTC时间所经过的秒数

## strlen函数注意事项

当字符串中含二进制时，使用strlen一定要注意，strlen判断长度以'\0'为截止符的

## sprintf输出位数不足左补0用法

char   caTmp[12+1];

sprintf(caTmp,**"%012ld**",caBuf);

## 取结构体变量偏移量

#define  offset (type,field) (size\_t)((char \*)&( (type\*0)->field ) - (char \*)(type \*)0)

## 字符串查找函数

1、void \* memchr(const  void \*s, intc, size\_t n):

2、 char   \* strchr (const char\*s, int c);

**3、**char \* strrchr <const char \* s, int  c):

4、 char \*strstr (cons! char \*haystack. const char \*needle);

## sscanf函数用%[]来分隔字符串

1. 常见用法。

　　char buf[512] = ;

　　sscanf("123456 ", "%s", buf);

　　printf("%s\n", buf);

　　结果为：123456

　　2. 取指定长度的字符串。如在下例中，取最大长度为4字节的字符串。

　　sscanf("123456 ", "%4s", buf);

　　printf("%s\n", buf);

　　结果为：1234

　　3. 取到指定字符为止的字符串。如在下例中，取遇到空格为止字符串。

　　sscanf("123456 abcdedf", "%[^ ]", buf);

　　printf("%s\n", buf);

　　结果为：123456

　　4. 取仅包含指定字符集的字符串。如在下例中，取仅包含1到9和小写字母的字符串。

　　sscanf("123456abcdedfBCDEF", "%[1-9a-z]", buf);

　　printf("%s\n", buf);

　　结果为：123456abcdedf

　　5. 取到指定字符集为止的字符串。如在下例中，取遇到大写字母为止的字符串。

　　sscanf("123456abcdedfBCDEF", "%[^A-Z]", buf);

　　printf("%s\n", buf);

　　结果为：123456abcdedf

　　6、给定一个字符串iios/12DDWDFF@122，获取 / 和 @ 之间的字符串，先将 "iios/"过滤掉，再将非'@'的一串内容送到buf中

　　sscanf("iios/12DDWDFF@122", "%\*[^/]/%[^@]", buf);

　　printf("%s\n", buf);

　　结果为：12DDWDFF

　　7、给定一个字符串““hello, world”，仅保留world。（注意：“，”之后有一空格）

　　sscanf(“hello, world”, "%\*s%s", buf);

　　printf("%s\n", buf);

　　结果为：world

　　%\*s表示第一个匹配到的%s被过滤掉，即hello被过滤了

　　如果没有空格则结果为NULL。

　　sscanf的功能很类似于正则表达式, 但却没有正则表达式强大,所以如果对于比较复杂的字符串处理,建议使用正则表达式.

　　sscanf,表示从字符串中格式化输入

　　上面表示从str中，输入数字给x，就是32700

　　久以前，我以为c没有自己的split string函数，后来我发现了sscanf；一直以来，我以为sscanf只能以空格来界定字符串，现在我发现我错了。

　　sscanf是一个运行时函数，原形很简单：

　　int sscanf(

　　const char \*buffer,

　　const char \*format [,

　　argument ] ...

　　);

　　它强大的功能体现在对format的支持上。

　　我以前用它来分隔类似这样的字符串2006:03:18:

　　int a, b, c;

　　sscanf("2006:03:18", "%d:%d:%d", a, b, c);

　　以及2006:03:18 - 2006:04:18:

　　char sztime1[16] = "", sztime2[16] = "";

　　sscanf("2006:03:18 - 2006:04:18", "%s - %s", sztime1, sztime2);

　　但是后来，我需要处理2006:03:18-2006:04:18

　　仅仅是取消了‘-’两边的空格，却打破了%s对字符串的界定。

　　我需要重新设计一个函数来处理这样的情况？这并不复杂，但是，为了使所有的代码都有统一的风格，我需要改动很多地方，把已有的sscanf替换成我自己的分割函数。我以为我肯定需要这样做，并伴随着对sscanf的强烈不满而入睡；一觉醒来，发现其实不必。

　　format-type中有%[]这样的type field。如果读取的字符串，不是以空格来分隔的话，就可以使用%[]。

　　%[]类似于一个正则表达式。[a-z]表示读取a-z的所有字符，[^a-z]表示读取除a-z以外的所有字符。

　　所以那个问题也就迎刃而解了:

　　sscanf("2006:03:18 - 2006:04:18", "%[0-9,:] - %[0-9,:]", sztime1, sztime2);

## printf("%lf")与scanf("%lf")

printf()的%f既可以输出float类型的变量也可以输出double类型的变量（根据默认参数提升规则）float型会被提升为double型

因此printf只会看到双精度型double，严格的讲%lf是没有定义的但是在很多系统可能会接受它，

要确保可移植性就要坚持使用%f

对于scanf就不一样了，它接受指针，这里没有类似的类型提升，向float型存储和double型存储不一样

因此scanf区分%f和%lf

short   int  类型的变量scanf输入的时候必要用scanf("%hd");

## printf("%\*d", width, x);

printf("%\*d", width, x);格式符中中的\*星号表示，参数列表中的一个int值用来表示域的宽度，从而可以实现可变的域宽度

## printf("%ld\n")

long   int  类型的变量必须用用“%ld”格式输出

## 如何用printf输出%

printf("%%\n");即可

printf输出%之所以困难是因为%是printf的转义字符

任何时候printf遇到%都会等待一下一个字符，然后决定如何处理。

而%%字符序列就被定义成了单独的%字符

## if (0 == x)与if (x == 0)

if (0 == x)比if (x == 0)更好因为可以避免if(0 = x)的错误，0=x 在编译时会报错

## 函数是通过指针调用的

例如：

int    r，（\*fp）， func() ;

fp = func ;

r = (\*fp)();

最后一句r = (\*fp)() ； fp是一个函数指针，\*fp是个函数，

函数总是通过指针来调用的，所有“真正的”函数名在表达式和初始化中，总是隐式地退化为指针，无论fp是函数名还是指针，

r = fp();都是合法的且能正确工作，使用显示的\*号依然允许，且为了保证在较老的编译器的可移植性，这也是推荐的用法

## c语言中函数参数是通过值传递的

c语言中函数参数是通过值传递的即传入的是实参的一个副本

## \*p++

后缀++和--操作符比前缀一元操作符的优先级高，\*p++和\*(p++)等价，自增p幷返回p自增 之前所指向的值

## return 语句不能返回指向栈的指针

return 语句不能返回指向栈的指针

函数内定义的局部变量在栈中分配内存，函数执行完后会释放栈内存，如果函数返回函数内定义的局部变量的地址，会导致在函数的调用出，取得的是一个无效的地址。

例子：

Int  \*f1()

{

Int  temp = 1;

    return &temp;

}

Int  \*f2()

{

Int  tt= 99;

    return & tt;

}

Int main()

{

Int  \* p;

p=f1();

printf(“p=[%s]\n”,\*p);

f2();

printf(“p=[%s]\n”,\*p);

}

程序的执行结果为：

P=[1]

P=[99]

但是从程序来看应该两次都输出1才对，

这个错误是由，函数返回局部变量的地址引起的。p=f1（）会将函数f1（）中定义的局部变量temp地址返回给p，函数调用结束后，temp所占的存储空间被收回，但是p中任保留temp的地址。调用f2（）时，原来temp使用过的存储区域被分配给f2（）的局部变量tt，f2（）将此区域的内容改为99，因为p任指向该区域，所以p的值就被改写为99了。

## 正则表达式常用元字符

|  |  |
| --- | --- |
| **元字符** | **释义** |
| ^ | 只匹配行首 |
| $ | 只匹配行尾 |
| \* | 一个单字符后紧跟 \*，匹配 0 个或多个此单字符 |
| [ ] | 匹配[ ] 内字符。可以是一个单字符，也可以是字符序列。可以使用 - 表示[ ] 内字符序列范围，如用 [ 1 - 5 ] 代替[ 1 2 3 4 5 ] |
| \ | 用来屏蔽一个元字符的特殊含义。因为有时在 shell 中一些元字符有特殊含义。 \ 可以使其失去应有意义 |
| [^] | 匹配不在指定字符组内的字符 |
| \< | 词首定位符 |
| \> | 词尾定位符 |
| . | 匹配任意单字符 |
| x\{n\} | 用来匹配字符 x重复出现次数。 n 为次数 |
| x\{n， \} | 用来匹配字符 x重复出现次数，但次数最少为 n |
| x\{n， m\} | 用来匹配字符 x重复出现次数，但字符 x 出现次数在n 与 m之间 |

## return不能返回指向栈内存的指针

return 不能返回指向栈内存的指针，因为栈内存在函数结束时会自动销毁。

## 不用第三个变量交换两个数

int a ;

int b;

a = a + b;

b = a - b;

a = a - b;

或者：

a = a ^ b;

b = a ^ b;

a = a ^ b;

## ASSERT()是干什么用的？

ASSERT()是一个调试程序时经常使用的宏，在程序运行时它计算括号内的表达式，

如果表达式为FALSE (0), 程序将报告错误，并终止执行。

如果表达式不为0，则继续执行后面的语句。

这个宏通常原来判断程序中是否出现了明显非法的数据，

如果出现了终止程序以免导致严重后果，同时也便于查找错误。例如，变量n在程序中不应该为0，如果为0可能导致错误，你可以这样写程序：

......

ASSERT( n != 0);

k = 10/ n;

......

ASSERT只有在Debug版本中才有效，如果编译为Release版本则被忽略。

assert()的功能类似，它是ANSI C标准中规定的函数，它与ASSERT的一个重要区别是可以用在Release版本中

## 变量溢出---经典问题

下面这个程序执行后会有什么错误或者效果:

#define MAX 255

int main()

{

    unsigned char A[MAX],i;

    for (i=0;i<=MAX;i++)

      A[i]=i;

}

解答：MAX=255,数组A的下标范围为:0..MAX-1,这是其一,其二 当i循环到255时,

循环内执行: A[255]=255;这句本身没有问题，但是返回for (i=0;i<=MAX;i++)语句时,

由于unsigned char的取值范围在(0..255),i++以后i又为0了..无限循环下去.

注：char类型为一个字节，取值范围是[-128，127]，unsigned char [0 ,255]

## strcpy和memcpy

strcpy和memcpy主要有以下3方面的区别。

1、复制的内容不同。strcpy只能复制字符串，而memcpy可以复制任意内容，例如字符数组、整型、结构体、类等。

2、复制的方法不同。strcpy不需要指定长度，它遇到被复制字符的串结束符"\0"才结束，所以容易溢出。memcpy则是根据其第3个参数决定复制的长度。

3、用途不同。通常在复制字符串时用strcpy，而需要复制其他类型数据时则一般用memcpy

## 字符串查找函数

1、void \* memchr(const  void \*s, intc, size\_t n):

2、char \* strchr (const char\*s, int c);

**3、**char \* strrchr (const char \* s, int  c):

4、 char \*strstr (const char \*haystack. const char \*needle);

## 正态分布算法

#include <math.h>

E-期望，V-标准差

编译的时候一定要加上lm库

doulbe gaussrand(double E,double V)

{

static double V1, V2, S;

static int phase = 0;

double X;

if ( phase == 0 )

{

do

{

double U1 = (double)rand() / RAND\_MAX;

double U2 = (double)rand() / RAND\_MAX;

V1 = 2 \* U1 -1;

V2 = 2 \* U2 -1;

S = V1 \* V1 + V2 \* V2;

}while(S >= 1 || S == 0)

X = V1 \* sqrt(-2 \* log(S) / S);

}

else

{

X = V2 \* sqrt(-2 \* log(S) / S);

}

phase = 1 - phase;

X = X \* V + E;

return X;

}

## 字节顺序问题

小端字节顺序：低位字节放在内存的低地址端，高位字节放在内存的高地址端

大端字节顺序：高位字节放在内存的低地址端，低位字节放在内存的高地址端

socket通信时，网络字节顺序是大端字节顺序，socket通信时如果传输的数据中有多字节的数值类型的数据时一定要用htonl或htons或ntohs或ntohl进行转换，只有单字节字符型数据不存在大小端的问题

AIX系统是大端字节顺序，linux系统是小端字节顺序，在AIX中程序和linux系统程序进行通信时一定要进行字节顺序转换

## #pragma pack

#pragma pack(n)

每个特定平台上的编译器都有自己的默认“对齐系数”(也叫对齐模数)。程序员可以通过预编译命令#pragma pack(n)，n=1,2,4,8,16来改变这一系数，其中的n就是你要指定的“对齐系数”。

规则：

　　1、数据成员对齐规则：结构(struct)(或联合(union))的数据成员，第一个数据成员放在offset为0的地方，以后每个数据成员的对齐按照#pragma pack指定的数值和这个数据成员自身长度中，比较小的那个进行。

　　2、结构(或联合)的整体对齐规则：在数据成员完成各自对齐之后，结构(或联合)本身也要进行对齐，对齐将按照#pragma pack指定的数值和结构(或联合)最大数据成员长度中，比较小的那个进行。

#pragma pack(4)

struct node{

  int e;  
  char f;  
  short int a;  
  char b;

};

struct node n;

printf("%d\n",sizeof(n));

我自己算的结果是16，结果实际结果是：



然后结构体内部数据成员变动一下位置：

#pragma pack(4)

struct node{

  char f;  
  int e;  
  short int a;  
  char b;};

struct node n;

printf("%d\n",sizeof(n));



将对齐位数强制定位2

#pragma pack(2)

struct node{

  char f;  
  int e;  
  short int a;  
  char b;};

struct node n;

printf("%d\n",sizeof(n));



将对齐位数强制定位1

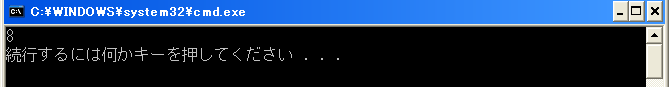
#pragma pack(1)

struct node{

  char f;  
  int e;  
  short int a;  
  char b;};

struct node n;

printf("%d\n",sizeof(n));



看着输出结果和文字描述有点晕，下面简单说一下俺的判定规则吧：

其实之所以有内存字节对齐机制，就是为了最大限度的减少内存读取次数。我们知道CPU读取速度比内存读取速度快至少一个数量级，所以为了节省运算花费时间，只能以牺牲空间来换取时间了。

下面举例说明如何最大限度的减少读取次数。

#pragma pack(1)

struct node{

  char f;  
  int e;  
  short int a;  
  char b;};

struct node n;

printf("%d\n",sizeof(n));

这里强制按照1字节进行对齐，可以理解成所有的内容都是按照1字节进行读取（暂且这样理解，因为这样可以很好的理解内存对其机制），其他所有的数据成员都是1字节的整数倍，所以也就不用进行内存对其，各个成员在内存中就按照实际顺序进行排列，结构体实际长度为8

#pragma pack(2)

struct node{

  char f;  
  int e;  
  short int a;  
  char b;};

struct node n;

printf("%d\n",sizeof(n));

这里强制按照2字节进行对齐。如果内存分布仍然是连续的话，那么int e就得三次才能读到CPU中，所以为了“讲究”int e的读取，所以在char f之后预留1BYTE，最后的char b也是如此，所以长度为10

#pragma pack(4)

struct node{

  char f;  
  int e;  
  short int a;  
  char b;};

struct node n;

printf("%d\n",sizeof(n));

这里强制按照4字节进行对齐。所以char f后要预留3BYTE，而short int a 和 cha b可以一次读取到CPU(按照4字节读取)，所以长度为12

如果#pramga pack(n)中的n大于结构体成员中任何一个成员所占用的字节数，则该n值无效。编译器会选取结构体中最大数据成员的字节数为基准进行对齐

## ld命令

### ld -m

ld –m 产生一个内存映射或列表，显示在可执行文件中什么地方放入了哪些符号。它同时显示了同一个符号的多个实例，通过查看报告的内容，用户可以判断是否发生了Interpositioning.

### ld -D

显示链接-编辑过程和所包含的输入文件。如果需要监视从archive中提取对象的过程。这个选项尤其有用。它同时显示运行时绑定信息。

ld是一个复杂的程序，还有很多其他选项和约定未在此说明。对于绝大多数应用来说，这些说明已经做够了。如果需要知道更多它的知识，下面提供了4种途径，按复杂程度，分列如下：

1. 使用ldd命令，列出执行文件的动态依赖集。这条命令会告诉你动态链接的程序所需要的函数库
2. ld程序的 –Dhelp选项能提供一些信息，有助于查找链接过程出现的问题。
3. 查看ld程序的在线文档
4. 阅读SunOs Linker and Libariies Manua(位于801-2869-10部分)

## CC 编译器手工加库

cc cjson.c -o cjson -I/chk/usr/include -L/chk/usr/lib libumcom.so

cc gaussrand.c -o gaussrand -lm

## CC 通过编译器提供的-G参数选项创建动态库

使用编译参数-G 即可，步骤分析如下：

1 ） 编译源程序

cc –c show.c -o show.o

2 ） 创建动态库

cc –G show.o -o show.so 这种方法，简单明了，

省掉好多重复的操作。

也可以直接:cc -G show.c -o show.so

## cc和gcc编译器在编译的时候加入全局宏定义

如果程序中有如 #ifdef \_REDHAT\_LINUX或者\_AIX\_预编译命令

#ifdef \_REDHAT\_LINUX

………

#endif

或者程序的日志开关

#ifdef \_DEBUG\_

Log（）;

#endif

可以在程序编译的时候利用编译器的参数-g -D来控制

如：在编译的时候定义宏变量\_REDHAT\_LINUX

cc -g -D \_REDHAT\_LINUX

gcc -g -D \_REDHAT\_LINUX

## 动态库与静态库的区别

1. 动态库体积小，节省内存空间。

静态库是在编译的时候就将库函数链接到可执行程序中，

动态库是在可执行程序执行时才链接到可执行程序中，动态库在被内核映射到内存后，会被使用到的所以可执行程序共享，即动态库在内存中只有一份拷贝，而静态库在被所有调用的可执行程序中都有一份拷贝

1. 静态库在可执行程序编译完成后可以删掉，而动态库在可执行程序编译后，不能删掉否则影响程序的运行。

动态库链接 是一种just-in-time（JIT）链接，程序在运行时必须能找到它们所需要的函数库。链接器通过把库文件名或路径名植入可执行文件中来做到这一点，所以动态库的路径不能随意移动否在运行时会导致失败

## gcc编译过程的4个阶段

从.c文件到可执行文件的整个编译过程。通常在使用gcc编译程序时，编译过程通常会分为4个阶段，即预处理（pre-processing），编译（compiling），汇编（assembling），链接（linking）。

### 1、预处理（pre-processing）

在预处理阶段，一般输入的是.c文件，而输出的是.i文件。在此阶段中通常会处理源文件中的预处理命令，比如#define，#include，#ifdef等命令。如果想要生成这种.i中间文件，那么可以使用下面命令：

gcc -E hello.c -o hello.i

### 2、编译（compiling）

在编译阶段，输入的是.i中间文件，输出的是.s汇编语言文件。可以使用下面的命令：

gcc -S hello.i -o test.s

### 3、汇编（assembling）

在汇编阶段，输入的是上一步产生的.s文件，产生的是二进制机器代码.o文件。此阶段对应的命令如下：

gcc -c test.s -o test.o

### 4、链接（linking）

链接阶段，输入的是.o二进制机器代码文件，连同其他的（如果有的话）机器代码文件和库文件一起汇集成一个可执行的二进制代码文件。

gcc test.o -o test

以上是一个程序编译完整的四个阶段。一般来说我们会将前三个阶段一步搞定，那么整个编译过程可以用下面两条命令就完成：

1 gcc -c test.c -o test.o

2 gcc test.o -o test

## gcc编译成动态库

1. 编译成二进制文件(-c只编译不链接)

gcc -o 二进制文件名.o -c 需要编译的文件名 $(CCFLG) -I$(INCSP) –L$(LIBSP)

CCFLG = -fPIC -std=c99 –wall -wno-implicit-function-declartion –wno-parentheses –wno-missing-braces -pipe -wpointer -arith -wsign-compare -funsigned-char

INCSP = /usr/local/include

LIBSP = /usr/local/lib64

1. 将二进制文件链接成动态库

gcc -o 动态库名.so -shared $(OBJS)

$(OBJS)为二进制对象列表

## gcc编译成静态库

ar $(SLFLG) -o静态库名.a $(OBJS)

$(OBJS)为二进制对象列表

SLFLG = -ruc

ar:archive（档案文件，把。。。存档）的缩写。

## -S选项编译成汇编指令

cc -S -Xc banana.c

这个-S选项使编译停在汇阶段，把汇编语言指令输出到bannana.c文件中。

-Xc选项告诉编译器拒绝任何不符合ANSI C的代码结构。当编写源代码时始终使用这个选项是一个好主意，因为它有助于程序获得最大的可移植性。

## 动态库调用流程

void \* pHandle = NULL;

int   (\*func)() = NULL;

char  \* pcaErrMsg = NULL;

int  iResult = 0;

pHandle= dlopen("动态库的绝对路径", RTLD\_NOW);

if (pHandle == NULL)

{

     异常处理

}

func=( int (\*)() )dlsym(pHandle,调用函数名)

if (( pcaErrMsg  =  dlerror() )  != NULL)

{

    dlclose(pHandle );

}

iResult  =(\*pHandle )(参数)

根据iResult的返回码异常处理

dlclose(pHandle )

if (( pcaErrMsg  =  dlerror() )  != NULL)

{

   return -1;

}

## AIX中生成动态库

对UNIX 平台，动态库的创建大同小异，都是通过编译器（实际上为ld ）生成。

除此之外，

在AIX 平台，系统提供了丰富的工具创建动态库。

从大的范围看有两种不同的方法可以创建动态库，

一种是按照传统的方式，编写输出符号文件，创建动态库。

一种是通过编译器提供的参数选项，自动生成输出符号文件创建动态库,下面将逐一介绍

### 1、按照传统的方式创建动态库

按照传统的方法创建动态库，关键的问题是如何编写输出符号文件，

VisualAge C/C++这一产品给我们提供了一个比较好的工具CreateExportList 帮助我们完

成。参考下面的例子，

将给出详细的步骤说明，如何创建输出符号文件。

测试用例为C 程序文件，文件名为show.c 和trans.c ，源码分别为：

/\*

\*\* show.c

\*/

#include <stdio.h>

void show(char \*pt){

printf("The showing text : %s/n", pt);

}

/\*

\*\* trans.c

\*/

#include <string.h>

void trans(char \*str1) {

char str[20];

strcpy(str, str1);

show(str);

}

上面的两个示例程序中，show.c 包含函数show() ，trans.c 包含函数

trans() ，借助于CreateExportList工具，首先生成输出符号文件，步骤如下：

1 ）编译源程序show.c 和trans.c

xlc –c show.c

xlc –c trans.c

生成相应的目标文件show.o 和trans.o

2 ）创建包含目标文件的纯文本文件

创建此文件的目的是帮助我们收集动态库包含的目标文件，这一步并不是必需的，假设文件名为

objlist ，内容如下：

show.o

trans.o

3 ）创建输出符号文件，文件包含动态库要输出的所有符号，假定文件名为exp.f

/usr/vac/bin/CreateExportList exp.f –f $(objlist)

如果没有objlist 文件，也可以将目标文件逐一列示：

/usr/vac/bin/CreateExportList exp.f show.o trans.o

查看文件exp.f 内容为：

show

trans

4 ）编辑输出符号文件，在文件起始处增加#!动态库名，假定要创建的动态库名为libct.so ，如下：

#!libct.so

5 ）编辑输出符号文件，删除不希望输出的符号。CreateExportList 创建的输出文件可能包含所有的函数，

无论是调用的还是被调用，需要删除掉不希望输出的符号，或调用的系统函数。

输出符号文件创建完成后，接下来，就可以创建动态库了。前面已经

提过，动态库是 由编译器创建的，这一点与静态库的创建有本质的区别，

如下：

xlc –bE:exp.f –bnoentry -bM:SRE –olibct.so show.o trans.o

本质上，这是有ld 完成的，xlc 只是将参数传递给ld 。这可以节省部分工

作，这是因为，相对于xlc ，ld 是比较最底层的操作，需要对系统提供的库比较熟悉。如果直接使用ld ，必须要清楚，创建库需要的所有资源。

假如我们简单的将以上命令的xlc 替换为ld ，请看以下输出结果：

[root@ibmp630#]ld -bE:exp.f -bnoentry -bM:SRE -o libct.so show.o trans.o

ld: 0711-317 ERROR: Undefined symbol: .printf

ld: 0711-317 ERROR: Undefined symbol: .strcpy

ld: 0711-345 Use the -bloadmap or -bnoquiet option to obtain more information

正确的使用命令，如下：

ld –bE:exp.f –bnoentry -bM:SRE –olibct.so show.o trans.o –lc

至此，按照传统的方法创建动态库已经完成。可以看到，以上的步骤

比较多，且 繁琐，

### 2、通过编译器提供的-G参数选项创建动态库

使用编译参数-G 即可，步骤分析如下：

1 ） 编译源程序

cc –c show.c -o show.o

2 ） 创建动态库

cc –G show.o -o show.so 这种方法，简单明了，

省掉好多重复的操作。

也可以直接:cc -G show.c -o show.so

## BBS段

BSS段是“Block Started by Symbol ”(有符号开始块)的缩写，它是旧式IBM704汇编程序的一个伪指令，UNIX借用了这个名字，至今依然沿用，有些人喜欢把它记作“Better Save Space”（更有效的节省空间）。由于BSS段只保存没有值的变量，所以事实上它并不需要保存这些变量的映像。运行时所需要的BSS段的大小会记录在目标文件中（Object），但BSS段（不像其他段）并不占用目标文件的任何空间。

## size命令

size 可执行程序

它可以告诉你可执行程序文件中的三个段（代码段、数据段和bss段）的大小

## nm工具

nm -x 可执行文件

## 两个浮点数比较相等

float a = 20;

float b = 40 – 20 ;

不能用 if ( a == b)来判断两个数是否相等

一定要用两个数相减去绝对值与0.00000001比较

if ( fabs(a - b) < 0.00000001)来判断

浮点数可以比较大小，但是不能直接判断两个浮点数相等。

## 内存泄露问题

程序中申请的内存后，没有free掉，会导致系统内存耗完dump机

## 数组越界问题

C语言不会对数据越界做检查，存放在数组中的内容如果超过了数组的大小会导致数组越界，非常危险，因为数组越界后会改变数组附件其他变量的数值，是程序发生一些不可预知的致命性错误

## 函数中在（stack）分配过大变量问题

在函数中定义的变量如果没有加static都是在堆中分配内存的, 否则在栈中分配内存

linux系统中用户栈的大小限制为8M(ulimit -a 查看stack size的大小)，如果在函数中定义变量大小超过8M，函数执行就会异常，特别是定义字符数组的时候一定要注意，如果实在要定义超过8M大小字符数组变量，可以在定义数组前加上static，使变量在静态数据区分配内存

## socket通信常见问题

1、字节顺序问题

单字节类型的变量不存在字节顺序问题，如char类型

但是多字节类型的变量如数字类型int long float   double等变量通过socket传输时就要注意字节顺序问题

字节顺序分为：大端顺序、小端顺序、网络字节顺序

大端顺序：多字节变量的高字节存放到内存的低地址端，低字节存放到内存的高地址端

小端顺序：多字节变量的低字节存放到内存的低地址端，高字节存放到内存的高地址端

网络字节顺序：采用大端顺序，数据在网络上传输时采用的是大端字节顺序

不同的系统会采用不同的字节顺序：AIX采用大端顺序，linux采用小端顺序

所以在AIX系统运行的程序与在linux系统运行的程序互相通信时必须调用如下函数进行字节顺序转换

htonl、htons，ntohl、ntohs

验证系统采用哪种字节顺序测试程序：

void   main()

{

     int            num =  0x12345678;

     printf("num[0]=[%x]\n"  ,  \*(((char\*)&num+0)));

     printf("num[1]=[%x]\n"  ,  \*(((char\*)&num+1)));

     printf("num[2]=[%x]\n"  ,  \*(((char\*)&num+2)));

     printf("num[3]=[%x]\n"  ,  \*(((char\*)&num+3)));

}

    如果输出是：

    num[0]=12

    num[1]=34

    num[2]=56

    num[3]=78

则说明系统是采用的大端字节顺序

    如果输出是：

    num[0]=78

    num[1]=56

    num[2]=34

    num[3]=12

则说明系统是采用小端字节顺序

}

2、读写的超时时间

读超时时间如果设置的的不当很容易出现read串包问题

例如：客户端write完后等待read如果write和read的间隔设置的过短

由于超时时间设置的太短，很容易超时，如果本次读超时了，函数就退出了，

程序继续进行下一轮的write和read，服务器端发送过来的read内容还存在于socket中

从而造成下一轮read的内容是还是上一轮服务器发送过来的内容

## C语言写的windows关机小程序

int main()

{

char cmd[20]="shutdown -s -t ";

char t[5]="0";

int c;

system("title C语言关机程序"); //设置cmd窗口标题

system("mode con cols=48 lines=25"); //窗口宽度高度

system("color f0"); //可以写成 red 调出颜色组

system("date /T");

system("TIME /T");

printf("----------- C语言关机程序 -----------\n");

printf("1.实现10分钟内的定时关闭计算机\n");

printf("2.立即关闭计算机\n");

printf("3.注销计算机\n");

printf("0.退出系统\n");

printf("-------------------------------------\n");

scanf("%d",&c);

switch(c) {

case 1:

printf("您想在多少秒后自动关闭计算机？（0~600）\n");

scanf("%s",t);

system(strcat(cmd,t));

break;

case 2:

system("shutdown -p");

break;

case 3:

system("shutdown -l");

break;

case 0:

break;

default:

printf("Error!\n");

}

system("pause");

return 0;

}

## 程序的4大区

栈区：局部变量，如果未初始化变量值未知

堆区：malloc分配的变量，如果未初始化变量值未知

全局数据区：全局变量和static变量，数据区的变量未初始化时，变量会初始化为0

代码区：代码指令

## signal函数原型

void ( \*signal(int sig, void (\*func)(int)) )(int);

## sigwait函数

## Linux多线程编程

### 多线程编程与多进程编程中信号处理的区别

在[Linux](http://lib.csdn.net/base/linux)的多线程中使用信号机制，与在进程中使用信号机制有着根本的区别，可以说是完全不同。在进程环境中，对信号的处理是，先注册信号处理函数，当信号异步发生时，调用处理函数来处理信号。它完全是异步的（我们完全不知到信号会在进程的那个执行点到来！）。然而信号处理函数的实现，有着许多的限制；比如有一些函数不能在信号处理函数中调用；再比如一些函数read、recv等调用时会被异步的信号给中断(interrupt)，因此我们必须对在这些函数在调用时因为信号而中断的情况进行处理（判断函数返回时 enno 是否等于 EINTR）

但是在多线程中处理信号的原则却完全不同，它的基本原则是：将对信号的异步处理，**转换成同步处理**，也就是说用一个线程专门的来“同步等待”信号的到来，而其它的线程可以完全不被该信号中断/打断(interrupt)。这样就在相当程度上简化了在多线程环境中对信号的处理。而且可以保证其它的线程不受信号的影响。这样我们对信号就可以完全预测，因为它不再是异步的，而是同步的（我们完全知道信号会在哪个线程中的哪个执行点到来而被处理！）。而同步的编程模式总是比异步的编程模式简单。其实多线程相比于多进程的其中一个优点就是：多线程可以将进程中异步的东西转换成同步的来处理。

### sigwait函数

 #include <signal.h>

 int sigwait(const sigset\_t \**set*, int \**sig*);

sigwait是**同步**的等待信号的到来，而不是像进程中那样是异步的等待信号的到来。sigwait函数使用一个信号集作为他的参数，并且在集合中的任一个信号发生时返回该信号值，解除阻塞，然后可以针对该信号进行一些相应的处理。

在多线程代码中，总是使用sigwait或者sigwaitinfo或者sigtimedwait等函数来处理信号。

 而不是signal或者sigaction等函数。因为在一个线程中调用signal或者sigaction等函数会改变所有线程中的，信号处理函数。而不是仅仅改变调用signal/sigaction的那个线程的信号处理函数。

### ****pthread\_sigmask函数****

 #include <signal.h>

 int pthread\_sigmask(int *how*, const sigset\_t \**set*, sigset\_t \**oldset*);

How:

SIG\_BLOCK:     结果集是当前集合参数集的并集  
SIG\_UNBLOCK: 结果集是当前集合参数集的差集  
SIG\_SETMASK: 结果集是由参数集指向的集

  每个线程均有自己的信号屏蔽集（信号掩码），可以使用pthread\_sigmask函数来屏蔽某个线程对某些信号的响应处理，仅留下需要处理该信号的线程来处理指定的信号。实现方式是：利用线程信号屏蔽集的继承关系

  （在主进程中对sigmask进行设置后，主进程创建出来的线程将继承主进程的掩码）

### ****pthread\_kill函数****

 #include <signal.h>

 int pthread\_kill(pthread\_t *thread*, int *sig*);

   在多线程程序中，一个线程可以使用pthread\_kill对同一个进程中指定的线程（包括自己）发送信号。注意在多线程中  一般不使用kill函数发送信号，因为kill是对进程发送信号，结果是：正在运行的线程会处理该信号，如果该线程没有注册信号处理函数，那么会导致整个进程退出。

### ****1.Linux进程与线程****

Linux进程创建一个新线程时，线程将拥有自己的栈（因为线程有自己的局部变量），但与它的创建者共享全局变量、文件描述符、信号句柄和当前目录状态。

Linux通过fork创建子进程与创建线程之间是有区别的：fork创建出该进程的一份拷贝，这个新进程拥有自己的变量和自己的PID，它的时间调度是独立的，它的执行几乎完全独立于父进程。

进程可以看成一个资源的基本单位，而线程是程序调度的基本单位，一个进程内部的线程之间共享进程获得的时间片。

### ****2.\_REENTRANT宏****

在一个多线程程序里，默认情况下，只有一个errno变量供所有的线程共享。在一个线程准备获取刚才的错误代码时，该变量很容易被另一个线程中的函数调用所改变。类似的问题还存在于fputs之类的函数中，这些函数通常用一个单独的全局性区域来缓存输出数据。

为解决这个问题，需要使用可重入的例程。可重入代码可以被多次调用而仍然工作正常。编写的多线程程序，通过定义宏\_REENTRANT来告诉编译器我们需要可重入功能，这个宏的定义必须出现于程序中的任何#include语句之前。

\_REENTRANT为我们做三件事情，并且做的非常优雅：

（1）它会对部分函数重新定义它们的可安全重入的版本，这些函数名字一般不会发生改变，只是会在函数名后面添加\_r字符串，如函数名gethostbyname变成gethostbyname\_r。

（2）stdio.h中原来以宏的形式实现的一些函数将变成可安全重入函数。

（3）在error.h中定义的变量error现在将成为一个函数调用，它能够以一种安全的多线程方式来获取真正的errno的值。

### **3.线程的基本函数**

大多数pthread\_XXX系列的函数在失败时，并未遵循UNIX函数的惯例返回-1，这种情况在UNIX函数中属于一少部分。如果调用成功，则返回值是0，如果失败则返回错误代码。

#### 1.线程创建

#include <pthread.h>

int pthread\_create(pthread\_t \*thread, pthread\_attr\_t \*attr, void \*(\*start\_routine)(void \*), void \*arg);

参数说明：

thread：指向pthread\_create类型的指针，用于引用新创建的线程。

attr：用于设置线程的属性，一般不需要特殊的属性，所以可以简单地设置为NULL。

\*(\*start\_routine)(void \*)：传递新线程所要执行的函数地址。

arg：新线程所要执行的函数的参数。

调用如果成功，则返回值是0，如果失败则返回错误代码。

#### 2.线程终止

#include <pthread.h>

void pthread\_exit(void \*retval);

参数说明：

retval：返回指针，指向线程向要返回的某个对象。

线程通过调用pthread\_exit函数终止执行，并返回一个指向某对象的指针。注意：绝不能用它返回一个指向局部变量的指针，因为线程调用该函数后，这个局部变量就不存在了，这将引起严重的程序漏洞。

#### 3.线程同步

#include <pthread.h>

int pthread\_join(pthread\_t th, void \*\*thread\_return);

参数说明：

th：将要等待的线程，线程通过pthread\_create返回的标识符来指定。

thread\_return：一个指针，指向另一个指针，而后者指向线程的返回值。

一个简单的多线程Demo（thread1.c）：

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/monkey_d_meng/article/details/5628663)

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void \*thread\_function(void \*arg);

char message[] = "Hello World";

int main()

{

    int res;

    pthread\_t a\_thread;

    void \*thread\_result;

    res = pthread\_create(&a\_thread, NULL, thread\_function, (void \*)message);

    if (res != 0)

    {

        perror("Thread creation failed!");

        exit(EXIT\_FAILURE);

    }

    printf("Waiting for thread to finish.../n");

    res = pthread\_join(a\_thread, &thread\_result);

    if (res != 0)

    {

        perror("Thread join failed!/n");

        exit(EXIT\_FAILURE);

    }

    printf("Thread joined, it returned %s/n", (char \*)thread\_result);

    printf("Message is now %s/n", message);

    exit(EXIT\_FAILURE);

}

void \*thread\_function(void \*arg)

{

    printf("thread\_function is running. Argument was %s/n", (char \*)arg);

    sleep(3);

    strcpy(message, "Bye!");

    pthread\_exit("Thank you for your CPU time!");

}

编译这个程序时，需要定义宏\_REENTRANT：

gcc -D\_REENTRANT thread1.c -o thread1 –lpthread

运行这个程序：

$ ./thread1输出：

thread\_function is running. Argument was Hello World

Waiting for thread to finish...

Thread joined, it returned Thank you for your CPU time!

Message is now Bye!

这个例子值得我们去花时间理解，因为它将作为几个例子的基础。

pthread\_exit(void \*retval)本身返回的就是指向某个对象的指针，因此，pthread\_join(pthread\_t th, void \*\*thread\_return);中的thread\_return是二级指针，指向线程返回值的指针。

可以看到，我们创建的新线程修改的数组message的值，而原先的线程也可以访问该数组。如果我们调用的是fork而不是pthread\_create，就不会有这样的效果了。原因是fork创建子进程之后，子进程会拷贝父进程，两者分离，相互不干扰，而线程之间则是共享进程的相关资源。

#### **4.线程的同时执行**

接下来，我们来编写一个程序，以验证两个线程的执行是同时进行的。当然，如果是在一个单处理器系统上，线程的同时执行就需要靠CPU在线程之间的快速切换来实现了。

我们的程序需要利用一个原理：即除了局部变量外，所有其他的变量在一个进程中的所有线程之间是共享的。

在这个程序中，我们是在两个线程之间使用轮询技术，这种方式称为忙等待，所以它的效率会很低。在本文的后续部分，我们将介绍一种更好的解决办法。

下面的代码中，两个线程会不断的轮询判断flag的值是否满足各自的要求。

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/monkey_d_meng/article/details/5628663)

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

int flag = 1;

void \*thread\_function(void \*arg);

int main()

{

    int res;

    pthread\_t a\_thread;

    void \*thread\_result;

    int count = 1;

    res = pthread\_create(&a\_thread, NULL, thread\_function, NULL);

    if (res != 0)

    {

        perror("Thread creation failed");

        exit(EXIT\_FAILURE);

    }

    while (count++ <= 20)

    {

        if (flag == 1)

        {

            printf ("1");

            flag = 2;

        }

        else

        {

            sleep(1);

        }

    }

    printf("/nWaiting for thread to finish.../n");

    res = pthread\_join(a\_thread, &thread\_result);

    if (res != 0)

    {

        perror("Thread join failed");

        exit(EXIT\_FAILURE);

    }

    exit(EXIT\_SUCCESS);

}

void \*thread\_function(void \*arg)

{

    int count = 1;

    while (count++ <= 20)

    {

        if (flag == 2)

        {

            printf("2");

            flag = 1;

        }

        else

        {

            sleep(1);

        }

    }

}

编译这个程序：

gcc -D\_REENTRANT thread2.c -o thread2 –lpthread

运行这个程序：

$ ./thread2

121212121212121212

Waiting for thread to finish...

### **5.线程的同步**

在上述示例中，我们采用轮询的方式在两个线程之间不停地切换是非常笨拙且没有效率的实现方式，幸运的是，专门有一级设计好的函数为我们提供更好的控制线程执行和访问代码临界区的方法。

本小节将介绍两个线程同步的基本方法：信号量和互斥量。这两种方法很相似，事实上，它们可以互相通过对方来实现。但在实际的应用中，对于一些情况，可能使用信号量或互斥量中的一个更符合问题的语义，并且效果更好。

##### **5.1用信号量进行同步**

###### 1.信号量创建

#include <semaphore.h>

int sem\_init(sem\_t \*sem, int pshared, unsigned int value);

参数说明：

sem：信号量对象。

pshared：控制信号量的类型，0表示这个信号量是当前进程的局部信号量，否则，这个信号量就可以在多个进程之间共享。

value：信号量的初始值。

###### 2.信号量控制

#include <semaphore.h>

int sem\_wait(sem\_t \*sem);

int sem\_post(sem\_t \*sem);

sem\_post的作用是以**原子操作**的方式给信号量的值加1。  
 sem\_wait的作用是以**原子操作**的方式给信号量的值减1，但它会等到信号量非0时才会开始减法操作。如果对值为0的信号量调用sem\_wait，这个函数就会等待，直到有线程增加了该信号量的值使其不再为0。

###### 3.信号量销毁

#include <semaphore.h>

int sem\_destory(sem\_t \*sem);

这个函数的作用是，用完信号量后对它进行清理，清理该信号量所拥有的资源。如果你试图清理的信号量正被一些线程等待，就会收到一个错误。

与大多数Linux函数一样，这些函数在成功时都返回0。

下面编码实现输入字符串，统计每行的字符个数，以“end”结束输入：

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/monkey_d_meng/article/details/5628663)

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <semaphore.h>

#define SIZE 1024

void \*thread\_function(void \*arg);

char buffer[SIZE];

sem\_t sem;

int main()

{

    int res;

    pthread\_t a\_thread;

    void \*thread\_result;

    res = sem\_init(&sem, 0, 0);

    if (res != 0)

    {

        perror("Sem init failed");

        exit(EXIT\_FAILURE);

    }

    res = pthread\_create(&a\_thread, NULL, thread\_function, NULL);

    if (res != 0)

    {

        perror("Thread create failed");

        exit(EXIT\_FAILURE);

    }

    printf("Input some text. Enter 'end' to finish/n");

    while (scanf("%s", buffer))

    {

        sem\_post(&sem);

        if (strncmp("end", buffer, 3) == 0)

            break;

    }

    printf ("/nWaiting for thread to finish.../n");

    res = pthread\_join(a\_thread, &thread\_result);

    if (res != 0)

    {

        perror("Thread join failed");

        exit(EXIT\_FAILURE);

    }

    printf ("Thread join/n");

    sem\_destroy(&sem);

    exit(EXIT\_SUCCESS);

}

void \*thread\_function(void \*arg)

{

    sem\_wait(&sem);

    while (strncmp("end", buffer, 3) != 0)

    {

        printf("You input %d characters/n", strlen(buffer));

        sem\_wait(&sem);

    }

    pthread\_exit(NULL);

}

编译这个程序：

gcc -D\_REENTRANT thread2.c -o thread2 –lpthread

运行这个程序：

$ ./thread3

Input some text. Enter 'end' to finish

123

You input 3 characters

1234

You input 4 characters

12345

You input 5 characters

end

Waiting for thread to finish…

Thread join

通过使用信号量，我们阻塞了统计字符个数的线程，这个程序似乎对快速的文本输入和悠闲的暂停都很适用，比之前的轮询解决方案效率上有了本质的提高。

##### **5.2用互斥量进行线程同步**

另一种用在多线程程序中同步访问的方法是使用互斥量。它允许程序员锁住某个对象，使得每次只能有一个线程访问它。为了控制对关键代码的访问，必须在进入这段代码之前锁住一个互斥量，然后在完成操作之后解锁它。

用于互斥量的基本函数和用于信号量的函数非常相似：

#include <pthread.h>

int pthread\_mutex\_init(pthread\_mutex\_t \*mutex, const pthread\_mutexattr\_t, \*mutexattr);

int pthread\_mutex\_lock(pthread\_mutex\_t \*mutex);

int pthread\_mutex\_unlock(pthread\_mutex\_t \*mutex);

int pthread\_mutex\_destory(pthread\_mutex\_t \*mutex);

与其他函数一样，成功时返回0，失败时将返回错误代码，但这些函数并不设置errno，所以必须对函数的返回代码进行检查。互斥量的属性设置这里不讨论，因此设置成NULL。

我们用互斥量来重写刚才的代码如下：

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/monkey_d_meng/article/details/5628663)

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <semaphore.h>

#define SIZE 1024

char buffer[SIZE];

void \*thread\_function(void \*arg);

pthread\_mutex\_t mutex;

int main()

{

    int res;

    pthread\_t a\_thread;

    void \*thread\_result;

    res = pthread\_mutex\_init(&mutex, NULL);

    if (res != 0)

    {

        perror("Mutex init failed!");

        exit(EXIT\_FAILURE);

    }

    res = pthread\_create(&a\_thread, NULL, thread\_function, NULL);

    if (res != 0)

    {

        perror("Thread create failed!");

        exit(EXIT\_FAILURE);

    }

    printf("Input some text. Enter 'end' to finish/n");

    while (1)

    {

        pthread\_mutex\_lock(&mutex);

        scanf("%s", buffer);

        pthread\_mutex\_unlock(&mutex);

        if (strncmp("end", buffer, 3) == 0)

            break;

        sleep(1);

    }

    res = pthread\_join(a\_thread, &thread\_result);

    if (res != 0)

    {

        perror("Thread join failed!");

        exit(EXIT\_FAILURE);

    }

    printf("Thread joined/n");

    pthread\_mutex\_destroy(&mutex);

    exit(EXIT\_SUCCESS);

}

1. void \*thread\_function(void \*arg)
2. {
3. sleep(1);
5. while (1)
6. {
7. pthread\_mutex\_lock(&mutex);
8. printf("You input %d characters/n", strlen(buffer));
9. pthread\_mutex\_unlock(&mutex);
10. if (strncmp("end", buffer, 3) == 0)
11. break;
12. sleep(1);
13. }

编译这个程序：

gcc –D \_REENTRANT thread4.c -o thread4 –lpthread

运行这个程序：

$ ./thread4

Input some text. Enter 'end' to finish

123

You input 3 characters

1234

You input 4 characters

12345

You input 5 characters

end

You input 3 characters

Thread joined

#### **6.线程的属性**

之前我们并未谈及到线程的属性，可以控制的线程属性是非常多的，这里面只列举一些常用的。

如在前面的示例中，我们都使用的pthread\_join同步线程，但其实有些情况下，我们并不需要。如：主线程为服务线程，而第二个线程为数据备份线程，备份工作完成之后，第二个线程可以直接终止了，它没有必要再返回到主线程中。因此，我们可以创建一个“脱离线程”。

下面介绍几个常用的函数：

(1)int pthread\_attr\_init (pthread\_attr\_t\* attr);

功能：对线程属性变量的初始化。

attr：线程属性。

函数返回值：成功：0，失败：-1

(2) int pthread\_attr\_setscope (pthread\_attr\_t\* attr, int scope);

功能：设置线程绑定属性。

attr：线程属性。

scope：PTHREAD\_SCOPE\_SYSTEM(绑定)；PTHREAD\_SCOPE\_PROCESS(非绑定)

函数返回值：成功：0，失败：-1

(3) int pthread\_attr\_setdetachstate (pthread\_attr\_t\* attr, int detachstate);

功能：设置线程分离属性。

attr：线程属性。

detachstate：PTHREAD\_CREATE\_DETACHED(分离)；PTHREAD\_CREATE\_JOINABLE（非分离）

函数返回值：成功：0，失败：-1

(4) int pthread\_attr\_setschedpolicy(pthread\_attr\_t\* attr, int policy);

功能：设置创建线程的调度策略。

attr：线程属性；

policy：线程调度策略：SCHED\_FIFO、SCHED\_RR和SCHED\_OTHER。

函数返回值：成功：0，失败：-1

(5) int pthread\_attr\_setschedparam (pthread\_attr\_t\* attr, struct sched\_param\* param);

功能：设置线程优先级。

attr：线程属性。

param：线程优先级。

函数返回值：成功：0，失败：-1

(6) int pthread\_attr\_destroy (pthread\_attr\_t\* attr);

功能：对线程属性变量的销毁。

attr：线程属性。

函数返回值：成功：0，失败：-1

(7)其他

int pthread\_attr\_setguardsize(pthread\_attr\_t\* attr,size\_t guardsize);//设置新创建线程栈的保护区大小。

int pthread\_attr\_setinheritsched(pthread\_attr\_t\* attr, int inheritsched);//决定怎样设置新创建线程的调度属性。

int pthread\_attr\_setstack(pthread\_attr\_t\* attr, void\* stackader,size\_t stacksize);//两者共同决定了线程栈的基地址以及堆栈的最小尺寸（以字节为单位）。

int pthread\_attr\_setstackaddr(pthread\_attr\_t\* attr, void\* stackader);//决定了新创建线程的栈的基地址。

int pthread\_attr\_setstacksize(pthread\_attr\_t\* attr, size\_t stacksize);//决定了新创建线程的栈的最小尺寸（以字节为单位）。

例：创建优先级为10的线程。

pthread\_attr\_t attr;

struct sched\_param param;

pthread\_attr\_init(&attr);

pthread\_attr\_setscope (&attr, PTHREAD\_SCOPE\_SYSTEM); //绑定

pthread\_attr\_setdetachstate (&attr, PTHREAD\_CREATE\_DETACHED); //分离

pthread\_attr\_setschedpolicy(&attr, SCHED\_RR);

param.sched\_priority = 10;

pthread\_attr\_setschedparam(&attr, &param);

pthread\_create(xxx, &attr, xxx, xxx);

pthread\_attr\_destroy(&attr);

下面实现一个脱离线程的程序，创建一个线程，其属性设置为脱离状态。子线程结束时，要使用pthread\_exit，原来的主线程不再等待与子线程重新合并。代码如下：

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/monkey_d_meng/article/details/5628663)

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <pthread.h>

void \*thread\_function(void \*arg);

char message[] = "Hello World";

int thread\_finished = 0;

int main()

{

    int res;

    pthread\_t a\_thread;

    pthread\_attr\_t thread\_attr;

    res = pthread\_attr\_init(&thread\_attr);

    if (res != 0)

    {

        perror("Attribute creation failed");

        exit(EXIT\_FAILURE);

    }

    res = pthread\_attr\_setdetachstate(&thread\_attr, PTHREAD\_CREATE\_DETACHED);

    if (res != 0)

    {

        perror("Setting detached attribute failed");

        exit(EXIT\_FAILURE);

    }

    res = pthread\_create(&a\_thread, &thread\_attr, thread\_function, (void\*)message);

    if (res != 0)

    {

        perror("Thread creation failed");

        exit(EXIT\_FAILURE);

    }

    pthread\_attr\_destroy(&thread\_attr);

    while(!thread\_finished)

    {

        printf("Waiting for thread to say it's finished.../n");

        sleep(1);

    }

    printf("Other thread finished, bye!/n");

    exit(EXIT\_SUCCESS);

}

void \*thread\_function(void \*arg)

{

    printf("thread\_function is running. Argument was %s/n", (char \*)arg);

    sleep(4);

    printf("Second thread setting finished flag, and exiting now/n");

    thread\_finished = 1;

    pthread\_exit(NULL);

}

编译这个程序：

gcc -D\_REENTRANT thread5.c -o thread5 –lpthread

运行这个程序：

$ ./thread5

thread\_function is running. Argument: hello world!

Waiting for thread to finished...

Waiting for thread to finished...

Waiting for thread to finished...

Waiting for thread to finished...

Second thread setting finished flag, and exiting now

Other thread finished!

通过设置线程的属性，我们还可以控制线程的调试，其方式与设置脱离状态是一样的。

编译这个程序：

gcc -D\_REENTRANT thread5.c -o thread5 –lpthread

运行这个程序：

$ ./thread5

thread\_function is running. Argument: hello world!

Waiting for thread to finished...

Waiting for thread to finished...

Waiting for thread to finished...

Waiting for thread to finished...

Second thread setting finished flag, and exiting now

Other thread finished!

#### 7.取消一个线程

有时，我们想让一个线程可以要求另一个线程终止，线程有方法做到这一点，与信号处理一样，线程可以在被要求终止时改变其行为。

先来看用于请求一个线程终止的函数：

#include <pthread.h>

int pthread\_cancel(pthread\_t thread);

这个函数简单易懂，提供一个线程标识符，我们就可以发送请求来取消它。

线程可以用pthread\_setcancelstate设置自己的取消状态。

#include <pthread.h>

int pthread\_setcancelstate(int state, int \*oldstate);

参数说明：

state：可以是PTHREAD\_CANCEL\_ENABLE允许线程接收取消请求，也可以是PTHREAD\_CANCEL\_DISABLE忽略取消请求。

oldstate：获取先前的取消状态。如果对它没兴趣，可以简单地设置为NULL。如果取消请求被接受了，线程可以进入第二个控制层次，用pthread\_setcanceltype设置取消类型。

#include <pthread.h>

int pthread\_setcanceltype(int type, int \*oldtype);

参数说明：

type：可以取PTHREAD\_CANCEL\_ASYNCHRONOUS，它将使得在接收到取消请求后立即采取行动；另一个是PTHREAD\_CANCEL\_DEFERRED，它将使得在接收到取消请求后，一直等待直到线程执行了下述函数之一后才采取行动：pthread\_join、pthread\_cond\_wait、pthread\_cond\_timedwait、pthread\_testcancel、sem\_wait或sigwait。

oldtype：允许保存先前的状态，如果不想知道先前的状态，可以传递NULL。

默认情况下，线程在启动时的取消状态为PTHREAD\_CANCEL\_ENABLE，取消类型是PTHREAD\_CANCEL\_DEFERRED。

下面编写代码thread6.c，主线程向它创建的线程发送一个取消请求。

[cpp] view plaincopyprint?

1. #include <pthread.h>

2. #include <stdio.h>

3. #include <stdlib.h>

4.

5. void \*thread\_function(void \*arg);

6.

7. int main()

8. {

9. int res;

10. pthread\_t a\_thread;

11. void \*thread\_result;

12.

13. res = pthread\_create(&a\_thread, NULL, thread\_function, NULL);

14. if (res != 0)

15. {

16. perror("Thread create failed!");

17. exit(EXIT\_FAILURE);

18. }

19.

20. sleep(4);

21. printf("Canceling thread.../n");

22.

23. res = pthread\_cancel(a\_thread);

24. if (res != 0)

25. {

26. perror("Thread cancel failed!");

27. exit(EXIT\_FAILURE);

28. }

29.

30. printf ("Waiting for thread to finished.../n");

31.

32. res = pthread\_join(a\_thread, &thread\_result);

33. if (res != 0)

34. {

35. perror ("Thread join failed!");

36. exit(EXIT\_FAILURE);

37. }

38.

39. printf("Thread canceled!");

40.

41. exit(EXIT\_FAILURE);

42. }

43.

44. void \*thread\_function(void \*arg)

45. {

46. int i;

47. int res;

48.

49. res = pthread\_setcancelstate(PTHREAD\_CANCEL\_ENABLE, NULL);

50. if (res != 0)

51. {

52. perror("Thread setcancelstate failed!");

53. exit(EXIT\_FAILURE);

54. }

55.

56. res = pthread\_setcanceltype(PTHREAD\_CANCEL\_DEFERRED, NULL);

57. if (res != 0)

58. {

59. perror("Thread setcanceltype failed!");

60. exit(EXIT\_FAILURE);

61. }

62.

63. printf("thread\_function is running.../n");

64.

65. for (i = 0; i < 10; i++)

66. {

67. printf("Thread is still running (%d).../n", i);

68. sleep(1);

69. }

70. pthread\_exit(0);

71. }

编译这个程序：

gcc -D\_REENTRANT thread6.c -o thread6 –lpthread

运行这个程序：

$ ./thread6

thread\_function is running...

Thread is still running (0)...

Thread is still running (1)...

Thread is still running (2)...

Thread is still running (3)...

Canceling thread...

Waiting for thread to finished...

#### 8.多线程

之前，我们所编写的代码里面都仅仅是创建了一个线程，现在我们来演示一下如何创建一个多线程的程序。

[cpp] view plaincopyprint?

1. #include <pthread.h>

2. #include <stdio.h>

3. #include <stdlib.h>

4.

5. #define NUM 6

6.

7. void \*thread\_function(void \*arg);

8.

9. int main()

10. {

11. int res;

12. pthread\_t a\_thread[NUM];

13. void \*thread\_result;

14. int index;

15.

16. for (index = 0; index < NUM; ++index) {

17. res = pthread\_create(&a\_thread[index], NULL, thread\_function, (void \*)index);

18. if (res != 0)

19. {

20. perror("Thread create failed!");

21. exit(EXIT\_FAILURE);

22. }

23. sleep(1);

24. }

25.

26. printf("Waiting for threads to finished.../n");

27.

28. for (index = NUM - 1; index >= 0; --index)

29. {

30. res = pthread\_join(a\_thread[index], &thread\_result);

31. if (res == 0)

32. {

33. printf("Picked up a thread:%d/n", index + 1);

34. }

35. else

36. {

37. perror("pthread\_join failed/n");

38. }

39. }

40.

41. printf("All done/n");

42.

43. exit(EXIT\_SUCCESS);

44. }

45.

46. void \*thread\_function(void \*arg)

47. {

48. int my\_number = (int)arg;

49. int rand\_num;

50.

51. printf("thread\_function is running. Argument was %d/n", my\_number);

52. rand\_num = 1 + (int)(9.0 \* rand()/(RAND\_MAX + 1.0));

53. sleep(rand\_num);

54. printf("Bye from %d/n", my\_number);

55. pthread\_exit(NULL);

56. }

编译这个程序：

gcc -D\_REENTRANT thread7.c -o thread7 –lpthread

运行这个程序：

$ ./thread7

thread\_function is running. Argument was 0

thread\_function is running. Argument was 1

thread\_function is running. Argument was 2

thread\_function is running. Argument was 3

thread\_function is running. Argument was 4

Bye from 1

thread\_function is running. Argument was 5

Waiting for threads to finished...

Bye from 5

Picked up a thread:6

Bye from 0

Bye from 2

Bye from 3

Bye from 4

Picked up a thread:5

Picked up a thread:4

Picked up a thread:3

Picked up a thread:2

Picked up a thread:1

All done

## 创建分离线程

### 1、定义线程id和线程属性

pthread\_t tid

pthread\_attr\_t atrrDetach

### 2、初始化线程属性

If ( 0 != pthread\_init\_attr(&atrrDetach))

{

异常处理

}

### 3、设置线程的分离状态为分离

If ( 0 !=pthread\_attr\_setdetachstate(&atrrDetach，PTHREAD\_CREATE\_DETACHED))

{

异常处理

}

### 4、创建分离线程

If ( -1 == pthread\_create(&tid, &atrrDetach,func,NULL))

{

异常处理

}

func为处理函数名，函数的类型如下：

void \* func(void \*);

### 5、释放线程内存空间

If ( 0 != pthread\_attr\_destroy(&atrrDetach))

{

异常处理

}

### 编译的时候加上lpthread库

## 线程创建模板

typedef void \* (\* ThreadProc) (void \*p)

typedef struct PROCS\_THRD\_PRMTR

{

ThreadProc pProc;

void \* pPrmtr;

pthread\_attr\_t atrrPrmtr;

pthread\_t tThreadId;

} PROCS\_THRD\_PRMTR;

int CreatePrcs(PROC\_THRD\_PRMTR \*pPrcsThrdPrmtr)

{

int iRet = 0;

iRet = pthread\_attr\_init(&(pPrcsThrdPrmtr-> atrrPrmtr));

if (iRet != 0)

{

}

iRet=pthread\_attr\_setdetachstate(&(pPrcsThrdPrmtr->atrrPrmtr),PTHREAD\_CREATE\_DETACHED);

if (iRet != 0)

{

}

#ifndef \_REDHAT\_LINUX\_

iRet = pthread\_attr\_setscope(&(pPrcsThrdPrmtr->atrrPrmtr),PPHREAD\_SCOPE\_PROCESS);

#else

iRet = pthread\_attr\_setscope(&(pPrcsThrdPrmtr->atrrPrmtr),PPHREAD\_SCOPE\_SYSTEM);

#endif

if (iRet != 0)

{

}

iRet = pthread\_create(&(pPrcsThrdPrmtr-> tThreadId),

&(pPrcsThrdPrmtr->atrrPrmtr),

pPrcsThrdPrmtr-> pProc,

pPrcsThrdPrmtr-> pPrmtr

);

if (iRet != 0)

{

}

iRet = pthread\_attr\_destory(&(pPrcsThrdPrmtr->atrrPrmtr));

if (iRet != 0)

{

}

return iRet;

}

## char \*p = “abc”与char a[]=“abc”的区别

“abc”是常量字符串，是存放到只读区域的，是不能通过指针p来修改常量字符串的即不能通过memcpy(p,”def”,3)、strcpy(p,”def”)、p[0] = ‘d’,这些操作编译时都会报错

char a [] = “abc”是定义了一个数组a，该数组被初始化成了a、b、c、‘\0’

数组a定义时就分配内存了，a是可读写的，因此可以通过memcpy(a,”def”,3)、strcpy(a,”def”)、a[0] = ‘d等操作’来修改数组a的内容。

## [一个误解: 单个服务器程序可承受最大连接数“理论”上是“65535”](http://www.cnblogs.com/tianzhiliang/archive/2011/06/13/2079564.html)

请注意，这里有两个词分别被我标记上了引号，一个是“理论”，一个是“65535”。强调“理论”这个词，是想特别明确误解者的意思：就是说，这个值是不可能被打破的，是铁板钉丁的。而65535这个数字的由来，很多人想当然地将它与port最大值联系起来。的确，TCP的端口数，最大值确实为65535。但是，这并不代表一个服务器可以接受的连接数就是这个值，很多人之所以把这两个概念搞混淆是因为对socket和port没有更深的认识和理解。我们先来回想一下服务器服务的先后过程：服务器创建监听socket- >与对外服务的端口号绑定->开始listen->客户端连接到服务器对应的port->服务器accept为新的客户端产生新的socket->基于这个新的socket与客户端交换数据。从以上流程来看，最大值为65535的“端口号”这个重要的东东，我们只用了一次，就是执行bind的时候！而以后创建的socket，说白了就是一个可以进行网络IO操作的HANDLE而已，它跟端口号的牵扯仅限bind以及作为客户端连接服务器的识别端口号的时候，一旦accept产生了socket，这个端口号，对服务器和新客户端的通信而言就不再有任何意义。而服务器可承载的连接数最大量，不就是能产生多少个客户端的socket吗？这个socket值即使与端口号无关，又何来65535的“理论”上限？我再一次地将“理论”二字用引号括起，是因为在有的操作系统中，默认的配置会将socket最大值设定为65535，但这个值是可以改的！端口号仅仅是门牌地址，不能因为鸟巢体育场能容纳10万名观众就搞10万个门牌号码吧，那电话簿要多厚啊!

## 一个建立socket接受数据的函数例子

typedef struct \_cswt\_channel\_inf

{

Int iSendFd;

Int iRecvFd;

Int istat;

char caDownTime[8+1];

time\_t tAbsoluteTime;

char caDownReason;

pthread\_mutex\_t channelLock;

pthread\_t tRecvThreadId

}T\_CswtChannelInf;

T\_CswtChannelInf gsCswtChannelInf;

long lRcvChkInterval = 5;

int giSndRetryCnt

pthread\_mutex\_init(&gsCswtChannelInf.channelLock);

void signalReconnect(const char \* downReason)

{

time\_t tInterval = 0;

time\_t tNow = 0;

tNow = time(NULL);

if (tNow == -1)

{

异常处理；

return;

}

/\*获取通信通道的互斥锁\*/

If ( pthread\_mutex\_trylock(&gsCswtChannelInf.channelLock) != 0 )

{

异常处理；

return;

}

tInterval = tNow – gsCswtChannelInf.tAbsoluteTime;

if ( tInterval < lRcvChkInterval )

{

pthread\_mutex\_unlock(&gsCswtChannelInf.channelLock);

}

/\*注册重建原因如通道信息中\*/

gsCswtChannelInf.istat = 0;

strcpy(gsCswtChannelInf.caDownReason, downReason);

strcpy(gsCswtChannelInf. caDownTime,basGetSysTime())

gsCswtChannelInf. tAbsoluteTime = tNow;

/\*释放通信通道互斥锁\*/

pthread\_mutex\_unlock(&gsCswtChannelInf.channelLock);

/\*发起重建链接\*/

SpawnConnectThread();

return ;

}

SpawnConnectThread()

{

pthread\_t tid

pthread\_attr\_t atrrDetach

/\*初始化线程属性\*/

If ( 0 != pthread\_init\_attr(&atrrDetach))

{

异常处理

}

If ( 0 !=pthread\_attr\_setdetachstate(&atrrDetach，PTHREAD\_CREATE\_DETACHED))

{

异常处理

}

If ( -1 == pthread\_create(&tid, &atrrDetach,DechConnect ,NULL))

{

异常处理

}

If ( 0 != pthread\_attr\_destroy(&atrrDetach))

{

异常处理

}

}

void \* DechConnect(void \*)

{

/\*获取通信通道的互斥锁\*/

If ( pthread\_mutex\_trylock(&gsCswtChannelInf.channelLock) != 0 )

{

异常处理；

return;

}

/\*清理通道将对应的套接字关闭\*/

CleanUpChannel();

/\*链接远程服务器\*/

gsCswtChannelInf. iRecvFd = Connect2Remote(caRemoteIp,caRemotePort);

if( gsCswtChannelInf. iRecvFd < 0 )

{

pthread\_mutex\_unlock(&gsCswtChannelInf.channelLock);

异常处理；

return;

}

gsCswtChannelInf. iSendFd = gsCswtChannelInf. iRecvFd;

/\*修改通道的状态为已链接\*/

gsCswtChannelInf.iStat = 1;

/\*重置重连次数全局变量\*/

giSndRetryCnt = 0;

/\*设置空闲报文发送闹钟\*/

SetUpAlarm();

/\*打印通道信息\*/

PrintChannel();

pthread\_mutex\_unlock(&gsCswtChannelInf.channelLock);

}

int Connect2Remote(const char \* remoteIp, const char \* remotePort)

{

Int commFd = -1;

Int on;

struct sockaddr\_in server;

commFd = socket(AF\_INET,SOCK\_STREAM,0);

on = 1;

setsockopt(commFd,SOL\_SOCKET,SO\_REUSEADDR,&on,sizeof(on)):

setsockopt(commFd,SOL\_SOCKET,SO\_KEEPALIVE,&on,sizeof(on)):

server.sin\_family = AF\_INET;

server.sin\_addr.s\_addr = inet\_addr(remoteIp);

server.sin\_port = htons((short)atoi(remotePort));

if (connect(commFd,(struct sockaddr \*) server,sizeof(server)) != -1 )

{

return commFd;

}

else

{

close(commFd);

return -1;

}

}

void CleanUpChannel(void )

{

struct timespec tDelayTime;

tDelayTime.tv\_sec = 3;

tDelayTime.tv\_nsec = 0;

/\*关闭链接\*/

If (gsCswtChannelInf.iSendFd >0)

{

shutdown(gsCswtChannelInf.iSendFd);

close(gsCswtChannelInf.iSendFd);

gsCswtChannelInf.iSendFd = -1;

}

If (gsCswtChannelInf.iRecvFd >0)

{

shutdown(gsCswtChannelInf. iRecvFd);

close(gsCswtChannelInf. iRecvFd);

gsCswtChannelInf. iRecvFd = -1;

}

pthread\_delay\_np(&tDelayTime);

}

void SetUpAlerm(void )

{

signal(SIGALRM,SendCheck);

alerm(iSendInerval);

}

/\*发送空闲测试报文\*/

void SendCheck(int sig)

{ /\*空闲测试报文\*/

If( write(gsCswtChannelInf.iSendFd,”0000”,4) <= 0 )

{ /\*如果发送次数超限，则进程退出\*/

If( giSndRetryCnt >= iSndTetryTimes )

{

exit(-1);

}

giSndRetryCnt++;

signalReconnect();

}

giSndRetryCnt = 0;

SetUpAlerm();

}

## Socket选项设置

在利用socket技术编写通信程序时,为了保证通信双方可以正常通信，需要设置相关的选项，其中有的选项是协议相关的。在不同的系统中，socket选项有不同的默认值。这里只规定几个主要的socket选项设置，其他选项均使用系统默认值。

a) 保持socket的“LINGER”选项为缺省状态,即“关闭”状态。这个选项影响到使用TCP协议的socket关闭操作的行为。设置该选项为“关闭”状态，使socket关闭操作保持默认行为，即close()函数调用立即返回，如果socket发送缓冲区中还有数据，则系统会发送这些数据。

b) 设置socket的“REUSEADDR”为“打开”状态。设置这个选项可以保证socket监听进程在异常退出并重新启动后，仍可以成功绑定到原监听端口。该选项主要用在监听socket连接请求的服务器端。

c) 设置socket的“KeepAlive”为“打开”状态，设置这个选项可以保证在socket连接没有流量时，自动开始发送KeepAlive侦测包，侦测socket是否已经断开。

## socket通信函数

int redis\_writeo(int fd,void \*ptr,size\_t nbytes,int msecs )

{

If ( msecs == 0 )

{

return redis\_writeo\_wait(fd, ptr, nbytes);

}

else if( msecs > 0 )

{

redis\_writeo\_timeo(fd, ptr, nbytes, msec);

}

else

{

return -1;

}

}

int redis\_writeo\_wait(int fd, const void \* vptr, size\_t n )

{

size\_t nleft;

ssize\_t nwritten;

const char \*ptr;

ptr = vptr;

nleft = n;

while (nleft > 0)

{

If ( (nwritten = write(fd, ptr, nleft)) <= 0 )

{

If(errno == EINTR)

{

nwritten = 0;

}

else

{

retrun -2

}

}

nleft-= nwritten;

ptr+= nwritten;

}

return 0;

}

int redis\_writeo\_timeo(int fd, const void \* vptr, size\_t n, int msec)

{

size\_t nleft;

ssize\_t nwritten;

const char \* ptr;

int secleft;

int flags;

ptr = vptr;

nleft = n;

secleft = msec;;

flags = fcntl(fd, F\_GETFL ,0);

fcntl(fd, F\_SETFL, flags|O\_NONBLOCK);

while ( (nleft > 0) && (secleft >0 ) )

{

If ( redis\_is\_writeable(fd,1000) )

{

If((nwritten = write(fd, ptr, nleft)) <= 0)

{

If(errno == EINTR)

{

nwritten = 0;

}

else

{

retrun -2

}

}

nleft-= nwritten;

ptr+= nwritten;

}

Secleft-=1000;

}

fcntl(fd, flags, 0);

/\*超时\*/

if ( (secleft <=0) && (nleft >0) )

{

return -4;

}

return 0;

}

int redis\_is\_readable(int fd, int ms)

{

int ret;

fd\_set fs;

struct timeval tv;

FD\_ZERO(&fs);

FD\_SET(&fs)

if ( ms == 0 )

{

ret = select(fd+1, &fs, NULL, NULL, NULL );

}

else

{

tv.tv\_sec = ms / 1000;

tv.tv\_usec = (ms % 1000) \* 1000;

ret = select(fd+1, &fs, NULL, NULL, &tv);

}

return ret;

}

Int redis\_is\_writeable(int fd, int ms)

{

int ret;

fd\_set fs;

struct timeval tv;

FD\_ZERO(&fs);

FD\_SET(&fs)

if ( ms == 0 )

{

ret = select(fd+1, NULL, &fs, NULL, NULL );

}

else

{

tv.tv\_sec = ms / 1000;

tv.tv\_usec = (ms % 1000) \* 1000;

ret = select(fd+1, NULL, &fs, NULL, &tv);

}

return ret;

}

## size命令

size 可执行程序

## nm工具

nm -x 可执行程序

## dump工具

## 蠕虫程序的攻击原理

蠕虫程序的攻击原理就是将数据转换成代码。

代码和数据的区别也可以认为是编译时和运行时的分界线。编译器的绝大部分工作都跟翻译代码有关，必要的数据存储管理的绝大部分在运行时进行。

## C语言常用工具

### 用于检查源代码的工具

|  |  |  |
| --- | --- | --- |
| 工具 | 位置 | 作用 |
| cb | 随编译器附带 | C程序美化器，在源文件中运行这个工具，可以使源文件有标准的布局和缩进格式。来自Berkeley |
| indent |  | 与cb作用相同。来自AT&T |
| cflow | 随编译器附带 | 打印程序中调用者与被调用者的关系 |
| cscope | 随编译器附带 | 一个基于ASCII的C程序的交互式浏览器。我们在操作系统小组中使用，用于检查头文件修改的效果 |
| ctags | /usr/bin` | 创建一个标签文件，共vi编辑器使用。标签文件能加快检查程序源文件的速度 |
| lint | 随编译器附带 | C程序检查器 |
| sccs | /usr/ccs/bin | 源代码版本控制系统 |
| vgrind | /usr/bin | 格式器，用于打印漂亮的c列表 |

### 检查可执行文件的工具

|  |  |  |
| --- | --- | --- |
| 工具 | 位置 | 作用 |
| dig | /usr/ccs/bin | 目标代码反编译工具 |
| dump -Lv | /usr/ccs/bin | 打印动态链接信息 |
| ldd | /usr/bin | 打印所需要的动态链接 |
| nm | /usr/ccs/bin | 打印目标文件的符号表 |
| strings | /usr/bin | 查看嵌入二进制文件中的字符串。用于查看二进制文件中可能产生的错误信息、内置文件名和符号名或版本或版权信息 |
| sum | /usr/bin | 打印文件的检验与程序块计数 |
|  |  |  |

### 帮助调试的工具

|  |  |  |
| --- | --- | --- |
| 工具 | 位置 | 作用 |
| truss | /usr/bin | 打印可执行文件所进行的系统调用，它可以查看二进制文件正在干什么，为什么阻塞或失败 |
| ps | /usr/bin | 显示进程的状态 |
| ctrace | 随编译器附带 | 修改你的源文件，文件执行时按行打印 |
| debugger | 随编译器附带 | 交互式调试器 |
| file | /usr/bin | 告诉你一个文件包含的内容 |

## 内存字节单位

|  |  |
| --- | --- |
| 单位 | 大小 |
| Kilo | 2的10次方 |
| Mega | 2的20次方 |
| Giga | 2的30次方 |
| Tera | 2的40次方 |
| Bubba | 2的50次方 |

## setjmp和longjmp函数

setjmp和longjmp是操作过程活动记录实现的，许多程序员并不知道这个强大的机制，因为它是C语言所特有的，它们部分弥补了C语言有限的转移能力。这两个函数协同工作如下所示：

* setjmp(jmp\_buf j)必须首先被调用. 它表示“使用变量j”记录现在的位置。函数返回0。
* longjmp(jmp\_buf j, int i)可以接着被调用。表示“回到j所记录的位置，让它看上去像是从原先的setjmp()函数返回一样。但是函数返回-1，使代码能够知道它实际上是通过longjmp()返回的”熬不拗口？
* 当使用longjmp（）时，j的内容会被销毁。

setjmp保存了一份程序计数器和当前的栈顶指针。如果喜欢，也可以保存一些初始值。longjmp()恢复这些值，有效的转移控制并把状态重置回保存状态的时候。这被称作“展开堆栈（unwinding stack）”，因为从堆栈中展开过程活动记录，直到取得保存在其中的值。尽管longjmp会导致转移，但它和goto又有不同，区别如下：

* longjmp和goto的区别

goto语句不能跳出C语言当前的函数（这也是longjmp取名由来，它可以跳的很远，甚至可以跳到其他文件的函数中）。

longjmp只能跳到曾经到过的地方。在执行setjmp的 地方任留有一个过程活动记录。从这个角度讲，longjmp更像是“从何处来（come from）”而不是“往哪里去（go to）”。Longjmp接受一个额外的整型参数幷返回它的值，这可以知道是由longjmp转移到这里的还是从上条语句执行后自然而来到这里的。

下面代码显示了setjmp()和longjmp()一例：

#include<setjmp.h>

jmp\_buf buf;

void banna(void)

{

printf(“in banna()\n”);

longjmp(buf,1);

/\*以下代码不会被执行\*/

printf(“you will never see this ,because longjmp”);

}

main()

{

If (setjmp(buf))

{

printf(“I am back in main\n”);

}

else

{

printf(“first time through \n”);

banna();

}

}

输出结果如下：

first time through

in banna()

I am back in main

需要注意的地方是：保证局部变量在longjmp过程中一直保持它的值的唯一可靠方法是把它声明为volatile(这适用于那些值在setjmp执行和longjmp返回之间会改变的变量)

setjmp/longjmp最大的用途是错误恢复。只要还没有从函数中返回，一旦发现一个不可恢复的错误，可以把控制转移到主输入循环，幷从那里重新开始。有些人使用setjmp/longjmp从一串无数的函数调用中立即返回。还有一些人用它们防范潜在的危险代码。setjmp和longjmp在C++中变异为更普通的异常处理机制”catch”和”throw”

像goto一样，setjmp和longjmp使得程序难以理解和调试。如果不是出于特殊需要，最好避免使用它们。

## 常见段错误

* 引用一个包含非法值的指针
* 引用一个空指针（常常由于系统程序中返回指针，幷为经过检查就是用）
* 在为得到正确的权限时进行访问。例如：试图往一个只读的文本段写入值，就会引起段错误
* 用完了堆栈或堆空间（虚拟内存虽然巨大但绝非无限）

以发生频率为序，最终可能导致段错误的常见编程错误是：

* 坏指针值错

在指针赋值之前就用它来引用内存，或者向库函数传送一个坏指针（不要上当！如果调试器显示系统程序中出现了段错误，幷不是因为系统程序引起了段错误，问题很可能还存在于自己的代码中）。第三种可能导致坏指针的原因是对指针进行释放之后，再访问它的内容。可修改free语句，在指针释放之后再将它置为空值

free（p），p = NULL；

这样，如果在指针释放之后继续使用该指针，至少程序能再终止之前进行信息转存。

* 改写（overwrite）错误：

越过数组边界写入数据，在动态分配的内存两端之外写入数据或改写一些堆管理数据结构（在动态分配的内存之前的区域写入数据就很容易发生这种情况）

* 指针释放引起的错误：

释放同一个内存块两次，或释放一块未曾使用malloc分配的内存，或释放仍在使用中的内存，或释放一个无效的指针。一个极为常见的与释放内存有关的错误就是在for(p=start;p;p=p->next)这样的循环中迭代一个链表，幷在循环体内使用free(p)语句。这样，在下一次循环迭代时，程序就会对已经释放的指针引用操作，从而导致不可预料的结果。

当程出现坏指针时，什么样的结果都有可能发生，一种广被接受的说法是，如果“你走运”，指针将指向你的地址空间之外，这样第一次使用该指针时就会是程序进行信息转存后终止。如果”你不走运”，指针将指向你的地址空间之内，幷损坏（改写）它所指向的内存的任何信息，这将引起隐晦的Bug，非常难以捕捉。

### 库函数调用和系统调用区别何在？

简明的回答是函数库调用是语言或应用程序的一部分，而系统调用是操作系统的一部分。

**库函数调用VS系统调用**

|  |  |
| --- | --- |
| 库函数调用 | 系统调用 |
| * 在所有ANSI C编译器版本中，C函数库是相同的 * 它调用库函数中的一个程序 * 与用户程序相联系 * 在用户地址空间执行 * 它运行的时间属于“用户”时间 * 属于过程调用，开销较小 * 典型的C函数库调用：system、fprintf，malloc | * 各个操作系统的系统调用是不同的 * 它调用系统内核服务 * 是操作系统的一个进入点 * 在内核地址空间执行 * 它的运行时间属于“内核”时间 * 需要在切换到内核上下文环境后切换回来，开销较大 * 在UNIX中大约90个系统调用chdir,fork,write,brk |

库行数通常比行内展开的代码慢，因为它需要付出函数调用的开销，但系统调用比库函数调用还要慢很多，因为它需要把上下文环境切到内核模式。纯粹从性能上考虑来说，应该尽可能地少用系统调用，但是，必须记住的是，许多C函数库中的程序是通过系统调用来实现的

### 文件描述符与文件中的不同

这个问题是前面一个问题的延续，所有操纵文件的UNIX程序或者使用文件指针或者使用文件描述符来标识他们正在操作的文件，事实上答案非常直截了当，它取决于你对UNIX I/O的熟悉程度以及对各种因素利弊的权衡。

所有操作文件的系统调用都接受一个文件描述符作为参数，或者把它作返回值返回。

在Sun的 编译器中，文件描述符是一个小整数（通常在0-255之间），用于索引开放文件的每个进程表，系统I/O调用有create(),open(),write(),close(),ioctl()等，他们不是ANSI C的一部分，不会存在于非UNIX环境。如果使用了他们，程序将失去可移植性。因此，建立一组标准I/O库调用是必要的 ，ANSI C现在规定所有的编译环境都必须支持他们。

为了确保可移植性，应该使用标准的I/O库调用，如fopen(),fclose(),putc(),fseek()等，它们中的绝大多数名字中带有一个“f”，这些系统调用都接受一个类型为指向FILE结构的指针的参数。FILE指针指向一个流结构，它在<stdio.h>中定义。结构中的内容根据不同的编译器有所不同，在UNIX中通常是开放文件的每个进程表的一个条目。在经典情况下，它包含了缓冲区、所有用于提示缓冲区中有多少字节是实际的文件数数据的变量以及提示流状态的标志（如ERROR和EOF）等。

* 所以，文件描述符就是开放文件进程表的一个偏移量。它用于UNIX系统调用中，用于标识文件。
* FILE指针保存了一个FILE结构的地址。FILE 结构用于开放的I/O流。它用于ANSI C标准I/O库调用中，用于标识文件。

## 经典面试题

### 如何能检测到链表中存在循环

首次，排除一种特殊情况，就是3个元素的链表中第2个元素的后面是第1个元素。设置两个指针p1和p2，p1指向第一个元素，p2指向第二个元素，看看他们是否相等，如果相等就属于上述这种特殊情况，如果不等，把p1后移一个元素，p2指向后移两个元素。检测两指针的值，如果相等，说明链表中存在循环。如果不相等，继续按照前述方法进行。尽管有可能要对这个链表经过几次遍历才能检测出来。

### C语言中不同的增值语句的区别何在

考虑下面四条语句：

x = x + 1;

++x;

x++;

x += 1

显然，这四条语句的功能是相等的，它们都是把x的值增加1。如果像现在这样不考虑前后的上下文环境，它们之间没有什么区别。应试者需要（显式的或隐式的）提供上下文环境，以便回答这个问题并找出这四条语句的区别。

### 库函数调用和系统调用区别何在？

简明的回答是函数库调用是语言或应用程序的一部分，而系统调用是操作系统的一部分。

**库函数调用VS系统调用**

|  |  |
| --- | --- |
| 库函数调用 | 系统调用 |
| * 在所有ANSI C编译器版本中，C函数库是相同的 * 它调用库函数中的一个程序 * 与用户程序相联系 * 在用户地址空间执行 * 它运行的时间属于“用户”时间 * 属于过程调用，开销较小 * 典型的C函数库调用：system、fprintf，malloc | * 各个操作系统的系统调用是不同的 * 它调用系统内核服务 * 是操作系统的一个进入点 * 在内核地址空间执行 * 它的运行时间属于“内核”时间 * 需要在切换到内核上下文环境后切换回来，开销较大 * 在UNIX中大约90个系统调用chdir,fork,write,brk |

库行数通常比行内展开的代码慢，因为它需要付出函数调用的开销，但系统调用比库函数调用还要慢很多，因为它需要把上下文环境切到内核模式。纯粹从性能上考虑来说，应该尽可能地少用系统调用，但是，必须记住的是，许多C函数库中的程序是通过系统调用来实现的

### 文件描述符与文件中的不同

这个问题是前面一个问题的延续，所有操纵文件的UNIX程序或者使用文件指针或者使用文件描述符来标识他们正在操作的文件，事实上答案非常直截了当，它取决于你对UNIX I/O的熟悉程度以及对各种因素利弊的权衡。

所有操作文件的系统调用都接受一个文件描述符作为参数，或者把它作返回值返回。

在Sun的 编译器中，文件描述符是一个小整数（通常在0-255之间），用于索引开放文件的每个进程表，系统I/O调用有create(),open(),write(),close(),ioctl()等，他们不是ANSI C的一部分，不会存在于非UNIX环境。如果使用了他们，程序将失去可移植性。因此，建立一组标准I/O库调用是必要的 ，ANSI C现在规定所有的编译环境都必须支持他们。

为了确保可移植性，应该使用标准的I/O库调用，如fopen(),fclose(),putc(),fseek()等，它们中的绝大多数名字中带有一个“f”，这些系统调用都接受一个类型为指向FILE结构的指针的参数。FILE指针指向一个流结构，它在<stdio.h>中定义。结构中的内容根据不同的编译器有所不同，在UNIX中通常是开放文件的每个进程表的一个条目。在经典情况下，它包含了缓冲区、所有用于提示缓冲区中有多少字节是实际的文件数数据的变量以及提示流状态的标志（如ERROR和EOF）等。

* 所以，文件描述符就是开放文件进程表的一个偏移量。它用于UNIX系统调用中，用于标识文件。
* FILE指针保存了一个FILE结构的地址。FILE 结构用于开放的I/O流。它用于ANSI C标准I/O库调用中，用于标识文件。

### 编写代码，确定一个变量是有符号数还是无符号数

要回答这个问题，你必须在特定的编译器中确定一个给定的类型是有符号数，还是无符号数，在ANSI C中，“char”既可以是有符号数，也可以是无符号数，这是编译器决定的。当你编写的需要移植到多个平台时，知道类型是不是有符号数就非常有用了，如果该类型在所有的编译器编译都是恒定的，那就在理想不过了。

你无法用函数，实现目的，函数形式参数的类型是在函数内部定义的，所以它无法穿越调用这一关。因此，你必须编写一个宏，根据参数的声明对它进行处理。

接下来就是区别宏的参数到底是一个类型还是一个类型的值，假定参数是一个值，无符号数的本质是它永远不会是负的，有符号数的本质特征是对最左边一个位取补将会改变它的符号

#define ISUNSIGNED(a) (a>=0 && ~a>=0)

#define ISUNSIGNED (type) ((type)0-1 > 0)

第二个从类型来判断，没有问题。

而第一个只能用在K&R C里，在ANSI C里就不行了。当这个宏被用在int/unsigned int时，没有任何问题。但是当使用在char和short上就会出错。

分析： 此宏在int/unsigned int好使的原因是宏中的a的精度始终不会发生变化，而当a是char和short时，表达式~a>=0会先将a变为int然后再取反（因为常量0的类型为int），而int当然是有符号的，对它取反就有可能小于0。

要修正这个错误其实很简单，那就是要保证先进行取反操作再做精度提升，我们可以采用对a取反a=~a（这样就不会有类型提升），判断结束后再取一次反，恢复a的值。

这样得到新的代码：

#define ISUNSIGNED(a) (a >=0 && (a=~a,a >=0 ? (a=~a,1):(a=~a,0)))

另一种方法是：把变量的最高位置1，然后判断变量是否大于0。如果大于0，就是无符号数；否则就是有符号数。

#defined ISUNSIGNED(a) (a|=(0x1<<(sizeof(a)\*8-1))>0?1:0)

### 打印一个二叉树的值的时间复杂度是多少？

这个问题是面试者在申请Intel编译器小组的职位时被问到的。现在，关于复杂度理论首先需要知道的是关于大O表示法。O(N)表示当N（通常是需要处理的对象数量）增长时，处理时间几乎是按线性增长的。类似，O(N2)，表示当N增长时，处理时间的增长要快得多，大致是按照N的平方增长的，关于时间复杂度理论你其次需要知道的是在一棵二叉树中，所有的操作的时间复杂度都是O(log(n)),所以很多程序员不假思索的作出了这个回答。**错误！**

这个问题有点类似于Dan Rather著名的“频率是什么”问题，这个问题用于干扰、混淆、和激怒对方而不是真的想对方咨询信息。要打印一棵二叉树的所有节点的值，你必须对它们诸葛访问，所以时间复杂度是O（N）

### execve()系统调用成功，它将返回什么？

execve()函数用参数中的可执行文件替换调用者进程的映像幷开始执行。所以，当execve系统调用成功执行后，并不会返回一个值。

### 从文件中随机提取一个字符

这是Microsoft喜欢使用的问题之一，主考官要求面试者编写一些代码，实现从一个文件（文件的内容是许多字符串）中随机提取一个字符串。解决这个问题经典的方法是读取文件，对字符串进行计数，并记录每个字符串的位移位置。然后，在1和字符串总数之间取一个随机数，根据选中字符串的偏移位置取出该字符串。

但是，主考官设置了一些条件，使这个问题的难度大大增加。它要求只能按顺序遍历文件一次，并且不能用表格来存储，对于这个问题，主考官的主要兴趣在于你如何解决问题的过程。如果你提问，他会给你一些提示，所以大多数面试者最终都能获得答案。主考官对你的满意程度取决于你获取答案的速度。

基本的技巧是在辛存的字符串中挑选，并在过程中不断更新。从计算的角度看这个方法是非常低效的，所以它很容易被忽略。你打开文件并保存第一个字符串，此时就有了一个备选字符串，幷有100%的可能性选中它。保存这个字符串，继续读入一个字符串，这样就有了两个备选字符串，选中每个的可能性都是50%。选中其中之一，然后丢弃另一个再读入下一个字符串，按照新字符串33%原先辛存的字符串67%的概率，在两者之间选择一个，然后保存新选中的字符串。根据这个方法，一次对整个文件进行处理，在其中的每一步，读入字符串N，在它和前一个辛存的字符串之间进行选择。当达到文件末尾的时候，最后一个辛存字符串就是从整个文件中随机提取的那个字符串。

这是一个非常难的问题，你要么依靠尽可能的提示获得答案，要么就预先做好充分准备，提前阅读本书。

## a.out目标文件格式

### 1执行头部分(exec header)

该部分含有一些参数（exec结构体）是有关目标文件的整体结构信息。例如：代码区和数据区的长度、未初始化数据区的长度、对应源文件名以及目标文件创建时间等。内核使用这些参数据把执行文件加载到内存中幷执行，而链接程序ld使用这些参数将一些模块文件组合成一个可执行文件。这是目标文件唯一必要的组成部分。

### 2代码段(text segment)

由编译器或汇编器生成的二进制指令代码和数据信息，含有程序执行时被加载到内存中的指令代码和数据。可以以只读形式被加载。

### 3数据段（data segment）

由编译器或汇编器生成的二进制指令代码和数据信息，这部分代码含有已经初始化过的数据，总是被加载到可读写内存中。

### 4代码重定位部分（text relocation）

含有供链接程序使用的记录数据。在组合目标模块文件时用于定位代码段中的指针和地址。当链接程序需要修改目标代码的地址时就需要修正和维护这些地方。

### 5数据重定位部分(data relocation)

类似于代码重定位段部分，但作用于数据段中指针的重定位。

### 6符号表（symbol table）

这部分同样含有供链接程序使用的记录数据。这些记录数据保存着模块文件中定义的全局符号以及需要从其他模块文件中输入的符号或者有链接器定义的符号，用于在模块文件之间对命名的变量和函数进行交叉引用

### 7字符串表(string table)

该部分含有与符号名相对应的字符串。用于调试程序调试目标代码，与连接过程无关。

### 8使用objdump -x 目标文件名

objdump -x 目标文件名

用来查看模块文件和执行文件中文件执行头结构的具体值。

### 9使用hexdump -h 目标文件名

hexdump -h 目标文件名

来查看模块文件和执行文件中文件执行头结构的具体值。

# socket通信

## setsockopt函数SO\_REUSEADDR选项

SO\_REUSEADDR可以用在以下四种情况下。   
    (摘自《Unix网络编程》卷一，即UNPv1)   
    1、当有一个有相同本地地址和端口的socket1处于TIME\_WAIT状态时，而你启  
动的程序的socket2要占用该地址和端口，你的程序就要用到该选项。   
    2、SO\_REUSEADDR允许同一port上启动同一服务器的多个实例(多个进程)。但  
每个实例绑定的IP地址是不能相同的。在有多块网卡或用IP Alias技术的机器可  
以[**测试**](http://lib.csdn.net/base/softwaretest)这种情况。   
3、SO\_REUSEADDR允许单个进程绑定相同的端口到多个socket上，但每个soc  
ket绑定的ip地址不同。这和2很相似，区别请看UNPv1。   
    4、SO\_REUSEADDR允许完全相同的地址和端口的重复绑定。但这只用于UDP的  
多播，不用于TCP。

## recv和send等在调用时会被信号给中断

read、write、recv、send等报文接受和发送函数调用时会被异步的信号给中断(interrupt)，因此我们必须对在这些函数在调用时因为信号而中断的情况进行处理，**即判断函数返回时 enno 是否等于 EINTR。**

## listen和accept的区别和作用

### ****listen函数****

**摘要:**listen函数使用主动连接套接口变为被连接套接口，使得一个进程可以接受其它进程的请求，从而成为一个服务器进程。在TCP服务器编程中listen函数把进程变为一个服务器，并指定相应的套接字变为被动连接。

listen函数在一般在调用bind之后-调用accept之前调用，它的函数原型是：

intlisten(int sockfd, int backlog)

**参数sockfd**

被listen函数作用的套接字，sockfd之前由socket函数返回。在被socket函数返回的套接字fd之时，它是一个主动连接的套接字，也就是此时系统假设用户会对这个套接字调用connect函数，期待它主动与其它进程连接，然后在服务器编程中，用户希望这个套接字可以接受外来的连接请求，也就是被动等待用户来连接。由于系统默认时认为一个套接字是主动连接的，所以需要通过某种方式来告诉系统，用户进程通过系统调用listen来完成这件事。

**参数backlog**

这个参数涉及到一些网络的细节。进程处理一个一个连接请求的时候，可能还存在其它的连接请求。因为TCP连接是一个过程，所以可能存在一种半连接的状态，有时由于同时尝试连接的用户过多，使得服务器进程无法快速地完成连接请求。如果这个情况出现了，服务器进程希望内核如何处理呢？内核会在自己的进程空间里维护一个队列以跟踪这些完成的连接但服务器进程还没有接手处理或正在进行的连接，这样的一个队列内核不可能让其任意大，所以必须有一个大小的上限。这个backlog告诉内核使用这个数值作为上限。

毫无疑问，服务器进程不能随便指定一个数值，内核有一个许可的范围。这个范围是实现相关的。很难有某种统一，一般这个值会小30以内。

### ****accept函数****

**摘要**:accept()用来接受参数s的socket连接，它的函数原型是：

intaccept(int s,struct sockaddr \* addr,int \* addrlen)

服务程序调用accept函数从处于监听状态的流套接字s的客户连接请求队列中取出排在最前的一个客户请求，并且创建一个新的套接字来与客户套接字创建连接通道，如果连接成功，就返回新创建的套接字的描述符，以后与客户套接字交换数据的是新创建的套接字；如果失败就返回 INVALID\_SOCKET。该函数的第一个参数指定处于监听状态的流套接字；[**操作系统**](http://lib.csdn.net/base/operatingsystem)利用第二个参数来返回新创建的套接字的地址结构；操作系统利用第三个参数来返回新创建的套接字的地址结构的长度。

listen只是把套接字从主动变为被动，并限制链接数；剩下的问题就是accept的，它会检测的；  
listen意思是监听：但是它不是一直在监听，accept才是；

理解:listen函数不会阻塞，它只是相当于把socket的属性更改为被动连接，可以接收其他进程的连接。listen侦听的过程并不是一直阻塞，直到有客户端请求连接才会返回，它只是设置好socket的属性之后就会返回。监听的过程实质由操作系统完成。但是accept会阻塞（也可以设置为非阻塞），如果listen的套接字对应的连接请求队列为空（没有客户端连接请求）,它会一直阻塞等待。

### select 函数详解

select()函数主要是建立在fd\_set类型的基础上的。fd\_set（它比较重要所以先介绍一下）是一组文件描述字(fd)的集合，它用一位来表示一个fd（下面会仔细介绍），对于fd\_set类型通过下面四个宏来操作：

fd\_set set;

FD\_ZERO(&set);       /\* 将set清零使集合中不含任何fd\*/

FD\_SET(fd, &set);    /\* 将fd加入set集合 \*/

FD\_CLR(fd, &set);    /\* 将fd从set集合中清除 \*/

FD\_ISSET(fd, &set);  /\* 测试fd是否在set集合中\*/

过去，一个fd\_set通常只能包含<32的fd（文件描述字），因为fd\_set其实只用了一个32位矢量来表示fd；现在,UNIX系统通常会在头文件中定义常量FD\_SETSIZE，它是数据类型fd\_set的描述字数量，其值通常是1024，这样就能表示<1024的fd。根据fd\_set的位矢量实现，我们可以重新理解操作fd\_set的四个宏：

fd\_set set;

FD\_ZERO(&set);

将set的所有位置0，如set在内存中占8位则将set置为00000000

FD\_SET(0, &set);

将set的第0位置1，如set原来是00000000，则现在变为10000000，这样fd==1的文件描述字就被加进set中了

FD\_CLR(4, &set);

将set的第4位置0，如set原来是10001000，则现在变为10000000，这样fd==4的文件描述字就被从set中清除了

FD\_ISSET(5, &set);

测试set的第5位是否为1，如果set原来是10000100，则返回非零，表明fd==5的文件描述字在set中；否则返回0。

select函数的接口比较简单：

int select(int nfds, fd\_set \*readset, fd\_set \*writeset,

fd\_set\* exceptset, struct timeval \*timeout);

功能：

测试指定的fd可读？可写？有异常条件待处理？

参数：

nfds

需要检查的文件描述字个数（即检查到fd\_set的第几位），数值应该比三组fd\_set中所含的最大fd值更大，一般设为三组fd\_set中所含的最大fd值加1（如在readset,writeset,exceptset中所含最大的fd为5，则nfds=6，因为fd是从0开始的）。设这个值是为提高效率，使函数不必检查fd\_set的所有1024位。

readset

     用来检查可读性的一组文件描述字。

writeset

     用来检查可写性的一组文件描述字。

exceptset

     用来检查是否有异常条件出现的文件描述字。(注：错误不包括在异常条件之内)

timeout

有三种可能：

1. timeout=NULL（阻塞：直到有一个fd位被置为1函数才返回）

2. timeout所指向的结构设为非零时间（等待固定时间：有一个fd位被置为1或者时间耗尽，函数均返回）

3.  timeout所指向的结构，时间设为0（非阻塞：函数检查完每个fd后立即返回）

返回值：

返回对应位仍然为1的fd的总数。

使用select函数的过程一般是：

先调用宏FD\_ZERO将指定的fd\_set清零，然后调用宏FD\_SET将需要测试的fd加入fd\_set，接着调用函数select测试fd\_set中的所有fd，最后用宏FD\_ISSET检查某个fd在函数select调用后，相应位是否仍然为1。

以下是一个测试单个文件描述字可读性的例子

//这段代码将指定测试Socket的描述字的可读可写性，因为Socket使用的也是fd

uint32 SocketWait(TSocket \*s,bool rd,bool wr,uint32 timems)

{

     fd\_set rfds,wfds;

#ifdef \_WIN32

     TIMEVAL tv;

#else

     struct timeval tv;

#endif   /\* \_WIN32 \*/

     FD\_ZERO(&rfds);

     FD\_ZERO(&wfds);

     if (rd)     //TRUE

          FD\_SET(\*s,&rfds);   //添加要测试的描述字

     if (wr)     //FALSE

          FD\_SET(\*s,&wfds);

     tv.tv\_sec=timems/1000;     //second

     tv.tv\_usec=timems%1000;     //ms

     for (;;) //如果errno==EINTR，反复测试缓冲区的可读性

          switch(select((\*s)+1,&rfds,&wfds,NULL,

          (timems==TIME\_INFINITE?NULL:&tv)))  //测试在规定的时间内套接口接收缓冲区中是否有数据可读

         {                                              //0－－超时，-1－－出错

         case 0:     /\* time out \*/

              return 0;

         case (-1):    /\* socket error \*/

              if (SocketError()==EINTR)

                   break;

              return 0; //有错但不是EINTR

          default:

              if (FD\_ISSET(\*s,&rfds)) //如果s是fds中的一员返回非0，否则返回0

                   return 1;

              if (FD\_ISSET(\*s,&wfds))

                   return 2;

              return 0;

         };

}

## UDP广播

### 创建UDP套接字

UDPSockFD = socket(AF\_INET,SOCK\_DGRAM,0)

### bind套接字

struct sockaddr\_in receiving\_from

bzero(&receiving\_from,sizeof(receiving\_from));

receiving\_from.sin\_family = AF\_INET

receiving\_from.sin\_addr.s\_addr = inet\_addr(“192.168.0.0.1”)

receiving\_from.sin\_port = htons(10006)

bind(UDPSockFD,(struct sockaddr\*)(&receiving\_from), sizeof(receiving\_from))

### setsocktopt

struct ip\_mreq mreq;

mreq.imr\_multiaddr.s\_addr = inet\_addr(“192.168.0.0.1”);

mreq.imr\_interface.s\_addr = inet\_addr(INADDR\_ANY);

setsockopt(UDPSockFD,IPPROTO\_IP,IP\_ADD\_MEMBERSHIP, &mreq,sizeof(mreq));

### select

fd\_set ReceiveBroad,ReadSock

FD\_CLR(&ReceiveBroad)

FD\_SET(UDPSockFD, &ReceiveBroad);

long lMaxSoc = UDPSockFD

for(;;)

{

ReadSock = ReceiveBroad;

nFound = 0;

nFound = select(lMaxSoc+1，& ReadSock，NULL，NULL，NULL);

if (nFound <=0){ continue;}

FD\_ISSET(UDPSockFD,& ReadSock)

{

recv\_from(UDPSockFD,buffer,sizeof(buffer),0,(struct sockaddr \*) SendAddr,sizeof(SendAddr))

}

}

## socket编码常见错误

### 1、accept句柄耗尽

服务器每次accept一个链接之后，都会分配新的socket资源。Linux下对每个进程所能使用的文件句柄数是有限制的，默认是1024。扣除stdin,stdout,stderr,只有1021个句柄可用。一旦出现了accept的连接数超过了这个限制后就会很尴尬：accept返回-1，errno = 24,即EMFILE。如果采用Select、poll 或者epoll LT方式，系统会不停通知你，但是你没法拿到那个fd，也就不能处理，甚至连close都不能。只能尴尬地让cpu跑在100%,期待着某个客户端断线，某个句柄恰好被close，然后才能处理之。

解决办法有两种，

1，   准备一个空闲的nullfd = open(“/dev/null”), close(nullFd),然后accept,close(socketFd)，最后在nullfd = open(“/dev/null”)，缺点，并不是线程安全的。

2，   直接在代码中限制最大允许连接数，一旦超过则直接踢出。缺点，需要额外的关注。

### 2、Socket资源不释放

服务器未能有效地检测和处理socket关闭事件，导致最后socket资源被耗尽，再也不能连入新的连接。

解决办法:keepalive、网络连接心跳检测，定时关闭不活动的连接。

### 3、发送缓冲使用不当，数据包乱序

当有数据要发送时，无视发送缓冲直接调用send函数。在同一连接中会发生数据A发送了一部分，又发送数据B的一部分，在发送数据A的一部分。令接收者无所适从。

解决办法：从不直接调用send发送数据，应将发送数据放入发送缓冲中处理。

### 4、网络串包

将A的信息发给了B。通常发生在服务器用socketFd1接受A的请求，服务器处理中。A掉线，服务器close(socketFd1)。这时B再上线，POSIX标准要求每次打开文件的时候必须使用当前最小可用的文件描述符号，于是B使用了socketFd1句柄。服务器就将A的应答发送给了B.

解决办法：

1. socket错误处理时不能直接close了事。

2，处理应答时不能直接将数据包丢给socket句柄，要建立某种机制确认socket句柄是你想发送的那个。

### 5、服务器端太多的TIME\_WAIT

 主动关闭的Socket端会进入TIME\_WAIT状态，并且持续2MSL时间长度，MSL就是maximum segment lifetime(最大分节生命期），这是一个IP数据包能在互联网上生存的最长时间，超过这个时间将在网络中消失。MSL在RFC 1122上建议是2分钟，而源自berkeley的TCP实现传统上使用30秒，因而，TIME\_WAIT状态一般维持在1-4分钟。如果每次关闭socket的动作都由服务器发起，那么TIME\_WAIT状态就会留在服务端,服务端的内核就需要维护更多的状态。收到ip包，做hash运算，hlist冲突的概率更大。

解决办法：使用shutdown(sockfd, SHUT\_WR)通知客户端，由客户端执行close，将TIME\_WAIT留在客户端。

### 6、没有处理SIGPIPE引起服务器崩溃

当一个进程向某个已收到RST的Socket执行写操作时，内核将向进程发送一个SIGPIPE信号。该信号的默认行为是终止进程。

解决办法：忽略该信号。

struct sigaction act;

act.sa\_handler = SIG\_IGN;

if (sigaction(SIGPIPE, &act, NULL) == 0) {

LOG("SIGPIPE ignore");

}

### 7、阻塞I/O没有处理EINTR、非阻塞I/O没有处理EAGAIN

对于慢系统调用，像accept, receive这样有可能无法返回的函数的进程一旦捕获了某个信号，并且相应的信号处理函数返回时，该系统调用有可能返回一个EINTR错误。

非阻塞模式下调用了阻塞操作，在该操作没有完成就返回EAGAIN。EAGAIN不会破坏socket的同步，不用管它。

# Linux系统

## [linux & xp 双系统 重装的问题](http://www.cnblogs.com/alamps/archive/2010/03/04/1677931.html)

\* 重装 xp

linux & xp 双系统 重装 xp 后，linux 的 grub 就不起作用了，通过以下步骤可以在启动时重新显示 grub 界面：

\* 进入 xp 系统

\* 从 http://download.gna.org/grub4dos 下载 grub4dos 文件，应该是个压缩包；

\* 文件夹选项中设置显示所有文件和文件夹；

\* 解压 grub4dos 压缩包，将 grldr,grub.exe,menu.lst 三个文件放在 c盘 根目录下(如果没有c盘，找那个有 boot.ini 文件的盘)

\* 找到 c盘下 的 boot.ini 文件，取消其只读属性，在最后加上：Start with GRUB4DOS

\* 重启电脑，启动界面选择 Start with GRUB4DOS ，选择 enter command line

\* 输入 find /boot/grub/stage1 ，可以看到类似 (hd0,5) 的结果，下面就以 (hd0,5) 为例

\* 输入 root (hd0,5) 回车，注意 root 后有个 空格

\* 输入 setup (hd0) 回车，主义 setup 后有个 空格

\* 输入 reboot 重新启动

\* 重启后可以发现已经是 grub 界面了，爽了；

\*

\* 测试下 linux 和 xp 系统是否都能成功进入，linux 应该肯定没有问题，如果xp提示无法启动，则有可能是重装 xp 时，分区改变了，按下面的进行修正；

\* 如果xp是因为分区改变导致无法通过grub启动，则按照如下步骤修正：

\* 进入 linux 系统，我下面是以 ubuntu9.0.4 desktop 为例的；

\* 打开命令行，转换为超级管理员身份，依次执行命令：

cd /boot/grub

cp menu.lst menu.lst.bak

vi menu.lst

编辑 menu.lst 时，注意找 对应着 xp 的 “title” 5个字母，我的如下：

# on /dev/sda1

title Microsoft Windows XP Professional

rootnoverify (hd0,1)

修改 (hd0,1) 为 (hd0,n) 其中 n 是你的xp对应的分区，如果你不知道自己xp所在的分区，可以多实验几个直到从grub能启动xp，如果一直都不行，那可能不是分区的问题，是其它什么问 题；

\*

\*

\*

\* 重装 linux

重装后应该没什么问题，还是从 grub 启动；

====================================================================================================

重装xp后恢复ubuntu引导

(2009-01-15 18:32:03)

安装 ubuntu 之后因为种种原因需要重新安装 xp ，这样就会将硬盘的 mbr （主引导扇区）覆盖，导致 ubuntu 无法启动，网上对于这种情况提供了很多解决办法，但是有些复杂。经过参考网上的一些资料，以及自己的实践，提供给大家两种最简便的方法：

准备：

1. 下载 grub4dos 文件，http://download.gna.org/grub4dos/ 将 grldr 、 grub.exe 、 menu.lst 三个文件放在 C 盘根目录下

2. 打开我的电脑，选择工具 -> 文件夹选项 -> 查看 -> 选中显示所有文件和文件夹

3. 将 c 盘根目录下的 boot.ini 文件只读属性去掉，在最后加上 c:\grldr="Start Grub4Dos" ，保存

4. 重新启动计算机，进入 ”Start with GRUB4DOS”

方法一：

进入 ”Start with GRUB4DOS” 后，可以看见一个 “find and boot Linux with menu.lst already installed” 的选项了，选择它，就会自动去搜索机器上的 ubuntu ，然后选择 ubuntu 就可以正常启动 ubuntu 了

优点 ：什么都不用做，简单方便

缺点 ：每次都需要系统去搜索 ubuntu 所在的分区，不能一劳永逸

PS ：关于那个 “find and boot Linux with menu.lst already installed” 的选项，其实是搜索磁盘里面的 ubuntu 分区，如果每一次进入都搜索分区未免有些多余。所以我们可以在进入 ubuntu 以后把 /boot/grub/menu.lst 文件拷贝到 c 盘根目录下，这样进入 ”Start with GRUB4DOS” 就不用在检测 ubuntu 的分区了。

方法二：

进入 ”Start with GRUB4DOS” 后，可以看见一个 "enter command line" 选项，选择它输入 root (hdX,Y) ，如果不知道 x 是多少先输入：

#> find /boot/grub/stage1

根据返回的具体结果再输入，我机器的返回结果是（ hd0 ， 2 ）：

#>root (hd0,2)

执行成功之后，最后输入：

#> setup (hd0)

重启，可爱的 GRUB 启动栏又回来了！

优点 ：不用每次都搜索硬盘

缺点 ：需要敲两三条命令

PS ： root 和（ hd0,2) 之间是有一个空格的， setup 和（ hd0 ）之间也是有空格的。 root (hdX,Y) 中的 hdX 的这个 X ，是硬盘数，从零开始，如果是一个硬盘就是 hd0 。 Y 是根所在的分区，像我安装在 2, 这个 Y 就用 2 。

## ubuntu系统单用户模式下重置root密码

1. 启动时候一直按住shift键

以前的Ubuntu版本，启动grub的时候，有一个grub loading，press ESC to get the menu的提示，所以按ESC就可以看到grub界面

现在的Ubuntu没有了，现在的做法是，启动的时候按住shift键，就可以看到grub界面了。

（接下来就修改启动参数，加入single，然后按ctrl+x启动，就能看到一个ncurse的界面，选择drop to root prompt，就可以得到一个root的terminal然后就可以重新设置root口令了）

2. 修改grub文件来完成

点击(此处)折叠或打开

1.sudo gedit /etc/default/grub

修改如下：

GRUB\_HIDDEN\_TIMEOUT=0

#GRUB\_HIDDEN\_TIMEOUT\_QUIET=true

GRUB\_TIMEOUT=0

2.sudo update-grub

下面展开说明：

首先修改 /etc/default/grub.cfg文件，参考以下config文件注释掉GRUB\_HIDDEN\_TIMEOUT=0

点击(此处)折叠或打开

# If you change this file, run 'update-grub' afterwards to update

# /boot/grub/grub.cfg.

# For full documentation of the options in this file, see:

# info -f grub -n 'Simple configuration'

GRUB\_DEFAULT=0

#GRUB\_HIDDEN\_TIMEOUT=0

GRUB\_HIDDEN\_TIMEOUT\_QUIET=true

GRUB\_TIMEOUT=10

GRUB\_DISTRIBUTOR=`lsb\_release -i -s 2> /dev/null || echo Debian`

GRUB\_CMDLINE\_LINUX\_DEFAULT="quiet splash"

GRUB\_CMDLINE\_LINUX=""

# Uncomment to enable BadRAM filtering, modify to suit your needs

# This works with Linux (no patch required) and with any kernel that obtains

# the memory map information from GRUB (GNU Mach, kernel of FreeBSD ...)

#GRUB\_BADRAM="0x01234567,0xfefefefe,0x89abcdef,0xefefefef"

# Uncomment to disable graphical terminal (grub-pc only)

#GRUB\_TERMINAL=console

# The resolution used on graphical terminal

# note that you can use only modes which your graphic card supports via VBE

# you can see them in real GRUB with the command `vbeinfo'

#GRUB\_GFXMODE=640x480

# Uncomment if you don't want GRUB to pass "root=UUID=xxx" parameter to Linux

#GRUB\_DISABLE\_LINUX\_UUID=true

# Uncomment to disable generation of recovery mode menu entries

#GRUB\_DISABLE\_RECOVERY="true"

# Uncomment to get a beep at grub start

#GRUB\_INIT\_TUNE="480 440 1"

然后执行"sudo update-grub"。这样以后在grub.cfg中会多出以下代码

点击(此处)折叠或打开

### BEGIN /etc/grub.d/30\_os-prober ###

if [ "x${timeout}" != "x-1" ]; then

if keystatus; then

if keystatus --shift; then

set timeout=-1

else

set timeout=0

fi

else

if sleep --interruptible 3 ; then

set timeout=0

fi

fi

fi

### END /etc/grub.d/30\_os-prober ###

最后简单介绍下其中的几个参数

点击(此处)折叠或打开

1.GRUB\_HIDDEN\_TIMEOUT=0

此配置将影响菜单显示。若设置此选项,将在此时间内隐藏菜单而显示引导画面。

菜单将会被隐藏,除非在此行开头加上一个 # 符号。(# GRUB\_HIDDEN\_TIMEOUT=0)。

GRUB 2 第一次执行时将会寻找其他操作系统。若没有其他操作系统被检测到,菜单将会配置为隐藏。若辨认出其他操作系统,菜单将会显示。

若是大于 0 的整数,系统将会依此配置的秒数暂停,但不会显示菜单。

0 则菜单不会显示,也不会有延迟。

使用者可以在启动时按住 SHIFT 键不放以强制显示菜单。

启动过程中,系统将会检查 SHIFT 键状态。若无法辨识按键状态,会有一个短时间的延迟让使用者可通过按下 ESC 键来显示菜单。

2.GRUB\_HIDDEN\_TIMEOUT\_QUIET=true

true 不显示倒计时。屏幕将会是空白的。

false 在 GRUB\_HIDDEN\_TIMEOUT 中配置的时间,空白屏幕上会有一个倒数计时器。

3.GRUB\_TIMEOUT=10

此命令将顺从 GRUB\_HIDDEN\_TIMEOUT 配置,除非 GRUB\_HIDDEN\_TIMEOUT 被注释掉(#)。若 GRUB\_HIDDEN\_TIMEOUT 启用,则当菜单显示时,GRUB\_TIMEOUT 将会只执行一次。

配置此值为 -1 将会导致菜单一直显示,直到用户选择。

GRUB 2 菜单默认为隐藏,除非其他操作系统被系统检测到。若没有其他操作系统,此行将会被注释掉,除非使用者修改它。为了在每次启动时显示菜单,去掉此行的注释并使用 1 或更大的值。

## [Linux写时拷贝技术(copy-on-write)](http://www.cnblogs.com/biyeymyhjob/archive/2012/07/20/2601655.html)

**COW技术初窥：**

在Linux程序中，fork（）会产生一个和父进程完全相同的子进程，但子进程在此后多会exec系统调用，出于效率考虑，linux中引入了“写时复制“技术，也就是只有进程空间的各段的内容要发生变化时，才会将父进程的内容复制一份给子进程。

那么子进程的物理空间没有代码，怎么去取指令执行exec系统调用呢？

在fork之后exec之前两个进程用的是相同的物理空间（内存区），子进程的代码段、数据段、堆栈都是指向父进程的物理空间，也就是说，两者的虚拟空间不同，但其对应的物理空间是同一个。当父子进程中有更改相应段的行为发生时，再为子进程相应的段分配物理空间，如果不是因为exec，内核会给子进程的数据段、堆栈段分配相应的物理空间（至此两者有各自的进程空间，互不影响），而代码段继续共享父进程的物理空间（两者的代码完全相同）。而如果是因为exec，由于两者执行的代码不同，子进程的代码段也会分配单独的物理空间。

 在网上看到还有个细节问题就是，fork之后内核会通过将子进程放在队列的前面，以让子进程先执行，以免父进程执行导致写时复制，而后子进程执行exec系统调用，因无意义的复制而造成效率的下降。

**COW详述：**

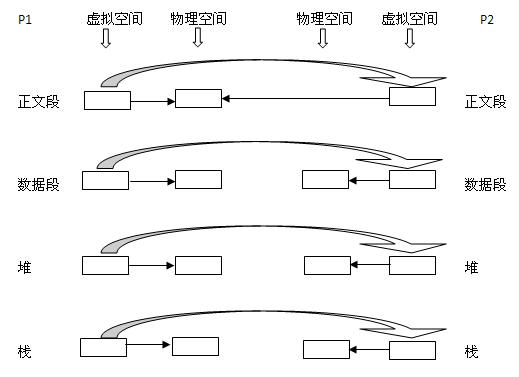
现在有一个父进程P1，这是一个主体，那么它是有灵魂也就身体的。现在在其虚拟地址空间（有相应的数据结构表示）上有：正文段，数据段，堆，栈这四个部分，相应的，内核要为这四个部分分配各自的物理块。即：正文段块，数据段块，堆块，栈块。至于如何分配，这是内核去做的事，在此不详述。

1.现在P1用fork()函数为进程创建一个子进程P2，

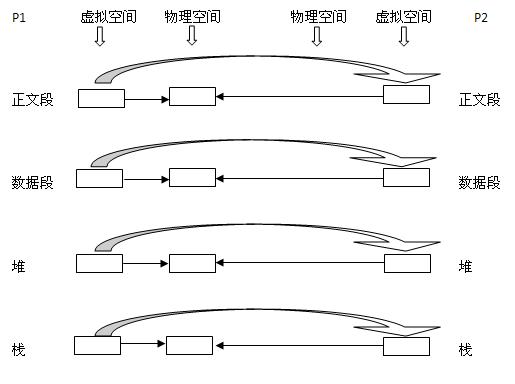
内核：

（1）复制P1的正文段，数据段，堆，栈这四个部分，注意是其内容相同。

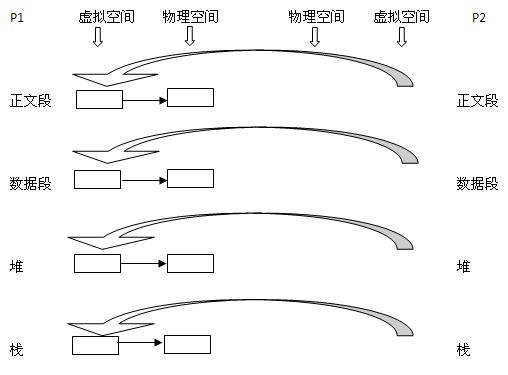
（2）为这四个部分分配物理块，P2的：正文段－＞PI的正文段的物理块，其实就是不为P2分配正文段块，让P2的正文段指向P1的正文段块，数据段－＞P2自己的数据段块（为其分配对应的块），堆－＞P2自己的堆块，栈－＞P2自己的栈块。如下图所示：同左到右大的方向箭头表示复制内容。



2.写时复制技术：内核只为新生成的子进程创建虚拟空间结构，它们来复制于父进程的虚拟究竟结构，但是不为这些段分配物理内存，它们共享父进程的物理空间，当父子进程中有更改相应段的行为发生时，再为子进程相应的段分配物理空间。



3.vfork()：这个做法更加火爆，内核连子进程的虚拟地址空间结构也不创建了，直接共享了父进程的虚拟空间，当然了，这种做法就顺水推舟的共享了父进程的物理空间



通过以上的分析，相信大家对进程有个深入的认识，它是怎么一层层体现出自己来的，进程是一个主体，那么它就有灵魂与身体，系统必须为实现它创建相应的实体， 灵魂实体与物理实体。这两者在系统中都有相应的数据结构表示，物理实体更是体现了它的物理意义。以下援引LKD

传统的fork()系统调用直接把所有的资源复制给新创建的进程。这种实现过于简单并且效率低下，因为它拷贝的数据也许并不共享，更糟的情况是，如果新进程打算立即执行一个新的映像，那么所有的拷贝都将前功尽弃。Linux的fork()使用写时拷贝（copy-on-write）页实现。写时拷贝是一种可以推迟甚至免除拷贝数据的技术。内核此时并不复制整个进程地址空间，而是让父进程和子进程共享同一个拷贝。只有在需要写入的时候，数据才会被复制，从而使各个进程拥有各自的拷贝。也就是说，资源的复制只有在需要写入的时候才进行，在此之前，只是以只读方式共享。这种技术使地址空间上的页的拷贝被推迟到实际发生写入的时候。在页根本不会被写入的情况下—举例来说，fork()后立即调用exec()—它们就无需复制了。fork()的实际开销就是复制父进程的页表以及给子进程创建惟一的进程描述符。在一般情况下，进程创建后都会马上运行一个可执行的文件，这种优化可以避免拷贝大量根本就不会被使用的数据（地址空间里常常包含数十兆的数据）。由于Unix强调进程快速执行的能力，所以这个优化是很重要的。这里补充一点：**Linux COW与exec没有必然联系**

PS：实际上COW技术不仅仅在Linux进程上有应用，其他例如C++的String在有的IDE环境下也支持COW技术，即例如：

string str1 = "hello world";

string str2 = str1;

之后执行代码:

str1[1]='q';

str2[1]='w';

在开始的两个语句后，str1和str2存放数据的地址是一样的，而在修改内容后，str1的地址发生了变化，而str2的地址还是原来的,这就是C++中的COW技术的应用，不过VS2005似乎已经不支持COW。

## 设置Linux系统IP

1.调试时修改IP,仅在当前生效,重启后恢复为原有IP

ifconfig eth0 192.168.63.27 netmask 255.255.255.0 route add default gw 192.168.63.1 up

2.永久生效(即重启后也能生效)

方法1:配置网卡的配置文件

修改/etc/sysconfig/network-scripts/ifcfg-eth0

然后重启服务service network restart生效,或者/etc/init.d/network restart生效

方法2:

将ifconfig eth0 192.168.63.27 netmask 255.255.255.0 route add default gw 192.168.63.1 保存在/etc/init.d/rc.local文件中

## 启动服务

chkconfig --list | grep telnet 查看telnet服务

service vsftpd status/start/stop

service iptables stop 防火墙

## 用户ftp无法上传文件问题

设置selinux无效

setenforce 0

如果永久无效则

vi /etc/sysconfig/selinux

SELINUX=enforcing改成SELINUX=disabled

## ORA-27125: unable to create shared memory segment

解决方法：

[oracle@yans1 ~]$ id oracle

uid=500(oracle) gid=502(oinstall) groups=502(oinstall),501(dba)

[oracle@yans1 ~]$ more /proc/sys/vm/hugetlb\_shm\_group

0

下面用root执行下面的命令，将dba组添加到系统内核中：

root# echo 501 > /proc/sys/vm/hugetlb\_shm\_group

# Linux Epoll介绍

## 1. Epoll是何方神圣？

Epoll可是当前在Linux下开发大规模并发网络程序的热门人选，Epoll 在Linux2.6内核中正式引入，和select相似，其实都I/O多路复用技术而已，并没有什么神秘的。

其实在Linux下设计并发网络程序，向来不缺少方法，比如典型的Apache模型（Process Per Connection，简称PPC），TPC（Thread PerConnection）模型，以及select模型和poll模型，那为何还要再引入Epoll这个东东呢？那还是有得说说的…

### Epoll的优点

\* 事件驱动，高性能；  
\* 轻量级，专注于网络；   
\* 跨平台，支持 Windows、Linux、Mac Os等；   
\* 支持多种 I/O多路复用技术， epoll、poll、dev/poll、select 和kqueue 等；

\* 支持 I/O，定时器和信号等事件；

## 2. 常用模型的缺点

如果不摆出来其他模型的缺点，怎么能对比出Epoll的优点呢。

### 2.1 PPC/TPC模型

这两种模型思想类似，就是让每一个到来的连接一边自己做事去，别再来烦我。只是PPC是为它开了一个进程，而TPC开了一个线程。可是别烦我是有代价的，它要时间和空间啊，连接多了之后，那么多的进程/线程切换，这开销就上来了；因此这类模型能接受的最大连接数都不会高，一般在几百个左右。

### 2.2 select模型

1. 最大并发数限制，因为一个进程所打开的FD（文件描述符）是有限制的，由FD\_SETSIZE设置，默认值是1024/2048，因此Select模型的最大并发数就被相应限制了。自己改改这个FD\_SETSIZE？想法虽好，可是先看看下面吧…

2. 效率问题，select每次调用都会线性扫描全部的FD集合，这样效率就会呈现线性下降，把FD\_SETSIZE改大的后果就是，大家都慢慢来，什么？都超时了？？！！

3. 内核/用户空间 内存拷贝问题，如何让内核把FD消息通知给用户空间呢？在这个问题上select采取了内存拷贝方法。

### 2.3 poll模型

基本上效率和select是相同的，select缺点的2和3它都没有改掉。

## 3. Epoll的提升

把其他模型逐个批判了一下，再来看看Epoll的改进之处吧，其实把select的缺点反过来那就是Epoll的优点了。

3.1. Epoll没有最大并发连接的限制，上限是最大可以打开文件的数目，这个数字一般远大于2048, 一般来说这个数目和系统内存关系很大，具体数目可以cat /proc/sys/fs/file-max察看。

3.2. 效率提升，Epoll最大的优点就在于它只管你“活跃”的连接，而跟连接总数无关，因此在实际的网络环境中，Epoll的效率就会远远高于select和poll。

3.3. 内存拷贝，Epoll在这点上使用了“共享内存”，这个内存拷贝也省略了。

## 4. Epoll为什么高效

Epoll的高效和其数据结构的设计是密不可分的，这个下面就会提到。

首先回忆一下select模型，当有I/O事件到来时，select通知应用程序有事件到了快去处理，而应用程序必须轮询所有的FD集合，测试每个FD是否有事件发生，并处理事件；代码像下面这样：

int res = select(maxfd+1, &readfds, NULL, NULL, 120);

if(res > 0)

{

for(int i = 0; i < MAX\_CONNECTION; i++)

{

if(FD\_ISSET(allConnection[i],&readfds))

{

handleEvent(allConnection[i]);

}

}

}

// if(res == 0) handle timeout, res < 0 handle error

Epoll不仅会告诉应用程序有I/0事件到来，还会告诉应用程序相关的信息，这些信息是应用程序填充的，因此根据这些信息应用程序就能直接定位到事件，而不必遍历整个FD集合。

Int res = epoll\_wait(epfd, events, 20, 120);

for(int i = 0; i < res;i++)

{

handleEvent(events[n]);

}

## 5. Epoll关键数据结构

前面提到Epoll速度快和其数据结构密不可分，其关键数据结构就是：

struct epoll\_event {

\_\_uint32\_t events; // Epoll events

epoll\_data\_t data; // User datavariable

};

typedef union epoll\_data {

void \*ptr;

int fd;

\_\_uint32\_t u32;

\_\_uint64\_t u64;

} epoll\_data\_t;

可见epoll\_data是一个union结构体,借助于它应用程序可以保存很多类型的信息:fd、指针等等。有了它，应用程序就可以直接定位目标了。

## 6. 使用Epoll

既然Epoll相比select这么好，那么用起来如何呢？会不会很繁琐啊…先看看下面的三个函数吧，就知道Epoll的易用了。

int epoll\_create(int size);

生成一个Epoll专用的文件描述符，其实是申请一个内核空间，用来存放你想关注的socket fd上是否发生以及发生了什么事件。size就是你在这个Epoll fd上能关注的最大socket fd数，大小自定，只要内存足够。

Int epoll\_ctl(int epfd, int op, int fd, struct epoll\_event \*event);

控制某个Epoll文件描述符上的事件：注册、修改、删除。其中参数epfd是epoll\_create()创建Epoll专用的文件描述符。相对于select模型中的FD\_SET和FD\_CLR宏。

Int epoll\_wait(int epfd,structepoll\_event \* events,int maxevents,int timeout);

等待I/O事件的发生；参数说明：

epfd:由epoll\_create() 生成的Epoll专用的文件描述符；

epoll\_event:用于回传代处理事件的数组；

maxevents:每次能处理的事件数；

timeout:等待I/O事件发生的超时值；

返回发生事件数。

相对于select模型中的select函数。

## 7. 例子程序

下面是一个简单Echo Server的例子程序，麻雀虽小，五脏俱全，还包含了一个简单的超时检查机制，简洁起见没有做错误处理。

[cpp] view plain copy

//

// a simple echo server using epoll in linux

//

// 2009-11-05

// 2013-03-22:修改了几个问题，1是/n格式问题，2是去掉了原代码不小心加上的ET模式;

// 本来只是简单的示意程序，决定还是加上 recv/send时的buffer偏移

// by sparkling

//

#include <sys/socket.h>

#include <sys/epoll.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <fcntl.h>

#include <unistd.h>

#include <stdio.h>

#include <errno.h>

#include <iostream>

using namespace std;

#define MAX\_EVENTS 500

struct myevent\_s

{

int fd;

void (\*call\_back)(int fd, int events, void \*arg);

int events;

void \*arg;

int status; // 1: in epoll wait list, 0 not in

char buff[128]; // recv data buffer

int len, s\_offset;

long last\_active; // last active time

};

// set event

void EventSet(myevent\_s \*ev, int fd, void (\*call\_back)(int, int, void\*), void \*arg)

{

ev->fd = fd;

ev->call\_back = call\_back;

ev->events = 0;

ev->arg = arg;

ev->status = 0;

bzero(ev->buff, sizeof(ev->buff));

ev->s\_offset = 0;

ev->len = 0;

ev->last\_active = time(NULL);

}

// add/mod an event to epoll

void EventAdd(int epollFd, int events, myevent\_s \*ev)

{

struct epoll\_event epv = {0, {0}};

int op;

epv.data.ptr = ev;

epv.events = ev->events = events;

if(ev->status == 1){

op = EPOLL\_CTL\_MOD;

}

else{

op = EPOLL\_CTL\_ADD;

ev->status = 1;

}

if(epoll\_ctl(epollFd, op, ev->fd, &epv) < 0)

printf("Event Add failed[fd=%d], evnets[%d]\n", ev->fd, events);

else

printf("Event Add OK[fd=%d], op=%d, evnets[%0X]\n", ev->fd, op, events);

}

// delete an event from epoll

void EventDel(int epollFd, myevent\_s \*ev)

{

struct epoll\_event epv = {0, {0}};

if(ev->status != 1) return;

epv.data.ptr = ev;

ev->status = 0;

epoll\_ctl(epollFd, EPOLL\_CTL\_DEL, ev->fd, &epv);

}

Int g\_epollFd;

myevent\_s g\_Events[MAX\_EVENTS+1]; // g\_Events[MAX\_EVENTS] is used by listen fd

void RecvData(int fd, int events, void \*arg);

void SendData(int fd, int events, void \*arg);

// accept new connections from clients

void AcceptConn(int fd, int events, void \*arg)

{

struct sockaddr\_in sin;

socklen\_t len = sizeof(struct sockaddr\_in);

int nfd, i;

// accept

if((nfd = accept(fd, (struct sockaddr\*)&sin, &len)) == -1)

{

if(errno != EAGAIN && errno != EINTR)

{

}

printf("%s: accept, %d", \_\_func\_\_, errno);

return;

}

do

{

for(i = 0; i < MAX\_EVENTS; i++)

{

if(g\_Events[i].status == 0)

{

break;

}

}

if(i == MAX\_EVENTS)

{

printf("%s:max connection limit[%d].", \_\_func\_\_, MAX\_EVENTS);

break;

}

// set nonblocking

int iret = 0;

if((iret = fcntl(nfd, F\_SETFL, O\_NONBLOCK)) < 0)

{

printf("%s: fcntl nonblocking failed:%d", \_\_func\_\_, iret);

break;

}

// add a read event for receive data

EventSet(&g\_Events[i], nfd, RecvData, &g\_Events[i]);

EventAdd(g\_epollFd, EPOLLIN, &g\_Events[i]);

}while(0);

printf("new conn[%s:%d][time:%d], pos[%d]\n", inet\_ntoa(sin.sin\_addr),

ntohs(sin.sin\_port), g\_Events[i].last\_active, i);

}

// receive data

void RecvData(int fd, int events, void \*arg)

{

struct myevent\_s \*ev = (struct myevent\_s\*)arg;

int len;

// receive data

len = recv(fd, ev->buff+ev->len, sizeof(ev->buff)-1-ev->len, 0);

EventDel(g\_epollFd, ev);

if(len > 0)

{

ev->len += len;

ev->buff[len] = '\0';

printf("C[%d]:%s\n", fd, ev->buff);

// change to send event

EventSet(ev, fd, SendData, ev);

EventAdd(g\_epollFd, EPOLLOUT, ev);

}

else if(len == 0)

{

close(ev->fd);

printf("[fd=%d] pos[%d], closed gracefully.\n", fd, ev-g\_Events);

}

else

{

close(ev->fd);

printf("recv[fd=%d] error[%d]:%s\n", fd, errno, strerror(errno));

}

}

// send data

void SendData(int fd, int events, void \*arg)

{

struct myevent\_s \*ev = (struct myevent\_s\*)arg;

int len;

// send data

len = send(fd, ev->buff + ev->s\_offset, ev->len - ev->s\_offset, 0);

if(len > 0)

{

printf("send[fd=%d], [%d<->%d]%s\n", fd, len, ev->len, ev->buff);

ev->s\_offset += len;

if(ev->s\_offset == ev->len)

{

// change to receive event

EventDel(g\_epollFd, ev);

EventSet(ev, fd, RecvData, ev);

EventAdd(g\_epollFd, EPOLLIN, ev);

}

}

else

{

close(ev->fd);

EventDel(g\_epollFd, ev);

printf("send[fd=%d] error[%d]\n", fd, errno);

}

}

void InitListenSocket(int epollFd, short port)

{

int listenFd = socket(AF\_INET, SOCK\_STREAM, 0);

fcntl(listenFd, F\_SETFL, O\_NONBLOCK); // set non-blocking

printf("server listen fd=%d\n", listenFd);

EventSet(&g\_Events[MAX\_EVENTS], listenFd, AcceptConn, &g\_Events[MAX\_EVENTS]);

// add listen socket

EventAdd(epollFd, EPOLLIN, &g\_Events[MAX\_EVENTS]);

// bind & listen

sockaddr\_in sin;

bzero(&sin, sizeof(sin));

sin.sin\_family = AF\_INET;

sin.sin\_addr.s\_addr = INADDR\_ANY;

sin.sin\_port = htons(port);

bind(listenFd, (const sockaddr\*)&sin, sizeof(sin));

listen(listenFd, 5);

}

int main(int argc, char \*\*argv)

{

unsigned short port = 12345; // default port

if (argc == 2)

{

port = atoi(argv[1]);

}

// create epoll

g\_epollFd = epoll\_create(MAX\_EVENTS);

if(g\_epollFd <= 0) printf("create epoll failed.%d\n", g\_epollFd);

// create & bind listen socket, and add to epoll, set non-blocking

InitListenSocket(g\_epollFd, port);

// event loop

struct epoll\_event events[MAX\_EVENTS];

printf("server running:port[%d]\n", port);

int checkPos = 0;

while(1){

// a simple timeout check here, every time 100, better to use a mini-heap, and add timer event

long now = time(NULL);

for(int i = 0; i < 100; i++, checkPos++) // doesn't check listen fd

{

if(checkPos == MAX\_EVENTS) checkPos = 0; // recycle

if(g\_Events[checkPos].status != 1) continue;

long duration = now - g\_Events[checkPos].last\_active;

if(duration >= 60) // 60s timeout

{

close(g\_Events[checkPos].fd);

printf("[fd=%d] timeout[%d--%d].\n", g\_Events[checkPos].fd, g\_Events[checkPos].last\_active, now);

EventDel(g\_epollFd, &g\_Events[checkPos]);

}

}

// wait for events to happen

int fds = epoll\_wait(g\_epollFd, events, MAX\_EVENTS, 1000);

if(fds < 0){

printf("epoll\_wait error, exit\n");

break;

}

for(int i = 0; i < fds; i++){

myevent\_s \*ev = (struct myevent\_s\*)events[i].data.ptr;

if((events[i].events&EPOLLIN)&&(ev->events&EPOLLIN)) // read event

{

ev->call\_back(ev->fd, events[i].events, ev->arg);

}

if((events[i].events&EPOLLOUT)&&(ev->events&EPOLLOUT)) // write event

{

ev->call\_back(ev->fd, events[i].events, ev->arg);

}

}

}

// free resource

return 0;

}

# 汇编

## Linux AT&T汇编错误：Error: invalid instruction suffix for `push'

在 Linux 使用 AT&T 语法写汇编程序时

程序中有一条命令为：pushl %eax

使用 as 编译时出现错误：

Error: invalid instruction suffix for `push'

原因是 64 位系统和 32 位系统的差别引起的

**解决方法为：**

在代码开头添加 .code32 即可

# dbx调试程序

## 查看core是哪个文件造成的:

dbx 可执行文件名 core, 然后敲where

# gdb调试程序

GDB是一个强大的命令行调试工具。大家知道命令行的强大就是在于，其可以形成执行序列，形成脚本。UNIX下的软件全是命令行的，这给程序开发提代供了极大的便利，命令行软件的优势在于，它们可以非常容易的集成在一起，使用几个简单的已有工具的命令，就可以做出一个非常强大的功能。

于是UNIX下的软件比Windows下的软件更能有机地结合，各自发挥各自的长处，组合成更为强劲的功能。而Windows下的图形软件基本上是各自为营，互相不能调用，很不利于各种软件的相互集成。在这里并不是要和Windows做个什么比较，所谓“寸有所长，尺有所短”，图形化工具还是有不如命令行的地方。

用GDB调试程序

GDB概述

————

GDB是GNU开源组织发布的一个强大的UNIX下的程序调试工具。或许，各位比较喜欢那种图形界面方式的，像VC、BCB等IDE的调试，但如果你是在UNIX平台下做软件，你会发现GDB这个调试工具有比VC、BCB的图形化调试器更强大的功能。所谓“寸有所长，尺有所短”就是这个道理。

一般来说，

## GDB主要帮忙你完成下面四个方面的功能：

1、启动你的程序，可以按照你的自定义的要求随心所欲的运行程序。

2、可让被调试的程序在你所指定的调置的断点处停住。（断点可以是条件表达式）

3、当程序被停住时，可以检查此时你的程序中所发生的事。

4、动态的改变你程序的执行环境。

从上面看来，GDB和一般的调试工具没有什么两样，基本上也是完成这些功能，不过在细节上，你会发现GDB这个调试工具的强大，大家可能比较习惯了图形化的调试工具，但有时候，命令行的调试工具却有着图形化工具所不能完成的功能。让我们一一看来。

一个调试示例

——————

源程序：tst.c

1 #include

2

3 int func(int n)

4 {

5 int sum=0,i;

6 for(i=0; i

7 {

8 sum+=i;

9 }

10 return sum;

11 }

12

13

14 main()

15 {

16 int i;

17 long result = 0;

18 for(i=1; i<=100; i++)

19 {

20 result += i;

21 }

22

23 printf("result[1-100] = %d \n", result );

24 printf("result[1-250] = %d \n", func(250) );

25 }

编译生成执行文件：（Linux下）

hchen/test> cc -g tst.c -o tst

使用GDB调试：

hchen/test> gdb tst <---------- 启动GDB

GNU gdb 5.1.1

Copyright 2002 Free Software Foundation, Inc.

GDB is free software, covered by the GNU General Public License, and you are

welcome to change it and/or distribute copies of it under certain conditions.

Type "show copying" to see the conditions.

There is absolutely no warranty for GDB. Type "show warranty" for details.

This GDB was configured as "i386-suse-linux"...

(gdb) l <-------------------- l命令相当于list，从第一行开始例出原码。

1 #include

2

3 int func(int n)

4 {

5 int sum=0,i;

6 for(i=0; i

7 {

8 sum+=i;

9 }

10 return sum;

(gdb) <-------------------- 直接回车表示，重复上一次命令

11 }

12

13

14 main()

15 {

16 int i;

17 long result = 0;

18 for(i=1; i<=100; i++)

19 {

20 result += i;

(gdb) break 16 <-------------------- 设置断点，在源程序第16行处。

Breakpoint 1 at 0x8048496: file tst.c, line 16.

(gdb) break func <-------------------- 设置断点，在函数func()入口处。

Breakpoint 2 at 0x8048456: file tst.c, line 5.

(gdb) info break <-------------------- 查看断点信息。

Num Type Disp Enb Address What

1 breakpoint keep y 0x08048496 in main at tst.c:16

2 breakpoint keep y 0x08048456 in func at tst.c:5

(gdb) r <--------------------- 运行程序，run命令简写

Starting program: /home/hchen/test/tst

Breakpoint 1, main () at tst.c:17 <---------- 在断点处停住。

17 long result = 0;

(gdb) n <--------------------- 单条语句执行，next命令简写。

18 for(i=1; i<=100; i++)

(gdb) n

20 result += i;

(gdb) n

18 for(i=1; i<=100; i++)

(gdb) n

20 result += i;

(gdb) c <--------------------- 继续运行程序，continue命令简写。

Continuing.

result[1-100] = 5050 <----------程序输出。

Breakpoint 2, func (n=250) at tst.c:5

5 int sum=0,i;

(gdb) n

6 for(i=1; i<=n; i++)

(gdb) p i <--------------------- 打印变量i的值，print命令简写。

$1 = 134513808

(gdb) n

8 sum+=i;

(gdb) n

6 for(i=1; i<=n; i++)

(gdb) p sum

$2 = 1

(gdb) n

8 sum+=i;

(gdb) p i

$3 = 2

(gdb) n

6 for(i=1; i<=n; i++)

(gdb) p sum

$4 = 3

(gdb) bt <--------------------- 查看函数堆栈。

#0 func (n=250) at tst.c:5

#1 0x080484e4 in main () at tst.c:24

#2 0x400409ed in \_\_libc\_start\_main () from /lib/libc.so.6

(gdb) finish <--------------------- 退出函数。

Run till exit from #0 func (n=250) at tst.c:5

0x080484e4 in main () at tst.c:24

24 printf("result[1-250] = %d \n", func(250) );

Value returned is $6 = 31375

(gdb) c <--------------------- 继续运行。

Continuing.

result[1-250] = 31375 <----------程序输出。

Program exited with code 027. <--------程序退出，调试结束。

(gdb) q <--------------------- 退出gdb。

hchen/test>

好了，有了以上的感性认识，还是让我们来系统地认识一下gdb吧。

使用GDB

————

一般来说GDB主要调试的是C/C++的程序。要调试C/C++的程序，首先在编译时，我们必须要把调试信息加到可执行文件中。使用编译器（cc/gcc/g++）的 -g 参数可以做到这一点。如：

> cc -g hello.c -o hello

> g++ -g hello.cpp -o hello

如果没有-g，你将看不见程序的函数名、变量名，所代替的全是运行时的内存地址。当你用-g把调试信息加入之后，并成功编译目标代码以后，让我们来看看如何用gdb来调试他。

## 启动GDB的方法有以下几种：

1、gdb program

program也就是你的执行文件，一般在当然目录下。

2、gdb core

用gdb同时调试一个运行程序和core文件，core是程序非法执行后core dump后产生的文件。

3、gdb

如果你的程序是一个服务程序，那么你可以指定这个服务程序运行时的进程ID。gdb会自动attach上去，并调试他。program应该在PATH环境变量中搜索得到。

GDB启动时，可以加上一些GDB的启动开关，详细的开关可以用gdb -help查看。我在下面只例举一些比较常用的参数：

-symbols

-s

从指定文件中读取符号表。

-se file

从指定文件中读取符号表信息，并把他用在可执行文件中。

-core

-c

调试时core dump的core文件。

-directory

-d

加入一个源文件的搜索路径。默认搜索路径是环境变量中PATH所定义的路径。

GDB的命令概貌

## gdb的命令可以使用help命令来查看

启动gdb后，就你被带入gdb的调试环境中，就可以使用gdb的命令开始调试程序了，gdb的命令可以使用help命令来查看，如下所示：

/home/hchen> gdb

GNU gdb 5.1.1

Copyright 2002 Free Software Foundation, Inc.

GDB is free software, covered by the GNU General Public License, and you are

welcome to change it and/or distribute copies of it under certain conditions.

Type "show copying" to see the conditions.

There is absolutely no warranty for GDB. Type "show warranty" for details.

This GDB was configured as "i386-suse-linux".

(gdb) help

List of classes of commands:

aliases -- Aliases of other commands

breakpoints -- Making program stop at certain points

data -- Examining data

files -- Specifying and examining files

internals -- Maintenance commands

obscure -- Obscure features

running -- Running the program

stack -- Examining the stack

status -- Status inquiries

support -- Support facilities

tracepoints -- Tracing of program execution without stopping the program

user-defined -- User-defined commands

Type "help" followed by a class name for a list of commands in that class.

Type "help" followed by command name for full documentation.

Command name abbreviations are allowed if unambiguous.

(gdb)

gdb的命令很多，gdb把之分成许多个种类。**help命令只是例出gdb的命令种类**，如果要看种类中的命令，可以使用help 命令，如：help breakpoints，查看设置断点的所有命令。也可以直接help 来查看命令的帮助。

gdb中，输入命令时，可以不用打全命令，只用打命令的前几个字符就可以了，当然，命令的前几个字符应该要标志着一个唯一的命令，在Linux下，你可以敲击两次TAB键来补齐命令的全称，如果有重复的，那么gdb会把其例出来。

示例一：在进入函数func时，设置一个断点。可以敲入break func，或是直接就是b func

(gdb) b func

Breakpoint 1 at 0x8048458: file hello.c, line 10.

示例二：敲入b按两次TAB键，你会看到所有b打头的命令：

(gdb) b

backtrace break bt

(gdb)

示例三：只记得函数的前缀，可以这样：

(gdb) b make\_ <按TAB键>

（再按下一次TAB键，你会看到:）

make\_a\_section\_from\_file make\_environ

make\_abs\_section make\_function\_type

make\_blockvector make\_pointer\_type

make\_cleanup make\_reference\_type

make\_command make\_symbol\_completion\_list

(gdb) b make\_

GDB把所有make开头的函数全部例出来给你查看。

示例四：调试C++的程序时，有可以函数名一样。如：

(gdb) b 'bubble( M-?

bubble(double,double) bubble(int,int)

(gdb) b 'bubble(

你可以查看到C++中的所有的重载函数及参数。（注：M-?和“按两次TAB键”是一个意思）

要退出gdb时，只用发quit或命令简称q就行了。

GDB中运行UNIX的shell程序

————————————

在gdb环境中，你可以执行UNIX的shell的命令，使用gdb的shell命令来完成：

shell

调用UNIX的shell来执行，环境变量SHELL中定义的UNIX的shell将会被用来执行，如果SHELL没有定义，那就使用UNIX的标准shell：/bin/sh。（在Windows中使用Command.com或cmd.exe）

还有一个gdb命令是make：

make

可以在gdb中执行make命令来重新build自己的程序。这个命令等价于“shell make ”。

在GDB中运行程序

————————

当以gdb 方式启动gdb后，gdb会在PATH路径和当前目录中搜索的源文件。如要确认gdb是否读到源文件，可使用l或list命令，看看gdb是否能列出源代码。

## gdb清屏

gdb清屏：CTRL+l

ctrl+a或ctrl+x可以来回切换layout模式

## gdb运行程序

在gdb中，运行程序使用r或是run命令。程序的运行，你有可能需要设置下面四方面的事。

### 1、程序运行参数。

set args 可指定运行时参数。（如：set args 10 20 30 40 50）

show args 命令可以查看设置好的运行参数。

### 2、运行环境。

path

可设定程序的运行路径。

show paths 查看程序的运行路径。

set environment varname [=value] 设置环境变量。如：set env USER=hchen

show environment [varname] 查看环境变量。

### 3、工作目录。

cd

相当于shell的cd命令。

pwd 显示当前的所在目录。

### 4、程序的输入输出。

info terminal 显示你程序用到的终端的模式。

使用重定向控制程序输出。如：run > outfile

tty命令可以指写输入输出的终端设备。如：tty /dev/ttyb

调试已运行的程序

————————

两种方法：

1、在UNIX下用ps查看正在运行的程序的PID（进程ID），然后用gdb PID格式挂接正在运行的程序。

2、先用gdb 关联上源代码，并进行gdb，在gdb中用attach命令来挂接进程的PID。并用detach来取消挂接的进程。

暂停 / 恢复程序运行

—————————

调试程序中，暂停程序运行是必须的，GDB可以方便地暂停程序的运行。你可以设置程序的在哪行停住，在什么条件下停住，在收到什么信号时停往等等。以便于你查看运行时的变量，以及运行时的流程。

当进程被gdb停住时，你可以使用info program 来查看程序的是否在运行，进程号，被暂停的原因。

## gdb查看程中的宏操作

默认编译的时候，调试过程是看不见宏的值的。编译时候需要给选项。-g3

gcc  -g3 -o test.c  test

gdb test

察看宏（macro)命令：

|  |
| --- |
| print macroname(...) |

就像察看一个普通变量一样。如果只想看宏的展开形势，就用如下命令：

|  |
| --- |
| macro expand macroname(...) |

查看展开形式

还可以用info macro macroname 查看宏定义。

|  |
| --- |
| info macro macroname |

## 打印结构体变量类型

|  |
| --- |
| ptype struct 类型名 |

## 打印寄存器组的信息

|  |
| --- |
| info reg |

## 为什么没有core文件生成呢

有时候程序down了, 但是core文件却没有生成.

core文件的生成跟你当前系统的环境设置有关系, 可以用下面的语句设置一下, 然后再运行程序便生成了core文件.

|  |
| --- |
| ulimit -c unlimited  gdb a.out core |

## gdb中next命令与step命令的区别

step命令：

单步调试如果有函数调用，则进入函数；与命令next不同的是，next是不进入调用的函数的

## gdb中退出函数命令finish

finish：退出函数

## 显示内存命令

x/FMT ADDRESS

ADDRESS:内存地址

FMT：格式，由两部分组成fmt字符和size字符

fmt字符有：o(octal),x(hex),d(decimal),u(unsigned decimal),t(binary),f(float),a(address),i(instruction),c(char),s(string)

size字符有:b(bype),h(halfword),w(word),g(giant,8 bytes)

## 让GDB记录输出信息到文件

|  |
| --- |
| set logging on    -> 打开记录功能。 |

|  |
| --- |
| set logging off -> 关闭记录功能。 |

|  |
| --- |
| show logging -> 显示记录功能中每个选项的设置。  Future logs will be written to gdb.txt.   ---> 表示log 没有打开  Curre  ntly logging to "gdb.txt".    --> 表示log 已经打开 |

下面不常用的功能，供了解  
set logging file <file name> -> 改变记录文件，默认记录文件是gdb.txt。  
set logging overwrite [on|off] -> 默认这个项目是关闭，  
也就是以添加的形式将记录信息写入文件，这样之前文件中的信息不会被覆盖掉。  
注意：如果设置这个选项的时候记录功能已经打开，需要关闭记录功能再重新打开才能起作用。  
set logging redirect [on|off] -> 设置输出信息只记录到文件不作显示。  
默认这个项目是关闭的，GDB将输出信息到终端和记录文件。如果打开这个项目，GDB将只输出信息到记录文件。  
注意：如果设置这个选项的时候记录功能已经打开，需要关闭记录功能再重新打开才能起作用。  
注意：记录功能只能记录GDB的输出信息，被调试程序的输出信息仍然会输出到终端。

## gdb 调试运行的进程

gdb -p <进程号>

或者:

gdb

attach <进程号>

## gdb 调试跟踪多进程程序

gdb只能跟踪一个进程（默认是跟踪父进程），而不能同时跟踪多个进程，

可以设置gdb跟踪父进程还是子进程, 命令如下:

set follow-fork-mode parent 跟踪父进程, 默认

set follow-fork-mode child  跟踪子进程

## gdb暂停程序

在gdb中，我们可以有以下几种暂停方式：断点（BreakPoint）、观察点（WatchPoint）、捕捉点（CatchPoint）、信号（Signals）、线程停止（Thread Stops）。如果要恢复程序运行，可以使用c或是continue命令。

### 一、设置断点（BreakPoint）

我们用break命令来设置断点。正面有几点设置断点的方法：

break

在进入指定函数时停住。C++中可以使用class::function或function(type,type)格式来指定函数名。

break

在指定行号停住。

break +offset

break -offset

在当前行号的前面或后面的offset行停住。offiset为自然数。

break filename:linenum

在源文件filename的linenum行处停住。

break filename:function

在源文件filename的function函数的入口处停住。

break \*address

在程序运行的内存地址处停住。

break

break命令没有参数时，表示在下一条指令处停住。

break ... if

...可以是上述的参数，condition表示条件，在条件成立时停住。比如在循环境体中，可以设置break if i=100，表示当i为100时停住程序。

查看断点时，可使用info命令，如下所示：（注：n表示断点号）

info breakpoints [n]

info break [n]

### 二、设置观察点（WatchPoint）

观察点一般来观察某个表达式（变量也是一种表达式）的值是否有变化了，如果有变化，马上停住程序。我们有下面的几种方法来设置观察点：

watch

为表达式（变量）expr设置一个观察点。一量表达式值有变化（写）时，马上停住程序。

rwatch

当表达式（变量）expr被读时，停住程序。

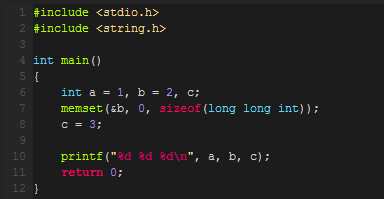
awatch

当表达式（变量）的值被读或被写时，停住程序。

info watchpoints

列出当前所设置了的所有观察点。

首先来看以下一段简单的代码



显然，第7行代码是有问题，那么这个错误的memset会造成什么后果呢？

  我们运行以下两个指令看下程序的输出：

   g++ -g main.c -o main.o

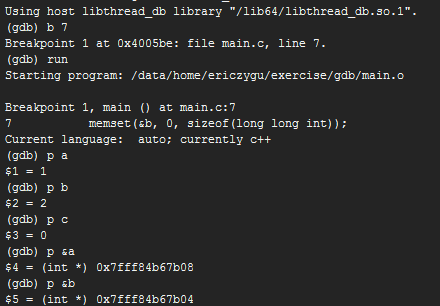
   ./main.o

  程序的输出是0 0 3，变量a的值因为memset被错误地修改了。

因为这是个很短的程序，所以我们能很轻松地看出第7行的代码是有问题的。这里的memset会越界，错误地修改了a的值。在实际情况下，当我们发现某个变量的值不符合预期时，一般的做法是先查下这个变量的引用，找到对该变量有写操作的地方(在本例中，对变量a的写操作只有一处，即第6行)。当我们发现所有的写操作和逻辑都不会产生该非法值时，可以认定程序中有越界的情况。越界的情况在实际项目中是非常令人头痛的。一是问题的根源难以定位：在本例中异常的数据是a，但其根本原因是对b操作不当造成的。二是越界之后程序的行为是未定义的，而除了回档之外也找不到到更好的方法来还原数据。

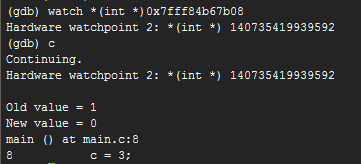
  在开发环境下，我们可以使用gdb来辅助定位越界这一问题，这里用的是watch命令。

  我们将断点设在第7行，并运行程序。可以分别看下a，b，c的值，可以看到这个时候三个变量的值都是正常的(实际这个时候变量c的值也是未定义的，看编译器怎么处理)。我们再分别打印变量a和变量b的地址，可以看到a的地址比b大4 。(回忆下，一个int型变量占4个字节，而栈上的内存是从大到小分配的)。



这个我们使用watch监控变量a值得变量，一种做法是直接watch a，我的习惯是watch地址，watch地址的方法更加具有普适性一些。

具体的指令是 watch \*(int \*)0x7fff84b67b08。然后我们继续执行程序看会在哪一步停下来，



程序在执行到第8行时发现watch的一段内存有变化，旧的值是1，新的值是0。这个时候我们回去看程序，就能发现是第7行这个memset错误地清空了a的内存空间。

### 三、设置捕捉点（CatchPoint）

你可设置捕捉点来补捉程序运行时的一些事件。如：载入共享库（动态链接库）或是C++的异常。设置捕捉点的格式为：

catch

当event发生时，停住程序。event可以是下面的内容：

1、throw 一个C++抛出的异常。（throw为关键字）

2、catch 一个C++捕捉到的异常。（catch为关键字）

3、exec 调用系统调用exec时。（exec为关键字，目前此功能只在HP-UX下有用）

4、fork 调用系统调用fork时。（fork为关键字，目前此功能只在HP-UX下有用）

5、vfork 调用系统调用vfork时。（vfork为关键字，目前此功能只在HP-UX下有用）

6、load 或 load 载入共享库（动态链接库）时。（load为关键字，目前此功能只在HP-UX下有用）

7、unload 或 unload 卸载共享库（动态链接库）时。（unload为关键字，目前此功能只在HP-UX下有用）

tcatch

只设置一次捕捉点，当程序停住以后，应点被自动删除。

### 四、维护停止点

上面说了如何设置程序的停止点，GDB中的停止点也就是上述的三类。在GDB中，如果你觉得已定义好的停止点没有用了，你可以使用delete、clear、disable、enable这几个命令来进行维护。

#### clear

|  |
| --- |
| clear 断点行号  clear 断点函数 |

1、清除所有设置在指定行上的停止点。

2、清除所有设置在函数上的停止点。

#### delete

|  |
| --- |
| delete [breakpoints] [range...] |

删除指定的断点，breakpoints为断点号。如果不指定断点号，则表示删除所有的断点。range 表示断点号的范围（如：3-7）。其简写命令为d。

#### disable

比删除更好的一种方法是disable停止点，disable了的停止点，GDB不会删除，当你还需要时，enable即可，就好像回收站一样。

|  |
| --- |
| disable [breakpoints] [range...] |

disable所指定的停止点，breakpoints为停止点号。如果什么都不指定，表示disable所有的停止点。简写命令是dis.

#### enable

|  |
| --- |
| enable [breakpoints] [range...] |

enable所指定的停止点，breakpoints为停止点号。

enable [breakpoints] once range...

enable所指定的停止点一次，当程序停止后，该停止点马上被GDB自动disable。

enable [breakpoints] delete range...

enable所指定的停止点一次，当程序停止后，该停止点马上被GDB自动删除。

### 五、停止条件维护

前面在说到设置断点时，我们提到过可以设置一个条件，当条件成立时，程序自动停止，这是一个非常强大的功能，这里，我想专门说说这个条件的相关维护命令。一般来说，为断点设置一个条件，我们使用if关键词，后面跟其断点条件。并且，条件设置好后，我们可以用condition命令来修改断点的条件。（只有break和watch命令支持if，catch目前暂不支持if）

break xx if condition

修改断点号为bnum的停止条件为expression。

condition

清除断点号为bnum的停止条件。

还有一个比较特殊的维护命令ignore，你可以指定程序运行时，忽略停止条件几次。

ignore

表示忽略断点号为bnum的停止条件count次。

### 六、为停止点设定运行命令

我们可以使用GDB提供的command命令来设置停止点的运行命令。也就是说，当运行的程序在被停止住时，我们可以让其自动运行一些别的命令，这很有利行自动化调试。对基于GDB的自动化调试是一个强大的支持。

commands [bnum]

... command-list ...

end

为断点号bnum指写一个命令列表。当程序被该断点停住时，gdb会依次运行命令列表中的命令。

例如：

break foo if x>0

commands

printf "x is %d\n",x

continue

end

断点设置在函数foo中，断点条件是x>0，如果程序被断住后，也就是，一旦x的值在foo函数中大于0，GDB会自动打印出x的值，并继续运行程序。

如果你要清除断点上的命令序列，那么只要简单的执行一下commands命令，并直接在打个end就行了。

### 七、断点菜单

在C++中，可能会重复出现同一个名字的函数若干次（函数重载），在这种情况下，break 不能告诉GDB要停在哪个函数的入口。当然，你可以使用break 也就是把函数的参数类型告诉GDB，以指定一个函数。否则的话，GDB会给你列出一个断点菜单供你选择你所需要的断点。你只要输入你菜单列表中的编号就可以了。如：

(gdb) b String::after

[0] cancel

[1] all

[2] file:String.cc; line number:867

[3] file:String.cc; line number:860

[4] file:String.cc; line number:875

[5] file:String.cc; line number:853

[6] file:String.cc; line number:846

[7] file:String.cc; line number:735

> 2 4 6

Breakpoint 1 at 0xb26c: file String.cc, line 867.

Breakpoint 2 at 0xb344: file String.cc, line 875.

Breakpoint 3 at 0xafcc: file String.cc, line 846.

Multiple breakpoints were set.

Use the "delete" command to delete unwanted

breakpoints.

(gdb)

可见，GDB列出了所有after的重载函数，你可以选一下列表编号就行了。0表示放弃设置断点，1表示所有函数都设置断点。

### 八、恢复程序运行和单步调试

当程序被停住了，你可以用continue命令恢复程序的运行直到程序结束，或下一个断点到来。也可以使用step或next命令单步跟踪程序。

continue [ignore-count]

c [ignore-count]

fg [ignore-count]

恢复程序运行，直到程序结束，或是下一个断点到来。ignore-count表示忽略其后的断点次数。continue，c，fg三个命令都是一样的意思。

#### continue

继续运行程序直到下一个**断**

#### step

单步跟踪，如果有函数调用，他会进入该函数。进入函数的前提是，此函数被编译有debug信息。很像VC等工具中的step in。后面可以加count也可以不加，不加表示一条条地执行，加表示执行后面的count条指令，然后再停住。

#### next

同样单步跟踪，如果有函数调用，他不会进入该函数。很像VC等工具中的step over。后面可以加count也可以不加，不加表示一条条地执行，加表示执行后面的count条指令，然后再停住。

set step-mode

set step-mode on

打开step-mode模式，于是，在进行单步跟踪时，程序不会因为没有debug信息而不停住。这个参数有很利于查看机器码。

set step-mod off

关闭step-mode模式。

#### finish

运行程序，直到当前函数完成返回。并打印函数返回时的堆栈地址和返回值及参数值等信息。

#### until 或 u

当你厌倦了在一个循环体内单步跟踪时，这个命令可以运行程序直到退出循环体。

#### stepi 或 si 或nexti 或 ni

单步跟踪一条机器指令！一条程序代码有可能由数条机器指令完成，stepi和nexti可以单步执行机器指令。与之一样有相同功能的命令是 “display/i $pc” ，当运行完这个命令后，单步跟踪会在打出程序代码的同时打出机器指令（也就是汇编代码）

### 九、信号（Signals）

信号是一种软中断，是一种处理异步事件的方法。一般来说，操作系统都支持许多信号。尤其是UNIX，比较重要应用程序一般都会处理信号。UNIX定义了许多信号，比如SIGINT表示中断字符信号，也就是Ctrl+C的信号，SIGBUS表示硬件故障的信号；SIGCHLD表示子进程状态改变信号； SIGKILL表示终止程序运行的信号，等等。信号量编程是UNIX下非常重要的一种技术。

GDB有能力在你调试程序的时候处理任何一种信号，你可以告诉GDB需要处理哪一种信号。你可以要求GDB收到你所指定的信号时，马上停住正在运行的程序，以供你进行调试。你可以用GDB的handle命令来完成这一功能。

#### handle

handle SIGKILL 当程序收到SIGKILL信号时程序gdb会停下来

handle all

在GDB中定义一个信号处理。信号可以以SIG开头或不以SIG开头，可以用定义一个要处理信号的范围（如：SIGIO-SIGKILL，表示处理从SIGIO信号到SIGKILL的信号，其中包括SIGIO，SIGIOT，SIGKILL三个信号），也可以使用关键字all来标明要处理所有的信号。一旦被调试的程序接收到信号，运行程序马上会被GDB停住，以供调试。其可以是以下几种关键字的一个或多个。

nostop

当被调试的程序收到信号时，GDB不会停住程序的运行，但会打出消息告诉你收到这种信号。

stop

当被调试的程序收到信号时，GDB会停住你的程序。

print

当被调试的程序收到信号时，GDB会显示出一条信息。

noprint

当被调试的程序收到信号时，GDB不会告诉你收到信号的信息。

pass

noignore

当被调试的程序收到信号时，GDB不处理信号。这表示，GDB会把这个信号交给被调试程序会处理。

nopass

ignore

当被调试的程序收到信号时，GDB不会让被调试程序来处理这个信号。

info signals

info handle

查看有哪些信号在被GDB检测中。

### 十、线程（Thread Stops）

如果你程序是多线程的话，你可以定义你的断点是否在所有的线程上，或是在某个特定的线程。GDB很容易帮你完成这一工作。

break thread

break thread if ...

linespec指定了断点设置在的源程序的行号。threadno指定了线程的ID，注意，这个ID是GDB分配的，你可以通过“info threads”命令来查看正在运行程序中的线程信息。如果你不指定thread 则表示你的断点设在所有线程上面。你还可以为某线程指定断点条件。如：

(gdb) break frik.c:13 thread 28 if bartab > lim

当你的程序被GDB停住时，所有的运行线程都会被停住。这方便你你查看运行程序的总体情况。而在你恢复程序运行时，所有的线程也会被恢复运行。那怕是主进程在被单步调试时。

查看栈信息

—————

当程序被停住了，你需要做的第一件事就是查看程序是在哪里停住的。当你的程序调用了一个函数，函数的地址，函数参数，函数内的局部变量都会被压入“栈”（Stack）中。你可以用GDB命令来查看当前的栈中的信息。

下面是一些查看函数调用栈信息的GDB命令：

backtrace

bt

打印当前的函数调用栈的所有信息。如：

(gdb) bt

#0 func (n=250) at tst.c:6

#1 0x08048524 in main (argc=1, argv=0xbffff674) at tst.c:30

#2 0x400409ed in \_\_libc\_start\_main () from /lib/libc.so.6

从上可以看出函数的调用栈信息：\_\_libc\_start\_main --> main() --> func()

backtrace

bt

n是一个正整数，表示只打印栈顶上n层的栈信息。

backtrace <-n>

bt <-n>

-n表一个负整数，表示只打印栈底下n层的栈信息。

如果你要查看某一层的信息，你需要在切换当前的栈，一般来说，程序停止时，最顶层的栈就是当前栈，如果你要查看栈下面层的详细信息，首先要做的是切换当前栈。

frame

f

n是一个从0开始的整数，是栈中的层编号。比如：frame 0，表示栈顶，frame 1，表示栈的第二层。

up

表示向栈的上面移动n层，可以不打n，表示向上移动一层。

down

表示向栈的下面移动n层，可以不打n，表示向下移动一层。

上面的命令，都会打印出移动到的栈层的信息。如果你不想让其打出信息。你可以使用这三个命令：

select-frame 对应于 frame 命令。

up-silently 对应于 up 命令。

down-silently 对应于 down 命令。

查看当前栈层的信息，你可以用以下GDB命令：

frame 或 f

会打印出这些信息：栈的层编号，当前的函数名，函数参数值，函数所在文件及行号，函数执行到的语句。

info frame

info f

这个命令会打印出更为详细的当前栈层的信息，只不过，大多数都是运行时的内内地址。比如：函数地址，调用函数的地址，被调用函数的地址，目前的函数是由什么样的程序语言写成的、函数参数地址及值、局部变量的地址等等。如：

(gdb) info f

Stack level 0, frame at 0xbffff5d4:

eip = 0x804845d in func (tst.c:6); saved eip 0x8048524

called by frame at 0xbffff60c

source language c.

Arglist at 0xbffff5d4, args: n=250

Locals at 0xbffff5d4, Previous frame's sp is 0x0

Saved registers:

ebp at 0xbffff5d4, eip at 0xbffff5d8

info args

打印出当前函数的参数名及其值。

info locals

打印出当前函数中所有局部变量及其值。

info catch

打印出当前的函数中的异常处理信息。

## 查看源程序

### 一、显示源代码

GDB 可以打印出所调试程序的源代码，当然，在程序编译时一定要加上-g的参数，把源程序信息编译到执行文件中。不然就看不到源程序了。当程序停下来以后， GDB会报告程序停在了那个文件的第几行上。你可以用list命令来打印程序的源代码。还是来看一看查看源代码的GDB命令吧。

#### list linenum

显示程序第linenum行的周围的源程序。

#### list function

显示函数名为function的函数的源程序。

#### list -

显示当前行前面的源程序。

一般是打印当前行的上5行和下5行，如果显示函数是是上2行下8行，默认是10行，当然，你也可以定制显示的范围，使用下面命令可以设置一次显示源程序的行数。

#### set listsize

设置一次显示源代码的行数。

#### show listsize

查看当前listsize的设置。

#### list命令还有下面的用法：

list ,

显示从first行到last行之间的源代码。

list ,

显示从当前行到last行之间的源代码。

list +

往后显示源代码。

一般来说在list后面可以跟以下这们的参数：

行号。

<+offset> 当前行号的正偏移量。

<-offset> 当前行号的负偏移量。

哪个文件的哪一行。

函数名。

哪个文件中的哪个函数。

<\*address> 程序运行时的语句在内存中的地址。

### 二、搜索源代码

不仅如此，GDB还提供了源代码搜索的命令：

forward-search

search

向前面搜索。

reverse-search

全部搜索。

其中，就是正则表达式，也主一个字符串的匹配模式，关于正则表达式，我就不在这里讲了，还请各位查看相关资料。

### 三、指定源文件的路径

某些时候，用-g编译过后的执行程序中只是包括了源文件的名字，没有路径名。GDB提供了可以让你指定源文件的路径的命令，以便GDB进行搜索。

directory

dir

加一个源文件路径到当前路径的前面。如果你要指定多个路径，UNIX下你可以使用“:”，Windows下你可以使用“;”。

directory

清除所有的自定义的源文件搜索路径信息。

show directories

显示定义了的源文件搜索路径。

### 四、源代码的内存

你可以使用info line命令来查看源代码在内存中的地址。info line后面可以跟“行号”，“函数名”，“文件名:行号”，“文件名:函数名”，这个命令会打印出所指定的源码在运行时的内存地址，如：

(gdb) info line tst.c:func

Line 5 of "tst.c" starts at address 0x8048456 and ends at 0x804845d .

还有一个命令（disassemble）你可以查看源程序的当前执行时的机器码，这个命令会把目前内存中的指令dump出来。如下面的示例表示查看函数func的汇编代码。

(gdb) disassemble func

Dump of assembler code for function func:

0x8048450 : push %ebp

0x8048451 : mov %esp,%ebp

0x8048453 : sub $0x18,%esp

0x8048456 : movl $0x0,0xfffffffc(%ebp)

0x804845d : movl $0x1,0xfffffff8(%ebp)

0x8048464 : mov 0xfffffff8(%ebp),%eax

0x8048467 : cmp 0x8(%ebp),%eax

0x804846a : jle 0x8048470

0x804846c : jmp 0x8048480

0x804846e : mov %esi,%esi

0x8048470 : mov 0xfffffff8(%ebp),%eax

0x8048473 : add %eax,0xfffffffc(%ebp)

0x8048476 : incl 0xfffffff8(%ebp)

0x8048479 : jmp 0x8048464

0x804847b : nop

0x804847c : lea 0x0(%esi,1),%esi

0x8048480 : mov 0xfffffffc(%ebp),%edx

0x8048483 : mov %edx,%eax

0x8048485 : jmp 0x8048487

0x8048487 : mov %ebp,%esp

0x8048489 : pop %ebp

0x804848a : ret

End of assembler dump.

## 查看运行时数据

在你调试程序时，当程序被停住时，你可以使用print命令（简写命令为p），或是同义命令inspect来查看当前程序的运行数据。print命令的格式是：

print

print /

是表达式，是你所调试的程序的语言的表达式（GDB可以调试多种编程语言），是输出的格式，比如，如果要把表达式按16进制的格式输出，那么就是/x。

### 一、表达式

print和许多GDB的命令一样，可以接受一个表达式，GDB会根据当前的程序运行的数据来计算这个表达式，既然是表达式，那么就可以是当前程序运行中的const常量、变量、函数等内容。可惜的是GDB不能使用你在程序中所定义的宏。

表达式的语法应该是当前所调试的语言的语法，由于C/C++是一种大众型的语言，所以，本文中的例子都是关于C/C++的。（而关于用GDB调试其它语言的章节，我将在后面介绍）

在表达式中，有几种GDB所支持的操作符，它们可以用在任何一种语言中。

@

是一个和数组有关的操作符，在后面会有更详细的说明。

::

指定一个在文件或是一个函数中的变量。

{}

表示一个指向内存地址的类型为type的一个对象。

### 二、程序变量

在GDB中，你可以随时查看以下三种变量的值：

1、全局变量（所有文件可见的）

2、静态全局变量（当前文件可见的）

3、局部变量（当前Scope可见的）

如果你的局部变量和全局变量发生冲突（也就是重名），一般情况下是局部变量会隐藏全局变量，也就是说，如果一个全局变量和一个函数中的局部变量同名时，如果当前停止点在函数中，用print显示出的变量的值会是函数中的局部变量的值。如果此时你想查看全局变量的值时，你可以使用“::”操作符：

file::variable

function::variable

可以通过这种形式指定你所想查看的变量，是哪个文件中的或是哪个函数中的。例如，查看文件f2.c中的全局变量x的值：

gdb) p 'f2.c'::x

当然，“::”操作符会和C++中的发生冲突，GDB能自动识别“::” 是否C++的操作符，所以你不必担心在调试C++程序时会出现异常。

另外，需要注意的是，如果你的程序编译时开启了优化选项，那么在用GDB调试被优化过的程序时，可能会发生某些变量不能访问，或是取值错误码的情况。这个是很正常的，因为优化程序会删改你的程序，整理你程序的语句顺序，剔除一些无意义的变量等，所以在GDB调试这种程序时，运行时的指令和你所编写指令就有不一样，也就会出现你所想象不到的结果。对付这种情况时，需要在编译程序时关闭编译优化。一般来说，几乎所有的编译器都支持编译优化的开关，例如，GNU 的C/C++编译器GCC，你可以使用“-gstabs”选项来解决这个问题。关于编译器的参数，还请查看编译器的使用说明文档。

### 三、数组

有时候，你需要查看一段连续的内存空间的值。比如数组的一段，或是动态分配的数据的大小。你可以使用GDB的“@”操作符，“@”的左边是第一个内存的地址的值，“@”的右边则你你想查看内存的长度。例如，你的程序中有这样的语句：

int \*array = (int \*) malloc (len \* sizeof (int));

于是，在GDB调试过程中，你可以以如下命令显示出这个动态数组的取值：

p \*array@len

@的左边是数组的首地址的值，也就是变量array所指向的内容，右边则是数据的长度，其保存在变量len中，其输出结果，大约是下面这个样子的：

(gdb) p \*array@len

$1 = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40}

如果是静态数组的话，可以直接用print数组名，就可以显示数组中所有数据的内容了。

### 四、输出格式

一般来说，GDB会根据变量的类型输出变量的值。但你也可以自定义GDB的输出的格式。例如，你想输出一个整数的十六进制，或是二进制来查看这个整型变量的中的位的情况。要做到这样，你可以使用GDB的数据显示格式：

x 按十六进制格式显示变量。

d 按十进制格式显示变量。

u 按十六进制格式显示无符号整型。

o 按八进制格式显示变量。

t 按二进制格式显示变量。

a 按十六进制格式显示变量。

c 按字符格式显示变量。

f 按浮点数格式显示变量。

(gdb) p i

$21 = 101

(gdb) p/a i

$22 = 0x65

(gdb) p/c i

$23 = 101 'e'

(gdb) p/f i

$24 = 1.41531145e-43

(gdb) p/x i

$25 = 0x65

(gdb) p/t i

$26 = 1100101

### 五、查看内存

你可以使用examine命令（简写是x）来查看内存地址中的值。x命令的语法如下所示：

x/

n、f、u是可选的参数。

n 是一个正整数，表示显示内存的长度，也就是说从当前地址向后显示几个地址的内容。

f 表示显示的格式，参见上面。如果地址所指的是字符串，那么格式可以是s，如果地十是指令地址，那么格式可以是i。

u 表示从当前地址往后请求的字节数，如果不指定的话，GDB默认是4个bytes。u参数可以用下面的字符来代替，b表示单字节，h表示双字节，w表示四字节，g表示八字节。当我们指定了字节长度后，GDB会从指内存定的内存地址开始，读写指定字节，并把其当作一个值取出来。

表示一个内存地址。

n/f/u三个参数可以一起使用。例如：

命令：x/3uh 0x54320 表示，从内存地址0x54320读取内容，h表示以双字节为一个单位，3表示三个单位，u表示按十六进制显示。

### 六、自动显示

你可以设置一些自动显示的变量，当程序停住时，或是在你单步跟踪时，这些变量会自动显示。相关的GDB命令是display。

display

display/

display/

expr是一个表达式，fmt表示显示的格式，addr表示内存地址，当你用display设定好了一个或多个表达式后，只要你的程序被停下来，GDB会自动显示你所设置的这些表达式的值。

格式i和s同样被display支持，一个非常有用的命令是：

display/i $pc

$pc是GDB的环境变量，表示着指令的地址，/i则表示输出格式为机器指令码，也就是汇编。于是当程序停下后，就会出现源代码和机器指令码相对应的情形，这是一个很有意思的功能。

下面是一些和display相关的GDB命令：

undisplay

delete display

删除自动显示，dnums意为所设置好了的自动显式的编号。如果要同时删除几个，编号可以用空格分隔，如果要删除一个范围内的编号，可以用减号表示（如：2-5）

disable display

enable display

disable和enalbe不删除自动显示的设置，而只是让其失效和恢复。

info display

查看display设置的自动显示的信息。GDB会打出一张表格，向你报告当然调试中设置了多少个自动显示设置，其中包括，设置的编号，表达式，是否enable。

### 七、设置显示选项

GDB中关于显示的选项比较多，这里我只例举大多数常用的选项。

set print address

set print address on

打开地址输出，当程序显示函数信息时，GDB会显出函数的参数地址。系统默认为打开的，如：

(gdb) f

#0 set\_quotes (lq=0x34c78 "<<", rq=0x34c88 ">>")

at input.c:530

530 if (lquote != def\_lquote)

set print address off

关闭函数的参数地址显示，如：

(gdb) set print addr off

(gdb) f

#0 set\_quotes (lq="<<", rq=">>") at input.c:530

530 if (lquote != def\_lquote)

show print address

查看当前地址显示选项是否打开。

set print array

set print array on

打开数组显示，打开后当数组显示时，每个元素占一行，如果不打开的话，每个元素则以逗号分隔。这个选项默认是关闭的。与之相关的两个命令如下，我就不再多说了。

set print array off

show print array

set print elements

这个选项主要是设置数组的，如果你的数组太大了，那么就可以指定一个来指定数据显示的最大长度，当到达这个长度时，GDB就不再往下显示了。如果设置为0，则表示不限制。

show print elements

查看print elements的选项信息。

set print null-stop

如果打开了这个选项，那么当显示字符串时，遇到结束符则停止显示。这个选项默认为off。

set print pretty on

如果打开printf pretty这个选项，那么当GDB显示结构体时会比较漂亮。如：

$1 = {

next = 0x0,

flags = {

sweet = 1,

sour = 1

},

meat = 0x54 "Pork"

}

set print pretty off

关闭printf pretty这个选项，GDB显示结构体时会如下显示：

$1 = {next = 0x0, flags = {sweet = 1, sour = 1}, meat = 0x54 "Pork"}

show print pretty

查看GDB是如何显示结构体的。

set print sevenbit-strings

设置字符显示，是否按“\nnn”的格式显示，如果打开，则字符串或字符数据按\nnn显示，如“\065”。

show print sevenbit-strings

查看字符显示开关是否打开。

set print union

设置显示结构体时，是否显式其内的联合体数据。例如有以下数据结构：

typedef enum {Tree, Bug} Species;

typedef enum {Big\_tree, Acorn, Seedling} Tree\_forms;

typedef enum {Caterpillar, Cocoon, Butterfly}

Bug\_forms;

struct thing {

Species it;

union {

Tree\_forms tree;

Bug\_forms bug;

} form;

};

struct thing foo = {Tree, {Acorn}};

当打开这个开关时，执行 p foo 命令后，会如下显示：

$1 = {it = Tree, form = {tree = Acorn, bug = Cocoon}}

当关闭这个开关时，执行 p foo 命令后，会如下显示：

$1 = {it = Tree, form = {...}}

show print union

查看联合体数据的显示方式

set print object

在C++中，如果一个对象指针指向其派生类，如果打开这个选项，GDB会自动按照虚方法调用的规则显示输出，如果关闭这个选项的话，GDB就不管虚函数表了。这个选项默认是off。

show print object

查看对象选项的设置。

set print static-members

这个选项表示，当显示一个C++对象中的内容是，是否显示其中的静态数据成员。默认是on。

show print static-members

查看静态数据成员选项设置。

set print vtbl

当此选项打开时，GDB将用比较规整的格式来显示虚函数表时。其默认是关闭的。

show print vtbl

查看虚函数显示格式的选项。

### 八、历史记录

当你用GDB的print查看程序运行时的数据时，你每一个print都会被GDB记录下来。GDB会以$1, $2, $3 .....这样的方式为你每一个print命令编上号。于是，你可以使用这个编号访问以前的表达式，如$1。这个功能所带来的好处是，如果你先前输入了一个比较长的表达式，如果你还想查看这个表达式的值，你可以使用历史记录来访问，省去了重复输入。

### 九、GDB环境变量

你可以在GDB的调试环境中定义自己的变量，用来保存一些调试程序中的运行数据。要定义一个GDB的变量很简单只需。使用GDB的set命令。GDB的环境变量和UNIX一样，也是以$起头。如：

set $foo = \*object\_ptr

使用环境变量时，GDB会在你第一次使用时创建这个变量，而在以后的使用中，则直接对其賦值。环境变量没有类型，你可以给环境变量定义任一的类型。包括结构体和数组。

show convenience

该命令查看当前所设置的所有的环境变量。

这是一个比较强大的功能，环境变量和程序变量的交互使用，将使得程序调试更为灵活便捷。例如：

set $i = 0

print bar[$i++]->contents

于是，当你就不必，print bar[0]->contents, print bar[1]->contents地输入命令了。输入这样的命令后，只用敲回车，重复执行上一条语句，环境变量会自动累加，从而完成逐个输出的功能。

### 十、查看寄存器

要查看寄存器的值，很简单，可以使用如下命令：

info registers

查看寄存器的情况。（除了浮点寄存器）

info all-registers

查看所有寄存器的情况。（包括浮点寄存器）

info registers

查看所指定的寄存器的情况。

寄存器中放置了程序运行时的数据，比如程序当前运行的指令地址（ip），程序的当前堆栈地址（sp）等等。你同样可以使用print命令来访问寄存器的情况，只需要在寄存器名字前加一个$符号就可以了。如：p $eip。

## 改变程序的执行

一旦使用GDB挂上被调试程序，当程序运行起来后，你可以根据自己的调试思路来动态地在GDB中更改当前被调试程序的运行线路或是其变量的值，这个强大的功能能够让你更好的调试你的程序，比如，你可以在程序的一次运行中走遍程序的所有分支。

### 一、修改变量值

修改被调试程序运行时的变量值，在GDB中很容易实现，使用GDB的print命令即可完成。如：

(gdb) print x=4

x=4这个表达式是C/C++的语法，意为把变量x的值修改为4，如果你当前调试的语言是Pascal，那么你可以使用Pascal的语法：x:=4。

在某些时候，很有可能你的变量和GDB中的参数冲突，如：

(gdb) whatis width

type = double

(gdb) p width

$4 = 13

(gdb) set width=47

Invalid syntax in expression.

因为，set width是GDB的命令，所以，出现了“Invalid syntax in expression”的设置错误，此时，你可以使用set var命令来告诉GDB，width不是你GDB的参数，而是程序的变量名，如：

(gdb) set var width=47

另外，还可能有些情况，GDB并不报告这种错误，所以保险起见，在你改变程序变量取值时，最好都使用set var格式的GDB命令。

### 二、跳转执行

一般来说，被调试程序会按照程序代码的运行顺序依次执行。GDB提供了乱序执行的功能，也就是说，GDB可以修改程序的执行顺序，可以让程序执行随意跳跃。这个功能可以由GDB的jump命令来完：

jump

指定下一条语句的运行点。可以是文件的行号，可以是file:line格式，可以是+num这种偏移量格式。表式着下一条运行语句从哪里开始。

jump

这里的 是代码行的内存地址。

注意，jump命令不会改变当前的程序栈中的内容，所以，当你从一个函数跳到另一个函数时，当函数运行完返回时进行弹栈操作时必然会发生错误，可能结果还是非常奇怪的，甚至于产生程序Core Dump。所以最好是同一个函数中进行跳转。

熟悉汇编的人都知道，程序运行时，有一个寄存器用于保存当前代码所在的内存地址。所以，jump命令也就是改变了这个寄存器中的值。于是，你可以使用“set $pc”来更改跳转执行的地址。如：

set $pc = 0x485

### 三、产生信号量

使用singal命令，可以产生一个信号量给被调试的程序。如：中断信号Ctrl+C。这非常方便于程序的调试，可以在程序运行的任意位置设置断点，并在该断点用GDB产生一个信号量，这种精确地在某处产生信号非常有利程序的调试。

语法是：signal ，UNIX的系统信号量通常从1到15。所以取值也在这个范围。

signal命令和shell的kill命令不同，系统的kill命令发信号给被调试程序时，是由GDB截获的，而signal命令所发出一信号则是直接发给被调试程序的。

### 四、强制函数返回

如果你的调试断点在某个函数中，并还有语句没有执行完。你可以使用return命令强制函数忽略还没有执行的语句并返回。

return

return

使用return命令取消当前函数的执行，并立即返回，如果指定了，那么该表达式的值会被认作函数的返回值。

### 五、强制调用函数

call

表达式中可以一是函数，以此达到强制调用函数的目的。并显示函数的返回值，如果函数返回值是void，那么就不显示。

另一个相似的命令也可以完成这一功能——print，print后面可以跟表达式，所以也可以用他来调用函数，print和call的不同是，如果函数返回void，call则不显示，print则显示函数返回值，并把该值存入历史数据中。

## 在不同语言中使用GDB

——————————

GDB支持下列语言：C, C++, Fortran, PASCAL, Java, Chill, assembly, 和 Modula-2。一般说来，GDB会根据你所调试的程序来确定当然的调试语言，比如：发现文件名后缀为“.c”的，GDB会认为是C程序。文件名后缀为 “.C, .cc, .cp, .cpp, .cxx, .c++”的，GDB会认为是C++程序。而后缀是“.f, .F”的，GDB会认为是Fortran程序，还有，后缀为如果是“.s, .S”的会认为是汇编语言。

也就是说，GDB会根据你所调试的程序的语言，来设置自己的语言环境，并让GDB的命令跟着语言环境的改变而改变。比如一些GDB命令需要用到表达式或变量时，这些表达式或变量的语法，完全是根据当前的语言环境而改变的。例如C/C++中对指针的语法是\*p，而在Modula-2中则是p^。并且，如果你当前的程序是由几种不同语言一同编译成的，那到在调试过程中，GDB也能根据不同的语言自动地切换语言环境。这种跟着语言环境而改变的功能，真是体贴开发人员的一种设计。

下面是几个相关于GDB语言环境的命令：

### show language

查看当前的语言环境。如果GDB不能识为你所调试的编程语言，那么，C语言被认为是默认的环境。

### info frame

查看当前函数的程序语言。

### info source

查看当前文件的程序语言。

如果GDB没有检测出当前的程序语言，那么你也可以手动设置当前的程序语言。使用set language命令即可做到。

当set language命令后什么也不跟的话，你可以查看GDB所支持的语言种类：

(gdb) set language

The currently understood settings are:

local or auto Automatic setting based on source file

c Use the C language

c++ Use the C++ language

asm Use the Asm language

chill Use the Chill language

fortran Use the Fortran language

java Use the Java language

modula-2 Use the Modula-2 language

pascal Use the Pascal language

scheme Use the Scheme language

于是你可以在set language后跟上被列出来的程序语言名，来设置当前的语言环境。

# makefile

## 格式

目标（target）:依赖文件(prerequisites) ...

命令（command）

只要依赖文件（输入）时间戳比目标文件（输出）新，就会执行命令

## makefile中的自动变量

### 代表目标文件自动化变量:$@

'$@' 表示目标对象

### 代表依赖文件自动化变量

**依赖文件自动化变量分为两类**

**依赖文件列表名自动化变量用于类似将多个\*.o文件归档成静态库文件或动态库文件**

#### 依赖文件名列表$^、$+、$？

'$^' 表示规则中所的依赖文件(去重复)

‘$+’ 类似“$^”，但是它保留了依赖文件中重复出现的文件

‘$?’ 所有比目标文件更新的依赖文件列表，空格分割。如果目标是静态库文件名，代

表的是库成员（.o 文件）。

#### 单个依赖文件名:$<、$%、$\*

**‘**$<’ 规则的第一个依赖文件名

‘$%’ 当规则的目标文件是一个静态库文件时，代表静态库的一个成员名

‘$\* ‘表示文件中除后缀以外的部分。例如：“foo.c”则“$\* ”的值为：“foo ”

'$@' 表示目标对象

'$^' 表示规则中所的依赖文件

'$<' 表示规则中的第一个依赖文件

有时候依赖文件中包含有头文件,而这些头文件不需要在命令中说明，此时自动变量$<就表示第一个依赖文件

例如:

foo.o:foo.c def.h hack.h

cc -c $(CFLAGS) $< -o $@

自动化变量

模式规则中，规则的目标和依赖文件名代表了一类文件名；规则的命令是对所有这

一类文件重建过程的描述，显然，在命令中不能出现具体的文件名，否则模式规则失去

意义。那么在模式规则的命令行中该如何表示文件，将是本小节的讨论的重点。

假如你需要书写一个将.c 文件编译到.o 文件的模式规则，那么你该如何为gcc 书写

正确的源文件名？当然了，不能使用任何具体的文件名，因为在每一次执行模式规则时

源文件名都是不一样的。为了解决这个问题，就需要使用“自动化变量”，自动化变量

的取值是根据具体所执行的规则来决定的，取决于所执行规则的目标和依赖文件名。

下面对所有的自动化变量进行说明：

$@

表示规则的目标文件名。如果目标是一个文档文件（Linux中，一般称.a 文件为

文档文件，也称为静态库文件），那么它代表这个文档的文件名。在多目标模式

规则中，它代表的是哪个触发规则被执行的目标文件名。

$%

当规则的目标文件是一个静态库文件时，代表静态库的一个成员名。例如，规则

的目标是“foo.a(bar.o)”，那么，“ $%”的值就为“bar.o”，“ $@ ”的值为“foo.a”。

如果目标不是静态库文件，其值为空。

$<

规则的第一个依赖文件名。如果是一个目标文件使用隐含规则来重建，则它代表

由隐含规则加入的第一个依赖文件。

$?

所有比目标文件更新的依赖文件列表，空格分割。如果目标是静态库文件名，代

表的是库成员（.o 文件）。

自动化变量“$?”在显式规则中也是非常有用的，使用它规则可以指定只对更新

以后的依赖文件进行操作。例如，静态库文件“libN.a ”，它由一些.o 文件组成。这个规

则实现了只将更新后的.o 文件加入到库中：

     lib: foo.o bar.o lose.o win.o

             ar r lib $?

$^

规则的所有依赖文件列表，使用空格分隔。如果目标是静态库文件，它所代表的

只能是所有库成员（.o 文件）名。一个文件可重复的出现在目标的依赖中，变量

“$^”只记录它的一次引用情况。就是说变量“$^”会去掉重复的依赖文件。

$+

类似“$^”，但是它保留了依赖文件中重复出现的文件。主要用在程序链接时库

的交叉引用场合。

$\*

在模式规则和静态模式规则中，代表“茎”。“茎”是目标模式中“% ”所代表的

部分（当文件名中存在目录时，“茎”也包含目录（斜杠之前）部分，可参考  10.5.4

模式的匹配  一小节）。例如：文件“dir/a.foo.b”，当目标的模式为“a.%.b ”时，

“$\* ”的值为“dir/a.foo ”。“茎”对于构造相关文件名非常有用。

自动化变量“$\* ”需要两点说明：

1 对于一个明确指定的规则来说不存在“茎”，这种情况下“$\* ”的含义发

生改变。此时，如果目标文件名带有一个可识别的后缀（参考  10.7 后

缀规则  一节），那么“$\* ”表示文件中除后缀以外的部分。例如：“foo.c”

则“$\* ”的值为：“foo ”，因为.c 是一个可识别的文件后缀名。GUN make

对明确规则的这种奇怪的处理行为是为了和其它版本的make兼容。通

常，在除静态规则和模式规则以外，明确指定目标文件的规则中应该避

免使用这个变量。

2  当明确指定文件名的规则中目标文件名包含不可识别的后缀时，此变量

为空。

以上罗列的自动量变量中。其中有四个在规则中代表文件名（$@ 、$<、$%、$\* ）。

而其它三个的($^,$+,$?)在规则中代表一个文件名列表。GUN make 中，还可以通过这七个自动化

变量来获取一个完整文件名中的目录部分和具体文件名部分。在这些变量中加入“D”

或者“F”字符就形成了一系列变种的自动环变量。这些变量会出现在以前版本的make

中，在当前版本的make中，可以使用“dir”或者“notdir”函数来实现同样的功能（可

参考  8.3  文件名处理函数  一节）。

$(@D)

表示目标文件的目录部分（不包括斜杠）。如果“$@ ”是“dir/foo.o ”，那么“$(@D) ”

的值为“dir”。如果“$@ ”不存在斜杠，其值就是“. ”（当前目录）。注意它和 函

数“dir”的区别！

$(@F)

目标文件的完整文件名中除目录以外的部分（实际文件名）。如果“$@ ”为

“dir/foo.o ”，那么“$(@F) ”只就是“foo.o”。“$(@F) ”等价于函数“$(notdir

$@) ”。

$(\*D)

$(\*F)

分别代表目标“茎”中的目录部分和文件名部分。

$(%D)

$(%F)

当以如“archive(member) ”形式静态库为目标时，分别表示库文件成员

“member”名中的目录部分和文件名部分。它仅对这种形式的规则目标有效。

$(<="">

$(<="">

分别表示规则中第一个依赖文件的目录部分和文件名部分。

$(^D)

$(^F)

分别表示所有依赖文件的目录部分和文件部分（不存在同一文件）。

$(+D)

$(+F)

分别表示所有依赖文件的目录部分和文件部分（可存在重复文件）。

$(?D)

$(?F)

分别表示被更新的依赖文件的目录部分和文件名部分

在讨论自动化变量时，为了和普通变量（如：“CFLAGS ”）区别，我们直接使用了

“$<”的形式。这种形式仅仅是为了和普通变量进行区别，没有别的目的。其实对于

自动化变量和普通变量一样，代表规则第一个依赖文件名的变量名实际上是“< ”，我

们完全可以使用“$(<) ”来替代“$<”。但是在引用自动化变量时通常的做法是“$<”，

因为自动化变量本身是一个特殊字符。

GUN make同时支持“Sysv”特性，允许在规则的依赖列表中使用特殊的变量引

用（一般的自动化变量只能在规则的命令行中被引用）“$$@”、“$$(@D)”和“$$(@F)”

（注意：要使用“$$”），它们分别代表了“目标的完整文件名”、“目标文件名中的目

录部分”和“目标的实际文件名部分”。这三个特殊的变量只能用在明确指定目标文件

名的规则中或者是静态模式规则中，不用于隐含规则中。另外Sysv make 和GNU make

对规则依赖的处理也不尽相同。Sysv make对规则的依赖进行两次替换展开，而GUN

make对依赖列表的处理只有一次，对其中的变量和函数引用直接进行展开。

自动化变量的这个古怪的特性完全是为了兼容Sysv  版本的makefile文件。在使用

GNU make 时可以不考虑这个，也可以在Makefile中使用伪目标“.POSIX ”来禁止这一

特性

## 双后缀规则

.原后缀.目标后缀:

编译命令

例如：

.c.s:

$(CC) $(CFLAS) \

-nostdinc -Iinclude -S -o $\*.s $<

此时$<表示'\*.c'文件名

# redis

## 常用命令

dbsize

info 可查看数据是否过期

flushall清空数据

ttl + key记录的存活时间，单位为秒

# Shell脚本1

## [ps命令详解](http://www.cnblogs.com/wangkangluo1/archive/2011/09/23/2185938.html)

ps命令详解

有时候系统管理员可能只关心现在系统中运行着哪些程序，而不想知道有哪些进程在运行。由于一个应用程序可能需要启动多个进程。所以在同等情况下，进程的数 量要比程序多的多。为此从阅读方面考虑，管理员需要知道系统中运行的具体程序。要实现这个需求的话，就需要利用命令ps来帮忙。

要对进程进行监测和控制，首先必须要了解当前进程的情况，也就是需要查看当前进程，而 ps 命令就是最基本同时也是非常强大的进程查看命令。使用该命令可以确定有哪些进程正在运行和运行的状态、进程是否结束、进程有没有僵死、哪些进程占用了过多 的资源等等。总之大部分信息都是可以通过执行该命令得到的。

名称：ps

使用权限：所有使用者

使用方式：ps [options] [--help]

说明：显示瞬间行程 (process) 的动态

参数：ps的参数非常多, 在此仅列出几个常用的参数并大略介绍含义

-A 列出所有的进程

-w 显示加宽可以显示较多的资讯

-au 显示较详细的资讯

-aux 显示所有包含其他使用者的行程

############################################################

常用参数：

-A 显示所有进程（等价于-e）(utility)

-a 显示一个终端的所有进程，除了会话引线

-N 忽略选择。

-d 显示所有进程，但省略所有的会话引线(utility)

-x 显示没有控制终端的进程，同时显示各个命令的具体路径。dx不可合用。（utility）

-p pid 进程使用cpu的时间

-u uid or username 选择有效的用户id或者是用户名

-g gid or groupname 显示组的所有进程。

U username 显示该用户下的所有进程，且显示各个命令的详细路径。如:ps U zhang;(utility)

-f 全部列出，通常和其他选项联用。如：ps -fa or ps -fx and so on.

-l 长格式（有F,wchan,C 等字段）

-j 作业格式

-o 用户自定义格式。

v 以虚拟存储器格式显示

s 以信号格式显示

-m 显示所有的线程

-H 显示进程的层次(和其它的命令合用，如：ps -Ha)（utility）

e 命令之后显示环境（如：ps -d e; ps -a e）(utility)

h 不显示第一行

############################################################

ps命令常用用法（方便查看系统进程）

1）ps a 显示现行终端机下的所有程序，包括其他用户的程序。

2）ps -A 显示所有进程。

3）ps c 列出程序时，显示每个程序真正的指令名称，而不包含路径，参数或常驻服务的标示。

4）ps -e 此参数的效果和指定"A"参数相同。

5）ps e 列出程序时，显示每个程序所使用的环境变量。

6）ps f 用ASCII字符显示树状结构，表达程序间的相互关系。

7）ps -H 显示树状结构，表示程序间的相互关系。

8）ps -N 显示所有的程序，除了执行ps指令终端机下的程序之外。

9）ps s 采用程序信号的格式显示程序状况。

10）ps S 列出程序时，包括已中断的子程序资料。

11）ps -t<终端机编号> 　指定终端机编号，并列出属于该终端机的程序的状况。

12）ps u 　以用户为主的格式来显示程序状况。

13）ps x 　显示所有程序，不以终端机来区分。

最常用的方法是ps -aux,然后再利用一个管道符号导向到grep去查找特定的进程,然后再对特定的进程进行操作。

############################################################

运行 ps aux 的到如下信息：

root:# ps aux

USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND

smmsp 3521 0.0 0.7 6556 1616 ? Ss 20:40 0:00 sendmail: Queue runner@01:00:00 f

root 3532 0.0 0.2 2428 452 ? Ss 20:40 0:00 gpm -m /dev/input/mice -t imps2

htt 3563 0.0 0.0 2956 196 ? Ss 20:41 0:00 /usr/sbin/htt -retryonerror 0

htt 3564 0.0 1.7 29460 3704 ? Sl 20:41 0:00 htt\_server -nodaemon

root 3574 0.0 0.4 5236 992 ? Ss 20:41 0:00 crond

xfs 3617 0.0 1.3 13572 2804 ? Ss 20:41 0:00 xfs -droppriv -daemon

root 3627 0.0 0.2 3448 552 ? SNs 20:41 0:00 anacron -s

root 3636 0.0 0.1 2304 420 ? Ss 20:41 0:00 /usr/sbin/atd

dbus 3655 0.0 0.5 13840 1084 ? Ssl 20:41 0:00 dbus-daemon-1 --system

Head标头：

USER 用户名

UID 用户ID（User ID）

PID 进程ID（Process ID）

PPID 父进程的进程ID（Parent Process id）

SID 会话ID（Session id）

%CPU 进程的cpu占用率

%MEM 进程的内存占用率

VSZ 进程所使用的虚存的大小（Virtual Size）

RSS 进程使用的驻留集大小或者是实际内存的大小，Kbytes字节。

TTY 与进程关联的终端（tty）

STAT 进程的状态：进程状态使用字符表示的（STAT的状态码）

R 运行 Runnable (on run queue) 正在运行或在运行队列中等待。

S 睡眠 Sleeping 休眠中, 受阻, 在等待某个条件的形成或接受到信号。

I 空闲 Idle

Z 僵死 Zombie（a defunct process) 进程已终止, 但进程描述符存在, 直到父进程调用wait4()系统调用后释放。

D 不可中断 Uninterruptible sleep (ususally IO) 收到信号不唤醒和不可运行, 进程必须等待直到有中断发生。

T 终止 Terminate 进程收到SIGSTOP, SIGSTP, SIGTIN, SIGTOU信号后停止运行运行。

P 等待交换页

W 无驻留页 has no resident pages 没有足够的记忆体分页可分配。

X 死掉的进程

< 高优先级进程 高优先序的进程

N 低优先 级进程 低优先序的进程

L 内存锁页 Lock 有记忆体分页分配并缩在记忆体内

s 进程的领导者（在它之下有子进程）；

l 多进程的（使用 CLONE\_THREAD, 类似 NPTL pthreads）

+ 位于后台的进程组

START 进程启动时间和日期

TIME 进程使用的总cpu时间

COMMAND 正在执行的命令行命令

NI 优先级(Nice)

PRI 进程优先级编号(Priority)

WCHAN 进程正在睡眠的内核函数名称；该函数的名称是从/root/system.map文件中获得的。

FLAGS 与进程相关的数字标识

############################################################

例子:

查看当前系统进程的uid,pid,stat,pri, 以uid号排序.

ps -eo pid,stat,pri,uid –sort uid

查看当前系统进程的user,pid,stat,rss,args, 以rss排序.

ps -eo user,pid,stat,rss,args –sort rss

############################################################

ps 为我们提供了进程的一次性的查看，它所提供的查看结果并不动态连续的；如果想对进程时间监控，应该用 top 工具。

1、ps 的参数说明：

ps 提供了很多的选项参数，常用的有以下几个：

l 长格式输出；

u 按用户名和启动时间的顺序来显示进程；

j 用任务格式来显示进程；

f 用树形格式来显示进程；

a 显示所有用户的所有进程（包括其它用户）；

x 显示无控制终端的进程；

r 显示运行中的进程；

ww 避免详细参数被截断；

我们常用的选项是组合是 aux 或 lax，还有参数 f 的应用。

2、ps aux 或 lax 输出的解释：

USER 进程的属主；

PID 进程的ID；

PPID 父进程；

%CPU 进程占用的CPU百分比；

%MEM 占用内存的百分比；

NI 进程的NICE值，数值大，表示较少占用CPU时间；

VSZ 进程虚拟大小；

RSS 驻留中页的数量；

TTY 终端ID

STAT 进程状态（有以下几种）

D 无法中断的休眠状态（通常 IO 的进程）；

R 正在运行可中在队列中可过行的；

S 处于休眠状态；

T 停止或被追踪；

W 进入内存交换（从内核2.6开始无效）；

X 死掉的进程（从来没见过）；

Z 僵尸进程；

< 优先级高的进程

N 优先级较低的进程

L 有些页被锁进内存；

s 进程的领导者（在它之下有子进程）；

l 多进程的（使用 CLONE\_THREAD, 类似 NPTL pthreads）；

+ 位于后台的进程组；

WCHAN 正在等待的进程资源；

START 启动进程的时间；

TIME 进程消耗CPU的时间；

COMMAND 命令的名称和参数；

3、应用举例：

[root@localhost ~]# ps -aux |more

可以用 | 管道和 more 连接起来分页查看。

[root@localhost ~]# ps -aux > ps001.txt

把结果输出到ps001.txt文本中并保存。

[root@localhost ~]# more ps001.txt

这里是把所有进程显示出来，并输出到ps001.txt文件，然后再通过more 来分页查看。

4、kill 终止（杀死）进程，有十几种控制进程的方法，下面是一些常用的方法:

[root@localhost ~]#kill -STOP [pid]

发送SIGSTOP (17,19,23)停止一个进程，而并不消灭这个进程。

[root@localhost ~]#kill -CONT [pid]

发送SIGCONT (19,18,25)重新开始一个停止的进程。

[root@localhost ~]#kill -KILL [pid]

发送SIGKILL (9)强迫进程立即停止，并且不实施清理操作。

[root@localhost ~]#kill -9 -1

终止你拥有的全部进程。

SIGKILL 和 SIGSTOP 信号不能被捕捉、封锁或者忽略，但是，其它的信号可以。所以这是你的终极武器。

############################################################

利用ps命令向管理员报告执行中的程序：

有时候系统管理员可能只关心现在系统中运行着哪些程序，而不想知道有哪些进程在运行。由于一个应用程序可能需要启动多个进程。所在在同等情况下，进程的数 量要比程序多的多。为此从阅读方面考虑，管理员需要知道系统中运行的具体程序。要实现这个需求的话，就需要利用命令ps来帮忙。

一、ps命令显示结果的含义。

当需要查看系统中执行的程序时，虽然ps 命令不是唯一的命令，但绝对是使用的最频繁的命令。如下图所示，就是执行ps命令后显示的结果。

在命令行中输入命令ps，就可以显示系统中当前运行的所有应用程序。如上图所示，如果输入ps命令，其显示结果主要有四部分内容。首先是PID，这是程序 的ID号。才作系统就是利用这个ID号来唯一的标识应用程序，而不是利用命令来辨认。当需要强制关闭应用程序时，就需要用到这个PID号码。其次是 TTY，这个字段表示用户使用的终端代码。Pts表示用户是采用远程登录的。第三个参数TIME表示这个程序所消耗的CPU时间，注意这个时间不是程序开 始运行的时间。最后一个参数CMD就表示程序的名字。

二、让系统报告详细的信息。

在使用ps命令时，如果不采用任何的可选项，则其显示的信息是非常有限的，而且往往只显示当前用户所运行的程序。当系统管理员需要知道应用程序更加详细的 运行信息时，如想要知道这个应用程序内存、CPU的占用率情况时，那么就需要加入一些可选项。如系统管理员需要一并查看其它用户所执行的应用程序时，就需 要在这个命令后面采用可选项-al。如此的话，系统会列出系统中所有用户运行的所有程序。如想要知道某个程序CPU与内存的使用情况，而不是只简单的现实 其CPU的使用时间，那么就需要在这个命令后面加入参数-l ，即使用ps –l命令可以让系统显示出应用程序的详细运行信息。关于上面各个字段所代表的含义，大家若有兴趣的话可以查看相关的帮助。一般来说，系统管理员关心的只是 程序的PID号码、内存与CPU的使用率、命令的名字、用户采用的终端等等。其它的信息对于系统管理员来说价值不是很大。

三、查看后台运行的程序。

默认情况下，ps命令只显示前台运行的程序，而不会显示后台运行的程序。但是并非所有的程序都是在前台运行。正常情况下，隐藏在后台运行的程序数量要比前 台运行的程序多的多。如随着操作系统启动而启动的不少系统自带程序，其运行的方式都是后台运行。而且有时候，系统出现问题往往是由于后台程序所造成的。如 常见的木马等程序都是在后台所运行的。为此系统管理员相对来说，更加想要知道在后台运行着哪些程序。

如果要查看后台运行的程序，那就比较复杂一点。因为在不同版本的Linux操作系统中，要显示后台进程其所采用的可选项是不同的。如在红帽子 Linux操作系统中，其实采用参数的形式而不是可选项。即采用ps aux命令可以显示出所有的应用程序(包括前台与后台的)。参数与可选项的差异主要在前面又没有这个-符号。如果带有这个符号的就表示这是一个可选项。而 如果不带的，就表示这事一个参数。这个符号一般情况下可不能够省。在该写的地方没写，或者不需要些的地方偏偏加上了，则系统都会提示错误信息，说找不到这 个命令。而在其它的Linux系统版本中，可能不能够识别这个aux参数。如在一些Linux操作系统版本中，需要采用-a可选项来完成这个任务。由于系 统版本之间的差异，给系统管理员带来了不少的麻烦。不过值得庆幸的是，各个操作系统版本中都有在线的帮助。如果系统管理员在使用一个新版本的操作系统时， 不知道要显示全部进程该使用哪个可选项时，可以利用ps --heip等命令来查看系统帮助。不过美中不足的是，系统在线帮助都是英文的，对系统管理员的英文水平是一个不小的考验。不过如果要作Linux系统管 理员，这点英文底子还是要有的。因为最新的Linux技术基本上都是先出来英文文档的。其实要掌握最先进的操作系统书籍，大部分的计算机书籍都是英文的。

四、对程序列表进行排序。

当运行的应用程序比较多时，系统管理员需要对应用程序进行排序。Ps命令的排序功能是比较强的。主要是因为这个命令有一个--sort参数(注意在这个参 数前面采用的是两个小横杆符号，各位读者不要以为是笔者写错了)。在这个参数后面加上系统管理员想要的排序字段，就可以进行排序了。如这个命令ps –A --sort cmd，就表示显示系统所有的应用程序，并根据程序命令来进行排序。在Linux操作系统参数中，还有一个比较麻烦的事情，就是参数大小写不同往往代表着 不同的含义。如上面这个命令，将大写字母A换成是小写字符a，则结果就完全两样了。大写字符A表示所有的应用程序，而小写字符a则表示“all w/ tty except session leaders”。两者有本质的区别。通过这个差异可以用来过滤不同终端登陆帐户所运行的应用程序。

在ps命令中，不少参数都有这种情况。运行ps --help查看这个命令的所有参数，就可以看到类似的大小写不同代表不同含义的情况还有很多。如大写字符O与小写字符o、大写字符U与小写字符u等等。 这些差异无形之中增加了系统管理员维护操作系统的难度。系统管理员要掌握这么多的参数以及参数大小写之间的差异，往往是不可能的。系统管理员除了平时要多 使用使用常用的参数来增加值观的印象，这个系统命令的在线帮助确实也是少不了的。这个系统帮助可以给管理员使用系统命令提供在线的指导。

五、报告特定程序的运行情况。

当系统中运行的程序比较多时，通过对程序名字排序可以帮助管理员找到自己所关心的程序。但是这仍然不是最简便的方式。如现在系统管理员在其它操作系统中发 现有一个叫做threadx的木马程序在系统后台运行。为此管理员需要在其它电脑上查看是否也有这个木马程序在运行。此时该如何处理呢?利用排序功能，对 程序的名字进行排序(注意不是对程序的PID进行排序，因为即使程序相同，启动的时间不同或者操作系统中已经启动程序的数量不同，这个PID号码也就不 同。也就是说这个PID号码是自动生成的)。这在一定程度上可以帮助管理员加快程序查找的速度。不过笔者认为，如果系统管理员能够使用管道符与 grep等查询命令可能会更快的找到自己所需要的应用程序信息。

如现在系统管理员若使用这个命令ps aux |grep “threadx”，会出现什么情况呢?首先系统会列出当前系统中所有运行的应用程序(包括前台运行与后台运行的)。然后将输出的结果通过管道符号|传递 给命令grep。然后命令grep就会在ps命令输出的结果中查找是否有threadx这个程序运行的信息。如果有的话，则会在窗口中显示这个程序的信 息，而过滤掉其它应用程序的运行信息。可见采用管道符跟其它查询命令结合，可以帮助系统管理员在最短的时间内找到自己想了解的应用程序的信息。

跟一些特殊符号结合还可以实现一些更加负责的功能。如跟cat命令结合使用可以统计出符合条件的程序数量。如在命令后面加入>符号，可以将输出的结 果定位到一个文件中。如跟通配符结合使用，可以查询出满足特定条件的一类程序。如还可以查询出某个程序父程序或者其子程序的运行信息等等。

## linux的hosts文件

/etc/hosts

发现ssh 无法登陆Linux的eth0网卡ip:10.0.2.15

尝试了各种方法，ssh服务重启，关闭Linux的防火墙等，发现都没用，

最后在/etc/hosts加上了一条和127.0.0.1一样的配置就可以了

## 检查Linux服务是否关闭

chkconfig

## 关闭linux防火墙

/etc/init.d/iptables stop

## linux中将秒转换成时间

date -d @1233631748

## set,env和export这三个命令的区别

set 显示当前shell的变量，包括当前用户的变量 env 显示当前用户的变量 export 显示当前导出成用户变量的shell变量 每个shell有自己特有的变量（set）显示的变量，这个和用户变量是不同的，当前用户变量和你用什么shell无关，不管你用什么shell都在，比如HOME,SHELL等这些变量，但shell自己的变量不同shell是不同的，比如BASH\_ARGC， BASH等，这些变量只有set才会显示，是bash特有的，export不加参数的时候，显示哪些变量被导出成了用户变量，因为一个shell自己的变量可以通过export “导出”变成一个用户变量。

shell变量包括两种变量

1.本shell私有的变量：通过赋值语句定义好的变量，可以通过如下方法定义shell变量

       A1="1234"

       delcare A2="2345"

2.用户的环境变量：通过export语法导出的shell私有变量，可以通过如下方法导出用户环境变量

      A1="1234"

      export A1  #先定义再导出

      export A3="34"

导出成的用户环境变量可以在所有的shell中看到

env显示用户的环境变量；

set 显示当前shell的定义的私有变量，包括用户的环境变量，按变量名称排序；

export 显示当前导出成用户变量的shell变量，并显示变量的属性(是否只读)，按变量名称排序；

declare 同set 一样，显示当前shell的定义的变量，包括用户的环境变量

## tput命令

tput 命令将通过 terminfo 数据库对您的终端会话进行初始化和操作。通过使用 tput，您可以更改几项终端功能，如移动或更改光标、更改文本属性，以及清除终端屏幕的特定区域。

### 1、设置文本属性

#### tput Color Capabilities—文本颜色属性

##### tput setab [0-7]

– Set a background color using ANSI escape

##### tput setb [0-7]

– Set a background color

##### tput setaf [0-7]

– Set a foreground color using ANSI escape

##### tput setf [0-7]

– Set a foreground color

##### Color Code for tput:

0 – Black

1 – Red

2 – Green

3 – Yellow

4 – Blue

5 – Magenta

6 – Cyan

7 – White

#### tput Text Mode Capabilities:-文本模式属性

tput bold – Set bold mode-粗体模式

tput dim – turn on half-bright mode

tput smul – begin underline mode-下划线模式

tput rmul – exit underline mode-推出下划线模式

tput rev – Turn on reverse mode

tput smso – Enter standout mode (bold on rxvt)-输入突出模式

tput rmso – Exit standout mode-退出突出模式

tput sgr0 – Turn off all attributes-关闭所有属性

#### 例子

##### 使输出的字符串有颜色，底色，加粗

#!/bin/bash

printf $(tput setaf 2; tput bold)'color show\n\n'$(tput sgr0)

for((i=0; i<=7; i++)); do

echo $(tput setaf $i)"show me the money"$(tput sgr0)

done

printf '\n'$(tput setaf 2; tput setab 0; tput bold)'background color show'$(tput sgr0)'\n\n'

for((i=0,j=7; i<=7; i++,j--)); do

echo $(tput setaf $i; tput setab $j; tput bold)"show me the money"$(tput sgr0)

done

exit 0

done

exit 0



##### 输出格式控制函数

#!/bin/bash

# $1 str print string

# $2 color 0-7 设置颜色

# $3 bgcolor 0-7 设置背景颜色

# $4 bold 0-1 设置粗体

# $5 underline 0-1 设置下划线

function format\_output(){

str=$1

color=$2

bgcolor=$3

bold=$4

underline=$5

normal=$(tput sgr0)

case "$color" in

0|1|2|3|4|5|6|7)

setcolor=$(tput setaf $color;) ;;

\*)

setcolor="" ;;

esac

case "$bgcolor" in

0|1|2|3|4|5|6|7)

setbgcolor=$(tput setab $bgcolor;) ;;

\*)

setbgcolor="" ;;

esac

if [ "$bold" = "1" ]; then

setbold=$(tput bold;)

else

setbold=""

fi

if [ "$underline" = "1" ]; then

setunderline=$(tput smul;)

else

setunderline=""

fi

printf "$setcolor$setbgcolor$setbold$setunderline$str$normal\n"

}

format\_output "Yesterday Once More" 2 5 1 1

exit 0

### ****2.光标属性****

tput clear -清屏

tput sc -保存当前光标位置

tput cup row col - 将光标移动到 row col

tput civis -光标不可见

tput cnorm -光标可见

tput rc -显示输出

## cp -r与cp -R的区别

cp -r与cp -R都可以递归拷贝目录及子目录下的文件

二者的区别如下：

cp -r 只能拷贝普通文件（以普通文件的方式拷贝）

cp -R不仅能拷贝普通文件，还可以拷贝FIFO文件(管道文件)，块设备和字符设备文件（这些文件是以重新建立的，而不是拷贝）

详见 ，man cp手册

## lsof命令入门

### 用户信息

你也可以获取各种用户的信息，以及它们在系统上正干着的事情，包括它们的网络活动、对文件的操作等。

#### 使用-u显示指定用户打开了什么

# **lsof -u daniel**

-- snipped --

Dock155 daniel txt REG 14,22798436823208/usr/lib/libicucore.A.dylib

Dock155 daniel txt REG 14,21580212823126/usr/lib/libobjc.A.dylib

Dock155 daniel txt REG 14,22934184823498/usr/lib/libstdc++.6.0.4.dylib

Dock155 daniel txt REG 14,2132008823505/usr/lib/libgcc\_s.1.dylib

Dock155 daniel txt REG 14,2212160823214/usr/lib/libauto.dylib

-- snipped --

#### 使用-u user来显示除指定用户以外的其它所有用户所做的事情

# **lsof -u ^daniel**

-- snipped --

Dock155 jim txt REG 14,22798436823208/usr/lib/libicucore.A.dylib

Dock155 jim txt REG 14,21580212823126/usr/lib/libobjc.A.dylib

Dock155 jim txt REG 14,22934184823498/usr/lib/libstdc++.6.0.4.dylib

Dock155 jim txt REG 14,2132008823505/usr/lib/libgcc\_s.1.dylib

Dock155 jim txt REG 14,2212160823214/usr/lib/libauto.dylib

-- snipped --

### 杀死指定用户所做的一切事情

可以消灭指定用户运行的所有东西，这真不错。

# **kill -9 `lsof -t -u daniel`**

### 命令和进程

可以查看指定程序或进程由什么启动，这通常会很有用，而你可以使用lsof通过名称或进程ID过滤来完成这个任务。下面列出了一些选项：

#### 使用-c查看指定的命令正在使用的文件和网络连接

# **lsof -c syslog-ng**

COMMAND PID USER FD TYPE DEVICE SIZE NODE NAME

syslog-ng 7547 root cwd DIR 3,340962/

syslog-ng 7547 root rtd DIR 3,340962/

syslog-ng 7547 root txt REG 3,31135241064970/usr/sbin/syslog-ng

-- snipped --

#### 使用-p查看指定进程ID已打开的内容

# **lsof -p 10075**

-- snipped --

sshd 10068 root mem REG 3,334808850407/lib/libnss\_files-2.4.so

sshd 10068 root mem REG 3,334924850409/lib/libnss\_nis-2.4.so

sshd 10068 root mem REG 3,326596850405/lib/libnss\_compat-2.4.so

sshd 10068 root mem REG 3,3200152509940/usr/lib/libssl.so.0.9.7

sshd 10068 root mem REG 3,346216510014/usr/lib/liblber-2.3

sshd 10068 root mem REG 3,359868850413/lib/libresolv-2.4.so

sshd 10068 root mem REG 3,31197180850396/lib/libc-2.4.so

sshd 10068 root mem REG 3,322168850398/lib/libcrypt-2.4.so

sshd 10068 root mem REG 3,372784850404/lib/libnsl-2.4.so

sshd 10068 root mem REG 3,370632850417/lib/libz.so.1.2.3

sshd 10068 root mem REG 3,39992850416/lib/libutil-2.4.so

-- snipped --

#### -t选项只返回PID

# **lsof -t -c Mail**

350

#### 文件和目录

通过查看指定文件或目录，你可以看到系统上所有正与其交互的资源——包括用户、进程等。

#### 显示与指定目录交互的所有一切

# **lsof /var/log/messages/**

COMMAND PID USER FD TYPE DEVICE SIZE NODE NAME

syslog-ng 7547 root 4w REG 3,3217309834024/var/log/messages

#### 显示与指定文件交互的所有一切

# **lsof /home/daniel/firewall\_whitelist.txt**

### 高级用法

与tcpdump类似，当你开始组合查询时，它就显示了它强大的功能。

#### 显示daniel连接到1.1.1.1所做的一切

# **lsof -u daniel -i @1.1.1.1**

bkdr 1893 daniel 3uIPv63456 TCP 10.10.1.10:1234->1.1.1.1:31337(ESTABLISHED)

#### 同时使用-t和-c选项以给进程发送 HUP 信号

# **kill -HUP `lsof -t -c sshd`**

#### lsof +L1显示所有打开的链接数小于1的文件

这通常（当不总是）表示某个攻击者正尝试通过删除文件入口来隐藏文件内容。

# **lsof +L1**

(hopefully nothing)

显示某个端口范围的打开的连接

# lsof -i @fw.google.com:2150=2180

结尾

本入门教程只是管窥了lsof功能的一斑，要查看完整参考，运行man lsof命令或查看在线版本。希望本文对你有所助益，也随时欢迎你的评论和指正。

## vi中大小写转换命令

个数+~

例如：

转换(大写转小写，小写转大写)10个字母: 10~

## ulimit -a 查看stack size的大小

ulimit -a 查看stack size的大小

## 查看当前系统的语言

echo $LANG

## 查看安装的语言包

输入locale命令，如果有zh\_CN表示已经安装了中文语言，如果没有需要下载

## ln命令

ln是linux中又一个非常重要命令，它的功能是为某一个文件在另外一个位置建立一个同步的链接.当我们需要在不同的目录，用到相同的文件时，我们不需要在每一个需要的目录下都放一个必须相同的文件，我们只要在某个固定的目录，放上该文件，然后在 其它的目录下用ln命令链接（link）它就可以，不必重复的占用磁盘空间。

### 1．命令格式：

ln [参数][源文件或目录][目标文件或目录]

### 2．命令功能：

Linux文件系统中，有所谓的链接(link)，我们可以将其视为档案的别名，而链接又可分为两种 : 硬链接(hard link)与软链接(symbolic link)，硬链接的意思是一个档案可以有多个名称，而软链接的方式则是产生一个特殊的档案，该档案的内容是指向另一个档案的位置。硬链接是存在同一个文件系统中，而软链接却可以跨越不同的文件系统。

软链接：

1.软链接，以路径的形式存在。类似于Windows操作系统中的快捷方式

2.软链接可以 跨文件系统 ，硬链接不可以

3.软链接可以对一个不存在的文件名进行链接

4.软链接可以对目录进行链接

硬链接:

1.硬链接，以文件副本的形式存在。但不占用实际空间。

2.不允许给目录创建硬链接

3.硬链接只有在同一个文件系统中才能创建

这里有两点要注意：

第一，ln命令会保持每一处链接文件的同步性，也就是说，不论你改动了哪一处，其它的文件都会发生相同的变化；

第二，ln的链接又分软链接和硬链接两种，软链接就是ln –s 源文件 目标文件，它只会在你选定的位置上生成一个文件的镜像，不会占用磁盘空间，硬链接 ln 源文件 目标文件，没有参数-s， 它会在你选定的位置上生成一个和源文件大小相同的文件，无论是软链接还是硬链接，文件都保持同步变化。

ln指令用在链接文件或目录，如同时指定两个以上的文件或目录，且最后的目的地是一个已经存在的目录，则会把前面指定的所有文件或目录复制到该目录中。若同时指定多个文件或目录，且最后的目的地并非是一个已存在的目录，则会出现错误信息。

### 3．命令参数：

必要参数:

-b 删除，覆盖以前建立的链接

-d 允许超级用户制作目录的硬链接

-f 强制执行

-i 交互模式，文件存在则提示用户是否覆盖

-n 把符号链接视为一般目录

-s 软链接(符号链接)

-v 显示详细的处理过程

选择参数:

-S “-S<字尾备份字符串> ”或 “--suffix=<字尾备份字符串>”

-V “-V<备份方式>”或“--version-control=<备份方式>”

--help 显示帮助信息

--version 显示版本信息

## 修改系统语言为中文

1 临时更换语言

如果只是临时更换linux系统的语言环境，可以通过输入设置 LANG=语言名称， 如中文是

zh\_CN.UTF-8

2 修改系统默认语言

以上方法是通过修改设置系统默认的语言配置

如vi   /etc/sysconfig/i18n （注意改好之后重启一下系统）

3 其他注意事项

如果按照以上方法设置修改设置中文语言还是不行的话，注意您的链接终端选择的编码。

如xshell为例，把终端的编码选择中文**，**或者UTF8即可

## 清理IPC资源

AIX系统中：ipcs  -mqs|grep   $USER|awk '{print  "ipcrm -"  $1$2}'|sh  -x (清理IPC资源)

Linux系统中：

ipcs –m|grep $USER|awk ‘{print “ipcrm -m” $2}’|sh –x （清理共享内存）

ipcs -q|grep $USER|awk ‘{print “ipcrm -q” $2}’|sh –x （清理消息队列）

ipcs –s|grep $USER|awk ‘{print “ipcrm -s” $2}’|sh –x （清理信号量）

## shell中=字符串比较与赋值的区别

字符串复制：str1=str2

=两边不能有空格

字符串比较：[ str1 = str2 ]

=两边必须有空格

## 用-z判空时一定要加引号“”

判断str变量是否为空：[-z “$str”]变量str一定要加引号

## shell脚本字符串处理常用方法

### 一、构造字符串

直接构造

STR\_ZERO=hello

STR\_FIRST="i am a string"

STR\_SECOND='success'

重复多次

#repeat the first parm($1) by $2 times

strRepeat()

{

local x=$2

if [ "$x" == "" ]; then

x=0

fi

local STR\_TEMP=""

while [ $x -ge 1 ];

do

STR\_TEMP=`printf "%s%s" "$STR\_TEMP" "$1"`

x=`expr $x - 1`

done

echo $STR\_TEMP

}

举例：

STR\_REPEAT=`strRepeat "$USER\_NAME" 3`

echo "repeat = $STR\_REPEAT"

### 二、赋值与拷贝

直接赋值

与构造字符串一样

USER\_NAME=terry

从变量赋值

ALIASE\_NAME=$USER\_NAME

### 三、联接

直接联接两个字符串

STR\_TEMP=`printf "%s%s" "$STR\_ZERO" "$USER\_NAME"`

使用printf可以进行更复杂的联接

### 四、求长

获取字符串变量的长度：${#string}

求字符数(char)

COUNT\_CHAR=`echo "$STR\_FIRST" | wc -m`

echo $COUNT\_CHAR

求字节数(byte)

COUNT\_BYTE=`echo "$STR\_FIRST" | wc -c`

echo $COUNT\_BYTE

求字数(word)

COUNT\_WORD=`echo "$STR\_FIRST" | wc -w`

echo $COUNT\_WORD

### 五、比较

相等比较 str1 = str2

不等比较 str1 != str2

注意=，左右两边一定要有空格

举例：

if [ "$USER\_NAME" = "terry" ]; then

echo "I am terry"

fi

小于比较

#return 0 if the two string is equal,

return 1 if $1 < $2,

else 2

strCompare()

 { local x=0 if [ "$1" != "$2" ]; then x=2 localTEMP=`printf "%s\n%s" "$1" "$2"` local TEMP2=`(echo "$1"; echo "$2") |sort` if [ "$TEMP" = "$TEMP2" ]; then x=1 fi fi echo $x }

### 六、测试

判空   -z str

判非空  -n str

注意：使用-z 时变量一定要加引号

如：if [ -z "$str" ]

是否为数字

# return 0 if the string is num, otherwise 1

strIsNum()

{

local RET=1

if [ -n "$1" ]; then

local STR\_TEMP=`echo "$1" | sed 's/[0-9]//g'`

if [ -z "$STR\_TEMP" ]; then

RET=0

fi

fi

echo $RET

}

举例：

if [ -n "$USER\_NAME" ]; then

echo "my name is NOT empty"

fi

echo `strIsNum "9980"`

### 七、分割

以符号＋为准，将字符分割为左右两部分

使用sed

举例：

命令 date --rfc-3339 seconds 的输出为

2007-04-14 15:09:47+08:00

取其＋左边的部分

date --rfc-3339 seconds | sed 's/+[0-9][0-9]:[0-9][0-9]//g'

输出为

2007-04-14 15:09:47

取+右边的部分

date --rfc-3339 seconds | sed 's/.\*+//g'

输出为

08:00

以空格为分割符的字符串分割

使用awk

举例：

STR\_FRUIT="Banana 0.89 100"

取第3字段

echo $STR\_FRUIT | awk '{ print $3; }'

### 八、子字符串

字符串1是否为字符串2的子字符串

# return 0 is $1 is substring of $2, otherwise 1

strIsSubstring()

{

local x=1

case "$2" in

\*$1\*) x=0;;

esac

echo $x

}

Shell字符串截取

一、Linux shell 截取字符变量的前8位，有方法如下：

1.expr substr “$a” 1 8

2.echo $a|awk ‘{print substr($0,1,8)}’

3.echo $a|cut -c1-8

4.echo $

5.expr $a : ‘\(.\\).\*’

6.echo $a|dd bs=1 count=8 2>/dev/null

二、按指定的字符串截取

1、第一种方法:

•${varible##\*string} 从左向右截取最后一个string后的字符串

•${varible#\*string}从左向右截取第一个string后的字符串

•${varible%%string\*}从右向左截取最后一个string后的字符串

•${varible%string\*}从右向左截取第一个string后的字符串

“\*”只是一个通配符可以不要

例子：

$ MYVAR=foodforthought.jpg

$ echo ${MYVAR##\*fo}

rthought.jpg

$ echo ${MYVAR#\*fo}

odforthought.jpg

2、第二种方法：只能在bash中可以这么用

${varible:n1:n2}:截取变量varible从n1开始的n2个字符，组成一个子字符串。可以根据特定字符偏移和长度，使用另一种形式的变量扩展，来选择特定子字符串。

试着在 bash 中输入以下行：

$EXCLAIM=cowabunga

$echo ${EXCLAIM:0:3}

cow

$echo ${EXCLAIM:3:7}

abunga

这种形式的字符串截断非常简便，只需用冒号分开来指定起始字符和子字符串长度

三、按照指定要求分割：

比如获取后缀名

ls -al | cut -d “.” -f2

shell (bash) 比较运算符

                                  运算符 描述 示例

文件比较运算符

-e  filename 如果filename存在，则为真 [ -e /var/log/syslog ]

-d  filename 如果filename为目录，则为真 [ -d /tmp/mydir ]

-f   filename 如果filename为常规文件，则为真 [ -f /usr/bin/grep ]

-L  filename 如果filename为符号链接，则为真 [ -L /usr/bin/grep ]

-r   filename 如果filename可读，则为真 [ -r /var/log/syslog ]

-w  filename 如果filename可写，则为真 [ -w /var/mytmp.txt ]

-x   filename 如果filename可执行，则为真 [ -L /usr/bin/grep ]

filename1-n tfilename2 如果filename1比filename2新，则为真 [ /tmp/install/etc/services -nt /etc/services ]

filename1-o tfilename2 如果filename1比filename2旧，则为真 [ /boot/bzImage -ot arch/i386/boot/bzImage ]

字符串比较运算符[size=-1]（请注意引号的使用，这是防止空格扰乱代码的好方法）

-z string 如果string长度为零，则为真 [ -z "$myvar" ]

-n string 如果string长度非零，则为真 [ -n "$myvar" ]

string1 = s tring2  如果string1与string2相同，则为真 [ "$myvar" = "one two three" ]

string1 != string2 如果string1与string2不同，则为真 [ "$myvar" != "one two three" ]

算术比较运算符

num1  -eq  num2 等于 [ 3 -eq $mynum ]

num1  -ne  num2 不等于 [ 3 -ne $mynum ]

num1  -lt    num2 小于 [ 3 -lt $mynum ]

num1  -le  num2 小于或等于 [ 3 -le $mynum ]

num1  -gt  num2 大于 [ 3 -gt $mynum ]

num1 -ge  num2 大于或等于 [ 3 -ge $mynum ]

## 目录的rwx权限

目录的r：允许列出目录下的文件和子目录

目录的w：允许在目录下新建和删除目录下的文件

目录的x：允许访问目录

## 算术扩展

**算术扩展语法：**

**$((算术式))**

如果数字以0开头，则视为八进制数字，如016等于十进制的14

如果数字以0x或0X开头，则视为十六进制数字

不同进制的数字，可使用“基底#数字”的表示法，其中基底的范围为2-64，

其中数字如果超过9，则a代表10，b代表11，

其他以此类推

如15进制15#1b，其值为1\*15+11=26

     2进制2#1001，其值为1\*8+1=9

**bash中：\*\*表示是乘方运算，**如$((2\*\*5))，表示求2的5次方，值为32

**外部程序expr做算术运算：**

**语法：expr   表示式**

expr是外部程序和shell的版本没有关系几乎可以在所有的平台上都可以执行，

如果注重跨平台和可移植性，脚本开发可以用expr做算术运算

在使用expr时要特别注意“表示式”中是否含有与bash shell的特殊字符，

如 \* 、|、<、>、！、&、（、）等等

如果有的话要用\予以转义，否则会报错

例如r=$(expr 2 \\*  4),echo $r,其值为8

注意：expr没有乘幂功能，即r=$(expr 2 \*\* 5)是错误的语法，无法执行

递增操作   r=0；r=expr  ${r}   + 1

**对比样式：**

**例如：r=$(expr  "string"    :  st)**

**由字符串“string”开头，对比样式st，返回符合样式st的字符数**

**也可以写成r = $(expr match "string" st)**

**找出字符串的位置：**

**r=$(expr  index  "string"  in)**

**取出子字符串**

**r=$(expr  "string"   2  3)**

**从第二个字符t开始取出3个长度的子字符串，r的值为“tin”**

**计算字符串的长度**

**r=$(expr  length  "string")**

**bash中可以使用$[算术式] 做算术运算**

**语法：$[算术式]**

**加:r=$[4+5]**

**减:r=$[4-5]**

**乘:r=$[4\*5]**

**除:r=$[4/5],如果有余数则舍弃**

**求余数：r=$[4%5]**

**乘方：r=$[4\*\*5]**

**递增1：r=5;r=$[r+1],之后r值为6**

declare  -i     I  声明变量I的属性为整数

I=8+16

echo $I

由于I是整数，8+16就不再是字符串，而是一个算式，因此bash会对它进行运算

declare注意事项：

1、运算符和操作数之间不可以有空格，要紧密连接

2、特殊字符不必使用\转义例如乘法 \*，乘方 \*\*

3、算式中可以包含其他变量，变量之前 不必加上$

let用法注意事项同declare，另外可以使用空格符让表达式可读性高一点，

但此时必须用引号包含表达式才行，例如let "i  =  i  +  5"

例：

let I=8+6

let I=16-8

let I=5\*3

let I=2\*\*3

## shell中declare和let内置命令

declare  -i     I  声明变量I的属性为整数

I=8+16

echo $I

由于I是整数，8+16就不再是字符串，而是一个算式，因此bash会对它进行运算

declare注意事项：

1、运算符和操作数之间不可以有空格，要紧密连接

2、特殊字符不必使用\转义例如乘法 \*，乘方 \*\*

3、算式中可以包含其他变量，变量之前 不必加上$

let用法注意事项同declare，另外可以使用空格符让表达式可读性高一点，

但此时必须用引号包含表达式才行，例如let "i  =  i  +  5"

例：

let I=8+6

let I=16-8

let I=5\*3

let I=2\*\*3

## shell脚本-命令替换

把命令执行后的标准输出，赋值给指定变量，在这个过程中，默认会自动删除换行字符

有以下两种语法：

新式的写法：变量名称=$(命令)（建议用新式的写法）

旧式的写法：变量名称=`命令`

例如：

file=“/etc/passwd”

IFS=' '

fc=$(cat  $file)或fc=`cat $file`

或者

file=“/etc/passwd”

fc=$(<  $file)或fc=`< $file`  (<是输出重定向符号)

命令替换可以包含两种或两种以上的命令,命令替换的值是最后一次指令执行的结果

例如：pdir=$(cd ..;pwd)

在命令替换里也可以再包含其他的命令替换

例如：

 r=$(du -s $(pwd))

 取得工作目录占用磁盘用量总和

## shell中常用变量

HOME：用户主目录变量

PATH:命令的搜索路径，用：分隔

IFS：分隔符

PS1：主提示符

PS2：次提示符，默认为>

PS3：select选单提示符,默认为#？

PS4：追踪shell程序各行的提示符，默认为+

$n：程序参数,$0表示程序的名称,$1程序的第一个参数，$2、$3、。。。。$n以此类推

$\*:代表所有的位置参数，幷视为一个字符串，例如：test.sh ABC 123 xyz,

        $\*的内容为字符串"ABC 123 xyz"

$@:代表所有的位置参数，但$@代表各个位置参数组成的串行，例如：test.sh ABC 123 xyz,

$@的内容为字符串"ABC"  "123"   "xyz"这三个字符串

$#位置参数的个数，例如：test.sh ABC 123 xyz,$#的值为3

$?上一个命令执行结束后的返回值，通常0代表执行成功，非0代表执行失败

TMOUT:终端的超时时间

## 变量扩展

**一、变量扩展---测试存在性及空值**

${变量名-默认值}，如果变量不存在，变量则为默认值

${变量名：-默认值}，如果变量不存在或值为空，变量值则为默认值

特殊用法:[  -n ${DEBUG:-} ]  &&  set  -v,这句话一般放在脚本的开头

${变量名：=默认值}，如果变量不存在或值为空，变量值则为默认值

${变量名：？提示信息}，如果变量不存在或值为空，则提示错误信息，脚本停止执行

${变量名 ：+真值}，如果变量存在且不为空，则返回真值

例如：fgrep.sh

#!/usr/bin/sh

exec grep  -F  ${1:+"$@"}

  -F标示使用固定字符串作为寻找样式

  ${1:+"$@"}标示的意思是若$1(脚本的第一个参数)存在，

  则返回所有的参数$@

总结：

上述各式中若去掉：则只测试存在性，不测试空值

记忆方法

：空，测空值

-负向，测不存在性

=赋值，给一个变量设一个默认值

？有问题，检查条件是否完备后再执行吧

+正向，测试存在

二、变量扩展---取子符串或变量长度

  字符串的第一个字符，编号为0，右邻的字符编号，依次增加1，例如：“sheller”字符s的编号为0，r的编号为6

取子字符串的语法有两种：

语法一：${变量：位置起点}

例如：myname="yiyouliang"

substr=${myname:3}

则echo $substr，屏幕显示为uliang

语法二：${变量:位置起点：长度}

例如：myname="yiyouliang"

substr=${myname:0:3}

则echo $substr，屏幕显示为yiyo

取变量位置参数：

例如：substr.sh   x   y   z,这道指令中，第一个参数是x ,第二个参数是y，第三个参数是z,$@代表所有的参数

$1的值为x，$2的值为y,$3的值为z，命令本身用$0表示，其值为substr.sh

像$0,$1,$2,$3这样的参数称为位置参数

取得部分位置参数的用法有以下二式：

${@:起点}，由起点开始取得后面的所有的位置参数

${@:起点：个数}，由起点开始取得指定个数的位置参数

三、变量扩展---取字符串长度

语法：${#变量名}

总结：

${#变量名}   取字符串长度

${#数组[@]} 取得数组元素个数

${#数组[\*]} 取得数组元素个数

取字符长度还可以用外部程序expr来做

方法1：expr  length  "字符串"

方法2：expr "字符串":'.\*'   这里:后接的.\*  是一个代表任意多个字符的字符串样式，

expr会根据此样式来对比“字符串”等于是计算出字符串的长度

四、变量扩展---对比样式

  A、由字符串前面对比，删除相符者

    由前面对比，删除最短的：

    语法1：${变量#样式}

    例如：filename="usr/sbin/ntpdate"

        r={filename#/\*/}

        echo $r

  如果对比的样式是/\*/,即凡一对斜对线之间有字符串者(空字符串亦可)，对比符合

  因#表示由前面取最短的，所以对比到的最短字符串为/usr/，故将它删除，然后传回剩下的字符串，幷设值给r

echo 显示结果为sbin/ntpdate

由前面对比，删除最长的：

语法2：${变量##样式}

例如：filename="usr/sbin/ntpdate"

  r={filename##/\*/}

    echo $r

    如果对比的样式是/\*/,即凡一对斜对线之间有字符串者(空字符串亦可)，对比符合

    因##表示由前面取最长的，所以对比到的最长字符串为/usr/sbin/，故将它删出，然后传回剩下的字符串，幷设值给r

    echo 显示结果为ntpdate

B、由字符串后面对比，删除相符者

  由后面对比，删除最短的：

  语法1：${变量名%样式}

  例如：filename="usr/sbin/ntpdate"

      r={filename%/\*}

      echo $r

  如果对比的样式是/\*,即凡斜线后面有字符串者(空字符串亦可)，对比符合 因%表示由后面取最短的，所以对比到的最短字符串为/ntpdate，故将它删出，然后传回剩下的字符串，幷设值给r

echo 显示结果为usr/sbin

由后面对比，删除最长的：

语法2：${变量名%%样式}

例如：dns="www.baiidu.com.cn"

  r={dns%%.\*}

echo $r

如果对比的样式是,\*,凡是.后面有字符串者(空字符串亦可)，对比符合

因%%表示由后面取最长的，所以对比到的最长字符串为.baiidu.com.cn，故将它删出，然后传回剩下的字符串，幷设值给r

echo 显示结果为www

总结：

  ${变量#样式}       由前面对比，删除最短的

  ${变量##样式}      由前面对比，删除最长的

  ${变量%样式}      由后面对比，删除最短的

  ${变量%%样式}     由后面对比，删除最长的

  记忆法：

    #经常放在数字前面表示个数，所以看到它，就是记它为由前面对比，一个#删最短，两个#删最长

  %经常放在数字后面表示个数，所以看到它，就是记它为由后面对比，一个%删最短，两个%%删最长

**C、取代或删除部分字符串**

**只替换第一个比较符合的字符串**

**语法：${变量/样式/替换字符串**

**替换全部比较符合的字符串**

**语法：${变量//样式/替换字符串}**

**删除第一个符合样式的字符串**

**语法：${变量/样式/}**

**删除全部个符合样式的字符串**

**语法：${变量//样式/}**

**删除 要求样式在句首**

**语法：${变量/#/}**

**删除 要求样式在句尾**

**语法：${变量/%/}**

**D、取变量名称列表或数组索引列表**

**取变量名称列表**

**语法：${!开头字符串@}或者${!开头字符串\*}**

**E、取数组索引列表**

**语法：${!数组变量[@]}或者${!数组变量[\*]}**

**例如：ar=（a b c d e）**

**echo ${!ar[@]}，显示0 1 2 3**

## shell中数组用法

bash的数组，其元素个数没有限制，数组的索引由0开始，但不一定要连续（可以跳号）。索引也可以用算术表达式表示，如1+2.

用法如下：

A[0]=5

A[1]=10

A[2]=28

这里建立了一个数组叫A，共有3个元素，第1个元素为A[0],设值为5，第2个元素为A[1]设置为10，第3个元素为A[2]设置为28，其索引值分别是0、1、2.

数组建立后，若有需要，随时可以增加元素，例如：

A[3]="bash shell"

这里增加第4个元素，且元素值是一个字符串

取用某个素组元素的值的语法如下：

${数组[索引值]}，

以上例来说，第2个元素为${A[1]}

建立数组可以一次指定每个元素的值，做法如下：

B=（23 88 89 66）

echo ${B[2]},显示为99

建立数组时也可以指定个别元素的索引，做法如下：

c=([3]=77 [1]=100 [5]=66)

echo $c([5])

也可以指定某一元素的索引，其余交由系统接续

E=(124 [8]=166 88)

数组E的第一元素是124，索引为0，第9个元素是166，索引值为8，下一个元素的索引值自动续接为9，其元素值为66

取出数组所有的元素

bash可以一次取出所有的元素，以数组B为例，语法如下:

echo ${B[@]},结果显示为23 88 99 66

把@改成\*，一样可以取出所有的元素，如下所示：

echo ${B[\*]},

这两者的区别是，${B[@]}取到4个空白分割的数字，但${B[\*]}得到一整个字符串"23 88 99 66"

取得数组元素的个数：

语法如下：

${#数组[@]}或者${#数组[\*]}

如果数组元素是字符串还可以用以下语法取得元素的长度

${#数组[索引值]}

取消数组或数组元素

取消数组：unset A

取消某一个元素 unset A[索引值]

## Here Document

Here Document的基本语法为：

命令 << 标记

。。。。。

。。。。。

。。。。。

标记

这样会把命令和标记之间的内容，利用转向输入的方式交给该命令去处理。

标记可以是任何字符串。标记选用的原则：尽量选不常用的字符组合，避免该字符串出现在输入的内容中造成bash误判

Here Document 也支持变量替换，在输入的内容中，如果有变量，bash在转向前，会先替换成变量值

用例：

From='From : me@example.edu.cn'

To='To : you@example.edu.cn'

Subject='Subject:Hello World'

Msg='新年快乐'

Em='20160314.txt'

cat > $em <<Here

$From

$To

$Subject

$Msg

Here

控制Here Document的格式

Here Document支持两种控制输出格式的的做法：

### 关闭变量替换的功能

有时候，希望按照原来的样子转向输出，不做任何变量替换。做法如下：

cat << 'Here'

line 1 is good

$WOW

Here

对标记Here加上单引号，即可关闭变量替换

$WOW还是保持原样，不做变量替换

### 去掉每行之前的TAB字符

用例：

cat <<-'Here'

line 1 is good

$WOW

Here

### 利用Here Document做多行批注

原本bash只支持单行注释，即#右边的任何字符串，bash都会忽略

利用Here Document可做多行批注

例：

：<<Do-NotThing

这是第一行批注

这是第二行批注

这是第三行批注

这是第四行批注

其他行，请类推

Do-NotThing

以上是多行批注的范例，行1中，：代表“什么都不做”的空命令，这个区域在script中，bash会忽略不执行，因此，达到批注的效果

### 利用Here Document 打包C(或其他程序语言的)原始码

这是Cracker散布安装漏洞程序时最喜欢的用法，举例如下：

#!/bin/bash

echo "正在产生hello.c ....."

echo

cat <<'EOF' >hello.c

#include <stdio.h>

int main()

{

printf("Hello World\n");

return 0；

}

EOF

echo "正在编译hello.c ....."

echo

#编译hello.c产生执行文件

gcc -o hello hello.c

#编译成功则执行

if [ $？ -eq 0 ] ;then

echo "正在执行hello...."

echo

./hello

else

echo "Compile error:hello.c"

fi

## shell变量不存在与NULL的区别

unset   变量名；取消变量定义，让变量不存在

变量名=；设置变量内容为空

## alias 别名

执行alias 显示目前设置的别名

alias  别名=指令，定义一个新的别名

需要注意两件事，第一、=（等号）两旁不可以有空白；第二、如果等号右边的指令中含有空白，需要用单引号含括该指令

## shell中一对单引号中不可以出现单引号,就算用转义字符也不行

shell中一对单引号中不可以出现单引号,就算用转义字符也不行

例如

echo 'this is jack's book'

echo 'this is jack\'s book'

解决办法：去掉最外面的一对单引号

echo  this is jack's book

或改用双引号

echo  "this is jack's book"

或使用接续的方式

echo ' this is jack'\''s book'

## /etc/passwd文件格式

账号：x：UID：GID：用户信息:主目录位置:登录shell程序

x为密码栏，加密后保存在/etc/shadow文件中

## $LOGNAME和$USER的区别

$LOGNAME指登陆主机的用户名

$USER指当前环境中的使用者用户名

例如：当前登陆账户是root，使用su命令切换成使用者ols3后，$USER的内容变成了ols3，

但$LOGNAME的内容还是root

如果要完全切换成使用者的环境就要用  su  -   ols3，此时，$LOGNAME和$USER都是ols3

## script 记录命令产生的信息

script  log.txt  将命令产生的结果保存在log.txt中，执行exit可以退出script

## eval内置命令

eval  $HIST  等同于执行history命令

## shopt设置bash的行为模式

shopt或shopt -p显示目前各选项的开关状态

shopt -s 启用选项

shopt -u关闭选项

shopt -o 使用和set -o 一样

## 设置shell各属性的开关状态

set   -o    显示shel各属性的的状态

set   -o    vi 打开shell的vi属性开关

set   +o   vi 关闭shell的vi属性开关

## 在父进程环境中执行shell脚本

**.**  脚本名

source     脚本名

## unix 系统中的文件类型

lsp\_bcd共七种类型:

l:链接文件

[s:socket通信文件](file:///s:socket通信文件)

[p:管道文件](file:///p:管道文件)

-:普通文件

[b:块文件](file:///b:块文件)

[c:字符文件](file:///c:字符文件)

d:目录文件

## shell脚本中文件的3种身份4种权限

3种身份是：

[u:文件主](file:///u:文件主)(user)

[g:文件组](file:///g:文件组)(group)

[o:其他人(other)](file:///o:其他人(other))

4中权限是：

[r:读权限](file:///r:读权限)(read)

[w:写权限](file:///w:写权限) (write)

[x:可执行权限](file:///x:可执行权限)(execute)

[s:特殊权限](file:///s:特殊权限) set user id(s)  set group id(s)  sticky bit(t),简称sst

如果在执行一个文件时，执行者可以暂时变成“文件主”的身份，

则称具有特殊权限set user id,其权限组rwx改用为rws来表示，

这种文件会在原权限值（假设是0755）之前加上4000，记为4755

如果在执行一个文件时，执行者可以暂时变成“文件组”的身份，

则称具有特殊权限set grop id,其权限组r\_x改用为r\_s来表示，

这种文件会在原权限值（假设是0755）之前加上2000，记为2755

如果一个目录，其中的文件只有文件主才能删除，则称该文件具有sticky bit,其权限组记为rwt,

文件会在原权限值（假设是0777）之前加上1000，记为1777

## shell脚本排错方法

sh  -v   脚本名  //检查shell脚本中关键字等语法错

sh  -x  脚本名  //追踪脚本中的逻辑

## aix中把时间转换成秒数

perl –e ‘print time “\n”’

## vi编辑器

在vi中，必须牢记它是有两个状态的 ---- 输入状态与命令状态。

由输入状态切换 到命令状态，必须ESC键；

而从命令状态转换到输入状态的命令比较多，有a， A，i，I，s，S，c，C，o和O等，

命令模式下：

w后移动一个单词

e 后移动一个单词

b 前移动一个单词

f  x  光标跳到x字符上

. 重复一次命令

r 替换当前字符

R 替换后续所有字符

D删除当前字符的后面字符

dd删除当前行

光标移到想要被复制词的词首，输入dw,删除一个单词

有时候不想费劲看多少行或复制大量行时，可以使用标签来替代

光标移到起始行，输入ma

光标移到结束行，输入mb

光标移到粘贴行，输入mc

然后 :'a,'b co 'c   把 co 改成 m 就成剪切了

要删除多行的话，可以用 ：5, 9 de

光标移到想要被复制词的词首，输入yw，复制一个单词

光标移到想要被复制字符的首位置，若想复制5个字符输入：5yl

命令行模式下输入

6,9 co 12

复制第6行到第9行之间的内容到第12行后面。

j 下移一个字符

k 上移一个字符

h 左移一个字符

l 由移一个字符

vi中大小写转换命令:数字+~

### VI中的多行删除与复制

法一：

单行删除，：1（待删除行）d

多行删除 ，：1,10d

法二：

光标所在行，dd

光标所在行以下的N行，Ndd

方法1：

光标放到第6行，

输入：2yy

光标放到第9行，

输入：p

此方法适合复制少量行文本的情况，复制第6行（包括）下面的2行数据，放到第9行下面。

方法2：

命令行模式下输入

6,9 co 12

复制第6行到第9行之间的内容到第12行后面。

方法3：

有时候不想费劲看多少行或复制大量行时，可以使用标签来替代

光标移到起始行，输入ma

光标移到结束行，输入mb

光标移到粘贴行，输入mc

然后 :'a,'b co 'c 把 co 改成 m 就成剪切了

要删除多行的话，可以用 ：5, 9 de

### vi中大小写转换命令

个数+~

例如：

转换(大写转小写，小写转大写)10个字母: 10~

### vi中替换命令

vi/vim 中可以使用 :s 命令来替换字符串。  
:s/vivian/sky/ 替换当前行第一个 vivian 为 sky  
:s/vivian/sky/g 替换当前行所有 vivian 为 sky

:n,$s/vivian/sky/ 替换第 n 行开始到最后一行中每一行的第一个 vivian 为 sky  
:n,$s/vivian/sky/g 替换第 n 行开始到最后一行中每一行所有 vivian 为 sky  
n 为数字，若 n 为 .，表示从当前行开始到最后一行

:%s/vivian/sky/（等同于 :g/vivian/s//sky/） 替换每一行的第一个 vivian 为 sky  
:%s/vivian/sky/g（等同于 :g/vivian/s//sky/g） 替换每一行中所有 vivian 为 sky

可以使用 # 作为分隔符，此时中间出现的 / 不会作为分隔符  
:s#vivian/#sky/# 替换当前行第一个 vivian/ 为 sky/

:%s+/oradata/apras/+/user01/apras1+ （使用+ 来 替换 / ）： /oradata/apras/替换成/user01/apras1/

## vi脚本中输入”^M”字符

ctrl+V ctrl+M

## AIX系统安装

安装前必须已满足IBM的装机条件和所有的硬件设备已连接好。

打开外接设备和系统的电源，假设从光盘启动，在电源打开以后系统将从光盘启动。

在选择好控制台和安装所用到的语言环境之后系统将出现提示信息供用户选择所需要的功能

Welcome to Base Operating System

Installation and Maintenance

Type the numbers of your choice and press Enter. Choice indicated by>>>

1. Start Install now with Default Settings

2. Change/Show Installation Settings and Install

3. Start Maintenance Mode for System Recovery

88. Help?

>>>Choice[1]:

其中第三项是为了修复系统中出现的某些错误而设置的单用户环境。

第一项是如果不希望改动任何缺省设置的话，就选择1开始安装。

第二项是更改或查看系统安装的设置。在第二项的子菜单中，有三个选项可以被改变：

Installation Settings

Enter type 0 or press Enter to install with current settings, or type the number of the setting you want to change and press Enter.

1. System Settings

Method of installation-------------------------------------New and Complet Overwrite

Disk where you want to install----------------------------hdisk0

2. Primary Language Environment Settings ( After install )

Cultural Convention-----------------------------------C ( POSIX )

Language------------------------------------------------C ( POSIX )

Keyboard------------------------------------------------C ( POSIX )

3. Install Trusted Computing Base---------------------------no

0. Install with the settings listed above

88. Help?

99. Previous Menu

>>>Choice[1]:

其中选项1是全新安装。使用这种方法安装系统将会覆盖用户所选择目标盘上的所有数据。

选项2是迁移安装。 这种方法常用来做系统升级(例如从AIX3.2升级到AIX4.1)，使用这种方法安装时，除了tmp以外的所有文件系统的内容都会被保留。但一些AIX设备驱动的软件必须重新安装。

选项3是保留安装。 如果用户希望保留rootvg卷组中的用户数据，则可选择这种安装方法。此时/usr、/tmp、/var和/文件系统中的所有数据都将被覆盖，系统安装完后还须重新配置。

当以上参数都选择完后就可以开始安装系统了。

系统安装完后会自动重新启动，且出现install\_assist的画面供用户进行基本的配置。

其它实用的技巧

1、查看软件版本的命令

# oslevel

# lslpp -l

oslevel 是用来查看AIX操作系统的版本。

而 lslpp -l 是用来查看系统具体每个文件包的版本。

例子：

# lslpp -l > lpp.txt

则 lpp.txt 中包含如下信息：l

Fileset Level State Description ----------------------------------------------------------------------------Path: /usr/lib/objrepos

IMNSearch.bld.DBCS 1.2.0.4 COMMITTED NetQuestion DBCS Buildtime Modules

IMNSearch.bld.SBCS 1.2.1.3 COMMITTED NetQuestion SBCS Buildtime Modules

IMNSearch.rte.DBCS 1.2.0.4 COMMITTED NetQuestion DBCS Search

Engine

IMNSearch.rte.SBCS 1.2.1.3 COMMITTED NetQuestion SBCS Search

Engine

IMNSearch.rte.httpdlite 1.1.1.1 COMMITTED NetQuestion Local HTTP

Daemon

Java.adt.docs 1.1.6.0 COMMITTED Java Documentation Java.adt.includes 1.1.6.4 COMMITTED Java Application Development Toolkit Includes

2、查看所有出错信息

在系统运行时，一些系统错误记录会记录在 errlog 中，其中有些错误还会在终端上 显示出来，检查错误日志可用以下命令：

# errpt | more查看系统所有的记录

IDENTIFIER TIMESTAMP T C RESOURCE\_NAME DESCRIPTION

E85C5C4C 0426104399 P S CFGLFT SOFTWARE PROGRAM ERROR

2BFA76F6 0426104099 T S SYSPROC SYSTEM SHUTDOWN BY USER

9DBCFDEE 0426104399 T O errdemon ERROR LOGGING TURNED ON

192AC071 0426103999 T O errdemon ERROR LOGGING TURNED OFF

AA8AB241 0423132999 T O clstrmgr OPERATOR NOTIFICATION

其中

IDENTIFIER 为错误编号，当需要检查详细信息时常会用到。

TIME STAMP 为时间标签，它记录的是出错时间，

其格式：月月日日时时分分年年

T 为Type ，它记录的是错误类型

P ：为永久错误，需引起注意

T ：为临时错误。

C为Class，它记录的是错误类型

H ： Hardware

S ： Software

O ： Errloger command messages

U ： undetermined

RESOURCE\_NAME 为错误来源

DESCRIPTION为错误描述

查看系统详细记录内容

# errpt -aj <IDENTIFIER>|pg

其中IDENTIFIER为错误编号，如 # errpt -aj 0426104399 | pg

查看系统所有的硬件出错记录

# errpt -dH

3、文件系统的操作

(1) 、列出所有的文件系统

# lsfs

它的作用是列出所有的文件系统，其输出如下：

Name Nodename Mount Pt VFS Size Options Auto

Accounting

/dev/hd4 -- / jfs 163840 -- yes

no

/dev/hd1 -- /home jfs 16384 -- yes

no

/dev/hd2 -- /usr jfs 2818048 -- yes

no

/dev/hd9var -- /var jfs 16384 -- yes

no

/dev/hd3 -- /tmp jfs 32768 -- yes

no

/dev/lv00 -- /ibmcxx jfs 835584 rw no

(2)、 列出所有 mount 了的文件系统

# mount

其输出如下：

node mounted mounted over vfs date options

-------- --------------- --------------- ------ ------------ ---------------

/dev/hd4 / jfs Nov 16 12:47 rw,log=/dev/hd8

/dev/hd2 /usr jfs Nov 16 12:47 rw,log=/dev/hd8

/dev/hd9var /var jfs Nov 16 12:47 rw,log=/dev/hd8

/dev/hd3 /tmp jfs Nov 16 12:47 rw,log=/dev/hd8

/dev/hd1 /home jfs Nov 16 12:49 rw,log=/dev/hd8

(3)、 查看各文件系统的使用情况

# df -k

其输出如下：

Filesystem 1024-blocks Free %Used Iused %Iused Mounted on

/dev/hd4 81920 68568 17% 1724 5% /

/dev/hd2 1409024 430896 70% 28467 9% /usr

/dev/hd9var 8192 6540 21% 293 15% /var

/dev/hd3 16384 15352 7% 51 2% /tmp

/dev/hd1 8192 3820 54% 54 3% /home

(4) 、启动、停止和检测 HACMP

a用(#smit clstart)启动 HACMP

b用(#tail -f /tmp/hacmp.out)跟踪输出

c用(#netstat -in)查看 en0、en1的状态,看是否已从 boot 地址切换成service地址

d用(#lsvg -o) 查看是否 datavg 已经激活

e正常启动后，用(#smit clstop)通过 takeover 方式 down掉一个

node，查看是否 IP 被另一个node接管

f停止HACMP，用(#smit clstop)通过graceful方式或force方式停用HACMP

-graceful方式是node停用HACMP并释放资源

-force方式是node停用HACMP但不释放资源

-take over方式是node停用HACMP,并让另一个node接管资源

## 配置和运行TCP/IP

1、在配置TCP/IP前，需知道：

主机名和域名

IP地址和子网掩码

路由器的IP地址

域名服务器的IP地址

smit tcpip

-Minimum Configuration & Startup

Minimum Configuration & Startup

To Delete existing configuration data, please use Further Configuration menus

Type or select values in entry fields.

Press Enter AFTER making all desired changes.

[Entry Fields]

\* HOSTNAME [ ]

\* Internet ADDRESS (dotted decimal) [ ]

Network MASK (dotted decimal) [ ]

\* Network INTERFACE en0

NAMESERVER

Internet ADDRESS (dotted decimal) [ ]

DOMAIN Name [ ]

Default GATEWAY Address [ ]

(dotted decimal or symbolic name)

Your CABLE Type N/A +

START Now no +

F1=Help F2=Refresh F3=Cancel F4=List

Esc+5=Reset Esc+6=Command Esc+7=Edit Esc+8=Image

Esc+9=Shell Esc+0=Exit Enter=Do

在域中输入正确的值，按ENTER键即可完成TCP/IP的配置。

如果值有变化，第二次可用smit chinet来设置。例如：

smit chinet

[Entry Fields]

Network Interface Name en0

Internet ADDRESS (dotted decimal) []

Network MASK (dotted decimal) []

Current STAT up

F1=Help F2=Refresh F3=Cancel F4=List

Esc+5=Reset Esc+6=Command Esc+7=Edit Esc+8=Image

Esc+9=Shell Esc+0=Exit Enter=Do

2、TCP/IP的启动

TCP/IP启动时需要两个启动文件：/etc/rc.net和/etc/rc.tcpip。文件/etc/rc.net用于配置网络界面，设置路由；而/etc/rc.tcpip用于启动tcpip后台服务进程。

二、TCP/IP常用的命令

1、ping

2、telnet

3、ftp

/etc/ftpusers

4、netstat显示网络状态，例如：

(1) netstat -in

(2) netstat -rn

三、网络文件系统NFS

NFS是一种能较好共享远程文件的方案。它采用客户/服务器结构，NFS服务器提供共享文件供客户访问，

NFS客户通过网络请求服务器的资源。一台主机可同时作为NFS服务器和NFS客户机。

1、NFS的服务进程

NFS客户机的进程：biod, rpc.statd, rpc.locked

NFS服务器的进程：rpc.mountd, nfsd, rpc.statd, rpc.lockd

2、配置NFS服务器

在配置NFS以前，需要确认已经：

安装和配置了TCP/IP

安装了NFS软件(bos.net.nfs)

明确了网络中那些是NFS服务器和客户机

确定了NFS服务器上需要共享的目录

smit nfs

-Network File System(NFS)

-Add a Directory to Exports List

Add a Directory to Exports List

Type or select values in entry fields.

Press Enter AFTER making all desired changes.

[Entry Fields]

\* PATHNAME of directory to export [ ] /

\* MODE to export directory read-write +

HOSTS & NETGROUPS allowed client access [ ]

Anonymous UID [-2]

HOSTS allowed root access [ ]

HOSTNAME list. If exported read-mostly [ ]

Use SECURE option? no +

Public filesystem? no +

\* EXPORT directory now, system restart or both both +

PATHNAME of alternate Exports file [ ]

F1=Help F2=Refresh F3=Cancel F4=List

Esc+5=Reset Esc+6=Command Esc+7=Edit Esc+8=Image

Esc+9=Shell Esc+0=Exit Enter=Do

其中PATHNAME of directory to export表示共享出去的目录的路径

MODE to export directory表示共享目录的模式

HOSTS & NETGROUPS allowed client access表示允许访问该共享目录的客户机名，如果空白，则允许所有的客户机访问

3、配置NFS客户机

(1)、用nkdir命令建立一个本地安装点

mkdir /home/mntpt

(2)、启动NFS客户进程

smit mknfs

Start NFS

Type or select values in entry fields.

Press Enter AFTER making all desired changes.

[Entry Fields]

\* START NFS now, on system restart or both both +

F1=Help F2=Refresh F3=Cancel F4=List

Esc+5=Reset Esc+6=Command Esc+7=Edit Esc+8=Image

Esc+9=Shell Esc+0=Exit Enter=Do

(3)安装网络文件系统

mount server1:/export\_file /home/mntpt

SMIT工具

SMIT(SMITTY)为用户提供了一个灵活的工具来管理系统。下面介绍一些SMIT中符号和功能键的意义

\* 所需要的值，这个域必须填入值

# 这个域需要填入数字

/ 这个域需要填入路径

X 这个值需要填入一个16进制的数

？ 这个值不会显示

+ 一个下拉式列表，按F4键显示列表

F1(ESC-1) 显示上下文帮助

F2(ESC-2) 刷新

F3(ESC-3) 注销，返回上一级菜单

F4(ESC-4) 列表，给出可选的列表值

F5(ECS-5) 初始化，恢复一个输入域的原始数据

F6(ESC-6) 命令，显示要执行的AIX命令

F7(ESC-7) 编辑，编辑一个下拉式菜单或下拉表中的一项

F8(ESC-8) 镜像，将当前的屏幕存到一个文件中并将当前的快速路径显示出来

F9(ESC-9) 外壳，启动一个子外壳

F10(ESC-10)退出，立即退出SMIT菜单

ENTER 开始执行当前行

/TEXT 在输出中查找文本

n查找下一个出现要查找的文本的地方

SMIT的日志文件在$HOME目录下，名为smit.log，它保存了所有被访问的菜单和对话框，还有所有被执行的命令和输出，以及运行过程中产生的错误。smit.script文件则保存了SMIT执行的所有的AIX命令。

## 查看用户最大进程数

lsattr  -El   sys0

maxupro   选项的值表示每个用户的最大进程数

## AIX系统卷组

AIX系统的数据存储方式：

在AIX中，是这样来组织多个硬盘上的数据的：

物理卷PV 单独的硬盘 /dev/hdisk0 , /dev/hdisk1... 可细分为

物理分区PP

卷组VG 一个或多个硬盘组成。

从逻辑上讲，在一个卷组VG中，可定义若干个逻辑卷LV。它也可细分为逻辑分区LP。这些逻辑卷LV可用作分页空间(Paging Space)、系统日志(SystemLog)，最多的是用作文件系统FileSystem的载体，这种文件系统称为日志文件系统(JournaledFileSystem)。

在系统初始安装时，一个名为rootvg的卷组被用来装载BIOS。在rootvg卷组中，有一些逻辑分区被用来装载一些特定的系统数据，例如启动数据hd5、日志记录(JL)hd8、分页空间(PagingSpace)hd6等。这样rootvg总是被激活的。

在每一个卷组VG内的物理卷PV上，至少一个或多个很重要的数据区--卷组描述区(VGDA)。同一个卷组VG的各个物理卷PV上的VGDA内容，正常时应该是一致的。

在一个卷组VG中，用户可以定义若干个逻辑卷LV，一个逻辑卷在物理上可能是不连续的，有可能跨越了一个卷组中的多个物理卷。可用mklv来创建。一个逻辑卷的大小可利用逻辑分区的数目来确定。每个逻辑分区最多可有三个物理分区与之对应，多个物理分区对应于一个逻辑分区的情况，通常用于对逻辑卷生成镜像。

这样就有这么一些概念：

卷组VG

物理卷PV

物理分区PP

逻辑卷LV

逻辑分区LP

常用命令：

# lsvg显示卷组VG的信息，例如：

(1) lsvg -o

(2) lsvg

(3) lsvg vg02

(4) lsvg -l vg02

#varyoffvg使卷组VG不被激活，例如：

(1) varyoffvg cx\_vg

#varyonvg使卷组VG被激活，例如：

(1) varyonvg cx\_vg

# lspv显示物理卷PV的信息，例如：

(1) lspv hdisk0

(2) lspv -p hdisk2

(3) lspv

hdisk0 0000000012345678 rootvg

hdisk1 10000BC876543258 rootvg

hdisk2 ABCD000054C23486 cx\_vg

# lslv 显示逻辑卷LV的信息，例如：

(1) lslv lv03

(2) lslv -p hdisk2

(3) lslv -l lv03

# lsps显示Paging space的信息，例如：

(1) lsps -a

Page Space Phy Vol Vol Grp Size %Used Active Auto

hd6 hdisk0 rootvg 1024MB 30 yes yes

(2) lsps -s

# lsfs显示文件系统的信息，例如：

(1) lsfs

(2) lsfs -v jfs

## 文件系统

文件系统概述

在AIX中，文件系统是由若个文件和目录组成的分级树形结构。文件系统通过一个挂接点mount point把自己连接到系统的一个节点上，这样文件系统可以方便地安装、拆卸和备份、恢复。

在AIX中，一个文件系统总是独占一个逻辑卷，因此系统逻辑卷对文件系统有一定的限制。当然，逻辑卷既可做为一个文件系统的载体，也可做为它用。AIX支持的文件系统类型有日志文件系统jfs，网络文件系统nfs，光盘文件系统cdrfs等等。

对于一般的日志文件系统，系统在/etc/filesystems文件中都有具体的定义。例如：

/:

dev=/dev/hd4

vol="root"

mount=automatic

check=false

vfs=jfs

log=/dev/hd8

type=bootfs

/home:

dev=/dev/hd1

vol="/home"

mount=true

check=true

vfs=jfs

log=/dev/hd8

/home/user:

dev=/dev/lv00

vfs=jfs

log=/dev/hd8

mount=false

check=true

options=rw

其中check属性表示如果使用fsck命令检测文件系统，此文件是否在检测之列。AIX在系统启动时要调用fsck来检测指定文件系统的完整形。

mount属性有三个选项：automatic表示在系统初启时此文件系统将被自动载入，相反则用false表示；true表示使用mount all命令来加载。

vfs表示文件系统的类型，如JFS、NFS。

log表示此文件系统的日志记录数据将写到的逻辑卷。

options表示此文件系统的操作属性，如rw表示为可读写。

2、文件系统的结构

在AIX中，有五个特定的文件系统，它们是系统安装时自动生成的。

(1)、/ root文件系统包括AIX系统运行的一些关键性的文件和目录，其中比较重要的有目录/etc，/dev，/usr，/var，/home，文件/unix等。

在/etc目录下，包括了有关系统的一些配置文件和命令文件。

在/dev目录下，包含了所有的硬件设备的逻辑文件。

/unix文件则包含了UNIX操作系统的核心程序

(2)、/usr文件系统是AIX中最庞大的一个文件系统，它挂接在root文件系统的/usr目录下。

它包括的内容最常见的有：

/usr/bin目录包括有各种二进制命令文件和SHELL脚本程序。

/usr/lib目录中主要是各种与平台无关的库文件。

/usr/lpp目录下包含了安装到AIX中的各种应用软件产品。

/usr/sbin目录提供给管理员各种用于系统管理的工具。

(3)、/var文件系统包括了一些有关电子邮件、日志记录、事件跟踪等方面的一些应用程序的文本记录。例如：

/var/adm/wtmp用户的登录记录

/var/news系统的消息记录

/var/tmp/\*有关的临时文件

/var/adm/sulog用户使用SU命令的记录

(4)、为了保存一些临时文件，AIX中设有一个专门的文件系统/tmp。

(5)、/home文件系统则是专门用来放置用户数据。

除了这些由系统建立的文件系统外，用户也可以用smit mkfs来建立自己的文件系统。

3、管理文件系统

#smit fs

File Systems

List All File Systems

List All Mounted File Systems

Add/Change/Show/Delete File Systems

Mount a File Systems

Mount a Group of File Systems

Unmount a File Systems

Unmount a Group of File Systems

Verify a File Systems

Backup a File Systems

Restore a File Systems

List Contents of a Backup

三、逻辑卷的管理

对于逻辑卷，在/dev中对应一个流格式和一个块格式的逻辑文件描述：

brw-rw---- 1 root system 19,0 Apr 01 12 : 30 /dev/hdisk0

crw-rw---- 1 root system 19,0 Apr 01 12 : 30 /dev/rhdisk0

brw-rw---- 1 root system 10,6 Apr 01 12 : 30 /dev/hd2

crw-rw---- 1 root system 10,6 Apr 01 12 : 30 /dev/rhd2

它主要包括有：

Journaled file system日志文件系统

Paging space分页空间

Journal log日志记录

Boot Logical Volume引导逻辑卷

Raw device裸设备

Dump Space

我们可以用smit lvm来进行管理和配置

四、分页空间

分页空间是位于硬盘上的固定空间，其中保存着一些位于虚拟内存中且当前不被访问的数据。分页空间也可称为交换空间，实际上就是一个逻辑卷。

在实际系统运行中，如果系统实内存太小，那么位于实内存中最近没有被访问到的数据将被从实内存移到分页空间中，以便为其它的急需要内存资源的作业释放空间。

在AIX中，缺省的分页空间是/dev/hd6逻辑卷。

对于一个系统来说，分页空间的管理通常是对以下两个方面来进行的：

(1)、分页空间的数量

太多的分页空间是一种不必要的浪费，但太少则会影响系统的运行效率，更有甚者可能会造成系统终止。建议在30%~70%之间。

(2)、分页空间的分布

分页空间是被频繁访问的区域，所以应当尽量放在硬盘的中心。

五、系统的备份与恢复

数据的备份和恢复是UNIX系统中一个重要的工作。在备份过程中，常常要使用compress和pack命令对数据进行压缩。恢复时用uncompress、unpack解压缩。

数据备份从方式上来说有如下三种：

系统备份 保留整个操作系统的备份映像，可用来恢复操作系统。

全备份 保留所有用户的全部数据文件和配置信息。

增强式备份 只对上次备份后的修改部分作备份，备份速度快。

在系统安装完毕后或系统有任何修改，都应该对ROOTVG作系统备份，方法是：

# smit mksysb

[TOP] [Entry Fields]

WARNING: Execution of the mksysb command will

result in the loss of all material

previously stored on the selected

output medium. This command backs

up only rootvg volume group.

\* Backup DEVICE or FILE [] +/

Create MAP files? no +

EXCLUDE files? no +

List files as they are backed up? no +

Generate new /image.data file? yes +

EXPAND /tmp if needed? no +

Disable software packing of backup? no + [MORE...2]

Esc+1=Help Esc+2=Refresh Esc+3=Cancel Esc+4=List

Esc+5=Reset Esc+6=Command Esc+7=Edit Esc+8=Image

Esc+9=Shell Esc+0=Exit Enter=Do

进入smit界面后，把device一栏设成磁带机，然后按回车就行了。

对于非rootvg卷组，可用savevg命令或smit savevg来作备份。对于一般用户可用backup命令来进行。例如：

backup [-u] [-level] [-f device] [file system]

其中允许使用参数实现增强式备份，如-0，-1，-2，...，等等，其中零级备份即表示全备份。除此以外，AIX还提供了tar、cpio、dd、copy、flcopy、tctl等，请查看随机的CD-ROM文件。

注意：数据库的备份应使用其提供的方法，才能保证数据库的完整性。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*AIX基础教程（下）\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

设备的管理和使用

一、物理设备和逻辑设备

1、物理设备是指以某种方式与计算机系统相连的实际硬件。例如显示器、终端、磁带机等。

2、逻辑设备是用户或应用程序访问物理设备的界面，用户或应用程序通过逻辑设备来访问物理设备。

3、在很多情况下，一个物理设备会对应多个逻辑设备，例如：

/dev/fd0

/dev/fd0.18

/dev/fd0h

/dev/fd0l

/dev/rfd0

/dev/rfd0.18

/dev/rfd0h

/dev/rfd0l

以上所有的逻辑设备全部对应于同一个物理设备(软盘驱动器)。此外，系统中有一些逻辑设备只是一些虚拟设备，它们不对于任何实际的物理设备，例如/dev/error和 /dev/null。

4、块设备是一种将信息存放在定长块中的设备。块设备是一种随机访问设备，用户可以随时访问设备中的任意一块。常见的块设备有：

cd0 只读光驱

fd0 软盘驱动器

hd1 lv00 逻辑卷

hdisk0 硬盘

5、字符设备是一种原始的、面向字符流的顺序存取设备，每次按顺序读写一个字符。常用的字符设备有：

console, lft0, tty 终端

rmt0 磁带机

tok0, ent0 网卡

mem 内存

rhd1, rlv00 逻辑卷

rhdisk0 物理卷

二、设备的分类层次

1、在AIX系统中，按类、子类、型号三个层次对设备进行分类。

类(class)是按设备的功能划分的，例如打印机类、硬盘类、适配卡类等。

子类(subclass)是按设备的某种相似性划分的，例如适配卡类可分为PCI子类和MCA子类。

型号(type)是按设备的机型或生产厂家划分的，例如磁带机可分为4mm12gb、 8mm20gb等型号。

2、获取设备信息

# lsdev [-option]

在AIX系统中，有两个关于设备的数据库----Predefined与Customized。Predefined数 据库中记录了AIX系统本身可以支持的所有设备种类；Customized数据库中则记录 了目前你的系统已配置/安装的设备类型。

Customized数据库中又有两类设备。一种叫Defined设备；另一种叫Available设备。 Defined设备是指它的驱动程序已安装，但系统还没有找到该设备。Available设备则 是指系统已找到该设备及其驱动程序，因此你可以直接使用这个设备。

这条命令的参数如下：

-P：查看系统Predefined数据库中的设备

-H：输出时，在第一行加上各列的Title

-c： 指出要查询的设备类型

-C：查看系统Customized数据库中的设备

例子：

# lsdev -PH

查看系统可以支持的所有设备，其输出如下：

classtype subclass description

adapterethernet isa IBM ISA Ethernet Adapter

adapterppa sio Standard I/O Parallel Port Adapter

adaptertokenring mca Token-Ring High-Performance Adapter

disk 400mb scsi 400MB SCSI Disk Drive

disk osdisk scsi Other SCSI disk

memory memory sys Memory Card

printerosp rs232 Other serial printer

tape 150mb scsi 150MB ?-Inch Tape Drive

tty tty rs232 Asynchronous Terminal

adapterner810 pci Standard SCSI I/O Controller

. . . . . .

# lsdev -Pc tape

查看系统可以支持的所有磁带机类型，其输出如下：

tape1200mb scsi1.2GB ?-Inch Tape Drive

tape150mbscsi150MB ?-Inch Tape Drive

. . . . . .

# lsdev -CH

查看系统内各个设备的信息，我们可以看到如下信息：

namestatuslocationdescription

sys0Available00-00System Object

bus0Available00-00PCI Bus

bus1Available00-10ISA Bus

ppa0Available00-00-0PStandard I/O Parallel Port Adapter

lp0Available00-00-0P-00IBM 4201 Model Proprinter II

sa0Available00-00-S1Standard I/O Serial Port 1

tty0Available00-00-S1-00Asynchronous Terminal

mem0Available00-0D16MB Memory Card

scsi0Available00-01SCSI I/O Controller

hdisk0Available00-01-00-0,0400MB SCSI Disk Drive

rmt0Defined 00-01-00-5,C150MB ?-Inch Tape Drive

ent0Available00-02Ethernet Adapter

# lsdev -Cc memory

查看所有属于memory设备的信息，其输出如下：

mem0Available00-0D16MB Memory Card

mem1Available00-0432MB Memory Card

3、获取设备的具体信息

# lsattr [-option] [resource name]

这条命令是用来获取某一设备的具体信息的。

每个逻辑设备在系统中都有一个 Location Code ，这个号码因设备的类型及与之所 连的适配卡而定。

这条命令的参数如下：

-C：从ODM库中提取customized设备的信息。

-E：看设备的具体属性。

-l ： 列出逻辑设备名。

-c ： 按类来列出设备。

例子：

# lsattr -El sys0

查看系统本身的配置信息。其输出如下：

keylocknormalState of system keylock at boot timeFalse

maxbuf20Maximum number of pages in block

I/O BUFFER CACHETrue

4、显示系统配置、诊断、VPD(vital product data)的信息

# lscfg

例如：

(1) lscfg

INSTALLED RESOURCE LIST

The following resources are installed on your machine.

+/- = Added/Deleted from Diagnostic Test List.

\* = NOT Supported by Diagnostics.

+ sysplanar0 00-00 System Planar

+ fpa0 00-00 Floating Point Processor

+ mem0 00-0A Memory Card

+ mem1 00-0B Memory Card

+ ioplanar0 00-00 I/O Planar

\* f2bus0 00-00 Micro Channel Bus

+ rs2320 00-01 RS232 Card

+ tty0 00-01-0-01 RS232 Card Port

- tty1 00-01-0-02 RS232 Card Port

(2) lscfg -v

INSTALLED RESOURCE LIST WITH VPD

The following devices are installed in your system.

sysplanar0 00-00 System Planar

Part Number.........342522

EC Level............254921

Serial Number.......353535

fpa0 00-00 Floating Point Processor

mem0 00-0A Memory Card

EC Level............990221

三、设备的状态

在AIX中，设备可以处在下列三种状态之一：

1、Undefined未定义状态

如果设备处于此状态，则系统没有该设备的任何信息。

2、Defined已定义状态

处于此状态，说明系统已经存有该设备的全部信息，并已经给设备分配好逻辑设备名和端口，但当前设备还不能被使用。

3、Available可用状态

处于此状态的设备能被正常使用。

设备可以在这三种状态之间进行转换，可通过命令mkdev -dl和 rmdev -dl来进行，或通过smit dev来实现。

例如：

(1) mkdev -l rmt0

rmt0 Available

(2) rmdev -l rmt0

rmt0 Defined

(3) rmdev -dl rmt0

rmt0 deleted

四、串行设备

串行设备是与系统串口相连的设备，一般需要手工配置。

1、显示所有已定义的TTY设备

lsdev -Cctty

smit tty

2、增加一个TTY设备

smit tty

-Add a TTY

-TTY (rs232 rs422)

-Parent Adapter(sa0 Available 00-00-s1 Standard I/O Serial Port1)

(sa1Available 00-00-s2 Standard I/O Serial Port2)

(sa2 Available 00-03-11 8 port Asynchronous Adapter)

Add a TTY

Type or select values in entry fields.

Press Enter AFTER making all desired changes.

[TOP] [Entry Fields]

TTY type tty

TTY interface rs232

Description Asynchronous Terminal

Parent adapter sa0

\* PORT number [] +

Enable LOGIN disable +

BAUD rate [9600] +

PARITY [none] +

BITS per character [8] +

Number of STOP BITS [1] +

TIME before advancing to next port setting [0] +#

TERMINAL type [dumb]

FLOW CONTROL to be used [xon] + [MORE...31]

F1=Help F2=Refresh F3=Cancel F4=List

Esc+5=Reset Esc+6=Command Esc+7=Edit Esc+8=Image

Esc+9=Shell Esc+0=Exit Enter=Do

在这个对话框中，须填写PORT number一项，使用F4键得到选项列表，按所需的来选出合适的PORT number。

3、与TTY有关的命令

pdisable [-a] [device]命令暂时禁用一个TTY设备

penable命令则解除对TTY设备的禁用

stty [-a] [-g] [options]命令设置或显示TTY设备的属性

五、磁带机和软盘的使用

1、格式化软盘

format -d /dev/rfd0

注意：在UNIX系统中格式化的软盘不能在DOS中使用。

2、软盘拷贝

flcopy命令用于将一个软盘中的内容拷贝到另一张软盘上。

3、用软盘或磁带转存文件

使用cpio命令可将硬盘文件拷贝到磁带或软盘上。

例如，将当前目录下所有以C为后缀的文件拷贝到软盘上：

ls \*.C | cpio -ov > /dev/rfd0

例如，将当前目录的所有文件和子目录拷贝到软盘上：

find . -print | cpio -ov > /dev/rfd0

使用cpio命令可将文件从软盘或磁带拷贝恢复到硬盘上。

例如，将软盘的文件拷贝到硬盘：

cpio -iv < /dev/rfd0

例如，列出fd0的文件：

cpio -itr < /dev/rfd0

例如，将软盘的文件拷贝到当前目录：

cpio -idmv < /dev/rfd0

tar命令

tar cvf

tar xvf

tar tvf

4、磁带机的属性

(1)、块大小block-size

数据在磁带上是按块来存放的，块与块之间有一定的间隔标志。当对磁带进行读写操作时，需要知道块的大小。将块大小设置为较大的值时可以向磁带写入更多的信息，因为这样做会减少块与块之间间隔的数目。当块大小设置为零时，表示磁带机使用可变长度的块。

可通过smit tape--Change/Show Characteristics of a Tape Drive来设置或查看块大小。

(2)、缓冲区use DEVICE BUFFERS during writes

当缓冲区属性设为YES时，应用程序将数据写入设备缓冲区后即被告知完成写操作。当缓冲区属性设为NO时，只有当数据真正写入磁带后，应用程序才被告知完成写操作。此值缺省为YES。

## shell中常用过的环境变量

PATH 查询程序的路径变量(要分前后顺序) PATH=/usr/bin : /etc

PS1 shell的主提示符 PS1='$PWD>'

PS2 shell的第二提示符 PS2='>'

PWD 当前的工作目录

TMOUT 无命令输入退出的等待时间(秒) TMOUT=200

TERM 终端类型 TERM=ibm3151

HOME 用户的主目录 HOME=/home/user/

EDITOR 缺省的编辑器 EDITOR=/usr/bin/vi

用命令set可列出当前shell的变量

## ls 常用参数

-a：列出包含以 . 起始的隐藏档在内的所有文件名

-t：依照文件最后修改时间之顺序，依序列出文件名

-F：列出目前目录下之文件名及其类型。/结尾表示为目录名称，\*结尾表 示为执行档，@结尾表示为 symblic link

-l：列出目录下所有文件之许可权、拥有者、文件大小、修改时间及名称

-lg：同上，并显示出文件之拥有者群组名称

-R：显示出目录下，以及其所有子目录之文件名

## shell脚本赋值=和比较=的区别

赋值=：str="hello"

比较=：if [ str = "hello" ];then

       echo $str

    fi

赋值与比较区别在于：

赋值=左右两边不能有空格

比较=左右两边必须有空格

## shell脚本中break和continue

在循环过程中，有时候需要在未达到循环结束条件时强制跳出循环，Shell使用两个命令来实现该功能：break和continue。

**break命令**

break命令允许跳出所有循环（终止执行后面的所有循环）。

下面的例子中，脚本进入死循环直至用户输入数字大于5。要跳出这个循环，返回到shell提示符下，需要使用break命令。

复制代码 代码如下:

#!/bin/bash

while :

do

    echo -n "Input a number between 1 to 5: "

    read aNum

    case $aNum in

        1|2|3|4|5) echo "Your number is $aNum!"

        ;;

        \*) echo "You do not select a number between 1 to 5, game is over!"

            break

        ;;

    esac

done

**continue**

continue命令与break命令类似，只有一点差别，它不会跳出所有循环，仅仅跳出当前循环。

对上面的例子进行修改：

复制代码 代码如下:

#!/bin/bash

while :

do

    echo -n "Input a number between 1 to 5: "

    read aNum

    case $aNum in

        1|2|3|4|5) echo "Your number is $aNum!"

        ;;

        \*) echo "You do not select a number between 1 to 5!"

            continue

            echo "Game is over!"

        ;;

    esac

done

运行代码发现，当输入大于5的数字时，该例中的循环不会结束，语句复制代码 代码如下:

echo "Game is over!"

## shell脚本中一次读取一行记录

方法1：

#!/bin/sh

cat FILENAME  |while  read LINE

do

     echo  "$LINE"

done

exit  0

一次读取一行，但是最前面的空格会被忽略。

方法2：

 #!/bin/sh

for  LINE in  ` cat FILENAME `

do

     echo  $LINE

done

exit  0

这个并不一定是一次读取一行，而是按照空格为分隔符。

方法3：

 !/bin/sh

while  read LINE

do

       echo  $LINE

done  <  FILENAME

一次读取一行，但是最前面的空格会被忽略。

## shell执行开关

#!/usr/bin/sh的选项

{

    -e  如果读入一个命令失败就立即退出

    -n  读入命令但不执行

    -u  置换时把未设置的变量看成出错

    -v  当读入shell输入行时把他们显示出来

    -x  执行命令是把他们命令和他们的参数显示出来

}

## file命令

file   core  查看core文件是由那个进程产生的

## shell判断字符串是否为空

1、条件测试

str=""

if [ -z "$str"  ];then

fi

注意一定要加“”引号

2、取字符串长度判断

  $(#str)取字符串长度

## cut命令

cut命令可以从一个文本文件或者文本流中提取文本列。

 命令用法：

       cut -b list [-n] [file ...]

       cut -c list [file ...]

       cut -f list [-d delim][-s][file ...]

     上面的-b、-c、-f分别表示字节、字符、字段（即byte、character、field）；

      list表示-b、-c、-f操作范围，-n常常表示具体数字；

      file表示的自然是要操作的文本文件的名称；

      delim（英文全写：delimiter）表示分隔符，默认情况下为TAB；

      -s表示不包括那些不含分隔符的行（这样有利于去掉注释和标题）

上面三种方式中，表示从指定的范围中提取字节（-b）、或字符（-c）、或字段（-f）。

范围的表示方法：

N

 只有第N项

N-

 从第N项一直到行尾

N-M

 从第N项到第M项(包括M)

-M

 从一行的开始到第M项(包括M)

 从一行的开始到结束的所有项

## sort 命令

**sort的工作原理**

sort将文件的每一行作为一个单位，相互比较，比较原则是从首字符向后，依次按ASCII码值进行比较，最后将他们按升序输出。

[rocrocket@rocrocket programming]$ cat seq.txt

banana

apple

pear

orange

[rocrocket@rocrocket programming]$ sort seq.txt

apple

banana

orange

pear

**2 sort的-u选项**

它的作用很简单，就是在输出行中去除重复行。

[rocrocket@rocrocket programming]$ cat seq.txt

banana

apple

pear

orange

pear

[rocrocket@rocrocket programming]$ sort seq.txt

apple

banana

orange

pear

pear

[rocrocket@rocrocket programming]$ sort -u seq.txt

apple

banana

orange

pear

pear由于重复被-u选项无情的删除了。

**3 sort的-r选项**

sort默认的排序方式是升序，如果想改成降序，就加个-r就搞定了。

[rocrocket@rocrocket programming]$ cat number.txt

1

3

5

2

4

[rocrocket@rocrocket programming]$ sort number.txt

1

2

3

4

5

[rocrocket@rocrocket programming]$ sort -r number.txt

5

4

3

2

1

**4 sort的-o选项**

由于sort默认是把结果输出到标准输出，所以需要用重定向才能将结果写入文件，形如sort filename> newfile。

但是，如果你想把排序结果输出到原文件中，用重定向可就不行了。

[rocrocket@rocrocket programming]$ sort -r number.txt> number.txt

[rocrocket@rocrocket programming]$ cat number.txt

[rocrocket@rocrocket programming]$

看，竟然将number清空了。

就在这个时候，-o选项出现了，它成功的解决了这个问题，让你放心的将结果写入原文件。这或许也是-o比重定向的唯一优势所在。

[rocrocket@rocrocket programming]$ cat number.txt

1

3

5

2

4

[rocrocket@rocrocket programming]$ sort -r number.txt -o number.txt

[rocrocket@rocrocket programming]$ cat number.txt

5

4

3

2

1

**5 sort的-n选项**

你有没有遇到过10比2小的情况。我反正遇到过。出现这种情况是由于排序程序将这些数字按字符来排序了，排序程序会先比较1和2，显然1小，所以就将10放在2前面喽。这也是sort的一贯作风。

我们如果想改变这种现状，就要使用-n选项，来告诉sort，“要以数值来排序”！

[rocrocket@rocrocket programming]$ cat number.txt

1

10

19

11

2

5

[rocrocket@rocrocket programming]$ sort number.txt

1

10

11

19

2

5

[rocrocket@rocrocket programming]$ sort -n number.txt

1

2

5

10

11

19

**6 sort的-t选项和-k选项**

如果有一个文件的内容是这样：

[rocrocket@rocrocket programming]$ cat facebook.txt

banana:30:5.5

apple:10:2.5

pear:90:2.3

orange:20:3.4

这个文件有三列，列与列之间用冒号隔开了，第一列表示水果类型，第二列表示水果数量，第三列表示水果价格。

那么我想以水果数量来排序，也就是以第二列来排序，如何利用sort实现？

幸好，sort提供了-t选项，后面可以设定间隔符。（是不是想起了cut和paste的-d选项，共鸣～～）

指定了间隔符之后，就可以用-k来指定列数了。

[rocrocket@rocrocket programming]$ sort -n -k 2 -t : facebook.txt

apple:10:2.5

orange:20:3.4

banana:30:5.5

pear:90:2.3

我们使用冒号作为间隔符，并针对第二列来进行数值升序排序，结果很令人满意。

**7 其他的sort常用选项**

-f会将小写字母都转换为大写字母来进行比较，亦即忽略大小写

-c会检查文件是否已排好序，如果乱序，则输出第一个乱序的行的相关信息，最后返回1

-C会检查文件是否已排好序，如果乱序，不输出内容，仅返回1

-M会以月份来排序，比如JAN小于FEB等等

-b会忽略每一行前面的所有空白部分，从第一个可见字符开始比较。

有时候学习脚本，你会发现sort命令后面跟了一堆类似-k1,2，或者-k1.2 -k3.4的东东，有些匪夷所思。今天，我们就来搞定它—-k选项！

**1 准备素材**

$ cat facebook.txt

google 110 5000

baidu 100 5000

guge 50 3000

sohu 100 4500

第一个域是公司名称，第二个域是公司人数，第三个域是员工平均工资。（除了公司名称，其他的别信，都瞎写的^\_^）

**2 我想让这个文件按公司的字母顺序排序，也就是按第一个域进行排序：（这个facebook.txt文件有三个域）**

$ sort -t ‘ ‘ -k 1 facebook.txt

baidu 100 5000

google 110 5000

guge 50 3000

sohu 100 4500

看到了吧，就直接用-k 1设定就可以了。（其实此处并不严格，稍后你就会知道）

**3 我想让facebook.txt按照公司人数排序**

$ sort -n -t ‘ ‘ -k 2 facebook.txt

guge 50 3000

baidu 100 5000

sohu 100 4500

google 110 5000

不用解释，我相信你能懂。

但是，此处出现了问题，那就是baidu和sohu的公司人数相同，都是100人，这个时候怎么办呢？按照默认规矩，是从第一个域开始进行升序排序，因此baidu排在了sohu前面。

**4  我想让facebook.txt按照公司人数排序 ，人数相同的按照员工平均工资升序排序：**

$ sort -n -t ‘ ‘ -k 2 -k 3 facebook.txt

guge 50 3000

sohu 100 4500

baidu 100 5000

google 110 5000

看，我们加了一个-k2 -k3就解决了问题。对滴，sort支持这种设定，就是说设定域排序的优先级，先以第2个域进行排序，如果相同，再以第3个域进行排序。（如果你愿意，可以一直这么写下去，设定很多个排序优先级）

**5 我想让facebook.txt按照员工工资降序排序，如果员工人数相同的，则按照公司人数升序排序：（这个有点难度喽）**

$ sort -n -t ‘ ‘ -k 3r -k 2 facebook.txt

baidu 100 5000

google 110 5000

sohu 100 4500

guge 50 3000

此处有使用了一些小技巧，你仔细看看，在-k 3后面偷偷加上了一个小写字母r。你想想，再结合我们[上一篇文章](http://roclinux.cn/?p=1350)，能得到答案么？揭晓：r和-r选项的作用是一样的，就是表示逆序。因为sort默认是按照升序排序的，所以此处需要加上r表示第三个域（员工平均工资）是按照降序排序。此处你还可以加上n，就表示对这个域进行排序时，要按照数值大小进行排序，举个例子吧：

$ sort -t ‘ ‘ -k 3nr -k 2n facebook.txt

baidu 100 5000

google 110 5000

sohu 100 4500

guge 50 3000

看，我们去掉了最前面的-n选项，而是将它加入到了每一个-k选项中了。

**6 -k选项的具体语法格式**

要继续往下深入的话，就不得不来点理论知识。你需要了解-k选项的语法格式，如下：

[ FStart [ .CStart ] ] [ Modifier ] [ , [ FEnd [ .CEnd ] ][ Modifier ] ]

这个语法格式可以被其中的逗号（“，”）分为两大部分，Start部分和End部分。

先给你灌输一个思想，那就是“如果不设定End部分，那么就认为End被设定为行尾”。这个概念很重要的，但往往你不会重视它。

Start部分也由三部分组成，其中的Modifier部分就是我们之前说过的类似n和r的选项部分。我们重点说说Start部分的FStart和C.Start。

C.Start也是可以省略的，省略的话就表示从本域的开头部分开始。之前例子中的-k 2和-k 3就是省略了C.Start的例子喽。

FStart.CStart，其中FStart就是表示使用的域，而CStart则表示在FStart域中从第几个字符开始算“排序首字符”。

同理，在End部分中，你可以设定FEnd.CEnd，如果你省略.CEnd，则表示结尾到“域尾”，即本域的最后一个字符。或者，如果你将CEnd设定为0(零)，也是表示结尾到“域尾”。

**7 突发奇想，从公司英文名称的第二个字母开始进行排序：**

$ sort -t ‘ ‘ -k 1.2 facebook.txt

baidu 100 5000

sohu 100 4500

google 110 5000

guge 50 3000

看，我们使用了-k 1.2，这就表示对第一个域的第二个字符开始到本域的最后一个字符为止的字符串进行排序。你会发现baidu因为第二个字母是a而名列榜首。sohu和 google第二个字符都是o，但sohu的h在google的o前面，所以两者分别排在第二和第三。guge只能屈居第四了。

**8 又突发奇想，，只针对公司英文名称的第二个字母进行排序，如果相同的按照员工工资进行降序排序：**

$ sort -t ‘ ‘ -k 1.2,1.2 -k 3,3nr facebook.txt

baidu 100 5000

google 110 5000

sohu 100 4500

guge 50 3000

由于只对第二个字母进行排序，所以我们使用了-k 1.2,1.2的表示方式，表示我们“只”对第二个字母进行排序。（如果你问“我使用-k 1.2怎么不行？”，当然不行，因为你省略了End部分，这就意味着你将对从第二个字母起到本域最后一个字符为止的字符串进行排序）。对于员工工资进行排 序，我们也使用了-k 3,3，这是最准确的表述，表示我们“只”对本域进行排序，因为如果你省略了后面的3，就变成了我们“对第3个域开始到最后一个域位置的内容进行排序” 了。

**9 在modifier部分还可以用到哪些选项？**

可以用到b、d、f、i、n 或 r。

其中n和r你肯定已经很熟悉了。

b表示忽略本域的签到空白符号。

d表示对本域按照字典顺序排序（即，只考虑空白和字母）。

f表示对本域忽略大小写进行排序。

i表示忽略“不可打印字符”，只针对可打印字符进行排序。（有些ASCII就是不可打印字符，比如\a是报警，\b是退格，\n是换行，\r是回车等等）

**10 思考思考关于-k和-u联合使用的例子：**

$ cat facebook.txt

google 110 5000

baidu 100 5000

guge 50 3000

sohu 100 4500

这是最原始的facebook.txt文件。

$ sort -n -k 2 facebook.txt

guge 50 3000

baidu 100 5000

sohu 100 4500

google 110 5000

$ sort -n -k 2 -u facebook.txt

guge 50 3000

baidu 100 5000

google 110 5000

当设定以公司员工域进行数值排序，然后加-u后，sohu一行就被删除了！原来-u只识别用-k设定的域，发现相同，就将后续相同的行都删除。

$ sort  -k 1 -u facebook.txt

baidu 100 5000

google 110 5000

guge 50 3000

sohu 100 4500

$ sort  -k 1.1,1.1 -u facebook.txt

baidu 100 5000

google 110 5000

sohu 100 4500

这个例子也同理，开头字符是g的guge就没有幸免于难。

$ sort -n -k 2 -k 3 -u facebook.txt

guge 50 3000

sohu 100 4500

baidu 100 5000

google 110 5000

咦！这里设置了两层排序优先级的情况下，使用-u就没有删除任何行。原来-u是会权衡所有-k选项，将都相同的才会删除，只要其中有一级不同都不会轻易删除的:)（不信，你可以自己加一行sina 100 4500试试看）

**11 最诡异的排序：**

$ sort -n -k 2.2,3.1 facebook.txt

guge 50 3000

baidu 100 5000

sohu 100 4500

google 110 5000

以第二个域的第二个字符开始到第三个域的第一个字符结束的部分进行排序。

第一行，会提取0 3，第二行提取00 5，第三行提取00 4，第四行提取10 5。

又因为sort认为0小于00小于000小于0000….

因此0 3肯定是在第一个。10 5肯定是在最后一个。但为什么00 5却在00 4前面呢？（你可以自己做实验思考一下。）

答案揭晓：原来“跨域的设定是个假象”，sort只会比较第二个域的第二个字符到第二个域的最后一个字符的部分，而不会把第三个域的开头字符纳入比较范围。当发现00和00相同时，sort就会自动比较第一个域去了。当然baidu在sohu前面了。用一个范例即可证实：

$ sort -n -k 2.2,3.1 -k 1,1r facebook.txt

guge 50 3000

sohu 100 4500

baidu 100 5000

google 110 5000

## ftp命令

FTP命令是Internet用户使用最频繁的命令之一，不论是在DOS还是UNIX操作系统下使用FTP，都会遇到大量的FTP内部命令。熟悉并灵活应用FTP的内部命令，可以大大方便使用者，并收到事半功倍之效。下面以Win9x的DOS窗口中的FTP命令为例，介绍一下其用法。（本站注：若想在纯DOS下使用FTP命令，则需先加载网络设备如网卡或调制解调器的Packet驱动程序，再使用本站“网络工具”中的Wattcp FTP程序，其用法与下面介绍的类似，详情请见本站的“DOS使用”栏目）

FTP的命令行格式为： ftp -v -d -i -n -g [主机名] ，其中

-v 显示远程服务器的所有响应信息；

-n 限制ftp的自动登录，即不使用；

.n etrc文件；

-d 使用调试方式；

-g 取消全局文件名。

FTP使用的内部命令如下(中括号表示可选项):

1.![cmd[args]：在本地机中执行交互shell，exit回到ftp环境，如：!ls\*.zip

2.$ macro-ame[args]： 执行宏定义macro-name。

3.account[password]： 提供登录远程系统成功后访问系统资源所需的补充口令。

4.append local-file[remote-file]：将本地文件追加到远程系统主机，若未指定远程系统文件名，则使用本地文件名。

5.ascii：使用ascii类型传输方式。

6.bell：每个命令执行完毕后计算机响铃一次。

7.bin：使用二进制文件传输方式。

8.bye：退出ftp会话过程。

9.case：在使用mget时，将远程主机文件名中的大写转为小写字母。

10.cd remote-dir：进入远程主机目录。

11.cdup：进入远程主机目录的父目录。

12.chmod mode file-name：将远程主机文件file-name的存取方式设置为mode，如：chmod 777 a.out。

13.close：中断与远程服务器的ftp会话(与open对应)。

14.cr：使用asscii方式传输文件时，将回车换行转换为回行。

15.delete remote-file：删除远程主机文件。

16.debug[debug-value]：设置调试方式， 显示发送至远程主机的每条命令，如：deb up 3，若设为0，表示取消debug。

17.dir[remote-dir][local-file]：显示远程主机目录，并将结果存入本地文件local-file。

18.disconnection：同close。

19.form format：将文件传输方式设置为format，缺省为file方式。

20.get remote-file[local-file]： 将远程主机的文件remote-file传至本地硬盘的local-file。

21.glob：设置mdelete，mget，mput的文件名扩展，缺省时不扩展文件名，同命令行的-g参数。

22.hash：每传输1024字节，显示一个hash符号(#)。

23.help[cmd]：显示ftp内部命令cmd的帮助信息，如：help get。

24.idle[seconds]：将远程服务器的休眠计时器设为[seconds]秒。

25.image：设置二进制传输方式(同binary)。

26.lcd[dir]：将本地工作目录切换至dir。

27.ls[remote-dir][local-file]：显示远程目录remote-dir， 并存入本地文件local-file。

28.macdef macro-name：定义一个宏，遇到macdef下的空行时，宏定义结束。

29.mdelete[remote-file]：删除远程主机文件。

30.mdir remote-files local-file：与dir类似，但可指定多个远程文件，如：mdir \*.o.\*.zipoutfile 。

31.mget remote-files：传输多个远程文件。

32.mkdir dir-name：在远程主机中建一目录。

33.mls remote-file local-file：同nlist，但可指定多个文件名。

34.mode[modename]：将文件传输方式设置为modename， 缺省为stream方式。

35.modtime file-name：显示远程主机文件的最后修改时间。

36.mput local-file：将多个文件传输至远程主机。

37.newer file-name： 如果远程机中file-name的修改时间比本地硬盘同名文件的时间更近，则重传该文件。

38.nlist[remote-dir][local-file]：显示远程主机目录的文件清单，并存入本地硬盘的local-file。

39.nmap[inpattern outpattern]：设置文件名映射机制， 使得文件传输时，文件中的某些字符相互转换， 如：nmap $1.$2.$3[$1，$2].[$2，$3]，则传输文件a1.a2.a3时，文件名变为a1，a2。 该命令特别适用于远程主机为非UNIX机的情况。

40.ntrans[inchars[outchars]：设置文件名字符的翻译机制，如ntrans1R，则文件名LLL将变为RRR。

41.open host[port]：建立指定ftp服务器连接，可指定连接端口。

42.passive：进入被动传输方式。

43.prompt：设置多个文件传输时的交互提示。

44.proxy ftp-cmd：在次要控制连接中，执行一条ftp命令， .,该命令允许连接两个ftp服务器，以在两个服务器间传输文件。第一条ftp命令必须为open，以首先建立两个服务器间的连接。

45.put local-file[remote-file]：将本地文件local-file传送至远程主机。

46.pwd：显示远程主机的当前工作目录。

47.quit：同bye，退出ftp会话。

48.quote arg1，arg2...：将参数逐字发至远程ftp服务器，如：quote syst.

49.recv remote-file[local-file]：同get。

50.reget remote-file[local-file]：类似于get， 但若local-file存在，则从上次传输中断处续传。

51.rhelp[cmd-name]：请求获得远程主机的帮助。

52.rstatus[file-name]：若未指定文件名，则显示远程主机的状态， 否则显示文件状态。

53.rename[from][to]：更改远程主机文件名。

54.reset：清除回答队列。

55.restart marker：从指定的标志marker处，重新开始get或put，如：restart 130。

56.rmdir dir-name：删除远程主机目录。

57.runique：设置文件名唯一性存储，若文件存在，则在原文件后加后缀.1，.2等。

58.send local-file[remote-file]：同put。

59.sendport：设置PORT命令的使用。

60.site arg1，arg2...：将参数作为SITE命令逐字发送至远程ftp主机。

61.size file-name：显示远程主机文件大小，如：site idle 7200。

62.status：显示当前ftp状态。

63.struct[struct-name]：将文件传输结构设置为struct-name， 缺省时使用stream结构。

64.sunique：将远程主机文件名存储设置为唯一(与runique对应)。

65.system：显示远程主机的操作系统类型。

66.tenex：将文件传输类型设置为TENEX机的所需的类型。

67.tick：设置传输时的字节计数器。

68.trace：设置包跟踪。

69.type[type-name]：设置文件传输类型为type-name，缺省为ascii，如：type binary，设置二进制传输方式。

70.umask[newmask]：将远程服务器的缺省umask设置为newmask，如：umask 3

71.user user-name[password][account]：向远程主机表明自己的身份，需要口令时，必须输入口令，如：user anonymous my@email。

72.verbose：同命令行的-v参数，即设置详尽报告方式，ftp 服务器的所有响应都将显示给用户，缺省为on.

73.?[cmd]：同help.

## sftp与ftp

sftp   用户名@IP

ftp   IP,ftp时一定要采用bin模式传输

## cat -v 显示文件中不可以字符

cat  -v  filename  可以查看文件中有没有回车换行符

cat  -v filename  | tr -d "^M"  >  tmp  去掉文件中的回车换行符

## netstat网络命令

netstat -i 显示网络连接信息及统计信息

netstat -rn 显示核心路由表

netstat -I 网络设置名 端口号 监视端口情况

netstat -m 网络使用的内存空间情况

netstat -in 查看所有网卡IP地址

netstat -r  查看所有路由

ifconfig -a 查看所有网络配置信息

netstat -D  查看所有通讯包的统计

netstat -s -f inet 查看每种协议传输的数据量统计

netstat -v         查看每块网卡传输的数据量和状态统计

traceroute 10.188.182.1 跟踪IP

netpmon 可以监控关于网络行为的系统事件和性能以及网络行为对CPU的消耗。

netpmon -v 物理/逻辑资源的详细报告

ping 10.188.12.252 -l 17000 -t 以17000 bytes/包 ping 目标

增加并配置端口

　　route ADD 0 10.188.12.1

　　route add default 192.168.0.1 设置网关(或在/etc/defaultrouter文件中加入网关地址，重起机器就行)

arp -a 查看解析协议ARP表

## 实时监控程序内存使用情况

while  true;

do

ps aux|head -1

ps aux|grep $USER|grep billdwnsvr|grep -v grep

sleep 1

done

## 清除ipc资源

ipcs  -mqs|grep $USER|awk '{print "ipcrm -"  $1$2}'|sh –x

## lsdev命令

lsdev -Cc disk 系统设备信息（磁盘）

lsdev -Cc pdisk 显示阵列里的磁盘信息

lsdev -P 列出所有的设备, lsdev -Pc disk

lsdev -C -c if 显示网络接口描述

lsdev -C -c adapetr 显示适配描述

lsdev -C|grep Process|wc -l 显示CPU数量

lsdev -C           列出所有系统设备(包括所有状态：Undefined,Defined,Available)

lsdev -Cc adapter  列出系统现有设备

lsdev -PH          列出系统所有支持的设备

lsdev -P -r class  列出系统所有支持的设备分类(例如：swap,sys,tape,tty,printer,mouse,memory,lvm等)

lsdev -P -c disk -r subclass 列出系统所有支持的设备分类(磁盘)的子类(例如：SCSI,DAR,IDE,ISCSI等)

lsdev -P -c tape             列出所有支持的磁带设备

lsdev -l hdi\*                列出所有系统的SCSI Disk Drive

## ftp://root@10.188.12.250/ 在客户端登录AIX(用IE)

ftp://root@10.188.12.250/ 在客户端登录AIX(用IE)

## /etc/filesystems 记录所有的文件系统设置

/etc/filesystems 记录所有的文件系统设置

## /etc/inittab 相当于DOS的AUTOEXEC.BAT文件

/etc/inittab 相当于DOS的AUTOEXEC.BAT文件

## /etc/hosts 机器名IP对照表

/etc/hosts 机器名IP对照表

## /ect/services 查看端口

/ect/services 查看端口

## nohup command &与command &

nohup command & 使用户的后台进程在用户退出时仍然运行

command & 后台进程

## shell中$特殊变量的含义

$$ 表示当前进程的ID

$0 当前shell程序的名称

$# 传给当前shell Script的参数个数；

$\* 传给当前shell Script的第\*个参数,$1-$9,${10}......

$? 最近一个命令的返回值；

$! 最近一个后台进程的ID号;

## 双引号"" , 单引号'' ， `` 区别

``:``之间的内容作为一个命令,返回命令结果 如$now=`date`

单引号''  :   直接显示''间的内容，不予解释

双引号""  :   解释""间变量的内容

例如：

str=abc

echo "$str"

显示:abc

echo '$str'

显示:$str

## unset 与set

unset xxx 删除变量

set 命令：

功能说明：设置shell。

语  法：set [+-abCdefhHklmnpPtuvx]

补充说明：

用set 命令可以设置各种shell选项

或者列 出shell变量.

单个选项设置常用的特性.在某些选项之后-o参数将特殊特性打开.

在某些选项之后使用+o参数将关闭某些特性,

不带任何参数的set命 令将显示shell的全部变量

.除非遇到非法的选项,否则set总是返回ture

参  数：

allexport -a从设置开始标记所有新的和修改过的用于输出的变量

braceexpand  -B  允许符号扩展,默认选项

emacs 在进行命令编辑的时候,使用内建的emacs编辑器, 默认选项

errexit 如果一个命令返回一个非0退出状态值(失败),就退出.

histexpand  -H 在做临时替换的时候允许使用!和!! 默认选项

history 允许命令行历史,默认选项

ignoreeof 禁止coontrol-D的方式退出shell，必须输入exit。

interactive-comments  在交互式模式下， #用来表示注解

keyword   -k 为命令把关键字参数放在环境中

monitor  -m  允许作业控制

noclobber  -C  保护文件在使用重新动向的时候不被覆盖

noexec  -n  在脚本状态下读取命令但是不执行，主要为了检查语法结构。

noglob -d  禁止路径名扩展，即关闭通配符

notify -b  在后台作业以后通知客户

nounset -u在扩展一个没有的设置的变量的时候，    显示错误的信息

onecmd   -t 在读取并执行一个新的命令后退出

physical              -P                       如果被设置，则在使用pwd和cd命令时不使用符号连接的路径 而是物理路径

posix     改变shell行为以便符合POSIX要求

privileg 一旦被设置，shell不再读取.profile文件和env文件 shell函数也不继承任何环境

verbose  -v 为调试打开verbose模式

vi   在命令行编辑的时候使用内置的vi编辑器

xtrace  -x 打开调试回响模式

## aix 管理命令

正如您所知道的，AIX® 具有无数的命令，使您能够执行大量的任务。取决于您需要完成的任务，您可能会仅仅使用到这些命令的某个子集。这些子集随不同的用户和不同的需要而异。然而，有一些您通常要使用的核心命令。您需要这些命令来回答您自己的问题，或者为支持人员的询问提供答案。

在本文中，我将讨论这其中的一些核心命令。其目的旨在为您提供一个可用作便捷参考的列表。虽然这些命令的行为在所有 AIX 版本中都应该相同，但是仅在 AIX 5.3 下对它们进行了测试。

注意：

以下段落中讨论的 bootinfo 不是用户级别的命令，并且在 AIX 4.2 或更高版本中不受支持。

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

回页首

命令

内核

如何知道自己在运行 32 位内核还是 64 位内核？

要显示内核启用的是 32 位还是 64 位，可输入以下命令：

bootinfo -K

如何知道自己在运行单处理器还是多处理器内核？

/unix 是指向已启动内核的符号链接。要了解正在运行什么内核模式，可输入 ls -l /unix 并查看 /unix 链接到什么文件。下面是 ls -l /unix 命令的三种可能输出及其对应的内核：

/unix -> /usr/lib/boot/unix\_up           # 32 bit uniprocessor kernel

/unix -> /usr/lib/boot/unix\_mp           # 32 bit multiprocessor kernel

/unix -> /usr/lib/boot/unix\_64           # 64 bit multiprocessor kernel

注意：

AIX 5L Version 5.3 不支持单处理器内核。

如何从一种内核模式更改为另一种内核模式？

在安装过程期间，会缺省启用一种适合该 AIX 版本和操作中的硬件的内核。让我们使用前一个问题中的方法并假设启用了 32 位内核。我们还假设您希望在 64 位内核模式下启动。这可以通过按顺序执行以下命令来实现：

ln -sf /usr/lib/boot/unix\_64    /unix

ln -sf /usr/lib/boot/unix\_64    /usr/lib/boot/unix

bosboot -ad  /dev/hdiskxx

shutdown -r

/dev/hdiskxx 目录是启动逻辑卷 /dev/hd5 所在的位置。要弄清 hdiskxx 中有哪些 xx，可运行以下命令：

lslv -m hd5

注意：

在 AIX 5.2 中，缺省安装的是 32 位内核。在 AIX 5.3 中，缺省情况下会在 64 位硬件上安装 64 位内核，在 32 位硬件上安装 32 位内核。

硬件

如何知道我的计算机是否能够运行 AIX 5L Version 5.3？

AIX 5L Version 5.3 可在当前受支持的所有基于共用硬件参考平台（Common Hardware Reference Platform，CHRP）的 POWER 硬件上运行。

如何知道我的计算机是否基于 CHRP？

运行 prtconf 命令。如果是 CHRP 计算机，则字符串 chrp 会出现在 Model Architecture 行上。

如何知道我的 System p 计算机（硬件）是 32 位还是 64 位？

要显示硬件 32 位还是 64 位，可输入以下命令：

bootinfo -y

我的计算机有多少实际内存？

要显示以 KB 为单位的实际内存，可输入以下命令之一：

bootinfo -r

lsattr -El sys0 -a realmem

我的计算机是否可以运行 64 位内核？

需要 64 位硬件才能运行 64 位内核。

我系统中的设备属性值是什么？

要列出磁带设备 rmt0 的当前属性值，可输入以下命令：

lsattr -l rmt0 -E

要列出磁带设备 rmt0 的缺省属性值，可输入以下命令：

lsattr -l rmt0 -D

要列出 TTY 设备 tty0 的可能登录属性值，可输入以下命令：

lsattr -l tty0 -a login -R

要显示系统级别的属性，可输入以下命令：

lsattr -E -l sys0

我的系统有多少个处理器？

要显示您系统上的处理器数量，可输入以下命令：

lscfg | grep proc

我的系统上有多少个硬盘，当前正在使用哪些硬盘？

要显示系统上的硬盘数量，可输入以下命令：

lspv

如何列出有关特定物理卷的信息？

举例来说，若要了解有关 hdisk1 的详细信息，可运行如下命令：

lspv hdisk1

如何获得系统的详细配置？

输入以下命令：

lscfg

下列选项可以提供特定的信息：

-p     显示特定于平台的设备信息。该标志适用于 AIX 4.2.1 或更高版本。

-v     显示在自定义 VPD 对象类中找到的重要产品数据库（Vital Product Database，VPD）。

例如，要显示有关磁带驱动器 rmt0 的详细信息，可输入以下命令：

lscfg -vl rmt0

通过运行 prtconf 命令也可以获得非常类似的信息。

如何确定芯片类型、系统名称、节点名称、型号，等等？

uname 命令可以提供关于系统的详细信息。

uname -p     显示系统的芯片类型。例如，PowerPC。

uname -r     显示操作系统的版本号。

uname -s     显示系统名称。例如，AIX。

uname -n     显示节点名称。

uname -a     显示系统名称、节点名称、版本、计算机 ID。

uname -M     显示系统型号名称。例如，IBM, 9114-275。

uname -v     显示操作系统版本。

uname -m     显示运行系统的硬件的计算机 ID 编号。

uname -u     显示系统 ID 编号。

AIX

我的系统上在运行什么 AIX 主要版本、次要版本和维护级？

输入以下命令之一：

oslevel -r

lslpp -h bos.rte

如何确定某个特定的 AIX 级别缺少哪些文件集更新？

举例来说，若要确定 5300-04 缺少哪些文件集更新，可运行以下命令：

oslevel -rl 5300-04

我的系统上安装了什么 SP (Service Pack)？

要了解系统上当前安装了哪个 SP，可运行 oslevel -s 命令。对于安装了 TL4 和 SP2 的 AIX 5L Version 5.3 系统，示例输出如下：

oslevel –s

5300-04-02

我的系统上是否安装了 CSP (Concluding Service Pack)？

要确定系统上当前是否安装了某个 CSP，可运行 oslevel -s 命令。对于安装了 TL3 和 CSP 的 AIX 5L Version 5.3 系统，示例输出如下：

oslevel –s

5300-03-CSP

如何创建文件系统？

以下命令将在卷组 testvg 中创建一个大小为 10MB、安装点为 /fs1 的 jfs 文件系统：

crfs -v jfs -g testvg -a size=10M -m /fs1

以下命令将在卷组 testvg 中创建一个大小为 10MB、安装点为 /fs2 并具有只读权限的 jfs2 文件系统：

crfs -v jfs2 -g testvg -a size=10M -p ro -m /fs2

如何更改文件系统的大小？

若要将 /usr 文件系统的大小增加 1000000 个 512 字节的块，可输入以下命令：

chfs -a size=+1000000 /usr

注意：

在 AIX 5.3 中，JFS2 文件系统的大小还可以收缩。

如何安装 CD？

输入以下命令：

mount -V cdrfs -o ro /dev/cd0  /cdrom

如何安装文件系统？

以下命令将在 /test 目录中安装文件系统 /dev/fslv02：

mount /dev/fslv02 /test

如何安装所有缺省文件系统（/etc/filesystems 文件中标记有 mount=true 属性的所有标准文件系统）？

以下命令将安装所有此类文件系统：

mount {-a|all}

如何卸载文件系统？

输入以下命令可以卸载 /test 文件系统：

umount /test

如何显示已安装的文件系统？

输入以下命令可以显示有关所有当前已安装的文件系统的信息：

mount

如何删除文件系统？

输入以下命令可以删除 /test 文件系统：

rmfs /test

如何对文件系统进行碎片整理？

可以使用 defragfs 命令来改善或报告文件系统中的连续空间状态。例如，若要对文件系统 /home 进行碎片整理，可以使用以下命令：

defragfs /home

哪个文件集包含某个特定的二进制文件？

若要显示 /usr/bin/vmstat 包含 bos.acct，可输入以下命令：

lslpp -w /usr/bin/vmstat

或者若要显示 bos.perf.tools 包含 /usr/bin/svmon，可输入以下命令：

which\_fileset svmon

如何显示有关系统上已安装文件集的信息？

输入以下命令：

lslpp -l

如何确定我的系统上是否安装了所有维护级文件集？

输入以下命令：

instfix -i | grep ML

如何确定我的系统上是否安装了某个修复程序？

若要确定是否安装了 IY24043，可输入以下命令：

instfix -ik IY24043

如何按 APAR 安装单独的修复程序？

举例来说，若要从 /dev/cd0 安装 APAR IY73748，可输入以下命令：

instfix -k IY73748 -d /dev/cd0

如何验证文件集是否有必需的先决条件和是否已完全安装？

要显示需要安装或纠正哪些文件集，可输入以下命令：

lppchk -v

如何获得符号表示中的 loader 节头和符号条目的转储？

输入以下命令：

dump -Htv

如何确定已分配和使用的分页空间量？

输入以下命令：

lsps -a

如何增加分页空间？

可以使用 chps -s 命令来动态增加分页空间的大小。例如，如果希望将 hd6 的大小增加 3 个逻辑分区，您可以执行以下命令：

chps -s 3 hd6

如何减少分页空间？

可以使用 chps -d 命令来动态减少分页空间的大小。例如，如果希望将 hd6 的大小减少四个逻辑分区，您可以执行以下命令：

chps -d 4 hd6

如何知道我的系统是否能够使用同步多线程（Simultaneous Multi-threading，SMT）？

如果您的系统是运行 AIX 5L Version 5.3 的基于 POWER5 的系统，则它就能使用 SMT。

如何知道我的系统是否启用了 SMT？

如果不带任何选项运行 smtctl 命令，它将告诉您是否启用了 SMT。

32 位内核是否支持 SMT？

是的，32 位和 64 位内核都支持 SMT。

如何启用或禁用 SMT？

可以通过运行 smtctl 命令来启用或禁用 SMT。下面是该命令的语法：

smtctl [ -m off | on [ -w boot | now]]

可以使用以下选项：

-m off     将 SMT 模式设置为禁用。

-m on     将 SMT 模式设置为启用。

-w boot      如果在下一次系统重新启动前运行 bosboot 命令，则此选项使 SMT 模式更改在下一次和后续重新启动时生效。

-w now     使 SMT 模式更改立即生效，但不会延续到下一次重新启动以后。

如果既没有指定 -w boot 也没有指定 -w now 选项，则模式更改立即生效。如果在下一次系统重新启动前运行 bosboot 命令，所做的更改将延续到后续重新启动以后。

如何获得特定于分区的信息和统计信息？

lparstat 命令可以提供分区信息和利用率统计信息报告。此命令还可以显示 Hypervisor 信息。

卷组和逻辑卷

如何知道我的卷组是常规、大容量还是可扩展的？

对该卷组运行 lsvg 命令并查看 MAX PVs 的值。该值为 32 表示常规、128 表示大容量、1024 表示可扩展的卷组。

如何创建卷组？

可以使用以下命令，其中 spartition\_size 设置每个物理分区中的兆字节 (MB) 数，并且 partition\_size 是以 MB 为单位、从 1 到 1024 的值（对于 AIX 5.3 是从 1 到 131072）。partition\_size 变量必须等于 2 的幂（例如：1、2、4、8）。标准和大容量卷组的缺省值是保持在“每个物理卷最多只能有 1016 个物理分区”限制内的最低值。可扩展卷组的缺省值为每个物理卷容纳 2040 个物理分区的最低值。

mkvg -y name\_of\_volume\_group -s partition\_size list\_of\_hard\_disks

如何更改卷组的特征？

可以使用以下命令来更改卷组的特征：

chvg

如何创建逻辑卷？

输入以下命令：

mklv -y name\_of\_logical\_volume name\_of\_volume\_group number\_of\_partition

如何增加逻辑卷的大小？

举例来说，若要将 lv05 目录所表示的逻辑卷增加三个逻辑分区，可输入以下命令：

extendlv lv05 3

如何显示属于某个卷组（例如，rootvg）的所有逻辑卷？

可以通过输入以下命令来显示属于 rootvg 的所有逻辑卷：

lsvg -l rootvg

如何列出有关逻辑卷的信息？

可运行以下命令来显示有关逻辑卷 lv1 的信息：

lslv lv1

如何删除逻辑卷？

可以运行以下命令来删除逻辑卷 lv7：

rmlv lv7

rmlv 命令仅删除逻辑卷，但不删除其他实体，例如使用该逻辑卷的文件系统或分页空间。

如何镜像逻辑卷？

1.     mklvcopy LogicalVolumeName Numberofcopies

2.     syncvg VolumeGroupName

如何删除逻辑卷的副本？

可以使用 rmlvcopy 命令来删除逻辑卷的逻辑分区副本。若要减少属于逻辑卷 testlv 的每个逻辑分区的副本数量，可输入以下命令：

rmlvcopy testlv 2

现在该逻辑卷中每个逻辑分区最多只有两个物理分区。

有关卷组的问题

要显示系统中的卷组，可输入以下命令：

lsvg

要显示 rootvg 的所有特征，可输入以下命令：

lsvg rootvg

要显示 rootvg 所使用的磁盘，可输入以下命令：

lsvg -p rootvg

如何将磁盘添加到卷组？

输入以下命令：

extendvg   VolumeGroupName   hdisk0 hdisk1 ... hdiskn

如何确定我的硬盘所支持的最大逻辑磁道组 (LTG) 大小？

可以带 -M 标志使用 lquerypv 命令。此命令的输出给出以 KB 为单位的 LTG 大小。例如，在下面的示例中，hdisk0 的 LTG 大小为 256 KB。

/usr/sbin/lquerypv -M hdisk0

256

还可以在硬盘上运行 lspv 并查看 MAX REQUEST 的值。

syncvg 命令是做什么用的？

syncvg 命令用于同步过时的物理分区。它接受逻辑卷、物理卷或卷组名称作为参数。

例如，若要同步位于物理卷 hdisk6 和 hdisk7 上的物理分区，可以使用以下命令：

syncvg -p hdisk4 hdisk5

若要同步卷组 testvg 中的所有物理分区，可以使用以下命令：

syncvg -v testvg

如何替换某个磁盘？

1.     extendvg VolumeGroupName hdisk\_new

2.     migratepv hdisk\_bad hdisk\_new

3.     reducevg -d VolumeGroupName hdisk\_bad

如何克隆 rootvg（创建其副本）？

您可以运行 alt\_disk\_copy 命令来将当前 rootvg 复制到某个替代磁盘。下面的示例演示了如何将 rootvg 克隆到 hdisk1。

alt\_disk\_copy -d  hdisk1

网络

如何显示或设置网络参数值？

no 命令设置或显示网络优化参数的当前或下一次启动时的值。

如何获得我计算机的 IP 地址？

输入以下命令之一：

ifconfig -a

host Fully\_Qualified\_Host\_Name

例如，输入主机 cyclop.austin.ibm.com。

如何确定服务器上的网络接口？

以下两个命令中的任何一个都将显示网络接口：

lsdev -Cc if

ifconfig -a

若要获得有关某个特定网络接口（例如，tr0）的信息，可以运行以下命令：

ifconfig tr0

如何激活网络接口？

若要激活网络接口 tr0，可以运行以下命令：

ifconfig tr0 up

如何禁用网络接口？

举例来说，若要禁用网络接口 tr0，可以运行以下命令：

ifconfig tr0 down

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

AIX系统日常管理

1． 文件系统是否满

方法： df –k 可以以K为单位检查文件系统的使用率。（90%以上，需要调整）

2． 检查系统出错日志 使用errpt |more来检查

清除现有的log: Errclear 0

3． 检查系统合法/非法登陆情况

使用Last命令来检查来自登陆的地方。

4． 检查系统是否有巨大的Core文件生成

使用 find / -name core –print来检查。对Core文件，一般直接删除就可以了。

5． 系统性能检查：

a) CPU性能：使用Vmstat, topas来检查

b) 内存使用情况：也是使用topas, vmstat来检查

c) 检查IO平衡使用情况：使用iostat来检查

d) 交换空间使用情况：使用lsps –a来检查

6． Mail检查

7． Diag 一个月一次

第 2 楼：AIX的版本号代表什么意思?

AIX的版本号的格式通常为AIX x.x.x.x.例如: AIX 4.3.3.1.其代表意义分别为:

操作系统版本号(version).发行版号(release).改进版号(modification).修正版号(fix)

第 3 楼：AIX资源监控与调制工具

本文出自:http://www.ccw.com.cn/ 作者: 陆成新 (2002-04-15 06:02:01)

性能优化以及确定系统中的性能瓶颈是系统管理员的主要任务之一。在一个计算机系统中，CPU、内存、硬盘和网络是影响系统性能的主要因素，因此系统性能调整也主要在于如何在这些资源中获得某种平衡，以满足人们对系统性能的期望。性能调制需要很多技巧，知识以及经验，不能仅靠分析统计数字，图表就可取得，性能调制有时是一件复杂甚至是非常困难的任务。

如同其它UNIX系统一样，AIX也给系统管理员剪裁系统提供了非常丰富的手段。这里我们简单介绍 RS/6000 AIX系统中几个用于监控和调制多项系统资源的工具，每个工具的功能都很强，如想更透彻地了解这些命令的用法，请参考有关技术资料或手册。这里讲述的命令将不仅仅局限于CPU、硬盘、内存或网络资源的某个方面，它们可用于其中的一项或多项资源。

AIX监控工具

1、iostat

iostat命令主要通过观察物理磁盘的活跃时间以及他们的平均传输速度，监控系统输入/输出设备负载。根据iostat命令产生的报告，用户可确定一个系统配置是否平衡，并据此在物理磁盘与适配器之间更好地平衡输入/输出负载。

iostat 工具的主要目的是通过监控磁盘的利用率（tm\_act字段），而探测到系统中的I/O瓶颈。iostat还可用于确定CPU问题，辅助容量规划，并可以为最终解决I/O问题提供相关材料。vmstat和iostat联合使用，可捕获到确定与CPU，内存和I/O子系统有关的性能问题的必需数据。

iostat命令可产生下面四种类型的报告：

• tty和CPU利用情况

• 磁盘的利用情况

• 系统吞吐率

• 适配器吞吐率

2、netpmon

netpmon命令可以监控关于网络行为的系统事件和性能以及网络行为对CPU的消耗。netpmon命令在指定的监控周期报告网络行为。

netpmon 启动后直至发布trcstop命令终止它之前，一直在后台运行。如果使用缺省设置，trace命令将会在netpmon命令之后立即自动启动。另外， netpmon中还可用trcon命令选择在后面的某个时间跟踪。当这种跟踪用trcstop命令终止后，netpmon命令就会输出它的报告并退出。缺省时报告会输出到标准输出，需要时也可以重定向到某个文件。

netpmon命令还可以在一次先前产生的跟踪中以脱机模式使用。在这样的情况下，需要用gennames命令产生一个文件。该文件必须在trace终止后立即产生。

所产生的报告中包括CPU使用情况、网络设备驱动器I/O情况、互联网络套接字调用，以及网络文件系统（NFS）I/O信息：

• CPU use：netpmon命令报告线程和中断处理器对CPU的使用情况。该命令将网络相关行为的CPU使用情况与其它行为的CPU使用情况区分开。

• Network Device Driver I/O：netpmon命令监控网络适配器上所通过的I/O统计。

• Internet Socket Calls：netpmon命令在互联网络套接字上监控read，recv，recvfrom，write，send以及sendto子程序。ICMP，TCP，UDP这几个协议的每个进程都会予以报告。

• NFS I/O：netpmon命令监控客户NFS文件上的read和write子程序，NFS客户上的RPC请求以及NFS服务器的read和write请求。

3、PDT（性能诊断工具）

PDT通过收集和集中各种性能、配置和可用数据自动找出性能问题。PDT评估系统的当前状态并跟踪系统在工作量和性能上的变化。PDT数据收集和报告很容易起用，不需要更多的管理行为。

虽然许多常见的系统性能问题都有特定性，但PDT还试图用一些被认为性能好的系统中的通用概念来帮助它查找问题。这些概念包括：

• 资源的平衡使用

• 在限定范围操作

• 确定的工作量趋势

• 无错误操作

• 系统参数得到适当设置。

4、ps

ps 命令是UNIX系统中最常见的命令，它主要显示系统中关于进程的统计和状态信息，如进程ID，I/O行为以及CPU利用率等。利用ps命令提供的信息，可决定一个进程运行了多长时间，进程使用了多少CPU时间，以及进程是否受系统的惩罚。还可用ps命令确定进程使用了多少内存，完成多少I/O，进程的优先级以及是谁创建了进程。

下面这几个命令组合对于管理RS/6000 AIX系统有帮助：

（1）显示10个消耗CPU最多的进程：

# ps aux |head -1 ;ps aux |sort -rn +2 |head –10

（2）显示10个消耗存储空间最多的进程：

# ps aux |head -1 ;ps aux |sort -rn +3 |head -10

（3）按顺序显示系统中受罚的进程：

#ps -eakl |head -1 ;ps -eakl |sort -rn +5

（4）按优先级顺序显示系统中的进程：

#ps -eakl |sort -n +6 |head

（5）按处理时间为顺序显示系统中的前十个进程：

#ps vx |head -1 ;ps vx |grep -v PID |sort -rn +3 |head –10

（6）按实际内存使用的多少顺序显示系统中的前十个进程：

#ps vx |head -1 ;ps vx |grep -v PID |sort -rn +6 |head –10

（7）按换入页面的多少顺序显示系统中的前10个进程：

#ps vx |head -1 ;ps vx |grep -v PID |sort -rn +4 |head -10

5、vmstat

vmstat命令报告关于核心线程，虚拟内存，自陷（trap），磁盘以及CPU行为的统计。而且每种行为报告都被更细致地用百分比分别表示用户态、核态、空闲以及等待磁盘I/O等情况。

内核维持了对核心线程，换页以及中断行为的统计数据，而vmstat命令则通过使用knlist子程序和/dev/kmen伪设备驱动器访问这些数据。磁盘的输入/输出统计是通过设备驱动器维持的。对于磁盘，平均传输速度是通过使用活跃时间核传输信息数目决定的。而活跃时间百分比则是从报告期间驱动器忙的时间量计算出来的。

vmstat命令产生五种类型的报告：

• 虚存行为报告

• fork子进程情况报告

• 每个设备产生的中断情况报告

• 汇总报告

• 输入/输出行为报告

6、sar

sar命令报告CPU的使用情况，I/O以及其它系统行为。sar命令可以收集，报告以及保存系统行为信息。如果没有指定输入文件，则sar调用sarc命令访问系统数据。

用户可用让cron命令运行两个shell脚本（/usr/lib/sa/sa1和/usr/lib/sa2）以提供日统计和报表。在crontab文件 /var/spool/cron/crontabs/adm中包括了一些样本节，用于示范cron要在何时运行这些shell脚本。以这种方式收集到的数据对于确定系统的时间周期特征和决定峰值使用时间是有用的。

但要注意的是，sar命令自己运行时会产生相当数量的读写。因此最好在没有工作量的情况下运行sar统计，看看sar对总的统计数字有多大的影响。

7、topas

topas 命令用于监控各种系统资源，如CPU的使用情况，CPU事件和队列，内存和换页空间的使用，磁盘性能，网络性能以及NFS统计等。它还会报告指派给不同 WLM类的进程对系统资源的消耗情况。它还能报告系统中最热门的进程和工作量管理器（WLM）的热门类。有关WLM类信息只有在WLM激活时才会显示。 topas命令将热门进程定义为那些使用大量CPU时间的进程。topas命令没有作日志的选项，所有信息都是实时的。

topas命令利用 System Performance Measurement Interface（SPMI）API获得有关信息。正是因为通过SPMI API，使系统开销保持在最小程度。topas命令使用perfstat库调用访问perfstat内核扩展。

8、truss

truss命令跟踪一个进程的系统调用、所接收的信号以及招致的机器错。要检查的应用程序可在truss命令的命令行中指定，也可将truss命令挂在一个或多个已经在运行的进程上。

AIX调制工具

1、fdpr

fdpr命令改进用户级程序和库的执行时间和对实际内存的使用。fdr命令可以通过不同的操作，如删除不必要的指令和重组代码和数据，而实现这样的目标。fdr命令安装在目录/usr/bin下。

fdpr命令在三个不同阶段上，对原有的执行代码应用先进的优化技术从而为其构筑一个优化的可执行代码。这三个阶段分别是：

• 在阶段1，fdpr创建一个增加了某些装置（instrumented）的可执行程序。原有的可执行程序被保存为\_\_ProgramFile.save，而新版本被命名为\_\_ProgramFile.instr。

• 在阶段2，fdpr运行该增加了某些装置的可执行程序，并收集摘要（profiling）数据。该摘要数据被保存在一个叫\_\_ProgramFile.prof的文件中。运行执行程序时需要为它提供典型的输入数据，以使fdpr命令能够找出代码中可优化的部分。

• 在阶段3，fdpr命令使用阶段2中收集到的重要信息对可执行代码重新排序。这些重新排序涉及到这样一些任务：

（1）将那些高频度执行代码序列包装在一起。

（2）对条件分之重新排序，以改进硬件对分之条件的预测。

（3）将较少使用的代码部分移出来。

（4）内嵌一些热门函数。

（5）从重排序后的代码中删除掉NOP（空操作）指令。

另外，编译器中还提供了一个-qfdpr标志，用它可使编译器在执行代码中增加一些额外的信息，以辅助fdpr对该执行代码重新排序。但是，如果使用这个-qfdpr标志，则fdpr也只对那些用-qfdpr标志编译的模块重新排序。

2、schedtune

schedtune命令可以给抖动、进程挂起、时间片以及线程在锁上所能轮询的时间长度等设置准则。

用schedtune，可调整AIX中所设立的一组影响其内存负载控制机制的参数。Schedtune命令用于显示和修改那些用于检测系统内存是否在过度使用以致造成抖动的参数。Schedtune命令还能用于修改运行在系统上的进程的惩罚和衰减因子。在root用户下，用schedtune命令可做下面的事情：

• 决定用于确定抖动的准则。

• 决定哪个准则用于挂起进程。

• 决定在抖动终止后要等待多长时间才重新激活那些先前被挂起的进程。

• 决定被挂起的进程的最小数目。

• 调制调度优先级公式。

• 更改时间片数值。

• 决定在一把锁上轮询多长时间。

• 将schedtune值复位到它的缺省值。

需要注意的是，所有用schedtune作的修改在系统重启后都将丢失。为了确保所需的schedtune值在引导时能够置上，可在 /etc/inittab文件中插入适当的schedtune命令。如：schedt:2:once: /usr/samples/kernel/schedtune -s 65536

3、vmtune

vmtune命令负责显示和调整虚存管理器（VMM）和其它AIX部件使用的参数。系统中的根用户可动态修改包括下面这些参数：

• VMM页替换

• 永久文件读写

• 文件系统缓冲区结构（bufstructs）

• LVM缓冲区

• 裸输入/输出

• 换页空间参数

• 页删除

• 内存固定参数

第 4 楼：AIX常用命令

查看交换区信息：

lsps -a 显示交换区的分布信息

lsps -s 显示交换区的使用信息

slibclean 清除处理程序遗留的旧分页信息

smit mkps 建立交换区空间信息

swapon -a 启动所有的分页空间

/etc/swapspaces 存放分页空间表格信息

-------------------------------------------------

显示卷信息：

lsvg 显示卷的名称

lsvg -l rootvg 显示rootvg卷的详细信息

-------------------------------------------------

mount卷的方法：

varyonvg datavg 加载datavg卷

mount /dev/data1 加载datavg下的一个data1卷

裸设备类型：raw,jfs jfs可以转变成文件系统，而raw则不行

-------------------------------------------------

在裸设备上安装oracle系统：

修改裸设备的权限，如裸设备名为system01，安装数据库用户为oracle

chown oracle:dba /dev/system01

chown oracle:dba /dev/rsystem01

在使用文件时必须用rsystem01

-------------------------------------------------

smit快速路径名称：(smit：图形方式，smitty：字符方式)

dev 设备管理

diag 诊断

jfs 定期档案管理系统

lvm 逻辑卷册系统管理员管理

nfs NFS管理

sinstallp 软件安装及维护

spooler 列印队列管理

system 系统管理

tcpip TCP/IP管理

USER 使用者管理

clstart，clstop：启动和停止cluster

lssrc -g cluser：查看cluser的状态

-------------------------------------------------

查看已安装的软件信息：

ls -aF /usr/lpp (lpp:Licensed Program Products)

查看安装媒体内容：

installp -q -d /dev/cdrom -l

-------------------------------------------------

启动时自动加载文件系统信息：

需要加载的信息存放在/etc/filesystems

mount -t nf 加载所有在/ect/filesystems中定义type=nfs的文件系统

显示已加载的文件系统及状态： df -v,mount

-------------------------------------------------

查看错误日志信息：

errpt -a

-------------------------------------------------

有关TCP/IP的命令

网路卡：

smit chgenet,chgtok,chgfddi,opschange,mktty:adptr架构快速路径

smit mkinet,ppp:slip与ppp快速路径

ifconfig:config界面

位址：

/etc/hosts 静态主机表

/etc/resolv.conf 位址解析的名称服务器

/etc/named.boot 名称服务器架构

/etc/named.ca 根名称服务器快取

/etc/named.data 位址列表

/etc/named.rev 反转指标列表

nslookup 查询名称服务器资讯

网络路由：

route 管理路由

netstat -rn 列出定义的路由

routed 路由(daekmin rip)

gated 路由(daekmin rip、egp、hello)

/etc/gateways 已知网关

/etc/networks 已知网路

服务：

/etc/services

/etc/inetd.conf

TCP/IP群组子系统:

/etc/rc.net

startsrc -g tcpip 启动全部的tcpip子系统

startsrc -s inetd 启动主要internet

除错：

iptrace 启动封包追踪

ipreport 追踪结果格式化输出

netstat 网络统计

ping 检查是否可以到达

查看HACMP，外部硬盘信息:

lscfg -v

lsdev -Cc adapter

对等机器信息：

/etc/.rhosts

/etc/hosts.equiv

/etc/hosts

-------------------------------------------------

观察进程内存使用情况：

ps aux 观察参数%mem：内存使用百分比 RSS:实际使用内存

vmstat free的单位为块，缺省值为4096bytst

-------------------------------------------------

创建raw设备时选择的类型:

raw\_lv

第 5 楼：常用AIX论坛地址、资料下载地址、FTP地址

AIX论坛：

http://www.loveunix.net/ ; 《爱U家园》是大家的快乐空间

http://www.passcisco.com/bbs/list.asp?boardid=29 ; AIX使用者论坛

http://www.chinaunix.net/cgi-bin/bbs/forums.cgi?forum=5 ; chinaunix的AIX论坛

http://www.aixchina.com/ ; AIX中国论坛

http://www.swg.com.cn/bbs/cgi-bin/leoboard.cgi ; 思达奇公司的AIX 技术区

http://www.itpub.net/forumdisplay.php?s=&forumid=29 ; itpub的UNIX论坛

http://www.ibmusers.com/ ; 银信公司网站

http://bbs.interconn.com.cn/cgi-bin/forums.cgi?forum=2

AIX资料下载：

http://www.redbooks.ibm.com/ ;

http://www-1.ibm.com/servers/ese ... rary/hardware\_docs/

http://www-900.ibm.com/cn/support/download/r6download.shtml

http://publib.boulder.ibm.com/pseries/hacmp/hacmp\_index.html

http://www-1.ibm.com/servers/ese ... ary/hacmp\_docs.html

http://www.storage.ibm.com/hardsoft/products/ssa/docs/index.html

http://www.alphaworks.ibm.com/

http://www.ibm.com/certify

AIX练习:

ip:210.5.4.6 port:4003

user:openuser passwd:openuser

IBM AIX 和 pSeries信息中心，内容多多：

http://publib16.boulder.ibm.com/ ... nter/base/index.htm

首先感谢这些默默无闻为大家服务的同志！

ftp://unix2003:unix2003@166.111.121.3/ 推荐

ftp://chinaunix:unixchina@210.34.8.181/ 推荐

ftp://chinaunix:chinaunix@61.232.65.242/

ftp://firer2000:firer2000@211.141.67.2/

ftp://may:may@cnono.vicp.net/

第 6 楼：AIX操作系统使用心得

AIX是IBM公司开发的业界领先的优秀商务Unix操作系统，在可靠性、可用性、开放性、扩展性、高性能、安全性等方面都非常突出，尤其是在Internet的关键应用领域以及系统和硬件管理能力方面，其性能表现更为出色，受到了业界的普遍认可和广泛使用。以下是笔者几年来使用AIX 系统的一点心得，兹供使用该系统的其他读者参考。

1．如何禁止终端上的中断键（CTRL+C）？

在很多应用系统中，系统管理员希望普通用户只运行自己的应用程序，不能进入系统的shell提示符下，但缺省情况下当用户在终端上按CTRL+C键时就会退到系统提示符下。由于用户终端一般没有固定的端口号，为了禁止使用中断键，可采取下面办法：

（1）如果使用ksh, 可在$HOME/.profile中第一行加入如下内容：

trap "echo 'Abnormal operation'; exit" 123915

(2)如果使用csh(ksh亦可)，可用如下命令：

%stty intr ^!

如果恢复正常情况，键入下列命令：

% stty intr ^c

2．如何在shell中不回显（echo）字符？

在实际应用中，一般当我们在键盘上键入口令时不希望将其显示在屏幕上，为此可采用下面的两种办法：

•使用stty 命令

stty -echo # do not display password

echo "Enter password: \c"

read PASSWD #get the password

stty echo # restore standard configuration

•使用echo命令

设置保密属性：echo "\033[8m"

取消保密属性：echo "\033[m"

3．如何在某个目录及其所属子目录的所有文件中查找字符串？

在程序维护过程中，有时需要在某个目录及其所属子目录的所有文件中查找某一个字符串，为此可用下面两种方法（假设在\*.cp文件中查找字符串"abc"，结果放在文件out中）：

（1）cat /dev/null > out

find ./ -name "\*.cp" -exec grep "abc"{} >> out

（2）find ./ -name "\*.cp" | xargs grep "abc" > out

推荐使用第二种方法，因其系统开销小、速度快。

4．如何对/etc/inittab文件中的一行进行注释？

我们都知道在shell中使用"#"作为注释符号，但在/etc/inittab中注释一行的方法是在第一个字符前插入字符"："。

5．如何转换DOS和AIX两种格式的文本文件？

如欲转换DOS和AIX两种格式的文本文件，有两种方法：

（1）用ftp命令：设置ASCII传输类型，在一台运行AIX的机器和另外一台运行Windows的机器之间互相传送，这里不再赘述。

（2）使用AIX2dos或dos2AIX命令

如将DOS格式的文本文件转换为AIX格式，可用命令A：dos2AIX inputfile outfile，反之可用命令：AIX2dos inputfile outfile，关于dos2AIX和AIX2dos命令的详细用法可参阅"dos2AIX -h "和"AIX2dos -h "。注意要使用这两个命令，必须首先安装文件集bos.pci。

6．如何解决某一PV上的VGDA与ODM库不一致的问题？

在系统维护过程中，因为操作错误或其他特殊原因，有可能使某一PV上的LVCB和VGDA与其对应的ODM库不一致，导致ODM库紊乱，对PV的有关操作无法进行，这时可采用如下两个AIX命令加以解决：

redefinevg -d hdisk\_name vg\_name

该命令以指定PV上的LVM信息重新定义给定VG的ODM库。

或：synclvodm -P -v vgname

该命令同步或重建给定VG的ODM库和LVM信息。

7．如何设置用户的文件大小限制？

在AIX系统中，用户使用系统资源是有一定限制的。如用户缺省可创建或扩展的最大文件为1G（参见/etc/security/limits: fsize = 2097151, fsize\_hard=fsize 512-bytes blocks）。

如欲修改，可使用smit:

# smit chuser 选择用户，修改下面两项：

Soft FILE size [4194302]

# (2G,可根据需要设定)

Hard FILE size [4194302]

# (2G, 可根据需要设定)

用该用户身份登录，使用"ulimit -f "和"ulimit -Hf"可分别显示其fsize、fsize\_hard的大小。

8、如何按文件大小排序列出一个文件系统下的文件？

当监控某一文件系统的空间使用情况时，如果该文件系统剩余空间较少或已使用空间增长较快，则有必要排序列出该文件系统中所有大于某一给定字节数的文件，以便进一步维护管理。为此，可用如下命令：

# find [filesystem\_name] -xdev -size +[512-bytes bloks] -ls | sort -r -n -k7

以上所有命令在AIX4.3.3下全部测试通过。

第 7 楼：AIX环境如何找到活动最频繁的进程

1。下面命令输出中的'C'字段是一个进程的cpu-penalty值（0到120之间）,值越大表明进程越活跃。

# ps alxw | sort +5 -n

2．显示前10个累计占用CPU时间的进程

#ps -e|head -n 1;ps -e|grep -v"TIME|0:"|sort +2b -3 -n -r|head -n 10

3。显示前10个当前占用CPU时间的进程

#ps -ef|head -n 1;ps -ef|grep -v"C|0:00"|sort +3b -4 -n -r|head -n 10

4。显示前10个占用CPU应用的进程

#ps gu|head -n 1;ps gu|grep -v"CPU|kproc"|sort +2b -3 -n -r|head -n 10

第 8 楼：AIX中查看用户对系统资源的使用情况

在AIX中可以通过打开acct功能来查看用户的资源使用情况，主要包括：用户连接情况、用户CPU使用情况、

磁盘使用情况和打印机使用情况等。具体的步骤（共7步）有：

1. 在/var/adm/acct目录下创建三个子目录：nite,sum,fiscal

2. 以root登录，到/var/adm目录下，执行/usr/lib/acct/nulladm wtmp pacct。

3. 通过在/etc/rc文件中添加两行（最后一行"exit 0"之上）：

echo "Starting Process Accounting"

/usr/bin/su - adm -c /usr/lib/acct/startup

打开记帐功能。

4. 将/etc/filesystems中要加入磁盘记帐的文件系统相关节中的account值改成true

5. 修改/etc/qconfig文件，要加入打印机使用记帐的打印队列的描述节中添加一行，如：

lp0:

device=dlp0

acctfile = /var/adm/qacct # 添加行

6. 修改cron，例/var/spool/cron/crontabs/adm文件：

10 23 \* \* 0-6 /usr/lib/acct/runacct 2> /usr/adm/acct/nite/accterr > /dev/null

0 23 \* \* 0-6 /usr/lib/acct/dodisk > /dev/null 2>&1

0 \* \* \* \* /usr/lib/acct/ckpacct > /dev/null 2>&1

15 4 1 \* \* /usr/lib/acct/monacct > /dev/null 2>&1

7. 如果要使用adm用户进行记帐管理，首先要给它分配密码，然后在PATH环境变量中包括/usr/lib/acct以方便命令执行。

完成这些步骤后，系统会在步骤一生成的目录下定期生成报告文件，记录各用户的资源使用情况。

第 9 楼：AIX的系统备份和恢复

备份和恢复是系统管理员经常要做的事情, 主要包括rootvg备份和用户数据备份.

  1. 操作系统和系统程序的备份:

  将一盘新磁带或无用磁带插入磁带机

  #tctl -f /dev/rmt0 rewind

  #smit mksysb

  在"备份设备或文件"中添入"/dev/rmt0"后回车.

  系统会运行很长时间, 等到屏幕显示OK后拿出磁带. 这时候, 系统备份完成. 注意:

mksysb仅备份rootvg中已安装的文件系统.

  2. 用户数据备份

  • 常用磁带机选项

  /dev/rmt0: 若选择/dev/rmt0, 在插入磁带和写完一次磁带时, 磁带机都将磁带反绕到头.

因此, 下一次备份文件将覆盖本次备份.

  /dev/rmt0.1: 若选择/dev/rmt0.1, 则插入磁带和写完一次磁带时, 磁带机均不反绕磁带.

因此, 一盘磁带可以连续备份几个文件或文件系统.

  • #smit fs

  选择"备份文件系统"

  添入要备份的"文件系统名称"

  添入"/dev/rmt0.1"

  重复上述操作即可在同一盘磁带上备份多个文件系统.

  3. rootvg的恢复

  • 启动机器进入维护模式

  参见安装手册, 当出现"Welcome to Base Operating System Installation and Maintanence"时,

选3 "Start Maintenance Mode for System Recovery"

  • 恢复系统

  继续选4 "Install from a System Backup"

  出现"Choose mksysb Device"画面, 选"/dev/rmt0"并插入磁带后回车.这时候, 系统自动恢复操作系统.

  4. 用户数据恢复

  #tctl -f /dev/rmt0 rewind

  #smit fs

  选择"恢复文件系统"

  添入"设备名称"和"目标目录"

  系统会自动找到相应目录恢复.

第 10 楼：AIX如何自动启动和关闭软件的运行?

本文档讨论了如何定制系统, 以使用户定义的应用在系统引导过程中自动启动, 并在系统关闭时自动停止.

1. 自动启动定制的软件

在系统引导时, init进程顺序检测/etc/inittab文件, 决定在引导过程中进行什么样的操作.如果应用, 进程或其它的用户操作需要在系统引导过程中进行, 则要在/etc/inittab文件中加入相应的条目.通常的做法是创建一个名为rc.local的脚本文件, 将所有要在系统引导过程中启动的用户应用写入该脚本文件中. 该脚本文件可以在/etc/inittab文件中被调用, 通常是在所有系统应用和服务启动后, 在用户登录被允许之前被调用. 位置通常是在主控登录的条目前.

下面是一个例子:

rc.local:2:wait:/etc/rc.local > /dev/console 2>&1

在上面的条目中, init将在执行余下的条目之前等待该脚本程序的完成.如果进程, 应用或命令需要通过特定的用户运行, 则可在运行的命令中加入su命令.

如: su myuser "-c /usr/bin/myapp"

建议在脚本程序中加入应用开始启动和完成启动的输出提示.

如:

#!/bin/ksh

echo "Starting User Applications...."

echo "Starting Application XYZ"

su myuser "-c /usr/bin/myapp"

echo "Startup of User Applications completed"

2. 自动停止定制的软件

shutdown 命令用于关闭AIX系统. 在此操作中会关闭所有AIX子系统及文件系统.如果名为/etc/rc.shutdown的脚本程序存在, 系统会首先执行该脚本程序中的命令. 在该脚本程序中可以加入关闭用户应用的命令. 建议在脚本程序中加入应用停止运行和完成停止运行的输出提示.

如:

#!/bin/ksh

echo "Stopping User Applications...."

echo "Stopping Application XYZ"

[ commands to stop application ]

echo "Shutdown of User Applications completed"

第 11 楼：如何查看操作系统中是否安装了某个补丁程序?

使用命令:

lslpp -Bl Uxxxxxx

Uxxxxxx 是补丁程序的ID.

例: lslpp -Bl U465241 命令可用于查看系统中是否安装了ID为U4652241的补丁程序.

第 12 楼：如何检查AIX文件系统的病毒？

#virscan directory

例如: virscan /usr

第 13 楼：AIX如何解决"device busy"问题

当我们对网络或网卡进行操作时,经常会遇到"device busy", 而不让我们进行操作.

例如:

rmdev -l ent1

Method error (/etc/methods/ucfgdevice):

0514-062 Cannot perform the requested function because the

specified device is busy.

1.首先我们要确认网络接口已经"down"及 "detached".

使用以下脚本文件可以"down"及"detach"所有TCP/IP网络接口.

interfaces=`lsdev -Cc if -F name | grep -v lo0`

for interface in $interfaces

do

ifconfig $interface down

ifconfig $interface detach

done

2. 检查是否下列应用正在运行:

1) SNA : lssrc -g sna (检查SNA是否运行)

stopsrc -g sna(中断)

stopsrc -f -s sna(强行中断)

如果以上两条命令都不起作用: /usr/bin/sna -stop sna -t forced

如果仍不能中断: /usr/bin/sna -stop sna -t cancel

2) IPX: ps -ef |grep npsd

ps -ef |grep sapd (IPX是否运行)

/usr/lpp/netware/bin/stopnps (中断)

3) Netbios: ps -ef | grep netbios (是否运行)

mcs0 unload (中断)

以上步骤对于大部分"device busy"问题都可解决.

第 14 楼：何为AIX 5L?

AIX 5L 是AIX的下一代产品(当前为AIX V4)，它支持IBM POWER 和Intel 64位(IA-64)平台.“L” 指Linux affinity.

第 15 楼：使用命令行安装删除文件包

在AIX 中, 一般使用"smit"命令来安装和删除文件包,但有时经常会遇到"smit"本身被破坏的情况, 我们可以使用"installp"命令来安装及删除文件包.

安装: installp -ac -Q -d /dev/cd0

删除: installp -u

第 16 楼：何为AIX/Montery

AIX/Monterey是IBM下一发展阶段的UNIX操作系统，其中AIX核心为Intel系统带来了极易扩展、高度可用、具有业界非凡实力的UNIX操作系统所具有的全部优势。AIX/Monterey是一项主要的UNIX操作系统方案的成果，该项方案由IBM领导，并与SCO和 Intel合作，获得了业界领先的软件和系统厂商的广泛支持。AIX/Monterey是一个大规模的企业级UNIX操作系统，目前运行于各种Intel 32位体系架构（IA-32）及POWER体系架构之上。该产品将于今年晚些时候扩展到IA-64这一Intel的64位体系架构。 AIX/Monterey所支持的系统十分广泛，包括从部门级服务器到大型数据中心服务器。AIX/Monterey融合了多种操作系统的最佳特性，包括 AIX系统、IBM的DYNIX/ptx，以及SCO的UnixWare，其中DYNIX/ptx来自Sequent，运行NUMA-Q，而 UnixWare则是当今运行于Intel平台上的最大份额的一种UNIX操作系统。今年晚些时候，对于那些偏爱基于Intel64位Itanium3系统的客户而言，IBM将为其提供AIX/Monterey/64，使其拥有与运行于RS/6000上的AIX同样强大的功能。AIX/Monterey则将继续满足客户对跨越各种基于Intel和POWER的系统、行业性强的企业级UNIX平台的需求。我们将在AIX/Monterey中构建强大的 Linux兼容性，以帮助使Linux应用在AIX/Monterey上得以轻松实现，并帮助推动AIX/Monterey应用运行于未来版本的 Linux之中。应用的可移植性有望比Linux的成熟提前实现，这将为客户的顺利过渡提供坚实的基础。另外，我们还将与开放源码团体共同合作，将 AIX/Monterey技术应用于Linux之中，以便帮助建立起一种更好的Linux系统。

IBM完全能够帮助客户适应Linux的发展，其所具备的出众能力是其它任何UNIX厂商都无法企及的。对于运行于AIX/Monterey这一功能强大的全新操作系统之上的应用而言，IBM计划使其在未来能够轻松地移植到Linux中，并帮助客户保护其在硬件、应用、数据、过程和技术等方面的投资，因此，现今投资AIX/Monterey的客户可以充满自信地应对Linux的不断发展。

第 17 楼：如何查找系统中的某个文件存在于哪个文件集中?

当操作系统上由于未安装特定的文件集而无法执行某个命令时, 可以参考下面的方法确定所需的文件集.

首先确认系统中已安装了文件集 bos.content\_list:

# lslpp -l bos.content\_list

然后用下面的命令查找与指定文件相关的文件集:

# /usr/sbin/which\_fileset

如, # /usr/sbin/which\_fileset iostat

/usr/bin/iostat bos.acct

查找指定的文件集中包含哪些文件:

# lslpp -f

如, # lslpp -f bos.acct

....

/usr/bin/iostat

....

第 18 楼：如何减小 /var/adm/wtmp 文件的大小?

文件 /var/adm/wtmp 的大小会在每次用户登录时增长, 但从不自动减小. wtmp 文件的内容会被命令 last 使用, 用于显示登录系统和重启机器的情况.该文件不能被删除, 但它的内容可用如下命令清除:

# > /var/adm/wtmp

第 19 楼：AIX使用异步I/O(aio)提高系统性能

如果是同步I/O,当一个I/O操作执行时,应用程序必须等待,直到此I/O执行完.

相反,异步I/O操作在后台运行,I/O操作和应用程序可以同时运行,提高了系统性能.

使用异步I/O会提高I/O流量,如果应用是对裸设备进行操作,这种优势更加明显.

因此象数据库,文件服务器等应用往往会利用异步I/O,使得多个I/O操作同时执行.

1. 如何知道是否需要异步I/O?

\* 执行命令:"vmstat # #" 如果"wa"值超过25%.

\* 执行命令:"iostat # #" 如果"%tm\_act"值超过35%.

2. 查询有几个AIO服务器: pstat -a | grep aios | wc -l

3. 应该设几个AIO服务器?

一般为硬盘数的10倍,但不要超过80.

4. AIO驱动程序: bos.rte.aio

5. 配置AIO为可用: # smit chgaio ->

STATE to be configured at system restart [available]

第 20 楼：AIX如何管理Paging Space

Paging Space 的大小应符合应用的需要.太大会浪费硬盘空间,太小会影响系统的运行.Paging Space的确定原则: 1. 系统实际内存小于64MB, paging space= 2 \* RAM

2. 系统实际内存在 64MB to 256MB 之间, Page Space = RAM size + 16MB

3. 系统实际内存大于 256MB , Page Space = 512 + ( RAM - 256 ) \* 1.25

4. 对于AIX 4.3.3 当内存达到4GB 到8GB,因为AIX 4.3.3 使用延迟的(deferred )paging space.Paging Space主要根据应用的要求来定.

一般可初始3GB, 然后观察paging space的使用情况,如果使用率超过70%, 则需要增加paging space.

以上计算方法只是粗略地算法,用户可根据实际情况调整.

\* 如果系统出现以下几种情况,则说明需要扩大paging Space: 1. 下列任意错误信息之一:

INIT: Paging space is low

ksh: cannot fork no swap space

Not enough memory

Fork function failed

fork () system call failed

Unable to fork, too many processes

Fork failure - not enough memory available

Fork function not allowed. Not enough memory available.

Cannot fork: Not enough space

2. 使用命令" lsps -a" ,其结果%USED 列的平均值大于80%.

3. 使用命令"lsps -s" 其结果%USED 列大于80%.

\* 使用下列命令也可以检查paging space是否够用:

iostat

vmstat

lsps

第 21 楼：如何在AIX 4.3中安装联机手册？

使用联机手册之前，需安装相应的软件包。

解答 AIX 4.3 安装介质包含两张联机文档光盘（Documentation CD): Base Documentation CD 和 Extended Documentation CD。使用命令man查看系统命令之前

需要按照下列步骤安装相应的软件包： 1. 正确设置环境变量LANG的值，如“en\_US”为英文环境，“zh\_CN”为简体中文环境。可以用命令

export LANG=en\_US

设置。

2. 使用man查看系统命令之前必须安装下列软件包：

bos.html.en\_US.topnav.navigate - Top Level Navigation

bos.html.en\_US.nav - Online Navigation

bos.html.en\_US.cmds.cmds1 - AIX Commands Reference 1

bos.html.en\_US.cmds.cmds2 - AIX Commands Reference 2

bos.html.en\_US.cmds.cmds3 - AIX Commands Reference 3

bos.html.en\_US.cmds.cmds4 - AIX Commands Reference 4

bos.html.en\_US.cmds.cmds5 - AIX Commands Reference 5

bos.html.en\_US.cmds.cmds6 - AIX Commands Reference 6

使用命令

lslpp -l

检查上述软件包是否已经安装。

3. 如果需要查询AIX 系统调用及C语言库函数，则要安装软件包 bos.html.en\_US.techref.base - AIX Base Tech Ref

bos.html.en\_US.techref.commo - AIX Commo Tech Ref

4. Documentation CD中还包含许多其它的手册，可根据需要安装相应的软件包。

在AIX系统中安装软件包可使用smit管理菜单。在命令行输入

smit install\_all

选择:

1. CD-ROM 设备(通常为/dev/cd0)。

2. 将光标移至"SOFTWARE to install" 并按下或“+7”。

3. 将光标移至要安装的软件包并按下 或 “+7”选择要安装的软件包。

4. 选中所有要安装的软件包后按?lt;Enter> 开始安装，之后需再次按下确认后继续安装，直至成功。

此时就可以使用man查看系统命令了!

第 22 楼：如何在AIX中设置中文环境

环境

平台：RS/6000

软件版本： AIX 4.2 或更新版本

在AIX中使用中文有两种途径：第一是在安装AIX时选择中文语言，装好的系统自动显示中文（这种方法不推荐使用，它没有第二种方法使用起来灵活）。第二是安装AIX时选择英文,系统启动后手工设置中文环境，方法如下：

1. 将AIX系统盘的第一张光盘放入光驱；

2. 运行命令：

smitty

--> System Environments

--> Manage Languange Environment

--> Change/Show Primary Language Environment

--> Change/Show Cultural Convention, Language, or Keyboard

在随后显示的菜单中将光标分别移到以下字段：

Primary CULTURAL Convention

Primary LANGUAGE translation

Primary KEYBOARD

按下，从弹出的菜单中选择“IBM-eucCN”将上述字段改为简体中文，按下回车键后系统自动从光盘安装中文环境软件包。此操作完成后重新启动系统，操作界面即为简体中文。

需要输入中文时使用下列功能键切换输入方法：

AIX 4.3.3 以前的版本： + F1 --- + F4 切换到各种中文输入方法；

右 --- 切换到英文输入；

AIX 4.3.3： CTRL + [F2] : 智能 ABC ;

CTRL + [F4] : 拼音输入;

CTRL + [F5] : 五笔输入;

CTRL + [F6] : 郑码输入 ;

CTRL + [F7] : 表形码输入;

CTRL + [F9] : 内码输入;

CTRL + [F10] :英文半角;

此外，AIX还包含另外两种中文环境，即“UTF8”和“GBK”，它们与“IBM-eucCN”之区别在于包含了繁体汉字的使用。上述三种中文环境的设置方法均相同。

第 23 楼：AIX 的许可权限---ACL

ACL是对标准权限位的扩展。通过修改分配给个人或组的标准权限，对每个文件或目录进行更精细的控制。对每个组或用户，有3种权限分配情况：

PERMIT : 准许对文件或目录的特定权限。

DEMY : 限制对文件或目录的特定权限。

SPECIFY : 明确地定义文件或目录权限。

"acledit "命令用于建立ACL。首先必须设置文本编辑器.如：export EDITOR=/usr/bin/vi。

然后使用：acledit file\_name

屏幕上将会显示：

attributes:

base permissions

owner (rcunning): rwx

group (staff)： r—

others: ---

extended permissions

disabled

要设置扩展的权限，将“disabled ”设置改为“enabled”：

extend permissions

enabled

使用permit、deny 或specify关键字来定义扩展权限。前面的例子表明只有所有者能对这个文件写操作。组成员能读此文件而其他用户则没有任何许可权限。如果要使用户" joe"能够读写这个文件，用以下命令：

extended permissions

enabled

permit rw- u: joe.

要允许组用户joegroup读这个文件，用以下命令：

rermit r-- g: joegroup

你能够通过在同一行中合并多个条目来对权限进行微调。如果只想为pete提供读写权，而他是系统组的一部分，则用以下命令：

permit rw- u: pete, g: system

要为几个用户或组增加许可权限，则使用分行命令 ：

permit rw- u: joe

permit rw- u: pete

使用" ls -el"命令，查看ACL是否已在文件中设置。如"ls -el profile"命令显示：

— rwxw--------+

最后的+表示文件已具 有有效的ACL 。

注意：使用有数字争议的chmod 命令将使文件或目录的ACL无效。

第 24 楼：AIX管理经验谈

在AIX中可以通过打开acct功能来查看用户的资源使用情况， 主要包括：

用户连接情况、用户CPU使用情况、磁盘使用情况和打印机使用情况等。

具体的步骤（共7步）有：

1. 在/var/adm/acct目录下创建三个子目录：nite,sum,fiscal

2. 以root登录，到/var/adm目录下，执行/usr/lib/acct/nulladm wtmp pacct。

3. 通过在/etc/rc文件中添加两行（最后一行"exit 0"之上）：

echo "Starting Process Accounting" /usr/bin/su - adm -c /usr/lib/acct/startup

打开记帐功能。

4. 将/etc/filesystems中要加入磁盘记帐的文件系统相关节中的account值改成true

5. 修改/etc/qconfig文件，要加入打印机使用记帐的打印队列的描述节中添加一行，

如： lp0: device=dlp0 acctfile = /var/adm/qacct # 添加行

6. 修改cron，例/var/spool/cron/crontabs/adm文件：

10 23 \* \* 0-6 /usr/lib/acct/runacct \

2> /usr/adm/acct/nite/accterr > /dev/null

0 23 \* \* 0-6 /usr/lib/acct/dodisk > /dev/null 2>&1

0 \* \* \* \* /usr/lib/acct/ckpacct > /dev/null 2>&1

15 4 1 \* \* /usr/lib/acct/monacct > /dev/null 2>&1

7. 如果要使用adm用户进行记帐管理，首先要给它分配密码，

然后在PATH环境变量中包括/usr/lib/acct以方便命令执行。

完成这些步骤后，系统会在步骤一生成的目录下定期生成报告文件，

记录各用户的资源使用情况。

第 25 楼：AIX简介

AIX 全名为（Advanced Interactive Executive），它是IBM 公司的UNIX操作系统，

整个系统的设计从网络、主机硬件系统，到操作系统完全遵守开放系统的原则。

下面对AIX 作以介绍。

RS/6000 采用IBM 的UNIX操作系统-AIX作为其操作系统。这是一

个目前操作系统界最成功，应用领域最广，最开放的第二代的UNIX系

统。它特别适合于做关键数据处理（CRITICAL）。

AIX 包含了许多IBM 大型机传统受欢迎的特征，如系统完整性，系统可管理

性和系统可用性。

在 AIX 操作系统上，有许多的数据库和开发工具，用户除了选用已有的应用

软件外，还可以根据各自的需要进行开发。

此外，在AIX 之上，有一组功能强，使用方便的系统管理工具。对于异种平台

互存，互操作有很成熟的解决方案。

由于该 UNIX 的先进的内核技术和最好的开放性，因此，虽然RS/6000

从宣布到今天只有短短的5 年多的时间，它已在各行各业有了广泛的运用，

并在1993和1994年连续二年在MIDRANGE商用 UNIX 领域处于第一位。

RISC SYSTEM/6000的操作系统是AIX ，它是性能卓越的、开放的

UNIX，汇集了多年来计算机界在UNIX上的研究成果，以IBM 在计算机

体系结构、操作系统方面40多年极其丰富的经验。最大限度的使用RISC

技术，安装了象AIX 这样的具备工业界实力的UNIX操作系统。

它既可连接SAA 体系结构，又能与非IBM 系统的网络相连，因此，可以

和多数专业银行现有的系统实现互连，这对今后业务系统拓展将带来极大的

灵活性，并降低投资。

AIX 遵循一系列的国际标准：

\* IEEE POSIX1004.1－1990

\* X/OPEN 移植指南ISSUE3的基本级（XPG3）

\* AES/OS REVISION A （OSF/1 LEVEL 2 资格）

\* FIPS 151－1

\* AIX的编译器： XLC、C++(可选)、FORTRAN(可选)、PASCAL(可选)、COBOL(可选)

\* ADA 的编译器已达到XPG3“成员”级的认可。

\* AIX 支持多用户、多任务。

AIX有一些其它特性包括：

AIX 提供了3 种SHELL ：SYSTEM V的KORN、BOURNE SHELL和4.3BSDC

SHELL作为可选择的UNIX系统界面；

安全设施满足TCB （Trusted Computing Base）的C2级；

实时处理能力，这对于“面向交易”的应用至关重要（如零售业

和银行等），它使RS/6000 获得极高的响应和吞吐量；

虚拟存储管理，当需要时，可将一些不常用的模块转送至外存，

提高内存的可利用性。

先进的文件系统，使得系统管理更加有效，并提高了数据可靠性

以及完整性。

能兼容DOS 应用程序和数据。

InfoExplorer，快速信息超文本索引系统- 不仅包括文字，而且

对包含声音、图像的索引系统，这是个联机的文件接口。包括全部的

超文本的索引和查找，以及面向任务和坐标的多重导引和索引系统。

这个文字及图形索引系统以一个灵活的、基于任务的方式去使用详细

资料及培训资料。

高级系统管理工具（SMIT，System Management Interface Tool）。

提供一级菜单驱动程序，诸如完成软件的安装与设置、设备的设置及

管理、问题的测定、存贮管理等。可以自动地进行I/O 设备设置，

ASCII 终端也可充当系统控制台。在LAN 上可以进行远程系统的安装。

第 26 楼：AIX网络诊断及解决现场实例

AIX网络诊断及解决现场实例

http://www.chinaunix.net/ 作者:viperonline 发表于：2004-05-12 20:55:52

问题：有两台aix的测试机器，一台做应用，一台做数据库，都在同一个网段下（无双机），其中应用网络没有问题，而数据库这台网络有问题，内网下载只有30几K。

解决方法：对比法

由于这两台机器属于同一网段，机型完全一样，可以将两台机器的网络设置进行细致的对比，看看有何差异，然后改之。

使用netstat -v ent0 | grep -p "Specific Statistics"命令分别查看两台机器

应用：

netstat -v ent0 | grep -p "Specific Statistics"

10/100 Mbps Ethernet PCI Adapter II (1410ff01) Specific Statistics:

--------------------------------------------------------------------

Link Status: Up

Media Speed Selected: Auto negotiation

Media Speed Running: 100 Mbps Full Duplex

Receive Pool Buffer Size: 1024

Free Receive Pool Buffers: 1024

No Receive Pool Buffer Errors: 0

Receive Buffer Too Small Errors: 0

Entries to transmit timeout routine: 0

Transmit IPsec packets: 0

Transmit IPsec packets dropped: 0

Receive IPsec packets: 0

Receive IPsec packets dropped: 0

Inbound IPsec SA offload count: 0

Transmit Large Send packets: 0

Transmit Large Send packets dropped: 0

Packets with Transmit collisions:

1 collisions: 0 6 collisions: 0 11 collisions: 0

2 collisions: 0 7 collisions: 0 12 collisions: 0

3 collisions: 0 8 collisions: 0 13 collisions: 0

4 collisions: 0 9 collisions: 0 14 collisions: 0

5 collisions: 0 10 collisions: 0 15 collisions: 0

数据库：

netstat -v ent0 | grep -p "Specific Statistics"

10/100 Mbps Ethernet PCI Adapter II (1410ff01) Specific Statistics:

--------------------------------------------------------------------

Link Status: Up

Media Speed Selected: 100 Mbps Half Duplex

Media Speed Running: 100 Mbps Half Duplex

Receive Pool Buffer Size: 1024

Free Receive Pool Buffers: 1024

No Receive Pool Buffer Errors: 0

Receive Buffer Too Small Errors: 0

Entries to transmit timeout routine: 0

Transmit IPsec packets: 0

Transmit IPsec packets dropped: 0

Receive IPsec packets: 0

Receive IPsec packets dropped: 0

Inbound IPsec SA offload count: 0

Transmit Large Send packets: 0

Transmit Large Send packets dropped: 0

Packets with Transmit collisions:

1 collisions: 233543413 6 collisions: 0 11 collisions: 0

2 collisions: 0 7 collisions: 0 12 collisions: 0

3 collisions: 0 8 collisions: 0 13 collisions: 0

4 collisions: 0 9 collisions: 0 14 collisions: 0

5 collisions: 0 10 collisions: 0 15 collisions: 0

仔细对比发现数据库这台显然是双工模式与应用这台不同，应用的机器是自适应的，运行时是100M全双工模式，而数据库的机器限制为100M半双工模式，现在就是要将数据库这台改成与应用一样

更改双工模式的命令为smitty chgenet

但是在此之前要将需要更改的网卡down掉，以及所有经过此网卡的服务停止

步骤如下：

ifconfig -a

输出：

en0: flags=5e080863,80<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT,CHECKSUM\_OFFLOAD,PSEG,CHAIN>

inet 10.100.10.6 netmask 0xffffff00 broadcast 10.100.10.255

再输入：

ifconfig en0 down

ifconfig en0 detach

smitty chgennt

输出：

[Entry Fields]

Ethernet Adapter ent0

Description 10/100 Mbps Ethernet PCI Adapter II (1410ff01)

Status Available

Location 1L-08

Transmit descriptor queue size [512] +#

Receive descriptor queue size [512] +#

Software transmit queue size [8192] +#

Receive buffer pool size [1024] +#

Media Speed Auto\_Negotiation +

IPsec Offload no +

Enable hardware transmit and receive checksum yes +

Enable TCP Large Send Offload no +

Enable Link Polling no +

Time interval for Link Polling [500] +#

Enable ALTERNATE ETHERNET address no +

ALTERNATE ETHERNET address [0x000000000000] +

Apply change to DATABASE only no +

更改之后需要重新设置默认网关（原来的会丢失）

smitty tcpip

Minimum Configuration & Startup

输出：

HOSTNAME [TEST1-CRM-APP-CLS]

\* Internet ADDRESS (dotted decimal) [10.100.10.6]

Network MASK (dotted decimal) [255.255.255.0]

\* Network INTERFACE en0

NAMESERVER

Internet ADDRESS (dotted decimal) []

DOMAIN Name []

Default Gateway

Address (dotted decimal or symbolic name) [10.100.10.1]

Cost [0] #

Do Active Dead Gateway Detection? no +

Your CABLE Type N/A +

START TCP/IP daemons Now no +

最后将网卡启用

ifconfig en0 up

问题得到解决！！！！

我的主页：http://tlcshxd.3322.org/index.html

第 27 楼：IBM巡检流程

1、检查系统硬件情况：设备故障灯是否有亮

2、系统错误报告(errlog)

3、有否发给root用户的错误报告(mail)

4、检查hacmp.out,smit.log,boot.log

5、关键系统的文件使用率不大于80%

6、逻辑卷有否stale

7、内存交换区使用率是否超过70%

8、内存交换区的大小是否为物理内存的1.5倍

9、检查备份情况（有否系统备份、用户数据备份、磁带机是否需要清洗）

10、通信（网卡、IP、路由表、ping、/etc/hosts、DNS设置等）

11、是否有数据保护方式如RAID10/RAID5,是否有Hot spare

12、系统DUMP设置是否正确

13、检查系统参数是否正确

1)/etc/enviroment中的TZ不能有夏时制

2)如有数据库：Aio:available

3)HACMP中I/O pacing:High Water Mark/Low Water MArk:33/24

4)HACMP中Syncd:10

5)HACMP中Power Monitorff

14、检查rootvg是否有镜象

15、检查errdemon,srcmstr是否正常运行

16、机房环境（电压、湿度）

17、系统性能：有否性能瓶颈(topas,vmstat)

18、补丁程序（PTF）、微码（是否需要升级）

19、HACMP测试：Cluster Verification

20、系统硬件诊断

21、运行#snap -ac,生成文件snap+s/n.pax.Z

22、机器清洁

注：1~15为A类维护（季度维护），16~18为B类维护（半年维护），19~22为C类维护（年度维护）

我的主页：http://tlcshxd.3322.org/index.html

第 28 楼：AIX Useful Commands

compress -c file.txt > file.Z Create a compressed file.

uuencode (infile) (extract-file-name) > (output file)

Converts a binary file to an ASCII file for transfer by modem or email

uudecode (encoded file)

Extracts a binary file from encoded file and calls it the extract-file-name

examples :-

uuencode maymap maymap > maymap.enc

uudecode maymap.enc

od -c /tmp Displays contents of the /tmp directory file

ls -i Lists files with their inode numbers

echo \* Lists files, can be used if ls is corrupt/missing

chtz (timezone eg GMT0BST) Changes the timezone in /etc/environment file

chlang (language eg En\_GB) Changes the language in /etc/environment file

ar -v -t (archive file) List contents of an archive

ar -v -x (archive file) Extracts the archive

ar -v -t /usr/lib/libC-r.a Lists contents of the libC\_r.a library

find /source -print | cpio -pdm /target

Copying directories using cpio, creates /target/source directory.

dump -nTv (binary executable) Displays the contents of an executable file

dump -c Displays string information

dump -o Displays object file headers

dump -l Displays line numbers

dump -s Displays the text section

snap -ao /dev/rmt0 Create a snapshot onto tape

snap -ad (directory) Create a snapshot into a named directory other

&nbs, p; than the default (/tmp/ibmsupt)

/usr/dt/bin/dtconfig -d Disables desktop logins

/usr/dt/bin/dtconfig -e Enables desktop logins

/var/dt/Xpid PID of the dtlogin process

--------------------------------------------------------------------------------

LICENSES / SOFTWARE INSTALLATION

--------------------------------------------------------------------------------

lslicense Displays number of current user licenses

chlicense -u (number) Changes the number of user licenses

( ftp,rexec and rsh (without -i flag) do not need an AIX user license to be

able to access the system.

oslevel Returns operating system level

4 . 3 . 3 . 0 <--------- Preventive Maintenance Level

| | |

| | +----------------Modification

| +--------------------Release

+------------------------Version

oslevel -l 4.3.3.0 Displays all filesets that are "downlevel"

whence (program) Returns full path of program

whereis (program) Returms full path of program

what (program) Displays identifying info from the executable

like version number, when compiled.

lslpp -L all list all installed software

lslpp -L (program set name) Check if software installed

lslpp -f Lists filesets vs packages

lslpp -ha Lists installation history of filesets

lslpp -w /usr/bin/swapon Lists the fileset that the file belongs to

lppchk -c Checks file checksums against SWVPD

lppchk -l Checks symbolic links against SWVPD

instfix -ik (fix number eg IX66617) Checks id fix is installed

instfix -ik 4330-02\_AIX\_ML

instfix -i | grep ML Displays all ML's installed

instfix -k IX38794 -d /dev/cd0 Installs a fix from cdrom

/usr/sbin/install\_assist Smitty Installation Assistant

/usr/sys/inst.images/sys.bundles Software bundle files

alt\_disk\_install -c hdisk1 Clones a running rootvg onto hdisk1

alt\_disk\_install -w Wakes up alt vg

alt\_disk\_install -s Sends alt vg to sleep !

alt\_disk\_install -x Removes alt vg from disk

/usr/lpp/bosinst/blvset -d /dev/hdisk0 -p 4.2

Resets the pad string in the BLV to the correct AIX version. Needed if the

migration option is missing when installing.

installp -ad (device) (fileset) (level) Install apply and commit fileset

installp -pad (device) (fileset) (level) Preview install

installp -u (fileset) Remove fileset

installp -ld (device) List all software on device

example:-

installp -pad /dev/rmt0 X11.base 4.3.3.0

installp -C Cleans up after a premature cancel or interrupted installation.

--------------------------------------------------------------------------------

TERMINALS / DISPLAYS

--------------------------------------------------------------------------------

/usr/share/lib/terminfo Directory with all support terminal info files.

tty Displays what the tty/pty number of the terminal is.

termdef reports the termtype setup in smit for the tty port

that termdef is run on.

chdev -l (device eg tty1) -a term=vt100 Sets tty to a vt100 terminal type

penable tty0 adds getty line into /etc/inittab for tty0 and starts getty

pdisable tty0 disables the getty line and disables getty

penable / pdisable -a option is for all

stty erase ^? Set backspace key for vt100 terminals

stty erase ^H Set backspace key for wyse50 terminals

lscons Displays the console device

chcons -a login=enable (device eg /dev/tty1) Changes the console device

lsdisp Display adapter device information

chdisp Change default display used by LFT subsystem

Create ttys on ports 0 to 7 on adapter sa2 :-

for i in 0 1 2 3 4 5 6 7

do

mkdev -c tty1 -t tty -s rs232 -p sa2 -w$i -a login=enable -a term=vt100

done

portmir -t /dev/tty0 Mirror current terminal onto /dev/tty0

portmir -o Turns off port mirroring

--------------------------------------------------------------------------------

NETWORK

--------------------------------------------------------------------------------

host (ip or hostname) Resolves a hostname / ip address

hostname Displays hostname

hostname (hostname) Sets the hostname until next reboot

lsdev -Cc if Lists all available/defined network interfaces

chdev -l (device name) -a hostname=(hostname) Changes hostname permanently

chdev -l inet0 -a hostname=thomas

ifconfig (device name) Displays network card settings

ifconfig (device name) up Turns on network card

chdev -l (device name) -a state=up Turns on network card

ifconfig (device name) down Turns off network card

ifconfig (device name) detach Removes the network card from the

network interface list

ifconfig en0 inet 194.35.52.1 netmask 255.255.255.0 up

ifconfig lo0 alias 195.60.60.1 Create alias ip address for loopback

route (add/delete) (-net/-host) (destination) (gateway)

Adds or deletes routes to other networks or hosts, does not update

the ODM database and will be lost at reboot.

route add -net 194.60.89.0 194.60.90.4

chdev -l inet0 -a "net,-hopcount,1,-netmask,255.255.255.0,207.156.168.0,10.0.15.254"

(destination) ( gateway

Adds route and adds entry into ODM, route survives a reboot,

route -rn Display route table

odmget -q "attribute=route" CuAt Displays routes in the ODM.

lsattr -EHl inet0 Displays routes set in ODM and hostname

odmget -q "name=inet0" CuAt Displays routes set in ODM and hostname

refresh -s inetd Refresh inetd after changes to inetd.conf

kill -1 (inetd PID) Refresh inetd after changes to inted.conf

netstat -i Displays interface statistics

entstat -d (ethernet adapter eg en0) Displays ethernet statistics

arp -a Displays ip to mac address table from arp cache

no -a Displays network options use -o to set individual options or

-d to set individual options to default.

no -o option=value (this value is reset at reboot)

no -o "ipforwarding=1"

traceroute (name or ipaddress) Displays all the hops from source to

destination supplied.

ping -R (hostname or ipaddress) Same as traceroute except repeats.

spray (hostname or ipaddress) Send a stream of packets to a host

stopsrc -g tcpip Stops all running TCP/IP daemons

/etc/tcp.clean Stops all running TCP/IP daemons and removes all

lock files

/etc/rc.tcpip Start all TCP/IP daemons.

Do not use startsrc -g tcpip as this will start all

TCP/IP daemons including routed & gated

--------------------------------------------------------------------------------

N.F.S.

--------------------------------------------------------------------------------

exportfs Lists all exported filesystems

exportfs -a Exports all fs's in /etc/exports file

exportfs -u (filesystem) Un-exports a filesystem

mknfs Configures and starts NFS services

rmnfs Stops and un-configures NFS services

mknfsexp -d /directory Creates an NFS export directory

mknfsmnt Creates an NFS mount directory

mount hostname:/filesystem /mount-point Mount an NFS filesystem

nfso -a Display NFS Options

nfso -o option=value Set an NFS Option

nfso -o nfs\_use\_reserved\_port=1

我的主页：http://tlcshxd.3322.org/index.html

第 29 楼：BACKUPS

MKSYSB

------

mkszfile -f Creates /image.data file (4.x onwards)

mkszfile -X Creates /fs.size file (3.x)

mksysb -i (device of file) Creates a mksysb image

mksysb does not save any raw data and will not backup a filesystem that

is not mounted.

SAVEVG

------

savevg -if (device or file) (vg) Creates a savevg image

restvg -q -f (device or file) Restore from a savevg image

Ensure that the restvg command is run from /

mkvgdata (vg) Creates new vgname.data file

CPIO ARCHIVE

------------

find (filesystem) -print | cpio -ocv > (filename or device)

eg find ./usr/ -print | cpio -ocv > /dev/rmt0

CPIO RESTORE

------------

cpio -ict < (filename or device) | more Lists archive

cpio -icdv < (filename or device)

cpio -icdv < (filename or device) ("files or directories to restore")

eg cpio -icdv < /dev/rmt0 "tcpip/\*" Restore directory and contents

cpio -icdv < /dev/rmt0 "\*resolve.conf" Restore a named file

TAR ARCHIVE

-----------

tar -cvf (filename or device) ("files or directories to archive")

eg tar -cvf /dev/rmt0 "/usr/\*"

TAR RESTORE

-----------

tar -tvf (filename or device) Lists archive

tar -xvf (filename or device) Restore all

tar -xvf (filename or device) ("files or directories to restore")

use -p option for restoring with orginal permissions

eg tar -xvf /dev/rmt0 "tcpip" Restore directory and contents

tar -xvf /dev/rmt0 "tcpip/resolve.conf" Restore a named file

AIX ARCHIVE

-----------

find (filesystem) -print | backup -iqvf (filename or device)

Backup by filename.

eg find /usr/ -print | backup -iqvf /dev/rmt0

backup -(backup level 0 to 9) -f (filename or device) ("filesystem")

Backup by inode.

eg backup -0 -f /dev/rmt0 "/usr" -u option updates /etc/dumpdates file

AIX RESTORE

-----------

restore -qTvf (filename or device) Lists archive

restore -qvxf (filename or device) Restores all

restore -qvxf (filename or device) ("files or directories to restore")

(use -d for restore directories)

restore -qvxf /dev/rmt0.1 "./etc/passwd" Restore /etc/passwd file

restore -s4 -qTvf /dev/rmt0.1 Lists contents of a mksysb tape

BACKUPS ACROSS A NETWORK

------------------------

To run the backup on a local machine (cpio) and backup on the remote

machine's (remhost) tape drive (/dev/rmt0)

find /data -print | cpio -ocv | dd obs=32k | rsh remhost \

"dd ibs=32k obs=64k of=/dev/rmt0"

To restore/read the backup (cpio) on the remote machine

dd ibs=64k if=/dev/rmt0 | cpio -icvt

To restore/read the backup (cpio) on the local machine from the remote

machine's (remhost) tape drive (/dev/rmt0)

rsh remhost "dd ibs=64k obs=32k if=/dev/rmt0" | dd ibs=32k \

| cpio -icvt

To run the backup (cpio) on a remote machine (remhost) and backup to

the local machines tape drive (/dev/rmt0)

rsh remhost "find /data -print | cpio -ocv | dd ibs=32k" \

| dd ibs=32k obs=64k of=/dev/rmt0

tar cBf - . | rsh remhost "dd ibs=512 obs=512 of=/dev/rmt0"

Same as above but using tar instead of cpio.

--------------------------------------------------------------------------------

Copying diskettes and tape

--------------------------------------------------------------------------------

COPYING DISKETTES

-----------------

dd if=/dev/fd0 of=(filename) bs=36b

dd if=(filename) of=/dev/fd0 bs=36b conv=sync or flcopy

COPYING TAPES

-------------

dd if=/dev/rmt0 of=(filename)

dd if=(filename) of=/dev/rmt0 or tcopy

--------------------------------------------------------------------------------

VI Commands

--------------------------------------------------------------------------------

:g/xxx/s//yyy/ global change where xxx is to be changed by yyy

sed 's(ctrl v ctrl m)g//g' old.filename > new.filename

Strips out ^M characters from ascii files that have been transferred as binary.

To enter crontrol characters type ctrl v then ctrl ? where ? is whatever

ctrl character you need.

我的主页：http://tlcshxd.3322.org/index.html

第 30 楼：DEVICES

lscfg lists all installed devices

lscfg -v lists all installed devices in detail

lscfg -vl (device name) lists device details

bootinfo -b reports last device the system booted from

bootinfo -k reports keyswitch position

1=secure, 2=service, 3=normal

bootinfo -r reports amount of memory (/ by 1024)

bootinfo -s (disk device) reports size of disk drive

bootinfo -T reports type of machine

ie rspc,rs6ksmp,rspc or chrp

lsattr -El sys0 -a realmem reports amount of useable memory

mknod (device) c (major no) (minor no) Creates a /dev/ device file.

mknod /dev/null1 c 2 3

lsdev -C lists all customised devices ie installed

lsdev -P lists all pre-defined devices ie supported

lsdev -(C or P) -c (class) -t (type) -s (subtype)

chdev -l (device) -a (attribute)=(new value) Change a device attribute

chdev -l sys0 -a maxuproc=80

lsattr -EH -l (device) -D Lists the defaults in the pre-defined db

lsattr -EH -l sys0 -a modelname

rmdev -l (device) Change device state from available to defined

rmdev -l (device) -d Delete the device

rmdev -l (device) -SR S stops device, R unconfigures child devices

lsresource -l (device) Displays bus resource attributes of a device.

cfgmgr Configures devices

cfgmgr -i /dev/cd0 Configure devices and install drivers from /dev/cd0

if required

cfgmgr -S Run in serial, used with a larger number of disks

cfgmgr -l scsi0 Configure devices on adapter scsi0 only

diag Run hardware diagnostic menu

smitty diag Run hardware diagnostic menu

( 7020-40P and 7248-43P machines cannot run diagnostics, use diagnostics

in the SMS menus instead

diag -d (device) Run diagnostics against a device.

lsslot Displays all hot swap slots

lsslot -c pci Lists all pci hot swap slots

lsslot -c pci -a Lists all available pci hot swap slots

drslot Reconfgiures PCI hot-plug slots

drslot -i -c pci -s U0.1-P1-I3 Display a slot, flashes the LED next to

the slot so that it can be identified.

Power Management (PCI machines)

-------------------------------

pmctrl -a Displays the Power Management state

rmdev -l pmc0 Unconfigure Power Management

mkdev -l pmc0 Configure Power Management

--------------------------------------------------------------------------------

TAPE DRIVES

--------------------------------------------------------------------------------

rmt0.x where x = A + B + C

A = density 0 = high 4 = low

B = retension 0 = no 2 = yes

C = rewind 0 = yes 1 = no

tctl -f (tape device) fsf (No) Skips forward (No) tape markers

tctl -f (tape device) bsf (No) Skips back (No) tape markers

tctl -f (tape device) rewind Rewind the tape

tctl -f (tape device) offline Eject the tape

tctl -f (tape device) status Show status of tape drive

chdev -l rmt0 -a block\_size=512 changes block size to 512 bytes

(4mm = 1024, 8mm = variable but

1024 recommended)

dd if=/dev/rmt0 bs=128k count=1 | wc -c

Displays the block size of an unknow tape. Set block size to 0 first.

bootinfo -e answer of 1 = machine can boot from a tape drive

answer of 0 = machine CANNOT boot from tape drive

diag -c -d (tape device) Hardware reset a tape drive.

diag -c -d rmt0

tapechk (No of files) Checks Number of files on tape.

< /dev/rmt0 Rewinds the tape !!!

--------------------------------------------------------------------------------

PRINTERS / PRINT QUEUES

--------------------------------------------------------------------------------

splp (device) Displays/changes printer driver settings

splp /dev/lp0

export $LPDEST="pqname" Set default printer queue for login session

lsvirprt Lists/changes virtual printer attributes.

lsallq Displays all queues

rmvirprt -q queuename -d queuedevice Removes a virtual printer

qpri -#(job No) -a(new priority) Change a queue job priority.

qhld -#(job No) Put a hold on hold

qhld -r #(job No) Release a held job

qchk -A Status of jobs in queues

lpstat

lpstat -p(queue) Status of jobs in a named queue

qcan -x (job No) Cancel a job from a queue

cancel (job No)

enq -U -P(queue) Enable a queue

enable (queue)

enq -D -P(queue) Disable a queue

disable (queue)

qmov -m(new queue) -#(job No) Move a job to another queue

startsrc -s qdaemon Start qdaemon sub-system

lssrc -s qdaemon List status of qdaemon sub-system

stop -s qdaemon Stop qdaemon sub-system

我的主页：http://tlcshxd.3322.org/index.html

第 31 楼：FILE SYSTEMS

Physical Volumes (PV's)

-----------------------

lspv Lists all physical volumes (hard disks)

lspv (pv) Lists the physical volume details

lspv -l (pv) Lists the logical volumes on the physical volume

lspv -p (pv) Lists the physical partition usage for that PV

lspv -M (pv) Lists the PP allocation table for that PV.

If the PV state is "missing" but the disk is okay, use "varyonvg vg" to change

the state of the PV to "active".

chdev -l (pv) -a pv=yes Makes a new hdisk a pysical volume.

chpv -v r (pv) Removes a disk from the system.

chpv -v a (pv) Adds the removed disk back into the system.

chpv -a y (pv) Changes pv allocatable state to YES

chpv -a n (pv) Changes pv allocatable state to NO

migratepv (old pv) (new pv) Moves all LV's from one PV to another PV, both

PV's must be in the same volume group.

Migratepv cannot migrate striped logical volumes, use cplv and rmlv.

replacepv (old pv) (new pv) (4.3.3 onwards)

Volume Groups (VG's)

--------------------

lsvg Lists all volume groups

lsvg (vg) Lists the volume group details

lsvg -l (vg) Lists all logical volumes in the volume group

lsvg -p (vg) Lists all physical volumes in the volume group

lsvg -o Lists all varied on volume groups

varyonvg (vg) Vary On a volume group

varyonvg -f (vg) Forces the varyon process

varyonvg -s (vg) Vary on a VG in maintenance mode. LV commands can be

used on VG, but LV,s cannot be opened for I/O.

varyoffvg (vg) Vary Off a volume group

synclvodm (vg) Tries to resync VGDA, LV control blocks and ODM.

synclvodm -v (vg) Rebuilds the LVCB.

(the vg needs to be varied on before running synclvodm)

mkvg -y(vg) -s(PP size) (pv) Create a volume group

mkvg -y datavg -s 4 hdisk1

reducevg (vg) (pv) Removes a volume group

reducevg -d (vg) (pv) Removes a volume group and delete all LV's on the PV

reducevg (vg) (PVID) Removes the PVID disk reference from the VGDA when a

disk has vanished without the reducevg (vg) (pv)

command being run first.

reducevg -df (vg) (pv) Deletes all LV's from the VG and removes the VG

from the disk. If the last disk in the VG then

the VG is deleted.

extendvg (vg) (new pv) Adds another PV into a VG.

exportvg (vg) Exports the volume group, removes the VG entries and

removes all FS entries from /etc/filesystems but

leaves the mount points.

Note : Cannot export a VG if it has active paging space, turn off paging,

reboot before exporting VG. Exporting removes entries from filesystems

file but does not remove the mount points.

chvg -a y (vg) Auto Vary On a volume group at system start.

chvg -u (vg) Unlocks a locked volume group.

lqueryvg -Atp (pv) Details volume group info for the hard disk.

importvg -y (vg name) (pv) Import a volume group from a disk.

importvg (pv) Same as above but VG will be called vg00 etc.

4.3 onwards, importvg will automatically varyon the VG.

chvg -Q (y/n) (vg name) Turns on/off Quorum checking on a VG.

reorgvg (vg) (lv) Reorganised a fragmented LV, must state an LV at the

command line else the first LV in the VG is picked.

Does not reorg the PP's of striped LV's.

Logical Volumes (LV's)

----------------------

lslv (lv) Lists the logical volume details

lslv -l (lv) Lists the physical volume which the LV is on

lsattr -EHl (lv) Displays more logical volume details

mklv (vg) (No of PP's) (pv Name optional) Create a logical volume

mklv -y (lv) (PP's) (pv name optional) Creates a named logical volume

( use -t jfs2 when creating an LV for a JFS2 filesystem

chlv -n (new lv) (old lv) Rename a logical volume

chlv -x (number) (lv) Change max no of PP's

chlv -s n (lv) Turns of strickness on the LV

extendlv (lv) (extra No of PP's) Increase the size of an LV

rmlv (lv) Remove a logical volume

cplv -v (vg to copy to) -y (new lv) (lv) Copy an LV to a new LV

If copying a filesystem LV, umount the filesystem before copying, otherwise you

will have to fsck the the new LV before the filesystem can be mounted.

If copying a striped LV to an LV that is already created, and the stripe

size is different ( or not even striped , then these new parameters are

maintained when the data is copied to the new LV.

cplv -e (new lv) (old lv) Copy to an existing LV

( new lv must have type as copy use chlv -t copy (new lv) to change

mklv/extendlv -a = PP alocation policy

-am = middle -ac = center -ae = edge

-aie = inner edge -aim = inner middle

migratepv -l (lv) (old pv) (new pv)

Move a logical volume between physical volumes. Both physical volumes

must be in the same volume group !

mklv -y (lv) -t jfslog (vg) (No of PP's) (pv Name optional)

Creates a JFSlog logical volume.

logform (/dev/lv) Initialises an LV for use as an JFSlog

getlvcb -AT (lv) Displays Logical Volume Control Block information

File Systems (FS's)

-------------------

lsfs Lists all filesystems

lsfs -q (fs) Lists the file system details

lsjfs Displays data about all filesystems in CSV style format

mount Lists all the mounted filesystems

mount (fs or lv) Mounts a named filesystem

mount -a Mounts all filesystems

mount all

mount -r -v cdrfs /dev/cd0 /cdrom mounts cd0 drive over /cdrom

crfs -v jfs -d(lv) -m(mount point) -A yes

Will create a file system on the whole of the logical volume, adds entry into

/etc/filesystems and will create mount point directory if it does not exist.

( use -v jfs2 for JFS2 filesystems

crfs -v jfs -g(vg) -m(mount point) -a size=(size of fs) -A yes

Will create a logical volume on the volume group and create the file system on

the logical volume. All at the size stated. Will add entry into

/etc/filesystems and will create the mount point directory if it does not exist.

Use attribute "-a log=/dev/log01" to specify a jfslog devices

Use attrubute "-a bf=true" for a large file enabled filesystem

chfs -A yes (fs) Change file system to Auto mount in

/etc/filesystems

chfs -a size=(new fs size)(fs) Change file system size

chfs -m (new-mount-point) (fs) Change the file system mount point.

rmfs (fs) Removes the file system and will also remove the

LV if there are no onther file systems on it.

defrag -q (fs) Reports the fragment status of the file system.

defragfs -r (fs) Runs in report only defrag mode (no action).

defragfs (fs) Defragments a file system.

fsck (fs) Verify a file system, the file system must be unmounted!

fsck (-y or -n) (fs) Pre-answer questions either yes or no !

fsck -p (fs) Will restore primary superblock from backup copy if the

superblock is corrupt.

( or dd count=1 bs=4k skip=31 seek=1 if=/dev/lv00 of=/dev/lv00

Mirroring

---------

mklv -y (lv) -c(copies 2 or 3) (vg) (No of PP's) (PV Name optional)

Creates a mirrored named logical volume.

mklvcopy -s n (lv) (copies 2 or 3) (pv)

Creates a copy of a logical volume onto another physical volume. The physical

volume MUST be in the same volume group as the orginal logical volume !

rmlvcopy (lv) (copies 1 or 2) Removes logical volume copies.

rmlvcopy (lv) (copies 1 or 2) (pv) From this pv only!

syncvg -p (pv) Synchronize logical partion copies

syncvg -l (lv)

syncvg -v (vg)

mirrorvg (vg) (pv)

Mirrors the all the logical volumes in a volume group onto a new physical

volume. New physical volume must already be part of the volume group.

chfs -a splitcopy=/backup -a copy=2 /data1

Splits off a copy of a 3 way mirror and mount read only for use as an

offline backup.

--------------------------------------------------------------------------------

BOOT LOGICAL VOLUME (BLV) / PROCESSORS / KERNEL

--------------------------------------------------------------------------------

Mirroring does not work with the BLV as it is not a true logical volume,

bosboot must be run against the other disk after mirroring the rootvg.

bootlist -m (normal or service) -o displays bootlist

bootlist -m (normal or service) (list of devices) change bootlist

bootinfo -b Identifies the bootable disk

bootinfo -t Specifies type of boot

bosboot -a -d (/dev/pv) Creates a complete boot image on a physical volume.

mkboot -c -d (/dev/pv) Zero's out the boot records on the physical volume.

savebase -d (/dev/pv) Saves customised ODM info onto the boot device.

lslv -m hd5 &nbs, p; Find out which disk the BLV is on.

bootinfo -y Displays which kernel can be used, 32 or 64 bit

genkex Reports all loaded kernel extensions.

lsdev -Cc processor Lists all processors

lsattr -EHl proc0 Displays attributes of processor 0. AIX 5.1L will

display processor clock frequency.

--------------------------------------------------------------------------------

SYSTEM DUMP

--------------------------------------------------------------------------------

1, AIX 4.2.1 and greater supports system dump to paging space.

2, AIX 4.3.3 and greater supports system dump to mirrored paging space.

3, Primary dump device must be in the rootvg

4, Secondary dump device can be outside rootvg unless it is a paging device.

sysdumpdev -l Lists current dump destination.

sysdumpdev -e Estimates dumpsize of the current system in bytes.

sysdumpdev -L Displays information about the previous dump.

sysdumpstart -p Starts a dump and writes to the primary dump device.

sysdumpstart -s Starts a dump and writes to the secondary dump device.

(MCA machine can also dump if key is in service position and the reset

button is pressed)

sysdumpdev -p (dump device) -P Sets the default dump device, permanently

Analyse dump file :-

echo "stat\n status\n t -m" | crash /var/adm/ras/vmcore.0

snap -gfkD -o /dev/rmt0 Copy dump to tape to send to IBM support, uses tar.

--------------------------------------------------------------------------------

PAGING SPACE (PS's)

--------------------------------------------------------------------------------

lsps -a Lists out all paging space

lsps -s Displays total paging and total useage

lsps (ps)

swappon /dev/ps Activates a paging device eg /dev/paging00

swappoff /dev/ps Deactivates a paging device ( AIX 5.x only

mkps -s(No of PP's) -n -a (vg)

mkps -s(No of PP's) -n -a (vg) (pv)

-n = don't activate/swapon now -a = activate/swapon at reboot

mklv -b n -t paging -y hd6 (vg) (No of PP's) (pv)

Creates paging space using the mklv command.

chps -a n (ps) Turns off paging space.

chps -s(No of PP's) (ps) Increases paging space.

chps -d(No of PP's) (ps) Decreases paging space ( AIX 5.x only

chlv -n (new name) (old name) Change paging space name

rmps (ps) Remove paging space. PS must have been turned off

and then the system rebooted before it can be removed.

Note : Need to change the swapon entry in /sbin/rc.boot script if you are

changing the default paging space from /dev/hd6. You also need to

do a "bosboot -a -d /dev/hdiskx" before the reboot.

/etc/swapspaces File that lists all paging space devices that are

activated/swapon during reboot.

--------------------------------------------------------------------------------

SCHEDULING

--------------------------------------------------------------------------------

crontab -l List out crontab entrys

crontab -e Edit crontab entrys

crontab -l > (filename) Output crontab entrys to a file

crontab (filename) Enter a crontab from a file

crontab -r Removes all crontab entrys

crontab -v Displays crontab submission time.

/var/adm/cron/cron.allow File containing users allowed crontab use.

/var/adm/cron/cron.deny File containing users denied crontab use.

/var/adm/cron/crontab Directory containing users crontab entries.

/var/adm/cron/log Cron log file.

at (now + 2 minutes, 13:05, etc) {return} Schedule a job using at

Command or schell script {return}

{CTRL D}

echo "shutdown -Fr" | at now + 1 minute

at -l

atq Lists out jobs scheduled to run via at command

at -r (at job No)

atrm (at job No) Removes an at job scheduled to run.

/var/adm/cron/at.allow File containing users allowed at use.

/var/adm/cron/at.deny File containing users denied at use.

/var/adm/cron/atjobs Directory containing users at entries.

--------------------------------------------------------------------------------

SECURITY

--------------------------------------------------------------------------------

nulladm /var/adm/wtmp To recreate/clear down the wtmp file.

groups Lists out the groups that the user is a member of

我的主页：http://tlcshxd.3322.org/index.html

第 32 楼：AIX经验搜集

如何建立第二个root用户

AIX V3, V4

建立第二个root用户

步骤如下：

1. 添加一个用户

2. 手工修改/etc/passwd文件中的 user ID 和 group ID

3. 将user ID改为0.

如下，可对用户russ做改动:

将 russ:!:206:1::/u/russ:/bin/ksh

改为：russ:!:0:0::/u/russ:/bin/ksh

具有root权限的用户可执行一条命令，完成特定的任务。如下你可以建立一个用户(shutdown)负责系统的重启动.

在AIX 3.2.5中：

shutdown:!:0:0::/u/shutdown:/etc/shutdown -Fr

在AIX 4:

shutdown:!:0:0::/u/shutdown:/usr/sbin/shutdown -Fr

这个用户一注册，操作系统就会重起。

AIX 的许可权限

说明

如何查看许可权限

许可权限字母的意义

当产生文件或目录时, 如何决定性权限

如何改变文件或目录的所有者或组

如何改变文件或目录的权限

ACL

文件系统和目录安装点权限

NFS安装文件系统许可

本文介绍了AIX 如何控制文件和目录的权限 。所有文件和目录对以下对象都有权限控制 ： 所有者(通常是创造文件的人)

用户组(将该组作为组集之一部分或主组的任何人)

其他人(不是所有者或不属于这个群的用户以外的任何用户)

AIX 4.3版本和RS/6000产品资料可在以下网址获得 ：

http://www.rs6000.ibm.com/resource/aix\_resource/pubs/index/index html.

1.如何查看许可权限

执行命令" Is -l file\_name " 或" Is -ld directory-name" 会显示不同的数据, 左侧的一个 有10个字符的字串：

—rwxrwxrwx

该字符串可分解为3个许可权限集：

—rwx rwx rwx

| | |其他用户许可 。

| |

| |

| 把该组作为主组或作为他们组集之一部分的用户许可。

|

文件或目录所有者许可

这些命令同时也显示文件或目录的所有者和用户组 。

—rwxrwxrwx joe joegroup

许可 所有者 用户组

所显示的所有者名称来自 /etc/passwd。文件的inode 存储用户的Id 。如果你看到的是一个数字而不 是名称，这就是说/etc/passwd文件无法读取，或该文件不存在此用户id 。同样，用户组的名称来自/etc/group 文件.

系统首先检查你是否是所有者；如果是，你会得到所有者的许可权限 ，即使这个许可比 “用户群”或“其他人”许可权限集严格。

然后系统会检查你是否将该组列为主组或组集的一部分。如果是，你会得到组许可，尽管这些 许可限制比“其他人”许可权限集更严格。

如果你既不是所有者也不是组集的一部分，你会得到“其他人”的许可。

2. 许可权限的字母的意义

许可权限字母的意思因文件或目录的不同而有区别。读/写目录的权限与读/写目录所指文件的权限无关。

对于目录，许可权限字母的含义如下：

r --- 指读目录中文件列表的能力，如"ls"命令。

注 ：如果只有读目录列表的许可,则不可以在目录内部执行命令。

x ---目录搜索能力。可以在已知文件名时在目录中执行此文件。一般来说，大多数目录操作都要求读&reg;和搜索(x)权限。

w---同搜索(x)权限共同使用时,在目录中产生或删除文件的能力。如果目录具有写的权限，而没有在文件写的权限，你仍可以删除文件，但不可以修改文件.

t ---指链接权限。设置链接权限可防止除文件所有者、根用户和目录所有者之外的所有用户删除文件，虽然目录中的权限可能允许删除这个文件。在这种情况下，可为“其他人”设置搜索(x)位。

T---同t相同，但不为其他用户设置搜索(x)位。

s---不适用于所有者。用于用户组 ，它成为组的继承位(sgid,亦称set groupid) 。这个目录中创建的所有文件都与此目录有同样的组。

S---与s相同，但不为用户组设置搜索(x) 。

对于文件，许可字母有下列含义：

r--- 指阅读文件的能力。

w--- 指修改文件的能力。只有在目录中有写权限时才可以创建或删除文件。

x--- 指执行文件的能力。

3. 当产生文件或目录时,如何决定性权限？

用户id(uid)用来设置文件所有者。主组用来设置用户组(除非创建文件的目录有Sgid位设置)。

"umask "可设置初始权限。在命令行中键入"umask "命令，查看当前设置 ，如要修改" umask"设置，输入umask号码，如：

umask 022。

"umask" 命令不设置文本文件和脚本文件而只在目录中设置执行位。

如果你创建一个目录或文本文件，可按以下例子设置权限 ：

以umask 022为例 ：

777-022=755

666-022=644

执行"ls -l"，会显示以下权限，(r=4, w=2, x=1)

目录：rwxr-xr-x

文本文件：rw-r--r--

4.如何改变文件或目录的所有者或组？

"chown "和 "chgrp" 命令用于改变所有者和组。只有"根 "用户才能改变文件的所有者。" 根"用户或文件的拥有者可以改变这个文件的组。

使用Chgrp命令：chgrp (New\_group\_name) (file\_name)

使用Chown命令： Chown (new\_owner) (file\_name) 或 chown (new\_owner) (file\_name)

5.如何改变文件或目录的权限？

使用"chmod "命令来改变文件或目录的权限.权限可以用字母或数字表示。读 &reg; = 4 ,写(w) = 2 ,执行(或搜索)(x) = 1

这些数字加到一起就得到用于chmod命令的数字.(注意：如果您正在文件或目录中使用ACL，使用数字模式chmod命令会使ACL无效) 。

如 ：所有者权限：读+写+执行= 4+2+1=7

组 许 可 权 限 ： 读 + 写 = 4+2 =6

其他用户许可权限：阅读=4

chmod 764 file\_name。

如要设置特殊位如suid(设置用户ID)、sgid和链接数位，chmod命令需要第四个数字。

suid=4

sgid=2

link=1

如要在前面的例子中加入suid许可，它的命令是：

chmod 4764 file\_name

如果采用符号模式，首先确定要改变哪个位置所有者 、组[g] ，其他[o] 或所有[a])和要加(+)或减(- )哪些符号。要为所有者增加读和写的权限，可采用以下两种方式：

chmod u+r u+w (file\_name) 或 chmod u+rw (file\_name)

6. ACL

ACL是对标准权限位的扩展。通过修改分配给个人或组的标准权限，对每个文件或目录进行更精细的控制。对每个组或用户，有3种权限分配情况：

PERMIT : 准许对文件或目录的特定权限。

DEMY : 限制对文件或目录的特定权限。

SPECIFY : 明确地定义文件或目录权限。

"acledit "命令用于建立ACL。首先必须设置文本编辑器.如：export EDITOR=/usr/bin/vi。

然后使用：acledit file\_name

屏幕上将会显示：

attributes:

base permissions

owner (rcunning): rwx

group (staff)： r—

others: ---

extended permissions

disabled

要设置扩展的权限，将“disabled ”设置改为“enabled”：

extend permissions

enabled

使用permit、deny 或specify关键字来定义扩展权限。前面的例子表明只有所有者能对这个文件写操作。组成员能读此文件而其他用户则没有任何许可权限。如果要使用户" joe"能够读写这个文件，用以下命令：

extended permissions

enabled

permit rw- u: joe.

要允许组用户joegroup读这个文件，用以下命令：

rermit r-- g: joegroup

你能够通过在同一行中合并多个条目来对权限进行微调。如果只想为pete提供读写权，而他是系统组的一部分，则用以下命令：

permit rw- u: pete, g: system

要为几个用户或组增加许可权限，则使用分行命令 ：

permit rw- u: joe

permit rw- u: pete

使用" ls -el"命令，查看ACL是否已在文件中设置。如"ls -el profile"命令显示：

— rwxw--------+

最后的+表示文件已具 有有效的ACL 。

注意：使用有数字争议的chmod 命令将使文件或目录的ACL无效。

7.文件系统和目录安装点权限

文件系统安装在目录安装点上。安装点和文件系统都有权限。安装点的权限由文件系统创建时所采用的umask设置来决定。

虽然已安装文件系统的权限优先于安装点的权限，但安装点必须尽可能限制每个人的搜索权限(也就是111)来避免不可预测的结果。记住，安装文件系统后，不能看到安装点上的权限。在检查或改变安装点权限之前，必须卸载(umount )文件系统。

在创建文件系统时，缺省权限来自基本文件,同时sgid (组 继 承)位被设置。用户的umask不用于文件系统中而只用于基础安装点。

8.NFS安装文件系统许可

网络文件系统(NFS)安装在当地目录中。目录安装点在创建网络安装文件系统时建立。安装点在建立时使用当前的umask设置来决定许可权限。

NFS 安装文件系统使用一个叫作nobody的特殊用户id。这个uid一般都是很大的数字，以便使它不会和真实用户id发生冲突。除非NFS服务器在 /etc/passwd中有您的用户id(不是文本名) ，否则你在远程安装文件系统时所享有的权限只使用该假用户nobody。如果你的用户id碰巧与远程系统的某个有效id相吻合，你就会成为是由那个所有者创建的所有文件的所有者。这有可能导致无法预见的严重后果。如果你希望能够在远程系统中创建和拥有文件，你的本地系统和服务器系统必须具有与 /etc/passwd文件中相匹配的用户(用户名称和相同的id号码)。而且文件系统必须以能够读和写的形式输出。

"根"用户是一个特殊情况。由于“根”用户id在所有系统中都是0，如果没有特殊保护，任何安装该文件系统的系统都将成为该服务器系统的根。因此，NFS文件系统的输出必须带有一些主机名称的" 根"接入，以便使您可以从这些主机名称中要求特殊的"根"接入。如果文件系统不以这种方式输出，"根"就成为用户nobody.

如何限制用户改变密码

RS6000，AIX V4

如何限制用户改变密码?

可以用命令pwdadm -f ADMIN username来实现，如果想让用户恢复更改密码的权利，运行pwdadm -f ADMCHG username 来重置.

非root用户的登录问题

本文档描述了非root用户的登录及权限问题, 以及这些问题如何通过检查目录和文件的权限, 属主及属组来解决.

问题的症状

. 用户得到下面的错误信息, 可能指明组文件丢失或被破坏:

3004-010 设置终端属主和模式失败

/etc/passwd 文件中对应该用户的主组不能在/etc/group文件中被找到.

如 tps:!:215:1::/u/tps:/bin/ksh

在上例中, 组号为1. 检查/etc/group文件确认组号1存在.

. 只有root用户可以登录, 一般用户得到下面的错误信息:

3004-009 运行登录初始程序失败

或

系统不可用

. 执行命令 su - [user\_name] 时返回如下错误:

3004-505 不能设置进程环境

. 用户登录后得到如下错误信息:

0653-345 权限被拒绝

(当登录后进行任何操作时)

或

ksh: pwd: 不能访问父目录

(当登录后执行pwd命令时)

这些现象是由于用户不能执行登录初始程序或由于用户主目录的权限问题造成的.

检查问题文件和目录的步骤

下述步骤说明如何检查有权限问题的文件或目录.

如果任何文件或目录的权限有问题, 使用命令 chmod, chown 或 chgrp 更改相应的权限, 属主或属组.

如果符号链接丢失, 使用ln命令重建它.

例如, 要创建/bin链接到/usr/bin, 执行下面的命令:

ln -s /usr/bin /bin

步骤

1. 以root身份登录

2. 如果非root用户登录时得到的是系统不可用的错误, 则继续本步骤. 否则, 跳到下一步.

用命令 ls -l /etc/nologin 命令检查文件/etc/nologin.

如果文件/etc/nologin存在, 用命令 rm /etc/nologin 删除它.

对于AIX 4.x, 执行:

cd /

ls -al

输出举例:

drwxr-xr-x 19 bin bin 1024 Dec 12 21:14 .

drwxr-xr-x 19 bin bin 1024 Dec 12 21:14 ..

lrwxrwxrwx 1 bin bin 8 Nov 22 09:37 bin -> /usr/bin

drwxrwxr-x 4 root system 2048 Dec 12 21:12 dev

drwxr-xr-x 12 root system 2048 Dec 12 21:11 etc

drwxr-xr-x 5 bin bin 512 Nov 22 14:51 home

lrwxrwxrwx 1 bin bin 8 Nov 22 09:37 lib -> /usr/lib

drwxr-xr-x 20 bin bin 512 Nov 22 13:33 lpp

drwxr-xr-x 3 bin bin 512 Nov 22 09:37 sbin

lrwxrwxrwx 1 bin bin 5 Nov 22 09:37 u -> /home

drwxr-xr-x 20 bin bin 512 Nov 22 14:24 usr

drwxr-xr-x 12 bin bin 512 Nov 22 12:59 var

3. 执行:

ls -ld /usr/bin /usr/lib /tmp

输出举例:

drwxr-xr-x 3 bin bin 10752 Nov 22 12:53 /usr/bin

drwxr-xr-x 28 bin bin 4096 Dec 15 17:08 /usr/lib/

drwxrwxrwt 8 bin bin 2560 Jan 22 14:46 /tmp/

4. 执行:

ls -l /usr/bin/csh /usr/bin/ksh /usr/bin/bsh

输出举例:

-r-xr-xr-x 2 bin bin 341020 Nov 22 09:37 /usr/bin/bsh

-r-xr-xr-x 1 bin bin 154412 Nov 22 09:37 /usr/bin/csh

-r-xr-xr-x 4 bin bin 230148 Nov 22 09:37 /usr/bin/ksh

确定用户的主目录. 在这些步骤中, 假定用户的ID和目录为"user\_one".

5. 执行:

ls -ld u/user\_one (use path of user's directory)

输出举例:

-drwxr-xr-x 9 user\_one system 7680 Dec 24 15:00 /u/user\_one

该目录应被此用户所有, 并且此用户应对它有rwx权限.

6. 执行:

cd /u/user\_one

ls -al | pg

输出举例:

drwxr-xr-x 9 user\_one system 7680 Dec 24 15:00 .

drwxr-xr-x 71 bin bin 1536 Dec 14 09:37 ..

"."目录的所有者应为该用户. ".."目录的权限对于组和其它用户至少应为r-x.

7. 如果用户仍然有权限被拒绝的问题, 但没有登录的问题, 则可能是由于文件系统mount点的权限造成的.

为了检查mount点的权限, 文件系统必须首先被unmount. 一些文件系统的mount点则只能在进入系统维护模式的情况下进行检查.

如何屏蔽某一用户的ftp访问?

产品: AIX

平台: RS

机型: RS6000

如何屏蔽某一用户的ftp访问?

将被拒绝的用户名加入到/etc/ftpusersw文件中

var/adm/wtmp档案太大怎么办

产品：RS/6000

软件：AIX

/var/adm/wtmp文件保存所有用户登录的讯息,随著时间会增长到很大，/var/adm/wtmp档案太大时怎么办？

/var/adm/wtmp档案太大时,有时需要清理或编辑整理。

要清理它,执行cp /dev/null /var/adm/wtmp.

要编辑整理部分清理,用fwtmp命令先将文件wtmp变成ASCII格式的档案dummy.file:

/usr/sbin/acct/fwtmp < /var/adm/wtmp > dummy.file,

编辑之後用

/usr/sbin/acct/fwtmp -ic < dummy.file > /var/adm/wtmp

再将ASCII文件转变成二进位文件.

JESMSG显示屏幕,进行类似于上述2中的操作即可.

AIX 用户的系统资源使用限制

说明

适用操作系统

网络配置步骤

本文介绍的是AIX用户的系统资源使用限制，以及修改方法。

适用操作系统

AIX V4

网络配置步骤

AIX 用户使用的系统资源限制包括两个概念 --- 硬限制(hard limits) 和软限制(soft limits)。

hard limits自AIX 4.1版本开始引入。hard limits 应由AIX系统管理员设置，只有security组的成员可以将此值增大，

用户本身可以减小此限定值,但是其更改将随着该用户从系统退出而失效。使用下列命令可以查看hard limits的限定值：

ulimit -Ha

soft limits 是AIX核心使用的限制进程对系统资源的使用的上限值。此值可由任何人更改，但不能超出

hard limits值。这里要注意的是只有security组的成员可使更改永久生效，普通用户的更改在其退出系统

后将失效。使用以下命令可以查看soft limits的设置：

ulimit -a

下面为系统的soft limits的默认值：

3.2

4.1-4.3

===============

=================

fsize = 2097151

fsize = 2097151

core = 2048

core = 2048

cpu = 3600

cpu = -1

data = 131072

data = 262144

rss = 65536

rss = 65536

stack = 8192

stack = 65536

nofiles = 2000\*

nofiles=2000 \*

\* 该值(nofiles)只能在AIX 4.3.1 或以后的版本中更改。

上述定义作为默认值存放在文件 /etc/security/limits 中，在新用户被加进系统后生效。直接更改此文件中的定义值

需要将系统重新启动以便使更改生效。将相应值该为"-1" 表示不受 soft limits的限制（unlimited)。

下面我们将就各字段逐一进行介绍：

fsize 用户创建的文件大小限制。此定义值（512字节为单位）为该用户可以生成的最大文件的大小。

core 生成的core文件大小的限制（512字节为单位）。

cpu 用户进程可用cpu的限定值（以秒为单位）。普通用户只能将此值减小，root可以将此值增大。这里要注意的

是进程使用CPU的时间取决于AIX Kernel（核心程序）进程调度算法，该值在此仅做参考。

data 进程数据段大小的限定值（以字节为单位）。

stack 进程堆栈段大小的限定值（以字节为单位）。

rss 进程常驻内存段的限定值（以字节为单位）。AIX核心并不参考此限定。

nofiles 进程中打开文件的最大数量。此限定在AIX 4.3.1之前的版本中固定为2000。在AIX 4.3.1及其之后的版本中

可将此值增大至32767。

下面介绍三种修改上述限定值的方法：

1. 编辑文件/etc/security/limits，直接修改各定义值。此更改在系统重新启动后生效。

2. 使用命令ulimit修改默认值。例如：

ulimit -f value

ulimit -c <value>

ulimit -t <value>

ulimit -d <value>

ulimit -s <value>

ulimit -m <value>

ulimit -n <value>

将修改fsize, core, cpu,data, stack, rss和nofiles的soft limit值。

3. 使用命令chuser修改某用户的限定值。例如：

chuser fsize=<value> <username>

chuser core=<value> <username

chuser cpu=<value> <username>

chuser data=<value> <username>

chuser limit=<value> <username>

chuser rss=<value> <username>

chuser nofiles=<value> <username>

将用户“username”的soft limits改为值“value”。

chuser hard\_fsize=<value> <username>

chuser hard\_core=<value> <username>

chuser hard\_cpu=<value> <username>

chuser hard\_data=<value> <username>

chuser hard\_limit=<value> <username>

chuser hard\_rss=<value> <username>

chuser hard\_nofiles=<value> <username>

将用户“username”的hard limits改为值“value”。

[技术文档]目录:

[技术文档]说明/内容摘要:

[技术文档]详述:

更改HACMP环境里网卡的ip地址

AIX，HACMP

如何更改HACMP环境里网卡的ip地址？

更改HACMP环境里网卡的ip地址的步骤如下：

1.运行命令:

/usr/sbin/cluster/utilities/cllsif > /tmp/cllsif.orig ，

/tmp/cllsif.orig文件里就包含您的HACMP网卡当前的ip地址；

2.停止您系统上正在运行的与当前ip地址有关所有应用；

3. 停止所有有关节点系统的HACMP服务：

smitty hacmp

Cluster Services

Stop Cluster Services

4.停止所有有关节点系统的TCPIP服务 :

stopsrc -g tcpip

5.更改各个节点的HACMP网卡的boot和standby地址：

smitty tcpip

Minimum Configuration and Startup

6.在各个节点上编辑/etc/hosts文件，更改成要求的ip地址：

boot,service和standy的ip地址；

7. 在各个节点上启动TCPIP服务：

startsrc -g tcpip

8. 用ping或者telnet命令来测试并确保各个节点在网络中是可访问的；

9.在HACMP的配置里更改ip地址：

smitty hacmp

Cluster Configuration

Cluster Topology

Configure Adapters

Change/Show and Adapter

10. 同步cluster的拓扑结构：

smitty hacmp

Cluster Configuration

Cluster Topology

Synchronize Cluster Topology

11.检验环境：

smitty hacmp

Cluster Configuration

Cluster Verification

12. 运行命令：

/usr/sbin/cluster/utilities/cllsif > /tmp/cllsif.new ,

比较/tmp/cllsif.new文件和先前的/tmp/cllsif.orig文件，所有ip地址的改变都将反映出来；

13.把各个节点上除了rootvg外的所有卷组都置为不可用状态：

varyoffvg vgname ；

14.启动 HACMP服务：

smitty hacmp

Cluster Services

Start Cluster Services

15. 如果用了DNS服务，对DNS服务器做必要的更改；

16. 用新的ip地址进行HACMP的资源接管测试。

如何搜集TCP/IP问题的相关资料

在TCP/IP发生问题时，可搜集相关信息，寄给IBM来诊断

这篇文章主要描述了如何搜集TCP/IP问题的相关资料，然后可以把这些资料传给IBM做技术分析，通过本方法搜集的信息资料比较完全，以避免来回搜集资料的烦琐。这些搜集的资料既可以用来诊断比较简单明显的问题，也可以用来诊断比较复杂的问题甚至是系统bug。

步骤如下：

一、

df命令检查/tmp空间，因为snap将在/tmp下产生ibmsupt目录，并将搜集的所有资料放到该目录下，所以该目录必须有较大空间存放资料，如果/tmp不够，请扩充之，如果不愿在该目录下存放资料，可在snap命令后用 -d 参数指定新的目录。

二、

运行snap -r删除上次所做的snap资料，将出现

The following directories and files will be deleted:

----------------------------------------------------

/tmp/ibmsupt/filesys (directory)

/tmp/ibmsupt/general (directory)

/tmp/ibmsupt/kernel (directory)

/tmp/ibmsupt/nfs (directory)

/tmp/ibmsupt/other (directory)

/tmp/ibmsupt/tcpip (directory)

/tmp/ibmsupt/testcase (directory)

Do you want me to remove these directories (y/n)?

选择y，删除上次所做的snap资料

运行snap -gGtkfn 命令搜集资料，将出现

Checking space requirement for general information..............................

........................................ done.

..Checking space requirement for tcpip information..................... done.

Checking space requirement for nfs information.............. done.

Checking space requirement for kernel information............. done.

Checking space requirement for filesys information..................... done.

Checking for enough free space in filesystem... done.

\*\*\*\*\*\*\*\*Checking and initializing directory structure

Creating /tmp/ibmsupt/filesys directory tree... done.

Creating /tmp/ibmsupt/kernel directory tree... done.

Creating /tmp/ibmsupt/nfs directory tree... done.

Creating /tmp/ibmsupt/tcpip directory tree... done.

Creating /tmp/ibmsupt/general directory tree... done.

Creating /tmp/ibmsupt/general/diagnostics directory tree... done.

Creating /tmp/ibmsupt/testcase directory tree... done.

Creating /tmp/ibmsupt/other directory tree... done.

\*\*\*\*\*\*\*\*Finished setting up directory /tmp/ibmsupt

Gathering general system information......... done.

Including Pd\* ODM files in general information.......... done.

Gathering scanout information..done.

Gathering tcpip system information...................... done.

Gathering nfs system information............... done.

Gathering kernel system information.............. done.

Gathering filesys system information.................................. done.

该命令将搜集文件系统、内核、NFS、TCPIP等等方面的信息。

三、

用vi编辑器，生成readme文件如下

/tmp/ibmsupt/testcase/readme

在该文件中回答下列问题：

1、发生问题的程序名或设备名

2、如果发生问题的是一个网络设备，请描述网络适配器的种类

3、尽可能地描述问题，包括：

a、如果问题是关于网络性能，请说明目前的网络吞吐量和期望值，避免用“慢”这种词来描述，请指名速度比如20Kbytes/sec。

b、如果该问题是关于某程序的，请说明仅针对于该程序还是多个程序，如果多个程序，请指明之。

c、如果问题比较易于重现，请说明重现方法以及错误输出。

d、如果问题是关于一个non-AIX程序，请说明其和AIX的关联关系，以及AIX所产生的错误

e、这么问题是否是第一次产生，如果是，请说明其发生前系统或网络所发生的变动

4、填入任何其它你以为相关的信息

四、

以下步骤仅用于问题比较易于重现的情况

1、在有问题的机器上运行iptrace

startsrc -s iptrace -a "-b -d <IP address of remote client> /tmp/ibmsupt/testcase/iptrace.bin"

2、重现问题

3、运行命令

stopsrc -s iptrace

4、运行netstat命令

netstat -v > /tmp/ibmsupt/testcase/netstat-v.out

netstat -in > /tmp/ibmsupt/testcase/netstat-in.out

netstat -rn > /tmp/ibmsupt/testcase/netstat-rn.out

netstat -D > /tmp/ibmsupt/testcase/netsat-D.out

netstat -s > /tmp/ibmsupt/testcase/netsat-s.out

netstat -an > /tmp/ibmsupt/testcase/netsat-an.out

netstat -m > /tmp/ibmsupt/testcase/netsat-m.out

5、运行命令

arp -an > /tmp/ibmsupt/testcase/arp-an.out

no -a > /tmp/ibmsupt/testcase/no-a.out

五、

遵循以下步骤将文档发给IBM

1、snap -c 将所搜集的文件归档，该命令将产生

/tmp/ibmsupt/snap.pax.Z或/tmp/ibmsupt/snap.tar.Z，如例：

#snap -c

Creating compressed pax file...

Starting pax/compress process... Please wait... done.

-rw------- ----1 0 ----0 ------717829 Sep 17 11:34 snap.pax.Z

2、将该文件发送给IBM

在AIX中如何对用户卷组进行卷组备份

在AIX中可以使用mksysb对系统rootvg进行备份，对于其他用户卷组该如何备份？在本文将着重介绍在AIX中如何创建、验证和恢复用户卷组的备份。

由于在AIX中 mksysb命令只能备份rootvg中文件系统的内容，所以还需要有其他途径来备份用户卷组上的内容。使用系统中的savevg 命令就可为用户卷组 创建一卷组备份，使用restvg 命令恢复savevg 所做的备份映像 ，包括重新创建用户卷组、逻辑卷和文件系统等操作。

需要注意的是，所有用户卷组上的逻辑卷的定义会写到备份设备上并在重建卷组时创建，但只有已mount的文件系统中的内容才会在执行savevg命令 时备份到备份介质上。Savevg命令不能备份裸设备上的内容。

附加下载文件：savevg.doc

AIX中如何解决不能释放光驱的问题

产品: RS6000

平台: AIX

在AIX上由于异常终止或其它原因经常会遇到不能正常释放光驱的问题

#fuser -kxuc /dev/cd#

或#fuser /dev/cd# 列出当前访问此设备的进程，然后使用kill -9 pid终止所有进程。

"Volume Group Locked" 恢复步骤

产品：AIX

平台：RS/6000

遇到错误消息 “ 0516-366 putlvodm: volume group rootvg is locked, try again”，可按下列步骤进行恢复。

本文所述方法针对AIX V4.2 或更新的版本。

逻辑卷操作的异常终止有时会造成卷组（VG）被锁定，这时需用命令

chvg -u [vgname]

将被锁定的卷组解锁。“vgname”为被锁定的卷组 。

查看SSA磁盘中pdisk与hdisk的对应关系

AIX V4

怎样查看SSA磁盘中pdisk与hdisk的对应关系?

ssaxlate命令显示pdisk与hdisk的对应关系.

#ssaxlate -l LogicalDiskName

#ssaxlate -l PhysicalDiskName

无法删除文件系统

产品：RS6000

平台：RS

软件版本：AIX 4.3

可以mount或unmount一个文件系统，但却无法删除该文件系统，在删除时，系统报错：0516-306：get lvodm: unable to find <lvname> in the device confgiguration data.

发生此错误的原因可能是ODM数据库中的定义与实际不一致。可以用命令 lsvg -l rootvg 检查该文件系统的类型，若类型显示为？？？，则用命令 synclvodm -P rootvg 和 syncvg -v rootvg 进行同步，之后再用命令 lsvg -l rootvg 显示出正确的文件类型。此时可以成功删除该文件系统。

AIX 5.1文件系统类型简介

本文简单介绍AIX 5.1支持的文件系统类型。包括：

JFS(Journaled File System)

JFS2(Enhanced Journal File System)

GPFS(General Parallel File System)

RAM File System

Journaled File System

JFS 是AIX 4.3.3及其以前版本以及AIX 5.1 32位核心环境中的默认文件系统类型。JFS可以根据所记录的文件系统中的文件元数据（metadata）修复被损坏的文件系统。通过文件系统日志，系统可以将文件元数据的更改信息记录到文件系统的预留区域中，而对文件的写操作则在此日志更新完成之后进行。这样的文件系统结构有利于最大限度地对文件系统进行修复。

Enhanced Journal File System

Enhanced JFS (也称为 JFS2) 提供了比JFS 更优的扩展性能（如文件系统的大小等）。另外，在64位AIX核心中JFS2是系统默认的文件系统类型。当然，用户无论在32位还是在64位环境中都可以选择是使用JFS2还是JFS。在这里我们建议在32位核心中使用JFS文件系统，在64位核心中使用JFS2。

JFS2的创建和使用方法与JFS相同，即通过smit完成。

General Parallel File System (GPFS)

GPFS 是高性能，共享磁盘式文件系统，可以快速访问共存于一个群集（Cluster）中所有节点的数据。 GPFS通过多个磁盘条带化IO可以提供高性能IO服务，同时通过提供日志管理，文件复制以及服务器和磁盘的相互接管功能，提供了高可用性。

有关GPFS的使用，请参见IBM Redbook: GPFS on AIX Cluster: High Performance File System Administration Simplified。IBM Redbook站点为：

http://www.redbooks.ibm.com/

Network File System

NFS 属于分布式文件系统，允许用户从远程访问文件及目录，并且视这些文件及目录为本地资源一样。例如，用户可以使用操作系统提供的命令创建，删除或修改远程文件及目录。

RAM File System

RAM 盘是将内存中的一块空间仿真为磁盘。相对于物理磁盘，使用RAM盘可以极大地提高IO性能，但是建议在针对不需要永久保存的文件IO操作环境中使用 RAM盘，因为在系统宕机或重启后RAM盘中的所有数据会丢失。

使用系统命令 mkramdisk可以创建RAM盘。下面的例子说明如何创建一个20MB的RAM盘，并在其上创建文件系统：

mkramdisk 40000

ls -l /dev | grep ram

mkfs -V jfs /dev/ramdiskx

mkdir /ramdisk0

mount -V jfs -o nointegrity /dev/ramdiskx /ramdiskx

其中x 是RAM盘的逻辑设备号。删除RAM盘时，使用命令rmramdisk。此外，RAM 盘在系统重启后被自动删除。

如何替换镜像的卷组中故障的物理卷

rs6000,pSeries

如何替换镜像的卷组中故障的物理卷

以下情形替换镜像卷组中与物理卷相关的故障或正在发生故障的磁盘。

在指示信息中，使用配置管理器配置新的磁盘（名为 hdisk10），然后使用replacepv命令替换镜像卷组中的物理卷而不丢失物理卷的内容，

该镜像卷组至少部分地驻留在故障的磁盘驱动器（名为 hdisk02）上。完成以下过程无需重新引导或调度停机时间。

1. 选择一个至少与故障的磁盘一样大容量的新磁盘驱动器。

2. 利用超级权限，运行配置管理器配置新的磁盘。在命令行输入以下命令：

cfgmgr -l hdisk10

-l 标志允许仅配置指定的设备以及任何“子”设备。如果没有该标志，则 cfgmgr 命令对整个系统运行配置管理器。

3. 使用以下命令替换物理卷，使它可以开始使用新磁盘。

注:如果逻辑卷的镜像是旧的，则 replacepv 命令无法正确工作。

replacepv hdisk02 hdisk10

4. 如果相关的镜像卷组为 rootvg，必须再运行以下命令将故障磁盘从引导映象清除并向引导映象添加新的磁盘。

5. chpv -c hdisk02

6. bootlist hdisk10

bosboot -a

chpv -c 命令将 hdisk02 从引导映象清除。

bootlist 命令将 hdisk10 添加到系统可能从其引导的引导设备列表。

bosboot -a 命令在缺省的引导逻辑卷上创建完整的引导映象。

此时，物理卷 hdisk02 现在映射到新配置的 hdisk10。

pSeries

重新设置未知的 root 用户密码

以下步骤描述了当系统的 root 用户密码不可用或未知时如何恢复对 root 用户特权的访问。

以下步骤需要一些系统停机时间。可能的话，请调度好停机时间，使其尽可能少地影响作业量，从而防止可能的数据或功能的丢失。

1. 请将与当前安装程序同样的版本和级别的产品介质插入相应驱动器。

2. 打开机器电源。

3. 当图标屏幕出现或听到蜂鸣声的时候，重复按下 F1 键直到出现系统管理服务菜单。

4. 选择多引导。

5. 选择从何处安装。

6. 选择放有产品介质的设备，然后选择安装。

7. 选择 AIX 版本图标。

8. 通过按 F1 键然后按 Enter 键将您的当前系统定义为系统控制台。

9. 选择首选语言的号码，然后按 Enter 键。

10. 通过输入 3 并按 Enter 键选择启动维护方式以进行系统恢复。

11. 选择访问根卷组。系统会显示一条消息，说明如果在此处更改了根卷组，则不重新启动将无法返回“安装”菜单。

12. 输入 0 并按 Enter 键。

13. 从列表中输入相应的卷组号码并按 Enter 键。

14. 通过输入 1 并按 Enter 键选择访问该卷组并启动 shell。

15. 在 #（数字符号）提示状态下，在命令行提示符后输入 passwd 命令以重新设置 root 用户密码。例如：

16. # passwd

17. 更改 root 用户密码

18. root 用户新密码：

请再次输入新的密码：

19. 要将缓冲区中的任何东西写入硬盘并重新引导系统，请输入以下命令：

sync;sync;sync;reboot

当登录屏幕出现的时候，在步骤15中设置的密码现在允许访问 root 用户特权。

AIX系统级命令简介

在AIX操作系统上有很多的命令。这里介绍一些系统级的命令，它将有助于回答一些常见问题。大家以此做参考，并补充修改。

以下命令在AIX 5.1上测试通过。

以下命令在AIX 5.1上测试通过。

• 关于内核

显示AIX系统内核是32位还是64位：

bootinfo -K

如何改变内核模式（32位或64位）

/unix文件连接到一个可引导的映像。通过命令ls -l /unix查看：

/unix -> /usr/lib/boot/unix\_up # 32 bit uniprocessor kernel

/unix -> /usr/lib/boot/unix\_mp # 32 bit multiprocessor kernel

/unix -> /usr/lib/boot/unix\_64 # 64 bit multiprocessor kernel

在AIX系统安装时，缺省安装的内核是32位。可以用如下命令更改内核模式：

ln -sf /usr/lib/boot/unix\_64 /unixln -sf /usr/lib/boot/unix\_64 /usr/lib/boot/unixbosboot -ad /dev/hdiskxxshutdown -r

注意：/dev/hdiskxx是指引导逻辑卷/dev/hd5所在的硬盘。可通过下面命令来查看xx是几：

lslv -m hd5

• 关于硬件

显示机器硬件是32位还是64位：

bootinfo -y

查看机器的物理内存是多少：

bootinfo -r

或

lsattr -El sys0 -a realmem

查看机器是否支持64位内核（是否64位硬件）

/usr/sbin/bootinfo -p

如果返回32，则表示硬件是32位的；如果返回的是chrp，则表示硬件是64位的机器。

显示当前磁带设备rmt0的属性：

lsattr -l rmt0 -E

显示缺省的磁带设备rmt0的属性：

lsattr -l rmt0 -D

显示终端设备tty0的登录属性：

lsattr -l tty0 -a login -R

显示系统级属性：

lsattr -E -l sys0

查看当前系统有多少CPU：

lscfg | grep proc

查看当前系统有多少硬盘而且是否被使用：

Lspv

查看当前系统的详细配置：

lscfg -pv

也可以显示某一设备的配置：

lscfg -vl rmt0

查看当前系统的芯片名称、系统名、节点名、型号等：

uname -p # 显示芯片名称，例如：powerpc

uname -r # 显示操作系统的发行号

uname -s # 显示系统名，例如：AIX

uname -n # 显示节点名

uname -a # 显示系统名、节点名、版本、机器ID

uname -M # 显示型号，例如：IBM,7046-B50

uname -v # 显示操作系统版本

uname -m # 显示机器ID

• 关于AIX

查看AIX的版本、发行号、ML（Maintenance Level）级别：

oslevel -r

或

lslpp -h bos.rte

如何改变文件系统大小，例如：将/usr文件系统增加1000000字节：

chfs -a size=+1000000 /usr

如何使用CD：

mount -V cdrfs -o ro /dev/cd0 /cdrom

查看本机的IP地址：

Ifconfig -a

或

host Fully\_Qualified\_Host\_Name

例如：host cyclop.austin.ibm.com

查找哪一个文件集（fileset）包含特定的文件，例如：查找/usr/bin/vmstat属于哪一个文件集。

lslpp -w /usr/bin/vmstat

显示哪一个文件集包含/usr/bin/svmon：

Which\_fileset svmon

查看某一级别的ML中所有文件是否都已经安装：

instfix -i | grep ML

如何确定某一补丁（fix）是否安装，例如：查看IY24043是否安装

instfix -ik IY24043

显示哪些文件集需要安装或修改：

lppchk -v

查看交换区（paging space）的分配和使用：

lsps -a

• 关于卷组和逻辑卷

创建卷组：

mkvg -y name\_of\_volume\_group -s partition\_size list\_of\_hard\_disks

partition\_size单位是MB，它是1到1024之间的一个数。（2的次幂，例如：1, 2, 4, 8, 16, 32等，缺省是4MB。）

创建逻辑卷：

mklv -y name\_of\_logical\_volume name\_of\_volume\_group number\_of\_partition

显示当前系统的所有卷组：

Lsvg

显示卷组rootvg的详细信息：

lsvg rootvg

显示属于rootvg的所有硬盘：

lsvg -p rootvg

如何添加一个硬盘到卷组中：

extendvg Volume\_Group\_Name hdisk0 hdisk1 ... hdiskn

如何替换一个硬盘：

1． extendvg Volume\_Group\_Name hdisk\_new

2． migratepv hdisk\_bad hdisk\_new

3． reducevg -d Volume\_Group\_Name hdisk\_bad

如何做一个逻辑卷镜像：

1． mklvcopy Logical\_Volume\_Name Number\_of\_copies

2． Syncvg Volume\_Group\_Name

在AIX上，有一个很方便的系统管理命令smit。上面提到的命令中，如逻辑卷操作部分就可以smit轻松的完成。

这里介绍的命令只是AIX中的一小部分。更多更详细的内容请参考AIX系统手册和技术红皮书。

数据急救(1): 如何恢复误删除的文件

当我们在目录中为了方便使用#rm -rf \* 删除文件时,有时会发现自己今天真是倒霉，在执行后发现#pwd竟然不是自己想要删除文件的目录,这时才发现备份是多么重要啊。(可惜呀，没做,:- (), (这就是系统超级管理员常犯的错误，手太快)，没办法，以头戕地吧。但你在遭遇了巨痛后，如果还没有完全丧失斗志，那么下面这篇文章也许能帮你找会你的文件和自信。希望和成功往往也只是一念之间\*\_\*\和DOS,winX,以及众多unix系统一样，当文件被删除时，其所占据的物理空间并没有真正从系统中移除，但他们的状态确是未分配空间，所以这时只要没有新的文件产生或者文件变化发生在同一个文件系统中，那么我们完全有可能用系统提供的fsdb (filesystem debugger)来恢复我们的数据特别感谢此文的英文原文作者，他们还有收费的恢复工具，需要时请访问

<http://www.compunix.com/>

备注： 其中包含图片和网页链接，如果看不到，请下载附件中的word文档。

(1) 如何恢复误删除的文件？

(2) 举例说明(红色为我们应输入的)

(3) 相关知识介绍

MAKE A BACKUP OF THE ENTIRE FILESYSTEM BEFORE PERFORMING THESE STEPS!!!

ELSE ( BANG !!!!! ).

Steps to recover a deleted file

-------------------------------

1) "ls -id {dir}"

(where dir is directory where file resided)

Record INODE number for next step.

36872

2) Unmount the filesystem.

3) "fsdb /{Mountpoint}" or "fsdb /dev/{LVname}"

(where Mountpoint is the filesystem mount point, and LVname is

the logical volume name of the filesystem)

4) "{INODE}i"

(where INODE is the inode number recorded in step 1)

This will display the inode information for the directory. The

field a0 contains the block number of the directory.

The following steps assume only field a0 is used. If a value

appears in a1, etc, it may be necessary to repeat steps #5 and

#6 for each block until the file to be recovered is found.

5) "a0b"

(moves to block pointed to by field "a0" of this inode)

6) "p128c"

(prints 128 bytes of directory in character format)

Look for missing filename. If not seen, repeat this step until

filename is found. Record address where filename begins. Also

record address where PRIOR filename begins. If filename does

not appear, return to step #5, and selecting a1b, a2b, etc.

Note that the address of the first field is shown to the far left.

Increment the address by one for each position to the right,

counting in octal.

7) "a0b"

(moves to block pointed to by field "a0" of this inode)

If the filename was found in block 1, use a1b instead, etc.

8) "p128e"

(prints first 128 bytes in decimal word format)

Find the address of the file to recover (as recorded in step 6)

in the far left column. If address is not shown, repeat until found.

## su与su - 的区别

**su - username 携带环境变量**

**su username 只是切换用户，但是环境变量还是之前用户的**

## ln命令

**ln是linux 中又一个非常重要命令，它的功能是为某一个文件在另外一个位置建立一个同不的链接，这个命令最常用的参数是 -s，具体用法是：ln –s 源文件 目标文件。**

**当我们需要在不同的目录，用到相同的文件时，我们不需要在每一个需要的目录下都放一个必须相同的文件，我们只要在某个固定的目录，放上该文件，然后在 其它的目录下用ln命令链接（ link）它就可以，不必重复的占用磁盘空间。例如： ln –s /bin/less /usr/local/bin/less**

**-s 是代号（ symbolic）的意思。**

**这里有两点要注意：第一， ln命令会保持每一处链接文件的同步性，也就是说，不论你改动了哪一处，其它的文件都会发生相同的变化；第二， ln的链接又 软链接和硬链接两种，软链接就是 ln –s \*\* \*\*，它只会在你选定的位置上生成一个文件的镜像，不会占用磁盘空间，硬链接 ln \*\* \*\*，没有参数-s ， 它会在你选定的位置上生成一个和源文件大小相同的文件，无论是软链接还是硬链接，文件都保持同步变化。**

**如果你用 ls察看一个目录时，发现有的文件后面有一个 @的符号，那就是一个用ln命令生成的文件，用 ls –l命令去察看，就可以看到显示的 link的路径了。**

**指令详细说明**

**指令名称 : ln**

**使用权限 : 所有使用者**

**使用方式 : ln [options] source dest，其中 option 的格式为 :**

**[-bdfinsvF] [-S backup-suffix] [-V {numbered,existing,simple}]**

**[--help] [--version] [--]**

**说明 : Linux/Unix 档案系统中，有所谓的连结 (link)，我们可以将其视为档案的别名，而连结又可分为两种 : 硬连结(hard link) 与软连结 (symbolic link)，硬连结的意思是一个档案可以有多个名称，而软连结的方式则是产生一个特殊的档案，该档案的内容是指向另一个档案的位置。硬连结是存在同一个档 案系统中，而软连结却可以跨越不同的档案系统。**

**ln source dest 是产生一个连结 (dest)到 source ，至于使用硬连结或软链结则由参数决定。**

**不论是硬连结或软链结都不会将原本的档案复制一份，只会占用非常少量的磁碟空间。**

**-f : 链结时先将与 dest 同档名的档案删除**

**-d : 允许系统管理者硬链结自己的目录**

**-i : 在删除与 dest 同档名的档案时先进行询问**

**-n : 在进行软连结时，将 dest 视为一般的档案**

**-s : 进行软链结 (symbolic link)**

**-v : 在连结之前显示其档名**

**-b : 将在链结时会被覆写或删除的档案进行备份**

**-S SUFFIX : 将备份的档案都加上 SUFFIX 的字尾**

**-V METHOD : 指定备份的方式**

**--help : 显示辅助说明**

**--version : 显示版本**

**范例 :**

**将档案 yy 产生一个 symbolic link : zz**

**ln -s yy zz**

**将档案 yy 产生一个 hard link : zz**

**ln yy xx﻿**

## ctr+c与ctr+z的区别

**ctrl+c和ctrl+z 都是中断命令 ,但是他们的作用却不一样 .**

**ctrl+c是强制中断程序的执行 ,**

**而ctrl+z的是将任务中断 ,但是此任务并没有结束,他仍然在进程中他只是维持挂起的状态 ,**

**用户可以使用fg/bg操作继续前台或后台的任务 ,fg命令重新启动前台被中断的任务 ,**

**bg命令把被中断的任务放在后台执行 .**

**例如:**

**当你vi一个文件是 ,如果需要用shell 执行别的操作 ,但是你又不打算关闭vi,因为你得**

**存盘推出 ,你可以简单的按下ctrl+z,shell会将 vi进程挂起~, 当你结束了那个 shell操作之后, 你可以用fg命令继续 vi你的文件.**

## if-else结构

1、if  条件测试；then

         命令区域

     fi

或

     if  条件测试

     then

     命令区域

2、if  条件测试；then

         命令区域

      else

         命令区域

     fi

或

if  条件测试

then

         命令区域

else

         命令区域

fi

3、if   条件测试1；then

          命令区域

     elif 条件测试2 ; then

          命令区域

     elif 条件测试3 ; then

          命令区域

     else

         命令区域

     fi

   或

if   条件测试1

then

          命令区域

elif 条件测试2 ;

then

          命令区域

elif 条件测试3

then

          命令区域

else

         命令区域

fi

## shell中命令的结束状态

shell中每一个命令执行结束后，都会传回一个结束状态值，

如果执行成功，传回0，

如果执行失败则传回非0值

shell中用内置变量$?用来储存每个命令执行后的返回状态值

## shell中条件测试的写法

1、执行某一个命令的结果

例如：

if grep -q  "^root"   /etc/passwd;then

   echo "find an command "

else

   echo "not found"

fi

2、传回某一命令执行结果的相反值

例如：

if ！grep -q  "^root"   /etc/passwd;then

   echo "not found"

else

   echo "find an command "

fi

3、使用复合命令:（（算式））

如果算式的运算结果不为0，则为逻辑真，否则为逻辑假

例如：

（（2-2））为逻辑假

（（2-3））为逻辑真

（（2<3））为逻辑真

4、使用bash关键词‘[[’、']]'组成的式子：[[  判断式     ]]

注意：[[ 的前面和]]的后面都至少有一个以上的空格才行

if [[  2<3 ]];then

   echo "2<3"

else

  echo  "2>3"

fi

5、使用内置命令 test  判断式

6、使用内置命令:[ 判断式 ]

[  判断式 ] 和 test 判断式   的用法是相同的，两者可以互换

7、使用 -a、-o进行逻辑组合

例如：

[ -r filename1  -a  -x filename2 ]如果filename1可读且filename2可执行，则为真

[ -r filename1  -o  -x filename2 ]如果filename1可读或filename2可执行，则为真

8、命令1 && 命令2

     && 称为逻辑AND，其运作方式是：如果“命令1”执行结果为真，才会执行"命令2",两个都为真，则返回真

例如：

if [ 3>2 ]  && [ 4>3 ];then

    echo "真"

else

    echo "假"

fi

&&的特性经常拿来用作if的隐形用法。例如：

[ -z "$PS1" ]  &&  return

9、命令1 ||  命令2

|| 称为逻辑OR，其运作方式是：如果“命令1”执行为假，才会执行“命令2”，两个命令中有个为真，则返回真

10、&& 和 ||合用

**注意事项：**

**1、[[  ]] 和test 、[  ]的意思和用法是相近的，但[[  ]] 比test和[ ]更自由一点，**

**因为[[  ]] 不必担心bash中的某些特殊字符对运算符的影响，不必写一堆转义字符**

**如[[  str < xyz ]]是正确的语法，但是在[ ] 中却要写成 [ str  \< xyz ]，这种陷阱很容易忘记**

**在[[  ]]中<、>、&&、||等可以自由使用，不必使用转义字符**

**在bash中(())中也不必理会上述提到的特殊字符的影响**

**2、在[[ 判断式 ]] 中如果使用 ==和 ！= 且运算符右方的字符串没有加上单引号或双引号，**

**则==和！=会视为想要对比该字符串所形成的样式，如果相符，则为真**

**自bash3.2版开始，支持在[[  ]]中运用新的对比运算符=~,该运算符的右边不能有引号含括，否则此功能无法使用**

**例如：**

**a="str"**

**if[ [  $a == ??? ]]的意思是说，用$a的值，对比样式???(3个字符的字符串)**

**如果改成**

**if [ $a == "???" ]的意思是判断$a的值和字符串"???"是否相等**

## 文件属性的判断式

-a       文件     如果文件存在

-b       文件     如果文件存在，且文件为块文件

-c       文件     如果文件存在，且文件是字符设备文件

-d       文件     如果文件存在，且文件是目录文件

-f       文件     如果文件存在，且文件是一般文件

-g       文件     如果文件存在，且文件设立了set group id属性

-u       文件     如果文件存在，且文件设立了set user id属性

-h       文件     如果文件存在，且文件为符号链接文件

-k       文件     如果文件存在，且文件设立了sticky位的属性

-p       文件     如果文件存在，且文件为管道文件

-r     文件     如果文件存在，且文件具有可读属性

-w       文件     如果文件存在，且文件具有可写属性

-x       文件     如果文件存在，且文件具有可执行属性

-s     文件    如果文件存在，且文件的大小大于0

-t      文件描述符  如果文件描述符是开启的，且链接了某一个终端

-e     文件   如果文件存在

-O 文件 如果文件存在且为有效使用者id所拥有

-G 文件 如果文件存在且为有效的群组Id所有拥有

-L 文件 如果文件存在且为符号链接文件

-S 文件 如果文件存在且是socket文件

-N 文件 如果文件存在且上次读取后，曾修改过

文件1 -nt 文件2 文件1比文件2新；或者文件1存在，但文件2不存在

文件1 –ot 文件2 文件1比文件2旧；或者文件2存在，但文件1不存在文件1 –ef文件2 文件1和文件2参考到相同的设备和inode编号

## 字符串的条件判断式

-z 字符串 如果字符串的长度为0

-n 字符串 如果字符串的长度不为0

字符串 如果字符串的长队不为0

字符串1 == 字符串2 如果两个字符串相同

字符串 1 = 字符串2 如果两个字符串相同

字符串1 < 字符串2 如果字符串1小于字符串2

字符串1 > 字符串2 如果字符串1大于字符串2

注意如果 < 、>出现在[ ]或test 命令中，要用\转义特殊字符的意义，即要写成

[ 字符串1 \< 字符串2] 和[字符串1 \> 字符串2]，但如果出现在[[ ]]中，则不必

## 算式的条件判断

参数1 –eq 参数2 参数1与参数2的值相同

参数1 –ne 参数2 参数1与参数2的值不相同

参数1 –lt 参数2 参数1小于参数2的值

参数1 -le 参数2 参数1小于或等于参数2的值

参数1 –gt 参数2 参数1的值大于 参数2的值

参数1 –ge 参数2 参数1的值大于或等于参数2的值

## BASH选项的条件判断式：

-o set的选项名称 如果选项是开启的状态

## case 条件判断

if条件判断中提到的if条件判断，一旦要进行条件测试的情况多了起来的话，if和elif的语法就会变得很冗长，此时case条件判断的优势就凸显出来了

case语法：

case 待测项 in

样式串行1）

命令区域1；；

样式串行2）

命令区域2；；

。。。。。。。。

\*）

命令区域；；

esac

如果待测项和某一样式对比符合的话，case会选择执行该样式后面的命令区域

命令区域，可以是单一指令或多行命令，但最后一个指令要以；；结束

如果样式串行中有好几个字符串要对比，要使用|隔开，字符| 有或的意思

样式串可以写成‘样式串行’，或省略左边的括号，写成‘样式串行）’。

样式\*）通常放在case最后的一个区域，用来捕捉不符合指定样式 的其他所有情况。

## for的语法

for 变量 in 串行

do

命令区域

done

其中串行是一些字符串的组合，彼此用$IFS所定义的分隔符(如空格符)隔开

for 循环中 “in 串行”这一段可以省略不写，此时bash会由命令行读取各个参数，轮循代入变量中，作用同一下用法：

for 变量 in $@

do

命令区域

done

$@代表命令行的所有参数

for 命令经常用在分割字符串行，取出各字段元素

IFS=‘：’

PL=”root:x:0:0:root:/root:/bin/Bash”

for f in $PL

do

echo $f

done

比较经典的用法是取出CSV文件各行的字段，CSV文件是一种用，（逗号）分割数据字段的纯文本文件

取出CSV文件各行字段的做法如下：

declare -I i=0

for line in $(cat csvfile.txt)

do

i=i+1

echo -n “第$i行的字段有：”

save\_ifs=$IFS

IFS=‘，’

for f in $line

do

echo -n $f ‘ ‘

done

IFS=$save\_ifs

done

for循环也经常用在处理目录中的文件

DIR=“/root”

for f in $(ls $DIR)

do

echo ‘文件’：$f

done

for循环还有一种用法，和传统C语言for很像

for （（初始条件；终止条件；异动项））

do

命令区域

done

例如

declare -I i，sum

for （（i=1；i<=10；i++））

do

let sum+=I;

done

echo $sum

for无穷循环

for（（；1；））

do

echo ‘日不落’

done

## while循环的语法

while 条件测试

do

命令区域

done

使用while循环读取文件内容

While read line

do

echo $line

done < cvsfile

又例如：

IFS=‘：’

While read f1 f2 f3 f4 f5 f6 f7

do

echo “账号：$f1,login shell是：$f7”

done< /etc/passwd

while无穷循环：

while ((1))

do

命令区域

done

或者

while [1]

do

命令区域

done

或者

while true

do

命令区域

done

或者

while ：

do

命令区域

done

## until循环语法：

until 条件测试

do

命令区域

done

如果条件测试为假，就进入循环

## Select命令

Select 选项变量 in 串行

do

命令区域

done

select 命令建立一个列表，列表的选项，即串行的各个字段。列表提示符号是由bash的内置变量PS3来定义，默认值是#? ,只要改变PS3的值，用户可自定义 提示符号。Select会给各个选项一个编号，编号由1开始递增。当使用者键入编号，选择其中某一选项时，改选项的内容就设值给选项变量，而键入的编号值则放入REPLY变量中

使用者没键入一次编号，就执行一次命令区域的命令

要想结束select列表，可在命令区域输入break或按ctrl+D结束输入

例如：

PS3=’请选择：’

Select f in \*

do

echo “你键入的编号是：$REPLY,选择的文件是：$f”

done

又例如：

PS3=’请选择：’

Menu=”/ /root /etc /home /usr/local /var/log”

Select f in $Menu

do

echo “你键入的编号是：$REPLY,选择的文件是：$f”

done

各选项之间使用$IFS分割字符变量定义的字符隔开（默认是空格）

运用select 可以快速建立具有列表功能的工具程序，例如：账号管理系统、软件安装系统、文件管理系统等。

## break和continue

对于前面提到的4种循环for、while、until、select而言，如果想提前结束循环；可以在循环中使用break命令。执行break时，会跳出一层的循环，如果想跳出多层循环，可以在break命令之后加上层数n（n一定要大于或等于1）

Continue是结束本次循环，也可以在continue命令之后加上层数n

## shell函数的语法

语法1：

function 函数名称（）

{

命令区域

}

语法2：

关键字function 可以省略，以下写法也可以：

函数名称（）

{

命令区域

}

语法3：

如果使用 关键字function,那么函数名称后面的小括号也可以省略

function 函数

{

命令区域

}

函数的调用直接写出函数名即可

函数名称，

如果要传入参数，可以把参数选项写在函数名之后，格式如下：

函数名称 参数1 参数2 参数3 参数4 。。。

执行函数时，函数中最后一个命令的传回值代表函数的结束状态。执行函数，如果遇到return命令，就立即返回，回到调用函数的下一个命令，此时函数的返回值为0

函数的作用范围：

函数仅在定义的shell环境中有效，bash执行函数时，并不会另外再开启一个子shell，如果要传递函数给予子shell环境中使用，可以使用内置命令export 和-f选项， 语法如下：

export -f 函数名称

变量的作用范围：

如果没有特别设置变量的属性，那么，在script中自定义的变量称为全局变量。它的作用范围在整个script中有效，这样的话shell函数中使用变量就很没有保障了，因为如果在script中的其他地方使用了同名的变量就会影响其他地方的 变量的使用，所以在定义变量时最好先规划好变量的作用范围，使用内置命令local 可以是变量属性为函数私有，变量的作用范围为函数范围内。

移动位置参数：

使用内置命令shift可以往前移动位置参数的值，语法如下：

Shift n，其中n是正整数，代表向前移动的次数。n可以省略不写，代表移动一次，执行shift n ，$（n+1）的值会放入$1，以执行shift命令来说（不指定次数）,$2的值会放入$1，$3的值会放入$2，$4的值会放入$3,其他类推。

## 用set指定位置参数值：

Bash的内置命令set，可以指定位置参数的值

例如：

declare -I i=0；

set 61 62 63 64 65 66 67 68

for p in $@

do

（（i++））

echo “第$i个位置参数\$$i = $p”

done

如果要一次重置所有的位置参数，使其值为空，可执行：set –

在程序中运用位置参数传递参数值给函数时，原有的位置参数会被保存，其内涵值不会受到影响。

## 命令行的选项和参数getopts

getopts 语法：

getopts 选项 选项变量

其中，选项行是由各选项的单一字符组成；

如：u：ah

如果选项字符后面，接上：，则表示 选项需要提供一个参数，如果这里的u后面有：，如果在执行script时，选项u没有提供额外的参数，那么bash就会显示“option requires an argument --u”的错误信息，如果不想出现这样的错误，可在选项前面加上：

如：：u：ah

选项变量的作用：getopts由命令行取得选项，把它放在选项变量中，如果选项变量需要额外的参数，参数值会放在 OPTARG这个变量中

例子：

while getopts u:ah opt

do

case $opt in

u)

echo “提供了选项u和参数：$OPTARG”；；

a)

echo “提供了选项a”；；

h)

echo “提供了选项h”；；

\*)

；；

esac

done

## shell脚本库函数

如果某些函数经常使用,就可以把这些文件函数抽出来,集中成一个文件,这个文件称为函数库,往后只要调用这个函数库,就可以加载这些函数备用

调用函数库的语法:

. 路径/函数库文件名

Source 路径/函数库文件名

## 文件代码

操作系统为每一个打开的文件赋予一个编号（0开始）作为追踪文件之用，如对文件进行的读写操作都可以用这个文件编号进行沟通

操作系统默认指定3个文件代码：0给标准输入，1给标准输出，2给标准出错输出

，这个3个文件在建立shell环境时开启

## 开启文件代码

开启文件的语法：

fd <>文件

开启文件，指定文件代码为fd，使用这种方式开启的文件，可供读写

例：exec 6 <> test.txt

## 关闭文件代码

如果文件不使用了，最好关闭文件，交回调用文件代码所占用的系统资源

关闭输入文件代码语法：

fd < &-

例：exec 6 <&-

关闭输出文件代码语法：

fd>&-

例：exec 6 >&-

## 复制文件代码

有时为了操作已经开启的文件代码（0、1、2）为了不影响原有文件代码的作用，最好先复制一份，等操作结束，在还原回去

复制转向输入的文件代码m，存成文件代码n，使n链接到m

n <& m

复制转向输出的文件m，存成文件n，使n链接到m

n>& m

例：

exec 1 >& 6，将标准输出链接到文件代码6，凡是原本输出到标准输出的内容，会转存到文件代码6

例：

exec 6 <& 0将标准输入0，存成文件代码6，即建立文件代码0的链接

例：

#！/bin/bash

datafile=’datafile1’

exec 3<>$datafile

while read <&3

do

echo $REPLY

done

exec 3<&—

上例又可以写成如下形式：

#！/bin/bash

datafile=’datafile1’

exec 3<&0

exec <$datafile

while read

do

echo $REPLY

done

exec 0<&3 3<&-

## REPlY变量

read命令默认会把读取的数据放在变量REPLY中

## 转向输入

语法：

fd < 文件

左边省略fd，fd默认的是0

例子：

wc –l < /var/log/error.log

转向输入后标准输入和error.log建立链接。wc改由error.log读取行数

但wc并不知道它是从哪里读取的数据，反正wc默认是从标准输入读取数据

例：

exec 6 < /var/log/error.log

wc –l <&6

## 转向输出

fd > 文件

左边fd不存在，fd默认的是1,即标准输出转向,常见运用有以下两种

例：

>test.txt

开启空文件test.txt,以下做法效果同上

：> test.txt

：为bash的空命令，什么也不做，转向输出的效果，产生空文件test.txt

储存命令的执行输出

ls > dirs.txt

左方代码不空

exec 6> out.txt //开启文件out.txt作为输出，文件代码6。

echo ‘hello world!’ 1>&6 //将标准输出的信息，储存在文件代码6。

fd >|文件，同>,但>|不理会noclobber选项，只要文件存在，都会强制覆盖，清空文件内容

ls >| dirs.txt

## 转向附加

fd >>文件

## 标准出错伴随标准输出做转向

语法：

>&文件

或

&>文件

Shell指令运行时产生的错误不输出到屏幕转向输出到文件里

例如：ntpdate &> /dev/null

&>的作用，和下面的写法是相同的

>文件 2&>1

例如:ntpdate >/dev/null 2>&1

## 远程客户可通过"login, ftp"登录, 但不可通过"telnet"登录

1. 使用命令"ps -ef" 查看"telnetd"进程是否启动;

2. 检查文件/etc/services中的"telnet port"是否为"23", 如果不是,改为"23"，然后执行" refresh -s inetd".

## 在AIX中设置中文环境

在AIX中使用中文有两种途径：第一是在安装AIX时选择中文语言，装好的系统自动显示中文（这种方法不推荐使用，它没有第二种方法使用起来灵活）。第二是安装AIX时选择英文,系统启动后手工设置中文环境，方法如下：

1. 将AIX系统盘的第一张光盘放入光驱；

2. 运行命令：

    smitty

        --> System Environments

            --> Manage Languange Environment

                --> Change/Show Primary Language Environment

                    --> Change/Show Cultural Convention, Language, or Keyboard

在随后显示的菜单中将光标分别移到以下字段：

Primary CULTURAL Convention

Primary LANGUAGE translation

Primary KEYBOARD

  按下<F4>，从弹出的菜单中选择“IBM-eucCN”将上述字段改为简体中文，按下回车键后系统自动从光盘安装中文环境软件包。此操作完成后重新启动系统，操作界面即为简体中文。

    需要输入中文时使用下列功能键切换输入方法：

        AIX 4.3.3 以前的版本：

|  |  |
| --- | --- |
|  | <Shift> + F1 --- <Shift> + F4 切换到各种中文输入方法；  右<Alt> --- 切换到英文输入； |

        AIX 4.3.3：

|  |  |
| --- | --- |
|  | CTRL + [F2] : 智能 ABC ;  CTRL + [F4] : 拼音输入;  CTRL + [F5] : 五笔输入;  CTRL + [F6] : 郑码输入 ;  CTRL + [F7] : 表形码输入;  CTRL + [F9] : 内码输入;  CTRL + [F10] :英文半角; |

此外，AIX还包含另外两种中文环境，即“UTF8”和“GBK”，它们与“IBM-eucCN”之区别在于包含了繁体汉字的使用。上述三种中文环境的设置方法均相同。

## 怎样查看RS/6000的物理内存

使用命令

lsdev -Cc memory

查看RS/6000配置的物理内存设备，下面为其输出示例：

|  |  |  |  |
| --- | --- | --- | --- |
|  | mem0 | Available | 00-00 Memory |
|  | L2cache0 | Available | 00-00 L2 Cache |

再使用命令

    lsattr -El mem0

输出如下

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | size | 512 | Total amount of physical memory in Mbytes | False |
|  | goodsize | 512 | Amount of usable physical memory in Mbytes | False |

此例说明机器的物理内存为512MB。如果前面lsdev的输出中有设备名 mem1，则使用同样的命令查看其对应的大小并依此类推。L2cache0 为系统二级缓存(Level 2 Cache)的设备名。同样，使用命令

    lsattr -El L2cache0

可以查看其大小。

## 检查AIX上的病毒

#virscan directory

例如: virscan /usr

## 如何查看系统启动信息

操作系统在启动时会在主控台屏幕上显示一些信息。启动后，你仍可以用alog 命令查看这些信息 以及其它没有在屏幕上显示的启动信息，命令如下：

# alog -o -t boot

## 如何禁止终端上的中断(CTRL-C)键

在很多应用系统中，系统管理员希望普通用户只运行自己的应用程序，不能进入系统的shell提 示符下。但通常当用户在终端上敲CTRL-C键时就会退到系统提示符下。为了避免这种情 况发 生，可采取下面措施：

# smit tty

  -> Change / Show Characteristics of a TTY

      -> 选择需要修改的tty端 ?

          -> 将菜单中的INTERRUPT character的值设为^@

但在有些情况下，用户终端可能没有固定的端口号，这时可使用命令行方式，方法如下：

# stty intr ^@

## 如何查看系统的错误报告(error report)

用errpt -a 命令可以看到系统错误日志的详细信息，下面就是一个例子：

LABEL: TAPE\_ERR1

IDENTIFIER: 4865FA9B

Date/Time: Wed Sep 24 10:52:09

Sequence Number: 118

Machine Id: 00800020C000

Node Id: e20

Class: H

Type: PERM

Resource Name: rmt0

Resource Class: tape

Resource Type: 4mm4gb

Location: 04-C0-00-1,0

VPD:

|  |  |  |
| --- | --- | --- |
|  | Manufacturer.......... | ARCHIVE |
|  | Machine Type and Model.. | IBM4326NP/RP !D |
|  | Device Specific.(Z1)... | 562B |
|  | Serial Number........ | DR083T5 0011 |
|  | Device Specific....... | (LI) 30303131 |
|  | Part Number....... | 21H5154 |
|  | FRU Number.......... | 21H5172 |
|  | EC Level .......... | D48101 |
|  | Device Specific.(Z0).... | 0180020283000018 |
|  | Device Specific.(Z3).... | L1 |

Description

TAPE OPERATION ERROR

Probable Causes

TAPE

User Causes

MEDIA DEFECTIVE

DIRTY READ/WRITE HEAD

|  |  |
| --- | --- |
|  | Recommended Actions  FOR REMOVABLE MEDIA, CHANGE MEDIA AND RETRY  PERFORM PROBLEM DETERMINATION PROCEDURES |

Detail Data

SENSE DATA

0601 0000 1101 0000 0100 0000 0000 0000 0102 0000 F000 0300 0000 010A 0000 0000

3B00 0025 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

系统记录的错误类(class)有如下几类：

\* hardware(H): 设备或介质错误

\* software(S): 应用程序错误，系统程序错误，系统内核错误

\* operator(O): 操作员提示信息

\* undetermined(U): 错误不确定

系统的错误类型(type)分为:

\* Permanent(PERM): 永久性，不能恢复。

\* Impending(PEND): 设备或元件不能使用

\* Temporary(Temp): 存在错误但被更正

\* Information(INFO): 系统信息

\* Performance(PERF): 设备或元件的性能低于可接受的程度。

\* Unknown(UNKN): 不能确定错误的程度

## 共享UNIX主机上的打印机

在 TCP/IP 网络环境中，一台UNIX主机上的打印机可被其它UNIX主机共享。提供打印机的主机称 为打印机服务器，使用远程打印机的机器称为工作站(client)。

下面举例说明。假设打印机服务器的主机名为sys1，工作站的主机名为sys2。

|  |
| --- |
| D:\Documents\Doc\Aix\技术问题\AIX问题解答.files\sample-7.jpg |

在打印机服务器一端的配置如下：

a. 按加本地 (local)打印机的方法加一打印机及打印队列。在本例中，假设打印机为lp1, 打印队列名 字为lp1。

b. 通过smit菜单打开(enable) lpd 进程(daemon)。

smit printer

-> Printer Spooling

    -> Manage Print Server

        -> Start the Printer Server Subsystem(lpd daemon)

c. 把 client 端的主机名sys2 加到 /etc/hosts.lpd 文件中。

在 client 端:

配置一远程打印队列：

# smit mkpq

-> 选择 remote - Printer Attached to Remote Host

    -> 选择 Standard processing 进入下面菜单：

Add a Standard Remote Print Queue

Type or select values in entry fields.

Press Enter AFTER making all desired changes.

|  |  |  |
| --- | --- | --- |
|  |  | [Entry Fields] |
|  | \* Name of QUEUE to add | [rlp] |
|  | \* HOSTNAME of remote server | [sys1] |
|  | \* Name of QUEUE on remote server | [lp1] |
|  | TYPE of print spooler on remote server | AIX Version 3 or 4 + |
|  | DESCRIPTION of printer on remote server | [] |
|  |  |  |
|  |  |  |

输入打印队列名称：rlp，打印机服务器的名称: sys1， 及此打印队列对应打印机服务上的那一个打印队列：lp1。

AIX中提供的打印命令都适用于远程队列的打印， 例:

|  |  |
| --- | --- |
|  | # lp -d rlp /etc/passwd  # qprt -P rlp /etc/passwd |

## 如何减少文件系统中的碎块

当一个文件系统中有很多小的碎块时，虽然文件系统看起来有很多空间，但不能存储大的文件。用defragfs命令可以减少文件系统中的碎块，增加连续空间。用法如下：

# defragfs /var

defragfs有两个选项：

-q: 显示文件系统当前的状态。

-r: 显示文件系统当前及如果运行defragfs命令将要变成的状态。

## telnet提示"所有的网络端口已用"

1.检查pty0是否可用:lsdev -Cl pty0

2.检查telnet守护进程是否启动:lssrc -t telnet

3.检查/etc/security/login.cfg中maxlogins是否不够大

4.smitty pty检查最大可用终端是否够用

5.no -a|grep thewall检查网络缓冲是否够用

## 如何查找系统中的某个文件存在于哪个文件集中?

当操作系统上由于未安装特定的文件集而无法执行某个命令时, 可以参考下面的方法确定所需的文件集.

首先确认系统中已安装了文件集 bos.content\_list:

# lslpp -l bos.content\_list

然后用下面的命令查找与指定文件相关的文件集:

# /usr/sbin/which\_fileset <filename>

如, # /usr/sbin/which\_fileset iostat

    /usr/bin/iostat bos.acct

查找指定的文件集中包含哪些文件:

# lslpp -f <fileset>

如, # lslpp -f bos.acct

   ....

   /usr/bin/iostat

   ....

<etc>

## 如何创建 /dev/null 文件

1. mknod /dev/null c 2 2

2. chmod 666 /dev/null

## 如何在AIX 4.3中安装联机手册？

AIX 4.3 安装介质包含两张联机文档光盘（Documentation CD): Base Documentation CD 和 Extended Documentation CD。使用命令man查看系统命令之前

需要按照下列步骤安装相应的软件包：

|  |  |
| --- | --- |
| 1. | 正确设置环境变量LANG的值，如“en\_US”为英文环境，“zh\_CN”为简体中文环境。可以用命令      export LANG=en\_US  设置。 |
| 2. | 使用man查看系统命令之前必须安装下列软件包：   |  |  | | --- | --- | | bos.html.en\_US.topnav.navigate | - Top Level Navigation | | bos.html.en\_US.nav | - Online Navigation | | bos.html.en\_US.cmds.cmds1 | - AIX Commands Reference 1 | | bos.html.en\_US.cmds.cmds2 | - AIX Commands Reference 2 | | bos.html.en\_US.cmds.cmds3 | - AIX Commands Reference 3 | | bos.html.en\_US.cmds.cmds4 | - AIX Commands Reference 4 | | bos.html.en\_US.cmds.cmds5 | - AIX Commands Reference 5 | | bos.html.en\_US.cmds.cmds6 | - AIX Commands Reference 6 |   使用命令  lslpp -l <filesetname>  检查上述软件包是否已经安装。 |
| 3. | 如果需要查询AIX 系统调用及C语言库函数，则要安装软件包 bos.html.en\_US.techref.base - AIX Base Tech Ref  bos.html.en\_US.techref.commo - AIX Commo Tech Ref |
| 4. | Documentation CD中还包含许多其它的手册，可根据需要安装相应的软件包。  在AIX系统中安装软件包可使用smit管理菜单。在命令行输入  smit install\_all |

选择:

|  |  |
| --- | --- |
| 1. | CD-ROM 设备(通常为/dev/cd0)。 |
| 2. | 将光标移至"SOFTWARE to install" 并按下<F4>或“<Esc>+7”。 |
| 3. | 将光标移至要安装的软件包并按下<F7> 或 “<Esc>+7”选择要安装的软件包。 |
| 4. | 选中所有要安装的软件包后按?lt;Enter> 开始安装，之后需再次按下<Enter>确认后继续安装，直至成功。 |

此时就可以使用man查看系统命令了!

## 如何自动启动和关闭软件的运行

**1. 自动启动定制的软件**

在系统引导时, init进程顺序检测/etc/inittab文件, 决定在引导过程中进行什么样的操作.如果应用, 进程或其它的用户操作需要在系统引导过程中进行, 则要在/etc/inittab文件中加入相应的条目.通常的做法是创建一个名为rc.local的脚本文件, 将所有要在系统引导过程中启动的用户应用写入该脚本文件中. 该脚本文件可以在/etc/inittab文件中被调用, 通常是在所有系统应用和服务启动后, 在用户登录被允许之前被调用. 位置通常是在主控登录的条目前.

下面是一个例子:

  rc.local:2:wait:/etc/rc.local > /dev/console 2>&1

在上面的条目中, init将在执行余下的条目之前等待该脚本程序的完成.如果进程, 应用或命令需要通过特定的用户运行, 则可在运行的命令中加入su命令.

如: su - myuser -c "/usr/bin/myapp"

建议在脚本程序中加入应用开始启动和完成启动的输出提示.

如:

  #!/bin/ksh

  echo "Starting User Applications...."

  echo "Starting Application XYZ"

  su myuser "-c /usr/bin/myapp"

  echo "Startup of User Applications completed"

**2. 自动停止定制的软件**

shutdown命令用于关闭AIX系统. 在此操作中会关闭所有AIX子系统及文件系统.如果名为/etc/rc.shutdown的脚本程序存在, 系统会首先执行该脚本程序中的命令. 在该脚本程序中可以加入关闭用户应用的命令. 建议在脚本程序中加入应用停止运行和完成停止运行的输出提示.

如:

  #!/bin/ksh

  echo "Stopping User Applications...."

  echo "Stopping Application XYZ"

  [ commands to stop application ]

  echo "Shutdown of User Applications completed"

## 使用ftp自动传输文件

1. 在用户根目录下创建文件.netrc

2. 设置.netrc的属性为600，owner 为ftp用户

3. 定制.netrc的内容.如:

machine <host\_name> login <ftp\_user> password <ftp\_user\_password>

macdef init

get file1

put file2

....

quit

<space\_line>

注：.netrc必须以一空行结束。

## 如何屏蔽某一用户的ftp访问

将被拒绝的用户名加入到/etc/ftpusersw文件中

## 异步I/O (aio)

如果是同步I/O,当一个I/O操作执行时,应用程序必须等待,直到此I/O执行完.

相反,异步I/O操作在后台运行,I/O操作和应用程序可以同时运行,提高了系统性能.

使用异步I/O会提高I/O流量,如果应用是对裸设备进行操作,这种优势更加明显.

因此象数据库,文件服务器等应用往往会利用异步I/O,使得多个I/O操作同时执行.

|  |  |
| --- | --- |
| 1. | 如何知道是否需要异步I/O?  \* 执行命令:"vmstat # #" 如果"wa"值超过25%.  \* 执行命令:"iostat # #" 如果"%tm\_act"值超过35%. |
| 2. | 查询有几个AIO服务器: pstat -a | grep aios | wc -l |
| 3. | 应该设几个AIO服务器?  一般为硬盘数的10倍,但不要超过80. |
| 4. | AIO驱动程序: bos.rte.aio |
| 5. | 配置AIO为可用: # smit chgaio ->         STATE to be configured at system restart   [available] |

## libc.a 文件恢复

在解答用户的问题时, 经常会遇到系统无法启动,报错:can't find file or damaged file libc.a.当遇到这种情况时, 我们可以按以下步骤恢复:

1. 从CDROM 启动系统, 进入维护模式, 然后执行下列命令:

  # mount /dev/hd4 /mnt

  # mount /dev/hd2 /mnt/usr

  # export PATH=/mnt/usr/sbin:/mnt/usr/bin

  # cd /mnt

  # restore -xvqf /../SPOT/usr/sys/inst.images/bos

  ./usr/ccs/lib/libc.a

  LIBPATH=/usr/ccs/lib:/usr/lib

  export LIBPATH

  ln -s /usr/ccs/lib/libc.a /mnt/usr/lib/libc.a

2. 检查文件系统:

  fsck /dev/hd1

  fsck /dev/hd2

  fsck /dev/hd3

  fsck /dev/hd4

  fsck /dev/hd9var

3.重新启动系统:

  reboot

## kill -l 查看系统的信号列表

linux中kill –l 或trap -l

## man \* signal 查看某个信号的使用手册

如：man 9 signal 查看9号信号的使用手册

## kill 发送信号

kill -信号 进程编号

或者

Kill -s 信号 进程编号

或者

Kill -n 信号 进程编号

“-信号”可使信号代码或者信号名称。Kill可后接多个进程编号，kill会把指定的信号都传送给这些进程。

如果省略“-信号”，则会默认传送SIGTERM（代码15）信号给进程。

在进程中，可另行定义TERM的处理方式或忽略改信号；执行kill –TERM,未必按照你的意思终止进程。因此，如果要强制终止，应改用

Kill -9 进程编号

或

Kill -KILL 进程编号

## killall 传递信号的对象是进程的名称（在linux系统中）

Killall -信号 进程名称

Killall使用的选项和kill相同；和kill不同的是，killall传送的信号的对象是进程的名称。

## svmon命令

svmon -P进程号;sleep 15 ; svmon -P进程号;

## trap 命令

### trap 的语法1：

**trap ‘触发命令’ 信号串行**

向shell注册信号，只要系统产生次特定的信号，trap指令就会触发，幷执行trap命令中的‘触发命令’

例：trap ‘echo “收到ABRT信号。。。。。。。”’ ABRT

本例向shell注册了一个ABRT的信号，只要系统出现ABRT信号，就会执行echo指令

向目前你shell发出ABRT信号的方法：

Kill -ABRT $$

其中$$代表目前shell进程的编号

注意：bash会对trap指令读取两次，一次是读取trap指令内容时，一次是执行触发指令时。因此最好使用单引号含括“触发指令”，以避免bash先对进行变量替换等动作，影响触发指令的正常执行

trap的语法2**：**

**trap - 信号串行**

如果trap的语法中没有触发命令，而是-或者空白，则可恢复信号串行原本的作用

例如：trap ‘-’ ABRT

本例恢复 ABRT信号的原本用法

### trap的语法3：

**trap ‘’ 信号串行**

忽略系统发出的“信号串行”

例如：想忽略使用者按下的CTRL+C组合键，可写成

trap ‘ ’ INT

trap的语法4**：**

**trap -p**

显示出trap想shell注册的信号列表

## 正则表达式

**. 代表任意字符**（要想比对.这个字符本身需加上转义字符\,即\.）

**^在行首**

**$在行尾**

**[…..]字符集合：**

字符集合常见用法：

[A-Z]一个大写字符

[a-z]一个小写字母

[0-9]一个数字

[^a-z]除了小写字符之外的一个字符

[^a-zA-Z]一个非英文字母的字符

[^a-zA-Z0-9]一个非英文字母且非数字的字符

^出现在[]括号里的第一个位置，代表非不是之意

**\*出现0个以上**

**\{\}指定字符符合的个数**

指定前面(左邻)字符的个数

例如：\{3,5\}，代表前面的字符有3-5个。[a-z]\{3,5\}代表小写字符组成的字符串，长度是3-5

**\(\)把对比符合的字符串暂时保存起来**

**例如:**H\(\)y表示保存H和y之间的3个字符，若要提取保存的字符串，可用位置参数，\1代表第一个保存的字符串，\2代表第二个，其他类推

## sed命令用法

语法：

sed ‘样式命令’ 文件

如果文件中某一行符合“样式”，就执行指定的sed命令，如删除（d）,取代(s)，打印(p)

这里的样式使用一对//含括，表示寻找之意，也可以指定数据行的范围，例如，1,6表示作用范围是由第1行到第6行；/AAA/,/DDD/表示作用范围是从含有AAA的数据行，到含有DDD的数据行，需要注意的是，sed并不会改变文件的内容,sed的工作方式是读取文件的内容，经流编辑之后，把结果显示到标准输出，因此，如果想存储sed的处理结果，得自行运用转向输出将结果存成其他文件

以下介绍下面的用法：

**sed语法1：删除某一段范围的数据行**

sed ‘1,4d’ datafile1

**sed语法2：把含有样式的数据行删掉**

sed ‘/La/d’ datafile2

把含有La的行删掉，剩余的显示出来。其中//代表搜索之意

sed ‘/[0-9]\(3\)/d‘ datafile3

把含有3位数字的行删除，剩余的显示出来

sed ‘/^$/d’ datafile5

删除datafile5的空白行，^代表开头，$代表尾部，两者之间没有任何字符，代表该行是一空白行

**sed语法3：把不含样式的数据行删掉**

sed ‘/La/!d’ datafile3

把不含La的行删掉，这里！是否定意思

**sed语法4：把含有样式的数据行显示出来**

sed ‘/La/p‘ datafile3

把含有La的行显示出来。其中p是sed命令，它会把目前的数据显示出来，但因为sed默认也会显示不符合的数据行，所以应该用一下指令

sed -n ‘ /La/p’ datafile5

选项n会抑制sed显示出其他资料行的默认操作，只显示符合样式的数据行

**sed语法5：取代**

sed –n ‘s/La/Oo/p‘ datafile3

这里的s是取代的意思，第对//中包含的字符串（La）是搜索的目标，第二对//包含的是置换的字符串（Oo）。它会把数据行中的字符串La换成Oo，

请注意，上面指令，只会替换第一个出现的La而已，如要全部替换，应再加上全局的命令g，如下所示：

sed ‘s/La/Ol/gp’ datafile5

取代的用法还有以下几个：

例1：sed -n ‘s/La//p’ datafile5

把每一行出现的La删掉（把La替换成空字符，就是删除）

例2：sed ‘s/^…//p‘ datafile5

把每一行开头的三个字符删掉

例3：sed ‘s/…$//p’ datafile5

**sed语法6：取符合样式的字符串**

sed –n ‘s/\(La\)/\1Oo/p’ datafile5

把找到的La 存起来，用\1取回来再使用。这个指令作用的结果：若数据行含有La字符串，则第一个出现的La会置换成LaOo，然后在显示这些含有La的数据行。

**sed语法7：找到符合样式的数据行后，在进行取代操作。**

例1：sed -n ‘/AAA/s/234/567/p’ datafile6

找到含有AAA的那一行之后，将234换成567

例2：sed –n ‘/AAA/,/DDD/s/B/567/p’ datafile3

将含有AAA到含有DDD的那几行，皆将其中的B换成567

例3：sed -n ‘2,4s/B/567/p’ datafile3

将2到4行，皆将其中的B换成567

在设计script时,我们经常会利用sed置换系统配置文件里的关键词,以开启或关闭某个

设置选项,若用bash来做,可能要花很大的力气,但用sed来做,通常只要一行指令就可以搞定

例:如果想开启vsftpd匿名登录功能

anonyftp1.sh

!# /bin/bash

#修改vsftpd的配置文件，开放匿名FTP服务。

VSFTPD\_conf=’/etc/vsftpd.conf’

TMP\_file=”tmp/tmp.$$”

将anonymous\_enable选项，设成YES,这样，vsftpd就会开启匿名FTP登入的功能。

sed ‘s/^.\*anonyous\_enable=.\*/anonymous\_enable=YES/’ $ VSFTPD\_conf > $TMP\_file

mv $TMP\_file $VSFTPD\_conf

由于在/etc/vsftpd.conf中，这一行的默认值可能是：

anonymous\_enable=no

或者被#注释掉了

#anonyous\_enable=no

因此，笔者决定采用样式^.\*anonymous\_enable=.\*来做比对，因为^表示由行首开始比对，紧接着0个以上的字符，如此，这就能包含#可能出现的情况，接着再用anonymous\_enable=.\*比对anonyous\_enable=NO或者anonyous\_enable=YES，即可。

找到符合的数据行之后，进而取代，然后把指定形态的字符串换成anonymous\_enable=YES,再将修改结果转向存储至临时盘中

上例，只能开启匿名登录，其实，做法还可以更有弹性一点，例如：执行时，加上选项on或off，即可切换；开启或关闭匿名登录

anonyftp.sh

#!/bin/bash

#修改vsftpd的配置文件，切换；‘开启/关闭’匿名FTP服务

if [ $# -ne 1 ];then

echo “usage:$0 on或$0 off”

exit 1

fi

OPT=$1

case “$OPT” in

[Oo][Nn]) CMD=’YES’;;

[Oo][Ff] ) CMD=’NO’;;

\*)

echo ‘选项格式错误！请用on或off来切换匿名登录的开关’

exit 1

;;

esac

VSFD\_conf=’/etc/vsftpd.conf’

TMP\_file=”/tmp/tmp.$$”

sed s‘^.\*anonymous\_enable=.\*/anonymous\_enable=$CMD/ $VSFTPD\_conf >$TMP\_file’

mv -f $TMP\_file $VSFTPD\_conf

使用方法：

要想关闭匿名登录，请执行：

anonyftp.sh off

要想打开匿名登录，请执行：

anonyftp.sh on

修改CSV文件名字段的顺序：

以下是数据文件datafile6.csv,想要把第2个字段和第4个字段调换

所在乡镇，学校名称，学校网址，校长姓名，学校电话，VOIP前三码，学校地址

做法如下：

Chcsv.sh

#!/bin/bash

TMPF=’/tmp/tmp.$$’

cat datafile6.csv |awk –F, ‘BEGIN{OFS=“，”}{print $1,$4,$3,$2,$5,$6,$7}’ >$TMPF

mv –f $TMPF datafile6.csv

## awk 命令用法

awk使用可以处理数据，产生格式化报表的语言，功能相当强大。Awk的工作方式是读取数据文件，将每一行数据视为一条记录，每笔记录以字段分隔符成若干字段，然后输出各个字段的值。

以下是执行ps auxw的输出片段:

USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND

root 1 0.0 0.0 164 92 7 Ss Apr09 0:01 init [5]

例如：仅用以下指令，就可以取得所有进程的pid

ps auxw |awk ‘{print $2}’

awk 对每一条记录，都会套用一个“样式{动作}”；如果改行符合样式，就执行指定的操作。样式。样式或操作之一，可以省略。如果只有样式，表示要显示符合样式的数据行，如果只有操作，表示对每一数据行都执行该项操作。

以下是awk的常用作用格式：

awk “样式” 文件：把符合样式的数据行显示出来

awk ‘操作’ 文件：对每一行都执行{}中的操作

awk ‘样式{操作}’ 文件：对符合样式的数据行，执行{}中的操作

### awk用法1：awk ‘/La/’ datafile3

显示包含La的数据行

### awk用法2：awk ‘{print $1,$2}’ datafile3

显示dataf3每一行的第1和第2个字段

### awk用法3：awk ‘/La/{print $1,$2}’ datafile3

将含有La关键词的数据行的第1及第2个字段显示出来

### aw的用法4：awk -F: ‘/^ola3/{print $2,$3}’ /etc/passwd

使用选项-F，指定：为分隔符，账号ols3的uid（第3个字段）及gid（第4个字段）显示出来

### awk的用法5：awk –F：‘BEGIN{OFS=”+++”}/^ola3/{print $1,$2,$3,$4,$5}’ /etc/passwd

以：为分隔字符，+++为输出字段分隔符，将账号ols3的第1-5栏显示出来

则执行结果

Ola3+++x+++1002+++1002+++

本例中，BEGIN{}区域指示awk一开始先做初始化的操作，即设定OFS=”+++”。变量OFS的作用是存储输出字段的分隔符。接着寻找ola3的账号行，找到后用print打印出第1-第5个字段，且彼此用+++隔开

实例应用：

取得网卡的IP：

Ifconfig|grep ‘inet\_addr:’|grep Bcast|awk ‘print $2’|awk -F:’{print $2}’

取得网络设备名称：

cat /proc/net/dev|awk ‘/eth.:|ppp.:|wlan.:/{print $1}’

本例中，-F：把分隔符设为：，而且采用多选一的样式/eth.:|ppp.:|wlan.:/

这个样式的意思是：设备名称可以是eth0、ppp1、wlan1这3个其中之一。一旦找到符合样式的字符串后，去掉：，取其中的第一个域值，因此，可能的答案是eth0或ppp1或wlan1

取得系统内存大小：

cat /proc/meminfo|awk ‘/MemTotal/{print $2}’

/proc/meminfo记载主机内存相关数据，其中MemTotal为内存大小，其样本值如下：

MemTotal：223128 kb

修改CSV文件文件各个字段的顺序：

以下是数据文件datafile6.csv

所在乡镇，学校名称，学校网址，校长姓名，学校电话，VOIP前三码，学校地址

chcsv.sh

#!/bin/bash

TMPF=’tmp/tmp.$$’

cat datafile6.csv|awk -F,‘BEGIN{OFS=”,”}{print $1,$4,$3,$2,$5,$6,$7}’ >$TMPF

mv –f $TMPF datafile6.csv

## ps（process status）命令

ps命令用来显示目前内存中的进程

ps的功能非常详尽，支持多种选项，其语法如下：

ps 选项

ps的语法可分为System V和BSD这两种风格

System V（以下简称UNIX）风格规定，选项可以分开或合并写在一起，但一定要以**-**开头

例如：ps -ax或ps –a –x

BSD风格，选项可以分开或写在一起，且可以不用**-**

例如：ps ax或ps a x

有些系统中的ps命令只支持一种风格，但对GNU/Linux而言，两种都支持。除此之外，Linux版ps是读取/proc这个虚拟目录中的文件，以获得目前核心及进程的信息，称为：/proc base的ps版本。

Linux版本的ps，除了上述两项风格之外，也支持GNU长选项。GNU长选项使用两个连续的**--**开头，例如：

ps – pid $$,这个命令使用长选项—pid来指定要显示的进程编号。$$表示要显示的是目前的bash shell的进程信息。

Linux版的ps是属于procps计划中的一个程序

以下是ps命令的常见用法：

**显示目前所有的进程：**

UNIX风格的用法：

ps –e

ps –ef

ps –eF

ps –ely

选项e表示要选出所有的进程，和-A同义

选项-f是控制显示格式，表示要列出完整的进程信息。

选项-F也是控制显示格式，表示要比-f列出更多额外的信息。

-l和-y均为控制显示格式的选项：-l表示使用长格式，-y表示不要显示旗标，这两个选项经常放在一起使用，且-y只能和-l合用

BSD风格的用法：

ps ax

ps axu

ps auxw

ps auxww

选项a表示要列出和终端有关的进程。如果和x合用表示要列出所有的进程

u是控制显示格式的选项，表示要以用户导向的角度来列出进程。

w是控制显示格式的修饰选项，表示要用较宽的行数来显示进程信息，超过宽度的信息会被截掉。如果使用两个w，则不限制显示宽度，信息会全部显示出来，不会被截断

**列出进程的树桩结构**

unix风格的用法：

ps –ejH

选项-e同前述

-j是控制显示格式的选项，表示要以工作控制(jobs control)的格式来显示

-H是控制显示格式的修饰选项，表示要列出进程的层次结构

BSD风格的用法：

ps axjf

选项ax合用表示要列出所有的进程

j表示控制显示格式的修饰选项，表示要列出进程的阶层结构

这两个指令，都可以看出进程之间的父子关系

**自定义输出字段**

ps支持自定义输出字段的功能，使用选项-o和o都可以

UNIX风格的用法：

ps –eo pid ,ni,user,args

BSD风格的用法：

ps axo pid,ni,user,args

上述自定义的4个字段：pid进程编号、ni是执行优先级、user使用者名称、args是执行进程和参数

ps 可供选用的输出字段很多，详情请参考man ps中的”STANDARD FORMAT SPECIFIERS”这一节说明

**列出某一进程的程序名称和参数**

UNIX风格的用法

ps –p 1 –o args

BSD风格的用法

ps p 1 o args

上述指令列出进程编号1的程序名称

**列出某一使用者的进程或用该用户权利执行的进程：**

ps –U root -u root u

这里的选项-U，指的是建立进程的使用者名称，称为real user ID（RUID）：-u指的是进程拥有的权限等同于改使用者，即effective user ID(EUID);最后那个u是指用户导向的格式输出进程信息。

其他的ps选项，可执行man ps 或ps –help查阅

接下来，说明ps列出的进程的信息中各个字段的意义

ps auxw的执行结果片断

标题栏各个字段的意义如下：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| USER | PID | %CPU | %MEM | VS2 | RSS | TTY | STAT | START | TIME | COMMAND |
| 执行身份 | 进程编号 | CUP资源使用率 | 内存使用率 | 进程占用虚拟内存的大小 | 进程在真实内存中占用的大小 | 控制改进程的终端 | 进程状态 | 开始执行时间 | 进程已经运行时间 | 执行命令 |

除了这些之外，还有一个常见的字段PPID.PPID是父进程编号，例如：ps –ef 的执行结果中就有这个字段。

ps 显示的进程信息，除了各字段的意义要了解之外，进程状态(STAT)也很重要，它使用数种代码不同的执行状态

以下是进程的执行状态列表：

|  |  |
| --- | --- |
| D | 不可中断的休眠（通常是进行输入输出）I/O的进程 |
| R | 可执行的 |
| S | 等待接受工作（休眠） |
| T | 被追踪或被停止 |
| W | 分页（只适用于Linux核心2.4以前的版本） |
| X | 死亡 |
| Z | 僵尸 |
| < | 高优先级的工作 |
| N | 低优先级的工作 |
| L | 已分页锁定在内存中 |
| s | 一个session leader |
| + | 在前台的进程群组中 |

## top命令

前述ps命令是用来观察目前进程的状态信息。如果想要持续观察，那么，可以使用top，top命令除了列出进程信息之外，也会显示主机的重要系统信息，如工作负载、进程统计、CPU使用率、内存和交换空间的耗用情况等

行1top，显示系统的负载情况，由左至右，代表：系统现在时间、已开机时间、目前登录系统的用户数、过去1分钟、5分钟、15分钟等待执行的进程平均数。

行2Tasks，显示进程的统计，由左至右，代表：进程总数、正在执行的进程数、休眠的进程数、暂停的进程数、僵尸进程数

行3Cpu(s),显示CPU使用率，由左至右，代表：ur指使用者执行的进程占用CPU的百分比，sy指核心系统内部占用CPU的百分比，ni指renice的进程占用

行4Mem，显示目前内存使用的情形，由左至右，代表：内存总量、已使用内存量、尚未使用内存量、缓冲区的大小。

行5Swap，显示目前置换空间使用的情形，由左至右，代表：置换空间总量、已使用的置换空间、尚未使用的置换空间、块取大小。

行6为空白行

行7位进程信息的各个字段，其意义如下表：

|  |  |
| --- | --- |
| PID | 进程编号 |
| USER | 拥有进程的使用者 |
| PR | 进程的优先级 |
| NI | 进程的nice值 |
| VIRT | 进程占用的虚拟内存大小 |
| RES | 进程常驻时占用内存的大小 |
| SHR | 可分享个其他进程的内存大小 |
| S | 进程状态 |
| %CPU | 进程占用CPU百分比 |
| %MEM | 进程占用真实内存大小 |
| TIME+ | 进程占用的CPU时间 |
| COMMAND | 程序名称和参数 |

top（UNIX中是topas）命令的操作方式是使用单一命令的字符，例如：按h或？可以显示出操作方法的说明，按q可结束top的执行

以下是操作说明：按h或？可展示出操作方法的说明。

离开top的方法：按q结束top，当然，按ctrl+c组合建也可以

## topas 命令

aix中是使用topas命令持续观察进程的状态和top一样按h或？可以显示出操作方法的说明，按**~**，可以以nmon的方式显示

## nice命令

nice语法：

nice -n nice值 程序

指定程序执行时的优先级为某一nice值，其中nice值的范围是-20~19，但只有root才能运用-20~0这个区段。若省略-n，不指定nice值，则预设使用nice值10来执行该程序。

## renice 命令

renice的语法

renice nice值 -p PID -q群组 -u使用者名称

调整某一个已执行进程的优先级

## PRI和NI

PRI也是进程优先级的一种表示法，和nice值（NI）不同的是，PRI永远以0或正数来表示，它和NI之间的关系是：

PRI=NI+20

## 工作控制

Bash在互动模式下，允许用户暂停某一进程，稍后再回复执行该进程。像这种赋予使用使用者控制进程的机制，称为工作控制（job control）

以实例说明工作控制

Sleep1.sh

#!/bin/bash

while true

do

sleep 20

done

先执行sleep1.sh，然后按ctrl+z,，组合键暂停该进程，接着执行bg命令，把该进程丢到背景中取执行（这些操作和接着执行sleep1.sh & 效果是一样的），执行jobs命令，列出目前的工作（若执行jobs –l，可额外显示工作中的编号），然后执行fg命令，把进程拉回前台，最后按ctrl+c组合键，终端sleep1.sh的执行，结果中

其，[1]代表工作编号为1，+号代表它是最近的一个工作

使用jobs –l列出工作时，可发现第一个工作记录为[1]-,第二个工作标记为[2]+,其中-代表前一个工作的意思。

使用%%或%+可调用现在的工作，使用%-则可以调用前一个工作

除了%%，%+，%-，的用法之外，还可以使用以下方法调用某一工作编号的进程，将他置于前台执行%n，等同与fg %n(aix中是用fg %n，不能用%n)

若将工作编号1、2、6置于背景执行，可写成

%1 &

%2 &

%6 &

执行% n & ，等同于bg %n（aix中只只能用bg %n）

另外，还可以用字符串来表示或搜寻工作的名称，%?sl,表示要调用程序名称中含有sl的工作，？代表搜寻之意，%./sl表示要找出程序开头是./sl的工作

这里，整理一下前述的工作符号

|  |  |
| --- | --- |
| 符号 | 意义 |
| + | 代表当前工作 |
| - | 代表前一个工作 |
| %%,%+ | 调出现在的工作 |
| %- | 调出前一工作 |
| %n | 调用工作编号n的进程，置于前台执行 |
| %n & | 调用工作编号n的进程，置于背景执行 |
| %?sl | 调用含有sl字符串的工作 |
| %./sl | 调用文件名开头是./sl的工作 |

请注意，若键入exit准备离开bash shell时还有处于暂停状态的工作，则bash会显示提示信息：There are stopped jobs (目前还有暂停的工作)，此时终端接口并不会消失。但若是在键入一次exit，暂停的工作会被终止，终端程序随即结束

### 常用的工作控制命令&

将指令丢到背景执行

#### 用途1：将单一指令丢到背景执行

find ./ -name ‘\*.doc’ -print &

#### 用途2：把多个指令丢到背景执行

(find / -name ‘\*.txt’ > check.log; find / -name ‘\*.conf’ >>check.log) &

这里使用（）开启一个子shell执行find等复合指令，然后用&全丢到背景去执行

### bg

将暂停工作丢到背景执行

#### 用法1：bg

将现在的工作丢到背景

#### 用法2：bg n或bg %n

把工作编号n的工作丢到背景

#### 用法3：bg %?sl

把程序名称含有sl字符串的工作丢到背景

使用ctrl+z组合键暂停进程，然后配合bg命令把工作丢到背景，和执行“指令 &”的效果是一样的；&就是丢到背景的意思

上述 n ，%n，%?sl等称为“工作规格”

### fg

将暂停的工作拉回前台执行

fg 工作规格

将符合工作规格的工作，丢到前台执行

#### 用法1：fg

将目前的工作丢到前台

#### 用法2：fg %2

将编号2的工作丢到前台

### jobs

列出工作

jobs或jobs –l

### disdown

移除工作

disdown %1

移除编号1的工作

### kill

传送信号给进程或工作

传送信号给进程，请参考前面内容，至于传送信号给工作，用法差不多，只不过kill后接的工作编号，而不是PID

例：kill -TERM %2

### wait

等待子进程结束，传回最后一个命令的执行状态

例如：wait 1877,或wait %2

等待进程编号1877，或编号2的工作执行结束

### suspend

暂停目前的shell

用法1：suspend

暂停一个非login的shell

用法2：suspend –f

暂停目前的login shell

### 进程替换

进程替换可让我们把标准输出，一次倒给许多进程作为输入；或者由多个进程取得其输出，然后到给某个程序处理

进程替换的方法主要有两个：

**命令 < (指令行) ,<和（之间没有空格**

这是通过设备文件/dev/fd/n或缓存文件，把进程的输出倒给<左方的命令来读取

**> (指令行)，>和（之间没有空格**

这是通过设备文件/dev/fd/n或缓存盘，把数据提供给()中的进程去读取

例1：

echo > (true)

echo < (true)

借由例1，观察此进程替换是借用哪一个设备文件来做输出入的中介

例2：一次读取一行数据放在数组中

read –a arr < <(echo 123 45 97 101 88)

选项a的意思是指定arr为一个数组变量，这样会把123放入数组arr的一个元素，45放入第二个，其他类推。取数组元素值，如下：echo ${arr[2]}

可以把()中的echo改换成其他命令，往后，会发现这种用法非常有效果。

例3：

dirA=”usr/bin”

dirB=”usr/local/bin”

diff <(ls $dirA) <(ls $dirB)

比较$dirA和$dirB这两个目录内容的差异

例4：

tar cvf <(gzip –c > test.tgz) /root/tmp

这里，tar把/root/tmp目录打包，然后，将结果写入/dev/fd/n(n表某一正整数)，接着（）中的进程会由/dev/fd/中读取数据，然后交给gzip做压缩，幷转向输出存成test.tgz,

其实，它的效果和以下指令一样

tar cvfz test.tgz /root/tmp

## 历史指令

在shell互动模式下，执行过的指令会被记录下来，利用上下箭头键，可以往后往前查看这些指令，而且离开shell之后，bash会将这些指令储存在一个文件中，称为历史脚本文件，下次再进入shell时，还可以浏览或调用这些旧指令

这个历史脚文件的路径文件名，由bash的内容置变量HISTFILE定义，以下指令可显示这个变量的内容

echo $HISTFILE

通常，此变量的内容为：

~/.Bash\_history

换言之，历史脚本文件位于每一个账号的家目录中，以root而言，该文件位于/root/.Bash\_history;以osl3这个账号来说，历史脚本文件是位于/home/osl3/.Bash\_hostory.这个文件必须是普通的文本文件才行，如果该文件是目录,Bash就无法存入历史指令了

有些cracker入侵主机后，为了避免留下记录，会故意把Bash\_history改成目录。不过，这种做法很容易被有经验的管理员识破

如果想要关闭历史指令的功能，可使用Bash的内置命令set

set +o history

这里的+o是关闭选项的意思，-o则是启用的意思

每一个shell互动环境结束后，究竟有多少个指令会被记录到历史脚本文件中呢？这由Bash的内置变量HISTSIZE决定，通常其默认值是500，而历史脚本文件最多能存入的指令数量，则由内置变量HISTFILESIZE决定，通常其值也是500。也就是说，Bash\_history最多存500个指令

为了精简历史脚本文件，可利用HISTIGNORE控制某些指令不予存入，如常见的ls、cd、ps指令，重复的指令也应只存一个就好

用例：

export HISTIGNORE=ls:ps:cd:t\*:\&

这里设定ls、ps、cd三道命令不存入历史脚本文件；t\*代表凡是t开头的命令，都不存入历史脚本文件；&代表最后一个历史指令，这表示，如果一直键入重复的指令，则只存入一个

另外,内置变量HISTCONTROL也可以控制指令是否存入历史脚本文件

HISTCONTROL有三个可能的设置值，各设置值的意义如下：

Ignorespace:凡是开头有空格符的都不存入历史脚本文件

Ignoredups:连续重复的指令，只存一个

Ignoreboth:结合前两者的功能

常用的做法是使用第三个：

export HISTCONTROL=ignoreboth

自Bash3.1版开始，新增了一个内置变量HISTTIMEFORMAT,如果设定这个变量为某一个时间格式，则执行history命令时，会在每个命令前显示日期时间，范例如下：

HISTTIMEFORMAT=”%c:”

Bash有两个内置命令和历史指令有关，以下分别说明

**history命令**

history可以列出历史指令。这里假设HISTFILE定义的历史脚本文件是Bash\_history.

以下是history命令的几个用法：

history

不加任何选项的话，history会列出全部的使用过的指令。列出时，在每个指令前会有编号

history n

列出最后几个指令

history –d n

删除第n条历史指令

history c

清空Bash\_history文件的内容

history –r 文件

将文件内容附加在Bash\_history文件的后面

history –w

将目前的历史记录写入Bash\_history文件

**fc命令**

fc命令可以列出登入主机后最近执行过的指令，关于这项功能，请参考5.1内置命令中fc内置命令的说明。除了列出命令的功能，fc还可以调出过去的指令来编辑，然后在执行编辑过的命令

fc ssh

这里，fc会调出最近一个以ssh开头的历史指令，幷使用默认的编辑器来编辑该指令

编辑内容如下：

ssh [osl3@example.edu.cn](mailto:osl3@example.edu.cn)

调用默认编辑器的规则是：FCEDIT变量定义的编辑器优先使用，如果FCEDIT没有定义，则使用EDITOR变量定的编辑器，如果都没有定义，则使用vi

Fc也有搜寻的功能，范例如下

fc –s ls

这里，fc会搜寻最近一个以ls开头的命令，幷执行该命令

**历史指令扩展**

历史指令扩展有3个功能：

调用历史指令，更方便快速

在当前的命令行中，调用先前的指令，幷安插参数

快速修正之前错误的命令

在介绍用法之前，有个重要的选项要注意一下

在Bash中，选项histverify和历史指令扩展是否立即执行有关（该选项默认是关闭的）

如果histverify设为off（即关闭）则历史指令的扩展完成后，该指令立即执行；但若histverify设为on，则历史指令扩展后，会先把扩展结果显示在命令行以供编辑，幷不会立即执行，可以利用这个选项功能，把某一历史指令调出来修改，之后，再予以执行

打开histverify的方法如下：

shopt –s histverify

设置完之后，可以执行shopt观看选项开关的情况

接下来，介绍历史指令扩展的用法:

**!n：调用编号n的历史指令**

假设history命令执行的结果部分内容如下：

72 cd /var

73 ls /tmp

74 cd /home/ols3

75 ps auxw

!73,bash会扩展!73成编号73的历史指令，也就是执行ls /tmp

**!-n：调用前往倒数的第n个历史指令**

**!!：执行前一历史指令和!-1的作用相同**

**!字符串：执行最近一个以“字符串“开头的历史指令**

**!?字符串?：执行最近一个含有“字符串”的历史指令**

**^字符串1^字符串2^：将前一个历史指令中的“字符串1”换成“字符串2”，然后执行替换后的指令**

**!#：代表在命令行中，到目前为止所键入的字符串**

例：date;!#

这里的!#会被date;取代，执行指令变成

date;date

等于执行两次date

**!!:$：取得前一个命令的最后一个参数**

假设前一个命令是ls /tmp/home,则ls的最后一个参数为/home

cd !!:$,会扩展成cd /home,然后执行指令，即切换成/home目录

**!ls:1找出最近一个以ls开头的指令的第一个参数**

cd !ls:l会扩展成cd /tmp,然后执行该指令，即切换成/tmp目录

在本例用法中，ls可换成其他字符串；1也可以换成其他数字n，代表第n个参数。若n是0，则代表该命令本身。

另外，第1个参数也可以可用^来表示

cd !ls:^

**!ls:\*找出最近一个以ls开头的指令的所有参数**

## 取得目录内容

利用ls取得目录内容，这里假设要查看的目录是~/tmp.设定变量DestDir如下：

DestDir=~/tmp

注意：~不可以放在单引号或双引号中，否则会失去家目录的替换效果

利用命令替换中提到命令替换功能，可取得DestDir目录的内容，方法如下：

DirCnt=$(ls $DestDir)

此时变量DirCnt的内容即为DestDir目录内容的文件列表（但不包含隐藏文件），运用for循环，便可逐一处理列表中的每一个文件，如下所示：

for f in $DirCnt

do

echo $f #这里可以置换成其他处理操作

done

如果想要包括隐藏文件，可改用一下写法：

DirCnt=$(ls –A $DestDir)

ls 的选项A和a的作用差不多，两者的差异是-a会把 **.** 及 **..** 也列入

可以利用一下指令，大量建立扩展名是.txt的文件。declare –i i=0;while(true);do ((i++));touch $i.txt;done

如果要停止建立文件，则可以按[ctrl+c]组合键予以终止

上述方法，在文件数量不多时，变量DirCnt占用的内存的情况不严重，此方法尚可适用，但如果目录中的文件非常多时，不但执行速度慢，耗用大量的内存，而且为了展开\*.txt的通配符，可能会出现“Bash：/bin/ls;Argument list too long”参数行过长的错误信息，而无法正常运作。这时，可运用

## 参数长度限制

ls 、cp、rm等命令，有时会因参数太长而报“argument list too long”，不同的操作系统，参数行的长度限制可能会有所不同，可执行getconf ARG\_MAX或linux/limits.h(Linux)、sys/syslimits.h(BSD)中ARG\_MAX的定义值

## 判断文件或目录是否存在

If [ -f 文件名]；then

处理

fi

if [ -d 目录名 ]；then

处理

fi

## 判断文件的类型

以下使用“[判断式] && 指令”的隐形if-then语法来进行判断

[ -b /dev/sda ] && echo “块设备文件存在”

[ -c /dev/tty1 ] && echo “字符设备文件存在”

[ -S /var/run/mysqld/mysqld.sock ] && echo “socket文件存在”

[ -p $f ] && echo “管道文件存在”

[ -L /etc/localtime ] && echo “符号链接文存在”

如果不管文件的类型，只求判断文件是否存在，可使用-e判断式

## 判断文件的权限

[ -r /tmp/test.txt ] && echo “文件可读取”

[ -w /tmp/test.txt ] && echo “文件可写”

[ -x /tmp/test.txt ] && echo “文件可执行”

[ -s /tmp/test.txt ] && echo “文件大小不为空”

需要特别注意的是，某些系统中，虽然文件不具读取、写入的权限，但如果执行script的身份是root，由于root具有最高权限，所以任可以判断得到：改文件是可读取或写入的

## 建立文件

### 建立一般文件

使用转向输出

>test.txt

配合空命令：，也可以

:>test.txt

使用空命令：的理由是，可增加script的可移植性，因为在某些shell环境下，使用”>文件”来开启新文件的方式，可能会造成语法错误。

### 建立隐藏文件

:> .test.txt

使用touch命令也可以

touch test.txt

touch 原本用来修改文件时间，但是如果test.txt不存在，则会开启一个新的文件

以上方法都可以建立一个空文件。它的文件权限，会受到umask值的影响，如果umask是0222，那么新文件的权限即为0666 – 0022 = 0644

以下方法，在建立文件时，即指定初始化的内容

echo ‘Hello Bash’ > test.txt

以下方法，则是利用ln命令，建立另一个文件的副本

ln test.txt test2.txt

此时，新增一个test.txt的文件复本称为test2.txt.test.txt和test.txt2的连接计数均为2，表示这个文件有两个复本

像这种使用ln不加任何选项建立的文件，称为“硬式连接”，这也是一种普通文件。这种文件有一个特性，当修改一个文件的内容时，另一个文件的内容也会跟着做相同的改变，但如果把其中一个文件删除，另一个文件任然会存在，只是会把文件计数减1

## 建立目录

mkdir testdir

如果目录不存在，则执行上述指令后，即可建立一个空目录testdir,新建目录的权限，则受到umask值的影响。如果umask为0022，则新目录的权限为0777 – 0022 = 0755

也可以在建立目录时直接指定目录的权限值，这样就可以不收umask值的影响。做法如下：

mkdir –m 711 testdir

这里使用-m指定testdir目录建立时的权限值为0711

## 建立区块设备文件

所谓区块设备文件，就是用来存储数据的媒体，或可供读取的接口设备。前者如硬盘的分区/dev/hda1,后者如CDROM等外接设备/dev/cdrom或/dev/scd0

以下指令可以建立一个和/dev/hda1相同属性的区块设备文件

mknod myhda1 b 3 3 ,这里使用的指令mknod，可以建立特殊的文件，如区块设备文件、字符设备文件、FIFO文件，它的基本语法是：

mknod 选项 文件名 文件形态 主设备代码（MAJOR）、次设备代码（MINOR）

其中文件的形态由以下代码组成

|  |  |
| --- | --- |
| 代码 | 文件形态 |
| b | 区块设备文件 |
| c | 字符设备文件 |
| P | FIFO文件 |

选项通常是“-m 文件权限值” 用来指定建立的设备文件的文件属性，至于主要设备代码，称为MAJOR，次要设备代码称为MINJOR,则是linux核心用来辨识接口设备的一种方式。由于linux核心把任何东西，由于linux核心把任何东西，包括硬件设备，都设为文件，因此，必须有一套识别的机制。其中MAJOR是指硬件的分类，而MINJOR则是该硬件分类中在细分其特殊的属性。如IDE硬盘分属MAJOR为3的类别，而第一个IDE分区/dev/hda1,则使用MINOR 1来和其他分区做区别，记为（3,1），其他分区依次类推，如/dev/hda3,记为（3,3）

，只要MAJOR和MINOR的配对值相同，即视为相同属性的设备，依次，前述指令

mknod myhda1 b 3 1意思就很清楚了：建立一个区块设备文件名为myhda1 ，文件的形态是区块设备文件（b），主要设备代码MAJOR为3，次要设备代码MINJOR为1。

也可以在建立设备文件时，直接指定文件权限，例如：

mknod –m 666 myhda1 b 3 1 ,那么，不同的设备文件，其MAJOR和MINOR各是多少呢？linux核心原始代码套件中，在Documentation/dev/devices.txt有十分清楚的定义。如果只要是快速查询到MAJOR而已，则可参考/usr/include/linux/major.h

## 建立字符设备文件

所谓字符设备文件是指可以读取序行字符的周边，如键盘、串行端口、鼠标、终端等。

以下指令，可以建立一个和/dev/null相同属性的字符设备文件：

mknod -m 666 nullfile c 1 3

这样一来，nullfile便和/dev/null具有相同的属性。也就是说，凡是丢给nullfile的内容，均会立即消失

可以用nullfile来清空其他文件的内容，这在管理巨大文件记录是，特别好用。

另外，顺便一提，/dev/null这个文件对系统的正常运行十分重要，如果这个文件不慎丢失，系统可能发生一些异常现象，从而无法正常工作，如无法正常登入主机。在了解上述建立nullfile的方法后，如果要重建/dev/null ，就十分简单了，方法如下：

mknod –m 666 /dev/null 1 3

请特别牢记这个建立/dev/null文件的方法！有时候，你会发现；重建这个文件，可救回主机一命

## /dev/null文件的作用

### 可以用来清空大的文件

cp /dev/null 文件名

### /dev/null这个文件对系统的正常运行十分重要

如果这个文件不慎丢失，系统可能发生一些异常现象，从而无法正常工作，如无法正常登入主机。在了解上述建立nullfile的方法后（mknod -m 666 nullfile c 1 3 ），如果要重建/dev/null ，就十分简单了，方法如下：

mknod –m 666 /dev/null 1 3

## 建立FIFO文件

FIFO文件是一种命名管道，具有先进先出读出的序行特性

以下指令可建立一个FIFO文件

mknod myfifo p

特别注意，在建立FIFO文件时，不必（也不可以）指定MAJOR和MINOR,那么这个文件怎么用？在一个shell环境中使用echo命令，把一个字符串输出到myfifo，称为First in（先进入者），而echo则会暂停下来，等待其他进程读取

在另一个shell环境中，使用cat命令读取来自myfifo的输出，称为First Out(先取出)，而之前的echo之指令则因数据已输出完毕而执行完毕。

如上所述，FIFO文件这种命令命名管道的特性，在设计script时，可让两个不同的进程通过FIFO文件，进行数据传递，从而达到内部进程通信的目的。

这里补充说明一下，在linux中，利用mkfifo程序，也可以建立FIFO文件。

mkfifo myfifo

这个效果和mknod建立的FIFO文件是完全一样的。

## 建立socket文件

这里使用Red Hat Linux的mksock来建立socket文件，范例如下：

mksock mysocket

mksock属于MAKDEV套件的一部分。MAKDEV的原始代码程序包中，含有mksock.c.这个程序可单独编译，方法如下：

make mysock

接着将mksock复制放入/usr/local/sbin中，方便以后调用。

另外，在/dev中有一个重要的/dev/log文件，如果不慎遗失或误删了，可以用mksock重建

mksock /dev/log

chmod 666 /dev/log

## 建立符号链接文件

符号链接文件是一种快捷方式的概念，可指向其他文件建立链接，而且可跨越不同的系统分区，是管理目录、备份系统、建立工作环境时十分重要的应用技巧

建立符号链接的方法如下：

Ln 选项 来源文件 目的文件

### 建立新的符号链接文件

Ln -s test.txt tfile

建立一个新的符号链接文件tfile，指向test.txt

### 删除原符号链接文件，重建新的符号链接文件

Ln –af test.txt tfile

如果tfile原已存在，则先予以删除，再建立一个新文件指向test.txt

### 建立指向目录的链接文件

Ln -s /root/tmp mytmp

一旦和目录建立链接之后，使用cd mytmp,就等于进入/root/tmp目录是一样的

## 取得文件的属性

在Linux中，每一文件都有13个属性，分别是：

* 设备编号
* Inode编号
* 文件模式
* 链接文件数量
* 文件拥有这编号
* 设备标识符
* 文件隶属群组编号
* 设备标识符
* 文件大小
* 取文件的时间：atime
* 修改文件的时间：mtime
* 改变文件模式的时间：ctime
* 区块大小
* 配置区块数

取得文件属性，最方便的方法是利用stat这个程序。stat可以用来取得两种信息：一种是文件系统；一种是文件。

### stat的基本用法

#### 取得文件系统的信息

stat –f /dev/hda1

其中，选项-f表示要取用的是文件系统（分区），而不是文件

#### 取得文件属性

stat test.sh

每一个文件都有3种时间（称为时间戳timestamps），对这3种时间，一般人经常混淆不清，这里做个说明：

* Access time（atime）：是指取用文件的时间，所谓取用，常见的操作如:使用编辑器查看文件的内容、使用cat命令显示其内容、使用cp命令把该文件复制成其他文件，或者在这个文件上运用grep 、sed、more、less、tail、head等命令，凡是读取而不修改文件的操作，均会改变文件的Access time
* Modify time（mtime）:是指修改文件内容的时间，只要文件内容有改动（如使用转向或追加转向）或存盘的操作，就会改变文件的Modify time 。平常我们使用ls –l指令查看文件，看到的就是Modify time。
* Change time（ctiime）：是指文件属性或文件位置改动的时间

要特别注意几点：

* 如果修改文件（使用编辑器存盘或使用>、>>转向操作），则Modify time和Change time会同步成写入的时间，但Access time不变。
* 如果执行touch文件，则3种时间都会变成一样
* 使用ln –s做文件的软式链接，会改变文件的取用时间Access time
* 执行ls –al查看一般文件，不会改这三种时间，但是如果这个文件是符号链接文件则会更改其取用时间Access time

了解上述基本概念后，接下来就可以把stat运用在是script设计上了，Stat支持许多取得文件属性的格式代码，其基本语法如下：

##### stat –c 格式代码 文件

例如，想要取得文件的大小，可执行以下指令：

stat –c %s test.sh

利用之前的提到的命令替换的技巧，就可以把文件的大小放入变量中备用

FileSize=$(stat –c %s test.sh)

下面列出stat支持的格式代码，如果，想取用某一个文件属性，只要按上面的范例模仿即可

* 适用于文件的格式代码：

|  |  |
| --- | --- |
| 格式代码 | 准备取用的文件属性 |
| %A | 用文件权限码来表示，如-rwx-r-r— |
| %a | 用八进制数字表示文件权限 |
| %b | 占用的区块数量（blocks） |
| %B | 用%b计算区块数量时，每一块的的大小，预设是512bytes |
| %D | 用16进制表示的设备编号 |
| %d | 用10进制表示的设备编号 |
| %F | 文件形态(例如：一般文件、目录、区块、字符设备文件、socket文件、FIFO、符号链接文件) |
| %f | Raw mode以16进制表示 |
| %G | 文件拥有的组名 |
| %g | 文件拥有的群组编号（即GID） |
| %h | 硬件连接的文件的链接数量 |
| %i | Inode编号 |
| %N | 将符号链接文件和其指向的的文件名，用引号包含，例如‘1.sh’-> ‘test.sh’ |
| %n | 文件名 |
| %o | IO区块的的大小，预设是4096bytes |
| %s | 文件的大小 |
| %T | 16进制表示Minor次要设备代码 |
| %t | 16进制表示Major只要设备代码 |
| %U | 文件拥有者的使用者名称（如root） |
| %u | 文件拥有者的使用者编号 |
| %X | 取用时间，以1990年1月1日至取用时的秒数来表示 |
| %x | 取用时间 |
| %Y | 修改时间，以1990年 1月1日至修改时的秒数来表示 |
| %y | 修改时间 |
| %Z | 属性修改时间，以19900年1月1日至属性改动时的秒数来表示 |
| %z | 属性改动时间 |

* 适用于文件系统（即分区）的格式代码

|  |  |
| --- | --- |
| 格式代码 | 想要取用的文件属性 |
| %a | 一般使用者可自由运用的区块数 |
| %b | 全部区块数 |
| %c | 全部的文件节点数 |
| %d | 可取用的文件节点数 |
| %f | 可取用的区块数 |
| %i | 用16进制数表示的文件系统代码 |
| %d | 最大的文件名长度 |
| %n | 文件名 |
| %s | 最佳传输的文件大小 |
| %T | 用易于了解的方式，表示文件系统的格式 |
| %t | 用16进制数表示文件系统的格式 |

除了上述文件的属性之外，在linux ext2/ext3 文件系统中，文件还可以拥有一族特殊的属性。这些特殊的属性，可使用chattr和lsattr进行管理。chattr用来改变文件的属性，lsattr用来查看文件的属性。

### Chattr的使用方法：

增加特殊属性

chattr + 属性代码 文件

使用+表示各个属性可个别的加上，不影响文件的原有的特殊属性。

设定固定属性

chattr = 属性代码 文件

使用=，表示直接指定文件的属性，原已有存在但未被指定的属性会消失

删除特殊属性

chattr –属性代码 文件

使用-，表示要移除某些属性

可用的属性代码有：AacDdijsSu.q。其中常用的属性如下：

|  |  |  |
| --- | --- | --- |
| 代码 | 属性作用 | 操作者的身份 |
| A | 文件取用时间不改变 | 所有拥有者 |
| a | 只能使用转向附加的方式开启文件进行写入 | 只有root可以 |
| i | 不能修改文件 | 只有root可以 |
| D | 该文件不需要用dump做备份 | 所有使用者 |

设定取用文件后，取用时间不改变

chattr +A test.sh

使用chattr改变文件的属性后，可以用lsattr来查看修改的结果

Lsattr的用法：

Lsattr 文件

一旦设定了这项属性后，不管是使用编辑器查看文件还是使用cat显示文件的内容，改文件的取用时间不会改变

## diff对比文件的差异

diff是一种以行和行进行对比文件的差异性的工具，其基本用法如下：

diff 选项 文件1 文件2

如果对比的两个文件内容完全一样，则$?传回0，如果不同，则传回1

利用diff，我们就可以来撰写一个判断两个文件内容是否相同的工具了

#!/bin/bash

shopt –s –o nounset

DIFF=”/usr/bin/diff”

[ $# -ne 2 ] && echo “使用法：$0文件1 文件2” && exit 2

File1=$1

File2=$2

$DIFF $File1 $File2 &> /dev/null

exit $?

diff的其他用法：

程序的改动部分，我们称为patch文件，或补丁

下面以diff来示范，如何产生程序代码的patch文件

产生patch的基本语法如下：

diff 选项 原文件 新文件 > patch 文件

选项说明如下：

-u 表示采用称为“unified context”的diff格式，这种格式产生比较小而紧致的差异文件，是目前最受欢迎的格式。

# Shell脚本2

## [ulimit 命令详解](http://www.cnblogs.com/zengkefu/p/5649407.html)

Linux对于每个用户，系统限制其最大进程数。为提高性能，可以根据设备资源情况，设置各linux 用户的最大进程数

可以用ulimit -a 来显示当前的各种用户进程限制。

下面我把某linux用户的最大进程数设为10000个：

     ulimit -u 10240

     对于需要做许多 socket 连接并使它们处于打开状态的 Java 应用程序而言，

     最好通过使用 ulimit -n xx 修改每个进程可打开的文件数，缺省值是 1024。

     ulimit -n 4096 将每个进程可以打开的文件数目加大到4096，缺省为1024

     其他建议设置成无限制（unlimited）的一些重要设置是：

     数据段长度：ulimit -d unlimited

     最大内存大小：ulimit -m unlimited

     堆栈大小：ulimit -s unlimited

     CPU 时间：ulimit -t unlimited

     虚拟内存：ulimit -v unlimited

     暂时地，适用于通过 ulimit 命令登录 shell 会话期间。

     永久地，通过将一个相应的 ulimit 语句添加到由登录 shell 读取的文件中， 即特定于 shell 的用户资源文件，如：

1)、解除 Linux 系统的最大进程数和最大文件打开数限制：

        vi /etc/security/limits.conf

        # 添加如下的行

        \* soft noproc 11000

        \* hard noproc 11000

        \* soft nofile 4100

        \* hard nofile 4100

       说明：\* 代表针对所有用户，noproc 是代表最大进程数，nofile 是代表最大文件打开数

2)、让 SSH 接受 Login 程式的登入，方便在 ssh 客户端查看 ulimit -a 资源限制：

        a、vi /etc/ssh/sshd\_config

             把 UserLogin 的值改为 yes，并把 # 注释去掉

        b、重启 sshd 服务：

              /etc/init.d/sshd restart

3)、修改所有 linux 用户的环境变量文件：

    vi /etc/profile

    ulimit -u 10000

    ulimit -n 4096

    ulimit -d unlimited

    ulimit -m unlimited

    ulimit -s unlimited

    ulimit -t unlimited

    ulimit -v unlimited

 保存后运行#source /etc/profile 使其生效

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

有时候在程序里面需要打开多个文件，进行分析，系统一般默认数量是1024，（用ulimit -a可以看到）对于正常使用是够了，但是对于程序来讲，就太少了。

修改2个文件。

1./etc/security/limits.conf

vi /etc/security/limits.conf

加上：

\* soft nofile 8192

\* hard nofile 20480

2./etc/pam.d/login

session required /lib/security/pam\_limits.so

/\*\*\*\*\*\*\*\*\*\*

另外确保/etc/pam.d/system-auth文件有下面内容

session required /lib/security/$ISA/pam\_limits.so

这一行确保系统会执行这个限制。

/\*\*\*\*\*\*\*\*\*\*\*

3.一般用户的.bash\_profile

#ulimit -n 1024

重新登陆ok

ulimit 的作用

  =======================

ulimit：显示（或设置）用户可以使用的资源的限制（limit），这限制分为软限制（当前限制）和硬限制（上限），其中硬限制是软限制的上限值，应用程序在运行过程中使用的系统资源不超过相应的软限制，任何的超越都导致进程的终止。

参数 描述

ulimited 不限制用户可以使用的资源，但本设置对可打开的最大文件数（max open files）

和可同时运行的最大进程数（max user processes）无效

-a 列出所有当前资源极限

-c 设置core文件的最大值.单位:blocks

-d 设置一个进程的数据段的最大值.单位:kbytes

-f Shell 创建文件的文件大小的最大值，单位：blocks

-h 指定设置某个给定资源的硬极限。如果用户拥有 root 用户权限，可以增大硬极限。任何用户均可减少硬极限

-l 可以锁住的物理内存的最大值

-m 可以使用的常驻内存的最大值,单位：kbytes

-n 每个进程可以同时打开的最大文件数

-p 设置管道的最大值，单位为block，1block=512bytes

-s 指定堆栈的最大值：单位：kbytes

-S 指定为给定的资源设置软极限。软极限可增大到硬极限的值。如果 -H 和 -S 标志均未指定，极限适用于以上二者

-t 指定每个进程所使用的秒数,单位：seconds

-u 可以运行的最大并发进程数

-v Shell可使用的最大的虚拟内存，单位：kbytes

-x

范例1：

[root@localhost proc]# ulimit -a

core file size (blocks, -c) 100

data seg size (kbytes, -d) unlimited

file size (blocks, -f) unlimited

pending signals (-i) 2047

max locked memory (kbytes, -l) 32

max memory size (kbytes, -m) unlimited

open files (-n) 1024

pipe size (512 bytes, -p) 8

POSIX message queues (bytes, -q) 819200

stack size (kbytes, -s) 8192

cpu time (seconds, -t) unlimited

max user processes (-u) 2047

virtual memory (kbytes, -v) unlimited

file locks (-x) unlimited

[root@localhost proc]#

输出的每一行由资源名字、（单位，ulimit命令的参数）、软限制组成。详细解释：

参数 描述

core file size core文件的最大值为100 blocks，

data seg size 进程的数据段可以任意大

file size 文件可以任意大

pending signals 最多有2047个待处理的信号

max locked memory 一个任务锁住的物理内存的最大值为32kB

max memory size 一个任务的常驻物理内存的最大值

open files 一个任务最多可以同时打开1024的文件

pipe size 管道的最大空间为4096字节

POSIX message queues POSIX的消息队列的最大值为819200字节

stack size 进程的栈的最大值为8192字节

cpu time 进程使用的CPU时间

max user processes 当前用户同时打开的进程(包括线程)的最大个数为2047

virtual memory 没有限制进程的最大地址空间

file locks 所能锁住的文件的最大个数没有限制

范例2：通过ulimit命令来限制文件的大小，从而导致拷贝命令的失败

[root@localhost]ls temp.txt

ls: temp.txt: 没有那个文件或目录

[root@localhost]ulimit -f 1 #设置创建文件的最大块(一块=512字节)

[root@localhost]cat a.c > temp.txt

文件大小超出限制

文件a.c的大小是5002字节,而我们设定的创建文件的大小是512字节x1块=512字节

1、修改用戶進程可打開文件數限制

在Linux平台上，無論編寫客戶端程序還是服務端程序，在進行高並發TCP連接處理時，最高的並發數量都要受到系統對用戶單一進程同時可打開文件數量的限制(這是因為系統為每個TCP連接都要創建一個socket句柄，每個socket句柄同時也是一個文件句柄)。可使用ulimit命令查看系統允許當前用戶進程打開的文件數限制：

[speng@as4 ~]$ ulimit -n

1024

這表示當前用戶的每個進程最多允許同時打開1024個文件，這1024個文件中還得除去每個進程必然打開的標準輸入，標準輸出，標準錯誤，服務器監聽socket，進程間通訊的unix域socket等文件，那麼剩下的可用於客戶端socket連接的文件數就只有大概1024-10=1014個左右。也就是說缺省情況下，基於Linux的通訊程序最多允許同時1014個TCP並發連接。

對於想支持更高數量的TCP並發連接的通訊處理程序，就必須修改Linux對當前用戶的進程同時打開的文件數量的軟限制(soft limit)和硬限制(hardlimit)。其中軟限制是指Linux在當前系統能夠承受的範圍內進一步限制用戶同時打開的文件數；硬限制則是根據系統硬件資源狀況(主要是系統內存)計算出來的系統最多可同時打開的文件數量。通常軟限制小於或等於硬限制。

修改上述限制的最簡單的辦法就是使用ulimit命令：

[speng@as4 ~]$ ulimit -n<file\_num>

上述命令中，在<file\_num>中指定要設置的單一進程允許打開的最大文件數。如果系統回顯類似於"Operation notpermitted"之類的話，說明上述限制修改失敗，實際上是因為在<file\_num>中指定的數值超過了Linux系統對該用戶打開文件數的軟限製或硬限制。因此，就需要修改Linux系統對用戶的關於打開文件數的軟限制和硬限制。

第一步，修改/etc/security/limits.conf文件，在文件中添加如下行：

speng soft nofile 10240

speng hard nofile 10240

其中speng指定了要修改哪個用戶的打開文件數限制，可用'\*'號表示修改所有用戶的限制；soft或hard指定要修改軟限制還是硬限制；10240則指定了想要修改的新的限制值，即最大打開文件數(請注意軟限制值要小於或等於硬限制)。修改完後保存文件。

第二步，修改/etc/pam.d/login文件，在文件中添加如下行：

session required /lib/security/pam\_limits.so

這是告訴Linux在用戶完成系統登錄後，應該調用pam\_limits.so模塊來設置系統對該用戶可使用的各種資源數量的最大限制(包括用戶可打開的最大文件數限制)，而pam\_limits.so模塊就會從/etc/security/limits.conf文件中讀取配置來設置這些限制值。修改完後保存此文件。

第三步，查看Linux系統級的最大打開文件數限制，使用如下命令：

[speng@as4 ~]$ cat /proc/sys/fs/file-max

12158

這表明這台Linux系統最多允許同時打開(即包含所有用戶打開文件數總和)12158個文件，是Linux系統級硬限制，所有用戶級的打開文件數限制都不應超過這個數值。通常這個系統級硬限制是Linux系統在啟動時根據系統硬件資源狀況計算出來的最佳的最大同時打開文件數限制，如果沒有特殊需要，不應該修改此限制，除非想為用戶級打開文件數限制設置超過此限制的值。修改此硬限制的方法是修改/etc/rc.local腳本，在腳本中添加如下行：

echo 22158 > /proc/sys/fs/file-max

這是讓Linux在啟動完成後強行將系統級打開文件數硬限制設置為22158。修改完後保存此文件。

完成上述步驟後重啟系統，一般情況下就可以將Linux系統對指定用戶的單一進程允許同時打開的最大文件數限制設為指定的數值。如果重啟後用ulimit-n命令查看用戶可打開文件數限制仍然低於上述步驟中設置的最大值，這可能是因為在用戶登錄腳本/etc/profile中使用ulimit -n命令已經將用戶可同時打開的文件數做了限制。由於通過ulimit-n修改系統對用戶可同時打開文件的最大數限制時，新修改的值只能小於或等於上次ulimit-n設置的值，因此想用此命令增大這個限制值是不可能的。所以，如果有上述問題存在，就只能去打開/etc/profile腳本文件，在文件中查找是否使用了ulimit-n限制了用戶可同時打開的最大文件數量，如果找到，則刪除這行命令，或者將其設置的值改為合適的值，然後保存文件，用戶退出並重新登錄系統即可。

通過上述步驟，就為支持高並發TCP連接處理的通訊處理程序解除關於打開文件數量方面的系統限制。

2、修改網絡內核對TCP連接的有關限制

在Linux上編寫支持高並發TCP連接的客戶端通訊處理程序時，有時會發現儘管已經解除了系統對用戶同時打開文件數的限制，但仍會出現並發TCP連接數增加到一定數量時，再也無法成功建立新的TCP連接的現象。出現這種現在的原因有多種。

第一種原因可能是因為Linux網絡內核對本地端口號範圍有限制。此時，進一步分析為什麼無法建立TCP連接，會發現問題出在connect()調用返回失敗，查看系統錯誤提示消息是"Can't assign requestedaddress"。同時，如果在此時用tcpdump工具監視網絡，會發現根本沒有TCP連接時客戶端發SYN包的網絡流量。這些情況說明問題在於本地Linux系統內核中有限制。其實，問題的根本原因在於Linux內核的TCP/IP協議實現模塊對系統中所有的客戶端TCP連接對應的本地端口號的範圍進行了限制(例如，內核限製本地端口號的範圍為1024~32768之間)。當系統中某一時刻同時存在太多的TCP客戶端連接時，由於每個TCP客戶端連接都要佔用一個唯一的本地端口號(此端口號在系統的本地端口號範圍限制中)，如果現有的TCP客戶端連接已將所有的本地端口號佔滿，則此時就無法為新的TCP客戶端連接分配一個本地端口號了，因此系統會在這種情況下在connect()調用中返回失敗，並將錯誤提示消息設為"Can't assignrequested address"。有關這些控制邏輯可以查看Linux內核源代碼，以linux2.6內核為例，可以查看tcp\_ipv4.c文件中如下函數：

static int tcp\_v4\_hash\_connect(struct sock \*sk)

請注意上述函數中對變量sysctl\_local\_port\_range的訪問控制。變量sysctl\_local\_port\_range的初始化則是在tcp.c文件中的如下函數中設置：

void \_\_init tcp\_init(void)

內核編譯時默認設置的本地端口號範圍可能太小，因此需要修改此本地端口範圍限制。

第一步，修改/etc/sysctl.conf文件，在文件中添加如下行：

net.ipv4.ip\_local\_port\_range = 1024 65000

這表明將系統對本地端口範圍限制設置為1024~65000之間。請注意，本地端口範圍的最小值必須大於或等於1024；而端口範圍的最大值則應小於或等於65535。修改完後保存此文件。

第二步，執行sysctl命令：

[speng@as4 ~]$ sysctl -p

如果系統沒有錯誤提示，就表明新的本地端口範圍設置成功。如果按上述端口範圍進行設置，則理論上單獨一個進程最多可以同時建立60000多個TCP客戶端連接。

第二種無法建立TCP連接的原因可能是因為Linux網絡內核的IP\_TABLE防火牆對最大跟踪的TCP連接數有限制。此時程序會表現為在connect()調用中阻塞，如同死機，如果用tcpdump工具監視網絡，也會發現根本沒有TCP連接時客戶端發SYN包的網絡流量。由於IP\_TABLE防火牆在內核中會對每個TCP連接的狀態進行跟踪，跟踪信息將會放在位於內核內存中的conntrackdatabase中，這個數據庫的大小有限，當系統中存在過多的TCP連接時，數據庫容量不足，IP\_TABLE無法為新的TCP連接建立跟踪信息，於是表現為在connect()調用中阻塞。此時就必須修改內核對最大跟踪的TCP連接數的限制，方法同修改內核對本地端口號範圍的限制是類似的：

第一步，修改/etc/sysctl.conf文件，在文件中添加如下行：

net.ipv4.ip\_conntrack\_max = 10240

這表明將系統對最大跟踪的TCP連接數限制設置為10240。請注意，此限制值要盡量小，以節省對內核內存的佔用。

第二步，執行sysctl命令：

[speng@as4 ~]$ sysctl -p

如果系統沒有錯誤提示，就表明系統對新的最大跟踪的TCP連接數限制修改成功。如果按上述參數進行設置，則理論上單獨一個進程最多可以同時建立10000多個TCP客戶端連接。

\*\*\*\*\*\*\*注意\*\*\*\*\*\*\*

sysctl -p 報錯net.ipv4.ip\_conntrack\_max" is an unknown key 則：modprobe ip\_conntrack

o

## du命令查看文件大小

-s显示文件大小的综合

-a显示每个文件的大小

-m是以MB为计量单位

-g以GB为计量单位

du –am \* 显示当前目录下每个文件的大小

du –sm \* 显示当前目录下所有文件大小的总和

du –sm \*|sort –k1显示文件大小并按大小排序

## col -b 命令

用于将man一类有pause屏的文本连续起来（去掉pause）

例如：man cp |col -b > cp.man

## tee命令

相当于read 和echo > file的组合：提供从键盘读入文件的功能

从STDIN读一段字符串，写入文件中

[root@nm testsh]# tee test.txt

hello 键盘输入

hello 屏幕回显

[root@nm testsh]# cat test.txt

hello 证明写入文件中

[root@nm testsh]# tee -a test.txt a:append追加入文件

i m mac

i m mac

who are you

who are you

## 建立临时文件

建立临时文件需要注意一下几点：

1、不同的脚本建立的临时文件不可重复，以免造成意外的影响。

2、程序结束之前要记得清理临时文件，以免在系统中留下垃圾

3、临时文件可以包含有机密数据，建立临时文件时，应注意文件权限及程序安全。

由于每个shell进程编号不会重复，可以使用进程编号变量$$来定义临时文件

但这个方法有如下缺点：

通常会忘了，在脚本结束前清理临时文件；使用进程编号变量$$不够随机，容易被猜中（小于65535种可能而已）

临时文件权限没有做好把关，在脚本执行过程中恐遭人填入假数据。暴露了临时文件的位置，可能有遗漏信息的疑虑。

比较好的方式是改用系统建立临时文件的机制，使用mktemp的用法：

mktemp是用来建立唯一性的临时文件的工具，

具体语法如下:

mktemp 选项 临时文件模板

用例：

### 1、直接执行mktemp

如果不加任何选项和模板，直接执行mktemp，它会在/tmp目录下产生tmp.XXXXXX的临时文件,

其文件名末6个字符，由现行的进程编号和随机的英数字组成。

使用命令替换的技巧即可取得临时文件名，作用如下：

TMP\_file=$(mktemp)

### 2、指定临时文件模板

执行mktemp时，可以指定临时盘的模板，做法如下：

mktemp /tmp/tfile.XXXXXX

请注意：临时文件模板一定要是"文件名.XXXXXX"的格式（少一个X都不行），否则会发生错误。当临时文件建立成功时，

那6个X会被mktemp随机替换成6个英数字元。模板文件的路径不一定得在/tmp目录下，指定其他任何位置均可。

例如：

tmpfile=$(mktemp /root/tmp/sss.XXXXXX)

如果不想mktemp在发生错误时产生任何信息，可加上-q选项

mktemp -q /tmp/tfile.XXXXXX

### 3、指定临时盘的前置目录名

使用-p可指定mktemp建立临时盘的前置路径的目录名称，例如：

mktemp -p /root/.tmp123 myfile.XXXXXX

### 4、使用-t选项，按mktemp预先设置的方式，决定临时盘的路径

如果mktemp执行时有使用-t选项，则mktemp会先检查环境变量TMPDIR是否有定义，如果有，则以该变量的内容为临时盘的路径，如果无，则使用

-p所指定的前置路径，如果未使用-p则默认在/tmp目录下建立临时文件

### 5、如果使用选项-d，则建立时暂存目录

用例：mktemp -d tmpdir.XXXXXX

## 环境变量NLS\_LANG

NLS\_LANG是一个环境变量，用于定义语言，地域以及字符集属性。对于非英语的字符集，NLS\_LANG的设置就非常重要。

NLS：‘National Language Support (NLS)’ 当我们设定一种nls的时候实际上我们是为oracle在存放数据时指定了他的语种所特有的一些表达形式，比如我们选择chinese,那么它的中文字符如何存放,按什么规则排序，货币如何表示，日期格式也就被设定了。

NLS\_LANG参数由以下部分组成:NLS\_LANG=<Language>\_<Territory>.<Clients Characterset> NLS\_Language 指定: - Oracle（错误）信息的语言 - 日和月份的名称 注意：NLS\_LANGUAGE与插入和查询的\*数据\*的语言无关。

NLS\_Territory 指定: - 货币和数字格式 - 计算星期和天数的范围和惯例

客户端字符集（CLIENTS CHARACTERSET）: - 定义Oracle客户端，客户应用使用的编码 \* 或者它要符合您Microsoft Windows代码页 （GUI工具的ACP, 命令提示符的CHCP 值） \* 或者为Unicode WIN32应用设置为UTF8/AL32UTF8。

常见的值可以参见[Oracle Database Client Globalization Support](http://docs.oracle.com/html/B10131_02/gblsupp.htm)

### Linux/Unix下的设置

export NLS\_LANG=AMERICAN\_AMERICA.ZHS16GBK

export NLS\_LANG=SIMPLIFIED CHINESE\_CHINA.ZHS16GBK

DB中的NLS\_LANG DB也有自己的NLS\_LANG参数，来标识数据库的语言，地域，字符集。可以通过以下的SQL来查得， SELECT \* FROM v$nls\_parameters; OR SELECT USERENV ('language') FROM DUAL;

客户端的NLS\_LANG和DB的NLS\_LANG DB的NLS\_LANG和客户端环境变量中NLS\_LANG尽可能的保持一致。Setting the NLS\_LANG Environment Variable for Oracle Databases 提供了如何保持一致的方法。 那么为什么要保持一致呢？那是因为如果字符集不匹配，有可能在DB和Client间相互转换的过程中，出现字符转换错误的情况。比如Oracle nls\_lang tips中举的例子。

### Windows中设置NLS\_LANG

1.命令提示符中设置为环境变量

如果您在命令行中设置NLS\_LANG为环境变量，它将覆盖注册表和系统属性中的NLS\_LANG的定义。在命令提示符中，使用“set”命令，例如：

LANGUAGE设置为中文时，提示如下：

C:\Users\TianPan>set nls\_lang=SIMPLIFIED CHINESE\_CHINA.ZHS16GBK

C:\Users\TianPan>sqlplus /nolog

SQL\*Plus: Release 11.2.0.1.0 Production on 星期五 11月 28 10:07:21 2014

Copyright (c) 1982, 2010, Oracle. All rights reserved.

SQL> conn scott/tiger 已连接。

SQL> select sysdate from dual;

SYSDATE -------------- 28-11月-14

当Language设置为American时，提示信息则变为英文:

C:\Users\TianPan>set nls\_lang=AMERICAN\_AMERICA.ZHS16GBK

C:\Users\TianPan>sqlplus /nolog

SQL\*Plus: Release 11.2.0.1.0 Production on Fri Nov 28 10:07:51 2014

Copyright (c) 1982, 2010, Oracle. All rights reserved.

SQL> conn scott/tiger Connected. SQL> select sysdate from dual;

SYSDATE ------------ 28-NOV-14

在Windows下通过set nls\_lang来设置，只是Session级别的，关闭cmd窗口后再打开，就又变为原来的设置了。如需永久生效，可以修改注册表的参数。

2.注册表设置（永久生效）

默认情况下，windows上的Oracle安装使用注册表来定义这个设置。 版本 10g 及以上: HKEY\_LOCAL\_MACHINE\SOFTWARE\ORACLE\KEY\_<oracle\_home\_name> 这里您会看到一个条目名为NLS\_LANG

在64位windows平台上安装32位软件，会使用32位兼容性路径 HKEY\_LOCAL\_MACHINE\SOFTWARE\Wow6432Node\ORACLE\KEY\_<oracle\_home\_name>

3.操作系统环境变量

尽管注册表是Microsoft Windows设置的主要存储位置，它不是唯一可以设置参数的地方。

虽然不推荐，但您可以设置NLS\_LANG为系统属性中的系统或者用户变量。这个设置会被所有Oracle home使用。

设置位置： '我的电脑' -> '属性' > '高级' -> '环境变量'

因为这些环境变量比注册表中已经设置的参数级别高，所以除非您有非常好的理由否则不要在这个位置设置Oracle参数。

## locale –a

locale –a可以查看当前系统已经安装和支持的字体字符编码

## locale命令

locale命令可以显示本地自然语言的相关信息，locale决定系统如何显示文字、数字、日期、货币及资料的排序方式。Locale的命令格式是：

语系\_地区.文字编码

例如：中国流行的locale是zh\_CN.GBK，指的是中国大陆地区（CN）使用中文语系（zh），文字的编码是GBK，近年来各种系统逐渐支持 UTF-8编码， UTF-8编码变成国际主流，因此

zh\_CN.UTF-8的locale也较为流行。

在主机系统中有两个环境变量与locale的设定有关：

declare –x LANG=”zh\_CN.GBK”

declare –x LC\_ALL=”zh\_CN.GBK”

如果LC\_ALL变量有设置值，LC\_ALL变量的值会覆盖LANG,以及locale目前所有变量的值

## FTP自动登录

利用输入转向（Here Document）可以实现ftp的自动登录

#!/bin/bash

FileName=”test.gz”

IP=”172.17.248.180”

USER=”ch\_txn02”

PASS=”ch\_txn02”

ftp -n $IP << AUTOEOF

user $USER $PASS

bin

cd /root

get test.gz

quit

AUTOEOF

## SSH自动登录

FTP协议在登录和传输过程中没有加密，极易被别有用心的人使用sniffer之类的工具窃取密码，ssh则改进了这个缺点。

如果要建立ssh自动登录机制，首先要在ssh客户端产生一组公钥和私钥，然后将公钥上传至SSH Server端,幷做好相应的设置（如：权限属性），私钥则留在client端，接着就可以利用scp进行远程复制的操作，不必输入任何密码。它的原理是：当用户要登录远程主机时，只要指定采用身份文件（identity）来识别，服务端就会拿出公钥的变化内容向client端询问，如果client端能以它的私钥求出这项变化内容的原因，则满足服务端的挑战（challenge），即身份识别成功。因此免用密码即可登录。

产生公私钥的方法，可采用OpenSSH提供的ssh-keygen,做法如下：

ssh-keygen –f 密钥文件名 -t 使用的算法

其中密钥文件名可以自定义，而可选用的算法有RSA和DSA两种，一下范例采用RSA算法，密钥文件名为id\_rsa(此为身份文件)

ssh-keygen -f id\_rsa -t rsa

当程序询问密码和第二次确认密码时，请按【Enter】键表示不设密码，此时会在工作目录中产生私钥文件id\_rsa以及公钥文件id\_rsa.pub,

接下来，把公钥文件上传到SSH Server端的/root/.SSH目录下，幷把文件名改为authorized\_keys。接着登录SSH Server,以root身份执行以下权限的设定（请自行把它写成脚本备用）：

cd /root

chmod go –w **.**  (把群组和其他身份对root目录的写权限拿掉)

cd .ssh

chmod 700 **.**（把.ssh目录本身的权限改成700）

chmod 600 \* (把.ssh目录下所有的文件的权限改成600，即只有root可以读写)

然后离开SSH Server（假设是192.168.1.2）回到客户端，使用选项-i，指定使用当前在目录下面的私钥文件id\_rsa(此即为指定身份文件)进行ssh连接测试：

ssh [root@192.168.1.2](mailto:root@192.168.1.2) -i id\_rsa

如果连接SSH Server时没有询问密码即可登录，表示测试成功。请先离开SSH Server

接下来，在client端把私钥文件id\_rsa移入.ssh目录下，然后再测试一下

ssh [root@192.168.1.2](mailto:root@192.168.1.2)

如果不询问密码就能登录，那就大功告成

接下来说明scp的用法。scp是secure copy的意思，即安全的复制，特别适合用于对远程主机的文件复制

scp的语法：

将远程主机中的文件复制到local 主机

scp 账号@主机：/目录/文件 /目录/文件

例：

scp [root@192.168.1.2:/var/www/index.php](mailto:root@192.168.1.2:/var/www/index.php) tmp/testidx.php

将远程主机中的文件复制到local主机：

scp /目录/文件 账号@主机：/目录/文件

例：

scp tmp/testidx.php [root@192.168.1.2:/var/www/index.php](mailto:root@192.168.1.2:/var/www/index.php)

如果要复制整个目录，可以在scp之后加上-r选项，如果要保持文件的属性，可以加上-p选项

自动登录SSH，幷不是只能执行scp而已，任何指令都行！例如：

ssh 192.168.1.2 `echo hello world`

就是在自动登录远程主机192.168.1.2之后，在改主机中执行echo指令，但执行结果响应到local端

再以下指令为例，可将远程主机中的/etc/hosts文件的内容显示出来。

ssh 192.168.1.2 `cat /etc/hosts`

自动登录SSH的机制，功能相当强大，有了它，就可以把远程主机当成好像local主机一样来操作。

## scp命令

scp的语法：

将远程主机中的文件复制到local 主机

scp 账号@主机：/目录/文件 /目录/文件

例：

scp [root@192.168.1.2:/var/www/index.php](mailto:root@192.168.1.2:/var/www/index.php) tmp/testidx.php

将远程主机中的文件复制到local主机：

scp /目录/文件 账号@主机：/目录/文件

例：

scp tmp/testidx.php [root@192.168.1.2:/var/www/index.php](mailto:root@192.168.1.2:/var/www/index.php)

如果要复制整个目录，可以在scp之后加上-r选项，如果要保持文件的属性，可以加上-p选项

## ifconfig –a

获取主机的IP地址

## uname命令

uname -s 核心名称

uname -n 核心所在的网络节点名（即主机名）

uname -r 核心版本

uname -v 核心编译版次

uname -m核心所在主机硬件

uname -c 操作系统种类

uname -a 核心所有的信息

## free命令

free会把真是的内存和置换空间（SWAP）使用的情况显示出来，如果加上-b,-k,-m选项可分别使用byte，kb，mb来表示内存的大小

也可以指定每隔几秒中就更新一次显示，做法如下：

free -s 5

## fdisk命令-列出磁盘容量

列出磁盘容量

fdisk -l /dev/磁盘设备名

例如：fdisk –l /dev/sda

## df命令-列出磁盘用量

列出磁盘用量

## W和top命令

W和top命令可以列出主机系统平均负载值

## find命令

find 目录 -name “文件名或模式” -exec 命令(如ls -alr) {} \;

其中{}代表找到的文件,\;表示命令结束（;在bash中是分割命令的符号，此处用\转义）

find 目录 -name “文件名或模式” |xargs grep –ni “内容”

## du –sm查看目录的大小

-s显示文件大小的综合

-a显示每个文件的大小

-m是以MB为计量单位

-g以GB为计量单位

## 系统端口

系统的端口范围从0-65535.其中0-1023保留给系统服器专用，而且，只有以root权限执行者，才能使用此段范围，称为特权端口（privilege port）,1024-65535称为非特权端口。

## netstat –anp查看连接状态

-a表示所有未连接和连接的socket都列出来，-n表示全部以数字显示，不对主机名，端口、用户等信息反查其对应的名称

-p选项会列出每一个和socket有关的程序名称和进程编号（PID）(请记牢)

netstat -tan 查看tcp连接

netstat -uan 查看udp连接

## lsof命令

lsof会列出主机中已开启的文件，包括一般文件、目录、区域文件、字符文件、串行(stream),甚至是网络连接。

## md5sum和sha1sum工具

checksum是一种固定长度的信息摘要，所谓信息摘要是说，当原来的信息量很大时，可以用数学的方法，找到一段足以代表该信息的长度比较小的摘要，而且，任何两个信息的摘要完全相同的几率，可以说非常小，如果有，则称为碰撞（collision），这种能产生信息摘要的函数称为hash函数，常见的hash数学有md5和sha1

md5产生消息摘要的长度为128位（即32个字符），sha1的信息摘要长度为160位（40个字符），虽然md5已经被证实遭到破解，sha1也被发现有部分密码学上的缺陷，但在一般的用途上，足以胜任。

哈希函数可以运用在文件完整性的审核上。其原因是只要文件有改动哪怕是1个位，checksum值就不一样，因此只要checksum值不变，就表示文件没有被篡改过。

大部分linux/BSD系统都会提供两个checksum的工具，一个是md5sum，一个是sha1sum

### md5sum的用法：

#### 产生checksum值

md5sum /bin/ps > ps.md5

#### 检查checksum值是否正确

md5sum -c ps.md5

### sha1sum用法：

#### 产生checksum值

sha1sum /bin/ps > ps.sha1

#### 检查checksum值是否正确

sha1sum -c ps.sha1

## csum 命令

AIX系统中csum命令生成和校验文件的checksum(和linux中的md5sum类似)

### 产生checksum值

csum -o 输出文件名 -h [MD5 | SHA1] 文件名

如果不指定-h，默认是MD5,即使用MD5哈希算法

例：

csum -o test.md5 -h MD5 test.c

csum -o test.sha1 -h SHA1 test.c

### 检查checksum

csum -I test.md5 -h MD5

csum -I test.sha1 -h SHA1

## tar命令

tar的选项有：

c-create

x-extract

f-file

v-verbose

z-gzip

j-bzip

### 建立tar文件

tar –cvf tar文件 文件或目录

例如：tar –cvf etc.tar /etc

### 建立tar文件时，再加压缩

常见的压缩工具是gzip和bzip2,其中bzip2的压缩比较高

tar cvzf etc.tgz /etc

选项z表示要用gzip把etc.tar压缩，最后的存储的文件名为etc.tgz，也可以写成.tar.gz

### 使用bzip2的做法：

tar –cvjf etc.tar.bz2 /etc

### 查看tar文件的内容

tar tvf etc.tar

选项t是列出tar文件的内容（只看，不解开）

如果tar文件已经过压缩，那么，如果要查看文件包，那么，如果要查看文件包的内容，必须使用适当的选项，如z调用gzip，或j调用bzip2

例如：查看gzip压缩过的tar文件：

tar -tvzf etc.gzip

查看bzip压缩过的tar文件：

tar -tvjf etc.bz2

### 解开tar文件

tar -xvf etc.tar

#### 解开用gzip压缩过的tar文件

tar –xvzf etc.tar.tgz

#### 解开用bzip2压缩过的tar文件

tar –xvzf etc.tar.bz2

#### 解压到指定目录

指定的目录一定要先存在

tar –xvzf etc.tar.tgz –C /root/tmp/test

选项C表示要切换到指定目录，而且/root/tmp/test一定要先存在才行。

一下做法可保证顺利执行：

Dir=/root/tmp/test

[ -d $Dir ] || mkdir –p “$Dir” && tar –xvzf etc.tar.tgz -C /root/tmp/test

#### 添加文件到tar文件

tar –rvf etc.tar test.txt

选项r表示在tar文件后面添加文件

需要注意的是，选项r和gzip和bzip2无法合用，如果要在tgz和tar.bz2中加入文件，必须先解压文件

#### 更新tar文件中的文件

更新tar文件中的文件可以用选项u

tar -uvf etc.tar test.txt

#### 合并tar文件

tar –Avf 1.tar etc.tar test.tar

将etc.tar、test.tar合并成1.tar

tar –Avf etc.tar test.tar

将etc.tar、test.tar合并成etc.tar

#### 删除tar中某一文件

tar -vf etc.tar --delete etc/hosts

#### 比较tar文件和实际的文件

首先要把tar文件和要比较的文件系统放在同一层目录，例如：etc.tar和etc都在根目录下，然后执行以下指令：

tar -dvf etc.tar

如果有不同就会出现警告信息。

## 清理共享内存命令

### AIX系统中：

ipcs  -mqs|grep   $USER|awk '{print  "ipcrm -"  $1$2}'|sh  -x (清理IPC资源)

### Linux系统中：

ipcs –m|grep $USER|awk ‘{print “ipcrm -m” $2}’|sh –x （清理共享内存）

## sync命令

在unix系统中内存就是磁盘inode的Cache，因此切断电源前如果不用sync命令把Cache的内容刷新到磁盘中，文件系统可能会损坏。

# 系统架构

## 高可用性HA

### 1通信层负载均衡

### 2应用层HA（High Available）

**主备Live-Standby**

-IBM HACMP

**双活Live-Live**

### 3数据库RAC

RAC，全称real application clusters，译为“实时应用集群”

## 可靠性

### 1链路的可靠性

* 心跳检测
* 消息序号和重发机制

### 2事务可靠性

* MQ的读写一致性
* 基于MQ的二阶段提交

### 3失败恢复和灾备切换

* 进程失败时LM尝试重启进程
* LM失败时DM尝试切换LM
* 主机故障时HA切换
* 异地灾备切换

# clear case

## ClearCase的一些基本概念

1. VOB--Versioned Object Base, ClearCase将所有管理的文件的各种版本都存储在这个VOB中，VOB可以看作是整个ClearCase SCM系统的中心[**数据库**](javascript:;)。  
     
   2. View--View分为SnapShot View和Dynamic View，Snapshot view是clearcase在服务器上存储的文件和目录的一个本地镜像，用户可以在本地进行修改，然后进行同步，要经常Update View保持最新的版本，Dynamic View是动态试图，他并不在本地存储任何文件，始终和服务器保持一致。  
     
   3. Reserved checkout vs unreserved checkout -- 一个文件可以被多个用户的多个view来unreserved checkout，但是同时只能有一个用户reserved checkout。当一个用户reserved checkout的时候，其他unreserved checkout的文件，不能checkin，只能等reserved checkout的文件被checkin之后才能够checkin。  
     
   4. hijacked文件--当用户创建Snapshot View的时候，本地的文件属性都是只读的，如果用户没有check out的情况下就对文件进行了修改，这个文件就为hijacked文件，此时这个文件已经脱离的ClearCase的控制，所以最好不要Hijacked 文件。  
     
   5. mastership--很多情况下，ClearCase都被部署为MultiSite的形式，特别是跨地域开发的时候，每个地方的开发人员都在本地的一个VOB副本上[**工作**](javascript:;)，叫做replica, ClearCase负责同步这些不同的VOB。为了避免冲突，ClearCase提供了一个排他修改的属性，叫做mastership。所有的VOB对象都有一个master replica。master replica 对这个对象有排他的修改操作权限，因此对于一个VOB对象只有master replica才能对他进行修改或删除。所以在你把新的文件或目录Add to source control的时候，最好要选择 “**Make current replica the master of all newly created branches**”。  
     
   6. merge文件--当一个用户check out一个文件进行了修改，在check in的时候如果clearcase发现这个文件和最新的版本有冲突的时候（可能是[**其他**](javascript:;)用户也对该文件进行了修改并已经check in），会提示要merge文件，这时候就可能需要手工的merge了。

## clear case 简介

ClearCase是一种配置管理工具，由Rational公司开发，是开发小组用来跟踪、管理软件开发过程各个工件的配置管理系统，ClearCase可以协助开发组织更好地管理软件开发进程。  
ClearCase可以和Rational公司的其他软件紧密结合，例如UCM、ClearQuest等等。  
ClearCase包括两套：ClearCaseLT和ClearCase(MultiSite)。前者可以用于在同一个局域网的开发小组，适合于中小型开发组织；ClearCase(MultiSite)则适应于分布于不同地理位置、不同局域网的开发小组，适合于大型的开发组织。  
因此，一般的公司使用ClearCaseLT就足够了。  
注：在后续[文章](http://www.it-crazy.com/)中，经常将ClearCase简称为CC。

2 ClearCaseLT工作原理

2.1系统组成

ClearCaseLT由ClearCaseLTServer和ClearCaseLTClient两个部分组成，其中服务器部分负责数据的集中管理；客户端部分则安装在各个需要使用ClearCase服务的机器上，以及主要开发人员的计算机上，属于典型的Client/Server结构。  
ClearCaseLTServer采用VOB(VersionedObjectBase)存储配置管理数据，用户通过视图（VIEW）的方式获取VOB中存储的数据；UCM方式（参见“项目管理”部分）中，主要通过Stream（流）和Activity（活动）管理项目。

2.2VOB、VIEW

VOB是ClearCase存储软件系统各个组成元素的所有版本数据的仓库，一个版本是其中包含的特定文件或目录；VIEW则是通过一定的规则选择出来的各个元素特定版本的集合，用户通过VIEW存取、修改各个元素。  
VOB和VIEW的关系如下图所示：  
视图有两种类型：快照视图（snapshotview）及动态视图（dynamicview）。快照视图，是将CC服务器中的视图内容拷贝到开发人员的机器中，开发人员需要经常与服务器同步以保持数据的一致性，快照视图的好处在于开发人员不必一直通过网络与CC服务器保持连接；动态视图则是动态的将CC服务器中的内容同步到开发人员的机器中，这就要求开发人员一直保持与服务器的网络连接。一般来讲，由管理员决定选用哪种视图。  
开发人员的开发涉及到两个视图：开发视图和集成视图。如果用户的名字为pat，参与的项目叫做test，那么两个视图缺省的名字为pat\_test和pat\_test\_integration。  
开发视图用于开发人员的开发过程，开发人员在开发视图中完成软件的开发、修改、提交等工作；集成视图的作用是存放开发人员完成的工作，使得开发人员可以通过该视图中的内容对其开发进行验证。

2.3CHECK-IN&CHECK-OUT

像其他配置管理工具一样，ClearCaseLT通过Check-in和Check-out的方式实现配置管理。Check-out一个文件时，ClearCase就会在视图中创建该文件的一个可编辑的版本，可以对该文件进行修改；Check-in一个文件时，ClearCase就在VOB中创建该文件的一个新的永久的版本，本地视图中对应的文件就会变成只读属性，无法修改。

2.4基线（Baseline）

在软件开发过程中，可以将各个元素的不同的版本组合成一个基线。通过基线表示软件项目开发达到了一定的要求，也可以说是里程碑。  
项目管理员可以根据情况设置相应的基线，并随着项目的发展逐步设置新的基线。

2.5活动（Activity）

活动是开发者为完成、提交一项开发任务而创建的，是用来记录开发者创建或修改的文件集合（变化集合）的对象，例如下图所示的修改BUG。一个活动包含一个组件（Component）两个或多个元件(Element)的多个版本。  
活动是UCM的基本对象。  
通过活动，开发者可以一次完成对多个文件的操作，而不必一一处理，可以极大的提高工作效率。

2.6流（Stream）

流记录了在项目的私有工作区和共享工作区中的所有活动历史。流同时也定义了开发者通过私有工作区可以存取相应元件（例如文件foo.c）的哪个版本。

3ClearCaseLT设置

3.1安装ClearCaseLTServer

安装ClearCaseLTServer的步骤参见后续章节的ClearCase安装部分。  
安装完毕ClearCaseLTServer以后，需要进行一些设置，主要是数据的准备、环境变量设置、权限设置等。

3.2生成基础VOB

ClearCase服务器安装完毕之后，会自动启动ClearCase服务器设置向导，设置初始的VOB。  
也可在以后单独运行该向导。  
如果需要额外的VOB数据，需要通过ClearCase管理控制台（AdministrationConsole）完成。

3.3设置环境变量

由于WindowsNT/2000中的BUG，ClearCase要正常完成授权管理，必须设置环境变量：  
a.打开控制面板，选择“系统”进入“系统特性”设置  
b.选择“高级”页面，点击“环境变量”按钮，进入环境变量设置。在“用户变量”一栏中点击“新建”按钮，输入变量名称：CLEARCASE\_PRIMARY\_GROUP变量值设置为将要访问ClearCase的用户组，例如CLEARCASE\_USERS，表示CLEARCASE\_USERS组可以访问ClearCase。（在安装ClearCaseClient的Windows2000/NT的机器里面也必须设置CLEARCASE\_PRIMARY\_GROUP，方法参照b；Windows98需要在autoexec.bat文件里面设置该环境变量）

3.4权限管理

权限管理在ClearCase的使用管理中十分重要，必须正确设置权限，才能正常使用ClearCase的各种功能。

3.4.1ClearCaseLT服务器端的权限设置

ClearCase通过Windows提供的用户组实现用户登录管理，因此必须设置相应的可以访问ClearCase功能的用户组。  
以上一节提供的数据为例，ClearCase管理员需要在ClearCase服务器上生成一个用户组CLEARCASE\_USERS，然后产生要使用ClearCase进行软件开发的用户账号，并且将这些账号指定到CLEARCASE\_USERS组。  
注意：指定的用户组名称必须和前面设置的环境变量CLEARCASE\_PRIMARY\_GROUP中的名字相同！

3.4.2ClearCase客户的设置

ClearCase客户端必须采用在服务器中提供的账户，登录到服务器域，才能使用ClearCase。

3.5其他事项

如果遇到问题，可以使用ClearCase提供的“RationalClearCaseLTDoctor”检查问题所在，然后根据它提供的方法解决问题。  
该工具在ClearCaseLT服务器和客户端都有。

4ClearCaseClient安装、配置

4.1配置LicenseServer（许可服务器）

运行ClearCase安装[程序](http://www.xuebuyuan.com/)，根据提示逐步进行，选择Client即可安装客户端、选择Server即可安装服务端。  
待安装完成后，在目标机中执行“开始”菜单栏中的RationalLicenseKeyAdministrator。首先点击按钮“取消”，然后点击“Settings”，弹出对话框后，点击“licenseFiles”，做相应软件授权设定。  
点击按钮“确定”配置成功。

4.2安装ClearCaseLT软件

要安装ClearCaseLTServer，则使用鼠标选中“RationalClearCaseLTServer”，然后点击按钮“下一步”，开始安装Server。  
要安装ClearCaseLTClient，则使用鼠标选中“RationalClearCaseLTClient”，然后点击按钮“下一步”，开始安装Client软件。  
根据提示，继续安装，就可以完成Server或者是Client的安装。

4.3ClientCase客户端配置

安装完毕客户端软件后，需要对客户端系统进行一些设置。  
首先配置用户组，具体方法参照错误！未找到引用源。对于Windows98中设置环境变量的方法，请自行参考相应的文档。  
其次设置网络属性，设置客户机的网络ID以及登陆的域。注意：需要在域中填写ClearCaseLTServer所在计算机的域名称。  
在Windows2000中，通过控制面板打开“系统”功能，进入“网络标识”页面，点击“属性”按钮，设置计算机名称和所属的域。  
用户每次登录计算机时，必须选择登录到域。输入在ClearCaseLTServer计算机上的用户名称和密码，这样才能使用ClearCase提供的服务。

5项目管理

使用CC进行项目管理，可以采用两种方式：BaseClearCase方式和UCM(UnifiedChangeManagement)方式。  
BaseClearCase方式是基于文件的管理方式，UCM方式则主要采用Activity方式进行项目管理。与BaseClearCase方式相比，UCM方式具有以下一些优点：  
λUCM比BaseClearCase更加有效  
λUCM使用Stream方式管理项目，比BaseClearCase采用Branch方式管理项目的方式减少了配置工作  
λUCM方式对基线的管理更加自动化并提供了附加的执行操作  
λUCM方式使用活动管理项目，更加简单，并提供了与ClearQuest工具集成的功能  
λUCM方式更加有利于并行开发模式  
下面对项目管理的讨论主要基于UCM方式。

5.1UCM基础

UCM方式采用一种迭代开发过程，开发人员在同一个UCM项目中工作。  
项目管理者负责创建项目，维护项目公共区域。  
一个项目包括了公共区域和多个私有工作区域，私有工作区域允许开发人员在活动上各自独立地进行工作。  
工作过程如下：  
λ项目管理者创建项目并且为项目中的元件确定初始基线集合；  
λ开发人员通过创建私有工作区域、获取项目基线内容，加入该项目；  
λ开发人员创建活动并且一次在一个活动上工作，与活动相关的文件集合称为变化集；  
λ当开发人员完成活动，并且在其私有工作区对其工作进行了测试后，通过执行交付将其工作与开发组共享。交付操作将开发者私有区域中的工作合并到项目共享区域。  
λ项目管理者在项目共享区域集成开发人员交付的工作。  
λ项目管理者定期在项目共享域创建新的基线，用来集成开发人员的工作。  
λ项目管理者执行快速验证测试，以保证新的基线可以正常工作。软件质量工程师将执行更多的扩展测试。  
λ项目管理者定期在基线的质量和稳定性提高后调整极限的晋升级别（如：创建、测试、发布）以反映适当的里程碑。当新的基线经过了足够的测试，项目管理者可以将其指定为推荐基线。  
λ开发者执行rebase操作来修改其私有工作区，使其包含新的推荐基线所确定的新版本集。  
λ开发者继续如下的开发循环：基于活动进行开发工作、发布完成的活动、根据新基线修改其私有工作区域。  
上述的任务可以分为两个循环：项目管理和开发

5.2创建项目

创建并配置一个项目，需要按照以下步骤进行：  
λ创建存放项目信息的仓库  
λ创建包含开发者使用的文件集组件  
λ创建基线，以便开发人员可以确定开始其工作的文件版本  
λ选择希望采取的开发策略  
注意：创建新的项目应当以安装CC时使用的用户登录，其他用户才能正常使用该项目。  
创建步骤：  
a.使用安装了CCLTServer的计算机，运行其中的”RationalClearCaseLTAdministrationConsole”，加入保存新的项目数据所需要的VOB。  
在此创建的VOB为TEST（具体是否创建VOB根据实际需求而定）。  
b.打开运行RationalClearCase程序组中的“RationalClearCaseLTProjectExplorer”，进入项目浏览器：创建新的项目，根据提示操作，选择缺省的选项，一直到第三步：NewProject–Step3  
c.添加项目使用的基线  
点击“Add”按钮：  
从组件列表中选择指定的组件（component），在此，选择刚刚创建的组件TEST，然后基线列表中出现了与组件相关的基线，选择一个基线，新的组件只有一个基线：xxxx\_INITIAL，此处选择的是TEST\_INITIAL基线。点击OK，这样，就为将要创建的项目选择了初始基线，点击下一步，进入步骤四。  
d.设置可修改组件、调整项目策略  
选择项目中需要修改的组件，在此选择TEST，实际应用中需要根据实际情况做出选择。  
点击按钮“Policies”，调整项目的策略，具体的项目策略应当根据实际情况进行调整，一般是将”ForAlldeliveries”策略设置为Enabled.  
e.继续选择缺省的配置，完成项目的创建。这样就生成了一个以缺省VOB为基础的新项目，新项目生成一个缺省的Integration\_Stream：testa\_Integration（这里假定刚刚生成的新项目名称为testa）

5.3通知开发组成员加入新的项目

创建了新项目后，需要通知、指导开发人员加入到新创建的项目中，具体的方法参见开发部分。

5.4管理项目

当通知开发人员加入到新项目开发中后，项目管理员的职责就是不断监视项目开发的进展情况，对不同开发者提交的[源代码](http://www.xuebuyuan.com/)进行相应的管理、验证工作；在开发进展到一定程度后，为项目生成新的基线；指定新的推荐基线，直到项目开发完毕。（参见UCM基础部分）

5.4.1生成新的基线

管理员需要根据开发人员的开发情况，经常针对当前开发生成新的基线，以便更好的保存项目的开发过程。  
操作过程如下：  
(1)启动”RationalClearCaseLTProjectExplorer”，选择当前的项目，选中项目的集成视图，点击鼠标右键，选择”Properties”功能，修改属性：进入Lock页面，将状态改为Locked。锁定IntegrationStream，防止用户提交新的数据。  
(2)选中项目的集成视图，点击鼠标右键，选择”makebaseline…”功能，在弹出的”makebaseline”对话框中，输入基线的名称、描述、类型，并且在”ViewContext”中选择对应的集成视图。上图中选择的是递增类型视图、以jim\_testa\_integration视图为基础。  
(3)按照(1)中的方法，将IntegrationStream的状态改为Unlocked。

5.4.2管理基线

在项目开发过程中会产生许多的基线，管理员需要根据项目开发的进展来管理基线，例如改变基线的属性、提升基线等。  
下面举例说明如何提升基线，这主要涉及到修改基线的属性。（可以在多个地方修改基线的属性，下面只是其中的一种）  
a.进入IntegrationStream属性功能  
b.选择基线  
c.调整基线属性  
在晋升(promotion)下拉框中，列出了缺省的晋升级别，根据当前项目的进展情况选择一个，然后点击“确定”按钮。  
REJECTED表示该基线所代表的内容被拒绝接受；INITIAL表示该基线为初始基线；BUILT表示该基线的内容已经创建成应用程序；TESTED表示该基线已经被测试过；RELEASED表示该基线已经发布。

5.4.3指定推荐基线

在一个开发团队并行开发项目的过程中，很可能出现各个开发人员修改不同步的情况，有时会导致对同一个文件出现不同的版本，这就需要管理员对此进行跟踪管理，采用的主要手段是推荐基线。  
采用推荐基线就是要求各个开发人员将他的开发视图中的内容与基线所代表的内容保持一致，以避免冲突。  
对于各个开发人员提交的开发数据，管理人员进行审查、测试，创建基线，当提交的[代码](http://www.xuebuyuan.com/)已经达到一定的稳定程度或者是一定规模后，管理人员可能认为需要让所有的开发人员与当前的代码保持一致，他就可以将认为可行的基线设置为推荐基线，然后要求各个开发人员rebase他们的开发视图与该推荐基线中的内容保持一致，然后再在此基础上继续开发。  
在一个项目的开发过程中，管理员不断的生成新的基线，根据实际情况变动推荐基线，以保证各开发人员工作的一致性。  
管理员可以根据实际情况添加、删除或者改变当前的基线。  
完成设置新的推荐基线后，需要通知各个开发人员rebase他们的开发视图，以便与推荐基线的内容保持一致。

6项目开发

6.1ClearCaseLTClient工具简介

ClearCaseLTClient安装成功后，上述各个工具中，RationalClearCaseLTExplorer将是开发人员最常使用的工具，其他一些工具许多都已经集成在RationalClearCaseLTExplorer中了。  
RationalClearCaseLTExplorer左边是ShortCutPane（快捷区），通过它可以方便的进行各项操作，包括两个部分：Toolbox（根据箱）和Views（视图），Toolbox中包括各种工具，Views中则是开发人员所参与的各个项目的视图。  
λToolbox中，GettingStarted中是使用帮助部分；UCM部分是统一变化管理部分，开发人员将主要使用该部分来进行开发过程的控制，功能包括：加入项目开发、提交数据、Rebase、打开项目浏览器；BaseClearCase部分是基础的CC功能，包括修改视图属性、创建新的视图、删除视图、修改快捷视图等。  
λ在Views中列出了开发人员参与的各个项目的视图，一般每个项目包括两个视图：开发视图和集成视图。上图显示的是项目testa的开发视图jim\_testa，该视图中尚无内容，其中可以看到一项view.dat，该文件表示了与视图相关的内容，开发过程中切勿删除此文件，否则CC将失去该视图，恢复操作很复杂。  
下面开始详细介绍ClearCase的使用。

6.2开发流程

开发人员的主要的工作流程是加入项目，然后基于活动实施开发，提交活动，根据管理员的要求rebase工作区，然后继续开发，直到开发完成。

6.3加入一个项目

要加入一个项目，主要点击RationalClearCaseLTExplorer/Toolbox/UCM中的”JoinProject按钮”（参见ClearCaseLTClient工具简介部分），然后根据提示进行操作，就可以完成加入项目开发的过程。  
加入项目后，点击菜单View/RefreshViewShortcuts，在左边面板的Views中就会出现项目的视图，同时，在计算机的硬盘（缺省的是C盘）中会产生两个目录：user\_project和user\_project\_integration。上面的例子中生成的目录为：jim\_testa、jim\_testa\_integration。  
以后，开发工作将会在user\_project中展开，测试工作将会在user\_project\_integration中进行。  
注：要想加入到项目开发，计算机的登录用户应当满足“ClearCaseLT设置”部分的要求。

6.4进行开发工作

加入到一个项目后，就可以开始着手开发工作，在开发过程中，可以使用不同的开发工具，可以提交的东西包括软件源代码、设计文档以及根据管理员要求可以提交的其他内容。开发人员可以使用开发工具直接在开发视图中进行修改。  
但是，在开发中一定需要记住一点：在开发视图中对文件进行修改之前，一定需要将准备修改的内容checkout，然后才能修改，修改完毕后，使用checkin加入到视图中。

6.4.1加入文件、目录

最初加入项目的开发人员可能面对的是一个空的项目，需要不断加入新文件或者是目录，这可以在RationalClearCaseLTExplorer中或者Windows资源管理器中完成。  
在开发视图对应的文件夹的快捷菜单出现了ClearCase的快捷操作方式。  
通过Windows资源管理器，在开发视图目录中生成新的目录，拷贝加入文件。例如：在jim\_testa/test目录中加入文件夹source、doc，source文件夹中加入新的文件home.html，这时新产生的目录和文件并没有纳入CC的管理之下，必须手工将其加入到CC管理，即AddtoSourceControl。  
首先选择需要加入到CC控制的目录或者是文件，然后通过Tools/AddtoSourceControl或者快捷按钮将其加入到版本控制中。  
添加到版本控制中要求选择“活动”，(UCM方式基于活动)，输入活动名称，活动被命名为addfolder。这样新加入的内容就加入到CC控制之下。  
注意：任何新加入的文件/目录都需要通过AddtoSourceControl加入到CC中。

6.4.2CheckOut

加入新的代码在开发过程中需要不断的修改，首先需要完成CheckOut工作。操作比较简单，选中需要CheckOut进行修改的目录或文件，然后通过Tools菜单或快捷菜单或快捷按钮，实现CheckOut功能。任何需要修改的文件，都要执行CheckOut功能，才能进行修改。  
执行了CheckOut之后，可以将ClearCase的各个工具关闭。

6.4.3修改

CheckOut以后，开发人员就可以对相应的文件进行修改了，可以在开发视图目录中直接修改，也可以在其他地方修改之后，覆盖该视图中原来的文件。

6.4.4CheckIn

修改后，要将新改动的内容加入到CC中，这就需要CheckIn。如果一次修改了许多文件，为了防止漏掉一些修改的内容，可以使用快捷菜单中的FindCheckouts功能，这样可以找出所有的被CheckOut的文件，选中这些文件，然后选择CheckIn，要取消这次修改，也可以选择UndoCheckout，当然也可以在View中选择某个或某些文件，从快捷菜单中选择CheckIn完成CheckIn操作。

6.5提交工作

当开发工作进行一定时间之后，要提交所作的改动，以便管理员及其他开发人员看到这些改动。  
这样，改动的部分就会加入到集成视图中，CC服务器也会得到改动的部分，管理员可以根据新的数据进行测试、生成新的基线、晋升基线、生成新的推荐基线等工作。  
当系统完成提交工作后，提交工作并没有真正的完成，CC会弹出对话框，请操作者确认：  
按照顺序操作：  
（1）点击OK，这时会打开一个集成视图窗口，以便开发者检验提交的数据是否正确，如果提交结果正常，关闭集成视图窗口；  
（2）如果确认提交的工作正常，点击Complete，系统将会把所作的改动提交到CC服务器中；如果认为提交的结果不正确，点击Close退出，继续修改、开发。  
（3）提示工作完成后，点击Close，完成提交工作。  
如果提交的过程中出现异常，CC会提示开发人员，按照提示操作就可以了。

6.6rebase

项目管理员在不断关注项目的开发进展情况时，如认为出现了一个比较好的版本，需要各个开发人员以其为基础继续开发时，管理人员就会建立一个基线，并将其设置为推荐基线，然后要求开发人员保持与推荐基线代表的内容一致，然后在该基线的基础上继续开发。  
这时开发人员需要执行rebase操作，使自己的开发视图与其他人保持一致，rebase也有多种方法，“提交工作”一节的图表示了其中的两种方式。

6.7异常处理

在开发复杂系统时，会产生开发人员的代码冲突问题，在提交工作时会出现一些问题，这时需要开发人员对提交的工作进行审查、选择正确的版本等。  
另外，还有其他可能会出现的一些异常，请参考帮助来处理。

6.8备份与还原

备份还原方法  
1．备份和还原VOB  
（1）备份VOB  
锁定VOB  
备份VOB的存储目录  
解锁VOB  
注意事项：遇到不能备份VOB的存储目录时，应在控制面板中关闭CC的服务。  
（2）还原VOB  
restore  
stopsandrestartsCC  
updatestheCCVOBregistry  
MergetheVOBdatabasesnapshotandVOBstoragedirectory  
Copiesthetemporarystoragedirectorytotargetlocation  
RunscheckvobtoresynchronizetheVOBdatabaseandstoragepools  
注意：不能还原网络镜像。  
步骤：step  
用高级管理用户登入安装CC的主机  
确认目标目录所在盘符有足够的空间来还原备份的VOB  
解锁VOB  
注意事项：遇到不能备份VOB的存储目录时，应在控制面板中关闭CC的服务。  
（2）还原VOB  
restore  
stopsandrestartsCC  
updatestheCCVOBregistry  
MergetheVOBdatabasesnapshotandVOBstoragedirectory  
Copiesthetemporarystoragedirectorytotargetlocation  
RunscheckvobtoresynchronizetheVOBdatabaseandstoragepools  
注意：不能还原网络镜像。  
步骤：step  
用高级管理用户登入安装CC的主机  
确认目标目录所在盘符有足够的空间来还原备份的VOB  
关闭处于工作状态的VIEW  
运行：  
vob\_restore/flex  
UNCnames(//host/share/rest-of-path)  
检查  
解锁  
“/vobstore/flex.vbs  
分析VOB和VIEW  
2．备份和还原VIEW  
一、备份步骤：  
（1）确定VIEW的存储目录使用CCAdministratorConsole工具或执行Lsview查看它的存储信息。获得View的存储路径。  
例如：  
c:/>cleartoollsview–longakp\_vu  
可以获得如下信息：  
viewonhost:Neptune  
viewserveraccesspath:  
/home/akp/views/akp.vwsc:/home/akp/view/akp.vws  
（2）将处于工作状态的VIEW设为只读  
cleartoolchview–readonlyakp\_vu  
（3）BackupView的存储目录  
（4）取消只读  
c:/cleartoolchview–readwriteakp\_vu  
（二）还原View  
（1）登入服务端主机  
（2）检查目标位置的剩余磁盘空间  
（3）转移目标位置的存储目录。更名或删除。  
例如：  
c:/home/akp/view>renakp.vwsakp.vws.old  
（4）还原备份文件  
%c:/>cleartoolrecoverview–tagakp\_vu  
6.9权限管理  
在域中分组管理。  
各用户加入不同的小组。  
不同小组对项目具有不同的权限。

7其他事项

ClearCase是一种先进的软件配置管理工具，功能强大、复杂，上述介绍只列出了其功能的九牛一毛，ClearCase的使用说明书超过了2000页。  
ClearCase还支持UNIX系统，混合平台开发等，除了支持图形界面以外，还支持命令行格式的操作。  
因此短短二十几页的介绍根本无法详述ClearCase所有功能，上述介绍只能起到入门的作用，真正熟练掌握、使用CC还需要多多实践、阅读使用手册。  
ClearCase提供的手册包括：  
ClearCase介绍  
ClearCase安装手册  
ClearCase软件开发手册  
ClearCase开发人员手册  
ClearCase管理手册  
ClearCase项目管理手册  
ClearCase参考手册  
等等。

# TUXEDO

## salt中加服务步骤

innersvc.wsdf

all.dep

wsloadcf  -y  all.dep

tmloadrepos   -i allsvc.mif    allsvc.repos

tmwsdlgen  -c  allsvc.wsdf   -o  allsvc.wsdl

## 清理共享内存命令

AIX系统中：ipcs  -mqs|grep   $USER|awk '{print  "ipcrm -"  $1$2}'|sh  -x (清理IPC资源)

Linux系统中：ipcs –m|grep $USER|awk ‘{print “ipcrm -m” $2}’|sh –x （清理共享内存）

## wsadmin

gws -i  GWWS1

## tuxedo的BBL启动失败的解决办法

a. 备份相应的事务日志 TLOG : cp TLOG TLOG\_Tuxedo\_bbl\_backup

b. 删除事务日志TLOG: rm TLOG

c. 使用tmadmin登录到tuxedo系统中重新建一个日志设备,方法如下:

  $tmadmin

  >crdl -b 1000 -z tuxedo配置文件ubb.cfg中TLOGDEVICE所配置的目录

  >crlog -m new\_tuxedo\_bbl

  >q

  $tmboot -y

  其中:

### crdl命令创建TLog的device

    -b 为日志设备大小 1000 block

    -z目录 ubb配置文件中\*MACHINES段的TLOGDEVICE保持一致

### crlog命令创建Tlog

    -m new\_tuxedo\_bbl: new\_tuxedo\_bbl是虚拟主机名

如果BBL启动还是失败，看看IPCKEY值配置是否与某个其他用户相同,且那个用户服务在运行.

## 同步与异步通信

**同步通讯**

**int tpcall((char \*)servicename, (char \*)bufptr, long length, (char \*\*)bufptr, (long \*)length, long flags)**

函数tpcall()有6个参数。如下：

 交易名

 请求数据缓冲

 请求数据缓冲的长度（仅缓冲类型为CARRY时需要）

 返回数据缓冲的地址。缓冲大小可以根据收到数据而改变。

 返回数据缓冲大小的地址

 标志量

标志量可以是以下值：

 TPNOTRAN   如有交易不调用

 TPNOBLOCK 如有阻塞不等待

 TPNOTIME          愿意等待直到超时

 TPSIGRSTRT       系统中断信号在TUXEDO完成后再发布

出错返回-1，错误原因如下：

TPEINVAL     参数错误

TPETRAN      交易相关错误

TPETIME      超时

ret = tpcall(“TOUPPER”,bufptr,len,&bufptr,&len,flags);

**ATMI提供2个异步通讯函数**

tpacall()和tpgetrply()

**int tpacall((char \*)service, (char \*)bufptr, long len, long flags);**

flag:TPNOBLOCK,TPSIGRSTRT,TPNOTIME,TPNOTRAN,TPNOCHANGE,TPGETANY

**int tpgetrply((int \*)handle, (char \*\*)bufptr, (long \*)len, long flags);**

flag:TPNOBLOCK,TPSIGRSTRT,TPNOTIME,TPNOTRAN

tpacall()有4个参数

²  交易名

²  请求数据缓冲

²  请求数据缓冲的长度（仅缓冲类型为CARRY时需要）

²  标志量（同tpcall()）

返回值是一个非负描述符（句柄），用于其后的tpgetrply()调用。如果tpacall()调用失败，句柄是-1，错误原因设置在tperrno，可能是以下值：

TPELIMIT  过多未处理的tpacall()

TPETIME   超时

tpacall()后总跟随一个tpgetrply()，有4个参数

²  变量地址，可以传入tpcall()所得句柄，或另设一个

²  返回数据缓冲的地址。缓冲大小可以根据收到数据而改变。

²  返回数据缓冲大小的地址

²  标志量

标志量的值可以是如下之一：

TPGETANY                     取第一个返回值，设句柄

TPNOCHANGE         取发送请求的匹配的类型的信息

参见tpcall()中的值

返回值0表示成功，-1表示失败，错误原因可能如下：

²  TPEINVAL         参数错误

²  TPEOTYPE              返回数据缓冲类型错误

²  TPETIME   超时

## 数据缓冲管理

(char \*)**tpalloc**((char \*)type, (char \*)subtype, long size)

(char \*)**tprealloc**((char \*)bufptr, long newsize)

void **tpfree**((char \*)bufptr)

## TPSVCINFO结构体

void tpservice(TPSVCINFO \*svcinfo);

只有一个参数，该参数是指向TPSVCINFO结构的指针(atmi.h)。该结构定义如下：

struct tpsvcinfo {

char name[32];   /\*service名(最大15个字符)\*/

long flags;      /\* client调用时指定的flags \*/

char \*d ata;      /\* 接收的数据地址 \*/

long len;        /\* 数据长度 \*/

int cd;         /\* 会话方式下的连接描述符 \*/

long appkey;     /\* 应用认证的key \*/

CLIENTID cltid;  /\* client ID \*/

};

## TUXEDO的ubb配置文件说明

简单介绍tuxedo的运行需要配置文件UBBCONFIG

tuxedo的运行需要配置文件UBBCONFIG，该文件由管理员编写，然后使用tmloadcf编译成二进制文件。它有以下几个部分组成：

1.RESOURCES

该部分包括整个应用系统的一些配置信息，必须存在而且是配置文件的第一部分。有如下几个参数：

(1).IPCKEY(必需)

用于指定tuxedo分配和存取IPC资源的key，范围是32,768到262,143，该值在系统是唯一的，不能和别的应用同时使用这个值。

(2).MASTER(必需)

指定系统的主节点，主节点控制着整个应用系统的启动和管理。它的值是一个Logical Machine Identifier(LMID)，LMID在后面的MACHINES定义。当主节点是集群时，需要指定两个LMID。

(3).UID、GID、PERM

指定系统的存取权限。UID和GID是tuxedo管理员的用户id和组id，PERM是一个八进制数指定读写权限。

(4).MAXACCESSERS、MAXSERVERS、MAXSERVICES

MAXACCESSERS：指定能连接到tuxedo一个节点的最大进程数。缺省是50，这包括所有的client、server和管理进程BBL等。增加该值将会消耗更多的系统信号量IPC资源。

MAXSERVERS：整个应用系统中最大的server进程数。包括管理server进程，缺省是50。提高该值需要共享内存资源。

MAXSERVICES：整个系统可以发布的service的最大数量。缺省是100。提高该值需要更多的共享内存资源。一个service可以被多个server发布，此时，它的数量要算多份。

(5).MODEL(必需)

有两个可选值：

SHM：应用运行在单机或多机下，使用全局共享内存；

MP：不使用共享内存，运行在网络多机环境下。

(6).OPTIONS

有两个可选值：

LAN：支持网络；

MIGRATE：支持server集群。

可同时指定这两个值，用逗号分隔。

(7).LDBAL

是否打开load balancing(负载均衡)功能。可设置为Y或N，缺省为N。

(8).NOTIFY

设置client接收unsolicited message(请求消息)的方法。有如下三个可选值：

IGNORE：client忽略此类消息；

DIPIN：当client调用tpchkunsol或其他ATML调用时接收此类消息；

SIGNAL：通过系统产生信号通知client接收消息。有两个信号可以产生：SIGUSR1和SIGUSR2，缺省是SIGUSR2。若要指定为SIGUSR1则需设置下面的USIGNAL参数。

缺省是DIPIN。

(9).USIGNAL

设置系统产生的信号是SIGUSR1还是SIGUSR2。

(10).SCANUNIT、SANITYSCAN、BLOCKTIME

管理server(BBL)定期对所属机器上的servers进行正常检测。下面是有关的设置。

SCANUNIT：指定检测粒度。单位是秒，值必须是5的倍数。缺省是10。

SANITYSCAN：在两次检测之前间隔几个SCANUNIT。缺省是12(120秒)。SANITYSCAN\*SCANUNIT不能大于300。

BLOCKTIME：一个消息超时前可阻塞的SCANUNIT。缺省是6(60秒)。BLOCKTIME\*SCANUNIT不能超过32767。

2.MACHINES

该部分包括整个应用系统中所有计算机的信息。在配置文件中必须存在，而且是第二部分。

每个主机有如下的属性：

(1).主机名

使用命令”uname -n”获得的名称。

(2).LMID(必需)

主机的逻辑id：logical identifier of the machine(LMID)。

(3).TUXCONFIG(必需)

配置文件所在的绝对路径(最大64字符)。

(4).TUXDIR(必需)

Tuxedo的安装绝对目录(最大64字符)。

(5).APPDIR(必需)

应用server所在的绝对目录(最大64字符)。

(6).ENVFILE

环境文件所在绝对路径(最大64字符)。

(7).ULOGPFX

用户日志文件的绝对路径和前缀。

其他

UID、GID、PERM、MAXACCESSERS、MAXCONV可以在此定义而覆盖RESOURCES部分对该计算机的定义。

3.GROUPS

该部分包括所有server组的定义。每台机器必须至少定义一个组。每个组有如下属性：

(1).组名

(2).GRPNO(必需)

组名对应的组号，它必须是唯一的数字。

(3).LMID(必须)

该组属于的逻辑主机id。

4.SERVERS

该部分包括所有server的信息。每个条目描述一个server。Server有如下属性：

(1).server名

server可执行程序名(用buildserver编译)。

(2).SRVGRP(必需)

该server所属的group id(在GROUPS定义)。

(3).SRVID(必需)

每个server在group里定义一个唯一的id号。

对于同一个可执行程序可以在SERVERS中有多个条目，即server名不是唯一的。

(4).CLOPT

指定传递给程序的命令行选项。命令行串由”--"分隔，之前的部分由标准main()处理，后面的部分(有的话)由tpsvrinit()处理，用户自定义的选项放在此处。

缺省CLOPT=“-A”，表示发布该server所有的services。

部分选项如下：

-e filename：重定向标准出错文件(缺省是$APPDIR/stderr)；

-o filename：重定向标准输出文件(缺省是$APPDIR/stdout)；

-p [L][low\_water][,[terminate\_time]][:[high\_water][,create\_time]]：允许server的动态伸缩；

-s services：发布特定的services(语法同buildserver)。该部分优先于buildserver；

-r：记录services的统计信息和标准错误。

(5).ENVFILE

每个server也可指定环境文件，它与MACHINES定义的ENVFILE都起作用，相同的变量以server的为主。

(6).RQADDR、RQPERM、REPLYQ、RPPERM

RQADDR：起用MSSQ(Multiple Server Single Queue)集合。指定一个名字，集合中其他server用同样的名字。

RQPERM：指定队列的访问权限(如0666)。

REPLYQ：属于MSSQ中server的service从队列中取消息的话，它必须回应该消息，这通过指定REPLYQ=Y完成。缺省REPLYQ=N。

RPPERM：当REPLYQ=Y时，指定相关权限。

(7).CONV

如果server是一个会话server，则该属性必须指定为Y。缺省是N，表示它不是一个会话server。

(8).SEQUENCE

指定server的启动顺序，范围是1-9,999。该值小的先于值大的启动；指定该值的server比没有指定的server先启动；如果server都没有指定该选项，则按定义的顺序启动。

server终止时按照启动的反序停止。

(9).MIN

指定启动server的最小实例数。缺省为1。

(10).MAX

指定server能启动的最大实例数。缺省等于MIN。

(11).RESTART、RCMD、MAXGEN、GRACE

RESTART：指定当server死后是否重新启动它。当RESTART=Y时，下面的选项有用。

RCMD：指定重启时同时执行的命令，它必须是一个可执行文件。

MAXGEN：在GRACE时间范围内能重启的次数(MAXGEN-1)。如果GRACE=0则表示重启没有限制。

GRACE：值必须大于等于0，小于等于2,147,483,647秒(大概70年)。缺省是24小时。

5.SERVICES

该部分包括应用中的所有service的信息。每个service一个条目。有如下属性：

(1).LOAD

当LDBAL为Y时，指定该service的负载因子。缺省是50。

(2).PRIO

指定service的优先级。缺省是50。值越大级别越高。

(3).SRVGRP

(4).BUFTYPE

## TUXEDO基本命令

Tuxedo基本命令：

服务启动：tmboot –y  如果只需要启动单个则tmboot –S servername

服务停止：tmshutdown –y  同样tmshutdown –S servername

tmadmin –v  查看 tuxedo 版本及license信息

tmloadcf -y ubb\_file  编译文本格式的ubb文件到二进制的ubb文件

tmloadcf -c ubb\_file   测试IPC资源最小需求

tmadmin>bbparms (bbp)    显示BB（Bulletin Board）的关键参数配置

tmadmin>bbstats (bbs)      显示当前BB（Bulletin Board）的统计信息

tmadmin>printclient (pclt)   显示当前连接tuxedo的客户端信息

tmadmin>printserver (psr)   显示 server 的信息

tmadmin>printservice (psc)  显示 service 的信息

tmadmin>printtrans (pt)     显示当前事务的信息

tmadmin>printqueue (pq)    显示请求队列的信息

和Unix shell命令结合使用，例如：

echo psr| tmadmin | grep –v IDLE    取得在用的服务

echo psr| tmadmin | grep –v IDLE|wc -l

echo psr| tmadmin | sort +4

echo psc| tmadmin | sort +6

echo pq | tmadmin | sort +4

echo pclt | tmadmin | grep –v WSH|grep –v tmadmin|wc –l

还可以通过script的方式，对于命令结果进行更复杂的处理

tuxedo server和service区别

server是一些你写的应用，比如你的服务端可执行程序，service是可执行程序里所包含的一种服务，也就是一个函数。

其实简单点理解：SERVER就是集成了几个SERVICE的可执行程序。就好象一个服务器上可以安装多个应用服务一样。SERVICE是商业逻辑层面的。

从客户端的角度,他们只用知道SERVICE name和参数就可以了,不必关心是放在哪个server里面.但是从服务端的角度两个都要关心,

因为server是你的程序,你要编辑,编译,运行它,这个在运行的时候是一个进程,很多TUXEDO的管理是基于这个的;SERVICE是具体的服务,

server是一个进程，它接收client或其他server的请求和对它们发送应答信息。它包含一个或如果若干个services。

service是一个server程序的函数，它根据商业逻辑的需要执行一个特定的任务。

## tuxedo配置及服务编译

### tuxedo环境变量配置文件tux.env内容

export TUXDIR=/usr/bea/tuxedo10.0

export APPDIR=$APPHOME/bin/tux

export TUXCONFIG=$APPHOME/etc/tuxconfig

export BDMCONFIG=$APPHOME/etc/bdmconfig

export JAVA\_HOME=$TUXDIR/jre

export JVMLIBS=$JAVA\_HOME/jre/bin/j9vm: $JAVA\_HOME/jre/bin

export PATH=$TUXDIR/bin: $JAVA\_HOME/bin:$PATH

export COBCPY=:$TUXDIR/cobininclude

export COBOPT=”-C ANS85 -C ALIGN=8 -C NOIMBCOMP -C TRUNC=ANSI -C OSEXT=cbl”

export SHLIB\_PATH=$TUXDIR/lib:$JVMLIBS:$SHLIB\_PATH

export LIBPATH=$TUXDIR/lib:$JVMLIBS:$LIBPATH

export LD\_LIBRARY\_PATH=$TUXDIR/lib:$JVMLIBS:$LD\_LIBRARY\_PATH

export WEBJAVADIR=$TUXDIR/udataobj/webgui/java

#view文件名(多个文件用“，”分割)

export VIEWFILES=view\_txnmsgdef.V,view\_mgmmsgdev.V

export VIEWDIR=$APPHOME/bin/tux

######SALT环境变量#######

#saltconfig生成目录

export SALTCONFIG=$APPHOME/etc/saltconfig

#fml32文件目录

export FLDTBLDIR32=$APPHOME/salt

#fml32文件名(多个文件用“，”分割)

export FIELDTBLS32=userMsg.fml32,BillTransNotifyService.fml32,JFSmsReciveConfirmWS.fml32

#.xsd文件目录

export XSDDIR=$APPHOME/salt

#.xsd文件名(多个文件用“，”分割)

export XSDFILES=userMsg\_1.xsd,BillTransNotifyService\_1.xsd

### tuxedo配置文件ubbconfig

\*RESOURCES

IPCKEY 88899

DOMAINID TDOM

MASTER PriMachine,SecMachine

MAXACCESSERS 9000

MAXSERVERS 2800

MAXSERVICES 32760

MODEL MP

OPTIONS MIGRATE,LAN

NOTIFY DIPIN

LDBAL Y

MAXGTT 2048

SCANUNIT 20

SANITYSCAN 6

BBLQUERY 6

DBBLWAIT 1

BLOCKTIME 3

\*MACHINES

P570\_N\_1 LMID=PriMachine

TUXDIR=”/usr/local/bea/tuxedo10.0”

APPDIR=”/chk/usr/ch\_txn02/chacc/bin/tux”

TUXCONFIG=”/chk/usr/ch\_txn02/ch\_acc/etc/tuxconfig”

ULOGPFX=”/chk/usr/ch\_txn02/ch\_acc/log/tux/ULOG”

TLOGDEVICE=”/chk/usr/ch\_txn02/ch\_acc\_TLOG”

TLOGNAME=”TLOG”

TYPE=”AIX”

NETLOAD=20

MAXWSCLIENTS=320

UID=257

GID=203

P570\_N\_2 LMID=SecMachine

TUXDIR=”/usr/local/bea/tuxedo10.0”

APPDIR=”/chk/usr/ch\_txn02/chacc/bin/tux”

TUXCONFIG=”/chk/usr/ch\_txn02/ch\_acc/etc/tuxconfig”

ULOGPFX=”/chk/usr/ch\_txn02/ch\_acc/log/tux/ULOG”

TLOGDEVICE=”/chk/usr/ch\_txn02/ch\_acc\_TLOG”

TLOGNAME=”TLOG”

TYPE=”AIX”

NETLOAD=20

MAXWSCLIENTS=320

UID=255

GID=202

\*NETWORK

PriMachine NADDR=”//172.17.248.180:46006”

NLSADDR=”//172.17.248.180:46007”

SecMachine NADDR=”//172.17.248.181:46006”

NLSADDR=”//172.17.248.181:46007”

\*GROUPS

GROUPPRI1

LMID=PriMachine GRPNO=1

GROUPPRI2

LMID=SecMachine GRPNO=2

\*SERVERS

TMMETADATA SRVGRP=GROUPPRI1 SRVID=1 CLOPT=”-A -- -f /chk/usr/ch\_txn02/ch\_acc/salt/allsvc.repos”

GWWS SRVGRP=GROUPPRI1 SRVID=2 CLOPT=”-A -- -IGWWS1“

GWWS SRVGRP=GROUPPRI1 SRVID=3 CLOPT=”-A -- -IGWWS2“

TMMETADATA SRVGRP=GROUPSEC1 SRVID=4 CLOPT=”-A -- -f /chk/usr/ch\_txn02/ch\_acc/salt/allsvc.repos”

GWWS SRVGRP= GROUPSEC1 SRVID=5 CLOPT=”-A -- -IGWWS3“

GWWS SRVGRP= GROUPSEC1 SRVID=6 CLOPT=”-A -- -IGWWS4“

##PPCHACC01##

cswttxnsvr SRVGRP=GROUPPRI1 SRVID=7 RESTART=Y MAXGEN=5 GRACE=10 MIN=1

MAX=1 RQADDR=”Q1” REPLY=Y CLOPT=”-A -s CSWTTXNSVC\_01:CSWTTXNSVC --g1001 ”

cswtcommsvr SRVGRP= GROUPPRI1 SRVID=8 RESTART=Y MAXGEN=20 GRACE=3600

CLOPT=”-A –s CSWTCOMMSVC\_A:CSWTCOMMSVC -- -x172.17.178.231 –y33123 -zA”

##PPCHACC02##

cswttxnsvr SRVGRP= GROUPSEC1 SRVID=1 RESTART=Y MAXGEN=5 GRACE=10 MIN=1

MAX=1 RQADDR=”Q2” REPLY=Y CLOPT=”-A -s CSWTTXNSVC\_01:CSWTTXNSVC --g1001 ”

cswtcommsvr SRVGRP= GROUPSEC1 SRVID=2 RESTART=Y MAXGEN=20 GRACE=3600

CLOPT=”-A –s CSWTCOMMSVC\_A:CSWTCOMMSVC -- -x172.17.178.231 –y33123 -zA”

### 编译ubbconfig配置文件

tmloadcf -y ubbconfig

### 编译tuxedo服务程序

如果源程序为usracctsvr.c

PRGTARG=usracctsvr

OBJS= usracctsvr.o

TUXSVCS= -s USRACCTSVC

LIBS=

buildserver -o $ (PRGTARG) -f $(OBJS) $TUXSVCS -f $(LIBS)

# MQ中间件

## 查看MQ错误码

mqrc 错误码

## 查看MQ版本

dspmqver

## 查看队列管理器

dspmq

命令在mqm的bin目录下

## 基础概念

对于MQ，我们需要知道4个名词：队列管理器、队列、消息、通道；对于编程设计人员,通常更关心消息和队列,对于维护管理人员,通常 会更关心队列管理器和通道。

如果我们把队列管理器比作是数据库,那么队列就是其中的一张表,消息就是表中的一条记录。

### 1.队列：

我们可以简单地把队列看成一个容器，用于存放消息。

### 2.队列管理器：

队列管理器构建了独立的 MQ 的运行环境，它是消息队列的管理者，用来维护和管理消息队列。

### 3.消息：

MQ中的最小对象；默认情况下，消息缺省可以达到 4MB。消息可以分成持久消息和非持久消息。所谓“持久”的 意思,就是在MQ 队列管理器重启动后,消息是否仍然能保持。持久的消息写入或读出队列的同时会在 Log 中记录,所以性能上比非持久消息差不少。

### 4.通道：

通道则是两个队列管理器之间的一种单向的点对点的通信连接， 消息在通道中只能单向流动。队列管理器之间的通信是通过配置通道来实现 的,通道两侧的队列管理器对这个通道的相关参数应该能对应起来。在通道上可以配置不同的通信协议,这样就使得编程接口与通信协议无关。通道两端的 配置必须匹配,且名字相同,否则无法连通。

### 5.runmqsc：

命令行交互界面管理工具；作为维护人员的我们，与MQ打交道有两种方式，一种是通过MQ提供的二进制命令工具（在mq安装目录的bin目录下），另一种方式则是通过命令行交互管理工具；这二者在功能上有很多是重合的，但并非完全可替代；

RUNMQSC是一个通用的 MQ 对象管理工具，使用 MQSC命令集可以对 MQ 对象进行

全方位的管理，也是各种管理方式最直接、最全面的一种。

RUNMQSC 运行的命令集称为 MQSC (MQ Script Command)"；在 RUNMQSC 中大小写无关,所有的命令会先转换成全大写再提交执行。所以如果要 表示大小相关的字串,比如对象名,则用引号将字串包住。

输入以下命令启动MQSC命令：runmqsc [queueManagerName]

ok，概念介绍完了，下面就是通过示例来学习如何使用他们；

首先，我们需要创建队列管理，用来存放队列。有了队列管理器后，就可以创建队列；队列创建了，就可以用来放入和取出消息；

## mq是如何实现通讯的？

mq的通讯方式有两种，通俗的说就是mq之间进行通讯，开发的程序和mq之间的通讯。

mq之间进行通讯：通过发送接收通道建立tcp连接进行消息传输，称为server对server

开发的程序和mq之间的通讯：通过服务器连接通道进行传输，client对server

## 如何配置两台mq使之相互进行通讯？

首先要规划好两个队列管理器之间使用的ip和端口，假设我们使用

ip 端口

192.168.0.1 1414

192.168.0.2 1415

### 第一步 建立队列管理器

crtmqm -lc -lf 100 -lp 3 -ls 3 QM1

解释下：

-lc 是采用循环日志

-lf 是每块日志的大小，4k为单位的，100就是100\*4k

-lp 是主逻辑日志的数量

-ls 是辅逻辑日志的数量

QM1 是队列管理器名称

### 第二步 启动队列管理器

strmqm QM1

### 第三步 定义队列管理器中的队列和通道等

先运行runmqsc QM1首先要保证运行该命令的用户属于mqm组

运行完后进入mq命令窗口

定义本地队列 def ql(QL1)

先解释什么是本地队列，然后解释命令的含义（以下同）

本地队列是存储信息的盒子，用户可以从本地队列里取消息，对方发送消息的目的地也是本地

队列。

def是 define的缩写，mq支持一些命令的缩写。

ql是queue local的缩写，表示本地队列，括号内是本地队列名

定义远程队列 def qr(QR1) rname(QL2) rqname(QM2) xmitq(QT1)

远程队列是相对于本地队列的，当用户希望往另一个队列管理器发消息的时候，配置好远程队

列，用户直接放消息到该队列就可以，mq会传输到另一方的本地队列中。

以上面的例子说明，当我们把消息放入该远程队列后，消息会传输到QM2队列管理器中的QL2队

列中。

qr queue remote的缩写

rname 指定的远程队列管理器上的队列名

rqname 远程队列管理器

xmitq 所要用的传输队列

定义传输队列 def ql(QT1) usage(xmitq) trigger trigtype(first) initq

(system.channel.initq) trigdata(QM1.QM2)

传输队列是传输的介质，消息是通过传输队列进行传输的。

usage 用途xmitq是传输队列

trigger 消息触发开关

trigtype 触发类型第一条消息触发

initq 初始队列

trigdata 触发数据

定义发送通道 def chl(QM1.QM2) chltype(sdr) conname('192.168.0.2(1415)') trptype

(tcp) xmitq(QT1)

发送通道就相当于建立一个tcp的连接

chl channel的缩写

chltype 通道类型sdr是发送通道

conname 连接名包括对方的ip和端口

trptype 通讯类型tcp通讯

xmitq 使用的传输队列

定义接受通道 def chl(QM2.QM1)

接收通道是被动的，只定义名字就可以。大家注意，接收通道的名字一定要和发送通道名一致

，他们是靠名字来匹配。

### 第四步 配置监听器

是对方mq管理器来探测，本地要给对方一个回应，监听器就是起这个作用的。

如果是5.3版本 只能在命令行里运行 runmqlsr -m QM1 -t tcp -p 1414

如果是6.0版本 可以runmqsc QM1里运行 def listener(LSR.QM1) trptype(tcp) port(1414)

control(qmgr)

解释下 trptype 监听类型

port 监听端口

control 监听控制，如果是qmgr则在队列管理器启动的时候监听也自动启动。

### 第五步 配置另外一个队列管理器

简单的说一下，和上面的差不多，只不过名字不一样。大家自己尝试下：）

写的手累了,下次补充!

继续

上面我们说完了如何建队列管理器，接下来我们说说建完以后如何测试两边是不是正常传输，

我们可以用命令行方式向远程队列中放入测试消息，以aix为例，

用/usr/mqm/samp/bin/amqsput命令就可以放入消息，格式为：

amqsput QR1 QM1

解释下：QR1是你要放入的队列名，QM1是你要放入的队列管理器名。

输入以上命令后就可以写入消息了，一下回车就是发送一条消息，两次回车就是退出。

用/usr/mqm/samp/bin/amqsget命令就可以取消息，格式为

amqsget QL2 QM2

解释下：QL1是你要取消息的队列名，QM2是你要取消息的队列管理器名。

输入完命令后就会显示所有消息。

不过要注意一点，命令行方式的输入和取消息有字节限制。

如果只是简单的浏览一下消息可以使用

/usr/mqm/samp/bin/amqsbcg命令

格式和取消息一样，但是该命令不会把消息取出来，运行完该命令后消息还是保存在队列中。

正常情况下消息的传输流程（只说正常的，排错一会再说）

QR1 -> QT1 -> QL2

消息被放入到远程队列中，远程队列通过传输队列传输，最后传输到QM2中的本地队列。

## 出现问题，我们怎么办？

### 1 不能放入消息。

一般这种情况应该大部分是远程队列中的传输队列那个参数配置的不正确。

还有可能是队列的允许放入这个参数设置成了禁止。基本上就这两种情况。

### 2 发送通道和接收通道的状态不是running

首先说明，如果长时间没有消息传输，通道的状态会变成不活动状态，这是正常现象。

如果你手动启动通道后，通道状态还不是running，那先查看错误日志（两边的队列管理器都要查看）

/var/mqm/qmgrs/QM1/errors中的错误日志，通常编号01的日志是最新日志。

常见情况是网络不通导致的通道不通！所以首先要保证网络是正常的，我们可以同过telnet对方的IP加监听端口的方法来查看是不是正常。

telnet 192.168.0.2 1415

再有的情况是两边的配置属性有问题，如两边发送和接收通道名不一致，发送通道的连接名配置错误，发送通道中的传输队列配置错误。

我们也可以执行mq中的一个命令来查看通道是不是正常

runmqsc QM1

ping chl(QM1.QM2)

ping操作来查看两边的通道是不是正常，如果正常会返回ping完成。

### 3 放入的消息没有到QM2的队列中

注意：消息一定要放入远程队列中，如果放入传输队列中消息会被放入死信队列中。（上面忘记定义死信队列了，晕）

再有看看远程队列中的属性是不是配置错误，如rname，rqname，xmitq等属性。

也有可能是发送接收队列的消息序列号不一致。如果不一致做一下reset操作。

还有可能是上一批消息没有提交。可以做一个resolve操作。

也是要先看错误日志

### 4 消息到达QM2队列QL2中，但是取不出来

QL2的允许取出属性是不是被禁止了。

这样再查看以下QM2到QM1的传输是不是正常。都正常就OK了。

## 创建队列管理器

eg：创建一个名为QM\_MEMDB的队列管理器：

crtmqm -q QM\_MEMDB

>Directory '/var/mqm/qmgrs/QM\_MEMDB' created.

所有创建的队列管理器在/var/mqm/qmgrs目录下都会有对应文件名的子目录生成；

（至于为什么是/var/mqm，这个配置在安装目录下的mq配置文件中配置samp/mqs.ini）

从这里我们可以确定，队列管理器的创建是一个持久化的操作，当MQ服务器停止后再启动时，队列管理器都还是存在的；

## 删除队列管理器

删除队列管理器ECIS\_QM

dltmqm ECIS\_QM

## 查看队列管理器

dspmq

## 运行队列管理器

runmqsc 队列管理器名

## 查看队列管理器中的所有队列

runmqsc 队列管理器名

display queue(\*)

display queue(队列名)

查看队列管理器的监听器

runmqsc 队列管理器名

display listener(\*)

display listener(监听器名)

## 启动队列管理器

启动名为QM\_MMEDB的队列管理器：

/opt/mqm/bin$./strmqm QM\_MEMDB

>WebSphere MQ queue manager 'QM\_MEMDB' started using V7.5.0.0.

（停止mq队列管理器：endmqm）

## 停止队列管理器

endmqm 管理器名

endmqm命令需要等待一段响应时间，一般情况下使用-i参数，已正常停止通道，如果多次停止无法生效，则使用-p参数强制停止

endmqm –p 队列管理器名

## 查看队列的深度

|  |
| --- |
| runmqs 队列管理器名  display qlocal(\*) curdepth |

## 设置本地队列Q 的最大深度为5

ALTER QLOCAL (Q) MAXDEPTH(5)

## 重新创建本地队列Q

如果Q已经存在，则将其全部属性重置为缺省属性

DEFINE QLOCAL (Q) REPLACE

## 创建远程队列

DEFINE QREMOTE(QRNAME) RNAME(AAA) RQMNAME(QMGRNAME) XMITQ(QTNAME)

## 显示所有远程队列

display qremote (\*)

## 定义死信队列

DEFINE QLOCAL(QUEUE) DEFPSIST(YES) REPLACE

## 设定队列管理器的死信队列

ALTER QMGR DEADQ(QUEUE)

## 设置MaxChannels和MaxActiveChannels属性（最大连接数）

MaxChannels和MaxActiveChannels分别代表队列管理器允许配置的通道的最大个数和允许同时运行的通道的个数，MaxChannels的缺省值是100，MaxActiveChannels的缺省值与MaxChannels相同。如果您的并发通道连接个数超过了100，您需要修改这两个参数。这对于大并发的Client/Server间通讯尤为重要。

在unix平台，修改qm.ini文件，添加如下所示（路径：/var/mqm/qmgrs/HQ\_SVR）：

Channels:

MaxChannels = 3000   （最大通道数）

MaxActiveChannels = 3000   （最大活动通道数）

PipeLineLength=2

AdoptNewMCA=ALL

然后重启MQ

## 定义持久性队列

DEFINE QLOCAL(QNAME) DEFPSIST (YES) REPLACE

## 传输队列管理

### 定义传输队列

DEFINE QLOCAL(Q\_TRANSFER) USAGE(XMITQ)

注：

Q\_TRANSFER:传输队列名

### 定义一个传输队列并带触发器

DEFINE QLOCAL(Q\_TRANSFER) USAGE(XMITQ) DEFPSIST(YES)  INITQ(SYSTEM.CHANNEL.INITQ) TRIGDATA(DP\_RCHANNEL) REPLACE

注：

参数1：传输队列名称(Q\_TRANSFER)

参数5：触发器数据(DP\_RCHANNEL)，也就是触发后要启动的通道。

### 在传输队列上定义触发器

alter qlocal(DPCIS\_Q\_TRANSFER) TRIGDATA('DP\_RCHANNEL') TRIGTYPE(every) TRIGGER initq('SYSTEM.CHANNEL.INITQ')

DPCIS\_Q\_TRANSFER:传输队列

DP\_RCHANNEL：触发器数据,即：发送通道，触发时启动该通道

## 通道管理

### 创建接收方通道

DEFINE CHANNEL(SDR\_ TEST)  CHLTYPE(RCVR)  REPLACE

define channel(Q\_R\_CHANNEL2) chltype(RCVR) replace

注：

参数1：接收方通道名称

replace:表示有该通道了替换

### 创建发送方通道

DEFINE CHANNEL(SDRNAME)  CHLTYPE(SDR)  CONNAME (‘100.100.100.215(1418)’)  XMITQ(QTNAME)  REPLACE

其中CHLTYPE可以是：SDR、SVR、RCVR、RQSTR、CLNTCONN、SVRCONN、CLUSSDR和CLUSRCVR。

define channel(DP\_RCHANNEL) chltype(SDR) conname('10.101.2.3 (1414)') xmitq(Q\_TRANSFER) replace

注：

通道名称：DP\_RCHANNEL (发送方通道的名称)

传输队列：Q\_TRANSFER  (发送方传输队列的名称)

### 启动通道

start channel(QM\_ORANGE.QM\_APPLE)

### 查看通道状态

dis chs(\*)

## 侦听器管理

### 定义侦听器

DEFINE LISTENER(TCPLS1) TRPTYPE(TCP) PORT(1414)  REPLACE

### 启动侦听器

start listener(TCPLS1)

### 显示侦听器信息

display listener(tcpls1)

## 创建触发

一般设置MQ触发器的目的有两种, 一是自动启动发送端通道, 二是监视队列消息, 一旦发现新的消息, 则利用触发器启动相应的处理进程

如果是利用触发器自动启动发送端通道, 使用方法1,

如果是利用触发器启动用户进程, 使用方法2

方法1

* A 在传输通道上设置触发器, 打开触发器控制, 类型为"第一个"
* B 初始队列为SYSTEM.CHANNEL.INITQ, 该队列为MQ专用的通道启动队列, 不需要手工启动其触发监视器
* C 触发器数据为发送端通道名称, 例如 QAG.50
* D 在发送端通道不活动的情况下, 在传输通道放入消息[**测试**](http://lib.csdn.net/base/softwaretest)

方法2

* A 创建一个触发启动队列, 其模板为SYSTEM.DEFAULT.INITIATION.QUEUE

runmqsc

define qlocal (MQ.TRIGER.INIQUEUE) like (SYSTEM.DEFAULT.INITIATION.QUEUE)

* B 定义进程 所谓进程就是触发器要触发的程序本例中进程名称：TRIGER.PROGRAM 应用程序标识为："Notepad.exe"
* C 在队列上设置触发器, 一般为本地队列, 打开触发器控制, 类型根据自己的需要选择"第一个", "每个" 还是 "根据消息数量"
* D 初始队列选择A中创建的触发启动队列, 例如MQ.TRIGER.INIQUEUE, 进程名称设置为B中定义的进程名称, 例如TRIGER.PROGRAM
* E 启动触发监视器 runmqtrm -m QAGWY -q MQ.TRIGER.INIQUEUE
* F 在本地队列放入消息进行测试

方法2也可以完成方法1的工作, 只需要将进程定义中的应用程序标识改为 runmqchl -m QAGWY -c QAG.50 就可以了, 但MQ中提供了专用的通道启动队列,

而该队列不需要手工启动触发监视器, 因此方法1比方法2的步骤要少, 配置也简单, 所以在利用触发器自动启动发送端通道的情况下, 还是方法1更好.

## 创建传输队列

define qlocal(CLV\_HQ\_TRAN) usage(xmitq) INITQ(SYSTEM.CHANNEL.INITQ) TRIGDATA(REV\_CLV) TRIGTYPE(EVERY) TRIGGER REPLACE

## 创建接收端通道C

DEFINE CHANNEL(C) CHLTYPE(RCVR)

## 创建发送方通道C ,连接对方的IP为10.10.10.10  端口为1414 通道连接的传输队列为XQ

DEFINE CHANNEL(C) CHLTYPE(SDR) CONNAME(’10.10.10.10 (1414)’) XMITQ(XQ)

## 创建请求方通道

DEFINE CHANNEL(D) CHLTYPE(RQSTR) CONNAME(’10.10.10.10 (1414)’)

## 创建服务器通道

DEFINE CHANNEL (D) CHLTYPE(SVR) xmitq (XQ1)

## 创建服务器连接通道

DEFINE CHANNEL(E) CHLTYPE(SVRCONN)  REPLACE

## 清理队列数据

clear ql(队列名)

## 查看CCSID

CCSID是IBM用来标识字符序列的标识代码。

display qmgr all

## 修改CCSID

ALTER QMGR [FORCE] CCSID(5488)

## 创建队列

队列及消息属于MQ的对象，MQ的对象管理一般使用mqsc命令交互工具来管理；

对于创建队列的操作，最好先写好放在一个文件里，然后调用；

eg,我们在队列管理器QM\_MEMDB中定义一个TEST1队列：

vi define\_memdb.tst：

define qlocal (TEST1) DEFPSIST(YES) MAXDEPTH(150000)

然后调用mqsc来创建：

/var/mqm/sh$runmqsc QM\_MEMDB < define\_memdb.tst > out.txt

创建的结果信息输出到out.txt:

1 : define qlocal (TEST1) DEFPSIST(YES) MAXDEPTH(150000)

AMQ8006: WebSphere MQ queue created.

One MQSC command read.

No commands have a syntax error.

All valid MQSC commands were processed.

## 向队列中放入消息

基本队列操作命令分成两类， 一类在 MQ Server 端运行， 它们是 amqsput、 amqsget、

amqsbcg。另一类在 MQ Client 端运行，它们是 amqsputc、amqsgetc、amqsbcgc

从Server端将消息放入队列的命令格式：

amqsput amqsput QueueName [QueueManagerName]

amqsput和 amqsputc 可以将消息放入队列中， 程序把之后的每一行标准输入作为一条独立的消息，读到 EOF 或空行时退出。注意，UNIX 上的 EOF 为 Ctrl+D。可以将标准输入重定向到文件。队列中每放入一条消息，队列深度增加一。

eg：

/var/mqm/sh$amqsput TEST1 QM\_MEMDB

查看队列中放入的消息

## amqsbcg 查看队列中消息

QM\_MEMDB中名为TEST1的队列：

amqsbcg TEST1 QM\_MEMDB

## 从队列中取出消息

命令格式：

amqsget 队列名 队列管理器名

amqsput 队列名 队列管理器名

amqsget 和 amqsgetc 可以将消息从队列中全部读出并显示。读空后再等待 15 秒，在这段时间内如果有新的消息到达会一并读出。强行中断该程序用 Ctrl+C 。amqsget 和 amqsgetc 执行后队列应该为空，即队列深度为零。

以上是单个服务器的单个队列管理器中完成的操作，如果需要跨队列管理器或跨机通信，我们需要MQ命令服务器和MQ监听器，当然，少不了建立通道；

命令格式：

amqsget 队列名 队列管理器名

amqsput 队列名 队列管理器名

## amqsget命令的目录

amqsget命令的目录一般在/opt/mqm/samp/bin下

## MQ命令服务器

WebSphere MQ 命令服务器是队列管理器的一个组件，用来对外来的命令消息进行解释和执行。在远程管理和编程管理的应用中，需要启动命令服务器。一个队列管理器最多只有一个命令服务器，缺省情况下在创建队列管理器时由系统一并创建。

启动与停止：

strmqcsv

endmqcsv

dspmqcsv

启动后执行下，观察状态，看到正在运行：

/opt/mqm/bin$dspmqcsv QM1

WebSphere MQ Command Server Status . . : Running

## MQ 监听器

WebSphere MQ 中监听器也是队列管理器的一个组件，用来监听外来的连接请求并相应

地做出反应。监听器通常需要先配置，然后才能运行，配置参数与监听器选择的通信协议有关。当然，也可以在第一次启动监听器时将配置参数传入，隐式地进行配置。一个队列管理器可以有多个监听器，分别应用于不同的通信协议或同一协议的不同参数。比如 TCP/IP 的不同端口。

通道的配置和建立下面单独拿出来讲解；

over。

## 定义监听器

DEFINE LISTENER(LSR\_4\_ECIS\_QM) TRPTYPE(TCP) PORT(1616) CONTROL(QMGR) REPLACE

## 启动监听器

START LISTENER(LSR\_4\_ECIS\_QM)

## 定义通道

DEFINE CHANNEL(ECIS.SVRCONN) CHLTYPE(SVRCONN) MCAUSER('mqm') REPLACE

## 启动通道

runmqchl –c ChlName –m ECIS\_QM

## 启动侦听

runmqlsr –t TCP –p 1616 –m ECIS\_QM

## 停止侦听

endmqlsr -m ECIS\_QM

## 查看监听器

display listener (LSR\_4\_ECIS\_QM)

## 定义队列

DEFINE QL(Q\_SVC2ADP\_4\_TELNET) REPLACE

DEFINE QL(Q\_SVC2ADP\_4\_JDBC) REPLACE

DEFINE QL(Q\_SVC2ADP\_4\_HTTP) REPLACE

DEFINE QL(Q\_SVC2ADP\_4\_SOCKET) REPLACE

## 删除队列

delete QL(Q\_SVC2ADP\_4\_TELNET)

## MQ通道配置

通道是用来连接两个队列管理器的；

在单个队列管理器内读写消息不需要建立通道；但在一个队列管理器中写入消息，而从另一个队列管理器中的队列取出消息，这就需要建立通道；

通道，对跨机和本机的队列管理器不做区分，对于两个队列管理器，不论是否分布在同一个机器上，配置方式都是一样的，所不同的就是ip的地址配置；

通道类型

通信双方的通道类型配对并不是可以随意排列组合的，共有六种。（详见《精通MQ》1.2.4 p24）

Sender/Receiver 是所有连接中最简单、最常用的一种。Sender 是通道主动方，也是

消息发送方。

Requester/Server 也是常用的一种连接方式。Requester 是通道主动方，但通道连接

后，它作为消息接收方，Server 是消息发送方。

Server/Receiver 与 Sender/Receiver 类似，Server 是消息的发送方，也是连接的主动

方。与 Sender 定义类似，Server 定义中必须指定 CONNAME 参数。

Sender/Receiver 通道是最常见的通道配置方式， Sender 作为通道的发送方也是通道连接的主动发起方，Receiver 作为通道的接收方也是通道连接的被动监听方。在 Receiver 端要配置并运行相应的监听器。

### 配置示例

以下以Sender/Receiver 作为示例：

在以下的配置脚本中，通道连接两个队列管理器 QM1 和 QM2。其中，QM1为 Sender，

QM2 为 Receiver。在 QM1 上配置了远程队列 QR 和传输队列 QX，其中 QR 指向队列管理器 QM2 上的本地队列 QL，且 QR 与 QX 对应，即凡是要放入 QR 队列的消息，在加上传输消息头后直接放入 QX 中等待发送。QM1 上配置 Sender 通道需要指定对方的通信参数 (IP地址和端口)，而这些参数必须与 QM2 上的监听器设置对应。Sender 通道与传输队列 QX 对应，表示凡是在 QX 中等待发送的消息最终都可以由该通道送出。双方通道必须同名。

在连接通道的时候，我们只需在 QM1 端启动通道 start channel (C)。

#### 1.建立队列管理器

首先，我们在10.6.159.147 建立两个队列管理器QM1、QM2，然后运行起来；

/var/mqm/sh$crtmqm QM1

/var/mqm/sh$crtmqm QM2

/var/mqm/sh$strmqm QM1

/var/mqm/sh$strmqm QM2

#### 2.建立队列和通道

##### 定义qm1的队列和通道创建脚本：

/var/mqm/sh$vi define\_qm1.tst

DEFINE QREMOTE (QR) RNAME (QL) RQMNAME (QM2) XMITQ (QX) REPLACE

DEFINE QLOCAL (QX) USAGE (XMITQ) REPLACE

DEFINE CHANNEL (C) CHLTYPE (SDR) TRPTYPE (TCP) CONNAME ('127.0.0.1 (1416)') XMITQ (QX) REPLACE

##### 创建qm1的队列和通道：

/var/mqm/sh$runmqsc QM1 < define\_qm1.tst > out

查看out文件，确认没有错误；

##### 定义qm2的队列和通道创建脚本：

/var/mqm/sh$vi define\_qm2.tst

DEFINE QLOCAL (QL) REPLACE

DEFINE CHANNEL (C) CHLTYPE (RCVR) TRPTYPE (TCP) REPLACE

创建qm2的队列和通道：

/var/mqm/sh$runmqsc QM2 < define\_qm2.tst > out

查看out文件，确认没有错误；

#### 3.在qm2运行监听器

在qm2上运行监听器，监听连接和发报请求；

/opt/mqm/bin$runmqlsr -m QM2 -t tcp -p 1416

（停止监听器的命令：endmqlsr -m QM2）

#### 4.运行通道

qm2的监听器不启动时，qm1上运行通道是无法启动的，因为无法建立起连接；

/opt/mqm/bin$runmqchl -c C -m QM1

或者通过mqsc来运行通道：

runmqsc QM1

start channel (C)

#### 5.发送测试报文

/var/mqm/sh$amqsput QR QM1

hello from qm1

#### 6.接收测试报文

/var/mqm/sh$amqsget QL QM2

>hello from qm1

一次全部接受回来

#### 7.不同的机器上

进一步，我们再另外一台机器上建立QM3队列管理器：

/var/mqm/sh$crtmqm QM3

/var/mqm/sh$strmqm QM3

队列的定义与qm2一样，创建：

/var/mqm/sh$runmqsc QM3 < define\_qm3.tst > out

运行监听器：

/opt/mqm/bin$runmqlsr -m QM3 -t tcp -p 1416

qm1上的队列需要修改下ip：

DEFINE QREMOTE (QR) RNAME (QL) RQMNAME (QM3) XMITQ (QX) REPLACE

DEFINE QLOCAL (QX) USAGE (XMITQ) REPLACE

DEFINE CHANNEL (C) CHLTYPE (SDR) TRPTYPE (TCP) CONNAME ('10.6.159.211 (1416)') XMITQ (QX) REPLACE

## 重置通道序列号

MQ通道如果双方长期未连接，或者有过切换动作时，或导致两边通信的序列号不一致，此时一般通道状态为PETRYING,无法正常通信，需要双方通道配合重置序列号

使用如下命令重置通道的序列号

|  |
| --- |
| runmqsc 通道名  stop chl(通道名)  reset chl(通道名)  start chl(通道名) |

注意：操作过程中，可以使用dis\_chs(通道名)或dis(\*)查看对应通道状态。且重置需要咋两端都进行，单方面的重置没有效果。

## 查看通道的状态

使用dis chs(通道名)或dis(\*)查看对应通道状态

## 逻辑定义

./JMSAdmin

DELETE QCF(T\_ECIS\_QCF)

DELETE Q(T\_ECIS\_Q\_SVC2ADP\_TELNET)

DELETE Q(T\_ECIS\_Q\_ADP2SVC\_TELNET)

DELETE Q(T\_ECIS\_Q\_SVC2ADP\_JDBC)

DELETE Q(T\_ECIS\_Q\_ADP2SVC\_JDBC)

DELETE Q(T\_ECIS\_Q\_SVC2ADP\_HTTP)

DELETE Q(T\_ECIS\_Q\_ADP2SVC\_HTTP)

DELETE Q(T\_ECIS\_Q\_SVC2ADP\_SOCKET)

DELETE Q(T\_ECIS\_Q\_ADP2SVC\_SOCKET)

DEFINE QCF(T\_ECIS\_QCF) DESC() TRAN(CLIENT) HOST(10.151.128.13) QMGR(T\_ECIS\_QM) CHAN(T\_ECIS.SVRCONN) PORT(2626) CCSID(5488)

DEFINE Q(T\_ECIS\_Q\_SVC2ADP\_TELNET) DESC() QMGR(T\_ECIS\_QM) QUEUE(Q\_SVC2ADP\_4\_TELNET) CCSID(5488)

DEFINE Q(T\_ECIS\_Q\_ADP2SVC\_TELNET) DESC() QMGR(T\_ECIS\_QM) QUEUE(Q\_ADP2SVC\_4\_TELNET) CCSID(5488)

DEFINE Q(T\_ECIS\_Q\_SVC2ADP\_JDBC) DESC() QMGR(T\_ECIS\_QM) QUEUE(Q\_SVC2ADP\_4\_JDBC) CCSID(5488)

DEFINE Q(T\_ECIS\_Q\_ADP2SVC\_JDBC) DESC() QMGR(T\_ECIS\_QM) QUEUE(Q\_ADP2SVC\_4\_JDBC) CCSID(5488)

DEFINE Q(T\_ECIS\_Q\_SVC2ADP\_HTTP) DESC() QMGR(T\_ECIS\_QM) QUEUE(Q\_SVC2ADP\_4\_HTTP) CCSID(5488)

DEFINE Q(T\_ECIS\_Q\_ADP2SVC\_HTTP) DESC() QMGR(T\_ECIS\_QM) QUEUE(Q\_ADP2SVC\_4\_HTTP) CCSID(5488)

DEFINE Q(T\_ECIS\_Q\_SVC2ADP\_SOCKET) DESC() QMGR(T\_ECIS\_QM) QUEUE(Q\_SVC2ADP\_4\_SOCKET) CCSID(5488)

DEFINE Q(T\_ECIS\_Q\_ADP2SVC\_SOCKET) DESC() QMGR(T\_ECIS\_QM) QUEUE(Q\_ADP2SVC\_4\_SOCKET) CCSID(5488)

# ESQL嵌入式sql

## whenever

在每条嵌入式SQL语句之后立即编写一条检查SQLCODE/SQLSTATE值的程序，是一件很繁琐的事情。为了简化错误处理，可以使用WHENEVER语句.

WHENEVER语句是SQL预编译程序的指示语句，而不是可执行语句。它通知预编译程序在每条可执行嵌入式SQL语句之后自动生成错误处理程序，并指定了错误处理操作。

### 如何处理三种异常处理：

WHENEVER SQLERROR action：表示一旦sql语句执行时遇到错误信息，则执行action，action中包含了处理错误的代码（SQLCODE<0）。

WHENEVER SQLWARNING action：表示一旦sql语句执行时遇到警告信息，则执行aciton，即action中包含了处理警报的代码（SQLCODE=1）。

WHENEVER NOT FOUND action：表示一旦sql语句执行时没有找到相应的元组，则执行action，即action包含了处理没有查到内容的代码（SQLCODE=100）。

针对上述三种异常处理，用户可以指定预编译程序采取以下三种行为（action）：

whenever …GOTO：通知预编译程序产生一条转移语句。

whenever…CONTINUE：通知预编译程序让程序的控制流转入到下一个主语言语句。

whenever…CALL：通知预编译程序调用函数。

其完整语法如下：

WHENEVER {SQLWARNING | SQLERROR | NOT FOUND} {CONTINUE |DO BREAK| GOTO stmt\_label | CALL function()}

例：WHENEVER的作用

EXEC SQL WHENEVER sqlerror GOTO errormessage1;

EXEC SQL DELETE FROM test WHERE eid < 10000;

EXEC SQL DELETE FROM test WHERE eid < 40000;

EXEC SQL WHENEVER sqlerror CONTINUE;

EXEC SQL UPDATE test SET ename="999";

EXEC SQL WHENEVER sqlerror GOTO errormessage2;

errormessage1:

printf("SQL DELETE error: %ld\n, sqlcode);

exit();

errormessage2:

printf("SQL INSERT error: %ld\n, sqlcode);

exit();

由于第一个WHENEVER语句的作用，前面两个DELETE语句中任一语句内的一个错误会在errormessage1中形成一个转移指令。由于一个WHENEVER语句替代前面WHENEVER语句，所以，嵌入式UPDATE语句中的一个错误会直接转入下一个程序语句中。

从上面例子看出，WHENEVER/CONTINUE语句的主要作用是取消先前的WHENEVER语句的作用。WHENEVER语句使得对嵌入式SQL错误的处理更加简便。

# Db2数据库

## 查看数据库错误码

db2 “? sqlxxx”

## 事务的ACID特性

ACID是数据库事务正确执行的4个基本要素的缩写

A：atomicity（原子性）

C：consitency(一致性)

I： isolation（隔离性）

D：durability(持久性)

## 数据库数据3大完整性

实体完整性：是指保证表中所有的行唯一

参照完整性：是指保证主关键字（被引用表）和外部关键字（引用表）之间的参照关系。

用户自定义完整性：是指用户自己定义的一个列的输入有效性，例如是否允许为空值

## 数据库的范式（5NF）

## Oracle GoldenGate简介

[Oracle](http://www.linuxidc.com/topicnews.aspx?tid=12) Golden Gate软件是一种基于日志的结构化数据复制备份软件，它通过解析源 数据库在线日志或归档日志获得数据的增量变化，再将这些变化应用到目标数据库，从而实现源数据库与目标数据库同步。Oracle Golden Gate可以在异构的IT基础结构（包括几乎 所有常用操作系统平台和数据库平台）之间实现大量数据亚秒一级的实时复制,从而在可以 在应急系统、在线报表、实时数据仓库供应、交易跟踪、数据同步、集中/分发、容灾、数 据库升级和移植、双业务中心等多个场景下应用。同时，Oracle Golden Gate 可以实现一 对一、广播(一对多)、聚合(多对一)、双向、点对点、级联等多种灵活的拓扑结构。

RAC环境下管理OGG-HA <http://www.linuxidc.com/Linux/2012-07/65796.htm>

RAC环境下配置OGG同步 <http://www.linuxidc.com/Linux/2012-07/65794.htm>

GoldenGate单向表DML同步 <http://www.linuxidc.com/Linux/2013-04/82942.htm>

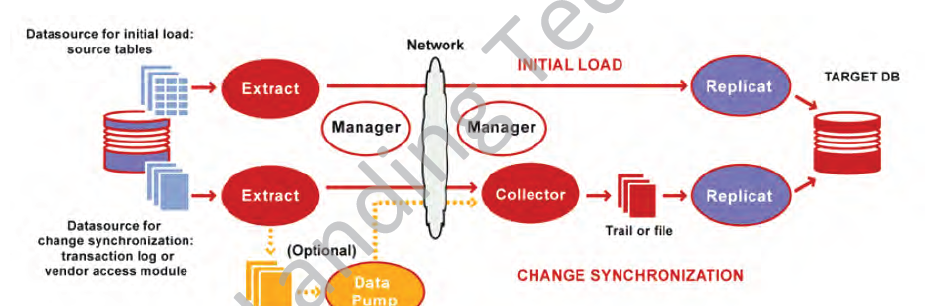
Oracle GoldenGate 系列：Extract 进程的恢复原理 <http://www.linuxidc.com/Linux/2013-04/82563.htm>

Oracle GoldenGate安装配置 <http://www.linuxidc.com/Linux/2013-02/79455.htm>

Oracle goldengate的OGG-01004 OGG-1296错误 <http://www.linuxidc.com/Linux/2011-08/40951.htm>

二、技术结构

和传统的逻辑复制一样，Oracle GoldenGate 实现原理是通过抽取源端 的redo log或者archive log，然后通过TCP/IP投递到目标端，最后解析还原应用到 目标端，使目标端实现同源端数同步。以下是OracleGoldenGate的技术架。



三、OGG进程

1、Manager进程

Manager进程是GoldenGate的控制进程，运行在源端和目标端上。它主要作用有以下几个方面：启动、监控、重启Goldengate的其他进程, 报告错误及事件，分配数据存储 空间，发布阀值报告等。

2、Extract进程

Extract运行在数据库源端，负责从源端数据表或者日志中捕获数据。

3、Pump进程

pump进程运行在数据库源端。

其作用是如果源端使用了本地的trail文件， 那么pump 进程就会把trail以数据块的形式通过TCP/IP协议发送到目标端，这通常也是推荐的 方式。pump进程本质是extract进程的一种特殊形式，如果不使用trails文件，那么 就是extract进程在抽取完数据以后，直接投递到目标端。

4.Trail文件

为了更有效、更安全的把数据库事务信息从源端投递到目标端。GoldenGate引进 trail文件的概念。前面提到extract抽取完数据以后Goldengate会将抽取的事务信息转化为一种GoldenGate专有格式的文件。然后pump负责把源端的trail文件投递到目标端， 所以源、目标两端都会存在这种文件。trail文件存在的目的旨在防止单点故障，将事务信 息持久化，并且使用checkpoint机制来记录其读写位置，如果故障发生，则数据可以根据 checkpoint记录的位置来重传。

5.Replicat进程

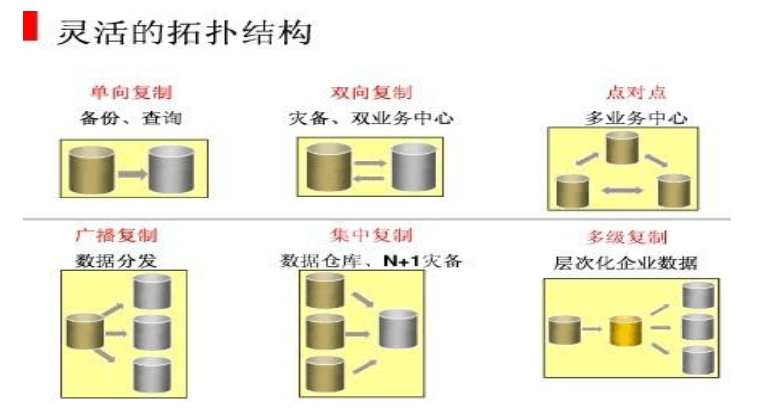
Replicat进程，通常我们也把它叫做应用进程。运行在目标端，是数据传递的最后一站，负责读取目标端trail文件中的内容，并将其解析为DML或DDL语句，然后应用到目标数据库中。

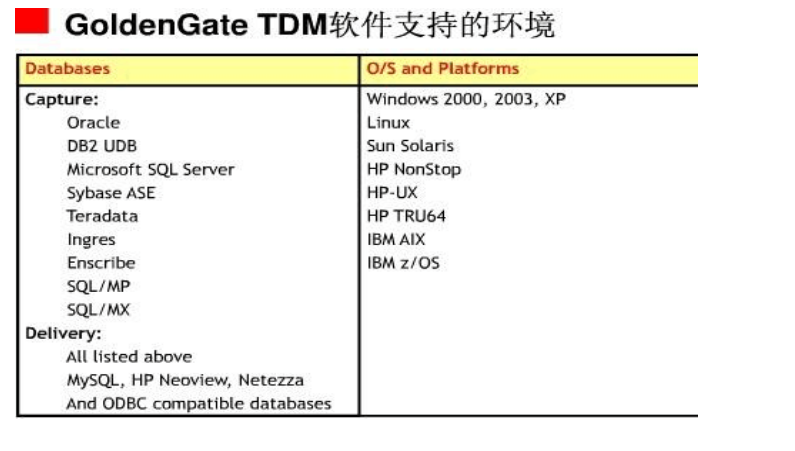
6.GGSCI

GGSCI是GoldenGate Software Command Interface 的缩写，

它提供了十分丰富的 命令来对Goldengate进行各种操作，如创建、修改、监控GoldenGate进程等等。

三、应用的拓展结构、支持平台和数据库





## hex()函数，十六进制转换函数

## db2用户授权

db2 grant insert,delete,update,select on tablename to user $DBUSER

## commit操作不能和游标一起用混用

commit操作会断开会话连接，如果和游标一起使用，会导致游标关闭报-501。

## 统计TPS的SQL

select   shijian,count(\*)

from

(

select  to\_char(rec\_crt\_ts,'YYYYMMDDHH24MI') as

shijian

from

tbl\_chswt\_swt\_log1

where

timestampdiff(2,char(rec\_upd\_ts -rec\_crt\_ts))>5

and rec\_crt\_ts  between

'2015-10-21-10.00.00.000000'

and

'2015-10-21-10.00.00.000000'

)

group by

shijian

with ur

## db2 常用的时间函数

B2时间函数是我们最常见的函数之一

1. --获取当前日期：
3. select current date from sysibm.sysdummy1;
4. values current date;
6. --获取当前日期
7. select current time from sysibm.sysdummy1;
8. values current time;
9. --获取当前时间戳
10. select current timestamp from sysibm.sysdummy1;
11. values current timestamp;
13. --要使当前时间或当前时间戳记调整到 GMT/CUT，则把当前的时间或时间戳记减去当前时区寄存器：
15. values current time -current timezone;
16. values current timestamp -current timezone;
18. --获取当前年份
20. values year(current timestamp);
22. --获取当前月
23. values month(current timestamp);
25. --获取当前日
26. values day(current timestamp);
28. --获取当前时
29. values hour(current timestamp);
31. --获取分钟
32. values minute(current timestamp);
34. --获取秒
35. values second(current timestamp);
37. --获取毫秒
38. values microsecond(current timestamp);
40. --从时间戳记单独抽取出日期和时间
42. values date(current timestamp);
43. values VARCHAR\_FORMAT(current TIMESTAMP,'yyyy-mm-dd');
44. values char(current date);
45. values time(current timestamp);
47. --执行日期和时间的计算
49. values current date+1 year;
50. values current date+3 years+2 months +15 days;
51. values current time +5 hours -3 minutes +10 seconds;
53. --计算两个日期之间的天数
55. values days(current date)- days(date('2010-02-20'));
57. --时间和日期换成字符串
59. values char(current date);
60. values char(current time);
62. --要将字符串转换成日期或时间值
64. values timestamp('2010-03-09-22.43.00.000000');
65. values timestamp('2010-03-09 22:44:36');
66. values date('2010-03-09');
67. values date('03/09/2010');
68. values time('22:45:27');
69. values time('22.45.27');
71. --计算两个时间戳记之间的时差：
73. --秒的小数部分为单位
74. values timestampdiff(1,char(current timestamp - timestamp('2010-01-01-00.00.00')));
75. --秒为单位
76. values timestampdiff(2,char(current timestamp - timestamp('2010-01-01-00.00.00')));
77. --分为单位
78. values timestampdiff(4,char(current timestamp - timestamp('2010-01-01-00.00.00')));
79. --小时为单位
80. values timestampdiff(8,char(current timestamp - timestamp('2010-01-01-00.00.00')));
81. --天为单位
82. values timestampdiff(16,char(current timestamp - timestamp('2010-01-01-00.00.00')));
83. --周为单位
84. values timestampdiff(32,char(current timestamp - timestamp('2010-01-01-00.00.00')));
85. --月为单位
86. values timestampdiff(64,char(current timestamp - timestamp('2010-01-01-00.00.00')));
87. --季度为单位
88. values timestampdiff(128,char(current timestamp - timestamp('2010-01-01-00.00.00')));
89. --年为单位
90. values timestampdiff(256,char(current timestamp - timestamp('2010-

## db2 快速清理数据库表数据

db2 "import from /dev/null of ixf   replace  表名"

## 左链接和右链接

join可以分主次表 外联接有三种类型：完全外联，左联，右联．

完全外联包含两张表的所有记录．

左联是以左边的表为主，右边的为辅，右联则相反

左连接where只影向右表，右连接where只影响左表。

例如：

A表(a1,b1,c1)

B表(a2,b2)

a1 b1  c1 a2 b2

01 数学 95 01 张三

02 语文 90 02 李四

03 英语 80 04 王五

select A.\*,B.\* from A left outer join B on(A.a1=B.a2)

结果是:

a1 b1  c1 a2   b2

01 数学 95 01   张三

02 语文 90 02   李四

03 英语 80 NULL NULL

select A.\*,B.\* from A right outer join B on(A.a1=B.a2)

结果是:

a1   b1  c1   a2   b2

01   数学 95   01   张三

02   语文 90   02   李四

NULL NULL NULL 04   王五

## db2 sqlcode=668解决方法

解决方法：db2   reorg  table   表名   即可

总结：

reorg  table  <tablename> 通过重构行来消除“碎片”数据幷压缩信息，对表进行重组

runstats  on table  <tableschema>.<tablename>  收集<tablename>表的统计局信息

reorgchk   on   table  all  确定是否需要对表进行重组，对于所有表自动执行runstats  很有用

## 连接数据库

db2 connect to <数据库名> user <用户名> using <密码> --连接到远端数据库

## db2修改密码

db2 connect to 数据库名 user  用户名 using 旧密码 new 新密码 confirm 新密码

## to\_char函数

to\_char（时间字段，“YYYYMMDDHH24MISS”）

## except、except all、union、union all、intersect

union  all-并运算

union-并运算，但会去掉重复纪录并按第一列排序

except all-相当于做减运算

except

intersect  all-相当于做交运算

intersect

## 取数据库时间戳

values   current timestamp

## db2查看死锁

1、db2top  -d  数据库名   -n 节点名  -u 用户名  -p 密码  -i

2、db2pd  -d 数据库名   -wlocks

3、db2 list applications show detail  | grep   dead

## sql语句大全

一、基础

1、说明：创建数据库

CREATE DATABASE database-name

2、说明：删除数据库

drop database dbname

3、说明：备份sql server

--- 创建 备份数据的 device

USE master

EXEC sp\_addumpdevice 'disk', 'testBack', 'c:\mssql7backup\MyNwind\_1.dat'

--- 开始 备份

BACKUP DATABASE pubs TO testBack

4、说明：创建新表

create table tabname(col1 type1 [not null] [primary key],col2 type2 [not null],..)

根据已有的表创建新表：

A：create table tab\_new like tab\_old (使用旧表创建新表)

B：create table tab\_new as select col1,col2… from tab\_old definition only

5、说明：删除新表

drop table tabname

6、说明：增加一个列

Alter table tabname add (column col type)

注：列增加后将不能删除。DB2中列加上后数据类型也不能改变，唯一能改变的是增加varchar类型的长度。

7、说明：添加主键： Alter table tabname add primary key(col)

说明：删除主键： Alter table tabname drop primary key(col)

8、说明：创建索引：create [unique] index idxname on tabname(col….)

删除索引：drop index idxname

注：索引是不可更改的，想更改必须删除重新建。

9、说明：创建视图：create view viewname as select statement

删除视图：drop view viewname

10、说明：几个简单的基本的sql语句

选择：select \* from table1 where 范围

插入：insert into table1(field1,field2) values(value1,value2)

删除：delete from table1 where 范围

更新：update table1 set field1=value1 where 范围

查找：select \* from table1 where field1 like ’%value1%’ ---like的语法很精妙，查资料!

排序：select \* from table1 order by field1,field2 [desc]

总数：select count as totalcount from table1

求和：select sum(field1) as sumvalue from table1

平均：select avg(field1) as avgvalue from table1

最大：select max(field1) as maxvalue from table1

最小：select min(field1) as minvalue from table1

11、说明：几个高级查询运算词

A： UNION 运算符

UNION 运算符通过组合其他两个结果表（例如 TABLE1 和 TABLE2）并消去表中任何重复行而派生出一个结果表。当 ALL 随 UNION 一起使用时（即 UNION ALL），不消除重复行。两种情况下，派生表的每一行不是来自 TABLE1 就是来自 TABLE2。

B： EXCEPT 运算符

EXCEPT 运算符通过包括所有在 TABLE1 中但不在 TABLE2 中的行并消除所有重复行而派生出一个结果表。当 ALL 随 EXCEPT 一起使用时 (EXCEPT ALL)，不消除重复行。

C： INTERSECT 运算符

INTERSECT 运算符通过只包括 TABLE1 和 TABLE2 中都有的行并消除所有重复行而派生出一个结果表。当 ALL 随 INTERSECT 一起使用时 (INTERSECT ALL)，不消除重复行。

注：使用运算词的几个查询结果行必须是一致的。

12、说明：使用外连接

A、left （outer） join：

左外连接（左连接）：结果集几包括连接表的匹配行，也包括左连接表的所有行。

SQL: select a.a, a.b, a.c, b.c, b.d, b.f from a LEFT OUT JOIN b ON a.a = b.c

B：right （outer） join:

右外连接(右连接)：结果集既包括连接表的匹配连接行，也包括右连接表的所有行。

C：full/cross （outer） join：

全外连接：不仅包括符号连接表的匹配行，还包括两个连接表中的所有记录。

12、分组:Group by:

   一张表，一旦分组 完成后，查询后只能得到组相关的信息。

    组相关的信息：（统计信息） count,sum,max,min,avg  分组的标准)

    在SQLServer中分组时：不能以text,ntext,image类型的字段作为分组依据

   在selecte统计函数中的字段，不能和普通的字段放在一起；

13、对数据库进行操作：

   分离数据库： sp\_detach\_db; 附加数据库：sp\_attach\_db 后接表明，附加需要完整的路径名

14.如何修改数据库的名称:

sp\_renamedb 'old\_name', 'new\_name'

二、提升

1、说明：复制表(只复制结构,源表名：a 新表名：b) (Access可用)

法一：select \* into b from a where 1<>1（仅用于SQlServer）

法二：select top 0 \* into b from a

2、说明：拷贝表(拷贝数据,源表名：a 目标表名：b) (Access可用)

insert into b(a, b, c) select d,e,f from b;

3、说明：跨数据库之间表的拷贝(具体数据使用绝对路径) (Access可用)

insert into b(a, b, c) select d,e,f from b in ‘具体数据库’ where 条件

例子：..from b in '"&Server.MapPath(".")&"\data.mdb" &"' where..

4、说明：子查询(表名1：a 表名2：b)

select a,b,c from a where a IN (select d from b ) 或者: select a,b,c from a where a IN (1,2,3)

5、说明：显示文章、提交人和最后回复时间

select a.title,a.username,b.adddate from table a,(select max(adddate) adddate from table where table.title=a.title) b

6、说明：外连接查询(表名1：a 表名2：b)

select a.a, a.b, a.c, b.c, b.d, b.f from a LEFT OUT JOIN b ON a.a = b.c

7、说明：在线视图查询(表名1：a )

select \* from (SELECT a,b,c FROM a) T where t.a > 1;

8、说明：between的用法,between限制查询数据范围时包括了边界值,not between不包括

select \* from table1 where time between time1 and time2

select a,b,c, from table1 where a not between 数值1 and 数值2

9、说明：in 的使用方法

select \* from table1 where a [not] in (‘值1’,’值2’,’值4’,’值6’)

10、说明：两张关联表，删除主表中已经在副表中没有的信息

delete from table1 where not exists ( select \* from table2 where table1.field1=table2.field1 )

11、说明：四表联查问题：

select \* from a left inner join b on a.a=b.b right inner join c on a.a=c.c inner join d on a.a=d.d where .....

12、说明：日程安排提前五分钟提醒

SQL: select \* from 日程安排 where datediff('minute',f开始时间,getdate())>5

13、说明：一条sql 语句搞定数据库分页

select top 10 b.\* from (select top 20 主键字段,排序字段 from 表名 order by 排序字段 desc) a,表名 b where b.主键字段 = a.主键字段 order by a.排序字段

具体实现：

关于数据库分页：

  declare @start int,@end int

  @sql  nvarchar(600)

  set @sql=’select top’+str(@end-@start+1)+’+from T where rid not in(select top’+str(@str-1)+’Rid from T where Rid>-1)’

  exec sp\_executesql @sql

注意：在top后不能直接跟一个变量，所以在实际应用中只有这样的进行特殊的处理。Rid为一个标识列，如果top后还有具体的字段，这样做是非常有好处的。因为这样可以避免 top的字段如果是逻辑索引的，查询的结果后实际表中的不一致（逻辑索引中的数据有可能和数据表中的不一致，而查询时如果处在索引则首先查询索引）

14、说明：前10条记录

select top 10 \* form table1 where 范围

15、说明：选择在每一组b值相同的数据中对应的a最大的记录的所有信息(类似这样的用法可以用于论坛每月排行榜,每月热销产品分析,按科目成绩排名,等等.)

select a,b,c from tablename ta where a=(select max(a) from tablename tb where tb.b=ta.b)

16、说明：包括所有在 TableA 中但不在 TableB和TableC 中的行并消除所有重复行而派生出一个结果表

(select a from tableA ) except (select a from tableB) except (select a from tableC)

17、说明：随机取出10条数据

select top 10 \* from tablename order by newid()

18、说明：随机选择记录

select newid()

19、说明：删除重复记录

1),delete from tablename where id not in (select max(id) from tablename group by col1,col2,...)

2),select distinct \* into temp from tablename

  delete from tablename

  insert into tablename select \* from temp

评价： 这种操作牵连大量的数据的移动，这种做法不适合大容量但数据操作

3),例如：在一个外部表中导入数据，由于某些原因第一次只导入了一部分，但很难判断具体位置，这样只有在下一次全部导入，这样也就产生好多重复的字段，怎样删除重复字段

alter table tablename

--添加一个自增列

add  column\_b int identity(1,1)

delete from tablename where column\_b not in(

select max(column\_b)  from tablename group by column1,column2,...)

alter table tablename drop column column\_b

20、说明：列出数据库里所有的表名

select name from sysobjects where type='U' // U代表用户

21、说明：列出表里的所有的列名

select name from syscolumns where id=object\_id('TableName')

22、说明：列示type、vender、pcs字段，以type字段排列，case可以方便地实现多重选择，类似select 中的case。

select type,sum(case vender when 'A' then pcs else 0 end),sum(case vender when 'C' then pcs else 0 end),sum(case vender when 'B' then pcs else 0 end) FROM tablename group by type

显示结果：

type vender pcs

电脑 A 1

电脑 A 1

光盘 B 2

光盘 A 2

手机 B 3

手机 C 3

23、说明：初始化表table1

TRUNCATE TABLE table1

24、说明：选择从10到15的记录

select top 5 \* from (select top 15 \* from table order by id asc) table\_别名 order by id desc

三、技巧

1、1=1，1=2的使用，在SQL语句组合时用的较多

“where 1=1” 是表示选择全部    “where 1=2”全部不选，

如：

if @strWhere !=''

begin

set @strSQL = 'select count(\*) as Total from [' + @tblName + '] where ' + @strWhere

end

else

begin

set @strSQL = 'select count(\*) as Total from [' + @tblName + ']'

end

我们可以直接写成

错误！未找到目录项。

set @strSQL = 'select count(\*) as Total from [' + @tblName + '] where 1=1 安定 '+ @strWhere 2、收缩数据库

--重建索引

DBCC REINDEX

DBCC INDEXDEFRAG

--收缩数据和日志

DBCC SHRINKDB

DBCC SHRINKFILE

3、压缩数据库

dbcc shrinkdatabase(dbname)

4、转移数据库给新用户以已存在用户权限

exec sp\_change\_users\_login 'update\_one','newname','oldname'

go

5、检查备份集

RESTORE VERIFYONLY from disk='E:\dvbbs.bak'

6、修复数据库

ALTER DATABASE [dvbbs] SET SINGLE\_USER

GO

DBCC CHECKDB('dvbbs',repair\_allow\_data\_loss) WITH TABLOCK

GO

ALTER DATABASE [dvbbs] SET MULTI\_USER

GO

7、日志清除

SET NOCOUNT ON

DECLARE @LogicalFileName sysname,

@MaxMinutes INT,

@NewSize INT

USE tablename -- 要操作的数据库名

SELECT  @LogicalFileName = 'tablename\_log', -- 日志文件名

@MaxMinutes = 10, -- Limit on time allowed to wrap log.

@NewSize = 1  -- 你想设定的日志文件的大小(M)

Setup / initialize

DECLARE @OriginalSize int

SELECT @OriginalSize = size

FROM sysfiles

WHERE name = @LogicalFileName

SELECT 'Original Size of ' + db\_name() + ' LOG is ' +

CONVERT(VARCHAR(30),@OriginalSize) + ' 8K pages or ' +

CONVERT(VARCHAR(30),(@OriginalSize\*8/1024)) + 'MB'

FROM sysfiles

WHERE name = @LogicalFileName

CREATE TABLE DummyTrans

(DummyColumn char (8000) not null)

DECLARE @Counter    INT,

@StartTime DATETIME,

@TruncLog   VARCHAR(255)

SELECT @StartTime = GETDATE(),

@TruncLog = 'BACKUP LOG ' + db\_name() + ' WITH TRUNCATE\_ONLY'

DBCC SHRINKFILE (@LogicalFileName, @NewSize)

EXEC (@TruncLog)

-- Wrap the log if necessary.

WHILE @MaxMinutes > DATEDIFF (mi, @StartTime, GETDATE()) -- time has not expired

AND @OriginalSize = (SELECT size FROM sysfiles WHERE name = @LogicalFileName)

AND (@OriginalSize \* 8 /1024) > @NewSize

BEGIN -- Outer loop.

SELECT @Counter = 0

WHILE   ((@Counter < @OriginalSize / 16) AND (@Counter < 50000))

BEGIN -- update

INSERT DummyTrans VALUES ('Fill Log') DELETE DummyTrans

SELECT @Counter = @Counter + 1

END

EXEC (@TruncLog)

END

SELECT 'Final Size of ' + db\_name() + ' LOG is ' +

CONVERT(VARCHAR(30),size) + ' 8K pages or ' +

CONVERT(VARCHAR(30),(size\*8/1024)) + 'MB'

FROM sysfiles

WHERE name = @LogicalFileName

DROP TABLE DummyTrans

SET NOCOUNT OFF

8、说明：更改某个表

exec sp\_changeobjectowner 'tablename','dbo'

9、存储更改全部表

CREATE PROCEDURE dbo.User\_ChangeObjectOwnerBatch

@OldOwner as NVARCHAR(128),

@NewOwner as NVARCHAR(128)

AS

DECLARE @Name    as NVARCHAR(128)

DECLARE @Owner   as NVARCHAR(128)

DECLARE @OwnerName   as NVARCHAR(128)

DECLARE curObject CURSOR FOR

select 'Name'    = name,

   'Owner'    = user\_name(uid)

from sysobjects

where user\_name(uid)=@OldOwner

order by name

OPEN   curObject

FETCH NEXT FROM curObject INTO @Name, @Owner

WHILE(@@FETCH\_STATUS=0)

BEGIN

if @Owner=@OldOwner

begin

   set @OwnerName = @OldOwner + '.' + rtrim(@Name)

   exec sp\_changeobjectowner @OwnerName, @NewOwner

end

-- select @name,@NewOwner,@OldOwner

FETCH NEXT FROM curObject INTO @Name, @Owner

END

close curObject

deallocate curObject

GO

10、SQL SERVER中直接循环写入数据

declare @i int

set @i=1

while @i<30

begin

    insert into test (userid) values(@i)

    set @i=@i+1

end

案例：

有如下表，要求就裱中所有沒有及格的成績，在每次增長0.1的基礎上，使他們剛好及格:

    Name     score

    Zhangshan   80

    Lishi       59

    Wangwu      50

    Songquan    69

while((select min(score) from tb\_table)<60)

begin

update tb\_table set score =score\*1.01

where score<60

if  (select min(score) from tb\_table)>60

  break

else

    continue

end

数据开发-经典

1.按姓氏笔画排序:

Select \* From TableName Order By CustomerName Collate Chinese\_PRC\_Stroke\_ci\_as //从少到多

2.数据库加密:

select encrypt('原始密码')

select pwdencrypt('原始密码')

select pwdcompare('原始密码','加密后密码') = 1--相同；否则不相同 encrypt('原始密码')

select pwdencrypt('原始密码')

select pwdcompare('原始密码','加密后密码') = 1--相同；否则不相同

3.取回表中字段:

declare @list varchar(1000),

@sql nvarchar(1000)

select @list=@list+','+b.name from sysobjects a,syscolumns b where a.id=b.id and a.name='表A'

set @sql='select '+right(@list,len(@list)-1)+' from 表A'

exec (@sql)

4.查看硬盘分区:

EXEC master..xp\_fixeddrives

5.比较A,B表是否相等:

if (select checksum\_agg(binary\_checksum(\*)) from A)

     =

    (select checksum\_agg(binary\_checksum(\*)) from B)

print '相等'

else

print '不相等'

6.杀掉所有的事件探察器进程:

DECLARE hcforeach CURSOR GLOBAL FOR SELECT 'kill '+RTRIM(spid) FROM master.dbo.sysprocesses

WHERE program\_name IN('SQL profiler',N'SQL 事件探查器')

EXEC sp\_msforeach\_worker '?'

7.记录搜索:

开头到N条记录

Select Top N \* From 表

-------------------------------

N到M条记录(要有主索引ID)

Select Top M-N \* From 表 Where ID in (Select Top M ID From 表) Order by ID   Desc

----------------------------------

N到结尾记录

Select Top N \* From 表 Order by ID Desc

案例

例如1：一张表有一万多条记录，表的第一个字段 RecID 是自增长字段， 写一个SQL语句， 找出表的第31到第40个记录。

select top 10 recid from A where recid not  in(select top 30 recid from A)

分析：如果这样写会产生某些问题，如果recid在表中存在逻辑索引。

    select top 10 recid from A where……是从索引中查找，而后面的select top 30 recid from A则在数据表中查找，这样由于索引中的顺序有可能和数据表中的不一致，这样就导致查询到的不是本来的欲得到的数据。

解决方案

1， 用order by select top 30 recid from A order by ricid 如果该字段不是自增长，就会出现问题

2， 在那个子查询中也加条件：select top 30 recid from A where recid>-1

例2：查询表中的最后以条记录，并不知道这个表共有多少数据,以及表结构。

set @s = 'select top 1 \* from T   where pid not in (select top ' + str(@count-1) + ' pid  from  T)'

print @s      exec  sp\_executesql  @s

9：获取当前数据库中的所有用户表

select Name from sysobjects where xtype='u' and status>=0

## 查询数据库中的所有sequence

select \* from syscat.sequences

## 取sequence序列值

values   nextval for    **sequencename**

## 如何在CLP执行操作系统的命令?

  如何在CLP执行操作系统的命令?

　 在命令前加"!"作为前缀

　 DB2=>! ls

## db2top命令简介----db2监控工具

db2top -d TEST -n mynode -u user -p passwd -V skm4 -B -i 1

命令参数如下所示：

-d TEST # 数据库名称

-n mynode # 节点名

-u user # 用户标识

-p passwd # 密码

-V skm4 # 模式名称

-B # 启用粗体

-i 1 # 屏幕更新时间间隔：1 秒

## db2look工具

db2look -d $DBNAME -e -o out.sql -i $DBUSERNAME -w $DBPWD -t tablename

不加-t参数，则导出数据库中所有的表建表语句DDL

db2 -tvf  <脚本名称>.sql 把上述导出的表结构导入到数据库表结构

## load命令

db2 "load client from D:\xx.txt of del insert/replace into tblName"(不需要写日志，但插入前表必须存在，不能create tabel)

db2 "load client from D:\xx.txt of del restart/terminate into tabName"(当个导入数据出现问题被强制中断时，此表会被加锁，通过此命令可以解锁)

## import 导入数据

以默认分隔字符加载， 默认为“，”

db2 “import from <数据文件名data.del > of del commitcount 5000 insert into <表名>”

以指定分隔符“#”加载

db2 “import from <数据文件名data.del > of del modified by coldel# commitcount 5000 insert into <表名>”

coldel和#间不能有空格，是连在一起的，否则会报错

## insert\_update导入数据

db2 “import from <数据文件名data.del > of del insert\_update into <表名>(主键，要更新的字段)”

## 利用游标更新删除数据

update

表名

set

列名=值{......}

where

current of

游标名

delete

from

表名

where

current of 游标名

## export 导出数据

以默认分隔字符导出， 默认为“，”

db2 “export to <数据文件名data.del > of del select \* from <表名>”

以指定分隔符“#”导出

db2 “export to <数据文件名data.del > of del modified by coldel# select \* from <表名>”

coldel和#间不能有空格，是连在一起的，否则会报错

## 导入文件的中的DDL语句

db2 -tvf  <脚本名称>.sql

## db2move工具

db2move是一个集成式的数据移动工具，它支持导出export、导入import、装入load三种操作方式。其实这三种操作方式是通过简单的db2 export 、db2 import、db2 load指令来完成的

语法：

db2move <数据库名> <action> [<option> <value>]

首先要指定数据库名和要执行的操作<action>:export 、import 、load,然后指定一个选项来定义操作的范围，例如，可以将一个限制在特定的<option>：表(-tn)、表空间(-ts)、表的创建者(-tc)、或模式名(-sn),另外也可以用-tf选项，此时文件作为参数，其中列出了要导出的表名，在改文件中每一行只能列出一个完整的表名

db2move <数据库名> export -u <用户名> -p <密码>

将数据库中的全部数据导出到当前目录，每个表中的内容都储存在一个ixf文件中，每个ixf文件都有一个与之相对应的.msg文件，msg文件是描述从表中导出数据时的信息的。另外，还有两个文件,db2move.lst用来记录.ixf文件、msg文件与表的一一对应关系，EXPORT.out记录的是导出数据时的屏幕输出

db2move <数据库名> export -u <用户名> -p <密码> -tn <表名>

将数据中某张表的数据导出到当前目录

db2move <数据库名> import -u <用户名> -p <密码> ，将上面导出的数据重新导入数据库

db2move <数据库名> load -u <用户名> -p <密码>

## 备份整个数据库数据

db2 backup db db2name<数据库名称>

## 还原数据库

db2 restore db <db2name>

## 启动数据库

db2start

## 停止数据库

db2stop

## 创建数据库使用utf-8编码

create database <数据库名> using codeset utf-8 territory CN

数据

## 数据库编目

1. 编目远程节点

db2 uncatalog node  **<远程节点名>**

db2 catalog tcpip node **<远程节点名>** remote <IP地址> server <端口>

1. 编目数据库

db2 uncatalog database <远程数据库实例名>

本地数据库实例名(url下的jdbc:db2:inst2tc)

db2 catalog database <远程数据库实例名> as <本地数据库实例名> at node  **<远程节点名>**

1. 测试数据库连接

db2 connect to <本地数据库实例名> user user\_name USING password

## 设置当前数据库模式

db2 set current schema $DBSCHMA

## 取数据库序列值

values nextval for 序列名

## 查看数据库参数

db2 get dbm cfg

db2 get db cfg for <数据库名>

## 修改数据库日志文件大小

db2 update db cfg for <数据库名> using logbufsz 1024

修改完了之后，应执行下面的命令使其生效

db2 stop

db2 start

## DB2[数据库](http://lib.csdn.net/base/14)的日志满，修改数据库日志的参数

可以根据情况来修改如下参数：

    实用程序堆大小：db2 update db cfg for <dbname> using    UTIL\_HEAP\_SZ  50000

    修改日志缓冲区：db2 update db cfg for <dbname> using LOGBUFSZ    10240

    修改日志文件大小：db2 update db cfg for <dbname> using LOGFILSIZ 204800

    修改主日志文件个数：db2 update db cfg for <dbname> using LOGPRIMARY 80

    修改辅助日志文件个数：db2 update db cfg for <dbname> using LOGSECOND 176

    自动语句统计信息：db2 "update db cfg for middledb using AUTO\_STMT\_STATS off"

对于日志文件的个数：主日志个数+辅助日志个数<=256

使用命令：db2 get db cfg for <库名>获取db2数据库的配置参数，如下：

数据库  的数据库配置

 数据库配置发行版级别                                    = 0x0a00

 数据库发行版级别                                        = 0x0a00

 数据库地域                                              = CN

 数据库代码页                                            = 1386

 数据库代码集                                            = GBK

 数据库国家／地区代码                                    = 86

 数据库整理顺序                                          = UNIQUE

 备用整理顺序                              (ALT\_COLLATE) =

 数据库页大小                                            = 4096

 动态 SQL 查询管理                      (DYN\_QUERY\_MGMT) = DISABLE

 对此数据库的发现支持                      (DISCOVER\_DB) = ENABLE

 缺省查询优化类                           (DFT\_QUERYOPT) = 5

 并行度                                     (DFT\_DEGREE) = 1

 在算术异常时继续                      (DFT\_SQLMATHWARN) = NO

 缺省刷新有效期                        (DFT\_REFRESH\_AGE) = 0

 缺省维护的选项（DFT\_MTTB\_TYPES）的表类型                = SYSTEM

 保留的高频值的数目                     (NUM\_FREQVALUES) = 10

 保留的分位点数目                        (NUM\_QUANTILES) = 20

 备份暂挂                                                = NO

 数据库是一致的                                          = YES

 前滚暂挂                                                = NO

 复原暂挂                                                = NO

 启用的多页文件分配                                      = NO

 恢复状态的日志保留                                      = RECOVERY

 日志记录状态的用户出口                                  = YES

 Data Links 标记到期时间间隔（秒）           (DL\_EXPINT) = 60

 Data Links 写标记初始时间间隔           (DL\_WT\_IEXPINT) = 60

 副本的 Data Links 数目                  (DL\_NUM\_COPIES) = 1

 删除后的 Data Links 时间（天数）        (DL\_TIME\_DROP)  = 1

 大写的 Data Links 标记                       (DL\_UPPER) = NO

 Data Links 标记算法                          (DL\_TOKEN) = MAC0

 数据库堆（4KB）                                (DBHEAP) = 600

 数据库共享内存大小（4KB）             (DATABASE\_MEMORY) = AUTOMATIC

 目录高速缓存大小（4KB）               (CATALOGCACHE\_SZ) = (MAXAPPLS\*4)

 日志缓冲区大小（4KB）                         (LOGBUFSZ) = 8

 实用程序堆大小（4KB）                    (UTIL\_HEAP\_SZ) = 5000

 缓冲池大小（页）                             (BUFFPAGE) = 250

 扩充存储段大小（4KB）                   (ESTORE\_SEG\_SZ) = 16000

 扩充存储段的数目                      (NUM\_ESTORE\_SEGS) = 0

 锁定列表的最大存储量（4KB）                   (LOCKLIST) = 50

 应用程序组内存集的最大大小（4KB）     (APPGROUP\_MEM\_SZ) = 30000

 应用程序组堆的内存百分比              (GROUPHEAP\_RATIO) = 70

 最大应用程序控制堆大小（4KB）         (APP\_CTL\_HEAP\_SZ) = 128

 共享排序的排序堆域值（4KB）            (SHEAPTHRES\_SHR) = (SHEAPTHRES)

 排序列表堆（4KB）                            (SORTHEAP) = 256

 SQL 语句堆（4KB）                            (STMTHEAP) = 2048

 缺省应用程序堆（4KB）                      (APPLHEAPSZ) = 256

 程序包高速缓存大小（4KB）                  (PCKCACHESZ) = (MAXAPPLS\*8)

 统计信息堆大小（4KB）                    (STAT\_HEAP\_SZ) = 4384

 检查死锁的时间间隔（毫秒）                  (DLCHKTIME) = 10000

 每个应用程序的锁定百分比列表                 (MAXLOCKS) = 22

 锁定超时（秒）                             (LOCKTIMEOUT) = -1

 更改的页阈值                           (CHNGPGS\_THRESH) = 60

 异步页清除程序的数目                   (NUM\_IOCLEANERS) = 1

 I/O 服务器的数目                        (NUM\_IOSERVERS) = 3

 索引排序标志                                 (INDEXSORT) = YES

 顺序检测标志                                (SEQDETECT) = YES

 缺省预取大小（页）                    (DFT\_PREFETCH\_SZ) = 16

 跟踪修改的页数                               (TRACKMOD) = ON

 容器的缺省数目                                          = 1

 缺省表空间扩展数据块大小(页)            (DFT\_EXTENT\_SZ) = 32

 活动应用程序的最大数目                       (MAXAPPLS) = 10000

 活动应用程序的平均数目       (AVG\_APPLS) = 1

 每个应用程序的最大打开数据库文件数           (MAXFILOP) = 64

 日志文件大小（4KB）                         (LOGFILSIZ) = 5000

 主日志文件的数目                           (LOGPRIMARY) = 35

 辅助日志文件的数目                          (LOGSECOND) = 15

 已更改的至日志文件的路径                   (NEWLOGPATH) =

 日志文件路径                                            = E:\DB2\NODE0000\SQL00

001\SQLOGDIR\

 溢出日志路径                          (OVERFLOWLOGPATH) =

 镜像日志路径                            (MIRRORLOGPATH) =

 首个活动日志文件                                        = S0000578.LOG

 磁盘上已满的块日志                    (BLK\_LOG\_DSK\_FUL) = NO

 事务使用的最大活动日志空间的百分比            (MAX\_LOG) = 0

 1 个活动 UOW 的活动日志文件的数目        (NUM\_LOG\_SPAN) = 0

 组落实计数                                  (MINCOMMIT) = 1

 软检查点前回收的日志文件的百分比              (SOFTMAX) = 100

 启用的恢复的日志保留                        (LOGRETAIN) = RECOVERY

 启用的日志记录的用户出口                     (USEREXIT) = ON

 HADR 数据库角色                                          = STANDARD

 HADR 本地主机名                        (HADR\_LOCAL\_HOST) =

 HADR 本地服务名称                       (HADR\_LOCAL\_SVC) =

 HADR 远程主机名                       (HADR\_REMOTE\_HOST) =

 HADR 远程服务名称                      (HADR\_REMOTE\_SVC) =

 远程服务器的 HADR 实例名              (HADR\_REMOTE\_INST) =

 HADR 超时值                               (HADR\_TIMEOUT) = 120

 HADR 日志写同步方式                      (HADR\_SYNCMODE) = NEARSYNC

 第一个日志归档方法                        (LOGARCHMETH1) = USEREXIT

 logarchmeth1 的选项                        (LOGARCHOPT1) =

 第二个日志归档方法                        (LOGARCHMETH2) = OFF

 logarchmeth2 的选项                        (LOGARCHOPT2) =

 故障转移日志归档路径                     (FAILARCHPATH) =

 错误时重试日志归档次数                   (NUMARCHRETRY) = 5

 日志归档重试延迟（秒）                 (ARCHRETRYDELAY) = 20

 供应商选项                                  (VENDOROPT) =

 启用的自动重新启动                        (AUTORESTART) = ON

 索引重新创建时间和重做索引构建               (INDEXREC) = SYSTEM (ACCESS)

 在索引构建期间记录页                    (LOGINDEXBUILD) = OFF

 loadrec 会话的缺省数目                (DFT\_LOADREC\_SES) = 1

 要保留的数据库备份的数目               (NUM\_DB\_BACKUPS) = 12

 恢复历史保留时间（天数）              (REC\_HIS\_RETENTN) = 366

 TSM 管理类                              (TSM\_MGMTCLASS) =

 TSM 节点名                               (TSM\_NODENAME) =

 TSM 所有者                                  (TSM\_OWNER) =

 TSM 密码                                 (TSM\_PASSWORD) =

 自动维护                                   (AUTO\_MAINT) = OFF

 自动数据库备份                       (AUTO\_DB\_BACKUP) = OFF

 自动表维护                           (AUTO\_TBL\_MAINT) = OFF

 自动 runstats                       (AUTO\_RUNSTATS) = OFF

   自动统计信息概要分析              (AUTO\_STATS\_PROF) = OFF

   自动概要文件更新                  (AUTO\_PROF\_UPD) = OFF

   自动重组                               (AUTO\_REORG) = OFF

## 列出所有数据库实例

db2ilist

## 查看当前数据库管理实例

db2 get instance

## 查看版本

db2level

## 设置实例启动时是否自动启动

db2iauto -on <实例名> 自动启动

db2iauto -off <实例名> 不自动启动

## 查看数据库中所有的表

db2 list tables for all

## 优化命令 reorg、runstats

数据库使用一段时间，数据空间会变得越来越大，一些delete掉的数据任存在数据库中，占用数据空间，影响数据库性能。因此需要定期运行reorg、runstats命令，清除已经删除的数据

db2 reorg table 表名

db2 runstats on table <模式名>.<表名> with distribution and detailed indexes all

数据库使用一段时间，数据空间会变得越来越大，一些delete掉的数据任存在数据库中，占用数据空间，影响数据库性能。因此需要定期运行reorg、runstats命令，清除已经删除的数据

db2   reorg   table 表名

db2   runstats on table [模式名].[表名] with distribution and detailed indexes all

#### runstats的作用

一个SQL在写完并运行之后，其实我们只是告诉了DB2去做什么，而不是如何去做。而，具体的如何去做，就取决于优化器。优化器为了生成最优的执行计划，就得掌握当前的系统信息，目录中的统计信息等等。

runstats命令就是用来收集数据库对象的状态信息，这对优化器生成最优的执行计划至关重要。

#### 什么时候需要runstats

·在给表创建一个index后，我们最好做一次runstat，否则可能index没有生效。不过有说法称在8.2版本以后的DB2中，会在INDEX之后自动进行runstats；

·.在对table做了一次reorg后，记得要做一次runstats。因为对表做reorg，会修改表的很多信息，比如高水位等，所以做一次runstats,可以更新统计信息。

·当表里数据发生了比较大的变化，一般来说，大约表里面的数据量的10%-20%发生了变化，就应该作一次runstats。这些变化包括删除，修改，插入。对于一些非常大的表，比方在数据仓库的项目里面，某些事实表非常巨大。这个时候，完整的对一个大表作runstats可能花费时间相当大，DB2 8.1里面支持我们对这些大表作抽样，比方说只对20%的数据作runstats，这样的话，一般来说也能保证得到正确的执行计划。当然首先要确保这个表里面的数据最好分布比较均匀。

·.当你在分区(DPF)数据库里面使用了REDISTRIBUTE DATABASE PARTITION GROUP这个命令，那么就需要用runstats来收集新的统计信息。

## db2set 查看或设置客户端db2变量的值

db2set -all 可以查看数据库的codepage编码

db2set db2codepage=1386 设置数据库codepage编码为简体中文

设置完后需重启数据库生效

db2set -all查看客户端codepage的值要和用db2 get db cfg 查看的值一样

## 查看数据库的编码

db2 get db cfg|grep “code set”

## 查看数据库配置

db2 get db cfg

## db2 客户端查询返回结果超时selectForRecvTimeout解决方法

db2set

DB2客户端配置参数 DB2TCP\_CLIENT\_RCVTIMEOUT

 这个参数的作用是：如果客户端等了这么久，还没有收到服务端的返回结果，那么客户端就会终止掉连接。

DB2TCP\_CLIENT\_RCVTIMEOUT=0表示不设置超时时间

## 查看数据库的所有连接

db2 list application

## 查看数据库中系统表

db2 list tables for system

## 查看数据库目录

db2 list db directory

## 查看节点目录

db2 list node directory

## 查看表空间tablespaces

db2 list tablespaces show detail

db2 list tablespaces

db2 list tablespace containers for 2  show detail 查看tablespace id=2使用容器所在目录

## 查看数据库包packages

db2 list packages for all

## 绑定数据库bind文件

db2 bind \*.bnd

## 查看表结构

db2 describe table 表名 show detail

db2 describe table tablename

## 查看表的索引

db2 describe indexes for table 表名 show detail

db2 describe indexes for table 表名

## 创建索引

### 非唯一索引

create index ind\_empno on emp(empno)；

create index 索引名 on 表名(列名)

### 唯一索引

create unique index ind\_empno on emp(empno)；

create unique index 索引名 on 表名(列名)

### 簇索引

drop index ind\_emp;

create index ind\_emp on emp(empno) cluster;

create index 索引名 on 表名(列名) cluster

### 唯一簇索引

create unique index ind\_sal on u\_emp(sal) cluster;

create unique index 索引名 on 表名(列名) cluster

## 删除索引

drop index 索引名;

## [深入理解DB2索引（Index）](http://blog.csdn.net/idber/article/details/8096941)

索引（Index）是[数据库](http://lib.csdn.net/base/14)管理系统中一个非常重要的数据结构，索引的合理使用能够极大提高数据库系统的性能。那么，什么是索引？索引有时如何提高数据库系统性能的呢？

阅读本文时建议参考：《[深入理解数据库磁盘存储（Disk Storage）](http://blog.csdn.net/idber/article/details/8087473)》

1.主键一定是唯一性索引，唯一性索引并不一定就是主键；

2.一个表中可以有多个唯一性索引，但只能有一个主键；

3.主键列不允许空值，而唯一性索引列允许空值。

索引概念

以一本书为例，通常一本书开头会有目录，而后才是正文，通过目录中每行左侧的标题和右侧的页码，我们可以快速定位到需要阅读的页面，而无需一页一页翻阅到该页面。数据库中的索引就像目录，它能帮助数据库管理系统快速定位到表中符合查询条件的数据行。索引实际上也是数据表的组成部分之一（数据表的存储包括数据页面+索引页面）

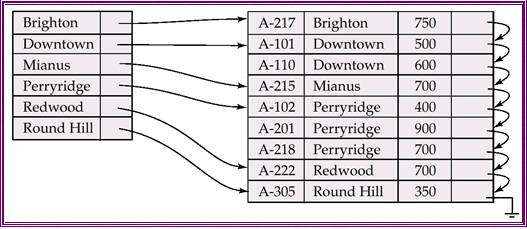
定义：数据库索引实际上是一个单独的、物理的数据库结构，它是某个表中一列或若干列值的集合和相应的指向表中物理标识这些值的数据页的逻辑指针的清单（序列）。

从索引的定义中可以得到：

1.索引是一个物理数据结构，也就是说，它是需要保持到物理磁盘上的，和普通的表数据一样要占用磁盘空间，也是存储在数据页上的。

2.索引是一个清单（或者说序列），由两部分组成：数据库表中的一列或多利的列值的集合、指向这些列值的数据页的逻辑指针。

可以把索引看作是一张只有两列的表，一列是普通数据表的列值（key-value），另一列是该列值的行对应的数据页的逻辑指针（Row-Pointer），这个逻辑指针可以理解为就是RID。需要说明的是，根据数据库产品和索引类型的不同，逻辑指针的结构也各不相同。如下图是索引的一个概览性描述，右侧是数据表页，左侧是对应的索引页。还有一点，从图上可以看到，表中的行之间是有指针相连的，即数据页中的每个记录除了存放数据，还会包含一个指针，指向其下一行记录。各记录形成一种链表结构。当然，这种结构并不是所有数据库系统都会采用的。



索引怎样提高性能

考虑这样一种情况：如上图，银行数据库的account表有编号（ID），城市（City）和余额（Balance）三列。表中一共有10万行数据。现在要列出属于Mianus这个城市的所有账号，查询语句为：

Select \* From account Where city=‘Mianus’

为了找出满足条件的查询，数据库管理器必须扫描account表中的所有行，逐行匹配，即表扫描。这会向缓冲池调入大量的数据页，执行大量I/O操作无疑是非常影响性能的。

如果在city列建立索引，则DB2查询优化器会在索引中扫描匹配的行，然后根据该行的逻辑指针找到那些满足条件的表数据行并将对应的数据页调入缓冲池，从而大大减少I/O操作。【一般而言，索引是可以常驻缓冲池的，所以对索引的扫描可以无需进行耗时的I/O操作】

当然上面的索引工作机制的描述是直观性的，真正的索引工作机制下文再讨论。

索引的分类--按结构分

索引按照结构可以分为有序索引（ordered index）和散列索引（hash index）两种基本类型。其中有序索引是基于值的顺序排序，根据值的排序进行索引值的查找。而散列索引则是基于将值平均分布到若干散列桶（hash bucket）。根据散列函数确定索引值所在的散列桶。

有序索引

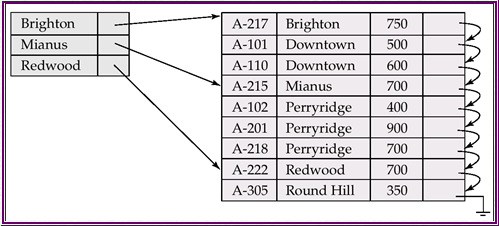
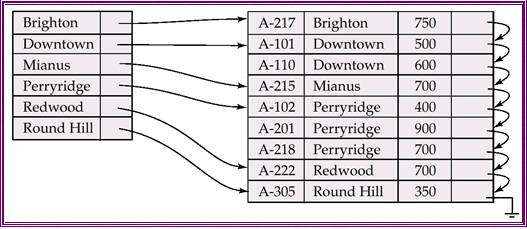
顺序文件索引

有序索引的一种最简单实现形式是顺序文件索引。顺序文件索引结构更加类似一张两列表，所有的索引值顺序排列，进行查询的时候逐行扫描索引中的各行记录，找到匹配项后就能根据逻辑指针找到表数据行的数据页。

那么，是不是建立索引的列的所有列值都必须成为索引中的索引值呢？不是。

稠密索引（dense index）和稀疏索引（sparse index）

我们将表中所有列值建立一个索引记录的索引称为稠密索引（dense index），将只为某些列值建立索引记录的索引称为稀疏索引（sparse index），如图：



稠密索引与 稀疏索引

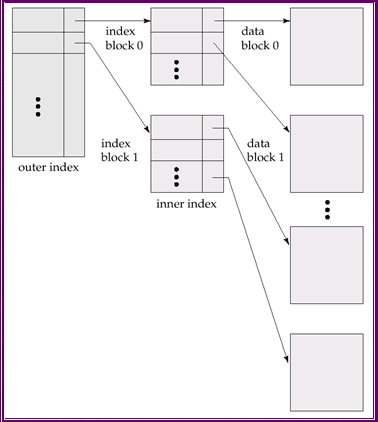
左侧的稠密索引很好理解，右侧的稀疏索引是怎样工作的呢？该索引只为Brighton、Mianus，和Redwood建立的索引记录，当需要查找Downtown时，由于顺序上Downtown在Brighton和Mianus之间，所以就根据Brighton这条记录的指针找到Brighton的数据页，从Brighton这一行记录开始根据记录的指针逐个查找，直到找到全部Downtown的行记录为止。

很显然，对于稀疏索引，数据行在数据页中必须按照索引记录的顺序而顺序排列才可能有效，否则有的列值就可能找不到或者找不全。但是和稠密索引相比，稀疏索引有着节省存储空间的优势。

多级索引

有时候，即便使用稀疏索引，由于数据量大且索引列上的列值多，索引本身也会变得非常大而难于有效处理（索引过大就难以保证常驻内存，进行磁盘读取的话，索引越大，需要的I/O越多），为了处理这种问题，我们使用了多级索引技术，其思想就是对索引建立索引。以一个二级索引为例，先根据外层索引定位到内层索引相应位置，在根据内层索引定位到表数据的数据页。这样的多级索引结构肯定会使得外层索引中的索引记录会少很多，如何便可以只将外层索引常驻内存从而达到减少I/O的目的了。

当然，无论是哪一级索引，都既可以是稠密索引也可以是稀疏索引（最内层可能只能是稠密索引）。实际上，索引记录本身就是顺序存储的，所以建立在索引上的索引通常是稀疏索引，这样才能更好的发挥多级索引的作用。

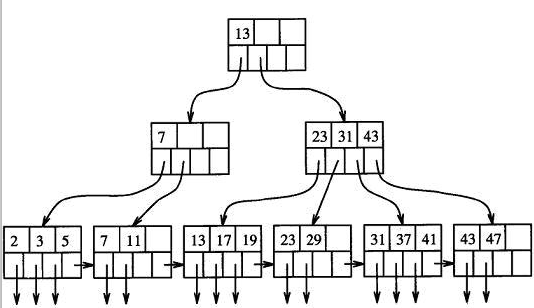


B+树索引

顺序文件索引的最大缺点在于，随着表数据行的增大和索引的增大，索引查找性能（多级索引的复杂度，索引扫描量）和数据顺序扫描性能（数据在增删改等操作中会变得非常混乱，难以维护）都会下降。虽然这种性能下降可以通过表重组和索引重组解决，但是频繁的重组同样是我们不愿意看到的。

因此，一种新的索引结构被广泛接受，这就是B+树索引。B+树是B树（平衡查找树，一种多路查找树）的一个重要变种（参考：[B 树、B- 树、B+ 树和B\* 树](http://blog.csdn.net/idber/article/details/8087521)），B+树索引结构是使用最广泛，在数据插入和删除的情况下仍能保持其执行效率的几种索引结构之一。目前DB2的索引结构就是B+树索引，Oracle的普通索引结构是B+树索引的变种B\*树索引。

B+树索引采用平衡树（balanced tree）结构，其中根结点到每个叶结点的路径长度都是相同的（所有叶结点都在同一层），树中除根结点外，每个非叶结点有[n/2] （这里的[]符号表示取天棚，如[3/2]=2。天棚符号显示不出。下同）到n个子结点，每个页结点有[(n-1)/2]到n-1个索引值（key-value，我们称为搜索码：search key）。至于n是什么，后面会解释。一个B+树的例子：其中n=4，索引值（搜索码）是2,3,5,7,11,13,17,19,23,29,31,37,41,43,47.



B+树索引结构

既然DB2的索引是B+树索引，那么B+树索引在磁盘上到底是怎样存储的呢？

前面介绍过，索引和表数据行一样，存储在数据页中。事实上，一个索引数据页就是索引B+树的一个结点！

每个结点在数据页中的存储方式是这样的：

http://img.my.csdn.net/uploads/201210/22/1350903638_2620.png

其中P1，P2，.......，Pn是n个指针；K1，K2，......，Kn-1是n-1个搜索码（索引值），这n-1个搜索码是按照从小到大的顺序排列的。

对于非叶结点：

P1指向另一个结点数据页，这个结点数据页中的所有搜索码值都小于K1（按某种排序规则规定的大于小于关系，比如Brighton<Downtown）；

P2指向另一个结点数据页，这个结点数据页中的所有搜索码值都大于或等于K1且都小于K2；

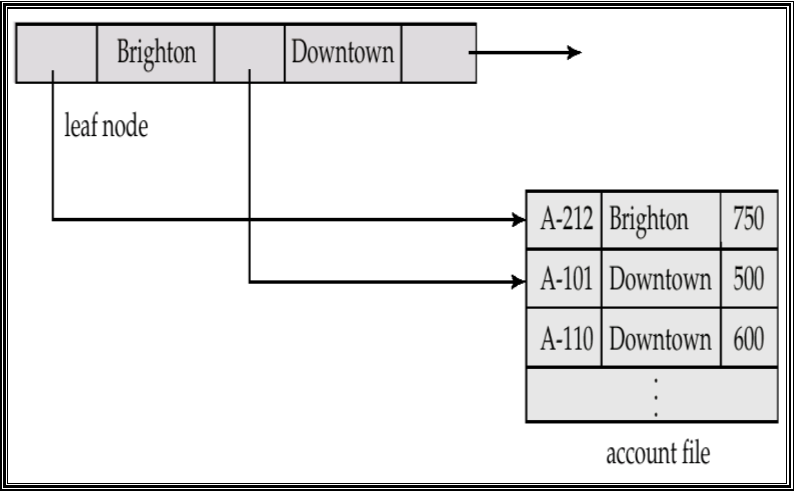
依此类推，Pn指向的结点数据页中所有搜索码值都大于或等于Kn-1。

而对于叶结点：

P1就是K1搜索码所对应的表数据行的逻辑指针，直接指向数据行的地址（页号+槽号）；

同理，P2指向K2搜索码数对应的表数据行的逻辑指针，......，Pn-1指向Kn-1搜索码数对应的表数据行的逻辑指针；

最后的Pn指针则指向该B+树中的下一个页结点。



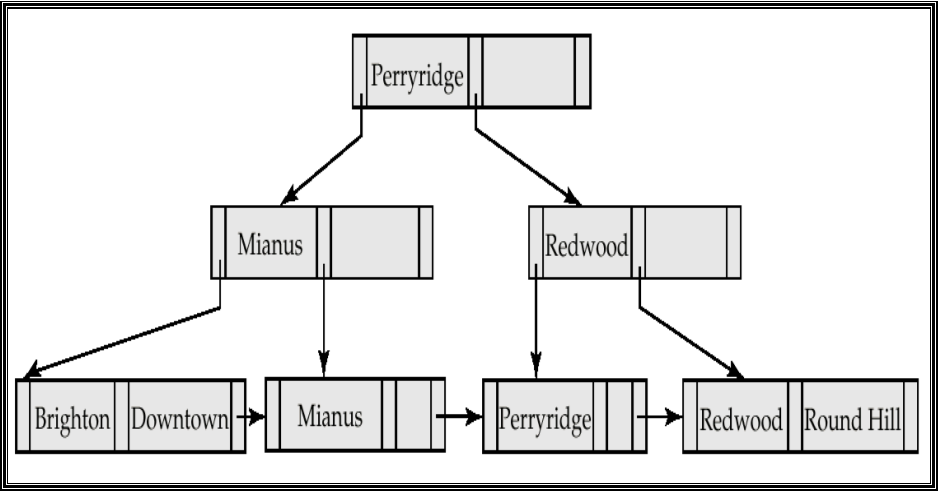
B+树索引分析

我们认为一个（指针，搜索码）对，即一个（P，K）对是一个索引项（index entry，一个索引项就是索引数据页中的一条记录）。假设一个B+树索引的一个索引项大小为40B，一个数据页中可以存储的索引项数大致是4KB/40B=100个。那么，n=100。这就是B+树索引中的n的来历。它完全是由索引数据页大小和索引项大小决定的。（当然，这个例子中按计算值来说，n应该等于101，这样才是100个搜索码，101个指针，才有100个索引项。但是由于页面空间利用和方便计算的原因，直接取个大概值100）

按照树结构的层数公式：层数=http://img.my.csdn.net/uploads/201210/22/1350905825_7054.png (K表示总共的搜索码数量，m表示每个结点的子结点数)。由于非叶结点的子结点数为[n/2]到n-1，那么如果所有非叶结点的子结点都是[n/2]个，B+树索引的层数为http://img.my.csdn.net/uploads/201210/22/1350905921_1433.png；如果所有非叶结点的子结点都是n-1个，索引B+树的层数则为http://img.my.csdn.net/uploads/201210/22/1350906014_5246.png。也就是说，B+树索引的层数最多为http://img.my.csdn.net/uploads/201210/22/1350905921_1433.png层。由于B+树根结点到所有叶结点的路径都一样长，而且所有搜索码都在叶结点上，所以，查找任意一个搜索码所需要的路径长度都是一样的。

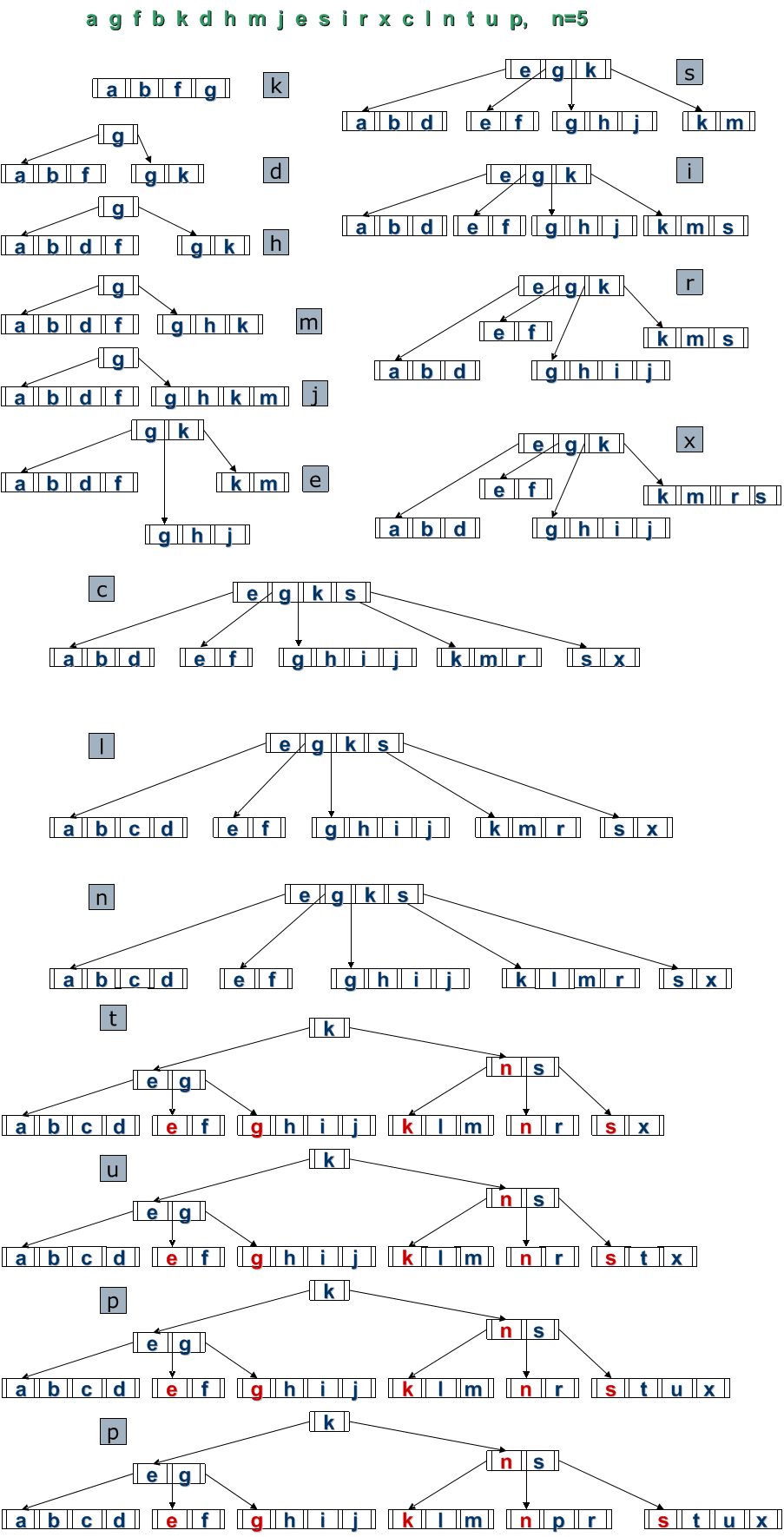
继续假设一个表中建立了索引的列上有100万个不同的列值，即有100万个搜索码（索引值），则B+树索引的层数最高为http://img.my.csdn.net/uploads/201210/22/1350906506_1410.png层。可见，通常DB2B+树索引的层数不会超过3层。（100万个不同的搜索码才4层，很少需要在这样的列上建立索引。除非是主键。主键默认是有唯一性索引的）层数越低表示查找到所需的搜索码越快，定位相应的数据行的物理位置越快。

下面以n=3为例看看上面account表的city列的索引的结构是怎样的：



B+树索引生成

下面的图详细演示了一棵n=5的B+树索引的生成过程（叶结点之间的指针未标出）：



关于B+树的生成规则以及结点中搜索码的添加，删除和结点的合并分裂规则这里不作介绍，有空再写。

从B+树的结构和相关规则可以看出，B+树在插入和删除操作方面会相对复杂，不能会带来性能开销，还会增加空间开销。但是由于数据库中数据的查询操作通常是远多于删除和插入操作的，也就是说，B+树索引带来的稳定高性能查询优势是其他索引结构无法比拟的。插入和删除带来的开销实际上也是可以接受的，因为它减小了表重组和索引重组的代价。此外，由于结点有可能是半空的，这会造成空间的浪费。但是这种浪费依然是可接受的。

多值指针

还有一个问题在之前的讨论中一直被回避着，那就是，如果一个索引值（搜索码）对应着表中多个数据行（这种情况是极为普遍的，比如上面account表中的Perryridge），那么，该叶结点的指针该指向谁呢？

如果是顺序文件索引，可以通过指向第一个满足条件的行，再根据各行的next指针来遍历所有满足条件的行（这一点上文有解释）。可是B+树索引并没有这样的行间next指针结构。

包括Oracle数据库在内的大多数数据库产品采用的策略是使用散列桶（hash bucket）。即对于一个搜索码对应多个数据行的情况，其叶结点指针指向一个散列桶，再通过遍历散列桶来获取所有满足条件的行地址。（这个策略类似B树索引结构）

而DB2则是利用其优秀的索引压缩技术而采用了一种新的解决方案，DB2使用一种逻辑块结构来解决搜索码值重复的问题，如图：



一个逻辑块包含一个前缀（Prefix，大小为2B），一个搜索码（Keyval）和一个与该搜索码对应的RID（大小为4B）列表。规定一个逻辑块中的搜索码后跟的RID的个数最多为255个。假设搜索码A有N个不同的数据行与之对应，则意味着一个搜索码对应N个RID。如果N<=255，则一个逻辑块中能够包含该搜索码以及其全部RID。如果N>255，则需要另外生成一个逻辑块，该逻辑块中的搜索码仍然是之前的搜索码A，RID列表中为剩下的RID。如果仍然无法容纳剩余的全部RID，则继续生成该搜索码的逻辑块，直到能够全部容纳为止。多个逻辑块则以前缀Prx来进行区分（前缀不仅能区分同一搜索码的不同逻辑块，还能区分不同搜索码的逻辑块）。如此，一个逻辑块类似于一个索引项。但是这里的索引项跟上面无重复搜索码的索引项是有着本质的区别的。

找到一篇详细介绍DB2索引压缩技术的论文，有时间再翻一翻：

<http://www2.hawaii.edu/~lipyeow/pub/vldb09-indexcompression.pdf> （很值得研究的一篇论文）

散列索引

有序索引的一个缺点是必须访问索引结构来定位数据，或者必须使用多路查找，这些都导致相对过多的I/O操作。而基于散列（hashing）技术的索引则能够避免访问索引结构。介绍散列索引前先理解一下散列的相关概念。

静态散列

散列技术中有一个重要的概念，那就是桶（bucket）。桶表示能够存储一条或多条记录的存储单位。通常一个桶就是一个磁盘块，但也可能大小不定。我们用K表示所有搜索码的集合，用B表示所有桶的集合。散列技术的思想就是：将所有的搜索码K及其相关信息按照某种规则（称这种规则为散列函数h）分散到各个散列桶中（就是一种映射关系）。当查找某个搜索码时，就可以根据规则计算出该搜索码所在的桶，然后在桶中找到搜索码及其信息。这种方法可以在一开始就定位到小范围内进行查找工作，效率可想而知是很高的。

散列函数

散列技术所要解决的首要问题就是散列函数的确定了。不好的散列函数可能导致所有搜索码只被分配到单独几个散列桶中，这既导致散列桶空间的浪费，也使得散列查找的优势丧失（不得不在一个桶中查找大量记录）。良好的散列函数至少需要满足两大特性：分布是均匀的；分布是随机的。散列函数的确定是非常灵活的，可以非常简单（比如简单求余，根据余数的不同将搜索码分配到不同桶中），也可以异常复杂。不作过多讨论。

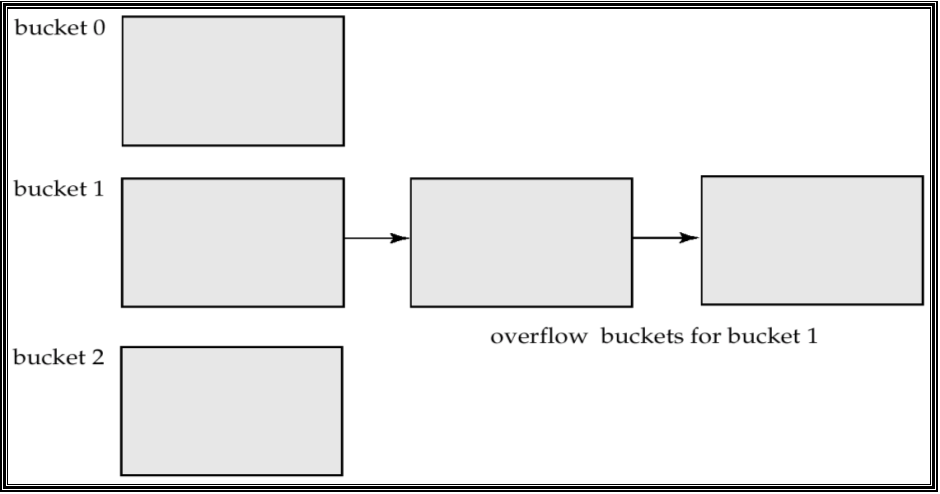
桶溢出

当插入一条记录，而根据散列函数映射到的散列桶已经没有存储空间存放该记录的搜索码，就会发生桶溢出（bucket overflow）。发生桶溢出的可能原因有：

1.桶不足（Insufficient bucket）。即桶的总数不够，当然这种问题一定程度上是由于散列函数的不合理造成的，它导致大量的搜索码通过散列函数只能映射到少数几个桶中。当然这也是无法完全避免的，任何散列函数都只能将搜索码映射到有限的桶中，任何桶的存储空间也是有限的。但是搜索码的数量可以是无限的。

2.偏斜（Skew）。即大量的搜索码分布到少数几个桶中，其他的桶中的记录很少或者没有。这种情况一方面也是由于散列函数选取不当，另一方面也可能是特定的搜索码集合本身具有一定的耦合性（比如多个记录具有相同的搜索码）。

解决这一问题的策略就是使用溢出桶（overflow bucket）。即如果一个搜索码映射到的桶已经满了，则为这个桶增加一个溢出桶，将这个搜索码存储在溢出桶中。如果溢出桶也满了，就再增加一个溢出桶，如此反复。散列桶和它的溢出桶通过链表连接起来，称为溢出链（overflow chaining）。



很显然，一个散列桶上挂的溢出桶越多，散列桶所拥有的优势也就丧失越多，因为不得不进行更多的查找操作。

以上的散列技术应用与索引就产生了散列索引了。

散列索引将搜索码及其相应的逻辑指针封装成一个散列结构，将散列函数作用于搜索码，将这些散列结构分配到不同的散列桶中。当进行查找时，根据搜索码找到装有含该搜索码的散列结构的散列桶，然后在散列桶中更加搜索码找到对应的散列结构，得到散列结构中的逻辑指针就能找到对应的表数据的数据页及槽地址了。

以一个桶能够存储2个封装了搜索码和逻辑指针的散列结构为例（以ID主码为搜索码）：



以上就是使用静态散列技术构造散列索引的方法了。

动态散列

静态散列索引的一个主要的缺陷是，随着散列函数的确定，桶地址集合B也就确定了。（比如将散列函数定义为对10求余，则散列桶就只有10个：0-9）。随着数据库的不断增大，就需要使用溢出桶处理溢出问题了，但过多的溢出桶会导致效率的明显下降。

一种处理方法是一开始就确定一个能使用足够多散列桶的散列函数，但这必然造成前期存储空间的巨大浪费。另一种方法是周期性的对散列结构进行重组。重新选择散列函数，重新分配桶，但这无疑是一个复杂而耗时的工作。

为了克服静态散列的缺陷，一些动态散列（dynamic hashing）技术被提出并应用与散列索引，以保证数据库的增大不会给性能带来大的影响。一种比较好的动态散列技术是可扩充散列（extendable hashing），当数据库增大或缩小时，通过桶的分裂或合并来适应数据库大小的变化。具体实现就不解释了。

### 索引的分类--按功能分

除了按照结构将索引分为有序索引和散列索引外，还可以按照索引功能将其分为唯一索引，非唯一索引，集群索引，非集群索引和MDC块索引这样的5种类型。

唯一索引（unique index）和非唯一索引（nonunique index）

实际上，索引的优势主要体现在两方面：提高查询效率和保证数据唯一性。其中保证数据唯一性的优势就是依靠唯一索引提供的。

唯一索引是指对某一列创建的索引必须保证每一个key-value（索引值，搜索码）只能对应一个Row-pointer。比如最开始的示意图中，一个Downtown的key-value就对应了两个表数据行的指针；一个Perryridge的key-value对应了3个表数据行的指针。这样的索引就是非唯一索引。

那么要保证索引是唯一索引，显然就必须保证对应表中的建立了索引的列上没有两行的列值是相同的。也就是说，唯一索引效果上等同于非唯一索引+索引列唯一性约束。

只有在表中某列上的所有列值都不相同是才能在该列上成功建立唯一索引。一旦在表中某列上建立了唯一索引，那么向表中插入的任何数据行都不允许在该索引列上出现重复值，否则插入失败。需要注意的是：创建了唯一索引的列上的数据允许为空，但是一旦有数据，就必须是唯一的。

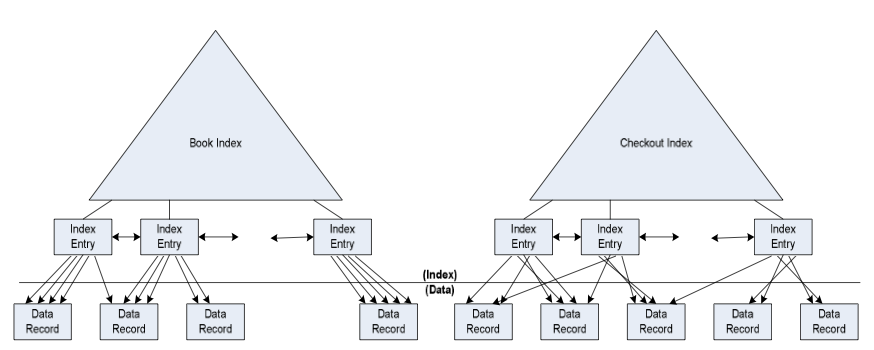
对于任何一张表，一旦指定了主键，那么数据库会默认在主键上创建一个唯一索引，有时称主键上的唯一索引为主键索引，主键索引其实就是在唯一索引和非空约束的组合实现（主键是不允许为空的）。另外，一旦为某个列建立了唯一性约束，数据库同样会默认在该列上创建一个唯一索引。

集群索引（clustered index）和非集群索引（nonclustered index）

概念

集群索引又称聚集索引，聚簇索引，其中聚簇索引是各数据库通用的常用的叫法，集群索引是DB2官方叫法。同时聚簇索引又称主索引（primary index）【注意：主索引并不表示该索引是建立在主键上的，事实上，主索引通常建立在非主键上】。聚簇索引的概念对于有序索引和散列索引都是有效的，但是对于有序索引才有意义。通常散列索引都是非聚簇索引。（当然散列聚簇索引也是存在的）

我们知道，无论是顺序文件索引还是B树、B+树、B\*树索引，这些有序索引中索引值（搜索码）都是有一定的排列顺序的。如果这些索引值（搜索码）对应的数据行在数据页存储空间中跟索引值的排列顺序是一致的，那么这样的索引就是聚簇索引。反之就是非聚簇索引（nonclustered index，非聚簇索引又称辅助索引：secondary index）。注意：这里只要求排列的顺序一致，并不要求索引值在索引数据页上是连续的，也不要求表数据行在常规数据页上是连续的。如图是DB2聚簇索引和非聚簇索引的示例：



很显然，由于聚簇索引要求数据页中的数据行顺序上与索引值的顺序保持一致，而数据行的排列顺序是一种物理上的顺序，不可能要求数据在磁盘上同时满足多种物理排序（就像不可能要求一群人人既按身高排队的同时又按照年龄排队一样），所以，一个表上只允许有一个聚簇索引。对于非聚簇索引则没有限制。

联系前面提到的稀疏索引和稠密索引的含义，可以知道，非聚簇索引必须是稠密索引，因为如果是稀疏索引，那么由于索引值与数据行排序的不一致，无法定位没有出现在索引值中的搜索码。

聚簇索引的优势

那么，既然聚簇索引有这么大的限制，聚簇索引存在的必要性在哪呢？

聚簇索引相较非聚簇索引唯一的优势是拥有更高的查询性能。参照上图，试想account表上的这样一条查询语句：

select \* from account where balance > 500

如果在balance列上建立了非聚簇索引，那么，balance > 500的行可能分布在各个数据页上，那么要查询到满足条件的所有行，就必须将大量的数据页调入缓冲池中（每一页上只有少数几行是满足要求的），而且，数据库的预取机制的效率就显得不怎么高。不仅如此，由于包含满足条件的行的数据页分散分布，数据在磁盘上很可能也分散到距离间隔比较大的扇区上。因此，这样的查询不仅I/O操作多而且I/O也更费时。

如果在balance列上建立的是聚簇索引，那么balance＞500的行很可能就分布在一个数据页内或者一个数据页中有大量满足条件的数据行，需要调入缓冲池的数据页就会少很多，预取机制也能较好的发挥作用。同时，数据在磁盘上也会分布在邻近的扇区。因此，这样的查询不仅I/O操作少 而且I/O相对省时。

聚簇率（clustering ratio）

使用聚簇索引固然有查询方面的优势，但是在数据插入方面就产生一个问题：新插入的数据应该放在哪个数据页的什么位置？由于聚簇索引要求数据行与索引值顺序保持一致，那新插入的数据行以及其索引值是否必须寻找相应位置执行插入操作呢？如果数据页中相应位置没有足够的空间插入该怎么办呢？可以想象，如果要保证顺序的严格一致，必然会导致大量的数据迁移，这样的花销是不可接受的。所以，真正的处理办法是允许有一定的顺序不一致出现，即便有些数据是无序的，仍然认为该索引是聚簇索引。

我们把索引中满足聚簇顺序条件的（索引值，数据行）对占该索引上所有（索引值，数据行）对的比例称为聚簇率，聚簇索引中85%以上的聚簇率是可接受的。否则就需要进行重组。

顺便提一个小技巧：

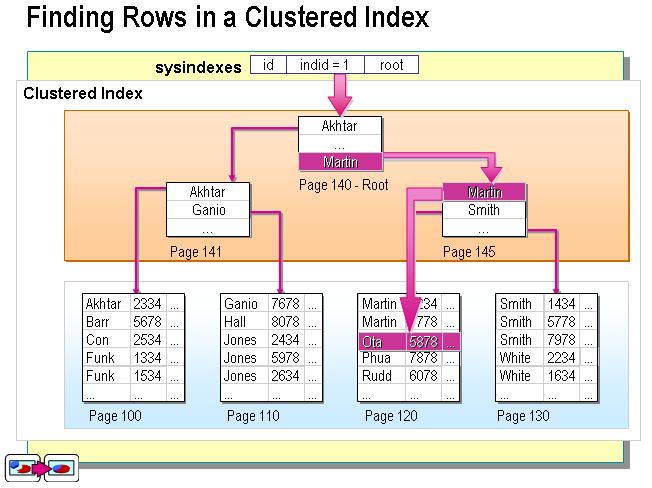
假设已经建好一张表，准备向表中导入大量数据，且要在某列上建立索引，那么：

如果是建立非聚簇索引，最好是先导入数据，然后建立索引。因为这样保证B+树索引建立时已经存在大量搜索码，无需在B+树生成后进行频繁的插入，合并，分裂操作。

如果是建立聚簇索引，最好先建立索引，然后导入数据。因为如果先导入了数据，再建索引，就必须对已经存在的数据进行耗时的重新排序。

聚簇索引的实现

还有一点需要说明，那就是，虽然各种数据库产品都有聚簇索引的概念，但具体的实现方式和索引结构也是有区别的。DB2的聚簇索引结构如上图所示，聚簇索引的叶结点和非聚簇索引一样，都是指向相应数据页的逻辑指针。即索引数据页和常规数据页是严格分开的。但是包括Oracle，SQL Server在内的大部分数据库的聚簇索引的页结点不是指向数据页的指针，而是页结点本身就是数据页。也就是说，索引数据页和常规数据页发生了融合，二者已经没有了严格的界限。如图（这是SQL Server数据库的聚簇索引和非聚簇索引的对比图，SQL Server数据库的索引是二叉树结构）：

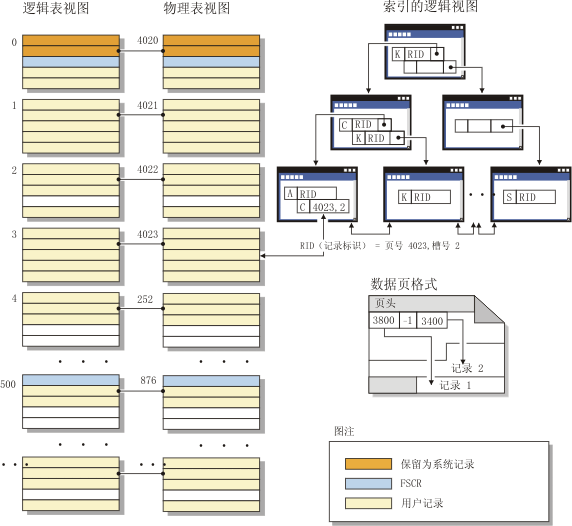


MDC块索引（MDC block index）

MDC块索引与多维集群表相关，还没有研究过。

标准表的表、索引和数据页的关系

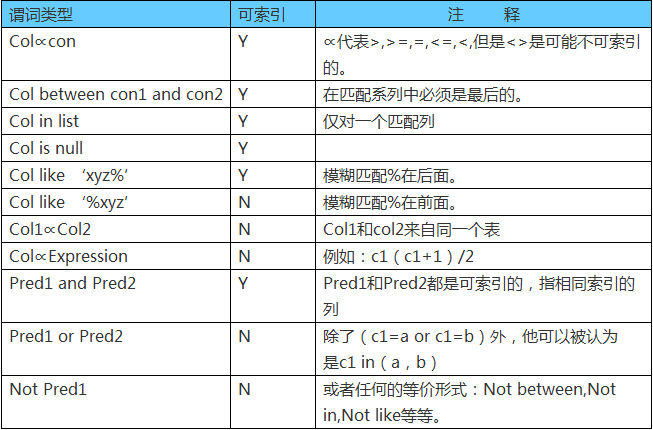
标准表是相对与多维集群表而言的，常规表就是标准表。其实标准表，索引和数据页的关系在这篇文章和《[数据库学习笔记----磁盘存储内部结构](http://blog.csdn.net/idber/article/details/8087473)》这两篇文章里已经解释得很清楚了，这里放上一张全景的关系结构图：



DB2索引优化

索引虽然能够大大提升查询效率，但是并不是对所有查询都适用的。比如对于“Select \* From account where balance != 500”这样的语句，balance列上的索引基本上是无效的。（这是很好理解的，“不等于”在B+树索引中怎么能查找呢？）

我们把查询语句中Where后面的表达式称为谓词。DB2中谓词能否使用索引的情况列表如下：



## 查看表： select \* from syscat.tables where Type = 'T'

## 查看用户： select \* from syscat.dbauth

## 查看权限： select \* from syscat.tabauth

## 断开与数据库的连接

db2 connect reset 或 db2 terminate

## 强制断开所有的数据库连接

db2 force application all

## 取出表中的前10条数据

select \* from <表名> fetch first 10 rows only

## char()函数

## trim()函数

## length()函数

## substr()函数

## coalesce(字段名，转换后的值)

coalesce(字段名，转换后的值),对是null的字段进行值转换，例如：select coalesce(id,1) from

<表名> 如果表中id字段为null，则将其值赋为1

## dayof week(日期)

dayofweek(日期):计算出日期中是周几(1是周日，2是周一，。。。。。,7是周六)

dayofweek\_iso(日期):计算出日期中是周几(1是周一，。。。。。,7是周日)

例如：dayofweek(date(2008-01-16))   --它会返回是4，代表星期三

         dayofweek\_iso(date(2008-01-16))   --它会返回是3，代表星期三

## dayofyear(日期)

一年中的第几天，范围在1-365范围之内，

注意参数中的日期格式是YYYY-MM-DD的形式，如果不是需要进行转换，否则函数不能用

## concat(参数1，连接值)

concat(参数1，连接值)，把参数1加上连接值组成一个新值

例如concat(‘aa’,’b’)，返回是aab

## SQL优化

### 1.mysql优化SQL步骤

#### 1. 通过 show status和应用特点了解各种 SQL的执行频率

       通过 SHOW STATUS 可以提供服务器状态信息，也可以使用 mysqladmin extende d-status 命令获得。 SHOW STATUS 可以根据需要显示 session 级别的统计结果和 global级别的统计结果。

       如显示当前session： SHOW STATUS like "Com\_%"; 全局级别：show global status;  
       以下几个参数对 Myisam 和 Innodb 存储引擎都计数：

1. Com\_select 执行 select 操作的次数，一次查询只累加 1 ；

2. Com\_insert 执行 insert 操作的次数，对于批量插入的 insert 操作，只累加一次 ；

3. Com\_update 执行 update 操作的次数；

4. Com\_delete 执行 delete 操作的次数；

以下几个参数是针对 Innodb 存储引擎计数的，累加的算法也略有不同：

1. Innodb\_rows\_read select 查询返回的行数；

2. Innodb\_rows\_inserted 执行 Insert 操作插入的行数；

3. Innodb\_rows\_updated 执行 update 操作更新的行数；

4. Innodb\_rows\_deleted 执行 delete 操作删除的行数；

       通过以上几个参数，可以很容易的了解当前[**数据库**](http://lib.csdn.net/base/14)的应用是以插入更新为主还 是以查询操作为主，以及各种类型的 SQL大致的执行比例是多少。对于更新操作的计 数，是对执行次数的计数，不论提交还是回滚都会累加。  
      对于事务型的应用，通过 Com\_commit 和 Com\_rollback 可以了解事务提交和回 滚的情况，对于回滚操作非常频繁的数据库，可能意味着应用编写存在问题。此外，以下几个参数便于我们了解数据库的基本情况：

1. Connections 试图连接 [**MySQL**](http://lib.csdn.net/base/14) 服务器的次数  
2. Uptime 服务器工作时间  
3. Slow\_queries 慢查询的次数

#### 2. 定位执行效率较低的SQL语句

     可以通过以下两种方式定位执行效率较低的 SQL 语句：  
     **1. 可以通过慢查询日志定位那些执行效率较低的 sql 语句**，用 --log-slow-queries[=file\_name] 选项启动时， mysqld 写一个包含所有执行时间超过long\_query\_time 秒的 SQL 语句的日志文件。可以链接到管理维护中的相关章节。  
     **2. 使用 show processlist查看当前MYSQL的线程，** 命令慢查询日志在查询结束以后才纪录，所以在应用反映执行效率出现问题的时候查 询慢查询日志并不能定位问题，可以使用 show processlist 命令查看当前 MySQL 在进行的线程，包括线程的状态，是否锁表等等，可以实时的查看 SQL 执行情况， 同时对一些锁表操作进行优化。

#### 3. 通过EXPLAIN 分析低效 SQL的执行计划

       通过以上步骤查询到效率低的 SQL 后，我们可以通过 explain 或者 desc 获取MySQL 如何执行 SELECT 语句的信息，包括 select 语句执行过程表如何连接和连接 的次序。

### 2. MySQL索引

#### 1. mysql如何使用索引

       索引用于快速找出在某个列中有一特定值的行。对相关列使用索引是提高SELECT 操作性能的最佳途径。  
       查询要使用索引最主要的条件是查询条件中需要使用索引关键字，如果是多列 索引，那么只有查询条件使用了多列关键字最左边的前缀时（**前缀索引**），才可以使用索引，否则 将不能使用索引。

       下列情况下， Mysql 不会使用已有的索引：  
      1、如果 mysql 估计使用索引比全表扫描更慢，则不使用索引。例如：如果 key\_part 1均匀分布在 1 和 100 之间，下列查询中使用索引就不是很好：  
        SELECT \* FROM table\_name where key\_part1 > 1 and key\_part1 < 90  
      2、如果使用**heap 表**并且 where 条件中不用＝索引列，**其他 > 、 < 、 >= 、 <= 均不使 用索引（MyISAM和innodb表使用索引）**；

      3、使用or分割的条件，如果or前的条件中的列有索引，后面的列中没有索引，那么涉及到的索引都不会使用。  
      4、如果创建复合索引，如果条件中使用的列不是索引列的第一部分；**（不是前缀索引）  
      4、如果 like 是以％开始；**  
      5、对 where 后边条件为字符串的一定要加引号，字符串如果为数字 mysql 会自动转为字符串，但是不使用索引。

#### 2. 查看索引使用情况

        如果索引正在工作， Handler\_read\_key 的值将很高，这个值代表了一个行被索引值读的次数，很低的值表明增加索引得到的性能改善不高，因为索引并不经常使 用。  
Handler\_read\_rnd\_next 的值高则意味着查询运行低效，并且应该建立索引补救。这个值的含义是在数据文件中读下一行的请求数。如果你正进行大量的表扫描，  
该值较高。通常说明表索引不正确或写入的查询没有利用索引。

       语法：  
        **mysql> show status like 'Handler\_read%';**

### 3. 具体优化查询语句

##### 1. 查询进行优化，应尽量避免全表扫描

    对查询进行优化，应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引

**.**    尝试下面的技巧以避免优化器错选了表扫描：

**·**  使用ANALYZE TABLE*tbl\_name*为扫描的表更新关键字分布。

**·** 对扫描的表使用FORCEINDEX告知MySQL，相对于使用给定的索引表扫描将非常耗时。

            SELECT \* FROM t1, t2 FORCE INDEX (*index\_for\_column*)   WHERE t1.*col\_name*=t2.*col\_name*；

**·** 用--max-seeks-for-key=1000选项启动**mysqld**或使用SET max\_seeks\_for\_key=1000告知优化器假设关键字扫描不会超过1,000次关键字搜索。

###### 1). 应尽量避免在 where 子句中对字段进行 null 值判断

        否则将导致引擎放弃使用索引而进行全表扫描，如：

        select id from t where num is null

        NULL对于大多数数据库都需要特殊处理，MySQL也不例外，它需要更多的代码，更多的检查和特殊的索引逻辑，有些开发人员完全没有意识到，创建表时NULL是默认值，但大多数时候应该使用NOT NULL，或者使用一个特殊的值，如0，-1作为默  认值。

        不能用null作索引，任何包含null值的列都将不会被包含在索引中。即使索引有多列这样的情况下，只要这些列中有一列含有null，该列    就会从索引中排除。也就是说如果某列存在空值，即使对该列建索引也不会提高性能。 任何在where子句中使用is null或is not null的语句优化器是不允许使用索引的。

        此例可以在num上设置默认值0，确保表中num列没有null值，然后这样查询：

         select id    from t where num=0

###### 2). 应尽量避免在 where 子句中使用!=或<>操作符

         否则将引擎放弃使用索引而进行全表扫描。  
         MySQL只有对以下操作符才使用索引：**<，<=，=，>，>=，BETWEEN，IN**，以及某些时候的LIKE。

        可以在LIKE操作中使用索引的情形是指另一个操作数不是以**通配符（%或者\_）开头**的情形。例如:  
         SELECT id FROM  t WHERE col LIKE 'Mich%'; #  这个查询将使用索引，  
         SELECT id FROM  t WHERE col  LIKE '%ike';   #这个查询不会使用索引。

###### 3). 应尽量避免在 where 子句中使用 or 来连接条件

        否则将导致引擎放弃使用索引而进行全表扫描，如：

        select id from t where num=10 or num=20

        可以 使用UNION合并查询： select id from t where num=10 union all select id from t where num=20

       在某些情况下，or条件可以避免全表扫描的。

**1 .where 语句里面如果带有or条件, myisam表能用到索引， innodb不行。**

**2 .必须所有的or条件都必须是独立索引**

[mysql or条件可以使用索引而避免全表](http://blog.csdn.net/hguisu/article/details/7106159)

###### 4) .in 和 not in 也要慎用，否则会导致全表扫描，

        如：

        select id from t where num in(1,2,3)

        对于连续的数值，能用 between 就不要用 in 了：

        Select id from t where num between 1 and 3

###### 5). 以%或\_开头的like查询也将导致全表扫描：

        select id from t where name like '%abc%' 或者

        select id from t where name like '%abc' 或者

        若要提高效率，可以考虑全文检索。

        而select id from t where name like 'abc%' 才用到索引

###### 6). 如果在 where 子句中使用参数，也会导致全表扫描。

       因为SQL只有在运行时才会解析局部变量，但优化程序不能将访问计划的选择推 迟到运行时；它必须在编译时进行选择。然而，如果在编译时建立访问计划，变量的值还是未知的，因而无法作为索引选择的输入项。如下面语句将进行全表扫描：

       select id from t where num=@num

       可以改为强制查询使用索引： select id from t with(index(索引名)) where num=@num

###### 7). 应尽量避免在 where 子句中对字段进行表达式操作

       这将导致引擎放弃使用索引而进行全表扫描。如：

       select id from t where num/2=100

       应改为:  select id from t where num=100\*2

###### 8).应尽量避免在where子句中对字段进行函数操作

       这将导致引擎放弃使用索引而进行全表扫描。如：

      select id from t where substring(name,1,3)='abc'   --name

      select id from t where datediff(day,createdate,'2005-11-30')=0--‘2005-11-30’

      生成的id 应改为:

      select id from t where name like 'abc%'

      select id from t where createdate>='2005-11-30' and createdate<'2005-12-1'

###### 9).不要在 where 子句中的“=”左边进行函数、算术运算或其他表达式运算

      否则系统将可能无法正确使用索引。

###### 10).索引字段不是复合索引的前缀索引

       例如 在使用索引字段作为条件时，如果该索引是复合索引，那么必须使用到该索引中的第一个字段作为条件时才能保证系统使用该索引，否则该索引将不会被使用，并且应尽可能的让字段顺序与索引顺序相一致。

###### 11).不要写一些没有意义的查询，

        如需要生成一个空表结构：

        select col1,col2 into #t from t where 1=0

        这类代码不会返回任何结果集，但是会消耗系统资源的，应改成这样： create table #t(...)

###### 12).很多时候用 exists 代替 in 是一个好的选择：

       select num from a where num in(select num from b)

       用下面的语句替换：

       select num from a where exists(select 1 from b where num=a.num)

###### 13).并不是所有索引对查询都有效

       SQL是根据表中数据来进行查询优化的，当索引列有大量数据重复时，SQL查询可能不会去利用索引，如一表中有字段sex，male、female几乎各一半，那么即使在sex上建了索引也对查询效率起不了作用。

###### 14).索引并不是越多越好

       索引固然可以提高相应的 select 的效率，但同时也降低了 insert 及 update 的效率，因为 insert 或 update 时有可能会重建索引，所以怎样建索引需要慎重考虑，视具体情况而定。一个表的索引数最好不要超过6个，若太多则应考虑一些不常使用到的列上建的索引是否有必要。

###### 15).应尽可能的避免更新 clustered 索引数据列，

       因为 clustered 索引数据列的顺序就是表记录的物理存储顺序，一旦该列值改变将导致整个表记录的顺序的调整，会耗费相当大的资源。若应用系统需要频繁更新 clustered 索引数据列，那么需要考虑是否应将该索引建为 clustered 索引。

###### 16).尽量使用数字型字段，

      若只含数值信息的字段尽量不要设计为字符型，这会降低查询和连接的性能，并会增加存储开销。这是因为引擎在处理查询和连接时会逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了。

###### 17).尽可能的使用 varchar/nvarchar 代替 char/nchar ，

      因为首先变长字段存储空间小，可以节省存储空间，其次对于查询来说，在一个相对较小的字段内搜索效率显然要高些。

###### 18).最好不要使用"\*"返回所有： select \* from t ，

     用具体的字段列表代替“\*”，不要返回用不到的任何字段。

###### 19).尽量使用表变量来代替临时表。

    如果表变量包含大量数据，请注意索引非常有限（只有主键索引）。

###### 20).避免频繁创建和删除临时表，以减少系统表资源的消耗。

###### 21).临时表并不是不可使用，

     适当地使用它们可以使某些例程更有效，例如，当需要重复引用大型表或常用表中的某个数据集时。但是，对于一次性事件，最好使用导出表。

###### 22).在新建临时表时，如果一次性插入数据量很大，那么可以使用 select into 代替 create table，避免造成大量 log ，以提高速度；

     如果数据量不大，为了缓和系统表的资源，应先create table，然后insert。

###### 23).如果使用到了临时表，在存储过程的最后务必将所有的临时表显式删除，先 truncate table ，然后 drop table ，这样可以避免系统表的较长时间锁定。

### 4. 游标的问题

#### 1).尽量避免使用游标，

      因为游标的效率较差，如果游标操作的数据超过1万行，那么就应该考虑改写。

#### 2).使用基于游标的方法或临时表方法之前

      应先寻找基于集的解决方案来解决问题，基于集的方法通常更有效。

#### 3).与临时表一样，游标并不是不可使用

     对小型数据集使用 FAST\_FORWARD 游标通常要优于其他逐行处理方法，尤其是在必须引用几个表才能获得所需的数据时。在结果集中包括“合计”的例程通常要比使用游标执行的速度快。如果开发时间允许，基于游标的方法和基于集的方法都可以尝试一下，看哪一种方法的效果更好。

#### 4).在所有的存储过程和触发器的开始处设置 SET NOCOUNT ON ，在结束时设置 SET NOCOUNT OFF

      无需在执行存储过程和触发器的每个语句后向客户端发送 DONE\_IN\_PROC 消息。

### 5. 事务的问题

尽量避免大事务操作，提高系统并发能力。

### 6. 数据量的问题

尽量避免向客户端返回[大数据](http://lib.csdn.net/base/20)量，若数据量过大，应该考虑相应需求是否合理。

### 7. COUNT优化

#### 1)count(\*) 优于count(1)和count(primary\_key)

　　很多人为了统计记录条数，就使用 count(1) 和 count(primary\_key) 而不是 count(\*) ，他们认为这样性能更好，其实这是一个误区。对于有些场景，这样做可能性能会更差，因为数据库对 count(\*) 计数操作做了一些特别的优化。

#### 2）count(column) 和 count(\*) 是不一样的

　　这个误区甚至在很多的资深工程师或者是 DBA 中都普遍存在，很多人都会认为这是理所当然的。实际上，count(column) 和 count(\*) 是一个完全不一样的操作，所代表的意义也完全不一样。

　　count(column) 是表示结果集中有多少个column字段不为空的记录

　　count(\*) 是表示整个结果集有多少条记录

##### 1.innodb引擎在统计方面和myisam是不同的，Myisam内置了一个计数器，

 Count(\*)在没有查询条件的情况下使用 select count(\*) from table 的时候，Myisam直接可以从计数器中取出数据。而innodb必须全表扫描一次方能得到总的数量

##### 2. 但是当有查询条件的时候，两者的查询效率一致。

##### 4. 主键索引count(\*)的时候之所以慢

InnoDB引擎:

[1]     数据文件和索引文件存储在一个文件中，主键索引默认直接指向数据存储位置。

[2]     二级索引存储指定字段的索引，实际的指向位置是主键索引。当我们通过二级索引统计数据的时候，无需扫描数据文件；而通过主键索引统计数据时，由于主键索引与数据文件存放在一起，所以每次都会扫描数据文件，所以主键索引统计没有二级索引效率高。

[3]     由于主键索引直接指向实际数据，所以当我们通过主键id查询数据时要比通过二级索引查询数据要快。

l  Myismm引擎

[1]     该引擎把每个表都分为几部分存储，比如用户表，包含user.frm，user.MYD和user.MYI。

[2]     User.frm负责存储表结构

[3]     User.MYD负责存储实际的数据记录，所有的用户记录都存储在这个文件中

[4]     User.MYI负责存储用户表的所有索引，这里也包括主键索引。

### 8. 优化order by语句

     基于索引的排序  
     MySQL的弱点之一是它的排序。虽然MySQL可以在1秒中查询大约15,000条记录，但由于MySQL在查询时最多只能使用一个索引。因此，如果WHERE条件已经占用了索引，那么在排序中就不使用索引了，这将大大降低查询的速度。我们可以看看如下的SQL语句:  
     SELECT \* FROM SALES WHERE NAME = “name” ORDER BY SALE\_DATE DESC;  
     在以上的SQL的WHERE子句中已经使用了NAME字段上的索引，因此，在对SALE\_DATE进行排序时将不再使用索引。为了解决这个问题，我们可以对SALES表建立复合索引:  
     ALTER TABLE SALES DROP INDEX NAME, ADD INDEX (NAME,SALE\_DATE)  
     这样再使用上述的SELECT语句进行查询时速度就会大副提升。但要注意，在使用这个方法时，要确保WHERE子句中没有排序字段，在上例中就是不能用SALE\_DATE进行查询，否则虽然排序快了，但是SALE\_DATE字段上没有单独的索引，因此查询又会慢下来。

     在某些情况中， MySQL可以使用一个索引来满足 ORDER BY子句，而不需要额外的排序。 where条件和order by使用相同的索引，并且order by 的顺序和索引顺序相 同，并且order by的字段都是升序或者都是降序。例如：下列sql可以使用索引。

     SELECT \* FROM t1 ORDER BY key\_part1,key\_part2,... ;  
     SELECT \* FROM t1 WHERE key\_part1=1 ORDER BY key\_part1 DESC, key\_part2 DESC;  
     SELECT \* FROM t1 ORDER BY key\_part1 DESC, key\_part2 DESC;  
    但是以下情况不使用索引：  
     SELECT \* FROM t1 ORDER BY key\_part1 DESC, key\_part2 ASC ； --order by 的字段混合 ASC 和 DESC  
     SELECT \* FROM t1 WHERE key2=constant ORDER BY key1 ；-- 用于查询行的关键字与 ORDER BY 中所使用的不相同  
     SELECT \* FROM t1 ORDER BY key1, key2 ；-- 对不同的关键字使用 ORDER BY ：

### 9. 优化GROUP BY

      默认情况下， MySQL 排序所有 GROUP BY col1 ， col2 ， .... 。查询的方法如同在查询中指定 ORDER BY col1 ， col2 ， ... 。如果显式包括一个包含相同的列的 ORDER BY  
子句， MySQL 可以毫不减速地对它进行优化，尽管仍然进行排序。如果查询包括 GROUP BY 但你想要避免排序结果的消耗，你可以指定 ORDER BY NULL禁止排序。  
例如 ：  
INSERT INTO foo  SELECT a, COUNT(\*) FROM bar GROUP BY a ORDER BY NULL;

### 10. 优化 OR

具体详解看：[mysql or条件可以使用索引而避免全表](http://blog.csdn.net/hguisu/article/details/7106159)

### 11. Explain解释说明

explain显示了mysql如何使用索引来处理select语句以及连接表。可以帮助选择更好的索引和写出更优化的查询语句。   
使用方法，在select语句前加上explain就可以了：   
如：

**[sql]** [view plain](http://blog.csdn.net/hguisu/article/details/5731629) [copy](http://blog.csdn.net/hguisu/article/details/5731629)

[print?](http://blog.csdn.net/hguisu/article/details/5731629)

1. explain **select** surname,first\_name form a,b **where** a.id=b.id

分析结果形式如下：   
**table |  type | possible\_keys | key | key\_len  | ref | rows | Extra**

EXPLAIN列的解释

1 table:

显示这一行的数据是关于哪张表的

2 type:

这是重要的列，显示连接使用了何种类型。从最好到最差的连接类型为：system、const、eg\_reg、ref、ref\_or\_null、 range、indexhe、 ALL。  
**system:**表仅有一行(=系统表)。这是const联接类型的一个特例  
      **const:(PRIMARY KEY或UNIQUE)**  
            表最多有一个匹配行，它将在查询开始时被读取。因为仅有一行，在这行的列值可被优化器剩余部分认为是常数。  
            const表很快，因为它们只读取一次！  
            const用于用常数值比较PRIMARY KEY或UNIQUE索引的所有部分时。  
            在下面的查询中，tbl\_name可以用于const表：

**[sql]** [view plain](http://blog.csdn.net/hguisu/article/details/5731629) [copy](http://blog.csdn.net/hguisu/article/details/5731629)

[print?](http://blog.csdn.net/hguisu/article/details/5731629)

1. **SELECT** \* **from** tbl\_name **WHERE** primary\_key=1；

**eq\_reg:key**  
  对于每个来自于前面的表的行组合，从该表中读取一行。这可能是最好的联接类型，除了const类型。  
           它用在一个索引的所有部分被联接使用并且索引是UNIQUE或PRIMARY KEY。  
           eq\_ref可以用于使用= 操作符比较的带索引的列。比较值可以为常量或一个使用在该表前面所读取的表的列的表达式。  
  在下面的例子中，MySQL可以使用eq\_ref联接来处理ref\_tables：

**[sql]** [view plain](http://blog.csdn.net/hguisu/article/details/5731629) [copy](http://blog.csdn.net/hguisu/article/details/5731629)

[print?](http://blog.csdn.net/hguisu/article/details/5731629)

1. **SELECT** \* **FROM** ref\_table,other\_table **WHERE** ref\_table.key\_column=other\_table.**column**;
2. **SELECT** \* **FROM** ref\_table,other\_table **WHERE** ref\_table.key\_column\_part1=other\_table.**column**
3. AND ref\_table.key\_column\_part2=1;

        **ref:key**  
 对于每个来自于前面的表的行组合，所有有匹配索引值的行将从这张表中读取。如果联接只使用键的最左边的前缀，  
          或如果键不是UNIQUE或PRIMARY KEY（换句话说，如果联接不能基于关键字选择单个行的话），则使用ref。  
          如果使用的键仅仅匹配少量行，该联接类型是不错的。  
          ref可以用于使用=或<=>操作符的带索引的列。  
 在下面的例子中，MySQL可以使用ref联接来处理ref\_tables：

**[sql]** [view plain](http://blog.csdn.net/hguisu/article/details/5731629) [copy](http://blog.csdn.net/hguisu/article/details/5731629)

[print?](http://blog.csdn.net/hguisu/article/details/5731629)

1. **SELECT** \* **FROM** ref\_table **WHERE** key\_column=expr;
2. **SELECT** \* **FROM** ref\_table,other\_table **WHERE** ref\_table.key\_column=other\_table.**column**;
3. **SELECT** \* **FROM** ref\_table,other\_table **WHERE** ref\_table.key\_column\_part1=other\_table.**column**
4. AND ref\_table.key\_column\_part2=1;

**ref\_or\_null:Or Is null**  
该联接类型如同ref，但是添加了MySQL可以专门搜索包含NULL值的行。在解决子查询中经常使用该联接类型的优化。  
        在下面的例子中，MySQL可以使用ref\_or\_null联接来处理ref\_tables：

**[sql]** [view plain](http://blog.csdn.net/hguisu/article/details/5731629) [copy](http://blog.csdn.net/hguisu/article/details/5731629)

[print?](http://blog.csdn.net/hguisu/article/details/5731629)

1. **SELECT** \* **FROM** ref\_table **WHERE** key\_column=expr OR key\_column **IS** NULL;

**range:=、<>、>、>=、<、<=、IS NULL、<=>、BETWEEN或者IN**  
 只检索给定范围的行，使用一个索引来选择行。key列显示使用了哪个索引。  
          key\_len包含所使用索引的最长关键元素。在该类型中ref列为NULL。  
当使用=、<>、>、>=、<、<=、IS NULL、<=>、BETWEEN或者IN操作符，用常量比较关键字列时，可以使用range：

**[sql]** [view plain](http://blog.csdn.net/hguisu/article/details/5731629) [copy](http://blog.csdn.net/hguisu/article/details/5731629)

[print?](http://blog.csdn.net/hguisu/article/details/5731629)

1. **SELECT** \* **FROM** tbl\_name **WHERE** key\_column = 10;
2. **SELECT** \* **FROM** tbl\_name **WHERE** key\_column BETWEEN 10 and 20;
3. **SELECT** \* **FROM** tbl\_name **WHERE** key\_column IN (10,20,30);
4. **SELECT** \* **FROM** tbl\_name **WHERE** key\_part1= 10 AND key\_part2 IN (10,20,30);

**indexhe:**  
该联接类型与ALL相同，除了只有索引树被扫描。这通常比ALL快，因为索引文件通常比数据文件小。  
当查询只使用作为单索引一部分的列时，MySQL可以使用该联接类型。  
     **ALL：**  
对于每个来自于先前的表的行组合，进行完整的表扫描。如果表是第一个没标记const的表，  
        这通常不好，并且通常在它情况下很差。通常可以增加更多的索引而不要使用ALL，  
        使得行能基于前面的表中的常数值或列值被检索出。

3 possible\_keys :

   显示可能应用在这张表中的索引。如果为空，没有可能的索引。可以为相关的域从WHERE语句中    
   选择一个合适的语句

4 key ：

 实际使用的索引。如果为NULL，则没有使用索引。很少的情况下，MYSQL会选择优化不足的索引  。  
 这种情况下，可以在SELECT语句中使用USEINDEX（indexname）来强制使用一个索引或者用IGNORE INDEX（indexname）来强制MYSQL忽略索引

5key\_len:

使用的索引的长度。在不损失精确性的情况下，长度越短越好

6 ref

显示索引的哪一列被使用了，如果可能的话，是一个常数

7 rows

MYSQL认为必须检查的用来返回请求数据的行数 (扫描行的数量)

8 Extra

 该列包含MySQL解决查询的详细信息  
 关于MYSQL如何解析查询的额外信息。将在表4.3中讨论，但这里可以看到的坏的例子是Using temporary和Using filesort，  
 意思MYSQL根本不能使用索引，结果是检索会很慢   
  
extra列返回的描述的意义   
  
**Distinct:**  
一旦MYSQL找到了与行相联合匹配的行，就不再搜索了   
**Not exists :**  
MYSQL优化了LEFT JOIN，一旦它找到了匹配LEFT JOIN标准的行， 就不再搜索了   
        面是一个可以这样优化的查询类型的例子：  
SELECT \* FROM t1 LEFT JOIN t2 ON t1.id=t2.id WHERE t2.id IS NULL；  
假定t2.id定义为NOT NULL。在这种情况下，MySQL使用t1.id的值扫描t1并查找t2中的行。  
        如果MySQL在t2中发现一个匹配的行，它知道t2.id绝不会为NULL，并且不再扫描t2内有相同的id值的行。  
        换句话说，对于t1的每个行，MySQL只需要在t2中查找一次，无论t2内实际有多少匹配的行。  
**Range checked for each Record（index map:#）**  
没有找到理想的索引，因此对于从前面表中来的每一个行组合，MYSQL检查使用哪个索引，并用它来从表中返回行。  
        这是使用索引的最慢的连接之一   
        MySQL没有发现好的可以使用的索引，但发现如果来自前面的表的列值已知，可能部分索引可以使用。  
        对前面的表的每个行组合，MySQL检查是否可以使用range或index\_merge访问方法来索取行。  
        关于适用性标准的描述参见7.2.5节，“范围优化”和7.2.6节，“索引合并优化”，  
        不同的是前面表的所有列值已知并且认为是常量。这并不很快，但比执行没有索引的联接要快得多。  
**Using filesort**  
看到这个的时候，查询就需要优化了。MYSQL需要进行额外的步骤来发现如何对返回的行排序。  
        它根据连接类型以及存储排序键值和匹配条件的全部行的行指针来排序全部行   
**Using index**  
列数据是从仅仅使用了索引中的信息而没有读取实际的行动的表返回的，  
        这发生在对表的全部的请求列都是同一个索引的部分的时候   
**Using temporary**  
看到这个的时候，查询需要优化了。这里，MYSQL需要创建一个临时表来存储结果，这通常发生在对不同的列集进行ORDER BY上，而不是GROUP BY上   
**Using where**  
使用了WHERE从句来限制哪些行将与下一张表匹配或者是返回给用户。如果不想返回表中的全部行，  
        并且连接类型ALL或index，这就会发生，或者是查询有问题   
Impossible WHERE noticed after reading const table...

### 12. SQL核心语句(非常实用的几个技巧)

#### 1) 插入数据

**批量插入:**

**[sql]** [view plain](http://blog.csdn.net/hguisu/article/details/5731629) [copy](http://blog.csdn.net/hguisu/article/details/5731629)

[print?](http://blog.csdn.net/hguisu/article/details/5731629)

1. **INSERT** mytable (first\_column,second\_column,third\_column)
2. **VALUES** ('some data','some more data','yet more data') ,
3. **VALUES** ('some data','some more data','yet more data') ,
4. **VALUES** ('some data','some more data','yet more data')

#### 2）清空数据表

**TRUNCATE** **TABLE**  `mytable`

注意：删除表中的所有记录，应使用TRUNCATE TABLE语句。注意这里为什么要用TRUNCATE TABLE语句代替DELETE语句:当你使用TRUNCATE TABLE语句时，记录的删除是不作记录的。也就是说，这意味着TRUNCATE TABLE要比DELETE快得多。

#### 3）用SELECT创建记录和表

　　INSERT语句与DELETE语句和UPDATE语句有一点不同，它一次只操作一个记录。然而，有一个方法可以使INSERT 语句一次添加多个记录。要作到这一点，你需要把INSERT语句与SELECT语句结合起来，象这样:

**[sql]** [view plain](http://blog.csdn.net/hguisu/article/details/5731629) [copy](http://blog.csdn.net/hguisu/article/details/5731629)

[print?](http://blog.csdn.net/hguisu/article/details/5731629)

1. **INSERT** mytable(first\_column,second\_column)
2. **SELECT** another\_first,another\_second  **FROM** anothertable **WHERE** another\_first='Copy Me!';

　　这个语句从anothertable拷贝记录到mytable.只有表anothertable中字段another\_first的值为'Copy Me!'的记录才被拷贝。

　　当为一个表中的记录建立备份时，这种形式的INSERT语句是非常有用的。在删除一个表中的记录之前，你可以先用这种方法把它们拷贝到另一个表中。

　　如果你需要拷贝整个表，你可以使用SELECT INTO语句。例如，下面的语句创建了一个名为newtable的新表，该表包含表mytable的所有数据:

**[sql]** [view plain](http://blog.csdn.net/hguisu/article/details/5731629) [copy](http://blog.csdn.net/hguisu/article/details/5731629)

[print?](http://blog.csdn.net/hguisu/article/details/5731629)

1. **SELECT** \* **INTO** newtable **FROM** mytable;

　　你也可以指定只有特定的字段被用来创建这个新表。要做到这一点，只需在字段列表中指定你想要拷贝的字段。另外，你可以使用WHERE子句来限制拷贝到新表中的记录。下面的例子只拷贝字段second\_columnd的值等于'Copy Me!'的记录的first\_column字段。

**[sql]** [view plain](http://blog.csdn.net/hguisu/article/details/5731629) [copy](http://blog.csdn.net/hguisu/article/details/5731629)

[print?](http://blog.csdn.net/hguisu/article/details/5731629)

1. **SELECT** first\_column **INTO** newtable
2. **FROM** mytable
3. **WHERE** second\_column='Copy Me!';

　　使用SQL修改已经建立的表是很困难的。例如，如果你向一个表中添加了一个字段，没有容易的办法来去除它。另外，如果你不小心把一个字段的数据类型给错了，你将没有办法改变它。但是，使用本节中讲述的SQL语句，你可以绕过这两个问题。

　　例如，假设你想从一个表中删除一个字段。使用SELECT INTO语句，你可以创建该表的一个拷贝，但不包含要删除的字段。这使你既删除了该字段，又保留了不想删除的数据。

　　如果你想改变一个字段的数据类型，你可以创建一个包含正确数据类型字段的新表。创建好该表后，你就可以结合使用UPDATE语句和SELECT语句，把原来表中的所有数据拷贝到新表中。通过这种方法，你既可以修改表的结构，又能保存原有的数据。

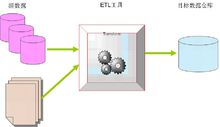
# ETL(Extrat-Transform-Load) 数据仓库技术

**ETL**，是英文 Extract-Transform-Load 的缩写，用来描述将数据从来源端经过抽取（extract）、转换（transform）、加载（load）至目的端的过程

ETL是构建数据仓库的重要一环，用户从[数据源](http://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E6%BA%90)抽取出所需的数据，经过[数据清洗](http://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E6%B8%85%E6%B4%97),最终按照预先定义好的数据仓库模型，将数据加载到数据仓库中去。

## 体系结构

### ETL架构

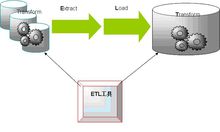
[](http://baike.baidu.com/pic/ETL/1251949/0/3c6d55fbb2fb43166357503e2aa4462309f7d32d?fr=lemma&ct=single)

在ETL架构中，数据的流向是从源数据流到ETL工具，ETL工具是一个单独的数据处理引擎，一般会在单独的硬件服务器上，实现所有数据转化的工作，然后将数据加载到目标数据仓库中，如果要增加整个ETL过程的效率，则只能增强ETL工具服务器的配置，优化系统处理流程（一般可调的东西非常少）。IBM的datastage和Informatica的powercenter原来都是采用的这种架构。

ETL架构的优势：

1. ETL可以分担数据库系统的负载（采用单独的硬件服务器）
2. ETL相对于EL-T架构可以实现更为复杂的数据转化逻辑
3. ETL采用单独的硬件服务器。.
4. ETL与底层的数据库数据存储无关。

### ELT架构

[](http://baike.baidu.com/pic/ETL/1251949/0/b219ebc4b74543a9567aef8814178a82b9011414?fr=lemma&ct=single)

在[ELT](http://baike.baidu.com/item/ELT/9352740)架构中，ELT只负责提供图形化的界面来设计业务规则，数据的整个加工过程都在目标和源的数据库之间流动，ELT协调相关的数据库系统来执行相关的应用，数据加工过程既可以在源数据库端执行，也可以在目标数据仓库端执行（主要取决于系统的架构设计和数据属性）。当ETL过程需要提高效率，则可以通过对相关数据库进行调优，或者改变执行加工的服务器就可以达到。一般数据库厂商会力推该中架构，像[Oracle](http://baike.baidu.com/item/Oracle)和[Teradata](http://baike.baidu.com/item/Teradata)都极力宣传ELT架构。

ELT架构的优势：

1. ELT主要通过数据库引擎来实现系统的可扩展性（尤其是当数据加工过程在晚上时，可以充分利用数据库引擎的资源）
2. ELT可以保持所有的数据始终在数据库当中，避免数据的加载和导出，从而保证效率，提高系统的可监控性。
3. ELT可以根据数据的分布情况进行并行处理优化，并可以利用数据库的固有功能优化磁盘I/O。
4. ELT的可扩展性取决于数据库引擎和其硬件服务器的可扩展性。
5. 通过对相关数据库进行性能调优，ETL过程获得3到4倍的效率提升一般不是特别困难。

# Oracle数据库

## 哈希函数DBMS\_UTILITY.GET\_HASH\_VALUE

SELECT

DBMS\_\_UTILITY.GET\_HASH\_VALUE(:OrigianalPasswd,100000000,power(2,30))

FROM

DUAL

用户的密码是如何存储的

1. 随机生成密码
2. 用随机生成的密码作为参数调用DBMS\_\_UTILITY.GET\_HASH\_VALUE
3. 生成的hash值存储在数据中
4. 用户每次登陆会再次调用DBMS\_\_UTILITY.GET\_HASH\_VALUE生成hash值，比较和数据库中的hash值是否一致

## oracle随机函数DBMS\_RANDOM.STRING(‘P’,7)

DB MS\_RANDOM.STRING(‘P’,7)

## 查出所有被锁住的表

select

b.owner TABLEOWNER,

b.object\_name TABLENAME,

c.OSUSER LOCKBY,

c.USERNAME LOGINID,

c.sid SID,

c.SERIAL SERIAL

from

v$locked\_object a,

dba\_objects b,

v$session c

where

b.object\_id = a.object\_id

AND

a.SESSION\_ID =c.sid;

--通过 SID, SERIAL 解锁

--alter system kill session 'SID, SERIAL';

## [ORACLE 中ROWNUM用法总结!](http://www.cnblogs.com/szlbm/p/5806070.html)

ORACLE 中ROWNUM用法总结!   
对于 Oracle 的 rownum 问题，很多资料都说不支持>,>=,=,between...and，只能用以上符号(<、<=、!=)，并非说用>,& gt;=,=,between..and 时会提示SQL语法错误，而是经常是查不出一条记录来，还会出现似乎是莫名其妙的结果来，其实您只要理解好了这个 rownum 伪列的意义就不应该感到惊奇，同样是伪列，rownum 与 rowid 可有些不一样，下面以例子说明

假设某个表 t1(c1) 有 20 条记录

如果用 select rownum,c1 from t1 where rownum < 10, 只要是用小于号，查出来的结果很容易地与一般理解在概念上能达成一致，应该不会有任何疑问的。

可如果用 select rownum,c1 from t1 where rownum > 10 (如果写下这样的查询语句，这时候在您的头脑中应该是想得到表中后面10条记录)，你就会发现，显示出来的结果要让您失望了，也许您还会怀疑是不谁删了一 些记录，然后查看记录数，仍然是 20 条啊？那问题是出在哪呢？

先好好理解 rownum 的意义吧。因为ROWNUM是对结果集加的一个伪列，即先查到结果集之后再加上去的一个列 (强调：先要有结果集)。简单的说 rownum 是对符合条件结果的序列号。它总是从1开始排起的。所以你选出的结果不可能没有1，而有其他大于1的值。所以您没办法期望得到下面的结果集：

11 aaaaaaaa  
12 bbbbbbb  
13 ccccccc  
.................

rownum >10 没有记录，因为第一条不满足去掉的话，第二条的ROWNUM又成了1，所以永远没有满足条件的记录。或者可以这样理解：

ROWNUM是一个序列，是oracle数据库从数据文件或缓冲区中读取数据的顺序。它取得第一条记录则rownum值为1，第二条为2，依次类 推。如果你用>,>=,=,between...and这些条件，因为从缓冲区或数据文件中得到的第一条记录的rownum为1，则被删除， 接着取下条，可是它的rownum还是1，又被删除，依次类推，便没有了数据。

有了以上从不同方面建立起来的对 rownum 的概念，那我们可以来认识使用 rownum 的几种现像

1. select rownum,c1 from t1 where rownum != 10 为何是返回前9条数据呢？它与 select rownum,c1 from tablename where rownum < 10 返回的结果集是一样的呢？  
因为是在查询到结果集后，显示完第 9 条记录后，之后的记录也都是 != 10,或者 >=10,所以只显示前面9条记录。也可以这样理解，rownum 为9后的记录的 rownum为10，因条件为 !=10，所以去掉，其后记录补上，rownum又是10，也去掉，如果下去也就只会显示前面9条记录了

2. 为什么 rownum >1 时查不到一条记录，而 rownum >0 或 rownum >=1 却总显示所以的记录  
因为 rownum 是在查询到的结果集后加上去的，它总是从1开始

3. 为什么 between 1 and 10 或者 between 0 and 10 能查到结果，而用 between 2 and 10 却得不到结果  
原因同上一样，因为 rownum 总是从 1 开始

从上可以看出，任何时候想把 rownum = 1 这条记录抛弃是不对的，它在结果集中是不可或缺的，少了rownum=1 就像空中楼阁一般不能存在，所以你的 rownum 条件要包含到 1

但如果就是想要用 rownum > 10 这种条件的话话就要用嵌套语句,把 rownum 先生成，然后对他进行查询。  
select \*   
from (selet rownum as rn，t1.\* from a where ...)  
where rn >10

一般代码中对结果集进行分页就是这么干的。

另外：rowid 与 rownum 虽都被称为伪列，但它们的存在方式是不一样的，rowid 可以说是物理存在的，表示记录在表空间中的唯一位置ID，在DB中唯一。只要记录没被搬动过，rowid是不变的。rowid 相对于表来说又像表中的一般列，所以以 rowid 为条件就不会有 rownum那些情况发生。  
另外还要注意：rownum不能以任何基表的名称作为前缀。

## 环境变量NLS\_LANG

NLS\_LANG是一个环境变量，用于定义语言，地域以及字符集属性。对于非英语的字符集，NLS\_LANG的设置就非常重要。

NLS：‘National Language Support (NLS)’ 当我们设定一种nls的时候实际上我们是为oracle在存放数据时指定了他的语种所特有的一些表达形式，比如我们选择chinese,那么它的中文字符如何存放,按什么规则排序，货币如何表示，日期格式也就被设定了。

NLS\_LANG参数由以下部分组成:NLS\_LANG=<Language>\_<Territory>.<Clients Characterset> NLS\_Language 指定: - Oracle（错误）信息的语言 - 日和月份的名称 注意：NLS\_LANGUAGE与插入和查询的数据的语言无关。

NLS\_Territory 指定: - 货币和数字格式 - 计算星期和天数的范围和惯例

客户端字符集（CLIENTS CHARACTERSET）: - 定义Oracle客户端，客户应用使用的编码 或者它要符合您Microsoft Windows代码页 （GUI工具的ACP, 命令提示符的CHCP 值） 或者为Unicode WIN32应用设置为UTF8/AL32UTF8。

常见的值可以参见[Oracle Database Client Globalization Support](http://docs.oracle.com/html/B10131_02/gblsupp.htm)

### Linux/Unix下的设置

export NLS\_LANG=AMERICAN\_AMERICA.ZHS16GBK

export NLS\_LANG=SIMPLIFIED CHINESE\_CHINA.ZHS16GBK

DB中的NLS\_LANG DB也有自己的NLS\_LANG参数，来标识数据库的语言，地域，字符集。可以通过以下的SQL来查得， SELECT \* FROM v$nls\_parameters; OR SELECT USERENV ('language') FROM DUAL;

客户端的NLS\_LANG和DB的NLS\_LANG DB的NLS\_LANG和客户端环境变量中NLS\_LANG尽可能的保持一致。Setting the NLS\_LANG Environment Variable for Oracle Databases 提供了如何保持一致的方法。 那么为什么要保持一致呢？那是因为如果字符集不匹配，有可能在DB和Client间相互转换的过程中，出现字符转换错误的情况。比如Oracle nls\_lang tips中举的例子。

### Windows中设置NLS\_LANG

1.命令提示符中设置为环境变量

如果您在命令行中设置NLS\_LANG为环境变量，它将覆盖注册表和系统属性中的NLS\_LANG的定义。在命令提示符中，使用“set”命令，例如：

LANGUAGE设置为中文时，提示如下：

C:\Users\TianPan>set nls\_lang=SIMPLIFIED CHINESE\_CHINA.ZHS16GBK

C:\Users\TianPan>sqlplus /nolog

SQL\*Plus: Release 11.2.0.1.0 Production on 星期五 11月 28 10:07:21 2014

Copyright (c) 1982, 2010, Oracle. All rights reserved.

SQL> conn scott/tiger 已连接。

SQL> select sysdate from dual;

SYSDATE -------------- 28-11月-14

当Language设置为American时，提示信息则变为英文:

C:\Users\TianPan>set nls\_lang=AMERICAN\_AMERICA.ZHS16GBK

C:\Users\TianPan>sqlplus /nolog

SQL\*Plus: Release 11.2.0.1.0 Production on Fri Nov 28 10:07:51 2014

Copyright (c) 1982, 2010, Oracle. All rights reserved.

SQL> conn scott/tiger Connected. SQL> select sysdate from dual;

SYSDATE ------------ 28-NOV-14

在Windows下通过set nls\_lang来设置，只是Session级别的，关闭cmd窗口后再打开，就又变为原来的设置了。如需永久生效，可以修改注册表的参数。

2.注册表设置（永久生效）

默认情况下，windows上的Oracle安装使用注册表来定义这个设置。 版本 10g 及以上: HKEY\_LOCAL\_MACHINE\SOFTWARE\ORACLE\KEY\_<oracle\_home\_name> 这里您会看到一个条目名为NLS\_LANG

在64位windows平台上安装32位软件，会使用32位兼容性路径 HKEY\_LOCAL\_MACHINE\SOFTWARE\Wow6432Node\ORACLE\KEY\_<oracle\_home\_name>

3.操作系统环境变量

尽管注册表是Microsoft Windows设置的主要存储位置，它不是唯一可以设置参数的地方。

虽然不推荐，但您可以设置NLS\_LANG为系统属性中的系统或者用户变量。这个设置会被所有Oracle home使用。

设置位置： '我的电脑' -> '属性' > '高级' -> '环境变量'

因为这些环境变量比注册表中已经设置的参数级别高，所以除非您有非常好的理由否则不要在这个位置设置Oracle参数。

## sqlplus中清屏命令

clear screen

## sqlplus中help命令

help 命令

## sqlplus中的show和set命令

SQL> show all --查看所有68个系统变量值   
SQL> show user --显示当前连接用户   
SQL> show error　　 --显示错误   
SQL> set heading off --禁止输出列标题，默认值为ON   
SQL> set feedback off --禁止显示最后一行的计数反馈信息，默认值为"对6个或更多的记录，回送ON"   
SQL> set timing on --默认为OFF，设置查询耗时，可用来估计SQL语句的执行时间，测试性能   
SQL> set sqlprompt "SQL> " --设置默认提示符，默认值就是"SQL> "   
SQL> set linesize 1000 --设置屏幕显示行宽，默认100   
SQL> set autocommit ON --设置是否自动提交，默认为OFF   
SQL> set pause on --默认为OFF，设置暂停，会使屏幕显示停止，等待按下ENTER键，再显示下一页   
SQL> set arraysize 1 --默认为15   
SQL> set long 1000 --默认为80

## oracle查询前10条记录

用rownum关键字

select \* from A where rownum<=10

## oracle中左连接，右连接，全连接以及（+）号用法

Oracle  外连接(OUTER JOIN)包括以下：

* 左外连接(左边的表不加限制)
* 右外连接(右边的表不加限制)
* 全外连接(左右两表都不加限制)

对应SQL：LEFT/RIGHT/FULL OUTER JOIN。 通常省略OUTER关键字， 写成：LEFT/RIGHT/FULL JOIN。

在左连接和右连接时都会以一张A表为基础表，该表的内容会全部显示，然后加上A表和B表匹配的内容。 如果A表的数据在B表中没有记录。 那么在相关联的结果集行中列显示为空值（NULL）。

对于外连接， 也可以使用“(+) ”来表示。 关于使用（+）的一些注意事项：

1. （+）操作符只能出现在WHERE子句中，并且不能与OUTER JOIN语法同时使用。
2. 当使用（+）操作符执行外连接时，如果在WHERE子句中包含有多个条件，则必须在所有条件中都包含（+）操作符。
3. （+）操作符只适用于列，而不能用在表达式上。
4. （+）操作符不能与OR和IN操作符一起使用。
5. （+）操作符只能用于实现左外连接和右外连接，而不能用于实现完全外连接。

用（+）来实现， 这个+号可以这样来理解： + 表示补充，即哪个表有加号，这个表就是匹配表。如果加号写在右表，左表就是全部显示，所以是左连接

用（+）来实现， 这个+号可以这样来理解： + 表示补充，即哪个表有加号，这个表就是匹配表。如果加号写在左表，右表就是全部显示，所以是右连接。

    左表和右表都不做限制，所有的记录都显示，两表不足的地方均为NULL。 全外连接不支持（+）写法。

**创建两张表，插入数据用于学习测试：**

[复制代码](javascript:void(0);)

CREATE TABLE t\_A (

id number,

name VARCHAR2(**10**)

);

CREATE TABLE t\_B (

id number,

name VARCHAR2(**10**)

);

INSERT INTO t\_A VALUES(**1**,'A');

INSERT INTO t\_A VALUES(**2**,'B');

INSERT INTO t\_A VALUES(**3**,'C');

INSERT INTO t\_A VALUES(**4**,'D');

INSERT INTO t\_A VALUES(**5**,'E');

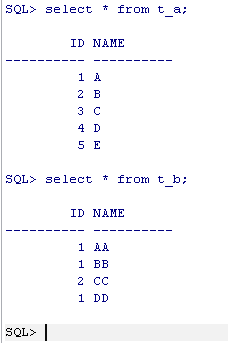
INSERT INTO t\_B VALUES(**1**,'AA');

INSERT INTO t\_B VALUES(**1**,'BB');

INSERT INTO t\_B VALUES(**2**,'CC');

INSERT INTO t\_B VALUES(**1**,'DD');

[复制代码](javascript:void(0);)



[回到目录](https://www.cnblogs.com/hehaiyang/p/4745897.html#_labelTop)

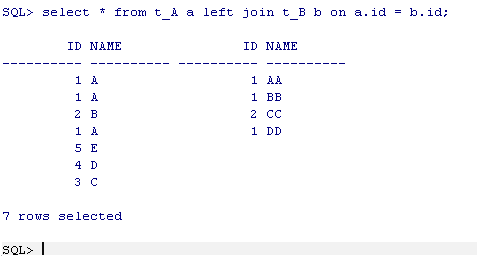
### 左外连接（LEFT OUTER JOIN/ LEFT JOIN）

　　LEFT JOIN是以左表的记录为基础的,示例中t\_A可以看成左表,t\_B可以看成右表,它的结果集是t\_A表中的全部数据，再加上t\_A表和t\_B表匹配后的数据。换句话说,左表(t\_A)的记录将会全部表示出来,而右表(t\_B)只会显示符合搜索条件的记录。t\_B表记录不足的地方均为NULL。

select \* from t\_A a left join t\_B b on a.id = b.id;

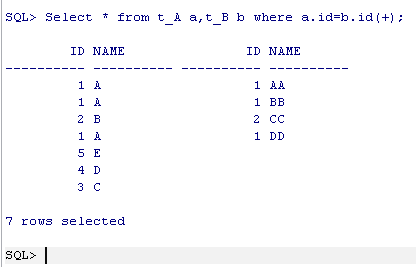
或

select \* from t\_A a left outer join t\_B b on a.id = b.id;



　　用（+）来实现， 这个+号可以这样来理解： + 表示补充，即哪个表有加号，这个表就是匹配表。如果加号写在右表，左表就是全部显示，所以是左连接。

Select \* from t\_A a,t\_B b where a.id=b.id(+);



[回到目录](https://www.cnblogs.com/hehaiyang/p/4745897.html#_labelTop)

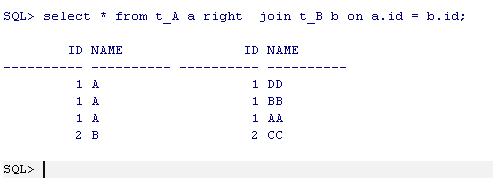
### 右外连接（RIGHT OUTER JOIN/RIGHT JOIN）

　　和LEFT JOIN的结果刚好相反,是以右表(t\_B)为基础的。它的结果集是t\_B表所有记录，再加上t\_A和t\_B匹配后的数据。 t\_A表记录不足的地方均为NULL。

select \* from t\_A a right join t\_B b on a.id = b.id;

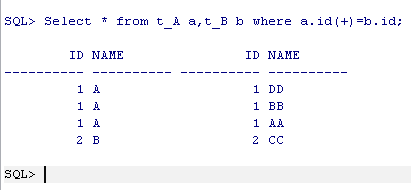
或

select \* from t\_A a right outer join t\_B b on a.id = b.id;



　　用（+）来实现， 这个+号可以这样来理解： + 表示补充，即哪个表有加号，这个表就是匹配表。如果加号写在左表，右表就是全部显示，所以是右连接。

Select \* from t\_A a,t\_B b where a.id(+)=b.id;



[回到目录](https://www.cnblogs.com/hehaiyang/p/4745897.html#_labelTop)

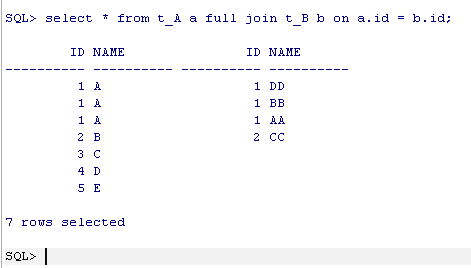
### 全外连接（FULL OUTER JOIN/FULL JOIN）

     左表和右表都不做限制，所有的记录都显示，两表不足的地方均为NULL。 全外连接不支持（+）写法。

select \* from t\_A a full join t\_B b on a.id = b.id;

或

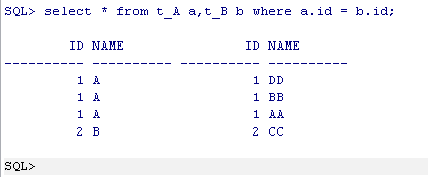
select \* from t\_A a full outer join t\_B b on a.id = b.id;



**补充**

select \* from t\_A a,t\_B b where a.id = b.id;

select \* from t\_A a join t\_B b on a.id = b.id;



select \* from t\_A a where a.id in (select b.id from t\_B b);

select \* from t\_A a where exists (select **1** from t\_B b where a.id = b.id);

## oracle启动线程功能

exec sql enable threads

## oracle导入imp导出exp

基本语法和实例：   
一、EXP:   
  有三种主要的方式（完全、用户、表）   
  1、完全：   
   EXP SYSTEM/MANAGER BUFFER=64000 FILE=C:\FULL.DMP FULL=Y   
   如果要执行完全导出，必须具有特殊的权限   
  2、用户模式：   
   EXP SONIC/SONIC BUFFER=64000 FILE=C:\SONIC.DMP OWNER=SONIC   
   这样用户SONIC的所有对象被输出到文件中。   
  3、表模式：  
   EXP SONIC/SONIC  BUFFER=64000 FILE=C:\SONIC.DMP OWNER=SONIC TABLES=(SONIC)   
   这样用户SONIC的表SONIC就被导出   
 二、IMP:   
   具有三种模式（完全、用户、表）   
   1、完全：   
    IMP SYSTEM/MANAGER BUFFER=64000 FILE=C:\FULL.DMP FULL=Y   
   2、用户模式：   
    IMP SONIC/SONIC    BUFFER=64000 FILE=C:\SONIC.DMP FROM USER=SONIC TO USER=SONIC   
    这样用户SONIC的所有对象被导入到文件中。必须指定FROM USER、TO USER参数，这样才能导入数据。   
   3、表模式：   
     EXP SONIC/SONIC    BUFFER=64000 FILE=C:\SONIC.DMP OWNER=SONIC TABLES=(SONIC)   
    这样用户SONIC的表SONIC就被导入。  
ORACLE数据库有两类备份方法。第一类为物理备份，该方法实现数据库的完整恢复，但数据库必须运行在归挡模式下（业务数据库在非归挡模式下运行），且需要极大的外部存储设备，例如磁带库；第二类备份方式为逻辑备份，业务数据库采用此种方式，此方法不需要数据库运行在归挡模式下，不但备份简单，而且可以不需要外部存储设备。  
　　  
　　数据库逻辑备份方法  
　　  
　　ORACLE数据库的逻辑备份分为三种模式：表备份、用户备份和完全备份。  
　　  
　　表模式  
　　  
　　备份某个用户模式下指定的对象（表）。业务数据库通常采用这种备份方式。  
　　  
　　若备份到本地文件，使用如下命令：  
　　  
　　exp icdmain/icd rows=y indexes=n compress=n buffer=65536  
　　feedback=100000 volsize=0  
　　file=exp\_icdmain\_csd\_yyyymmdd.dmp  
　　log=exp\_icdmain\_csd\_yyyymmdd.log  
　　tables=icdmain.commoninformation,icdmain.serviceinfo,icdmain.dealinfo  
　　  
　　若直接备份到磁带设备，使用如下命令：  
　　exp icdmain/icd rows=y indexes=n compress=n buffer=65536  
　　feedback=100000 volsize=0  
　　file=/dev/rmt0  
　　log=exp\_icdmain\_csd\_yyyymmdd.log  
　　tables=icdmain.commoninformation,icdmain.serviceinfo,icdmain.dealinfo  
　　  
　　注：在磁盘空间允许的情况下，应先备份到本地服务器，然后再拷贝到磁带。出于速度方面的考虑，尽量不要直接备份到磁带设备。  
　　  
　　用户模式  
　　  
　　备份某个用户模式下的所有对象。业务数据库通常采用这种备份方式。  
　　若备份到本地文件，使用如下命令：  
　　exp icdmain/icd owner=icdmain rows=y indexes=n compress=n buffer=65536  
　　feedback=100000 volsize=0  
　　file=exp\_icdmain\_yyyymmdd.dmp  
　　log=exp\_icdmain\_yyyymmdd.log  
　　若直接备份到磁带设备，使用如下命令：  
　　exp icdmain/icd owner=icdmain rows=y indexes=n compress=n buffer=65536  
　　feedback=100000 volsize=0  
　　file=/dev/rmt0  
　　log=exp\_icdmain\_yyyymmdd.log  
　　注：如果磁盘有空间，建议备份到磁盘，然后再拷贝到磁带。如果数据库数据量较小，可采用这种办法备份。

**以下为详细的导入导出实例：**

　　一、数据导出：

　　1、 将数据库TEST完全导出，用户名system 密码manager 导出到D：\daochu.dmp中

　　exp system/manager@TEST file=d：\daochu.dmp full=y

　　2、 将数据库中system用户与sys用户的表导出

　　exp system/manager@TEST file=d：\daochu.dmp owner=（system，sys）

　　3、 将数据库中的表table1 、table2导出

　　exp system/manager@TEST file=d：\daochu.dmp tables=（table1，table2）

　　4、 将数据库中的表table1中的字段filed1以"00"打头的数据导出

　　exp system/manager@TEST file=d：\daochu.dmp tables=（table1） query=\" where filed1 like '00%'\"

　　上面是常用的导出，对于压缩我不太在意，用winzip把dmp文件可以很好的压缩。

　　不过在上面命令后面 加上 compress=y  就可以了

　　二、数据的导入

 　 1、将D：\daochu.dmp 中的数据导入 TEST数据库中。

　　imp system/manager@TEST  file=d：\daochu.dmp

　　上面可能有点问题，因为有的表已经存在，然后它就报错，对该表就不进行导入。

　　在后面加上 ignore=y 就可以了。

　　2 将d：\daochu.dmp中的表table1 导入

　　imp system/manager@TEST  file=d：\daochu.dmp  tables=（table1）

　　基本上上面的导入导出够用了。不少情况我是将表彻底删除，然后导入。

## oracle导入imp导出exp

**一. 导出工具 exp**

1. 它是操作系统下一个可执行的文件 存放目录/ORACLE\_HOME/bin

exp导出工具将数据库中数据备份压缩成一个二进制系统文件.可以在不同OS间迁移  
    
它有三种模式：

a.  用户模式： 导出用户所有对象以及对象中的数据；  
b.  表模式： 导出用户所有表或者指定的表；  
c.  整个数据库： 导出数据库中所有对象。

2. 导出工具exp交互式命令行方式的使用的例子：

$exp test/test123@appdb  
Enter array fetch buffer size: 4096 > 回车  
Export file: expdat.dmp > m.dmp  生成导出的文件名  
(1)E(ntire database), (2)U(sers), or (3)T(ables): (2)U > 3  
Export table data (yes/no): yes > 回车  
Compress extents (yes/no): yes > 回车  
Export done in ZHS16GBK character set and ZHS16GBK NCHAR character set  
About to export specified tables via Conventional Path ...  
Table(T) or Partition(T:P) to be exported: (RETURN to quit) > cmamenu  要导出的表名  
. . exporting table                        CMAMENU       4336 rows exported  
Table(T) or Partition(T:P) to be exported: (RETURN to quit) >要导出的表名n  
Table(T) or Partition(T:P) to be exported: (RETURN to quit) > 回车  
Export terminated successfully without warnings.

导出命令：

复制代码 代码如下:

exp test/test123@orcl file=/opt/daochu.dmp full=y

其中test是用户名称，后一个test123是用户密码，orcl是数据库名称，file是存放要导出文件的url

**二.导入工具 imp**

1. 它是操作系统下一个可执行的文件 存放目录/ORACLE\_HOME/bin

imp导入工具将EXP形成的二进制系统文件导入到数据库中.

它有三种模式：

a.  用户模式： 导入用户所有对象以及对象中的数据；  
b.  表模式： 导入用户所有表或者指定的表；  
c.  整个数据库： 导入数据库中所有对象。  
        
只有拥有IMP\_FULL\_DATABASE和DBA权限的用户才能做整个数据库导入  
    
imp步骤：

(1) create table  (2) insert data  (3) create index (4) create triggers,constraints

2.导入工具imp交互式命令行方式的例子：

$ imp  
Import: Release 8.1.6.0.0 - Production on 星期五 12月 7 17:01:08 2001  
(c) Copyright 1999 Oracle Corporation.  All rights reserved.  
用户名:  test  
口令:\*\*\*\*  
连接到: Oracle8i Enterprise Edition Release 8.1.6.0.0 - 64bit Production  
With the Partitioning option  
JServer Release 8.1.6.0.0 - Production  
导入文件: expdat.dmp> /tmp/m.dmp  
输入插入缓冲区大小（最小为 8192 ) 30720>  
经由常规路径导出由EXPORT:V08.01.06创建的文件  
警告: 此对象由 TEST 导出, 而不是当前用户  
已经完成ZHS16GBK字符集和ZHS16GBK NCHAR 字符集中的导入  
只列出导入文件的内容(yes/no)：no>  
由于对象已存在, 忽略创建错误(yes/no)：no> yes  
导入权限(yes/no)：yes>  
导入表数据(yes/no)：yes>  
导入整个导出文件(yes/no)：no> yes  
. 正在将TEST的对象导入到 SCOTT  
. . 正在导入表                       "CMAMENU"       4336行被导入  
成功终止导入，但出现警告。

导入命令：

复制代码 代码如下:

$imp test/test123@orcl file=/opt/whufe.dmp full=y

其中test是用户名称，后一个test123是用户密码，orcl是数据库名称，file是存放要导入文件的url

## [Oracle 增加修改删除字段](http://www.cnblogs.com/laipDIDI/articles/2615210.html)

添加字段的语法：alter table tablename add (column datatype [default value][null/not null],….);

修改字段的语法：alter table tablename modify (column datatype [default value][null/not null],….);

删除字段的语法：alter table tablename drop (column);

## 分页SQL

假设要分页的表名为tab\_a

SELECT

COUNT(1) OVER(PARTITION BY FLAG) CNT,

ROW\_NUMBER OVER(PARTITION BY FLAG ORDER BY ‘字段’) RS,

QR.\*

FROM

(

SELECT

1 FLAG,

A.\*

FROM

(

SELECT

\*

FROM

tab\_a

) A

) QR

## Oracle数据库快照

需要将业务[**数据库**](http://lib.csdn.net/base/mysql)里所有的数据同步到另外一个处理服务器上。在做方案的时候，想了很多方法，当然最快的办法还是使用物理热备的方式。  
　 但是个人认为如果对于中大型数据库(我们的数据库有300G左右)最佳的选择还是使用SnapShot方式，即快照的方式。  
　　[**Oracle**](http://lib.csdn.net/base/oracle)数据库的快照是一个表，它包含有对一个本地或远程数据库上一个或多个表或视图的查询的结果。也就是说快照根本的原理就是将本地或远程数据库上的一个查询结果保存在一个表中。  
　　以下是我建立的Snapshot，目的是从业务数据库上将数据Copy到处理数据库上，是不同的两个服务器之间对数据copy。  
　　第一步：在处理服务器上的[**oracle**](http://lib.csdn.net/base/oracle)终端，建立database link，业务数据库服务器SID为TEST  
　　create database link TEST\_DBLINK.US.ORACLE.COM  
　　connect to AMICOS identified by AMICOS  
　　using ’test’;  
　　第二步：在业务数据库上对应的表建立快照日志  
　　Create snapshot log on A\_Table;  
　　第三步：建立Snapshot 快照名称为：Test\_SnapShot  
　　Create snapshot Test\_SnapShot  
　　REFRESH COMPLETE START WITH SYSDATE NEXT SYSDATE+1/24  
　　as select \* from [A\_Table@TEST\_DBLINK](mailto:A_Table@TEST_DBLINK)  
　　说明：REFRESH是刷新方法  
　　刷新方式有：COMPLETE和FAST两种，而START WITH是说明开始执行的时间。  
　　Next是下次执行的时间  
　　而AS以后是构成快照的查询方法。  
　　相关的方法：  
　　更改快照  
　　ALTER SNAPSHOT Test\_SnapShot  
　　REFRESH COMPLETE START WITH SYSDATE NEXT SYSDATE+1/2;  
　　手动刷新快照 在命令界面执行：  
　　EXEC DBMS\_SNAPSHOT.REFRESH(’Test\_SnapShot ’,’C’);  
　　第一个参数是要刷新的快照名  
　　第二个参数是刷新的方式，F----FAST, C---COMPLETE  
　　查看快照最后刷新的日期  
　　SELECT NAME,LAST\_REFRESH  
　　FROM ALL\_SNAPSHOT\_REFRESH\_TIMES;  
　　考试大整理最后非常的方案：  
　　1：为需要做Snapshot的表建立Snapshot日志  
　　create snapshot log on t1 with rowid; 这里使用ROWID建立日记的参数  
　　2：采用Fast的方式建立快照，使用rowid做为参考参数  
　　create snapshot fb\_test\_b refresh fast with rowid start with sysdate next sysdate+1/1440 as select \* from [fb\_test\_b@my\_dblink](mailto:fb_test_b@my_dblink);  
　　最好能按照rowid来建立快照。要不然就必须要为表建立Primary Key。

--名词说明：源——被同步的数据库  
      目的——要同步到的数据库

前6步必须执行,第6以后是一些辅助信息.

--1、在目的数据库上，创建dblink  
drop public database link dblink\_orc92\_182;  
Create public DATABASE LINK dblink\_orc92\_182 CONNECT TO bst114 IDENTIFIED BY password USING 'orc92\_192.168.254.111';  
--dblink\_orc92\_182 是dblink\_name  
--bst114 是 username  
--password 是 password  
--''orc92\_192.168.254.111'' 是远程数据库名

--2、在源和目的数据库上创建要同步的表(最好有主键约束,快照才可以快速刷新)  
drop table test\_user;  
create table test\_user(id number(10) primary key,name varchar2(12),age number(3));

--3、在目的数据库上，[**测试**](http://lib.csdn.net/base/softwaretest)dblink  
select \* from ;    //查询的是源数据库的表  
select \* from test\_user;

--4、在源数据库上，创建要同步表的快照日志  
Create snapshot log on test\_user;

--5、创建快照，在目的数据库上创建快照  
Create snapshot sn\_test\_user as select \* from ;

--6、设置快照刷新时间(只能选择一种刷新方式,推荐使用快速刷新,这样才可以用触发器双向同步)  
快速刷新  
Alter snapshot sn\_test\_user refresh fast Start with sysdate next sysdate with primary key;  
--oracle马上自动快速刷新，以后不停的刷新,只能在测试时使用.真实项目要正确权衡刷新时间.

完全刷新  
Alter snapshot sn\_test\_user refresh complete Start with sysdate+30/(24\*60\*60) next sysdate+30/(24\*60\*60);  
--oracle自动在30秒后进行第一次完全刷新，以后每隔30秒完全刷新一次

--7、手动刷新快照,在没有自动刷新的情况下,可以手动刷新快照.  
手动刷新方式1  
begin  
dbms\_refresh.refresh(''sn\_test\_user'');  
end;

手动刷新方式2  
EXEC DBMS\_SNAPSHOT.REFRESH(''sn\_test\_user'',''F'');  //第一个参数是快照名,第二个参数 F 是快速刷新 C 是完全刷新.

--8.修改会话时间格式  
ALTER SESSION SET NLS\_DATE\_FORMAT = ''YYYY-MM-DD HH24:MI:SS'';

--9.查看快照最后一次刷新时间  
SELECT NAME,LAST\_REFRESH FROM ALL\_SNAPSHOT\_REFRESH\_TIMES;

--10.查看快照下次执行时间  
select last\_date,next\_date,what from user\_jobs order by next\_date;

--11.打印调试信息  
dbms\_output.put\_line(''use ''||''plsql'');

--12.如果你只想单向同步,那么在目的数据库创建以下触发器(当源数据库表改变时,目的数据库表跟着改变,但目的数据库表改变时,源数据库表不改变).  
create or replace trigger TRI\_test\_user\_AFR  
  after  insert or update or delete on sn\_test\_user  
  for each row  
begin  
  if deleting then  
      delete from test\_user where >  end if;  
  if inserting then  
      insert into test\_user(id,name)  
      values(:new.id,:new.name);  
  end if;  
  if updating then  
     update test\_user set where >  end if;  
end TRI\_test\_user\_AFR;

--13.如果你想双向同步,请在源数据库中执行前6步,并在双方都创建以下触发器(当源数据库表改变时,目的数据库表跟着改变,目的数据库表改变时,源数据库表也改变)  
CREATE OR REPLACE TRIGGER BST114.TRI\_TEST\_USER\_AFR  
AFTER DELETE OR INSERT OR UPDATE  
ON BST114.SN\_TEST\_USER   
REFERENCING NEW AS NEW OLD AS OLD  
FOR EACH ROW  
declare  
    tmp\_id number(10):=-1;  
begin

  dbms\_output.put\_line(''begin'');  
  if inserting then  
      --select id into tmp\_id from test\_user where   
      for p in(select id from test\_user where >      loop  
        tmp\_id:=p.id;  
      end loop;  
        
      dbms\_output.put\_line(tmp\_id||''===------------'');  
      if (tmp\_id=-1) then  
          insert into test\_user(id,name,age)  
          values(:new.id,:new.name,:new.age);  
      end if;  
  end if;  
    
  if updating then  
     dbms\_output.put\_line(''updated'');  
     for p in(select name,age from test\_user where >     loop  
         if (p.name!=:new.name) or (p.age!=:new.age) then  
              update test\_user set where >         end if;  
     end loop;  
  end if;  
    
  if deleting then  
      dbms\_output.put\_line(''deleted'');  
      delete from test\_user where >  end if;  
  dbms\_output.put\_line(''end'');  
end TRI\_test\_user\_AFR;  
 --为防止双向同步触发器死循环,所以要在触发器中增加一些判断,阻止死循环.

--以上同步原理  
1.首先创建一个dblink,可以访问远程数据库  
2.在本地创建一个快照,映射远程数据表,当远程数据表有变化时,会反应到快照中.  
3.由于快照类似于视图表,所以在本地为快照创建一个触发器,当快照有变化时,会触发相应事件.  
4.在触发器中写同步数据的代码.

--附:快照刷新时间参数说明  
一天的秒数=24小时\*60分钟\*60钞  
所以要想在30秒后刷新,参数应该这样写 sysdate+30/(24\*60\*60)  
1分钟==sysdate+60/(24\*60\*60)

一天的分钟数=24小时\*60分钟  
一分钟也可以这样写 sysdate+1/(24\*60)  
30分钟==sysdate+30/(24\*60)  
60分钟==sysdate+60/(24\*60)

以此类推  
1小时==sysdate+1/24==sysdate+60/(24\*60)  
1天==sysdate+1  
一个月==sysdate+30

## 查看oracle配置

select \* from dba\_profiles;

## 查看表的约束

select   \*  from  user\_constraints where table\_name= 'MEDIA'

select   \*  from  all\_constraints  where table\_name= 'MEDIA'

select   \*  from  dba\_constraints  where table\_name= 'MEDIA'\

有时想删除一个表上的记录 , 由于创建的约束比较多 , 不知道约束在哪个表上 , 无法删除记录 , 用 CASCADE 也不行 . 这时就要找到约束所在的表 , 可以用以下 SQL 语句查询出约束所在的表 :

SELECT \* FROM  USER\_CONSTRAINTS  WHERE CONSTRAINT\_NAME = 'CONSTRAINT\_NAME';

再删除约束所在的表的具有约束限制的记录

## 查看所有的表

select table\_name from user\_tables; //当前用户拥有的表

select table\_name from all\_tables; //所有用户的表

select table\_name from dba\_tables; //包括系统表

select table\_name from dba\_tables where owner='用户名'

查看所有的数据库对象

select \* from user\_objects

## 查看表字段

select \* from user\_tab\_columns where Table\_Name='用户表';

select \* from all\_tab\_columns where Table\_Name='用户表';

select \* from dba\_tab\_columns where Table\_Name='用户表';

注意表名要大写

## 查看表注释

select \* from user\_tab\_comments

select \* from all\_tab\_comments

select \* from dba\_tab\_comments

## 查看字段注释

select \* from user\_col\_comments

select \* from all\_col\_comments

select \* from dba\_col\_comments

## oracle中插入blob数据

ociblbl \*pBlobLocator;

int iOffset = 1;

int iWriteAmt;

EXEC SQL BEGIN DECLARE SECTION

char caDummyMesg[30000+1];

EXEC SQL END DECLARE SECTION

EXEC SQL VAR caDummyMesg IS RAW[30000+1]

memset(caDummyMesg ,0,3 0000+1);

strcpy(caDummyMesg,”要插入的数据”);

EXEC SQL ALLOCATE :pBlobLocator;

EXEC SQL INSERT INTO 表名(,’blob字段名’,)

values(,empty\_blob(),)

returning ’blob字段名’ into :pBlobLocator

iWriteAmt = “插入数据长度”

EXEC SQL LOB WRITE ONE : iWriteAmt FROM : caDummyMesg INTO :pBlobLocator AT :iOffset;

EXEC SQL FREE :pBlobLocator;

## oracle中读取blob数据

EXEC SQL BEGIN DECLARE SECTION;

int iBlobLen = 0;

int iOffset = 1;

char caMesg[30000+1];

EXEC SQL END DECLARE SECTION;

EXEC SQL VAR caMesg IS RAW[30000+1];

EXEC SQL ALLOCATE :pBlobLocator;

EXEC SQL SELECT

‘ 二进制字段名’

FROM

表名

INTO

:pBlobLocator

EXEC SQL LOB DESCRIBE :pBlobLocator GET LENGTH INTO : iBlobLen;

EXEC SQL LOB READ : iBlobLen FROM :pBlobLocator AT :iOffset INTO

:caMesg;

EXEC SQL FREE :pBlobLocator;

## oracle中的hash函数

dbms\_utility.get\_hash\_value(‘strings', min, max)  
    strings：输入值  
    min：hash bucket最小值  
    max：hash bucket最大值

### ora\_hash('strings', N, 0 )

strings: 输入值  
    N：最大hash bucket的值  
    0：起始hash bucket值，缺省是1

## 清空表

truncate table

表名

## [ROW\_NUMBER() OVER() 函数的用法](http://blog.csdn.net/iw1210/article/details/11937085)

ROW\_NUMBER() OVER(partition by col1 order by col2) 表示根据col1分组，在分组内部根据col2排序，而此函数计算的值就表示每组内部排序后的顺序编号（组内是连续且唯一的）。

## listtagg（）函数用法

LISTAGG(XXX,YYY) WITHIN GROUP( ORDER BY ZZZ)

还有一个高级用法：就是over(partition by XXX)

with temp as(

select 'China' nation ,'Guangzhou' city from dual union all

select 'China' nation ,'Shanghai' city from dual union all

select 'China' nation ,'Beijing' city from dual union all

select 'USA' nation ,'New York' city from dual union all

select 'USA' nation ,'Bostom' city from dual union all

select 'Japan' nation ,'Tokyo' city from dual

)

select nation,listagg(city,',') within GROUP (order by city)

from temp

group by nation

with temp as(

select 500 population, 'China' nation ,'Guangzhou' city from dual union all

select 1500 population, 'China' nation ,'Shanghai' city from dual union all

select 500 population, 'China' nation ,'Beijing' city from dual union all

select 1000 population, 'USA' nation ,'New York' city from dual union all

select 500 population, 'USA' nation ,'Bostom' city from dual union all

select 500 population, 'Japan' nation ,'Tokyo' city from dual

)

select population,

nation,

city,

listagg(city,',') within GROUP (order by city) over (partition by nation) rank

from temp

## trunc()函数的用法

### 日期

1.select trunc(sysdate) from dual --2013-01-06 今天的日期为2013-01-06

select trunc(sysdate-n) from dual 取出前n天的日期

2.select trunc(sysdate, 'mm') from dual --2013-01-01 返回当月第一天.

3.select trunc(sysdate,'yy') from dual --2013-01-01 返回当年第一天

4.select trunc(sysdate,'dd') from dual --2013-01-06 返回当前年月日

5.select trunc(sysdate,'yyyy') from dual --2013-01-01 返回当年第一天

6.select trunc(sysdate,'d') from dual --2013-01-06 (星期天)返回当前星期的第一天

7.select trunc(sysdate, 'hh') from dual --2013-01-06 17:00:00 当前时间为17:35

8.select trunc(sysdate, 'mi') from dual --2013-01-06 17:35:00 TRUNC()函数没有秒的精确

### 数字

TRUNC（number,num\_digits）

Number 需要截尾取整的数字。

Num\_digits 用于指定取整精度的数字。Num\_digits 的默认值为 0。

TRUNC()函数截取时不进行四舍五入

9. select trunc(123.458) from dual --123

10.select trunc(123.458,0) from dual --123

11.select trunc(123.458,1) from dual --123.4

12.select trunc(123.458,-1) from dual --120

13.select trunc(123.458,-4) from dual --0

14.select trunc(123.458,4) from dual --123.458

15.select trunc(123) from dual --123

16.select trunc(123,1) from dual --123

17.select trunc(123,-1) from dual --120

## SELECT INTO 和 INSERT INTO SELECT 两种表复制语句详解

我们经常会遇到需要表复制的情况，如将一个table1的数据的部分字段复制到table2中，或者将整个table1复制到table2中，这时候我们就要使用SELECT INTO 和 INSERT INTO SELECT 表复制语句了。

### 1.INSERT INTO SELECT语句

语句形式为：Insert into Table2(field1,field2,...) select value1,value2,... from Table1

注意：

（1）要求目标表Table2必须存在，并且字段field,field2...也必须存在

（2）注意Table2的主键约束，如果Table2有主键而且不为空，则 field1， field2...中必须包括主键

（3）注意语法，不要加values，和插入一条数据的sql混了，不要写成:

Insert into Table2(field1,field2,...) values (select value1,value2,... from Table1)

由于目标表Table2已经存在，所以我们除了插入源表Table1的字段外，还可以插入常量。示例如下：

+ expand sourceview plaincopy to clipboardprint

--1.创建测试表

create TABLE Table1

(

a varchar(10),

b varchar(10),

c varchar(10)

)

create TABLE Table2

(

a varchar(10),

c varchar(10),

d int

)

--2.创建测试数据

Insert into Table1 values('赵','asds','90')

Insert into Table1 values('钱','asds','100')

Insert into Table1 values('孙','asds','80')

Insert into Table1 values('李','asds',null)

select \* from Table2

--3.INSERT INTO SELECT语句复制表数据

Insert into Table2(a, c, d) select a,c,5 from Table1

--4.显示更新后的结果

select \* from Table2

--5.删除测试表

drop TABLE Table1

drop TABLE Table2

### 2.SELECT INTO FROM语句

语句形式为：SELECT vale1, value2 into Table2 from Table1

要求目标表Table2不存在，因为在插入时会自动创建表Table2，并将Table1中指定字段数据复制到Table2中。示例如下：

view plaincopy to clipboardprint?

--1.创建测试表

create TABLE Table1

(

a varchar(10),

b varchar(10),

c varchar(10)

)

--2.创建测试数据

Insert into Table1 values('赵','asds','90')

Insert into Table1 values('钱','asds','100')

Insert into Table1 values('孙','asds','80')

Insert into Table1 values('李','asds',null)

--3.SELECT INTO FROM语句创建表Table2并复制数据

select a,c INTO Table2 from Table1

--4.显示更新后的结果

select \* from Table2

--5.删除测试表

drop TABLE Table1

drop TABLE Table2

注意：如果在sql/plus或者PL/SQL执行这条语句，会报"ORA-00905:缺失关键字"错误，原因是PL/Sql与T-SQL的区别。

T-SQL中该句正常，但PL/SQL中解释是:

select..into is part of PL/SQL language which means you have to use it inside a PL/SQL block. You can not use it in a SQL statement outside of PL/SQL.

即不能单独作为一条sql语句执行，一般在PL/SQL程序块(block)中使用。

如果想在PL/SQL中实现该功能，可使用Create table newTable as select \* from ...：

如： create table NewTable as select \* from ATable;

NewTable 除了没有键，其他的和ATable一样

### SQL SELECT INTO语法介绍

SQL SELECT INTO 语句可用于创建表的备份复件。

SELECT INTO 语句从一个表中选取数据，然后把数据插入另一个表中。

SELECT INTO 语句常用于创建表的备份复件或者用于对记录进行存档。

SQL SELECT INTO 语法

您可以把所有的列插入新表：

SELECT \* INTO new\_table\_name [IN externaldatabase] FROM old\_tablename

或者只把希望的列插入新表：

SELECT column\_name(s) INTO new\_table\_name [IN externaldatabase] FROM old\_tablename

#### SQL SELECT INTO 实例 - 制作备份复件

下面的例子会制作 "Persons" 表的备份复件：

SELECT \* INTO Persons\_backup FROM Persons

IN 子句可用于向另一个数据库中拷贝表：

SELECT \* INTO Persons IN 'Backup.mdb' FROM Persons

如果我们希望拷贝某些域，可以在 SELECT 语句后列出这些域：

SELECT LastName,FirstName

INTO Persons\_backup

FROM Persons

#### SQL SELECT INTO 实例 - 带有 WHERE 子句

我们也可以添加 WHERE 子句。

下面的例子通过从 "Persons" 表中提取居住在 "Beijing" 的人的信息，创建了一个带有两个列的名为 "Persons\_backup" 的表：

SELECT LastName,Firstname INTO Persons\_backup FROM Persons WHERE City='Beijing'

#### SQL SELECT INTO 实例 - 被连接的表

从一个以上的表中选取数据也是可以做到的。

下面的例子会创建一个名为 "Persons\_Order\_Backup" 的新表，其中包含了从 Persons 和 Orders 两个表中取得的信息：

SELECT Persons.LastName,Orders.OrderNo

INTO Persons\_Order\_Backup

FROM Persons

INNER JOIN Orders

ON Persons.Id\_P=Orders.Id\_P

## to\_date() 与 to\_char() 日期和字符串转换

TO\_CHAR 是把日期或数字转换为字符串  
TO\_DATE 是把字符串转换为[数据库](http://lib.csdn.net/base/mysql)中得日期类型转换函数

TO\_DATE格式(以时间:2016-07-25 11:45:25为例)

**Year:**

yy two digits 两位年 显示值:16

yyy three digits 三位年 显示值:016

yyyy four digits 四位年 显示值:2016

**Month:**

mm number 两位月 显示值:07

mon abbreviated 字符集表示 显示值:07月,若是英文版,显示jul

month spelled out 字符集表示 显示值:07月,若是英文版,显示july

**Day:**

dd number 当月第几天 显示值:25

ddd number 当年第几天 显示值:25

dy abbreviated 当周第几天简写 显示值:星期一,若是英文版,显示mon

day spelled out 当周第几天全写 显示值:星期一,若是英文版,显示monday

ddspth spelled out, ordinal twelfth

**Hour:**

hh two digits 12小时进制 显示值:11

hh24 two digits 24小时进制 显示值:11

**Minute:**

mi two digits 60进制 显示值:45

**Second:**

ss two digits 60进制 显示值:25

**其它**

Q digit 季度 显示值:4

WW digit 当年第几周 显示值:44

W digit 当月第几周 显示值:1

### 1. 日期和字符转换函数用法（to\_date,to\_char）

select to\_char(sysdate,'yyyy-mm-ddhh24:mi:ss') as nowTime from dual; //日期转化为字符串

select to\_char(sysdate,'yyyy') asnowYear from dual; //获取时间的年

select to\_char(sysdate,'mm') as nowMonth from dual; //获取时间的月

select to\_char(sysdate,'dd') as nowDay from dual; //获取时间的日

select to\_char(sysdate,'hh24') as nowHour from dual; //获取时间的时

select to\_char(sysdate,'mi') as nowMinute from dual; //获取时间的分

select to\_char(sysdate,'ss') as nowSecond from dual; //获取时间的秒

select to\_date('2004-05-0713:23:44','yyyy-mm-dd hh24:mi:ss') from dual//

### 2. select to\_char( to\_date(222,'J'),'Jsp') from dual

显示Two Hundred Twenty-Two

### 3.求某天是星期几

select to\_char(to\_date('2002-08-26','yyyy-mm-dd'),'day') from dual;

星期一

select to\_char(to\_date('2002-08-26','yyyy-mm-dd'),'day','NLS\_DATE\_LANGUAGE= American') from dual;

monday

设置日期语言

ALTER SESSION SET NLS\_DATE\_LANGUAGE='AMERICAN';

也可以这样

TO\_DATE ('2002-08-26', 'YYYY-mm-dd', 'NLS\_DATE\_LANGUAGE =American')

### 4. 两个日期间的天数

select floor(sysdate - to\_date('20020405','yyyymmdd')) from dual;

### 5. 时间为null的用法

select id, active\_date from table1

UNION

select 1, TO\_DATE(null) from dual;

注意要用TO\_DATE(null)

### 6.月份差

a\_date between to\_date('20011201','yyyymmdd') andto\_date('20011231','yyyymmdd')

那么12月31号中午12点之后和12月1号的12点之前是不包含在这个范围之内的。

所以，当时间需要精确的时候，觉得to\_char还是必要的

### 7. 日期格式冲突问题

输入的格式要看你安装的Oracle字符集的类型, 比如: US7ASCII, date格式的类型就是: '01-Jan-01'

alter system set NLS\_DATE\_LANGUAGE =American

alter session set NLS\_DATE\_LANGUAGE = American

或者在to\_date中写

selectto\_char(to\_date('2002-08-26','yyyy-mm-dd'),'day','NLS\_DATE\_LANGUAGE =American') from dual;

注意我这只是举了NLS\_DATE\_LANGUAGE，当然还有很多，

可查看

select \* from nls\_session\_parameters

select \* from V$NLS\_PARAMETERS

### 8. 查找2002-02-28至2002-02-01间除星期一和七的天数

select

count(\*)

from ( select rownum-1 rnum

from all\_objects

where rownum <= to\_date('2002-02-28','yyyy-mm-dd') -to\_date('2002-

02-01','yyyy-mm-dd')+1

)

where

to\_char( to\_date('2002-02-01','yyyy-mm-dd')+rnum-1, 'D' )

not in

( '1', '7' )

在前后分别调用DBMS\_UTILITY.GET\_TIME, 让后将结果相减(得到的是1/100秒, 而不是毫秒).

### 9. 查找月份

selectmonths\_between(to\_date('01-31-1999','MM-DD-YYYY'),to\_date('12-31-1998','MM-DD-YYYY'))"MONTHS" FROM DUAL;

1

select months\_between(to\_date('02-01-1999','MM-DD-YYYY'),to\_date('12-31-1998','MM-DD-YYYY'))"MONTHS" FROM DUAL;

1.03225806451613

### 10. Next\_day的用法

Next\_day(date, day)

Monday-Sunday, for format code DAY

Mon-Sun, for format code DY

1-7, for format code D

### 11 select to\_char(sysdate,'hh:mi:ss') TIME from all\_objects

注意：第一条记录的TIME 与最后一行是一样的

可以建立一个函数来处理这个问题

create or replace function sys\_date return date is

begin

return sysdate;

end;

select to\_char(sys\_date,'hh:mi:ss') from all\_objects;

### 12.获得小时数

extract()找出日期或间隔值的字段值

SELECT EXTRACT(HOUR FROM TIMESTAMP '2001-02-16 2:38:40') from offer

SQL> select sysdate ,to\_char(sysdate,'hh') from dual;

SYSDATE TO\_CHAR(SYSDATE,'HH')

-------------------- ---------------------

2003-10-13 19:35:21 07

SQL> select sysdate ,to\_char(sysdate,'hh24') from dual;

SYSDATE TO\_CHAR(SYSDATE,'HH24')

-------------------- -----------------------

2003-10-13 19:35:21 19

### 13.年月日的处理

select older\_date,

newer\_date,

years,

months,

abs(

trunc(

newer\_date-

add\_months( older\_date,years\*12+months )

)

) days

from ( select

trunc(months\_between( newer\_date, older\_date )/12) YEARS,

mod(trunc(months\_between( newer\_date, older\_date )),12 ) MONTHS,

newer\_date,

older\_date

from (

select hiredate older\_date,add\_months(hiredate,rownum)+rownum newer\_date

from emp

)

)

### 14.处理月份天数不定的办法

select to\_char(add\_months(last\_day(sysdate) +1, -2),'yyyymmdd'),last\_day(sysdate) from dual

### 15.找出今年的天数

select add\_months(trunc(sysdate,'year'), 12) - trunc(sysdate,'year')from dual

闰年的处理方法

to\_char( last\_day( to\_date('02' | | :year,'mmyyyy') ), 'dd' )

如果是28就不是闰年

### 16.yyyy与rrrr的区别

'YYYY99 TO\_C

------- ----

yyyy 99 0099

rrrr 99 1999

yyyy 01 0001

rrrr 01 2001

### 17.不同时区的处理

select to\_char( NEW\_TIME( sysdate, 'GMT','EST'), 'dd/mm/yyyy hh:mi:ss'),sysdate

from dual;

### 18.5秒钟一个间隔

Select TO\_DATE(FLOOR(TO\_CHAR(sysdate,'SSSSS')/300)\* 300,'SSSSS'),TO\_CHAR(sysdate,'SSSSS')

from dual

2002-11-1 9:55:00 35786

SSSSS表示5位秒数

### 19.一年的第几天

select TO\_CHAR(SYSDATE,'DDD'),sysdate from dual

310 2002-11-6 10:03:51

### 20.计算小时,分,秒,毫秒

select

Days,

A,

TRUNC(A\*24) Hours,

TRUNC(A\*24\*60 - 60\*TRUNC(A\*24)) Minutes,

TRUNC(A\*24\*60\*60 - 60\*TRUNC(A\*24\*60)) Seconds,

TRUNC(A\*24\*60\*60\*100 - 100\*TRUNC(A\*24\*60\*60)) mSeconds

from

(

select

trunc(sysdate) Days,

sysdate - trunc(sysdate) A

from dual

)

select \* from tabname

order by decode(mode,'FIFO',1,-1)\*to\_char(rq,'yyyymmddhh24miss');

//

floor((date2-date1) /365) 作为年

floor((date2-date1, 365) /30) 作为月

d(mod(date2-date1, 365), 30)作为日.

### 21.next\_day函数

返回下个星期的日期,day为1-7或星期日-星期六,1表示星期日

next\_day(sysdate,6)是从当前开始下一个星期五。后面的数字是从星期日开始算起。

12 3 4 5 6 7

日 一 二 三 四 五 六

---------------------------------------------------------------

select (sysdate-to\_date('2003-12-03 12:55:45','yyyy-mm-ddhh24:mi:ss'))\*24\*60\*60 from ddual

日期 返回的是天 然后转换为ss

### 22.round[舍入到最接近的日期](day:舍入到最接近的星期日)

select sysdate S1,

round(sysdate) S2 ,

round(sysdate,'year') YEAR,

round(sysdate,'month') MONTH ,

round(sysdate,'day') DAY from dual

### 23.trunc[截断到最接近的日期,单位为天] ,返回的是日期类型

select sysdate S1,

trunc(sysdate) S2, //返回当前日期,无时分秒

trunc(sysdate,'year') YEAR, //返回当前年的1月1日,无时分秒

trunc(sysdate,'month') MONTH , //返回当前月的1日,无时分秒

trunc(sysdate,'day') DAY //返回当前星期的星期天,无时分秒

from dual

### 24.返回日期列表中最晚日期

select greatest('01-1月-04','04-1月-04','10-2月-04') from dual

### 25.计算时间差

注:oracle时间差是以天数为单位,所以换算成年月,日

select floor(to\_number(sysdate-to\_date('2007-11-02 15:55:03','yyyy-mm-ddhh24:mi:ss'))/365) as spanYears from dual //时间差-年

select ceil(moths\_between(sysdate-to\_date('2007-11-0215:55:03','yyyy-mm-dd hh24:mi:ss'))) as spanMonths from dual //时间差-月

select floor(to\_number(sysdate-to\_date('2007-11-02 15:55:03','yyyy-mm-ddhh24:mi:ss'))) as spanDays from dual //时间差-天

select floor(to\_number(sysdate-to\_date('2007-11-02 15:55:03','yyyy-mm-ddhh24:mi:ss'))\*24) as spanHours from dual //时间差-时

select floor(to\_number(sysdate-to\_date('2007-11-02 15:55:03','yyyy-mm-ddhh24:mi:ss'))\*24\*60) as spanMinutes from dual //时间差-分

selectfloor(to\_number(sysdate-to\_date('2007-11-02 15:55:03','yyyy-mm-ddhh24:mi:ss'))\*24\*60\*60) as spanSeconds from dual //时间差-秒

### 26.更新时间

注:oracle时间加减是以天数为单位,设改变量为n,所以换算成年月,日

select to\_char(sysdate,'yyyy-mm-ddhh24:mi:ss'),to\_char(sysdate+n\*365,'yyyy-mm-dd hh24:mi:ss') as newTime fromdual //改变时间-年

select to\_char(sysdate,'yyyy-mm-dd hh24:mi:ss'),add\_months(sysdate,n) asnewTime from dual //改变时间-月

select to\_char(sysdate,'yyyy-mm-ddhh24:mi:ss'),to\_char(sysdate+n,'yyyy-mm-dd hh24:mi:ss') as newTime fromdual //改变时间-日

select to\_char(sysdate,'yyyy-mm-ddhh24:mi:ss'),to\_char(sysdate+n/24,'yyyy-mm-dd hh24:mi:ss') as newTime fromdual //改变时间-时

select to\_char(sysdate,'yyyy-mm-ddhh24:mi:ss'),to\_char(sysdate+n/24/60,'yyyy-mm-dd hh24:mi:ss') as newTime fromdual //改变时间-分

select to\_char(sysdate,'yyyy-mm-ddhh24:mi:ss'),to\_char(sysdate+n/24/60/60,'yyyy-mm-dd hh24:mi:ss') as newTimefrom dual //改变时间-秒

### 27.查找月的第一天,最后一天

SELECT Trunc(Trunc(SYSDATE, 'MONTH') - 1, 'MONTH') First\_Day\_Last\_Month,

Trunc(SYSDATE, 'MONTH') - 1 / 86400 Last\_Day\_Last\_Month,

Trunc(SYSDATE, 'MONTH') First\_Day\_Cur\_Month,

LAST\_DAY(Trunc(SYSDATE, 'MONTH')) + 1 - 1 / 86400 Last\_Day\_Cur\_Month

FROM dual;

## [Oracle行列转换](http://www.cnblogs.com/liunanjava/p/4961923.html)

### ****一、建表与插入数据****

1.1、建表

[复制代码](javascript:void(0);)

1 create table kecheng

2 (

3 id NUMBER,

4 name VARCHAR2(**20**),

5 course VARCHAR2(**20**),

6 score NUMBER

7 );

8 insert into kecheng (id, name, course, score)

9 values (**1**, '张三', '语文', **67**);

10 insert into kecheng (id, name, course, score)

11 values (**1**, '张三', '数学', **76**);

12 insert into kecheng (id, name, course, score)

13 values (**1**, '张三', '英语', **43**);

14 insert into kecheng (id, name, course, score)

15 values (**1**, '张三', '历史', **56**);

16 insert into kecheng (id, name, course, score)

17 values (**1**, '张三', '化学', **11**);

18 insert into kecheng (id, name, course, score)

19 values (**2**, '李四', '语文', **54**);

20 insert into kecheng (id, name, course, score)

21 values (**2**, '李四', '数学', **81**);

22 insert into kecheng (id, name, course, score)

23 values (**2**, '李四', '英语', **64**);

24 insert into kecheng (id, name, course, score)

25 values (**2**, '李四', '历史', **93**);

26 insert into kecheng (id, name, course, score)

27 values (**2**, '李四', '化学', **27**);

28 insert into kecheng (id, name, course, score)

29 values (**3**, '王五', '语文', **24**);

30 insert into kecheng (id, name, course, score)

31 values (**3**, '王五', '数学', **25**);

32 insert into kecheng (id, name, course, score)

33 values (**3**, '王五', '英语', **8**);

34 insert into kecheng (id, name, course, score)

35 values (**3**, '王五', '历史', **45**);

36 insert into kecheng (id, name, course, score)

37 values (**3**, '王五', '化学', **1**);

38 commit;

[复制代码](javascript:void(0);)



### **二、固定行列转换**

**2.1、Decode方式**

[复制代码](javascript:void(0);)

SELECT ID,NAME,

SUM(DECODE(course,'语文',score,**0**)) 语文,--这里使用max,min都可以

SUM(DECODE(course,'数学',score,**0**)) 数学,

SUM(DECODE(course,'英语',score,**0**)) 英语,

SUM(DECODE(course,'历史',score,**0**)) 历史,

SUM(DECODE(course,'化学',score,**0**)) 化学

FROM kecheng

GROUP BY ID ,NAME

[复制代码](javascript:void(0);)



**2.2、Case方式**

[复制代码](javascript:void(0);)

SELECT ID,NAME,

MAX(CASE WHEN course='语文' THEN score ELSE **0** END) 语文,

MAX(CASE WHEN course='数学' THEN score ELSE **0** END) 数学,

MAX(CASE WHEN course='英语' THEN score ELSE **0** END) 英语,

MAX(CASE WHEN course='历史' THEN score ELSE **0** END) 历史,

MAX(CASE WHEN course='化学' THEN score ELSE **0** END) 化学

FROM kecheng

GROUP BY ID ,NAME

[复制代码](javascript:void(0);)

结果与上方一样

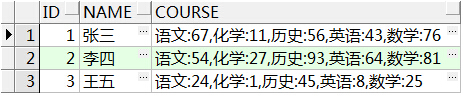
**2.3、wmsys.wm\_concat行列转换函数**

SELECT ID,NAME,

wmsys.wm\_concat(course || ':'||score) course

FROM kecheng

GROUP BY ID ,NAME;

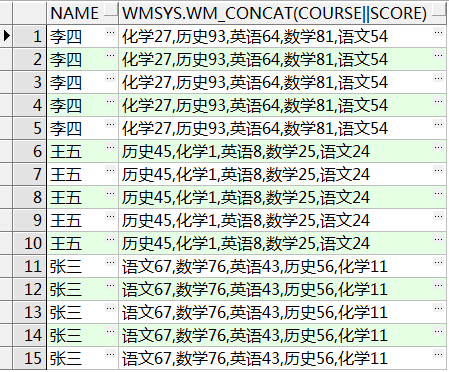


**2.4、使用over(partition by t.u\_id)用法**

SELECT NAME,

wmsys.wm\_concat(course ||score) OVER (PARTITION BY NAME)

FROM kecheng



### **三、动态**转**换**

**3.1、使用PL/SQL**

[复制代码](javascript:void(0);)

DECLARE

--存放最终的SQL

LV\_SQL VARCHAR2(**3000**);

--存放连接的SQL

SQL\_COMMOND VARCHAR2(**3000**);

--定义游标

CURSOR CUR IS

SELECT COURSE FROM KECHENG GROUP BY COURSE;

BEGIN

--定义查询开头

SQL\_COMMOND := 'SELECT NAME ';

FOR I IN CUR LOOP

--将结果相连接

SQL\_COMMOND := SQL\_COMMOND || ' ,SUM(DECODE(course,''' || I.COURSE ||

''',score,0)) ' || I.COURSE;

DBMS\_OUTPUT.PUT\_LINE(SQL\_COMMOND);

END LOOP;

SQL\_COMMOND := SQL\_COMMOND || ' from KECHENG group by name';

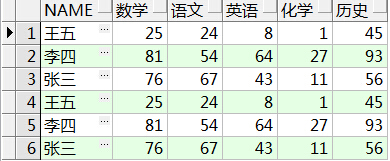
LV\_SQL := 'INSERT INTO temp\_ss ' || SQL\_COMMOND;

DBMS\_OUTPUT.PUT\_LINE(LV\_SQL);

EXECUTE IMMEDIATE LV\_SQL;

END;

[复制代码](javascript:void(0);)

temp\_ss 表  


## 嵌入式sql预处理

### 游标

将查询数据集sql以字符串的形式输出到宿主变量sCurString中

例如 ：strcpy(sCurString,”select \* from xxxx where xxxx”)

预编译sCurString

|  |
| --- |
| exec sql prepare FetchTableCurPre from :sCurString  if (sqlca.sqlcode){} |

声明游标

|  |
| --- |
| exec sql decare FetchTableCur cursor for FetchTableCurPre  if (sqlca.sqlcode){} |

打开游标

|  |
| --- |
| exec sql open FetchTableCur using  if (sqlca.sqlcode){} |

fetch 游标

|  |
| --- |
| exec sql fetch FetchTableCur into  if (sqlca.sqlcode){} |

关闭游标

|  |
| --- |
| exec sql close FetchTableCur  if (sqlca.sqlcode){} |

### 插入

将insert 语句以字符串的形式输出到宿主变量sInsertString中

例如 ：strcpy(sCurString,”insert into 表名(字段名1, 字段名2,…) values(值1, 值2,…)”)

预编译sCurString

|  |
| --- |
| exec sql prepare InsertrPre from : sInsertString  if (sqlca.sqlcode){} |

执行sql

|  |
| --- |
| exec sql execute InsertrPre using xxxx,xxxx,xxxx  if (sqlca.sqlcode){} |

## sqlca.sqlerrd[2]含义

sqlca.sqlerrd[2]当sql语句成功时,保存当前sql语句处理的行数.  
对于execute,insert,update,delete,select into反映成功处理的函数  
在open语句执行时,sqlerrd[2]置为0,在fetch后增值.

## Oracle的数据库，实例，服务名，SID

### 数据库

数据库（database）：物理操作系统文件或磁盘（disk）的集合。

比如常用的文件就是一种，在Oracle10G中，数据的存储有好几种。第一种是文件形式，也就是在你的磁盘中创建一批文件，然后在这些文件中存储信息。第二种就是磁盘阵列形式，这个是什么意思呢，这个就是说明数据库不是存放为某个文件，而是把一个或者多个磁盘格式化成Oracle的一种格式了，等于整个磁盘就是存放Oracle数据库的，不能作为别的用途。这样的优点是存储性能高，因为不再借助别的文件格式了，而是把整个磁盘都成为Oracle最适应的文件系统格式。当然还可能有别的形式，比如网络什么的。不过我们最常用的还是文件格式的，在文件格式中，数据库指的就是那些数据文件，控制文件以及REDO文件等等一系列文件。

数据库 = 重做文件 + 控制文件 + 数据文件 + 临时文件

### 实例instance

实例instance：一组Oracle后台进程/线程以及一个共享内存区，这些内存由同一个计算机上运行的线程/进程所共享。这里可以维护易失的、非持久性内容（有些可以刷新输出到磁盘）。就算没有磁盘存储，数据库实例也能存在。

instance其实就是指的操作系统中一系列的进程以及为这些进程所分配的内存块。在Oracle中，我们可以新建一个Oracle的Instance，这个时候虽然有了进程还有SGA等一系列的内存快，但是这个时候并没有把数据库文件读取进来。所以只是一个实例，在后来，你可以通过命令手动或者自动地把数据库文件加载进我们的数据库Instance中，这个时候的数据库才可以让我们真正的开始访问操作。

   所以说，数据库的应用如果想实现，数据库和数据库Instance是缺一不可的，如果只有数据库的那些文件，那么，只能代表数据在这个文件中，但是我们无法直接进行操作。而如果只有数据库Instance，那么我们虽然可以急性操作，但是也不知道操作哪些数据，操作生成的数据也无法保存等等。所以，当一个Oracle Instance真正Load了一个Oracle Database了以后，数据库才可以被我们使用。

在这里要注意一点的是，Oracle的实例在启动以后，只能load一次数据库，如果想把数据库与Instance断开，然后再重新挂在一个数据库Instance，那么就需要你首先把数据库Instance进程结束，然后重新建立这个instance的一个进程，再load另外一个数据库。否则肯定要抛除ORA-16169错误，说数据库已经被打开。因为一个数据库Instance在其生存期中最多只能load和打开一个instance.

ORACLE实例 = 进程 + 进程所使用的内存(SGA)

### 实例instance与数据库database的区别

ORACLE实例 = 进程 + 进程所使用的内存(SGA)

实例是一个临时性的东西，你也可以认为它代表了数据库某一时刻的状态！

数据库 = 重做文件 + 控制文件 + 数据文件 + 临时文件

数据库是永久的，是一个文件的集合。

ORACLE实例和数据库之间的关系

1. 临时性和永久性

2. 实例可以在没有数据文件的情况下单独启动 startup nomount , 通常没什么意义

3. 一个实例在其生存期内只能装载(alter database mount)和打开(alter database open)一个数据库

4. 一个数据库可被许多实例同时装载和打开(即RAC)，RAC环境中实例的作用能够得到充分的体现

5.一个Oracle数据库系统中可以同时安装几个数据库，每一个数据库对应一个唯  
一的实例，但是OPS系统除外，可以多个实例同时对一个数据库操作，称为并行服务  
器

### 服务名SERVICE\_NAMES

服务名 就是对外公布的名称，为网络监听服务，一个数据库可以对外公布多个服务名（SERVICE\_NAMES），service\_names是对外的服务名，是服务器端使用的

### 主机字符串

NET EASY CONFIG操纵的应该是主机字符串，是为客户端服务的，客户端也可以用多个主机字符串连接到同一个数据库服务器上。

“主机字符串”的叫法主要在SQL\*Plus中使用，是在客户端使用。

### SID

SID即是INSTANCE\_NAME

### 表空间

  一个数据库由一个或多个表空间组成，一个表空间只能属于一个数据库

  一个表空间由一个或多个多个数据文件组成，一个数据文件只能属于一个表空间

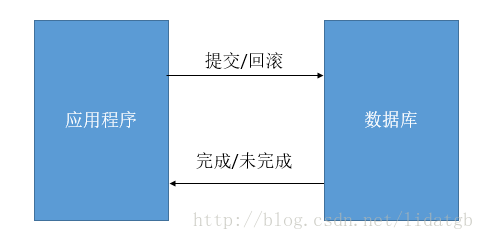
## [事务的一阶段提交协议和二阶段提交协议](http://blog.csdn.net/lidatgb/article/details/38468073)

  因为事务需要实现ACID，即原子性、一致性、隔离性、持久性，所以需要采用一定的机制来保证，通常采用的是分阶段提交的方式。

    XA:XA协议，规定事务管理器和资源管理器接口，采用二阶段提交协议。

### 一阶段提交协议

    一阶段提交协议相对简单，如下图：



    当然，前提是开启了事务，然后在应用程序发出提交/回滚请求后，[**数据库**](http://lib.csdn.net/base/mysql)执行操作，而后将成功/失败返回给应用程序，程序继续执行。

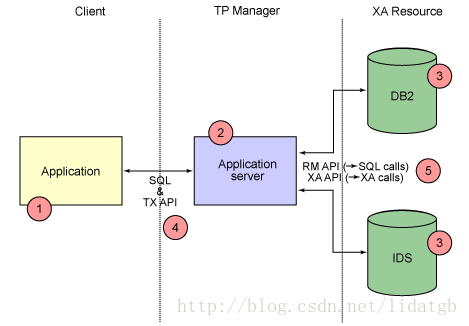
    一阶段提交协议相对简单，简单带来的优点就是，它不用再与其他的对象交互，节省了判断步骤和时间，所以在性能上是在阶段提交协议中最好的。

### 二阶段提交协议

    一阶段提交协议有其优点，但缺点也很明显：

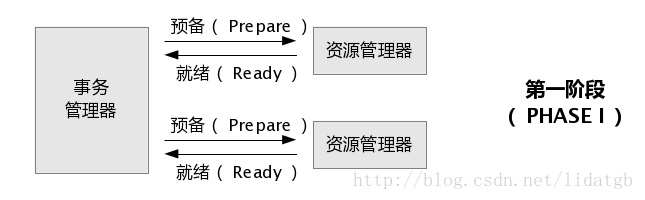
* 数据库确认执行事务的时间较长，出问题的可能性就随之增大。
* 如果有多个数据源，一阶段提交协议无法协调他们之间的关系。

   所以在一阶段协议的基础上，有了二阶段协议，二阶段协议的好处是添加了一个管理者角色，如下：



    很明显，二阶段协议通过将两层变为三层，增加了中间的管理者角色，从而协调多个数据源之间的关系，二阶段提交协议分为两个阶段。

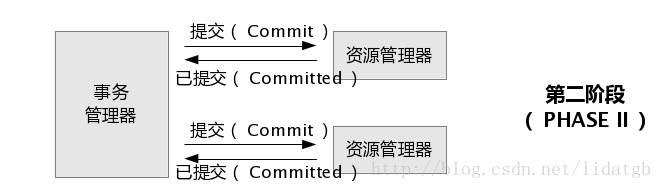
#### 第一阶段



    应用程序调用了事务管理器的提交方法，此后第一阶段分为两个步骤：

* 事务管理器通知参与该事务的各个资源管理器，通知他们开始准备事务。
* 资源管理器接收到消息后开始准备阶段，写好事务日志并执行事务，但不提交，然后将是否就绪的消息返回给事务管理器（此时已经将事务的大部分事情做完，以后的内容耗时极小）。

#### 第二阶段



    第二阶段也分为两个步骤：

* 事务管理器在接受各个消息后，开始分析，如果有任意其一失败，则发送回滚命令，否则发送提交命令。
* 各个资源管理器接收到命令后，执行（耗时很少），并将提交消息返回给事务管理器。

    事务管理器接受消息后，事务结束，应用程序继续执行。

#### 为什么要分两步执行？

一是因为分两步，就有了事务管理器统一管理的机会；二尽可能晚地提交事务，让事务在提交前尽可能地完成所有能完成的工作，这样，最后的提交阶段将是耗时极短，耗时极短意味着操作失败的可能性也就降低。

同时，二阶段提交协议为了保证事务的一致性，不管是事务管理器还是各个资源管理器，每执行一步操作，都会记录日志，为出现故障后的恢复准备依据。

#### 二阶段提交协议的弊端

二阶段提交协议的存在的弊端是阻塞，因为事务管理器要收集各个资源管理器的响应消息，如果其中一个或多个一直不返回消息，则事务管理器一直等待，应用程序也被阻塞，甚至可能永久阻塞。

### 事务与协议

    那么本地事务和分布式事务，分别采用的是哪些协议？我在[RedBooks](http://www.redbooks.ibm.com/redbooks/pdfs/sg246122.pdf)的一个文档中看到的是：

**Global transactions**

    Although the XAResource interface is intended to support two phase commit, the specification does not force an adapter to support two phase commit. However, if the resource adapter does implement XAResource it must also implement support for one phase commit. This allows the transaction manager to do one phase commit optimization (explained later) by setting the onePhase flag to true when doing acommit.……

**Local transactions**

    A local transaction is managed by the resource manager without the need for an   
external transaction manager, and can be utilized when only one resource is   
involved. **Local transactions only support one phase commit**, because they only   
reference one EIS.……

    大意是：虽然实现XA接口的目的是为了支持二阶段提交协议，但是它也支持一阶段提交协议。本地事务只支持一阶段提交；分布式事务默认采用的是二阶段提交，如果在分布式事务中非得使用一阶段提交协议，那么只要数据源多余一个就会抛出异常，如果只有一个数据源则正确执行。

总结

    一阶段提交协议和二阶段提交协议只是比较常用的两个，此外还有其他协议，可自行研究。

## 查看oracle的错误码

#### 切换到oracle用户

su – oracle

#### oerr ora 错误码

## 启动oracle数据库

#### 切换到oracle用户

su – oracle

#### sqlplus /nolog进入sqlplus

sqlplus /nolog进入sqlplus

#### 输入conn / as sysdba

#### 输入startup---启动数据库

#### 启动监听器lsnrctl start

退出sqlplus在oracle用户下启动监听器

lsnrctl start 启动监听器

lsnrctl status 查看监听器的状态

#### 输入shutdown –关闭数据库

## 启动oracle数据库脚本

#!/bin/sh

sqlplus ‘/as sysdba’ <<EOF

startup

EOF!

lsnrctl start

exit 0

## oracle连接数据库

sqlplus onl1/onl1@linuxdb

## oracle中listener.ora的位置

一般在 $ORACLE\_HOME/network/admin下面

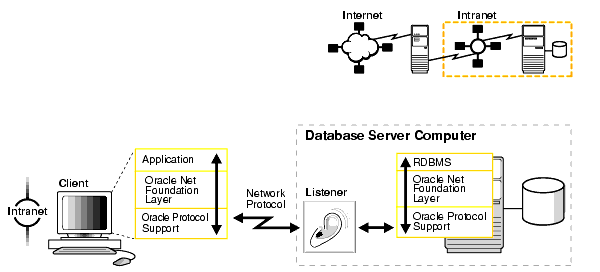
oracle

## oracle的初始化文件的位置

1、文本文件：$ORACLE\_HOME/dbs/initSID.ora （可以用文本编辑器查看，修改参数）  
2、二进制文件：$ORACLE\_HOME/dbs/spfileSID.ora （10G以后默认使用的参数文件）

## Oracle监听器 Listener

Oracle 监听器 Listener 是一个重要的数据库服务器组件，在整个 Oracle 体系结构中，扮演着重要的作用。它负责管理 Oracle 数据库和客户端之间的通讯，它在一个特定的网卡端口（默认是TCP 1521端口）上监听连接请求，并将连接转发给数据库，下面的部分，会从几个方面对监听器进行简单介绍。



### 1.监听器的功能

从当前的 Oracle 版本看，Listener 主要负责下面的几方面功能：

1. 监听客户端请求。监听器运行在数据库服务器之上，与 Oracle 实例（可为多个）相关关联，是一个专门的进程 process，在 Windows 的服务项目或者 Linux 的运行进程列表中，都会看到对应的运行进程。Windows 上名为 TNSLSNR，Linux/Unix 平台上是 lsnrctl。监听器守候在服务器制定端口（默认为：1521），监听客户端的请求。
2. 为客户端请求分配 Server Process。监听器只负责接听请求，之后将请求转接给 Oracle Server Process。在 Oracle 的服务模式下，客户端进程是不允许直接操作数据库实例和数据，而是通过一个服务进程 Server Process（也称为影子进程）作为代理。监听器接受到请求之后，就向操作系统（或者 Dispatcher 组件）要求 fork（或分配）一个 Server Process 与客户端相连。
3. 注册实例服务。本质上讲，Listener 是建立实例和客户端进程之间联系的桥梁。Listener 与实例之间的联系，就是通过注册的过程来实现的。注册的过程就是实例告诉监听器，它的数据库数据库实例名称 instance\_name 和服务名 service\_names。监听器注册上这样的信息，对客户端请求根据监听注册信息，找到正确的服务实例名称。目前 Oracle 版本中，提供动态注册和静态注册两种方式。
4. 错误转移 failover。failover 是 RAC 容错的一个重要方面功能，其功能是在数据库实例崩溃的时候，可以自动将请求转移到其他可用实例上的一种功能。可以提供很大程度上的可用性（Availability）功能。这个过程中，发现实例已经崩溃，并且将请求转移到其他实例上，就属于是 Listener 的功能。
5. 负载均衡衡量。在 RAC 架构中，Oracle 实现了负载均衡。当一个客户请求到来时，Oracle 会根据当前 RAC 集群环境中所有实例的负载情况，避开负载较高的实例，将请求转移到负载较低的实例进行处理。在早期 RAC 版本中，负载轻重的衡量是根据监听器当前维护连接数目来确定的，而不是实时查看多实例的负载。RAC 环境中的监听器之间进行沟通通信。

### 2.监听器的操作

监听器在 Windows 和 Linux/Unix 平台上，都是可以直接操作的。下面以 Linux 平台操作为例，其他平台类似。

在命令行窗口（CMD），输入 lsnrctl，就可以进入监听器控制窗口。

[oracle@newtest ~]$ lsnrctl

LSNRCTL for Linux: Version 10.2.0.4.0 – Production on 06-APR-2011 14:31:46  
Copyright (c) 1991, 2007, Oracle.  All rights reserved.  
Welcome to LSNRCTL, type “help” for information.

LSNRCTL>

通过输入命令 help，可以查看支持的监听器操作。下面仅介绍常用的几个:

查看当前监听器状态：

敲入 status，可以查看当前监听器的状态，对应操作日志信息和服务注册信息等内容。如下：

LSNRCTL> status

Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=127.0.0.1)(PORT=1521)))

STATUS of the LISTENER  
————————  
Alias LISTENER  
Version TNSLSNR for Linux: Version 10.2.0.4.0 – Production  
Start Date 18-MAR-2011 18:03:56  
Uptime 18 days 20 hr. 28 min. 37 sec  
Trace Level off  
Security ON: Local OS Authentication  
SNMP OFF

Listener Parameter File /home/oracle/product/10.2.0/db\_1/network/admin/listener.ora

Listener Log File /home/oracle/product/10.2.0/db\_1/network/log/listener.log

Listening Endpoints Summary…

(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=127.0.0.1)(PORT=1521)))  
Services Summary…  
Service “Test” has 2 instance(s).  
Instance “Test”, status UNKNOWN, has 1 handler(s) for this service…  
Instance “Test”, status READY, has 1 handler(s) for this service…  
Service “TestXDB” has 1 instance(s).  
Instance “Test”, status READY, has 1 handler(s) for this service…  
Service “Test\_XPT” has 1 instance(s).  
Instance “Test”, status READY, has 1 handler(s) for this service…  
The command completed successfully

LSNRCTL>

需要额外注意的下面几项内容。

监听程序参数文件 /home/oracle/product/10.2.0/db\_1/network/admin/listener.ora  //使用的参数

监听器启动时依据参数文件的，当然没有参数文件监听器也是可以启动，那时监听器依据默认的行为操作。这个参数指定了监听器参数文件的位置，listener.ora。这个文件是一个文本类型参数文件，描述了监听器监听端口，主机名称和静态注册信息。Listener.ora是随着数据库实例的建立之后，建立监听器的过程中动态配置的。

监听程序日志文件 /home/oracle/product/10.2.0/db\_1/network/log/listener.log //操作日志位置

监听器日志，是一个记录和描述监听器工作和错误的信息库。通过仔细研究日志内容，可以帮助我们发现当前的监听器的问题、解决连接故障和深入理解监听器工作原理。

上文中已经说明，监听器有注册功能，通过 status 命令（services命令同样效果）。可以查看到当前有什么服务被注册上，用来检查连接服务失败，是常用的工具。

开启/关闭监听器：

监听器的行为受到参数文件的控制，我们有时候需要调整监听器的参数。调整监听参数的方法，可以通过 Oracle 提供的 GUI 界面完成，还可以通过手工修改 listener.ora 来完成。

因为参数文件 listener.ora 是一个文本文件。Oracle 对于文本类型的参数文件，大多数情况下是不支持热加载的。pfile 和listener.ora 都是如此。

使用界面GUI，就是使用 Net Configuration Assistant 来配置。配置完成后，配置程序会自动重新启动监听器程序，来加载修改的参数文件。如果采用手工修改 listener.ora，就必须要手工的进行监听程序关闭和启动。

LSNRCTL> stop  
Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=127.0.0.1)(PORT=1521)))  
The command completed successfully

LSNRCTL> start  
Starting /home/oracle/product/10.2.0/db\_1/bin/tnslsnr: please wait…  
TNSLSNR for Linux: Version 10.2.0.4.0 – Production  
System parameter file is /home/oracle/product/10.2.0/db\_1/network/admin/listener.ora  
Log messages written to /home/oracle/product/10.2.0/db\_1/network/log/listener.log  
Listening on: (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=127.0.0.1)(PORT=1521)))  
Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=127.0.0.1)(PORT=1521)))

STATUS of the LISTENER  
————————  
Alias LISTENER  
Version TNSLSNR for Linux: Version 10.2.0.4.0 – Production  
Start Date 06-APR-2011 14:35:27  
Uptime 0 days 0 hr. 0 min. 0 sec  
Trace Level off  
Security ON: Local OS Authentication  
SNMP OFF  
Listener Parameter File /home/oracle/product/10.2.0/db\_1/network/admin/listener.ora  
Listener Log File /home/oracle/product/10.2.0/db\_1/network/log/listener.log  
Listening Endpoints Summary…

(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=127.0.0.1)(PORT=1521)))  
Services Summary…  
Service “Test” has 1 instance(s).  
Instance “Test”, status UNKNOWN, has 1 handler(s) for this service…  
The command completed successfully

LSNRCTL>

这里多说一句关于监听器的参数配置。使用GUI可以满足大部分的情况，而且可以避免拼写错误引发的监听器故障（监听器不会检查配置项目的正确与否）。但是，在一些比较复杂的情况下，比如一台机器绑定多个IP的情况，或者 RAC 的复杂功能配置上，还是倾向于使用手工编写 listener.ora 的方法。

此外，lsnrctl 提示行还提供了 reload 命令，可以在 Listener 启动的情况下，重新加载文件和 SID 信息。但是我还是比较喜欢 stop+start 的组合。

如果是在 Windows 平台下，启动关闭监听器还可以在系统服务项目列表中操作，效果是一样的。

### 3.监听器的工作过程

下面我们谈谈监听器的工作过程。一般，监听器作为一个独立 process 在操作系统中运行，守候在特定网络端口（默认为：1521），等待客户端请求的到来。注意：我们在客户端配置命名服务的时候，输入的1521也就是为了与监听器程序建立连接。

当一个请求“如期而至”，监听器对照已经注册的服务列表，查找对应的数据库实例信息，获取到指定实例的 ORACLE\_HOME 路径。相当于表明可以进行连接。

客户端与实例的交互不是直接的，是通过 Server Process 作为代理中介来实现的。所有指令 SQL 都是客户端通过 Server Process 发送到实例中，这种体系结构是 Oracle 对于实例和数据库文件一种保护机制。

当监听器获得请求之后，要从 Oracle 实例中分配一个 Server Process 与之对应。这里不同的 Oracle 连接方式存在一些差别。

如果是专用连接模式，也就是一个客户端连接对应一个 Server Process。监听器就会向 OS 请求 fork（创造）出一个 Server Process，与监听器尝试交互。

如果是共享连接模式，也就是多个客户端共享一个 Server Process（注意：这里还不是连接池）。监听器就会向 Dispatcher 进程（管理共享模式连接的进程）请求一个 Server Process 与之交互。

Server Process 与监听器的连接，实际上就是相互信息的交换。Server Process 将自身在 OS 中的进程编号、连接地址信息发给监听器。监听器将客户端信息传递给 Server Process。

监听器获取到 Server Process 的信息之后，将其返回给客户端连接程序。客户端获取到信息之后，进行重连接，根据返回的信息与 Server Process 在制定的服务器端口进行联系。

直到这个时候，客户端程序才将连接用户名、密码等信息发给 Server Process，进行登录验证等操作。监听器的工作也就到此结束。

这里面有一个技术细节，就是 Server Process 与客户端连接的时候，是允许不使用1521端口的。具体连接的端口，是带有随机因素的。在9i版本 Windows 平台下，如果安装了防火墙，只允许1521端口通信，是会带来一些连接问题。好在在其他平台上和之后的版本中，实现了一种端口共享技术，连接可以和监听器一起使用1521端口。

### 4.监听器的动/静态注册机制

介绍了监听器工作原理，下面我们说说动静态注册的机制。在上面的内容中，我们已经初步了解了注册的作用，注册就是将数据库作为一个服务注册到监听程序。客户端不需要知道数据库名和实例名，只需要知道该数据库对外提供的服务名就可以申请连接到数据库。这个服务名可能与实例名一样，也有可能不一样。在数据库服务器启动过程中，数据库服务器会向监听程序注册相应的服务（无论何时启动一个数据库，默认地都有两条信息注册到监听器中：数据库服务器对应的实例和服务。）相当于是这样：在数据库服务器和客户端之间有一监听程序（Listener），在监听程序中，会记录相应数据库对应的服务名（一个数据库可能对应有多个服务名），当客户端需要连接数据库时，只需要提供服务名，就可以建立客户端和服务器之间的连接。目前的 Oracle，支持静态注册和动态注册两种注册方式。

静态注册顾名思义，就是显示的指定出监听器程序要为那个实例以哪个服务名做监听。在启动监听器的时候，监听程序是不知道所监听实例服务是否存在。直到有客户端请求指定的服务。

静态注册就是实例启动时读取 listener.ora 文件的配置，将实例和服务注册到监听程序。无论何时启动一个数据库，默认地都有两条信息注册到监听器中：数据库服务器对应的实例和服务。

静态注册时，listener.ora 文件中的 GLOBAL\_DBNAME 向外提供服务名，listener.ora 文件中的 SID\_NAME 提供注册的实例名。下面是一个典型参数文件的结构：

# listener.ora Network Configuration File: /home/oracle/product/10.2.0/db\_1/network/admin/listener.ora  
# Generated by Oracle configuration tools.

SID\_LIST\_LISTENER =  
(SID\_LIST =  
(SID\_DESC =  
(SID\_NAME = PLSExtProc)  
(ORACLE\_HOME = /home/oracle/product/10.2.0/db\_1)  
(PROGRAM = extproc)  
)

(SID\_DESC =  
(GLOBAL\_DBNAME = Test)  
(ORACLE\_HOME = /home/oracle/product/10.2.0/db\_1)  
(SID\_NAME = Test)  
)  
)

LISTENER =  
(DESCRIPTION\_LIST =  
(DESCRIPTION =  
(ADDRESS = (PROTOCOL = TCP)(HOST = 127.0.0.1)(PORT = 1521))  
)  
)

其中，SID\_LIST 配置节点中的 SID\_DESC 就是配置静态注册的节点项目。SID\_LIST 里用来配置当前监听器注册的服务项目，通过多个 SID\_DESC 进行配置，每个 SID\_DESC 基本上就是一个配置项目。默认是动态注册的时候，只有 PLSExtProc 项目。

默认安装时，会安装一个 PL/SQL 外部程序（ExtProc）条目在 listener.ora 中，是 Oracle 为调用外部程序默认配置的监听，它的名字通常是 ExtProc 或 PLSExtProc，但一般不会使用它，可以直接从 listener.ora 中将这项移除，因为对 ExtProc 已经有多种攻击手段了，在不使用外部程序时，Oracle 也是建议删除的。PLSExtProc 是 PL/SQL external procdure 的意思，就是在 PL/SQL 中调用外部语句，如 C,Java 写的过程。现在，Oracle 已经全面支持 Java 了，这东西也就过时了，之所以继续保留是考虑到兼容以前老版本的数据库实例。有时可能会在多个数据库实例之间拷贝 listener.ora，请检查拷贝来的文件中是否含有不需要的服务，确保只留下的确需要的服务项目，减少监听器受攻击的面。其实 Oracle DBA 应该特别重视监听器的安全性，因为黑客可以轻易的侵入没有进行安全配置监听器的 Oracle 数据库实例。如果你想了解 Oracle 监听器安全配置方面的知识。

静态配置项目中，通过 GLOBAL\_NAME 配置服务项目，通过 SID\_NAME 指定数据库实例的名称，通过 ORACLE\_HOME 配置 Oracle 数据库软件安装的基本目录。

动态注册是与静态注册相对应的一种注册方法。同样也是通过 listener.ora 进行配置。下面是一个典型的配置参数文件：

# listener.ora Network Configuration File: /home/oracle/product/10.2.0/db\_1/network/admin/listener.ora  
# Generated by Oracle configuration tools.

SID\_LIST\_LISTENER =  
(SID\_LIST =  
(SID\_DESC =  
(SID\_NAME = PLSExtProc)  
(ORACLE\_HOME = /home/oracle/product/10.2.0/db\_1)  
(PROGRAM = extproc)  
)  
)

LISTENER =  
(DESCRIPTION\_LIST =  
(DESCRIPTION =  
(ADDRESS = (PROTOCOL = TCP)(HOST = 127.0.0.1)(PORT = 1521))  
)  
)

对比上面的静态注册文件，可以发现显示指定服务名称和数据库实例名的项目不见了。只有本地地址和监听器端口信息。

动态注册的动作是监听器所在主机上数据库实例完成。动态注册是 Oracle 上一个不能关闭的功能。实例的 background process PMON，每隔一段时间（一到两分钟）就会将实例的参数信息注册到监听器上，实现动态注册。

注册的信息是数据库参数 service\_name 和 instance\_name。可以通过 Show Parameter 命令查看。

SQL> show parameter instance\_name;

NAME TYPE VALUE  
———————————— ———– ——————————  
instance\_name string Test

SQL> show parameter service\_names;

NAME TYPE VALUE  
———————————— ———– ——————————  
service\_names string Test

一般数据库在建立的时候，这两个参数都是已经设置好的，不会轻易的发生变化。

如果 instance\_name 参数没有设置，则系统会选取参数 db\_name 作为实例名称返回。如果 service\_names 没有设置，则会将 db\_name 和 db\_domain 两个参数组合成服务名实现注册。

值得注意的一个问题是，service\_names 参数是一个可以指定多个服务名称的参数，以逗号分隔。也就是说，一个 Oracle 实例时可以同时作为多个服务名向外提供服务的。service\_name 是 Oracle 在新近版本中提出，替代 SID\_NAME 的概念。使用 Service 的指定方式，多个 Oracle 实例就可以统一提供一致的数据访问服务（也就是 RAC）。

通常，对 instance\_name 和 service\_name 两个参数，还是建议设置上。因为会在一定程度上影响到动态注册的效果。

如果没有显示的指定这两个参数，那么只有在数据库实例启动在监听器实例之后的时候，才会一次动态注册。一旦之后监听器重新启动，动态注册的信息就不会存在，而且很难再次注册上。所以，我们是推荐将这两个参数设置上的。

只有显示设置两个值的情况下，PMON 才会周期性的将注册信息加以注册。通过命令行，也可以强迫 PMON 立即执行一次注册操作。

SQL> alter system register ;  
System altered

最后，我们聊一聊动态注册的作用，支持错误转移 failover。注册操作的发动方是 PMON 后台进程，PMON 是 Oracle 实例最重要的后台进程，很多资料和 DBA 都是以该进程的状态确定数据库实例的状态。PMON 负责动态注册，如果不能进行动态注册，说明 PMON 已经失去了工作能力，也就意味着监听程序所服务的实例已经不能工作。

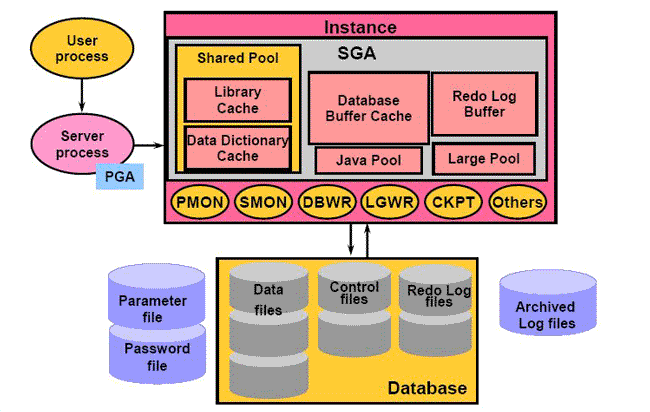
通过动态注册的机制，监听程序是可以知道所服务的服务器实例已经崩溃，对于客户端的请求，可以实现错误转移。

当然，我们这里谈到的崩溃是实例的崩溃。如果监听器程序崩溃了，情况又是另一种样子了。

我们可以使用命令 lsnrctl status 来查看某服务是静态注册还是动态注册。实例状态为 UNKNOWN 值时表明此服务是静态注册的设置。这时监听器用来表明它不知道关于该实例的任何信息，只有当客户发出连接请求时，它才检查该实例是否存在。动态注册的数据库通过状态信息中的状态 READY 或状态 BLOCKED（对于一个备用数据库）来指明。不管关闭何时数据库，动态注册的数据库都会动态地从 监听器注销，而与之相关的信息将从状态列表中消失。这样，不管数据库是在运行还是已经关闭，监听器总是知道它的状态。该信息将被用于连接请求的回退 （fallback）和负载平衡。

## Oracle数据库启动过程及状态详解（nomount、mount和open）

Oracle Server主要由**实例（instance）**和**数据库（database）**组成。**实例（instance）**由**共享内存（SGA）**和**后台进程系统**组成，**数据库（database）**是存储在磁盘上的一系列物理文件。

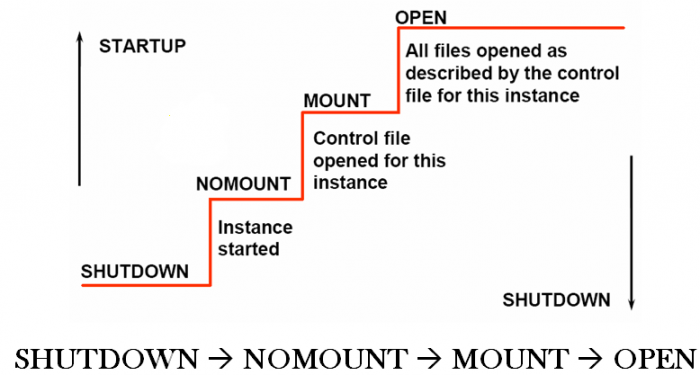


**SGA**主要由**Share Pool**（共享池，又分Library Cache和Data Dictionary Cache，前者临时存储最近执行过的语句代码等，后者临时存储数据位置、表定义及用户权限等）、**Database Buffer Cache**（数据缓冲区，临时存储读过的数据块）、**Redo Log Buffer**（重做日志缓冲区，临时存储数据库修改操作）、**Large Pool**（大池，分担Share Pool部分工作供共享服务器进程使用，如备份恢复、大型I/O操作、并行查询等）和**Java Pool**（Java池，分析Java语句）。

**后台进程**是数据库和操作系统进行交互的通道，后台进程的命名由ORACLE\_SID决定，ORACLE根据ORACLE\_SID来寻找参数文件启动实例。是Oracle数据库为保持最佳性能和协调多用户请求而设置的，主要有**DBWR**（将Database Buffer Cache数据写入Data Files）、**LGWR**（将Redo Log Buffer数据写入Redo Log Files）、**CKPT**（协调数据文件、控制文件和重做日志，将System Change Number即SCN写入到控制文件和数据文件头部，促使DBWR、LGWR执行）、**SMON**（System Monitor，3方面作用：instance recovery实例故障数据恢复、合并空闲碎片空间、回收临时段）、**PMON**（用户例程意外终止时处理事务，如回退事务、释放锁及其他资源等）、**ARCH**（将Redo Log Files写入Archive Log Files）、**CJQ0**（job queue coordinator）、**RVWR**（recover writer，为flashback database提供日志记录）等进程。

**数据库（database）**是指存储在磁盘上的一组物理文件，如**数据文件**（Data files，用于存储数据）、**控制文件**（Control files，存储数据文件、重做日志文件、归档日志文件位置及维护数据库完整性所需信息）和**重做日志文件**（Redo Log files，存储修改数据的所有操作记录以备故障后恢复），这三个是启动数据库必须的文件；另外还有**参数文件**（Parameter file，设置内存后台进程的启动等）、**归档日志文件**（Archived Log files，归档记录写满的重做日志文件的内容）和**口令文件**（Password file，验证用户名密码），这三个是非必须的文件。

Oracle启动分3个过程nomount、mount和open，这三个过程具体执行的工作如下：



**nomount状态**：启动实例。

Reading the initialization file from $ORACLE\_HOME/dbs in the following order:

从环境变量下dbs目录按如下顺序读取初始化文件：

-first  spfileSID.ora

首先，读取spfile+实例名.ora

-if not found then, spfile.ora

若未发现文件则读取spfile.ora

-if not found then, initSID.ora

若未发现文件则读取init+实例名.ora

Specifying the PFILE parameter with STARTUP overrides the default behavior.

指定pfile参数文件启动以替代默认启动方式

\*  Allocating the SGA

分配SGA

\*  Starting the background processes

启动后台进程

\*  Opening the alertSID.log file and the trace files

启动预警日志文件（记录实例生命周期内事件，如系统内部错误、数据块损坏、系统参数修改等）和追踪文件（记录SQL操作及时间消耗等）

The database must be named with the DB\_NAME parameter either in the initialization Parameter file or in the STARTUP command.

数据库必须用初始参数文件或启动命令中的DB\_NAME参数命名。

**mount状态**：关联实例与数据库，读取控制文件并获取数据文件和重做日志文件名称状态。

Mounting a database includes the following tasks:

装载数据库包括以下任务：

\* Associating a database with a previously started instance

将先前启动的实例与数据库相关联

\* Locating and opening the control files specified in the parameter file

从参数文件中找到控制文件位置并打开

\* Reading the control files to obtain the names and status of the data files and online redo log files.However,no checks are performed to verify the existence of the data files and online redo log files at this time.

从控制文件中读取数据文件及重做日志文件名称与状态，但是，此时并不检查数据文件与重做日志文件的存在性。

**open状态**：

opening the database includes the following tasks:

打开数据库包含以下任务：

opening the online data log files

打开数据文件

opening the onling redo log files

打开重做日志文件

If any of the datafiles or noline redo log files are not present when you attempt to open the database ,the oracle server returns an error.

若在打开数据库时数据文件或重做日志文件任何一个不存在，则Oracle服务器返回一个错误。

During this final stage,the oracle server verfies that all the data files and online redo log files can be opened and checks the consistency of the database. If necessary,the SMON background process initiates instance recovery.

在最后阶段，Oracle数据库验证数据文件和重做日志文件可否打开并检验数据库的一致性，若不一致，SMON后台进程将启动实例恢复。

### **shutdown的参数**

shutdown有四个参数：normal、transactional、immediate、abort，不带参数默认为normal。

#### shutdown normal：

不断开现在连接用户，阻止任何用户建立新的连接，包括管理员在内。已经连接的用户能够继续他们当前的工作，如递交新的更新事务，直到此用户自行断开连接。这样需要等待的时间长，可以查出现连用户，再通知其自行断开。所有的用户都断开连接，数据库才进行关闭操作，即关闭数据库、卸载数据库、终止例程。在这种情况下关闭的数据库在重新启动后，不会出现问题。启动时不需要实例恢复。

#### **shutdown transactional**：

阻止任何用户建立新连接，等待所有当前连接用户的未递交的活动事务提交完毕，然后立即断开用户的连接。所有的用户都断开连接则立即关闭数据库，进行关闭数据库、卸载数据库、终止进程等操作。这种方式，用户有可能正在算账，做复杂报表！一次数据库操作做不完的，在刚做了一次数据库操作后，将被断开，这样对用户有一定影响，启动时不需要实例恢复。

#### shutdown immediate：

阻止任何用户新的连接，同时限制当前连接用户开始新的事务。如果已连接用户有未完成的事务，则数据库系统不会等待他们完成，而是直接把当前未递交的事务回退。数据库系统不再等待用户主动断开连接，当未递交的事务回退成功后，系统会直接关闭、卸载数据库，并终止数据库进程，启动时不需要实例恢复。

#### **shutdown abort**：

当数据库出现故障时，可能以上三种方式都无法正常关闭数据库，则使用这种方法。强制结束当前正在执行的SQL语句，任何未递交的事务都不被回退！这种方法基本上不会对控制文件或者参数文件造成破坏，这比强制关机要好一点（在无法正常关闭数据库的时候），启动时自动进行实例恢复。

### startup参数

startup有7个参数：nomount、mount、open、pfile、force、restrict和

#### **startup nomount**：

通过参数文件，分配sga，启动数据库后台进程，不打开控制文件和数据文件，不能访问数据库。

#### **startup mount**：

仅给dba进行管理操作，不允许数据库用户访问。仅当前实例的控制文件被打开，数据文件未打开，在这个模式下可以进行如下操作：重命名数据文件、添加取消或重命名重做日志文件、设置归档模式、设置闪回、执行完整的数据库恢复操作等。

#### **startup open**：

startup的默认参数就是open，打开数据库，允许数据库的访问，当前实例控制文件中所描述的所有文件都已经打开。

#### **startup pfile**=FILENAME：

以FILENAME为初始化文件启动数据库，不是采用默认初始化文件。

#### **startup force**：

中止当前数据库的运行，并开始重新正常的启动数据库。

#### **startup restrict**：

只允许具有restricted session权限的用户访问数据库，该模式下登陆者可做如下操作：执行数据库数据的导出或导入、执行数据装载操作用SQL\*Loader、暂时阻止一般的用户使用数据、在某个移植过程和升级操作过程中restricted session登陆后可使用ALTER SYSTEM 语句来禁止RESTRICTED SESSION特性ALTER SYSTEM DISABLE RESTRICTED SESSION；如果是在非受限模式下打开的数据库，后来发现需要限制访问，此时可以使用带ENABLE RESTRICTED SESSION 子句的ALTER SYSTEM 语句来完成。

#### **startup recover**：

数据库启动，并开始介质恢复。

## linux下配置oracle远程连接数据库

下文为您介绍的oracle远程连接数据库方法实现的是在linux环境下，配置listener实现oracle远程连接数据库的方法

### 服务端：

1、增加一个listener：终端运行：$ORACLE\_HOME/bin/netca （必须在图形界面环境下），按照提示增加一个listener。

2、增加listener支持的服务：终端运行：$ORACLE\_HOME/bin/netmgr （图形界面下），在：Oracle Net Configuration -> Local -> Listeners -> xxxx (你刚才增加的那个listener)，从中选择“Database services”，增加你要服务的database。其中，注意Global Database Name就是你的对外的服务名字。然后保存并退出。

3、重启你的listener：终端运行：

$ORACLE\_HOME/bin/lsnrctl stop

$ORACLE\_HOME/bin/lsnrctl start

最后提示：

Service "XXXX" has 1 instance(s).

The command completed successfully.

好了，到现在服务端的listener配置按成了。

### 客户端：

1、配置一个“Local Service Name Configuration”：终端运行：$ORACLE\_HOME/bin/netca ，选择：Local Service Name Configuration，按照提示操作。其中，Service Name写和服务器端配置的时候添的那个“Global Database Name”；Host name写服务器IP（如果是在局域网内，也可些服务器名字）；最后给这个本地服务起一个名字（例如叫：LSN）。然后保存并退出。

2、通过sqlplus连接到服务器上的数据库：

格式：sqlplus 用户名/密码@本地服务的名字

例子：sqlplus user/password@LSN

## Oracle数据库解密函数decrypt\_new

select

decrypt\_new(user\_pwd)

from

表

## Oracle数据库加密函数encrypt\_new

select

encrypt\_new(user\_pwd)

from

表

## 查看oracle数据库的sid

查看SID

1.用sysdba身份登录 比如 conn /as  sysdba(或者sqlplus ‘/as sysdba’)

2.select instance\_name from v$instance;

oracle的sid在不同的系统环境下，查看方式不相同，方法分别如下：

1、windows 下查看注册表

开始 输入regedit 查看

HKEY\_LOCAL\_MACHINE\SOFTWARE\ORACLE\KEY\_OraDb11g\_home1\ORACLE\_SID就是

2、linux或者unix下可以

echo $ORACLE\_SID

注意：ORACLE\_SID要大写

3、在sql命令下可以查看

连接后查询：

SQL> select instance\_name from v$instance;

## 查看oracle数据库用户

查看用户名

select \* from dba\_users; --查看数据库里面所有用户，前提是你是有dba权限的帐号，如sys,system

select \* from all\_users;  --查看你能管理的所有用户！

select \* from user\_users; --查看当前用户信息 ！

## 查看数据库的版本

### 方法一：v$version

1. SQL> select \* from v$version;
3. BANNER
4. --------------------------------------------------------------------------------
5. Oracle Database 11g Enterprise Edition Release 11.1.0.7.0 - 64bit Production
6. PL/SQL Release 11.1.0.7.0 - Production
7. CORE    11.1.0.7.0      Production
8. TNS for Linux: Version 11.1.0.7.0 - Production
9. NLSRTL Version 11.1.0.7.0 - Production

### 方法二：product\_component\_version

1. SQL> select \* from product\_component\_version;
3. PRODUCT                                                 VERSION             STATUS
4. ---------------------------------------------------------------------------------------------
5. NLSRTL                                                  11.1.0.7.0           Production
6. Oracle Database 11g Enterprise Edition                  11.1.0.7.0           64bit Production
7. PL/SQL                                                  11.1.0.7.0           Production
8. TNS for Linux:                                          11.1.0.7.0           Production
10. SQL>

### 方法三：dbms\_output.put\_line( dbms\_db\_version.version )

1. SQL> SET SERVEROUTPUT ON
2. SQL> EXEC dbms\_output.put\_line( dbms\_db\_version.version );
3. 11
5. PL/SQL procedure successfully completed.
7. SQL>

## 查看dba用户

su – oracle

sqlplus /nolog

conn /as sysdba

select \* from dba\_users;

## oracle用户锁定与解锁

解锁用户，需要用具有dba权限的的用户登录数据库，执行如下语句：

|  |
| --- |
| Alter user user account unlock; |

将user替换成对应的用户名即可

登录失败次数配置参数，可以通过

|  |
| --- |
| Select username,profile from dba\_users; |

找到对应的用户所使用的profile文件，并通过

|  |
| --- |
| Select \* from dba\_profiles where resource\_name=’FAILED\_LOGIN\_ATTEMPS’ and profile=’DEFAULT’ |

语句,将DEFAULT为用户使用的profile文件进行查询，在查询结果即可看到登录失败允许次数设置

需要修正配置时，可通过语句

|  |
| --- |
| alter profile DEFAULT limit FAILED\_LOG\_ATTEMPTS num; |

将DEFAULT替换为用户对应的profile文件，并将num替换为设置的次数活unlimited 既可以

## oracle用户密码过期

修改密码，需要使用具有dba权限的用户登录数据库，执行如下语句

|  |
| --- |
| alter user user identified by password |

将user和password替换为期望的用户名和密码即可。

不允许重复的密码次数配置参数，可以通过

|  |
| --- |
| select \* from dba\_profile where resource\_name like ‘PASSWORD\_REUSE%’ and profile=’DEFAULT’; |

将DEFAULT替换为用户对应的profile查找相关参数设置，

PASSWORD\_REUSE\_TIME指定了密码不能重用的次数，PASSWORD\_REUSE\_MAX指定了密码可以重用至少修改过的次数。

如果希望查询密码过期的天数，可以通过

|  |
| --- |
| Select \* from dba\_profiles where resource\_name=’PASSWORD\_LIFE\_TIME’ and profile=’DEFAULT’; |

将DEFAULT替换为对应用户的profile查找可以得到密码过期天数

需要修改配置时，可以通过语句

|  |
| --- |
| alter profile DEFAULT limit PASSWORD\_LIFE\_TIME num |

将DEFAULT替换为用户对应的profile文件，并将num替换为设置的过期天数或者unlimited即可

## 归档日志清理

数据库无法登陆，报归档日志错误ORA-00257

原因是数据库归档日志已满

清理归档日志：

|  |
| --- |
| rman target 数据库名  crosscheck archivelog all;  delete force noprompt expired archivelog all;  delete force noprompt archivelog until time ‘sysdate-0.5’;  delete force noprompt archivelog all completed before ‘sysdate-0.5’ |

最后两条删除语句任选一种执行即可，0.5可以替换为任意想保留的天数，一般不建议清理所有的归档日志。

## Oracle重建stream

* 现象

数据同步异常，需要重建stream.

简单检测数据库同步方法如下：

|  |
| --- |
| -在线裤；状态非enable表示不正常  Select status from dba\_capture;  Select status from dba\_propagation;  -备份库：状态非enable表示不正常  Select status from dba\_apply;  -备份库：count(\*)大于等于1表示不正常  Select count(\*) from dba\_apply\_error; |

* 原因

资源不足，应用异常等。

* 解决

登录对应环境的在线库，执行如下命令：

|  |
| --- |
| cd /home/oracle/bin/rmbtodb/  rebuild StreamsRmbtodb |

根据环境数据库实例名的不同，路径可能略有差别

## Stream多次配置报错

* 现象

前次配置中断，接着重配stream时报错

* 原因

前一次配置过程中运行信息未清除，导致数据库认为尚有实例运行

* 解决

通过如下语句查询是否有未提交的事务：

|  |
| --- |
| sqlplus ‘/as sysdba’  select count(\*) from dba\_recoverable\_script |

如果结果不为0，则可执行如下语句删除残留记录：

|  |
| --- |
| Delete from dba\_recoverable\_script;  Commit; |

* 改进

建议每次配置stream前，预先检查该部分内容。

## Stream配置未提交事务阻塞

* 现象

Stream夯住

* 原因

存在未提交的事务，阻止了stream获取资源

* 解决

通过如下语句查询是否有未提交的事务：

|  |
| --- |
| select  gpro.inst\_id,  gse.sid,  gse.MACHINE,  gse.OSUSER,  gse.serial#,  gse.USERNAME,  gpro.spid,  gse.module,  gtr.start\_time  from  gv$process gpro,  gv$session gse,  gv$transaction gtr,  where  gpro.addr=gse.paddr  and  gse.taddr=gtr.addr  and  gtr.ubarec=0  and  gse.module not like ‘PL%’; |

检查stream是否夯住

|  |
| --- |
| select  aa.BLOCKING\_SESSION,  aa.\*  from  gv$session aa  where  aa.BLOCKING\_SESSION is not null; |

检查夯住stream事务

|  |
| --- |
| select  \*  from  gv$session  where  sid=\*\*; |

找到对应的事务后，重启相关的进程

## 表空间处理

查看表空间状态

|  |
| --- |
| -查看所有表空间当前大小及自动扩充的最大大小（M）  select  tablespace\_name,  bytes/1024/1024,  maxbytes/1024/1024  from  dba\_data\_files;  -查看剩余表空间当前大小  Select  Sum(bytes)/1024/1024,  tablespace\_name,  from  dba\_free\_space  group by  tablespace\_name; |

一般undo表空间命令为UNDOTSB+数字，如果把当前表空间已经扩展到最大且表空间剩余空间极小活已满，则需要扩展表空间

扩展表空间步骤：

首先用oracle用户登录后台服务器，执行如下命令查看Undo表文件位置

|  |
| --- |
| sqlplus ‘/as sysdba’  show parameter dump; |

一般UNDO表空间存在路径backgroud\_dump\_XX路径下，该路径下会有一些dbf文件，确认该目录下的磁盘空间足以支持必要的Undo文件扩充。

然后使用oracle用户登录数据库服务器，执行如下命令扩充Undo表空间

|  |
| --- |
| sqlplus ‘/as sysdba’  alter tablespace undotbs1 add datafile ‘oradata/rmbtbdb/UNDOTBS02.dbf’ size 1000m reuse autoextend on next 128m maxsize unlimited |

将undotbs1替换为UNDO表空间名称，将‘oradata/rmbtbdb/UNDOTBS02.dbf’替换为UNDO表空间路径和文件名，并确保路径正确及文件名与已有的不重复，执行命令即可扩展表空间

## 清理未提交事务

|  |
| --- |
| select  \*  from  v$session  where  username <> ‘SYS’;  select  \*  from  v$transaction;; |

通过上述命令找到链接的session，使用alter system kill session ‘sid.serial#’来清除相关的session即可清除该session上的未提交事务

## Oracle GoldenGate简介

[Oracle](http://www.linuxidc.com/topicnews.aspx?tid=12) Golden Gate软件是一种基于日志的结构化数据复制备份软件，它通过解析源 数据库在线日志或归档日志获得数据的增量变化，再将这些变化应用到目标数据库，从而实现源数据库与目标数据库同步。Oracle Golden Gate可以在异构的IT基础结构（包括几乎 所有常用操作系统平台和数据库平台）之间实现大量数据亚秒一级的实时复制,从而在可以 在应急系统、在线报表、实时数据仓库供应、交易跟踪、数据同步、集中/分发、容灾、数 据库升级和移植、双业务中心等多个场景下应用。同时，Oracle Golden Gate 可以实现一 对一、广播(一对多)、聚合(多对一)、双向、点对点、级联等多种灵活的拓扑结构。

RAC环境下管理OGG-HA <http://www.linuxidc.com/Linux/2012-07/65796.htm>

RAC环境下配置OGG同步 <http://www.linuxidc.com/Linux/2012-07/65794.htm>

GoldenGate单向表DML同步 <http://www.linuxidc.com/Linux/2013-04/82942.htm>

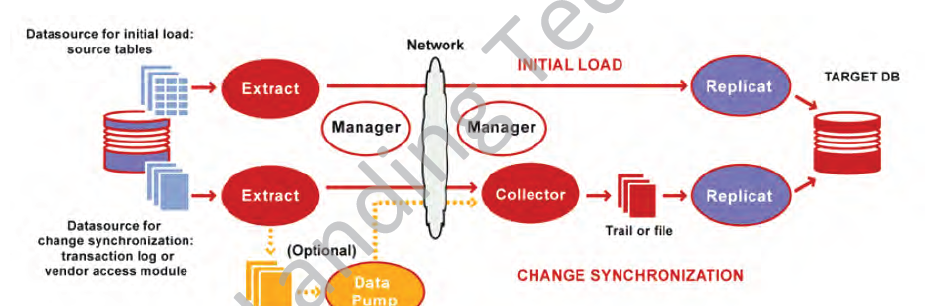
Oracle GoldenGate 系列：Extract 进程的恢复原理 <http://www.linuxidc.com/Linux/2013-04/82563.htm>

Oracle GoldenGate安装配置 <http://www.linuxidc.com/Linux/2013-02/79455.htm>

Oracle goldengate的OGG-01004 OGG-1296错误 <http://www.linuxidc.com/Linux/2011-08/40951.htm>

二、技术结构

和传统的逻辑复制一样，Oracle GoldenGate 实现原理是通过抽取源端 的redo log或者archive log，然后通过TCP/IP投递到目标端，最后解析还原应用到 目标端，使目标端实现同源端数同步。以下是OracleGoldenGate的技术架。



三、OGG进程

1、Manager进程

Manager进程是GoldenGate的控制进程，运行在源端和目标端上。它主要作用有以下几个方面：启动、监控、重启Goldengate的其他进程, 报告错误及事件，分配数据存储 空间，发布阀值报告等。

2、Extract进程

Extract运行在数据库源端，负责从源端数据表或者日志中捕获数据。

3、Pump进程

pump进程运行在数据库源端。

其作用是如果源端使用了本地的trail文件， 那么pump 进程就会把trail以数据块的形式通过TCP/IP协议发送到目标端，这通常也是推荐的 方式。pump进程本质是extract进程的一种特殊形式，如果不使用trails文件，那么 就是extract进程在抽取完数据以后，直接投递到目标端。

4.Trail文件

为了更有效、更安全的把数据库事务信息从源端投递到目标端。GoldenGate引进 trail文件的概念。前面提到extract抽取完数据以后Goldengate会将抽取的事务信息转化为一种GoldenGate专有格式的文件。然后pump负责把源端的trail文件投递到目标端， 所以源、目标两端都会存在这种文件。trail文件存在的目的旨在防止单点故障，将事务信 息持久化，并且使用checkpoint机制来记录其读写位置，如果故障发生，则数据可以根据 checkpoint记录的位置来重传。

5.Replicat进程

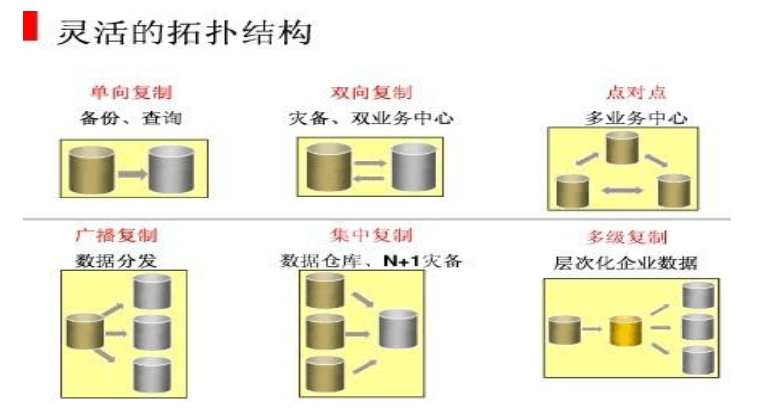
Replicat进程，通常我们也把它叫做应用进程。运行在目标端，是数据传递的最后一站，负责读取目标端trail文件中的内容，并将其解析为DML或DDL语句，然后应用到目标数据库中。

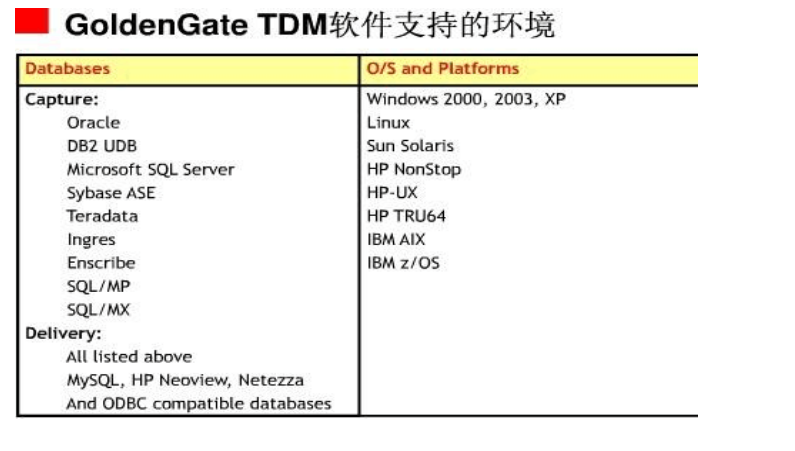
6.GGSCI

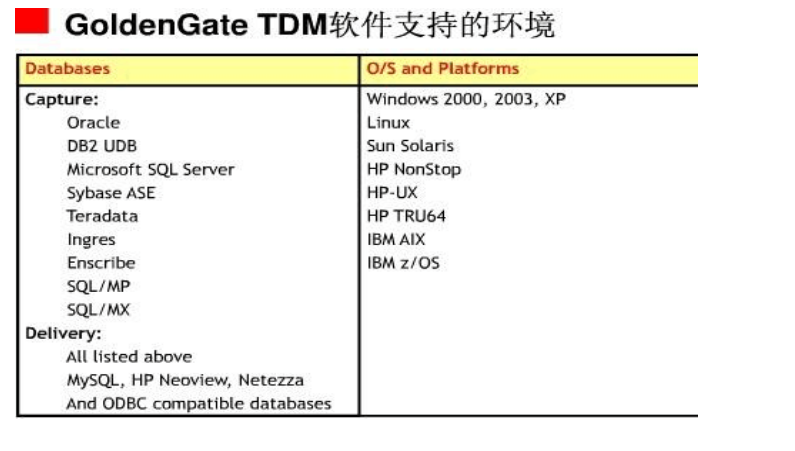
GGSCI是GoldenGate Software Command Interface 的缩写，

它提供了十分丰富的 命令来对Goldengate进行各种操作，如创建、修改、监控GoldenGate进程等等。

三、应用的拓展结构、支持平台和数据库







## Oracle RAC-实时应用集群

RAC，全称real application clusters，译为“实时应用集群”， 是Oracle新版数据库中采用的一项新技术，是高可用性的一种，也是[Oracle数据库](http://baike.baidu.com/view/1685727.htm)支持[网格计算](http://baike.baidu.com/view/20049.htm)环境的核心技术。

Oracle RAC主要支持Oracle9i、10g、11g版本，可以支持24 x 7 有效的[数据库应用系统](http://baike.baidu.com/view/1353688.htm)，在低成本[服务器](http://baike.baidu.com/view/899.htm)上构建高可用性[数据库系统](http://baike.baidu.com/view/7809.htm)，并且自由部署应用，无需修改代码。在Oracle RAC环境下，Oracle集成提供了[集群](http://baike.baidu.com/view/302477.htm)软件和存储管理软件，为用户降低了应用成本。当应用规模需要扩充时，用户可以按需扩展系统，以保证系统的性能。

(1)多[节点](http://baike.baidu.com/view/47398.htm)负载均衡;

(2)提供高可用：故障容错和无缝切换功能，将硬件和[软件错误](http://baike.baidu.com/view/1536002.htm)造成的影响最小化;

(3)通过[并行执行](http://baike.baidu.com/view/160033.htm)技术提高[事务](http://baike.baidu.com/view/121511.htm)响应时间----通常用于数据分析系统;

(4)通过横向扩展提高每秒交易数和连接数----通常对于联机事务系统;

(5)节约硬件成本，可以用多个廉价PC服务器代替昂贵的[小型机](http://baike.baidu.com/view/81485.htm)或大型机，同时节约相应维护成本;

(6)可扩展性好，可以方便添加删除节点，扩展硬件资源。

## Oracle物化视图

### ****物化视图（material view）是什么？****

物化视图是包括一个查询结果的数据库对象，它是远程数据的的本地副本，或者用来生成基于数据表求和的汇总表。

物化视图存储基于远程表的数据，也可以称为快照（类似于MSSQL Server中的snapshot，静态快照）。对于复制，物化视图允许你在本地维护远程数据的副本，这些副本是只读的。

如果你想修改本地副本，必须用高级复制的功能。当你想从一个表或视图中抽取数据时，你可以用从物化视图中抽取。

对于数据仓库，创建的物化视图通常情况下是聚合视图，单一表聚合视图和连接视图。（这个是基于本地的基表或者视图的聚合）。

物化视图，说白了，就是物理表，只不过这张表通过[Oracle](http://www.linuxidc.com/topicnews.aspx?tid=12)的内部机制可以定期更新，将一些大的耗时的表连接用物化视图实现，会提高查询的效率。当然要打开查询重写选项；

### **Material View的主要作用**

1. 实现两个数据库之间的数据同步，可以存在时间差。

2. 如果是远程链接数据库的场景时，提高查询速度。

（由于查询逻辑复杂，数据量比较大，导致每次查询视图的时候，查询速度慢，效率低下）

### ****物化视图的刷新方式和方法****

1. 刷新的方式

Fast

Complete

Fource

2. 刷新的方法

DBMS\_REFRESH.Refresh

DBMS\_MVIEW.Refresh

EXEC DBMS\_MVIEW.refresh('BXJ\_OBJECTS\_MV\_T1'，'C');

EXEC DBMS\_REFRESH.refresh('REP\_MVIEWGROUP');  
 **物化视图的刷新方式和方法**  
 (1).在源数据库建立mview log日志文件

        create materialized view log on w\_1 ;

----注：(TEST为表名或者视图名，关于视图上建立物化视图，见基于视图的物化视图

----创建物化视图语句：

(2).在统计数据建立materializad view  语法

Create materialized view MV\_TEST

----MVTEST为物化视图名

Build immediate

----创建时生成数据对应的是build deferred

Refresh fast

----增量刷新

On commit

----在基表有更新时提交，这里该句对视图无效

With rowid

----这里创建基于rowid的物化视图，对应的是 primary key

As

Select \* from TEST;

----生成物化视图数据语句

或者

　　　　CREATE MATERIALIZED VIEW MV\_TableName

BUILD IMMEDIATE --创建时立即刷新

REFRESH FORCE --如果可以快速刷新则进行快速刷新，否则完全刷新

ON DEMAND --刷新方式

START WITH SYSDATE --第一次刷新时间

NEXT SYSDATE+1/12 --刷新时间间隔

AS SELECT 1 id ‘A’ name FROM dual;

    (3).调用时进行刷新

        dbms\_refresh.refresh('W\_1')

### 创建物化视图

CREATE MATERIALIZED VIEW bxj\_objects\_mv\_t1 REFRESH FAST AS SELECT \* FROM apps.bxj\_objects\_t1;

CREATE MATERIALIZED VIEW bxj\_objects\_mv\_t2 REFRESH FORCE AS SELECT \* FROM bxj\_objects\_t2;

CREATE MATERIALIZED VIEW bxj\_objects\_mv\_t3 REFRESH COMPLETE AS SELECT \* FROM bxj\_objects\_t3;

### ON DEMAND和ON COMMIT物化视图的区别

ON DEMAND和ON COMMIT物化视图的区别在于其刷新方法的不同，ON DEMAND顾名思义，仅在该物化视图“**需要**”被刷新了，才进行刷新(REFRESH)，即更新物化视图，以保证和基表数据的一致性;而ON COMMIT是说，一旦基表有了COMMIT，即事务提交，则立刻刷新，立刻更新物化视图，使得数据和基表一致。  
  
**物化视图是物理表的么？**  
1) **物化视图在某种意义上说就是一个物理表(但并不仅仅是一个物理表)，这通过其可以被user\_tables查询出来，而得到佐证**;

2) **物化视图也是一种段(segment)，所以其有自己的物理存储属性**;

3) **物化视图会占用数据库磁盘空间，这点从user\_segment的查询结果，可以得到佐证**。

**查询物化视图上次刷新时间**

SELECT last\_refresh\_date

FROM user\_mviews

WHERE mview\_name = 'MV\_TABLENAME';

## [ORACLE自动断开数据库连接解决办法](http://www.cnblogs.com/Csharpblogs/archive/2011/08/18/2145074.html)

方法一、直接修改资源配置文件   
    分三个步骤在sqlplus环境下完成。   
    第一步，查询资源文件，找到CONNECT\_TIME所在的profile名。

|  |
| --- |
| select resource\_name,profile from dba\_profiles; |

    第二步，用alter命令修改profile中的值；   
        alter profile MONITORING\_PROFILE limit connect\_time unlimited;（或跟一个时间值，如1000，单位为分钟）   
        alter profile MONITORING\_PROFILE limit idle\_time unlimited;   
    第三步，用如下命令查看profile中的值；   
        select resource\_name,limit from dba\_profiles where profile='MONITORING\_PROFILE';   
  
    方法二：通过创建资源配置文件并进行用户分配来完成   
    分三个步骤在sqlplus环境下完成。[该方法不对其它数据库连接用户产生影响]   
    第一步，查看并修改resource\_limit的状态。   
    SQL>show   parameter   resource\_limit;   如果为FALSE，则将其修改为TRUE，否则配置不会生效：    
    SQL>alter system set resource\_file=true;              
    第二步，用create profile命令创建一个新的profile文件：    
    SQL>create profile [profile文件名] limit connect\_time unlimited idle\_time unlimited;    
    [示例：create profile test\_profile limit connect\_time unlimited idle\_time unlimited;]    
    第三步，将这个新的配置文件分配给该用户：    
    SQL>alter   user   [user\_name，如ROOT\_QY]   profile  [profile文件名];   
    [示例：alter user root\_qy profile test\_profile;]

## load data用法

load data

infile ‘ 文件名’

append into table 表名

fields terminated by ‘分隔符’

(

crt\_date date “YYYY-MM-DD”,

字段名,

)

## oracle db- link

二个oracle instance，如果需要在一个instance上，直接查询另一个instance上的数据，就要用到db-link

在创建database link的时候，Oracle在数据字典中保存相关的database link的信息，在使用database link的时候，Oracle通过Oracle Net用户预先定义好的连接信息访问相应的远程数据库以完成相应的工作。

建立database link之前需要确认的事项：  
确认从local database到remote database的网络连接是正常的，tnsping要能成功。  
确认在remote database上面有相应的访问权限。

### database link分类

类型Owner描述

|  |
| --- |
|  |
| Private | 创建database link的user拥有该database link | 在本地数据库的特定的schema下建立的database link。只有建立该database link的schema的session能使用这个database link来访问远程的数据库。同时也只有Owner能删除它自己的private database link。 |
| Public | Owner是PUBLIC. | Public的database link是数据库级的，本地数据库中所有的拥有数据库访问权限的用户或pl/sql程序都能使用此database link来访问相应的远程数据库。 |
| Global | Owner是PUBLIC. | Global的database link是网络级的，When an Oracle network uses a directory server, the directory server automatically create and manages global database links (as net service names) for every Oracle Database in the network. Users and PL/SQL subprograms in any database can use a global link to access objects in the corresponding remote database.  Note: In earlier releases of Oracle Database, a global database link referred to a database link that was registered with an Oracle Names server. The use of an Oracle Names server has been deprecated. In this document, global database links refer to the use of net service names from the directory server. |

### 创建dblink所需的权限

Privilege Database Required For

|  |
| --- |
|  |
| CREATE DATABASE LINK | Local | Creation of a private database link. |
| CREATE PUBLIC DATABASE LINK | Local | Creation of a public database link. |
| CREATE SESSION | Remote | Creation of any type of database link. |

### 创建db-link

基本语法  
CREATE [SHARED][PUBLIC] database link link\_name

      [CONNECT TO [user][current\_user] IDENTIFIED BY password]  
      [AUTHENTICATED BY user IDENTIFIED BY password]  
      [USING 'connect\_string']

说明：  
1) 权限：创建数据库链接的帐号必须有CREATE DATABASE LINK或CREATE PUBLIC DATABASE LINK的系统权限，用来登录到远程数据库的帐号必须有CREATE SESSION权限。这两种权限都包含在CONNECT角色中（CREATE PUBLIC DATABASE LINK权限在DBA中）。一个公用数据库链接对于数据库中的所有用户都是可用的，而一个私有链接仅对创建它的用户可用。由一个用户给另外一个用户授权私 有数据库链接是不可能的，一个数据库链接要么是公用的，要么是私有的。  
2)link :  当source端的数据库GLOBAL\_NAME=TRUE时，link名必须与远程数据库的全局数据库名global\_name）相同；否则，可以任意命名。

3)current\_user使用该选项是为了创建global类型的dblink。在分布式体系中存在多个数据库的话。如果想要在每一个数据库中都可以使用同样的名字来访问数据库a，那在每个数据库中都要创建一个到数据库a的db\_link，太麻烦了。所以现在有这个选项。你只要创建一次。所有的数据库都可以使用这个db\_link来访问了。要使用这个特性，必须有oracle nameserver或者ORACLE目录服务器。并且数据库a的参数global\_names=true.具体我也没有创建过，没有这个环境。  
4)connectstring：连接字符串，tnsnames.ora中定义远程数据库的连接串，也可以在创建dblink的时候直接指定。  
5)username、password：远程数据库的用户名，口令。如果不指定，则使用当前的用户名和口令登录到远程数据库，当创建connected user类型的dblink时，需要如果采用数据字典验证，则需要两边数据库的用户名密码一致。  
创建database link选项说明

取值说明

|  |
| --- |
|  |
| 不指定 | 默认值建立一个dedicated的连接，每一个使用database link的本地session都会对应有一个远程数据库的session。 |
| SHARED | 创建一个共享的数据库连接，同时要指定database link\_authentication。 使用shared方式的 database link是数据库会限制到远程数据库的连接的数量，这样以避免过多的连接对远程数据库造成太大的压力。在使用shared database link的时候，到database link的连接会在连接以后与本地连接断开，为防止未授权的session使用此链接而要求在创建shared database link的时候必须要指定database link\_authentication。 (from Oracle document: A shared database link is a link between a local server process and the remote database. The link is shared because multiple client processes can use the same link simultaneously.) |

共享链接更多资料  
共享数据库链接是指该链接的多个用户可以共享同一个底层网络连接。例如，在有四位用户的MTS（多线程服务器）环境下，每一个共享服务器进程都将与远程服务器有一个物理链接，这四位用户共享这两个链接。  
表面上，共享链接乍一听起来像是一件好事。在某些环境下的确如此，但是，当你考虑使用共享链接时，应当意识到这有许多局限性和警告：  
如果你使用一个专用的服务器连接来连接到你的本地数据库，链接只能在你从那些连接中创建的多重会话间共享 。在MTS环境里，每一个共享服务器进程潜在地打开一个链接。所有的会话被同一共享服务器进程提供并且分享被那个进程打开的任意共享链接。因为在MTS环境里的一个共享服务器进程能够服务于许多用户连接，共享链接的使用可能导致打开的链接远多于所必须的链接。用SHARED关键字建立共享数据库链接。还必须使用AUTHENTICATED BY 子句在远程系统上指定一有效的用户名和口令。如下命令建立一个共享的、公用的、连接用户数据库链接：  
    CREATE SHARED PUBLIC database link GNIS  
    AUTHENTICATED BY DUMMY\_USER IDENTIFIED BY SECRET  
    USING ‘GNIS’;

要获得创建链接和管理分布式系统的更多资料，请查阅Oracle Technology Network (http://otn.oracle.com/)。  
使用AUTHENTICATED BY子句稍微有些困扰，但是由于实现共享链接的方式安全性决定它是必须的。这个例子中的用户名和口令DUMMY\_USER/SECRET必须在远程系统上有效。然而，远程系统上使用的帐户仍就是连接用户的帐户。如果我以JEFF/SECRET登陆到我的本地数据库并使用我刚建好的共享链接，将会发生以下一系列事件：  
为了打开链接，Oracle使用DUMMY\_USER/SECRET向远程数据库授权。然后，Oracle试图使用HMAD/SECRET使我登陆到远程数据库。共享链接的主要目的是减少两个数据库服务器之间的底层网络连接数量。它们最适合于MTS环境，在那你拥有大量的通过这一链接访问远程数据库的用户。观念上，你想让用户数量超过共享服务器进程的数量。那么你可以通过为每一共享服务器进程打开一个链接而不是每位用户打开一个链接的方法，节省资源。

**database link使用方式**

取值说明

|  |
| --- |
|  |
| 不指定 | 默认值建立一个private的database link |
| PUBLIC | 公共连接，这样的连接可以被数据的所有的用户访问 |

**database link用户验证方法**

取值说明

|  |
| --- |
|  |
| 不指定 | 默认值采取Connected User的验证方法 |
| CONNECT TO CURRENT\_USER | 采取CURRENT\_USER的验证方式 |
| CONNECT TO user\_name IDENTIFIED BY password | 采取Fiexed User的验证方式 |

### database link创建举例

SQL StatementConnects To DatabaseConnects AsLink Type

|  |
| --- |
|  |
| CREATE database link sales.us.americas.acme\_auto.com USING ‘sales\_us’; | sales using net service name sales\_us | Connected user | Private connected user |
| CREATE database link foo  CONNECT TO CURRENT\_USER USING  ‘am\_sls’; | sales using service name am\_sls | Current global user | Private current user |
| CREATE database link sales.us.americas.acme\_auto.com CONNECT TO scott IDENTIFIED BY tiger USING ‘sales\_us’; | sales using net service name sales\_us | scott using password tiger | Private fixed user |
| CREATE PUBLIC database link sales CONNECT TO scott IDENTIFIED BY tiger USING ‘rev’; | sales using net service name rev | scott using password tiger | Public fixed user |
| CREATE SHARED PUBLIC database link sales.us.americas.acme\_auto.com  CONNECT TO scott IDENTIFIED BY tiger AUTHENTICATED BY anupam IDENTIFIED BY bhide USING ‘sales’; | sales using net service name sales | scott using password tiger, authenticated as anupam using password bhide | Shared public fixed user |

不使用TNS Name一例：  
CREATE database link link\_name  
CONNECT TO user IDENTIFIED BY screct  
USING '(DESCRIPTION =  
    (ADDRESS\_LIST =  
        (ADDRESS = (PROTOCOL = TCP)(HOST = sales.company.com)(PORT = 1521))  
    )  
    (CONNECT\_DATA =  
        (SERVICE\_NAME = sales)  
    )  
)';

例：

create public database link 链接名

connect to 用户名 identified by 密码

using '(DESCRIPTION =

(ADDRESS = (PROTOCOL = TCP)(HOST = 服务器IP)(PORT = 1521))

(CONNECT\_DATA =

(SERVER = DEDICATED)

(SERVICE\_NAME = orcltest)

)

)';

这样就创建了一个db-link，上面的“链接名”、“用户名”、"密码"等参数，大家自己调整一下（注：用户名/密码指连接到另一台instance上的连接用户名/密码）

### 查看系统的db-links

select \* from dba\_db\_links;

或者

select owner,object\_name from dba\_objects where object\_type='DATABASE LINK';

### 使用db-link

select \* from 表名@链接名;

如果觉得"@链接名"'太麻烦，可以用"同义词"简化操作

-- 最简单的用法

SELECT \* FROM table\_name@database link;   
-- 不想让使用的人知道database link的名字的时候  
-- 建一个别名包装一下   
CREATE SYNONYM table\_name FOR table\_name@database link;  
SELECT \* FROM table\_name;   
-- 或者，也可以建立一个视图来封装  
CREATE VIEW table\_name AS SELECT \* FROM  table\_name@database link;

### 删除database link

-- 删除public类型的database link

DROP PUBLIC database link link\_name;

-- 删除非public类型的database link

-- 注意：只有owner自己能删除自己的非public类型database link

DROP database link link\_name;

### 查看database link的信息

查看系统database link的基本信息  
DBA\_DB\_LINKS (ALL\_DB\_LINKS/USER\_DB\_LINKS)  
DBA\_DB\_LINKS视图为每一定义的链接返回一行。OWNER 列和DB\_LINK列分别显示了这一链接的所有者及名称。对公用数据库链接，OWNER列将包含’PUBLIC’。如果你建立固定用户链接，用户名应在DBA\_DB\_LINKS视图的USERNAME列里。ALL\_DB\_LINKS 视图和 USER\_DB\_LINKS视图与 DBA\_DB\_LINKS视图相类似-它们分别显示了你能够访问的所有链接及你所拥有的全部链接。   
COL OWNER FOR A15  
COL DB\_LINK FOR A25  
COL HOST FOR A25  
COL USERNAME FOR A15  
SELECT \* FROM DBA\_DB\_LINKS;   
DBA\_OBJECTS (ALL\_OBJECTS/USER\_OBJECTS)  
在这个视图里面只能查询到系统有那些database link以及他们的owner，创建时间等信息。  
COL OWNER FOR A15  
COL OBJECT\_NAME FOR A25  
COL OBJECT\_TYPE FOR A25  
SELECT OWNER,OBJECT\_NAME,OBJECT\_TYPE FROM DBA\_OBJECTS WHERE OBJECT\_TYPE LIKE 'database link';   
查看保存的Fixed user database link所保存的用户密码,该密码是经过加密的。  
这是比较危险的一件事，有必要对表sys.link$的权限进行限制。  
col host for a15  
col userid for a15  
col passwordx for a40  
col name for a20  
select name, host, userid, passwordx from sys.link$;

### 通过database link去SELECT远程数据库的一个表也是一个事务吗？

select \* from v$transaction;   
 -- 没有记录，说明没有事务

 -- 通过database link连接远程数据库，select 其中一个表  
  select \* from bbs\_news@mylink   
  select \* from v$transaction;   
 -- 发现有一条记录。  
解释  
    因为本地数据库只是将对应的sql发送给远程数据库执行，接受remote db返回的结果，但他并不知道是否该sql修改了数据； 所以需要为select 语句也标示一个事务。  
    具体可以参考otn 分布式数据库手册，所以在用database link远程访问时,要加上set transaction read only;   
close database link

ALTER SESSION CLOSE database link sales;

其实，dblink的相应属性对应了Oracle的数据字典link$,任何针对dblink的操作都是操作该数据字典。在9i的时候，如果Oracle的global\_name仅包括db\_name，也就是说DB\_DOMAIN的值为空。那么这个时候建立的数据库链，在数据库修改全局名GLOBAL\_NAME之后（修改为db\_name.db\_domain格式），会无法删除。  
如果要产生数据库链，必须将GLOBAL\_NAME改回DB\_NAME格式，即去掉后面的DOMAIN，但是这个时候，RENAME操作会自动添加域名，使得Oracle全局名无法恢复到初始状态。因此在这情况下，如果需要删除dblink，只能直接操作link$数据字典

delete from link$ where owner#=user\_id and name=dblink\_name

当然直接操作数据字典是危险的，最好做好备份，然后再进行操作。

利用dblink执行ddl

我们知道任何ddl语句都无法在dblink中直接执行，示例如下

SQL> desc db\_test;  
 Name                                      Null?    Type  
 ----------------------------------------- -------- ----------------------------  
 ID                                                 NUMBER(38)

SQL> drop table db\_test@DBLINK\_CONNECTED\_HR;   
drop table db\_test@DBLINK\_CONNECTED\_HR  
                   \*  
ERROR at line 1:  
ORA-02021: DDL operations are not allowed on a remote database

通过创建存储过程，使得能在dblink中执行ddl语句。 注意需在目标数据库的相应用户下创建存储过程 ，具体如下

SQL> exec dbms\_utility.EXEC\_DDL\_STATEMENT@dblink('drop table db\_test')  
PL/SQL procedure successfully completed.

SQL> desc db\_test;  
ERROR:  
ORA-04043: object db\_test does not exist

dblink的限制  
You cannot perform the following operations using database links:  
Grant privileges on remote objects  
Execute DESCRIBE operations on some remote objects. The following remote objects, however, do support DESCRIBE operations:  
        Tables  
        Views  
        Procedures  
        Functions  
Analyze remote objects  
Define or enforce referential integrity  
Grant roles to users in a remote database  
Obtain nondefault roles on a remote database. For example, if jane connects to the local database and executes a stored procedure that uses a fixed user link connecting as scott, jane receives scott's default roles on the remote database. Jane cannot issue SET ROLE to obtain a nondefault role.  
Execute hash query joins that use shared server connections  
Use a current user link without authentication through SSL, password, or NT native authentication

## decode函数详解

**decode**(条件,值1,返回值1,值2,返回值2,...值n,返回值n,缺省值)

该函数的含义如下：  
IF 条件=值1 THEN  
　　　　RETURN(翻译值1)  
ELSIF 条件=值2 THEN  
　　　　RETURN(翻译值2)  
　　　　......  
ELSIF 条件=值n THEN  
　　　　RETURN(翻译值n)  
ELSE  
　　　　RETURN(缺省值)  
END IF  
**decode**(字段或字段的运算，值1，值2，值3）

       这个函数运行的结果是，当字段或字段的运算的值等于值1时，该函数返回值2，否则返回值3  
当然值1，值2，值3也可以是表达式，这个函数使得某些sql语句简单了许多

使用方法：  
**1、比较大小**select **decode**(sign(变量1-变量2),-1,变量1,变量2) from dual; --取较小值  
sign()函数根据某个值是0、正数还是负数，分别返回0、1、-1  
例如：  
变量1=10，变量2=20  
则sign(变量1-变量2)返回-1，**decode**解码结果为“变量1”，达到了取较小值的目的。

**2、此函数用在SQL语句中，功能介绍如下：**

Decode函数与一系列嵌套的 IF-THEN-ELSE语句相似。base\_exp与compare1,compare2等等依次进行比较。如果base\_exp和 第i 个compare项匹配，就返回第i 个对应的value 。如果base\_exp与任何的compare值都不匹配，则返回default。每个compare值顺次求值，如果发现一个匹配，则剩下的compare值（如果还有的话）就都不再求值。一个为NULL的base\_exp被认为和NULL compare值等价。如果需要的话，每一个compare值都被转换成和第一个compare 值相同的数据类型，这个数据类型也是返回值的类型。

Decode函数在实际开发中非常的有用

**结合Lpad函数，如何使主键的值自动加1并在前面补0**select LPAD(decode(count(记录编号),0,1,max(to\_number(记录编号)+1)),14,'0') 记录编号 from tetdmis

eg:

select decode(dir,1,0,1) from a1\_interval

dir 的值是1变为0，是0则变为1

比如我要查询某班男生和女生的数量分别是多少?

通常我们这么写:

select count(\*) from 表 where 性别 ＝ 男；

select count(\*) from 表 where 性别 ＝ 女；

要想显示到一起还要union一下，太麻烦了

用decode呢，只需要一句话

select decode(性别，男，1，0），decode(性别，女，1，0） from 表

3、order by对字符列进行特定的排序

大家还可以在Order by中使用Decode。

例：表table\_subject，有subject\_name列。要求按照：语、数、外的顺序进行排序。这时，就可以非常轻松的使用Decode完成要求了。

select \* from table\_subject order by decode(subject\_name, '语文', 1, '数学', 2, , '外语',3)

## [lpad 函数](http://blog.csdn.net/bjnihao/article/details/6250417)

　lpad函数从左边对字符串使用指定的字符进行填充。从其字面意思也可以理解，l是left的简写，pad是填充的意思，所以lpad就是从左边填充的意思。

[编辑本段](http://baike.baidu.com/view/3058928.htm)语法

**语法格式如下：**

　　lpad( string, padded\_length, [ pad\_string ] )

**string**

　　准备被填充的字符串；

**padded\_length**

　　填充之后的字符串长度，也就是该函数返回的字符串长度，如果这个数量比原字符串的长度要短，lpad函数将会把字符串截取成从左到右的n个字符;

**pad\_string**

　　填充字符串，是个可选参数，这个字符串是要粘贴到string的左边，如果这个参数未写，lpad函数将会在string的左边粘贴空格。

**示例1：**

　　SQL> select lpad('abcde',10,'x') from dual;

　　LPAD('ABCDE',10,'X')

　　--------------------

　　xxxxxabcde

**示例2：**

　　SQL> select lpad('abcde',10,'oq') from dual;

　　LPAD('ABCDE',10,'OQ')

　　---------------------

　　oqoqoabcde

**示例3：**

　　SQL> select lpad('abcde',2) from dual;

　　LPAD('ABCDE',2)

　　---------------

　　ab

　　~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

　　与lpad函数对应的是rpad函数：

　　rpad函数从右边对字符串使用指定的字符进行填充，语法格式与lpad格式相同：

　　rpad(string,padded\_length,[pad\_string])

　　string

　　被填充的字符串

　　padded\_length

　　字符的长度，是返回的字符串的数量，如果这个数量比原字符串的长度要短，rpad函数将会把字符串截取成从左到右的n个字符;

　　pad\_string

　　是个可选参数，这个字符串是要粘贴到string的右边，如果这个参数未写，lpad函数将会在string的右边粘贴空格。

　　例如：

　　rpad('tech', 7); 将返回' tech'

　　rpad('tech', 2); 将返回'te'

　　rpad('tech', 8, '0'); 将返回'tech0000'

　　rpad('tech on the net', 15, 'z'); 将返回 'tech on the net'

　　rpad('tech on the net', 16, 'z'); 将返回 'tech on the netz'

## nvl函数

NVL函数是一个空值转换函数  
  
NVL（表达式1，表达式2）  
  
如果表达式1为空值，NVL返回值为表达式2的值，否则返回表达式1的值。该函数的目的是把一个空值（null）转换成一个实际的值。其表达式的值可以是数字型、字符型和日期型。但是表达式1和表达式2的数据类型必须为同一个类型。  
  
对数字型： NVL（ comm,0);  
  
对字符型 NVL( TO\_CHAR(comm), 'No Commission')  
  
对日期型 NVL（hiredate,' 31-DEC-99')

实用例子：

查询某个员工年薪，如果为空则用0代替

select (sal+nvl(comm,o))\*12 from emp where emp.pno=:pno;

这里，comm是未定义的变量，代指数字类型。 nvl返回为0.

nvl2函数  
  
 NVL2(表达式1，表达式2，表达式3）  
  
如果表达式1为空，返回值为表达式3的值。如果表达式1不为空，返回值为表达式2的值。  
  
例如 NVL2（comm，'sal+comm',sal)  
  
NVL2函数[**测试**](http://lib.csdn.net/base/softwaretest)comm  
  
如果comm为空，就返回sal 的值。如果 comm 不为空(null),就返回表达式 sal+comm的值。

## oracle-raw数据类型数据插入

exec sql begin decare section

char caDummyMesg[4000]

exec sql end decare section

exec sql var caDummyMesg is RAW（4000+1）

memset(caDummyMesg,’\0’,4000+1)

memcpy(caDummyMesg,要插入的数据,长度);

exec sql allocate :pBlobLocator;

exec sql insert into 表名(…..,字段名,….)

values(…..，empty\_blob()，…..)

returning 字段名 into : pBlobLocator

iWriteLen = 要插入数据长度

exec sql lob write one :iWriteLen from :caDummyMesg into : pBlobLocator at :ioffset

exec sql free :pBlobLocator

## bitand函数

bitand ，按位与操作。

select bitand(0,0) from dual --0

select bitand(1,0) from dual --0

select bitand(0,1) from dual --0

select bitand(1,1) from dual --1

另外，oracle本身没有bitor，bitxor函数，但是可以实现和bitand的转换：

BITOR(x,y) = (x + y) - BITAND(x, y)

## rawtohex函数

将字符串转换成16进制插入或更新到表中的BLOB字段中去

update 表名 set blob字段=rawtohex(‘字符串’)

## hextoraw函数

## to\_numer函数

## utl\_raw.cast\_to\_varchar2()函数

## utl\_raw.cast\_to\_raw()函数

# SecureCRT

## 全屏模式：alt+enter

## 设置鼠标右键拷贝

Options->Global Options->Teminal->Mouse

# Office

## Word中常用快捷键

Shift+alt+上(下)方向键：调整文档中段落的顺序，也可以调整word表格中的行顺序

F4键：重复上一步操作

ctrl+w ：关闭文档

shift+F3：更改字母的大小写

ctrl+u：下划线

ctrl+b：粗体

ctrl+i：斜体

ctrl+l：段落居左

ctrl+r：段落居右

ctrl+e：段落居中

ctrl+end：到文档末

ctrl+home：到文档开头

ctrl+y:重复上一步操作

ctrl+d： 调出字体栏

ctrl+alt+c：版权符号©

ctrl+alt+句号：省略号

ctrl+alt+r:注册商标符号®

ctrl+alt+t：商标符号™

## word中换行符与回车换行符

^l为换行符即shift+enter键

^p为回车换行符即enter键

## word中分节符

所谓的"节"，就是Word用来划分文档的一种方式。之所以引入"节"的概念，是因为我们在编辑文档的时候，有时并不是从头到尾所有的页面都采用相同的外观。例如，在某些文档中，几章之间可能使用不同的页面设置（页边距、装订线、页眉页脚等），在默认情况下，Word把整篇文档视为一节，即所有的页面都使用相同的页面设置、页眉页脚等，这有时不是我们所希望的。下面以word2003为例说一下word中分节符的用法。

1.分节符所包含的信息

    分节符可以包含以下信息: 页面方向(横向或纵向)、页边距、分栏状态、纵向对齐方式、行号、页眉和页脚样式、页码、纸型大小及纸张来源。

2.插入分节符

    插入分节符的办法为，选择"插入"\*"分隔符"，弹出"分节符"对话框。然后选择合适的分节符类型，点击"确定"按钮即可。注意: 我们可以把分节符当作一种隐藏的代码，它包含了所在的位置之前页面的各种信息(注意:一定是其所在位置之前的，用户在删除分节符的时候，了解这一概念非常必要)。

3.让分节符现形

    插入了分节符之后，您很可能看不到它。因为在我们最常用的"页面"视图模式下，通常是看不到分节符的。这时，我们可以单击"常用"工具栏上的"显示/隐藏编辑标记"按钮，让分节符现出原形。可以看出，分节符使用双行的虚线表示，同时括号里注明了该分节符的类型。

4.了解分节符的类型

    我们可以看到，在Word 2002中有4种分节符可供选择，它们分别是"下一页"、"连续"、"奇数页"和"偶数页"选项。下面就一一介绍各个分节符的具体作用。

下一页

    在插入此分节符的地方，Word会强制分页，新的"节"从下一页开始。如果要在不同页面上分别应用不同的页码样式、页眉和页脚文字，以及想改变页面的纸张方向、纵向对齐方式或者纸型，应该使用这种分节符。

连续

    插入"连续"分节符后，文档不会被强制分页。但是，如果"连续"分节符前后的页面设置不同，例如纸型和纸张走向等，即使选择使用"连续"分节符，Word也会在分节符处强制文档分页。而"连续"分节符的作用，主要是帮助用户在同一页面上创建不同的分栏样式或不同的页边距大小。尤其是当我们要创建报纸样式的分栏时，更需要连续分节符的帮助。

    当然，使用Word 6.0到Word 2002版本的用户在设置报纸分栏时，并不需要手工插入分节符。用户只需选定需要分栏的段落，然后选择合适的分栏数量即可，Word会自动在这些段落的前后插入连续分节符。如果需要取消分栏，可以直接删除相关分节符，这比重新设置分栏要容易得多。

奇数页

    在插入"奇数页"分节符之后，新的一节会从其后的第一个奇数页面开始(以页码编号为准)。在编辑长篇文稿，尤其是书稿时，人们一般习惯将新的章节题目排在奇数页，此时即可使用"奇数页"分节符。注意: 如果上一章节结束的位置是一个奇数页，也不必强制插入一个空白页。在插入"奇数页"分节符后，Word会自动在相应位置留出空白页。

偶数页

   “偶数页”分节符的功能与奇数页的类似，只不过是后面的一节从偶数页开始，在此不再赘述。

5.改变分节符属性

    插入分节符之后，如果需要改变其属性，无需删除该分节符并重新插入新的分节符。您可以把光标放置在需要改变分节符属性的"节"中(即分节符前面的任意位置)，然后选择"文件"\*"页面设置"，在"页面设置"对话框中选择"版式"选项卡，在"节的起始位置"选项中，选择新的起始位置即可。

6.删除分节符

    在需要对分节符进行删除的时候，要注意分节符中所保存的版式信息影响的是其前面的文字，而不是后面的。这一点请大家一定要记牢，不然在删除时容易选错删除对象，而把编辑好的文档弄得面目全非。

    如果需要一次性删除文档中所有的分节符，可以使用Word 2002的"查找替换"功能。在"查找和替换"对话框中的"查找内容"选项中填写"^b"(这2个字符代表分节符)，也可以单击"高级"按钮，在"特殊字符"列表中选取"分节符"选项，并替换为空，然后点击"全部替换"按钮，文档中所有的分节符即可全部删除。

    值得注意的是，在删除分节符时，该分节符前面的文字会依照分节符后面的文字版式进行重新排版。例如，如果把一篇文档分为两个小节，第一小节分两栏，第二小节分三栏。此时如果删除它们之间的分节符，那么整篇文档就会变成三栏版式。

     直接在视图中选页面宣示出分节符，选中他，然后用delete删除分节符

7.利用分节符进行快速排版

    分节符是排版的好帮手，不过它的作用可能并不局限于我们通常所想象的那些。例如，用户可以把设定好版式的分节符保存在自动图文集中，这样在以后使用时，直接插入该自动条目即可，省去了许多繁琐的排版工作。另外，如果需要在一个新的小节运用前面某个小节的版式(例如纸型及纸张方向等)，可以进入"普通"视图模式，复制包含前面小节排版信息的分节符，然后把它粘贴到需要设定同样版式的段落后。这样，新分节符上方的文字也自动遵循同样的版式。还有一点要提醒大家，如果您使用复制粘贴的方式复制或移动文字，而复制的文字中包含分节符，那么当该段文字被粘贴后，文档中位于新粘贴文字前面的文字也将按新粘贴文字中分节符所规定的版式进行排列。另外，由于在Word文档中，最后一小节的版式信息保存在文档的最后一个回车符中，所以当把最后一段连同回车符复制粘贴到文档的其他小节中时，最后一小节的版式同样会影响所粘贴文字前面的内容。

实例：

    有一篇论文，前几页不需要设置页眉页脚，也不需要添加页码。页眉页脚和页码只是从正文才开始添加。如何设置呢？

第一步：把光标放在正文开始的上一页最后，选择插入--分隔符，弹出分隔符对话框，在分节符类型区域选择“下一页”。这样从下一页开始，就是与上一页不同的节了，正文部分属于第2节。

第二步：把光标放在正文所在的节，选择视图--页眉和页脚，然后编辑页眉页脚以及页码。

注意：一定要注意页眉页脚工具栏中“连接到前一个”的使用，这时不应该按下该按钮，如果按下，则前一节（就是不应该加页眉页脚和页码的那几页）也会加上页眉页脚及页码。每次修改都应该注意该按钮是否被按下，因为word默认情况下是按下的。

最后关闭页眉页脚工具栏便可了。这样就达到了了不同页面实现不同的页眉页脚的目的。

当然，如果后面的章节或页面要使用跟前面不同的页面设置或者页眉页脚、页码等，还可以按这种方法增加不同的节来满足需要。

## word中enter和shift+enter

enter：段落标记，分了段落

shift+enter：人工换行符，未分段落，只是换行了

格式等是以段落为单位的

shift+enter是软回车( 回车后的段落仍属于前一个段落 ,只不过重新换行 ),enter是硬回车, 回车后的文字属于下一段落 .

它们的标记也是不同的 ,一个是向下的箭头 ,一个是弯曲箭头 .

## excel 数据有效性

Excel设置数据有效性实现单元格下拉菜单的3种方法

一、直接输入：

1.选择要设置的单元格，譬如A1单元格；

2.选择菜单栏的“数据”→“有效性”→出现“数据有效性”弹出窗口；

3.在“设置”选项中→“有效性条件”→“允许”中选择“序列”→右边的“忽略空值”和“提供下拉菜单”全部打勾→在“来源”下面输入数据，譬如“1,2,3,4,5,6,7,8,9”(不包括双引号，分割符号“,”必须为半角模式)→按“确定”就OK了，再次选择该A1单元格，就出现了下拉菜单。

二、引用同一工作表内的数据：

    如果同一工作表的某列就是下拉菜单想要的数据，譬如引用工作表Sheet1的B2：B5，B2：B5分别有以下数据：1、2、3、4，操作如下：

1.选择要设置的单元格，譬如A1单元格；

2.选择菜单栏的“数据”→“有效性”→出现“数据有效性”弹出窗口；

3.在“设置”选项中→“有效性条件”→“允许”中选择“序列”→右边的“忽略空值”和“提供下拉菜单”全部打勾→在“来源”下面输入数据“=$B$2:$B$5”，也可以按右边带红色箭头的直接选择B2：B5区域→按“确定”就OK了，再次选择该A1单元格，就出现了下拉菜单。

三、引用不同工作表内的数据（必须用到定义名称）：

如果不同工作表的某列就是下拉菜单想要的数据，譬如工作表Sheet1的A1单元格要引用工作表Sheet2的B2：B5区域，工作表Sheet2的B2：B5分别有以下数据：1、2、3、4，操作如下：

1.定义名称：菜单栏→“插 入”→“名称”→“定义”→弹出“定义名称”窗口，在“在当前工作薄中的名称”下面输入“DW”（可以自己随便明明）→“引用位置”下面输入“=Sheet2!$B$2:$B$5”，也可以按右边带红色箭头的直接选择B2：B5区域→按“添加”后再按“确定”完成第一步。

2.选择菜单栏的“数据”→“有效性”→出现“数据有效性”弹出窗口；

3.在“设置”选项中→“有效性条件”→“允许”中选择“序列”→右边的“忽略空值”和“提供下拉菜单”全部打勾→在“来源”下面输入“=DW”，“DW”就是刚刚定义好的名称，按“确定”就OK了，再次选择该A1单元格，就出现了下拉菜单。

# 术语简称解释

## Agile开发MoSCoW排序

优先级排序MoSCoW方法:  
Must：必须做的；  
  
Shoud：应该做的；  
  
Could：可以做的；  
  
Would not：不要做的。  
  
要按照这些顺序来做，保证Product Owner所需要的Must、Should完成，并力争Could能完成；在发生重要变更的时候，牺牲Could乃至Should保证变更。

## HACMP

HACMP（High Availability Cluster Muti-Processing)高可用性集群多处理

用于AIX 4.3的高可用性群集多处理（HACMP for AIX 4.3）是一种控制应用程序，它在使用高效集群内置的增强扩展性（ES）特征时可以链接多达32个RS/6000服务器或SP节点。[群集服务器](https://baike.baidu.com/item/%E7%BE%A4%E9%9B%86%E6%9C%8D%E5%8A%A1%E5%99%A8)或节点允许对数据进行并行访问，可以提供关键商业应用所要求的[冗余](https://baike.baidu.com/item/%E5%86%97%E4%BD%99)性和[容错性](https://baike.baidu.com/item/%E5%AE%B9%E9%94%99%E6%80%A7)。HACMP提供了[图形用户界面](https://baike.baidu.com/item/%E5%9B%BE%E5%BD%A2%E7%94%A8%E6%88%B7%E7%95%8C%E9%9D%A2)工具来帮助用户高效的安装、配置和管理群集。更重要的是，HACMP在配置和使用方面也很灵活。单处理器、对称多处理器（SMPs）和SP[节点](https://baike.baidu.com/item/%E8%8A%82%E7%82%B9)多可以集成到高效群集中。

计划性停机事件可能占据整个系统停机的大部分。HACMP允许通过[并行执行](https://baike.baidu.com/item/%E5%B9%B6%E8%A1%8C%E6%89%A7%E8%A1%8C)硬件、软件和其他维护活动使计划性停机减至最小，同时应用程序可以在其他节点上继续运行。可以将服务从一个群集节点移动至另一个群集节点，完成维护工作后再返回。

## VirtualBox中EFIUEFI的详细解释

### EFI

EFI == Extensible Firmware Interface == 扩展固件接口

从字面意思去看，也就很清楚了：

Extensible：扩展的；当然是针对的旧的BIOS来说，支持更多更好的功能，所以才称为扩展；

Firmware Interface：固件接口；之前旧的BIOS，其实也是固件接口，用来引导启动操作系统的。EFI，也是用来引导启动操作系统的。

### UEFI

根据Intel在：

[Extensible Firmware Interface (EFI) and Unified EFI (UEFI)](http://www.intel.com/content/www/us/en/architecture-and-technology/unified-extensible-firmware-interface/efi-homepage-general-technology.html)

中的解释：

EFI，是之前的叫法，现在改名叫做UEFI了。

UEFI= Unified EFI = Unified Extensible Firmware Interface = 统一的EFI = 统一的扩展固件接口

之所以把EFI改名为UEFI，估计采用更加统一的EFI，不要再各自为政了，使得大家都遵守这个规范，以便更好的推广此规范，鼓励大家都用这个东西。

VirtualBox手册中的解释是：

Enable EFI

This enables Extensible Firmware Interface (EFI), which replaces the legacy BIOS and may be useful for certain advanced use cases. Please refer to **Section 3.12, “Alternative firmware (EFI)”** for details.

大意为：

启用EFI，就是把之前旧的BIOS，换成了EFI。

换句话说，以前，启动系统是用的BIOS，如果你勾选了“启用EFI”，那么就不用BIOS，而是用EFI去启动系统了。

默认情况下，VirtualBox是用BIOS去启动（虚拟机）系统的，如果你需要（当前，你自己要知道你在干啥；不懂而去瞎设置，而导致无法启动虚拟机，那就是你自己的事情了）的话，可以去：

虚拟机的的设置->系统->主板->扩展特性->启用EFI(只针对某些操作系统)

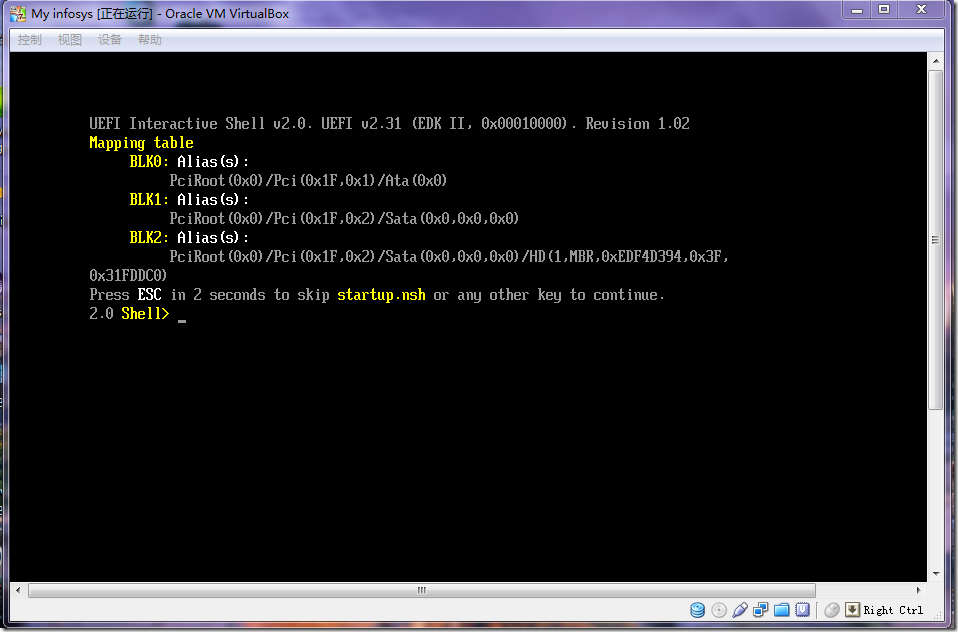
### 和EFI，UEFI相关的常见问题

#### 启用了EFI而导致无法启动虚拟机系统

比如有人:[1](http://zhidao.baidu.com/question/509967071.html),[2](http://zhidao.baidu.com/question/503041301.html)就遇到了，无法启动虚拟机的问题。

具体截图为：

[](http://www.crifan.com/files/pic/uploads/2012/12/UEFI-can-not-boot-vm_thumb.jpg)

[](http://www.crifan.com/files/pic/uploads/2012/12/uefi-can-not-boot-vm-in-virtualbox_thumb.png)

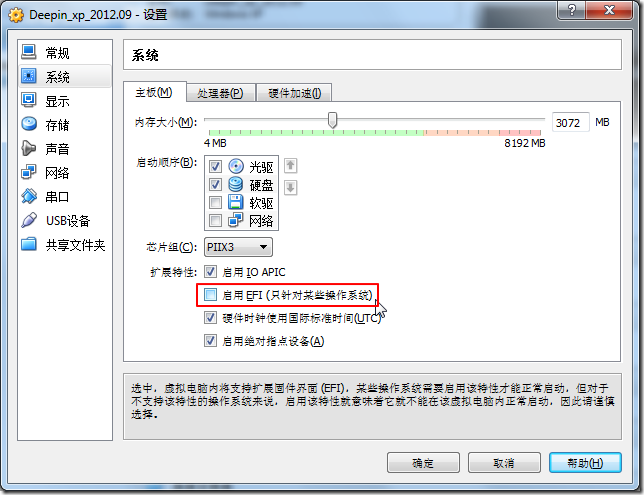
其具体原因则是：

自己在设置虚拟机的时候，不小心，勾选了那个“启用EFI(只针对某些操作系统)”

解决办法是：

取消选择“启用EFI(只针对某些操作系统)”，即可。

正确的设置如图（此处的截图是截取我自己的一个XP虚拟机中的配置）：

[](http://www.crifan.com/files/pic/uploads/2012/12/not-select-efi_thumb.png)

### 总结

在（VirtualBox中的）虚拟机设置的时候，没事的时候，尤其是你不懂的情况下，别乱选"启用EFI"。

否则，就会遇到，无法启动虚拟机的情况了。

## UTF-8编码

UTF-8（8-bit Unicode Transformation Format）是一种针对Unicode的可变长度字符编码，又称万国码。

Unicode 字符集，定义的是 "一个字符" 和 "一个整数"之间的映射（这样一个位置在字符集里称作 code point）  
其中，“中” 字对应的那个“整数” 是 01001110 00101101  
用 U + “整数” 这样的记法表示。  
“数” 被分成了很多区，最大的数现在不会超过 0x10FFFF。  
参见 [http://en.wikipedia.org/wiki/Unicode](//link.zhihu.com/?target=http%3A//en.wikipedia.org/wiki/Unicode)  
----  
有了这样的映射关系，接下来的问题是，如何在计算机里表示这些对应“字符”的“整数”。  
于是就有了 UTF-8/16/32 这些具体的编码方式。  
----  
UTF-32 最简单，直接用 32 位整数（4 Byte）表示字符对应的 “整数”，不管是什么字符，都用 32 位。  
----  
但是这样做浪费了很多空间。有些字对应的“数”很小，不需要 32-bits 就可以表示。为此人们制定了UTF-8 和 UTF-16 的“编码”方式。  
----  
具体介绍下 UTF-8。参见 [http://en.wikipedia.org/wiki/UTF-8](//link.zhihu.com/?target=http%3A//en.wikipedia.org/wiki/UTF-8)  
UTF-8 会用 1-6 个字节表示字符集中的一个“数”。  
----  
对于在 U + 0000 到 U + 007F 之间的”整数“，用一个 Byte 就可以表示。这个 Byte 的第一位是0，后 7 位 x 对应 code point。  
0xxxxxxx  
----  
对于在 U + 0080 到 U + 07FF 之间的“整数”，用两个 Byte 表示。  
110xxxxx | 10xxxxxx  
分别去掉这两个 Byte 前面的 110 和 10，把剩下的 x 拼起来就是对应字符的那个“数”了  
----  
中：11100100 10111000 10101101  
三个字节，格式是 3 | 1110xxxx | 10xxxxxx | 10xxxxxx  
把 x 对应的位提取出来：  
0100 111000 101101   
就是  
01001110 00101101  
了

## HBA-主机总线适配器

HBA:Host Bus Adapter

## NTP网络时间协议

NTP是网络时间协议(Network Time Protocol)，它是用来同步网络中各个计算机的时间的协议。

在计算机的世界里，时间非常地重要，例如对于火箭发射这种科研活动，对时间的统一性和准确性要求就非常地高，是按照A这台计算机的时间，还是按照B这台计算机的时间？NTP就是用来解决这个问题的，NTP（Network Time Protocol，网络时间协议）是用来使网络中的各个计算机时间同步的一种协议。它的用途是把计算机的时钟同步到[世界协调时](http://baike.baidu.com/view/2915317.htm)UTC，其精度在局域网内可达0.1ms，在互联网上绝大多数的地方其精度可以达到1-50ms。

它可以使计算机对其服务器或时钟源（如石英钟，GPS等等）进行时间同步，它可以提供高精准度的时间校正，而且可以使用加密确认的方式来防止病毒的协议攻击。

## 英语单词缩写规律

26个英文字母中有AEIOU5个元音字母，和1个半元音字母Y，单词缩写一般去掉元音字母。

## 汇票

汇票是出票人签发的，委托付款人在见票时或者在指定日期无条件支付确定的金额给收款人或者持票人的票据。

## CFETS

China Foreign Exchange Trade System

## CRepo-质押式回购

质押式回购（Pledge-style Repo）

### 质押式回购的概述

　　质押式回购是交易双方以[债券](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8)为[权利质押](http://wiki.mbalib.com/wiki/%E6%9D%83%E5%88%A9%E8%B4%A8%E6%8A%BC)所进行的短期[资金融通](http://wiki.mbalib.com/wiki/%E8%B5%84%E9%87%91%E8%9E%8D%E9%80%9A)业务。在质押式回购交易中，资金融入方（[正回购方](http://wiki.mbalib.com/wiki/%E6%AD%A3%E5%9B%9E%E8%B4%AD%E6%96%B9)）在将债券出质给资金融出方（逆回购方）融入资金的同时，双方约定在将来某一日期由正回购方向逆回购方返还[本金](http://wiki.mbalib.com/wiki/%E6%9C%AC%E9%87%91)和按约定[回购](http://wiki.mbalib.com/wiki/%E5%9B%9E%E8%B4%AD)利率计算的[利息](http://wiki.mbalib.com/wiki/%E5%88%A9%E6%81%AF)，逆回购方向正回购方返还原出质债券。

### 债券质押式回购交易的条件

　　（1）用于[债券回购](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8%E5%9B%9E%E8%B4%AD)的券种必须信誉高、流动性好。沪、深证券交易所[债券回购交易](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8%E5%9B%9E%E8%B4%AD%E4%BA%A4%E6%98%93)的抵押券主要是在交易所上市的[国债](http://wiki.mbalib.com/wiki/%E5%9B%BD%E5%80%BA)和[信用等级](http://wiki.mbalib.com/wiki/%E4%BF%A1%E7%94%A8%E7%AD%89%E7%BA%A7)在 [AAA](http://wiki.mbalib.com/wiki/AAA)以上的企业债券。

　　（2）在回购交易过程中，以券融资方应确保在回购成交至购回日期间，其在登记结算机构保留存放的回购抵押的债券量应大于融入资金量，否则将按卖空国债的规定予以处罚。

　　（3）债券回购交易过程中的以资融券方，在初始交易前必须将足够的[资金](http://wiki.mbalib.com/wiki/%E8%B5%84%E9%87%91)存入所委托的[证券营业部](http://wiki.mbalib.com/wiki/%E8%AF%81%E5%88%B8%E8%90%A5%E4%B8%9A%E9%83%A8)的[证券交易](http://wiki.mbalib.com/wiki/%E8%AF%81%E5%88%B8%E4%BA%A4%E6%98%93)结算[资金账户](http://wiki.mbalib.com/wiki/%E8%B5%84%E9%87%91%E8%B4%A6%E6%88%B7)。在回购期内不得动用[抵押债券](http://wiki.mbalib.com/wiki/%E6%8A%B5%E6%8A%BC%E5%80%BA%E5%88%B8)。

### 质押式回购的业务综述

　　质押式回购是交易双方以债券为权利质押所进行的短期资金融通业务。在质押式回购交易中，资金融入方（正回购方）在将债券出质给资金融出方（逆回购方）融入资金的同时，双方约定在将来某一日期由正回购方向逆回购方返还本金和按约定[回购利率](http://wiki.mbalib.com/wiki/%E5%9B%9E%E8%B4%AD%E5%88%A9%E7%8E%87)计算的利息，逆回购方向正回购方返还原出质债券。

### 质押式回购的一些相关规则

　　1、 现有的上海证券帐户可以交易新质押式回购。但原来已经做过回购登记的[证券帐户](http://wiki.mbalib.com/wiki/%E8%AF%81%E5%88%B8%E5%B8%90%E6%88%B7)，在撤销回购登记前，不能参加新质押式回购的交易。即非回购登记的证券账户可参加新质押式回购交易。深圳为现有的深圳证券帐户。

　　2、 自2006年5月8日起，[上交所](http://wiki.mbalib.com/wiki/%E4%B8%8A%E4%BA%A4%E6%89%80)交易系统不再接受对证券账户进行回购登记的申报，但接受回购注销申报。

　　3、 上交所所有[上市国债](http://wiki.mbalib.com/wiki/%E4%B8%8A%E5%B8%82%E5%9B%BD%E5%80%BA)（包含新上市国债，不包含[企业债](http://wiki.mbalib.com/wiki/%E4%BC%81%E4%B8%9A%E5%80%BA)）都可经入库申报后作为质押券，质押券将被换算成[标准券](http://wiki.mbalib.com/wiki/%E6%A0%87%E5%87%86%E5%88%B8)用于新质押式回购交易。

　　深圳除了质押式国债回购外，还支持企业债回购交易。

　　4、 上海结算公司以证券账户为单位对投资者的[回购融资](http://wiki.mbalib.com/wiki/%E5%9B%9E%E8%B4%AD%E8%9E%8D%E8%B5%84)额度进行总量核算，不同账户之间的标准券不可串用。[质押债券](http://wiki.mbalib.com/wiki/%E8%B4%A8%E6%8A%BC%E5%80%BA%E5%88%B8)与每笔融资回购交易不存在一一对应关系。

　　深圳结算公司以证券公司为单位对投资者的[回购融资](http://wiki.mbalib.com/wiki/%E5%9B%9E%E8%B4%AD%E8%9E%8D%E8%B5%84)额度进行总量核算，不同账户之间的标准券可串用。

　　5、 出入库申报不支持在集合竞价时间进行。

　　6、 上海t日买入的[国债](http://wiki.mbalib.com/wiki/%E5%9B%BD%E5%80%BA)可在t日申报提交为质押券，并可用于t日[回购融资](http://wiki.mbalib.com/wiki/%E5%9B%9E%E8%B4%AD%E8%9E%8D%E8%B5%84)。

　　深圳[t日](http://wiki.mbalib.com/wiki/T%E6%97%A5)买入的国债可在[t日](http://wiki.mbalib.com/wiki/T%E6%97%A5)申报提交为质押券，但不能用于t日[回购融资](http://wiki.mbalib.com/wiki/%E5%9B%9E%E8%B4%AD%E8%9E%8D%E8%B5%84)。

　　7、 上海质押库内的质押券可以实现当日实时替换；多余的质押券可以实时申报转回原证券帐户，并进行[现券交易](http://wiki.mbalib.com/wiki/%E7%8E%B0%E5%88%B8%E4%BA%A4%E6%98%93)。

　　深圳质押券可以申报转回原证券帐户，但转回的债券当日不能交易。

　　8、 回购到期日，融资方可以在可融资额度内进行新的融资回购，从而实现滚动融资；上海回购到期日，融资方可以申报将相关质押券转回原证券帐户，并可在当日卖出，卖出的资金可用于还到期购回款。

　　9、 上海[回购交易](http://wiki.mbalib.com/wiki/%E5%9B%9E%E8%B4%AD%E4%BA%A4%E6%98%93)的申报单位为手，1000元标准券为一手。申报数量必须为100手（10万资金）的整数倍，单笔数量不超过1万手（1000万资金）；回购价格为每百元资金到期的年收益率，最小变动单位为0.005元。

　　深圳回购交易的申报单位为张，100元标准券为1张。申报数量必须为10张的整数倍（1000元），单笔申报最大数量应不超过10万张（1000万元）；回购价格为每百元资金到期的年收益率，最小变动单位为0.001元。

　　10、 已经提交质押债券的上海投资者不得进行[指定交易](http://wiki.mbalib.com/wiki/%E6%8C%87%E5%AE%9A%E4%BA%A4%E6%98%93)变更。

　　11、 结算系统将在t日日终处理投资者通过[交易系统](http://wiki.mbalib.com/wiki/%E4%BA%A4%E6%98%93%E7%B3%BB%E7%BB%9F)申报的出入库指令。也就是说，交易系统日间回报成功的出入库申请是否能成功转出或转入质押库将在结算系统日终处理后才能确定。日间出入库成功的，结算系统日终处理时不一定成功。

　　12、 出库时的转出数量以元为单位，但必须是1000的整数倍。多出部分将被舍尾。如根据标准券余额计算可以转出1900元，由于舍尾只能转出1000元。出库申报的结果可以是部分成功部分失败。

　　13、国债质押期间的兑息直接发放给投资者的。质押期间的兑付，处理当日，系统首先检查兑付国债所涉出质证券账户有无[欠库](http://wiki.mbalib.com/wiki/%E6%AC%A0%E5%BA%93)。若无欠库，则根据标准券余额、兑付国债的[标准券折算率](http://wiki.mbalib.com/wiki/%E6%A0%87%E5%87%86%E5%88%B8%E6%8A%98%E7%AE%97%E7%8E%87)计算出可兑付国债数量，并予以兑付划付，且可兑付国债的划付出库优先于非兑付国债的出库。若有欠库，则全部暂不兑付。系统将该证券账户的国债兑付款以兑付权的形式保留在质押库内，并按最后一次该国债适用的折算率计算标准券，直至该国债兑付权兑付为止。

[[编辑](http://wiki.mbalib.com/w/index.php?title=%E8%B4%A8%E6%8A%BC%E5%BC%8F%E5%9B%9E%E8%B4%AD&action=edit&section=5)]

### 新质押式回购与过去的质押式回购的比较

　　根据《上海证券交易所债券交易实施细则》，上证所于2006年5月8日起推出了新质押式国债回购。新质押式回购与过去的质押式回购相比区别主要在以下几个方面：

　　1、新质押式回购按照证券账户进行[回购交易](http://wiki.mbalib.com/wiki/%E5%9B%9E%E8%B4%AD%E4%BA%A4%E6%98%93)和核算标准券库存。过去回购标准券是以证券公司为核算单位，新质押式回购交易必须按账户申报，并且按证券账户核算标准券库存。

　　2、质押库制度。新质押式回购的融资方在进行回购申报前必须申报质押券，作为质押券的现券将被实行“转移占有”。即投资者必须指明那些现券作为质押券，这些债券被过户到质押库中，作为质押券的债券不可以卖出。

　　3、[交易系统](http://wiki.mbalib.com/wiki/%E4%BA%A4%E6%98%93%E7%B3%BB%E7%BB%9F)进行前端检查。在回购交易时，交易系统按证券账户对回购交易进行前端检查。只有在标准券足额的情况下，融资回购申报才有效，这意味着，投资者在进行[回购融资](http://wiki.mbalib.com/wiki/%E5%9B%9E%E8%B4%AD%E8%9E%8D%E8%B5%84)前，必须首先申报足额的质押券，否则不能进行回购交易；融券方的资金检查还是由证券公司系统实行前端控制。

　　2006年5月8日后，新质押式回购与过去质押式回购并行，但两个品种在[证券代码](http://wiki.mbalib.com/wiki/%E8%AF%81%E5%88%B8%E4%BB%A3%E7%A0%81)和证券账户两个环节都相互隔离。

## OutRepo-债券买断式回购

债券买断式回购(Bonds outright repo)

### 债券买断式回购概述

　　债券买断式回购是[正回购方](http://wiki.mbalib.com/wiki/%E6%AD%A3%E5%9B%9E%E8%B4%AD%E6%96%B9)以出售[债券](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8)现券的方式，向[逆回购方](http://wiki.mbalib.com/wiki/%E9%80%86%E5%9B%9E%E8%B4%AD%E6%96%B9)融入资金，并按照约定[利率](http://wiki.mbalib.com/wiki/%E5%88%A9%E7%8E%87)和期限，以债券回购方式赎回债券现券。与封闭式相比，主要在于债券归属权发生改变，有一次卖出和一次赎回，成为两次[现券交易](http://wiki.mbalib.com/wiki/%E7%8E%B0%E5%88%B8%E4%BA%A4%E6%98%93)的组合。

　　按照上述交易机制，“债券买断式回购”的参与者可在依次递减的不同回购券种上进行多次回购操作。比如，投资者第一次选择了90天的交易品种进行[逆回购](http://wiki.mbalib.com/wiki/%E9%80%86%E5%9B%9E%E8%B4%AD)，20天后如果出现问题，他就可以通过[正回购](http://wiki.mbalib.com/wiki/%E6%AD%A3%E5%9B%9E%E8%B4%AD)，再把手中该笔债券在60天内的任何品种上卖出去拿回现金，可以规避一定的资金紧张的风险。

### 办理对象：

　　凡是进入全国[银行间债券市场](http://wiki.mbalib.com/wiki/%E9%93%B6%E8%A1%8C%E9%97%B4%E5%80%BA%E5%88%B8%E5%B8%82%E5%9C%BA)的金融机构和非金融机构都可叙做债券买断式回购业务。主要以商业银行、[外资银行](http://wiki.mbalib.com/wiki/%E5%A4%96%E8%B5%84%E9%93%B6%E8%A1%8C)、财务公司、[农信社](http://wiki.mbalib.com/wiki/%E5%86%9C%E4%BF%A1%E7%A4%BE)、城信社、[证券公司](http://wiki.mbalib.com/wiki/%E8%AF%81%E5%88%B8%E5%85%AC%E5%8F%B8)、基金公司、保险公司和财务公司等金融机构为主。

### 管理规定：

　　1． 最长期限不得超过91天。

　　2． 交易双方可以按照交易对手的信用状况[协商](http://wiki.mbalib.com/wiki/%E5%8D%8F%E5%95%86)设定保证金或保证券。

　　3． 任何一家市场参与者单只券种的待返售债券余额应小于该只债券流通量的20％。

　　4． 任何一家市场参与者待返售债券总余额应小于其在[中央国债登记结算有限责任公司](http://wiki.mbalib.com/wiki/%E4%B8%AD%E5%A4%AE%E5%9B%BD%E5%80%BA%E7%99%BB%E8%AE%B0%E7%BB%93%E7%AE%97%E6%9C%89%E9%99%90%E8%B4%A3%E4%BB%BB%E5%85%AC%E5%8F%B8)托管的自营债券总量的200％。

### 债券买断式回购的特点

　　债券买断式回购具有以下主要特点：

　　（一）实券过户制度，债券买断式回购直接以[证券账户](http://wiki.mbalib.com/wiki/%E8%AF%81%E5%88%B8%E8%B4%A6%E6%88%B7)为基础进行交易，即回购成交后，融资方的国债即被过户到融券方的账户中，回购到期，国债再反向过户；

　　（二）债券买断式回购实行履约金保证制度，参与者进行债券买断式回购交易必须交纳一定比例的[履约保证金](http://wiki.mbalib.com/wiki/%E5%B1%A5%E7%BA%A6%E4%BF%9D%E8%AF%81%E9%87%91)，冻结在结算公司，回购期满才可解冻；

　　（三）债券买断式回购引入了不履约申报制度，在国债债券买断式回购到期时，无论融资方还是融券方如果无法按期履约，均可以主动进行到期回购的“不履约申报”；

　　（四）债券买断式回购实行有限责任交收制度，一方违约或申报不履约后，履约金划归守约的对手方，双方交收责任解除，因此，这是一种有限责任的交收制度。

### 债券买断式回购在功能上的优点

　　债券买断式回购同时具有融资和融券功能。债券买断式回购可以满足持券方传统的融资需求，在市场看涨时，提供一种做多机制，对于其对手方而言，债券买断式回购满足了[融券](http://wiki.mbalib.com/wiki/%E8%9E%8D%E5%88%B8)的需求，在市场下跌时，可以提供一种[做空机制](http://wiki.mbalib.com/wiki/%E5%81%9A%E7%A9%BA%E6%9C%BA%E5%88%B6)。

　　此外，债券买断式回购还具有远期价格发现功能，债券买断式回购的[成交价格](http://wiki.mbalib.com/wiki/%E6%88%90%E4%BA%A4%E4%BB%B7%E6%A0%BC)实际上就是该债券的远期价格，它反映了市场对于债券未来一定时间内的价格预期，投资者可以根据债券买断式回购所揭示出来的远期价格制定相应的投资策略，规避[市场风险](http://wiki.mbalib.com/wiki/%E5%B8%82%E5%9C%BA%E9%A3%8E%E9%99%A9)。

### 债券买断式回购的主要功能

　　 债券买断式回购不仅具有与[质押式回购](http://wiki.mbalib.com/wiki/%E8%B4%A8%E6%8A%BC%E5%BC%8F%E5%9B%9E%E8%B4%AD)一样的融资功能，而且还具有[价格发现](http://wiki.mbalib.com/wiki/%E4%BB%B7%E6%A0%BC%E5%8F%91%E7%8E%B0)和避险[套利](http://wiki.mbalib.com/wiki/%E5%A5%97%E5%88%A9)的功能，同时还能提高债券市场的流动性。

　　1、活跃市场交易、提高[债券的流动性](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8%E7%9A%84%E6%B5%81%E5%8A%A8%E6%80%A7)

　　 在封闭式回购中，由于[逆回购方](http://wiki.mbalib.com/wiki/%E9%80%86%E5%9B%9E%E8%B4%AD%E6%96%B9)无权对质押的债券进行处置，造成了大量的债券冻结，使得可供交易的现券存量减少，严重影响了债券市场的流动性，限制了债券的交易，加剧了债券市场的[供求矛盾](http://wiki.mbalib.com/wiki/%E4%BE%9B%E6%B1%82%E7%9F%9B%E7%9B%BE)。随着货币市场的迅速发展，越来越多的债券被冻结、流动性越来越低、债券市场[供求矛盾](http://wiki.mbalib.com/wiki/%E4%BE%9B%E6%B1%82%E7%9F%9B%E7%9B%BE)进一步加剧。债券买断式回购最大的意义在于解冻了封闭式回购下的大量债券，解决了标的债券的二次利用问题，极大地提高了债券的流动性和利用效率。对活跃现券市场交易、提高现券交易的规模、解决债券供求矛盾等方面均具有重要意义。

　　2、[价格发现功能](http://wiki.mbalib.com/wiki/%E4%BB%B7%E6%A0%BC%E5%8F%91%E7%8E%B0%E5%8A%9F%E8%83%BD)

　　债券买断式回购是一方先将其持有的债券卖给另一方，并约定在未来的某一日期以约定的价格买回相等数量的同种债券的行为。它实际上是一项现券交易和一项相对应的远期交易的结合。由于债券买断式回购引入了远期价格揭示机制，其到期交易价格不仅反映了[融资融券](http://wiki.mbalib.com/wiki/%E8%9E%8D%E8%B5%84%E8%9E%8D%E5%88%B8)活动的成本，而且还反映了交易双方对利率变化趋势的预期以及对标的债券远期价格的预期，提高了市场的价格发现能力。此外，由于多种套利模式的存在，可以大大提高债券市场的[定价效率](http://wiki.mbalib.com/wiki/%E5%AE%9A%E4%BB%B7%E6%95%88%E7%8E%87)，使债券收益率曲线趋于平滑，债券[利率期限结构](http://wiki.mbalib.com/wiki/%E5%88%A9%E7%8E%87%E6%9C%9F%E9%99%90%E7%BB%93%E6%9E%84)更加合理。

　　3、避险套利功能

　　 目前的债券市场是一个单边的市场，只有在价格上涨时才能获利，投资者在预期[债券价格](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8%E4%BB%B7%E6%A0%BC)可能下跌的情况下，只有抛售手中的债券以避免损失。由于引入做空机制，债券买断式回购交易可以有效地解决这一问题。在升息预期下，逆回购方可以通过[卖空交易](http://wiki.mbalib.com/wiki/%E5%8D%96%E7%A9%BA%E4%BA%A4%E6%98%93)卖出所融入的标的券，待价格下跌后买回原券并到期还券，在回购期间内高卖低买所得差价即构成融券卖空交易的盈利。对于[债券持有人](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8%E6%8C%81%E6%9C%89%E4%BA%BA)来说，可以通过套期保值交易将所持债券余额与买断式逆回购下的融券[卖空](http://wiki.mbalib.com/wiki/%E5%8D%96%E7%A9%BA)操作匹配，[多头头寸](http://wiki.mbalib.com/wiki/%E5%A4%9A%E5%A4%B4%E5%A4%B4%E5%AF%B8)与[空头头寸](http://wiki.mbalib.com/wiki/%E7%A9%BA%E5%A4%B4%E5%A4%B4%E5%AF%B8)对冲，规避所持债券余额的市值波动风险。

　　4、催生多种[盈利模式](http://wiki.mbalib.com/wiki/%E7%9B%88%E5%88%A9%E6%A8%A1%E5%BC%8F)

　　由于债券买断式回购交易具有融券和做空的功能，所以在这基础上，通过各种组合可以派生出众多的新的[盈利模式](http://wiki.mbalib.com/wiki/%E7%9B%88%E5%88%A9%E6%A8%A1%E5%BC%8F)。除了传统的回购放大模式之外，利用债券买断式回购的做空机制，可以派生出逆回购放大模式，该模式的原理与传统的回购放大模式相似，只不过方向相反，在循环逆回购过程中不断放大空头头寸，实现空头效益的放大。通过将现券与买断式逆回购下的融券卖空操作匹配可以派生出[套期保值](http://wiki.mbalib.com/wiki/%E5%A5%97%E6%9C%9F%E4%BF%9D%E5%80%BC)模式。通过不同券种的走势分化进行组合操作可以派生出债券组合套利模式。通过不同回购品种利率的差异进行操作派生出[回购利率](http://wiki.mbalib.com/wiki/%E5%9B%9E%E8%B4%AD%E5%88%A9%E7%8E%87)组合套利模式。在债券买断式回购下投资者可以根据自己对市场的判断构建各种各样的组合模式，获取最大的收益。

### 债券买断式回购对市场的影响

　　 1、促进债券市场走向成熟和理性

　　 虽然和封闭式回购中循环正回购对市场产生的单边助涨效应一样，在债券买断式回购交易过程中，循环逆回购会放大卖空压力并对市场造成助跌动能，给市场造成短期的波动。但从长期来看，由于债券买断式回购增强了债券市场的流动性，大大提高了[债券市场](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8%E5%B8%82%E5%9C%BA)的定价效率，从而使债券市场的交易行为更加成熟和理性。同时由于跨市场套利的存在，银行间和交易所市场的联动性明显加强，成交更趋活跃，价差逐渐缩小，[市场收益率](http://wiki.mbalib.com/wiki/%E5%B8%82%E5%9C%BA%E6%94%B6%E7%9B%8A%E7%8E%87)曲线更合理。

　　2、为投资者提供了更多盈利机会

　　 由于债券买断式回购引入了做空机制，使得投资者不仅在债券价格上涨的情况下获利，而且在预期债券价格下跌的情况下也能够通过做空获利。这一点对银行、保险、基金来说尤其重要，因为[债券投资](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8%E6%8A%95%E8%B5%84)是他们资产结构中相当重要的组成部分，在市场下跌的过程中，不可能通过卖出所有债券来规避价格风险，债券买断式回购的推出为它们提供了有效的避险工具和盈利手段。另外，债券买断式回购派生出的众多新的[盈利模式](http://wiki.mbalib.com/wiki/%E7%9B%88%E5%88%A9%E6%A8%A1%E5%BC%8F)为投资者提供了的巨大的操作空间。

　　3、推动债券市场创新进程

　　由于债券买断式回购实质上是即期交易和反向远期交易的组合，因此随着债券买断式回购的启动，[国债远期交易](http://wiki.mbalib.com/wiki/%E5%9B%BD%E5%80%BA%E8%BF%9C%E6%9C%9F%E4%BA%A4%E6%98%93)的推出应该是顺理成章的事，而比远期交易更具标准化特点的[国债期货交易](http://wiki.mbalib.com/wiki/%E5%9B%BD%E5%80%BA%E6%9C%9F%E8%B4%A7%E4%BA%A4%E6%98%93)复出预计也会为时不远。所以债券买断式回购的推出对推动[国债市场](http://wiki.mbalib.com/wiki/%E5%9B%BD%E5%80%BA%E5%B8%82%E5%9C%BA)的创新具有重要的意义。

## SHIBOR-上海银行间同业拆放利率

上海银行间同业拆借利率*,*Shibor是Shanghai Inter bank Offered Rate.

### 上海银行间同业拆放利率

　　上海银行间同业拆放利率（Shanghai Interbank Offered Rate，简称Shibor），以位于上海的全国银行间同业拆借中心为技术平台计算、发布并命名，是由[信用等级](http://wiki.mbalib.com/wiki/%E4%BF%A1%E7%94%A8%E7%AD%89%E7%BA%A7)较高的银行组成报价团自主报出的人民币同业拆出利率计算确定的算术平均利率，是[单利](http://wiki.mbalib.com/wiki/%E5%8D%95%E5%88%A9)、无担保、批发性[利率](http://wiki.mbalib.com/wiki/%E5%88%A9%E7%8E%87)。

　　目前，对社会公布的Shibor品种包括隔夜、1周、2周、1个月、3个月、6个月、9个月及1年。

　　Shibor报价[银行团](http://wiki.mbalib.com/wiki/%E9%93%B6%E8%A1%8C%E5%9B%A2)现由16家商业银行组成。报价银行是公开市场[一级交易商](http://wiki.mbalib.com/wiki/%E4%B8%80%E7%BA%A7%E4%BA%A4%E6%98%93%E5%95%86)或外汇市场做市商，在中国货币市场上人民币交易相对活跃、[信息披露](http://wiki.mbalib.com/wiki/%E4%BF%A1%E6%81%AF%E6%8A%AB%E9%9C%B2)比较充分的银行。中国人民银行成立Shibor工作小组，依据《上海银行间同业拆放利率（Shibor）实施准则》确定和调整报价[银行团](http://wiki.mbalib.com/wiki/%E9%93%B6%E8%A1%8C%E5%9B%A2)成员、监督和管理Shibor运行、规范报价行与指定发布人行为。

　　全国银行间同业拆借中心受权Shibor的报价计算和信息发布。每个交易日根据各报价行的报价，剔除最高、最低各2家报价，对其余报价进行算术平均计算后，得出每一期限品种的Shibor，并于11:30对外发布。

### 上海银行间同业拆放利率（Shibor）技术方案

#### 基本点

　　·上海银行间同业拆放利率（Shanghai Interbank Offered Rate），简称Shibor。

　　·报价银行（Panel Banks）是公开市场一级交易商或外汇市场做市商，在中国货币市场上人民币交易相对活跃、信息披露比较充分的银行。

　　·期限品种包括O/N、1W、2W、1M、3M、6M、9M、1Y。

#### 报　价

　　·报价银行通过Shibor报价平台进行报价，报价时间不得晚于每个交易日的上午11：20（北京时间）。

　　·报价银行以年利率（％，Act/360，T+0）对各期限品种报价，保留4位小数。

　　·报价银行通过专线网进行报价，每家报价银行分配一个专属的报价页面。

#### 计　算

　　·每个[交易日](http://wiki.mbalib.com/wiki/%E4%BA%A4%E6%98%93%E6%97%A5)根据各报价银行的报价，剔除最高、最低各2家报卖价（Ask）。如果相同的最高（或最低）家数超过2家，系统随机剔除其中2家。

　　·11：20起系统自动对其余报价进行算术平均，计算得出每一期限品种的Shibor。

#### 发　布

　　·全国银行间同业拆借中心11：30起，通过网站对外发布当日Shibor。

## Cash Bond-现券买卖

### 什么是现券买卖

**现券买卖**是指交易双方以约定的[价格](http://wiki.mbalib.com/wiki/%E4%BB%B7%E6%A0%BC)转让债券[所有权](http://wiki.mbalib.com/wiki/%E6%89%80%E6%9C%89%E6%9D%83)的交易行为，即一次性的[买断](http://wiki.mbalib.com/wiki/%E4%B9%B0%E6%96%AD)[行为](http://wiki.mbalib.com/wiki/%E8%A1%8C%E4%B8%BA)。

### 现券买卖的内容

　　现券买卖是[债券交易](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8%E4%BA%A4%E6%98%93)中最普遍的[交易方式](http://wiki.mbalib.com/wiki/%E4%BA%A4%E6%98%93%E6%96%B9%E5%BC%8F)。债券买卖双方对[债券](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8)的[成交价](http://wiki.mbalib.com/wiki/%E6%88%90%E4%BA%A4%E4%BB%B7)达成一致，在交易完成后立即办理债券的[交割](http://wiki.mbalib.com/wiki/%E4%BA%A4%E5%89%B2)和[资金](http://wiki.mbalib.com/wiki/%E8%B5%84%E9%87%91)的交收，或在很短的时间内办理交割、交收[[1]](http://wiki.mbalib.com/wiki/%E7%8E%B0%E5%88%B8%E4%B9%B0%E5%8D%96#_note-a)。

　　现券买卖按[净价交易](http://wiki.mbalib.com/wiki/%E5%87%80%E4%BB%B7%E4%BA%A4%E6%98%93)，全价结算；[资产支持证券](http://wiki.mbalib.com/wiki/%E8%B5%84%E4%BA%A7%E6%94%AF%E6%8C%81%E8%AF%81%E5%88%B8)按每百元面额对应的[本金](http://wiki.mbalib.com/wiki/%E6%9C%AC%E9%87%91)进行报价。

　　现券买卖可采用[询价交易方式](http://wiki.mbalib.com/wiki/%E8%AF%A2%E4%BB%B7%E4%BA%A4%E6%98%93%E6%96%B9%E5%BC%8F)和点击成交交易方式。询价交易方式下可用意向报价、双向报价（仅适用资产支持证券）和对话报价，点击成交[交易方式](http://wiki.mbalib.com/wiki/%E4%BA%A4%E6%98%93%E6%96%B9%E5%BC%8F)下可用做市报价、点击[成交](http://wiki.mbalib.com/wiki/%E6%88%90%E4%BA%A4)报价和[限价](http://wiki.mbalib.com/wiki/%E9%99%90%E4%BB%B7)报价。

### 现券买卖的交易要素

　　（1）净价：不含[应计利息](http://wiki.mbalib.com/wiki/%E5%BA%94%E8%AE%A1%E5%88%A9%E6%81%AF)的[债券价格](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8%E4%BB%B7%E6%A0%BC)，单位为元/百元[面值](http://wiki.mbalib.com/wiki/%E9%9D%A2%E5%80%BC)。

　　（2）应计利息：上一[付息日](http://wiki.mbalib.com/wiki/%E4%BB%98%E6%81%AF%E6%97%A5)（或[起息日](http://wiki.mbalib.com/wiki/%E8%B5%B7%E6%81%AF%E6%97%A5)）至[结算](http://wiki.mbalib.com/wiki/%E7%BB%93%E7%AE%97)日之间累计的按百元面值计算的[债券发行](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8%E5%8F%91%E8%A1%8C)人应付给[债券持有人](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8%E6%8C%81%E6%9C%89%E4%BA%BA)的[利息](http://wiki.mbalib.com/wiki/%E5%88%A9%E6%81%AF)，单位为元/百元面值。

　　（3）全价：全价=净价+应计利息，单位为元/百元面值。

　　（4）待偿期：成交日至[债券到期日](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8%E5%88%B0%E6%9C%9F%E6%97%A5)的期间。

　　（5）券面总额：交易债券的总面额，单位为万元。

　　（6）交易金额：交易金额=净价/100×券面总额×10000，单位为元。

　　（7）应计利息总额：应计利息总额=[应计利息](http://wiki.mbalib.com/wiki/%E5%BA%94%E8%AE%A1%E5%88%A9%E6%81%AF)/100×券面总额×10000，单位为元。

　　（8）结算金额：结算金额=交易金额+应计利息总额，单位（9）[清算](http://wiki.mbalib.com/wiki/%E6%B8%85%E7%AE%97)速度：成交日（T）与结算日之间的[工作日](http://wiki.mbalib.com/wiki/%E5%B7%A5%E4%BD%9C%E6%97%A5)天数（N），有T+0和T+1两种。

　　（10）结算日：[债券](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8)交割和资金支付的日期。结算日=成交日+清算速度。

　　（11）[结算方式](http://wiki.mbalib.com/wiki/%E7%BB%93%E7%AE%97%E6%96%B9%E5%BC%8F)：包括[券款对付](http://wiki.mbalib.com/wiki/%E5%88%B8%E6%AC%BE%E5%AF%B9%E4%BB%98)、[见款付券](http://wiki.mbalib.com/wiki/%E8%A7%81%E6%AC%BE%E4%BB%98%E5%88%B8)和[见券付款](http://wiki.mbalib.com/wiki/%E8%A7%81%E5%88%B8%E4%BB%98%E6%AC%BE)。

　　（12）每百元本金额：每百元待清偿的本金金额，单位为元/每百元[面值](http://wiki.mbalib.com/wiki/%E9%9D%A2%E5%80%BC)（仅适用资产支持证券）。

（13）本金额：本金额=每百元本金额/100×券面总额×10000（仅适用资产支持证券）。

## SL债券借贷

Securities Loan

### 什么是债券借贷

　　债券借贷是指债券融入方以一定数量的[债券](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8)为质物，从债券融出方借入标的债券，同时约定在未来某一日期归还所借入标的债券，并由债券融出方返还相应质物的债券融通行为。目前，债券借贷已成为[国际债券市场](http://wiki.mbalib.com/wiki/%E5%9B%BD%E9%99%85%E5%80%BA%E5%88%B8%E5%B8%82%E5%9C%BA)广泛使用的重要工具之一。

### 债券借贷的作用

　　债券借贷可以增强[债券市场](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8%E5%B8%82%E5%9C%BA)的流动性，流动性的强弱是区分一个债券市场是[新兴市场](http://wiki.mbalib.com/wiki/%E6%96%B0%E5%85%B4%E5%B8%82%E5%9C%BA)还是[成熟市场](http://wiki.mbalib.com/wiki/%E6%88%90%E7%86%9F%E5%B8%82%E5%9C%BA)的重要标志之一。目前在我国债券市场中部分的可流通债券沉淀于各个机构的[投资](http://wiki.mbalib.com/wiki/%E6%8A%95%E8%B5%84)户中，在机构的交易户中用于买卖的债券比例比较有限。债券借贷通过[借入债券](http://wiki.mbalib.com/wiki/%E5%80%9F%E5%85%A5%E5%80%BA%E5%88%B8)融出方手中暂不用于交易的那部分债券进行买卖交易，可提高债券的周转速度，进而可增加[市场](http://wiki.mbalib.com/wiki/%E5%B8%82%E5%9C%BA)的流动性。

　　债券借贷提供了合理与有效的做空手段。在债券借贷过程中，债券融入方可以先卖出，到期时再以较低的[价格](http://wiki.mbalib.com/wiki/%E4%BB%B7%E6%A0%BC)从市场上买入归还给债券融出方，从而达到做空的目的。债券借贷业务明确债券融入方须付出[成本](http://wiki.mbalib.com/wiki/%E6%88%90%E6%9C%AC)，借贷的期限最多可达365天，并且可以单个或多个债券作为质押品，为投资者提供多样化投资与[风险规避](http://wiki.mbalib.com/wiki/%E9%A3%8E%E9%99%A9%E8%A7%84%E9%81%BF)的手段。

　　债券借贷能更方便地满足交易中对临时[头寸](http://wiki.mbalib.com/wiki/%E5%A4%B4%E5%AF%B8)的[需求](http://wiki.mbalib.com/wiki/%E9%9C%80%E6%B1%82)和结算的需求，降低存货风险和[交易成本](http://wiki.mbalib.com/wiki/%E4%BA%A4%E6%98%93%E6%88%90%E6%9C%AC)。以[做市商](http://wiki.mbalib.com/wiki/%E5%81%9A%E5%B8%82%E5%95%86)为例，做市商一般会对自己的做市券种保持一个比较高的头寸，以防做市券种被大量点击成交后头寸不足。引入债券借贷业务后，做市商可以通过“借券”来满足对做市券种的临时需求。对于非做市商同样也会经常面临类似的临时头寸需求。在过去，机构交易员会通过其他各种业务的组合方式或者通过私底下的合同进行“借券”，但多[业务组合](http://wiki.mbalib.com/wiki/%E4%B8%9A%E5%8A%A1%E7%BB%84%E5%90%88)的方式成本比较高，而私下的交易缺乏法律的保障，存在比较大的[信用风险](http://wiki.mbalib.com/wiki/%E4%BF%A1%E7%94%A8%E9%A3%8E%E9%99%A9)。债券借贷业务的推出将“借券”交易进行规范，并提供相应的[交易系统](http://wiki.mbalib.com/wiki/%E4%BA%A4%E6%98%93%E7%B3%BB%E7%BB%9F)和交易备案，为防范违约风险，促进市场有序、健康地发展提供有力的保障。

　　债券借贷业务为[债券持有人](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8%E6%8C%81%E6%9C%89%E4%BA%BA)提供新的盈利渠道，过去[债券持有人](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8%E6%8C%81%E6%9C%89%E4%BA%BA)只能通过债券的[利息收入](http://wiki.mbalib.com/wiki/%E5%88%A9%E6%81%AF%E6%94%B6%E5%85%A5)或者[债券买卖](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8%E4%B9%B0%E5%8D%96)的资本利差来获得盈利，债券借贷的融出方可以将原本不用于交易的债券借给债券融入方，并从债券融入方手中获得借贷[费用](http://wiki.mbalib.com/wiki/%E8%B4%B9%E7%94%A8)，从而获得额外的[收益](http://wiki.mbalib.com/wiki/%E6%94%B6%E7%9B%8A)。

## IRS-利率互换

利率互换(Interest Rate Swap)

### 利率互换概述

　　利率互换与[货币互换](http://wiki.mbalib.com/wiki/%E8%B4%A7%E5%B8%81%E4%BA%92%E6%8D%A2)在[互换交易](http://wiki.mbalib.com/wiki/%E4%BA%92%E6%8D%A2%E4%BA%A4%E6%98%93)中占主要地位。

　　利率互换是指两笔货币相同、债务额相同([本金](http://wiki.mbalib.com/wiki/%E6%9C%AC%E9%87%91)相同)、期限相同的[资金](http://wiki.mbalib.com/wiki/%E8%B5%84%E9%87%91)，作[固定利率](http://wiki.mbalib.com/wiki/%E5%9B%BA%E5%AE%9A%E5%88%A9%E7%8E%87)与[浮动利率](http://wiki.mbalib.com/wiki/%E6%B5%AE%E5%8A%A8%E5%88%A9%E7%8E%87)的调换。这个调换是双方的，如甲方以[固定利率](http://wiki.mbalib.com/wiki/%E5%9B%BA%E5%AE%9A%E5%88%A9%E7%8E%87)换取乙方的浮动利率，乙方则以浮动利率换取甲方的[固定利率](http://wiki.mbalib.com/wiki/%E5%9B%BA%E5%AE%9A%E5%88%A9%E7%8E%87)，故称互换。互换的目的在于降低[资金成本](http://wiki.mbalib.com/wiki/%E8%B5%84%E9%87%91%E6%88%90%E6%9C%AC)和[利率风险](http://wiki.mbalib.com/wiki/%E5%88%A9%E7%8E%87%E9%A3%8E%E9%99%A9)。利率互换与货币互换都是于1982年开拓的，是适用于[银行信贷](http://wiki.mbalib.com/wiki/%E9%93%B6%E8%A1%8C%E4%BF%A1%E8%B4%B7)和[债券筹资](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8%E7%AD%B9%E8%B5%84)的一种[资金融通](http://wiki.mbalib.com/wiki/%E8%B5%84%E9%87%91%E8%9E%8D%E9%80%9A)新技术，也是一种新型的避免[风险](http://wiki.mbalib.com/wiki/%E9%A3%8E%E9%99%A9)的金融技巧，目前已在国际上被广泛采用。

### 利率互换的前提条件

　　利率互换之所以会发生，是因为存在着这样的两个前提条件：

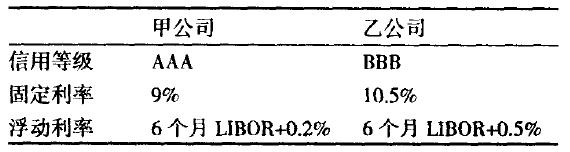
　　①存在品质加码差异

　　②存在相反的筹资意向

### 利率互换的原理

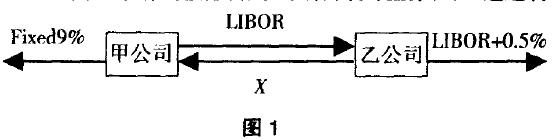
　　利率互换交易的基本原理就是[大卫•李嘉图的比较优势理论](http://wiki.mbalib.com/wiki/%E5%A4%A7%E5%8D%AB%E2%80%A2%E6%9D%8E%E5%98%89%E5%9B%BE%E7%9A%84%E6%AF%94%E8%BE%83%E4%BC%98%E5%8A%BF%E7%90%86%E8%AE%BA)与利益共享。首先，根据[比较优势理论](http://wiki.mbalib.com/wiki/%E6%AF%94%E8%BE%83%E4%BC%98%E5%8A%BF%E7%90%86%E8%AE%BA)，由于筹资双方[信用等级](http://wiki.mbalib.com/wiki/%E4%BF%A1%E7%94%A8%E7%AD%89%E7%BA%A7)、[筹资渠道](http://wiki.mbalib.com/wiki/%E7%AD%B9%E8%B5%84%E6%B8%A0%E9%81%93)、地理位置以及信息掌握程度等方面的不同，在各自的领域存在着[比较优势](http://wiki.mbalib.com/wiki/%E6%AF%94%E8%BE%83%E4%BC%98%E5%8A%BF)。因此，双方愿意达成协议，发挥各自优势，然后再互相交换债务，达到两者[总成本](http://wiki.mbalib.com/wiki/%E6%80%BB%E6%88%90%E6%9C%AC)的降低，进而由于利益共享，最终使得互换双方的筹资成本都能够得到一定的降低。下面举一个简单的例子进行说明。

　　假设有甲、乙两家公司，其[信用等级](http://wiki.mbalib.com/wiki/%E4%BF%A1%E7%94%A8%E7%AD%89%E7%BA%A7)及各自在固定利率市场和浮动利率市场上的[借款成本](http://wiki.mbalib.com/wiki/%E5%80%9F%E6%AC%BE%E6%88%90%E6%9C%AC)如下表所示：

[](http://wiki.mbalib.com/wiki/Image:%E5%88%A9%E7%8E%87%E4%BA%92%E6%8D%A21.jpg)

　　通过表中的数据可以看出，甲公司由于信用等级高，在浮动利率市场和固定利率市场都有优势，但是。不难发现两公司固定利率之差为1.5%，而浮动利率之差仅为O.3%。因此，可以认为甲公司在固定利率市场具有[比较优势](http://wiki.mbalib.com/wiki/%E6%AF%94%E8%BE%83%E4%BC%98%E5%8A%BF)。假设甲公司根据资产匹配的要求希望支付浮动利率利息而乙公司希望支付固定利率利息，如果二者按照各自原本的借款成本借款，总的[成本](http://wiki.mbalib.com/wiki/%E6%88%90%E6%9C%AC)为：6个月[LIBOR](http://wiki.mbalib.com/wiki/LIBOR)+O.2%+10.5%=6个月LIBOR+10.7%。根据双方的比较优势，甲公司借入[固定利率贷款](http://wiki.mbalib.com/wiki/%E5%9B%BA%E5%AE%9A%E5%88%A9%E7%8E%87%E8%B4%B7%E6%AC%BE)，乙公司借入[浮动利率贷款](http://wiki.mbalib.com/wiki/%E6%B5%AE%E5%8A%A8%E5%88%A9%E7%8E%87%E8%B4%B7%E6%AC%BE)，然后再进行互换，总的成本为：9%+6个月LIBOR+0.5%=6个月LIBOR+9.5%。很显然，相对于不进行互换而言一共节省成本1.2%。双方可以按照事先确定的比例分享这部分节省下来的成本。比如，按照利益均分原则每一方就可以节省0.6%。具体的操作流程可以由下图演示：

　　假设二者利益均分，即每一方节省0.6%，X=9.4%，那么，甲公司的实际贷款成本为：9%+LIBOR-9.4%=LIBOR-0.4%，而乙公司的[贷款](http://wiki.mbalib.com/wiki/%E8%B4%B7%E6%AC%BE)成本为：LIBOR+O.5%-LIBOR+9.4%=9.9%。双方成本都节省了0.6%。

[](http://wiki.mbalib.com/wiki/Image:%E5%88%A9%E7%8E%87%E4%BA%92%E6%8D%A2%E5%9B%BE1.jpg)

### 利率互换的特点

　　1.利率互换作为[金融衍生工具](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E8%A1%8D%E7%94%9F%E5%B7%A5%E5%85%B7)，为[表外业务](http://wiki.mbalib.com/wiki/%E8%A1%A8%E5%A4%96%E4%B8%9A%E5%8A%A1)，可以逃避[利率管制](http://wiki.mbalib.com/wiki/%E5%88%A9%E7%8E%87%E7%AE%A1%E5%88%B6)、税收限制等管制壁垒，有利于[资本](http://wiki.mbalib.com/wiki/%E8%B5%84%E6%9C%AC)的流动。

　　2.金额大，期限长，投机套利较难。绝大部分利率互换交易的期限在3—10年，由于期限较长，因此，投机套利的机会比较少，另外，利率互换一般属于[大宗交易](http://wiki.mbalib.com/wiki/%E5%A4%A7%E5%AE%97%E4%BA%A4%E6%98%93)，金额较大。

　　3.交易的成本较低，流动性强。利率互换是典型的场外市场交易([OTC](http://wiki.mbalib.com/wiki/OTC))工具，不能在交易所上市交易，可以根据[客户](http://wiki.mbalib.com/wiki/%E5%AE%A2%E6%88%B7)的具体要求进行设计[产品](http://wiki.mbalib.com/wiki/%E4%BA%A7%E5%93%81)，无[保证金](http://wiki.mbalib.com/wiki/%E4%BF%9D%E8%AF%81%E9%87%91)要求，交易不受时间、空间以及报价规则的限制。具体[交易事项](http://wiki.mbalib.com/wiki/%E4%BA%A4%E6%98%93%E4%BA%8B%E9%A1%B9)都由交易双方自主商定，交易手续简单，费用低。因此，成本较低，交易相当灵活。

　　4.风险较小。因为利率互换不涉及本金交易，[信用风险](http://wiki.mbalib.com/wiki/%E4%BF%A1%E7%94%A8%E9%A3%8E%E9%99%A9)仅取决于不同利率计算的利息差，而且其中包含了数个[计息期](http://wiki.mbalib.com/wiki/%E8%AE%A1%E6%81%AF%E6%9C%9F)，能够有效地避免[长期利率](http://wiki.mbalib.com/wiki/%E9%95%BF%E6%9C%9F%E5%88%A9%E7%8E%87)风险。

　　5.参与者信用较高。互换交易的双方一般信用较高，因为如果信用太低往往找不到合适的互换对手，久而久之，就被互换市场所淘汰。能够顺利参与交易的大都信用等级比较高。

### 利率互换的优点

　　①风险较小。因为利率互换不涉及[本金](http://wiki.mbalib.com/wiki/%E6%9C%AC%E9%87%91)，双方仅是互换利率，风险也只限于[应付利息](http://wiki.mbalib.com/wiki/%E5%BA%94%E4%BB%98%E5%88%A9%E6%81%AF)这一部分，所以风险相对较小；

　　②影响性微。这是因为利率互换对双方[财务报表](http://wiki.mbalib.com/wiki/%E8%B4%A2%E5%8A%A1%E6%8A%A5%E8%A1%A8)没有什么影响，现行的[会计规则](http://wiki.mbalib.com/wiki/%E4%BC%9A%E8%AE%A1%E8%A7%84%E5%88%99)也未要求把利率互换列在报表的附注中，故可对外保密；

　　③成本较低。双方通过互换，都实现了自己的愿望，同时也降低了筹资成本；

　　④手续较简，交易迅速达成。利率互换的缺点就是该互换不像[期货交易](http://wiki.mbalib.com/wiki/%E6%9C%9F%E8%B4%A7%E4%BA%A4%E6%98%93)那样有[标准化](http://wiki.mbalib.com/wiki/%E6%A0%87%E5%87%86%E5%8C%96)的[合约](http://wiki.mbalib.com/wiki/%E5%90%88%E7%BA%A6)，有时也可能找不到互换的另一方。

### 利率互换的功能

　　（1）降低[融资成本](http://wiki.mbalib.com/wiki/%E8%9E%8D%E8%B5%84%E6%88%90%E6%9C%AC)。出于各种原因，对于同种[货币](http://wiki.mbalib.com/wiki/%E8%B4%A7%E5%B8%81)，不同的[投资者](http://wiki.mbalib.com/wiki/%E6%8A%95%E8%B5%84%E8%80%85)在不同的[金融市场](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E5%B8%82%E5%9C%BA)的资信等级不同，因此融资的[利率](http://wiki.mbalib.com/wiki/%E5%88%A9%E7%8E%87)也不同，存在着相对的比较优势。利率互换可以利用这种相对比较优势进行互换套利以降低[融资成本](http://wiki.mbalib.com/wiki/%E8%9E%8D%E8%B5%84%E6%88%90%E6%9C%AC)。

　　（2）[资产负债管理](http://wiki.mbalib.com/wiki/%E8%B5%84%E4%BA%A7%E8%B4%9F%E5%80%BA%E7%AE%A1%E7%90%86)。利率互换可将固定利率[债权](http://wiki.mbalib.com/wiki/%E5%80%BA%E6%9D%83)（[债务](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%8A%A1)）换成[浮动利率债券](http://wiki.mbalib.com/wiki/%E6%B5%AE%E5%8A%A8%E5%88%A9%E7%8E%87%E5%80%BA%E5%88%B8)（债务）。

　　（3）对利率风险保值。对于一种货币来说，无论是固定利率还是浮动利率的持有者，都面临着利率变化的影响。对固定利率的[债务人](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%8A%A1%E4%BA%BA)来说，如果利率的走势下降，其可用更低成本重新融资，原先的固定利率对其不利；对于浮动利率的债务人来说，如果利率的走势上升，则成本会增大。

### 利率互换的交易机制

　　利率互换是受合同约束的双方在一定时间内按一定金额的本金彼此交换[现金流量](http://wiki.mbalib.com/wiki/%E7%8E%B0%E9%87%91%E6%B5%81%E9%87%8F)的协议。在利率互换中，若现有[头寸](http://wiki.mbalib.com/wiki/%E5%A4%B4%E5%AF%B8)为[负债](http://wiki.mbalib.com/wiki/%E8%B4%9F%E5%80%BA)，则互换的第一步是与债务利息相配对的[利息收入](http://wiki.mbalib.com/wiki/%E5%88%A9%E6%81%AF%E6%94%B6%E5%85%A5)；通过与现有受险部位配对后，借款人通过互换交易的第二步创造所需头寸。利率互换可以改变[利率风险](http://wiki.mbalib.com/wiki/%E5%88%A9%E7%8E%87%E9%A3%8E%E9%99%A9)。

[固定利率](http://wiki.mbalib.com/wiki/%E5%9B%BA%E5%AE%9A%E5%88%A9%E7%8E%87)支付者：在利率互换交易中支付固定利率；在利率互换交易中接受[浮动利率](http://wiki.mbalib.com/wiki/%E6%B5%AE%E5%8A%A8%E5%88%A9%E7%8E%87)；买进互换；是互换交易[多头](http://wiki.mbalib.com/wiki/%E5%A4%9A%E5%A4%B4)；称为支付方；是债券市场空头；对长期固定利率负债与浮动利率资产价格敏感。浮动利率支付者：在利率互换交易中支付浮动利率；在利率互换交易中接受[固定利率](http://wiki.mbalib.com/wiki/%E5%9B%BA%E5%AE%9A%E5%88%A9%E7%8E%87)；出售互换；是互换交易空头；称为接受方；是[债券市场](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8%E5%B8%82%E5%9C%BA)多头；对长期浮动利率负债与固定利率资产价格敏感。

### 利率互换的报价方式

　　在标准化的互换市场上，固定利率往往以一定年限的[国库券](http://wiki.mbalib.com/wiki/%E5%9B%BD%E5%BA%93%E5%88%B8)[收益率](http://wiki.mbalib.com/wiki/%E6%94%B6%E7%9B%8A%E7%8E%87)加上一个[利差](http://wiki.mbalib.com/wiki/%E5%88%A9%E5%B7%AE)作为报价。例如，一个十年期的[国库券](http://wiki.mbalib.com/wiki/%E5%9B%BD%E5%BA%93%E5%88%B8)收益率为6.2%，利差是68个基本点，那么这个十年期利率互换的价格就是6.88%，按市场惯例，如果这是利率互换的卖价，那么按此价格报价人愿意出售一个固定利率为6.88%，而承担浮动利率的风险。如果是买价，就是一个固定收益率6.2%加上63个基本点，即为6.83%，按此价格报价人愿意购买一个固定利率而不愿意承担浮动利率的风险。由于[债券](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8)的[二级市场](http://wiki.mbalib.com/wiki/%E4%BA%8C%E7%BA%A7%E5%B8%82%E5%9C%BA)上有不同年限的国库券买卖，故它的收益率适于充当不同年限的利率互换交易定价参数。[国库券](http://wiki.mbalib.com/wiki/%E5%9B%BD%E5%BA%93%E5%88%B8)的收益率组成利率互换交易价格的最基本组成部分，而利差的大小主要取决于互换市场的供需状况和竞争程度，利率互换交易中的价格利差是支付浮动利率的交易方需要用来抵补风险的一种费用。

**利率互换的报价表**

|  |  |  |  |
| --- | --- | --- | --- |
| **到期期限** | **银行支付固定利率** | **银行收取固定利率** | **当前**[**国库券利率**](http://wiki.mbalib.com/wiki/%E5%9B%BD%E5%BA%93%E5%88%B8%E5%88%A9%E7%8E%87)**（％）** |
| 2 | 2-yr.T+30点 | 2-yr.TN+38点 | 7.52 |
| 3 | 3-yr.T+35点 | 3-yr.TN+44点 | 7.71 |
| 4 | 4-yr.T+38点 | 4-yr.TN+48点 | 7.83 |
| 5 | 5-yr.T+44点 | 5-yr.TN+54点 | 7.90 |
| 6 | 6-yr.T+48点 | 6-yr.TN+60点 | 7.94 |
| 7 | 7-yr.T+50点 | 7-yr.TN+63点 | 7.97 |
| 10 | 10-yr.T+60点 | 10-yr.TN+75点 | 7.99 |
| 15 | 15-yr.T+90点 | 15-yr.TN+98点 | 6.08 |

　　注：表中的＂T＂是指[国库券](http://wiki.mbalib.com/wiki/%E5%9B%BD%E5%BA%93%E5%88%B8)，＂点＂是指基本点，一个基点为0.01%

### 利率互换的定价理论

　　利率互换的定价是寻找一种合适的固定利率，使某一笔新互换交易净现值为零。刁羽的文章中阐述了利率互换的定价理论分为无风险定价理论、单方违约风险定价理论和双方违约风险定价理论等三种，都是参考交易活动中的不同机构有没有[违约行为](http://wiki.mbalib.com/wiki/%E8%BF%9D%E7%BA%A6%E8%A1%8C%E4%B8%BA)而分析出的定价理论。现有的定价模型有零息票互换定价法、债券组合定价法及[远期利率协议](http://wiki.mbalib.com/wiki/%E8%BF%9C%E6%9C%9F%E5%88%A9%E7%8E%87%E5%8D%8F%E8%AE%AE)定价法等。

　　无风险定价理论是指在进行定价活动中不考虑[双方违约](http://wiki.mbalib.com/wiki/%E5%8F%8C%E6%96%B9%E8%BF%9D%E7%BA%A6)的情况，交易双方都按照预先签订的合约参与交易，最早是零息票互换定价法，把利率互换看成是两个债券的交换：固定利率和浮动利率债券。他们认为定价就是确定浮动利率债券价格，即[金融市场](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E5%B8%82%E5%9C%BA)的[利率期限结构](http://wiki.mbalib.com/wiki/%E5%88%A9%E7%8E%87%E6%9C%9F%E9%99%90%E7%BB%93%E6%9E%84)，这是定价的基础和关键。

　　单方违约风险定价理论中认为由于不同互换交易者进行交易时，某个交易者存在着违约的可能，那么在互换定价中加入违约的[风险补偿](http://wiki.mbalib.com/wiki/%E9%A3%8E%E9%99%A9%E8%A1%A5%E5%81%BF)是可以接受的。但是由于这种理论只考虑了一方交易者的违约风险，故其定价理论是单方违约定价理论。

　　双方违约风险定价理论主要是对交易者之间存在双向违约的风险下进行定价，Duffle和Huang(1996)推出了在双方都存在违约的情况下进行定价的理论模型，在这个模型中使用相关的[贴现率](http://wiki.mbalib.com/wiki/%E8%B4%B4%E7%8E%B0%E7%8E%87)来计算互换的[现值](http://wiki.mbalib.com/wiki/%E7%8E%B0%E5%80%BC)。

　　国内学者对利率互换定价归纳为两种方法，即多期远期合约法和[现值法](http://wiki.mbalib.com/wiki/%E7%8E%B0%E5%80%BC%E6%B3%95)。以上这些定价理论都是在假定一定的条件下得出的，能够得到一定的理论数据，但是在现实交易中还有许多情况会发生，所以定价模型的完善需要加入更多的变量，以使模型更加精确。

### 利率互换的内部控制

　　为了有效开展利率互换业务，企业应建立如下利率互换内部控制制度：

　　1.利率互换授权批准制度。利率互换业务管理制度、[业务流程](http://wiki.mbalib.com/wiki/%E4%B8%9A%E5%8A%A1%E6%B5%81%E7%A8%8B)及利率互换业务计划须经[董事会](http://wiki.mbalib.com/wiki/%E8%91%A3%E4%BA%8B%E4%BC%9A)或企业的最高管理当局批准：董事会或企业的最高管理当局授权总裁根据利率互换业务计划进行具体利率互换业务的批准，具体办法为：利率互换业务主管根据利率互换计划，与利率互换对手草拟利率互换合同，经[内部审计部门](http://wiki.mbalib.com/wiki/%E5%86%85%E9%83%A8%E5%AE%A1%E8%AE%A1%E9%83%A8%E9%97%A8)评估测算，提交测算报告，总裁根据测算报告结论判断决策，总裁批准后，盖章生效执行：如有利率互换[合同的变更](http://wiki.mbalib.com/wiki/%E5%90%88%E5%90%8C%E7%9A%84%E5%8F%98%E6%9B%B4)，也须报经总裁审批。

　　2.利率互换不相容职责分工制度。办理利率互换业务的不相容职责应相互分离、制约和监督。互换所涉及的不相容职责包括：

　　(1)利率互换计划的编制与审批利率互换计划由利率互换业务主管根据[财务计划](http://wiki.mbalib.com/wiki/%E8%B4%A2%E5%8A%A1%E8%AE%A1%E5%88%92)对资金的需求状况及利率波动情况编制。利率互换计划由董事会或企业最高管理当局根据利率互换主管呈报的利率互换业务计划审批.计划的编制与审批属于不兼容职责，应由不同的部门进行处理。

　　(2)利率互换业务项目的分析论证与执行由利率互换业务主管根据利率互换业务计划提出利率互换单项方案。具体内容有：利率互换对手、利率互换名义本金、利率互换价格。由企业内部审计部门对单项利率互换项目进行分析论证.主要内容为利率互换对手审核、利率互换定价审核。分析论证报告报总裁审批后，由利率互换业务部门执行。必须明确的是，分析论证与利率互换执行必须由不同部门来操作。

　　(3)利率互换业务的决策与[执行总裁](http://wiki.mbalib.com/wiki/%E6%89%A7%E8%A1%8C%E6%80%BB%E8%A3%81)负责单项利率互换业务的审批.利率互换业务部门负责利率互换业务的执行。两者为不相容职责，应分离。

　　(4)利率互换业务项目的处置审批与执行利率互换项目如须中止，应报总裁审批，总裁批准后，应由利率互换业务部门执行，此亦为不相容职责。应分开。

　　(5)利率互换业务的执行与相关会计记录承担利率互换[业务执行](http://wiki.mbalib.com/wiki/%E4%B8%9A%E5%8A%A1%E6%89%A7%E8%A1%8C)的部门不得进行利率互换业务的[会计记录](http://wiki.mbalib.com/wiki/%E4%BC%9A%E8%AE%A1%E8%AE%B0%E5%BD%95)。应由企业的财务部进行利率互换的[会计核算](http://wiki.mbalib.com/wiki/%E4%BC%9A%E8%AE%A1%E6%A0%B8%E7%AE%97)。

　　3.利率互换资产负债保管追踪制度。对于利率互换得到的资产和互换形成的负债要设置追踪查簿。及时记录换入的本金([名义本金](http://wiki.mbalib.com/wiki/%E5%90%8D%E4%B9%89%E6%9C%AC%E9%87%91))、利率、[汇率](http://wiki.mbalib.com/wiki/%E6%B1%87%E7%8E%87)、使用、回笼、归还以及对应的换出的本金(名义本金)、利率、汇率、收回、还贷等事项。所有经办人员须签名。严格规定互换所得资产的用途和使用审批手续，严格规定资金回笼渠道和时间，设置时间及金额预警。

　　4.与交易对手的定期对账制度。严格规定定期对账的时问，应由内部审计人员汇同[会计人员](http://wiki.mbalib.com/wiki/%E4%BC%9A%E8%AE%A1%E4%BA%BA%E5%91%98)、利率互换业务执行人员与利率互换对手定期对账，[对账](http://wiki.mbalib.com/wiki/%E5%AF%B9%E8%B4%A6)的内容包括：名义本金数量、收入利率、支付利率、应收应付利息金额等，并将对账状况记录于[备查簿](http://wiki.mbalib.com/wiki/%E5%A4%87%E6%9F%A5%E7%B0%BF)，如对账不符应及时查明原因，如存在重大差错应及找原因。并报告董事会。

　　5.利率互换会计核算制度。互换产生的资产和负债都要进行完整的会计记录(包括[会计帐簿](http://wiki.mbalib.com/wiki/%E4%BC%9A%E8%AE%A1%E5%B8%90%E7%B0%BF)和备查簿)，并对其增减变动及应收、应付利息进行相应的[会计核算](http://wiki.mbalib.com/wiki/%E4%BC%9A%E8%AE%A1%E6%A0%B8%E7%AE%97)。具体而言，应按每一项利率互换设置对应的换入，换出[明细分类帐](http://wiki.mbalib.com/wiki/%E6%98%8E%E7%BB%86%E5%88%86%E7%B1%BB%E5%B8%90)，详细记录其名称、金额、利率、汇率、日期、收入或支付的利息及手续费[汇兑损益](http://wiki.mbalib.com/wiki/%E6%B1%87%E5%85%91%E6%8D%9F%E7%9B%8A)等。

　　6.[内部审计](http://wiki.mbalib.com/wiki/%E5%86%85%E9%83%A8%E5%AE%A1%E8%AE%A1)。[内部审计人员](http://wiki.mbalib.com/wiki/%E5%86%85%E9%83%A8%E5%AE%A1%E8%AE%A1%E4%BA%BA%E5%91%98)经常评价企业规定的利率互换[管理制度](http://wiki.mbalib.com/wiki/%E7%AE%A1%E7%90%86%E5%88%B6%E5%BA%A6)及[工作流程](http://wiki.mbalib.com/wiki/%E5%B7%A5%E4%BD%9C%E6%B5%81%E7%A8%8B)是否得到贯彻执行：定期(按月)进行利率互换盘点询证；定期(每半年)对利率互换业务会计帐目进行审查，并编制[工作底稿](http://wiki.mbalib.com/wiki/%E5%B7%A5%E4%BD%9C%E5%BA%95%E7%A8%BF)和[内部审计报告](http://wiki.mbalib.com/wiki/%E5%86%85%E9%83%A8%E5%AE%A1%E8%AE%A1%E6%8A%A5%E5%91%8A)。由[内部审计部门](http://wiki.mbalib.com/wiki/%E5%86%85%E9%83%A8%E5%AE%A1%E8%AE%A1%E9%83%A8%E9%97%A8)牵头进行利率的[预测](http://wiki.mbalib.com/wiki/%E9%A2%84%E6%B5%8B)，并由内部审计部门对利率互换的定价进行审查，报总裁作为审批利率互换合同的依据。

　　7.利率互换定价审核制度。。利率互换的定价模型应是利率互换内部控制制度的一个重要的组成部份，企业应选定利率互换定价模型，报经董事会批准则。在签订利率互换合同之前。对利率互换的价格用定价模型进行测算，以测算价格作为基本数据，加入调节因子，从而确定利率互换合同的价格，在报总裁审批利率互换合同前，由内部审计部门进行评估和再测算.并将测算资料和结论一并报总裁，以确定是否签订此利率互换合同。

　　8.利率互换风险管理制度。利率互换的风险管理管理制度主要内容为：

　　1) 明确在企业总部专设机构进行利率互换的操作分支机构不得进行利率互换的操作，[分支机构](http://wiki.mbalib.com/wiki/%E5%88%86%E6%94%AF%E6%9C%BA%E6%9E%84)如需进行利率互换交易应委托总部专设机构操作。

　　2)强化利率互换定价审核规定由[内部审计机构](http://wiki.mbalib.com/wiki/%E5%86%85%E9%83%A8%E5%AE%A1%E8%AE%A1%E6%9C%BA%E6%9E%84)牵头选定利率预测模型，选定利率定性预测专家组，通过模型预测并结合专家组的[定性预测](http://wiki.mbalib.com/wiki/%E5%AE%9A%E6%80%A7%E9%A2%84%E6%B5%8B)，提高利率预测的准确性.再由内部审计部门对互换定价进行审核，并填写审核报告，报总裁作为审批利率互换合同的依据。

　　3)对交易对手的信用等级进行评估，规定各等组交易的限额对交易对手信用等级进行评估.可采用现行[银行信用](http://wiki.mbalib.com/wiki/%E9%93%B6%E8%A1%8C%E4%BF%A1%E7%94%A8)评估办法，各不同等级设置不同的利率互换交易上限，以控制风险。

　　4)设置抵押或担保交易对手为一般企业时，要求交易对手企业提供资产作为[抵押](http://wiki.mbalib.com/wiki/%E6%8A%B5%E6%8A%BC)，防范其违约行为；交易对手为[商业银行](http://wiki.mbalib.com/wiki/%E5%95%86%E4%B8%9A%E9%93%B6%E8%A1%8C)时，双方相互提供抵押担保，当交易对手信用明显恶化时，要求增加抵押的资产。

　　5)缔结[对冲交易](http://wiki.mbalib.com/wiki/%E5%AF%B9%E5%86%B2%E4%BA%A4%E6%98%93)，抵销市场风险规定一般企业作为交易对手进行利率互换交易时，本企业须缔结对冲利率互换以抵销[市场风险](http://wiki.mbalib.com/wiki/%E5%B8%82%E5%9C%BA%E9%A3%8E%E9%99%A9)。

　　6)选择交易双方利息轧差进行[结算](http://wiki.mbalib.com/wiki/%E7%BB%93%E7%AE%97)，以降低风险 。

[记帐](http://wiki.mbalib.com/wiki/%E8%AE%B0%E5%B8%90)时可详尽记录应收、应付利息金额。

　　但利息交割时规定采用利息轧差进行结算.以控制[交易风险](http://wiki.mbalib.com/wiki/%E4%BA%A4%E6%98%93%E9%A3%8E%E9%99%A9)。

　　建立、健全并切实执行利率互换的[内部控制制度](http://wiki.mbalib.com/wiki/%E5%86%85%E9%83%A8%E6%8E%A7%E5%88%B6%E5%88%B6%E5%BA%A6)，是实现利率互换管理的必要保证，是防范利率互换风险的根本措施。目前我国利率互换业务的开展尚处于初创阶段。各商业银行及相关机构尚未建立起完整的利率互换的内部控制体系，随着我国利率互换业务的日益增多，开展利率互换业务的商业银行及相关机构的不断增多，利率互换业务的内部控制制度的建立、健全和完善将是我们必须解决的重要问题。

### 利率互换的会计核算

#### 一、利率互换会计核算的原则

　　利率互换会计核算的关键是会计确认与计量。

　　1.利率互换合约的确认。[《企业会计准则第22号—金融工具确认和计量》](http://wiki.mbalib.com/wiki/%E3%80%8A%E4%BC%81%E4%B8%9A%E4%BC%9A%E8%AE%A1%E5%87%86%E5%88%99%E7%AC%AC22%E5%8F%B7%E2%80%94%E9%87%91%E8%9E%8D%E5%B7%A5%E5%85%B7%E7%A1%AE%E8%AE%A4%E5%92%8C%E8%AE%A1%E9%87%8F%E3%80%8B)中的第24条规定：“企业成为[金融工具](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E5%B7%A5%E5%85%B7)合同的一方时，应当确认一项[金融资产](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E8%B5%84%E4%BA%A7)或[金融负债](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E8%B4%9F%E5%80%BA)。”由于在合约签订后，[金融企业](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E4%BC%81%E4%B8%9A)即拥有并承担了日后利息互换的权利和义务，因而在合约签订日，交易双方应当在表内同时确认一项债权(金融资产)和债务(金融负债)。

　　2.利率互换合约的计量。《企业会计准则第22号—金融工具确认和计量》中的第30条规定：“企业初始确认金融资产或金融负债，应当按照[公允价值计量](http://wiki.mbalib.com/wiki/%E5%85%AC%E5%85%81%E4%BB%B7%E5%80%BC%E8%AE%A1%E9%87%8F)。”按照[公允价值](http://wiki.mbalib.com/wiki/%E5%85%AC%E5%85%81%E4%BB%B7%E5%80%BC)的定义，公允价值是指在[公平交易](http://wiki.mbalib.com/wiki/%E5%85%AC%E5%B9%B3%E4%BA%A4%E6%98%93)中，熟悉情况的交易双方自愿进行资产交换或者[债务清偿](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%8A%A1%E6%B8%85%E5%81%BF)的金额。事实上，公允价值不是一个独立的[计量属性](http://wiki.mbalib.com/wiki/%E8%AE%A1%E9%87%8F%E5%B1%9E%E6%80%A7)，而是包含[历史成本](http://wiki.mbalib.com/wiki/%E5%8E%86%E5%8F%B2%E6%88%90%E6%9C%AC)、[现行成本](http://wiki.mbalib.com/wiki/%E7%8E%B0%E8%A1%8C%E6%88%90%E6%9C%AC)、[现行市价](http://wiki.mbalib.com/wiki/%E7%8E%B0%E8%A1%8C%E5%B8%82%E4%BB%B7)以及现值在内的复合计量属性。

　　因此，利率交换中，由[利息支出](http://wiki.mbalib.com/wiki/%E5%88%A9%E6%81%AF%E6%94%AF%E5%87%BA)较高的一方向利息支出较低的一方支付按相同名义本金和两种利率形式计算出的利息差额，两者差额构成合约公允价值的组成部分，与[公允价值变动](http://wiki.mbalib.com/wiki/%E5%85%AC%E5%85%81%E4%BB%B7%E5%80%BC%E5%8F%98%E5%8A%A8)一道作为一项金融资产或金融负债计入资产负债表内。

　　3.账户设置。除存放中央银行款项(或其他有关科目)、利息收入、利息费用外，增加递延互换损益、应收利率互换、应付利率互换三个账户。

#### 二、利率互换会计核算实例

　　假如A方持有浮动利率借款，[利率水平](http://wiki.mbalib.com/wiki/%E5%88%A9%E7%8E%87%E6%B0%B4%E5%B9%B3)为[人民币](http://wiki.mbalib.com/wiki/%E4%BA%BA%E6%B0%91%E5%B8%81)6个月的[市场利率](http://wiki.mbalib.com/wiki/%E5%B8%82%E5%9C%BA%E5%88%A9%E7%8E%87)+20基本点。然而A方认为，人民币利率呈上升趋势，如果继续持有浮动利率债务，利息负担会越来越重，因此，希望将此浮动利率借款转换为固定利率借款。于是，与B方签订了一份开始于2006年3月1日的三年期的人民币利率互换合约。A方同意向B方支付由年固定利率4%和本金1.2亿人民币所计算的固定利息；而B方同意向A方支付由6个月期市场利率和同样本金支付的浮动利息。假设每半年交换一次利息。

　　双方将总共发生6次现金支付，第1次支付发生在2006年9月1日。

　　A方将支付B方固定利息：120百万×4%÷2=2.4百万

　　B方将支付A方浮动利息，该6个月的浮动利率必须在3月1日就确定，假定为3.2%，则支付的浮动利息为：120百万×3.2%÷2==1.92百万

　　第2、3、4、5、6次利息支付都同理，每一次固定利息的支付都是2.4百万，浮动利率则根据支付日前6个月确定的市场利率来计算。两个公司只需要支付两笔利息的差额。

　　为叙述方便，将A方支付固定利率4%并接受6个月市场利率的三年期利率互换的现金流状况列表如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 日期 | 6个月市场利率 | 浮动利率现金流 | 固定利率现金流 | [净现金流](http://wiki.mbalib.com/wiki/%E5%87%80%E7%8E%B0%E9%87%91%E6%B5%81) |
| 2006年3月1日 | 3.2% | - | - | - |
| 2006年9月1日 | 3.8% | +1.92 | -2.4 | -0.48 |
| 2007年3月1日 | 3.9% | +2.28 | -2.4 | -0.12 |
| 2007年9月1日 | 4.2% | +2.34 | -2.4 | -0.06 |
| 2008年3月1日 | 4.5% | +2.52 | -2.4 | +0.12 |
| 2008年9月1日 | 4.9% | +2.7 | -2.4 | +0.30 |
| 2009年3月1日 | 5.0% | +2.94 | -2.4 | +0.54 |

　　那么，A方应进行的会计处理如下：

　　1.2006年3月1日(签约日)[会计分录](http://wiki.mbalib.com/wiki/%E4%BC%9A%E8%AE%A1%E5%88%86%E5%BD%95)：

　　对利率互换合约按第一个付息期进行[会计确认](http://wiki.mbalib.com/wiki/%E4%BC%9A%E8%AE%A1%E7%A1%AE%E8%AE%A4)，

　　借：应收利率互换　　1.92

　　　　递延互换损益　　O.48

　　　　贷：应付利率互换　　2.4

　　2.2006年9月1日第一次[付息日](http://wiki.mbalib.com/wiki/%E4%BB%98%E6%81%AF%E6%97%A5)的会计分录：A方向B方支付利息差额，

　　借：应付利率互换　　0.48

　　　　贷：[存放中央银行款项](http://wiki.mbalib.com/wiki/%E5%AD%98%E6%94%BE%E4%B8%AD%E5%A4%AE%E9%93%B6%E8%A1%8C%E6%AC%BE%E9%A1%B9)　　0.48

　　同时，对利率互换合约按第二个付息期进行确认。

　　借：应收利率互换　　2.28

　　　　递延互换损益　　O.12

　　　　贷：应付利率互换　　2.4

　　3.2007年 3月 1日第二次付息日的会计分录：A方向B方支付利息差额，

　　借：应付利率互换　　0.12

　　　　贷：存放中央银行款项　　0.12

　　同时，对利率互换合约按第三个付息期进行确认，

　　借：应收利率互换　　2.34

　　　　递延互换损益　　0.06

　　　　贷：应付利率互换　　2.4

　　4.2007年9月1日第三次付息日的会计分录：A方向B方支付利息差额，

　　借：应付利率互换　　0.06

　　　　贷：存放中央银行款项　　0.06

　　同时，对利率互换合约按第四个付息期进行确认，

　　借：应收利率互换　　2.52

　　　　贷：应付利率互换　　2.4

　　　　递延互换损益　　O.12

　　5.2008年3月1日第四次付息日的会计分录：

　　A方向B方收取利息差额，

　　借：存放中央银行款项　　O.12

　　　　贷：应收互换利率　　O.12

　　同时，对利率互换合约按第五个付息期进行确认，

　　借：应收利率互换 　　2.7

　　　　贷：应付利率互换　　2.4

　　递延互换损益0.30

　　6.2008年9月1日第五次付息日的会计分录：

　　A方向B方收取利息差额，

　　借：存放中央银行款项　　O.3O

　　　　贷：应收互换利率　　O.3O

　　同时，对利率互换合约按第六个付息期进行确认，

　　借：应收利率互换　　2.94

　　　　贷：应付利率互换　　2.4

　　　　　　递延互换损益　　0.54

　　7.2009年3月1日互换合约到期，A方向B方收取利息差额，

　　借：存放中央银行款项　　O.54

　　　　贷：应收利率互换　　0.54

　　同时冲销有关[账面价值](http://wiki.mbalib.com/wiki/%E8%B4%A6%E9%9D%A2%E4%BB%B7%E5%80%BC)，

　　借：应付利率互换　　l3.74

　　　　贷：应收利率互换　　l3.74

　　在前3个付息期内，A方各月需平均摊销互换损失时，借记“利息费用”科目，贷记“递延互换损益”科目；在后3个付息期内，各月需平均摊销互换收益时，借记“递延互换损益”科目，贷记“利息收入”科目。

　　从以上的会计处理中可以看出，A方通过利率互换将浮动利率的借款转换为固定利率的借款，合约期内需要支付给B方利息共O.66百万，同时收取B方支付的利息共0.96百万，增加现金净流量O.3O百万，从而使通过利率互换降低筹资成本的效果十分明显。

[[编辑](http://wiki.mbalib.com/w/index.php?title=%E5%88%A9%E7%8E%87%E4%BA%92%E6%8D%A2&action=edit&section=12)]

### 利率互换对我国的现实意义

　　利率互换作为一种新型的金融衍生产品,在我国发展很快,特别是随着我国参与[国际金融](http://wiki.mbalib.com/wiki/%E5%9B%BD%E9%99%85%E9%87%91%E8%9E%8D)[资本运作](http://wiki.mbalib.com/wiki/%E8%B5%84%E6%9C%AC%E8%BF%90%E4%BD%9C)幅度的加大,利率互换已成为众多公司及银行之间常用的债务保值和资本升值的有效手段之一。从宏观角度看,利率互换对我国[金融资本](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E8%B5%84%E6%9C%AC)运作的意义为:

　　1)满足[投资主体](http://wiki.mbalib.com/wiki/%E6%8A%95%E8%B5%84%E4%B8%BB%E4%BD%93)[规避风险](http://wiki.mbalib.com/wiki/%E8%A7%84%E9%81%BF%E9%A3%8E%E9%99%A9)的要求

　　随着对外开放程度的加快,[利率市场化](http://wiki.mbalib.com/wiki/%E5%88%A9%E7%8E%87%E5%B8%82%E5%9C%BA%E5%8C%96)进程也必将越来越快。国内企业面临[市场化](http://wiki.mbalib.com/wiki/%E5%B8%82%E5%9C%BA%E5%8C%96)、乃至于国际化的竞争和冲击,必须要树立防范[金融风险](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E9%A3%8E%E9%99%A9)的意识,学会运用利率互换等金融衍生产品对自身的[债务资本](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%8A%A1%E8%B5%84%E6%9C%AC)进行合理运作,防止给企业带来金融损失,同时可提高企业的管理和运作水平。

　　2)促使金融市场的国际化,拓宽金融机构的业务范围

　　多种形式的利率互换业务发展,必然伴随着大量金融衍生工具的推出。在丰富投资主体资产组合的同时,也拓展了[金融机构](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E6%9C%BA%E6%9E%84)的业务范围,提升其在[国际市场](http://wiki.mbalib.com/wiki/%E5%9B%BD%E9%99%85%E5%B8%82%E5%9C%BA)上的[竞争力](http://wiki.mbalib.com/wiki/%E7%AB%9E%E4%BA%89%E5%8A%9B)。作为有实力、有头脑的公司、银行,绝不能仅满足于债务保值,而是要能通过积极主动的互换业务等市场运作,拓宽业务范围、增加自身的资本保有量。

　　3)深化[金融体制](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E4%BD%93%E5%88%B6)改革,加快利率市场化进程

　　我国的存[贷款利率](http://wiki.mbalib.com/wiki/%E8%B4%B7%E6%AC%BE%E5%88%A9%E7%8E%87)市场化程度还不高,缺乏有效地竞争,不利于形成反映[人民币](http://wiki.mbalib.com/wiki/%E4%BA%BA%E6%B0%91%E5%B8%81)供求状况的[市场利率](http://wiki.mbalib.com/wiki/%E5%B8%82%E5%9C%BA%E5%88%A9%E7%8E%87)。互换业务在我国推广的主要障碍就是没有形成权威性的基础[收益率曲线](http://wiki.mbalib.com/wiki/%E6%94%B6%E7%9B%8A%E7%8E%87%E6%9B%B2%E7%BA%BF)。因此,加大开展互换业务的力度,有助于推进利率的市场化进程。

　　4)有利于发展债券市场,丰富债券市场的品种结构

　　我国目前的债券市场期限结构和品种结构都不利于开发相关的衍生产品,所以,要重视[国债市场](http://wiki.mbalib.com/wiki/%E5%9B%BD%E5%80%BA%E5%B8%82%E5%9C%BA)在调节[供求关系](http://wiki.mbalib.com/wiki/%E4%BE%9B%E6%B1%82%E5%85%B3%E7%B3%BB)中的重要作用,同时积极发展利率互换等衍生产品市场。

### 利率互换会计核算实例

　　2007年3月16日甲银行与乙银行达成一笔人民币利率互换协议：

　　交易日：2007年3月16日

[起息日](http://wiki.mbalib.com/wiki/%E8%B5%B7%E6%81%AF%E6%97%A5)：2007年3月19日

　　到期日：2015年3月19日

　　名义本金额：人民币5000万元

　　甲银行支付：浮动利率FR007\*(7天回购定盘利率)，实际天数/365[复利](http://wiki.mbalib.com/wiki/%E5%A4%8D%E5%88%A9)

　　乙银行支付：固定利率，年利率3.61%，实际天数/365[单利](http://wiki.mbalib.com/wiki/%E5%8D%95%E5%88%A9)

　　浮动利率重置 [付息频率](http://wiki.mbalib.com/wiki/%E4%BB%98%E6%81%AF%E9%A2%91%E7%8E%87)：按季支付

　　甲银行会计分录：

　　1.2007年3月16日达成利率互换协议，甲银行将名义本金额记入表外：

　　收：衍生金融工具——利率互换(表外科目)50000000

　　2.2007年3月31日估值，随公允价值变动，如果上升，则按评估增值金额：

　　借：衍生金融工具——利率互换公允价值变动

　　贷：[公允价值变动损益](http://wiki.mbalib.com/wiki/%E5%85%AC%E5%85%81%E4%BB%B7%E5%80%BC%E5%8F%98%E5%8A%A8%E6%8D%9F%E7%9B%8A)——利率互换公允价值变动损益

　　公允价值评估减值做相反会计分录。

　　2007年4月30日、2007年5月31日[资产负债表日](http://wiki.mbalib.com/wiki/%E8%B5%84%E4%BA%A7%E8%B4%9F%E5%80%BA%E8%A1%A8%E6%97%A5)均要进行公允价值评估，并做如上会计分录。

　　3.2007年3月31日甲银行向交易对手乙银行支付浮动利率利息，按7天回购定盘利率支付利息，首个计息周期的FR007为1.5074%：

　　浮动利率利息=50000000×1.5074%×13÷365=26844.11(元)

　　借：[其他业务收入](http://wiki.mbalib.com/wiki/%E5%85%B6%E4%BB%96%E4%B8%9A%E5%8A%A1%E6%94%B6%E5%85%A5)——利率互换业务收入26844.11

　　贷：[银行存款](http://wiki.mbalib.com/wiki/%E9%93%B6%E8%A1%8C%E5%AD%98%E6%AC%BE)26844.11

　　同时，甲银行按固定利率3.61%收取对手乙银行支付的固定利率利息=50000000×3.61%×13÷365=64287.67(元)

　　借：银行存款64287.67

　　贷：其他业务收入——利率互换业务收入64287.67

　　4.2007年4月30日[计提](http://wiki.mbalib.com/wiki/%E8%AE%A1%E6%8F%90)应付利息，按7天[回购定盘利率](http://wiki.mbalib.com/wiki/%E5%9B%9E%E8%B4%AD%E5%AE%9A%E7%9B%98%E5%88%A9%E7%8E%87)3.3%支付利息，浮动利率利息=50000000×3.3%×1÷12=137500(元)

　　借：其他业务收入——利率互换业务收入137500

　　贷：应付利息——利率互换应付利息137500

　　同时计提[应收利息](http://wiki.mbalib.com/wiki/%E5%BA%94%E6%94%B6%E5%88%A9%E6%81%AF)，按固定利率3.61%收取，固定利率利息=50000000×3.61%×1÷12=150416.67(元)

　　借：应收利息——利率互换应收利息150416.67

　　贷：其他业务收入——利率互换业务收入150416.67

　　5月31日、6月30日也同样计提应付利息和应收利息。

　　5.2007年6月30日甲银行按季向交易对手乙银行实际支付浮动利率利息，按7天回购定盘利率(3.3%)支付的浮动利率利息=50000000×3.3%×3÷12=412500(元)

　　借：应付利息——利率互换应付利息412500

　　贷：银行存款412500

　　同时，甲银行收取对手乙银行按固定利率3.61%支付的固定利率利息=50000000X3.61%X3÷12=451250(元)

　　借：银行存款45l250

　　贷：应收利息——利率互换应收利息451250

　　6.利率互换交易终止，该笔交易的公允价值为0，因此须将上一[估值日](http://wiki.mbalib.com/wiki/%E4%BC%B0%E5%80%BC%E6%97%A5)的估值结果进行[红字冲销](http://wiki.mbalib.com/wiki/%E7%BA%A2%E5%AD%97%E5%86%B2%E9%94%80)，同时将名义本金额从[表外科目](http://wiki.mbalib.com/wiki/%E8%A1%A8%E5%A4%96%E7%A7%91%E7%9B%AE)付出。

　　付：衍生金融工具——利率互换(表外科目)50000000

　　红字冲销估值结果，评估增值金额：

　　借：衍生工具——利率互换公允价值变动(红字)

　　贷：公允价值变动损益——利率互换公允价值变动损益(红字)

　　公允价值评估减值则做相反分录。

　　7.2007年12月31日年终结转本年利润

　　借：其他业务收入——利率互换业务收入

　　公允价值变动损益——利率互换公允价值变动损益

　　贷：[本年利润](http://wiki.mbalib.com/wiki/%E6%9C%AC%E5%B9%B4%E5%88%A9%E6%B6%A6)

### 利率互换案例分析

#### 案例一：利率互换在金融机构中的应用

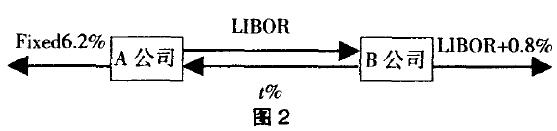
　　(一)利率互换在企业中的应用

　　1.降低企业的融资成本利率互换是一种基于比较优势和利益共享的交易，当一家企业在某一市场具有筹资优势，而在该市场筹资的类型与其所需不同时，可以通过互换得到其所需的筹资类型，同时，又降低了筹资成本。如前文所提到的两家企业所进行的互换交易。

　　2.规避利率风险在传统的[债务管理](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%8A%A1%E7%AE%A1%E7%90%86)中，债务一旦形成就必须履行下去，负债基本上没有流动性可言。相应的利率风险就不容易规避。而利率互换可以使负债更加灵活并转变成企业所需的形式，从而很好地规避了利率风险。例如，为了避免利牢的上升，交易者将浮动利率的负债与相同名义本金的固定利率负债进行互换，所收取的浮动利率冲抵原来负债利率支出，自己支付固定利率，从而将利率上升的[风险转嫁](http://wiki.mbalib.com/wiki/%E9%A3%8E%E9%99%A9%E8%BD%AC%E5%AB%81)出去。反之亦然。

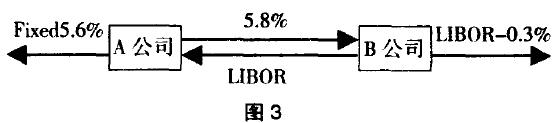
　　3.更好地进行资产负债的匹配企业的资产和负债往往不能很好的匹配，这样就会带来很大的风险，由于利率互换是以名义本金为基础进行交易，因此，就可以不经过资金转移，直接对资产和负债进行表外匹配。可以通过转换资产或转换负债两个方面来实现。下面分别通过例子对二者进行简要的说明。

　　(1)利用互换转换某项负债A公司原来持有一项利率为6.2%同定利率借款，B公司持有一项利率为LIBOR+0.8%的浮动利率借款。通过下图所示的互换：

[](http://wiki.mbalib.com/wiki/Image:%E5%88%A9%E7%8E%87%E4%BA%92%E6%8D%A2%E5%9B%BE2.jpg)

由图2可知，对于A公司来说，该互换将6.2%的[固定利率贷款](http://wiki.mbalib.com/wiki/%E5%9B%BA%E5%AE%9A%E5%88%A9%E7%8E%87%E8%B4%B7%E6%AC%BE)转换为LIBOR+0.2%的[浮动利率贷款](http://wiki.mbalib.com/wiki/%E6%B5%AE%E5%8A%A8%E5%88%A9%E7%8E%87%E8%B4%B7%E6%AC%BE)，实现了负债性质的转换。

　　(2)利用互换转换某项资产A公司拥有$l,000,000债券，期限5年，收益率为5.6%。B公司投资了一项为期5年的项目，投资额也为$l,000,000，收益为LIBOR-0.3%。通过下图所示的互换：

[](http://wiki.mbalib.com/wiki/Image:%E5%88%A9%E7%8E%87%E4%BA%92%E6%8D%A2%E5%9B%BE3.jpg.jpg)

　　由图3可知，对于A公司来说，该互换将收益率为5.6%的同定收益资产转换成收益为LIBOR-0.2%的浮动利率资产。实现了资产性质的转换。

　　除此之外，还可以进行[长期负债](http://wiki.mbalib.com/wiki/%E9%95%BF%E6%9C%9F%E8%B4%9F%E5%80%BA)(资产)与[短期负债](http://wiki.mbalib.com/wiki/%E7%9F%AD%E6%9C%9F%E8%B4%9F%E5%80%BA)(资产)的互换，以实现资产久期与负债久期的匹配。进行长短期互换时，双方由于期限的差别，需要签订一个“滚动”协议。也就是说，当第一份合约到期时，持有短期负债(资产)的公司就相同标的负债(资产)与对方再签订一份合约，如此“滚动”下去.最终达到双方负债(资产)久期的互换。

　　4.拓宽融资渠道某些企业由于信用等级等各方面的原因，融资渠道比较狭窄，或者不能获取实现对自己最适合的融资。通过利率互换，企业可以在自己最熟悉的市场七筹资，然后通过互换达到自己的目的。突破了某一市场对[企业融资](http://wiki.mbalib.com/wiki/%E4%BC%81%E4%B8%9A%E8%9E%8D%E8%B5%84)存在特定要求的限制，大大拓宽了融资渠道。

　　(二)利率互换在银行中的应用

　　1.有利于银行的资产负债管理随着利率是市场化的进程，商业银行所面临的利率风险越来越大。由于[商业银行负债](http://wiki.mbalib.com/wiki/%E5%95%86%E4%B8%9A%E9%93%B6%E8%A1%8C%E8%B4%9F%E5%80%BA)主要来自于居民存款，期限短，流动性高，受利率变化影响较大，而占资产大部分的贷款一般期限较长，以固定利率为主，从而导致了银行资产与负债期限结构和[利率结构](http://wiki.mbalib.com/wiki/%E5%88%A9%E7%8E%87%E7%BB%93%E6%9E%84)的严重不匹配，产生了较大的[利率敏感性缺口](http://wiki.mbalib.com/wiki/%E5%88%A9%E7%8E%87%E6%95%8F%E6%84%9F%E6%80%A7%E7%BC%BA%E5%8F%A3)。因此，商业银行可以通过一系列的利率互换来凋整资产和负债的久期和利率结构，使之更好的匹配，从而降低利率风险，保证其利润的稳定性。由于银行的资产负债结构和[保险公司](http://wiki.mbalib.com/wiki/%E4%BF%9D%E9%99%A9%E5%85%AC%E5%8F%B8)的恰好相反，因此，两者之间可以进行互换交易，达到双赢。两者之间的互换交易市场和渠道应该得到大力的发展。

　　2.有利于银行发展中问业务利率互换属于[场外交易](http://wiki.mbalib.com/wiki/%E5%9C%BA%E5%A4%96%E4%BA%A4%E6%98%93)，需要大量的[做市商](http://wiki.mbalib.com/wiki/%E5%81%9A%E5%B8%82%E5%95%86)和[经纪商](http://wiki.mbalib.com/wiki/%E7%BB%8F%E7%BA%AA%E5%95%86)，商业银行是充当这一角色的最好选择。作为利率互换双方的中介，银行可以赚取价差收入和无风险的佣金收入。另外，银行受到政府的监管以及各种法规的限制比较多，而利率互换又恰恰属于表外业务，不用改变银行的资产和负债就可以带来大量的收入。增加[中间业务](http://wiki.mbalib.com/wiki/%E4%B8%AD%E9%97%B4%E4%B8%9A%E5%8A%A1)的同时.可以减轻商业银行利润过于依赖存贷款利差的现象，又可以进一步降低银行所面临的利率风险。

　　其他的[金融机构](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E6%9C%BA%E6%9E%84)例如[基金公司](http://wiki.mbalib.com/wiki/%E5%9F%BA%E9%87%91%E5%85%AC%E5%8F%B8)、保险公司、[券商](http://wiki.mbalib.com/wiki/%E5%88%B8%E5%95%86)等都经常暴露在利率风险之中，因此，对互换也有很大的需求。例如，对于基金公司，当利率上升时，就可能面临赎回的压力，只能被动地将一些资产变现，从而改变了基金的整体[投资策略](http://wiki.mbalib.com/wiki/%E6%8A%95%E8%B5%84%E7%AD%96%E7%95%A5)，还会造成一段时间的净值下降。对于处理这种利率风险，利率互换将是一个很好的选择。券商也可以积极地参与到互换经纪业务中来，在进一步活跃整个[资金市场](http://wiki.mbalib.com/wiki/%E8%B5%84%E9%87%91%E5%B8%82%E5%9C%BA)的同时获取中介费用。

　　总之，作为目前应用最普遍的衍生金融工具之一，利率互换在[利率市场化](http://wiki.mbalib.com/wiki/%E5%88%A9%E7%8E%87%E5%B8%82%E5%9C%BA%E5%8C%96)的大背景下，必将发挥着越来越重要的作用。当然利率互换也是一把双刃剑，作为衍生金融工具的[杠杆效应](http://wiki.mbalib.com/wiki/%E6%9D%A0%E6%9D%86%E6%95%88%E5%BA%94)在放大收益的同时也放大了风险。在积极开展利率互换业务时，也应该采取一定的防范措施.尽可能减轻其消极的影响。

[[编辑](http://wiki.mbalib.com/w/index.php?title=%E5%88%A9%E7%8E%87%E4%BA%92%E6%8D%A2&action=edit&section=16)]

#### 案例二：利率互换实例

[国际资本市场](http://wiki.mbalib.com/wiki/%E5%9B%BD%E9%99%85%E8%B5%84%E6%9C%AC%E5%B8%82%E5%9C%BA)上发生的较早一次利率互换发生在1982年7月，当时，[德意志银行](http://wiki.mbalib.com/wiki/%E5%BE%B7%E6%84%8F%E5%BF%97%E9%93%B6%E8%A1%8C)发行了3亿美元7年期的[固定利率](http://wiki.mbalib.com/wiki/%E5%9B%BA%E5%AE%9A%E5%88%A9%E7%8E%87)[欧洲债券](http://wiki.mbalib.com/wiki/%E6%AC%A7%E6%B4%B2%E5%80%BA%E5%88%B8)的同时，通过与另外3家银行达成的[互换协议](http://wiki.mbalib.com/wiki/%E4%BA%92%E6%8D%A2%E5%8D%8F%E8%AE%AE)，交换成以[LIBOR](http://wiki.mbalib.com/wiki/LIBOR)为[基准利率](http://wiki.mbalib.com/wiki/%E5%9F%BA%E5%87%86%E5%88%A9%E7%8E%87)的[浮动利率](http://wiki.mbalib.com/wiki/%E6%B5%AE%E5%8A%A8%E5%88%A9%E7%8E%87)债务。这项交易使得双方能利用各自在不同金融市场上的[相对优势](http://wiki.mbalib.com/wiki/%E7%9B%B8%E5%AF%B9%E4%BC%98%E5%8A%BF)获得利益，即德意志银行按低于[LIBOR](http://wiki.mbalib.com/wiki/LIBOR)的利率支付浮动利息，而其他三家银行则通过德意志银行的较高的资信等级换得了优惠的固定利率债务。

[

#### 案例三：利率互换对商业银行的影响与对策

##### 一、利率互换业务对商业银行的作用

　　在基准利率和市场化利率分割监管的环境下，本身就可能存在利率互换交易发生的可能性。例如，银行资产负债管理中，资产负债会出现一定的[持续期](http://wiki.mbalib.com/wiki/%E6%8C%81%E7%BB%AD%E6%9C%9F)[缺口](http://wiki.mbalib.com/wiki/%E7%BC%BA%E5%8F%A3)，每家机构都会保持一定的[风险敞口](http://wiki.mbalib.com/wiki/%E9%A3%8E%E9%99%A9%E6%95%9E%E5%8F%A3)，如果银行A保持较大的正的持续期缺口(资产的持续期高于负债的持续期)，市场利率上升时，资产负债的净[市场价值](http://wiki.mbalib.com/wiki/%E5%B8%82%E5%9C%BA%E4%BB%B7%E5%80%BC)将下降较多，市场风险加大；而银行B恰恰相反，保持着较大的负的持续期缺口。为[有效控制](http://wiki.mbalib.com/wiki/%E6%9C%89%E6%95%88%E6%8E%A7%E5%88%B6)[风险敞口](http://wiki.mbalib.com/wiki/%E9%A3%8E%E9%99%A9%E6%95%9E%E5%8F%A3)，两家银行运用利率互换工具“互补遗缺”，在一定期限内，银行A浮动利率负债较多，但通过向银行B支付固定利率，将浮动利率性质的负债转换为固定利率负债，从而提高了负债的持续期。同样，银行B固定利率负债较多，通过向银行A支付浮动利率，将固定利率性质的负债转换为浮动利率负债，达到降低负债持续期的效果。这样，两家银行通过利率互换，就可以有效降低资产负债的持续期缺口，从而减少利率变化环境中的风险敞口和经营压力。

###### (一)利率互换有利于商业银行改善资产负债管理

　　在对资产和负债头寸进行管理时，可以利用[货币市场](http://wiki.mbalib.com/wiki/%E8%B4%A7%E5%B8%81%E5%B8%82%E5%9C%BA)和资本市场来管理，而用利率互换则更具有优势。由于利率互换是以名义本金为基础进行的，即它可以不经过真实[资金运动](http://wiki.mbalib.com/wiki/%E8%B5%84%E9%87%91%E8%BF%90%E5%8A%A8)，直接对资产负债额及其利率期限结构进行表外重组。在负债的利率互换中，付固定利率相当于借人一笔名义固定利率债务，会延长负债利率期限；付浮动利率相当于借入一笔名义浮动利率债务，会缩短负债的利率期限。而在资产利率互换中，收固定利率等于占有一笔名义固定利率债权，会延长资产的利率期限，而收浮动利率等于占有一笔名义浮动利率债权，会缩短资产的利率期限。为此，积极扩大运用利率互换工具，有利于提高[利率风险管理](http://wiki.mbalib.com/wiki/%E5%88%A9%E7%8E%87%E9%A3%8E%E9%99%A9%E7%AE%A1%E7%90%86)的效果，提高[经济效益](http://wiki.mbalib.com/wiki/%E7%BB%8F%E6%B5%8E%E6%95%88%E7%9B%8A)。

###### (二)利率互换有利于利率市场化的推进

　　由于长期以来利率的制定完全取决于[人民银行](http://wiki.mbalib.com/wiki/%E4%BA%BA%E6%B0%91%E9%93%B6%E8%A1%8C)，本币存贷款利率是由[人民银行](http://wiki.mbalib.com/wiki/%E4%BA%BA%E6%B0%91%E9%93%B6%E8%A1%8C)制定的，外币存贷款利率是人民银行委托[中国银行](http://wiki.mbalib.com/wiki/%E4%B8%AD%E5%9B%BD%E9%93%B6%E8%A1%8C)制定的。其他银行都采用“盯住”的方式在基准利率上下小幅波动，利率的完全浮动将对商业银行的业务经营产生难以预料的影响。利率市场化将加剧同业问的竞争，冲击传统业务，缩小存贷款利差，加大各类利率风险的暴露，减少净利息收入。

　　利率市场化将使[利率风险管理](http://wiki.mbalib.com/wiki/%E5%88%A9%E7%8E%87%E9%A3%8E%E9%99%A9%E7%AE%A1%E7%90%86)成为各商业银行竞争的关键和焦点。而利率互换则是各商业银行规避利率风险，实现资产保值增值的重要手段，若预期利率上升，浮动利率负债的交易方，为避免利率上升带来的增加融资成本的损失，就可以与负债数额相同的固定利率和交易者进行互换，所收的浮动利率与原负债相抵，而仅支出固定利率，从而避免利率上升而带来的增加融资成本的风险。同样，对固定利率的借款者而言，若预期利率下降，为享受利率下降带来融资成本降低的好处，也可以将固定利率转换成浮动利率。

###### (三)利率互换有利于推动商业银行参与国际竞争

[经济全球化](http://wiki.mbalib.com/wiki/%E7%BB%8F%E6%B5%8E%E5%85%A8%E7%90%83%E5%8C%96)、[金融一体化](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E4%B8%80%E4%BD%93%E5%8C%96)的潮流必然使中国经济、金融日益国际化。我国的[外贸依存度](http://wiki.mbalib.com/wiki/%E5%A4%96%E8%B4%B8%E4%BE%9D%E5%AD%98%E5%BA%A6)持续提高，目前已经超过80％ ；[金融开放](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E5%BC%80%E6%94%BE)度不断提高，1996就实现了人民币[经常项目](http://wiki.mbalib.com/wiki/%E7%BB%8F%E5%B8%B8%E9%A1%B9%E7%9B%AE)下的可兑换，2005年实施了以市场供求为基础，参考“[一揽子货币](http://wiki.mbalib.com/wiki/%E4%B8%80%E6%8F%BD%E5%AD%90%E8%B4%A7%E5%B8%81)”的[浮动汇率制](http://wiki.mbalib.com/wiki/%E6%B5%AE%E5%8A%A8%E6%B1%87%E7%8E%87%E5%88%B6)。[外资金融机构](http://wiki.mbalib.com/wiki/%E5%A4%96%E8%B5%84%E9%87%91%E8%9E%8D%E6%9C%BA%E6%9E%84)已经大量进入中国市场，部分已获准经营人民币业务。我国[利用外资](http://wiki.mbalib.com/wiki/%E5%88%A9%E7%94%A8%E5%A4%96%E8%B5%84)规模连续居世界前列。随着我国外债规模的不断扩大，[外债](http://wiki.mbalib.com/wiki/%E5%A4%96%E5%80%BA)利率风险骤增。[美国全国期货协会](http://wiki.mbalib.com/wiki/%E7%BE%8E%E5%9B%BD%E5%85%A8%E5%9B%BD%E6%9C%9F%E8%B4%A7%E5%8D%8F%E4%BC%9A)主席[汉森](http://wiki.mbalib.com/wiki/%E6%B1%89%E6%A3%AE)曾经指出：“中国对其[外汇贷款](http://wiki.mbalib.com/wiki/%E5%A4%96%E6%B1%87%E8%B4%B7%E6%AC%BE)不进行[套期保值](http://wiki.mbalib.com/wiki/%E5%A5%97%E6%9C%9F%E4%BF%9D%E5%80%BC)而带来的损失，已经远远超过了中国过去在期货交易中受到的损失。”因此，如果我们不加紧进入国际金融衍生品市场，加快推进国内[金融衍生品市场](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E8%A1%8D%E7%94%9F%E5%93%81%E5%B8%82%E5%9C%BA)建设，是要付出惨痛代价的。

###### (四)有利于防范金融风险、维护[金融稳定](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E7%A8%B3%E5%AE%9A)

　　目前，我国金融市场规模持续扩大，交易产品和交易工具不断增加。截至2007年末，我国债券市场存量达12．3万亿元，比2006年增长33．2％ ；债券总成交量达到62．6万亿元，比2006年增长63％ ；累计债券发行量为41 805．03亿元，比2006年增长40％。同时，我国商业银行和[政策性银行](http://wiki.mbalib.com/wiki/%E6%94%BF%E7%AD%96%E6%80%A7%E9%93%B6%E8%A1%8C)[资产负债表](http://wiki.mbalib.com/wiki/%E8%B5%84%E4%BA%A7%E8%B4%9F%E5%80%BA%E8%A1%A8)上都积累了巨大的利率风险需要对冲。以[固定利率债券](http://wiki.mbalib.com/wiki/%E5%9B%BA%E5%AE%9A%E5%88%A9%E7%8E%87%E5%80%BA%E5%88%B8)的平均持有期4．4年测算，利率每上升l％ ，商业银行[固定利率债券](http://wiki.mbalib.com/wiki/%E5%9B%BA%E5%AE%9A%E5%88%A9%E7%8E%87%E5%80%BA%E5%88%B8)投资市值将损失1 200亿元。作为规避债券利率风险的强烈需求，而利率互换是实现这个目标最有效的工具。

##### 二、商业银行拓展利率互换业务的对策

###### (一)降低[市场准入门槛](http://wiki.mbalib.com/wiki/%E5%B8%82%E5%9C%BA%E5%87%86%E5%85%A5%E9%97%A8%E6%A7%9B)

　　2004年，[银监会](http://wiki.mbalib.com/wiki/%E9%93%B6%E7%9B%91%E4%BC%9A)发布实施了《金融机构衍生产品交易管理暂行办法》(以下简称《办法》)，用于监管[银监会](http://wiki.mbalib.com/wiki/%E9%93%B6%E7%9B%91%E4%BC%9A)管理范围内金融机构的衍生产品交易业务。银监会在起草这个《办法》的时候，人民币衍生产品市场尚未建立，金融机构的衍生产品交易业务仅限于外币。《办法》虽然考虑到了将来人民币衍生产品市场发展的需要，但是对建立人民币衍生产品市场后金融机构对衍生产品的需求估计不足，限制了绝大多数金融机构开展人民币衍生产品交易业务。银监会的《办法》颁布后，中资银行中仅有十几家(四大商业银行、[国开行](http://wiki.mbalib.com/wiki/%E5%9B%BD%E5%BC%80%E8%A1%8C)和[进出口行](http://wiki.mbalib.com/wiki/%E8%BF%9B%E5%87%BA%E5%8F%A3%E8%A1%8C)、部分全国性[股份制银行](http://wiki.mbalib.com/wiki/%E8%82%A1%E4%BB%BD%E5%88%B6%E9%93%B6%E8%A1%8C))获得了开展业务的资格的金融机构，意味着他们既不能开展衍生产品造市，投机性交易业务，也不能作为衍生产品的最终用户，以避险为目的开展衍生产品交易业务。

　　在利率市场化的形势下，金融市场所有参与者都要承担利率波动的风险，不允许这些中小金融机构以利率避险为目的进行衍生产品交易对这些机构是不公平的，也不利于提高[市场效率](http://wiki.mbalib.com/wiki/%E5%B8%82%E5%9C%BA%E6%95%88%E7%8E%87)。因此，银行监管部门根据人民币衍生产品[市场监管](http://wiki.mbalib.com/wiki/%E5%B8%82%E5%9C%BA%E7%9B%91%E7%AE%A1)规则，把衍生产品市场的参与者明确分为“中介”和“最终用户”两类，允许所有市场参与者作为衍生产品的“最终用户”，以避险为目的开展衍生产品交易业务。

　　对于[证券](http://wiki.mbalib.com/wiki/%E8%AF%81%E5%88%B8)、[保险](http://wiki.mbalib.com/wiki/%E4%BF%9D%E9%99%A9)、基金等机构的衍生产品交易业务，目前没有明确的说法。为了使这些机构适应利率市场化的要求，也应该允许他们作为衍生产品的“最终用户”，以避险为目的开展衍生产品交易，对于条件好的券商，应鼓励他们创造条件，开展中介业务。

###### (二)提高利率互换的流动性

　　影响互换的[市场流动性](http://wiki.mbalib.com/wiki/%E5%B8%82%E5%9C%BA%E6%B5%81%E5%8A%A8%E6%80%A7)因素主要有两个：一是市场买方和卖方严重不平衡。买方是众多的商业银行，而卖方主要是[国家开发银行](http://wiki.mbalib.com/wiki/%E5%9B%BD%E5%AE%B6%E5%BC%80%E5%8F%91%E9%93%B6%E8%A1%8C)一家。“一对多”的[市场结构](http://wiki.mbalib.com/wiki/%E5%B8%82%E5%9C%BA%E7%BB%93%E6%9E%84)使做市商难以开展业务。二是做市商套期手段和工具不足。由于没有债券和[利率期货](http://wiki.mbalib.com/wiki/%E5%88%A9%E7%8E%87%E6%9C%9F%E8%B4%A7)、不允许卖空债券、[货币市场](http://wiki.mbalib.com/wiki/%E8%B4%A7%E5%B8%81%E5%B8%82%E5%9C%BA)和债券市场基础工具流动性不高，使得做市商难以及时进行[套期交易](http://wiki.mbalib.com/wiki/%E5%A5%97%E6%9C%9F%E4%BA%A4%E6%98%93)，不得不承担利率敞口风险。这两个因素严重制约了做市商开展[做市业务](http://wiki.mbalib.com/wiki/%E5%81%9A%E5%B8%82%E4%B8%9A%E5%8A%A1)。为提高流动性，可以从以下几个方面开展工作：一是引入[保险](http://wiki.mbalib.com/wiki/%E4%BF%9D%E9%99%A9)、基金等[非银行机构](http://wiki.mbalib.com/wiki/%E9%9D%9E%E9%93%B6%E8%A1%8C%E6%9C%BA%E6%9E%84)进入互换市场、创造不同的需求，改变市场“一对多”的结构；二是尽快推出融券业务，为做市商套期提供工具；三是探索与互换市场“一对多”结构特点相适应的交易方式，提高市场效率。

###### (三)加快培育货币市场

　　货币市场利率指数影响利率互换市场发展，即没有一个适当的货币市场利率指数，既能反映货币市场利率水平，又有大量的[资本市场工具](http://wiki.mbalib.com/wiki/%E8%B5%84%E6%9C%AC%E5%B8%82%E5%9C%BA%E5%B7%A5%E5%85%B7)，包括长期浮动利率债券和长期浮动利率贷款的利率与之挂钩。目前，货币市场流动性最高的交易品种是隔夜和7天期回购，其利率指数水平比较客观地反映了货币市场的资金供求状况。

　　但是，长期浮动利率贷款和浮动利率债券的利率都不与[回购利率](http://wiki.mbalib.com/wiki/%E5%9B%9E%E8%B4%AD%E5%88%A9%E7%8E%87)挂钩，因此，若用[回购利率](http://wiki.mbalib.com/wiki/%E5%9B%9E%E8%B4%AD%E5%88%A9%E7%8E%87)指数作为利率互换的参考利率，其应用受很大限制。在目前情况下，主要是那些在债券市场进行交易套利和在人民币[远期外汇市场](http://wiki.mbalib.com/wiki/%E8%BF%9C%E6%9C%9F%E5%A4%96%E6%B1%87%E5%B8%82%E5%9C%BA)进行造市、投机和[套利交易](http://wiki.mbalib.com/wiki/%E5%A5%97%E5%88%A9%E4%BA%A4%E6%98%93)的金融机构有实现需求。

　　互换市场的发展迫切需要加快培育货币市场，形成有足够流动性和浓度的货币市场，形成期限3个月或6个月的货币市场利率指数，满足浮动利率债券和浮动利率贷款等[资本市场工具](http://wiki.mbalib.com/wiki/%E8%B5%84%E6%9C%AC%E5%B8%82%E5%9C%BA%E5%B7%A5%E5%85%B7)的需要，这样才能从根本上解决利率互换浮动端参考利率的问题。今后还要从几个方面着手发展货币市场：一是银行在[银行间市场](http://wiki.mbalib.com/wiki/%E9%93%B6%E8%A1%8C%E9%97%B4%E5%B8%82%E5%9C%BA)发行记账式[商业票据](http://wiki.mbalib.com/wiki/%E5%95%86%E4%B8%9A%E7%A5%A8%E6%8D%AE)和大额存单。[商业票据](http://wiki.mbalib.com/wiki/%E5%95%86%E4%B8%9A%E7%A5%A8%E6%8D%AE)或大额存单的发行制度应适应货币市场的要求。简化发行和上市程序，允许以[询价](http://wiki.mbalib.com/wiki/%E8%AF%A2%E4%BB%B7)方式发行。通过一段时问的培育，由中介机构编制期限3个月或6个月的[票据市场](http://wiki.mbalib.com/wiki/%E7%A5%A8%E6%8D%AE%E5%B8%82%E5%9C%BA)利率指数，这个指数代表了主要银行在货币市场融资的成本，可以作为浮动利率债券、浮动利率贷款的参考利率，也就具备了作为利率互换浮动端参考利率的条件。这种做法在[新兴市场](http://wiki.mbalib.com/wiki/%E6%96%B0%E5%85%B4%E5%B8%82%E5%9C%BA)国家和我国台湾地区都有成熟的经验。二是建立货币市场[做市商制度](http://wiki.mbalib.com/wiki/%E5%81%9A%E5%B8%82%E5%95%86%E5%88%B6%E5%BA%A6)，由具备实力的银行向市场提供期限1年以内拆借／回购双边报价，承担维持流动性的义务，提高货币市场资金配置和[定价效率](http://wiki.mbalib.com/wiki/%E5%AE%9A%E4%BB%B7%E6%95%88%E7%8E%87)。

###### (四)完善利率互换产品

　　首先，逐步放开银行间开展外币对人民币的互换业务是互换市场的主要途径。不同币种之间的互换多数混合了利率互换，商业银行可以从中摸索利率互换方面的经验，培育客户群，培养互换方面的专业人才。目前，在香港等地外币对人民币的掉期业务量已经达到相当的规模，一些国内银行实际上已经开展了这项业务，最近银监会正式批准了中国商业银行试点外币对人民币的掉期业务，相信将有更多的商业银行获准这项业务资格。

　　其次，积极开办[债券互换](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8%E4%BA%92%E6%8D%A2)和贷款互换业务。如前所述，我国开展[债券互换](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8%E4%BA%92%E6%8D%A2)的条件已经基本成熟，债券市场规模较为适度。债券持有者可以利用债券互换规避所持有债券的利率风险，商业银行可以尝试用债券互换来规避固定利率贷款的利率风险，这样，可以促进商业银行的固定利率贷款业务；当固定利率货款达到一定规模后，就可以逐渐开展商业银行贷款间及[商业银行借款](http://wiki.mbalib.com/wiki/%E5%95%86%E4%B8%9A%E9%93%B6%E8%A1%8C%E5%80%9F%E6%AC%BE)问的利率互换，进而推动整个利率互换业务的发展。

###### (五)规避商业银行利率互换业务的风险

　　任何事物都有其两面性，利率互换作为一种衍生工具，更是如此。它的产生一方面给人们提供了新的有效的风险转移工具，另一方面也带来了新的风险，应进行正确管理。利率互换交易的风险主要来自于两个方面，一是内部风险，二是外部风险。内部风险包括因交易决策部门对[市场预测](http://wiki.mbalib.com/wiki/%E5%B8%82%E5%9C%BA%E9%A2%84%E6%B5%8B)不当，导致[决策风险](http://wiki.mbalib.com/wiki/%E5%86%B3%E7%AD%96%E9%A3%8E%E9%99%A9)；因[交易员](http://wiki.mbalib.com/wiki/%E4%BA%A4%E6%98%93%E5%91%98)操作错误，导致的[操作风险](http://wiki.mbalib.com/wiki/%E6%93%8D%E4%BD%9C%E9%A3%8E%E9%99%A9)；因交易[会计报表](http://wiki.mbalib.com/wiki/%E4%BC%9A%E8%AE%A1%E6%8A%A5%E8%A1%A8)资料不全、提示度不高，导致的[会计风险](http://wiki.mbalib.com/wiki/%E4%BC%9A%E8%AE%A1%E9%A3%8E%E9%99%A9)等。外部风险包括交易违约的信用风险，金融标的价格不利变动的价格风险，市场供求失衡、交易不畅导致的[流动性风险](http://wiki.mbalib.com/wiki/%E6%B5%81%E5%8A%A8%E6%80%A7%E9%A3%8E%E9%99%A9)，[清算](http://wiki.mbalib.com/wiki/%E6%B8%85%E7%AE%97)故障或[清算价值](http://wiki.mbalib.com/wiki/%E6%B8%85%E7%AE%97%E4%BB%B7%E5%80%BC)变动导致的[清算风险](http://wiki.mbalib.com/wiki/%E6%B8%85%E7%AE%97%E9%A3%8E%E9%99%A9)，以及法规不明或交易不受法律保障而导致的法律风险等。

　　商业银行规避利率互换风险的主要措施有：一是建立和完善人民银行监测体系，提高市场化利率水平透明度。建立对金融机构法人利率市场化微观机制建设情况的备案、评估制度，为稳步推进利率市场化改革提供政策建议，建立和完善市场利率水平定期公布制度，引导金融机构理性确定产品价格，维护利率竞争秩序，建立对[民间借贷利率](http://wiki.mbalib.com/wiki/%E6%B0%91%E9%97%B4%E5%80%9F%E8%B4%B7%E5%88%A9%E7%8E%87)的监测和调研制度，为完善市场利率调控体系服务。

　　二是引导和督促金融机构尽快完善利率定价和利率风险管理机制。以收益与[风险成本](http://wiki.mbalib.com/wiki/%E9%A3%8E%E9%99%A9%E6%88%90%E6%9C%AC)相对称原则为基础，不断完善[经营管理制度](http://wiki.mbalib.com/wiki/%E7%BB%8F%E8%90%A5%E7%AE%A1%E7%90%86%E5%88%B6%E5%BA%A6)，逐步提高利率定价和风险防范能力。通过完善[公司治理结构](http://wiki.mbalib.com/wiki/%E5%85%AC%E5%8F%B8%E6%B2%BB%E7%90%86%E7%BB%93%E6%9E%84)，真正确立受[资本充足率](http://wiki.mbalib.com/wiki/%E8%B5%84%E6%9C%AC%E5%85%85%E8%B6%B3%E7%8E%87)约束的[利润最大化](http://wiki.mbalib.com/wiki/%E5%88%A9%E6%B6%A6%E6%9C%80%E5%A4%A7%E5%8C%96)[经营目标](http://wiki.mbalib.com/wiki/%E7%BB%8F%E8%90%A5%E7%9B%AE%E6%A0%87)；加快建立利率定价管理及利率风险管理体系和[风险溢价](http://wiki.mbalib.com/wiki/%E9%A3%8E%E9%99%A9%E6%BA%A2%E4%BB%B7)评测体系，及时为贷款定价提供基础数据；同时通过建立[内部转移价格](http://wiki.mbalib.com/wiki/%E5%86%85%E9%83%A8%E8%BD%AC%E7%A7%BB%E4%BB%B7%E6%A0%BC)，培育[正向激励](http://wiki.mbalib.com/wiki/%E6%AD%A3%E5%90%91%E6%BF%80%E5%8A%B1)机制为利率定价和风险管理机制提供技术支持。

　　三是加强金融机构利率定价与利率风险管理培训，提高对利率互换风险的认识和管理水平。利率互换交易工作，既为规避和对冲利率风险提供了一个有效工具，也有利于整个金融市场的稳定，是“加快实现利率市场化环境，打破我国汇率市场改革的[瓶颈](http://wiki.mbalib.com/wiki/%E7%93%B6%E9%A2%88)”的[一箭双雕](http://wiki.mbalib.com/wiki/%E4%B8%80%E7%AE%AD%E5%8F%8C%E9%9B%95)之举，但应充分认知其风险规避，应进一步加强制度和监管上的约束，通过采取诸如对发行资格、市场参与资格、以及交易种类严格控制和审查以趋利避害，达到促进利率互换市场健康稳步发展的目的。

## FRA-远期利率协议

### 远期利率协议概述

　　远期利率协议是一种[远期合约](http://wiki.mbalib.com/wiki/%E8%BF%9C%E6%9C%9F%E5%90%88%E7%BA%A6)，买卖双方([客户](http://wiki.mbalib.com/wiki/%E5%AE%A2%E6%88%B7)与[银行](http://wiki.mbalib.com/wiki/%E9%93%B6%E8%A1%8C)或两个银行同业之间)商定将来一定时间点(指[利息](http://wiki.mbalib.com/wiki/%E5%88%A9%E6%81%AF)起算日)开始的一定期限的[协议利率](http://wiki.mbalib.com/w/index.php?title=%E5%8D%8F%E8%AE%AE%E5%88%A9%E7%8E%87&action=edit)，并规定以何种[利率](http://wiki.mbalib.com/wiki/%E5%88%A9%E7%8E%87)为参照利率，在将来利息起算日，按规定的协议利率、期限和本金额，由当事人一方向另一方支付协议利率与参照利率利息差的贴现额。

　　在这种协议下，交易双方约定从将来某一确定的日期开始在某一特定的时期内借贷一笔利率固定、数额确定，以具体[货币](http://wiki.mbalib.com/wiki/%E8%B4%A7%E5%B8%81)表示的[名义本金](http://wiki.mbalib.com/wiki/%E5%90%8D%E4%B9%89%E6%9C%AC%E9%87%91)。远期利率协议的买方就是名义借款人，如果[市场利率](http://wiki.mbalib.com/wiki/%E5%B8%82%E5%9C%BA%E5%88%A9%E7%8E%87)上升的话，他按协议上确定的利率支付利息，就避免了[利率风险](http://wiki.mbalib.com/wiki/%E5%88%A9%E7%8E%87%E9%A3%8E%E9%99%A9)；但若市场利率下跌的话，他仍然必须按协议利率支付利息，就会受到损失。远期利率协议的卖方就是名义贷款人，他按照协议确定的利率收取利息，显然，若[市场利率](http://wiki.mbalib.com/wiki/%E5%B8%82%E5%9C%BA%E5%88%A9%E7%8E%87)下跌，他将受益；若市场利率上升，他则受损。

### 远期利率协议的功能

　　通过固定将来实际交付的利率而避免了利率变动的[风险](http://wiki.mbalib.com/wiki/%E9%A3%8E%E9%99%A9)

　　利率用[利差](http://wiki.mbalib.com/wiki/%E5%88%A9%E5%B7%AE)[结算](http://wiki.mbalib.com/wiki/%E7%BB%93%E7%AE%97)，资金流动量小，为银行提供了一种管理利率风险而又无需改变资产[负债结构](http://wiki.mbalib.com/wiki/%E8%B4%9F%E5%80%BA%E7%BB%93%E6%9E%84)的有效工具

　　远期利率协议具有简便、灵活、不须支付[保证金](http://wiki.mbalib.com/wiki/%E4%BF%9D%E8%AF%81%E9%87%91)等优点

### 远期利率协议的常见术语

* 合同金额——借贷的[名义本金额](http://wiki.mbalib.com/wiki/%E5%90%8D%E4%B9%89%E6%9C%AC%E9%87%91%E9%A2%9D)
* [合同货币](http://wiki.mbalib.com/wiki/%E5%90%88%E5%90%8C%E8%B4%A7%E5%B8%81)——货币币种
* 交易日——远期利率协议成交的日期
* 结算日——名义借贷开始日期
* 确定日——确定参照利率的日期
* 到期日——合约结束之日
* 合约期——结算日至到期日的天数
* 合约利率——协议中双方商定的利率
* 参照利率——某种市场利率

### FRA的价格与报价

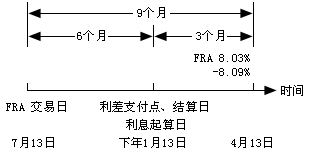
　　FRA的价格是指从利息起算日开始的一定期限的协议利率，FRA的报价方式和[货币市场](http://wiki.mbalib.com/wiki/%E8%B4%A7%E5%B8%81%E5%B8%82%E5%9C%BA)拆出拆入利率表达方式类似，但FRA报价多了合约指定的协议利率期限。具体FRA行情可通过路透终端机的“FRAT”画面得到。FRA市场定价是每天随着市场变化而变化的，实际交易的价格要由每个报价银行来决定。

**FRA市场报价举例**

|  |  |  |
| --- | --- | --- |
| 7月13日 | [美元](http://wiki.mbalib.com/wiki/%E7%BE%8E%E5%85%83) | FRA |
|  | 3×6 | 8.08‰～8.14‰ |
|  | 2×8 | 8.16‰～8.22‰ |
|  | 6×9 | 8.03‰～8.09‰ |
|  | 6×12 | 8.17‰～8.23‰ |

　　　　表1

　　表1报价第三行“6×9、8.03％～8.09％”的市场术语作如下解释：“6×9”(6个月对9个月，英语称为six against nine)是表示期限，即从交易日(7月13日)起6个月末(即次年1月13日)为[起息日](http://wiki.mbalib.com/wiki/%E8%B5%B7%E6%81%AF%E6%97%A5)，而交易日后的9个月末为到期日，协议利率的期限为3个月期。它们之间的时间关系参见图2。

[](http://wiki.mbalib.com/wiki/Image:%E8%BF%9C%E6%9C%9F%E5%88%A9%E7%8E%87%E5%8D%8F%E8%AE%AE%E6%8A%A5%E4%BB%B7%E5%9B%BE2.jpg)

　　“8.03％~8.09％”为报价方报出的FRA买卖价：前者是报价银行的买价，若与[询价](http://wiki.mbalib.com/wiki/%E8%AF%A2%E4%BB%B7)方成交，则意味着报价银行(买方)在结算日支付8.03％利率给询价方(卖方)，并从询价方处收取参照利率。后者是报价银行的卖价，若与询价方成交，则意味着报价银行(卖方)在结算日从询价方(买方)处收取8.09％利率，并支付参照利率给询价方。

### FRA的利息计算

　　在起息日如何支付利息，可按以下步骤进行：

　　首先，计算FRA协议期限内利息差。该利息差就是根据当天参照利率(通常是在结算日前两个营业日使用[LIBOR](http://wiki.mbalib.com/wiki/LIBOR)来决定结算日的参照利率)与协议利率结算利息差，其计算方法与货币市场计算利息的惯例相同，等于本金额X利率差X期限(年)。

　　其次，要注意的是，按惯例，FRA差额的支付是在协议期限的期初(即利息起算日)，而不是协议利率到期日的最后一日，因此利息起算日所交付的差额要按参照利率[贴现](http://wiki.mbalib.com/wiki/%E8%B4%B4%E7%8E%B0)方式计算。

　　最后，计算的A有正有负，当A＞0时，由FRA的卖方将利息差贴现值付给FRA的买方；当A＜0时，则由FRA的买方将利息差贴现值付给FRA的卖方。

[[编辑](http://wiki.mbalib.com/w/index.php?title=%E8%BF%9C%E6%9C%9F%E5%88%A9%E7%8E%87%E5%8D%8F%E8%AE%AE&action=edit&section=6)]

### 远期利率协议结算金的计算[[1]](http://wiki.mbalib.com/wiki/%E8%BF%9C%E6%9C%9F%E5%88%A9%E7%8E%87%E5%8D%8F%E8%AE%AE#_note-0)

　　结算金=\frac{(r_r-r_k)\times A\times\frac{D}{B}}{1+r_r\times\frac{D}{B}}

*rr*=参照利率

*rk*=合约利率

A=合约金额

D=合约期间

B=年基准

**例题**

　　2006年4月10日，某财务公司经理预测从2006年6月16日到9月15日的3个月（92天）的远期资金需求，他认为，利率可能上升，因此，他想[对冲](http://wiki.mbalib.com/wiki/%E5%AF%B9%E5%86%B2)利率上升的风险，便于4月10日从[中国银行](http://wiki.mbalib.com/wiki/%E4%B8%AD%E5%9B%BD%E9%93%B6%E8%A1%8C)买进远期利率协议。

**条件：**

* 合约金额：10000000元
* [交易日](http://wiki.mbalib.com/wiki/%E4%BA%A4%E6%98%93%E6%97%A5)：2006年4月10日
* 结算日：2006年6月16日
* 到期日：2006年9月15日
* 合约年利率：6.75%
* 年基准：360天

　　如果在结算日6月16日的3个月全国[银行业](http://wiki.mbalib.com/wiki/%E9%93%B6%E8%A1%8C%E4%B8%9A)[同业拆借利率](http://wiki.mbalib.com/wiki/%E5%90%8C%E4%B8%9A%E6%8B%86%E5%80%9F%E5%88%A9%E7%8E%87)（参考利率）为7.25%，高于合约利率，则按照远期利率协议银行须补偿公司一定量的[现金](http://wiki.mbalib.com/wiki/%E7%8E%B0%E9%87%91)，运用上面的公式计算支付金额

　　结算金= \frac{(7.25%-6.75%)\times 10000000\times \frac{92}{360}}{1+7.25%\times \frac{92}{360})}=12545.34（元）

　　至此，远期利率协议就终止了，该公司可以将[借款成本](http://wiki.mbalib.com/wiki/%E5%80%9F%E6%AC%BE%E6%88%90%E6%9C%AC)锁定在6.75%。

### 远期利率协议的价值

　　考虑时刻 t 的两个远期利率协议，它们的名义本金均为A，约定的未来期限均为*T* \* − *T*，第一个FRA的协议利率采用市场远期利率*rF*，第二个FRA的协议利率为*rK*。显然，这两个FRA之间的唯一不同就是*T* \* 时刻的利息支付。换句话说，t时刻第二个FRA与第一个FRA的[价值差异](http://wiki.mbalib.com/wiki/%E4%BB%B7%E5%80%BC%E5%B7%AE%E5%BC%82)就是T^\*时刻不同利息支付的现值

\left(Ae^{r_K(T^*-T)-Ae^{r_F(T^*-T)}}\right)e^{-r^*(T^*-t)}　　　　　　　（1）

　　由于第一个FRA中的协议利率为理论远期利率，其远期价值应为零。则第二个FRA的价值就等于式（1）。

　　式（1）适合于任何协议利率为的远期利率协议价值的计算。

### 利用FRA进行风险管理的技巧

　　FRA是防范将来利率变动风险的一种[金融工具](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E5%B7%A5%E5%85%B7)，其特点是预先锁定将来的利率。在FRA市场中，FRA的买方是为了防止利率上升引起[筹资成本](http://wiki.mbalib.com/wiki/%E7%AD%B9%E8%B5%84%E6%88%90%E6%9C%AC)上升的风险，希望在现在就锁定将来的筹资成本。用FRA防范将来利率变动的风险，实质上是用FRA市场的盈亏抵补现货资金市场的风险，因此FRA具有预先决定筹资成本或预先决定[投资报酬率](http://wiki.mbalib.com/wiki/%E6%8A%95%E8%B5%84%E6%8A%A5%E9%85%AC%E7%8E%87)的功能。

[[编辑](http://wiki.mbalib.com/w/index.php?title=%E8%BF%9C%E6%9C%9F%E5%88%A9%E7%8E%87%E5%8D%8F%E8%AE%AE&action=edit&section=9)]

### FRA与利率期货的联系与区别

　　从形式上看，FRA具有[利率期货](http://wiki.mbalib.com/wiki/%E5%88%A9%E7%8E%87%E6%9C%9F%E8%B4%A7)类似的优点，即避免利率变动风险的功能，但它们之间也有区别，归纳如下表所示。

|  |  |  |
| --- | --- | --- |
|  | **FRA** | **利率期货** |
| 交易形态 | [场外交易市场](http://wiki.mbalib.com/wiki/%E5%9C%BA%E5%A4%96%E4%BA%A4%E6%98%93%E5%B8%82%E5%9C%BA)成交，交易金额和交割日期都不受限制，灵活简便 | 交易所内交易，标准化契约交易 |
| [信用风险](http://wiki.mbalib.com/wiki/%E4%BF%A1%E7%94%A8%E9%A3%8E%E9%99%A9) | 双方均存在信用风险 | 信用风险极小 |
| 交割前的[现金流](http://wiki.mbalib.com/wiki/%E7%8E%B0%E9%87%91%E6%B5%81) | 不发生现金流 | 每日[保证金账户](http://wiki.mbalib.com/wiki/%E4%BF%9D%E8%AF%81%E9%87%91%E8%B4%A6%E6%88%B7)内有现金净流动 |
| 适用货币 | 一切可兑换货币 | 交易所规定的货币 |

### 远期外汇综合协议与远期利率协议的区别

[远期外汇综合协议](http://wiki.mbalib.com/wiki/%E8%BF%9C%E6%9C%9F%E5%A4%96%E6%B1%87%E7%BB%BC%E5%90%88%E5%8D%8F%E8%AE%AE)与远期利率协议的最大区别在于：前者的保值或投机目标是两种货币间的利率差以及由此决定的远期差价，后者的目标则是一国利率的绝对水平。

　　但两者也有很多相似之处：

　　1、标价方式都是m×n，其中m表示合同签订日到结算日的时间，n表示合同签订日至到期日的时间。

　　2、两者都有五个时点，即合同签订日、起算日、确定日、结算日、到期日，而且有关规定均相同。

　　3、名义本金均不交换。

## CRMW-信用风险缓释凭证

信用风险缓释凭证（Credit Risk Mitigation Warrant，简称CRMW）

### 什么是信用风险缓释凭证

　　根据2010年10月29日，[中国银行间市场交易商协会](http://wiki.mbalib.com/wiki/%E4%B8%AD%E5%9B%BD%E9%93%B6%E8%A1%8C%E9%97%B4%E5%B8%82%E5%9C%BA%E4%BA%A4%E6%98%93%E5%95%86%E5%8D%8F%E4%BC%9A)发布的《[银行间市场](http://wiki.mbalib.com/wiki/%E9%93%B6%E8%A1%8C%E9%97%B4%E5%B8%82%E5%9C%BA)信用风险缓释工具试点业务指引》第十条，**信用风险缓释凭证**，是指由标的实体以外的机构创设的，为凭证[持有人](http://wiki.mbalib.com/wiki/%E6%8C%81%E6%9C%89%E4%BA%BA)就标的[债务](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%8A%A1)提供信用风险保护的，可[交易](http://wiki.mbalib.com/wiki/%E4%BA%A4%E6%98%93)[流通](http://wiki.mbalib.com/wiki/%E6%B5%81%E9%80%9A)的有价凭证。

### 信用风险缓释凭证的实质

　　从[定义](http://wiki.mbalib.com/wiki/%E5%AE%9A%E4%B9%89)和形式上看，信用风险缓释凭证是由凭证创设人面向[投资者](http://wiki.mbalib.com/wiki/%E6%8A%95%E8%B5%84%E8%80%85)“一对多”发行的、可在[二级市场](http://wiki.mbalib.com/wiki/%E4%BA%8C%E7%BA%A7%E5%B8%82%E5%9C%BA)[流通转让](http://wiki.mbalib.com/wiki/%E6%B5%81%E9%80%9A%E8%BD%AC%E8%AE%A9)的、附带[信用](http://wiki.mbalib.com/wiki/%E4%BF%A1%E7%94%A8)保护[权利](http://wiki.mbalib.com/wiki/%E6%9D%83%E5%88%A9)的[证券](http://wiki.mbalib.com/wiki/%E8%AF%81%E5%88%B8)。

　　从实质上看，信用风险缓释凭证是[标准化](http://wiki.mbalib.com/wiki/%E6%A0%87%E5%87%86%E5%8C%96)的[信用风险缓释合约](http://wiki.mbalib.com/wiki/%E4%BF%A1%E7%94%A8%E9%A3%8E%E9%99%A9%E7%BC%93%E9%87%8A%E5%90%88%E7%BA%A6)，从而可以在不同投资者之间[买卖](http://wiki.mbalib.com/wiki/%E4%B9%B0%E5%8D%96)[转让](http://wiki.mbalib.com/wiki/%E8%BD%AC%E8%AE%A9)；从[市场运行](http://wiki.mbalib.com/wiki/%E5%B8%82%E5%9C%BA%E8%BF%90%E8%A1%8C)框架上看，信用风险缓释凭证与[债券](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8)类似，有创设登记、发行销售、交易[结算](http://wiki.mbalib.com/wiki/%E7%BB%93%E7%AE%97)、注销等一系列[流程规范](http://wiki.mbalib.com/wiki/%E6%B5%81%E7%A8%8B%E8%A7%84%E8%8C%83)信用风险缓释凭证的[管理](http://wiki.mbalib.com/wiki/%E7%AE%A1%E7%90%86)。

### 信用风险缓释凭证的优势

　　信用风险缓释凭证相对于合约的优势在于，可以通过对信用风险缓释凭证创设实行[市场化](http://wiki.mbalib.com/wiki/%E5%B8%82%E5%9C%BA%E5%8C%96)的登记[制度](http://wiki.mbalib.com/wiki/%E5%88%B6%E5%BA%A6)，实现对信用风险缓释凭证创设规模的[事前控制](http://wiki.mbalib.com/wiki/%E4%BA%8B%E5%89%8D%E6%8E%A7%E5%88%B6)，并能对[市场](http://wiki.mbalib.com/wiki/%E5%B8%82%E5%9C%BA)交易情况进行即时监测，在[风险](http://wiki.mbalib.com/wiki/%E9%A3%8E%E9%99%A9)防范上十分便利。具体而言：

#### 一是符合资质的机构才可创设凭证。

[信用风险](http://wiki.mbalib.com/wiki/%E4%BF%A1%E7%94%A8%E9%A3%8E%E9%99%A9)缓释凭证面向市场[公开发行](http://wiki.mbalib.com/wiki/%E5%85%AC%E5%BC%80%E5%8F%91%E8%A1%8C)，对创设机构的[资本](http://wiki.mbalib.com/wiki/%E8%B5%84%E6%9C%AC)实力、[信用风险管理](http://wiki.mbalib.com/wiki/%E4%BF%A1%E7%94%A8%E9%A3%8E%E9%99%A9%E7%AE%A1%E7%90%86)能力、交易定价能力都有很高的要求，因此，只有符合一定资质条件的机构才具备创设凭证的能力。

　　通过对凭证创设机构提出资质要求，有利于保护凭证持有人的权利，维护市场稳定。在具体的操作上，可以由[银行间市场](http://wiki.mbalib.com/wiki/%E9%93%B6%E8%A1%8C%E9%97%B4%E5%B8%82%E5%9C%BA)[金融衍生产品](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E8%A1%8D%E7%94%9F%E4%BA%A7%E5%93%81)专业委员会（简称专业委员会）提出和制定资质要求，由[交易商](http://wiki.mbalib.com/wiki/%E4%BA%A4%E6%98%93%E5%95%86)协会秘书处进行备案认可，从而实现市场化的[自律](http://wiki.mbalib.com/wiki/%E8%87%AA%E5%BE%8B)[管理模式](http://wiki.mbalib.com/wiki/%E7%AE%A1%E7%90%86%E6%A8%A1%E5%BC%8F)。

#### 二是凭证创设需要登记。

　　信用风险缓释凭证创设可以实行市场化的登记制度，创设机构将凭证说明书、凭证公告等登记[材料](http://wiki.mbalib.com/wiki/%E6%9D%90%E6%96%99)提交交易商协会后，由交易商协会[组织](http://wiki.mbalib.com/wiki/%E7%BB%84%E7%BB%87)银行间市场[金融衍生产品市场](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E8%A1%8D%E7%94%9F%E4%BA%A7%E5%93%81%E5%B8%82%E5%9C%BA)专家召开[会议](http://wiki.mbalib.com/wiki/%E4%BC%9A%E8%AE%AE)，由专家集体做出是否[接受](http://wiki.mbalib.com/wiki/%E6%8E%A5%E5%8F%97)信用风险缓释凭证创设登记的[决定](http://wiki.mbalib.com/wiki/%E5%86%B3%E5%AE%9A)。

　　金融衍生产品专家必须具有丰富的[经济](http://wiki.mbalib.com/wiki/%E7%BB%8F%E6%B5%8E)[金融理论](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E7%90%86%E8%AE%BA)知识和金融衍生产品从业经验，并熟知[相关法](http://wiki.mbalib.com/wiki/%E7%9B%B8%E5%85%B3%E6%B3%95)律法规、享有较高职业声誉，可以通过交易商协会[会员](http://wiki.mbalib.com/wiki/%E4%BC%9A%E5%91%98)推荐，协会[常务理事](http://wiki.mbalib.com/wiki/%E5%B8%B8%E5%8A%A1%E7%90%86%E4%BA%8B)会审议的方式从银行间市场金融衍生产品[从业人员](http://wiki.mbalib.com/wiki/%E4%BB%8E%E4%B8%9A%E4%BA%BA%E5%91%98)中选取。经专家会议审议通过后，交易商协会对信用风险缓释凭证进行登记，创设机构可以开展后续的信用风险缓释凭证发[行销](http://wiki.mbalib.com/wiki/%E8%A1%8C%E9%94%80)售工作。

#### 三是凭证进行统一登记和托管。

　　信用风险缓释凭证发行销售结束后，在[中央结算公司](http://wiki.mbalib.com/wiki/%E4%B8%AD%E5%A4%AE%E7%BB%93%E7%AE%97%E5%85%AC%E5%8F%B8)办理凭证登记、托管手续，然后即可在银行间市场[交易](http://wiki.mbalib.com/wiki/%E4%BA%A4%E6%98%93)[流通](http://wiki.mbalib.com/wiki/%E6%B5%81%E9%80%9A)，具体步骤与[债券](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8)的登记、托管、交易十分相似。

### 信用风险缓释凭证的应用

　　信用风险缓释凭证是第三方机构开发的针对特定[债务](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%8A%A1)的一种[有价证券](http://wiki.mbalib.com/wiki/%E6%9C%89%E4%BB%B7%E8%AF%81%E5%88%B8)，它绑定了特定的债务，能够在[二级市场](http://wiki.mbalib.com/wiki/%E4%BA%8C%E7%BA%A7%E5%B8%82%E5%9C%BA)交易和流通，是和CDS及其类似的[标准化](http://wiki.mbalib.com/wiki/%E6%A0%87%E5%87%86%E5%8C%96)[产品](http://wiki.mbalib.com/wiki/%E4%BA%A7%E5%93%81)。

　　信用风险缓释凭证在登记、托管和[清算制度](http://wiki.mbalib.com/wiki/%E6%B8%85%E7%AE%97%E5%88%B6%E5%BA%A6)上都是集中进行，对[市场](http://wiki.mbalib.com/wiki/%E5%B8%82%E5%9C%BA)的透明度及[杠杆率](http://wiki.mbalib.com/wiki/%E6%9D%A0%E6%9D%86%E7%8E%87)进行[有效控制](http://wiki.mbalib.com/wiki/%E6%9C%89%E6%95%88%E6%8E%A7%E5%88%B6)。

如：某公司发行了5年期[中期票据](http://wiki.mbalib.com/wiki/%E4%B8%AD%E6%9C%9F%E7%A5%A8%E6%8D%AE)，[投资人](http://wiki.mbalib.com/wiki/%E6%8A%95%E8%B5%84%E4%BA%BA)是某[银行](http://wiki.mbalib.com/wiki/%E9%93%B6%E8%A1%8C)，为了对该笔中期票据进行保障，核心[交易商](http://wiki.mbalib.com/wiki/%E4%BA%A4%E6%98%93%E5%95%86)某[保险公司](http://wiki.mbalib.com/wiki/%E4%BF%9D%E9%99%A9%E5%85%AC%E5%8F%B8)开发了一种凭证并[销售](http://wiki.mbalib.com/wiki/%E9%94%80%E5%94%AE)给[其他投资](http://wiki.mbalib.com/wiki/%E5%85%B6%E4%BB%96%E6%8A%95%E8%B5%84)者，该凭证可以在市场中自由流通和[转让](http://wiki.mbalib.com/wiki/%E8%BD%AC%E8%AE%A9)。市场交易中，部分投资人可赚取 中间[差价](http://wiki.mbalib.com/wiki/%E5%B7%AE%E4%BB%B7)，而银行希望降低资本占用以缓释信用风险。如果发行票据的公司不能 按期还款，那么CRMW的[持有人](http://wiki.mbalib.com/wiki/%E6%8C%81%E6%9C%89%E4%BA%BA)就可以直接向保险公司要求赔偿。

## CFAE-北京所

[](http://wiki.mbalib.com/wiki/Image:%E5%8C%97%E9%87%91%E6%89%80.jpg)

北京金融资产交易所(Beijing Financial Assets Exchange; 北金所)

### 什么是北京金融资产交易所

**北京金融资产交易所**（简称**北金所**）是经北京市人民政府批准设立的专业化[金融资产](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E8%B5%84%E4%BA%A7)交易机构，于2010年5月30日正式揭牌[运营](http://wiki.mbalib.com/wiki/%E8%BF%90%E8%90%A5)。

　　北金所是财政部指定的金融类[国有资产](http://wiki.mbalib.com/wiki/%E5%9B%BD%E6%9C%89%E8%B5%84%E4%BA%A7)交易平台以及[中国银行间市场交易商协会](http://wiki.mbalib.com/wiki/%E4%B8%AD%E5%9B%BD%E9%93%B6%E8%A1%8C%E9%97%B4%E5%B8%82%E5%9C%BA%E4%BA%A4%E6%98%93%E5%95%86%E5%8D%8F%E4%BC%9A)的指定交易平台。业务范围涵盖金融企业国有资产交易、[信贷资产交易](http://wiki.mbalib.com/wiki/%E4%BF%A1%E8%B4%B7%E8%B5%84%E4%BA%A7)、[信托产品交易](http://wiki.mbalib.com/wiki/%E4%BF%A1%E6%89%98%E4%BA%A7%E5%93%81)、债券产品交易、[私募股权交易](http://wiki.mbalib.com/wiki/%E7%A7%81%E5%8B%9F%E8%82%A1%E6%9D%83)、[黄金交易](http://wiki.mbalib.com/wiki/%E9%BB%84%E9%87%91%E4%BA%A4%E6%98%93)等，为各类金融资产提供从登记、交易到[结算](http://wiki.mbalib.com/wiki/%E7%BB%93%E7%AE%97)的全程式服务。

北金所采用[有限责任公司](http://wiki.mbalib.com/wiki/%E6%9C%89%E9%99%90%E8%B4%A3%E4%BB%BB%E5%85%AC%E5%8F%B8)的[组织形式](http://wiki.mbalib.com/wiki/%E7%BB%84%E7%BB%87%E5%BD%A2%E5%BC%8F)，由北京产权交易所、信达投资有限公司、中国光大投资公司、北京华融综合投资公司、华能资本服务有限公司共同出资组建。

### 北京金融资产交易所的市场资质

　　北金所是[中国银行间市场交易商协会](http://wiki.mbalib.com/wiki/%E4%B8%AD%E5%9B%BD%E9%93%B6%E8%A1%8C%E9%97%B4%E5%B8%82%E5%9C%BA%E4%BA%A4%E6%98%93%E5%95%86%E5%8D%8F%E4%BC%9A)指定交易平台。根据双方合作协议，交易商协会支持北金所参与金融市场产品创新研发，并利用北金所平台进行创新产品与[融资工具](http://wiki.mbalib.com/wiki/%E8%9E%8D%E8%B5%84%E5%B7%A5%E5%85%B7)的发行交易；同时积极引导、鼓励协会相关会员进入北金所开展相关[金融资产](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E8%B5%84%E4%BA%A7)交易业务。

### 北京金融资产交易所的功能

　　金融是[经济](http://wiki.mbalib.com/wiki/%E7%BB%8F%E6%B5%8E)的血脉。中国经济的蓬勃发展，离不开[金融市场](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E5%B8%82%E5%9C%BA)的支持。中国的金融资产存量已达到150万亿元[人民币](http://wiki.mbalib.com/wiki/%E4%BA%BA%E6%B0%91%E5%B8%81)，但由于流动机制与交易平台的缺乏，这些巨额金融资产中的很大一部分都无法顺畅流动，无法充分助推中国[经济发展](http://wiki.mbalib.com/wiki/%E7%BB%8F%E6%B5%8E%E5%8F%91%E5%B1%95)。

　　北京金融资产交易所的成立，为包括金融[国有资产](http://wiki.mbalib.com/wiki/%E5%9B%BD%E6%9C%89%E8%B5%84%E4%BA%A7)、[金融不良资产](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E4%B8%8D%E8%89%AF%E8%B5%84%E4%BA%A7)、[信贷资产](http://wiki.mbalib.com/wiki/%E4%BF%A1%E8%B4%B7%E8%B5%84%E4%BA%A7)、[信托产品](http://wiki.mbalib.com/wiki/%E4%BF%A1%E6%89%98%E4%BA%A7%E5%93%81)、[私募股权](http://wiki.mbalib.com/wiki/%E7%A7%81%E5%8B%9F%E8%82%A1%E6%9D%83)等在内的各类[金融资产](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E8%B5%84%E4%BA%A7)流动提供了一个有效[渠道](http://wiki.mbalib.com/wiki/%E6%B8%A0%E9%81%93)，它是中国金融市场发展到一定阶段的自然产物，也为推动中国金融市场发展作出了有益探索。

中国正在向世界经济的顶峰坚实迈进，与此相应，中国金融必将迎来[跨越式发展](http://wiki.mbalib.com/wiki/%E8%B7%A8%E8%B6%8A%E5%BC%8F%E5%8F%91%E5%B1%95)。作为中国[金融市场化](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E5%B8%82%E5%9C%BA%E5%8C%96)改革的“试验田”和市场基础设施，北京金融资产交易所将“立足北京、辐射全国、走向世界”，致力打造一个国内一流并具有一定国际影响力的专业化金融资产交易平台，成为推动中国金融资产有效流动和[金融资源](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E8%B5%84%E6%BA%90)合理配置的积极力量。

## CDS-信用违约互换

信用违约互换(credit default swap，CDS)，也称信用违约掉期、信贷违约掉期

### 信用违约互换概述

**信用违约互换(credit default swap，CDS)**是国外[债券市场](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8%E5%B8%82%E5%9C%BA)中最常见的[信用衍生产品](http://wiki.mbalib.com/wiki/%E4%BF%A1%E7%94%A8%E8%A1%8D%E7%94%9F%E4%BA%A7%E5%93%81)。在信用违约互换交易中，违约互换购买者将定期向违约互换出售者支付一定费用(称为信用违约互换[点差](http://wiki.mbalib.com/wiki/%E7%82%B9%E5%B7%AE))，而一旦出现信用类事件(主要指[债券](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8)主体无法偿付)，违约互换购买者将有权利将[债券](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8)以[面值](http://wiki.mbalib.com/wiki/%E9%9D%A2%E5%80%BC)递送给违约互换出售者，从而有效规避[信用风险](http://wiki.mbalib.com/wiki/%E4%BF%A1%E7%94%A8%E9%A3%8E%E9%99%A9)。由于信用违约互换[产品定义](http://wiki.mbalib.com/wiki/%E4%BA%A7%E5%93%81%E5%AE%9A%E4%B9%89)简单、容易实现[标准化](http://wiki.mbalib.com/wiki/%E6%A0%87%E5%87%86%E5%8C%96)，交易简洁，自90年代以来，该[金融产品](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E4%BA%A7%E5%93%81)在国外发达[金融市场](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E5%B8%82%E5%9C%BA)得到了迅速发展。

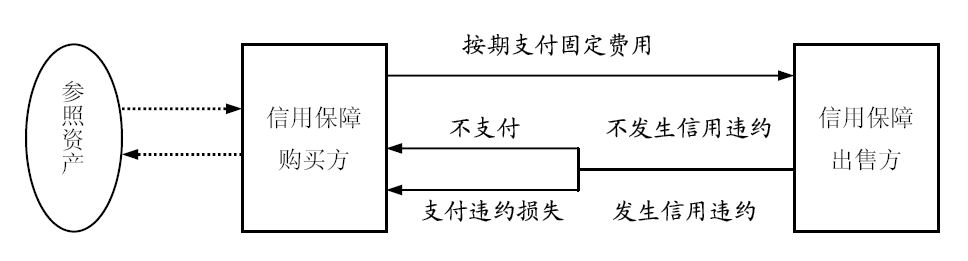
　　对于[投资者](http://wiki.mbalib.com/wiki/%E6%8A%95%E8%B5%84%E8%80%85)，规避信用风险的方法一种是根据[信用评级](http://wiki.mbalib.com/wiki/%E4%BF%A1%E7%94%A8%E8%AF%84%E7%BA%A7)直接要求[信用利差](http://wiki.mbalib.com/wiki/%E4%BF%A1%E7%94%A8%E5%88%A9%E5%B7%AE)，另一种就是购买诸如信用违约互换等[信用衍生品](http://wiki.mbalib.com/wiki/%E4%BF%A1%E7%94%A8%E8%A1%8D%E7%94%9F%E5%93%81)。如果[投资组合](http://wiki.mbalib.com/wiki/%E6%8A%95%E8%B5%84%E7%BB%84%E5%90%88)中[企业债券](http://wiki.mbalib.com/wiki/%E4%BC%81%E4%B8%9A%E5%80%BA%E5%88%B8)发债体较多、行业分布集中度低，则直接要求每只债券一定信用利差即可有效降低组合整体信用风险损失；但如果组合中[企业债券](http://wiki.mbalib.com/wiki/%E4%BC%81%E4%B8%9A%E5%80%BA%E5%88%B8)数目不多、[行业集中度](http://wiki.mbalib.com/wiki/%E8%A1%8C%E4%B8%9A%E9%9B%86%E4%B8%AD%E5%BA%A6)高，不能有效分散信用风险，购买信用违约互换即成为更现实的做法，产生与通过分散资产来降低组合风险的同等作用。

　　由于在购买信用违约互换后，[投资者](http://wiki.mbalib.com/wiki/%E6%8A%95%E8%B5%84%E8%80%85)持有企业债券的信用风险理论上降低为零，我们可以因此认为企业[债券收益](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8%E6%94%B6%E7%9B%8A)(y)、信用违约互换[点差](http://wiki.mbalib.com/wiki/%E7%82%B9%E5%B7%AE)(s)和[无风险利率](http://wiki.mbalib.com/wiki/%E6%97%A0%E9%A3%8E%E9%99%A9%E5%88%A9%E7%8E%87)(r)存在等式关系s=y－r。在具体交易中，[国债收益率](http://wiki.mbalib.com/wiki/%E5%9B%BD%E5%80%BA%E6%94%B6%E7%9B%8A%E7%8E%87)以及[利率互换](http://wiki.mbalib.com/wiki/%E5%88%A9%E7%8E%87%E4%BA%92%E6%8D%A2)(swaprate)收益率都可用作无风险利率，而[交易商](http://wiki.mbalib.com/wiki/%E4%BA%A4%E6%98%93%E5%95%86)对具体企业债券市场报价则[简单表](http://wiki.mbalib.com/wiki/%E7%AE%80%E5%8D%95%E8%A1%A8)述为在同期限无风险利率基础上加上信用违约互换点差水平。

**信用违约互换**诞生于2004年，当时在全球进行[交易](http://wiki.mbalib.com/wiki/%E4%BA%A4%E6%98%93)的[国家](http://wiki.mbalib.com/wiki/%E5%9B%BD%E5%AE%B6)也仅仅限于美国和欧洲，但由于它发展迅速，仅仅三年时间就已经位列全球信用衍生品交易量的第二位，英国银行家协会出版的《British Bankers’Association一一Credit Derivatives Report 2006》显示信用违约指数产品已经成为[信用衍生产品](http://wiki.mbalib.com/wiki/%E4%BF%A1%E7%94%A8%E8%A1%8D%E7%94%9F%E4%BA%A7%E5%93%81)中的第二大类，成为[信用违约掉期](http://wiki.mbalib.com/wiki/%E4%BF%A1%E7%94%A8%E8%BF%9D%E7%BA%A6%E6%8E%89%E6%9C%9F)市场上的热门投资点。

### 信用违约互换结构图

　　信用违约互换是将参照资产的信用风险从信用保障买方转移给信用卖方的交易。信用保障的买方向愿意承担风险保护的保障卖方在合同期限内支付一笔固定的费用；信用保障卖方在接受费用的同时，则承诺在合同期限内，当对应信用违约时，向信用保障的买方赔付违约的损失。对应参照资产的信用可是某一信用，也可是一篮子信用。如果一篮子信用中出现任何一笔违约，信用保障的卖方都必须向对方[赔偿损失](http://wiki.mbalib.com/wiki/%E8%B5%94%E5%81%BF%E6%8D%9F%E5%A4%B1)，其结构如图1所示：

[](http://wiki.mbalib.com/wiki/Image:%E4%BF%A1%E7%94%A8%E8%BF%9D%E7%BA%A6%E4%BA%92%E6%8D%A2_.jpg)

　　图：信用违约互换

### 国内信用违约互换设计

　　目前国内的企业债券几乎都有银行或者有实力机构担保，偿付的风险基本没有，但是这不代表国内就彻底没有信用违约互换出现基础。以下两种情况可促使信用违约互换出现:

　　1)目前的企业债券大多是[长期债券](http://wiki.mbalib.com/wiki/%E9%95%BF%E6%9C%9F%E5%80%BA%E5%88%B8)，横跨不止一个[经济周期](http://wiki.mbalib.com/wiki/%E7%BB%8F%E6%B5%8E%E5%91%A8%E6%9C%9F)，[企业信用](http://wiki.mbalib.com/wiki/%E4%BC%81%E4%B8%9A%E4%BF%A1%E7%94%A8)状况不一定会恒定良好。如果一旦发债企业出现偿付问题，企业债偿付将转由担保机构完成，此时企业债券即由可交易流通的债券转变为失去流动性的担保机构债务。投资者将失去[资产组合](http://wiki.mbalib.com/wiki/%E8%B5%84%E4%BA%A7%E7%BB%84%E5%90%88)操作的灵活性，并将调整资产组合的[风险价值](http://wiki.mbalib.com/wiki/%E9%A3%8E%E9%99%A9%E4%BB%B7%E5%80%BC)或其他[风险指标](http://wiki.mbalib.com/wiki/%E9%A3%8E%E9%99%A9%E6%8C%87%E6%A0%87)。一旦市场利率环境发生变化，投资者也不能通过买卖该企业债来获得收益或规避[利率风险](http://wiki.mbalib.com/wiki/%E5%88%A9%E7%8E%87%E9%A3%8E%E9%99%A9)。所以说，发债企业的信用风险仍会导致投资者损失，尽管它并不带来投资者本金或预期票息收益的损失。

　　2)在国内机构投资实践工作中，[风险控制](http://wiki.mbalib.com/wiki/%E9%A3%8E%E9%99%A9%E6%8E%A7%E5%88%B6)较严的机构往往设有[内部评级](http://wiki.mbalib.com/wiki/%E5%86%85%E9%83%A8%E8%AF%84%E7%BA%A7)，内部评级不高的企业债券该类机构也不会轻易投资，此时如果有信用违约互换的支持，该类机构[投资行为](http://wiki.mbalib.com/wiki/%E6%8A%95%E8%B5%84%E8%A1%8C%E4%B8%BA)可能将有所变化。

　　因此说，尽管目前企业债券本息偿付基本不存在问题，但信用风险仍然会影响机构的[投资收益](http://wiki.mbalib.com/wiki/%E6%8A%95%E8%B5%84%E6%94%B6%E7%9B%8A)和行为，这为信用违约互换的产生提供了基础。国内实力雄厚的大型机构完全可以出售信用违约互换给企业债券投资者，通过创造[信用衍生产品](http://wiki.mbalib.com/wiki/%E4%BF%A1%E7%94%A8%E8%A1%8D%E7%94%9F%E4%BA%A7%E5%93%81)增进其他业务合作，并实实在在形成一笔表外收入，而购买信用违约互换的机构则可以增加资产组合灵活性和资产组合品种多样性。

[短期融资券](http://wiki.mbalib.com/wiki/%E7%9F%AD%E6%9C%9F%E8%9E%8D%E8%B5%84%E5%88%B8)是另一个信用违约互换可以出现的领域。虽然短期融资券期限较短，在如此短时间内发债企业出现偿付风险可能不大，国外对[商业票据市场](http://wiki.mbalib.com/wiki/%E5%95%86%E4%B8%9A%E7%A5%A8%E6%8D%AE%E5%B8%82%E5%9C%BA)的长期跟踪研究也表明这类品种信用风险很小。但随着发债企业数目迅速增多，极少数发债企业突然出现兑付危机的概率还是存在的。因此说在短期融资券领域信用违约互换仍有创造与应用的价值。

　　在国内市场交易大量应用信用违约互换之前，信用违约互换在国内的应用价值可能会在投资银行领域。随着企业债及短期融资券[市场规模](http://wiki.mbalib.com/wiki/%E5%B8%82%E5%9C%BA%E8%A7%84%E6%A8%A1)的扩大，[债券发行](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8%E5%8F%91%E8%A1%8C)体将不仅仅限于实力雄厚的[大型企业](http://wiki.mbalib.com/wiki/%E5%A4%A7%E5%9E%8B%E4%BC%81%E4%B8%9A)，[中小企业](http://wiki.mbalib.com/wiki/%E4%B8%AD%E5%B0%8F%E4%BC%81%E4%B8%9A)也将成为发行主体。由于[国内信用](http://wiki.mbalib.com/wiki/%E5%9B%BD%E5%86%85%E4%BF%A1%E7%94%A8)评级机制发展滞后，市场还不能有效对信用风险进行甄别和[定价](http://wiki.mbalib.com/wiki/%E5%AE%9A%E4%BB%B7)，[承销商](http://wiki.mbalib.com/wiki/%E6%89%BF%E9%94%80%E5%95%86)为[中小企业](http://wiki.mbalib.com/wiki/%E4%B8%AD%E5%B0%8F%E4%BC%81%E4%B8%9A)进行投资银行服务可能会遇到市场需求瓶颈。此时如果[主承销商](http://wiki.mbalib.com/wiki/%E4%B8%BB%E6%89%BF%E9%94%80%E5%95%86)实力雄厚，完全可以通过向[投资人](http://wiki.mbalib.com/wiki/%E6%8A%95%E8%B5%84%E4%BA%BA)出售信用违约互换的方式为发债主体进行信用增强，并进而吸引更多的投资者。特别是在[短期融资券](http://wiki.mbalib.com/wiki/%E7%9F%AD%E6%9C%9F%E8%9E%8D%E8%B5%84%E5%88%B8)发行领域，[商业银行](http://wiki.mbalib.com/wiki/%E5%95%86%E4%B8%9A%E9%93%B6%E8%A1%8C)通过出售信用违约互换方式可以有效增强中小企业信用水平，从而使规模略小一些的企业成为发债体，丰富[信用市场](http://wiki.mbalib.com/wiki/%E4%BF%A1%E7%94%A8%E5%B8%82%E5%9C%BA)品种范围。

　　在[债券市场](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8%E5%B8%82%E5%9C%BA)经历连续多月的上涨后，[国债](http://wiki.mbalib.com/wiki/%E5%9B%BD%E5%80%BA)、[政策金融债](http://wiki.mbalib.com/wiki/%E6%94%BF%E7%AD%96%E9%87%91%E8%9E%8D%E5%80%BA)等无信用风险[债券收益率](http://wiki.mbalib.com/wiki/%E5%80%BA%E5%88%B8%E6%94%B6%E7%9B%8A%E7%8E%87)愈来愈低，机构普遍开始寻找新的[投资方向](http://wiki.mbalib.com/wiki/%E6%8A%95%E8%B5%84%E6%96%B9%E5%90%91)和利润来源，[金融创新](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E5%88%9B%E6%96%B0)逐渐成为机构关心的领域。作为一种[金融创新](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E5%88%9B%E6%96%B0)，信用违约互换有其存在发育的土壤，在[银行间市场](http://wiki.mbalib.com/wiki/%E9%93%B6%E8%A1%8C%E9%97%B4%E5%B8%82%E5%9C%BA)这种柜台市场中，机构间完全可以签署协议，通过[表外业务](http://wiki.mbalib.com/wiki/%E8%A1%A8%E5%A4%96%E4%B8%9A%E5%8A%A1)创新获取收益或者[规避风险](http://wiki.mbalib.com/wiki/%E8%A7%84%E9%81%BF%E9%A3%8E%E9%99%A9)。当然，创新的初始阶段市场交易不会频繁，但不断坚持创新思路、不断坚持实践创新应是低收益率环境下市场成员的工作重点之一。

### 信用违约互换的优点

　　1、信用违约互换自身流动性强。首先，其[交易效率](http://wiki.mbalib.com/wiki/%E4%BA%A4%E6%98%93%E6%95%88%E7%8E%87)高、[交易成本](http://wiki.mbalib.com/wiki/%E4%BA%A4%E6%98%93%E6%88%90%E6%9C%AC)低。[标准化](http://wiki.mbalib.com/wiki/%E6%A0%87%E5%87%86%E5%8C%96)的特点使[交易商](http://wiki.mbalib.com/wiki/%E4%BA%A4%E6%98%93%E5%95%86)可迅速实现大规模的[交易量](http://wiki.mbalib.com/wiki/%E4%BA%A4%E6%98%93%E9%87%8F)，而其交易成本更只有一个[基点](http://wiki.mbalib.com/wiki/%E5%9F%BA%E7%82%B9)的二分之一，远远小于单一[标的资产](http://wiki.mbalib.com/wiki/%E6%A0%87%E7%9A%84%E8%B5%84%E4%BA%A7)信用违约互换的[交易成本](http://wiki.mbalib.com/wiki/%E4%BA%A4%E6%98%93%E6%88%90%E6%9C%AC)，因此，更受信用市场参与者的青睐其次，[市场](http://wiki.mbalib.com/wiki/%E5%B8%82%E5%9C%BA)接受程度高。信用违约互换是公认的整体市场信用风险的一个关键性指标，能够比单一标的资产**信用违约互换**更迅速的反应市场的基本情况，并有化解[系统风险](http://wiki.mbalib.com/wiki/%E7%B3%BB%E7%BB%9F%E9%A3%8E%E9%99%A9)的作用，获得[经销商](http://wiki.mbalib.com/wiki/%E7%BB%8F%E9%94%80%E5%95%86)和业界的广泛支持。

　　2、可以增强信用衍生品市场的流动性。信用违约互换不仅自身的流动性高，还对整个信用衍生品市场流动性的增加有显著的推动作用，这主要体现在两方而:首先，信用违约指数其[样本](http://wiki.mbalib.com/wiki/%E6%A0%B7%E6%9C%AC)仅有125种，交易者可以通过复制信用违约互换在信用衍生产品市场上作多头或者[空头](http://wiki.mbalib.com/wiki/%E7%A9%BA%E5%A4%B4)的交易。在[市场价格](http://wiki.mbalib.com/wiki/%E5%B8%82%E5%9C%BA%E4%BB%B7%E6%A0%BC)发生[暴跌](http://wiki.mbalib.com/wiki/%E6%9A%B4%E8%B7%8C)或者暴涨时，交易者就可以在两个[市场](http://wiki.mbalib.com/wiki/%E5%B8%82%E5%9C%BA)上作[对冲交易](http://wiki.mbalib.com/wiki/%E5%AF%B9%E5%86%B2%E4%BA%A4%E6%98%93)来[回避风险](http://wiki.mbalib.com/wiki/%E5%9B%9E%E9%81%BF%E9%A3%8E%E9%99%A9)，于是[信用衍生产品](http://wiki.mbalib.com/wiki/%E4%BF%A1%E7%94%A8%E8%A1%8D%E7%94%9F%E4%BA%A7%E5%93%81)整体市场的流动性就增加了。其次，[指数](http://wiki.mbalib.com/wiki/%E6%8C%87%E6%95%B0)交易本身就是一个增强[市场流动性](http://wiki.mbalib.com/wiki/%E5%B8%82%E5%9C%BA%E6%B5%81%E5%8A%A8%E6%80%A7)的特征之一，当能够通过信用违约指数基点的变化来对冲信用风险后，[机构投资者](http://wiki.mbalib.com/wiki/%E6%9C%BA%E6%9E%84%E6%8A%95%E8%B5%84%E8%80%85)就能够迅速的针对整个市场信用风险状况来调节信用风险头寸，从而促进了信用衍生品市场的交易量的增加。

　　3、可以平抑对冲信用风险，化解[系统性风险](http://wiki.mbalib.com/wiki/%E7%B3%BB%E7%BB%9F%E6%80%A7%E9%A3%8E%E9%99%A9)。信用违约指数交易发展迅速而且吸引了大量的交易者，这主要是因为[信用](http://wiki.mbalib.com/wiki/%E4%BF%A1%E7%94%A8)[违约](http://wiki.mbalib.com/wiki/%E8%BF%9D%E7%BA%A6)指数为交易者提供了以低廉的成本来快速分散、购买或出售[信用风险](http://wiki.mbalib.com/wiki/%E4%BF%A1%E7%94%A8%E9%A3%8E%E9%99%A9)。信用违约指数的交易者可以通过购买或者出售信用违约指数来管理信用风险头寸，从而化解系统风险

　　4、[定价](http://wiki.mbalib.com/wiki/%E5%AE%9A%E4%BB%B7)具有一定的透明性。信用违约互换指标的定价是每天免费提供的，[投资者](http://wiki.mbalib.com/wiki/%E6%8A%95%E8%B5%84%E8%80%85)可以清楚了解其定价过程，便于投资者自己判断[指标](http://wiki.mbalib.com/wiki/%E6%8C%87%E6%A0%87)的定价公正与否，这实际上相当于向[社会](http://wiki.mbalib.com/wiki/%E7%A4%BE%E4%BC%9A)提供了一个[监督](http://wiki.mbalib.com/wiki/%E7%9B%91%E7%9D%A3)的[窗口](http://wiki.mbalib.com/wiki/%E7%AA%97%E5%8F%A3)，增加人们对信用违约互换产品的信赖众所周知，[定价](http://wiki.mbalib.com/wiki/%E5%AE%9A%E4%BB%B7)的透明、精确是[风险管理](http://wiki.mbalib.com/wiki/%E9%A3%8E%E9%99%A9%E7%AE%A1%E7%90%86)功能有效发挥的重要条件之一，相较其他[金融衍生品](http://wiki.mbalib.com/wiki/%E9%87%91%E8%9E%8D%E8%A1%8D%E7%94%9F%E5%93%81)，信用违约互换在这方而的表现是比较突出的，这就[保证](http://wiki.mbalib.com/wiki/%E4%BF%9D%E8%AF%81)了信用违约互换在[交易](http://wiki.mbalib.com/wiki/%E4%BA%A4%E6%98%93)过程中更具有[可靠性](http://wiki.mbalib.com/wiki/%E5%8F%AF%E9%9D%A0%E6%80%A7)，也促进了市场的优化。

### 信用违约期权与信用违约互换的区别

[信用违约期权](http://wiki.mbalib.com/wiki/%E4%BF%A1%E7%94%A8%E8%BF%9D%E7%BA%A6%E6%9C%9F%E6%9D%83)与信用违约互换的区别并不大，因为在信用事件发生后，期权的购买方总会执行期权以获得补偿。不同的是，在信用违约互换中存在相当于名义[贷款本金](http://wiki.mbalib.com/wiki/%E8%B4%B7%E6%AC%BE%E6%9C%AC%E9%87%91)转移的问题，即[信用保险](http://wiki.mbalib.com/wiki/%E4%BF%A1%E7%94%A8%E4%BF%9D%E9%99%A9)买方可以以[贷款本金](http://wiki.mbalib.com/wiki/%E8%B4%B7%E6%AC%BE%E6%9C%AC%E9%87%91)为基数按双方商定的基点支付费用，而[信用违约期权](http://wiki.mbalib.com/wiki/%E4%BF%A1%E7%94%A8%E8%BF%9D%E7%BA%A6%E6%9C%9F%E6%9D%83)不存在这个问题。另外，在[支付方式](http://wiki.mbalib.com/wiki/%E6%94%AF%E4%BB%98%E6%96%B9%E5%BC%8F)上，信用违约互换是在[合约](http://wiki.mbalib.com/wiki/%E5%90%88%E7%BA%A6)有效期内，定期的多次支付，而信用违约期权为一次性的支付[期权费](http://wiki.mbalib.com/wiki/%E6%9C%9F%E6%9D%83%E8%B4%B9)。

## WI-预发行交易

预发行交易（When-issuedtrading，简称WItrading）

所谓债券预发行交易，是指某只债券在发行公告后、正式招标发行前，市场对该债券进行买卖，并在未来某一天进行资金与债券交割的行为。从本质上说，债券预发行交易是一种短期的[债券](http://www.baike.com/sowiki/%E5%80%BA%E5%88%B8?prd=content_doc_search)远期交易。目前，美国、[英国](http://www.baike.com/sowiki/%E8%8B%B1%E5%9B%BD?prd=content_doc_search)、[法国](http://www.baike.com/sowiki/%E6%B3%95%E5%9B%BD?prd=content_doc_search)等发达[债券市场](http://www.baike.com/sowiki/%E5%80%BA%E5%88%B8%E5%B8%82%E5%9C%BA?prd=content_doc_search)均存在债券预发行交易。债券预发行交易在保证招标债券价格发现的连续性、避免投资者投标的盲目性以及减少债券招标的市场不确定性等方面具有重要意义。

## ABS-资产支持类证券

资产支持型证券（Asset Backed Securities），也简称为ABS，是以[金融资产](http://baike.baidu.com/view/965479.htm)为支持型的[债券](http://baike.baidu.com/view/26411.htm)或票据

典型的资产包括除[抵押贷款](http://baike.baidu.com/view/184702.htm)以外的应收账款，比如信用卡应收账款、[汽车贷款](http://baike.baidu.com/view/1149925.htm)、房屋制造合同以及房地产贷款等等。资产支持型证券与其它大多数[债券](http://baike.baidu.com/view/26411.htm)的不同之处在于它们的信用价值（发行流通中的90%以上[信用评级](http://baike.baidu.com/view/65910.htm)为3A级）并不取决于[标的资产](http://baike.baidu.com/view/1597125.htm)初始持有人的偿付能力。

持有初始贷款的金融机构（包括银行、信用卡供应商、汽车融资公司和消费融资公司）通过一种被称为“[证券化](http://baike.baidu.com/view/1556649.htm)’”的过程，将它们的贷款转化为可以在市场上交易的证券。贷款的初始持有人通常被称为资产支持型证券的发行人，但事实上，他们只是这些证券的发起人，而并不是[直接发行](http://baike.baidu.com/view/500402.htm)的人。这些金融机构将[贷款出售](http://baike.baidu.com/view/2110767.htm)给“特殊目的机构”（SPV），而“特殊目的机构”的唯一功能就是购买这些贷款并将它们证券化。其后，这些通常以公司形式存在的 “特殊目的机构”将证券化后的贷款出售给一家[信托机构](http://baike.baidu.com/view/975398.htm)。信托机构再将贷款重新打包组合成能够产生利息的证券，并真正发行它们。贷款由发起人向“特殊目的机构”的[真实出售](http://baike.baidu.com/view/2682839.htm)提供了一种“[破产隔离](http://baike.baidu.com/view/1428769.htm)”机制，将信托机构与发起人隔离开来。然后，这些证券由投资银行承销，向投资者出售，并通过内部、外部或两者同时进行的一种或多种附加保护措施进行[信用增级](http://baike.baidu.com/view/1433073.htm)。　资产支持型证券构成了债务市场中相对较新但增长迅速的部分。第一种[资产](http://baike.baidu.com/view/42564.htm)支持型证券在1985年发行，在那一年，市场中公开出售发行的资产支持型证券有 120亿美元。到1998年上半年，市场上新发行的此类证券就达到1022亿美元，已快速接近了1997年的全年发行量1851亿美元。据估计，从 1985年到1998年第二季度，总共发行了8820亿美元的资产支持型证券。

## SWT-利率互换期权

swapping options

公司使用利率互换期权的原因和使用利率互换掉期是一样的,主要都是为了限制两种风险.  
这两种风险分别为:  
1. 汇率波动的风险  
2. 浮动利率和[固定利率](https://www.baidu.com/s?wd=%E5%9B%BA%E5%AE%9A%E5%88%A9%E7%8E%87&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PhR1PHNWrjubn1ubnHI90ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En16kPHT1rj61)之间差额的变化  
  
试想现在英国利率(0.5%)较人民币利率(一年期6.65%)低,你现在借入了100万钱的英镑,一年以后只需要偿还0.5%的利息,当时如果你通过人民币来融资,你的融资成本会大为上升.所以就算你的公司在中国,你也会很乐于使用英镑融资.  
  
但是现在还有另外一个问题,就是英镑和人民币之间的汇率是会变的,为了计算简易,设现在英镑兑人民币是10:1,设兑换货币和国际汇款不需要任何费用,如果在一年以后[人民币贬值](https://www.baidu.com/s?wd=%E4%BA%BA%E6%B0%91%E5%B8%81%E8%B4%AC%E5%80%BC&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PhR1PHNWrjubn1ubnHI90ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En16kPHT1rj61)6.15%(6.15%+0.5%=6.65%)以上,那么你还不如就在国内通过人民币融资,为了限制这种汇率变化的风险,公司就会使用利率互换期权,购买利率互换期权的公司有权利决定是否实行利率互换,如果选择实行,那么就按协议规定的固定或者浮动人民币数额来支付利息,如果选择不实行就依然用英镑支付利息.  
  
如果你是这个以英镑融资的[中国公司](https://www.baidu.com/s?wd=%E4%B8%AD%E5%9B%BD%E5%85%AC%E5%8F%B8&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PhR1PHNWrjubn1ubnHI90ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En16kPHT1rj61),当[人民币贬值](https://www.baidu.com/s?wd=%E4%BA%BA%E6%B0%91%E5%B8%81%E8%B4%AC%E5%80%BC&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PhR1PHNWrjubn1ubnHI90ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En16kPHT1rj61)6.05%以上时(设利率互换期权的期权费用为0,实际中还需要考虑期权费用),你就会选择实行利率互换协议,反之,如果[人民币贬值](https://www.baidu.com/s?wd=%E4%BA%BA%E6%B0%91%E5%B8%81%E8%B4%AC%E5%80%BC&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PhR1PHNWrjubn1ubnHI90ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En16kPHT1rj61)幅度在6.05%以下甚至升值了,那你就不会选择实行期权  
  
利率互换期权实行的条件:   
借款币种兑本币的升值产生的额外利息+兑换货币的费用>0  
购买利率互换期权的盈利条件:  
借款币种兑本币的升值产生的额外利息+兑换货币的费用>购买利率互换期权的费用  
  
浮动利率和[固定利率](https://www.baidu.com/s?wd=%E5%9B%BA%E5%AE%9A%E5%88%A9%E7%8E%87&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PhR1PHNWrjubn1ubnHI90ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En16kPHT1rj61)之间差额的变化也是一样的道理,购买期权的人为了防止自己发行的[浮动利率债券](https://www.baidu.com/s?wd=%E6%B5%AE%E5%8A%A8%E5%88%A9%E7%8E%87%E5%80%BA%E5%88%B8&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PhR1PHNWrjubn1ubnHI90ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En16kPHT1rj61)利率过高,会购买[固定利率](https://www.baidu.com/s?wd=%E5%9B%BA%E5%AE%9A%E5%88%A9%E7%8E%87&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3PhR1PHNWrjubn1ubnHI90ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En16kPHT1rj61)的互换期权来限制风险  
  
最早利率互换期权和利率掉期协议都是在双方都有不同币种的借款的时候才签订的,但实际上,也可以在没有任何借款的情况下签发,购买还有实行,这种行为就不再是为了限制风险,而仅仅是一种投机,在这种情况下,跨币种的利率互换期权就和简单的外汇期权作用差不多了

## 利率上限(interest rate cap)

**利率上限**是指买卖双方就未来某一时期商定一个[固定利率](http://wiki.mbalib.com/wiki/%E5%9B%BA%E5%AE%9A%E5%88%A9%E7%8E%87)作为利率上限，如果协议规定的[市场利率](http://wiki.mbalib.com/wiki/%E5%B8%82%E5%9C%BA%E5%88%A9%E7%8E%87)(通常为LIBOR)超过上限利率，则由卖方将市场利率与上限利率的[差额支付](http://wiki.mbalib.com/wiki/%E5%B7%AE%E9%A2%9D%E6%94%AF%E4%BB%98)给买方，但买方在协议签订时，必须支付卖方一定费用。

## 利率下限期权（Floor Options）

利率下限期权又称地面期权或保底期权，即通过固定一个最低 利率 回 避利率降低的风险。 利率下限期权出售者保证购买者可以在一定时期内获得一个最低利率 ，购买者支付 期权费 。如果在结算日参考利率低于 期权合约 中最低利 率，则 期权 出售者将利差付给购买者。此时购买者获得一个选择权， 同 [利率上限](https://www.baidu.com/s?wd=%E5%88%A9%E7%8E%87%E4%B8%8A%E9%99%90&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLmH0dnhf3PHuBmvwbnHD40ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En1bknWb1rj6dPH64nHckn1md)期权 类似，他可以在利率上升中得益， 又可防止利率低于期权合约中最低利率所产生的风险， 具体操作办法与[利率上限](https://www.baidu.com/s?wd=%E5%88%A9%E7%8E%87%E4%B8%8A%E9%99%90&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLmH0dnhf3PHuBmvwbnHD40ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En1bknWb1rj6dPH64nHckn1md)期权大同小异。

## IMIX协议

IMIX协议全称银行间市场业务数据交换协议（Inter-bank Market Information eXchange Protocol是一种用于在银行间市场交易活动（包括人民币、外汇和衍生品等各类产品的交易、发行、确认和清结算等）中实现金融信息实时传输的电子通信协议。IMIX协议是金融行业标准，起草单位为[中国外汇交易中心](http://baike.baidu.com/view/295668.htm)。经全国金融标准化委员会审核通过，该标准于2011年6月2日由中国人民银行发布实施，标准号JR/T 0066-2011。[1-2]

2016年2月25日，基于IMIX协议标准开发的[ISO20022](http://baike.baidu.com/view/10301014.htm)（金融服务：金融业通用报文方案）外汇交易后确认和交易获取的报文正式注册发布。这两组报文是国际标准化组织（[ISO](http://baike.baidu.com/subview/1007/5909092.htm)）首次发布中国金融领域的相关报文。[3]

IMIX协议基于FIX协议制定，其数据模型兼容了FIX协议。FIX协议全称金融信息交换协议（Financial Information eXchange Protocol），是被国际金融界广泛使用的金融标准。FIX协议基于Tag/Value格式制定，提供覆盖交易前、中、后的应用层报文，以及用于控制数据传输的会话层报文。

IMIX协议制定过程中遵循了稳定性和开放性原则，结合国内银行间市场的实际业务对FIX协议进行了扩展，规定了应用环境、会话机制、报文格式、报文定义、扩展方式、安全加密、数据完整性、数据字典等内容。目前，IMIX协议中定义的报文已覆盖了银行间市场产品的报价、订单、交易执行、行情数据发布和成交回报等交易活动环节

## FIX协议

FIX协议全称金融信息交换协议（Financial Information eXchange Protocol），是被国际金融界广泛使用的金融标准。FIX协议基于Tag/Value格式制定，提供覆盖交易前、中、后的应用层报文，以及用于控制数据传输的会话层报文。

## CDN

CDN的全称是Content Delivery Network，即[内容分发网络](http://baike.baidu.com/view/8689800.htm)。其基本思路是尽可能避开互联网上有可能影响数据传输速度和稳定性的瓶颈和环节，使内容传输的更快、更稳定。通过在网络各处放置[节点服务器](http://baike.baidu.com/view/3787429.htm)所构成的在现有的互联网基础之上的一层智能[虚拟网络](http://baike.baidu.com/view/747782.htm)，CDN系统能够实时地根据[网络流量](http://baike.baidu.com/view/411702.htm)和各节点的连接、负载状况以及到用户的距离和响应时间等综合信息将用户的请求重新导向离用户最近的服务节点上。其目的是使用户可就近取得所需内容，解决 Internet网络拥挤的状况，提高用户访问网站的响应速度。

## AI

AI:Artificial Intelligence 人工智能

## IEEE

IEEE：电气和电子工程师协会，全称是Institute of Electrical and Electronics Engineers

## ps:process status

## Linux系统诞生、发展、成长的5个重要支柱

UNIX系统、MINIX系统、GNU计划、POSIX标准、Internet网络。

Linux内核版本已经开发到4.6.x版，版本号中第2个数字若是奇数表示是正在开发的版本，不能保证系统的稳定性。

UNIX最初源代码版本为Linux提供了实现的基本原理和算法；

GNU计划为Linux系统提供了丰富且免费的各种实用工具；

POSIX标准的出现为Linux提供了实现与标准兼容系统的参考指南；

MINIX操作系统为Linux的诞生起到了不可忽缺的参考；

Internet是Linux成长和壮大的必要环境。

CMOS:Complementary Metal Oxide Semiconductor 互补金属氧化物半导体

## 沙箱

Sandboxie(又叫沙箱、沙盘)即是一个虚拟系统程序。

近年来，随着网络安全问题的日益突出，人们更多的将沙箱技术应用于网上冲浪方面。从技术实现角度而言，就是从原有的阻止可疑程序对系统访问，转变成将可疑程序对磁盘、注册表等的访问重定向到指定文件夹下，从而消除对系统的危害

## 麦道夫黑洞

指美国历史上数额最大的诈骗案，因作案者为“大慈善家、社会的支柱、代表华尔街品质”的纳斯达克前董事会主席麦道夫而得名，其行骗时间长达20年、数额高达500亿美元。其手法就是利用传统的“庞氏骗局”手法行骗。“庞氏骗局”是金字塔传销的模板之一，是一种非法集资，得名于查尔斯·庞兹，其实质就是“借钱还钱”，整个行骗过程并没有实质的利润产生，投资者的回报完全是后来者新资金及自己扩大再投资带来的钱。

麦道夫“黑洞”吞噬500亿美元事件像惊悚连续剧一样，每隔几天就爆一个受害人。汇丰刚确认10亿美元资金卷入，又爆出受害者大导演斯皮尔伯格、诺贝尔文学奖得主威瑟尔……受害人名单仍在不断加长。

这个拥有近半个世纪“白璧无瑕”从业记录的“明星经纪人”，这个被视为“大慈善家、社会的支柱、代表华尔街品质”的纳斯达克前董事会主席，竟然行骗长达20年、高达500亿美元，成为美国历史上数额最大的诈骗案。

从数年前的美国安然案到今年雷曼兄弟破产，再到500亿美元天价诈骗，全球惊愕：金融塔尖华尔街的原来不过是群魔乱舞之地，此次，麦道夫不折不扣地给了华尔街又一耳光。

麦道夫一直是与 巴菲特 齐名的“超五星级”举足轻重的金融人物。20年里，他公司名下的账户每个月能像时钟一样准时产生将近1%的收益，无论牛市、熊市。“麦道夫是犹太版的国库券。他是那么的安全。”“如果麦道夫是骗子，他将把全世界一半人拖下水。”他被称为金融神话。

东窗事发后，人们才发现，此人骗术一点儿也不高明，也无需若干精算师、博士用数学模块计算，它古老、常见，甚至连美国乡下的老婆婆都能一眼识破。

这个简单的“拆东墙补西墙”骗过了美国、欧洲、日本等国的无数权贵。可见这个“不乱花钱、以乐善好施著称、跟邻居亲切地喊名字打招呼的和蔼可亲的胖子”麦道夫是个心理高手。他以“神秘的魅力和排他的方式”来引诱受害者上钩。他把自己的投资策略冠以无人能懂的“分裂转换”，高深莫测得令所有人觉得有人生来就比自己知道得多。他还把他的圈子搞成排他性的“有钱人的俱乐部”，巧妙地将投资者的恐惧从担心亏钱转换成了担心在这场赚钱大戏中掉队。进入麦道夫的“俱乐部”成为身份显贵的象征。

说到底，麦道夫能用最古老的骗术一骗20年，无非是利用了人的幼稚和贪婪。正如牛顿说：“我可以计算天体运行的轨道，却无法计算人性的疯狂。”

是啊，财富造成的贪婪人，比贪婪造成的富人要多。再完美的制度也限制不了人性的贪婪。“太阳底下没有新事”，历史总在不断重演。

## 庞氏骗局

庞氏骗局是对金融领域投资诈骗的称呼，金字塔骗局（Pyramid scheme）的始祖，很多非法的传销集团就是用这一招聚敛钱财的，这种骗术是一个名叫查尔斯·庞兹的投机商人“发明”的。庞氏骗局在中国又称“拆东墙补西墙”，“空手套白狼”。简言之就是利用新投资人的钱来向老投资者支付利息和短期回报，以制造赚钱的假象进而骗取更多的投资。

查尔斯·庞兹（Charles Ponzi）是一位生活在19、20世纪的意大利裔投机商，1903年移民到美国，1919年他开始策划一个阴谋，骗子向一个事实上子虚乌有的企业投资，许诺投资者将在三个月内得到40%的利润回报，然后，狡猾的庞兹把新投资者的钱作为快速盈利付给最初投资的人，以诱使更多的人上当。由于前期投资的人回报丰厚，庞兹成功地在七个月内吸引了三万名投资者，这场阴谋持续了一年之久，才让被利益冲昏头脑的人们清醒过来，后人称之为“庞氏骗局”。

“庞氏骗局”源自于一个名叫查尔斯-庞兹（Charles Ponzi，1882

地设陷阱非法集资

地设陷阱非法集资

-1949）的人， 他是一个意大利人，1903年移民到美国。在美国干过各种工作，包括油漆工，一心想发大财。他曾因伪造罪在加拿大坐过牢，在美国亚特兰大因走私人口而蹲过监狱。经过美国式发财梦十几年的熏陶，庞兹发现最快速赚钱的方法就是金融，于是，从1919年起，庞兹隐瞒了自己的历史来到了波士顿，设计了一个投资计划，向美国大众兜售。

这个投资计划说起来很简单，就是投资一种东西，然后获得高额回报。但是，庞兹故意把这个计划弄得非常复杂，让普通人根本搞不清楚。1919年，第一次世界大战刚刚结束，世界经济体系一片混乱，庞兹便利用了这种混乱。他宣称，购买欧洲的某种邮政票据，再卖给美国，便可以赚钱。国家之间由于政策、汇率等等因素，很多经济行为普通人一般确实不容易搞清楚。其实，只要懂一点金融知识，专家都会指出，这种方式根本不可能赚钱。然而，庞兹一方面在金融方面故弄玄虚， 另一方面则设置了巨大的诱饵，他宣称，所有的投资，在90天之内都可以获得40%的回报。而且，他还给人们“眼见为实”的证据：最初的一批“投资者”的确在规定时间内拿到了庞兹所承诺的回报。于是，后面的“投资者”大量跟进。

在一年左右的时间里，差不多有4万名波士顿市民，傻子一样变成庞兹赚钱计划的投资者，而且大部分是怀抱发财梦想的穷人，庞兹共收到约1500万美元的小额投资，平均每人“投资”几百美元。当时的庞兹被一些愚昧的美国人称为与哥伦布、马可尼（无线电发明者）齐名的最伟大的三个意大利人之一，因为他像哥伦布发现新大陆一样“发现了钱”。庞兹住上了有20个房间的别墅，买了100多套昂贵的西装，并配上专门的皮鞋，拥有数十根镶金的拐杖，还给他的妻子购买了无数昂贵的首饰，连他的烟斗都镶嵌着钻石。当某个金融专家揭露庞兹的投资骗术时，庞兹还在报纸上发表文章反驳金融专家，说金融专家什么都不懂。

1920年8月，庞兹破产了。他所收到的钱，按照他的许诺，可以购买几亿张欧洲邮政票据，事实上，他只买过两张。此后，“庞兹骗局”成为一个专门名词，意思是指用后来的“投资者”的钱，给前面的“投资者”以回报。庞兹被判处5年刑期。出狱后，他又干了几件类似的勾当，因而蹲了更长的监狱。1934年被遣送回意大利，他又想办法去骗墨索里尼，也没能得逞。1949年，庞兹在巴西的一个慈善堂去世。死去时，这个“庞氏骗局”的发明者身无分文。

## HCE

Host-based Card Emulation)，

## ApplePay的Token体系与流程实现

1.手机终端传递PAN至Token Requestor;

　　2.Token Requestor向Token SP发起Token申请，发送PAN;

　　3.Token SP根据PAN，生成对应的Token并返回至Token Requestor，同时建立PAN至Token的映射并存储在Token库中，与发卡方共享此Token库;

　　4.Token Requestor将返回的Token传递至NFC终端。

　　ApplePay支付流程实现：线下支付

　　1.用户商店购物，用Apple Pay支付时，将手机终端靠近商户POS，NFC激活，提示用户扫描指纹支付，商户POS从手机终端内读取Payment Token、Token Expiry Date、Token Cryptogram（根据Token等数据动态生成）及其他标准的非接数据；

　　2.商户POS向收单机构上送支付请求数据;

　　3.收单机构完成例行检查，通过支付网络转发支付请求;

　　4.支付网络调用Token service provider服务，校验Token请求合法性，通过PAN与Token对照关系，重新获取PAN、 PAN Expiry Date等数据。

　　5.完成De-Tokenize 后，支付网络向发卡行上送将替换为PAN相关数据的支付请求，发卡行完成相应校验后，完成支付扣款；

　　6.发卡行完成支付后，返回PAN给支付网络；

　　7.支付网络接收到发卡行支付结果，调用Token service provider服务,重新根据PAN,Token映射关系，查找PAN对应的Token；

　　8.支付网络将token、PAN后四位及其他数据发送至收单机构；

　　9.收单机构通知商户终端；

　　10.商户终端根据结果，通知用户支付成功或失败。

　　Token体系概述，Apple Pay引入的Token体系中，除了传统电子支付参与方外，新增了2个参与方。

　　Token的应用原理：Token SP根据Token Requestor提供的PAN(主帐号)生成Token后，将Token作为PAN的替代值流转在支付的各个环节，使得在支付流程中，独一无二的PAN只在Token SP、转接方、发卡方间传递，由于三者专线连接且彼此互信，且当Token被检测到风险或到期时，将再次生成新Token替代，从而大幅降低支付过程中PAN泄漏的可能性，极大地提高了PAN的安全性。

Token体系概述-Q&A

　　1、 Q：Token的用途和目的? A: 作为PAN的替代值，大幅度降低了在交易流程中PAN泄漏的安全隐患。

　　2、 Q: Token替代PAN，安全性如何得到提升? A：Token替代PAN，使得PAN只在Token SP、转接方、发卡方间传递，而这三者理论上是互信的，且是专线连接，从而大幅度降低了在交易流程中PAN泄漏的安全隐患。

　　3、 Q：Token作为PAN的替代值，如何解决在传统交易流程中，也存在的与PAN类似的泄漏隐患? A：当Token被检测到风险或到期时，再次生成新Token替代。

　　4、 Q：Token生成和验证完成后，后续的传统交易流程是否改变?

　　A: 仅就Apply Pay目前披露的信息而言，完成Token生成和验证后，Token将作为PAN的替代值被使用，不改变后续的传统交易流程。

　　5、 Q：Token体系中，谁能承担新引入的Token Requestor和Token SP角色?

　　A：可以是新玩家，也可以是传统交易流程中的老玩家，规范并没有强制定义谁去承担这两个角色，所以不同的应用场景，可能这两个角色的承担者不同。

　　a) 能获得用户PAN的实体是天然的Token Requestor，如支付宝钱包、各银行手银等;

　　b) 能得到发卡方授权，建立PAN至Token的映射的实体是Token SP，如转接方或发卡方等，其中转接方是Token SP最可能的承担者。

　　c) Token SP控制PAN至Token的映射这一核心数据，在Token系统中十分关键，而Token Requestor相对不重要。

　　6、 Q：Token在iPhone6中的存储位置?

　　A：规范中并未强制规定Token的存储位置，可选的有远程存储、SE、TEE等多种存储方式，仅就Apply Pay披露的信息而言，合理怀疑Token存储在iPhone6的eSE中。

　　7、 Q：Token体系在国内技术实现的难点在何处?

　　A：首先要建立Token Requestor和Token SP系统;其次，Token传递路径上的所有参与方(POS、收单方、转接方、发卡方)均需要按照规范进行相应的系统升级，以保证良好的互操作性和一致性。

　　8、 Q：Apple Pay方案中运营商似乎被边缘化了?

　　A：只要是eSE模式，运营商都可能被边缘和管道化。SE控制权的博弈过程中，移动终端厂商一步步积累了愈来愈深厚的技术和业务实力，运营商的优势将越来越小，合理预期运营商被边缘和管道化是大势所趋。

　　9、 Q：Apply Pay方案在国内是否能成功复制? A：Apply Pay在NFC原有的产业链上又新增了Token Requestor和Token SP，成功复制Apply Pay需要两个前提，或者拥有庞大的NFC手机用户体量(掌控eSE)，或者成为数量稀缺的Token SP(掌控PAN至Token的映射核心数据)。原先看好小米，可惜米4不带NFC。

## TUXEDO全称：

 Transaction for UNIX (TUX) + distributed client – server communications (Extended for Distributed Operation)

## SALT全称：

Architecture   Leveraging    Tuxedo

## ANSI全称

American National Stardards Institute

## POP 和 IMAP协议、SMTP

与POP3协议类似，IMAP（Internet消息访问协议）也是提供面向用户的邮件收取服务。

POP:是Post Office Protocol 邮局协议的简称。

POP3 使用 TCP 作为传输协议，端口号是110。

POP3 不支持对服务器上邮件进行扩展操作，此过程由更高级的 IMAP4 完成。

IMAP:是Internet Mail Access Protocol交互式邮件存取协议的简称。

IMAP协议运行在TCP/IP协议之上，使用的端口是143.

它与POP3协议的主要区别是用户可以不用把所有的邮件全部下载，

可以通过客户端直接对服务器上的邮件进行操作。

其实说个简单的，就是如果你不对客户端做一个特殊设置的话，

默认情况下，你使用POP3协议下载邮件，

会把服务器上的所有邮件下载到客户端，而服务器上将不再存储。

但是IMAP在默认情况下，下载但是不删除邮件服务器上的文件。

SMTP（Simple Mail Transfer Protocol）即简单邮件传输协议,

它是一组用于由源地址到目的地址传送邮件的规则，

由它来控制信件的中转方式。SMTP协议属于TCP/IP协议簇，

它帮助每台计算机在发送或中转信件时找到下一个目的地。

通过SMTP协议所指定的服务器,就可以把E-mail寄到收信人的服务器上了，

整个过程只要几分钟。SMTP服务器则是遵循SMTP协议的发送邮件服务器，用来发送或中转发出的电子邮件。

## 费雪公式

MV=PQ

M[货币需求量](http://www.haosou.com/s?q=%E8%B4%A7%E5%B8%81%E9%9C%80%E6%B1%82%E9%87%8F&ie=utf-8&src=wenda_link)

V[货币流通速度](http://www.haosou.com/s?q=%E8%B4%A7%E5%B8%81%E6%B5%81%E9%80%9A%E9%80%9F%E5%BA%A6&ie=utf-8&src=wenda_link)(一定时间内货币转手几次)（物理上速度用velocity）

P商品价格

Q商品量

## 4S店中的S代表什么意思

4S店是一种以“四位一体”为核心的汽车特许经营模式，包括整车销售（Sale）、零配件（Sparepart）、售后服务（Service）、信息反馈等（Survey）。 4S店一般采取一个品牌在一个地区分布一个或相对等距离的几个专卖店，按照生产厂家的统一店内外设计要求建造，投资巨大，动辄上千万，甚至几千万，豪华气派。

## CSV文件格式

comma-separated value

## NFC功能的手机实现卡模拟的两种方式

在一部配备NFC功能的手机实现卡模拟，目前有两种方式：一种是基于硬件的，称为[虚拟卡](http://baike.baidu.com/view/485903.htm)模式(Virtual Card Mode);一种是基于软件的，被称为主机卡模式(Host Card Mode  )。

在虚拟卡模式下，需要提供安全模块[SE](http://baike.baidu.com/view/477740.htm)(Secure Elemen)，[SE](http://baike.baidu.com/view/477740.htm)提供对敏感信息的安全存储和对交易事务提供一个安全的执行环境。[NFC](http://baike.baidu.com/subview/917495/5282340.htm)芯片作为非接触通讯前端，将从外部读写器接收到的命令转发到SE，然后由SE处理，并通过NFC控制器回复。

在主机卡模式下，不需要提供SE，而是由在手机中运行的一个应用或云端的服务器完成SE的功能，此时[NFC](http://baike.baidu.com/subview/917495/5282340.htm)芯片接收到的数据由操作系统或发送至手机中的应用，或通过移动网络发送至云端的服务器来完成交互。两种方式的特点都是绕过了手机内置的SE的限制。这一标准的妙处在于，它不需要整个行业为了控制安全元件而争斗。

## ApplePay的Token体系与流程实现

1.手机终端传递PAN至Token Requestor;

　　2.Token Requestor向Token SP发起Token申请，发送PAN;

　　3.Token SP根据PAN，生成对应的Token并返回至Token Requestor，同时建立PAN至Token的映射并存储在Token库中，与发卡方共享此Token库;

　　4.Token Requestor将返回的Token传递至NFC终端。

　　ApplePay支付流程实现：线下支付

　　1.用户商店购物，用Apple Pay支付时，将手机终端靠近商户POS，NFC激活，提示用户扫描指纹支付，商户POS从手机终端内读取Payment Token、Token Expiry Date、Token Cryptogram（根据Token等数据动态生成）及其他标准的非接数据；

　　2.商户POS向收单机构上送支付请求数据;

　　3.收单机构完成例行检查，通过支付网络转发支付请求;

　　4.支付网络调用Token service provider服务，校验Token请求合法性，通过PAN与Token对照关系，重新获取PAN、 PAN Expiry Date等数据。

　　5.完成De-Tokenize 后，支付网络向发卡行上送将替换为PAN相关数据的支付请求，发卡行完成相应校验后，完成支付扣款；

　　6.发卡行完成支付后，返回PAN给支付网络；

　　7.支付网络接收到发卡行支付结果，调用Token service provider服务,重新根据PAN,Token映射关系，查找PAN对应的Token；

　　8.支付网络将token、PAN后四位及其他数据发送至收单机构；

　　9.收单机构通知商户终端；

　　10.商户终端根据结果，通知用户支付成功或失败。

　　Token体系概述，Apple Pay引入的Token体系中，除了传统电子支付参与方外，新增了2个参与方。

　　Token的应用原理：Token SP根据Token Requestor提供的PAN(主帐号)生成Token后，将Token作为PAN的替代值流转在支付的各个环节，使得在支付流程中，独一无二的PAN只在Token SP、转接方、发卡方间传递，由于三者专线连接且彼此互信，且当Token被检测到风险或到期时，将再次生成新Token替代，从而大幅降低支付过程中PAN泄漏的可能性，极大地提高了PAN的安全性。

Token体系概述-Q&A

　　1、 Q：Token的用途和目的? A: 作为PAN的替代值，大幅度降低了在交易流程中PAN泄漏的安全隐患。

　　2、 Q: Token替代PAN，安全性如何得到提升? A：Token替代PAN，使得PAN只在Token SP、转接方、发卡方间传递，而这三者理论上是互信的，且是专线连接，从而大幅度降低了在交易流程中PAN泄漏的安全隐患。

　　3、 Q：Token作为PAN的替代值，如何解决在传统交易流程中，也存在的与PAN类似的泄漏隐患? A：当Token被检测到风险或到期时，再次生成新Token替代。

　　4、 Q：Token生成和验证完成后，后续的传统交易流程是否改变?

　　A: 仅就Apply Pay目前披露的信息而言，完成Token生成和验证后，Token将作为PAN的替代值被使用，不改变后续的传统交易流程。

　　5、 Q：Token体系中，谁能承担新引入的Token Requestor和Token SP角色?

　　A：可以是新玩家，也可以是传统交易流程中的老玩家，规范并没有强制定义谁去承担这两个角色，所以不同的应用场景，可能这两个角色的承担者不同。

　　a) 能获得用户PAN的实体是天然的Token Requestor，如支付宝钱包、各银行手银等;

　　b) 能得到发卡方授权，建立PAN至Token的映射的实体是Token SP，如转接方或发卡方等，其中转接方是Token SP最可能的承担者。

　　c) Token SP控制PAN至Token的映射这一核心数据，在Token系统中十分关键，而Token Requestor相对不重要。

　　6、 Q：Token在iPhone6中的存储位置?

　　A：规范中并未强制规定Token的存储位置，可选的有远程存储、SE、TEE等多种存储方式，仅就Apply Pay披露的信息而言，合理怀疑Token存储在iPhone6的eSE中。

　　7、 Q：Token体系在国内技术实现的难点在何处?

　　A：首先要建立Token Requestor和Token SP系统;其次，Token传递路径上的所有参与方(POS、收单方、转接方、发卡方)均需要按照规范进行相应的系统升级，以保证良好的互操作性和一致性。

　　8、 Q：Apple Pay方案中运营商似乎被边缘化了?

　　A：只要是eSE模式，运营商都可能被边缘和管道化。SE控制权的博弈过程中，移动终端厂商一步步积累了愈来愈深厚的技术和业务实力，运营商的优势将越来越小，合理预期运营商被边缘和管道化是大势所趋。

　　9、 Q：Apply Pay方案在国内是否能成功复制? A：Apply Pay在NFC原有的产业链上又新增了Token Requestor和Token SP，成功复制Apply Pay需要两个前提，或者拥有庞大的NFC手机用户体量(掌控eSE)，或者成为数量稀缺的Token SP(掌控PAN至Token的映射核心数据)。原先看好小米，可惜米4不带NFC。

## HCE究竟为nfc业务带来了什么好处

面对[移动支付](https://www.baidu.com/s?wd=%E7%A7%BB%E5%8A%A8%E6%94%AF%E4%BB%98&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9nyfvPhn4nH63nWDYmyw-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPWD3PW0LnjDsnWcYrjT1njRz)，以及扑面而来的[O2O电子商务](https://www.baidu.com/s?wd=O2O%E7%94%B5%E5%AD%90%E5%95%86%E5%8A%A1&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9nyfvPhn4nH63nWDYmyw-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPWD3PW0LnjDsnWcYrjT1njRz)的机会，NFC始终是一道朦胧的风景。NFC的不温不火，一直让坚守NFC阵营的人们期待这只是黎明前的黑暗，不过，转眼间黎明前的黑暗已经10年。10年来，NFC一直只听楼梯响不见佳人来，害苦了赌上一切、压宝NFC的芸芸众生。

NFC经历了终端功能从功能机到智能机，终端品牌从[诺基亚](https://www.baidu.com/s?wd=%E8%AF%BA%E5%9F%BA%E4%BA%9A&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9nyfvPhn4nH63nWDYmyw-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPWD3PW0LnjDsnWcYrjT1njRz)到三星，终端业务管道从互联网到移动互联网，现在看起来，NFC依然是那个NFC，[苹果](https://www.baidu.com/s?wd=%E8%8B%B9%E6%9E%9C&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9nyfvPhn4nH63nWDYmyw-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPWD3PW0LnjDsnWcYrjT1njRz)已经推出了ibeacon，笑看云起云落，义无反顾的抛弃了NFC。说不清是对NFC坚守者的嘲笑还是不屑。

只所以出现如今的局面，原因可能千万，但其中NFC发展始终有一道坎，就是如何解决松耦合的[无线通信](https://www.baidu.com/s?wd=%E6%97%A0%E7%BA%BF%E9%80%9A%E4%BF%A1&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9nyfvPhn4nH63nWDYmyw-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPWD3PW0LnjDsnWcYrjT1njRz)与安全的平衡，这道坎上人头攒动，终端制造商、卡商、通信运营商、业务提供商，各显神通。提出的方案也是五花八门，全终端、[SIM卡](https://www.baidu.com/s?wd=SIM%E5%8D%A1&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9nyfvPhn4nH63nWDYmyw-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPWD3PW0LnjDsnWcYrjT1njRz)、SD卡，冲着的都是那个解决安全问题的SE(安全单元)。可惜，在这场百花争鸣的盛宴中谁也不能一统江湖，你方唱吧我登场，但均是[叫好不叫座](https://www.baidu.com/s?wd=%E5%8F%AB%E5%A5%BD%E4%B8%8D%E5%8F%AB%E5%BA%A7&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9nyfvPhn4nH63nWDYmyw-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPWD3PW0LnjDsnWcYrjT1njRz)，不能带来突破。

目前，中国出台的[移动支付](https://www.baidu.com/s?wd=%E7%A7%BB%E5%8A%A8%E6%94%AF%E4%BB%98&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9nyfvPhn4nH63nWDYmyw-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPWD3PW0LnjDsnWcYrjT1njRz)标准是NFC-SWP，即是终端包含NFC芯片，[SIM卡](https://www.baidu.com/s?wd=SIM%E5%8D%A1&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9nyfvPhn4nH63nWDYmyw-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPWD3PW0LnjDsnWcYrjT1njRz)包含SE，是一个运营商、银行相对都可以接受的方案。当然，又要换手机又要换卡的方案还要看最终用户是不是待见。到目前为止，仅仅是这锅汤依然不温不火的煲着。

[谷歌](https://www.baidu.com/s?wd=%E8%B0%B7%E6%AD%8C&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9nyfvPhn4nH63nWDYmyw-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPWD3PW0LnjDsnWcYrjT1njRz)在移动互联网的盛宴中，起的不能算早，甚至也看不到在这个领域横征暴敛一夜暴富的潜力，但赔本赚吆喝之余，不声不响的占据了智能终端半壁江山的android却为[谷歌](https://www.baidu.com/s?wd=%E8%B0%B7%E6%AD%8C&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9nyfvPhn4nH63nWDYmyw-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPWD3PW0LnjDsnWcYrjT1njRz)带来业界实打实的话语权，虽然不是终端制造商，也不是通信运营商，但对[操作系统](https://www.baidu.com/s?wd=%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9nyfvPhn4nH63nWDYmyw-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPWD3PW0LnjDsnWcYrjT1njRz)开放体系的掌握，无人能及，以至于android无论出什么官方的更新都可能变成移动终端上事实上的工业标准。当[谷歌](https://www.baidu.com/s?wd=%E8%B0%B7%E6%AD%8C&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9nyfvPhn4nH63nWDYmyw-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPWD3PW0LnjDsnWcYrjT1njRz)宣布在andorid 4.4的系统中支持HCE时，着实朝着NFC这一潭死水中扔了个大炮仗。

HCE说白了就是模拟NFC和SE的通信的协议和实现。HCE没有实现SE，但是HCE以NFC与SE通信的方式告诉NFC读卡器后面有SE支持，从而以虚拟SE的方式完成NFC业务的安全保障。HCE可以让读卡器信以为真有SE，那么没有SE芯片，HCE用谁来充当SE呢，解决方案要么是云端的模拟，要么是本地软件的模拟。

谷歌提出HCE的技术并不是创新和发明，早在2011年，黑莓已经提出了类似的方案(吐槽一句，其实对黑莓而言，真的没必要提出模拟SE的方案，黑莓终端是自家的硬件，搞个全终端方案不就好了吗，何必绕圈子。当然，也许HCE可以节省SE的成本是黑莓方案背后的秘密)。美国的移动互联网创新者simplytapp也早提出了云端SE的概念，并通过CyanogenMOD的非官方发行android ROM发布了类似谷歌HCE的实现，也就是说，拥有android手机的用户可以通过root 系统后刷机的方式利用非官方ROM使用类似于谷歌HCE的功能。因此，谷歌HCE只不过是谷歌根据业界的创新和未来战略提出的HCE官方的实现而已。不过这个官方实现，对于谷歌来说，也是一个开创新的业务模式的创新，这样有争议的创新，官方发布也具备了一些值得称道的勇气。

谷歌的HCE只是模拟了NFC与SE的通信，至于SE的实现则是空白，没有提出解决方案和实现。针对SE，谷歌留下的是空间但同时也是责任，负责安全的SE如何通过本地化的软件或者远程的云端实现，并且能够保障安全性，着实是个头疼的问题，传统意义上而言，没有独立硬件支撑的密钥存储和运算机制，安全都要打一个折扣，这也是为什么一个SE把NFC产业折腾的欲仙欲死的根源所在。谷歌聪明的把这个难题留白，给各路神仙留下了空间，同时，也相对的把安全责任的包袱给甩得一干二净。

HCE究竟为NFC业务带来了什么好处，需要从NFC支持的业务谈起，首先需要澄清的是NFC不等同于[移动支付](https://www.baidu.com/s?wd=%E7%A7%BB%E5%8A%A8%E6%94%AF%E4%BB%98&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9nyfvPhn4nH63nWDYmyw-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPWD3PW0LnjDsnWcYrjT1njRz)，移动支付只是NFC诸多应用中安全要求最高的应用之一，在移动互联网业务爆发的时代，NFC应用的范围也日益广泛，会员卡、优惠券、电子票等基于身份、凭证的应用有些虽然可以划入广义的移动支付的范畴，但和需要牌照的支付和银行的支付在安全和业务监管要求的程度上不尽相同。这些类型的NFC应用安全要求不像账户中有大量资金的银行类支付业务，虽然需要对用户身份、凭证的安全性做出确认，避免假冒和伪造，但还未上升到金融业务的高度，因此如何在安全、便利和业务发展之间寻求平衡十分重要。同时，NFC是嫁接移动互联网与线下，共同完成[O2O电子商务](https://www.baidu.com/s?wd=O2O%E7%94%B5%E5%AD%90%E5%95%86%E5%8A%A1&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9nyfvPhn4nH63nWDYmyw-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPWD3PW0LnjDsnWcYrjT1njRz)活动的关键要素，完全称得上是移动互联网O2O业务的依赖的核心交互技术手段，是O2O行业发展的[不遑多让](https://www.baidu.com/s?wd=%E4%B8%8D%E9%81%91%E5%A4%9A%E8%AE%A9&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9nyfvPhn4nH63nWDYmyw-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPWD3PW0LnjDsnWcYrjT1njRz)的关键助力。O2O业务的发展，在没有HCE的时代，同样受至于NFC的发展，堪称裹足不前(当然，基于二维码、RFID的一些业务也在风风火火开展，只不过是有不小的局限性而已)。谷歌HCE的出现为这些非金融支付类的O2O业务打开了一扇门，在一定程度上提供安全的模拟手段，经济的解决SE的问题，为业务的蓬勃发展大有助力。

至于金融级别的移动支付，HCE是否可以登堂入室，第一要看Visa、master、银联是否能够解决本地软件、远程云端SE的安全问题，第二要看，技术上可以解决后上述机构和政府层面是否愿意接受这样的方案。目前，国内有个别运营商和银行在小规模试用类似HCE的技术手段，希望推动移动支付的发展，但是否能够成为标准，形成规模，还需要时间考验。因此，HCE对移动支付行业发展的影响，暂时不好评论。

HCE为NFC的发展带来了价值，同样也存在着一定的安全风险。HCE带来了繁花似锦，也免不了苍蝇蚊虫，HCE给O2O的业务发展打开了一扇门，也给投机取巧恶意钻营者开了一扇窗。本地软件模式和云端模拟远程实现SE，安全级别没有硬件芯片级别的安全级别高，在通信和使用中，存在SE密钥安全泄露的问题。另外，也存在假冒具备安全芯片第三方SE的可能性，即使使用了安全芯片SE的金融支付应用，也可以被人为地使用HCE假冒和伪造，带来较高的风险。伪造和假冒别人的SE技术成本降低，SE造成的欺诈也会如影随行。因此，配套的对SE类型和业务的鉴别和认证机制建设和运营迫在眉睫。如果没有认证的SE大行其道，欺骗欺诈满天飞，HCE营造的NFC业务发展的大好前途只会黯然收场。甚至是牵连目前已经具备硬件芯片SE的应用也会受到影响，中国制定的移动支付标准的根本或者也会受到动摇，即使发行了SIM卡的SE也会受到不小的挫折。

从谷歌一贯的作风来看，谷歌对安全控制的形象一直不佳，甚至有些安全手段的控制力度只能用匪夷所思来形容。例如，在应用认证机制中，允许开发者自签名证书，用笑柄来形容并不为过，可以堂而皇之的使用第三方的名义发布应用，没有权威机构发行的证书是对PKI/CA机制不了解造成，有时候，没文化真可怕。

当然，谷歌可以解释为安全方面留白太多是给终端制造商和通信运营商以控制和运营安全机制的机会。不过，从终端制造商和通信运营商来看，真正能把这个机制建设和运营好的，也只能说是寥寥无几。因此这种留白，只能说是个天大的漏洞。

同样，面对HCE时代的SE认证机制，如果终端制造商和通信运营商没有把手里的权力和责任使用好完善模拟SE的实现并提出安全解决方案，建立对SE的鉴别和认证机制，避免模拟SE漏洞百出，避免伪造和假冒第三方SE大行其道。那么结局也会相当悲催，最可能的结果是彻底埋葬NFC的前途，唱响NFC的最终葬歌。

总体来说，谷歌的HCE布局的先手优势明显，非常到位的解决了困扰NFC以及O2O业务发展的瓶颈，同时为产业链留下了可以深耕的空间，开发、销售、运营本地软件模拟和远程云端模拟SE是一门不错的生意。这种方式，即成全了谷歌开放的名声，也减轻了安全的相关责任。同时，谷歌的HCE对本地软件和远程云端的SE并未建立起完备安全的认证机制，可能造成欺诈风险肆虐，从而影响HCE的推广和发展，甚至会影响NFC的发展。

作为一个开放平台提供商，谷歌可以不运营这样的机制，但是谷歌需要提供这样的机制和解决方案。这个机制的运营可以由终端制造商和通信运营商来承担。当然，在谷歌不具备该机制的前提下，终端制造商和通信运营商应该及时补位，提出模拟SE安全性和SE鉴别认证机制，从而针对性的解决HCE发布带给NFC发展的问题。避免形式恶化后不可收拾的崩盘。

## 轧差

轧差是指利用抵销、合同更新等法律制度，最终取得一方对另一方的一个数额的净债权或净债务，如市场交易者之间，可能互有内容相同，方向相反的多笔交易，在结算或结束交易时，可以将各方债权在相等数额内抵销，仅支付余额。

## 3C认证

3C认证的[全称](http://baike.baidu.com/view/3216501.htm)为“[强制性产品认证制度](http://baike.baidu.com/view/161696.htm)”，它是中国政府为保护[消费者](http://baike.baidu.com/view/99548.htm)人身安全和国家安全、加强产品[质量](http://baike.baidu.com/subview/13930/5063889.htm)管理、依照法律法规实施的一种产品合格评定制度。所谓3C认证，就是中国强制性产品[认证制度](http://baike.baidu.com/view/137160.htm)，英文名称China Compulsory Certification，英文缩写[CCC](http://baike.baidu.com/view/77862.htm)。

## 眼镜的参数

### 折射率

　折射率是反映镜片材料折射能力的一个参数。折射率值越大，光线进入镜片偏离得就越远。一般情况下，折射率值越大，镜片厚度越小，镜片的重量越轻。

### 色散系数

色散现象是指由于不同颜色的光线，波长不同，折射率也不同，当白色光线被折射后，所产生的颜色像差即为色散现象。材料的色散能力可以由阿贝数来描述。阿贝数的大小与材料的色散成反比，即阿贝数越大，表示该材料的色散越小。通常情况下，眼镜片材料的阿贝数为30-60。一般眼镜片的折射率越高，色散越大，阿贝数就越低。

### 光透射比

按镜片的透光程度，也分为四类，可以叫“透射比分类”。  
1、43%~80%。  
2、18%~43%。  
3、8%~18%。  
4、3%~8%。  
按照要求，驾驶用镜透射比要大于8%，所以第4类[太阳镜](https://www.baidu.com/s?wd=%E5%A4%AA%E9%98%B3%E9%95%9C&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3nvRkmym1rHT1PW79nHFW0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EP1DzPjb1Pj6vPjD1nWTLrHcd)是不能开车带的。

### 顶焦度

顶焦度是一种度量单位的名称，是用来表述透镜对光线屈折能力大小的，在数值上等于透镜焦距（以米为单位）的倒数。以f代表焦距，F代表顶焦度即为F=1/f。

顶焦度表示单位为米的倒数(m-1)，单位名称为屈光度,符号为“D”。例如：球面镜可以表示为-2.00Ds，即可矫正200度近视；+3.00Ds表示矫正300度远视。

如果镜片顶焦度大于眼睛的屈光度，眼镜会造成超过屈光度部分新的屈光不正；如果镜片顶焦度小于眼睛的屈光度，眼镜就不能完全矫正屈光不正[1]  。配装眼镜中眼镜镜片顶焦度偏差必须符合GB10810.1-2005《眼镜镜片》标准要求。

# 电脑常见问题解决方法

## DOS命令行清屏命令

DOS命令行清屏命令:cls

## 怎么解决电脑为只能上QQ不能上网

电脑只能上qq不能打开网页主要由：

lsp出错;

DNS设置错误;

系统中毒;

三大问题导致，下面来一一介绍

1.lsp是tcp/ip等协议的接口，而打开网页就需要这两个协议的支持，只能上qq不能上网大部分都是由于这一问题导致；

2.DNS是用来解析网址的，简单的说你在上网时输入的网址要通过DNS转换成与之相对应的IP地址，如果这个环节出错了那么也会导致电脑只能上qq不能打开网页；

3.现在有不少病毒和木马会劫持浏览器，一旦用户电脑中毒之后，会锁定主页或者修改hosts文件等来达到不能上网的目的，这样用户就无法求助，甚至可以将杀毒软件的云查杀功能废除。

1.我们要修复lsp，可以通过系统命令和lsp修复工具进行修复，这里教大家用系统命令进行修复还原。点击“开始”，“运行”，输入“cmd”回车，然后输入“netsh winsock reset”回车，这样就可以修复lsp了，最好重启下电脑再查看只能上qq不能上网的问题是否解决。

2.重新设置DNS，大多数情况DNS我们都是选择的“自动获得DNS服务器地址”，如果需要更换DNS服务器地址，那么一定要根据你所在地更换，例如你在北京，那么就要选北京的DNS服务器地址。大家可以参考巨盾提供的国内各地DNS服务器IP地址进行更换。

3.对于电脑中毒引起的问题，可以先用杀毒软件或木马专杀工具进行杀毒，然后再检查hosts文件有没有还原，hosts文件存放在 “C:\WINDOWS\system32\drivers\etc”目录下，大多数时候我们都可以将这个文件清空，避免hosts文件导致只能上qq不能打开网页的问题。

# 事件处理总结

## 问题现象及应急

## 原因

## 后续处理措施

## 经验教训