Java/JavaEE

学习笔记

作者：续祥

2008年11月1日

内容目录

==========

Unix 学习笔记

==========

主讲：程欣

一、Unix 前言

1969 年在贝尔实验室发明了 Unix
非常稳定
扩展名没什么用
Solaris 10: 5.11
服务器：三大公司垄断（IBM HP SUN）

二、Unix 基础知识

1、什么叫 daemon(精灵进程):
name pid ppid
A 1000
B 1001 1000
操作系统管理：
子进程在结束的时候，父进程要回收子进程的资源
比如说进程号，进程的 V(Virtual)表

如果子进程结束了，父进程没有时间回收子进程的资源，子进程叫：僵死进程
如果父进程提前结束，子进程叫孤儿进程
init 1 号进程，管理操作系统所有的孤儿进程，孤儿院进程

daemon（后台常驻留的精灵进程）：

1. 孤儿进程
2. 必须脱离终端
3. 必须 chdir("/"),让当前的程序可以卸载。

2、登录服务器

右键打开终端
telnet 192.168.1.250
login : briup
passwd: briup(不显示)
id（显示坐标）
su(切换用户)
passwd（修改密码）
pwd（显示当前路径）
users(显示当前在线用户)
who（详细显示当前在线人数）
csh
tcsh
bash

telnet 192.168.1.250
login:xuxiang
new password:

三、Unix 常用命令

1、关于档案/目录处理的指令:

1） ls
这是最基本的档案指令。 ls 的意义为 "list"，也就是将某一个目录或是某一个档案的内容显示出来。

如果你在 ls 指令後头没有跟任何的档案名，它将会显示出目前目录中所有档案。

也可以在 ls 後面加上所要察看的目录名称或档案的名称，如

% ls /home2/X11R5

% ls first

ls 有一些特别的参数，可以给予使用者更多有关的资讯，如下:

-a : 在 UNIX 中若一个目录或档案名字的第一个字元为 "." , 则使用 ls
将不会显示出这个档案的名字，我们称此类档案为隐藏档。如 tcsh

的初设档 .tcshrc；如果我们要察看这类档案，则必须加上参数 -a 。

-l : 这个参数代表使用 ls 的长(long)格式，可以显示更多的资讯，
如档案存取权，档案拥有者(owner)，档案大小，档案最後更新
日期，甚而 symbolic link 的档案是 link 那一个档等等。如下

% ls -l

drwx--x--x 2 jjtseng 512 Aug 8 05:08 18
drwx--x--x 2 jjtseng 512 Aug 8 22:00 19
-rw------- 1 jjtseng 566 Aug 8 05:28 makefile

2） cp
cp 这个指令的意义是复制("COPY") , 也就是将一个或多个档案复制成另
一个档案或者是将其复制到另一个目录去。

cp 的用法如下:

cp f1 f2 : 将档名为 f1 的档案复制一份为档名为 f2 的档案。
cp f1 f2 f3 ... dir : 将档案 f1 f2 f3 ... 都以相同的档名复制一
份放到目录 dir 里面。
cp -r dir1 dir2 : 将 dir1 的全部内容全部复制到 dir2 里面。

cp 也有一些参数，如下:

-i : 此参数是当已有档名为 f2 的档案时，若迳自使用 cp 将会将原来 f2
的内容掩盖过去，因此在要盖过之前必须先询问使用者一下。如使用者
的回答是 y(yes)才执行复制的动作。

-r : 此参数是用来做递回复制用，可将一整颗子树都复制到另一个
目录中。

3） mv
mv 的意义为 move , 主要是将一档案改名或换至另一个目录。如同 cp ，它也有
三种格式:

mv f1 f2 : 将档名为 f1 的档案变更成档名为 f2 的档案。
mv dir1 dir2 : 将档名为 dir1 的目录变更成档名为 dir2 的目录。
mv f1 f2 f3 ... dir : 将档案 f1 f2 f3 ... 都移至目录 dir 里面。

mv 的参数有两个，-f 和 -i , 其中 -i 的意义与 cp 中的相同，均是 interactive 询问之意。而 -f 为强迫(force) , 就是不管有没有同名的档案，反正我就是要搬过去，所有其他的参数遇到 -f 均会失效。

4） rm

rm 的意义是 remove ，也就是用来杀掉一个档案的指令。在 UNIX 中一个被杀掉的档案除非是系统恰好有做备份，否则是无法像 DOS 里面一样还能够救回来的。所以在做 rm 动作的时候使用者应该要特别小心。

rm 的格式如下:

rm f1 f2 f3 .....

而 rm 的参数比较常用的有几个: -f , -i , 与 -r

-f : 将会使得系统在删除时，不提出任何警告讯息。
-i : 在除去档案之前均会询问是否真要除去。
-r : 递回式的删除。

小心不要随便使用 rm -rf , 否则有一天你会"欲哭无泪".....

5） mkdir

mkdir 是一个让使用者建立一个目录的指令。你可以在一个目录底下使用 midir 建立一个子目录，使用的方法如下:

mkdir dirname1 [dirname2 ...]

如此你就可以建立一个或多个目录。

6） chdir (cd)

这是让使用者用来转移工作目录用的。
chdir 的用法如下:

chdir dirname

如此你就可以将目前的目录转移到 dirname 这一个目录去。
或使用 "chdir .." 来转移到上一层目录。

7） rmdir

相对於 mkdir ，rmdir 是用来将一个"空的"目录杀掉的。如果一个目录下面

没有任何档案，你就可以用 rmdir 指令将其除去。rmdir 的使用法如下:

rmdir dirname1 [dirname2 ....]

如果一个目录底下有其他的档案， rmdir 将无法将这个目录杀掉，除非使用 rm 指令的 -r 选项。

8） pwd
pwd 会将目前目录的路径(path)显示出来，例如:

9） cat/more/less
以上三个指令均为察看档案内容的指令。cat 的意义是猫....不不不，是 concatenate ，在字典上的意思是"连结,将…串成锁状"(语出资工电子词典 cdict)，其实就是把档案的内容显示出来的意思。 cat 有许多奇怪的参数，较常为人所使用的是 -n 参数，也就是把显示出来的内容加上行号。 cat 的用法如下:

cat [-n] :自标准输入读进内容，你可以用 pipe 将别的程式的输出转向给 cat .
cat [-n] filename : 将 filename 的内容读进来，显示在标准输出上。

问题在於 cat 它是不会停下来的，因此并不好用(试想如果一个萤幕二十四行，而一个档案四百行，cat 一出来将会劈哩啪啦不断的卷上去，使用者很难据此得到他们所需的资讯。) 所以才有人又写了 more 出来。

more , 跟据蔡文能老师的说法是"再多一点就好"，more 可以将所观察的档案跟据终端机的形态一页页的显示出来，再根据使用者的要求换页或卷行。如果使用者要在某一个档案中搜寻一个特定的字串，则按 / 然後跟著打所要搜寻的单字即可进行搜寻。more 也可以找得到。more 的使用法如下:

more filename

如果你在使用中觉得已经看到了所要看的部份，可以按'q'离开 more 的使用。在使用中按'v' 亦可以使用编辑器来编辑所观看的档案。

less 的用法与 more 极类似，原先它就是为了弥补 more 只能往前方卷页的缺点而设计。 less 的用法如下:

less filename

其与 more 不同的是它可以按 y 来往上卷一行，并且可以用"?"来往回搜寻你所要找的单字。

10） chmod
chmod 为变更档案模式用(change mode) . 这个指令是用来更改档案的存取模式(access mode)。在 UNIX 一个档案上有可读(r)可写(w)可执行(x)三种模式, 分别针对该档案的拥有者(onwer)、同群者(group member)(你可以 ls -lg 来观看某一档案的所属的 group)，以及其他人(other)。一个档案如果改成可执行模式则系统就将其视为一个可执行档，而一个目录的可执行模式代表使用者有进入该目录之权利。chmod 就是用来变更一些档案的模式，其使用方式如下:

chmod [-fR] mode filename ...

其参数的意义如下:

-f Force. chmod 不会理会失败的动作。
-R Recurive. 会将所有子树下的所有子目录及档案改为你所要改成的模式。

mode 可以为一个三位或四位的八进位数字，来表示对某些对象的存取权。详情可参阅 chmod(1) 的 manual page 中有关 Absolute Modes 的说明。

或是用一个字串来表示，请参考 chmod(1) 的说明。

2、 关於 Process 处理的指令:

1） ps
ps 是用来显示目前你的 process 或系统 processes 的状况。
以下列出比较常用的参数:

其选项说明如下:
-a 列出包括其他 users 的 process 状况。
-u 显示 user - oriented 的 process 状况 。
-x 显示包括没有 terminal 控制的 process 状况 。
-w 使用较宽的显示模式来显示 process 状况 。

我们可以经由 ps 取得目前 processes 的状况，如 pid , running state 等。

2） kill
kill 指令的用途是送一个 signal 给某一个 process 。因为大部份送的都是

用来杀掉 process 的 SIGKILL 或 SIGHUP，因此称为 kill。kill 的用法为:

```
kill [ -SIGNAL ] pid ...
kill -l
```

SIGNAL 为一个 singal 的数字，从 0 到 31，其中 9 是 SIGKILL，也就是一般用来杀掉一些无法正常 terminate 的讯号。其馀讯号的用途可参考 sigvec(2)中对 signal 的说明。

你也可以用 kill -l 来察看可代替 signal 号码的数目字。kill 的详细情形请参阅 man kill。

3、 关於字串处理的指令:

1） echo

echo 是用来显示一字串在终端机上。□ echo -n 则是当显示完之後不会有跳行的动作。

2） grep/fgrep

grep 为一过滤器，它可自一个或多个档案中过滤出具有某个字串的行，或是自标准输入过滤出具有某个字串的行。

fgrep 可将欲过滤的一群字串放在某一个档案中，然後使用 fgrep 将包含有属於这一群字串的行过滤出来。

grep 与 fgrep 的用法如下:

```
grep [-nv] match_pattern file1 file2 ....
fgrep [-nv] -f pattern_file file1 file2 ....
```

-n 把所找到的行在行前加上行号列出
-v 把不包含 match_pattern 的行列出
match_pattern 所要搜寻的字串
-f 以 pattern_file 存放所要搜寻的字串

4、 网路上查询状况的指令:

1） man

man 是手册 (manual) 的意思。 UNIX 提供线上辅助(on-line help)的功能，

man 就是用来让使用者在使用时查询指令、系统呼叫、标准程式库函式、各种表格等的使用所用的。man 的用法如下:

man [-M path] [[section] title] .....
man [-M path] -k keyword ...

-M path man 所需要的 manual database 的路径。
我们也可以用设定环境变数 MANPATH 的方式来取代 -M 选项。
title 这是所要查询的目的物。
section 为一个数字表示 manual 的分类，通常 1 代表可执行指令，2 代表系统呼叫(system call)，3 代表标准函数，等等。

像下面 man 查询的片段:

SEE ALSO
apropos(1), cat(1V), col(1V), eqn(1), lpr(1), more(1),
nroff(1), refer(1), tbl(1), troff(1), vgrind(1), vtroff(1),
whatis(1), eqnchar(7), man(7), catman(8)

我们如要参考 eqnchar(7) 的资料，则我们就输入 man 7 eqnchar，
便能取得我们所要的辅助讯息。
-k keyword
用来将含有这项 keyword 的 title 列出来。

man 在 UNIX 上是一项非常重要的指令，我们在本讲义中所述之用法均仅只是一个大家比较常用的用法以及简单的说明，真正详细的用法与说明还是要请你使用 man 来得到。

2） who
who 指令是用来查询目前有那些人在线上。

3） w

w 指令是用来查询目前有那些人在线上，同时显示出那些人目前的工作。

4） ku
ku 可以用来搜寻整个网路上的 user，不像 w 跟 who 只是针对 local host 的查询. 而且 ku 提供让使用者建立搜寻特定使用者名单的功能。你可以建立一个档案 information-file 以条列的方式存放你的朋友的资料，再建立一个档案 hosts-file 来指定搜寻的机器名称。 ku 的指令格式可由 ku -h 得到。

5、 网路指令:

UNIX 提供网路的连接，使得你可以在各个不同的机器上做一些特殊的事情，如你可以在系上的 iris 图形工作站上做图形的处理，在系上的 Sun 上读 News，甚至到学校的计中去找别系的同学 talk。这些工作可以利用 UNIX 的网路指令，在你的位子上连到各个不同的机器上工作。如此一来，即使你在寝室，也能轻易的连至系上或计中来工作，不用像以前的人必须泡在冷冰冰的机房面。

这些网路的指令如下所述:

1） rlogin 与 rsh
rlogin 的意义是 remote login , 也就是经由网路到另外一部机器 login。
rlogin 的格式是:

rlogin host [-l username]

选项 -l username 是当你在远方的机器上的 username 和 local host 不同的时後，必须输入的选项，否则 rlogin 将会假设你在那边的 username 与 local host 相同，然後在第一次 login 时必然会发生错误。

rsh 是在远方的机器上执行某些指令，而把结果传回 local host。rsh 的格式如下:

rsh host [-l username] [command]

如同 rlogin 的参数 -l username , rsh 的 -l username 也是指定 remote host 的 username。而 command 则是要在 remote host 上执行的指令。如果没有指定 command，则 rsh 会去执行 rlogin，如同直接执行 rlogin。

不过 rsh 在执行的时候并不会像一般的 login 程序一样还会问你 password , 而是如果你没有设定 trust table , 则 remote host 将不会接受你的 request。

rsh 须要在每个可能会做为 remote host 的机器上设定一个档案，称为 .rhosts。这个档案每一行分为两个部份，第一个是允许 login 的 hostname , 第二个部份则是允许 login 的 username。例如，在 ccsun7.csie.nctu.edu.tw 上头你的 username 为 ysjuang , 而你的 home 下面的 .rhost 有以下的一行:

ccsun6.cc.nctu.edu.tw u8217529

则在 ccsun6.cc.nctu.edu.tw 机器上的 user u8217529 就可以用以下的方法来

执行 rsh 程式:

% rsh ccsun7.csie.nctu.edu.tw -l ysjuang cat mbox

将 ysjuang 在 ccsun7.csie.nctu.edu.tw 上的 mbox 档案内容显示在 local host ccsun6.cc.nctu.edu.tw 上。

而如果 .rhost 有这样的一行，则 ccsun6.cc.nctu.edu.tw 上的 user u8217529 将可以不用输入 password 而直接经由 rsh 或 rlogin login 到 ccsun7.csie.nctu.edu.tw 来。

注意:

.rhost 是一个设定可以信任的人 login 的表格，因此如果设定不当将会让不法之徒有可以乘机侵入系统的机会。 如果你阅读 man 5 rhosts ，将会发现你可以在第一栏用 + 来取代任何 hostname ，第二栏用 + 来取代任何 username 。

如一般 user 喜欢偷懒利用 " + username " 来代替列一长串 hostname ，但是这样将会使得即使有一台 PC 上跑 UNIX 的 user 有与你相同的 username , 也可以得到你的 trust 而侵入你的系统。这样容易造成系统安全上的危险。因此本系禁止使用这样子的方式写你的 .rhost 档，如果发现将予以停机直到你找中心的工作人员将其改正为止。 同理，如果你的第二个栏位为 + ，如" hostname + " ,则你是允许在某一部机器上的"所有" user 可以不用经由输入 password 来进入你的帐号，是壹种更危险的行为。所以请自行小心。

2） telnet
telnet 是一个提供 user 经由网路连到 remote host。
telnet 的 格式如下:

telnet [hostname | ip-address] [port]

hostname 为一个像 ccsun1 或是 ccsun1.cc.nctu.edu.tw 的 name address，ip-address 则为一个由四个小於 255 的数字组成的 ip address ，如 ccsun1 的 ip-address 为 140.113.17.173 ，ccsun1.cc.nctu.edu.tw 的 ip-address 为 140.113.4.11 。你可以利用 telnet ccsun1 或 telnet 140.113.17.173 来连到 ccsun1。

port 为一些特殊的程式所提供给外界的沟通点，如资工系的 MUD 其 server 便提供一些 port 让 user 由这些 port 进入 MUD 程式。详情请参阅 telnet(1) 的说明。

3） ftp

ftp 的意义是 File Transfer Program，是一个很常应用在网路档案传输的程式。ftp 的格式如下:

ftp [hostname | ip-address]

其中 hostname | ip-address 的意义跟 telnet 中的相同。

在进入 ftp 之後，如果与 remote host 连接上了，它将会询问你 username 与密码，如果输入对了就可以开始进行档案传输。

在 ftp 中有许多的命令，详细的使用方式请参考 ftp(1)，这里仅列出较常用的 cd , lcd , mkdir , put , mput , get , mget , binary , ascii , prompt , help 与 quit 的使用方式。

ascii 将传输模式设为 ascii 模式。通常用於传送文字档。

binary 将传输模式设为 binary 模式，通常用於传送执行档，压缩档与影像档等。

cd remote-directory 将 remote host 上的工作目录改变。

lcd [directory] 更改 local host 的工作目录。

ls [remote-directory] [local-file] 列出 remote host 上的档案。

get remote-file [local-file] 取得远方的档案。

mget remote-files 可使用通用字元一次取得多个档案。

put local-file [remote-file] 将 local host 的档案送到 remote host。

mput local-files 可使用通用字元一次将多个档案放到 remote host 上。

help [command] 线上辅助指令。

mkdir directory-name 在 remote host 造一个目录。

prompt 更改交谈模式，若为 on 则在 mput 与 mget 时每作一个档案之传输时均会询问。

quit/bye 离开 ftp .

利用 ftp，我们便可以在不同的机器上将所需要的资料做转移，某些特别的机器更存放大量的资料以供各地的使用者抓取，本校较著名的 ftp server 有 NCTUCCCA 与系上的 ftp.csie.nctu.edu.tw 。这些 ftp server 均有提供一个 user 称为 anonymous，一般的"外来客"可以利用这个 username 取得该 server 的公共资料。不过 anonymous 在询问 password 时是要求使用 anonymous 的使用者输入其 email address，以往有许多台湾的使用者在使用国外的 ftp server 时并没有按照人家的要求输入其 email address，而仅是随便打一些字串，引起许多 internet user 和管理者的不满，对台湾的使用者的风评变得很差，因此遵循各 ftp server 的使用规则也是一件相当重要的事。

6、 关於通讯用的指令:

1） write
这个指令是提供使用者传送讯息给另一个使用者，使用方式:
write username [tty]

2） talk/ytalk/cytalk/ctalk
UNIX 专用的交谈程式。会将萤幕分隔开为你的区域和交谈对象的区域，同时也可和不同机器的使用者交谈。使用方式:

talk username[@host] [tty]

3） mesg
选择是否接受他人的 messege , 若为 messege no 则他人的 messege 将无法传送给你，同时他也无法干扰你的工作。使用方法:

mesg [-n|-y]

4） mail/elm
在网路上的 email 程式，可经由此程式将信件 mail 给他人。 使用方式:

mail [username]
mail -f mailboxfile

如有信件，则直接键入 mail 可以读取你的 mail .

elm 提供较 mail 更为方便的介面，而且可做线上的 alias . 你可以进入 elm 使用上下左右键来选读取的信件，并可按 h 取得线上的 help 文件。

使用方式:

elm [usernmae]
elm -f mailboxfile

7、 编译器(Compiler):
Compiler 的用处在於将你所撰写的程式翻译成一个可执行档案。在资工系常用的程式语言是 C , pascal , FORTRAN 等。你可以先写好一个 C 或 Pascal 或 FORTRAN 的原始程式档，再用这些 compiler 将其翻成可执行档。你可以用这个方法来制造你自己的特殊指令。

1） cc/gcc (C Compiler)
/usr/bin/cc
/usr/local/bin/gcc

语法: cc [-o execfile] source
gcc [-o execfile] source

execfile 是你所希望的执行档的名称，如果没有加上 -o 选项编译出来的可执行档会以 a.out 做为档名。 source 为一个以 .c 做为结尾的 C 程式档。请参阅 cc(1) 的说明。

2） pc (Pascal Compiler)
/usr/local/bin/pc

语法: pc [-o execfile] source

execfile 是你所希望的执行档的名称，如果没有加上 -o 选项编译出来的可执行档会以 a.out 做为档名。 source 为一个以 .p 做为结尾的 Pascal 程式档。 请参阅 /net/home5/lang/man 中 pc(1) 的说明。

3） f77 (Fortran Compiler)
/net/home5/lang/f77

语法: f77 [-o execfile] source

execfile 是你所希望的执行档的名称，如果没有加上 -o 选项编译出来的可执行档会以 a.out 做为档名。 source 为一个以 .p 做为结尾的 FORTRAN 程式档。

8、 有关列印的指令:
以下为印表所会用到的指令，在本系的印表机有 lp1 , lp2 (点矩阵印表机), lw , sp , ps , compaq (雷射印表机)，供使用者使用。

1） lpr
lpr 为用来将一个档案印至列表机的指令。

用法:
lpr -P[printer] file1 file2 file3 ....

或
lpr -P[printer] < file1

例子:
lpr -Plp1 hello.c hello.lst hello.map
lpr -Plp1 < hello.c

前者以参数输入所要印出的档案内容，後者列印标准输入档案(standard input)的内容，因已将 hello.c 转向到标准输入，故会印出 hello.c 的档案内容。

2） lpq
lpq 是用来观察 printer queue 上的 Jobs 。

用法:
lpq -P[printer]

3） lprm
lprm 是用来取消列印要求的指令。 通常我们有时会印错，或是误送非文字档资料至 printer , 此时就必须利用 lprm 取消列印 request ，以免造成资源的浪费。

用法:
lprm -P[printer] [Jobs id | username]

lprm 用来清除 printer queue 中的 Jobs , 如果你使用 Job Id 作为参数，则它将此 Job 自 printer queue 清除，如果你用 username 作为参数，则它将此 queue

中所有 Owner 为此 username 的 Jobs 清除。

9、 更改个人使用资料:

1） passwd
passwd 是用来更改你的使用密码，用法为:

passwd [username]

在使用 passwd 的时候，它会先问你的旧密码，然後询问两次要更改的密码，确定
无误後才将你的密码改掉。

2） chsh
chsh 是提供使用者更换 login shell 的指令，你可经由此更换自己使用的 shell 。

四、 文件系统权限

权限	文件	目录
r（read）	cat 读取文件内容	ls 查看此目录
w (write)	修改文件内容	在此目录下删除，创建文件
x (execute)	执行此文件	cd 进入此目录

实例：浙江移动的验证授权系统的实现
****
营业厅里有很多角色:
r 4 00000100
w 2 00000010
z 1 00000001
p 5 00000101
&（位于运算）： 相同位置留下，不同取 0
if(p&4==4)具有读权限
if(p&4==2)具有写权限
if(p&4==1)具有执行权限

p 如果营业员权限为 p，如何判断此人具有读权限

浙江移动具如下权限：
登录 换卡 换号 换品牌 补卡 授权 重起 出帐

1 2 4 8 16 32 64 128

if(p=a+b+c)如果a,b,c都是不重复的2的n次方，则p和a,b,c做&运算肯定等于a,b,c
营业员：
p=1+2+16=19(权重)
if(p&1==1)可以登录
if(p&2==2)可以换卡
if(p&4==4)可以换号
****
练习：
1.熟练掌握各个命令
2.用chmod控制你home目录下的每个文件和目录
验证一点：如果把目录权限封住了，是否可以操作目录下的文件内容（cat）--->验证结果：不可以！
ls -la
==========

五、软硬连接的区别

1.软连接可以连接目录，硬连接不可以；
2.软连接可以跨分区，而硬连接不可以；
3.软连接依赖原文件，而硬连接不依赖。

六、范例程序

1、King.java(金字塔)：

```
/**
        % vi King.java
        % source   /etc/.login
        % javac King.java
        % java King
*/
public class King{
        public static void main(String args[]){
                for(int i=0;i<10;i++){
                        for(int j=10;j>i;j--){
                                System.out.print(" ");
                        }
```

```
                        for(int j=0;j<i*2+1;j++){
                                System.out.print("$");
                        }
                System.out.println();
                }
        }
}
```

2、Server.java（聊天室）：

```
import java.io.*;
import java.net.*;
import java.util.*;

public class Server{
        public static  Vector users=new Vector();;

        public static void main(String args[]){
                int port=10000;
                ServerSocket server=null;
                try{
                        server=new ServerSocket(port);
                        System.out.println("server start up on port "+port);
                        while(true){
                                Socket s=server.accept();
                                User user=new User(s);
                                user.start();
                        }
                }catch(Exception e){
                        e.printStackTrace();
                }finally{
                        if(server!=null){
                try{server.close();}catch(Exception e){}
                        }
                }
        }
}
class User extends Thread{
        private String userName;
        private InetAddress ip;
        private Socket s;
        public User(Socket s){
                this.s=s;
        }
        public void run(){
                InputStream input=null;
                OutputStream output=null;
                try{
```

```
input=s.getInputStream();
output=s.getOutputStream();
PrintWriter pw=new PrintWriter(
new OutputStreamWriter(output));
BufferedReader buf=new BufferedReader(new InputStreamReader(input));
pw.print("enter your name :");
pw.flush();
userName=buf.readLine();
ip=s.getInetAddress();
Server.users.add(this);
sendAll(userName+" has joid us");
long t=0;
while(true){
        String line=buf.readLine();
        if(line.equals("bye")){
                sendAll(userName+" has leave us ");
                Server.users.remove(this);
                break;

        }
        if(line.equals("users")){
                Iterator it=Server.users.iterator();
                while(it.hasNext()){
User user=(User)it.next();
String n=user.getUserName();
InetAddress i=user.getIp();
pw.println(i+"\t"+n);

                }
pw.flush();
        continue;
        }


if(line==null||line.trim().length()==0)continue;
long s=System.currentTimeMillis();
if(t!=0&&(s-t)/1000<3){
        pw.println("flush denied ");
        pw.flush();
        continue;
}
t=s;
if(line.startsWith("kill")){
        try{
StringTokenizer stk=new StringTokenizer(line);
        stk.nextElement();
        String u=(String)stk.nextElement();
                Iterator it=Server.users.iterator();
                while(it.hasNext()){
User user=(User)it.next();
```

```
                String n=user.getUserName();
                if(n.equals(u)){
                        sendAll(n+" was killed by "+userName);
                        it.remove();
                        user.interrupt();
                        break;
                }
                                }

                        }catch(Exception e){

                        }

                }

                if(line.equals(":)")) {
                        sendAll(userName+" 嘿嘿干笑了一声，统统告诉我密码");
                        continue;
                }
                if(line.equals(":(")){
                        sendAll(userName+" 耷拉着脸说：我今天不太高兴，不要惹我 ");
                        continue;
                }
                sendAll(userName+" said : "+ line);

                }
        }catch(Exception e){
                e.printStackTrace();
        }finally{
                if(s!=null)try{s.close();}catch(Exception e){}
        }
}

public synchronized void sendMessage(String message){

        OutputStream output=null;
        try{
                output=s.getOutputStream();
                PrintWriter pw=new PrintWriter(
                new OutputStreamWriter(output));
                pw.println(message);
                pw.flush();

        }catch(Exception e){}
}
public void sendAll(String message){
        Iterator it=Server.users.iterator();
        while(it.hasNext()){
                User user=(User)it.next();
```

```
                    user.sendMessage(message);
                }
        }

        public String getUserName(){return userName;}
        public InetAddress getIp(){return ip;}
}
```

登录命令：telnet 192.168.1.250 10000

==========

七、网络基本命令

1、/etc/inet/inetd.conf 是 solaris10 以前的配置方式，现在已经作废(vi /etc/inet/inetd.conf)

2、输入命令：vi /etc/services

```
"/etc/services" [Read only] 119 lines, 4228 characters
swat            901/tcp                 # Samba Web Adm.Tool
#
# The following customer-specific entries were found in the services file
# prior to an upgrade.  Note that service names and their corresponding
# port numbers must be registered with IANA, http://www.iana.org, and
# entries not registered as such may not be preserved automatically by
# future upgrades.
#
tcpmux          1/tcp
echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
systat          11/tcp          users
daytime         13/tcp
daytime         13/udp
netstat         15/tcp
chargen         19/tcp          ttytst source
chargen         19/udp          ttytst source
ftp-data        20/tcp
ftp             21/tcp
ssh             22/tcp                  # Secure Shell
telnet          23/tcp
"/etc/services" [Read only] 119 lines, 4228 characters
```

解释：20 口传命令，21 口传文件

3、新版本的 solaris 中，靠 svcs 命令来管理所有的服务

svcs 列出所有的服务

svcadm enable telnet

svcadm disable telnet
svcadm disable ftp
svcadm enable ftp
svcadm disable ssh
svcamd enable ssh

ftp-data 20
ftp 21
ssh 22
telnet 23
smtp 25
web 80
pop3 110

==========

常见应用软件：
oracle 1521
tomcat 8080
resin 8080
weblogic 7001
pointbase 9092
mysql 3306

1000 口内的程序默认是为超级用户准备的
1000 内的端口默认是为系统服务准备的
自定义的程序必须大于 1000 口，如果小于 1000，
则必须用超级用户(root,Administrator)来运行

ssh briup@192.168.1.250
yes

3、查看所经历的路由：
/usr/sbin/traceroute 192.168.1.1
/usr/sbin/traceroute 61.129.57.158
traceroute to 61.129.57.158 (61.129.57.158), 30 hops max, 40 byte packets
1 192.168.1.1 (192.168.1.1) 0.198 ms 0.094 ms 0.085 ms
2 * * *
3 *^C$

telnet:
apache web 服务器
resin/tomcat java 应用服务器
squid 缓存服务器 就是给图片和网页在内存中缓存起来

rlogin 192.168.1.250 -l briup

Password:
Last login: Wed Apr 30 10:43:43 from 192.168.3.74
Sun Microsystems Inc. SunOS 5.10 Generic January 2005
You have new mail.

4、ftp:下载和上传文件
ftp 192.168.1.250
user : briup
pass: briup

cd 切换服务器上的目录
lcd 切换本地目录
dir/ls 查看服务器上的文件列表
bye 断开，退出
bin 传输二进制文件（图片，电影，doc，ppt 都是）
asc 传输文本文件 （.txt .java 写字板打开肉眼能识别的）
用 bin 模式肯定不会错的
get 下载
put 上传
mget 下载多个
mput 上传多个
hash 打印进度条
prompt 下载，上传前是否提示

八、Shell scripts:脚本编程

1、脚本
.sh 意味着本脚本是用 sh 的语法
.csh cshell 的语法
.bash bash 的语法
注意：95%的脚本是用 sh 写的，比较通用
#!/bins/sh 声明本 shell 遵守 bourne shell 的格式

2、在 shell 中
$* 代表命令行传进来的多个参数
$1 第一
$2 第二个
`命令率先执行,将结果赋值给等号左边` 点

'字符串赋值给等号左边' 单引号

A=`echo hello` A 是 hello

B='echo hello' B 是 echo hello

3、lcf.sh
vi lcf.sh

```
#!/bin/sh

for oldname in $*
do
        if [ -f $oldname ]
        then
        newname=`echo $oldname |tr '[A-Z]' '[a-z]'`
                if [ -f $oldname != $newname ]
                then
                        mv $oldname $newname
                fi

        fi

done
```

:wq

```
# chmod u+x lcf.sh
# touch A B C
# lcf.sh A B C
# ls
```

4.crontab:定时调用

/etc/cron.d/cron.allow
/etc/cron.d/cron.deny 两个文件用于对 crontab 进行访问控制

我喜欢的用法：
$crontab -l > a
$vi a //编辑 a 文件
$crontab a 重新初始化 crontab 列表

另外的用法：
$crontab -e //然后对 crontab 文件进行编辑，最后保存。

当第一次使用 crontab 的时候可能会报一个错误，告诉你没有临时文件

$crontab a //随便找个文件初始化一下就可以
==========

crontab 文件格式
* * * * * /bin/ls

分钟 小时 天 月 星期几 /bin/ls 可执行程序

briupserver4% setenv EDITOR vi
briupserver4% crontab -e
"/tmp/crontabo8aGOk" 0 lines, 0 characters
* * * * * date >> /temp/xuxiang.txt
briupserver4% tail -f /temp/xuxiang.txt
tail: cannot open input
briupserver4% crontab -l
* * * * * date >> /temp/xuxiang.txt
briupserver4% tail -f /temp/xuxiang.txt
tail: cannot open input
briupserver4% crontab -r
briupserver4% rm /temp/xuxiang,txt
/temp/xuxiang,txt: No such file or directory
briupserver4% crontab -l
crontab: can't open your crontab file.

crontab 是超级用户运行的，所以你在脚本中打印是看不到结果的，应该把结果>>某个文件中,文件也要全路径，不能使用 ~

crontab 中加载的顺序：
登录
1. /etc/.login
2.~/.cschr
3.~/.login
登出
1、~/ .logout
这些文件如何产生?
1. cp /etc/skel/* ~
2.mv ~/local.cshrc ~/.cshrc
3.vi ~/ .cshrc
加入自定义命令
4.立即生效: source ~/.cshrc
或者重新登录
source 是 csh 专有命令

在环境变量中，$具有特殊的含义：

“$...”中$是带有含义的

‘$...’中就是一个字符而已，没有任何含义

path 是查找命令的位置和顺序
briupserver4% echo $path
/bin /usr/bin /usr/ucb /etc .
briupserver4% ping
ping: Command not found
briupserver4% set path=($path /usr/sbin)
briupserver4% echo $path
/bin /usr/bin /usr/ucb /etc . /usr/sbin

777-033=744-执行权限＝644
%ls -l udir
udir: No such file or directory
%umask
22
%umask 255
%mkdir abc
%ls -la abc
total 4
dr-x-w--w- 2 xuxiang ajd0805 512 Apr 30 15:59 .
drwxrwxrwx 3 xuxiang ajd0805 512 Apr 30 15:59 ..
%touch beans
%ls -la beans
-r---w--w- 1 xuxiang ajd0805 0 Apr 30 15:59 beans

Unix 笔记于 2008 年 5 月 1 日全部完成！！！

Core Java 学习笔记

==========

主讲：丁长明(Kevin Ding)

第一章 JAVA 前言

1、JAVA 的特点

1) 简单(Java 语法是 C++语法的一个“纯净”版本)；

2) 可移植性 (一次编译到处运行)

3) 面向对象

4) 分布式(Java 把打开套接字连接等繁琐的网络任务变得非常容易)
5) 健壮性(Java 编译器会检查出很多其他语言在运行时刻才显示出来的错误；Java 采用的指针模型可以消除重写内存和数据崩溃的可能)
6) 多线程(多线程编程的简单性是 Java 成为流行的服务器端开发语言的主要原因之一)
7) 安全(用 Java 可以构建防病毒和防篡改的系统)
9) 动态(Java 可随意增加新的方法以及实例变量，而客户端却不需做任何的更改)
10)体系结构中立(字节码与计算机体系结构无关，只要存在运行时系统，可在多种处理器上执行)

2、JAVA 的运行原理
1)编译: *.java 文件 ----------> *.class 文件
2)运行: *.class 文件 -----> 加载 -----> JVM（JAVA 虚拟机）

3、JAVA 的目录
1）JRE ——————— 运行环境
2）SRC ——————— 类库
3）BIN ——————— 应用程序(二进制，JAVA 工具)
4）demo ——————— 范例
5）include ——————— C 语言

4、JDK (Java Development Kit)

JDK 是整个 Java 的核心，包括了 Java 运行环境（Java Runtime Envirnment），一堆 Java 工具和 Java 基础的类库(rt.jar)。不论什么 Java 应用服务器实质都是内置了某个版本的 JDK。因此掌握 JDK 是学好 Java 的第一步。最主流的 JDK 是 Sun 公司发布的 JDK，除了 Sun 之外，还有很多公司和组织都开发了自己的 JDK，例如 IBM 公司开发的 JDK，BEA 公司的 Jrocket，还有 GNU 组织开发的 JDK 等等。其中 IBM 的 JDK 包含的 JVM（Java Virtual Machine）运行效率要比 Sun JDK 包含的 JVM 高出许多。而专门运行在 x86 平台的 Jrocket 在服务端运行效率也要比 Sun JDK 好很多。但不管怎么说，我们还是需要先把 Sun JDK 掌握好。

1）JDK 的下载和安装（JAVA 开发包的安装）
1.下载（http://java.sun.com）：JDK 又叫做 J2SE （Java2 SDK Standard Edition），可以从 Sun 的 Java 网站上下载到，http://java.sun.com/j2se/downloads.html ，JDK 当前最新的版本是 J2SDK1.4.2，建议下载该版本的 JDK，下载页面在这里：http://java.sun.com/j2se/1.4.2/download.html。

2.安装（Linux 下直接运行 shell）：下载好的 JDK 是一个可执行安装程序，默认安装完

毕后会在 C:\Program Files\Java\目录下安装一套 JRE（供浏览器来使用），在 C:\j2sdk1.4.2 下安装一套 JDK（也包括一套 JRE）。然后我们需要在环境变量 PATH 的最前面增加 java 的路径 C:\j2sdk1.4.2\bin。这样 JDK 就安装好了。

2）JDK 的命令工具

JDK 的最重要命令行工具：

java： 启动 JVM 执行 class

javac： Java 编译器

jar： Java 打包工具

javadoc： Java 文档生成器

这些命令行必须要非常非常熟悉，对于每个参数都要很精通才行。对于这些命令的学习，JDK Documentation 上有详细的文档。

5、设置环境变量(3 个):

1）csh(服务器)

```
setenv JAVA_HOME /opt/java/jdk/jdk 1.5.0_06
setenv PATH /opt/java/jdk/jdk 1.5.0_06/bin:$PATH
setenv CLASSPATH .
```

2）bsh/ksh(本机)

```
vi .bashrc
export JAVA_HOME=/opt/java/jdk/jdk 1.5.0_06     // JDK 安装路径
export PATH=/opt/java/jdk/jdk 1.5.0_06/bin:$PATH   // 操作系统本身的，不属于 JAVA 特有的
export CLASSPATH=.   // 设置为点，默认为当前目录
```

3).检查

```
echo $PATH
. .bashrc
java
javac
```

6、类装载机制：分为四个级别

1）启动类装载器 $jdk/jre/lib/rt.jar

2) 扩展类装载器 $jdk/jre/lib/ext/*

3) 系统类装载器 $CLASSPATH

```
Helloworld------->Helloworld.class
System  System(.)
```

双亲委托机制:系统类------>扩展类----->启动类(委托),上一级别先要装载，装载不成功时才开始装载。

报出错误：classNotFound
4) 网络类装载器

作用：保证 JAVA 的优良特性

7、JAVA 程序
源文件:在最顶层只包括一个 public 类型的类/接口，文件名与类/接口名相同并以.java 作为文件后缀。
1) package: 包（一个程序的完整名称是包的名字+程序的局部名字，降低程序的命名冲突）
包:package ,在源文件中 this identify 只能放在第一行，且最多只能是一行
一个将类和接口组织在一块的实体，在文件系统中以目录/文件夹型式呈现。
2）import: 导入（导入若干个其它包中的半成品代码）
3）class: 类(程序主体)------->若干个 ，public 可以不要,当为 public class 时，程序只能是一个。
4) mathod: 入口点是固定的（public static void main）,在 C/C++中叫函数

范例：
规定包名：com.briup.ch01
规定程序（类）名：HelloJava
(1)编写: vi ch01/HelloJava.java

```
package hello.briup.ch01;
/**
 * This is my first java program,
 * which output the string "hello java,so simple"
 */
import java.lang.*;
public class HelloJava{
        /**
         * The entry of program
         */
    public static void main(String[] args){
          System.out.println("hello java, so simple");
    }
}
:wq
```

(2)编译: javac -d . ch01/HelloJava.java
解释：javac(启动编译器) -d .(当前目录下) ch01/HelloJava.java
查看目录：dir（-d 的作用：创建目录）

如果没有第一行 package，那么可以省去" -d . ",否则必须加上！
(3)运行(run)：java com.briup.ch01.HelloJava（注意这是一个完整的名字）
如果不在当前目录下，则利用命令 export CLASSPATH=？修改路径(注意？代表目录名称）--------->一般不轻易修改已设定好的路径，故很少用
或者是临时运行：java -cp ？ com.briup.ch01.HelloJava

==========

第二章 标识符，关键字，类型

1、3 种类型的注释：
1）单行注释： // 注释内容
2）多行注释： /* 注释内容 */
3）文档注释： /** 注释内容 */
示例：

```
/*
 *公司
 *程序员
 *日期
 *功能
 */
```

生成帮助文档：javadoc -d ******* *.java
示例：javadoc -d docs ch01/HelloJava.java

查看 doc 文件：*/docs/index.html
解决乱码问题：查看----->字符编码----->Unicode

设置 Tab 宽度-----> :set sts=4

2、Java 中的语法规则：
1）Java 语句以分号结束；
2）Java 的代码块包含在大括号内；
3）忽略空格“space” and 换行符“Tab”,它们无特殊含义。

3、标识符:程序元素的名字
1) 用以命名类、方法和变量、以及包；

遵守 JAVA 的命名规范；

类名的每个单词都以大写字母开头；

方法和变量名第一个单词的首字母小写，其他的大写；

2) 以字母、“_”或“$”开头；

3）大小写敏感，即大小写严格区分；

4) 无长度限制。

4、关键字：就是一类已经被启用的标识符

1）所有 Java 关键字都是小写英文字符串，不同于 C 语言中的 TRUE、FALSE、NULL 等。

2） “goto”（标号：无条件的转向标号所标注的程序位置）and “const”: Java 中不建议使用，在 JAVA 语言里无意义，但是作为关键字保留。c/c++中也不建议使用!

3） “true” , “false” and “null" ：不属于关键字，属于保留字。

5、基本数据类型:决定了内存空间的大小以及对内存空间所存放的值的类型的限制。

逻辑型；文本型；整数型；浮点型

1) 整数（数值）型

Type	Bytes	Bits	Range
byte	1B	8 位	-128 到 127
short	2B	16 位	-2^15 到 2^15-1
int	4B	32 位	-2^31 到 2^31-1
long	8B	64 位	-2^63 到 2^63-1

2) 浮点类型

float	4B	32 位	--------> 单精度
double	8B	64 位	--------> 双精度

3) 文本型（字符类型）

char	2B	16 位

4) 逻辑型（布尔型）

boolean false/true 讨论它占了多少字节是没有任何意义的，大多数情况下是占了 32bit.

注： 1）char 是无符号的 16 位整数,字面值必须用单引号括起来，例如‘a’；

2）String 是类，非原始数据类型，本质上不属于基本数据类型，属于复杂类型；

3）长整型数字有一个后缀为“L”或“l”，八进制前缀为“0”，十六进制前缀为“0x”；

4）默认整数类型为 int,默认浮点类型为 double；

5）float 数据类型必须有一个后缀为“f”或“F”,Double 数据类型后可以跟后缀“D”或“d”。

6）在 JAVA 中的编码都是 Unicode 编码，长度都是以 16bit 作为单位的，不分中英文。

7）脚下留心：String,StringBuffer,StringBuilder 的区别?

1. String：不可变字符串，在大量字符串进行连接时不使用 String；

2.StringBuffer：Java5 中的可变字符串，线程安全的；

3.StringBuilder：线程不安全的，性能较高，没有多线程的情况下进行大量字符串连接时使用；

6、内存的逻辑分区：

1) 栈区：满足 first in last out（先进后出）的分配原则 -----> 通常存放局部变量(在方法内部定义的变量)；

2) 堆区：自由存储区

c/c++: 动态分配的内存,指针变量；

java : 对象

3) 代码区：专门用于存放函数的区域。

函数指针：函数在代码区当中的地址

指针函数：函数的返回值为指针类型的函数称为指针函数

4) 静态，常量存储区

7、推荐命名规则

1) 类/接口名：单个单词的首字母大写，复合词则每个单词的首字母都大写；

2) 方法、变量名：第一个单词首字母小写，其余单词的首字母大写；

3) 常量名称：全部字母大写，单词间用“_”连接；

4) 包名：全部字母小写。

8、程序范例：

```
briup@briup:~/corejava> mkdir ch02
briup@briup:~/corejava> vi ch02/Person.java

package com.briup.ch02;

public class Person{
    String name;   //实例变量
    boolean gender;   //实例变量
    int age;    //实例变量
```

```
    public void increAge(){
        age = age + 5;
    }
    public void decreAge(){
        age = age - 5;
    }
}
:wq
```

```
briup@briup:~/corejava> javac -d . ch02/Person.java
briup@briup:~/corejava> vi ch02/PersonTest.java
```

```
package com.briup.ch02;

public class PersonTest{
    public static void main(String[] args){
    //primitiva data type declaration
    int i; //给 i 开辟了空间
    //primitiva data type assignment
    i = 5;

    //compex data type declaration
    Person p;   //局部变量 p
    p = new Person();  //new 表示创建一个对象
    p.name = "kevin ding";
    p.gender = true;
    p.age = 34;
    p.increAge();
    System.out.println(p);
    System.out.println(p.name);
    System.out.println(p.gender);
    System.out.println(p.age);
    System.out.println("**************");
    p.decreAge();
    System.out.println(p);
    System.out.println(p.name);
    System.out.println(p.gender);
    System.out.println(p.age);
```

```
    }
}
:wq

briup@briup:~/corejava> javac -d . ch02/PersonTest.java
briup@briup:~/corejava> java com.briup.ch02.PersonTest

程序运行结果：
com.briup.ch02.Person@82ba41
kevin ding
true
39
**********
com.briup.ch02.Person@82ba41
kevin ding
true
34
```

9、类与对象

语言层面	实际应用
类	群体
对象（object）	个体
实例（instance）	分配内存空间
实例化	创建对象
成员变量（member variable）	属性（attribute）
成员方法（member method）	操作（operration）
引用（reference）	唯一标识/地址
引用型变量（reference variable）	

练习：1.定义一个 Teacher 类，体现所有的老师都有姓名（String），性别（boolean），年龄（int），薪水（double）四个属性，同时每个老师都能打印自己的四个信息，增加薪水，增加年龄。

2.编写一个测试程序 TeacherTest,构建老师对象，填入该对象的四个信息（姓名，性别，年龄，薪水）值，改变该老师的薪水，年龄并打印四个信息。

3.定义一个桌子类，表示所有桌子的特点：形状（shape）、腿的个数（legCount）、颜色（color），并且提供操作可以任意修改桌子的颜色（modifyColor），增加或减少腿的个数（modifyLegCount），改变桌子的形状（modifyShape），桌子只有三种形状：circle,rectangle,square.

4.编写测试程序 TableTest,构建桌子对象，填入其信息和打印桌子的信息。

5.思考如何控制桌子腿的个数不能低于 3.

范例：

```
briup@briup:~/corejava> vi ch02/Table.java

package com.briup.ch02;

public class Table{
    String shape;
    int legCount;
    String color;


    public void modifyLegCount(int in){
        if(in>=3){
          legCount = in;
         }
         else{
         System.out.println("桌子腿数不满足要求");
          }
    }
    public void modifyColor(String in){
        color = in;
    }
    public void modifyShape(String in){
        shape = in;
    }
    public void print(){
       System.out.println(shape);
       System.out.println(legCount);
       System.out.println(color);
    }
}

briup@briup:~/corejava> javac -d . ch02/Table.java
briup@briup:~/corejava> vi ch02/TableTest.java

package com.briup.ch02;

public class TableTest{
```

```
    public static void main(String[] args){
    //primitiva data type declaration
    int i;
    //primitiva data type assignment
    i = 5;

    //compex data type declaration
    Table t;
    t = new Table();
    t.shape = "square";
    t.legCount = 3;
    t.color = "yellow";
    System.out.println(t);
    t.print();
    System.out.println("==========================");
    t.modifyShape("circle");
    t.modifyLegCount(4);
    t.modifyColor("red");
    System.out.println(t);
    t.print();
    }
}
```

```
briup@briup:~/corejava> javac -d . ch02/TableTest.java
briup@briup:~/corejava> java com.briup.ch02.TableTest
程序运行结果：
com.briup.ch02.Table@82ba41
square
3
yellow
==========================
com.briup.ch02.Table@82ba41
circle
4
red
```

==========

第三章 表达式与流控制

1、变量和作用域

1) 局部变量

$范围：定义在一个方法内部的变量；

$作用域：其作用域为所在代码块，也称为临时变量、栈变量；

$生命周期：从创建开始到方法（函数）结束为止；

$特点：在使用之前必须要对其初始化；

$一律存放在栈中。

2) 实例变量

$范围：定义在一个类内部但不在方法中的变量；

$作用域：其作用域为整个类，能够被操作的范围是类的内部；

$生命周期：对象被创建的时候开始，对象被垃圾回收的时间结束；

$特点：如未定义其初值，系统会自动对实例变量进行默认初始化（默认值为 null），使用之前必须创建类的实例；

$每一个对象的实例变量的值是相互独立的；

$一律存放在堆中。

注意：1.局部变量要先赋值，再进行运算，而实例变量均已经赋初值。这是局部变量和实例变量的一大区别；

2.局部变量不允许范围内定义两个同名变量。实例变量和局部变量允许命名冲突，但在局部变量的作用域内局部变量优先，访问实例变量须使用 this.variableName。

3）默认数值

类型	默认值
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000' 空格
boolean	false
String	null

2、操作符

1）>> and <<（移位）：高位正数补零，负数补一（即前面是零补零，前面是一补一）；

>>>（移位） ：无符号右移，即高位始终补零;

2）&（与） : 相同保留，不同取零；

|（或） : 相同保留，不同取一；

^（异或）: 相同取零，不同取一；

~（取反）: 全部相反。

3）&&（短路与）：前面为假，表达式为假，后面不须计算；

4）||（短路或）：前面为真，表达式为真，后面不计算；

5）赋值运算：

a+=b 和 a=a+b 这俩个式子是否相同？如果不同说明为什么？

答：不相同，因为假如 int a=10，double b=0.1，则有

a+=b; // 相当于 a=(int)(a+b)，不需要进行强制类型转换，以等号左边的数据类型为准

a=a+b; // int=double，需要强制类型转换，以等号右边的数据类型为准

6）条件运算：A?a:b

1."?"前面的一定是一个独立的表达式；

2.若 A 为 True,则取":"前面的；否则取":"后面的。

7）脚下留心：写个例子说明&&和&,||和|的区别？

1. &&逻辑与运算符，&位与运算符；

2. &也可以进行逻辑运算，&不具有短路功能。

if((1==1)&(1==2))

3. ||逻辑或运算符，|位或运算符；

4. |也可以进行逻辑运算，|不具有短路功能。

if((1==1)|(1==2))

例：

```
public class Test {
    public static void main(String[] args) {
        int a = 1;
        int b = 1;
        if( a== 1 || b++ == 1){
            System.out.println("b="+b); //输出 b=1;若改为“|”则输出 b=2
        }
    }
}
```

3、类型（造型）转换：

1）分类：隐式转换：在 JAVA 中，低精度向高精度的转换（即窄类型向宽类型的转换）是隐形的，称为隐式转换，转换时不用任何说明就一定能转换成功。

显式转换：在 JAVA 中，高精度向低精度的转换（即宽类型向窄类型的转换）是显式的，称为显式转换，转换时必须对其进行强制性转换说明才能转换成功。

2）原则：在类型转换的时候保证数据的完整性

3）基本类型数据之间的转换：

1. byte ——→ short ——→ int ——→ long
2. char ——→ int ----→ float
3. float ——→ double
4. long ----→ float
5. long ----→ double
6. int ——→ double

注： 1) 实箭头表示无信息损失的转换，虚箭头表示转换可能引起损失；

2）int 和 float 同为 32 位，但 float 要有几位表示幂的位数，在精度位数上要比 int 要小，所以有可能会有损失。long 转为 double 同理；

3）char 和 short 同为 16 位，但 char 属无符号数，其范围为 0~2^16, short 的范围为-2^15~2^15-1 , 所以 char 和 short 不能相互转换，它们之间无论谁转谁都要进行强制性（显式）说明；

4）byte、short、char 属 child 型，在计算时会自动转为 int 型，然后转为更大范围类型(long、short、double)。

4）强制类型转换：

1.语法：圆括号中给出要转换的目标类型，随后是待转换的变量名。例：doublc x=9.997;int nx = (int)x;

2.如果试图强制转换的类型超出了目标类型的范围，结果是一个被截取的不同的数值;

3.不能在布尔值和任何数字类型间强制类型转换，如确要转换，可使用条件表达式，例：b?1:0。

5）转换的二种类型：

1.赋值

```
double d = 1.0f;
int i = (int)d;
```

2.方法调用

```
double converValue(int value){
        return value;
  }
```

6) 数值计算转换：-9.232e20+1; //12.5E300 为科学计数法，表示 12.5x10^300

4、流控制

1）判断结构: if(){...}

if(){...}else{...}

if(){...}else if(){...}else{...}

2）switch 块：

```
switch(){
        case variable:{...}
        case variable:{...}
        ……
        default:
                System.out.println("default");
}
```

注解：1.switch()内数据类型为 child 类型 byte short char 自动转换为 int,且只能是这四种类型；

2.每一个 case 块中都有 break 语句；

3.当与每一个 case 块都不能匹配上时执行 default 块，该块没有位置限制。

3）循环语句：

1.for(int i=0;i<n;i++){...}

范例：loop

```
for(int i=0;i<n;i++){
                for(int j=0;j<m;j++){
                        if(3==j){
                        break loop;     //loop 为标签：只能用在循环语句中，循环嵌套中
用于跳到外层循环。
                        }
                }
        }
```

脚下留心：1.int x,a=6,b=7;

x=a++ + b++; //a=7,b=8,x=13

2.int x=6;x=~x;//6 的二进制 00110，取反得 11001，再转成补码(取反加一)：10111=-7

2.while(){...}

3.do{...} while(); ----------> 加分号

注意：1）do while 语句后面必须加“;”

2）该语句是先执行后判断，即该循环至少执行一次。

4) break：跳出本层循环；

continue：结束本次循环，重新开始下一轮循环。

==========

范例程序：

1.位运算：

```
package com.briup.ch03;
```

```
public class BitWiseTest{
    public static void main(String[] args){
        int i = 5, j = 3;
        System.out.println((i&j));
        System.out.println((i|j));
        System.out.println((i^j));
        System.out.println((~i));
    }
}
```

2.比较循环：

1）briup@briup:~/corejava> vi ch03/IfTest.java

```
package com.briup.ch03;

public class IfTest{
    public static void main(String[] args){
        int i = 5, j = 4;
        if(i>j){
           System.out.println(i + ">" + j);
        }else if (i<j){
           System.out.println(i + "<" + j);
        }else{
           System.out.println(i + "=" + j);
        }
    }
}
```

```
briup@briup:~/corejava> javac -d . ch03/IfTest.java
briup@briup:~/corejava> java com.briup.ch03.IfTest
5>4
```

2）briup@briup:~/corejava> vi ch03/Comparison.java

```
package com.briup.ch03;

public class Comparison{
    int i ;
    int j;
    public void print(){
        if(i > j){
```

```
            System.out.println(i + " > " + j);
        }else if(i < j){
            System.out.println(i + " < " + j);
        }else{
            System.out.println(i + " = " + j);
        }
    }
}
briup@briup:~/corejava> javac -d . ch03/Comparison.java
briup@briup:~/corejava> vi ch03/IfTest.java
package com.briup.ch03;

public class IfTest{
    public static void main(String[] args){
        Comparison c = new Comparison();
        c.i = 5;
        c.j = 4;
        c.print();

        c.j = 80;
        c.print();

        c.i = 100;
        c.print();
    }
}
briup@briup:~/corejava> javac -d . ch03/IfTest.java
briup@briup:~/corejava> java com.briup.ch03.IfTest
5 > 4
5 < 80
100 > 80
```

3.求和循环：

```
briup@briup:~/corejava> vi ch03/Sum.java
package com.briup.ch03;

public class Sum{
    int start;
    int end;
    public void sumByFor(){
        int sum = 0;
```

```
        for(int i = start;i <= end;i=i + 1){
            sum += i;
        }
        System.out.println("start:\t" + start + "\tend\t" + end + "\tsum\t" + sum);
         //其中"\t"为空格符，另"\n"为换行符
    }
    public void sumByWhile(){
        int sum = 0;
        int i = start;
        while(i <= end){
            sum +=i;
            i += 1;
        }
        System.out.println("start:\t" + start + "\tend\t" + end + "\tsum\t" + sum);
    }
    public void sumByDoWhile(){
        int sum = 0;
        int i = start;
        do{
            sum += i;
            i += 1;
        }
        while(i <= end);
        System.out.println("start:\t" + start + "\tend\t" + end + "\tsum\t" + sum);
    }
}
briup@briup:~/corejava> javac -d . ch03/Sum.java
briup@briup:~/corejava> vi ch03/SumTest.java
package com.briup.ch03;

public class SumTest{
    public static void main(String[] args){
        Sum s = new Sum();
        s.start=1;
        s.end=100;
        s.sumByFor();
```

```
        s.start=51;
        s.sumByFor();
        System.out.println("==========================================");
        s.sumByWhile();
        System.out.println("==========================================");
        s.sumByDoWhile();
    }
}
briup@briup:~/corejava> javac -d . ch03/SumTest.java
briup@briup:~/corejava> java com.briup.ch03.SumTest
start:  1      end    100    sum    5050
start:  51     end    100    sum    3775
==========================================
start:  51     end    100    sum    3775
==========================================
start:  51     end    100    sum    3775
```

4.二进制循环：

```
briup@briup:~/corejava> vi ch03/Bin.java
package com.briup.ch03;

public class Bin{
    int data;
    public void outBin(){
        System.out.print("Data:\t" + data + "\t");
        for(int i = 31; i >= 0; i -= 1){
            int newData = (data >> i);
            newData &= 1;
            System.out.print(newData);
    }
    System.out.println();
    }
}
briup@briup:~/corejava> javac -d . ch03/Bin.java
briup@briup:~/corejava> vi ch03/BinTest.java
package com.briup.ch03;

public class BinTest{
    public static void main(String[] args){
        Bin b = new Bin();
```

```
        b.data = 5;
        b.outBin();
        b.data = -5;
        b.outBin();
        b.data = (-42342 >>> 1);
        b.outBin();
    }
}
briup@briup:~/corejava> javac -d . ch03/BinTest.java
briup@briup:~/corejava> java com.briup.ch03.BinTest
Data:   5       00000000000000000000000000000101
Data:   -5      11111111111111111111111111111011
Data:   2147462477      01111111111111111010110101001101
```

==========

练习：

1.编写一个类 MaxMin，包括三个成员变量（i,j,k），提供操作 print()，能够打印三个数字中的最大值和最小值。

2.编写一个测试程序 MaxMinTest，测试上述类的运行是否正确。

3.if 语句改为 switch 块。

4.用 while 和 do while 循环改写 for 循环。

==========

第四章 数组

1、声明数组

1) 一组相同类型(可以是类)数据的有序的集合，声明数组须指明其类型;

2) 一个数组是一个对象：意味着数组一类存放于堆中，且必须得 new（创建）出来之后才能使用，所有的对象都具有相同的属性 length;

3) 注意声明一个数组并没有创建一个对象;

4) 数组能以下列形式声明：

int[] i 或 int i[]

Car[] c 或 Car c[]

注意：C++中只能使用 Car c[]，而在 JAVA 中推荐使用 Car[] c.

2、创建数组：new

1) 创建基本数据类型数组： int[] i = new int[2];

2) 创建引用数据类型数组： Car[] c = new Car[100];

3) 数组创建后有初始值：

数字类型为0 布尔类型为 false 引用类型为 null

4) 边界问题：索引号是以 0 开始的。

3、初始化数组

1) 初始化、创建、和声明分开：

```
int[] i;
i = new int[2];  //分配（开辟）了两个空间，存放的是对象的引用
i[0] = 0;
i[1] = 1;
```

2) 初始化、创建、和声明在同一时间：

```
int[] i = {0,1};
Car[] c = {new Car(),new Car()};
```

4、多维数组

1) 有效：不对称数组----->在 java 中允许，但在 c++中是不允许的；

```
int[][] i1 = new int[2][3];
int[][] i2 = new int[2][];
i2[0] = new int[2],i2[1] = new int[3];
```

*C++中 int[][] =new int[][3];有效

2) 无效

```
int[][] i1 = new int[][3];
```

3) 高维数组创建必须从低维度向高维度依次序创建。

5、数组长度 ---------- 数组的属性 length

```
int[] i = new int[5];
int len = i.length;//lenth = 5;
Student[][] st = new Student[4][6];
len = st.length;//lenth = 4,行数;
len = st[0].length;//lenth = 6，第一行列数;
```

思考：请问以下哪段代码哪个可正确执行？(a,c)

a. char[] i = {'a','b'}; i = new char[]{'b','c'};

b. char[] i = {'a','b'}; i = {'b','c'};

c. char[] i = new char[2]; i = new char[]{'b','c'};

d. char[] i = new char[2]; i = {'b','c'};

6、数组拷贝：拷贝一个数组到另一个数组（覆盖）。

```
System.arrayCopy(Object src, int srcPos, Object dest, int destPos, int length);
```

System.arrayCopy(被拷贝数组的引用，被拷贝数组的起始位置，目标数组的引用，目标数组的起始位置，拷贝元素的个数);

==========

范例程序：

```
briup@briup:~/corejava> vi ch04/Address.java
package com.briup.ch04;

public class Address{
    String city;
    String street;
    int    code;
    public void print(){
        System.out.println("City:\t"+city);
        System.out.println("Street:\t"+street);
        System.out.println("Code:\t"+code);
    }
}
briup@briup:~/corejava> javac -d . ch04/Address.java
briup@briup:~/corejava> vi ch04/Person.java
package com.briup.ch04;

public class Person{
    String name;
    boolean gender;
    int age;
    Address[] address;

    public void print(){
       System.out.println("Name:\t"+name);
       System.out.println("Gender:\t"+gender);
       System.out.println("Age:\t"+age);
       for(int i=0;i<address.length;i++){
           address[i].print();
       }
    }
}
briup@briup:~/corejava> javac -d . ch04/Person.java
briup@briup:~/corejava> vi ch04/PersonTest.java
package com.briup.ch04;
```

```
public class PersonTest{
   public static void main(String[] args){
      Person p = new Person();
      p.name="kevin ding";
      p.gender=true;
      p.age=34;

      Address[] adrs = new Address[2];
      adrs[0]=new Address();
      adrs[0].city="shanghai";
      adrs[0].street="guotai road";
      adrs[0].code=88;

      adrs[1]=new Address();
      adrs[1].city="beijing";
      adrs[1].street="changan";
      adrs[1].code=1;

      p.address=adrs;
      p.print();
   }
}
briup@briup:~/corejava> javac -d . ch04/PersonTest.java
briup@briup:~/corejava> java com.briup.ch04.PersonTest
```

练习：1.思考：p,new person();name,gender,age,address[]{adrs[0],adrs[1]}

2.编写一个 Car 类表示车的信息（车高 height,车门数 doors，若干个车轮 wheels），每个车轮（wheel）包括两个信息（颜色 color,半径 radius）。

3.编写测试程序 CarTest,构建 Car 对象，填入车的完整信息并输出其完整信息。

==========

第五章 对象和类（OO 思想）

1、基本概念：

1）封装：a.隐藏信息 b.使方法的实现细节不可见 c.统一对所有用户的接口 d.提高效率

2）继承：通过已有的类去扩展新的类。

3）多态：同一类域的不同对象在调用相同方法的时候表现不同的现象(针对对象的行为而

言)。

2、方法的定义格式： <modifiers><return_type><name>([argument_list>])[throws <exception>]{<block>}

1)<modifiers>：访问修饰符（可以为若干个），没有先后顺序。例如 public static，static public，private 等。
2)<return_type>：方法的返回值类型（表示函数的出口），一个方法没有返回值时就必须定义为 void，构造方法没有返回类型。
3)<name>：方法名，方法名表示方法在代码区当中的地址。
4)([<argument_list>])：，参数列表（表示函数的入口），可选。
5)[throw<exception>]：声明抛出的异常类型，可选。
6){<block>}:方法体（即方法中的执行的代码）。

3、参数的传递方式：按值传递；按引用传递
1) 8 种基本数据类型的参数传递：按值传递是对应参数的值拷贝的过程，因此不会改变传入参数的值（是将其对象复制一份）。
2) 引用类型的参数传递：按引用传递，有可能对所传入的“参数所指向的值”产生影响（是其对象的引用复制一份，并没有复制对象）。
3）Java 语言总是使用传值调用，这意味着方法得到的只是所有参数值的拷贝。因此对于基本数据类型的参数传递方法不能修改传递给它的任何参数变量的内容；而对于引用类型的参数传递的是传递该对象的引用，方法中并不能改变对象变量，但能通过修改该变量的引用修改其所指向的对象。

4、this 关键字：当前对象的引用。
1）当局部变量和实例变量出现命名冲突时，Java 中表现为就近原则，因而 java 处理这种现象是使用 this.variableName 表示实例变量；
2）就两种用法：
(1) this.variableName（实例变量） ---> 用于区分同名的实例变量和局部变量；
(2) this（参数列表） ---> 用于调用本类的构造器。
3）关键字 this：是个隐式参数，代表被构造的对象；

```
public class Person{
        private String name;
                public void setName(String name){
        this.name=name;     // this.name 为实例变量
        }
    }
```

4) 如果构造器的第一个语句具有形式 this(...)，那么这个构造器将调用同一类中的其他构

造器。

5) 在构造器中this()必须放在方法的第一行。

5、封装

1）编码阶段的封装:

1.数据隐藏（信息隐藏）：private修饰符

a.类的编写者可以控制类的信息访问者（或类的使用者）的行为；

b.当一个类的内部信息的表达方式发生变化时，它不会影响类的使用者。

2.行为的不可见：

a.方法实现细节的不可见；

b.统一的接口；

c.增强可维护性。

2）编译阶段的封装：方法的重载

条件：1.方法名必须相同（在同一类中有若干个方法相同）；

2.参数列表必须不同：表现在(1)参数个数不同 (2)参数类型不同 (3)参数顺序不同；

3.返回类型无关紧要(返回值类型是否相同对重载没有任何的影响)；

4.Overloading方法对于不匹配的参数是从低向高转换的:Byte—short—int—long—float—double。

注意：1.在程序的编译过程中根据变量类型来找相应的方法，Overloading被认为是“编译时的多态”。

2.Overloading只是为方便程序员编程的解决方法，编译之后的方法名实际加上了各参数类型成为唯一的名字。

6、对象创建和初始化的过程（JVM的工作）：

1）在堆区开辟对象本身所在的存储空间，返回对象在堆区所在的地址给对象的引用(在栈区开辟对象的引用所在的存储空间)；

2）对对象的属性进行默认初始化；

3）显式初始化；

1.显式初始化：声明一个变量时同时对其赋值；

2.显式初始化可以帮助我们改变默认值，并可以给人为的实际变量赋予具有实际意义的值。

4）调用构造器对对象进行初始化（对象的构建过程中，目的是简化代码）。

构造器：1.就是一个特殊的函数（人为的），生成对象的过程中调用的方法，但构造器并不能创建对象，也叫构造方法；

2.特点：它的名字和类名完全相同；没有返回类型；

3.作用：能够让类的使用者提供一个初始化的值；就做一些初始化工作，初始化

要求尽可能分离在其方法中，然后在构造器中调用。

4.构造器调用：this()----->表示一个构造器调用本类中另外一个构造器。

5.系统（编译器）在程序员没有创建构造器的情况下将自动生成默认的参数列表为空的，方法体也为空的构造器；如果程序员提供了构造器，系统将不再提供默认的构造器。

6.构造器的重载：可以构造多个构造器，但多个构造器的参数列表一定不同，参数顺序不同即属于不同的构造器。

7.构造方法也可以是其他的限制符：private、protected、default 等(注意：private 一般用在 singleton 模式中)。

8.在一个对象的生成周期中构造器只用一次，一旦这个对象生成，那么这个构造器失效。

9.接口不能创建实例，因为没有构造器。

7、继承（主要目的：代码复用）

1）条件：必须满足“is a”关系才可以继承；

2）语法：pulic class subclass extends superclass{...}；

3）单根继承：不允许多重继承；一个子类只能有一个父类，但一个父类可以有若干个子类；JAVA 中所有类都有一个祖先类 Object（没有父类）；

4）父类中的构造器不能继承，但可以被调用；方法和成员变量都可以被继承；

5）Super 关键字: 表示父类数据区的引用

1.也是个隐形参数，代表被构造对象的父类；

2.同样也必须在构造器的第一行；

3.就两种用法：

(1)调用父类构造器：作为子类中的任何一个构造器必须调用父类中的构造器，表现为

a.显式调用：“super（传递给父类的参数值）”----->调用父类构造器并对其属性进行初始化，且必须放在构造器的第一行；

b.隐式调用：如果在子类构造器中省略调用父类构造器的语法，则编译器将自动插入代码 super()默认调用父类的无参构造器，如果父类中不存在无参构造器的话，系统将会显示编译出错。

(2)调用父类中的变量或方法：“super.父类的方法名（）”----->调用父类中同名的方法（当子类中含有与调用父类方法同名的方法时必须加“super.”，否则不用）。

注意：当子类发现父类中的方法并不适合自己时，就要覆盖父类中的方法。

6）弊端: 1.减少了代码的灵活性；

2.破坏了耦合性原则——低耦合，高内聚。

8、多态：

1) 多态的概念：不同类型的对象虽然是相同的操作但执行的是不同的过程；

2) 注意：对象的类型永远所固定的；
3) 编译阶段看变量类型，运行阶段看对象类型；
4) 运行期问题：具体调用哪个方法在运行阶段才能确立。
5) 操作符 instanceof：
1.语法：(引用型变量 instanceof 类名）；
2.涵义：判断引用型变量所指的对象是否是在操作符后的类的自身或者是它的子类中。
3.返回结果是 boolean 型；
6) 转换：
1.上朔造型：将子类类型变量转换成父类类型变量是隐式的；
2.下朔造型：将父类类型转换成类类型是显式的(强制转换前需利用 instanceof 对其进行类型鉴定)；
显式转换成功的前提条件是被转换的变量所指向的对象类型一定要是转换以后类型的自身或者是它的子类。
7) 方法覆盖（Methods Overridden，也叫方法重写）：
1.范围：父子类之间；

2.条件：a.具有相同的方法名，相同的参数列表和相同的返回类型；
b.可见范围不能被缩小（private 为最小的可见范围，public 为最大的可见范围，默认可见范围是位于二者之间的）；
访问控制符：
权限高
public 全部可见
protected 本类可见，同包可见，子类可见
default 本类可见，同包可见
private 本类可见
权限低
c.异常不能被扩大（IOException 为比 Exception 小的异常）。
d.final 修饰的方法不能被重写。
e.静态方法不能被覆盖，但可以被子类中同名的方法所隐藏。

3.切记：a.所有的属性取决于编译时类型；
b.所有的静态方法取决于编译时类型；
c.所有的普通方法取决于运行时类型（多态）。
8) 多态的三种表现形式：
1. 父类 引用 = new 子类();
接口 引用 = new 实现类();

2. 方法声明：public void method(父类 参数){};
方法调用：method(new 子类());

3. public 父类 method(){
return new 子类();
}

9、面向对象与面向过程

1）对象的概念：

1.什么是对象：EVERYTHING IS OBJECT（万物皆对象）

2.所有的事物都有两个方面：

a．属性（有什么）：用来描述对象。

b．方法（能够做什么）：告诉外界对象有那些功能。

“后者以前者为基础”

3.一个对象的属性也可以是一个对象:这是一种对象的关联（associate）

```
public class Student{
        private String name;---------对象
        private int age;---------------基本类型
        private boolean gender;
        public void study(){}---------方法
}
```

2）为什么要使用面向对象：

1.面向对象符合人类看待事物的一般规律。

a.对象的方法的实现细节是屏蔽的，只有对象方法的实现者了解细节。

b.方法的定义非常重要。方法有参数，也可能有返回值。分析对象主要从方法开始。通过类来看待对象，类是对象的抽象。

c.注意区分：对象（本身）、对象的实现者、对象的调用者。

2.采用面向对象方法可以使系统各部分各司其职、各尽所能。

a.对象之间的耦合性一定要低（比如不同硬盘和不同主板之间的关系），这样才能使每个对象本身做成最好的。

b.对于对象的要求：高内聚、低耦合，这样容易拼装成为一个系统。实现高内聚就是要最大限度地提高复用性（复用性好是因为高内聚）。可复用性是 OOP(面向对象编程)的基础。

3）比较面向过程的思想和面向对象的思想：

1.面向过程的思想：由过程、步骤、函数组成，以过程为核心；

面向对象的思想：以对象为中心，先开发类，得到对象，通过对象之间相互通信实现功能。

2.面向过程是先有算法，后有数据结构。

面向对象是先有数据结构，然后再有算法。

4）在用面向对象思想开发的过程中，可以对对象进行复用，如无法进行复用则开发新的对象。

1.开发过程是利用简单对象的多个简单方法来实现复杂的功能。

2.从语法上来看，一个类是一个新的数据类型。

3.在面向对象编程中，除了简单数据类型，就是对象类型。

==========

练习：构建一个图形库，包含基本图形如下：点（point),三角形（Trigle),圆（circle),矩形（rectangle), 正方形(square), 其中 point 具有两个属性（x,y)和绘制点的方法(draw),三角形具有三个点构成并且可以 draw(),圆具有一个点（圆心）和半径（radius)并且可以 draw，矩形具有一个顶点和长，宽，并且可以绘制出来，正方形具有一个顶点和边长且可以绘制出来，请用 point, Trigle, circle, Rectangle, square 类，并完成他们内部 draw 实现的方法。编写测试程序，构建上述各种图形并绘制出来（打印）。

==========

第六章 高级语言特性

1、修饰符 static

1）静态类变量（修饰成员变量）：

1.Static int data 语句说明 data 为类变量，为一个类的共享变量，是所有对象共享的，它不属于任何对象，是属于整个类的（静态方法也是一样）。

2.Static 定义的是一块为整个类共有的一块存储区域。

3.其变量可以通过类名去访问：类名.变量名（与通过对象引用访问变量是等价的）。

2）静态方法（修饰方法）：

1.Public static void printData(){}：表明此类方法为类方法（静态方法），访问是在编译期完成，执行效率比较高。

2.静态方法不需要有对象，可以使用类名调用（不需要实例也可以调用静态方法）。

3.静态方法中不能访问类的非静态成员，包括成员变量和方法；只能访问本类中的静态变量和其它静态方法。因为此时是通过类调用的，没有对象的概念。方法中 this.data 和 super.data 是不可用的。

原因：从根本上说，静态变量不管类是否实例化都会存在，而实例变量只有类实例化了才存在。直接调用静态方法时并不确定实例变量是否存在。

4. 一般情况下，主方法是静态方法，所以 JVM 可以直接调用它，主方法为静态方法是因为它是整个软件系统的入口，而进入入口时系统中没有任何对象，只能使用类调用。

5.静态方法不能被覆盖，如果子类中有和父类重名的静态方法，虽然编译通过，但它并不能实现多态，所以不能称作覆盖。例如：

```
class Super{
    static public void show(){System.out.println("in Super");}
}
class Sub extends Super{
    static public void show(){System.out.println("in Sub");}
}
public class  Test {
    public static void main(String[] arg) {
        Super s = new Sub();
        s.show();
    }
}
```

执行结果是： in Super

3）静态代码块（修饰没有名字的代码块）：

1.只被执行一次；

2.初始化块在类被加载后首先被运行，不管类是否实例化，而且只执行这一次

3.作用：一般用来初始化一些复杂类型的静态变量。

4）静态内部类（注意：只能修饰成员内部类）：

```
class Out{
    public static class Inner{}
}
```

5）Static 通常用于 Singleton 模式开发：

Singleton 模式(单例模式)：是一种设计模式，高于语法，可以保证一个类在整个系统中仅有一个对象。

1.问题域：

系统中你需要获得某个类的唯一实例，所有客户端对它的访问都将通过一个公

共的访问点获得。

2.解决方案：创建一个类并使其

a.定义一个私有的构造器；

b.定义一个私有、静态的实例变量指向自己(类型是自己的类型)；

c.定义一个公有、静态的访问方法 getInstance()用于返回该类的唯一实例(注：懒汉式需同步)。

3.分类：

a.饿汉式：在类装载的时候就进行实例化；

b.懒汉式：在公有、静态的访问方法(同步：用 synchronized 修饰)中进行实例化，用的多一些。

实现 1：

```
public class ConnectionFactory{
    private static Connection conn;
    private Connection(){
      if(conn==null)
      conn = new Connction();
     }
    public Connection getInstance(){
       return conn;
    }
}
```

实现 2：

```
public class ConnectionFactory{
        private static Connection conn;
        static{
                conn = new Connection();
        }
        public static Connection getInstance(){
                return conn;
        }
}
```

2、修饰符 final

1）final 型变量（修饰变量时）：

1. 当利用 final 修饰一个变量（属性）的时候，此时该变量变为常量。

注意：a. JAVA 命名规范中常量全部字母要求大写：Final int AGE=10；

b. 常量的地址不可改变，但在地址中保存的值（即对象的属性）是可以改变的。

2. 在 JAVA 中利用 public static final 的组合方式对常量进行标识（固定格式）。

3. Final 变量是在整个类被创建时候被赋值，之后就不能改变了。

注意：a. 对于 final 变量，如果在声明和构造的时候均不进行赋值，将显示编译出错。

b. 对于利用构造器对 final 变量进行赋值的时候，此时在构造之前系统设置的默认值被覆盖。

4. 常量赋值（这里的常量指的是实例常量，即成员变量）：

① 在初始化的时候通过显式声明赋值：Final int x=3；

② 在构造的时候赋值。

2）final 型方法（修饰方法时）：

1. final 方法不能被覆盖（不能被改写），只能被继承。

2. 为了保证方法的一致性（即不被改变），可将方法用 final 定义。

注意：a. 如果在父类中有 final 定义的方法，那么在子类中继承同一个方法。

b. 如果一个方法前有修饰词 private 或 static，则系统会自动在前面加上 final。即 private 和 static 方法默认均为 final 方法。

c. final 并不涉及继承，继承取决于类的修饰符是否为 private、default、protected 还是 public。也就是说，是否继承取决于这个类对于子类的可见性。

3. Final 和 abstract 永远不会同时出现。

3）final 类（修饰类时）：

1.final 类不能被继承，即 final 类没有子类。

2.可以用 final 保证用户调用时动作的一致性，可以防止子类覆盖情况的发生。

3.final 修饰类时表明同时类中的方法都是 final 型，但其变量不是。

3、修饰符 Abstract

1）抽象方法（修饰方法时）：子类继承抽象类时必须实现其中的抽象方法

1.当 abstract 用于修饰方法时，此时该方法为抽象方法，此时方法不需要实现，实现留给子类覆盖，子类覆盖该方法之后方法才能够生效。

2.抽象方法没有方法体，要求子类必须提供这个方法的实现。

注意比较：private void print(){}；此语句表示方法的空实现。

Abstract void print()； 此语句表示方法的抽象，无实现

2）抽象类（修饰类时）：

1. 如果将一个类声明为 abstract，表明此类不能创建对象（不能实例化），只能声明变量。

2. Abstract 类的设计是将子类的共性最大限度的抽取出来，以提高程序的统一性。

3. 带有抽象方法的类一定是抽象类，但抽象类可以没有抽象方法。

4. Abstract类可以作为编译时类型，但不能作为运行时类型。

4、接口

1) 接口是抽象类的另外一种形式(没有实例变量的抽象类);
2) 在一个接口中所有方法都是抽象方法;
3) 接口中所有变量都必须被定义为 final static；
4) 接口可以继承多个接口（可多重继承）；
5) 接口只有两种语法成分：静态常量和公有的抽象方法。

注：接口中的方法自动被置为 public, 因此在接口中声明方法并不需要提供 public 关键字。但在实现接口时，必须把方法声明为 public。

```
public interface IA{
    public abstract void method();
    void method2();
}
public interface IB{
    void method3();
}
public interface IC extends IA,IB{
    void method4();
}
public class C implements IC{}      // implements 表示对接口 IC 中方法的实现，共有 4 个
```

5、Object类：JAVA中有一个特殊的类，它是JAVA体系中所有类的父类（直接父类或者间接父类），此类中的方法可以使所有的类均能继承。以下介绍的三种属于Object类的方法：

1）操作符“==”和方法 equals()：

1.==：比较的是引用，表示其引用值是否相等，即是否指向同一对象；

注：“==”在任何时候都是比较地址，这种比较永远不会被覆盖。

2.equals: 比较的是对象，其默认比较规则就是"==";

注意：基本类型没有 equals()，只有复杂类型才有。

3.附加知识：字符串类为JAVA中的特殊类(String类在Java中为final类型)，一个字符串的值不可重复。因此在JVM（虚拟机）中有一个字符串池，专门用来存储字符串。如果遇到 String a="hello"时（注意没有new，不是创建新串），系统在字符串池中寻找是否有"hello"，此时若字符串池中没有"hello"，那么系统将此字符串存到字符串池中，然后将"hello"在字符串池中的地址返回给a。如果系统再遇到 String b="hello"，此时系统在字符

串池中可以找到"hello"字符串，并将其地址返回给 b，则此时 a 与 b 相同。

4.举例：

a. String a = new String("hello");

String b = new String("hello");

System.out.println(a==b);

则 a==b 的返回为结果为 false。 //因为==判断的是引用是否相等，并不是判断对象。

b. String a = "hello";

String b = "hello";

System.out.println(a==b);

系统的返回值为 true。

c. 故如果要比较两个字符串是否相同(而不是比较它们的地址是否相同)，可以对 a 调用 equals:

System.out.println(a.equals(b));

系统的返回值为 true。注意 equals()用来比较两个对象中字符串的顺序，即 a.equals(b)比较的是 a 与 b 的值。

5.注意下面程序：

```
student a=new student(“LUCY”,20);
student b=new student(“LUCY”,20);
System.out.println(a==b);
System.out.println(a.equals(b));
```

此时返回的结果为 false。因为 Student 继承的是 Object 的 equals()方法，此时 toString()等于 “==” ,为了实现对象的比较需要覆盖 equals()方法（加上这个定义，则返回 ture 或 false）。

6.实现标准 equals()方法的流程：

```
public boolean equals(Object o){
    if (this==o) return true;  //此时两者相同
    if (o==null) return false;
    if (! o instanceof Student) return false;  //不同类
    Student s = (Student)o; //强制转换
    if (s.name.equals(this.name)&&s.age==this.age) return true;
    else return false;
}
```

2）finalize 方法：当一个对象被垃圾收集器回收的时候调用的方法。

3）toString():是利用字符串来表示对象的方法，简化输出。

1.当我们直接打印定义的对象的时候，隐含的是打印 toString()的返回值（返回一个字符串）。

2.可以通过子类作为一个 toString()来覆盖父类的 toString()以取得我们想得到的表现形式，即当我们想利用一个自定义的方式描述对象的时候，我们应该覆盖 toString()。

6、内部类（Inner class）：

1）概念：内部类是指在一个类的内部再定义一个类。它可以访问外围类的私有成员，其语法特别繁琐，在实际开发中避免使用。

注意：所有使用内部类的地方都可以不用内部类，但使用内部类可以使程序更加的简洁，便于划分层次结构和命名规范。

2）特点：1. 缩小类的访问范围，比包还小；

2. 用内部类定义在外部类中不可访问的属性，这样就在外部类中实现了比外部类的 private 还要小的访问权限。

注意：内部类是一个编译时的概念，一旦编译成功，就会成为完全不同的两个类。对于一个名为 outer 的外部类和一个在其内部定义的名为 inner 的内部类，编译完成后就会出现 outer.class 和 outer$inner.class 两个不同的类。

3）内部类可为静态，也可用 PROTECTED 和 PRIVATE 修饰(而外部类不可以，顶级类只能使用 PUBLIC 和 DEFAULT 修饰)。

注意：JAVA 文件中没有任何一个 public class 可以类名和文件名不同，但内部类可以。

4）内部类的分类：与成员变量同一级别（成员内部类和局部内部类）；

与方法中的局部变量同一级别（静态内部类和匿名内部类(经常用到，必须掌握))。

5)成员内部类:

1.概念：作为外部类的一个成员存在，与外部类的属性、方法并列。是最复杂，用的最少的内部类。

2.特点：必须依附于外围类的实例而存在，能访问外围类的一切成员，成员内部类不能有静态属性。

3.在内部类中访问实例变量：this.属性

在内部类访问外部类中与它命名冲突的成员：Outer.this.member

在外部类的外部访问内部类，使用 out.inner

4.构建成员内部类实例：

a.在外围类以外的地方：(new outer()).new Inner();

b.在外围类内部：this.new Inner();

6）静态内部类：

1.静态内部类定义在类中，任何方法外，用static定义。是最简单的内部类，与外围类的名字不能相同。

2.静态内部类只能访问外部类的静态成员。

3.不需要构建外围类实例就可直接构建静态内部类的实例：这是静态内部类和成员内部类的区别，静态内部类的对象可以直接生成，而不需要通过生成外部类对象来生成。这样实际上使静态内部类成为了一个顶级类。静态内部类不可用private来进行定义。

```
Outer.Inner in = new Outer.Inner();
```

4.注意： a.当类与接口（或者是接口与接口）发生方法命名冲突的时候，此时必须使用内部类来实现。

b.用接口不能完全地实现多继承，用接口配合内部类才能实现真正的多继承。

例子：对于两个类，拥有相同的方法，另有一个robot类：class Robot extends Person implement Machine{}。此时run()不可直接实现。

```
interface Machine{
	void run();
}
class Person{
	void run(){
		System.out.println("run");
	}
}
class Robot extends Person{
	private class MachineHeart implements Machine{
		public void run(){
			System.out.println("heart run");
		}
	}
	public void run(){
		System.out.println("Robot run");
	}
	Machine getMachine(){
		return new MachineHeart();
	}
}
class Test{
	public static void main(String[] args){
		Robot robot = new Robot();
		Machine m = robot.getMachine();
		m.run();
		robot.run();
	}
}
```

6）局部内部类：

1.概念：在方法中定义的内部类称为局部内部类。

2.与局部变量类似，在局部内部类前不能使用修饰符 public,protected,private 和 static，其范围为定义它的代码块。

3.局部内部类可以访问外部类的一切成员，还可以访问这个方法中由 final 修饰的局部变量（即常量）。

4.可见范围缩小到一个方法的内部：在类外不可直接访问局部内部类（保证局部内部类对外是不可见的）。

注意：在方法中才能调用其局部内部类。

7）匿名内部类：

1.概念：匿名内部类是一种特殊的局部内部类，它是通过匿名类实现接口。

注意：匿名内部类没有名字，没有关键词 class，没有 extends 和 implements,也没有构造器。

2.一个匿名内部类一定是在 new 的后面，隐含着继承一个父类或者实现一个接口（或实现一个类），没有类名，根据多态，我们使用其父类名。

3.利用父类或接口的名字代替 new 后面的名字来创建对象。

4.利用父类或接口声明变量：接口名 对象名 = new 接口名(){...};

注意：new 是一个完整的语句，注意一定要在花括号后加分号；

如果一个对象编译时的类型是接口，那么其运行的类型为实现这个接口的类。

5.因其为局部内部类，那么局部内部类的所有限制都对其生效。

注意：匿名内部类是唯一一种没有构造器的类。因匿名内部类无构造器，所以其使用范围非常的有限。

6.匿名内部类在编译的时候由系统自动起名 Out$1.class。

7、包装器类：Wrapper Class

功能：将基本类型数据包装成对象，例如

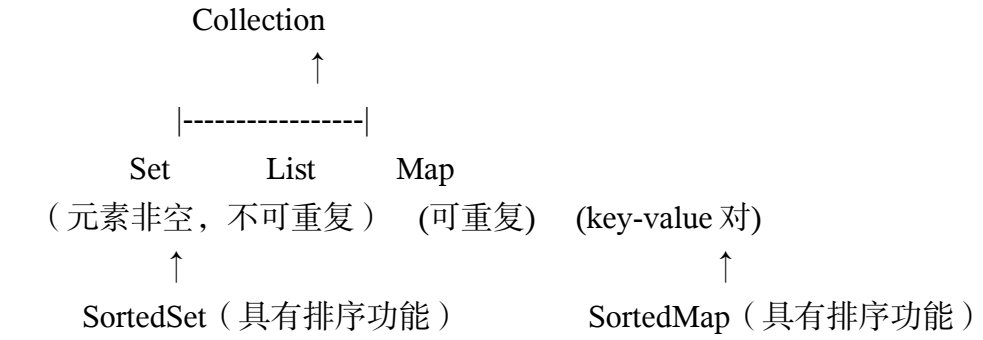
```
int i = 5; 或  String s = "5";
Integer integer = new Integer(i 或 s);  //装箱（box）：包装成对象
int j = integer.intValue();                //拆箱（unbox）：提取数据
对于 String 型数据来说以上两步相当于：int j = Integer.parseInt(s);
```

8、集合 Collection:

1）概念：集合是指一个可以容纳多个对象(不是引用)的对象，这个集合对象主要用来管理和维护一系列相似的对象。

2）集合三要素：若干个接口，接口的实现类和算法(一系列方法和实现类交合在一起)。

3）集合接口类层次：位于 package java.util.*;



注意：1.Set: 集合类中不允许有重复对象;

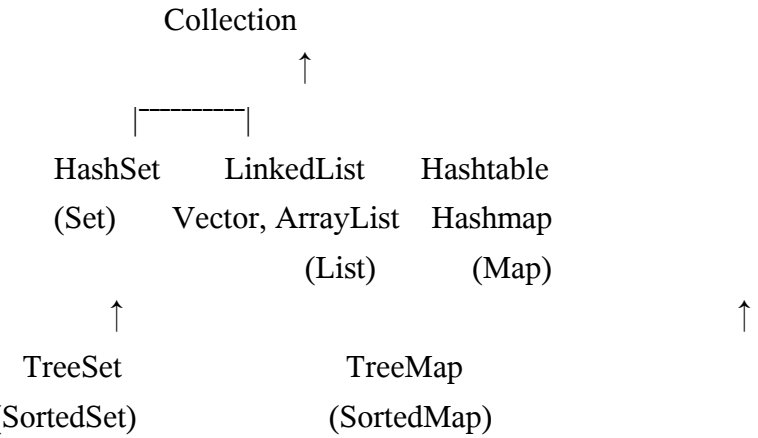
2.SortedSet: 和 Set 接口相同，但元素按升序排列;

3.List: 元素加载和移出时按照顺序，可以保存重复对象。

4.Map: 存储了唯一关键字辨识和对应的值(key-value 对)。

5.SortedMap: 和 Map 类相同，但对象按他们的关键字升序排列。

4）集合类层次 ：（注：JAVA1.5 对 JAVA1.4 的最大改进就是增加了对范型的支持）



5）Collection 接口的方法：

```
add(Object o)
addAll(Collection c)
contains(Object o)
containsAll(Collection c)
remove(Object o)
removeAll(Collection c)
clear()
equals(Object o)
isEmpty()
iterator()
size()
toArray()
toArray(Object[] o)
```

6）Iterator（迭代器）：读集合的一个工具

```
public interface Iterator{
```

```
boolean hasNext();
Object next();
void remove();
}
```

7）五个最常用的集合类之间的区别和联系：

1．ArrayList: 元素单个，效率高，多用于查询

底层是 Object 数组，所以 ArrayList 具有数组的查询速度快的优点以及增删速度慢的缺点。

2．LinkedList:元素单个，多用于插入和删除

a.经常在增删操作较多而查询操作很少的情况下使用：队列和堆栈。

队列：先进先出的数据结构；

栈：后进先出的数据结构。

b.在 LinkedList 的底层是一种双向循环链表。在此链表上每一个数据节点都由三部分组成：前指针（指向前面的节点的位置），数据，后指针（指向后面的节点的位置）。最后一个节点的后指针指向第一个节点的前指针，形成一个循环。

c.双向循环链表的查询效率低但是增删效率高。

d.注意：使用栈的时候一定不能提供方法让不是最后一个元素的元素获得出栈的机会。

ArrayList 和 LinkedList 在用法上没有区别，但是在功能上还是有区别的。

3．Vector: 元素单个，线程安全，多用于查询

a.与 ArrayList 相似，区别是 Vector 是重量级的组件，使用时消耗的资源比较多。

b.结论：在考虑并发的情况下用 Vector（保证线程的安全）；在不考虑并发的情况下用 ArrayList（不能保证线程的安全）。

4．HashMap: 元素成对，元素可为空

5．HashTable: 元素成对，线程安全，元素不可为空

6.Map 遍历的形式：

1）
```
Set ks = map.keySet();
Iterator it = ks.iterator();
while(it.hasNext()){
        Object key = it.next();
        Object value = map.get(key);
}
```

2）
```
Collection values = map.values();
```

```
Iterator it = values.iterator();
while(it.hasNext()){
        Object value = it.next();
}
```

3）
```
Set<Map.Entry> entrySet = map.entrySet();
for(Map.Entry entry:entrySet){
        Object value = entry.getValue();
}
```

8）知识点：

1.java.util.stack（stack 即为堆栈）的父类为 Vector。可是 stack 的父类是最不应该为 Vector 的。因为 Vector 的底层是数组，且 Vector 有 get 方法（意味着它可能访问到并不属于最后一个位置元素的其他元素，很不安全）。对于堆栈和队列只能用 push 类和 get 类。故 Stack 类以后不要轻易使用。

2.实现栈一定要用 LinkedList。在 JAVA1.5 中，collection 由 queue 来实现队列。

3.Set-HashSet 实现类：

a.遍历一个 Set 的方法只有一个：迭代器（interator）。

b.HashSet 中元素是无序的（这个无序指的是数据的添加顺序和后来的排列顺序不同），而且元素不可重复。

c.HashSet 底层用的也是数组。

d.在 Object 中除了有 finalize()，toString()，equals()，还有 hashCode()。当向数组中利用 add(Object o)添加对象的时候，系统先找对象的 hashCode：

int hc = o.hashCode(); //返回的 hashCode 为整数值。

Int I = hc%n; //n 为数组的长度，取得余数后，利用余数向数组中相应的位置添加数据，以 n 为 6 为例，如果 I=0 则放在数组 a[0]位置，如果 I=1,则放在数组 a[1]位置。如果 equals()返回的值为 true，则说明数据重复。如果 equals()返回的值为 false，则再找其他的位置进行比较。这样的机制就导致两个相同的对象有可能重复地添加到数组中，因为他们的 hashCode 不同。

如果我们能够使两个相同的对象具有相同 hashcode，才能在 equals()方法中返回真。因此在实际中，我们定义 Student 对象时要覆盖它的 hashcode。因为 String 类是自动覆盖的，所以当比较 String 类的对象的时候，就不会出现有两个相同的 string 对象的情况。现在在大部分的 JDK 中，都已经要求覆盖了 hashCode。

结论：如将自定义类用 hashSet 来添加对象，一定要覆盖 hashcode()和 equals()这两个方法，覆盖的原则是保证当两个对象相同时 hashcode 返回相同的整数，而且 equals()返回的值为 True。如果偷懒，没有设定 equals()，就会造成返回 hashCode 虽然结果相同，但在程序执行的过程中会多次地调用 equals()，从而影响程序执行的效率。

我们要保证相同对象的返回的 hashCode 一定相同，也要保证不相同的

对象的 hashCode 尽可能不同（因为数组的边界性，hashCode 还是可能相同的）。

例子：

```
public int hashCode(){
    return name.hashcode()+age;
}
```

这个例子保证了相同姓名和年龄的记录返回的 hashCode 是相同的。

e.使用 hashSet 的优点：

hashSet 的底层是数组，其查询效率非常高。而且在增加和删除的时候由于运用的 hashCode 的比较确定添加元素的位置，所以不存在元素的偏移，所以效率也非常高。因为 hashSet 查询和删除和增加元素的效率都非常高。但是 hashSet 增删的高效率是通过花费大量的空间换来的：因为空间越大，取余数相同的情况就越小。HashSet 这种算法会建立许多无用的空间。

f.使用 hashSet 类时要注意，如果发生冲突，就会出现遍历整个数组的情况，这样就使得效率非常的低。

4.比较:

Collections 类（工具类 ---> 全是 static 方法）：

```
Public static int binarySearch(List list,Object key)
Public static void Sort(List list,Comparator com)
```

Comparator 接口：

```
Int compare(Object a,Object b)
Boolean equals(Object o)
```

注意：集合的最大缺点是无法进行类型判定（这个缺点在 JAVA1.5 中已经解决），这样就可能出现因为类型不同而出现类型错误。解决的方法是添加类型的判断。

例子：

```
import java.util.*;
public class  Test {
    public static void main(String[] arg) {
    ArrayList al = new ArrayList();
    Person p1 = new Person("kevin");
    Person p2 = new Person("robin");
    Person p3 = new Person("terry");
    al.add(p1);
    al.add(p2);
    al.add(p3);
    Collections.sort(al,p1);
    for(Iterator it = al.iterator();it.hasNext();){
    Person p = (Person)it.next();
    System.out.println(p.name);
```

```
        }
        }
    }
    class Person implements java.util.Comparator{
    public String name;
    public Person(String name){
        this.name = name;
    }
    public int compare(Object a,Object b){
        if(a instanceof Person && b instanceof Person){  //类型的判断
        Person pa = (Person)a;
        Person pb = (Person)b;
        return pa.name.compareTo(pb.name);
        }
        return 0;
    }
        public boolean equals(Object a){return true;}
    }
```

9）总结：谈谈 Java 中的集合框架?

1.ArrayList,Vector,LinkedList 的存储性能和特性?

2.Collection(集合的最上层接口)与 Collections(工具类)的区别?

3.怎样使 ArrayList 成为线程安全的?

根据集合存放数据的特点，分为两类：

1）只存单值： List , Set

ArrayList：以数组的方式管理数据，线程不安全的，适合于做大量的查询操作；

思考：怎么变成线程安全的(java.util.Collections)?

-----> synchronizedList(List<T> list)方法；

LinkedList：线程安全的,以双向链表的方式管理数据，适合于做大量的增加、删除操作；

Vector：不适合查询，线程安全的，适合处理多线程集合；

2）存 Key-Value 键值对： Map

HashMap（线程不安全的）

Hashtable（线程安全的）

TreeMap(根据 HashCode 进行排序）

9、反射（JAVA 中最抽象并且是最难理解的概念，也叫镜像）

1）作用：

1.动态构建类的实例：确定一个对象的类;

2.获得类的内部信息：获得一个类的修改符、变量、方法、构器函数、和父类的相关信息;

找出哪些常量和方法是从一个接口声明的;

创建一个在运行时才知道名称的类;

调用对象的方法;

3.改变类的一些特性。

2）获取镜像仅有的三种方式：

1.知道类的对象：

```
object o = xxxx;
Class clazz = o.getClass();
```

2.知道类名获取镜像：

```
java.uil.ArrayList
Class clazz = ArrayList.class;
```

3.知道存放类名的字符串：

```
String s = "xxxxx";  //不知类名
Class clazz = Class.forName(s);
```

3）通过反射修改类的特性：

1. 修改一个对象的实例变量值

```
Field.set(instance,value);
```

2. 调用一个对象上的方法

```
Method.invoke（instance,new Object[]{value1,value2});
```

3. 调用一个类中的构造器

```
Constructor.newInstance{new Object[]{value1,value2});
```

4）程序范例：

```
briup@briup:~/corejava> vi collection/DynamicObject.java
package com.briup.collection;

import java.util.*;

public class DynamicObject{
    public static void main(String[] args) throws Exception {
        if(args.length<1){
            System.out.println("Usage: java "+
            DynamicObject.class.getName()+" XXXX");
        System.exit(0);
        }
        String className = args[0];
```

```
                    Class clazz = Class.forName(className);
                    Collection conn = (Collection)clazz.newInstance();
                    conn.add(new Integer(32));
                    conn.add(new Integer(6454));
                    conn.add(new Integer(-4242));
                    Iterator iter = conn.iterator();
                    while(iter.hasNext()){
                    System.out.println(iter.next());
                    }
        }
        }
briup@briup:~/corejava> javac -d . collection/DynamicObject.java
briup@briup:~/corejava> java com.briup.collection.DynamicObject
Usage: java com.briup.collection.DynamicObject XXXX
briup@briup:~/corejava> java com.briup.collection.DynamicObject java.util.ArrayList
32
6454
-4242
briup@briup:~/corejava> java com.briup.collection.DynamicObject java.util.TreeSet
-4242
32
6454
```

==========

思考：单例模式（保证一个类只有一个对象）

```
public class Singleton{                    //充分利用修饰符 static
        private static Singleton instance;
        private Singleton(){}              // 控制构造器不能被调用,私有
        public static Singleton getInstance(){
            if(instance == null)
                instance = new Singleton();
            return instance;
        }
}
```

==========

第七章 异常

1、异常的基本概念 ：

1）异常的发生改变程序流程；

2）异常的发生是有条件的：当一个异常事件发生时，一个异常被抛出；

3）响应处理异常的代码被称为 exception handler;
4）exception handler：捕获异常；
5）异常处理能让你集中精力在一个地方解决问题，然后将处理错误的代码分开放在另一个地方。
6）异常的发生：1.没有预料到的程序错误；2.人为制造的。

2、捕获异常
1) 设置一个 try/catch 的代码块;
2) 如果 try 块内的任何代码抛出了由 catch 子句指定的异常，则
a. 程序跳过 try 块中的其他代码;
b. 程序执行 catch 从句中的处理器代码。
3) 如 try 块内没有抛出异常，直接跳过 catch 从句内容。
4) 如 try 块内抛出的异常没有在 catch 从句中指定, 则该方法会立即退出。

3、处理异常
1）如何控制 try 的范围：根据操作的连动性和相关性，如果前面的程序代码块抛出的错误影响了后面程序代码的运行，那么这个我们就说这两个程序代码存在关联，应该放在同一个 try 中。
2）对已经查出来的异常，有 throws(消极)和 try catch（积极）两种处理方法。
对于 try catch 放在能够很好地处理异常的位置（即放在具备对异常进行处理的能力的位置），如果没有处理能力就继续上抛。
3）当我们自己定义一个异常类的时候必须使其继承 Excepiton 或者 RuntimeException。
4) 对子类方法抛出的异常不能超出父类方法 throws 指令的范围。如父类方法不抛出任何异常，在子类方法中必须捕捉每一个“已检查异常”。

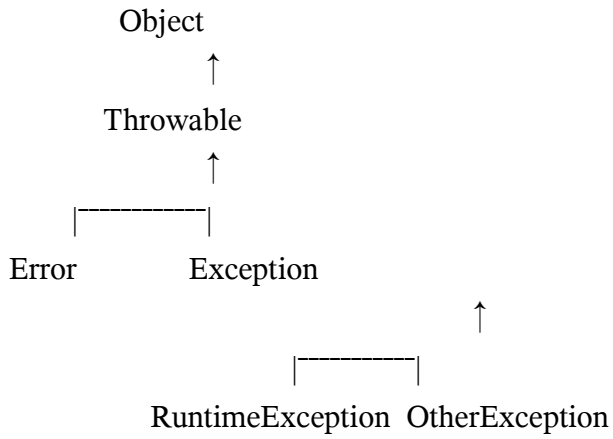
5、捕捉多个异常
1) 一个 catch 块是一个异常，其数目不受任何限制；
2) 如前面 catch 从句捕获异常，将直接跳过后面 catch 从句内容；
3) 建议按异常类型的子类 --> 超类的顺序排列 catch 从句的先后顺序。

6、finally 块：
无论是否捕获异常，都会执行 finally 从句中的代码(程序结束前必须执行的代码)。
例子：finally{conn.close();}

7、异常调用栈：异常的发生点
1）哪个方法调用的代码发生异常，返回到调用方法的地方;
2）main 方法调用的代码发生异常，返回到虚拟机。

3）getMessage()：取到关于异常的简短说明信息；
printStackTrace()：打印异常的发生点以及所有的调用点。

8、异常层次（分类）：
1) 起源于 Error 的类，代表不常用的环境(通常是硬件层面)；
2) 应用程序不能够从 Error 中恢复正常；
3) 所有的 Java 异常都起源于 Exception，由应用程序负责；
4) RuntimeExcepiton:也称为未检查异常，编译器不要求；
5) 未检查异常无须捕获；
6) 其它异常:也称为检查异常，必须处理（编译器）。



9、一些未检查的异常：
1) java.lang.ArithmeticException（算术异常） 如：除 0;
2) java.lang.NullPointerException（空指针异常） 如：没初始化一个 References 便使用;
3) java.lang.ArrayIndexoutofBoundsException（数组越界） 如：调用一个有十个元素的 Array 的第十一个元素的内容;
4) java.lang.NumberFORMatException（数据格式转换异常） 如：Integer.parseInt("a");
5) java.lang.SecurityException（安全异常）：一般碰不到
6) java.lang.NegativeArraySizeExcepriotn（数组长度负数异常）

10、写你自己的异常
1) 要做的仅仅是从 Exception 继承或者是从 Exception 的一个子类衍生出自己需要的类即可；
2) 习惯为每一个异常类提供一个默认的构造器以及一个包含详细信息的构造器。

11、抛出你自己的异常

1) 在方法的定义中增加一个 throws 修饰符，以通知调用者可能会抛出一个异常;
2) 在方法中构造一个该类的实例，然后抛出该实例。

范例程序：异常的处理

```
briup@briup:~/corejava> vi ch07/DivTest1.java
package com.briup.ch07;

public class DivTest1{
        public static void main(String[] args){
                int i=5,j=0,result=0;
                result=div(i,j);
                System.out.println("result: "+result);
        }
        public static int div(int i,int j){
                int res = 0;
                res = i/j;
                System.out.println("res: "+res);
                return res;
        }
}
briup@briup:~/corejava> javac -d . ch07/DivTest1.java
briup@briup:~/corejava> java com.briup.ch07.DivTest1
Exception in thread "main" java.lang.ArithmeticException: / by zero
            at com.briup.ch07.DivTest1.div(DivTest1.java:11)
            at com.briup.ch07.DivTest1.main(DivTest1.java:6)
briup@briup:~/corejava> cp ch07/DivTest1.java ch07/DivTest2.java
briup@briup:~/corejava> vi ch07/DivTest2.java
package com.briup.ch07;

public class DivTest2{
        public static void main(String[] args){
                int i=5,j=0,result=0;
                try{                                    // 异常的处理
                result=div(i,j);
                System.out.println("result: "+result);
                }catch(ArithmeticException e){
                System.out.println("catch exception");
                System.out.println(e);
                System.out.println(e.getMessage());             // 打印异常的简短说明信息
                e.printStackTrace();                    // 打印异常的所有发生点和所有调用点
                }finally{
                System.out.println("finish");
                }
          System.out.println("main end");
        }
        public static int div(int i,int j){
```

```
            int res = 0;
            res = i/j;
            System.out.println("res: "+res);
            return res;
        }
}
briup@briup:~/corejava> javac -d . ch07/DivTest2.java
briup@briup:~/corejava> java com.briup.ch07.DivTest2
catch exception
java.lang.ArithmeticException: / by zero
/ by zero
java.lang.ArithmeticException: / by zero
        at com.briup.ch07.DivTest2.div(DivTest2.java:21)
        at com.briup.ch07.DivTest2.main(DivTest2.java:7)
finish
main end
```

==========

思考：

```
public int method(){
    int i = 5, j = ?;           // ?取 0 或其它值
    try{
      i / j;
      return 1;
    }catch(Exception e){
      return 2;
    }finally{
      return 3;                 // 不管有无异常返回结果都是 3
    }
      // return 4;              // 加上之后永远无法执行
}
```

==========

第八章 图形用户接口：GUI

1、图形包：

javax.swing.*;图形界面升级包

java.awt.*:基本图形界面包

java.awt.event.*:图形界面事件处理包（用于实现使用者交互）

2、AWT（抽象窗口工具箱）组成要素：

1.Component:组件；

2.Container（容器）：特殊的组件，它里面可以包含其它的组件；

3.LayoutManager（布局管理器）：控制组件在容器中的排列风格；

注意：一个容器中只能有一种排列风格。

3、步骤：1.确立容器；

2.设置布局管理器；

3.加入组件到容器中；

4.设置事件处理器。

第十一章 事件处理模型

1、事件：对窗口内进行的所有动作称之为事件。

2、模型三要素：

1）事件源：被观察者

2）事件对象：交换的信息

3）事件监听器：观察者

3、Ob(观察者模式)：交互关系

1）被观察者：事件源

1.必须记录对事件源感兴趣的监听器；

2.提供方法用于加入和删除监听器；

3.在事件源内部要包含一段逻辑创造事件对象，并将其传递给事件监听器。

注：所有组件可能成为事件源

2）中间关系：事件对象

在事件对象里必须要记录是谁产生了事件对象，必须包含一个引用型变量来指明事件源是哪一个对象；

3）观察者：事件监听器

在事件侦听器里要提供一个方法能够接收事件对象(信息)。

第十三章 线程(Threads)

1、线程原理

1）概念：线程是在一个时间段看到的CPU的多个执行流程；

线程在语言层面上以对象的形式而存在

2）与进程的区别：进程是数据独占的，而线程是数据共享的（所以需要处理数据并发）；

3）并发原理：宏观并行，微观串行。

OS(操作系统)将一段时间分为多个时间片，每个时间片CPU只能运行一个任务。

4）线程组成要素：处理机（独立的）；代码（即可共享也可不共享）；数据（即可共享也可不共享）。

2、构建线程的方式：

1）继承java.lang.Thread：符合面向对象的思想

```
class MyThread extends Thread{
        public void run(){
                //需要进行执行的代码，如循环。
        }
}
```

启动线程：

```
public class TestThread{
        public static void main(){
                Thread t1 = new Mythread();
                T1.start();
        }
}
```

2）实现java.lang.Runnable接口：解决了单继承的限制（这种实现可再继承其他类）

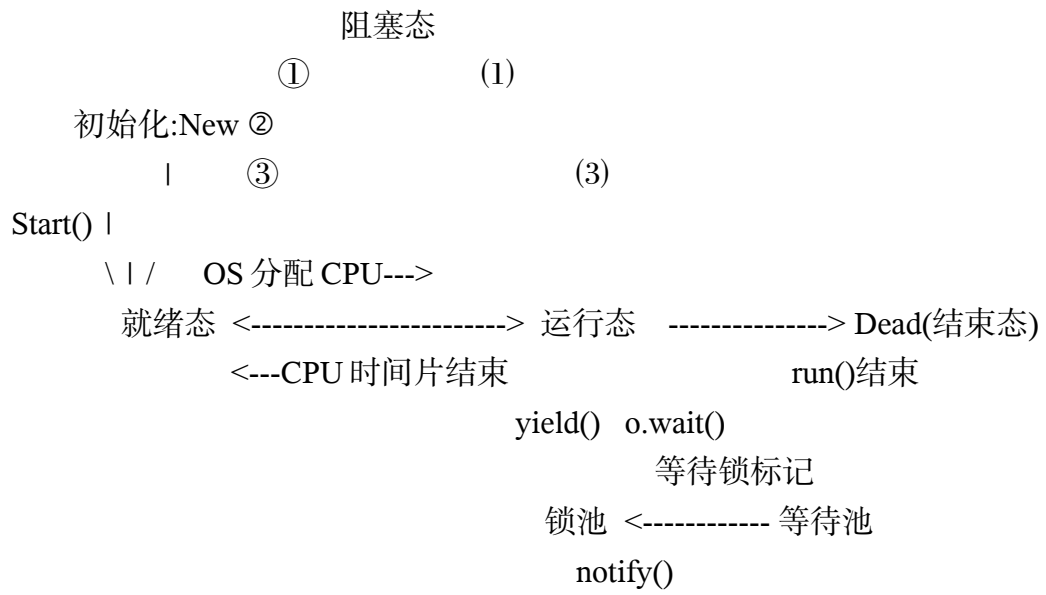
```
class MyThread  implements Runnable{
        public  void run(){
                //需要进行执行的代码，如循环。
        }
}
```

启动线程时不同前者：

```
public static void main(){
        Runnable myThread = new MyThread();
        Thread t = new Thread(myThread);
        t.start();
}
```

3、线程的生命周期（状态转换）：

1）下面为线程中的7种非常重要的状态：（有的书上只有前五种状态：而将“锁池”和“等待池”都看成是“阻塞”状态的特殊情况：这种认识也是正确的，但是将“锁池”和“等待池”单独分离出来有利于对程序的理解）



2）注意：图中标记依次为

①输入完毕；②wake up；③t1退出

⑴如等待输入（输入设备进行处理，而CUP不处理），则放入阻塞，直到输入完毕。

⑵线程休眠sleep()

⑶t1.join()：指停止main()，然后在某段时间内将t1加入运行队列，直到t1退出，main()才加入可运行队列。

特别注意：①②③与(1)(2)(3)是一一对应的。

4、Thread的方法

1）public static void sleep(long millis) throws InterruptedException{}

1.括号中以毫秒为单位,使线程停止一段时间，间隔期满后，线程不一定立即恢复执行。

2.当main()运行完毕，即使在结束时时间片还没有用完，CPU也放弃此时间片，继续运行其他程序。

```
Try{
        Thread.sleep(1000);
}Catch(InterruptedException e){e.printStackTrace();}
```

2）public final void join() throws InterruptedException{}

1.表示其他运行线程放弃执行权，进入阻塞状态，直到调用线程结束。

2.实际上是把并发的线程变为串行运行。

3.线程的优先级：1-10，越大优先级越高，优先级越高被 OS 选中的可能性就越大(不建议使用，因为不同操作系统的优先级并不相同，使得程序不具备跨平台性，这种优先级只是粗略地划分)。

注：程序的跨平台性 ---> 除了能够运行，还必须保证运行的结果。

3）public static void field()

使当前线程马上交出执行权，回到可运行状态，等待 OS 的再次调用。

4）public final Boolean isActive()

验证当前线程是否是活动的，不管它是否正在运行。

5、互斥（共享数据的并发处理）

1）概念：两个线程修改共享资源时会出现数据的不一致，产生排斥。

2）解决方法：为避免这种现象采用对访问的线程做限制的方法。利用每个对象都有一个 monitor(锁标记)，当线程拥有这个锁标记时才能访问这个资源，没有锁标记便进入锁池。JAVA 中任何一个对象上有且仅有一把锁。

1．synchronized 修饰代码块：对括号内的对象加锁，只有拿到锁标记的对象才能执行该代码块

```
public void push(char c){
    synchronized(this){
        ...
    }
}
```

2．synchronized 修饰方法：对当前对象的加锁，只有拿到锁标记的对象才能执行该方法

```
public synchronized void push(char c) {
    ...
}
```

注意：1.方法的 synchronized 特性本身不会被继承，只能覆盖。

2.解决互斥问题的核心是一定要找到共享对象。

3.线程因为未拿到锁标记而发生阻塞进入锁池（lock pool）。每个对象都有自己的一个锁池的空间，用于放置等待运行的线程。由系统决定哪个线程拿到锁标记并运行。

6、死锁：

1）概念：若干个线程共享有限的资源，某个线程只能获得一部分共享对象的锁，而拿不到足够的锁，导致每个线程都无法正常运行。

注意：锁标记如果过多，就会出现线程等待其他线程释放锁标记，而又都不释放自己的锁标记供其他线程运行的状况，就是死锁。

2）避免死锁：所有线程以相同的顺序去获得锁

死锁的两种处理方法：统一排列锁顺序(解决不同方法中对多个共享资源的访问)

对象 1 的方法：

```
synchronized(a)
synchronized(b)
```

对象 2 的方法

```
synchronized(a)
synchronized(b)
```

7、同步：线程间通信(也就是线程间的相互依赖)

1）概念：若干个线程中一个线程必须依赖另一个线程运行到一定的程度才能执行。

注意：线程间通信使用的空间称之为对象的等待池（wait pool），该队列也是属于对象的空间。

2）解决方法：

1.找共享对象；

2.独立去看每个线程类内部情况；

3.必须确立线程的 wait 发生在 notify 之前（使用标识变量）；

注：线程由于调用了 wait()而导致阻塞一定会释放锁。

3）流程：

1.进入等待池：使用 Object 类中 wait()的方法，在运行状态中，线程调用 wait()，此时表示线程将释放自己所有的锁标记和 CPU 的占用，同时进入这个对象的等待池。等待池的状态也是阻塞状态，只不过线程释放自己的锁标记。

wait()：交出锁和 CPU 的占用;

2.退出等待池进入锁池：

a. notify()：将从对象的等待池中移走一个任意的线程，并放到锁池中，那里的对象一直在等待，直到可以获得对象的锁标记。

b. notifyAll(): 将从等待池中移走所有等待那个对象的线程并放到锁池中，只有锁池中的线程能获取对象的锁标记，锁标记允许线程从上次因调用 wait()而中断的地方开始继续运行。

注意：只能对加锁的资源进行 wait()和 notify()。

5）注：在 java.io 包中 Vector 和 HashTable 之所以是线程安全的，是因为每个方法都有 synchronized 修饰。

8、线程的优先级：

Thread.yield()：把资源让给比它高的优先级线程，如果它的优先级是 10 无用

setPrioirity()：设置优先级 1 ~ 10，好字越大优先级越高，默认为 5

==========

第十四章 标准 I/O 流与文件

1、I/O 流基础：

1）Input/Output：指跨越出了 JVM 的边界，与外界进行数据交换。

注意：1.输入：以应用程序为准，往程序中放入代码

2.输出：程序往外发送

3.输入/输出是针对 JVM 而言。

2）流的分类：

1.从数据类型分：字节流和字符流

2.从数据方向分：输入流和输出流

3.从流的功能分：节点流和过滤流

3）对文件的操作

1.File 类(java.io.File)：可表示文件或者目录(在 JAVA 中文件和目录都属于这个类中，而且区分不是非常的明显)。

a. File 下的方法是对磁盘上的文件进行磁盘操作，但是无法读取文件的内容。

b. 注意：创建一个文件对象和创建一个文件在 JAVA 中是两个不同的概念。前者是在虚拟机中创建了一个文件，但却并没有将它真正地创建到 OS 的文件系统中，随着虚拟机的关闭，这个创建的对象也就消失了。而创建一个文件才是在系统中真正地建立一个文件。

c. 例如：File f = new File("1.txt"); //创建一个名为 1.txt 的文件对象

f.createNewFile(); //真正地创建文件

2.File 的方法

Boolean createNewFile(); //创建文件

Boolean mkdir(); //创建目录

Boolean mkdirs(); //创建多个目录

Boolean delete(); //删除文件

Boolean deleteOnExit(); //在进程退出的时候删除文件，这样的操作通常用在临时文件的删除。

String[] list(); //返回当前 File 对象下所有文件和目录名（相对路径）

File[] listFiles(); //返回当前 File 对象的所有 Files 对象，可用 getName()获得文件名。

isDirectory()和 isFile(); //判断究竟是目录还是文件。

String getParent(); //得到父类文件名

File getParentFile(); //。。。

String getPath(); //。。。路径

3.处理跨平台性

a.对于命令：File f = new File("d:\\a\\b\\1.txt")

这个命令不具备跨平台性，因为不同的 OS 的文件系统的分隔符是不相同的。

b.使用 File 类的 separator 属性，返回当前平台的文件分隔符。

```
File newD = new File("aa"+File.separator+"bb"+File.separator+"cc");
File newF = new File(newD,"mudi.txt");
try{
newD.mkdirs();
newF.createNewFile();
}catch(Exception e){}
```

2、I/O 输入输出

1）InputStream 类：所有字节输入流的父类，如 FileInputStream，ObjectInputStream，PipedInputStrean。

1.三个基本的 read()方法

a. int read()：从流里读出的一个字节或者-1(实际读了多长)；

b. int read(byte[])：将数据读入到字节数组中，并返回所读的字节数(期望读多长)；

c. int read(byte[], int , int)：两个 int 参数指定了所要填入的数组的子范围。

2.其它方法

a. void close()：关闭流，如使用过滤器流，关闭栈顶部的流，会关闭其余的流。

b. int available()：返回可从流中读取的字节数。

c. skip(long)：丢弃了流中指定数目的字符。

d. boolean markSupported()

e. void mark(int)

f. void rese()

2）OutputStream 方法

1.三个基本的 read()方法

a. void write()

b. void write(byte[])

c. void write(byte[], int , int)

2.其它方法

a. void close(): 关闭流，如使用过滤器流，关闭栈顶部的流，会关闭其余的流。

b. void flush(): 允许你强制执行写操作。

3.注意：在流中 close()方法由程序员控制。因为输入输出流已经超越了 JVM 的边界，所以有时可能无法回收资源。

原则：凡是跨出虚拟机边界的资源都要求程序员自己关闭，不要指望垃圾回收。

3）FileInputStream 和 FileOutputStream

1.结点流，使用磁盘文件。

2.要构造一个 FileInputStream, 所关联的文件必须存在而且是可读的。

3.要构造一个 FileOutputStream 而输出文件已经存在，则它将被覆盖。

FileInputStream infile = new FileInputStream("myfile.dat");

FileOutputStream outfile = new FileOutputStream("results.dat");

FileOutputStream outfile = new FileOutputStream("results.dat",true); //参数为 true 时输出为添加，为 false 时为覆盖。

FileOutputStream 类代码：(为什么能建文件)

```
public FileOutputStream(String name){
        this(name!=null?new File(String):null,false);
}
```

4）DataInputStream 和 DataOutputStream：过滤流，输出输入各种数据类型。

通过流来读写 Java 基本类，注意 DataInputStream 和 DataOutputStream 的方法是成对的。

writeBoolean(boolean b) ------以 1 byte 数据传送

writeByte(int) ------以 1 byte 数据传送

writeBytes(String s) --------以 byte 序列数据传送

writeChar(int v) ————————以 2 byte

writeChars(String s)----------以 2 byte 序列

writeDouble(double d) -------以 8 byte
writeInt(int v)
writeLong(long l)
writeShort(short s)

writeUTF(String)----------能输出中文!

5）ObjectInputStream 和 ObjectOutputStream：过滤流，处理对象的持久化。

```
Object o = new Object();
FileOutputStream fos = new FileOutputStream("Object.txt");
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(o);
oos.close();
FileInputStream fis = new FileInputStream(“Object.txt”);
ObjectInputStream ois = new ObjectInputStream(fis);
Object o = (Object)Ois.readObject();
ois.close();
```

6）BufferInputStream 和 BufferOutputStream：过滤流，可以提高 I/O 操作的效率

1.用于给节点流增加一个缓冲的功能。

2.在 JVM 的内部建立一个缓冲区，数据先写入缓冲区，等到缓冲区的数据满了之后再一次性写出，效率很高。

3.使用带缓冲区的输入输出流的速度会大幅提高，缓冲区越大，效率越高（这是典型的牺牲空间换时间）。

4.切记：使用带缓冲区的流，如果数据数据输入完毕，使用 flush 方法将缓冲区中的内容一次性写入到外部数据源。用 close()也可以达到相同的效果，因为每次 close 都会使用 flush。一定要注意关闭外部的过滤流。

7） PipedInputStream 和 PipedOutputStream：用来在线程间通信.

```
PipedOutputStream pos = new PipedOutputStream();
PipedInputStream pis = new PipedInputStream();
try{
        pos.connect(pis);
        new Producer(pos).start();
        new Consumer(pis).start();
}
catch(Exception e){
        e.printStackTrace();
}
```

8）Reader 和 Writer

1.Java 技术使用 Unicode 来表示字符串和字符，而且提供 16 位版本的流，以便用类似的方法处理字符。

2.InputStreamReader 和 OutputStreamWriter 作为字节流与字符流中的接口。

3.如果构造了一个连接到流的 Reader 和 Writer，转换规则会在使用缺省平台所定义的字节编码和 Unicode 之间切换。

4.字节流与字符流的区别.

5.编码与解码：编码是把字符转换成数字存储到计算机中；把数字转换成相应的字符的过程称为解码。

6.编码方式的分类：

ASCII（数字、英文）:1 个字符占一个字节（所有的编码集都兼容 ASCII）

ISO8859-1（欧洲）：1 个字符占一个字节

GB-2312/GBK：1 个字符占两个字节

Unicode: 1 个字符占两个字节（网络传输速度慢）

UTF-8：变长字节，对于英文一个字节，对于汉字两个或三个字节。

9）BufferedReader 和 PrinterWriter：两种好用的字符 I/O 类，都是过滤流。

BufferedReader 的方法：readLine():String

PrintWriter 的方法：println(…String,Object 等等)

10）随机存取文件

1.实现了二个接口：DataInput 和 DataOutput;

2.只要文件能打开就能读写;

3.通过文件指针能读写文件指定位置;

4.可以访问在 DataInputStream 和 DataOutputStream 中所有的 read()和 write()操作;

5.在文件中移动方法:

a. long getFilePointer(): 返回文件指针的当前位置。

b. void seek(long pos): 设置文件指针到给定的绝对位置。

c. long length(): 返回文件的长度。

11）持久化

1.将一个对象存放到存储器上称为持久化;

2.只有对象的数据被串行化;

3.标记为 transient 关键字的数据不被串行化。

12）RandomAccessFile 类

RandomAccessFile(String name,String mode)
RandomAccessFile(File name,String mode)

注意：重要的方法

length()：long 返回文件长度

seek(long len)：void 定位文件指针

readLine()：String

writeUTF(String str)：void

12）对象序列化与反序列化

1. 对象序列化：将对象转换为字节流保存起来，并在日后还原这个对象，这种机制叫

做对象序列化；

2. 一个对象要想能够实现序列化，必须实现 Serializable 接口或 Externalizable 接口；

3. 当一个对象被序列化时，只保存对象的非静态成员变量，不能保存任何的成员方法和静态的成员变量；

4. 当我们利用对象的反序列化重新构建对象的时候，它并不会调用这个对象当中的任何构造器，它仅仅是根据我们先前保存对象的状态信息在内存中重新还原这个对象；

5. 如果一个可序列化的对象包含一个不可序列化的对象的引用(例如 Thread 对象)，那么整个序列化操作将会失败，并且会抛出一个 NotSerializableException。我们可以将这个引用标记为 transient，这个对象就不会参与到序列化中，先前对象仍然可以序列化。

6. 对象的序列化：可利用 ObjectOutputStream 中的 writeObject()方法；

对象的反序列化：可利用 ObjectInputStream 中的 readObject()方法；

注意：这两个方法声明为 private 时，在类外仍然是可以访问的，是 java 中仅有的两个特例!

13）字节流与字符流

源地址 Java 程序
目标地址

字节流：
硬盘 FileInputStream FileOututStream 硬盘
键盘 System.in System.out
控制台
内存 ByteArrayInputStream ByteArrayOutputStream 内存

字符流：
硬盘 FileReader FileWriter
硬盘
键盘 new InputStreamReader(System.in) System.out 控制台
内存 CharArrayReader CharArrayWriter 内存
StringReader StringWtriter

==========

第十五章 网络编程

1、网络基础知识

1）网络:由不同物理位置彼此相对独立的计算机联机组成。

2）IP 地址：IP 地址是一个逻辑地址，IP 网络中每台主机都必须有一个惟一的 IP 地址；因特网上的 IP 地址具有全球唯一性；IP 地址为 32 位，4 个字节，常用点分十进制的格式表示，例如 192.168.0.16。IP 地址分为以下几类：

1. A 类地址

A 类地址第 1 字节为网络地址，其它 3 个字节为主机地址。另外第 1 个字节的最高位固定为 0。

A 类地址范围：1.0.0.1 到 126.155.255.254。

A 类地址中的私有地址和保留地址：10.0.0.0 到 10.255.255.255 是私有地址(所谓的私有地址就是在互联网上不使用，而被用在局域网络中的地址)；127.0.0.0 到 127.255.255.255 是保留地址，用做循环测试用的。

2. B 类地址

B 类地址第 1 字节和第 2 字节为网络地址，其它 2 个字节为主机地址。另外第 1 个字节的前两位固定为 10。

B 类地址范围：128.0.0.1 到 191.255.255.254。

B 类地址的私有地址和保留地址：172.16.0.0 到 172.31.255.255 是私有地址；169.254.0.0 到 169.254.255.255 是保留地址。如果你的 IP 地址是自动获取 IP 地址，而你在网络上又没有找到可用的 DHCP 服务器，这时你将会从 169.254.0.0 到 169.254.255.255 中临得获得一个 IP 地址。

3. C 类地址

C 类地址第 1 字节、第 2 字节和第 3 个字节为网络地址，第 4 个个字节为主机地址。另外第 1 个字节的前三位固定为 110。

C 类地址范围：192.0.0.1 到 223.255.255.254。

C 类地址中的私有地址：192.168.0.0 到 192.168.255.255 是私有地址。

4. D 类地址

D 类地址不分网络地址和主机地址，它的第 1 个字节的前四位固定为 1110。

D 类地址范围：224.0.0.1 到 239.255.255.254

3）Mac 地址：每个网卡专用地址，也是唯一的。

1.端口(port)：是一种抽象的软件结构(包括一些数据结构和 I/O 缓冲区)，端口使用一个 16 位的数字来表示，它的范围是 0~65535(OS 中可以有 65536（2^16）个端口)，1024 以下的端口号保留给预定义的服务，例如 http 使用 80 端口。

a.进程通过端口交换数据，连线的时候需要输入 IP 也需要输入端口信息。

b.计算机通信实际上是主机之间的进程通信，进程的通信就需要在端口进行联系。

如：192.168.0.23:8080(8080 为端口号)

c.端口是一种抽象的软件结构，与协议相关：TCP23 端口和 UDT23 端口为两个不同的概念。

d.端口应该用 1024 以上的端口，以下的端口都已经设定功能。

2.协议：为进行网络中的数据交换（通信）而建立的规则、标准或约定(协议=语义+语法+规则)，不同层具有各自不同的协议。

注意：不同层的协议是完全不同的。

4）OSI Model（开放系统互联模型）：

OSI 参考模型包括 7 个层次：应用层、表示层、会话层、传输层、网络层、数据链路层、物理层。

网络层：寻址、路由（指如何到达地址的过程）

传输层：端口连接

5）TCP/IP 模型：

TCP/IP 模型包括 4 个层次：应用层、传输层、网络层、网络接口

1. Application(FTP,HTTP,TELNET,POP3,SMPT)
2. Transport(TCP,UDP)
3. Network(IP,ICMP,ARP,RARP)
4. Link(Device driver,….)

注意：

ARP(Address Resolution Protocol，地址解析协议)：将 IP 地址转换成 Mac 地址；

RARP(Reflect Address Resolution Protocol，反相地址解析协议)：与上相反；

ICMP(Internet Control Message Protocol)：检测链路连接状况，利用此协议的工具有 ping， traceroute。

2、网络程序设计

1）socket(套接字)：为了能够方便的开发网络应用软件，由美国伯克利大学在 Unix 上推出了一种应用程序访问通信协议的操作系统调用 socket。后来，套接字又被引进了 Windows 等操作系统中。Java 语言也引入了套接字编程模型。

2）基于 TCP 的 socket 编程：

1.TCP Socket：

a. 服务器分配一个端口号，服务器使用 accept()方法等待客户端的信号，信号一到打开 socket 连接，从 socket 中取得 OutputStream 和 InputStream。

b. 客户端提供主机地址和端口号使用 socket 端口建立连接，得到 OutputStream

和 InputStream。

2. 建立 TCP 服务器端

服务器程序编写：

① 调用 ServerSocket(int port)创建一个服务器端套接字，并绑定到指定端口上；

② 调用 accept()，监听连接请求，如果客户端请求连接，则接受连接，返回通信套接字；

③ 调用 Socket 类的 getOutputStream()和 getInputStream 获取输出流和输入流，开始网络数据的发送和接收；

④ 最后关闭通信套接字。

程序范例：

```
ServerSocket server = new ServerSocket(post)
Socket connection = server.accept();
ObjectInputStream put = new ObjectInputStream(connection.getInputStream());
ObjectOutputStreamo put = newObjectOutputStream(connection.getOutputStream());
处理输入和输出流;
关闭流和 socket。
```

3. 建立 TCP 客户端

客户端程序编写：

① 调用 Socket()创建一个流套接字，并连接到服务器端；

② 调用 Socket 类的 getOutputStream()和 getInputStream 获取输出流和输入流，开始网络数据的发送和接收；

③ 最后关闭通信套接字。

程序范例：

```
Socket connection = new Socket(127.0.0.1, 7777);
ObjectInputStream input = new ObjectInputStream(connection.getInputStream());
ObjectOutputStream utput = new ObjectOutputStream(connection.getOutputStream());
处理输入和输出流;
关闭流和 socket。
```

3）建立 URL 连接

1.URL 和 URLConnection 类封装了大量复杂的从远端服务器获取信息的操作，其工作在 TCP/IP 的应用层。

2.支持 HTTP 和 FTP 资源

```
URL url = new URL("http://www.enterinfo.net")
```

URL url = new URL("ftp://TonyDeng@enterinfo.net")

3.打开一个网络连接

URLConnection urlconn = url.openConnection()

4.打开一个输入流

inputStream is = url.openStream()

或者

BufferInputReader bir = new BufferInputReader(new InputStreamReader(urlconn.getInputStream());

4）基于 UDP 的 socket 编程：

1. UDP socket：这种信息传输方式相当于传真，信息打包，在接受端准备纸。

2. 特点：

a.基于 UDP 无连接协议；

b.不保证消息的可靠传输；

c.它们由 Java 技术中的 DatagramSocket 和 DatagramPacket 类支持。

DatagramSocket(邮递员)：对应数据报的 Socket 概念，不需要创建两个 socket，不可使用输入输出流。

DatagramPacket(信件)：数据包，是 UDP 下进行传输数据的单位，数据存放在字节数组中，其中包括了目标地址和端口以及传送的信息(所以不用建立点对点的连接)。

3. DatagramPacket 的分类：

用于接收：DatagramPacket(byte[] buf,int length)
DatagramPacket(byte[] buf,int offset,int length)

用于发送：DatagramPacket(byte[] buf,int length,InetAddress address,int port)
DatagramPacket(byte[] buf,int offset,int length,InetAddress address, int port)

注：InetAddress 类网址用于封装 IP 地址没有构造方法，通过
InetAddress.getByAddress(byte[] addr):InetAddress
InetAddress.getByName(String host):InetAddress
...

4. 建立 UDP 服务器

接收端程序编写：

① 调用 DatagramSocket(int port)创建一个数据报套接字，并绑定到指定端口上；

② 调用 DatagramPacket(byte[] buf, int length)，建立一个字节数组以接收 UDP 包；

③ 调用 DatagramSocket 类的 receive()，接收 UDP 包；

④ 最后关闭数据报套接字。

程序范例：

```
byte[] buf = new byte[1024];
DatagramSocket datagramSocket = new DatagramSocket(13);      // set port
DatagramPackage intputPackage = new DatagramPackage(buf,buf.length);
datagramSocket.receive(inputPackage);
DatagramPackage outputPackage = new DatagramPackage(buf,buf.length,
inetAddress,port);
datagramSocket.send(outputPackage);
```

没建立流所以不用断开。

5. 建立 UDP 客户端

发送端程序编写：

① 调用 DatagramSocket()创建一个数据报套接字；

② 调用 DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port)，建立要发送的 UDP 包；

③ 调用 DatagramSocket 类的 send()，发送 UDP；

④ 最后关闭数据报套接字。

程序范例：

```
byte[] buf = new byte[1024];
DatagramSocket datagramSocket = new DatagramSocket();        //不用设端口
DatagramPackage outputPackage=new DatagramPackage(Buf, buf.length,
serverAddress, serverPort);
DatagramPackage inputPackage=new DatagramPackage(buf,buf.length);
datagramSocket.receive(inputPackage);
```

Core Java 笔记于 2008 年 5 月 20 日全部完成！！！

Tiger 学习笔记

==========

主讲：刘涛(Robin)

day1

第一章 Java5.0 新特性

1、 j2se5.0：Tiger

j2se6.0：Mustang

2、jdk5 新特性：

1）泛型（Generics）：参数化类型 *****

```
List<student> list
list.add(Student);
list.add(Dog); //complie error
(Student)list.get(2)
ClassCastException
```

2)自动装箱/拆箱 *****

```
Integer i = new Integer(10);
Integer j = 10;  //jdk5.0 自动装箱
int a = i.intValue();
int b = i;  //jdk5.0 自动拆箱
```

3）对 for 循环功能的增强 *****

```
for(定义循环变量；循环条件；改变循环变量){
.....
}
for(int i=0;i<=100;i++){.....}
for(定义集合中数据类型变量：集合){.....}
for(Student s : list){
.....
}
```

4）增加了类型安全的枚举类型

```
class,interface,enum
```

5）可变参数

```
sum(int... args){      //参数个数不确定，可用“...”代替
        for(int i=0;i<args.length;i++){
                total+=i;
        }
        rerurn total;
}

main(){
        sum(1,3,5);
}
```

6）静态导入

```
import java.io.*;
import java.net.*;
import static java.lang.System.*; //导入包中静态的属性或者方法(如 System.out.println();)
import static java.lang.Math.*;
```

out.println();
random();

7)元数据(Metadata)：用来描述其它数据的数据
Annotation(标注、注释) ---> EJB 3.0

第二章 基础知识

1、自动装箱、拆箱
1）自动装箱：将基本数据类型转换成包装类型
2）自动拆箱：将包装类型转换成基本数据类型
注意：自动装箱、拆箱由编译器帮我们实现，我们不需要进行其它处理

Integer ----> int i.intValue()
String ----> int Integer.parseInt(String i)
int ----> Integer Integer.valueOf(int i)

IDE：Eclipse，IBM，Borland公司，JBuilder

创建一个自动装箱、拆箱测试例子：
tiger_jd0805 //new Project
src //Project layout 中选第二个
--com.briup.day1(package) //new package
--BoxingTest //new class

自动格式化：ctrl+shift+f
自动导包：ctrl+shift+o
自动注释：ctrl+shift+/
取消注释：ctrl+shift+\
列出所有快捷方式：ctrl+shift+l

&&和&的区别：
1）&&逻辑与运算符，&位与运算符
2）&也可以进行逻辑运算，&不具有短路功能
if((1==1)&(1==2))

2、for循环

for(定义集合中存放数据相同类型的变量:要遍历集合){}

1)语法：for(variable : collection/Array){}

2)比较

1.jdk5.0之前

```
for(Iterator iter=list.iterator;iter.hasNext();){
        Object o = iter.next();
        System.out.println(o);
}
```

2.jdk5.0之后

```
for(Object o : list)
        System.out.println(o);
```

注意：在迭代输出时不能删除元素----->缺陷

3、类型安全的枚举类型

1）enum：可以定义一个类型，而且还可以限制该类型的取值在某个范围内

2）语法：enum enumName{枚举值1，...，枚举值n}

3）适用位置：包中、类中，但不能在方法中

4）注意：1.所有的enum类型都继承自java.lang.Enum；

2.自定义的enum类型不能再有子类；

3.枚举类型的取值默认为public static final；

4.枚举类型中可以定义属性和方法；

5.构造器必须是私有的；

6.枚举类型的取值就是其实例对象，它会调用该枚举类型的构造器，还要实现枚举类型中所有的抽象方法；

7.枚举类型可以实现其它接口；

```
src
    --com.briup.day1
        --EnumTest
        --TafficLight
        --MyGrade
        --EnumSetTest
        --EnumMapTest
        --VarargsTest
```

面试题：

1.单例(单太)模式：饿汉式、懒汉式

1）私有的构造器；

2）私有、静态的属性（本类的类型）；

3）共有的方法暴露静态属性。

举例：

```
public class Student {
        private static Student s = new Student(); //饿汉式(懒汉式：private static Student s)
        private Student(){}
        public synchronized static Student newInstance(){
                if(s == null)
                        s = new Student();
                return s;
        }
        public static void main(String[] args) {
                Student s1 = Student.newInstance();
                Student s2 = Student.newInstance();
                Student s3 = Student.newInstance();
                System.out.println(s1);
                System.out.println(s2);
                System.out.println(s3);
        }
}
```

2.谈谈 java 中的集合框架?

根据集合存放数据的特点，分为两类：

1）只存单值： List , Set

ArrayList：以数组的方式管理数据，线程不安全的，适合于做大量的查询操作；

思考：怎么变成线程安全的(java.util.Collections)?

-----> synchronized List(List<T> list)方法；

LinkedList：线程安全的,以双向链表的方式管理数据，适合于做大量的增加、删除操作；

Vector：不适合查询，线程安全的，适合处理多线程集合；

2）存 Key-Value 键值对： Map

HashMap（线程不安全的）

Hashtable（线程安全的）

TreeMap(根据 HashCode 进行排序）

4、可变参数

1）注意：1.可变参数只能用在方法的参数列表中

2.可变参数可以指定 0 个或多个实参

3.可变参数在参数列表中只能出现 1 次

2）重载：1.发生在同一个类中

2.相同的方法名

3.不关心返回类型

4.参数列表必须有所不同

5.构造器也可以被重载

3）重写：1.发生在父子类之间

2.相同的方法名

3.相同的返回值类型

4.相同的参数列表

5.可见性不能被缩小

6.异常不能被扩大

7.final 方法不能被重写（也叫覆盖）

8.静态方法不能被覆盖，但可以被子类中同名的方法所隐藏

切记：1.所有的属性取决于编译时类型

2.所有的静态方法取决于编译时类型

3.所有的普通方法取决于运行时类型（多态）

作业：1.String,StringBuffer,StringBuilder 的区别?

String：不可变字符串（固定字符串），在大量字符串进行连接时不使用 String；

StringBuffer：Java5 中的可变字符串，线程安全的,不考虑线程的时候使用；

StringBuilder：线程不安全的，性能较高，考虑多线程的情况下进行大量字符串连接时使用；

2.纸牌游戏：52 张牌发给 4 个人，每人 10 张

牌型：13（枚举）

花色：4（循环）

String s = "ACE of Hearts"; //表示红桃 A，其它自己想

注意：所有牌放到集合中（准备牌）；

如何打乱集合顺序（洗牌）：java.util.Collections ---> shuffle(注意与 java.util.Collection 的区别)；

如何去掉集合中部分元素（发牌）：subString，subList.

day2

5、静态导入（static import）：

1）语法：import static ...

2）功能：并没有增加大的功能性的改进，只是允许我们将静态的属性和方法导入到程序中，以简化我们的编程；

3）注意：1.import 导入的是包中的类或接口；

2.import static 导入的是类或接口中静态的属性和方法。

src
--com.briup.day2
--StaticImportTest；
--MyStatic

6、格式化输入输出&Scanner（引入 C 和 C++中的 printf）:

1）格式化输出语法：System.out.printf(format(格式),args（参数）,...);
2）format 格式：[argument][flag][width(宽度)][.precision（精度）]type
%[flag][width][.precision]type
1.type：是必须的,如：%d %s %f，注意有几个百分号后面就跟几个参数。
2.argument:利用“<”重复传入上一个参数值，而不是传入下一个参数
3.flag:指定输出格式化的标记，如："%-5s"中的“-”表示字符左对齐；“+”表示输出的数字带+号。
4.width：指定输出字符的宽度,表示字符的个数，如"%5s"中的"5"表示字符宽度为 5。
5..precision：表示小数的位数，如"%5.2f"中的“.2”表示输出 2 位小数。
3)Scanner:用于输入的类，可以利用他获得从键盘上输入的信息
例如：Scanner s = new Scanner(System.in)//获得字节流
int n = s.nextInt();

练习：接收键盘输入，然后将内容打印在控制台上（System.in）；
用 java.util.Scanner 完成同样的功能（参考 API）；

7、从控制台上向 java 程序内部传递参数有几种方式?
1）应用程序参数
run...--->Java Application--->右键 New（ArgTest）--->Arguments--->Program aguments--->briup1 briup2 briup3
在程序内部通过 main 中 args 取得：如 java Test briup1 briup2 briup3
2）虚拟机参数
java -Dname=briup -Dage=10
在程序内部通过 java.util.Property prop = System.getProperties();获取
例：Properties.load(new FileInputStream(zh.property))；
Properties(key,value);
Properties.getProperty(key);

Name:

zh.property

name=名字:

en.property
name=Name:

-Dname=briup
-Dage=10

第三章 泛型

1、泛型：

1）定义：即参数化类型，允许我们对放入集合中的具体类型进行限制；

2）语法：类名/接口名<类型参数>；例如 List<Student>;

3）优点：1.使用泛型可以保证你的代码是类型安全的，类型检查会在编译阶段完成，在运行时不会出现 ClassCastException;

2.可以增加程序的可读性；

4）运用：实际的开发中，主要运用在集合（Collection）中
HashSet TreeSet ArrayList LinkedList

2、子类型

1）对象类型和引用之间可以有继承关系；

2）对象类型和引用类型的类型参数间不能有继承关系；

3）放到集合中的元素的类型为类型参数本身，或者是其子类型。

3、通配符<?>

当集合中的具体类型不确定时，我们可以使用通配符，表示集合中可以存放所有 Object 及其子类型的元素；

4、限制通配符<? extends typeName>

表示集合中的元素为 typeName 所指定的类型，或其子类型；

限制类型是在定义自己的泛型的时候使用的；

自己定义的泛型的语法：

1）没有使用限制的情况下

modiflers keyword(class/interface) ClassName/InterfaceName<E>/<K,V>

E,K,V 表示占位

数量决定于开发应用需求

2）对于泛型类型有限制的情况

modiflers keyword(class/interface) ClassName/InterfaceName<T extends SuperClass>

T 表示占位，可以接受父类以及所有的子类

5、定义泛型类

1）声明一个类/接口，后面加上泛型信息： class A<E>{}

2）类型参数一般用一个大写字母表示，只表示一个占位作用；

3）在整个泛型类中，我们都可以使用该类型参数；

4）类型参数的实际类型，由调用者在调用时给出。

Map.Entry

--EntryMap

6、裸类型

1.定义：类/接口使用时，如果没有给出泛型信息，我们就称之为裸类型；

2.裸类型在使用时是类型不安全的，在运行时可能会出现 ClassCastException.

7、类型擦除

泛型只在编译阶段起作用，在编译阶段编译器通过泛型信息来进行类型检查；

在运行阶段所有的泛型信息都不存在了，即在.Class 文件中所有的泛型信息已经被剔除掉了

——TypeEraseTest.java

练习：1.有两个字符串，a="1983 年 03 月 20 日"，b="2008 年 08 月 08 日"，问两个字符串所指代的日期相差多少天(java.text.SimpleDateFormat)?

2.如何用最简单的方法构建多个内容相同的对象，要求对一个对象内容的修改，不会影响到其它对象？(涉及到 java 克隆机制)

```
Student s1 = new Student("briup",10,...);
Student s2 = s1;
s1.setName("ibm");
s2.getName();
```

3.求某一年是否是闰年，给出程序(java.util.GregorianCalendar)。

闰年：4 年闰，100 年不闰，400 年闰。

day3

第四章 Annotation

1、Annotation(标注、注释):

1）概念：是对 java 中注释功能的增强，可以用来对类、属性、方法等进行标注；

2）功能：使用 Annotation 可以改变我们的编程风格，允许我们进行声明式编程，我们只要给出声明，大量的代码可以由工具帮我们自动产生；

```
Employee{
    String name;
```

```
        int age;
        ...
        transient double salary;
    }
```

2、Sun 公司给出的标准 Annotation:

1）@Deprcated：用来标识类、属性、方法等。已经过时，不建议使用了。
---> StandardAnnotation

2）@Override：用来标识某一个方法是重写父类中的方法，编译器会看到此标识会自动检查该方法是否满足重写要求。

3）@SuppressWarnings('unchecked')：禁止产生警告信息

Warning：警告，可以不进行处理.

引申：
Error
Exception(异常)
1.checked exception:
必须要处理，要么 try...catch 捕获，要么 throws 向上抛出(冒泡处理机制)；如 IOException

注意：

1. try{...}finally{...} 没有 catch 块也是可以的
2.
```
interface A{
        public void say() throws e1;
}
interface B{
        public void say() throws e2;
}
interface C extends A,B{
        public void say() throws ...?
}
```
a.如果 e1（父类）和 e2（子类）有继承关系，抛 e2 或不抛;
b.如果 e1 和 e2 没有继承关系，不抛。

2.unchecked exception:可以不进行处理，如 NullPointerException,ClassCastExeption

3、相关知识：

1）POJO：Plain Old Java Object
com.briup.pojo

2）javadoc

4、定义 Annotation

1）说明：

1.Annotation 是一种特殊的接口；

2.使用@interface 声明 Annotation: public @interface Info{}
---> Info

3.Annotation 中可以包含方法(元素)；

4.Annotation 中的方法不能有参数，不能抛异常；

5.Annotation 中方法的返回值类型有限制，只能是 primitives,String,Class,enums,annotaions 或者是以上类型的数组；

6.Annotation 的方法可以有默认值：methodName() default defaultValues;

2）使用 Annotation

1.Annotation 的使用和 public、static、final 等类似，可以对类、属性、方法等进行修饰、限制

2.使用语法：

```
@annotationName(
        methodName = Value,
        ...
        methodName = value
    )
```

5、Annotation 分类

1）标记 Annotation:

1.在 Annotation 中没有任何方法（元素）,只起到一个标识作用

2.语法：@interface annotationName{}

3.使用：@annotationName(),如@Deprecated,@Override

2）单值 Annotation：

1.在 Annotation 中只有一个方法（元素）

2.语法：@interface annotationName{
        String value();
    }

3.使用：@annotationName("value");
如：SuppressWarnings("unchecked")

3）普通 Annotation：

1.在 Annotation 中可以出现多个方法（元素）

2.语法：@interface annotationName{

```
            method1;
            ...
            methodn;
        }
    3.使用：@annotationName(
            method1 = value,
            ...
            methodn = value
        )
      如:@Info
```

6、Meta-Annotation：元 Annotation

1）作用：用来描述、限制其它 Annotation 的 Annotation

2）Sun 公司提供了两个常用的元 Annotation：

1.@Retention

用来描述、限制 Annotation 的持久力，即 Annotation 的有效时长

取值：a.SOURCE:表示 Annotation 只在原文件中有效

b.CLASS：表示 Annotation 在编译阶段有效，默认取值

c.RUNTIME：表示 Annotation 在运行阶段有效

语法：@Retention(...)

@Retention 对应 java 类：java.lang.annotation.Retention;

取值对应 java 类：static java.lang.annotation.RetentionPolicy.*;

2.@Target

用来描述、限制 Annotation 所修饰的对象

取值：a.TYPE：表示该 Annotation 可以修饰 class,interface,enum 类型

b.FILED：表示该 Annotation 只能修饰属性

c.METHOD:表示该 Annotation 只能修饰方法

.....

@Target 对应 java 类：java.lang.annotation.Target;

取值对应 java 类：static java.lang.annotation.ElementType.*;

7、在程序内部处理 Annotation

1）reflect:反射

```
Class Test{
        public static void main(String[] args){
            String name = args[0];
```

```
                Class.forName(name).newInstance();
            }
            public void hello(){}
    }
```

2）获取镜像：Test a = new Test();

1.Class.forName(className);
2.a.getClass();
3.Test.class;

3）java.lang.reflect.*;

Method

Feild

---> MetaTest

作业：1.谈一下深度克隆（深拷贝）和浅克隆（浅拷贝）？

2.1）写一个 Annotation，并用@Retention,@Target 进行限制？

2）写一个有多个方法的类 A，对类中的部分方法用 Annotation 进行修饰？

3）再写一个测试类，统计一下类 A 中有多少个方法被 Annotation 修饰，有多少方法没被 Annotation 修饰，并用反射调用一下所有被 Annotation 修饰的方法？

Tiger 笔记于 2008 年 5 月 28 日全部完成！！！

XML 学习笔记

==========

主讲：何兆睿
hezr@briup.com

XML 前言

课程内容：

1、基本语法

2、DTD（文档类型定义）

3、XML Namespace

4、XML Schema

5、XML 解析（使用 Java）

1）使用程序语言（必须支持解析 XML）对 XML 文件进行读写

2）两种解析模型：DOM 和 SAX

第一章 XML Overview

1、XML：扩展标记语言

1） 概念： 标记——代码由两部分组成，标记和内容
扩展——标记完全自定义，可以根据需要自行扩展

2） 作用：用于保存特定的信息，内容为保存的信息，标记用于指定保存信息的含义
例如：<id>10</id> -----> 开始标记 内容 结束标记

3） 保存大量数据（扮演数据库角色）的弊端：纯文本格式，效率较低
保存少量数据的优势：数据含义明确，标准化格式，便于维护和管理

4） 主要应用领域：
1、配置文件（重点）
2、系统间信息的交换

5） 与 HTML 的区别：
HTML——超文本标记语言
共同点：都是标记语言
不同点：HTML 的标记指定的是信息的显示格式，XML 的标记指定的是信息的内容含义，HTML 的标记预定义，
XML 的标记自定义。
如：<h1>XML(/h1> HTML
<name>XML</name> XML

2、 XML 的规范：
1）结构良好的（well-formde）：符合 XML 基本语法的文档
语法要求： a、大小写敏感；
b、有且仅有一个根标记；
c、开始标记必须对应结束标记；
d、如果标记没有内容可以使用<标记/>的格式（省略结束标记）；
e、标记之间的关系只有两种：平行或嵌套，不允许交叉；
XML 可以使用属性表示信息，信息的逻辑含义与使用内容表示相同
如：<id>10</id>相当于<id value="10"/>
格式：属性名="属性值"
f、属性必须写在开始标记中，具有特定的格式，属性值必须以""或"包含；

g、每个标记中可以定义任意数量的属性，但属性名称必须不同；

注意：虽然属性和标记在数据逻辑上完全相同，但考虑到解析实现的可操作性，通常建议使用标记内容的方式保存信息。

2）有效的（valid）：在1的基础上，符合DTD或XML Schema的格式要求

由DTD或XML Schema的约束特定XML文件的标记和属性的使用规则，以实现除语法格式标准化外的内容格式标准化，为标准化解析和其它应用构建平台；

3、 注意：

XML格式：符合XML语法要求的文件都是XML格式的文件，扩展名有几十种，常用的有.xml .xsd .tld 等；

XML文件：扩展名为.xml的文件

4、 <?xml version="1.0" encoding="UTF-8"?>

XML的指示信息，用于对XML文件进行说明

可以不写（使用默认值），如果写必须出现在文件的第一行

注意：如果在文件中出现中文信息，通常将encoding属性设置为gb2312或gbk

语法：<?...?>

5、 注释：<!--... ...-->

注释中的内容不被处理，但如果出现中文，encoding属性也需要作相应的设定

6、 标记

1）标记中可以包含同名标记

2）混合标记——标记中既包含子标记又包含内容

在解析中，混合标记操作相对复杂，通常不建议使用

处理方法： 将混合标记中的内容部分替换一个合适的子标记，该标记用于包含内容；如果内容分散，可以使用多个同名标记；

例如：

```
<chapter>                          <chapter>
    Chapter information                <content>Chapter information</content>
    <para>What is XML</para>   --->    <para>What is XML</para>
    <para>What is HTML</para>          <para>What is HTML</para>
    More chapter information           <content>More chapter information</content>
<chapter>                          <chapter>
```

7、 XML数据类型：

PCDATA：处理XML时将会对数据进行处理，所有没有说明的内容都属于该类型（99%）

CDATA：处理 XML 时不对数据进行处理，需要使用特定的语法进行说明，在内容包含大量特殊字符时使用较多

例如：
```
<nameXML>
<![CDATA[
<name common="freddy" breed="springer-spaniel">
        Sir Frederick of Ledyard's End
</name>
        ]]>
</nameXML>
```

8、 标记和属性的命名：

基本与 Java 相同，但不能使用"xml*"和"XML*"作为标记和属性的名称

9、 DTD 和 XML Schema

1）DTD：原有的用于约束 XML 格式的文档格式，存在很多缺点，已经逐渐被 XML Schema 取代，但仍然有大量的

系统在沿用早期的配置方式（使用 DTD）

2）XML Schema：DTD 的升级版本

10、命名空间：

1) Java 以 package 形式出现

如 com.briup.md02.Student 和 com.briup.md05.Student

2) xml 以 namespace 形式出现

如<namespace:标记名称>.....</namespace:标记名称>

11、Web Service

SOAP

第二章 DTD

DTD：Document Type Definition

不是 XML 格式，扩展名为.dtd

定义标记：

<!ELEMENT 标记名 (内容)>

注意：每个<!ELEMENT>用于定义一个标记，无论这些标记在 xml 中的关系如何，在 dtd 中都是平行的

内容的常用形式：

1、#PCDATA：表示内容为文本内容（信息），对应 XML 的 PCDATA+CDATA

2、子标记

符号的含义

, 顺序出现

? 0-1

* 0-无穷

+ 1-无穷

| 选择左右两边元素的其中一个

注意：在内容位置出现的所有标记都必须被定义

3、空

使用 EMPTY 关键字表示，没有括号

表示该标记是空标记，不能包含内容和子标记

4、任意内容

使用 ANY 关键字表示，没有括号

表示该标记的内容不作限制，不建议使用（使用则失去 DTD 存在的意义）

注意：当出现(a|b|c)*或(a|b|c)+时，是对整体执行*或+，子标记可以为以下结构：

<a>_
_
<c>_</c>
.....

在 xml 中，有三种方式定义与 dtd 的关联，符合关联的 dtd 文件的格式要求的 xml 文件为有效的 xml 文件(valid)

1、内部 dtd：直接将 dtd 的内容写在 xml 文件中

在 xml 中的格式：

<!DOCTYPE 根标记 [DTD 的内容]>

缺点：定义的 dtd 只能被当前 xml 文件使用

2、（文件）系统 dtd：dtd 文件位于和 xml 文件相同的文件系统中

格式：<!DOCTYPE 根标记 SYSTEM "dtd 文件路径">

如：<!DOCTYPE address SYSTEM "address.dtd" >

SYSTEM 表示关联 dtd 的类型

3、外部 dtd：dtd 文件来自于任何可以访问的外部资源位置

格式：<!DOCTYPE 根标记 PUBLIC "dtd 描述" "dtd 的 URL">

PUBLIC 表示关联 dtd 的类型

定义属性类型

1、ID：属性值不能重复

2、枚举类型：不使用关键字，格式为(枚举值1|枚举值2|...)，

该类型属性值必须指定为列出的枚举值中的1个

3、CDATA：属性值可以是任意内容

定义默认值

1、#REQUIRED：该属性必须出现

2、#IMPLIED：该属性为可选的

3、直接指定默认值

4、#FIXED 值：该属性的值是固定的，不能指定其它值

实体：

1、预定义的实体：五个，对应的内容为><"'&

a>b a>b

2、自定义的实体：按照需求定义，关系为实体名称和实体内容的一一对应，使用时指定实体名称，相当于将对应的实体内容放到指定位置

实体名称：briup 实体内容：aaaaaaaaaaaaaaaaaaaaaa

<value>bbbb&briup;ccc</value>相当于<value>bbbaaaaaaaaaaaaaaaaaaaaaaccc</value>

定义实体：

1、直接使用 ENTITY 定义

2、使用扩展名为.ent 的文件定义实体

文件内容为实体内容

在 DTD 文件中使用<!ENTITY 实体名称 SYSTEM "实体文件路径">方式定义

在 xml 文件中使用方式与 1 相同

DTD 缺陷：

1、不能支持命名空间，因为命名空间是属于 xml 的，而 dtd 不是 xml

2、无法精确定义数据类型，dtd 对标记内容只有#PCDATA 一种

3、无法精确定义标记，如某标记出现 2-5 次

4、对于属性也存在类似问题(2)

以上问题在 xml Schema 中可以得到解决

第三章 XML Namespaces

1、命名空间：

作用：由于 xml 的标记是完全自定义的，所以出现同名不同义的标记的可能性很大，

为了更好的对这些标记进行区分，通常可以给标记增加一个新的名字，这个名字就是命名空间；

Java 的包名使用域名的反序保证唯一性，如：
域名为 www.briup.com，则包名为 com.briup.xxx

xml 的命名空间采用 URI 的方式保证唯一性，如：
域名为 http://www.briup.com，则命名空间为 http://com.briup.com/xxx
注意：URI 外观类似于网络地址，但实际只是利用该机制保证其唯一，不一定是有效的访问地址。

```
<catalogEntry>
    <book>
        <title>this book</title>
    <isbn>0001</isbn>
    <author>
        <title>Dr.</title>
        <lastName>Zhu</lastName>
        <firstName>George</firstName>
    </author>
    </book>
</catalogEntry>
```

2、定义命名空间：
在标记中定义，语法类于定义属性，允许在一个标记中定义任意数量的命名空间
格式：xmlns:前缀="命名空间"
使用命名空间：
为需要指定命名空间的标记和属性增加命名空间的前缀
命名空间的有效范围（作用域）：
在定义命名空间的标记和其子标记中都可以使用（包含这些标记的属性）

```
<catalogEntry>
    <book:book xmlns:book="http://www.briup.com/book"
                    xmlns:author="http://www.briup.com/author">
    <book:title>this book</book:title>
    <book:isbn>0001</book:isbn>
    <book:author>
        <author:title>Dr.</author:title>
        <author:lastName>Zhu</author:lastName>
```

```
            <author:firstName>George</author:firstName>
        </book:author>
        </book:book>
    </catalogEntry>
```

指定两个命名空间

http://www.briup.com/book 和 http://www.briup.com/author

<book>的子标记指定命名空间为 http://www.briup.com/book

<author>的子标记指定命名空间为 http://www.briup.com/author

3、命名空间的前缀(prefix):

1）通常命名空间的内容很长，为了在使用中简化，使用前缀进行对应，前缀通常使用有实际意义的单个单词组成；

2）注意:命名空间可以保证全世界范围的唯一性，而前缀只需要在当前 xml 文件中唯一，所以可以使用较短的字符串；

3）前缀只是命名空间使用时的一个代号，真正用于识别的名字依然是命名空间，类似于数据库中的表名和别名；

4、 标记的完整语法格式：

<前缀:标记名>...</前缀:标记名>

标记名——局部名称

前缀:标记名——全限定名（注意：不是全名）

属性的完整语法格式：

前缀:属性名="属性值"

5、No Namespace——没有命名空间

当标记或属性没有指定任何命名空间时，称为 no namespace；

也就是指标记或属性只有 localname 部分，通常不建议使用；

6、Default Namespace——默认命名空间

基本概念：前缀为空字符串，对应一个具体的命名空间，使用时可以不使用任何前缀，但指定了具体的命名空间，提高代码的可读性，同时大大简化了代码

如果一个 xml 文件中存在多个命名空间，通常选择使用最多的命名空间作为默认命名空间

语法：xmlns="命名空间"

代码 1：定义了默认命名空间

<book>的命名空间为"http://www.briup.com/book",id 属性属于 no namespace，默认命名空间对属性无效；

注意：标记没有指定前缀并不一定属于 no namespace

如果定义了 default namespace，则属于 default namespace;

属性没有指定前缀一定属于 no namespace

```
<book xmlns="http://www.briup.com/book" id="001">
        ... ...
</book>
```

代码 2：定义了命名空间

<book>的命名空间是 no namespace

```
<book xmlns:book="http://www.briup.com/book">
        ... ...
</book>
```

代码 3：没有定义命名空间

<book>的命名空间是 no namespace

```
<book>
        ... ...
</book>
```

7、对于默认命名空间与属性不能直接关联的解决方法：

1）定义默认命名空间用于所有的标记

2）定义命名空间指向与默认命名空间相同的 URI，用于属性

（两个前缀指向同一个命名空间）

8、Namespace 的用途：

对于开发工具和大多数用于浏览和查看 xml 的工具（如浏览器），不会对命名空间的有效性和正确性作任何的检查和校验；通过代码操作 xml 文件时，可以读取标记或属性的命名空间信息，用于识别标记和属性及实现一些扩展的功能。

练习：

```
<book:catalogEntry xmlns:book="http://www.briup.com/book"
xmlns:author="http://www.briup.com/author">
  <book:book>
    <book:title>this book</book:title>
    <book:isbn book:id="001"/>
    <book:author>
      <author:title>Dr.</author:title>
```

```
            <author:lastName>Zhu</author:lastName>
            <author:firstName>George</author:firstName>
        </book:author>
    </book:book>
</book:catalogEntry>
```

保持原有命名空间定义，合理使用默认命名空间简化代码；

```
<catalogEntry xmlns="http://www.briup.com/book"
xmlns:book="http://www.briup.com/book"
xmlns:author="http://www.briup.com/author">
    <book>
        <title>this book</title>
        <isbn book:id="001"/>
        <author xmlns="http://www.briup.com/author">
            <author:title>Dr.</author:title>
            <author:lastName>Zhu</author:lastName>
            <author:firstName>George</author:firstName>
        <author>
    <book>
<catalogEntry>
```

DTD 不支持 Namespace，其对 xml 文件的格式要求只指定 localname

XML Schema 支持 Namespace，其对 xml 文件的格式要求可以指定 lacalname 和 namespace

如：

DTD 可以指定文件的根标记<book>；

XML Schema 可以指定文件的根标记为<book>，同时必须属于"http://www.briup.com/book"这个 namespace，但对前缀不作任何要求；

第四章 XML Schema

1、XML Schema:

1）XML Schema 是 XML 格式，扩展名为.xsd

2）XML Schema 类似于 Java 中的集合，本身是对象，用于存放对象

3）XML Schema 本身是 xml，用于限定 xml 的格式

2、类型系统：

Schema 拥有强大的，具有良好扩展性的类型系统，包括两部分

1）内置类型：如整数，字符串

2）扩展类型(自定义)：将内置类型或扩展类型整合定义新的类型，在符合语法要求的情况

下可以任意定义

xsd 的格式有国际标准组织定义，其文件结构本质就是定义
Schema 类型系统的方式，所有标记定义必须属于"http://www.w3.org/2001/XMLSchema"这个命名空间

3、格式：
1）根标记为<schema>
2）定义 xml 标记使用
a.使用内置类型
<element name="标记名" type="标记类型"/>
如：<element name="id" type="int"/>
对应的 xml 文件内容为：<id>100</id>

b.使用扩展类型
<element name="标记名">具体的扩展细节</element>

扩展类型分为：
简单类型<simpleType>：只包含内容
复杂类型<complexType>：包含属性或子标记或两者都有
如：
<id>100</id> 简单类型
<id num=100/> 复杂类型
3）定义简单类型
方法一：定义标记，将类型作为标记子标记定义
在类型只被一个标记使用时通常用该方式

```
<simpleType>
    <restriction base="integer">
        <minInclusive value="2"/>
        <maxInclusive value="5"/>
    </restriction>
</simpleType>
```

当需要某个类型基础上做范围缩小时，可以使用以上格式
<restriction base="基础类型">具体的缩小方式</restriction>
<minInclusive> 最小值(含)
<minExclusive> 最小值(不含)
<maxInclusive> 最大值(含)
<maxExclusive> 最大值(不含)

方法二：定义类型，为类型取名，在标记定义时通过 type 属性指定使用的类型
在类型需要被多个标记使用时使用该方式可以减少代码冗余

4）定义复杂类型

情况 1：包含内容和属性，没有子标记

```
<complexType>
    <simpleContent>
        <extension base="内容的类型">
            <attribute name="属性名称" type="属性类型" default="属性默认值"/>
            <!--更多的属性-->
        </extension>
    </simpleContent>
</complexType>
```

定义组：

形式 1：所有标记必须顺序出现

```
<group name="组名">
    <sequence>标记</sequence>
</group>
```

形式 2：所有标记必须顺序出现，但顺序不作要求

```
<group name="组名">
    <all>标记</all>
</group>
```

形式 3：选择其中的任何一个标记出现

```
<group name="组名">
    <choise>标记</choise>
</group>
```

注意：组不是复杂类型，在定义复杂类型时包含子标记时可以使用已经定义的组，在一组标记被多个复杂类型使用时可以减少代码的冗余。

情况 2：只有属性，没有内容和子标记

```
<complexType>
    <attribute ... .../>
</complexType>
```

情况 3：只包含子标记

两种格式:

格式一：

```
<complexType>
```

```
        <!--引用已经定义的组-->
    </complexType>
    格式二：
    <complexType>
        <!--定义包含的子标记，语法与组的定义相同（三种组合方式）-->
    </complexType>
```

情况 4：包含子标记和内容（混合方式，不建议使用）

与情况 3 的定义方式相同，在<complexType>标记中增加属性 mixed="true"(该属性默认为 false)

情况 5：任何复杂类型中属性需要自定义类型

```
<!--外层标记省略-->
    <attribute ... ...>
        <simpleType>
        <!--定义具体的类型及约束,与标记的简单类型格式相同-->
        </simpleType>
    </attribute>
```

情况 6：包含子标记和属性

```
<complexType mixed="..."> //true 就是有内容，false 就是没有内容
<!--定义子标记-->
<!--定义属性-->
</complexType>
```

4、命名空间的使用和关联：

1）xsd 文件

```
<schema
        xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.briup.com/xml"
        elementFormDefault="qualified">
```

定义 xsd 文件使用的 namespace:

xmlns="http://www.w3.org/2001/XMLSchema"

定义 xsd 文件定义的格式的 namespace

targetNamespace="http://www.briup.com/xml"

定义 targetNamespace 的有效范围：

elementFormDefault="qualified"表示所有标记都应用 targetNamespace

elementFormDefault="unqualified"表示根标记应用 targetNamespace，其它标记

应用 no namespace，是默认形式

2）xml 文件

```
<quantity xmlns='http://www.briup.com.cn/xml'
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation='http://www.briup.com.cn/xml/schema.xsd'>
```

定义当前文件使用的 namespace，通常与对应的 xsd 文件的 targetNamespace 一致；
xml="http://www.briup.com/xml"

定义应用 xsd 文件的 xml 文件的固定的 namespace
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

第五章 XML 解析

XML 解析：使用程序语言（必须支持解析 XML，我们使用 Java）对 XML 文件进行读写操作；

两种解析模型：DOM 和 SAX

1、DOM：

1）最常用的解析模型

2）操作方式：读取时，将整个 XML 文件读入内存，保持原有文件的所有结构，依据节点的关系或全文档查询进行读取；写入时，先进行读取（增加内容）或创建新的文档（新建 XML 文件），然后依据节点的关系进行写入操作，写入完成后同步到文件。

3）优点：可读可写，操作直观；

4）缺点：操作大文件时占用资源较多；

使用 JavaBean 保存读取的信息

JavaBean 规范：

注意：不是所有的 JavaBean 都必须完全遵循规范！

1.实现 Java 的序列化接口；

2.属性执行封装（private 修饰，提供 public 的 get 和 set 方法）；

3.包含无参数构造器；

符合以上条件的 Java 类称为 JavaBean，又称为 pojo 类型。

相关的类：

1.DocumentBuiderFactory：获得 dom 解析器的工厂类

2.DocumentBuilder：dom 解析器所属的类

3.Document：文档对象所属的类

普通的节点的输出形式：[节点名称: null]
[saluation: null]
文字节点的输出形式：[#text: 节点内容]
[#text: Ms.]
转换为字符串之后的输出：节点内容
Ms.

ftp://briup:briup@192.168.1.220/jd0806/xml
model javabean
dom 操作类型

注意：原始的 xml 文件是没有任何间隙的，所有的标记和内容连成一片，DOM 方式默认生成的文件使用该结构，该结构可以避免因为空白可能带来的读取的偏差，在不涉及使用人员的情况下（XML 文件完全由代码生成和读取）建议使用该格式。如果需要便于程序员阅读，可以通过参数指定。

2、SAX：

1）使用事件处理模型操作 XML 文件

2）操作原理：将 XML 文件以流的形式读入，通过捕捉其中的语法元素（通过事件）执行相应的代码，以此达到解析的目的；

捕捉以下 7 个语法元素：

1.文档开始
2.文档结束
3.标记开始
4.标记结束
5.内容
6.命名空间开始
7.命名空间结束

3）优点：采用分段读入的方式（每次读入 2048 字节，由解析器控制），资源消耗少；

4）缺点：不可往回读取，同时无法执行写操作；

事件三要素：

1.事件源：被读取的 XML 文件；

2.事件对象：由 SAX 解析器维护，保存与读取操作有关的信息；

3.事件监听器：需要自定义，实现特定的接口或继承特定的类型，覆盖七个方法分别对应 XML 文件中的七类语法元素，当读取特定语法元素时执行相应的方法；

XML 笔记于 2008 年 7 月 20 日全部完成！！！

Oracle 学习笔记

==========

主讲：邓远辉

前言

一、关系型数据库基础知识

1、why?
Java 编写的程序不可避免要和数据库交互；
Oracle 是商业领域市场占有率第一的产品；

2、what?
使用 SQL 语句操作 Oracle 进行 CRUD:
1）数据
2）数据库对象

CRUD：
1) Create 增
2) Retrieval 查
3) Update 改
4) Delete 删

s_student
---
name class gendar birthday tel
zs jd0806 male 1986-10-20 123
zs jd0806 male 1986-10-20 456
ls jd0806 male 1986-10-20 789

s_student
---
name class gendar birthday id(pk)
ls jd0806 male 1986-10-20 1
ww jd0806 male 1985-2-12 2

s_tel
---
tel name student_id(fk)
123 ls 1
456 ls 1
789 ls 1
666 ww 2
888

3、数据冗余

Primary Key(主键)
作用：唯一标识表中记录；
取值：1）唯一；
2）非空；

Foreign Key(外键)
作用：建立表间记录对应关系；
取值：1）要么为空；
2）要么为所引用列的列值；

4、数据库对象

1.表（table）
2.视图（view）：虚拟表，表现为一至多张表的部分数据内容；
本质上是一条有名称的查询语句；
3.sequence：用来产生依次递增/递减的值；
4.索引（index）：用来提升查询效率的对象；
5.同义词（synonym）：对象的别名，方便对对象的访问；
6.编程单元（program unit）:使用 PL/SQL 语言编写的
procedure,functiion,package,trigger

5、约束类型(5 种)：
1.主键
2.外键
3.非空
4.唯一
5.check

二、oracle 基础知识

数据库：

Microsoft	:	SQL Server
Oracle	:	Oracle
IBM	:	DB2
SUN	:	MySQL
		Sybase
...		...

默认端口号：1521

---

telnet 192.168.1.250 1521

SQL：结构化的查询语言(Structured Query Language)，也是一种智能语言，只关心结果，不关心过程；能以相同形式访问不同类型的数据库；

由 ANSI（美国国家标准委员会）维护；

1986	SQL1.0	SQL86
1992	SQL2.0	SQL92
1999	SQL3.0	SQL99

PL/SQL：Procedure Language(过程语言)

由 Oracle 公司制定，在 SQL 基础上增加了一些过程性的结构（变量、类型、条件、循环等），

只能访问 Oracle 数据库；

SQL*Plus：由 Oracle 提供的客户端软件；

（命令：sqlplus jd0806/jd0806）

SQL 语句：

1.数据获取语言(Data Retrieval)

SELECT

对数据内容进行查询；

2.数据操作语言(DML)

INSERT,UPDATE，DELETE

对数据内容进行修改；

3.数据定义语言（DDL）

CREATE，ALTER，DROP，RENAME，TRUNCATE(删除表内容)

对数据库结构进行改变；

truncate 和 delete 的区别：

1）truncate 不能回滚，即不能恢复；

2）truncate 会释放被删除记录所占空间；

4.事务控制(Transaction control)

COMMIT(提交),ROLLBACK(事务回滚),SAVEPOINT(设置事务的回滚点)

事务控制操作命令；

5.数据控制语言（DCL）

GRANT,REVOKE

对权限进行授予和收回；

第一章 Selecting Rows

1、distinct：相当的记录值，只显示一次。

1）例如：select distinct name from s_empl;

2）使用 distinct 时注意：

1.位置：紧跟在 select 之后；

s_student

```
id      name    score
1       zs      100
2       ls      90
3       ww      100
```

```
select          name, distinct score
from            s_student; //invalid
```

```
NAME            SCORE
-------- ---------
                        100
ls                      90
```

2.后跟多个列的处理：多列内容联合起来相同才显示一次；

2、登录方式：

1）sqlplus 用户名/密码

2）sqlplus 用户名

passwd:密码

3）sqlplus

username:用户名

passwd:密码

查看进程：$ ps -ef | grep sqlplus

3、三种数据类型：

1）数值：number 或 number(p,s)

p:精度，s:刻度（小数位数）

1234.567 number(6,2) --------> 1234.57
0.001234 number(2,4) --------> 0.0012
1234 number(3,-1)--------> 1230
1204 number(3) --------> 超出精度

```
SQL> create table t (n number(6,2));
SQL> insert into t values(1234.567);
SQL> select * from t;
```

2）字符：char, varchar, varchar2

char 与 varchar 的区别：

1.内容存储时占用的存储空间；

定长：char （声明固定长度）
'a' -> char(10) -> 10
'ab' -> char(10) -> 10

变长：varchar （根据实际存储的长度为准）
'a' -> varchar(10) -> 1
'ab' -> varchar(10) -> 2

2.所允许存储的最大长度；

char: 2000

varchar: 4000

varchar 与 varchar2 的区别:

1.制定厂商不同:

varchar:ANSI

varchar2:Oracle

2.承诺不同；

3）日期：date '01-JAN-08'

世纪，年，月，日，时，分，秒

1234 -> $99,999.99 -> $1,234.00
1234 -> $00,000.00 -> $01,234.00
1234 -> $999 -> ######

4、sql*plus 命令：

1）编辑命令：

A[PPEND] text：在当前行后追加内容；

C[HANGE]/old/new：用新内容替换当前行第一处符合条件的内容；

CL[EAR] BUFF[ER]：清空 buffer；

DEL：删除当前行；

DEL n：删除指定行；

I[NPUT] text：

L[IST] n：显示指定行内容；

n text：整行替换；

l：显示 buffer 中的命令；

/：重复执行 buffer 中的命令；

2）设置暂停命令：

设置：set pause {off|on|text}

显示：show pause

3）文件命令：

SAVE filename[.sql] [replace]：将 buffer 中的内容保存形成文件(自动追加后缀.sql)；

例如： save xuxiang
!more xuxiang.sql

GET filename：将文件内容写入 buffer，再敲入“/”重复执行该命令；

START filename：同“/”，执行文件内容；

@ filename：将文件内容 load 进 buffer 并执行；

EDIT filename：编辑；

SPOOL filename：

EXIT/QUIT：退出；

4）对列显示的格式进行控制：col[umn] [column_name] [option];

clear : 清除列的格式

format : 改变列的格式

heading :

justify : 对齐

如：column salary：查询 salary 列的格式

column salary clear：清空列的格式

column last_name heading 'Employee|name' format a15;

"|"代表换行；"a15"代表列的宽度为 15 个字符；

column salary justify left format $99,990.00;

column start_date format a8 null 'Not hired';

第二章 Limiting Selected Rows

1、order by

作用：根据指定列的内容对查询结果进行排序；

注意：1）位置：始终位于语句最后；

2）字句中允许出现的内容：

1. 列名

2. 别名

```
select      id,salary s
from        s_emp
order by    s desc;
```

3. 表达式

```
select      id,salary
form        s_emp
order by    salary *12;
```

4. 列/表达式出现在 select 后的序号

```
select      id,salary
form        s_emp
order by    2 desc;
```

3）order by 后面跟多列内容的处理：

先按前面列内容排序，

如果前面列内容重复，

则进一步按后面列决定重复记录的次序；

2、where

作用：对查询结果进行过滤与筛选；

注意：1）位置：紧跟在 from 子句之后；

2）内容：表达式

between...and... 限制为一个区间

in (...,...) 限制查找为固定的几个值

is null

is not null

like

%：代表 0 或多个字符

_：代表单一字符

\：代表转义字符

sql: select * from xuxiang_student where name like 'b____1%';

第三章 Single Row Functions

语法：function_name(column|expression,[arg1,arg2,...])

1、字符函数

LOWER： Converts to lowercase(转化为小写字符)
lower('SQL Course') -----> sql course

UPPER： Converts to uppercase(转化为大写字符)
upper('SQL Course') -----> SQL COURSE

INITCAP： Converts to initial capitalization
initcap('SQL Course') -----> Sql Course

CONCAT： Concatenates values(合成字符，同“||”)
concat('Hello','String') -----> HelloString

SUBSTR： Returns substring(获得部分字符)
substr('String',1,3) -----> Str

LENGTH： Returns number of characters(获得字符串的长度)
length('String') -----> 6

NVL： Converts a null value(空值置换函数)

2、数值函数

ROUND： Rounds value to specified decimal(取舍)
ROUND (45.923, 2) -----> 45.92
ROUND (45.923, 0) -----> 46
ROUND (45.923) -----> 46
ROUND (45.923, -1) -----> 50

TRUNC： Truncates value to specified decimal(去尾)
TRUNC (45.923, 2) -----> 45.92
TRUNC (45.923) -----> 45
TRUNC (45.923, -1) -----> 40

MOD： Returns remainder of division(取余)
MOD(1600,300) -----> 100

3、日期函数

1）Oracle 存储日期的内置格式：
Century, year, month, day, hours, minutes, seconds

2）默认的日期显示格式： DD-MON-YY.

3）sysdate：返回系统日期和时间的函数

4）dual：存储系统日期和时间的一个视图

```
select sysdate from dual;
select to_char(sysdate,'yyyy-mm-dd hh24:mi:ss')
select to_char(sysdate,'yyyy-mm-dd-hh-mm-ss')
select to_char(sysdate,'year month ddsp day ddspath')
select to_char(sysdate,'dy mon Mon MOn mON')
```

MONTHS_BETWEEN： 两个日期的相差月份数；
MONTHS_BETWEEN('01-SEP-95','11-JAN-94') -----> 19.774194

ADD_MONTHS： 在指定日期上增加月份；
ADD_MONTHS('11-JAN-94',6) -----> '11-JUL-94'

NEXT_DAY： 指定日期接下来的一天；
NEXT_DAY('01-SEP-95','FRIDAY') -----> '08-SEP-95'

LAST_DAY： 指定日期所在月份的最后一天；
LAST_DAY('01-SEP-95') -----> '30-SEP-95'

ROUND： 取舍；
ROUND('25-MAY-95','MONTH') -----> 01-JUN-95
ROUND('25-MAY-95 ','YEAR') -----> 01-JAN-95

TRUNC： 去尾；
TRUNC('25-MAY-95 ','MONTH') -----> 01-MAY-95
TRUNC('25-MAY-95 ','YEAR') -----> 01-JAN-95

4、转换函数

to_char： converts a number or date string to a character string.

to_number：converts a character string containing digits to a number.

to_date： converts a character string of a date to a date value.

注意：date 和 number 之间不能相互转换。

5、一个语句由多个子句组成；

```
SQL> select     id,last_name
  2  from   s_emp;
SQL> SELECT     id,last_name FROM s_emp;

SQL> select     salary+salary*commission_pct/100 s
  2  from   s_emp;
```

6、空值置换函数：如果前面的字段是空值,则用后面的字段代替。

结构：NVL(字段名,与前面的字段的类型要相同);

```
SQL> select    salary+salary*nvl(commission_pct,0)/100 s
  2  from      s_emp;
SQL> select    salary+nvl(salary*commission_pct,0)/100 s
  2  from      s_emp;
SQL> select    salary+nvl(salary*commission_pct/100,0) s
  2  from      s_emp;
```

7、save 文件名[.sql] [replace]

文件名:可以使用.sql，不加扩展名的话就默认为.sql

replace:表示可以覆盖，在 save 保存为.sql 文件后，想重新再保存,就要加上 replace

第四章 Displaying Data from Multiple Tables

1、what is a join?

1）where 子句；

2）能建立起两张表间记录的对应关系；

2、join type：

1）等连接：

```
select   s_dept.name,s_region.name
from     s_dept,s_region
where    s_dept.region_id = s_region.id;
```

注意：若无 where 子句则会产生两张表记录乘积的结果。

2）不等连接；

3）外连接：使用（+）

1.查询所有员工名字以及对应部门名字

所有员工 -> 相关联的部门表缺记录 -> +置于部门表这边

2.查询所有员工名字以及对应部门名字

所有员工 -> 相关联的员工表缺记录 -> +置于员工表这边

注意：当两张表出现不匹配的记录时要将其全部显示出来，必须使用外连接。

4）自连接；

```
1.select        s.name, f.name
  from s_dir s, s_dir f
  where         s.parent_id = f.id;

2.select        e.last_name,m.last_name
  from s_emp e,s_emp m
  where         e.id = m.manager_id(+);
```

第五章 Group Function(组函数)

1、常用组函数：默认不处理空值

a. 只能处理数值类型

avg, sum

b. 处理多种数值类型

max, min, count

1)select max(salary),min(salary),max(salary)-min(salary)
from s_emp;
2)s_student

```
-----------------------------------
id      name   score   class
1       zs             100          jd0806
2       ls             90           jd0806
3       ww             95           ajd0806
4       lz             80           ajd0806
-----------------------------------
select  avg(score)  from  s_student;
```

2、group by 子句

1.作用：将表中记录按特定列内容分组，然后将分组后的记录内容分别进行组函数运算；

2.注意：1) 位置：在 where 或 from 子句后；
2) 各子句执行的次序：
i. where
ii. group by
iii. group function
iv. order by
3) group by 子句中出现有多列内容的处理：多列内容联合起来相同作为一组
注意：先进行头一个字段的分组，如果遇到相同的字段，在进行下一个字段的分组；

3.如果 select 语句中包含组函数，但是 select 子句中有些列没有和组函数使用，那么这些列必须出现在 group by 子句中；

eg：s_emp

```
-----------------------------------
title     salary
t1              1000
t2              900
t3              800
-----------------------------------
select  title,  avg(salary)  from  s_emp;   // invalid
```

4.允许组函数出现的位置：
1）select 子句；
2）having 子句；

3）order by 子句；

3、having 子句

作用：与 where 同，但允许出现有 group function

注意：1）位置：紧跟在 group by 子句后；

2）各子句执行次序：

i. where
ii. group by
iii. group function
iv. having
v. order by

练习：查询工资低于公司平均工资员工的名称以及具体工资。

```
select      last_name,salary
from        s_emp
group by    last_name, salary
having      salary<avg(salary);     // invalid
```

步骤一：
```
select      avg(salary)
from    s_emp;
```

步骤二：
```
select      last_name, salary
from    s_emp
where   salary < 1750;
```

合并：
```
select  last_name,salary
from s_emp
where   salary < (select        avg(salary)
from    s_emp);
```

第六章 Sub queries(子查询)

1、what is subquery?

1）sub：嵌套在其它语句中，先执行；

2）query：一条完整的 select 语句；

3）subquery 可以位于哪些子句中?

a.where 子句

b.having 子句

c.from 子句

4）rownum(行号)：

rownum 列中保存的是行号,这个是系统给定的,不需要用户去创建；

注意：rownum 不能用于 between...and...语句，也不能使用>=与>和=(但可以等于 1)，只能用 < 或 <= 。

经典题型：

步骤一：select title
from s_emp
where last_name = 'Smith';

步骤二：select last_name
from s_emp
where title = '?');

整合：select last_name
from s_emp
where title = (select title from s_emp where last_name='Smith');

查询员工表中前三条记录的 last_name, id, salary;

```
select    last_name,id,salary
from      s_emp
where     rownum<4;
```

2、分页查询：

例如：

1）查询员工表中第 11 至第 20 条记录的 last_name, id, salary;

```
select    last_name, id, salary
from      (
          select  last_name, id, salary, rownum  r
          from    s_emp
)
where     r between 11 and 20;
```

2）查询员工表中工资前三名员工的工资以及 last_name；

笨方法:

```
create table xuxiang_emp1000
as
select  salary,last_name,rownum r
from            s_emp
order by        salary desc;

select  salary,last_name
from            (select salary,last_name,rownum r from xuxiang_emp1000)
where           r between 1 and 3;
```

好方法:

```
select  salary,last_name
from            (select salary,last_name
                from            s_emp
                order by        salary desc)
where           rownum <=3;
```

第七章 指定运行时变量

1、如何定义运行时变量：&

```
where last_name=&n;
Enter value for n : 'a'
where last_name='a';
where last_name='&n';
Enter value for n : a
where last_name=a;
```

定义变量:

```
select id,last_name from s_emp where id<&id;
select id from s_emp where last_name=&n;
select id from s_emp where last_name='&n';
```

set verify off|on：变量使用时提示信息开关

2、给变量传值：

1）运行时临时传递

2）使用 accept 传值

```
accept id number prompt 'please enter value for id: '
```

3）执行.sql 文件传值

```
SQL>  @myFile id last_name
```

例如：

```
SQL> !vi tony_0707.sql
"tony_0707.sql" [New file]
select  &1,&2
from    &3;
SQL> @tony_0707 id last_name s_emp

SQL> select     &c1,&c2
  2  from       &c3
  3  where      id<&c4;
```

编辑脚本文件：!vi tony_0706.sql

运行脚本文件：@tony_0706 / start tony_0706

3、解除绑定：undefine id

第八章 数据模型和数据库设计

1、读懂 E-R 图

E-R 图：Entity Ralationship Model

1）读懂属性前符号含义；

o：可选，对应列无约束；

*：一定有值，对应列应加 not-null 约束；

#*：非空、唯一，对应列为主键列或主键组成列；

(#)*：对应列加 not-null , unique 约束；

2）读懂关系连接含义；

----：对应表不一定有记录，对表结构没任何影响；

——：对应表一定有记录，外键列应再加上 not-null 约束；

伞状符号：多，该实体对应表应增加 foreign key 列；

竖线：UID bar，外键列同时也是主键组成列；

（注意：一个表如果有主键，只能有一个主键，但可由多个列组成该主键(联合主键)）

primary key (主键)：

作用:唯一区分表中记录;

取值要求:

1)唯一：联合唯一

2)非空：单列非空

一个表如果有主键,只能有一个主键,但可由多个列, 组成该主键(联合主键)

2、根据 E-R 图生成表实例图

Prinary key(pk):主键

作用：唯一区分表中不同记录；

取值：1）唯一（联合唯一）；

2）非空（单列非空）；

注意：一个表最多只有一个主键，但可以由多列组成

Foreign Key(fk):外键

作用：建立表间记录关系

取值：1）要么为空；

2）要么为所引用列的列值；

a. 所引用列有主键约束或唯一约束；

b. 所引用列即可来自本表，也可来自其他表；

第九章 Creating Tables(创建表)

例：

create table s_order(

```
        id      number(10)
        name    varchar2(20)
        num     number(10)
);
```

约束名称：

1）每一个约束均有一个唯一名称；

通过该名称可以对约束进行删除，修改操作；

2）名称即可以自定义也可由系统自动分配；

自定义：　　表名_列名_约束简写

自动分配：　sys_c00001

创建约束形式：

1）列级别

a.约束信息和列信息以 空格 分隔；

b.有些约束仅能通过列级别形式加：

not null;

2）表级别

a.约束信息和列信息间以 逗号 分隔；

b.有些约束仅能通过表级别形式加：

联合主键、联合外键、联合唯一键；

foreign key 外键中通过 references 去指定表名 列名

查看表是否有外键约束:

```
select   constraint_name,constraint_type
from             user_constraints
where            table_name = '要查看的表的大写表名';
```

复制表:只能拷贝 not null 约束,主键和外键非空唯一都不能被拷贝；

```
SQL> create table xuxiang_nn_test(
  2  id        number(10) not null,
  3  name      varchar2(20)
  4            constraint xuxiang_nn_test_name_nn
  5            not null
  6  );

create           table   xuxiang_uk_test(
id       number(10) unique,
name             varchar2(20),
```

```
constraint xuxiang_uk_test_name_uk
unique(name)
);

SQL> create table xuxiang_pk_test(
  2  id        number(10)      primary key,
  3  name      varchar2(20)
  4  );

SQL> create table xuxiang_pk_test2(
  2  id        number(10),
  3  name      varchar2(20),
  4  primary key(id)
  5  );

s_emp
-------------------------------jh
id        name   dept_id(fk)
1         zs              1
2         ls              2
3         ww              3

s_dept
--------------------------------
id        name
1         d1
2         d2

create table xuxiang_fk_test(
id        number(10),
p_id      number(10)
references xuxiang_pk_test(id)
);

SQL> create table xuxiang_fk_test2(
  2  id        number(10),
  3  p_id      number(10),
  4  constraint xuxiang_fk_test2_p_id_fk
  5  foreign key(p_id)
  6  references xuxiang_pk_test(id)
  7  );

xuxiang_dept
--------------------------------
id(pk)  name
```

```
1               d1
2               d2

xuxiang_emp
------------------------------
id(pk)  name    dept_id(fk)
1               zs              1
2               ls              2
3               ww              1
4               zl              1

create table s_dept(
        id      number(7)
                constraint s_dept_id_pk
                primary key,
        name    varchar2(25)
                constraint s_dept_name_nn
                not null,
        region_id       number(7)
                constraint s_dept_region_id_fk
                references s_region(id),
        constraint s_dept_name_region_id_uk
        unique(name,region_id)
);

create table xx_item(
        id      number(10),
        price   number(10,2),
        quantity        number(10),
        order_id        number(10)
                        constraint xx_item_order_id_fk
                        references s_order(id),
        constraint xx_item_id_order_id_pk
        primary key(id,order_id)
);

create table xuxiang_emp2000
        as
        select    id,last_name name,salary sal
        from s_emp
        where     id<6;
```

第十章 Oracle Data Dictionary(数据字典表)

1、数据字典表的分类

user：当前用户创建对象的信息；

user_tables:当前用户创建的 表 对象信息；

user_views:当前用户创建的 视图 对象信息；

user_indexs:当前用户创建的 索引 对象信息；
user_objects:当前用户创建的 对象 简要信息；

all：当前用户所能访问对象的信息；
all_tables:当前用户访问 表 对象信息；
all_views:当前用户访问 视图 对象信息；
all_indexs:当前用户访问 索引 对象信息；
all_objects:当前用户访问 对象简要 信息；

dba：所有用户创建对象信息；
dba_tables
dba_views
...

v$：数据库性能数据、版本等内容(数据库版本、日志、环境设置等信息)
v$version:保存有数据库当前的版本信息内容
ex:desc v$version
select * from v$version;

2、常用数据字典表的使用

dictionary：数据字典表的名称以及说明
user_objects:
user_tables:
user_constraints:
user_cons_columns:

ex: desc dictionary (只有两列,一是 table_name 数据字典表的表名,二是 comments 是对数据字典表的描述)
select * from dictionary where table_name = 'USER_TALBES';

primary key : p
foreign ket : r
check : c
unique : u
not null : c

user_constraints 所包含的列： constraint_name constraint_type table_name
user_cons_columns 所包含的列： constraint_name column_name table_name

外键的定义：

列级别：

ex：

create table 表名(列名属性 constraint 所要起的约束名称 references 所引用表（所引用列名));

表级别:

ex:

create table 表名(列名属性,..., constraint 约束名称 foreign key(做为外键的列名) references 所引用表名(所引用列名));

user_constraints
---
constraint_name constraint_type table_name

user_cons_columns
---
constraint_name column_name table_name

```
select    c.constraint_name,
          c.constraint_type,
          m.column_name
from      user_constraints c,
          user_cons_columns m
where     c.constraint_name=m.constraint_name
and       c.table_name='S_EMP';
```

第十一章 Manipulating Data(对表的操作)

1、DML

1）insert

```
insert into table_name[(column_name,...)]
values(v1,v2,...);

insert into table_name[(column_name,...)]
subquery;
```

2）update

```
update table_name
set    column_name=new_value[,...]
[where ...]
```

3）delete

```
delete  [from] table_name
[where ...]
```

2、Transaction Control：事务控制

事务：将多条操作组合在一起，同时成功，同时失败；

作用：保证数据的完整、一致；

四大特性： A 原子性：将多个操作组合在一个不能再细分的操作；

C 一致性：事务操作前后数据是一致的；

例如转帐前后帐户余额总和相等；

I 隔离性：一个事务看不到另一个事务中数据；

D 持久性：事务结束应将中间数据持久写入数据库中，以便所有用户可查；

操作：声明事务边界

1）开始边界

a. 通过 SQL*Plus 连入数据库；

b. 上一个事务结束；

2）结束边界

a. 正常结束：commit;

b. 异常结束：rollback ---> 回复到事务开始或之间某个位置状态；

指定回滚处: rollback to 回滚点的名字

c. 设置回滚点位置：savepoint

设置回滚点名: savepoint 回滚点的名字

s_account

id	name	balance
1	zs	1000
2	ls	1000

```
update s_account set balance=balance-100
where  id=1;

update s_account set balance=balance+100
where  id=2;
```

对事务的操作注意点如下:

*事务的自动提交：commit,create，如果遇到 create 时，就直接对事务进行提交，DDL 都有自己提交事务的功能；

*事务回滚：commit 后就不能对事务进行 rollback; 因为事务已经被提交了，不能再对事务进行回滚，也不能对其 savepoint；

*使用 rollback 和 savepoint 必须在事务没有被提交前使用，必须在遇到 commit 或者 create 之前使用

只针对 update insert delete 等操作;

create -> DDL -> 另一事务可查
insert -> DML -> 另一事务不可查
commit：事务结束命令

	commit 或 rollback 之前	commit 之后	rollback 之后
当前事务操作数据	当前事务可查 其他事务不可查	所有事务可查	数据丢失 所有事务不可查
回滚点	存在	丢失	丢失
锁	持有	释放	释放

使用 SQL 语句操作 Oracle 进行 CRUD:

1、数据：
- C: insert
- R: select
- U: update
- D: delete

2、数据库中对象：

Table:
C: create table
R: select * from user_tables
  where table_name='?';
U: alter table table_name
  add/modify/drop ...;
D: drop table table_name;

第十二章 Altering Tables and Constraints

1、更新表列(只能修改四种约束,五种约束中只有 not null 不可以)

1）新增表列

```
alter table table_name
add     (column_name column_type [default default_value] [not null]);
```

2）修改表列

```
alter table table_name
modify (column_name column_type [default default_value] [not null]);
```

a.相同类型

number(3) --> number(10) //valid

number(10)-->number(3)//视情况而定

b.不同类型

number(3)-->varchar2(20)//valid

varchar2(20)-->number(20)//视情况而定

c.新增非空约束-->//视情况而定

3）删除表列

```
alter table table_name
drop    column column_name;
```

4）修改列名

```
alter table table_name
rename column old_column_name
to new_column_name;
```

范例：

```
SQL> create table xuxiang_tran_test(
  2  id        number(10)
  3            primary key,
  4  name      varchar2(20)
  5  );
SQL> commit;
SQL> insert into xuxiang_tran_test
  2  values(1,'zs');
SQL> savepoint a;
SQL> select * from xuxiang_tran_test;
SQL> update xuxiang_tran_test
  2  set       name='ls'
  3  where     id=1;
SQL> savepoint b;
SQL> insert into xuxiang_tran_test
  2  values(2,'ww');
SQL> delete from xuxiang_tran_test;
SQL> select * from xuxiang_tran_test;
SQL> rollback to b;
SQL> select * from xuxiang_tran_test;
SQL> rollback to a;
SQL> select * from xuxiang_tran_test;
SQL> rollback;
SQL> select * from xuxiang_tran_test;
```

2、更新约束

1）新增约束

```
alter table table_name
add [constraint constraint_name]
constraint_type(column_name);
```

2）生效/失效约束

```
alter table table_name
enable/disable constraint constraint_name;
```

3）删除约束

```
alter table table_name
drop constraint constraint_name;
```

DML:
insert, update, delete

DDL:
create, alter, drop

```
SQL> create table xuxiang_cons_test(
  2  id        number(10),
  3  name      varchar2(20)
  4  );
SQL> alter table xuxiang_cons_test
  2  add       primary key(id);
SQL> alter table xuxiang_cons_test
  2  add constraint xuxiang_cons_test_name_uk
  3  unique(name);
SQL> select              constraint_name,
  2               constraint_type
  3  from        user_constraints
  4  where               table_name='XUXIANG_CONS_TEST';
SQL> alter     table xuxiang_cons_test
  2  add       (gender varchar2(1) not null);
SQL> desc xuxiang_cons_test
SQL> alter table xuxiang_cons_test
  2  drop      constraint xuxiang_cons_test_name_uk;
SQL> alter table xuxiang_cons_test
  2  drop      primary key;
SQL> drop table xuxiang_cons_test;  //删除表，注意删除后将不能恢复
SQL> create table xuxiang_emp6000
  2  as
  3  select    *
  4  from      s_emp;
SQL> rename xuxiang_emp6000 to xuxiang_emp8888;  //重命名表名
```

核心：使用 SQL 语句操作 Oracle 数据库进行 CRUD：

1、表数据：

C：insert into table_name values(...)

R：select ... from ...

U：update table_name set ... where ...

D：delete from table_name where ...

2、数据库对象

Table:

C: creat table table_name(...);

R: select * from user_tables
  where table_name='?';

U: alter table table_name
  add/modify/drop ...;

D: drop table table_name;

第十三章 Creating Sequences

1、what is sequence?

数据库中用来产生依次递增/减值的对象，主要用于作为主键值;

一个 sequence 对象，可服务于多张表，为了提升访问性能，可以使用缓存；

```
SQL> create sequence xuxiang_seq_first;
SQL> select     *
  2  from       user_sequences
  3  where      sequence_name='XUXIANG_SEQ_FIRST';
SQL> column sequence_name format a15;      //设置格式：宽度，目的为了使其一行显示
SQL> /
```

2、CRUD sequence

C: create sequence sequence_name

[increment by n] ---> n：步长值，可以为负，默认为 1

[start with n] ---> n：初值，设定主键产生的初始值，默认为 1

[maxvalue n] ---> n：最大值，步长为正时默认为 1*10^27

[minvalue n] ---> n：最小值，默认为 1

[cycle|nocycle] ---> 是否循环

[cache n|nocache]; ---> 是否放入缓存中,如放入值有多少个，默认为 20

R: select *
  from user_sequences
  where sequence_name='?';

U: alter sequence sequence_name
  [];

D: drop sequence sequence_name;

nextval:
先从内存中获取下一个 sequence 值，如无，则从硬盘中获取；

currval:
始终从内存中获取当前 sequence 值；

注意：在 sequences 当中初始值不可以被修改；
当删除 sequences 的时候，以 sequeces 获得的以前 table 中获得的记录还存在；

例如：

```
SQL> select    xuxiang_seq_first.nextval
  2  from      dual;
SQL> select    xuxiang_seq_first.currval
  2  from      dual;
SQL> select    *
  2  from      user_sequences
  3  where     sequence_name='XUXIANG_SEQ_FIRST';

SQL> create table xuxiang_seq_t(
  2  id        number(10)     primary key,
  3  name      varchar2(20)
  4  );
SQL> insert into xuxiang_seq_t(id,name)
  2  values(xuxiang_seq_first.nextval,'briup');
SQL> /                                          //重复执行
SQL> insert into xuxiang_seq_t(id,name)
  2  values(xuxiang_seq_first.nextval,'briup'||to_char(xuxiang_seq_first.currval));
SQL> /
SQL> select    *
  2  from      xuxiang_seq_t;
SQL> drop sequence xuxiang_seq_first;  //删除 sequence
```

第十四章 Creating View(创建视图)

1、what is view?
虚拟表，一致多张表的部分数据；
本质：一个有名称的查询语句；

s_emp

```
id      name   salary
1       zs             1500
2       ls             3000

select   id,name from s_emp;
desc s_emp;
select id,name,salary from s_emp;   //弊端：工资泄露，目的是不想泄露表中的一些机密

select          object_type              //查看对象的类型：表或视图
from            user_objects
where           object_name='?';
```

2、CRUD view

```
        C: create view view_name[(n1,n2...)]
           as
           subquery;
        R：select       view_name,text
           from         user_views
           where        view_name='?';
        U: create or replace view_name[(n1,n2...)]
           as
           subquery;
        D: drop view view_name;

SQL> create or replace view xuxiang_view_first
  2  as
  3  select    id,last_name,salary
  4  from      xuxiang_emp8888;
SQL> desc xuxiang_view_first;
SQL> select    *
  2  from      xuxiang_view_first;
SQL> insert into xuxiang_view_first          //注意：并不是所有视图都能进行 DML 操作，只有简单视图才可以。
  2  values(26,'jd0806',2000);
SQL> create or replace view xuxiang_view_first(            //改变别名
  2  id,name,sal)
  3  as
  4  select    id,last_name,salary
  5  from      s_emp
  6  where     id<5;

SQL> create or replace view xuxiang_v
  2  as
```

```
  3  select     id,last_name,salary
  4  from       xuxiang_emp168
  5  with read only;
SQL> select     view_name,
  2            text
  3  from       user_views
  4  where      view_name='XUXIANF_V';

SQL> create or replace view xuxiang_read_only(   //创建简单视图，不允许插值
  2    id,name,sal)
  3  as
  4    select  id,last_name,salary
  5    from    s_emp
  6    with read only;
SQL> select     view_name,text
  2  from       user_views
  3  where      view_name='XUXIANG_READ_ONLY';
SQL> column view_name format a30
SQL> drop view xuxiang_read_only;
```

第十五章 Creating Indexes(创建索引)

1、what is index?

rowid 是系统给定的 id 值

数据库中一种用来提升查询速度的对象;

1）按一定算法单独维护加了索引的列值;

2）按一定算法迅速定位;

3）通过 rowid 对符合条件记录进行定位;

注意：索引并不是总能提升效率;

2、CRUD

```
C: create index index_name
   on table_name(column_name);
R: select        *
   from          user_indexs
   where         index_name='?';
D: drop index index_name;
```

3、查寻方式:

B*Tree

位图
反向
降序
函数

```
SQL> select     id,name,region_id,rowid,rownum  //rowid 可以迅速获取对应列的内容
  2  from      s_dept;

SQL> create table xuxiang_table(
  2    id      number(10)      primary key,
  3    name    varchar2(20)
  4  );
SQL> create index xuxiang_table_name_idx
  2  on          xuxiang_table(name);
```

第十六章 权限的授予与收回

权限：

1）系统权限

DBA: 创建用户 -----> 超级用户
删除用户
备份数据
user: 创建各种对象 -----> 普通用户
Schema: 用户及用户所创建对象的一个集合

2）数据（对象）权限

创建用户：create user 用户名 identified by 密码; ---> 此命令只能针对 DBA 用户；
创建完用户后，还不能对此用户进行交互；
必须要先对用户进行分配权限；

权限授予：grant [权限] to 用户名;

权限: create table,
create sequence,
create view,
create session,
create procedure

grant connect,resource,create view to 用户名
resource： create table,

create sequence,
create session,
connect: create procedure,

修改用户：alter user 用户名 identified by 用户名; ---> 此命令每个用户都可以任意修改

限制用户：grant 功能 on 表名 to 用户名; ---> 当前用户名表的 select 权限给指定用户名

功能表示：select/delete/update/insert....;

grant 功能 on 列名 to 用户名/public with grant option;

---> with grant option ---> 允许权限传递 public ---> 表示所有公共的

收回权限：revoke 功能 on 表名 from 用户名;

同义词的创建：create synonym 同义词名称 for 表名;

```
# su - oracle
sqlplus '/as sysdba'
SQL> show user
SQL> create user ajd0806
  2  identified by ajd0806;
SQL> grant    connect,resource,create view    //授予权限
  2  to          ajd0806;
SQL> conn ajd0806/ajd0806
SQL> create table xuxiang_table(
  2  id       number(10)     primary key,
  3  name     varchar2(20)
  4  );
SQL> insert into xuxiang_table
  2  values (1,'xuxiang');
SQL> grant    select on xuxiang_table to jd0806;
SQL> conn jd0806/jd0806
SQL> select    *
  2  from     ajd0806.xuxiang_table;
SQL> revoke  select                          //收回权限
  2  on                xuxiang_table
  3  from      jd0806;
SQL> create    synonym xx                    //为对象取别名
  2  for      xuxiang_emp1000;
```

第十七章 Summary

经典练习题：录像租界系统

----------Title_table----------
```
create table xuxiang_title (
id number(10)
constraint xuxiang_title_pk primary key,
title varchar2(60)
constraint xuxiang_title_title_nn not null,
description varchar2(400)
constraint xuxiang_title_description_nn not null,
rating varchar2(4)
constraint xuxiang_title_rating_ck check(rating in( 'G','PG','R','NC17','NR')),
category varchar2(20) default 'DEFAULT'
constraint xuxiang_title_category_ck check(category
in('DRAMA','COMEDY','ACTION','CHILD','SCIFI','DOCUMENTARY')),
release_date date,
price number(10,2)
constraint xuxiang_title_price_nn not null);
```
注意:对于 rating 来说,要求指定值,所以使用 check,因为 check 的意思就是指定其值;

而对于 category 来说,也是使用 check,但是 default 来说,是指定默认值;

----------Member_table----------
```
create table xuxiang_member(
id number(10)
constraint xuxiang_member_id_pk primary key,
last_name varchar2(25)
constraint xuxiang_member_last_name_nn not null,
first_name varchar2(25) ,
address varchar2(100) ,
city varchar2(30) ,
phone varchar2(15) ,
join_date date default sysdate
constraint xuxiang_member_join_date_nn not null);
```
注意:对于 join_date 来说,要求指定系统当前日期,所以指定默认值是 sysdate 函数,指定其值;

----------Reservation_table----------
```
create table xuxiang_reservation(
res_date date,
title_id number(10)
constraint xuxiang_reservation_title_id_fk references xuxiang_title(id),
member_id number(10)
constraint xuxiang_reservation_member_id_fk references xuxiang_member(id),
constraint xuxiang_reservation_res_date_pk primary key(res_date,title_id,member_id));
```
注意:此表要求联合主键,不要急着给 res_date 指定 primary key,因为指定联合主键的话,要求要用表方式进行约束;

----------Title_copy_table----------
create table xuxiang_title_copy(
id number(10),
title_id number(10)
constraint xuxiang_title_copy_title_id_fk references xuxiang_title(id),
status varchar2(15)
constraint xuxiang_title_copy_status_ck check(status
in('AVAILABLE','DESTROYED','RENTED','RESERVED')),
constraint xuxiang_title_copy_id_pk primary key(id,title_id));
注意:在此在强调一下 check 的用法,check(要指定的字段 in('值 1','值 2','...','值 n'));

----------Rental_table----------
create table xuxiang_rental(
book_date date default sysdate,
member_id number(10)
constraint xuxiang_rental_member_id_fk references xuxiang_member(id),
copy_id number(10),
act_ret_date date,
exp_ret_date date default sysdate+2,
constraint xuxiang_rental_xuxiang_copy_id_fk foreign key(copy_id,title_id) references
xuxiang_title_copy(id,title_id),
title_id number(10)
constraint xuxiang_rental_title_id_fk references xuxiang_title(id),
constraint xuxiang_rental_id_pk primary key(book_date,copy_id,title_id));
注意:此表最为最麻烦,因为在此要创建联合外键与联合主键,要注意的是要用表级别约束;
---
drop table xuxiang_title cascade constraints;
drop table xuxiang_member cascade constraints;
drop table xuxiang_title_copy cascade constraints;
drop table xuxiang_rental cascade constraints;
drop table xuxiang_reservation cascade constraints;
注意:此些内容,是为了不占用其他更多的空间,使用完后就将其删除掉;

注意 drop 后面的 cascade constraints，这个是用于删除联合主键的。

:::::::::::::: 精点课上实例::::::::::::

==========

s_item->项目表 s_customer->客户表 s_product->产品表 s_emp->员工表 s_dept->部门表 s_ord->定单表

==========

select to_char(sysdate ,'yyyy-mm-dd hh24:mi:ss') from dual;
select to_char(trunc(sysdate) ,'yyyy-mm-dd hh24:mi:ss') from dual
---
Date --->to_char --->Char
Date <---to_date <---Char

Num --->to_char --->Char
Num <---to_number <---Char

时间:

YY 机制:世纪与当前时间世纪同值;以系统当前时间来进行计算日期

RR 机制:计算前五十年,和后五十年的世纪值

```
create table a(start_date date);
insert into a values(to_date('05-08-08',rr-mm-dd') ;
select to_char(n,'yyyy') from dual;
select last_name,to_char(start_date,'yyyy-mm-dd hh24:mi:ss) from s_emp;
select id,last_name,to_char(salary,'$99,999.99') from s_emp where id<5;
```

select to_char(1234,'999') from dual; -----> 会显示为#### 后面的 999 是代表转换字符后所要保留的位数

```
select to_number('1')+1 from dual;
select to_number(to_char(sysdate,'yyyy'))+1 from dual;
select
to_number(to_char(sysdate,'yyyy'))+to_number(to_char(sysdate,'mm'))+to_number(to_char(sysdate,'dd')) from dual;
select to_date('2008-08-08','yyyy-mm-dd') from dual;
select to_char(to_date('2008-08-10 23:59:38','yyyy-mm-dd hh24:mi:ss'),'mi-ss-dd yyyy-hh24-mm') from dual;
select last_name,name from s_emp,s_dept;
```

笛卡尔乖积:

```
select last_name,name from s_emp,s_dept where s_emp.dept_id=s_dept.id;
```

关联:

```
select s_dept.name,s_region.name from s_dept,s_region where s_dept.region_id = s_region.id;
select d.name,r.name from s_dept d,s_region r where d.region_id=r.id;
```

查看当使用的用户:show user

改变当前的用户:conn 用户名/密码 ex:conn scott/tiger

```
select grade from salgrade     where 5000 >= losal and 5000 <=hisal;
                                                                    where 5000 between losal and hisal;
select e.sal,g.grade,g.losal,g.hisal from emp e, salgrade g where e.sal between g.losal and g.hisal;
```

xuxiang_emp 表名

ID	NAME	DEPT_ID
1	zs	1
2	ls	

xuxiang_dept 表名

ID	NAME
1	d1
2	d2

查询所有员工名字以及对应部门名字

```
==========
select e.name , d.name from xuxiang_emp e,xuxiang_dept d where e.dept_id=d.id;
NAME                 NAME
-------- -------------
zs                   d1
==========
select e.name , d.name from xuxiang_emp e,xuxiang_dept d where e.dept_id=d.id(+);
NAME                 NAME
-------- --------------
zs                   d1
ls
==========
```

查询所有部门名字以及对应员工名字

```
==========
select d.name , e.name from xuxiang_emp e,xuxiang_dept d where e.dept_id(+)=d.id;
NAME                 NAME
-------  -------------
d1                   zs
d2
```

(+)的作用是:左边加右外联接,右边加左外联接,加号应该放在缺记录的一边.

在此之前,假想 s_emp 是两张表,

主管理者没有所对应的被管理人员,所以如果不加外连接的话,就会不显示主管理者的信息内容,因为条件中是查找有被

管理人员的人.

```
select worker.last_name,manager.last_name from s_emp worker,s_emp manager where
worker.manager_id=manager.id(+);
---------------------------------------------------------------------------------------------------------
select u.name,r.name from s_dir u,s_dir r where u.parent_id=r.id(+);
---------------------------------------------------------------------------------------------------------
```

&n 可以单行进行给值,下面就是一个实用的例子,

```
SQL> &n
SP2-0042: unknown command "&n" - rest of line ignored.
SQL> select ;
select
*
ERROR at line 1:
ORA-00936: missing expression
SQL> 1 &n
SQL> l
1* &n
SQL> /
Enter value for n: select id from s_emp where id=1
old 1: &n
new 1: select id from s_emp where id=1
ID
----------
```

```
1
SQL> desc s_emp
Name                          Null?    Type
----------------- ---------- ------------------
ID                            NOT NULL NUMBER(7)
LAST_NAME                     NOT NULL VARCHAR2(25)
FIRST_NAME                             VARCHAR2(25)
USERID                                 VARCHAR2(8)
START_DATE                             DATE
COMMENTS                               VARCHAR2(255)
MANAGER_ID                             NUMBER(7)
TITLE                                  VARCHAR2(25)
DEPT_ID                                NUMBER(7)
SALARY                                 NUMBER(11,2)
COMMISSION_PCT                         NUMBER(4,2)
SQL> select to_char(start_date-1,'d') from s_emp
SQL> select to_char(start_date-1,'d') start_date from s_emp;
SQL> column start_date format a20;
SQL> /
START_DATE
--------------------
5
4
1
6
7
5
1
6
7
3
1
11 rows selected.
SQL> @xuxiang id last_name
old 1: select &1,&2
new 1: select id,last_name
ID         LAST_NAME
---------- -------------------------
1          Velasquez
SQL> !clear
SQL> column start_date clear
SQL> select upper(last_name)||
2 ' earns'|| to_char(salary,'fm$99,999')||' monthly but wants '||to_char(salary*3,'fm$99,999')||' .'
"Dream Salaries" from s_emp;
Dream Salaries
---------------------------------------------------------------
VELASQUEZ earns$2,500 monthly but wants $7,500 .
NGAO earns$1,450 monthly but wants $4,350 .
NAGAYAMA earns$1,400 monthly but wants $4,200 .
QUICK-TO-SEE earns$1,450 monthly but wants $4,350 .
```

```
ROPEBURN earns$1,550 monthly but wants $4,650 .
URGUHART earns$1,200 monthly but wants $3,600 .
MENCHU earns$1,250 monthly but wants $3,750 .
BIRI earns$1,100 monthly but wants $3,300 .
CATCHPOLE earns$1,300 monthly but wants $3,900 .
HAVEL earns$1,307 monthly but wants $3,921 .
MAGEE earns$1,400 monthly but wants $4,200 .
11 rows selected.
SQL> select e.last_name,e.dept_id,d.name from s_emp e,s_dept d where e.dept_id = d.id;
LAST_NAME       DEPT_ID     NAME
--------------- ----------  ----------------------
Velasquez       50                  Administration
Ngao                    41                  Operations
Nagayama                31                  Sales
Quick-To-See 10                     Finance
Ropeburn                50                  Administration
Urguhart                41                  Operations
Menchu                  42                  Operations
Biri                    43                  Operations
Catchpole       44                  Operations
Havel                   45                  Operations
Magee                           31                  Sales
11 rows selected.
SQL> select e.last_name,d.name,r.name from s_emp e,s_dept d,s_region r where e.dept_id = d.id
and r.id = d.region_id and
e.commission_pct is not null;
LAST_NAME NAME                NAME
------------------        -------------------------
Magee                               Sales
North   America
Giljum                              Sales
South   America
Sedeghi                             Sales
Africa/Middle East

LAST_NAME NAME                NAME
------------------                  -------------------------
Nguyen                              Sales
Asia
Dumas                               Sales
Europe
SQL> select s_emp.last_name,s_dept.name from s_emp,s_dept where s_emp.dept_id = s_dept.id
and s_emp.last_name = 'Smith';
LAST_NAME                   NAME
---------------- -------------------------
Smith                       Operations
SQL> select s_product.name,
s_product.id,
s_item.quantity "ORDERED"
from s_product,s_item
```

```
where s_product.id=s_item.product_id and s_item.ord_id = 101;
NAME                                                    ID
        ORDERED
-------------------------------     -------- ----------
Grand Prix Bicycle Tires                30421         15
Pro Curling Bar                                 40422         30
Prostar 10 Pound Weight                 41010         20
Prostar 100 Pound Weight                41100         35
Major League Baseball                           50169         40
Griffey Glove                                   50417         27
Cabrera Bat                                             50530         50
7 rows selected.
SQL> select s_customer.id,
s_emp.last_name
from s_emp,s_customer
where s_emp.id = s_customer.sales_rep_id
order by s_emp.last_name;
ID              LAST_NAME
------  -------------------------
205     Dumas
206     Dumas
211     Dumas
215     Dumas
208     Dumas
201     Giljum
210     Giljum
204     Magee
209     Magee
214     Magee
213     Magee
11 rows selected.
SQL> select s_customer.id "Customer ID",
2 s_customer.name "Customer Name",
3 s_ord.id "Order ID"
4 from s_customer,s_ord
5 where s_customer.id = s_ord.customer_id(+);
Customer ID  Customer Name                  Order ID
-------------    ------------------------  ------------
201                     Unisports                               97
202                     OJ Atheletics           98
203                     Delhi Sports            99
204                     Womansport                      100
204                     Womansport                      111
205                     Kam's Sporting Goods    101
206                     Sportique                       102
207                     Sweet Rock Sports
208                     Muench Sports                   103
208                     Muench Sports                   104
209                     Beisbol Si!                     105
11 rows selected.
```

```
SQL> select w.last_name EMP_NAME,w.id EMP_ID,
m.last_name MGR_NAME,m.id MGR_ID
from s_emp w,s_emp m
where w.manager_id = m.id;
EMP_NAME          EMP_ID      MGR_NAME          MGR_ID
---------------   ----------  -------------     -----------
Ngao              2           Velasquez         1
Nagayama          3           Velasquez         1
Quick-To-See 4                Velasquez 1
Ropeburn          5           Velasquez         1
Urguhart          6           Ngao              2
Menchu            7           Ngao              2
Biri              8           Ngao              2
Catchpole 9                   Ngao              2
Havel             10          Ngao              2
Magee             11          Nagayama          3
Giljum            12          Nagayama          3
11 rows selected.
```

----------精点例题----------

```
SQL> select w.last_name , w.id , m.last_name , m.id from s_emp w,s_emp m where w.manager_id
= m.id(+);
LAST_NAME          ID          LAST_NAME          ID
----------------- -----    --------------        ------
Ropeburn          5           Velasquez         1
Quick-To-See      4           Velasquez         1
Nagayama          3           Velasquez         1
Ngao              2           Velasquez         1
Havel             10          Ngao              2
Catchpole         9           Ngao              2
Biri              8           Ngao              2
Menchu            7           Ngao              2
Urguhart          6           Ngao              2
Dumas             15          Nagayama          3
Nguyen            14          Nagayama          3
11rows selected.
SQL> select c.name customer,p.id product_id,i.quantity
2 from s_customer c,s_product p,s_item i,s_ord o
3 where c.id = o.customer_id
4 and o.id = i.ord_id
5 and i.product_id = p.id
6 and o.total > 100000;
CUSTOMER          PRODUCT_ID        QUANTITY
-----------------  --------------   -------------
Womansport        10011             500
Womansport        10013             400
Womansport        10021             500
Womansport        30326             600
Womansport        41010             250
Womansport        30433             450
```

Womansport 10023 400
Kuhn's Sports 20106 50
Kuhn's Sports 20201 130
Kuhn's Sports 30421 55
Kuhn's Sports 30321 75
11 rows selected.

第五章 function
SQL> select max(total) "Highest",
2 min(total) "Lowest"
3 from s_ord;
Highest Lowest
---------- ----------
1020935 377
SQL> select title job,max(salary) maximum,min(salary) minimum from s_emp group by title;
JOB MAXIMUM MINIMUM
---------- ---------- ----------
President 2500 2500
Sales Representative 1525 1400
Stock Clerk 1400 750
VP,Administration 1550 1550
VP, Finance 1450 1450
VP, Operations 1450 1450
VP, Sales 1400 1400
Warehouse Manager 1307 1100
8 rows selected.

数据字典表 user_constraints:
SQL> select constraint_name,constraint_type from user_constraints where table_name =
'XUXIANG_NN_TEST';
CONSTRAINT_NAME C
------------------------------ -
SYS_C0013231 C
XUXIANG_NN_TEST_NAME_NN C

创建唯一约束表:
---
create table xuxiang_uk_test(
id number(10) unique,
name varchar2(20),
constraint xuxiang_uk_test_name_uk
unique(name));

创建主键表:

列级别创建:
SQL> create table xuxiang_pk_test(
2 id number(10) primary key
3);
Table created.

表级别创建:

```
SQL> create table xuxiang_pk_test2(
2 id number(10) ,name varchar2(20),
3 primary key(id)
4 );
```

创建联合主键:

```
create table xuxiang_pk(
id number(10),
name varchar2(20),
constraint xuxiang_pk_id_name_pk
primary key(id,name)
);
```

创建外键表:

列级别创建:

```
create table xuxiang_fk_test(
id number(10),
parent_id number(10)
references xuxiang_pk_test(id)
);
```

创建联合外键:

```
create table xuxiang_fk_test2(
id number(10),
name varchar2(20),
foreign key(id,name)
references xuxiang_pk(id,name)
);
```

级联删除:

check:担心误输入

列级别:

```
create table xuxiang_ck_test(
id number(10) primary key ,
name varchar2(20) unique,
gendar char(10)
check(gendar in ('f','m'))
);
create table xuxiang_ck_test(
id number(10) primary key ,
name varchar2(20) unique,
gendar char(10) ,
check(gendar in ('f','m'))
);
```

创建关联表:

被外键所关联的表:

```
SQL> create table xuxiang_dept(
```

2 id number(10) primary key
3);
外键相关联表:
SQL> create table xuxiang_emp(
2 id number(10) primary key,
3 name varchar2(20) ,
4 dept_id number(10),
5 constraint xuxiang_emp_dept_id_fk foreign key(dept_id) references xuxiang_dept(id)
5);
第 219 个 ppt 的实意图去创建表:
---
create table s_dept(
id number(7) constraint s_dept_id_pk primary key,
name varchar2(25) constraint s_dept_name_nn not null,
region_id number(7) constraint s_dept_region_id_fk references s_region(id),
constraint s_dept_name_region_id_uk
unique(name,region_id
);
SQL> create table xuxiang_item1(
id number(10) ,
price number(10,2),
quantity number(10) ,
order_id number(10) references s_order(id),
primary key(id,order_id)
);
create table xuxiang_item(
id number(10) ,
price number(10,2),
quantity number(10) ,
order_id number(10) connstraint s_order_id_fk references s_order(id),
constraint s_item_name_order_id_uk primary key(id,order_id)
);
create table xuxiang_item(
id number(10),
price number(10,2),
quantity number(10),
order_id number(10) constraint xuxiang_item_order_id_fk references s_order(id),
列级别定义外键:
---
create table xuxiang_item(
id number(10),
price number(10,2),
quantity number(10),
order_id number(10),
constraint xuxiang_item_order_id_fk foreign key(order_id) references s_order(id)
);
复制表:只能拷贝 not null 约束,主键和外键非空唯一都不能被拷贝;
SQL>create table xuxiang_emp1000

```
2 as
3 select *
4* from s_emp
```

测试数据字典表的使用:

```
SQL>select table_name from user_tables
2 where table_name like 'XUXIANG%';
```

查看对象名称

```
SQL> select distinct object_type from user_objects;
```

输出结果:

```
OBJECT_TYPE
-------------------
INDEX
SEQUENCE
TABLE
```

查看约束名称与约束类型:

```
SQL> select constraint_name,constraint_type from user_constraints where
table_name='XUXIANG_EMP1000';
```

输出结果

```
CONSTRAINT_NAME                C
-----------------------        -----
SYS_C0013976                   C
SYS_C0013977                   C
SQL> select user_const.constraint_name ,
2 user_cons.constraint_name columns,
3 user_const.table_name constraints_table,
4 user_const.constraint_type constraints_type,
5 user_cons.table_name
6 from user_constraints user_const,user_cons_columns user_cons
7 where user_const.table_name='TONY_EMP1000' and
8 user_cons.table_name = 'TOMY_EMP1000';
SQL> select c.constraint_name,
2 c.constraint_type,
3 u.column_name
4 from user_constraints c,
5 user_cons_columns u
6 where c.constraint_name=u.constraint_name
7 and c.table_name='TON_EMP1000';
```

第十一章内容

```
inset into s_emp(id,first_name,last_name,userid,salary,start_date)
values(26,'Donna','Smith',USER,null,sysdate);
```

USER:是一个函数,返回值是字符型,返回当前的用户名给所对应的字段附值

一次插入多条记录(借用子查询)

```
SQL> insert into xuxiang_emp2000
2 select * from s_emp
3 where id between 1 and 10;
```

10 rows created.

更新操作的几个实例:

```
update xuxiang_emp1000 set salary=salary+200;
update xuxiang_emp1000 set salary = salary-200 where id not in (1,2);
update xuxiang_emp1000 set name = null where id = 3;
```

事务的回滚与提交

```
create table xuxiang_tran(id number(10) primary key,name varchar2(20));
insert into xuxiang_tran (id,name) values(1,'wanglixian');
savepoint a;
update xuxiang_tran set name='wanglixian001' where id=1;
insert into xuxiang_tran (id,name) values(2,'wanglixian2');
insert into xuxiang_tran (id,name) values(3,'wanglixian3');
select * from xuxiang_tran;
rollback to a;
select * from xuxiang_tran;
commit;
select * from xuxiang_tran;
```

第十二章:alter 的操作

Add 的操作:

```
SQL> alter table xuxiang_tran add (birthday date, tel varchar2(20));
SQL> desc xuxiang_tran;
Name          Null?                    Type
-------- ------------      ------------------
ID                        NOT NULL            NUMBER(10)
NAME                                                VARCHAR2(20)
BIRTHDAY                                      DATE
TEL
```

Modify 的操作:

将 tel 的 varchar2 改变成 number 类型

```
SQL> alter table xuxiang_tran modify(tel number(11));
```

将 name 加上非空约束

```
SQL> alter table xuxiang_tran modify(name not null);
SQL> desc xuxiang_tran
Name          Null?                    Type
-------          ------------      -----------------
ID                        NOT NULL            NUMBER(10)
NAME                  NOT NULL            VARCHAR2(20)
BIRTHDAY                                      DATE
TEL NUMBER(11)
```

Drop 的操作:

```
SQL> alter table xuxiang_tran drop column name;
SQL> desc xuxiang_tran
Name          Null?                    Type
----------       ------------      ---------------
ID                        NOT NULL            NUMBER(10)
```

```
BIRTHDAY                                  DATE
TEL                                       NUMBER(11)
```

Rename 的操作:

```
SQL> alter table xuxiang_tran rename column birthday to bir;
SQL> desc xuxiang_tran
Name          Null?                Type
-------       ------------      ---------------
ID                    NOT NULL          NUMBER(10)
BIR                                     DATE
TEL                                     NUMBER(11)
```

给 bri 加上唯一约束:

```
SQL> alter table xuxiang_tran add constraint xuxiang_tran_bir_uk unique(bir);
```

通过数据字典表来查看唯一约束是否增加:

```
SQL> select constraint_name,constraint_type from user_constraints where
table_name='XUXIANG_TRAN';
CONSTRAINT_NAME              C
----------------------       -------
SYS_C0014741                 P
XUXIANG_TRAN_BIR_UK                 U
```

删除掉约束:

```
SQL> alter table xuxiang_tran drop constraint xuxiang_tran_bir_uk;
SQL> select constraint_name,constraint_type from user_constraints where
table_name='XUXIANG_TRAN';
CONSTRAINT_NAME              C
----------------------       -------
SYS_C0014741                 P
```

第十三章

创建 sequence

```
SQL> create sequence xuxiang_seq1;
SQL> column sequence_name format a10
SQL> select * from user_sequences where sequence_name='XUXIANG_SEQ1';
SEQUENCE_N MIN_VALUE MAX_VALUE INCREMENT_BY C O CACHE_SIZE
LAST_NUMBER
---------- ---------- ---------- ------------ - - ---------- -----------
XUXIANG_SEQ1 1 1.0000E+27 1 N N 20 1
SQL> select xuxiang_seq1.nextval from dual;
NEXTVAL
----------
1
SQL> insert into xuxiang_first values(xuxiang_seq1.nextval,'xuxiang00'||xuxiang_seq1.currval);
-------------------------------------------------------------------------------------------
```

第十四章:

创建视图:

```
SQL> create of replace view xuxiang_v
2 as
```

```
3 select id,last_name,salary from s_emp
4 where id in(1,3,5);
**********复制表**********
SQL> create table xuxiang_emp1000 as select * from tony_emp1000;
```

给视图字段修改别名:

```
SQL> create or replace view xuxiang_v3
2 as
3 select id,last_name name,salary sal
4 from s_emp;
View created.
SQL> desc xuxiang_v3;
Name          Null?          Type
---------     ----------     ----------
ID                    NOT NULL          NUMBER(7)
NAME                  NOT NULL          VARCHAR2(25)
SAL NUMBER(11,2)
SQL> create or replace view xuxiang_v4 (id,name,sal)
2 as
3 select id,last_name,salary
4 from s_emp;
View created.
SQL> desc xuxiang_v4;
Name          Null?          Type
-------  ------------      -----------------
ID                    NOT NULL          NUMBER(7)
NAME                  NOT NULL          VARCHAR2(25)
SAL
create or replace view xuxiang_v4 (id,name,sal)
as
select id,last_name,salary
from xuxiang_emp1000
```

with read only;---------->只读,只能进行查询操作

Orcal SQL 笔记于 2008 年 7 月 25 日全部完成！！！

JDBC 学习笔记

==========

主讲：刘涛

第一天：

数据库：Oracle(Oracle),SQL Server(微软),MySQL(Sun),DB2(IBM),Sybase,.....
DB -----> RDBMS

第一章 JDBC Overview(JDBC 综述)

1、JDBC 的概念：

ODBC：Open DataBase Connectivity，开放数据库互联，微软公司提供的，是 C 语言与数据库交互的一种方式。

JDBC：Java DataBase Connectivity，java 数据库互联；

java 程序和数据库交互的“标准接口”，1996 年由 SUN 公司制定，JDBC TM(上标为 TM:是 SUN 公司注册的商标,TradMark)。

API：Application Programming Interface(应用程序接口)，别人已经写好的关于类和接口的一些集合。

JDBC API：Java 程序语言访问数据库的标准接口。

2、JDBC 可以做什么？

1）连接数据库；

2）发送 SQL 语句；

3）处理结果集；

3、JDBC API 的分类：

1）应用程序开发接口：应用程序开发人员使用

2）驱动程序开发接口：驱动程序开发人员使用

4、JDBC 驱动的分类：

1）JDBC－ODBC 桥加上 ODBC

将 JDBC 调用转换为 ODBC 调用

application--->jdbc-odbc 桥--->odbc--->db

性能低，现在已很少使用。

2）本地 API 部分用 java 写的驱动

将 JDBC 调用转换为 RDBMS 客户端程序调用

application--->jdbc driver(type2)--->client library--->db

需要在安装数据库的客户端程序。

3）JDBC 网络纯 java 驱动

将 JDBC 调用转换为与 RDBMS 无关的网络协议，然后由相应的服务器进行解析并调用 DB

application--->jdbc driver(type3)--->server--->db

4）本地协议纯 java 驱动(*)

将 JCBC 调用直接转换为 RDBMS 的调用

application--->jdbc driver(type4)--->db

5、JDBC 编程常用的包：

java.sql--->java 2 平台下 JDBC 的主要功能，标准版(J2SE)

javax.sql--->java 2 平台下 JDBC 增强功能，企业版(J2EE)

6、URL：

1）URL--->Uniform Resource Locator(统一资源定位器)，如：http://www.google.com/

2）我们使用 URL 来确定一个数据库(正确的 Driver,正确的主机,正确的协议，正确的用户名和密码)；

3）数据库 URL 的语法：jdbc:子协议:子名字

例如：Oracle URL--->jdbc:oracle:thin:@192.168.1.250:1521:briupdb

1.oracle:

driver: oracle.jdbc.driver.OracleDriver
URL: jdbc:oracle:thin:@192.168.0.200:1521:briupdb

2.pointbase:

driver: com.pointbase.jdbc.jdbcUniversalDriver
URL: jdbc:pointbase:server://localhost:9092/ec_port

3.mysql:

driver: com.mysql.jdbc.Driver
URL: jdbc:mysql://localhost:3306/briupdb

中文问题：
jdbc:mysql://localhost:3306/briupdb?useUnicode=true&characterEncoding=gb2312

7、javax.sql 包 JDBC2.0 的增强功能

1) 数据源接口；
2) 连接池；
3) 分布式事务；
4) 行集；

第二章 Using JDBC(JDBC 的使用)

1、JDBC 编程的六大步骤:

1）注册驱动；

a）通过类加载器

Class.forName(driverName);

b）直接实例化驱动

Driver driver = new DriverName();
DriverManager.registerDriver(driver);

c）通过 jdbc.drivers

java -Djdbc.drivers=DriverName[:DriverName]

如：右键 ---> Run As ---> Run... ---> Java Application ---> new ---> Arguments ---> VM arguments ---> -Djdbc .drivers=oracle.jdbc.driver.OracleDriver --->Run

2）建立连接；

a）DriverManager.getConnection(url,user,password);

b）drier.connect(url, properties);

注意：一般配合着注册驱动的第二步一起使用。

3）创建 statement;

Statement:

Connection.createStatement();

PreparedStatement:

Connection.prepareStetement(String sql);

CallableStatement:

Connection.prepareCall(String sql);

statement：用来发送 SQL 语句

statement：异构的 statement，用于执行静态 SQL 语句；

PreparedStatement：同构的 statement，表示预编译的 SQL 语句的对象，然后可以使用此对象高效地多次执行该语句；

CallableStatement：同构的 statement，用于执行 SQL 存储过程的接口。

注意： 1.调用存储进程或函数时用 CallableStatement；

2.sql 语句的结构是否相同。

4）执行 sql 语句；

ResultSet stm.executeQuery(...):

返回值为 ResultSet 类型，用来表示查询的结果集；

int stm.executeUpdate(...):

返回值为 int 类型，用来表示执行的 sql 语句影响的记录数；如果是 select，则返回 0；

boolean stm.execute(...):

返回值为 boolean 类型，用来表示执行的 sql 语句是否有结果集返回；如果有为 true，否则为 false;

5）处理结果集；

位置标识法；

名字标识法；

6）释放资源；

将释放资源的操作写在 finally 语句块中；

2、范例程序：

sqlplus jd0806/jd0806

首先建一张表：
```
create table xuxiang_student(
name   varchar2(25),
age            number(3)
);
```

Project: JDBC
Package: com.briup.basic
Class: --FirstJdbc

下载驱动包：192.168.1.250
jd0806/jdbc/lib/ojdbc14.jar

导入 JDBC 驱动包：建立 JDBC 驱动包和项目连接的方式
项目名--->右键--->Properties--->Java Build Path--->Libraries
--->Add External JARs...--->导入添加 JDBC 驱动包的路径

查询 Eclipse 的快捷键：ctrl+shift+l

PL/SQL:
```
create table day_x(1...31)
(
        id
        name
        ...
)
```

Oracle 函数：count(),lower(),upper(),round(),...

3、作业：

建立一张表，用 JDBC 向表中插入 10 条记录，修改 id 为 1 的记录，删除 id 为 2 的记录，最后将结果打印在控制台上（要求和数据库连接相关的、易变的信息不能硬编码在程序中）。

```
create table xuxiang_user(
id             number(7)          primary key,
name   varchar2(25),
age            number(3)
);
```

第二天：

com.briup.common
--JdbcConnectionFactory
--JdbcTemplate(CRUD)

第四章 JDBC Advanced Topics(JDBC 高级特性)

1、JDBC 编程中的异常用 SQLException 表示，是 checked exception,必须对其进行处理。

常用方法：

getMessage(): 获取错误信息

printStackTrace(): 打印堆栈信息

getErrorCode(): 获取错误代码号，如：ORA-00923

2、SQLWarnings: jdbc 警告信息，无须处理；

获取 jdbc 警告信息：

Connection.getWarnings();
Statement.getWarnings();
ResultSet.getWarnings();

3、metadata: 元数据，用来描述其它数据的数据；

DatabaseMetaData: 用来描述数据库的数据；

如：数据库的名字、数据库改版等

获取：DatabaseMetaData dbmd = Connection.getMetaData();

ResultSetMetaData: 用来描述结果集的数据；

如：结果集有多少列、每一列什么名字、什么类型等

获取：ResultSetMetaData rsmd = ResultSet.getMetaData();

4、Transaction: 事务

1）概念：把一组操作封装成一个原子，这组操作要么同时成功，要么同时失败(如银行的转帐)；

2）事务的 ACID 特性：

1.原子性(Atomicity)：事务所封装的一组操作不可分割；

2.一致性(Consistency)：事务执行前和事务执行后，数据所处的状态要保持一致；

3.隔离性(Isolation)：事务在执行时彼此间互不影响；

4.持久性(Durability)：事务一旦提交，状态将持久改变；

3）事务的提交：

1.自动提交：默认建立的连接会自动提交事务，每执行一个操作将提交一次；

2.手动提交事务：

```
con.setAutoCommit(false);
...
```

```
con.commit();
con.rallback();
```

4）多事务并发访问时可能出现的问题：

脏读(Dirty read)：读取了其它事务没有提交的数据；

不可重复读(Repeatable read)：在同一事务内部多次读取的数据内容不一致；

幻影读--->虚读(Phantom read)：在同一事务内部多次读取的记录数不一致；

5）隔离级别：

1.TRANSACTION_NONE：不支持事务，现已很少使用；

指示事务不受支持的常量。

2.TRANSACTION_READ_UNCOMMITTED：允许读取其它事务未提交的数据；

指示可以发生脏读、不可重复读和虚读的常量。

3.TRANSACTION_READ_COMMITTED：只能读取已经提交的数据，可解决脏读问题，是比较常用的；

指示防止发生脏读的常量；不可重复读和虚读有可能发生。

4.TRANSACTION_REPEATABLE_READ： 可解决不可重复读问题；

指示防止发生脏读和不可重复读的常量；虚读有可能发生。

5.TRANSACTION_SERIALIZABLE：可解决脏读、不可重复读、幻影读问题；

指示防止发生脏读、不可重复读和虚读的常量。

注意：1）隔离力度越大，性能就低一些。

2）仅 READ_COMMITTED 和 SERIALIZABLE 是有效的事务处理级。

设置隔离级别：

```
con.setTransactionIsolation(Connection.TRANSACTION_READ_COMMITTED);
```

默认隔离级别：TRANSACTION_READ_COMMITTED

	read uncommitted	read committed	repeatable	serializable
dirty read	n	y	y	y
repeatable read	n	n	y	y
phandom read	n	n	n	y

第三天：

第五章 JDBC2.0 Core Features(JDBC2.0 新特性)

1、JDBC2.0 对 ResultSet 功能的增强：

1）可滚动性：允许回滚，能够快速定位；

1.可以进行绝对定位、相对定位

2.可以向前滚动、向后滚动；

2）可更新性：通过修改结果集，可直接影响到数据库中的数据；

对 ResultSet 进行 update,delete,insert 操作，实际影响的是数据库中的数据；

2、JDBC2.0 对 Statement 功能的增强：允许批处理

3、JDBC2.0 对 SQL3 功能的增强：增加了高级数据类型

1）Blob：大的二进制流，如将电影存放到数据库中

2）Clob：大数据量的文本，如小说《红楼梦》

...

4、connection.createStatement(对别人的修改是否敏感，是否支持可更新性)

对别人的修改是否敏感：

TYPE_SCROLL SENSITIVE：敏感

TYPE_SCROLL INSENSITIVE：不敏感

是否支持可更新性：

CONCUR_READ_ONLY：不支持

CONCUR_UPDATABLE：支持

如：Statement stm = connection.createStatement(ResultSet.TYPE_SCROLL SENSITIVE,ResultSet.CONCUR_UPDATABLE)

com.briup.core

--ScrollTest

--UpdateTest

--StatementBatchTest

--PreparedStatementBatchTest

利用 PL/SQL 向表中插入 10 条记录：

```
begin
    for i in 1..10 loop
        insert into xuxiang_user
        values(i,'briup'||i,i*10);
    end loop;
    commit;
end;
/
```

jd0806/jdbc/code

--BlobTest.java

--face.gif

--table.sql

5、作业：

利用普通的 Statement 操作：

1）删除表中第二条记录；

利用 PL/SQL 向表中插入 20 条记录：

```
begin
        for i in 1..20 loop
                insert into xuxiang_user
                values(i,'briup'||i,i*10);
        end loop;
        commit;
end;
/
```

查询表中第二条记录：

```
select  id,name,age
from        (select id,name,age,rownum r from xuxiang_user)
where       r=2;
```

删除表中第二条记录：

```
delete from xuxiang_user where rowid=(
        select rid from(
                select rowid rid,rownum rnum
                from xuxiang_user)
        where rnum=2);
```

删除表中相同的记录并留一条：

```
delete from student a
where a.rowid!=(
        select max(b.rowid) from student b
        where a.name=b.name and a.age=b.age);
```

查询两个表中的(含空值)的记录：

```
select  e.last_name,d.name
from        s_emp e,s_dept d
where       e.dept_id(+)=d.id;
union
select  e.last_name,d.name
from        s_emp e,s_dept d
where       e.dept_id=d.id(+);
```

集合操作符：

union/union all/minus/intersect

2）取出表中第 11 到第 20 条记录；

法一：

```
select  id,name,age
```

```
from    (select id,name,age,rownum r from  xuxiang_user)
where   r between 11 and 20;

法二：
select  id,name,age
from    xuxiang_user
where   rownum<=20
minus
select  id,name,age
from    xuxiang_user
where   rownum<=10;
```

第四天

第六章 JDBC2.0 Standard Extensions

1、连接池：共享有限的数据库连接、提高系统性能

1）在应用启动时，可以将一定数量的连接放到连接池中；

2）能够从连接池中获取到连接；

3）能够将连接放回到连接池中；

4）在应用关闭时，可以将所有连接关闭；

产品描述 | 市场价格

Oracle 10g for NT /2000 标准版（5 user）	36540.00
Oracle 10g for Liunx 标准版 (5user)	50400.00
Oracle 10g for NT /2000 标准版 (支持一个 CPU)	41790.00
Oracle 10g for NT /2000 标准版 (支持多个 CPU)	102900.00
Oracle 10G 单 CPU 25 用户 企业版	15066*用户数
Oracle 10G 企业版 单 CPU 无限用户	753228.00

com.briup.extension

--PoolManager(4 个方法)

--DataSourceTest

2、DataSource: 数据源，用来创建 Connection 的工厂，可以认为是 DriverManager 的替代者，一般结合 JNDI 一起来使用

1）将程序转化为.exe 文件：exe4j,installAnywhere(收费)

2）ip: www.google.com
www.baidu.com

3） JDNI：Java Naming and Directory Interface(Java 命名目录接口)
服务器端：注册资源，将具体的资源同逻辑名字相绑定；
客户端：通过注册的逻辑名获取资源使用；

Tomact,Weblogic...: 服务器（不是计算机，而是一个软件）

ls -a

http://www.google.com
ftp://
file://

4） ORM：Object Relationship Mapping(对象关系映射)
ORM 工具：hibernate,ibatis...

com.briup.orm.simple
--Student
com.briup.orm.pojo
com.briup.orm.dao
--UserDao
com.briup.orm.dao.impl
--UserDaoImpl
com.briup.orm.test
--UserDaoTest

```
create table xuxiang_student(
id          number(7)          primary key,
name   varchar2(25),
age         number(3)
);
```

5）POJO：Plain Old Java Object(最古老的 Java 类)

6） DAO：Data Accessor Object
DAO 模式＝Data Accessor 模式 + Active Domain Object 模式
1. Data Accessor 模式：实现数据访问和业务逻辑的分离，使用接口进行降偶；
2. Active Domain Object 模式：实现业务数据的对象化封装；

3、Distributed Transaction：分布式事务，跨多个数据库的事务
需要通过事务管理器来帮我们管理事务，要使用两阶段提交协议（JTA）

4、RowSet：行集，当数据库驱动不支持结果集的可滚动性和可更新性时，我们可以借助行集来实现；

第七章 Row Set Implementation

略...

第五天：

第三章 OOAD and Database Design

1、ORM：将内存中的对象映射到关系型数据库中(对象关系映射)

面向对象的 java　　关系型数据库

类　　表
属性　　字段(列)
关系　　表
OID　　pk

```
class Student{          table sudent(
id                              id
name                    name
age                             age
bornDate                bornDate
}                               )
```

注意：并不是所有的属性都必须要映射成表中的字段，派生属性不需要映射
派生属性，即可以由其他属性计算出来的属性

```
one-to-one:
class Person{           table person(
...                             ...
Card card                       cardid
}                               )
class Card{             table card(
...                             ...
}                               )

insert into person(1,'zhangsan',10)
insert into person(2,'zhangsan',10)

one-to-many:
class Customer{         table customer(
...                             ...
```

List addresses
}
class Address{
...
Customer c
}

)
table address(
...
cid
)

many-to-many: 多对多关系在数据库中没有办法直接实现，需再引入一张关联表，内含关联属性

class Student{
...
List courses
}
class Course{
...
List students
}
class s_c{
...
grade
}

table student(
...
)
table course(
...
)
table student_course(
sid
cid
grade
)

注意：多对多映射要引入关联表，关联表中有属性时要引入关联类

2、 pk 分类：

1）自然主键：主键本身具有商业含义；

2）逻辑主键(代理主键)：主键本身没有商业含义；

3、 获取唯一 OID 的方法：

1)单行表：只有一个字段，用来存放唯一的 oid

每次从单行表中获取到唯一的 oid，然后将 oid 的值修改，多用户并发访问时，要进行同步控制，性能比较低；

2)高低值：oid 由高值和低值两部分组成

高值存放在单行表中，在会话开始时由应用程序取到低值，并由应用程序进行递增，然后添加到高值的后面，组成 oid;

com.briup.oid
--SingleRow
--HighLow

```
create table xuxiang_singleValue(
nextval                 number(3)
);
insert into xuxiang_singleValue values(100);
commit;
```

```
high:100
int low:1

int id = Integer.parserInt('100'+low);

1001
low++;
1002
...
for(int i=0;i<5;i++){
insert ...(id
}

create table xuxiang_highLow(
nextval          number(3)
);
insert into xuxiang_highLow values(100);
commit;
```

4、 有继承关系的类进行映射：

映射方式	是否支持多态	更性维护是否方便	出具(打印)报表是否方便
1）每个类映射成一张表	支持	方便	不方便
2）每个具体的子类映射成一张表	不支持	不方便	方便
3）整个类层次映射成一张表	支持	不方便	方便

多态：在同一张表中，能否存放不同类类型的数据信息；

第八章 JDBC and Application Architecture

1、分层好处：

1）将大的复杂的项目分解；

2）每一层至多和相邻的上下层交互，对每一层的改变最多影响两层；

3）支持重用；

2、两层架构：

1）单一的服务器，进行扩展比较困难；

2）耦合性较强，往往是牵一发而动全身；

3、三层架构：

1）表示层：用户接口，负责与用户进行交互，如检查用户输入数据，显示应用输出数据；

2）业务逻辑层：处理具体的业务，如网上订书下订单，既要给客户通知、又要给客

服通知；

3）数据层：与数据库进行交互；

C/S：Client/Server ---> 客户服务器架构（如 QQ）

B/S：Browser/Server ---> 浏览器服务器架构（如 163 邮箱），通过 AJAX 可以完成在浏览器中拖动窗口的功能

作业：ORM

```
Order                                   Orderline
table Order(                            Orderline(
id                                              id
name                                    ...
...                                             orderid
)                                               )

class Order{                            class Orderline{
List orderlines                 Order order
...                                             ...
}                                               }
```

级联（保存，删除，修改，查询）

```
create table xuxiang_order(
id              number(7)       primary key,
style           varchar2(50),
total           number
);

create table xuxiang_orderline(
id              number(7)       primary key,
product         varchar2(50),
price           number,
count           number,
oderid          number(7) references xuxiang_order(id)
);
```

Order.java, Orderline.java
OrderDaoImplimplements OrderDao
OrderDaoTest

JDBC 笔记于 2008 年 7 月 30 日全部完成！！！

Hibernate 学习笔记

==========

主讲：刘涛(Robin)

第一天：

第一章 Hibernate 入门

1、持久化：

狭义：将数据持久地保存(将处于临时状态的数据持久的保存到文件或数据库中)；

广义：对数据进行 CRUD 操作；

employee(
id
name
age
deptid
addressid
)

2、ORM：将内存中的对象映射到关系型数据库中；

面向对象的 java	关系型数据库
类	表
属性	字段(列)
关系	表
OID	PK

注意：派生属性不需要映射

派生属性，即可以由其他属性计算出来的属性

3、公用持久化模板的要求：

1）提供 API 供客户使用，执行 CRUD 操作；

2）提供一种语言访问数据库（面向对象的）；

如 HQL：Hibernate Query Language

3）允许客户制定表和类之间的映射关系，一般写在 xml 中；

4）提供一些优化机制，来提高性能，如：缓存、延迟加载等；

4、什么是 Hibernate?

1）中间件，介于应用程序和数据库之间的软件；

2）持久化工具，封装了和数据库访问的细节，可以让程序开发人员更加关注于业务逻辑的处理；

3）ORM 工具，将面向对象的 java 同关系型数据库进行映射，可以实现内存中的数据和数据库中数据的双向同步更新；

5、Hibernate 涉及的文件：

1）pojo：封装数据；

2）配置文件：hibernate.properties 或 hibernate.cfg.xml，用来配置和数据库连接相关的信息；

3）映射文件：xx.hbm.xml(hbm：hibernate mapping)，用来映射表和类之间的对应关系；

4）hibernate API：

1.核心接口：CRUD 操作，事务处理等；

2.回调接口：当特定时间触发时，回调客户端程序；

3.类型：hibernate 有自己定义的类型；

4.可扩展接口：可以以 hibernate 为基础进行扩展；

6、核心接口：

1）org.hibernate.cfg.Configuration：

1.对 hibernate 进行配置，根启动 hibernate;

2.创建 SessionFactory;

2）org.hibernate.SessionFactory:

1.重量级的(非常的占用资源)，不要频繁的创建，删除，一般一个数据库对应一个 SessionFactory；

2.线程安全的；

3.缓存 sql 语句；

如：student(
id
name
age
)

缓存的四条 sql 语句：

```
select id,name,age from student where id=?
delete from student where id=?
update student set name=?,age=? where id=?
insert into student values(?,?,?)
```

4.维护着 hibernate 的二级缓存（可选的）；

5.创建 Session；

3）org.hibernate.Session:

1.轻量级的，可以频繁的创建和删除，一般一个事务中的操作对应一个Session;

2.线程不安全的(多线程访问需要进行线程控制)；

3.维护着 hibernate 的一级缓存（必用的）；

4.可以执行 CRUD 操作；

```
session.save(...);
session.delete(...);
session.update(...);
session.get(...);
session.load(...);
```

4）org.hibernate.Transaction：

hibernate 中的事务控制，包括：jdbc 事务、JTA 事务、CORBA 事务；

5）org.hibernate.Query：

通过 HQL 执行查询操作； ---> "Hibernate Query Language"
session.createQuery(String queryString)

sql：select id,name,age from student

hql：from Student

6）org.hibernate.Criteria：

擅长于做动态查询

7、Hibernate 编程步骤：

1）创建 pojo 类；

2）创建配置文件（包含数据库连接的相关信息，以及连接的映射文件）；

3）创建映射文件；

4）调用 hibernate 的 API；

1.创建 Configuration;

2.创建 SessionFactory;

3.创建 Session;

4.启动事务，session.beginTransaction();

5.执行 CRUD 操作；

6.提交或回滚事务；

transaction.commit()，transaction.rollback()

```
create table xuxiang_student(
id                    number(7)          primary key,
```

```
name        varchar2(25),
age             number,
gendar      varchar2(10),
address     varchar2(50)
);
```

添加 Hibernate 驱动：项目名--->右键--->Properties--->Java Build Path--->Libraries--->Add Library...
--->MyEclipse libraries--->Hibernate 3.0 Core Libraries

```
src
    --com.briup.first
        --StudentTest
```

DTD，XML，Schema

第二章 对象/关系映射基础

1、映射文件：xx.hbm.xml

在同一个映射文件中，可以映射多个 pojo 类，但建议只映射一个；

2、常用标签介绍：

1）<class>：用来映射表

2）<id>：用来映射主键

3）name：类中 set/get 方法后的名字

4）column：表中的列名

5）type：hibernate 数据类型

6）<generator>：用来制定主键的生成方式

1. increment：由 hibernate 按照自增长方式产生唯一主键，注意在多线程访问时不要使用；

对应的 java 类：org.hibernate.id.IncrementGenerator

2. identity：由支持 identity 的数据库产生主键；

--org.hibernate.id.IdentityGenerator

```
id number primary key identity
```

3. sequence：由数据库 sequence 来产生唯一主键；

--org.hibernate.id.SequenceGenerator

```
create sequence xuxiang_sequence
start with 100;
```

4. hilo：由数据库按高低值算法产生唯一主键（主键高值存在于表中）；

--org.hibernate.id.TableHiLoGenerator

算法：id=hi*(max_lo+1)+lo

如下，则有：id=100*(100+1)+0=10100

```
create table xuxiang_hi(
nextval        number
);
insert into xuxiang_hi values(100);
commit;
```

5. seqhilo：由数据库按高低值算法产生唯一主键（主键高值存在于 sequence 中）；

--org.hibernate.id.SequenceHiLoGenerator

6. assign：手动指定主键值；

7）<property>：　　用来映射普通的列；

8）<discriminator>：　　识别器，整个类层次映射成一张表时使用；

9）<one-to-one>：　　用来映射一对一关系；

10）<many-to-one>：　　用来映射多对一关系；

11）<many-to-many>：　　用来映射多对多关系；

12）<composite-id>：　　用来映射联合主键；

com.briup.second.composite

```
create table xuxiang_student2(
id1                    number,
id2                    number,
name          varchar2(25),
age           number,
primary key(id1,id2)
);
```

13）<component>：　　用来映射组合关系；

com.briup.second.component

--User,Address

```
create table xuxiang_user(
id                     number primary key,
name          varchar2(25),
age                    number,
city          varchar2(20),
street        varchar2(50)
);
```

14）<subclass>：　　用来映射子类；

15）<formula>：　　计算表达式，映射派生属性时经常使用；

User

--Date bornDate

3、作业：建一个表 product，对其进行 CRUD 操作，要求主键由 sequence 生成，映射派生属

性 totalPrice;

```
create table xuxiang_product(
id                  number          primary key,
name        varchar2(25),
price       number,
count       number
);
```

第二天：

4、Hibernate 中的类型：

1）值类型：没有映射文件，不能直接映射到数据库中；

基本数据类型，对象类型(如 Address)

2）实体类型：有映射文件，可以直接映射到数据库中；

第三章 关联关系映射

1、一对一映射：

1）唯一外键关联；

表上：

table user	table address
id<PK>	id
name	city
...	...
addressid<FK>	

类上：

class User	class Address
id	id
name	city
...	...
Address address	User user

对应的映射文件：

User.hbm.xml

```
<many-to-one name="address" class="Address" column="addressid" unique="true" />
```

注意：由于<one-to-one>已被主键关联占用，为了与其区别，故引入了<many-to-one>!

Address.hbm.xml

```
<one-to-one name="user" class="User" />
```

com.briup.third.one2one.fk

```
create table xuxiang_address(
```

```
id              number          primary key,
city        varchar2(20),
street      varchar2(50)
);
create table xuxiang_user(
id                  number          primary key,
name            varchar2(25),
age                 number,
addressid       number references xuxiang_address(id)
);
```

2）主键关联；

表上：

```
table user                          table address
id<PK>                                  id<PK,FK>
name                                city
...                                     ...
```

类上：

```
class User                          class Address
id                                      id
name                                city
...                                     ...
Address address             User user
```

映射文件：

User.hbm.xml

```
    <one-to-one name="address" class="Address" />
```

Address.hbm.xml

```
    <one-to-one name="user" class="User" />
    <generator class="foreign">
        <param name="property">user</param>
    </generator>
```

com.briup.third.one2one.pk

```
create table xuxiang_user2(
id              number          primary key,
name        varchar2(25),
age             number
);
create table xuxiang_address2(
id              number      references xuxiang_user2(id),
city        varchar2(20),
street      varchar2(50),
primary key(id)
);
```

2、一对多映射：

1）单向：只能在类上体现，表上是不能体现的，故单向和双向的表是完全相同的；

表上：

table user　　table address

id<PK>　　id

name　　city

...　　...

addressid<FK>

类上：

class User　　class Address

id　　id

name　　city

...　　...

Address address

映射文件：

User.hbm.xml

<many-to-one name="address" class="Address" column="addressid" not-null="true"/>

Address.hbm.xml

不需要作任何配置；

com.briup.third.many2one.unidirect

```
create table xuxiang_address3(
id              number          primary key,
city        varchar2(20),
street      varchar2(50)
);
create table xuxiang_user3(
id                  number          primary key,
name            varchar2(25),
age                 number,
addressid       number references xuxiang_address3(id)
);
```

2）双向：

表上：

table user　　table address

id<PK>　　id

name　　city

...　　...

addressid<FK>

类上：

class User　　class Address

id　　id

name city
... ...
Address address List users

映射文件：
User.hbm.xml
```
<many-to-one name="address" class="Address"/>
```
Address.hbm.xml
```
<set name="users">
    <key column="addressid"/>
    <one-to-many class="User"/>
</set>
```

注意：

cascade：设置级联

sava-update：级联保存、更新

delete：级联删除

none：不级联，默认值

all：级联保存、更新、删除

inverse：在映射一对多关系时，一般将该属性设置为 true，表示表间的关联关系由一方设置，减少 update 语句，提高性能。

com.briup.third.many2one.bidirect

练习：创建表 customer，order，建立双向一对多关联关系，向数据库中插入 1 个客户、两个订单；

```
create table xuxiang_customer(
id          number      primary key,
name        varchar2(25),
age         number
);
create table xuxiang_order(
id          number      primary key,
name        varchar2(25),
price       number,
customerid  number references xuxiang_customer(id)
);
```

第三天：

3、多对多映射：

1）单向：

2）双向：

表上：

table user　　table user_address　　table address

id<PK>　　userid<PK,FK>

id<PK>

name　　addressid<PK,FK>　　city

age

street

...

...

类上：

class User　　class Address

id　　id

name　　city

age　　street

...　　...

Set addresses　　Set　　users

映射文件：

User.hbm.xml:

```
<set name="addresses" table="user_address">
    <key column="userid"/>
    <many-to-many column="addressid" class="Address"/>
</set>
```

Address.hbm.xml:

```
<set name="users" table="user_address" inverse="true">
    <key column="addressid"/>
    <many-to-many column="userid" class="User"/>
</set>
```

com.briup.third.many2many

```
create table xuxiang_user4(
id          number          primary key,
name        varchar2(25),
age         number
);
create table xuxiang_address4(
id          number          primary key,
city        varchar2(20),
street      varchar2(50)
);
create table xuxiang_user4_address4(
userid      number          references xuxiang_user4(id),
addressid   number          references xuxiang_address4(id),
primary key(userid,addressid)
);
```

第四章 操纵持久化对象

1、 hibernate对象的状态：

	是否同session关联	数据库中是否有对应记录
1）自由状态	n	n
2）持久化状态	y	y
3）游离状态	n	y

dynamic-update="true"：动态生成update语句

2、 session.get(...)：

1）数据库中无对应记录，返回null;

2）直接到数据库中去查；

3）使用代理类；

session.load(...)：

1）数据库中无对应记录，抛出ObjectNotFoundException;

2）先到缓存中去查，缓存中没有，再到数据库中去查；

3）不使用代理类；

代理类(ProxyClass)：

OID（Object id）---> PK

3、 脏检查：对于持久化状态的对象，hibernate会进行脏检查，自动将内存中的改变更新到数据库中；

4、 cascade：级联

com.briup.fourth.cascade

--Order，Orderline

```
create table xuxiang_order2(
id                  number          primary key,
name        varchar2(50),
style       varchar2(10)
);
create table xuxiang_orderline2(
id                  number          primary key,
product     varchar2(50),
price       number,
count       number,
orderidnumber       references xuxiang_order2(id)
);
insert into xuxiang_order2 values(100,'myorder','cash');
```

```
insert into xuxiang_order2 values(101,'myorder2','cash');
insert into xuxiang_orderline2 values(10,'bread',20,5,100);
insert into xuxiang_orderline2 values(11,'milk',3,10,100);
commit;
```

5、 批处理：引入 for 循环和 if 语句

第五章 事务控制

1、事务

1）事务的 ACID 特性：

1.原子性：事务所封装的一组操作不可分割，要么同时成功，要么同时失败

2.一致性：事务操作之前和事务操作以后，数据的完整性保持一致

3.隔离性：多个事务同时执行，但不会互相影响

4.持久性：事务执行成功，状态将持久保存

2）事务控制

```
Transaction tran = session.beginTransaction();
tran.commit();
tran.rollback();
session.setFlushMode(...)
        FlushMode.ALWAYS    ：执行 query 之前先 flush
        FlushMode.AUTO      ：默认值
        FlushMode.COMMIT    ：在 commit 前先 flush
        FlushMode.NEVER     ：session.flush()
```

注意：session.flush()的作用是将缓存中的数据传入数据库

3）隔离问题

1.脏读：一个事务读取了别的事务未提交的操作

2.不可重复读：在同一事务中多次读取的数据信息不同（本事务没做任何修改操作）

3.幻影读：在同一事务中多次读取的数据记录数不同（本事务没做任何插入操作）

4.第一类丢失更新：多个事务对同一资源进行修改，某一事务的撤销，导致其他事务的提交也被撤销

5.第二类丢失更新：多个事务对同一资源进行修改，某一事务的提交，导致其他事务的提交被覆盖

解决方案：设置隔离级别

注意：隔离级别越高，锁的力度越细，解决的并发问题越多，但性能越低

4）锁：

1.悲观锁

常见的锁模式：

LockMode.NONE : 表示先到缓存中去检索，如果缓存中没有，再到数据库中去检索

LockMode.READ : 表示直接到数据库中去检索

LockMode.UPGRADE: 表示到数据库中去检索，并锁定相应的记录

```
select id,name,face from student for update  //通过 for update 加锁
nowait;
```

2.乐观锁：借助于 version 实现的（表、pojo 类、配置文件中都添加一个 int 型的 version）;

2、练习：

自己建一张表，批量插入 100 条记录，每次插入 20 条，将 100 条记录分成 10 页，每页 10 条，从外界传入一个参数值，来决定取出第几页的数据？（如：传 2 表示取出数据库中第 11 到 20 条记录）

```
create table xuxiang_friend(
id                  number          primary key,
name            varchar2(25),
telephone       varchar2(50),
address         varchar2(50)
);

int num = Integer.parseInt(args[0]); // 应用程序参数
String hql = "from Friend";
Query query = session.createQuery(hql);
query.setFirstResult((num-1)*10);
query.setMaxResults(10);
```

第四天：

第六章 高级映射

1、映射集合

1)Set: 无序不可以重复

```
create table xuxiang_item(
id                  number          primary key,
name            varchar2(25)
);
```

```
create table xuxiang_image(
itemid          number              references xuxiang_item(id),
filename        varchar2(50),
primary key(itemid,filename)
);
```

com.briup.sixth.set

映射文件：

```
<set name="images" table="xuxiang_image" lazy="true">
        <key column="itemid"></key>
        <element type="String" column="filename" not-null="true"></element>
</set>
```

注意：　　lazy：为 true 采用延迟加载策略，默认值

在程序内部我们可以手动级联加载，使用 Hibernate.initialize(...);

2）Bag: 无序可以重复

```
create table xuxiang_item2(
id                  number              primary key,
name            varchar2(25)
);
create table xuxiang_image2(
id                  number              primary key,
itemid          number              references xuxiang_item2(id),
filename        varchar2(50)
);
```

映射文件：

```
<idbag name="images" table="xuxiang_image2">
        <collection-id type="int" column="id">
                <generator class="increment"></generator>
        </collection-id>
        <key column="itemid"></key>
        <element type="string" column="filename" not-null="true"></element>
</idbag>
```

3）List: 有序可以重复

```
create table xuxiang_item3(
id              number primary key,
name   varchar2(25)
);
create table xuxiang_image3(
itemid   number references xuxiang_item3(id),
position number,
```

```
filename varchar2(50),
primary key(itemid,position)
);
```

映射文件：

```
<list name="images" table="xuxiang_image3">
        <key column="itemid"></key>
        <index column="position"></index>
        <element type="string" column="filename" not-null="true"></element>
</list>
```

4）Map: 有序不可以重复

```
create table xuxiang_item4(
id              number primary key,
name   varchar2(25)
);
create table xuxiang_image4(
itemid  number references xuxiang_item4(id),
imagename     varchar2(25),
filename        varchar2(50),
primary key(itemid,imagename)
);
```

映射文件：

```
<map name="images" table="xuxiang_image4" >
        <key column="itemid"></key>
        <index column="imagename" type="string"></index>
        <element type="string" column="filename" not-null="true"></element>
</map>
```

sort：指定集合排序规则

1.natural：按 ascii 码升序排序

2.order-by：指定数据库排序规则

如：order-by="imagename asc"

练习：根据文档完成 List、Map 的测试例子

2、有继承关系的类进行映射：

	是否支持多态	是否易于更新维护	是否易于出具报表
1）每个类一张表	y	y	n

2）每个具体子类一张表 n n y

3）整个类层次一张表 y y y

1）每个具体子类一张表：

```
create table xuxiang_cash_payment(
paymentid     number        primary key,
amount        number
);
create table xuxiang_credit_payment(
paymentid                   number        primary key,
amount                      number,
credit_card_type      varchar2(50)
);
create table xuxiang_cheque_payment(
paymentid     number        primary key,
amount        number,
chequeno      number
);
```

com.briup.sixth.concrate

2）整个类层次映射到一张表：

```
create table xuxiang_payment(
paymentid                   number        primary key,
amount                      number,
credit_card_type      varchar2(50),
chequeno                    number,
payment_type                varchar2(20)
);
```

com.briup.sixth.hierarchy

3、作业：模拟动态查询

自己建一张表，向表中插入多条记录

```
table(
id
name
age
gendar
city
...
)
```

java -Dage=10 -Dcity=shanghai

通过 JVM 传递多个查询条件参数，通过 hibernate 到数据库中查出满足条件的记录，并打印在控制台上

System.getProperties()

```
StringBuilder sb = new StringBuilder("from student ");
if(name!=null)
        sb.append("name=?");
if(age!=null)
        if(name!=null)
                sb.append("and age=?")
        else
                sb.append("age=?");
...
        "from Student where name='zhangsan'"
Query query = session.createQuery("from User where name=? and age=?");
query.setString(1,"...");
...
query.iterate();

com.briup.seventh.dynamic
```

第五天：

第七章 高级查询

1、hibernate 中检索对象的方式:

1）导航对象图检索：

根据已经获取的对象，去检索相关联的对象

```
User user = ...
Address address = user.getAddress();
```

2）OID 检索：

```
session.get(Student.class,OID)
session.load(Student.class,OID)
```

3）HQL 检索：

```
session.createQuery(HQL);
```

4）QBE(Query By Example)检索：

5）本地 SQL：

```
session.createSQLQuery(SQL);
```

```
begin
        for i in 1..30 loop
        insert into xuxiang_student
        values(i,'briup'||i,i,'male','shanghai');
        end loop;
        commit;
end;
/
```

2、hibernate 中绑定参数：

1）位置标识法：

```
"from Student where name=? and age=?"
query.setString(1,"briup');
query.setInt(1,20);
```

2）名字标识法：

```
"from Student where name=:myname and age=:myage"
query.setString("myname",'briup');
query.setint("myage",20);
```

注意：一般参数较多时，为了增加程序的可读性，我们使用名字标识法。

3、hibernate 中，可以将 hql 写在配置文件的<query>中，在程序内可以通过 session.getNameQuery(...)获取。

```
<query name="findStudentByName">
        <![CDATA[ from Student where name=:myname ]]>
</query>

Query query = session.getNamedQuery("findStudentByName");
query.setString("myname", "briup10");
Student s = (Student)query.uniqueResult();
System.out.println(s.getId());
```

4、通配符的使用：

查询名字像“b****1***”的值：

%：代表 0 或多个字符

_：代表单一字符

\：代表转义字符

```
sql: select * from xuxiang_student where name like 'b____1%';
```

5、投影查询：查整条记录的局部信息

```
"select name,age from Student where name=?"  //注意跟的是类名，而不是表名

com.briup.seventh.projection

String hql = "select name,age from Student";
Query query = session.createQuery(hql);
Iterator iter = query.iterate();
while(iter.hasNext()){
        Object[] row = (Object[])iter.next();
        String name = (String)row[0];
        int age = (Integer)row[1];
        System.out.println(name+":"+age);
}
```

select count(*) from xuxiang_student;

查询学生的最大年龄、最小年龄、平均年龄，打印在控制台上。

6、QBC(Query By Criteria)检索：

Criteria 擅长做动态查询
session.createCriteria(Class persistentClass)

QBE(Query By Example)

7、SQL 检索：

```
String sql = "select id,name from xuxiang_student where name=:name";
SQLQuery query = session.createSQLQuery(sql);
query.addScalar("id", Hibernate.INTEGER);
query.addScalar("name", Hibernate.STRING);
query.setString("name", "briup10");
Object[] rm = (Object[]) query.uniqueResult();
int id = (Integer) rm[0];
String name = (String) rm[1];
System.out.println(id+":"+name);
```

Hibernate 笔记于 2008 年 8 月 3 日全部完成！！！

HTML&JavaScript 学习笔记

==========

主讲：任青松

e-mail：renqs@briup.com

web ---> javaEE
HTML&JS

==========

HTML: 超文本标记语言

1、html 简介

1）html 是一种网络规范(95 年，版本是 html2.0)，由 SMGL（标准通用标记语言)语言发展而来，

依赖于 w3c 定制的一套标准(86) iso 8896

2）html 于 90 年和 web 一起在欧洲日内瓦的欧洲粒子实验室诞生

3）目前最新版本为 html4.01，于 99 年发布。

引申：1993 年 Oracle 进入中国，最早使用在保险行业，出售的是服务！

2、动态网页开发技术

1）CGI：应用程序和服务器衣交互的标准；

net： client-->server while(true)(监听)
(new Date()).toString

PERL：实现 CGI 的语言，基于进程的，效率一般。

2）PHP：跨平台，开放源码的嵌入式的服务器开发语言，可以实现和多种主流的数据库交互，灵活的处理 xml 文档；

优点：效率较高，对异常的处理比较完善，对内存的管理比较完善

3）ASP：微软的动态网页开发环境，内嵌于 IIS，使用 vbscript 和 javascript；

4）Servlet：基于 java 的，内部实现简单，是 CGI 的老的产品的代替(基于线程,效率较高)，作为 java 服务器端功能的扩展；

5）JSP：做为简化 servlet 开发的语言，在动态网页的开发中应用比较广泛；

3、html 的基本构成

1）html 顶端一行 html 的简单声明

2）html 的头部

3）html 的体部

```
<!-- 一行声名的头部：1)标准制定为 w3c 提供；2)html 的规范由 DTD 来限制，默认的为 strict.dtd；指定 DTD 使用的语言为英文；3)指定 html 的版本为 4.01，目的为浏览器解析提供信息 -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<!-- 声明浏览器里面不用显示的信息,比如一些搜索关键字,作者,描述等等 -->
    <head>
        <!--meta 元数据 提供一些页面的操作，帮我们规范页面，提供一些基本的信息供网络使用 -->
        <meta HTTP-EQUIV="Content-Type" CONTENT="text/html">
        <meta name="author" content="John" >
        <meta name="keywords" content="web development">
        <!--title：html 页面的标题，显示在浏览器上方 -->
```

```
        <title>HelloWorld Test</title>
    </head>
<!-- body：html 页面显示的内容 -->
    <body>
        Hello World !
    </body>
</html>
```

注意：1.声名的一行可以省略；
2.头部也可以省略；
3.体部包括 html 标签全部可以省略；

4、HTTP-EQUIV

1）Content-Type 和 Content-Language (显示字符集的设定)

说明：设定页面使用的字符集，用以说明主页制作所使用的文字以及语言，浏览器会根据此来调用相应的字符集显示 page 内容。

用法：<Meta http-equiv="Content-Type" Content="text/html; Charset=gb2312">
<Meta http-equiv="Content-Language" Content="zh-CN">

注意： 该 META 标签定义了 HTML 页面所使用的字符集为 GB2132，就是国标汉字码。如果将其中的“charset=GB2312”替换成“BIG5”，则该页面所用的字符集就是繁体中文 Big5 码。当你浏览一些国外的站点时，IE 浏览器会提示你要正确显示该页面需要下载 xx 语支持。这个功能就是通过读取 HTML 页面 META 标签的 Content-Type 属性而得知需要使用哪种字符集显示该页面的。如果系统里没有装相应的字符集，则 IE 就提示下载。其他的语言也对应不同的 charset，比如日文的字符集是“iso-2022-jp ”，韩文的是“ks_c_5601”。

Content-Type 的 Content 还可以是：text/xml 等文档类型；
Charset 选项：ISO-8859-1(英文)、BIG5、UTF-8、SHIFT-Jis、Euc、Koi8-2、us-ascii, x-mac-roman, iso-8859-2, x-mac-ce, iso-2022-jp, x-sjis, x-euc-jp, euc-kr, iso-2022-kr, gb2312, gb_2312-80, x-euc-tw, x-cns11643-1,x-cns11643-2 等字符集；Content-Language 的 Content 还可以是：EN、FR 等语言代码。

2）Refresh (刷新)

说明：让网页多长时间（秒）刷新自己，或在多长时间后让网页自动链接到其它网页。
用法：<Meta http-equiv="Refresh" Content="30">

<Meta http-equiv="Refresh" Content="5; Url=http://www.xia8.net">

注意：其中的5是指停留5秒钟后自动刷新到URL网址。

3）Expires (期限)

说明：指定网页在缓存中的过期时间，一旦网页过期，必须到服务器上重新调阅。

用法：<Meta http-equiv="Expires" Content="0">

<Meta http-equiv="Expires" Content="Wed, 26 Feb 1997 08:21:57 GMT">

注意：必须使用GMT的时间格式，或直接设为0(数字表示多少时间后过期)。

4）Pragma (cach模式)

说明：禁止浏览器从本地机的缓存中调阅页面内容。

用法：<Meta http-equiv="Pragma" Content="No-cach">

注意：网页不保存在缓存中，每次访问都刷新页面。这样设定，访问者将无法脱机浏览。

5）Set-Cookie (cookie设定)

说明：浏览器访问某个页面时会将它存在缓存中，下次再次访问时就可从缓存中读取，以提高速度。当你希望访问者每次都刷新你广告的图标，或每次都刷新你的计数器，就要禁用缓存了。通常HTML文件没有必要禁用缓存，对于ASP等页面，就可以使用禁用缓存，因为每次看到的页面都是在服务器动态生成的，缓存就失去意义。如果网页过期，那么存盘的cookie将被删除。

用法：<Meta http-equiv="Set-Cookie" Content="cookievalue=xxx; expires=Wednesday, 21-Oct-98 16:14:21 GMT; path=/">

注意：必须使用GMT的时间格式。

6）Window-target (显示窗口的设定)

说明：强制页面在当前窗口以独立页面显示。

用法：<Meta http-equiv="Widow-target" Content="_top">

注意：这个属性是用来防止别人在框架里调用你的页面。Content选项：_blank、_top、_self、_parent。

7）Pics-label (网页RSAC等级评定)

说明：在IE的Internet选项中有一项内容设置，可以防止浏览一些受限制的网站。

5、html 的标签

1）heading：标题

标题的标签：<h1>...</h1>

注意：html 里面的标题范围是：从<h1>到<h6>，如果超出范围不会报错，而是会按照浏览器默认大小格式显示。

2）Paragraphs：段落

段落标签： ...

html 里面显示的内容进行分段显示；

注意：段落会进行自动换行。

3）line breaks：换行

换行的标签：

注意：可以实现换行的标签 heading,par,lb

4）text formatting：文本格式

...: 粗体

...: 斜体

<big>...</big>

...: 强调体

...: 加强体

5）list：列表

1.无序列表

...

2.有序列表

...

3.自定义列表

<dl><dt>...</dt><dd>...</dd></dl>

6）Table：表格

```
<table border="1">          <!-- 注意：只有 0 表示不显示表格 -->
    <tr>            <!-- 表示某一行 -->
        <th></th>                   <!-- 表示标题 -->
    </tr>
    <tr>
        <td></td>                   <!-- 表示某一列 -->
    </tr>
</table>
```

注意： 表示空格。

7）frames：框架

frameset：框架集，即所有的框架集合体，类似于java里面的封装类；

frame：框架，一般嵌套frameset里面；

noFrame：当浏览器不支持的时候显示的替换框架；

iFrame：一般指当前框架的子框架，一般独立使用；

语法：<frameset cols=# rows=#> <!-- #表示框架的显示布局，一般由""里面写上布局参数；*表示平分页面 -->

```
<frameset cols="25%,70%,*">
    <frame src="left.htm" name=""> <!-- src表示我们的目标页面，name表示框架的名字 -->
    <frame>
</frameset>
```

8）links：链接

1.一张网页链接到另一张网页

```
<a href="">链接标题</a>
```

2.一张网页链接到同一张网页上的特定位置

a.设定链接的标记

```
<a name="标记名">
```

b.链接到地址的写法

```
<a href="#标记名">链接标题</a>
```

3.链接到不同的网页的特定位置

a.在所链接的页面的特定位置设定标记

```
<a name="标记名">
```

b.链接的地址写法

```
<a href="所链接的网页地址#标记名">链接标题</a>
```

超链接：left.htm

```
<body>
    <a href="url" target="left"></a>
</body>
```

apple.html； banana.htm； orange.htm

```
<body><table bgcolor="red"><th>name<th>color</table></body>
```

fruit.htm

```
<a href src="" target="right"></a>
```

index.html

<frameset>
welecome.htm --> right apple/banana

9）forms：表单

1.form：表单--data--->server--->接收(JavaBean)--->操作对象

2.action：1.制定 server 的路径；

2.跳转到一个新的页面；

3.发邮件；

4.操作一定的 web 编程访问路径*.do，*.action，...；

3.method：get，post

get	post
明文的	暗文
数据小于 256Byte 大于	
不安全	安全
提供搜索关键字	不提供

get 路径：<form action="../frames/apple.html" method="get">

file:///home/briup/Desktop/HTML%26JS/HTML/frames/apple.html?name=&passwd=&submit=%E6%8F%90%E4%BA%A4

post 路径：<form action="../frames/apple.html" method="post">

file:///home/briup/Desktop/HTML%26JS/HTML/frames/apple.html

```
<body>
    <form action="" method="">
        <table>
        <p>
        <ul>
        <input >
    </form>
</body>
```

10）images：图片

1.作为超链接；

2.作为背景；

3.展示；

```
<html>
    <head>
        <title>ImageTest</title>
    </head>
    <body background="2.jpg">
```

```
                <!-- show -->
                <img src="1.jpg" alt="Jingtian" width="220" height="340">

                <!-- link -->
                <a href="../forms/login.html">
                        <img src="3.jpg" width="160" height="220">
        </body>
</html>
```

11）多媒体：音频和视频

```
<embed src="1.mp3"                          音频文件的路径
        height="500" width="700"    视频高度和宽度
        autostart="true"                    是否自动播放
        nosave="true"                       暂存
        loop="true">                        循环播放
</embed>
```

12）Java Applet：

```
<applet code="HelloWorldApplet.class"
        width=190 height=160>
</applet>
```

13）滚动标签：

```
<marquee><font size=7 color=red>www.briup.com</font></marquee>
```

javaPx

6、CSS

CSS：层叠样式表(html4.01 版本的新特性)

```
//body 样式,表示 body 的背景颜色为绿色
body{background-color:green}
//段落样式,表示段落里面的字体为黄色,并且距离浏览器左边为 20 象素
p{color:yellow;margin-left:20px}
//表示标题 1 字体为宋体,并且显示为斜体大小为 30 象素
h1{font-family:宋体;font-size:30px;font-style:italic}
//表示标题 2 的颜色为蓝色
h2{color:blue}
表示列表前面有一个图片"speak.gif"
ul{list-style-image:url("speak.gif")}
//表示为 img 标签里面定义一个样式,图片在文字上方(与图片最上方平齐)
img.top{vertical-align:text-top}
//表示为 img 标签里面有一个样式,图片在文字下方
```

```
img.bottom{vertical-align:text-bottom}
// *#表示定义一个样式变量 fontstyle2
*#fontstyle2{color:white;font-size:40px}
```

style: 1)selector(目标)
2)property(参数)
3)value(参数的值)

引用样式的两种形式：1.直接调用；2.引用

```
<head>
    <style type="text/css">
      img.bottom{vertical-align:text-bottom}
    </style>
    <link rel="stylesheet" type="text/css" href="style.css"/>
</head>
<body>
    <h3 style="{color:blue}"></h3>
    <h4 id="fontstyle2"></h4>
    <img class="top">
</body>
```

css img.top

日志：log4j --> com.briup

==========

Java Script

一、JS 概述

1、JS 的概念：

1）JavaScript 是一种网页编程技术，大部分使用者将它用于创建动态交互网页；

富客户端(动态交互网页)：给用户一个美好的使用体验，系统的布局以及风格根据客户的意愿随时自由调整。

通过 Ajax 技术实现 ---> 异步调用 ---> JS 调用 ---> servlet

2）JavaScript 是一种基于对象和事件驱动的解释性脚本语言，具有与 Java 和 C 语言类似的语法；

1.对象：具有一定意义并且包含属性和行为的程序体。

2.基于对象：由于 javaScript 的对象操作基本上都是由其它语言封装提供的，而自己本身主要是起一个调用执行的作用，故称之为基于对象。java，c，php，jsp，html

3.事件：计算机的鼠标、键盘等外部输入设备以及例如时间的因素等引发的计算机程序操作。 setTimeout

4.基于事件：程序一直处于循环监听状态，当某一事件操作发生而去激发的行为操作的过程称为基于事件。

5.解释性脚本语言：在程序运行的时候，不用编译，直接读一行解释执行一行。

6.JavaScript 属于弱类型语言：JS 中所有变量声明都为 var xxx;

Java:

```
int a = 10;
double b = 11.0;
Date d = new Date();
```

JS:

```
var a = 10;
var b = 11.0;
var d = new Date();
```

3）JavaScript 是一种使用简单，功能强大的编程语言；

2、JavaScript 的用法

1）使用的位置：

头部，体部，html 标签之外，html 标签里面(head,body 之外);

2）调用方式：

1.事件激发；

2.直接引用 js(<script>标签里面)

3.直接定义(<script>标签体部)

3）js 文件：

javaScript 文件，可以独立定义为*.js，里面主要内容为 function 的声明。

3、JS 的发展历史

1）网景公司在 Netscape2.0 首先推出了 JavaScript；

2）微软公司从 IE3.0 开始提供对客户端 JavaScript 的支持，并另取名为 JScript；

3）微软公司从 IIS3.0 开始提供对服务器端 JScript 的支持；

4）微软推出 JScript.NET，基于.NET 框架的 JScript 具备了和 C#.NET 和 VB.NET 一样强大的功能；

4、JS 特点

1）简单易用

简洁易用，与 Java 有类似的语法

可以使用任何文本编辑工具编写

只需要浏览器就可以执行程序

2）解释执行

事先不编译

逐条执行

无需进行严格的变量声明

3）基于对象

内置大量现成对象，编写少量程序可以完成目标

5、JS 使用范围

1）适合做哪些事情：

1.客户端数据计算

2.客户端表单合法性验证

3.浏览器对象的调用

4.浏览器事件的触发

5.网页特殊显示效果制作

2）不适合做哪些事情：

1.大型应用程序

2.图像、多媒体处理

3.网络实时通讯应用

二、JS 的基础语法

类似于 java 的语法！！！

1、常见面试题：

```
int a = Integer.MAX_VALUE-10;
for(int b=a;b<=Integer.MAX_VALUE;b++){        //死循环，只有将<=变为<时才能输出10 次 hello!
        System.out.println("hello");

}
```

2、练习：calculate ---> 将十进制转换为二进制，同时以对话框的形式返回给客户；

```
<html>
    <head>
        <title>calculate Test</title>
        <script language="javascript">
            function cal(){
                var value = parseInt(document.getElementById("num").value);
                //注意 getElementByName()在 FireFox 下不支持
                var str = "";
                for(var j=31;j>=0;j--){
```

```
                        str += value>>j&1;
                    }
                    alert(str);
                    document.getElementById("num").value="";
                }
            </script>
        </head>
        <body>
            <input type="text" id="num" size=10>
            <input type="button" value="转为二进制" onclick="cal()";
        </body>
    </html>
```

3、prompt("","");确认对话框，第一个参数是问题，第二个参数是答案，最终执行结果是将答案返回给接收的变量！

三、JS 常用内置对象(11 种)

1、11 种内置对象简介：

类型	内置对象	介绍
数据对象	Number	数字对象
	String	字符串对象
	Boolean	布尔值对象
组合对象	Array	数组对象
	Math	数学对象
	Date	日期对象
高级对象	Object	自定义对象
	Error	错误对象
	Function	函数对象
	RegExp	正则表达式对象
	Global	全局对象

2、栅栏柱错误：一般情况下我们在编程中处理平均分配问题时，可以直接在平分对象中间插入条件，也可以在中间以及两边处理，切忌不能使用和平分数目一样的条件进行处理。

3、rub on rails：快速开发编程

String(负索引)：str => (-1)

计算器：

界面 ---> 按钮的操作(value-send->text) ---> this.value ---> text

四、DHTML(动态 HTML 对象)

1、DHTML 的定义

1）使用 JavaScript 和 CSS 级联样式表操作 HTML 创造出各种动态视觉效果统称为 DHTML

2）DHTML 是一种浏览器端的动态网页技术

2、DHTML 的功能

动态改变字体大小和字体颜色

动态设定文档元素的位置、内容，甚至隐藏和显示元素

可以通过事件响应机制制作动态折叠的树形结构和菜单

可以通过定时器制作时钟、日历

可以弹出对话框与用户进行交互

可以通过表单提交用户填写的信息

通过动态样式表可以设定更多的显示效果

...

3、DHTML 对象模型

1）将 HTML 标记、属性和 CSS 样式都对象化

2）可以动态存取 HTML 文档中的所有元素

3）可以使用属性 name 或 id 来存取或标记对象

4）改变元素内容或样式后浏览器中显示效果即时更新

5）DHTML 对象模型包括浏览器对象模型和 Document 对象模型

4、定时器:

setInterval 和 setTimeout 的区别：

1. setInterval 是周期性的，启动后除非采用 clearInterval 否则不会停止；

2. setTimeout 是一次性的，到时间执行完毕指定代码后就会自动停止，但是可在代码结束的地方再次用 setTimeout 定时达到与 setInterval 同样的效果。

五、JavaScirpt 高级技巧

1、正则表达式：

1）正则表达式的概念：

1.是一种模式匹配下文本替换、搜索、提取的强有力工具

2.在 JavaScript 下是一个用于处理字符串匹配的对象

2）正则表达式的功能：

1.查找文本中的字符串

2.替换文本中匹配的字符串

3.对表单输入值进行校验

3）Regular Expression 对象的定义：

```
rgExp=/pattern/flags
rgExp=new RegExp("pattern",["flags"])
```

其中 flags 标识有以下几个：

g：设定当前匹配为全局模式

i：忽略匹配中的大小写检测

m：多行搜索模式

4）email 地址有效性主要有以下规则：

1.必须在用户名后包含@；

2.@后加上的域名必须由一个或多个数字和字母组成；

3.域名类型后缀必须是 2 到 3 个字母组成；

4.在主机域名和域名类型后缀之间必须用点“.”隔开；

5.email 地址大小写无关。

分析正则表达式：/^[\w.-]+@([0-9a-z][\w-]+\.)+[a-z]{2,3}$/i

^匹配字符串的开始，$匹配字符串的结尾；

方括号[]内的字符如果不加上转义字符都代表本身含义；

在字符后加上减号：代表排除该字符，例如[\w.-]表示除了点以外的任意 ASCII 字符；

在字符后加上加号：代表一个或多个该字符；

[0-9a-z][\w-]+\. 表示一个或多个数字或字母组成，末尾加上点字符”.“；

[a-z]{2,3} 表示 2 到 3 个字母组成。

2、动态样式表(CSS)

1）动态样式表：指的是在 JavaScript 中直接对对象的 style 属性进行设定，HTML 组件中的 style 属性将立即起作用；

2）CSS 样式表

1.链入外部样式表文件 (Linking to a Style Sheet)；

先建立外部样式表文件（.css），然后使用 HTML 的 link 对象。

2.定义内部样式块对象 (Embedding a Style Block)；

可以在 HTML 文档的<HTML>和<BODY>标记之间插入一个<STYLE>...</STYLE>块对象。

3.内联定义 (Inline Styles)：指在对象的标记内使用对象的 style 属性直接定义少数的样式表属性；

```
<div style="font-size:36pt;font-family:黑体"
    onmouseover="this.style.color='#FF0000';"
    onmouseout="this.style.color='#000000';" >
    鼠标移过来，马上变颜色
</div>
```

HTML&JavaScript 笔记于 2008 年 8 月 5 日全部完成！！！

Servlet 学习笔记

==========

主讲：任青松

Servlet 前言

1、学习任务：

day1：了解 servlet，明确 web 应用程序的基本框架，熟悉 web 服务器的基本结构以及使用方式。除此之外，手工能书写一个简单的 servlet，并且部署运行。

day2：明确 form 表单配合 servlet 的使用，并且实现具有一定业务逻辑的 web 应用。

day3：servlet 的生命周期，以及一些高级方法。

day4：servlet 结合数据库的使用(JNDI)。

day5：综合处理复杂业务。

2、软件开发的模式(架构)：

C/S(Client/Server)：客户端会分担服务器的一些业务处理，降低服务器的压力，提高应用效率。--> ex: qq

缺点：1）升级维护比较麻烦；

2）资源要求，尤其是局域网使用比较高；

B/S：现阶段应用开发的主流架构，客户只需要打开浏览器，就可以进行所有操作。

优点：1）使用的资源相对较少；

2）升级维护非常方便(无缝升级)；

开发技术:

1）静态网页技术：html,js,css

2）动态网页技术：CGI,ASP,ASP.net,Servlet,jsp,php 等等

3、Servlet：开发使用 B/S 架构。

http://192.168.4.53:8080/myServlet/register.html?a=b

URL：协议://ipAddress:port/web 应用/具体的请求路径

1）协议：servlet 里面使用的主要是 http 协议；

2）port：webServer 的端口号；

3）webServer：可以部署 web 应用的软件，同时可以接收客户端的请求(request)，并且在服务器端做完处理，返回给客户端(response)；

ex: 开源的:tomcat(Apache-->美国的软件基金会),Jboss

商用的:weblogic(BEA),webSphere(IBM)

微软:IIS

4）web 应用的名称(也叫 ContextPath,上下文路径)：myServlet --> java 工程 name；

request.getContextPath(); ---> /myServlet

5）具体的请求(资源名称)：/register.html；

request.getServletPath(); ---> /register.html

request.getQueryString(); ---> a=b

4、什么是 Servlet?

1）sun 公司提供的 JavaEE 规范，主要用于开发 web 应用；

2）简单的说：servlet 就是一个使用了一定协议以及 Api 的 java 程序；

专业的说：servlet 是一套开发 web 应用的规范，其内部实现全部由 java 语言实现，具有 java 语言的一切优点。开发的过程是用 java 开发 java 应用。换句话说就是使用一些已经写好的 java 程序去实现一定的业务，即也就是结合业务调用对象以及方法的过程；

3）servlet 的主要实现都位于：javax.servlet.*以及 javax.servlet.http.*两个包下；

4）CGI 的替代品；

5）一个可以处理多用户请求的程序，处理方式：同一份代码，用线程来控制请求处理；servlet api 明确说明 servlet 不是线程安全的，原因是所有的请求处理都是在一个 Servlet 里面完成的。

第一章 Servlet Basic

1、CGI：公共网关接口协议，事实上是运行在服务器端的一段应用程序。可以开发 web 应用，并且可以使用多种语言(C,C++,VB,perl...)进行编程。

优点：多语言开发，可以被程序员很容易地接受；

缺点：移植性不好；客户端产生的一个请求，服务器端要产生一个进程去处理，系统消耗非常大。

2、http 协议：method --> get/post

状态：200(ok)

行为结果标识：500，404

3、MIME：描述一种因特网消息显示方式。

格式：消息的大类型/具体的类型

ex：text/html, text/xml, image/jpg

指定：setContentType("text/html");

4、servlet 是 CGI 的替代品，是因为：

首先，CGI 虽然支持多语言，但相对于同一种语言之间的调用，执行效率相对低；

其次，在安全方面，CGI 程序不如 java 稳定；

第三，在平台迁移方面，java 语言的程序具有绝对优势；

第四，CGI 脚本的使用对浏览器有要求。

注意：JSP 是 Servlet 的轻量级封装，是 Servlet 的替代品。

5、servlet(web)应用的请求过程：

1）客户打开浏览器，输入 url，使用 http 协议发起请求；

2）webServer 接收客户端的请求，加载 servlet 实例(第一次)，或者直接查找已经加载的实例；

3）webServer 根据 request 的请求数据，调用 service 方法(servlet 特有的业务处理方法，除过本身之外有两个替代品 doGet(),doPost())，处理业务逻辑，然后返回一个对象；

4）webServer 将返回的对象进行包装，通过 response 返回给客户端；

5）客户端接受 response 的信息进行显示或者处理。

6、UML：统一建模语言

物理视图：以物理图形式表现给用户的系统图。

时序图：应用根据执行时间顺序的表现图，画 servlet 的处理过程。

7、servlet组件的开发：

1）书写servlet类；

a.继承HttpServlet，并实现service方法

1.(doGet,doPost)-->一般情况下只实现一个，另外一个去调用该实现，

2.该方式比较完整，推荐使用。

```
class Test extends HttpServlet{
	public void doGet(HttpServletRequest request,HttpServletResponse response){
		doPost(request,response);
	}
	public void doPost(HttpServletRequest request,HttpServletResponse response){
		//在此方法中主要完成业务，构建动态页面显示给客户。
		String name = request.getParameter("name");
		method();
		response.setContentType("text/html");
		PrintWriter pw = response.getWriter();
		pw.println("<html>");
		...
		pw.println("</html>");
		pw.flush();
		pw.close();
	}
}
```

注意：所有的Stream流都是字符流，PrintWriter是字节流，Object是对象流。

在浏览器显示一个xml文档：

```
setContentType("text/xml");
<person>
	<firstName>terry</firstName>
	<lastName>hello</lastName>
</person>
```

http://localhost:8080/servlet/basic/first?name=xuxiang
http://localhost:8080/servlet/basic/xml?firstName=xu&lastName=xiang

b.继承一个抽象类GenericServlet，一般使用较少，操作多媒体等业务逻辑时使用。

```
class Test extends GenericServlet{
	public void service(req,res){

	}
}
```

.jpg--->程序--->发送给客户显示

```
res.setContentType("image/jpeg");
File file = new
File("/home/briup/workspace/servlet/src/com/briup/basic/1.jpg");
InputStream is = req.getInputStream();
FileInputStream fis = new FileInputStream(file);
OutputStream os = res.getOutputStream();
int temp;  //设置临时缓存区
while((temp=fis.read())!=-1)
        os.write(temp);
os.flush();
is/os.close();
```

c.实现接口 Servlet，在对 servlet 生命周期严格控制的需求里面使用。

```
class Test implements Servlet{
        //所有 servlet 类里面的方法都得定义，包括生命周期控制方法和
service 方法。
}
```

2）书写 servlet 部署文档

部署文档：web.xml，主要负责 webServer 和 web 应用之间的标识以及协作。

```
<servlet>
        <!-- servlet 的标识，可以任意指定  -->
        <servlet-name></servlet-name>
        <!-- servlet 类的全路径(包.类名) -->
        <servlet-class></servlet-class>
</servlet>

<servlet-mapping>
        <!-- servlet 的标识，和上面 servlet 标签里面的必需一致，作为匹
配条件  -->
        <servlet-name></servlet-name>
        <!-- 具体标识一个 servlet 的访问路径，url 里面 web 应用名后的部
分，必须以/开头  -->
        <url-pattern></url-pattern>
</servlet-mapping>
```

3）部署：

㈠ conf/tomcat-users.xml ---> 配置用户的信息以及权限。

```
<role rolename="manager"/>
<role rolename="admin"/>
<user username="briup" password="briup" roles="admin,manager"/>
```

㈡ 将我们的web应用部署在webServer上，然后启动webServer，在地址栏里面敲上地址进行访问。

tomcat:

1.bin：主要放置tomcat的脚本文件(包括各种操作系统的)

ex: start.bat,shutdown.bat(windows)
shutdown.sh,startup.sh(unix/linux)

```
chmod -R 755 bin
./startup.sh
http://localhost:8080

export JAVA_HOME=/opt/java/jdk/jdk1.5.0_06
export PATH=$JAVA_HOME/bin:$PATH
export CATALINA_HOME=/home/briup/jakarta-tomcat-5.0.28
export
CLASSPATH=.:$JAVA_HOME/lib:$CATALINA_HOME/common/lib/servlet-api.jar
```

2.common：存放tomcat和web应用公用的jar包，除此之外还有公用的公共类的.class文件。

3.conf：存放tomcat自身的配置信息以及用户和web应用的配置数据(数据源)。

4.logs：tomcat处理web程序的日志文件。

5.server：tomcat自身的工程以及应用程序，还有示例程序。

6.shared：基本用处同common。

7.work：jsp代码编译成servlet程序后存放的目录。

8.webapps：web应用的部署目录。

建立和工程名字一样的文件夹，将WebRoot下的所有内容拷贝到此文件夹即可。

启动tomcat，在地址栏里面敲url：

http://localhost:8080／servlet/basic/hello
http://localhost:8080/servlet/basic/hello?name=xuxiang

web应用的参数形式：

key=value

资源文件：

格式：key?value ?-->=,-," "

java类：java.util.properties

```
briup@briup:~> mkdir -p servlet/src
briup@briup:~> mkdir servlet/WebRoot
    -->jar/classes/.xml/jsp/php/html/image
briup@briup:~/servlet/src/com> touch MyFirstServlet.java
```

WebRoot
|
WEB-INF jsp/asp/php/css/js/html ...
|
lib classes .xml
| |
jars 编译好的类

在 WebRoot 下：
mkdir lib classes
touch web.xml

编译：javac -d WebRoot/WEB-INF/classes -cp WebRoot/WEB-INF/lib/servlet-api.jar src/com/MyFirstServlet.java

-cp：-classpath

8、手工方式开发一个 web 应用

1) 创建一个目录 Servlet,进入这个目录，创建一个 WEB-INF 目录

2) 进入 WEB-INF 目录,创建 classes 和 lib 目录以及 web.xml

3) 在.bashrc 中配置环境变量

环境变量：指系统启动时需要加载的一些常用的配置以及软件。

全局的环境变量：profile

局部环境变量：

sh ---> .profile

bash ---> .bashrc

export JAVA_HOME=/opt/... ---> jdk 的安装目录

export CLASS_PATH=.:$JAVA_HOME/lib ---> 二进制文件的存放路径

echo $JAVA_HOME

csh ---> .cshrc

setenv JAVA_HOME /opt/...

set PATH=(/usr/bin:...) ---> 可以使用的程序的源目录(ex：命令,$JAVA_HOME/bin)

linux 用户:

1) 打开一个终端

2) vi .bashrc

```
//CATALINA_HOME=/opt/jakarta-tomcat-5.0.28
CATALINA_HOME=/home/briup/jakarta-tomcat-5.0.28
CLASSPATH=.:$CATALINA_HOME/common/lib/servlet-api.jar
```

export CATALINA_HOME CLASSPATH

3) 保存退出

4) . ./.bashrc

5) echo $CLASSPATH(查看配置是否生效)

windows 用户:

1) 右击我的电脑-->高级--->环境变量

2) 在 CLASSPATH 的环境变量上增加一个 CATALINA_HOME 和 CLASSPATH

CATALINA_HOME=C:\jakarta-tomcat-5.0.28

CLASSPATH=.;%CATALINA_HOME%\common\lib\servlet-api.jar

4) 编写一个 myFirstServlet.java

```
package com;
import javax.servlet.http.*;
import java.io.*;
public class MyFirstServlet extends HttpServlet {
    ......
}
```

5) 在 web.xml 中描述这个 servlet

6)编译

javac -d WebRoot/WEB-INF/classes -cp WebRoot/WEB-INF/lib/servlet-api.jar src/com/MyFirstServlet.java

7) 新建文件夹和工程名字一样,将 webRoot 下的内容全部拷贝到 tomcat 下的 webapps 目录下

8) 启动 tomcat

进入 tomcat 下的 bin 目录，启动方式:

linux 用户:

右击鼠标-->打开终端-->运行 ./startup.sh 启动 tomcat

运行 ./shutdown.sh 关闭 tomcat

windows 用户:

双击 startup.bat 启动 tomcat

双击 shutdown.sh 关闭 tomcat

9) 在地址栏中输入 URL 访问

http://localhost:8080/servlet/basic/hello?name=xuxiang

第二章 Form 表单

1、from：表单 ---> 封装数据

2、web 应用：

client---req--->server---res--->client
|
\|/
form data(req.getParameter) html(div)

3、form 表单数据获取的方法：

register.html
http://localhost:8080/servlet/form/register.html

单值：text,radio,select,textarea
---req--->req.getParameter("name");
多值：checkbox
--->req.getValues("name")--->String[]
|
\|/
User ---> pojo 类 ---> ManagerUser.register(pojo)
|
\|/
Map/Properties map.put(pojo.name,pojo);
|
\|/
Servlet---make--->loginForm--input--->userName,passwd
|
\|/
验证用户名和密码--->根据用户名拿到 User 对象--->ManagerUser.login(String userName,String passwd)--->请求服务器验证(servlet)--->successPage

register.html(form)--->RegisterServlet(register)--->动态的 Html(请求 loginFormServlet)--->loginFormServlet(loginForm)--->loginFormHtml(login)--->loginServlet--->give client a result

相对路径：
servlet/form/register
servlet/form/register.html--->action"register"

4、form 表单中文乱码解决办法：

1）form 表单输入中文正常，以 post 方式提交到服务器，服务器端获得的字符为乱码：
request.setCharacterEncoding("UTF-8");

2）服务器端中文显示正常，而发送给客户端的字符显示乱码：
response.setCharacterEncoding("UTF-8");

3）如果表单输入中文正常，并且以 get 方式提交到服务器，服务器端接收的结果为乱码，需要修改 webServer 的配置：

tomcat/conf/server.xml--->Connector 添加属性 URIEncoding="UTF-8"

4）国际化解决方式，使用资源文件*.properties 配置键值对解决--->JSP 国际化

5、form 表单在 servlet 的使用：

1）form 表单的请求发送方式：

get：

1.get 信息一般放在头部；

2.信息一般小于 256byte;

3.一般来说传送的是不加密的数据；

4.get 一般用的就是临时的数据，换句话说，get 方式就是想从服务器端拿数据到客户端，并不是发送自身的有用数据到服务器端；

5.get 方式是我们 form 表单的默认的提交方式；

表现形式：<form method="get"/>

url 的头部?name=name&passwd=passwd

超链接使用的也是 get 方式

post：

1.post 信息一般放在体部(http 协议)；

2.信息长度一般较长；

3.加密的数据，或者安全性要求的数据必须以 post 方式处理；

4.主要是向服务器提供数据；

2）请求方式：

1.<form action="?">

?：可以是绝对路径，也可以是相对路径

2.超链接

3.直接在浏览器里面请求相应的 url

3）form 表单与服务器之间的编码转换

4）常用方法：

主要是参数的提取方式；

action="*.do" --->web.xml

<url-pattern>

/servlet/test or 通配

</url-pattern>

通配方式：

前通配：*.do，*.action

后通配：/servlet/test/*

注意：如果 web.xml 中 url-pattern 有重复的，请求时将以最后一个为准。

第三章 Servlets 生命周期

1、容器(container)：

1）web container：指存放 web 应用的一个虚拟的东西，是 webServer 的一部分。

当我们在运行 web 应用的时候，request 发送到 server，server 会将请求委托给一个 container 来处理。

2）container 分两种：

1.web Container：如 Servlet Container，JSP Container，...

2.EJB Container：处理 EJB 应用

注意：Tomcat 只有 WebContainer，没有 EJBContainer;

Weblogic，Jboss，Websphere 两种容器都有。

2、Servlet 的生命周期

1）装载并初始化一个 servlet;

1.装载

方式： a.server 启动时装载

<load-on-starup>num</load-on-startup>

num>=0 ---> 启动装载

num=0 ---> 表示最晚加载

>0 的值 ---> 越小越早加载

b.延迟装载

num<0 ---> 延迟加载

当我们不配置的情况下，会延迟加载；

当我们发送一个 request 到 server 端时装载；

2.初始化:

初始化一个 servlet

a.首先(container)调用 web.xml ---> 获得 ServletContext 对象

ServletContext：全局的 servlet 参数，可以被所有的 servlet 使用，在 web.xml 中配置(比如数据库连接参数)

```
<context-param>
    <param-name>passwd</param-name>
    <param-value>jd0806</param-value>
</context-param>
```

b.container 调用 ServletConfig 对象

ServletConfig：具体的 servlet 的参数，只能被自己使用；

```
<init-param>
<param-name>author</param-name>
```

<param-value>xuxiang</param-value>
</init-param>

c.将 context 加入到 config 中

此操作由 servlet 容器完成；

d.容器根据 config 初始化一个 servlet

3.特点：

首先一个 servlet 的初始化在整个生命周期中只执行一次；

其次所有的操作都是由容器完成的，容器会启动一个线程，将所有的初始化工作完成；

servlet 在创建之前的工作(在同一个线程中完成):
container-->web.xml-->servletContext-->servletConfig-->servlet

---- 读取 ----			
-----加入 ctx-	----------		
--读取加入 config----------		
---ctx 加入到 config-------	---------- ---	
----- init(config)----------	----------	----------

2）调用 init 方法，初始化变量；

```
class GenericServlet implements Servlet,ServletConfig{
    public void init(ServletConfig config) throws ServletException{
        this.config = config;
        this.init();
    }
    public init(){


    }
}
```

注意：首先调用有参的 init();

特点：在一个 servlet 里面只处理一次；

初始化工作完成--->servlet 实例创建成功，并且这个实例一直保存在 container 里面，并且处于 service 状态，直到调用 destroy 方法。初始化线程任务结束。

注意：整个 web 应用过程，只有一个 servlet 实例存在。

前两步不需要考虑多线程问题。

3）调用 servlet 的 service 方法，进行业务处理；

当有一个请求过来，servlet 实例会分配一个线程给用户，进行 service 操作(调

用 service 方法)。

```
class GenericServlet implements Servlet,ServletConfig{
        void service(ServletRequest req,ServletResponse res){

        }
}
class HttpServlet extends GnericServlet{
        void service(ServletRequest req,ServletResponse res){
                HttpServletRequest hreq = (HttpServletRequest)req;
                HttpServletResponse hres = (HttpServletResponse)res;
                String method = hreq.getMethod();
                if(method.equals("get")){
                        doGet(hreq,hres);
                }else if(method.equals("post")){
                        doPost(hreq,hres);
                }
                void doPost(HttpServletRequest hreq,HttpServletResponse hres){

                }
                void doGet(HttpServletRequest hreq,HttpServletResponse hres){

                }
        }
}
```

servlet 的多线程问题：

解决方式：1.将 servlet 类做成单例模式；

a.私有的构造器；

b.私有的本类的引用；

c.公有的提供初始化的方法--->调用私有构造器，返回本类的引用；

每有一个用户的请求，我们就产生一个 servlet 实例。这种方式效率极低，最不推荐使用!

2.给调用的方法加锁：synchronized

```
doGet(){
        method();
}
synchronized void method(){

}
```

使用条件：在某些效率要求不高的情况下使用。

3.使用类似于使用局部变量代替实例变量等公用的资源，使用私有的代替。

4）销毁 servlet;

container 去调用 destroy()方法结束 servlet 的生命，并且记录操作日志。

1）应用程序手动调用

2）webServer 正常关闭时调用

注意：在执行此方法关闭 servlet 的时候，其过程是等待所有的请求处理完毕，再关闭，这时候不再分配线程给其他的请求。

clear shutdown
kill -9 processnum

tomcat:

shutdown.sh--->正常关闭

eclipse：点击 shutdown--->正常关闭

强制关闭：不会调用 destroy()方法
ps -ef | grep tomcat
kill -9 tomacatpronum

eclipse：点击控制台的小红点

3、循环者的新娘/循环者的儿子：一种编程的现象

第四章 资源访问

1、JDBC 编程六大步：

1）注册 driver

1.class.forName(driver)
2.new oracle.jdbc.driver.OracleDriver();
3.java -Djdbc.drivers=oracle.jdbc.driver.OracleDriver
jdbc.drivers=System.gerProperties();

2）获得 Connection

1.conn = DriverManager.getConnection(url.userName,passwd);
2.driver.conn(url,properties);

3）获得 Statement

简单的 Statement: conn.createStatement();
PrepareStatement: conn.preparStatement(String sql);
CallableStatement: conn.prepareCall(String sql);

4）执行 sql 语句

```
stmt.execute(sql);
stmt.executeUpdate(sql);
stmt.executeQuery(sql);
pstmt.execute*();
```

5）获得并处理结果集

```
ResultSet rs = stmt.executeQuery(sql);
while(rs.next()){
    rs.getInt("name"); //名字、位置标识符
}
```

6）释放资源

```
if(rs!=null)rs.close();
if(stmt!=null)stmt.close();
if(conn!=null)conn.close();      //注意：不需要每一次都关闭
```

getConnection()
com.briup.jdbc.common
--ConnectionFactory
--jdbc.properties
oracle.dirver=oracle.jdbc.driver.OracleDriver
com.briup.jdbc.dao
--Interface--->IDao
void save(User user);
void delete(String userName);
List findAllUser();
--class IDaoImpl implements IDao
com.briup.jdbc.pojo
--User
register.html---form--->RegisterServlet(register)--->IDaoImpl.save(user)

查询所有用户--->FindAllUserServlet--->findAllUser()--->response--->在servlet中构建一个表格显示，并且每一个用户最后有一个删除字样的超链接---点击--->DeleteUserServlet---调用IDaoImpl.daleteUser(String userName)--->FindAllUserServlet显示删除后的效果

<a href="deleteUser?userName="+u.getUserName>--->req.getpar("userName");
findAllUser--->select username... from usertab;--->request.getpar()
?no--->findAllUser()--->List
遍历List--->user--->getProperties--->构建表格(table)

deleteUserServlet
dao.deleteUser(userName);
request.gerpar("userName);

写程序：

1/1/1原则：
1/3: thinking
1/3: coding
1/3: testing

2/8原则：
20%: coding
80%: testing

效率：数据库调优工程师

2、JNDI ：Java命名目录服务器

JDBC--->Connection

JNDI--->Connection

1）JNDI: java 命名目录服务器，本身是一个 javaEE(J2EE)规范。主要作用是提供一些资源的访问，本身有一套管理机制。

2）DateSource：数据源，提供了一系列的数据库 Connection 对象，存放在本身的连接池里面，用户有请求，直接分配一个连接给用户，用户使用完毕之后，会将连接放回 source 里面，不用重复创建，大大节省了资源消耗，大型的项目开发一般都使用。

3）Connectionpool 机制：本身会维护一定数量的连接，当用户比较多的时候，资源不够分配，会自动创建一些连接，当资源闲置的时候，又可自动的回收一部分资源。

“3p 模型的体现”

4）获得 Connection

```
Context ctx = new InitialContext();
DataSource ds = (DataSource)ctx.lookup("java:comp/env/jndiName");
Connection conn = ds.getConnection();
```

5）开发中我们一般使用 WebServer 配置数据源，目前的 webServer 都提供数据源支持。

6）绑定数据源和 web 应用，在 web.xml 中绑定。

配置数据源的步骤：

步骤一、将数据库驱动(ojdbc14 或者 class12)包放在 tomcat 的 common 的 lib 目录下；

步骤二、使用 Tomcat 的 admin 应用中以图形化的方式配置数据源：

Tomcat Administration --> Tomat Servlet --> Service --> Host --> Context --> Resources --> Data Source --> Create New Data Source

JNDI Name:jdbc/ds

Data Source URL：数据库的 URL

jdbc:oracle：thin:@192.168.1.220:1521:briupdb

JDBC Driver Class：oracle.jdbc.driver.OracleDriver

User Name：数据库的用户名

Password：数据库的密码

Max.Active Connections：最大的用户连接数(4)

Max.Idle Connections：当连接池中没有连接的时候，新创建的连接数(2)

Max.Wait for Connection：等待连接创建的时间(5000)

Validation Query：测试连接有语句

点击 save&commit changes

步骤三、web.xml 中配置资源引用（可选，新版本的 tomcat 不需要配置）；

```
<resource-ref>
    <description>
        Oracle Datasource example
    </description>
    <!-- jndi 的名字 -->
    <res-ref-name>jdbc/oracle</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
```

步骤四、在 Servlet 中使用以下代码访问：

```
Context ctx = new InitialContext();
DataSource ds = (DataSource)ctx.lookup("java:comp/env/jdbc/oracle");
Connection conn = ds.getConnection();
```

注：配置数据源之后，在 Tomcat/config/Catalina/..目录下会新建一个 xml 文件，里面存放着与数据源相关的信息。

3、资源访问定向：

1）服务器内部跳转：

1.首先获得分发器

分发器：主要是对资源进行请求转发的 servlet 对象。

RequestDispatcher

a.dispatch = request.getRequestDispatch("url");

url: 可以是相对路径，也可以是绝对路径。

b.dispatch = getServletConntext.getRequestDispatch("url");

url: 必须是绝对路径。

servletContext = super.getServletContext();

--->web 应用的路径（ ex：/sevlet)

换句话说 servletcontext 就是表示应用的路径。

servletContext.getInitPar("url");

url: 需要传递出去工程名(web 应用的路径)即以工程路径作为根路径，如："/form/addUser"

2.资源请求：两种服务器跳转方式

forward: 会清除请求应用的体部信息

include: 不会清除请求的体部信息

3.request 常用的几个方法

跳转的发起应用带上自己的一些处理结果：

request.setAttribute("key",value);

接受的应用根据 key 值拿到结果：

request.getAttribute(key);

删除 request 范围内的属性值(结果值)：

request.removeAttribute(key);

4.什么时候分别选用哪种服务器内部跳转方式?

当 client 需要两个页面的所有内容的时候，我们选用 include;

当我们不关心第一个页面体部信息，只关心其所处理的业务结果的时候选用 forward;

5.特点：

服务器内部跳转，我们 client 端的 url 不变，也就是说，用户不用去关心服务器的内部处理，这是 B/S 架构的特点之一。

client 端只发出一次请求 ---> 优点：server 可以隐藏 url 地址，在很大程度上提升了系统的安全性。

2）服务器的外部跳转(重定向)：

从一个 servlet 跳转为另一个 servlet，在操作过程中，发起了两次请求。一次为原来的 servlet--->request/response
--->client--->servlet 只是我们 client 体会不到原来的 servlet 的返回，直接跳转到第二个 servlet 处理之后的页面。

response.sendRedirect("url");

url: 可以使用绝对路径，也可以使用相对路径。

相对路径和绝对路径：

如果使用严格的编程规范，也就是目录结构非常成熟，建议使用相对路径；

要是不确定，尽量使用绝对路径，并且使绝对路径里面的变量尽量的灵活获取；

3）servlet 服务器内部跳转和外部跳转：

1.能使用 forward 就不要使用 sendRedirect;

a.性能方面：跳转次数

b.安全方面：url 的显示

情况：frame 框架搭建的页面在操作主页面结束后返回的部分页面内容不能显示：

若服务器内部跳转，实现了业务逻辑，但是返回给客户的信息里面还有新请求的 url 同级的一些资源，这时候由于路径问题，而不能显示。重定向即可解决。

2.内部跳转的使用一般要求效率和安全性，我们在开发过程中可以有意识的去

使用，而外部跳转一般是在不得已时才使用。

第五章 状态跟踪

1、http 协议是一种无状态协议，当用户发起一个请求，接收到 response 或者将请求转发，原来的请求就结束，也就是协议的一次操作结束。

其优点在于使 response 更灵活，并且不需要关心服务器的运行状态；

其缺点在于不能跟踪客户的信息到服务器，比如用户的用户名和密码，服务器并没有跟踪处理；

在这种状态下产生跟踪的解决方案：浏览器端的 cookie 和服务器端的 session。

2、cookie：浏览器的一个实现机制，主要用于在浏览器端(客户端)保存一些操作的信息。

1）在浏览器端，cookie 的存在形式是以数组的形式存在的。

2）每一个 cookie 是以键值对的形式存在的。

ex: userName xuxiang

session <---> jsessionid sessionid

3）拿到每一个 cookie，遍历整个 cookies[]

cookie.getName() ---> userName

cookie.getValue() ---> xuxiang

4）在服务器端获取 cookie

request.getCookie() ---> Cookie[]

1.当 cookie 不存在，也就是客户端没有保存 cookie!

```
Cookie cookie = new Cookie(key,value);
response.setCookie();
```

2.如果 cookie 存在，我们直接遍历拿到其值，进行服务器端的操作。

5）cookie 的部分属性

1.setMaxAge(int num); ---> 声明 cookie 的有效期，num 单位为秒

num<0: 表示在客户端不保存，但是在关闭浏览器后才丢失。

num=0: 删除重复的 cookie

num>0: 永久保存，也就是 cookie 中的数据保存的时间长度

2.setPath("/"): 表示只要发送到 server 的请求中的 cookie 都可以被 server 使用。

注意：cookie 一般位于我们请求的头部。

com.briup.status.CookieServlet

服务器处理 cookie 的一般思路：

1.先从客户端拿。要是能拿到，遍历显示其名字和内容。

2.要是拿到的值为空，就去创建，然后保存在客户端。

6）cookie 的缺陷：

1.cookie 中数据的安全性，极度不安全；

2.cookie 可以被禁用，此时所有的跟踪将消失；

3.当客户端 cookie 禁用，我们不能存储 cookie，即不能向服务器端发送 cookie 也不能从服务器端发送 cookie 到客户端。

练习一：使用 Cookie 实现各个用户的访问技术：

1）获取 cookie 数组中专门用于统计用户访问次数的 cookie 的值；

2）将值转换成 int 型；

3）将值加 1 并用原来的名称重新创建一个 Cookie 对象；

4）重新设置最大时效；

5）将新的 cookie 值输出；

实现过程：ip ---> new Cookie(ip count);

cookies[]--->遍历--->以 ip 为 key 的 cookie 存在，提取 value，进行++操作--->重新创建 key 相同的 cookie，设定生命值--->response 给 client

cookies[]--->遍历--->以 ip 为 key 的 cookie 不存在--->new Cookie(ip,count=1)--->response 给客户端

上两步实现了客户访问时明确得到客户的访问次数信息，但是一旦客户不去访问，服务器端就没有数据，即数据没有处理完整，造成丢失。于是可构建一个 Map 来存储此类信息，当下次客户的信息再次更新的时候，同时更新 Map 的信息。

练习二：使用 Cookie 识别访问者：

第一次访问，显示通用问候信息并提交文本输入框输入姓名和性别，以供创建 Cookie;

以后访问，显示个性化问候信息，如：欢迎张小明先生，不提供文本输入框；

1）调用 HttpServletRequest.getCookie()获取 Cookie 数组；

2）在循环中检索指定名字的 cookie 是否存在以及对对应的值是否正确；

3）如果是则退出循环并设置区别标识；

4）根据区别标识判断用户是否为初访者从而进行不同的操作；

3、HttpSession

1）Session：中文翻译为会话，如打电话。但是在我们牵扯到网络编程的技术里的会话已经不去考虑状态和连续性。

2）HttpSession：一种服务器端实现机制。将客户和服务器交互的信息维持在一个对象 HttpSession 里面，并且保存在服务器端。

存储机制：使用类似于散列表的数据结构来存储，同时会在客户端的 cookie 里面维持一个记录信息(在 cookie 里面维持，存储 session 的 id 到一个名字为 JSESSIONID 记录(cookie)里)。

如果客户端的 cookie 里面的 session 信息不存在，我们去请求 server，我们还可

以找回已经存在的session，这时候server会重新创建一个session保存在客户端。

3）session的获取：

request.getSession();
request.getSession(boolean)

如果boolean类型为true两个方法一样，表示先从client端发过来的request里面查找jsessionid，看是否有对应的session，要是没有，server会自动创建一个。

如果boolean为false，当server查询不到，会返回一个null对象。

session.set/get/removeAttribute(String key,Object obj);

4）session的生命周期：

1.session生命值的设置：

a.在web.xml中配置：

```
<session-config>
    <!-- 单位为分钟 -->
    <session-timeout>30</session-timeout>
</session-config>
```

b.session.setMaxInactiveInterval(30); //注意：单位为秒

c.注意：如果不设置，默认为30分钟！

2.生命结束：

a.直接调用session.invalidate()方法结束其生命；

b.生命到期；

c.关闭server。

5）session <---> cookie --->JSESSIONID

考虑：如果我禁用cookie,session怎么维持?

oldUrl：指我们原来请求的路径

response.encodeURL(oldURL); ---> 我们就会将jsessionid带回客户端，下次请求的url后面会追加jsessionid

JSESSIONID：指存储在客户端的cookie的名字，值为sessionId

jsessionid：在我们禁用cookie的时候，请求路径所带的jsessionid，以小写表示，与cookie中的key区分。

*.html ---> UserLoginServlet ---Session(userName,passwd)---> SuccessServlet ---link(两个超链接都去访问下一个servlet，区别是一个重写url，一个不重写)---> ListServlet

4、session和cookie的比较：

1）session存储在服务器，cookie存储在客户端；

2）禁用 cookie，cookie 不能操作，而 session 可以使用方法控制；

3）从安全角度讲，一定程度上，session 要比 cookie 安全。

5、servlet 中几个重要对象

1）ServletRequest(请求范围：requestScope)

创建：客户端发起请求的时候，由 web 容器创建；

消失：服务器端向客户端作出响应。

```
request.setAttribute(key,value);
Object value = request.getAttribute(key);
request.removeAttribute(key);
```

注：在进行服务器内部跳转的时候，如果需要传递数据那么一般都会将数据放在 request 对象中进行传递。

2）HttpSession(会话范围：sessionScope)

创建：客户端发起一个新的会话时；

销毁：会话结束

1. 停止 web 应用；

2. 会话超时：如果在一段时间内，客户端没有发起新的请求，并且这段时间已经超过了被认为超时的时间。

```
<session-config>
    <session-timeout>10</session-timeout>
</session-config>
```

3. 调用 HttpSession 对象的 invalidate 方法

```
session.invalidate();
```

```
session.setAttribute(key,value);
Object value = session.getAttribute(key);
session.removeAttribute(key);
```

注：1. session 中一般不宜存放过多数据；

2. 客户状态信息，在整个会话中经常需要使用的数据以及在进行客户端重定向时需要传递数据，那么我们可以将数据放在 session 对象中。

3）ServletContext(应用范围：applicationScope)

创建：web 应用启动的时候

销毁：web 应用停止的时候

ctx.setAttribute(key,value);
Object value = ctx.getAttribute(key);
ctx.removeAttribute(key);

注：这个对象对于一个 web 应用是唯一的，servletContext 对象中一般存放一些所有用户都要使用，并且在整个 web 应用运行过程中一直要使用的数据。

范围大小：requestScope<sessionScope<applicationScope
注意：放在 requestScope 中的数据，对于请求过程中经过的所有的 servlet 共享；放在 sessionScope 中的数据，对于在同一个会话中发起的所有的请求共享；放在 applicationScope 中的数据对所有的会话共享。

第六章 过滤器

1、Filter

1）对于客户端的请求和服务器端的 reponse，我们使用一段程序(filter)，在其传输之前进行检查，对不符合条件的 request 或者 response 进行拦截，阻止其执行。

2）Filter 不是一个 servlet，它是实现了 Filter 接口的 java 类。

```
public void destroy() {
        //销毁 filter，由容器来调用，只调用一次，在容器销毁的时候(即服务器停止的时候)。
}

public void doFilter(ServletRequest arg0, ServletResponse arg1,
        FilterChain arg2) throws IOException, ServletException {
        //filter 业务执行的代码块；
        //FilterChain 需要在此方法中调用；
        //doFilter(req,res)：其目的是传输 request 和 response;
        //在此操作之后的业务实现是在 response 返回的时候执行，而在 request 请求的时候不执行
        //在整个 filter 中会多次被调用。
}

public void init(FilterConfig arg0) throws ServletException {
        //初始化：
        //初始化 filter 到一个实例，保存在 webContraint 中；
        //同时可以初始化一些 filter 配置的参数，配置类型和 Servler 局部参数配
```

置类似；

```
        //整个生命周期中只被调用一次。
    }
```

2、思考：我们能否用 filter 控制我们对 servlet 的调用顺序？比如我们先调用 form 包下的，然后才能调用 status 包下的资源？

3、Wrappers(拦截)

封装一个拦截类：

1）必须继承 HttpServletRequestWrapper/HttpServletResponseWrapper;

2）必须提供一个参数类型为 HttpServletRequest/HttpServletResponse 的构造器；

3）需要定义一个输入或者输出流(CharArrayWriter/CharArrayReader)，用于在缓存中处理数据；

4）重写 response 的 getWriter()方法，其实质是 new PrintWriter(writer);

5）重写 Object 的 toString()，将 writer 流转换为字符串；

至此包装完毕。

new 拦截类(req/res) ---> test.getWriter

第七章 Web Application Life Cycle Events

1、监听器：对特定事件的处理进行监控的实现机制。

启动服务器；
client 发起请求；
server 接收；
处理业务；
response;
close sever;

2、三种级别的监听器：

1）application

2）Session

3）request

3、在实际应用中我们一般会去关心 HttpSession 级别的监听，同时对 request 级别的也需要常关心，application 级别的很少使用。原因是一般在编程的时候尽量使用作用范围小的对象，首选 request，再者去选择 session...

4、事件模型

1）事件源

web 容器

2）事件对象

1.web 应用的启动和停止

ServletContextEvent

2.会话的开始和结束

HttpSessionEvent

3.请求的开始和结束

ServletRequestEvent

4.往应用范围内存放、替换、删除数据

ServletContextAttributeEvent

5.往会话范围内存放、替换、删除数据

HttpSessionBindingEvent

6.往请求范围内存放、替换、删除数据

ServletRequestAttributeEvent

3）监听器

1.ServletContextListener(*)

2.HttpSessionListener(*)

3.ServletRequestListener

4.ServletContextAttributeListener

5.HttpSessionAttributeListener

6.ServletRequestAttributeListener

Servlet 笔记于 2008 年 8 月 8 日全部完成！！！

JSP 学习笔记

==========

主讲：邓远辉

JSP 前言

1、why?

提出：解决 Servlet 输出复杂动态网页内容的不足；

为了解决 PrintWriter.println("");的繁琐而引出的

Servlet = Java(多) + HTML(少) ---> 擅长做流程控制
JSP = HTML(多) + Java(少) ---> 擅长做界面显示

2、what?

如何让网页设计人员以简单的形式调用 Java 代码：

1）元素

1. 脚本元素： let servlet applet

a. 脚本小应用程序 ---> <% %>

注意：<% %>中的内容是 service()方法中的内容

b. 声明元素 ---> <%! %>

注意：<%! %>中的内容是 service()方法以外的内容

c. 表达式元素 ---> <%= %>

注意：<%= %>中的内容一定要能够转化形成字符串，是 out.println()中的内容！

2. 指令元素

a. <%@page contentType="text/html;charset=UTF-8" %>

b. <%@include file="*.html/jsp" %>: 可以实现页面内容重用，在转换阶段进行嵌入，也称作静态包含。

注意：它不能及时响应被包含内容的变化，故称作静态包含！

c. <%@taglib prefix="myTag" uri="http://www.briup.com/myTag" %>

3. 动作元素

a. 包含：<jsp:include page="" />: 在执行阶段进行嵌入，也称作动态包含。

b. 页面跳转：<jsp:forward page=""/>

或 <jsp:forward page=""><jsp:param name="" value=""></jsp:forward>

c. 传参：<jsp:param name="" value=""/>

或 <jsp:params>
{<jsp:param name="paramName" value="paramValue"/>}+
</jsp:params>

<jsp:useBean id="" class="" type="" scope=""/>
<jsp:setProperty property="*" name=""/>
<jsp:getProperty name="" property="" />

4. EL 元素

```
${?.?}
${?["?"]}
```

2）标记

1.自定义标记

a.标记处理类；

b.标记库描述文件；

c.JSP 文件；

2.JSTL

```
a.<c:set var="" value="" scope=""/>
b.<c:forEach var="" items="">
  </c:forEach>
```

3、JSP 的基本语法：

1）2 种注释类型；

2）3 个脚本元素；

3）3 个指令元素；

4）8 个动作元素。

第一章 Introduction to JSP

1、JSP：作为在服务器端显示信息的页面，它属于 JavaEE 中的一种规范。目前最新版本为 jsp2.0。

jsp 的工作原理：<比 serlvet 多了翻译阶段>

1）翻译阶段：当容器接受到客户端的请求后，先判断被请求的 jsp 是否存在，若不存在返回 404 错误。若存在，则先将该 jsp 翻译成一个 Servlet;

2）编译阶段：将 Servlet 的源文件编译成一个字节码文件，然后由容器将其装载进虚拟机并构造出一个实例对象(单例)；

3）运行阶段：按照 Servlet 生命周期(init(),service 方法,destory())的过程来运行 Servlet.

补充:

1）如果被请求的 jsp 已经有对应的 servlet 实例对象，那么直接运行该 servlet.

2）如果 jsp 源文件的内容发生变化，那么要重新将 jsp 翻译成 Servlet，再编译、装载和实例化对象，最后运行 servlet.

2、 静态网页：每一个人访问的页面内容都是相同的。

为了让网页动起来，SUN 公司推出了 Applet, Java

动态网页：同一个页面，不同人不同时间不同动作有不同内容。

1）客户端动态网页：—— JavaScript 技术

没有发出请求，页面会响应键盘、鼠标的动作；

2）服务器端动态网页：—— Servlet, JSP, EJB 等技术 ---> JavaEE 的三大组件

发现请求，由运行在服务器端的程序处理键盘、鼠标的动作；

3、JavaBean:

1）属性：封装数据；

2）通过 setter 以及 getter 方法对属性值进行访问：简化对属性的访问；

3）必须要有无参的构造方法：简化实例化；

4）实现接口：java.io.Serializable(序列化) ---> 非常方便的存、取属性值

4、MVC: Model View Control

MVC Web 应用

V(view—视图) ---> JSP(显示)

1. 提供界面供用户输入数据，提交请求；

2. 提供界面显示请求处理结果；

C(Controllor—控制器) ---> Servlet(控制和转发请求)

流程控制

M(Model—模型) ---> JavaBeans(其实就是一个 POJO 类)

业务逻辑处理

第二章 Creating,Deploying and Executing a JSP

1、创建一个 Web 工程

--Web Project: JSP

--WebRoot

--banner.html, helloWord.jsp

2、部署一个 JSP

/opt/java/server/jakata-tomcat-5.0.18

|

—— webapps

|

—— Root

/home/briup/workspace/JSP

|

—— WebRoot

|

—— banner.html, helloWord.jsp

第三章 JSP Scriplets,Directives and Action

1、 脚本小应用程序：JSP Scriplets
jsp-->extends HttpJspBase-->extends HttpServlet

<%="Hello World"%> --- > out.ptintln("Hello World");
<%=new java.util.Date()%> ---> out.println(new java.util.Date());
<%="abc"%> ---> out.println("abc");

number.jsp
(int)(Math.random()*100)+(int)(Math.random()*100)=?

add.jsp?n1=10&n2=20
10+20=30

<%System.out.println("abc");%> ---> System.out.println("abc");

counter.jsp
<% response.setContentType("text/html";charset="gb2312")) %>
您是第 个访问者!

2、 指令元素：
Page Directive：建议放在第一行，但不是必须的!

1）Which scripting language is used in JSP page
<%@page language="java"%><%--only java is permitted--%>

2）Super class of the servlet which results from the JSP page
<%@page extends="className"%><%--not recommended strongly--%>

3）Which classes are imported
<%@page import="java.util.*"%>
注意：只有 impor 可以多次重复出现，中间用空格隔开
如：<%@page import="java.util.*" import="java.io.*"%>
或 <%@page import="java.util.*,java.io.*"%>

4）whether participating in an (HTTP) session
<%@page session="true|false"%><%--"true" is default--%>

5）The buffering model for the initial out JspWriter to handle content output from the page
<%@page buffer="none|xxkb"%> <%--8kb is default--%>

6）Whether the buffered content should be flushed automatically
<%@page autoFlush="true|false"%><%--"true" is default--%>

7）How multithread is handled (deprecated)
<%@page isThreadSafe="true|false"%> <%--"true" is default--%>
false==SingleThreadModel

8）The returned content of Servlet.getServletInfo() method
<%@page info="Your first JSP page"%>

例如：<%@page info="author: briup"%>

9）Being intended to be the URL target of another JSP pages's errorPage
<%@page isErrorPage="true|false"%><%--"false" is default--%>

10）The URL to a resource which processes error
<%@page errorPage="error.jsp"%>

11）Which MIME type is generated
<%@page contentType="text/html;charset=GBK"%>

12）Which character encoding used for the JSP page
<%@page pageEncoding="GBK"%>

13）Whether EL expressions are ignored
<%@page isELIgnored="true|false"%>
<%--The default value varies depending on the web.xml version--%>

3、练习：输出九九乘法表

4、多态的三种表现形式：

1）父类 引用=new 子类();
接口 引用=new 实现类();

2）方法声明：public void method(父类 参数){};
方法调用：method(new 子类());

3）public 父类 method(){
return new 子类();
}

5、http://localhost:8080/jsp/ <==> http://localhost:8080/jsp/index.jsp
--index.jsp

6、注

<%-- 服务器端注释，不会发送到客户端 --%>
<!-- 客户端注释，会发送到客户端，但不会显示 -->

第四章 Working with Reusable Components(重用的组件)

1、组件：能实现特定功能的一至多个 API 的组合；

2、在 JSP 中使用 JavaBean:

语法：<jsp:useBean id="beanName(对象的引用)" class="javaBean 的类" type="javaBean 所声明的类型" beanName="beanName" scope="page|request|session|application"/>

或 <jsp:useBean id="beanName(对象的引用)" class="javaBean的类" type="javaBean所声明的类型" beanName="beanName" scope="page|request|session|application">
<jsp:setProperty name="beanName" property="*"/>
</jsp:useBean>

注意： 1. type没有的话则与class的取值相同!

2. <!-- t.setName(request.getParameter("name")) -->
<jsp:setProperty name="t" property="name"/>

<!-- t.setName(request.getParameter("n")) -->
<jsp:setProperty name="t" property="name" param="n"/>

<!-- t.setName("jd0806") -->
<jsp:setProperty name="t" property="name" value="jd0806"/>

<!-- t.setName(n) -->
<% String n="briup"; %> <!-- 注意：这里定义的是局部变量，若要定义实例变量，则用<%! %> -->
<jsp:setProperty name="t" property="name" value="<%=n%>"/>

<!-- t.set*(): 设置所有属性值 -->
<jsp:setProperty property="*" name="t"/>

1)
```
<jsp:useBean id="t" class="Teacher" scope="session"/>
Teacher t = session.getAttribute("t");
if(t==null){
        t = new Teacher();
        sesion.setAttribute("t",t);
}
```

2)
```
<jsp:useBean id="user" class="com.briup.User" scope="request"/>
com.briup.User user = (com.briup.User)request.getAttribute("user")
if(user==null){
        user=new com.briup.User();
        request.setAttribute("user",user);
}
```

3)
```
<jsp:useBean id="m" class="java.util.HashMap" type="java.util.Map" scope="session"/>
java.util.Map m = (java.util.Map)request.getAttribute("m");
if(user==null){
        m=new java.util.HashMap();
        session.setAttribute("m",m);
}
```

3、Class c = Class.forName("com.briup.Student").newInstance();
new Student();

第五章 Handling Exceptions(异常的处理)

1、异常处理

<%@page errorPage="errorpage.jsp" %> ---> 位于可能发生异常的页面

<%@page isErrorPage="true" %> ---> 位于处理异常的页面

注意：将 isErrorPage 的属性设为 true 才意味着可以创建一个局部变量 exception

<%=exception.getMessage() %> ---> 获取异常的描述信息

2、在配置文件 web.xml 中处理异常

```
<error-page>
        <error-code>404</error-code>
        <location>/notFound.html</location>
 </error-page>

 <error-page>
        <exception-type>java.lang.NumberFormatException</exception-type>
        <location>/error.jsp</location>
 </error-page>
</web-app>
```

第六章 Advanced Topics(高级主题)

1、隐含对象

1）request

request.getParameter(); ---> String 类型

request.getAttribute(); ---> Object 类型

2）pageContext: page scope

注意：通过获得 page, request, session, application, 可以跨越多个 request 共享对象，方便实例化隐含对象。

3）out: javax.servlet.jsp.JspWriter

类似于 PrintWriter 对象

4）通过 page 可以访问一些当前的方法

<%=page.getServletInfo()%>

2、session中属性对象为多个request所共享：

多个request共享同一个session ，并且共享同一个session中所放置的属性对象；

如：<%@page import="com.briup.Teacher" %>

```
<%
        Teacher t1=new Teacher("zs","zs");
        Teacher t2=new Teacher("ls","ls");

        session.setAttribute("t",t1);
        request.setAttribute("t",t2);
%>
<%=((Teacher)request.getAttribute("t")).getName()%>
```

问题：请问页面上将输出什么内容？

答案：ls

3、--WebPoject: estore_jsp

```
<from action="addline.do">                          addLine()
listBookStore.jsp ------------------>   controller.jsp

        BookStore.getBook(id)
id------------------------>Book
num------------------------>SalesLine

<from action="dropLine.do">                                 dropLine()
listCart.jsp ----------------------->      controller.jsp
```

listBookStore.jsp ---> addLine.do ---> controller.jsp ---> request.gtServletPath() ---> .do ---> cart.addLine()

request.getServletPath();

servlet中配置路径请求仅有的三种形式：

```
<servlet-mapping>
        <url-pattern>/addLine.do</url-pattern>
        <url-pattern>/*</url-pattern>
        <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

与数据库交互：

```
public class CustomerDaoImp implements CustomerDao{
        boolean login(String name,String password){
                Query query = session.creatQuery("form Customer name:"name");
                query.setString("name",name);
        }
        boolean register(Customer customer){
                session.save(customer);
```

```
        }
    }
```

4、Map 遍历的形式：

1）
```
Set ks = map.keySet();
Iterator it = ks.iterator();
while(it.hasNext()){
        Object key = it.next();
        Object value = map.get(key);
}
```

2）
```
Collection values = map.values();
Iterator it = values.iterator();
while(it.hasNext()){
        Object value = it.next();
}
```

3）
```
Set<Map.Entry> entrySet = map.entrySet();
for(Map.Entry entry:entrySet){
        Object value = entry.getValue();
}
```

5、TreeMap 的排序：

1）自然排序

key 对象实现接口 java.lang.Comparable

"1" --> "10" --> "2" --> "3" --> .....

2）人工排序

单独定义类型实现接口 java.util.Comparator

```
public int compare(Object x, Object y){
        0:      x==y
        >0:     x>y
        <0: x<y
}

private Map<String,Book> books = new TreeMap<String,Book>(new
Comparator(){
        public int compare(Object x, Object y) {
                String strX = (String)x;  // "1" "10"
                String strY = (String)y;

                int i = Integer.parseInt(strX);
                int j = Integer.parseInt(strY);

                return i-j;
        }
```

});

6、JSP 的 10 种内置对象：

1）Request：请求对象；
2）Response：响应对象；
3）Session：会话对象；
4）Application：应用程序对象；
5）Out：输出对象；
6）Cookie：客户端对象；
7）Config：配置对象；
8）Page：页面对象；
9）PageContext：页面上下文对象；
10）Exception：异常对象。

第七章 Custom Tags(自定义标记)

1、自定义标记的组成：

1）标记处理类：功能很强大 ---> Java 程序员(必须继承或实现某个特定的类)
2）JSP 文件： 使用很简单 ---> 网页设计人员
3）标记库描述文件：*.tld ---> 牵线搭桥

2、自定义标记实现的功能：

1）简单的向浏览器输出固定字符串；
2）通过获得属性值实现个性化输出；
3）循环显示本体内容；
4）处理本体内容(大写转换，小写转换，内容替换等)；
5）定义变量(往某个作用范围放置属性对象)；

3、标记库描述文件

<%@taglib prefix="" uri="" %>
uri: 识别 整个应用中 不同的标记库描述文件
prefix: 识别 当前 jsp 页面 所引入的标记库描述文件

--WebProject: jsp_jd0806
--Package: tag
--HelloWorld.java, PrintTable.java, IterationShow.java

4、 源地址 Java 程序 目标地址

字节流：

硬盘	FileInputStream	FileOututStream	硬盘
键盘	System.in	System.out	控制台
内存	ByteArrayInputStream	ByteArrayOutputStream	内存

字符流：

硬盘	FileReader new InputReader(System.in)	FileWriter System.out
键盘	CharArrayReader	CharArrayWriter
内存	StringReader	StringWtriter

5、处理本体内容

1）截取字符串

字符串：本质上是字符形成的数组；

```
String.substring(10);
String.substring(0,?);
String.indexOf("</h3>");

String.substring(0,String.indexOf("</h3>"));

String s = "abc";
s.substring(0,2) ---> "ab"
```

2）追加字符内容

第八章 EL 元素(Expression Language)

1、EL 元素

1）用于简化属性值，变量值的访问；

2）语法：${变量名称.属性名称}

${变量名称[属性名称]}

```
<%
    Student stu = new Student("zs","zs");
    request.setAttribute("s",stu);
%>
((Student)request.getAttribute("s")).getName()
${s.name}          <==>  ${s["name"]}
```

--elTest.jsp

2、page 隐含对象

```
public void _jspService(){
    PageContext pageContext;
    Object page = this; //HttpJspBase
```

}

第九章 Java Server Page Tag Libraries(JSTL)

1、JSTL：SUN 公司制定的一套标准的标记库

2、具体用法：

一. <c:out>

作用：显示数据的内容，就像是<%=scripting-language%>一样;

语法：1) 没有本体(body)内容

<c:out value="value" [escapeXml="{true|false}"] [default="defaultValue"]/>

2) 有本体(body)内容

<c:out value="value" [escapeXml="{true|false}"]>

default value

</c:out>

属性：1) value: 需要显示出来的值；

2) default: 如果 value 的值为 null, 则显示 default 的值;

3) escapeXml: 是否转换特殊字符, 如：< 转换成 , 默认情况下其值为 true;

Null 和错误处理：

假若 value 为 null, 会显示 default 的值，假若没有设定 default 的值，则会显示一个空的字符串;

范例：<c:out value="Hello JSP 2.0!!"/> <!--在网页上显示 Hello JSP2.0!!-->

<c:out value="${3+5}"/> <!--在网页上显示 8;-->

<c:out value="${param.data}" default="No Data"/>

<c:out value="<p>有特殊字符</p>"/>

<c:out value="<p>有特殊字符</p>" escapeXml="false"/>

二. <c:set>

作用：将变量储存至 JSP 范围中或是 JavaBean 的属性中;

语法：1) 在 value 的值储存至范围为 scope 的 varName 变量之中

<c:set value="value" var="varName" [scope="{page|request|session|application}"]/>

2) 在本体内容的值储存至范围为 scope 的 varName 变量之中

<c:set var="varName" [scope="{page|request|session|application}"]>

...本体内容

</c:set>

3) 将 value 的值储存至 target 对象的属性中

<c:set value="value" target="target" property="propertyName"/>

4) 将本体内容的数据储存至 target 对象的属性中

<c:set target="target" property="propertyName">

...本体内容

</c:set>

属性：1) value: 要被储存的值；

2) var ：欲存入的变量名称; 不能使用表达式来表示;

3) scope: var 变量的 JSP 范围，默认值是 page; 不能使用表达式来表示;

4) target:为一 JavaBean 或 java.util.Map 对象;

5) property: 指定 target 对象的属性;

范例：<c:set var="number" scope="request" value="${1+1}"/>

<c:set var="number" scope="session">

${3+5}

</c:set>

<c:set var="number" scope="request" value="${param.number}"/>

<c:set target="${user}" property="name" value="${param.Username}"/>

三. <c:remove>

作用：移除变量;

语法：<c:remove var="varName" [scope="{page|request|session|application}"]/>

属性：1) var ：欲移除的变量名称; 不能使用表达式来表示;

2) scope: var 变量的 JSP 范围，默认值是 page; 不能使用表达式来表示;

四. <c:catch>

作用：处理产生错误的异常状况，并且将错误信息储存至 varName 变量中。

语法：<c:catch [var="varName"]>

...欲抓取错误的部分

</c:catch>

属性：1) var: 用来储存错误信息的变量;

五. <c:if>

作用：和一般在程序中使用的 if 一样。

语法：1) 没有本体内容(body)

<c:if test="testCondition" var="varName" [scope="{page|request|session|application}"]/>

2) 有本体内容(body)

<c:if test="testCondition" [var="varName"] [scope="{page|request|session|application}"]>
本体内容
</c:if>

属性：1) test: 如果表达式的结果为 true, 则执行本体内容，false 相反
2) var : 用来储存 test 运算后的结果，即 true 或 false;
3) scope: var 变量的 JSP 范围

六. <c:forEach>

作用：循环控制，它可以将集合中的成员循序浏览一遍， 当条件符合时，就会持续重复执行<c:forEach>的本体内容。

语法：1) 迭代集合对象之所有成员

<c:forEach [var="varName"] items="collection" [varStatus="varStatusName"] [begin="begin"] [end="end"] [step="step"]>
本体内容
</c:forEach>

2) 迭代指定的次数

<c:forEach [var="varName"] [varStatus="varStatusName"] begin="begin" end="end" [step="step"]>
本体内容
</c:forEach>

属性：1) var : 用来存放现在指到的成员
2) items : 被迭代的集合对象
3) varStatus: 用来存放现在指到的相关成员信息, 它有四个属性：
index: 现在指到成员的索引；
count: 总共指到成员的总数；
first: 现在指到的成员是否为第一个成员；
last: 现在指到的成员是否为最后一个成员；
(见 c_forEach5.jsp)
4) begin: 开始的位置;
5) end : 结束的位置;
6) step : 每次迭代的间隔数;

注意：1) varName 的范围只存在<c:forEach>...</c:forEach>的本体中，如果超出了本体，则不能再取得 varName 的值。

<c:forEach items=${atts}" var="item">
</c:forEach>

${item}

${item}则不会显示 item 的内容

演示一下：<c:forEach>配套 List, Set 以及 Map 时使用的情景。

七. <c:import>

作用：把其它静态或动态文件包含至本身 JSP 网页。和<jsp:include>相比，它能包含不同 web application 或者是其他网站的文件。

语法：1) <c:import url="url" [context="context"] [var="varName"] [scope="{page|request|session|application}"]
[charEncoding="charEncoding"]>
本体内容
</c:import>

2) <c:import url="url" [context="context"] varReader="varReaderName" [charEncoding="charEncoding"]>
本体内容
</c:import>

属性：1) url: 一文件被包含的地址;

2) context: 相同 container 下，其他 web 站点必须以"/"开头;

3) var : 储存被包含的文件的内容(以 String 类型存入)。
当 var 属性存在的时候，被包含的文件内容导入后不会输出在网页上，而是以 String 的形式

4) scope : var 变量的 JSP 范围;

5) charEncoding: 被包含文件之内容的编码格式;

6) varReader : 储存被包含的文件的内容(以 Reader 类型存入)

示例: 1) <c:import url="http://java.sun.com"/>

2) <c:import url="ftp://202.168.0.202/data.txt"/> <!--支持 ftp 协议-->

3) <c:import url="Hello.jsp"/>

4) <c:import url="images/hello.txt"/>

5) <c:import url="/jsp/index.html" context="/others"/>
导入同一个服务器下 others 应用下的 jsp/index.html 文件内容,
注意：
a. 被包含的 web 站点必须在 server.xml 中定义过;
b. <context>的 crossContext 属性值必须为 true。这样，others 应用下的文件
才可以被其他 web 站点调用。

...
<context path="/others" docBase="others" crossContext="true" reloadable="true"/>
...

八. <c:url>

作用：用来产生 URL, 以前我们必须使用相对地址或是绝对地址去取得需要的图文件或文件，现在我们可以直接利用<c:url>从 Web 站点的角度来设定需要的图文件或文件的地址。

语法：1) <c:url value="value" [context="context"] [var="varName"]
[scope="{page|request|session|application}"]/>
2) <c:url value="value" [context="context"] [var="varName"]
[scope="{page|request|session|application}"]>
<c:param>标签
</c:url>

属性：1) value; 执行的 URL;
2) context: 相同 container 下，其他 web 站台必须以"/"开头;
3) var: 储存被包含文件的内容(以 String 类型存入)
4) scope: var 变量的 JSP 范围

示例：1) <c:url value="http://www.javaw.cn">
<c:param name="param" value="value"/>
</c:url>
执行结果产生一个网址为：http://www.javaw.cn?param=value
2) <a href="
<c:url value="http://www.javaw.cn">
<c:param name="param" value="value"/>
</c:url>Java 技术论坛

执行结果产生一个网址为：Java 技术论坛
3) <img src="<c:url value="/images/code.gif"/>"/>
执行结果产生一个网址为：

九. <c:redirect>

作用：将客户端的请求从一个 JSP 网页导向到其他文件

语法：1) 没有本体内容

<c:redirect url="url" [context="context"]/>

2) 本体内容代表查询字符串参数

<c:redirect url="url" [context="context"]>

<c:param>

</c:redirect>

属性：1) url: 导向的目标地址;

2) context: 相同 container 下，其他 web 站点必须以"/"开头

示例：<c:redirect url="http://localhost:8080">

<c:param name="param" value="value"/>

</c:redirect>

十. <fmt:setLocale>

作用：设定用户的语言区地区。

语法：<fmt:setLocale value="local" [variant="variant"]

[scope="{page|request|session|application}"]/>

属性：1) value: 地区代码。其中最少要有两个字母的语言代码，如：zh、en, 然后也可以再加上两个字母的国家和地区代码，如：US, TW, 两者可以由"-"或"_"

相连接起来，例如：zh_TW, zh_CN, en, en_US. 假若 value 为 null 时，则使用执行中的默认地域(locale)

2) varient: 供货商或浏览器的规格，例：WIN 代表 Windows, Mac 代表 Macintosh.

3) scope: 地区设定的套用范围;

十一. <fmt:requestEncoding>

作用：设定字符串编码，功能和 ServletRequest.setCharacterEncoding()相同;

语法: <fmt:requestEncoding [value="charseName"]/>

属性：1) value: 字符串编码,当我们设定 value 为 GB2312 时，表示将所有传送过来的字符串皆用 GB2312 编码处理，如果没有设定 value 属性，则它将会自动去寻找

合适的编码方式。

十二. <fmt:message>

作用：从指定的资源中把特定关键字中的值抓取出来;

语法：1) <fmt:message key="messageKey" [bundle="resourceBundle"]

[var="varName"] [scope="{page|request|session|application}"]/>

2) <fmt:message key="meesageKey" [bundle="resourceBundle"]

[var="varName"] [scope="{page|request|session|application}"]>

<fmt:param>
</fmt:message>
3) <fmt:message key="meesageKey" [bundle="resourceBundle"]
[var="varName"] [scope="{page|request|session|application}"]>
索引
(<fmt:param>标签)
</fmt:message>

属性： 1) key: 索引;
2) bundle: 使用的数据来源;
3) var: 用来储存国际化信息;
4) scope: var 变量的 JSP 范围;

十三. <fmt:param>

作用：动态设定数据来源中抓出内容的参数；

语法：1) 通过 value 属性设定参数值
<fmt:param value="messageParameter"/>
2) 通过本体内容设定参数值
<fmt:param>
本体内容
</fmt:param>

示例：1) Str3=today is {0, date}
其中{0,date}表示为一个动态变量, 0 代表第一个动态变量；date 代表该动态变量的类型。

十四. <fmt:bundle>

作用：设定本体内容的数据来源;

语法：<fmt:bundle basename="basename" [prefix="prefix"]>
本体内容
</fmt:bundle>

属性：1) basename: 要使用的资源名称，如果我们的 properties 文件为 Resource.properties, 那么
basename 的值为 Resource. basename 的值千万不可有任何文件类型。
2) prefix: 设定前置关键字。
例：当 properties 中内容如下：
requestinfo.label.method=Method:

requestinfo.label.requesturi=Request URI:

那么我们可以写成如下方式将其内容显示出来：

<fmt:bundle basename="Resource" prefix="requestinfo.label.">
<fmt:message key="method"/>
<fmt:message key="requesturi"/>
</fmt:bundle>

十五.<fmt:setBundle>

作用：设定默认的数据来源，或者也可以将它设定到属性范围中.

语法：<fmt:setBundle basename="basename" [var="varName"]
[scope="{page|request|session|application}"] />

属性：1) basename: 要使用的资源名称;
2) var : 储存资源的名称;
3) scope : var 变量的 JSP 范围;

说明：basename 设定要使用的数据源， 和<fmt:bundle>用法相同。如果没有设定 var 时，那么设定好的数据来源

将会变成默认的数据来源，让在同一网页或同一属性范围中<fmt:message>都可以直接默认使用此数据来

源。相反，如果设定 var 时，那么将会把此数据来源存入 varName 中，当<fmt:message>要使用时，必须使

用 bundle 这个属性来指定。如：

<fmt:setBundle baseName="Resource" var="resource" scope="session"/>
<fmt:message key="str1" bundle="${resouce}"/>

如没有设定 var 时，则只须写成如下：

<fmt:setBundle baseName="Resource"/>
<fmt:message key="str1"/>

一般<fmt:bundle>和<fmt:setBundle>都可以搭配<fmt:setLocale>使用，当我们有多种语言的数据业来

源时，可以将文件名取成 Resource_zh_TW.properties, Resource_en.properties 和 Resouce.properties

当我们将区域设定为 zh_TW, 那么使用<fmt:setBundle>或<fmt:bundle>时，将会默认读取 Resource_zh_TW.properties 资源文件，如果<fmt:setLocale>设定为 en 时，那么会默认抓取 Resource_en.properties 来使用，最后如果设定的区域没有符合的文件名，将使用 Resource.properties 来当做数据来源, 见 fmt_i18n.jsp.

十六.<fmt:formatNumber>

作用：依据设定的区域将数字改为适当的格式。

语法：<fmt:formatNumber value="numbericValue" [type="{number|currency|percent}"]

示例：<fmt:formatNumber value="123" type="currency"/> 显示为 $123

十七.<fmt:parseNumber>

作用：将字符串类型的数字、货币或百分比，都转为数字类型。

语法：<fmt:parseNumber value="numericValue" [type="{number|currency|percent}"]
[pattern="customerPattern"]
.../>

示例：<fmt:parseNumber value="500,800"/>显示为: 500800
<fmt:parseNumber value="$5000" type="currency"/> 显示为：5000

十八.<fmt:formatDate>

作用：格式化日期，时间

语法：<fmt:formatDate value="date" [type="{time|date|both}"]
[dateStyle="{default|short|medium|long|full}"]
[timeStyle="{default|short|medium|long|full}"]
/>

示例：<jsp:useBean id="now" class="java.util.Date"/>
<fmt:formatDate value="${now}"/>

十九.<fmt:parseDate>

作用：将字符串类型的时间或日期都转为日期时间类型；

语法：<fmt:parseDate value="date"

示例：<fmt:parseDate value="2003/2/17"/>

二十. <fmt:setTimeZone>

作用：设定默认时区或是将时区储存至属性范围中,方便以后使用

语法：<fmt:setTimeZone value="timeZone" [var="varName"]
[scope="{page|request|session|application}"]/>

属性：1) value: 使用的时区：可以为 EST, CST, MST 和 PST，为 null 时，则默认为 GMT 时区。

示例：<fmt:setTimeZone value="PST" scope="session"/>

二十一.<fmt:timeZone>

作用：设定暂时的时区;

语法：<fmt:timeZone value="timeZone">
本体内容
</fmt:timeZone>
示例：<fmt:timeZone value="PST">
<fmt:formatDate.../>
<fmt:formatDate.../>
...
</fmt:timeZone>

二十二.<sql:setDataSource>
作用：用来设定数据来源(DataSource)
语法：1) 直接使用已存在的数据来源
<sql:setDataSource dataSource="dataSource"
[var="varName"]
[scope="{page|request|session|application}"]/>
2) 使用 JDBC 方式，建立数据库联机
<sql:setDataSource url="jdbcUrl"
driver="driverClassName"
user="userName"
password="password"
[var="varName"]
[scope="{page|request|session|application}"]/>
示例：1) <sql:setDataSource dataSource="java:comp/env/jdbc/mysql"/>
2) <sql:setDataSource
dataSource="jdbc:mysql://localhost/test,com.mysql.jdbc.Driver,root,"/>
dataSource="url,driver,user,password"
3) <sql:setDataSource url="jdbc:mysql://localhost/test"
driver="com.mysql.jdbc.Driver"
user="root"
password=""/>

二十三.<sql:query>
作用：查询数据库的数据.
语法：1) 没有本体内容
<sql:query sql="sqlQuery" var="varName"
[scope="{page|request|session|application}"]

[dataSource="dataSource"]
[maxRows="maxRows"]
[startRow="startRow"]/>

2) 本体内容为查询指令

<sql:query sql="sqlQuery" var="varName"
[scope="{page|request|session|application}"]
[dataSource="dataSource"]
[maxRows="maxRows"]
[startRow="startRow"]>
...
sqlQuery
...
</sql>

属性：1) sql: SQL 语句(select);
2) dataSource: 数据来源;
3) maxRows: 设定最多可暂存的数据笔数;
4) startRow: 设定数据从第几笔开始, 以 0 为第一笔数据;
5) var: 储存查询结果, 必须的;
6) scope: var 变量的 JSP 范围;

查询结果存放在指定名称的属性中后，可以通过以下属性访问查询结果：
rows: 以字段名称当做索引的查询结果；
rowsByIndex: 以数字当作索引的查询结果；
columnNames: 字段名称;
rowCount: 查询到的数据笔数;
limitedByMaxRows: 取出最大数据笔数的限制;

二十四.<sql:update>

作用：执行修改操作(update,delete,create table 均可)

语法：1)没有本体内容

<sql:update sql="sqlUpdate" [var="varName"]
[scope="{page|request|session|application}"]
[dataSource="dataSource"]/>

2)本体内容为查询指令

<sql:update sql="sqlUpdate" [var="varName"]
[scope="{page|request|session|application}"]

[dataSource="dataSource"]>
...
sqlUpdate
...
</sql:query>

属性：1) sql: SQL 语法(update,insert,delete...)
2) dataSource: 数据来源;
3) var: 储存改变的数据笔数;
4) scope: var 变量的 JSP 范围;

二十五. <sql:transaction>

作用：提供事务支持，保证多个数据操作的完整性. <sql:transaction>主要是将所有必须同时执行的交易放在它的本体内容中，当本体内容有错误发生时，将
不会执行任何一个 SQL 语句，所以可保障交易机制的安全性。

语法：<sql:transaction [dataSource="dataSource"]
[isolation="{read_committed|read_uncommitted|repeatable|serializable}"]>
<sql:query>or<sql:update>
</sql:transaction>

二十六. <x:parse>

作用：解析 xml 文件;

语法：1) <x:parse doc="XMLDocument"
[var="var' [scope="{page|request|session|application}"]
.../>
2) <x:parse [var="var' [scope="{page|request|session|application}"]
...>
欲被解析的 XML 文件
</x:parse>

属性：1) doc: xml 文件;
2) var: 储存解析后的 XML 文件
3) scope: var 变量的 JSP 范围

示例：1) <c:import var="sample" url="http://www.javaw.cn/web.xml"/>
<x:parse doc="${sample}" var="sampleXml"/>
2) <x:parse doc="${sample}" var="sampleXml">
<c:import url="http://www.javaw.cn/web.xml"/>
</x:parse>

二十七.<x:out>

作用：取出 XML 中的字符串.

语法：<x:out select="XPathExpression" [escapeXml="{true|false}"]/>

属性：1) select: XPath 语句;

2) escapeXml: 是否转换特殊字符;

示例：<x:parse var="sampleXml">

```
<item>
    <name>car</name>
    <price>10000</price>
</item>
```

</x:parse>

显示<name>的内容,

1) <x:out select="$sampleXml//name"/>

2) <x:out select="$sampleXml/item/name"/>

二十八.<x:set>

作用：取出 XML 中的字符串, 储存至属性范围中

语法：<x:set select="XPathExpression" var="var" [scope="{page|request|session|application}"]/>

属性：1) select: XPath 语句;

2) var: 将从 XML 文件中取得的内容储存至 varName 中;

3) scope: var 变量的 JSP 范围;

示例：<x:parse var="sampleXml">

```
<item>
    <name>car</name>
    <price>10000</price>
</item>
```

</x:parse>

显示<name>的内容,

1) <x:out select="$sampleXml//name"/>

2) <x:out select="$sampleXml/item/name"/>

三十. fn:contains()

作用：判断某字符串是否在一字符串之中;

语法：${fn:contains(string, substring)} => boolean

属性：1) string: 原输入的字符串;

2) substring: 测试用的字符串;

3) Result: 回传 string 是否包含 substring. 若有，则回传 true; 若无，则回传 false;

三十一. fn:containsIgnoreCase()

作用：判断某字符串是否在已有字符串之中，并忽略其大小写;

语法：${fn:containsIgnoreCase(string, substring)} => boolean

三十二. fn:startsWith(string, prefix)

作用：判断一字符串是否以某一字符串为开头;

语法：${fn:startsWith(string, prefix)} => boolean

三十三. fn:endsWith(string, suffix)

作用：判断一字符串是否以某一字符串为结尾;

语法：${fn:endsWith(string, suffix)} => boolean

三十四. fn:escapeXml(String)

作用：用来转换转义符，例如：将<,>,',"和&转换成<,>,',"和&

语法: ${fn:escapeXml(String) => String

三十五. fn:indexOf(string, substring)

作用：回传某字符串到一字符串第一次吻合的位置;

语法: ${fn:indexOf(string, substring)} => int

三十六. fn:split(string, delimiters)

作用：将字符串分离成一字符串数组;

语法: ${fn:split(string, delimiters)} => string[]

三十七. fn:join(array, separator)

作用：将数组中的全部元素以指定字符串作为连接符，回传结合后的字符串;

语法: ${fn:join(array, separator)} => string

三十八. fn:replace(inputString, beforeSubstring, afterSubstring)

作用：将字符串中的某些子字符串用另一字符串取代;

语法: ${fn:replace(inputString, beforeSubstring, aterSubstring)} => string

属性：1) inputString: 原输入的字符串;

2) beforeString: 要替换的字符串;

3) afterSubstring: 替换成为的字符串;

三十九. fn:trim(string)

作用：去除字符串的前后空白;

语法: ${fn:trim(string)}

四十. fn:substring(string, beginIndex, endIndex)

作用：抽取字符串中的某一子字符串;

语法: ${fn:substring(string, beginIndex, endIndex)} => string

四十一. fn:substringAfter(string, substring)

作用：抽取字符串中某子字符串之后的字符串;

语法：${fn:substringAfter(string, substring)} => string

属性：1) string: 原输入的字符串;

2) substring: 某子字符串;

四十二. fn:substringBefore(string, substring)

作用：抽取字符串中某子字符串之前的字符串;

语法：${fn:substringBefore(string, substring)} => string

属性：1) string: 原输入的字符串;

2) substring: 某子字符串;

四十三. fn:toLowerCase(string)

作用：转换为小写字符;

语法：${fn:toLowerCase(string)} => string

四十四. fn:toUpperCase(string)

作用：转换为大写字符;

语法：${fn:toUpperCase(string)} => string

JSP 笔记于 2008 年 8 月 15 日全部完成！！！

Struts 学习笔记

==========

主讲：沈阅斌

Struts2.0

==========

第一章 MVC Model

1、struts: jsp+servlet

1）获得客户端参数
2）数据类型转换
3）数据格式验证
4）将参数包装成一个 pojo 类型
5）将需要显示的数据传递给 jsp
6）将请求转发给 jsp
7）jsp 显示 servlet 传递过来的数据

2、MVC 模式

1）控制器(controller)

1. 接收客户端请求
2. 调用模型的方法
3. 选择其中一个视图进行响应

2）模型(model)

1. 封装应用的状态信息
2. 提供方法可更改和查询这些状态信息
3. 一旦状态信息被更改,马上通知视图重新显示.

3）视图(view)

1. 是用户跟系统交互的接口
2. 可以发送请求给控制器
3. 一旦接收到模型的通知,马上重新显示模型中的数据.

3、web 应用框架(表现层框架)可以提供的功能：

1）自动获得客户端提交的参数；
2）自动进行数据类型转换和数据格式验证；
3）自动进行数据的包装；

4）能够更方便的将显示的数据传递给视图；

5）自动进行请求的转发；

6）提供标签库,能够更方便的构建视图；

7）国际化(il8n)的实现；

8）能够更方便的跟其他的框架进行集成；

第二章 Struts Basic

1、Struts: 最早的 MVC 开源框架

1）是 apache 组织基于 MVC 模式开发的开源的单点控制的 web 应用框架(framework)；

2）struts2 的核心是 WebWork2，继承了 webwork2 的血统，所以跟 struts1.x 有很大的区别，更类似于 WebWork；

3）struts2 是一个基于 MVC Model 2 的单点控制的 Web 应用框架；

2、使用 struts 的目的

可以快速构建整个软件系统的框架结构，将系统划分为三个核心的模块：

1）Model(模型层)

JavaBean:

1.处理业务逻辑 ---> 属于业务逻辑层

2.处理数据访问逻辑 ---> 属于数据访问层

EJB: SessionBean

ActionForm：由 struts 框架提供的一个类

作用：1)封装表单的字段信息

2)验证表单的字段信息

3)可以对表单中的字段信息重新赋值

2）View(视图层):Jsp Html

3）Controller(控制层)

ActionServlet：核心控制器

RequestProcessor：

Actions：业务逻辑控制器

注意：模型层包含了业务逻辑层和数据访问层；视图层和控制层属于表现层(web 层)。

3、struts 框架的优缺点：

1）优点：

1.可以提高软件系统的开发效率；

2.可以提高软件系统的维护性，扩展性和代码重用性；

3.适合于开发一些比较大的项目，可以降低架构的复杂性；

4.一个老牌的框架，支持的用户群很多；

2）缺点：

1.用户写的程序和 Struts 框架绑定住了，不能脱离框架单独测试运行

2.Struts 框架的运行机制非常复杂，不易于理解。

4、Struts 中组件的功能：

1)控制层的分类

1.ActionServlet：（公司的老总）

由 Struts 框架提供的一个类，用来接受所有客户端的*.do 请求， 但是他本身不会去处理请求，而是将请求委托给 RequestProcessor 全权处理；

2.RequestProcessor：（项目总监）

由 Struts 框架提供的一个类，用来接受 ActionServlet 转发过来的请求并处理请求，它会根据 struts.xml 中的配置调用不同的 Action 去全权处理客户端不同的请求；

3.Action：（程序员）

由 Struts 框架提供的一个类，它的功能和 Servlet 一样作为控制器来控制和转发请求，由它去调用模型层中的业务接口方法，并根据不同的结果选择不同的 jsp 视图显示；

注：我们用户需要自己写一个类继承 Action(跟写 Servlet 一样)。

2）模型层中的 ActionForm：由 struts 框架提供的一个类

作用：a.封装提交表单中的字段信息

b.对提交表单中的字段信息进行验证(服务器端验证)

c.可以对表单中的字段信息重新赋值

注意：1.我们用户也要定义一个类去继承 ActionForm；

2.一个 Action 可以配置一个 ActionForm，也可以不配置 ActionForm；一个 ActionForm 可以被多个 Action 使用。

Http://localhost:8080/struts/basic/register.do

contextPath: /struts_ajd0704

通过 request.contextPath()获得；

servletPath: /basic/register.do

通过 request.getServletPath()获得，如果 path 不正确会出现 404 错误；

5、基于 struts2 的 web 应用的环境搭建：

1) 创建 web 工程，导入 struts2 的 jar 包；

2) 在 web.xml 中配置主控器(FilterDispatcher);

```
<filter>
        <filter-name>struts2</filter-name>
        <filter-class>
                org.apache.struts2.dispatcher.FilterDispatcher
        </filter-class>
</filter>
<filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
</filter-mapping>
```

3) 在 src 下创建 struts.xml, struts.properties 以及 log4j.properties(日志包:将详细的错误信息打印在控制台上)。

6、Struts2 框架的执行流程：

1）用户发起请求(.action 结尾的请求表示请求一个 Action)；

2）struts2 框架的主控器(FilterDispatcher)接收用户发起的请求，然后判断这个请求是交给 Action 处理还是交给 web 组件处理。如果请求的 Action 或者 web 组件不存在,那么出现 404 错误。在整个处理过程中需要一个辅助对象：Action 映射器(ActionMapper)；

3）若第二步判断的结果是交给 Action 处理，并且存在对应的 Action。那么根据 struts.xml 文件中对此 Action 的配置信息，首先执行拦截此 Action 的所有拦截器，然后再执行请求的 Action 对象。在这个处理过程中需要辅助对象：Action 代理(ActionProxy)、配置管理器(ConfigurationManager)、ActionInvocation；

4）Action 执行完毕之后返回一个结果(此结果用字符串表示)，这个结果经过拦截 Action 的所有拦截器之后，返回给主控器。主控器根据此结果从配置文件中找到真正的路径，然后将请求转发给对应的视图；

5）由视图向客户端作出响应；

代理：两个对象实现共同接口或者继承共同父类，调用一个对象的方法等于调用另外一个对象的方法。

7、使用 Struts2 框架的基本步骤：

1）创建 jsp 页面作为视图，提供用户端输入表单的页面，注意下 action 属性的配置；

2）创建 Action 类，提供属性来存放客户端提交的数据，并且提供 getter/setter 方法，注意 property 的名字必须和客户端提交的参数名保持一致；提供 execute 方法来处理请求，使用字符串来表示处理的结果；

3）在 struts.xml 文件中配置 action，包括名字和类型，还需要在 action 元素中为 action 的 execute 方法的返回结果定义对应的 result 元素，主要是为了说明 action 返回的字符串代表的真正结果。

第三章 Action，Result & Struts2 Tag Library

1、Action 类的开发方法：

1）普通的 Java 类，不需要实现任何接口或者继承任何类；

2）实现一个 Action 接口；

优点：可以使用 Action 接口中定义的常量作为一个结果。

3）继承 ActionSupport 类；

优点：可以在自定义的 Action 类中使用 ActionSupport 类中定义的方法。

2、在 Action 类中访问 servlet 中的对象的方式：

1）通过 ActionContext 对象，对于每一个 Action 实例都有一个 ActionContext 对象；

```
ActionContext ac = ActionContext.getContext();
Map session = ac.getSession();
session.put("isLogin",true);
```

2）Action 类实现一些 aware 接口，然后 struts 框架中有一个拦截器自动会调用 aware 接口中的方法传入 servlet 中的对象；

```
ServletContextAware
public void setServletContext(ServletContext sc){}
ServletRequestAware
public void setServletRequest(HttpServletRequest req){}
ServletResponseAware
public void setServletResponse(HttpServletResponse res){}
```

3）通过 ServletActionContext 中定义的静态方法来获得 servlet 中的对象；

3、Action 的配置方式

Action 类中可以提供多个方法来处理多个不同类型的请求：

1）Action 类：提供多个方法(配置 Action 跟以前相同,只需指定 Action 的名字和 Action 的类型)

配置：声明 action 的名字和类型；

访问：必须以 actionName!methodName.action 这种方式访问；

如：

```
public class LoginRegisterAction{
        public String login(){}
        public String register(){}
}
<action name="LoginRegisterAction"
class="...LoginRegisterAction">
</action>
http://localhost:8080/struts2/LoginRegisterAction!login.action
```

2)Action类：提供多个方法(配置的时候要说明当前Action处理请求的方法名)

配置：为每一个处理请求的方法定义一个Action；

访问：actionName.action；

如：
```
public class LoginRegisterAction{
        public String login(){}
        public String register(){}
}
<action name="login" class="...LoginRegisterAction" method="login">
</action>
<action name="register" class="...LoginRegisterAction" method="register">
</action>
http://localhost:8080/struts2/login.action
```

3）Action: 提供多个方法(使用通配符简化第二种方式的配置)

配置：Action的名字采用通配符表示，方法为通配符的取值；

访问：methodNameActionName.action

如：
```
public class LoginRegisterAction{
        public String login(){}
        public String register(){}
}
<action name="*Action" class="...LoginRegisterAction" method="{1}">
</action>
http://localhost:8080/struts2/loginAction.action
```

4、解决访问Action不存在的处理办法：

1）设置默认Action：当访问的Action不存在时交给默认Action处理；

```
<default-action-ref name="actionName">
```

注意：这个action必须放在包中所有action的前面！

2）配置一个名字是一个通配符的action，这个action必须放在所有action后面配置；

```
<action name="*">
    <result>/404.jsp</result>
</action>
```

5、结果的配置：

1）局部的result

```
<result name="" type=""></result>
```

type取值： 1.dispatcher: 处理结果为作服务器内部跳转(跳转到web组件)

2.redirect: 作客户端重定向(重定向到web组件)

3.chain: 作服务器内部跳转(跳转到其他Action)

无 namespace: <result type="chain">login</result>

有 namespace: <result type="chain">
<param name="actionName">dashboard</param>
<param name="namespace">/secure</param>
</result>

4.redirectAction: 作客户端重定向(重定向到其他 Action)

注意：当 result 元素的 name 属性取值为 success 的时候，可以省略；type 属性值为 dispatcher 的时候，也可以省略。

如：<result name="success" type="dispatcher">
<param name="location">/ThankYou.jsp</param>
</result>

简化形式 1：
<result>
<param name="location">/ThankYou.jsp</param>
</result>

简化形式 2：
<result>/ThankYou.jsp</result>

2）全局的 result:
<global-results>
<result name=”error”>/Error.jsp</result>
<result name="success" type="dispatcher">/ThankYou.jsp</result>
</global-results>

6、OGNL：对象图导航语言

1）ognl 访问 ValueStack(值栈:存在于 request 范围)中的数据：

1.访问栈顶元素：存放在值栈中的对象，通过 ognl 表达式只能取栈顶对象；

<s:property/> //取栈顶的元素

2.访问值栈中某个对象的属性：如果是访问属性的话，那么会对值栈中的对象进行查找，直到找到这个属性返回；

<s:property value="属性名" />

注: a.如果有多个元素有相同名字的属性,那么 ognl 只能访问到第一个元素的属性；

b.即如果要访问的对象的属性在前面的对象中已经存在，那么就无法取出需要访问的对象的属性值；

3.访问值栈中维护的 Map 元素中的数据.

<s:property value="key" />

注:如果在 Map 元素前有元素的属性名跟 Map 中存放的数据的 key 相同,那么无法获得 Map 中的数据

2）ognl 访问 ActionContext(存在于 request 范围)中的数据：

1.访问 ActionContext 中维护的 Map 中的数据

<s:property value="#key" />

2.访问 request,session,appliaction,parameters,attr 等对象中存放的数据

<s:property value="#request.key" />
<s:property value="#session.key" />
<s:property value="#application.key" />
<s:property value="#parameters.key" /> //访问请求的参数
<s:property value="#attr.key" /> //从 page 到 application 查找数据

注意：1）只能在 struts2 标签的属性中使用，并且只能访问 ActionContext(存在于 request 范围)和 valueStack(存在于 request 范围)这两个对象中的数据；

2）与“jsp:el 表达式”的区别：

1.可以使用在任何地方；

2.可以访问 requestScope,sessionScope,cookie.....

${requestScope.user.name}
<s:property value="#request.user.name"/>

${person.age}
<s:property value="#attr.person.age"/>

7、struts2 的标签分类：

1）用户界面标签

表单标签和非表单标签

2）非用户界面标签

流程控制标签和数据访问标签

a. Control 标签:

1.if/elseif/else

<s:if test="expression">tag body;</s:if>
<s:elseif test="expression">tag body;</s:elseif>
<s:else>tag body;</s:else>

2.iterator

a.迭代 List/set/array

```
<s:iterator value="list/set/array">
    <s:property/>
</s:iterator>
```

b.迭代 Map

```
<s:iterator value="map">
    <s:property value="key"/>
    :
    <s:property value="value"/>
</s:iterator>
```

可选属性说明：

id：如果指定了 id 属性，那么迭代出来的数据除了放在 ValueStack 中之外，还被放入 ActionContext 中(key 就是 id)；

struts：如果需要获得当前迭代标签的一些状态信息(比如当前迭代出来的数据是第几条数据等)，那么可以指定这个属性，如果指定了这个属性，在 ActionContext 中会放入一个类型为 IteratorStatus 的对象，包含了当前的状态信息。

3.append：把多个集合组装成一个集合

```
<s:append id="newList">
    <s:param value="{'spring','hibernate','ejb'}"/>
    <s:param value="{'kevin','jack','tonny'}"/>
</s:append>

<s:iterator value="#newList">
    <s:property/><br/>
</s:iterator>
```

4.generator：按照某个分割符分割字符串变成一个集合

```
<s:generator sepatator="," val="'a,b,c,e,f'" id="strs">
</s:generator>

<s:iterator value="#attr.strs">
    <s:property/><br/>
</s:iterator>
```

5.merge：把多个集合合并成一个集合，同 append

```
1 2 3  1 2 3
append: 123123
merge: 112233
```

6.subset: 过滤集合中的数据

```
public class MyDecider implements SubsetIteratorFilter.Decider{
    public boolean decide(Object element)throws Exception{
        String str = (String)element;
        return str.indexOf("J2EE") > 0;
```

```
            }
        }

        <s:bean id="mydecider" name="MyDecider"/>
        <s:subset source="{'Java Tutorial','Spring2','Ajax based J2EE',
            'J2EE Enterprise Application'}" decider="#mydecider">
            <s:iterator>
                <s:property/><br/>
            </s:iterator>
        </s:subset>
```

7.sort：对集合中的元素进行排序

```
public class MyComparator implements Comparator{
    public int compare(Object element1, Object element2){
        return ((String)element1).length()-
((String)element2).length();
    }
}

<s:bean id="mycomparator" name="MyComparator"/>
<s:sort source="{'spring','struts2','hibernate','ejb'}"
comparator="#mycomparator">
    <s:iterator>
        <s:property/><br/>
    </s:iterator>
</s:sort>
```

b. Date 标签：

8.property：输出 ognl 表达式的值

```
<s:property value="attrName/#..."/>
```

9.set：将数据设定到某个范围内

```
<s:set name="" value="" scope=""/>
```

scope 取值：application,session,request,page,action(ActionContext), 默认为 session.

10.push：将某个对象放到值栈的栈顶

```
<s:put value="#request.user">
    <s:property/>
</s:put>
```

11.bean：创建指定类型的对象，并且将对象放入 ActionContext 对象中

```
<s:bean id="key" name="类型">
    <s:param name="name" value=""/>
</s:bean>
```

12.date：格式化输出日期类型的数据

<s:date name="日期对象" format="yyyy 年 MM 月 dd 日" nice="false"/>

13.action：在 jsp 页面上去调用 action

<s:action name="helo" namespace="/basic" executeResult="true"/>

14.debug：用于调试，通过这个标签可以查看 ValueStack 和 ActionContext 中存放的数据

<s:debug/>

15.include：包含其他页面，跟 jsp 的 include 动作元素类似

<s:include value="*.jsp"/>

16.param：一般与其他标签结合在一起使用，它不会单独使用

17.url：对一个 url 进行重写，可以在原来的 url 后追加 jsessionid 以及参数

```
<s:url value="delete.action">
    <s:param name="id" value="1"/>
</s:url>
---> delete.action?id=1;jsession=UKLJKUJOJKLJLKJJKJLK
```

18.i18n 和 text：主要是用于国际化的

i18n：读取某一组资源文件

text：输出与当前语言类型对应的资源文件中的内容

c. Form 标签

3）Ajax 标签

8、注意：在 struts2 中 action 如何将数据传递给 jsp 实现?

1）在 action 中将需要传递的数据放在 request 范围内传递过去，在 jsp 页面上从 request 范围内取出数据显示。

2）在 action 中将需要传递的数据作为 action 的一个实例变量(action 的引用会被 struts 框架放入值栈中)，在 jsp 页面上从 ValueStack 中取出数据显示。

第四章 Data Transfer & Type Converter

1、为什么需要类型转换?

b/s 架构的软件，客户端提交的数据都是文本类型的，但是服务器端接收到这些数据之后需要用不同的类型去表示，所以需要将文本数据与 java 类型之间进行转换，原本这些转

换工作要有开发人员通过编码来实现，现在可以交给 struts 框架处理。

1）可以将文本数据转换成 java 中的基本类型数据以及它的包装类型；

做法：需要在 action 中定义需要转换的类型的属性去接收

2）可以将文本数据转换成 java 中的 Date 类型；

3）可以将文本数据转换成数组类型，List 类型、Map 类型；

4）可以将文本数据转换成自定义的对象类型；

做法：在 action 中提供一个自定义对象类型的属性(getter/setter 方法)；在 jsp 页面中，输入框的名字定义为：

action 中的属性名.自定义对象类型的属性名

如：(action)Book book;(注意必须提供无参构造器) =====> (jsp)book.id, book.name, book.price

注意：1.类型转换器会在提交数据和显示数据的时候起作用；

2.如果在进行类型转换的时候出错(比如将"abc"转换成一个 Long 类型的数据)，struts 框架就不会再去执行 Action，而是直接返回一个逻辑名为"input"的结果给主控器；

2、自定义转换器的开发：

1）开发自定义转换器的原因：由于 struts 框架中提供的类型转换器是一些基本的类型转换器，对于一些特殊情况无法通过 struts2 框架内置的类型转换器进行转换。

2）自定义转换器的开发步骤：

1.创建一个转换器类，需要实现 TypeConverter 接口，一般会去继承 StrutsTypeConverter 这个抽象类实现两个方法：

convertFromString:

作用：将客户端提交的参数转换成一个 java 类型的对象，这个方法会在客户端提交数据的时候起作用；

convertToString:

作用：将 Action 中的属性值转换成一个字符串，这个方法会在通过 Ognl 表达式输出数据的时候起作用；

2.注册转换器类：

a.在 src 下创建 xwork-conversion.properties

并且声明：需要转换的对象类型=转换器类型

注意：被 struts2 框架自动读取，在这个文件中注册的转换器对整个 web 应用生效；

b.在与 Action 同一个包下创建 ActionClassName-conversion.properties

并且声明：需要转换的对象类型的属性名=转换器类型

注意：该文件注册的转换器只对局部的某个 Action 起作用；

3、拦截器：

1）什么是拦截器?

1.拦截器是 struts 框架中提供基础服务的一种组件，在 Action 执行前后执行；

2.当多个 Action 需要使用同一种功能的时候，那么我们可以将这个功能放在拦截器中实现；

3.拦截器是可重用的，如果有一个 Action 需要使用拦截器提供的功能，那么只需要让拦截器拦截这个 Action;

2）拦截器栈：多个拦截器

3）如果 Action 没有显式声明被某个拦截器拦截，那么默认就会被 struts 框架中定义的相关拦截器拦截；一旦声明了被某个拦截器拦截，那么 Action 就不会默认被 struts 框架中定义的相关拦截器拦截，需要显式的声明才行。

4）开发拦截器的步骤：

1.创建拦截器类，实现 Interceptor 接口或者继承 AbstractInterceptor 抽象类；

2.在 interceptors 元素中描述 interceptor;

3.在需要被拦截的 action 中描述 interceptor-ref;

注意：如果多个 Action 被多个相同的拦截器拦截，那么为了引用方便，我们可以将多个拦截器配置成一个拦截器栈，在 Action 中直接引用拦截器栈就等于引用了多个拦截器，我们甚至可以不在 Action 中显式的去引用拦截器而是配置一个 default-interceptor-def(默认拦截器)即可。

例：
```
<interceptors>
        <interceptor name="timer" class="com.briup.interceptor.TimerInterceptor">
        </interceptor>

        <interceptor name="authority"
class="com.briup.interceptor.LoginInterceptor">
        </interceptor>

        <interceptor-stack name="myStack">
            <interceptor-ref name="timer"/>
            <interceptor-ref name="authority"/>
            <interceptor-ref name="defaultStack"/>
        </interceptor-stack>
    </interceptors>

    <default-interceptor-ref name="myStack">
    </default-interceptor-ref>
```

```
<global-results>
    <result name="login" type="redirect">/basic/login.jsp</result>
</global-results>
```

5）struts 框架中提供的拦截器：

1.params：给 action 设置客户端提交的参数；

2.static-params：给 action 或者拦截器设置静态参数；

3.autowiring：将 spring 维护的某个 bean 设置到 action 中；

4.fileupload：提供文件上传的功能；

5.servlet-config：如果 action 实现 aware 接口，那么调用 aware 接口中定义的方法，将相关对象设置到 action 中；

6.workflow：对客户端提交的参数进行数据类型转换和格式验证；

7.validation：使用 xwork-validation 验证框架中定义的验证器进行数据格式验证；

8.exception：如果在 struts.xml 中声明了异常处理，那么由这个拦截器来完成或异常处理的工作；

.....

第五章 Validation(数据格式验证)

1、数据格式验证：

1）客户端验证

验证方式：JavaScript ---> js

优点：效率高，不会占用服务器端太多资源；

缺点：不安全；

注: xwork 验证框架可以在客户端自动产生 js 代码

2）服务器端验证

验证方式：

1.编码：编写 java 代码(Java 代码需写在一个名为 validate 的验证方法内)

注：当 Action 中只有一个处理请求的方法 execute 时，则只需要提供一个 validate 验证方法；如果 Action 中有多个处理请求的方法，那么需要为每个方法定义一个 validateXxx 方法(命名规则为 validate+处理请求的方法名)；这些验证方法在执行的时候只要产生 FieldError，就认为验证不通过，struts2 框架就会将请求转发到"input"结果上。

2.xwork 框架：编写验证文件"*-validation.xml"

当 Action 中只有一个 execute 的时候：ActionClassName-validation.xml

当 Action 中有多个方法时：ActionClassName-ActionMethod-validation.xml

验证器功能说明：

1) required；验证数据是不是为空
2) requiredstring：验证字符串是不是为空
3) int：验证 int 类型的数据的大小
4) date：验证日期类型的数据的大小
5) expression：验证 ognl 表达式(与下面这个验证器的区别？)
6) fieldexpression：验证 ognl 表达式(针对某个字段验证的)
7) email：验证数据是否符合 email 格式
8) url：验证数据是否符合 url 格式
9) visitor：验证复杂类型的数据
10) conversion：验证是否产生类型转换出错
11) stringlength：验证字符串的长度

2、使用 xwork 验证框架：

1) 验证 Action 的属性(非复杂类型)
   a) 在 Action 类同一个包下,创建验证文件 ActionClassName-validation.xml
   b) 在验证文件中,为 Action 的每一个需要验证格式的属性指定使用的验证器
2) 验证 Action 的属性(复杂类型)

方法一: a) 在 Action 类同一个包下,创建验证文件 ActionClassName-validation.xml

b) 在验证文件中,为 Action 的复合类型中的每一个需要验证格式的属性指定使用的验证器.

方法二: a) 在 Action 类同一个包下,创建验证文件 ActionClassName-validation.xml

b) 在验证文件中,为 Action 的复合类型指定 visitor 验证器.

c) 在复合类型同一个包下,创建验证文件 ClassName-contextName-validation.xml

d) 在复合类型的验证文件中,为复合类型中每一个需要验证格式的属性指定使用的验证器

注:当 Action 中有多个处理请求的方法时,针对每一个方法创建一个验证文件 ActionClassName-methodName-validation.xml

比如 RegisterAction 中接收注册信息的是一个 User 类型的参数，User 类型中包含了用户名、密码等信息。那么如何去验证 User 类型的这个 field 呢?

1.创建 RegisterAction-validation.xml

所有 field 的名字都需要定义成 user.*(比如 user.username)

2.创建 RegisterAction-validation.xml

```
<field name="user">
    <field-validator type="visitor">
        <param name="context'>userContext</param>
        <message>User:</message>
    </field-validator>
</field>
```

在 User 类型同一个包下创建 user-userContext-validation.xml，在这个验证文件中需要对 User 类型中的每一个属性使用基本的验证器验证。

3、验证器:

1）required:验证数据是不是为 null

```
<validator type="required">
 <param name="fieldName">username</param>
 <message>username must not be null</message>
</validator>
<field name="username">
 <field-validator type="required">
      <message>username must not be null</message>
 </field-validator>
</field>
```

2）requiredstring:验证字符串是不是为空

```
<validator type="requiredstring">
 <param name="fieldName">username</param>
 <param name="trim">true</param>
 <message>username is required</message>
</validator>
<field name="username">
<field-validator type="requiredstring">
    <param name="trim">true</param>
    <message>username is required</message>
</field-validator>
</field>
```

3）int:验证 int 类型的数据的大小

```
<validator type="int">
<param name="fieldName">age</param>
   <param name="min">20</param>
```

```
        <param name="max">50</param>
    <message>Age needs to be between ${min} and ${max}</message>
    </validator>

    <field name="age">
        <field-validator type="int">
          <param name="min">20</param>
          <param name="max">50</param>
          <message>Age needs to be between ${min} and ${max}</message>
                </field-validator>
    </field>
```

4）date:验证日期类型的数据的大小

```
    <validator type="date">
      <param name="fieldName">birthday</param>
      <param name="min">01/01/1990</param>
      <param name="max">01/01/2000</param>
      <message>Birthday must be within ${min} and ${max}</message>
    </validator>
    <field name="birthday">
        <field-validator type="date">
        <param name="min">01/01/1990</param>
        <param name="max">01/01/2000</param>
        <message>Birthday must be within ${min} and ${max}</message>
        </field>
    </field>
```

5）expression:验证 ognl 表达式(与下面这个验证器的区别?)

```
    <validator type="expression">
      <param name="expression"> .... </param>
      <message>Failed to meet Ognl Expression  .... </message>
    </validator>
```

6）fieldexpression:验证 ognl 表达式

```
    <validator type="fieldexpression">
   <param name="fieldName">myField</param>
   <param name="expression"><![CDATA[#myCreditLimit >
#myGirfriendCreditLimit]]></param>
   <message>My credit limit should be MORE than my girlfriend</message>
    <validator>
```

```
<!-- Field Validator Syntax -->
<field name="myField">
        <field-validator type="fieldexpression">
        <param name="expression"><![CDATA[#myCreditLimit >
#myGirfriendCreditLimit]]></param>
        <message>My credit limit should be MORE than my girlfriend</message>
        </field-validator>
</field>
```

7）email:验证数据是否符合 email 格式

```
<validators>
  <validator type="email">
    <param name="fieldName">myEmail</param>
    <message>Must provide a valid email</message>
  </validator>
</validators>
```

```
<!-- Field Validator Syntax -->
<field name="myEmail">
  <field-validator type="email">
    <message>Must provide a valid email</message>
  </field-validator>
</field>
```

8）url:验证数据是否符合 url 格式

```
<validator type="url">
  <param name="fieldName">myHomePage</param>
  <message>Invalid homepage url</message>
</validator>

<!-- Field Validator Syntax -->
<field name="myHomepage">
        <field-validator type="url">
  <message>Invalid homepage url</message>
        </field-validator>
</field>
```

9）visitor:验证复杂类型的数据

```
<validator type="visitor">
<param name="fieldName">user</param>
<param name="context">myContext</param>
```

```
<param name="appendPrefix">true</param>
</validator>
<field name="user">
<field-validator type="visitor">
    <param name="context">myContext</param>
    <param name="appendPrefix">true</param>
</field-validator>
</field>
```

10）conversion:验证是否产生类型转换出错

```
<validator type="conversion">
        <param name="fieldName">myField</param>
        <message>Conversion Error Occurred</message>
</validator>

<field name="myField">
        <field-validator type="conversion">
        <message>Conversion Error Occurred</message>
        </field-validator>
</field>
```

11）stringlength:验证字符串的长度

```
<validator type="stringlength">
                <param name="fieldName">myPurchaseCode</param>
                <param name="minLength">10</param>
  <param name="maxLength">10</param>
  <param name="trim">true</param>
  <message>Your purchase code needs to be 10 characters long</message>
</validator>

<!-- Field Validator Syntax -->
<field name="myPurchaseCode">
        <field-validator type="stringlength">
                <param name="minLength">10</param>
    <param name="maxLength>10</param>
    <param name="trim">true</param>
    <message>Your purchase code needs to be 10 characters long</message>
        </field-validator>
</field-name>
```

4、作业 1:

1. 将所有处理请求的方法方在一个 Action 中:BookAction.

list:列出所有的书

store:保存一本书

load:查找一本书

delete:删除一本书

remove:删除多本书

2. 采用 Action 的属性进行数据的传递给 jsp 页面显示

3. jsp 页面上遍历数据和输出数据采用 struts2 的标签和内置表达式

5、作业 2：

采用相关验证器验证下列数据，创建一个 RegisterAction(com.briup.validation 包下)，这个类需要继承 ActionSupport，提供接收注册信息的属性并且提供 getter/setter 方法。在同一个包下创建 RegisterAction-validation.xml 验证文件，并且编写此验证文件。在 struts.xml 文件中描述 RegisterAction。在 Webroot 上创建一个目录 validation，register.jsp(请放入 validation 目录下)提供给大家，放在 220 / jd0806/struts2 目录下。

用户名: 不能为空,长度在 6~12 个字符内

密码: 不能为空,并且包含数字和字符,长度在 4~25 之间

```
<field-validator type="regex">
    <param name="expression">
        <!CDATA[(\w{4, 25})]]>
    </param>
    <message>密码必须包含字符和数字,长度在 4 到 25 之间
</field-validator>
```

重复密码: 需要与密码相同

重复密码的验证，仅作参考!

```
<field name="repassword">
    <field-validator type="filedexpression">
        <param name="expression">
            <![CDATA[(password==repassword)]]>
        </param>
        <message>两次输入的密码必须相同!</message>
    </field-validator>
</field>
```

年龄: 必须在 20~50 之间

生日: 必须在 01/01/1990 到 01/01/2000 之间

邮箱: 符合 email 格式

个人主页: 必须符合 url 格式

第六章 Internationalization(I18N：国际化)

1、国际化(简写 i18n)处理：

资源文件(*.properties)：key=value

1）创建一组资源文件：

BaseName_language.properties

struts.properties(默认资源文件)

struts_en.properties(英文资源文件)

struts_zh.properties(中文资源文件)

注意：当用户选择的语言类型没有对应的资源文件时，采用默认资源文件中的内容作为显示的内容。

2）在 struts.xml 文件中配置一个 constant，指定这一组资源文件的 BaseName，让 struts 框架读取这一组资源文件。

<constant name="struts.custom.i18n.resources" value="struts"></constant>

3）在 jsp 页面上要显示的内容全部从资源文件中获取，使用<s:text name=""/>标签获取，根据当前使用的语言类型决定使用哪个资源文件中的内容作为显示的内容。

2、中文转换：

native2ascii *(源文件) *(目标文件)

命令：native2ascii a b ---> 将文件 a 转换成 unicode 编码到文件 b

3、对验证失败的错误信息也可以进行国际化，只要将验证器的 message 的 key 属性指定资源文件中的一个 key.

==========

struts1.x

一、开发 struts1 应用的基本步骤

1）首先导入 struts1 的所有 jar 包；

2）在 WEB-INF 下创建 struts-config.xml 这个核心配置文件；

3）在 web.xml 文件中配置 struts 的主控器 ActionServlet；

注：以上三步可以通过 MyEclipse 插件自动完成

4）创建一个 ActionForm 类，继承 ActionForm(可选)，在这个类中提供属性和

getter/setter方法来获得客户端提交的参数值；

5）创建一个Action类，继承Action。在execute方法中编写处理请求的代码，处理完成后返回一个ActionForward对象给主控器。

注：ActionForword实际上是一个包含了视图路径和转发请求方式的一个对象，这个对象可以有两种创建方式：

1.直接new一个，在创建的时候，通过构造器传入视图路径和转发请求方式等信息。

2.通过ActionMapping对象的findForward方法创建，这种方式实际上是在配置文件中配置转发请求的一些信息，然后根据这些信息创建一个ActionForward对象。

6）在struts-config.xml文件中配置Action；

7）在jsp页面上指定发送请求的路径，必须以.do结尾。

二、Struts的工作流程

装载并实例化ActionServlet

当Web Server启动装载Web应用(基于Struts)时，容器先将ActionServlet装载进虚拟机并实例化，接着调用init()方法，在init()中ActionServlet去解析struts-config.xml文件，将其中的元素信息封装到相应的对象中，对象被存放到Struts框架维护的集合中。

例如:

<form-bean>元素 --> ActionFormBean对象

<action>元素 --> ActionMapping对象

元素中的属性 --> 对象中的属性

注: ActionFormBean和ActionMapping对象被存放在Struts框架维护的HashMap中

HashMap formbeans(name,ActionFormBean对象引用)

HashMap actionconfigs(path,ActionMapping对象引用)

init()方法调用完毕后，ActionServlet处于service状态，可以接受客户端的请求。

(1) 接受客户端所有*.do请求

当客户端发出请求的URL以*.do结尾，那么这些请求将被ActionServlet接受。

(2) 调用RequestProcessor的process()方法

ActionServlet接受到*.do请求后，首先构造一个RequestProcessor对象(单例)，然后调用其process(request,response)方法，将请求委托给RequestProcessor来处理。

(3) 查找请求所对应的ActionMapping对象(定位自定义的Action)

RequestProcessor将请求URL中的servletPath(如:/basic/register.do)截取.do后变成/basic/register(path)，然后根据这个path到HashMap中查找对应的ActionMapping对象，若没

有找到，返回 404 错误代码到客户端，若找到也就能获得属性 type 指定的 Action。

(4) 查找或者构造 ActionForm 对象

根据<action>元素中属性 scope 指定的范围去查找是否存在配置的 ActionForm 对象，(例如:request.getAttribute(key))，key 就是属性 attribute 指定的值。若找到该对象，

就不会去构造一个新的 ActionForm 对象，若没有找到，先构造一个新的 ActionForm 对象，然后将该对象放置到 scope 指定的范围中(request.setAttribute(key,ActionForm 对象引用))。

注意: 若 Action 没有配置 ActionForm，那么(4)~(7)就不执行。

判断是否配置 ActionForm 的方式:

根据<action>元素中属性 name 的值(如:registerForm)去 HashMap 中查找对应的 ActionFormBean 对象，找到该对象也就找到了属性 type 指定的 ActionForm。

(5) 调用 ActionForm 的 reset()方法

调用此方法进行属性的初始化(基本数据类型设置为 0，布尔型设置为 false，String 设置为 null)。

注意: 此方法要用户自己去实现，但是若 ActionForm 对象存放在 request 中可以不实现该方法，若存放在 session 中最好实现该方法，对所有的属性进行初始化。

(6) 调用 ActionForm 的 setXXX()方法

调用此方法就将提交表单中的字段信息封装到 ActionForm 的属性中。

(7) 调用 ActionForm 的 validate()方法(可选)

如果<action>元素中的属性 validate 为 true，那么就调用 validate()方法进行表单字段的验证，该方法由用户提供实现。若设置 false 就不调用该方法。

注意: 该方法并返回一个 ActionErrors 对象(内置一个集合)，若集合中含有 ActionMessage 对象(封装错误消息)，验证就不通过，接着跳转到属性 input 指定的 jsp。若返回 null 或者 ActionErrors 对象内置的集合中不含有 ActionMessage 对象，验证就通过。

(8) 查找或者构造 Action 对象(单例)

RequestProcessor 首先判断在放置 Action 缓存(HashMap)中是否已经存在一个 Action 对象，若没有就构造一个新的 Action 对象，并放置到 Action 缓存中。

注意: Action 缓存是一个 HashMap，存放形式：

(className,Action 对象引用)，className 就是<action>元素中属性 type 指定的值。

(9) 调用 Action 的 execute()方法

此方法类似 Servlet 中的 service 方法，返回结果为一个 ActionForward 对象。

(9-1) 调用 ActionForm 的 getXXX()方法

在 Action 的 execute()中可以先调用 ActionForm 的 getXXX()方法来获得所有封装在 ActionForm 中的表单字段信息。

注意: 提交表单中的字段信息还可以通过 request.getParameter(name)来获得

(9-2) 调用模型层中的 service()方法(业务接口方法)

在 Action 的 execute()中调用模型层中的业务接口方法，处理业务逻辑。

(9-3) 调用 ActionMapping 的 findForward()方法

在 Action 的 execute()中最后调用此方法返回一个 ActionForward 对象。

(10) 调用 ActionForward 的 getPath()

调用此方法返回一个 ActionForward 对象中封装的服务器端资源的 path(Jsp、Servlet 或者 Action)。

(11) 进行跳转或者重定向

RequestProcessor 根据 ActionForward 封装的 path 和 redirect 的值进行跳转或者重定向到另外一个服务器端的资源(Jsp、Servlet 或者 Action)。

(12) 返回一个 response 到客户端

动态生成一个 html 加入到 response 中并返回到客户端，由浏览器解析显示。

三、struts-config.xml 配置文件中主要的元素和属性

1、<form-bean>元素 --> 对应于 ActionFormBean 对象

作用：用来定义一个 ActionForm，供 Action 使用；

包含的属性:

1) name: 指定 ActionForm 的名字，Action 在使用 ActionForm 时通过这个名字使用。

2) type: 指定 ActionForm 的类型 --> 指定用户自己定义的 ActionForm

2、<action>元素 --> 对应于 ActionMapping 对象

作用：用来定义一个 Action，处理客户端提交的请求

包含的属性:

1) path: 指定请求 Action 的路径，注意必须以"/"开始

2) type: 指定用户自己定义的 Action 的类型

3) name: 引用已定义的一个 ActionForm 来接收客户端提交的数据，要和<form-bean>元素中属性 name 的值对应

4) attribute: 指定 ActionForm 对象存放在某个范围内的 key；如不指定此属性，那么以 name 的值作为 key

5) scope: 指定 ActionForm 对象存放的范围

6) validate: 指定 ActionForm 中的 validate()方法是否被调用来验证数据(默认为 true)

true --> 调用

false --> 不调用

7) input: 指定当 validate()方法验证失败时要跳转的 jsp

3、<forward>元素 --> 对应于 ActionForward 对象(分为局部和全局两种)

作用：配置 Action 返回的结果信息

包含的属性:

1) name: 指定 forward 的名字

2) path: 指定 forward 的路径

3) redirect: 指定是否要重定向(默认为 false)

true --> 客户端重定向

false --> 服务器内部跳转

四、RequestProccessor

1、RequestProccessor 是 ActionServlet 的辅助对象，提供了一系列方法供主控器调用，比如把客户端提交的参数封装到 ActionForm 中，执行 Action 等等。

2、RequestProcessor 类的 process()方法：该方法负责实际的预处理请求操作，包括以下方法

1) processMultipart()

如果 HTTP 请求方式为 POST，并且请求的 contentType 属性以"multipart/form-data"开头，标准的 HttpServletRequest 对象将被重新包装，以方便处理"multipart"类型的 HTTP 请求。如果请求方式为 GET，或者 contentType 属性不是"multipart"，就直接返回原始的 HttpServletRequest 对象。

2) processPath()

获得请求 URL 的路径，这一信息可用于选择合适的 Struts Action 组件。

3) processLocale()

当 controllerConfig 对象的 locale 属性为 true，将读取用户请求中包含的 Locale 信息，然后把 Locale 实例保存在 session 范围内。

4) processContent()

读取 ControllerConfig 对象的 contentType 属性，然后调用

response.setContentType(contentType)方法，设置响应结果的文档类型和字符编码。

5) processNoCache()

读取 ControllerConfig 对象的 nocache 属性，如果 nocache 属性为 true，在响应结果中将加入特定的参数:Pragrma、Cache-Control 和 Expires，防止页面被存储在客户浏览器的缓存中。

6) processPreprocess()

该方法不执行任何的操作，直接返回 true。子类可以覆盖这个方法，执行客户化的预处理请求操作。

7) processMapping()

查找和用户请求的 URL 匹配的 AvctionMapping。如果不存在这样的 ActionMapping，则向用户返回恰当的错误信息。

8) processRoles()

先判断是否为 Action 配置了安全角色，如果配置了安全角色，就调用 isUserInRole()方法判断当前用户是否具备必备的角色，如果不具备，就结束请求处理流程，向用户返回恰当的错误信息。

9) processActionForm()

先判断是否为 ActionMapping 配置了 ActionForm，如果配置了 ActionForm，就先从 ActionForm 的存在范围内寻找该 ActionForm 实例。如果不存在，就创建一个新的实例。然后它保存在合适的范围中，保存时使用的属性 key 为 ActionMapping 的 attribute 属性。

10) processPopulate()

如果 ActionMapping 配置了 ActionForm，就先调用 ActionForm 的 reset()方法，再把请求中表单数据组装到 ActionForm 中。

11) processValidate()

如果 ActionMapping 配置了 ActionForm，并且 validate 属性为 true,就调用 ActionForm 的 validate()方法。如果 validate()返回的 ActionErrors 对象中包含 ActionMessage 对象，说明表单验证失败，就把 ActionErrors 对象存储在 request 范围中，再把请求转发到 ActionMapping 的 input 属性指定的组件中。如果 validate()验证成功(返回 null 或者 ActionErrors 中没有 ActionMessage 对象)，就继续执行下一步请求处理流程。

12) processForward()

判断是否在 ActionMapping 中配置了 forward 属性。如果配置了这个属性，就调用 RequestDispatcher 的 forward()方法，请求处理流程结束，否则继续下一步。

13) processInclude()

同上，差别在于调用 RequestDispatcher 的 include()方法。

14) processActionCreate()

先判断是否在 Action 缓存(存放所有 Action 的 HashMap，该 HashMap 类型的变量 actions 存放在 RequestProcessor 中)中存在这个 Action 实例，如果不存在，就创建一个 Action 实例，并把保存在 Action 缓存中。存放形式: actions(type,action)

15) processActionPerform()

该方法再调用 Action 实例的 excute()方法

16) processForwardConfig()

把 Action 的 excute()方法返回的 ActionForward 对象作为参数传递给它。RequestProcessor 根据 ActionForward 对象包含的请求转发信息来执行请求转发或者重定向。

五、Struts1 的执行流程

1）客户端发起.do 结尾的请求；

2）ActionServlet 接收请求，交给 RequestProcessor 处理(实际上就是调用 RequestProcessor 的 process 方法)；

3）RequestProccessor 根据请求路径查找对应的 ActionMapping;

4）RequestProccessor 查找或者创建一个 ActionForm 对象(ActionForm 可以被重用，存在于两个范围，Session 或者 Request)；

5）RequestProccessor 调用 ActionForm 的 reset 方法，重置状态信息；

6）RequestProccessor 调用 ActionForm 的 setXXX 方法，将客户端提交的参数封装到 ActionForm 中；

7）RequestProccessor 调用 ActionForm 的 validate 方法，如果此方法返回一个 ActionErrors 对象，并且这个对象中包含错误信息，那么认为验证失败，如果验证失败，那么将请求转发到 Action 元素的 input 属性指定的路径上去，反之验证成功；

8）RequestProccessor 查找或者创建一个 Action(Action 对象是单例的)；

9）RequestProccessor 调用 Action 对象的 execute 方法执行 Action，并且传入 ActionMapping，ActionForm，HttpServletRequest，HttpServletResponse 这四个对象；

10）在 Action 的 execute 方法中可以通过调用 ActionForm 的 getter 方法来获得客户端提交的数据，可以调用模型的方法来处理业务；

11）通过 ActionMapping 的 findForward 方法返回一个 ActionForward 对象或者直接 new 一个 ActionForward 对象；

12）主控器获得 Action 返回的 ActionForward 对象，然后根据这个对象中封装的信息，将请求转发(重定向或者内部跳转)到具体页面；

13）jsp 页面向客户端作出响应；

六、struts1的高级部分

1、DynaActionForm：动态 form

1）优点：使用动态 form 可以让开发人员不提供 ActionForm 类，全部通过配置的形式创建 ActionForm,而且比较灵活，容易修改和扩展。

2）步骤：

1.在 struts 的配置文件中配置一个 form-bean，类型 DynaActionForm 类型，并且为这个 action form 指定能够接收的参数名字和类型，通过 form-property 元素说明；

2.在 action 元素中引用动态 ActionForm 跟普通 form 的引用方式一样；

3.在 Action 处理去的方法中，需要将主控器传入的 ActionForm 强制转换成 DynaActionForm 类型，然后通过调用这个对象的 get*方法获得客户端提交的参数；

2、DispatcherAction extends Action

应用场合：多个表单交给同一个 Action 的不同方法处理

注：在 struts-config.xml 文件中一个 Action 类配置一个 Action

原理：客户端在提交请求的时候，需要提交一个方法名过去，DispatcherAction 的 execute 方法根据此方法名，通过反射调用对用的方法。

3、LookupDispatcherAction

应用场合：同一个表单的多个提交按钮，交给同一个 Action 的不同方法处理。

使用方法：

1）Action 类：首先继承 LookupDispatcherAction，实现 getKeyMethodMap 方法，这个方法返回的 Map 中存放提交按钮与访问方法的对应关系。Map 中的 key 就是按钮值在资源文件中对应的 key，Map 中的 value 就是方法名。

2）配置文件：Action 元素需要指定 parameter 属性，LookupDispatcherAction 可以根据 parameter 从 request 中获得一个按钮值，然后根据这个按钮值就可以找到对应的方法名。

3）jsp 页面：在一个表单中两个提交按钮的 name 属性取值就是 action 元素的 parameter 属性值，两个提交按钮的 value 属性取值就是资源文件中存放的提交按钮值。

4、MappingDispatcherAction

应用场合：基本跟 DispatcherAction 类似，区别就是这种类型的 Action 需要为每个方法配置一个 Action; 当 Action 中的每个方法需要使用不同 ActionForm 接收数据的时候，我们选择 MappingDispatcherAction.

七、struts1 中的国际化处理

1、创建资源文件；
ApplicationResources.properties
ApplicationResources_en.properties
ApplicationResources_zh.properties

2、在 struts-config.xml 文件中配置资源文件的 baseName，让框架读取这一组资源文件；
<message-resources />

3、在 jsp 页面上通过<bean:message/>标签输出资源文件中的内容。

4、更改使用的语言类型：
1）修改浏览器语言类型；
2）在 session 中设置 Globals.LOCALE_KEY 的读取；

Struts 笔记于 2008 年 8 月 26 日全部完成！！！

Ajax 学习笔记

==========

主讲：沈阅斌

Ajax 前言

1、胖客户端(富客户端)
特点：1）类似于传统的桌面系统；
2）系统的反应是非常迅速和灵敏的；
3）系统对使用者来说是非常友好的，可以提供丰富的提示功能；
4）用户在使用的过程中，所做的操作不会被中断。

2、Ajax:
1）用来开发比较复杂的 B/S 架构的客户端
2）AJAX(Asynchronous JavaScript And XML)：异步的 JavaScript 和 XML.
HTML：用来显示的
XML：是以数据为核心的

3、传统的 B/S 架构的软件：
1）同步的交互方式：请求 ===> 等待 ===> 响应式
2）缺点：1. 会中断用户操作；
2. 如果网络慢，会出现空白页面，对用户来说是非常不友好的；

3. 每次响应都会把一个新页面的所有内容替换整个浏览器上的所有内容，影响运行效率(局部刷新)。

Ajax 正文

1、什么是 AJAX?

1）它是一个客户端技术，是将原来一些老的技术进行结合使用，产生的一种新的应用；

2）可以跟任何的服务器端技术开发的程序进行交互；

3）AJAX 不是一门新的语言或者技术，只是将 XML，DOM，JS，CSS 等技术进行结合使用；

4）异步的交互方式。

2、在 AJAX 中使用的一些技术

1）JavaScript：编写基于 AJAX 的 web 应用，主要就是编写 js 代码；

2）Dom：1. 当服务器端响应回来的内容是一个 XML 文档的时候，客户就可以根据 dom 模型解析这个 xml 文档；

2. 客户端在进行局部刷新时，通过 dom 模型操纵整个页面；

3）CSS：控制显示样式，使得页面更加美观；

4）XMLHttpRequest：就是一个 js 对象，用来操纵 AJAX 引擎。

3、进行 AJAX 编程的基本步骤

1）客户端产生 js 的事件；

2）创建 XMLHttpRequest 对象(客户端浏览器内置对象)；

3）对 XMLHttpRequest 进行配置；

4）通过 AJAX 引擎发送异步请求；

5）服务器端接收请求并且处理请求，返回 html 或者 xml 内容；

6）XMLHttpRequest 调用一个 callback 函数(自己定义的)处理响应回来的内容；

7）页面被进行局部刷新。

4、XMLHttpRequest 对象的属性和方法

1）open("HTTP method","URL",syn/asyn):

设置 http method，请求路径以及请求方式；

2）send(content):

发送请求，content 为提交的参数。如果是 get 方式提交，content 为 null；

注意：提交参数的格式必须为 userName=briup, userName=briup&password=briup 等形式

3）onreadystatechange:

指定一个 js 的函数(callback 函数)，主要用来处理响应回来的内容；

当 XMLHttpRequest 对象的状态发生变化的时候，自动调用这个属性所指定的回调函数。

4）readyState:

返回 XMLHttpRequest 对象当前的状态.

0：没有初始化；1：装载中；2：装载结束；3：交互中；4：完成

5）status:

返回服务器端处理请求的情况(返回服务器端的错误编号：200，500 或 404 等)；

6）responseText:

返回服务器端响应回来的内容，得到的是文本内容；

7）responseXML:

返回服务器端响应回来的 xml 文档，得到的是一个 document 对象；

5、文档对象模型 DOM 中常用的属性和方法

属性：

1）childNodes：返回当前元素的所有子结点(数组)；

2）firstChild：返回当前元素的第一个子结点(未必是元素)；

3）lastChild：返回当前元素的最后一个子结点；

4）nextSibling：返回当前元素的下一个元素；

5）nodeValue：返回节点的值；

6）parentNode：返回父节点；

7）previousSibling：返回当前元素的前一个元素；

方法：

1）getElementById(id)：根据 id 返回一个元素；

2）getElementByTagName(tagName)：根据标签名返回一组元素；

3）getElemetnByName(name)：根据 name 返回一组元素；

4）hasChildNodes()：判断是否有子结点；

5）getAttribute(name)：获得属性值；

6、对页面内容进行局部更新的几种方式：

1）通过 DOM API 来更新当前页面内容；

2）通过 div 的 innerHTML 属性进行局部更新；

做法：在页面上使用 div 标签划分一块区域作为局部更新的区域，然后通过 js 将需要更新的内容直接赋给 div 的 innerHTML 属性即可。

7、服务器端返回 xml 内容的方式：

1）直接返回 xml 文件；

2）在 servlet 中调用 response 对象的 setContentType 方法，指定内容类型为 text/xml；

3）在 jsp 中将 page 指令的 contentType 属性指定成 text/xml 即可；

8、 Ajax 中下拉菜单的相关操作

1）获得某个下拉菜单的选中的选项的值

```
var province=document.getElementById("province");
var options=province.options;
var value="";
for(var i=0;i<options.length;i++){
        if(option[i].selected){
                value=option[i].value;
        }
}
```

2）清空下拉菜单的所有选项

```
function clearSelect(){
        var city=document.getElementById("city");
        while(city.hasChildNodes()){
                city.removeChild(city.childNodes[0]);
        }
}
```

3）构建选项，追加到下拉菜单上：

```
var citySelect=document.getElementById("city");
var option=new Option(name,value);
citySelect.options.add(option);
```

Ajax 在线聊天室的实现

要求:使用 ajax 以及相关 web 技术实现一个比较完善的在线聊天室

1、可以提供登陆功能，用户登陆之后进入在线聊天室，也可以进行匿名登陆.

2、进入聊天室之后，用户可以看到聊天记录,主要包含聊天者，聊天内容和时间.

3、可以通过一个表单进行聊天.

注:最好实现在聊天页面上显示的聊天记录为最新的 50 条记录.

并且按时间的先后顺序进行排序.

实现步骤:

1、提供一个聊天界面 chat.jsp(由我提供,已放到 220 上)

2、实现两个发请求的功能:

1. 输入一个内容之后可以进行聊天(发送消息)

只需你输入的内容发送到服务器端保存起来即可.

2. 实时的显示当前最新的聊天记录(查询消息)

使用定时器，定时发送请求到服务器端查询需要显示的聊天记录,并且在页面上进行显示.

比如(2 秒调用一次 queryMessage 函数):

```
<script type="text/javascript">
                    window.setInterval(queryMessage,2000);
</script>
```

注:聊天内容可以存放在服务器端的一个 Servlet 的实例变量中；一条聊天记录至少包含 sender,content,senddate，所以需要提供一个包装类型 Message，一个 Message 对象就是一条聊天记录。

AJAX 笔记于 2008 年 8 月 28 日全部完成！！！

OOAD & UML 学习笔记

==========

主讲：王正平
wangzp@briup.com

学习目标：

1）了解软件开发的过程

2）了解 OOA(面向对象分析)和 OOD(面向对象设计)

3）了解 UML(统一建模语言)，并利用 UML 来描述 OOA 和 OOD

4）了解 OO 的一些设计原则和设计模式

第一章 OOAD 概述

1、OO(面向对象)

将客户需要解决的问题抽象成一个个对象，通过这些对象之间的调用来形成一个可以运行的软件系统。

2、OO 的优点

1）对象比较符合人的思维方式

2）可以将数据和行为方式封装在对象中，便于重用和维护。

3）利用 OO 的特性(封装、继承、多态等)来编写程序，可以提供代码的灵活度。

3、对象的特点

1) 万事万物皆为对象

2) 对象是唯一的

3) 对象有自己的数据和行为

4) 对象在某一特定时刻有特定的属性值

5) 对象都是属于某一个类型的(如类或者接口等)

4、软件开发的注意阶段

1）OOA：What to do?

如：根据客户端需求要解决的问题定义一些类，但是不要去具体实现。

2）OOD：How to do?

如：对类进行细化，定义一些属性和方法的具体实现。

5、OOP(面向对象编程)的主要特征

1）抽象

将客户需求解决的问题提取符合要求的特征抽象成一个对象的思维过程。

抽象 抽象 抽象

客户需求要解决的问题---->对象---->类---->父类

2）封装

分为两个方面：

1.隐藏数据：在属性前面使用 private 修饰，那么要获得这些属性值的话，要调用一个 public 的方法来获得。

2.隐藏方法的实现细节：对象方法的具体实现细节对调用者是看不见的，调用者只要通过方法名调用此方法完成功能即可。

如：
```
public class Television{
    //调节电视机音量
    public void adjust(){
        //隐藏具体实现细节
    }
}
```

调用者：
```
Television tele = new Television();
tele.adjust();
```

3）继承

将多个类中出现的相同的特征或者行为抽取出来定义在一个单独的类(父类)中，那么这些作为子类的类可以继承(extends)这个父类，并可以把父类中所有的特征都继承下来。

继承的特点：

1.是"is a"的关系，单继承

2.继承是出现多态的基础

3.继承可以起到代码重用

4）多态

一个引用指向了多个不同类型的对象，多态发生在运行阶段，它是基于继承的。

如：
```
public class Ball{
    public void play(){
        System.out.println("玩球");
    }
}
public class FootBall extends Ball{
    public void play(){
        System.out.println("用脚玩球");
    }
}
public class BasketBall extends Ball{
    public void play(){
        System.out.println("用手玩球");
    }
}
main():
    Ball ball = new FootBall();
    ball.play();
    ball = new BasketBall();
    ball.paly();
```

多态的作用：

1.退耦(降低耦合度)：

用户只要调用父类或者接口中的方法而不用关心子类和实现类是如何对细节进行实现的。

2.提高代码灵活度

在不改变原有代码的基础上可以方便地增加新的功能。

多态的表现形式：

```
List list = new ArrayList();
Set set = new HashSet();

public void setList(List list){}
setList(new ArrayList());

public List getList(){
    return new ArrayList();
}
```

5）关联

当一个对象调用另外一个对象的方法或其它数据信息时，那么这两个对象之间就产生了关联。其关联关系可分为三种：

1.普通关联

特点：*使用局部变量

*整体(外部对象)不决定部分(内部对象)的生命周期

*部分可以服务于多个整体

如：
```
public class Person{
        public void use(Computer c){
                c.method();
        }
}
main():
Person p = new Person();
Computer c = new Computer();
p.use(c);
```

Person 对象和 Computer 对象之间属于普通关联。

2.聚合

特点：*使用实例变量

*整体不决定部分的生命周期

*部分可以服务于多个整体

如：
```
public class Order{
        private Orderline o1;
        public void setO1(Orderline o1){
                this.o1 = o1;
        }
}
```

3.组合

特点：*使用实例变量

*整体决定部分的生命周期

*部分仅服务于一个整体

如：
```
public class Person{
        private Computer c1;          //实例变量
        private Heart heart;          //实例变量
        public Person(){
                heart = new Heart();
        }
        public void setC1(Computer c){
                c1 = c;
        }
        public void use(){
```

```
            c1.method();
        }
    }
```

Person 对象与 Computer 对象属于聚合；

Person 对象与 Heart 对象属于组合。

关联关系从弱到强：普通关联-->聚合-->组合

6）内聚与耦合

1.内聚：度量一个类独立完成某项工作的能力

2.耦合：度量系统内或系统之间依赖关系的复杂度

设计原则：增加内聚，减少耦合

高内聚低耦合 --->“不要和陌生人说话”

6、OOAD 的开发过程

1）传统开发过程：瀑布模型

瀑布模型的缺点：

1.公司内部的开发人员不能同时进行工作，影响系统的进程和开发效率。

2.不能及时适应客户经常变更的需求。

2）统一软件开发过程：USDP

1.是一个迭代、递增的开发过程。

2.四个阶段：

a.初始化阶段

b.细化阶段

c.构建阶段

d.移交阶段

3.迭代、递增生命周期的优点：

a.降低成本

b.便于更好地维护项目进度

c.便于团队的协作开发

d.便于适应用户需求的动态变化

第二章 UML

1、UML(统一建模语言)：

UML 是一种图形化的语言，它可以帮助我们在 OOAD 过程中标识元素、构建模块、分析过程，并可通过文档说明系统中的重要细节。

2、UML 图的分类

1）静态模型

1.创建并记录一个系统的静态特征，描述系统包含的内容(有什么?)

2.反映一个软件系统基础、固定的框架结构

3.创建相关问题域主要元素的视图

4.静态建模包括：

—— 用例图（ use case diagrams ）

—— 类图（ class diagrams ）

—— 对象图（ object diagrams ）

—— 组件图（ component diagrams ）

—— 部署图（ deployment diagrams ）

2）动态模型

1.动态建模用以展示系统的行为，描述系统实现的操作方式和流程(怎么做?)

2.动态建模包括：

—— 时序图（ sequence diagrams ）

—— 协作图（ collaboration diagrams ）

—— 状态图（ state chart diagrams ）

—— 活动图（ activity diagrams ）

创建一个 java project: uml

画图工具：IBM ---> Rational Rose

第三章 需求与初始化分析

1、启动开发过程

—— 初始化系统开发环境

—— 分析初始工作流

—— 收集信息

—— 创建问题域

—— 创建用例图

—— 创建相关的组件图与部署图

2、收集的信息可来源于：

—— 客户最初的需求说明书

—— 领域专家

—— 客户与用户

—— 管理人员
—— 市场信息
—— 以前的项目

3、创建问题域
1）清晰展现与系统相关的区域和问题的范围
2）可以用图形或文本形式说明

4、数据字典：一份用于描述项目中所用词汇的文档

5、创建用例
1）用例表现用户与系统的交互
2）帮助理解系统需求与环境
3）用实线组成的椭圆表示
4）创建用例模型包含的组件：
—— 活动者
—— 用例
—— 系统
—— 泛化与关联关系

6、活动图
1）用于展现活动、过程与工作流
2）可用在任何你需要对活动建模的地方
3）包含下列元素
—— 活动
—— 判断
—— 流程的分离
—— 流程的汇合
—— 迭代
—— 对象流
—— 泳道

7、组件图
1）表现系统中的组件及其关系
2）使用祖件的好处：
—— 有助于降低系统复杂度
—— 从框架上看问题避免涉及过多细节

—— 可独立地查看某个组件

—— 组件的详细设计可推迟到具体场景中再完成

8、部署图

1）展现硬件设备之间的物理联系

2）可在项目的任意阶段加入

3）当有补充信息时可随时完善

4）节点中可包含组件

5）节点之间的连接（通讯方式）使用协议说明

第四章 分析阶段的静态建模

1、属性的划分

1）普通属性

2）推导属性

```
public class OrderLine{
        private int id;
        private double price;
        private int quantity;    //普通属性
        private double cost;             //推导属性

        public double getCost(){
                return price * quantity;
        }
}
```

OderLine 表：

```
------------------------------
id              price           quantity
 1               20.5                  10
```

2、关联的三种方式

1）1:1(双向)

a.类结构：

```
public class Car{
        private Engine engine;
        ...
        public void setEngine(){}
        public Engine getEngine(){}
}
public class Engine{
        private Car car;
        ...
```

```
        set get 方法
}
```

b.表结构：外键可放在任意一方

Car 表

```
---------------------------------
id              name            engine_id
```

Engine 表

```
---------------------------------
id              name
```

注意: 在表结构中永远是单向，从有外健的那张表指向外健引用主健的那张表。

2）1:*(双向)

a.类结构：

```
public class Person{
        private Set<car> sert = new HashSet<car>();
        ...
        set get 方法
}
public class Car{
        private Person person;
        ...
        set get 方法
}
```

b.表结构：外键放在多的一方

Person 表

```
---------------------------------
id              name
```

Car 表

```
---------------------------------
id              name            prison_id
```

3）*:*(双向)

a.类结构：

```
public class Person{
        private Set<Book> set = new HashSet<Book>();
        ...
        set get 方法
}
public class Book{
        private Set<Person> set = new HashSet<Person>();
        ...
```

```
        set get 方法
}
```

b.表结构：

Person 表

---

id name

Book 表

---

id name

Person_Book 表(join table)

---

person_id book_id

```
public class Bank{
        private Integer custRef;
        private Map<Integer,Customer> map = new HashMap<Integer,Customer>();

        public Customer findCustomer(Integer custRef){
                return map.get(custRef);
        }
}
```

3、分析继承关系的两种方式

1）泛化

从一组已经存在的类中抽取出公共的特征得到一个新类的过程。

2）特化

当要增加的新类与已存在的类有着相同的结构、功能和目的，但新类需要增加新的属性与方法。

4、关联

1）带角色名的关联：

角色名：表示类与类之间交互时扮演的“角色”，角色名最终以属性名的形式表示。

```
public class Customer{
        //product 在 UML 中是一个角色名
        private Product product;
}
public class Product{
        private Customer purchaser;
}
```

2）反射关联：

```
public class Person{
        private Person husband;
        public void married(Person husband){
                this.husband = husband;
        }
}
main():
        Person husband = new Person();
        Person wife = new Person();
        wife.married(husband);
```

3）普通关联：

```
public class Chef{
        public void chopsWith(Knife k){
                k.method();
        }
}
```

4）聚合：

```
public class Car{
        private CarRadio radio;        //实例变量
        public void setRadio(CarRadio radio){
                this.radio = radio;
        }
        public void use(){
                radio.method();
        }
}
```

5）组合：

```
public class Car{
        private Engine engine;
        public Car(){
                engine = new Engine();
        }
}
```

5、方法扩散

一般发生在聚合和组合中，指某个对象的方法去调用另外一个对象的方法。

```
public class Car{
        private Engine engine;
        public Car(){
                engine = new Engine();
        }
        //汽车加速
        public void pushAccelator(){
                //方法扩散
```

```
                engine.accelator();
        }
}
main():
Car c = new Car();
c.pushAccelator();
```

代理对象：Car 对象

被代理对象：Engine 对象

6、代理与委托

1）代理(Proxy)

若代理对象和被代理对象继承了同一个类或者实现同一个接口就称为 Proxy。

```
public class Engine extends AAA{}
public class Car extends AAA{
        private Engine engine;
        public Car(){
                engine = new Engine();
        }
        public void pushAccelator(){
                engine.accelator();
        }
}
```

2）委托(Delegate)

若代理对象和被代理对象没有继承同一个类或者实现同一个接口就称为 Delegate。

注意: 代理和委托发生在方法扩散中

第五章 分析阶段的动态建模

1、创建分析阶段的动态模型

在 OOAD 过程中，动态建模发生两次：

1）在逻辑分析阶段，确保系统中的每一个操作是可实现的

时序图

协作图

状态图

活动图

2）在物理设计阶段，具体细化类中的方法

2、责任

在OOAD的术语中，“责任”通常指：

1）一个类或类型知道什么（状态）

2）一个类或类型能做什么（行为）

第六章 设计原则与模式

1、设计原则

1）“开—闭”原则（OCP）

1.“开—闭”原则：一个软件实体应该对扩展开放，对修改关闭

2.如何做到“开—闭”

—— 抽象化是关键

3.对可变性的封装原则

a.可变性不应该散落在代码的很多角落里，而应该被封装到一个对象中

b.一种可变性不应当与另外一种可变性混合在一起

2）里氏代换原则（LSP）

1.里氏代换原则：任何基类适用的地方，子类一定也适用

2.从继承的角度实现“开—闭”

3）依赖倒转原则（DIP）

1.依赖倒转原则：要依赖于抽象，不要依赖于实现

2.三种耦合关系

—— 零耦合

—— 具体耦合

—— 抽象耦合

3.针对接口编程

```
public class C{
    //零耦合：没有用到任何的接口和实现类
    public void method1(){

    }
    //具体耦合：利用实现类
    public void method2(ArrayList list){
    list.add("aaa");
    }
    //抽象耦合：利用接口
    public void method3(List list){
```

```
        list.add("aaa");
        }
}
```

4）接口隔离原则（ISP）

1.接口隔离原则：使用多个专门的接口比使用单一的总接口要好。

2.定制服务

3.接口污染

5）组合/聚合复用原则（CARP）

组合/聚合复用原则：要尽量使用组合/聚合，而不是使用继承来达到复用目的。

```
public class Student{
        private int xh;
         private String name;
        /*
        private String city;
        private String street;
        private String zipcode;
        */
        private Address address;
        public Student(){
        address = new Address();
        }
}

public class Teacher{
        private int bh;
        private String name;
        private Address address;
}


public class Address{
        private String city;
        private String street;
        private String zipcode;
}
```

6）迪米特法则（LoD）

1.迪米特法则也称“最少知识原则”：一个对象或模块应该和其它对象和模块尽量少的通信

——“不要和陌生人说话”

2.对“高内聚低耦合”问题的解决

3.迪米特法则下的模式

—— 门面模式(EJB 中介绍)

—— 调停者模式(Facade 模式，中介者模式)

—— 前端控制器模式

—— 业务代表模式(Service 模式)

——— DAO 模式

2、设计模式

设计模式：用于描述一组对象与类之间的交互以解决某个场景中的设计问题

—— 通过架构模式提升

—— 设计模式中的主要元素：模式名、问题、解决方案、结果

1）工厂模式

1.问题域：

—— 你需要获得一组相关产品类的某个实例

—— 根据请求的不同，可以获得不同的对象

2.解决方案

—— 创建抽象工厂类，通过抽象工厂定义的抽象方法返回抽象产品

—— 创建具体工厂类，实现抽象方法返回具体产品

3.工厂模式(factory)分为:

a.工厂方法(静态)

b.抽象工厂

```
public abstract class Document{
        public abstract void open(String file);
        public abstract void save();
        public abstract void close();
        public abstract void write(String s);
}
public abstract class Application{
        public abstract Document createDoc();
        public Document newDocument(String file){
                Document doc = createDoc();
                doc.open(file);
```

```
                return doc;
            }
        }
        public class HTMLDocument extends Document{
            private PrintWriter pw;
            public void open(String file){
                FileWriter fw = new FileWriter(file);
                pw = new PrintWriter(fw);
            }
            public void write(String s){
                pw.println(s);
            }
            public abstract void save(){
                pw.flush();
            }
            public abstract void close(){
                pw.close();
            }
        }
        public class HTMLEditor extends Application {
            public Document createDoc() {
                return new HTMLDocument();
                }
        }
        public class TextDocument extends Document{
            ...
        }
        public class HTMLEditor extends Application {
            ...
        }
```

2）单例模式

1.问题域：

系统中你需要获得某个类的唯一实例，所有客户端对它的访问都将通过一个公共的访问点获得。

2.解决方案：

创建一个类并使其：

a.定义一个私有的构造器；

b.定义一个私有、静态的实例变量指向自己(类型是自己的类型)；

c.定义一个公有、静态的访问方法 getInstance()用于返回该类的唯一实例(注：懒汉式需同步)。

3.分类：

a.饿汉式：在类装载的时候就进行实例化

b.懒汉式：在公有、静态的访问方法(同步：用 synchronized 修饰)中进行实例化，用的多一些

3）组合模式

1.问题域：

—— 你需要描述对象之间整体—部分的关系(组合关系中菱形所在的一边为整体)；

—— 你需要使用同一个接口装配接口下的单纯元素(树叶)与复合元素(容器类)。

2.解决方案：

—— 创建一个抽象类作为“树根”，创建一组具体类作为“树叶”，并挑选一个具体类作为容器类保留对其它兄弟节点的引用(构建一个集合，增加 add(),remove(),getChild()方法)。

4）观察者模式

1.问题域：

—— 当事件发生时，你需要通知一组对象；

—— 当一个对象状态变化时，你需要通知其它对象，但有多少事先无法估计。

2.解决方案：

—— 创建一个抽象的主题类(被观察者)用于维护一组观察者；

—— 当主题类状态变化时，自动通知所有监听的观察者。

5）MVC 模式

1.问题域：

—— 需要显示模型的多个视图，控制器与模型和视图相互独立

2.解决方案：

—— 减少数据与具体视图的耦合，使得多个视图可以访问相同数据

—— 运行过程中，使用观察者模式为模型注册相关视图

—— 使用策略模式，视图可动态分配到控制器对象

3.mvc 模式

1)m(Moder) ---> 模型层

javabean：封装业务逻辑或者数据访问逻辑

SessionBean(EJB)：封装业务逻辑

ActionForm(Struts1)：封装提交表单中的信息

数据访问层：Dao 层

a) Dao 接口

b) javabean(jdbc Hibernate)：持久化层

c) 数据库

业务逻辑层：service 层

模型层：业务逻辑层和数据访问层统称为模型层

2)v(View) ---> 视图层

Jsp：用于显示

Html

3)c(Controller) ---> 控制层

Servlet：控制和转发请求

Struts1:

ActionServlet(主控制器)

Actions(业务逻辑控制器)

Struts2:

FiterDispatcher(主控制器)

Actions(业务逻辑控制器)

表现层：控制层和视图层统称为表现层

4.MVC 模式的发展演变：

a) [jsp] model 1

m: javabean

v: jsp

c: jsp

b) [jsp] model 2

m: javabean

v: jsp

c: servlet

c) mvc 模式(Web 框架)

m: javabean
v: jsp
c: servlet action

6）状态模式

1.问题域：

—— 对象的行为依赖其状态，对象必须根据其状态选择不同的行为方式(例如：由心情决定做某事)

2.解决方案：

—— 主对象称之为 Context，它有一个或多个依赖于状态的方法

—— 为每一个状态创建一个具体的状态类，每个状态类都实现了 Context 定义的方法

```
public abstract class State{
	protected static PrintWriter out;
	public static final State CLEAN = new CleanState();
	public static final State DIRTY = new DirtyState();
	public abstract void open(Document doc,String file);
	public abstract void save(Document doc);
	public abstract void edit(Document doc,String s);
	public abstract void close(Document doc);
}
public class CleanState extends State{
	public void open(Document doc,String file){
		FileWriter fw = new FileWriter(file);
		out = new PrintWriter(fw);
	}
	public void edit(Document doc,String s){
		out.println(s);
		//切换到 DIRTY 状态
		doc.setState(State.DIRTY);
	}
	...
}
public abstract class Document{
	protected State state; //在子类中实例化
	public abstract void open(String file);
	public abstract void edit(String s);
	public abstract void save();
	public abstract void close();
	public void setState(State state){
		this.state = state;
	}
```

```
}
public class HtmlDocument extends Document{
    public HtmlDocument(){
        state = State.CLEAN;
    }
    public void open(String file){
        state.open(this,file);
    }
    ...
}
```

参考：<<设计模式>>

---- 作者: 四人帮(gang of four)

---> 21 种设计模式

创建 java project: ooad

--src

--com.briup.pattern.factory.basic

--Creator.java

--Product.java

--ConcreteCreator1.java

--ConcreteCreator2.java

--ConcreteProduct1.java

--ConcreteProduct2.java

--Test.java

--src/com.briup.pattern.factory.sample

OOAD&UML 笔记于 2008 年 9 月 10 日全部完成！！！

EJB 学习笔记

==========

主讲：王正平

EJB 前言

1、打包：

jar: java archive file
war: web archive file
ear: enterprise archive file
archive: 存档

2、容器分为：Web 容器和 EJB 容器

Web 容器：装载以及管理 Web 应用(包括 servlet,jsp)
EJB 容器：装载以及管理 EJB 应用

Web 服务器：只有一个 Web 容器，只能部署 Web 应用，如：Tomcat。
应用服务器：Web 容器+EJB 容器，能部署 Web 应用和 EJB 应用，如：Weblogic(BEA)、WebSphere(IBM)。

注：服务器是一个进程，而容器实际上是一个线程。

3、开发任何一个应用的两个需求：
1）功能性需求
2）技术性需求

4、开发大型软件系统需要考虑的几个方面：
Remote Method Invocations(远程方法调用)
Load Balancing(负载均衡)：用集群来解决
Transparent Fail-over(透明容错)：当一台服务器出现问题，将请求交给另外一台服务器来处理
Back-end Integration(后端集成)：将 java 开发系统和非 java 开发系统结合起来
Transactions(事务)
Clustering 集群
Dynamic Redeployment(热部署)
Clean Shutdown(正常停机)
Logging and Auditing
Systems Management(系统管理)
Threading(线程)
Message-oriented Middleware(消息中间件)
Object Life Cycle(对象生命周期)
Resource pooling(资源池)
Security(安全) jaas(java authentication authorization service)java验证授权服务
Caching
…

EJB2.0

第一章 EJB 介绍

1、EJB(Enterprise JavaBean)
1）EJB 是一种用来简化企业级分布式应用开发、位于服务器端的可重用的重量级组

件。

2）EJB 跟 JavaBean 一样也可以来封装业务逻辑。

3）EJB 属于 j2ee 规范之一，EJB 版本有：

EJB1.0

EJB2.x(EJB2.0、EJB2.1)

EJB3.0

4）部署在应用服务器上的 EJB 组件可以获得一些中间件服务。

中间件：应用服务器

中间件服务：指中间件提供的一些事务、线程、日志、安全(技术性需求)等服务。

注：这里的中间件服务实际上是由中间件中的 EJB 容器提供的。

5）开发人员开发的 EJB 组件和供应商开发的应用服务器(中间件)都必须遵守 EJB 规范。

2、EJB 和 JavaBean 的比较

EJB 并不是 JavaBean，但是它具有 JavaBean 的一些特点: 如 1) 2) 3)

1）可以提供一些属性，并提供 set、get 方法；

2）用来封装业务逻辑；

3）都是一个可重用的组件；

4）EJB 组件比 JavaBean 组件大得多；

5）JavaBean 既可以运行在服务器端(非可视化)，也可以运行在客户端(可视化)，是一种轻量级别的组件(可以不受容器的管理)。

6）EJB 只能部署并运行在服务器端，是一种重量级别的组件(受 EJB 容器管理)。

3、RMI(Remote Method Invocation)

RMI 属于 j2ee 规范之一，利用这种机制可以在不同机器(不同 jvm)之间进行远程调用。

4、IIOP(Internet-Inter ORB 协议) ———> 远程调用采用的协议

这种协议主要在分布式应用的远程调用中使用，IIOP 协议最早来源于 CORBAR 技术，利用这种协议也可以在 EJB 应用和非 Java 应用之间进行远程通信。

5、JNDI(Java 命名和目录接口) ———> 远程调用采用的技术

JNDI 属于 j2ee 规范之一，利用 JNDI 规范中定义的 API(javax.naming.*)可以从目录服务器上查找事先注册上去的资源(一些对象，如: DataSource)。

注：在 EJB 应用中这些资源是容器帮我们注册在目录服务器上的，我们只需要通过 JNDI 查找就行。

JNDI 使用方式：

```
Context ctx = new InitialContext();
DataSource ds = (DataSource)ctx.lookup("java:comp/env/jdbc/ds");
Connection conn = ds.getConnection();
//下面根据 conn 去创建不同的 Statement
...
```

Context：接口，位于包 javax.naming.*下

java:comp/env/：jndi 上下文路径

jdbc/ds：jndi 名字

目录服务器：

Tomcat、Weblogic 等服务器是自带的(内置)，不需要提供

DMS：独立，专门的目录服务器(提供定位目录服务器的 url，以及目录服务器的 Driver)

通过 api 去查找事先注册上去的资源：

java.naming.* ---> 主要提供给开发人员使用

java.naming.spi.* ---> 提供给目录服务器的开发商使用

spi：Server Provider Interface(服务提供者接口)

目录服务器：

jndi 名字	位置
jdbc/ds	DataSource 对象的引用
...	...

对象分为：

命名对象：不可再分，如 String

目录对象：可再分、有自己的属性，如 DataSource、Student 等

6、Stub(存根) 服务器端创建，由EJB容器在启动的时候创建

特点: 它是由 EJB 容器在服务器端创建的一个对象，但是它是运行在客户端的。

作用:

1) 将客户端的请求信息编组(序列化)成字节流后发送到服务器端。

2) 将服务器端返回的结果(字节流)解组(反序列化)成本地格式的数据信息(java 能识别的)。

服务器端创建，由EJB容器在启动的时候创建

7、Skeleton(骨架)

特点: 它是由 EJB 容器在服务器端创建的一个对象，但是它是运行在服务器端的。

作用:

1) 将客户端发送过来的请求信息(字节流)解组成本地格式的数据信息，然后根据这些请求信息去调用对应的 EJB 组件。

2) 将调用 EJB 组件返回的结果编组后 response 到客户端。

第二章 First EJB

1、EJB 规范中定义的接口使用说明

1) javax.ejb.EJBObject

定义一个接口继承这个接口，定义的接口称为 Remote interface，EJB 容器会根据这个接口产生一个对象，这个对象称为 EJB Object(EJB 对象)，在这个接口中定义“业务接口方法”供客户端远程调用。

2) javax.ejb.EJBHome

定义一个接口继承这个接口，定义的接口称为 Home interface，EJB 容器会根据这个接口产生一个对象，这个对象称为 Home Object(Home 对象)，在这个接口中定义“创建和删除 EJB 对象的接口方法”供客户端远程调用。

3) javax.ejb.EJBLocalObject

定义一个接口继承这个接口，定义的接口称为 Local interface，EJB 容器会根据这个接口产生一个对象， 这个对象称为 EJB Local Object(本地 EJB 对象)，在这个接口中定义业务接口方法供本地调用。

4) javax.ejb.EJBLocalHome

定义一个接口继承这个接口，定义的接口称为 Local home interface，EJB 容器会根据这个接口产生一个对象，这个对象被称为 Local Home Object(本地 Home 对象)，在这个接口中定义创建和删除本地 EJB 对象的接口方法供本地调用。

5) javax.ejb.SessionBean(extends javax.ejb.EnterpriseBean)

定义一个类实现这个接口，定义的类被称为 Bean class(EJB 组件)，EJB 容器会根据这个类型创建对应的一个或者若干个对象，这些对象被称为 Enterprise Bean instance(Bean 实例)，在这个 Bean 类中定义业务方法具体去实现业务逻辑。

2、部署描述符(deployment descriptor)

---> xml 文档

1) ejb 部署描述文件

ejb-jar.xml：描述定义的远程、本地接口、Bean 类

2) 容器厂商指定的专有部署描述文件(这里以 weblogic 为例)

weblogic-ejb-jar.xml：可以描述 JNDI 的名字

3、开发一个 EJB 组件(SessionBean)的步骤(针对 EJB2.x)：

1) 定义一个 Remote interface，去继承 javax.ejb.EJBObject，在此接口中定义一些业务接口方法供客户端远程调用。

2) 定义一个 Home interface，去继承 javax.ejb.EJBHome，在此接口中定义创建和删除 EJB 对象的接口方法供客户端远程调用。

3) 定义一个 Local interface，去继承 javax.ejb.EJBLocalObject，在此接口中定义一些业务接口方法供客户端本地调用。

4) 定义一个 Local home interface，去继承 javax.ejb.EJBLocalHome，在此接口中定义创建和删除本地 EJB 对象的接口方法供客户端本地调用。

5) 定义一个 Bean 类(EJB 组件)，去实现 javax.ejb.SessionBean，在 Bean 类中提供一些业务方法来封装业务逻辑。

6) 提供两个部署描述文件 ejb-jar.xml 和 weblogic-ejb-jar.xml。

4、EJB 应用中的两种类型的异常：

1) 应用级别异常

一些 checked 类型异常，分为：用户自定义的异常(如:HelloException) 和 EJB 中内置的异常(如:CreateException)，这些异常可以通过 try catch 捕获处理，当出现这些异常后可以直接抛给客户端。

2) 系统级别异常

一些 unchecked 类型异常(RumtimeException)，当出现这些异常后，EJB 容器会将其先包装成 java.rmi.RemoteException 或者其子类型，然后抛给客户端，但是出现系统级别异常会引起事务的回滚。

5、SessionContext<<interface>> (extends EJBContext)

这是 EJB 容器针对 SessionBean 所提供的一种上下文，利用此上下文可以将 Bean 实例和 EJB 容器关联起来，Bean 实例可以从此上下文中获得一些 EJB 容器的环境信息(如: EJB 对象、

Home 对象、UserTransaction 等)。

注意: 1) 在实例化 Bean 类后，EJB 容器会调用 setSessionContext(SessionContext context) 方法将事先由 EJB 容器构造好的 SessionContext 类型的对象传入到 Bean 实例中。

2) 一个 Bean 实例对应一个 SessionContext 类型的对象。

6、Stub 和 Skeleton 对象

1) Stub 对象：

1.home stub 对象 ---> 是根据 Home interface 构造出来的

2.EO stub 对象 ---> 是根据 Remote interface 构造出来的

2) Skeleton 对象：

1.home skeleton 对象 ---> 是根据 Home interface 构造出来的

2.EO skeleton 对象 ---> 是根据 Remote interface 构造出来的

```
public Hello create(){
    //创建 EJB 对象
    return EOStub;
}
```

7、EJB 中远程调用采用的模式

1) factory(工厂模式)

发生在：Home 对象(工厂)和 EJB 对象(产品)之间

流程：Home 对象作为工厂调用工厂方法 create()创建出 EJB 对象。

2) proxy(代理模式)

发生在：

*Home Stub 对象(代理对象)和 Home 对象(被代理对象)之间

*EO Stub 对象(代理对象)和 EJB 对象(被代理对象)之间

3) delegate(委托模式)

发生在：EJB 对象(代理对象)和 Bean 实例(被代理对象)之间

流程：当 EJB 对象接受到客户端的请求后并不会去处理，而是将请求委托给 Bean 实例去处理(调用 Bean 实例中的同名业务方法去完成)。

第三章 EJB 原理

1、EJB 的三种类型

1) Session Bean(会话 Bean)：

以同步方式来处理业务过程，提供一些业务方法来封装业务逻辑。

2) Entity Bean(实体 Bean)：

是一个持久化对象，代表了业务数据。

3) Message Driven Bean(消息驱动 Bean)：

以异步方式来处理跟消息有关的业务过程，利用消息驱动 Bean 可以从消息中间件上接受和处理消息。

2、中间件

这里指应用服务器(如:weblogic)

3、中间件服务

指 EJB 容器提供的事务、线程、安全、日志等方面的服务。

4、获得中间件服务的方式

只要调用 EJB 容器所专有的 API(如:事务的 API 等)就可以获得 EJB 容器所提供的中间件服务(事务、线程、安全、日志等服务)。

5、显式中间件

在 Bean 实例中显式地去调用 EJB 容器所专有的 API(如:事务 API)来获得 EJB 容器提供的相应的中间件服务。

```
//通过 SessionContext 获得 UserTransaction 对象
UserTransaction userTx = context.getUserTransaction();
try {
        userTx.begin();
        System.out.println("启动事务");
        userTx.commit();
        System.out.println("提交事务");
} catch (Exception e) {
        try{
                userTx.rollback();
        }catch (Exception e2) {
                e2.printStackTrace();
        }
}
```

```
public class HelloBean implements SessionBean{
        ...
        //转帐
```

```
        transfer(){
            try{
                obj.begin();
                ...
                obj.commit();
            }catch(Exception e){
                obj.rollback();
            }
        }
    }
```

6、隐式中间件(默认)

在 Bean 实例中不用显式地去调用 EJB 容器所专有的 API，而是由请求拦截器(EJB 对象)去调用 EJB 容器所专有的 API 来获得 EJB 容器提供的中间件服务。

如：调用事务的 API

EJB 对象调用 obj.begin();

EJB 对象调用 Bean 实例中的业务方法

EJB 对象调用 obj.commit();

注意: 1) 请求拦截器调用那些 API 是在调用 Bean 实例的业务方法之前进行。

2) 若采用隐式中间件的话，必须在 ejb-jar.xml 中描述要提供的服务，这样请求拦截器才会去调用对应的 API，来获得 EJB 容器提供的相应服务。

```
<session>
    <!--<transaction-type>Container</transaction-type>--> //隐式
    <transaction-type>Bean</transaction-type>                    //显示
</session>
```

3) 以后基本上都是采用隐式中间件。

第四章 Session Bean(会话 Bean)

1、Session Bean 的分类

1）stateful session bean(有状态会话 Bean)

a)使用场合：

当客户端完成某个业务功能要向服务器端发起多次请求，调用服务器端 Bean 实例中的多个业务方法，并且这些业务方法中要使用从客户端传递过来的参数(客户端的状态信息)或者某些业务方法执行后产生的中间结果，那么在 Bean 实例中要提供变量来保存这些参数和中间结果以便其它被调用的业务方法使用。

b)特点

1.一个 Bean 实例只能被一个客户端使用；

2.可以保存客户端的状态信息或者一些业务方法执行后产生的中间结果。

客户端完成某个业务功能 A，需调用服务器端 Bean 实例中的多个业务方法才能实现：

```
public class BeanClass{
    private int total;
    public void a(int initData){
        this.total = initData;
    }
    public void b(int increment){
        this.total += increment;
    }
    public double c(){
        return total * 0.5;
    }
}
```

客户端完成功能 A：

```
a(10);  //request1 --> B1(一个 Bean 实例对应一个客户端)
b(5);   //request2 --> B1
double result = c();   //request3 --> B1
```

2）stateless session bean(无状态会话 Bean)

a)使用场合

客户端完成某个业务功能只要向服务器端发起一个请求，调用 Bean 实例中的一个业务方法即可。

b)特点

1.一个 Bean 实例可以被多个客户端共享使用。

2.也可以提供变量来保存客户端的状态信息或者中间结果，但是没有任何意义，因为当客户端向服务器发起一个请求后，EJB 容器会随机地选择一个 Bean 实例来处理请求。

2、在无状态会话 Bean 中引入 pool 机制的目的：

1）解决 EJB 中采用单线程模式，提高并发处理性能。

2）控制内存中 Bean 实例的数量，减少资源的开销。

注意：默认已经引入 pool，但是没有设置上限，并且 pool 中是空的。

在 EJB 中采用单线程方式访问 Bean 实例会降低并发访问的性能，那么在无状态会话 Bean 中引入 pool 来解决这个问题，开发人员可以在 pool 中预先放置几个事先创建好的 Bean

实例，这些 Bean 实例可以同时处理客户端的多个请求提高并发访问性能，也可以设置 pool 中放置 Bean 实例的上限。

3、pool 的运行原理

1) 当客户端向服务器端发起一个请求调用 pool 中的 Bean 实例时，EJB 容器会先判断 pool 中是否有空闲的 Bean 实例，若有，那么 EJB 对象(Wrapper 对象)就会将该请求委托给该 Bean 实例来处理。

2) 若没有空闲并且 pool 中的 Bean 实例数量没有到达设置的上限，那么 EJB 容器会创建一个新的 Bean 实例放置在 pool 中，然后那么 EJB 对象(Wrapper 对象)就会将该请求委托给这个新的 Bean 实例来处理。

3) 若没有并且 pool 中的 Bean 实例数量已经到达设置的上限，那么 EJB 容器不会再去创建一个新的 Bean 实例，此时，客户端的这个请求只能等待，等到有空闲的 Bean 实例时再去处理请求。

注意：在无状态会话 Bean 中默认提供了 pool 机制，但是并没有在 pool 中放置任何创建好的 Bean 实例和设置放置 Bean 实例的上限。

4、在无状态会话 Bean 中创建 Bean 实例的方式

1) 在 EJB 容器装载 EJB 应用时创建；

前提：在 weblogic-ejb-jar.xml 中描述一个<pool>，设置预先创建 Bean 实例的数量和上限。

```
<stateless-session-descriptor>
    <pool>
        <max-beans-in-free-pool>3</max-beans-in-free-pool>   //设置 Bean 实例的上限
        <initial-beans-in-free-pool>2</initial-beans-in-free-pool>//预先创建的数量
    </pool>
</stateless-session-descriptor>
```

2) 当客户端发起一个请求调用 pool 中 Bean 实例的业务方法时可能会创建。

5、在有状态会话 Bean 中引入 cache 机制的目的

1）控制内存中 Bean 实例数量；

2）提高处理并发访问性能。

由于在有状态会话 Bean 中只要客户端调用 create 方法就会触发服务器端创建一个新的 Bean 实例，为了控制内存中 Bean 实例的数量就引入 cache 来放置这些创建出来的 Bean 实例，当然利用 cache 机制也可以提高并发访问的性能，在 cache 中可以设置放置 Bean 实例的上限，但是不能预先创建 Bean 实例放置在 cache 中。

注意: 在有状态会话 Bean 默认没有提供 cache 机制，必须要引入，方式是在 weblogic-ejb-jar.xml 中描述一下 cache 即可，设置上限和钝化的策略(如:LRU-->最近最少使用策略)。

```
<stateful-session-descriptor>
<stateful-session-cache>
        <!-- 设置 cache 的上限，注意最多为 2 个并且不能预先去创建 -->
        <max-beans-in-cache>2</max-beans-in-cache>
        <cache-type>LRU</cache-type>
</stateful-session-cache>
</stateful-session-descriptor>
```

6、cache 的运行原理

1) 当客户端调用 create 方法向服务器端发起一个请求时就会触发服务器端(Home 对象)创建一个新的 Bean 实例并将其放置到 cache 中，但是在放置之前 EJB 容器会判断此时 cache 是否已满，若没有满那么直接将这个新的 Bean 实例放置到 cache 中。若已满那么 EJB 容器会按照某种策略(如:LRU)将 cache 中的一个或者多个 Bean 实例钝化(序列化)到外存中，然后将这个新的 Bean 实例放置到 cache 中。

2) 当客户端发起一个请求调用 cache 中某个 Bean 实例的业务方法时，EJB 容器会判断该 Bean 实例是否在 cache 中，若在，那么 EJB 对象(Wrapper 对象-->EJB3.0)就将请求委托给这个 Bean 实例来处理，若不在并且在外存中，EJB 容器先将该 Bean 实例激活(反序列化)进 cache 中，然后 EJB 对象再将请求委托给这个 Bean 实例来处理。

3) 在激活之前 EJB 容器也会判断此时 cache 是否已满，若满的话要先将 cache 中的 Bean 实例钝化，然后再激活。

7、钝化(序列化)和激活(反序列化)

1) 钝化(Swap out 或者 passivate)

将 cache 中的 Bean 实例序列化到外存中的一个过程。

注意：不能将正处于事务中的 Bean 实例进行钝化，必须等到事务结束方可钝化。

2) 激活(Swap in 或者 activate)

将钝化到外存中的 Bean 实例反序列化到 cache 中的一个过程。

8、在有状态会话 Bean 中创建 Bean 实例的方式

客户端通过 jndi 的 lookup 查找时就创建 Bean 实例（即：当客户端调用 create 方法触发服务器端(Home 对象)时创建 Bean 实例）。

9、无状态会话 Bean 中的生命周期回调方法

1) ejbCreate()

在无状态会话 Bean 中只能定义无参的 ejbCreate()，在此方法中可以进行一些资源的初始化工作(如:获得 Connection 等)，在调用 setSessionContext()之后 EJB 容器会调用此方法。

2) ejbRemove

此方法在 Bean 实例销毁之前被调用，在此方法中可以释放一些资源(如:Connection 等)

在下列情况下此方法被调用:

a) EJB 应用被卸载 EJB 容器去销毁 pool 中所有 Bean 实例前调用。

b) pool 中的 Bean 实例长时间不用并超过指定的时间时，EJB 容器在销毁这些 Bean 实例前调用。

注意：在客户端调用 remove()想销毁 Bean 实例是没有作用的。

3) ejbPassivate/ejbActivate()

这两个方法在无状态会话 Bean 中永远不会被调用。

4) setSessionContext()

此方法在实例化 Bean 之后被 EJB 容器调用，用于往 Bean 实例中传入一个事先由 EJB 容器构造好的 SessionContext 类型的对象。

注意：一个 Bean 实例对应一个 SessionContext。

10、有状态会话 Bean 中的生命周期回调方法

1) ejbCreate(args)/ejbCreate()

在有状态会话 Bean 有参和无参的 ejbCreate 方法都可以定义(可以同时定义，但是一般是定义有参的)，并且参数可以定义多个。在此方法中可以进行一些资源的初始化工作(如:获得 Connection 等)，在调用 setSessionContext()之后 Home 对象会调用此方法。

2) ejbRemove

此方法在 Bean 实例销毁之前被调用，在此方法中可以释放一些资源(如:Connection 等)

在下列情况下此方法被调用:

a) 在客户端调用 remove 方法后 EJB 对象在销毁对应的 Bean 实例前调用。

b) EJB 应用被卸载 EJB 容器去销毁 cahce 中所有 Bean 实例前调用。

c) cache 中的 Bean 实例长时间不用并超过指定的时间时，EJB 容器在销毁这些 Bean 实例前调用。

3) ejbPassivate

此方法在 Bean 实例钝化之前被 EJB 容器调用。

4) ejbActivate()

此方法在 Bean 实例激活之后被 EJB 容器调用。

5) setSessionContext()

此方法在实例化 Bean 之后被 Home 对象调用，用于往 Bean 实例中传入一个事先由 Home 对象构造好的 SessionContext 类型的对象。

11、Session Bean 中销毁 Bean 实例的方式

1）有状态会话 Bean

1.客户端调用方法(如：remove())触发 EJB 容器销毁 Bean 实例；

2.EJB 应用卸载时 EJB 容器会销毁所有的 Bean 实例。

2）无状态会话 Bean

只有 EJB 应用卸载时 EJB 容器会销毁所有的 Bean 实例，客户端调用方法想销毁没有作用。

第五章 消息驱动 Bean(MDB)

1、 MOM：Message-Oriented Middleware(消息中间件)，也叫 JMS Server。

JMS：Java Message Service(Java 消息服务)，是 j2ee 规范之一。

1）api：位于包 javax.jms.*下，供开发人员使用。

2）spi：供消息中间件的开发商使用。

2、JMS 中处理消息的两种主要方式

1) publish/subscribe(pub/sub)：发布/订阅

Topic：MOM 的目的地

特点:

a) 有多个消息生产者和多个消息消费者

b) 一条消息可以被多个消息消费者消费

c) 一个消息消费者也可以消费多条消息
d) 一条消息被消费后不会被 MOM 删除

2) point-to-point(PTP)

Queue：MOM 的目的地

特点:

a) 有多个消息生产者和一个消息消费者
b) 一条消息仅能被一个消息消费者消费
c) 一个消息消费者也可以消费多条消息
d) 一条消息被消费后会被 MOM 删除

3、消息订阅者(针对 publish/subscribe 方式)

1）非持久订阅者(默认)

必须先注册到 MOM 上才能消费到消息

2）持久订阅者

晚注册到 MOM 上也能消费到消息

注意：持久和非持久订阅仅出现在 publish/subscribe 方式中，在 ptp 中没有，但是在 ptp 中不管是否先或者晚注册都能消费到消息。

4、使用 MDB 的目的

1) 让 MDB 作为一个消息消费者可以简化程序代码，如:

* 让 EJB 容器从目录服务器上查找 MOM 驱动和目的地，并且创建 Connection 和 Session。

* EJB 容器将 MDB 注册到 MOM 上(在 EJB 应用装载时)

* 让 EJB 容器负责去接受 JMS 消息而 MDB 只要去接受 EJB 容器转交的消息进行处理即可。

2) MDB 运行在服务器端，是受 EJB 容器管理

可以获得 EJB 容器提供的中间件服务。

5、MDB(默认为非持久订阅者)消费消息的过程：

1) 当 EJB 容器装载 EJB 应用时(即 Weblogic 启动时)，先将所有的 MDB 组件注册到 MOM 上。

2) 当 EJB 容器从 MOM 中指定的目的地(EJB2.x 中需在 weblogic-ejb-jar.xml 中描述指定)接受到 JMS 消息后会转交给 MDB 来处理，方式如下：

* publish/subscribe(topic)

EJB 容器调用所有 MDB 组件中的某一个空闲的 Bean 实例的 onMessage()方法，将 JMS 消息转交这个 Bean 实例处理。

* ptp(queue)

EJB 容器调用某一个 MDB 组件中的某个空闲 Bean 实例的 onMessage 方法来处理。

********** EJB 3.0 **********

EJB3.0

第一章 EJB3.0 Foundation

1、开发一个 EJB 组件(针对 SessionBean)的步骤：

1）定义一个 Business interface(Remote/Local)，在此接口中定义一些业务接口方法供客户端远程或者本地调用，但是此接口不要继承 EJB3.0 规范中定义的任何接口。

2）定义一个 Bean 类实现 Business interface 即可，不用去实现任何组件接口(如：javax.ejb.SessionBean）。

3）提供一个 Weblogic-ejb-jar.xml(可选，当配置 pool 或 cache 时提供)。

注：1.在 EJB3.0 中以 Annotation 取代了 2.0 中的 ejb-jar.xml 文件。

2.Weblogic9 不支持 Annotation，所以 EJB3.0 只能采用 Weblogic10。

3.放 jar 文件到 base_domain 的 lib 目录下，进入 /home/briup/bea/user_projects/domains/base_domain/bin，用记事本打开 setDomainEnv.sh，然后拉到最后配置 CLASSPATH，在后面加上：${DOMAIN_HOME}/lib/*.jar，以便能够找到这些 jar 文件。 (*/)

4.创建 pointbase 数据库：运行 /home/briup/bea/wlserver_10.0/common/eval/pointbase/tools/startPointBase.sh 和 startPointBaseConsole.sh

pointbase 的 URL：jdbc:pointbase:server://localhost/xuxiangdb

2、EJB3.0 中开发 EJB 组件简化 API 的方式

1）提供一个 Business interface

2）提供一个 Bean 类实现 Business interface 即可

3）使用 Annotation 来替代 ejb-jar.xml

4）简化访问注册在目录服务器上资源的方式：

EJB2.x：通过 JNDI 的 lookup 方式访问

EJB3.0：a) 通过 JNDI 的 lookup 方式访问

b) 通过依赖注入(IOC/控制反转)访问

依赖注入: 将要查找的资源直接注入到要使用该资源的对象中。

注意：通过 JNDI 方式能访问到的资源都可以通过依赖注入来解决，反之不一定。

3、要求完成本地和远程调用的两种方式：

1）本地调用

1.将 HelloEJB.jar 和 HelloWEB.war 复制到 autodeploy 目录下

调用方式：http://localhost:7001/HelloWEB/hello.jsp

填完表单后提交

2.将 HelloEJB.jar 和 HelloWEB.war 打包成 HelloAPP.ear，方式如下：

创建目录结构：

```
HelloAPP---META-INF---application.xml
                    |-HelloEJB.jar
                    |-HelloWEB.war
```

```
cd HelloAPP
jar -cvf HelloAPP.ear *
```

将 HelloAPP.ear 复制到 autodeploy 中

调用方式：同上

2）远程调用

1.将 HelloWEB.war 复制到 Tomcat 的 webapps 目录下，并在 common/lib 中复制一个 weblogic.jar(采用 weblogic9 版本的 jar 文件)

2.将 HelloEJB.jar 复制到 weblogic10 的 autodeploy 目录下

3.启动 Tomcat 和 weblogic10

http://localhost:8080/HelloWEB/hello.jsp

第二章 Session Bean

1、在 EJB3.0 中定义 SessionBean 生命周期回调方法的方式(可选)

1) 定义在 Bean 类中

```
public class BeanClass{
@AnnotationName
public void method(){}
...
}
```

注意: 必须是无参的方法

2) 定义在回调监听器类中

```
public class CallBackListener{
    @AnnotationName
public void method(InvocationContext ctx){}
...
}
```

将回调监听器类和 Bean 类关联的方式:

```
@Interceptors(CallBackListener.class)
public class BeanClass{}
```

2、依赖注入(IOC/控制反转)

依赖容器或者一些框架(如:spring 等)将一个对象传入(注入)到另外一个对象的过程。

注意：这些对象都是由容器或者框架所构造

EJB3.0 中依赖注入的方式：

1) 通过 set 方法注入

特点: 将@Resource 注释在 set 方法前

2) 通过反射方式注入

特点: 将@Resource 注释在字段前

3) 通过 EJBContext 注入

特点: 将@Resource 注释在类前，EJB 容器先将 resource 注入到 EJBContext 中，再通过 EJBContext 的 lookup 方法来获得。

```
public class User{
    private Address address;
    //方式一：通过构造器注入
    public User(Address address){  --> 注入
        this.address = address;
    }
    public void setAddress(Address address){
        this.address = address;
    }
}
public class Address{}
main():
    User user = new User();
    Addreee address = new Address();
    //方式二：通过 set 方法注入(最常用的)
```

```
user.setAddress(address);      ---> 注入
```

第三章 消息驱动 Bean(MDB)

在 EJB3.0 中使用 MDB 作为消息消费者的方式：

注意：必须要实现 MessageListener 接口。

```
@MessageDriven(
        mappedName="jms/fileTopic",
        activationConfig={

@ActivationConfigProperty(propertyName="acknowledgeMode",propertyValue="Auto-acknowledge"),


@ActivationConfigProperty(propertyName="destinationType",propertyValue="javax.jms.Topic")
        }
)
public class WelcomeBean implements MessageListener {
        public void onMessage(Message arg0) {
                System.out.println("into onMessage()...");
                if(arg0 instanceof TextMessage){
                        TextMessage txtMsg = (TextMessage) arg0;
                        try {
                                System.out.println(txtMsg.getText());
                        } catch (JMSException e) {
                                e.printStackTrace();
                        }
                        System.out.println("消费消息成功！");
                }
        }
}
```

第四章 JPA(Java Persistence API)

对象模型	关系模型
类	表
对象	记录
属性	字段
OID	主键(PK)

1、JPA(Java Persistence API)

1) JPA 属于 EJB3.0 规范的一个重要组成部分，主要用来持久化的。

2) 在 JPA 中推崇 POJO 编程模型，这里的 POJO 就是一些 Entity。

3) 在 JPA 中支持基于 ORM 的技术或者持久化框架(如:Hibernate、TopLink 等)。

4) 在 JPA 的程序测试中可以脱离 EJB 容器，在 j2se 环境下进行。

5) 在 JPA 规范中提供了两套 API，分别是：

a)提供给 ejb 开发人员使用的 API(javax.persistence.*)：

利用这套 API 可以和数据库进行 CRUD 的操作。

b)提供给持久化框架开发商使用的 SPI(javax.persistence.spi.*)：

不同持久化框架的开发商要提供一个 Driver(持久化提供者，该驱动中封装了和不同数据库交互的细节)去实现其中的一个接口 javax.persistence.spi.PersistenceProvider。

2、Persistence Context(持久化上下文)

注意：我们访问实体必须要通过持久化上下文。

1）持久化上下文是 EntityManager(实体管理器)维护的一个缓存，我们调用 EntityManager 中的 API 将实体实例放置到持久化上下文中(与之关联)，可以从持久化上下文中获得实体实例。

Hibernate：

```
Student student = new Student();      //临时状态
session.save(student);  //持久化状态
```

JPA:

```
Account accouont = new Account);
    ---> Account 实例处于临时状态
EntityManager.persist(account);
    ---> Account 实例变成持久状态(受管状态)，那么该实体实例就和持久化上下文关联
```

2）持久化上下文的分类:

1.普通的持久化上下文(默认)

a.这种上下文的生命周期仅限于一个事务的范围，当启动一个事务就会产生一个普通的持久化上下文，当事务结束(提交或者回滚)，这种上下文也就随之结束。

b.这种上下文一般用于无状态会话 Bean 中。

```
@PersistenceContext
```

```
private EntityManager em;
```

2.扩展的持久化上下文

a.这种上下文的生命周期限于一个 Bean 实例生命周期的范围，当产生一个新的 Bean 实例，也就产生一个扩展的持久化上下文，这种上下文可以横跨若干个事务，当 Bean 实例被销毁，这种上下文也就结束。

b.这种上下文一般用于有状态会话 Bean 中。

c.设置扩展的持久化上下文的方式：

```
public class BeanClass{

@PersistenceContext(type=PersistenceContextType.EXTENDED,
                    [unitName="持久化单元的名字"])
    private EntityManager em;
    ...
}
```

注意: 若 persistence.xml 中只有一个持久化单元，那么 unitName 可以不写。

3、创建实体管理器实例的方式：

1）提供 PersistenceProvider 接口的实现类(驱动)，创建一个 PersistenceProvider 实例。

META-INF/persistence.xml

kodo：Weblogic 自带的一个持久化框架

2）EntityManagerFactory emf = Persistence.createEntityManagerFactory("持久化单元的名字");

3）EntityManger em = emf.createEntityManager();

4、将实体管理器实例注入到 Bean 实例中的方式：

```
public class BeanClass{
    @PersistenceContext
    private EntityManager em;
    ...
}
```

5、管理实体管理器的两种方式：

1) 容器管理实体管理器(推荐使用)

a) 将 EJB 容器创建好的实体管理器实例通过@PersistenceContext 注入到 Bean 实例中。

b) 先将 EJB 容器创建好的实体管理器实例注入到 SessionContext 中，再调用其中的 lookup 方法来获得实体管理器实例。

2) 应用管理实体管理器(脱离 EJB 容器)

开发人员在应用中自己编写代码来创建实体管理器实例，并将其注入到自己创建好的 Bean 实例中。

6、实体生命周期中的四种状态

状态	持久化上下文	数据库
new(新建)	未关联	无记录
managed(受管)	关联	有记录
detached(游离)	未关联	有记录
removed(删除)	关联	无记录

7、定义实体生命周期回调方法的方式

1) 实体监听器类：定义实体生命周期的回调方法

```
public class Listener{
        @AnnotationName
        public void method(Entity o){}
        ...
}
```

实体监听器类和实体关联方式：

```
@EntityListeners(Listener.class)
public class EntityClass{}
```

2) 定义在实体中

```
public class EntityClass{
        @AnnotationName
        public void method(){}
}
```

注意: 当实体状态发生变化，持久化提供者自动会调用对应的这些实体生命周期回调方法。

8、和持久化上下文关联的实体实例与数据库之间的同步

1) 实体实例同步到数据库

a) 调用 EntityManager.flush()进行同步

b) 设置同步模式：

EntityManager.setFlushMode(FlushModeType.AUTO/FlushModeType.COMMIT);

AUTO(默认): 表示在查询操作或者提交时自动进行同步。

COMMIT: 仅在提交时自动进行同步。

2) 数据库同步到实体实例

调用 EntityManager.refresh()

9、在 JPA 中处理事务并发访问的方式：

1) 设置乐观锁

在实体中定义一个属性 version，并用@Version 进行注释，这样持久化提供者自动会在表中创建一个 version 的字段(用于版本控制)。属性 version 的类型必须是下列类型：

int、Integer、short、Short、long、Long、java.sql.Timestamp(插入到数据库中可以精确到毫秒，而 java.util.Date 插入到数据库中则精确到天)。

2) 设置读/写锁(悲观锁)

调用 EntityManager.lock(Entity o,LockModeType.READ/LockModeType.WRITE);

10、一个 Bean 类去调用其它 Bean 类的方式：

1) EJB2.x

JNDI 方式(依赖查找)

2) EJB3.0

1. JNDI 方式(依赖查找)

2. 依赖注入

第五章 Advanced Persistence Concepts(JPA-高级持久化)

1、实体的继承关系

1）整个继承结构(类图)映射一张表

a) 根类中使用 Annotation

```
@Entity(name="entityName")
@Table(name="single_tbl")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)

@DiscriminatorColumn(name="DISC",discriminatorType=DiscriminatorType.STRING)
@DiscriminatorValue("rootClassDesc")
public class RootClassName{}
```

b) 子类中使用 Annotation

```
@Entity(name="entityName")
@DiscriminatorValue("subClassDesc")
```

```
public class SubClassName{}
```

2）每个类映射一张表

a) 根类中使用 Annotation

```
@Entity(name="entityName")
@Table(name="root_joined_tbl")
@Inheritance(strategy=InheritanceType.JOINED)
public class RootClassName{}
```

b) 子类中使用 Annotation

```
@Entity(name="entityName")
@Table(name="sub_joined_tbl")
public class SubClassName{}
```

3）每个子类映射一张表(不支持多态)

a) 根类中使用 Annotation

```
@MappedSuperclass
public class RootClassName{}
```

b) 子类中使用 Annotation

```
@Entity(name="entityName")
@Table(name="sub_concrete_tbl")
public class SubClassName{}
```

继承关系	支持多态	维护更新	出具报表
每个类映射一张表	支持	易	难
每个子类映射一张表	不支持	难	易
整个类图映射一张表	支持	难	易

2、实体的关联关系

1）1:1 (如:Order 和 Shipment)

a) 单向

1 方(Order)中使用 Annotation:

```
@Entity(name="entityName")
@Table(name="order_uni_tbl")
public class Order{
   ...
   @OneToOne(cascade=CascadeType.ALL,fetch=FetchType.EAGER)
   @JoinColumn(name="shipment_id",unique=true,referencedColumnName="id")
   private Shipment shipment;
}
```

1 方(Shipment)中使用 Annotation:

```
@Entity(name="entityName")
@Table(name="shipment_uni_tbl")
public class Shipment{}
```

b) 双向

1 方(Order)中使用 Annotation:

```
@Entity(name="entityName")
@Table(name="order_bi_tbl")
public class Order{
   ...
   @OneToOne(cascade=CascadeType.ALL,fetch=FetchType.EAGER)
   @JoinColumn(name="shipment_id",unique=true,referencedColumnName="id")
   private Shipment shipment;
}
```

1 方(Shipment)中使用 Annotation:

```
@Entity(name="entityName")
@Table(name="shipment_bi_tbl")
public class Shipment{
   ...
   @OneToOne(mappedBy="shipment")
   private Order order;
}
```

2）1:M (如:Company 和 Employee)

a) 单向

1 方(Company)中使用 Annotation:

```
@Entity(name="entityName")
@Table(name="company_uni_tbl")
public class Company{
   ...
   @OneToMany(cascade=CascadeType.ALL,fetch=FetchType.EAGER)
   [
    @JoinTable(
         name="company_employee_uni_tbl",

joinColumns={@JoinColumn(name="companyid",referencedColumnName="id")},

inverseJoinColumns={@JoinColumn(name="employeeid",referencedColumnName="id")})
   ]
   private Collection<Employee> employees = new ArrayList<Employee>();
}
```

注意: @JoinTable 可以不加，持久化提供者默认会创建一个中间表(桥表)。

M方(Employee)中使用 Annotation:

```
@Entity(name="entityName")
@Table(name="employee_uni_tbl")
public class Employee{}
```

b) 双向

1方(Company)中使用 Annotation:

```
@Entity(name="entityName")
@Table(name="company_bi_tbl")
public class Company{
   ...

@OneToMany(cascade=CascadeType.ALL,fetch=FetchType.EAGER,mappedBy="company")
   private Collection<Employee> employees = new ArrayList<Employee>();
}
```

M方(Employee)中使用 Annotation:

```
@Entity(name="entityName")
@Table(name="employee_bi_tbl")
public class Employee{
   ...
   @ManyToOne
   @JoinColumn(name="company_id",referencedColumnName="id")
   private Company company;
}
```

3）M:N (如:Student 和 Course)

a) 单向

M方(Student)中使用 Annotation:

```
@Entity(name="entityName")
@Table(name="student_uni_tbl")
public class Student{
   ...
   @ManyToMany(cascade=CascadeType.ALL,fetch=FetchType.EAGER)
   @JoinTable(name="student_course_uni_tbl",

joinColumns={@JoinColumn(name="student_id",referencedColumnName="id")},

inverseJoinColumns={@JoinColumn(name="course_id",referencedColumnName="id")}
   )
   private Collection<Course> courses = new ArrayList<Course>();
}
```

N方(Course)中使用 Annotation:

```
@Entity(name="entityName")
@Table(name="course_uni_tbl")
```

```
public class Course{}
```

b) 双向

M 方(Student)中使用 Annotation:

```
@Entity(name="entityName")
@Table(name="student_bi_tbl")
public class Student{
   ...
   @ManyToMany(cascade=CascadeType.ALL,fetch=FetchType.EAGER)
   @JoinTable(name="student_course_bi_tbl",

joinColumns={@JoinColumn(name="student_id",referencedColumnName="id")},

inverseJoinColumns={@JoinColumn(name="course_id",referencedColumnName="id")}
   )
   private Collection<Course> courses = new ArrayList<Course>();
}
```

N 方(Course)中使用 Annotation:

```
@Entity(name="course_bi_entity")
@Table(name="course_bi_tbl")
public class Course{
   ...


@ManyToMany(cascade=CascadeType.ALL,fetch=FetchType.EAGER,mappedBy="courses")
   private Collection<Student> students = new ArrayList<Student>();
}
```

3、Join Operation(连接)

```
company              employee
-------------------  ------------------------------------------------------
id      name         id              name          company_id
1       briup        10              kevin              1
                     11              terry            null
```

select c.name,e.name from company c,emplyee e where e.company_id=c.id(+)

左连接：将左侧的表中无关联的记录也全部查出来，注意是 Oracle 特有的

1）Left join(左连接)：

select c.name,e.name from company c left outer join employee e on e.company_id=c.id;

2）Inner join(内连接)：

select c.name,e.name from company c inner join employee e on e.company_id=c.id;

3）Fetch join

例：public List<Company> joinFind() {
Query query = em.createQuery("select c from company_bi_entity c"+" left join c.employees e");
return query.getResultList();
}

4、Group By 和 Having 子句

1）Group By 子句：

```
public List<Object[]> groupByFind() {
    Query query = em.createQuery("select e.gender,count(*) "+
        "from employee_bi_entity e "+"group by e.gender");
    return query.getResultList();
}
```

注意：如果不指定，则默认封装成一个集合 Vector{Object[],Object[],...}，返回 Object 数组。

Object[] ---> 一条记录

Object[]中的元素 ---> 记录中的字段

2）Having 子句：

"select e.gender,count(*) from employee_bi_entity e group by e.gender having e.gender=?"

5、投影(Projection)：

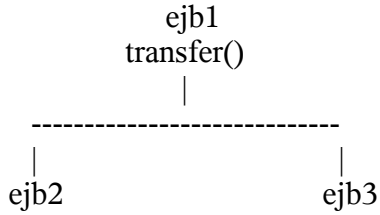
select name from company;

6、Object Construction in SELECT Statements

例：public List<CompanyEmployeeInfo> selectConstruct() {
Query query = em.createQuery("select new com.briup.relationship.one2many.bi.CompanyEmployeeInfo(c.name,e.name) "+"from company_bi_entity c join c.employees e");
return query.getResultList();
}

第六章 事务

1、分布式事务：指事务分布在多台数据库中(两阶段提交)。



withdraw() deposit()
| |
Oracle1 Oracle2

注意：数据库必须支持分布式事务，如 Oracle。

2、EJB 事务中的一些术语

1) 事务对象(事务组件)

指一些用来封装业务逻辑的 EJB 组件，在此组件中可以运行若干个事务。

2) 事务管理器

用来管理事务组件的事务操作，并且它是受 EJB 容器管理的，我们说 EJB 容器可以提供相应的事务服务(中间件服务)，实际上是 EJB 容器调用事务管理器来完成的。

3) 资源

一些持久源(文件、数据库等)，这里指数据库。

4) 资源管理器

指不同数据库的驱动。

3、JTA(Java Transaction API)：

属于 j2ee 规范之一，专门用来处理事务的，EJB 中事务是基于 JTA 的。JTA 规范中定义了一套用于处理事务的 API(javax.transaction.*)，主要提供了三大类接口：

1）javax.transaction.UserTransaction

---> 主要用于显示中间件

2）javax.transaction.TransactionManager

---> 主要用于隐式中间件

EJB2.x：EJB 对象(请求拦截器)

EJB3.0：Wrapper 对象(请求拦截器)

3）javax.transaction.xa.XAResource

---> 主要用于分布式事务中

注意：事务管理器实际是对上述接口的实现

启动事务：

EJB 事务：UserTransaction.begin();

Hibernate 事务：Transaction.begin();

JDBC 事务：Connection.begin();

注意：EJB 事务、Hibernate 事务都是基于 jdbc 事务的，它们只是对其进行了封装而已。

4、使用事务的动机(目的)：

ACID 特性（原子性-->主要动机、一致性、隔离性、持久性）

1) 原子操作(Atomic oprations)

将完成某个业务功能的一系列操作作为一个整体放在一个事务中进行，当某个操作失败，那么整个事务要回滚，只有当所有操作成功后事务才能正常地提交。

2) 在分布式应用中，当网络中断或者服务器瘫痪也要介入事务的控制。

3) 当多个用户同时去访问数据库中同一个数据时也要介入事务的控制。

5、事务的分类：

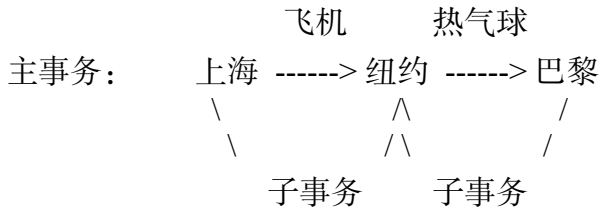
1）平面型事务(Flat transactions)

特点：事务的原子性，就一个事务。

2）嵌入型事务(Nested transactions，旅行商事务)

特点：由一个主事务和若干个子事务构成，其中一个子事务发生回滚，不会影响到主事务的回滚，只有当所有子事务提交后，主事务才能正常地提交。

例：



6、EJB 处理事务的三种方式：

1）CMT(Container Managed Transaction)

---> 隐式中间件

注意: 这是默认方式，并且推荐使用。

2）BMT(Bean Managed Transaction)

---> 显示中间件

3）客户端管理事务

在客户端(Servlet、Action 等)中调用事务的 API 来启动、提交或者回滚事务。

7、EJB 事务属性的取值(针对 CMT):

1) Required(默认)

若客户端在调用 EJB 组件时已经启动事务，则 EJB 组件参与其中；若没有启动事务，则 EJB 容器会为 EJB 组件启动新的事务。

2) RequiresNew(不管怎么样，都会启动新的事务)

若客户端在调用 EJB 组件时已经启动事务，则当前事务会被挂起，进而 EJB 容器会启动新的事务，EJB 组件执行相应的业务操作，EJB 容器会提交或者回滚事务。然后 EJB 容器会恢复原有的被挂起的事务。当然，若客户端在调用 EJB 组件时不存在事务，则不需要执行事务的挂起或恢复操作。

3) Supports(不会启动新的事务)

只有当调用 EJB 组件的客户端已经启动了事务时，这个 EJB 组件才会运行在事务中，反之 EJB 组件不会运行在事务中。

4) Mandatory

要求调用 EJB 组件的客户端必须运行在事务中，若未运行在事务中，则会抛出异常。

5) NotSupported(不支持)

EJB 组件根本不会参与到事务中。

6) Never

EJB 组件不能够参与到事务中，而且若调用它的客户端已经处于事务中，则容器会抛出 EJBException 异常。

8、分布式事务

1）二阶段提交方式：在 EJB 中采用二阶段提交方式来解决分布式事务(JTA 规范中指定的)。

第一阶段:

当执行提交命令 commit()后，主事务管理器会向网络中所有的事务管理器发出一个准备提交的命令，这些事务管理器接受到这个命令后都会返回一个运行报告(包含了执行的 sql 语句以及执行的异常信息)给主事务管理器，主事务管理器会判断这些运行报告中的 sql 执行是否正常，若正常那么将这些 sql 语句记录到事务日志中，便于以后由于数据库瘫痪或者其它原因导致数据丢失，可以用来执行事务日志中这些 sql 语句重新获得数据。

第二阶段:

当第一个阶段成功后，主事务管理器才会向所有的事务管理器发出一个真正提交的命令，所有事务管理器接受这个命令后，会通知对应的资源管理器(数据库驱动)将数据真正提交到数据库中。最后，所有事务管理器也要返回一个运行报告给主事务管理器。

2）对于分布式事务开发人员要做的事情：

1. 配置的数据源要支持二阶段提交；

2. 选择的数据库要支持分布式事务。

9、EJB 中用到的一些模式：

1）Facade(门面模式)：

SessionBean：SessionFacade ---> 充当门面

1.提供一些粗粒度方法

2.通过调用实体管理器方法 ---> 细粒度方法

2）DTO(Data Transfer Object)模式：数据传输对象

3）服务定位模式(Service locator) ---> JNDI 方式查找

注意：Struts 中采用的是命令(comand)模式，是基于 MVC 模式的，还利用了委托(delegate)模式！

EJB 笔记于 2008 年 9 月 22 日全部完成！！！

Spring 学习笔记

==========

主讲：沈阅斌

第一章 Spring 概述

1、轻量级框架

1）框架：Hibernate、Struts、Spring

1. Struts：web 层的框架

2. Hibernate：持久层的框架

3. Spring：综合框架；

2）SSH：Struts + Spring + Hibernate 三个框架的集成

2、EJB

1）使用 EJB 的好处：

1. 分布式处理

2. 对象的生命周期都是由 EJB 容器来管理的

3. 可以通过声明的方式使用容器提供的服务(比如事务服务)

Spring 可以提供声明型事务，引用了 EJB 的事务特性。

2）EJB3.0 的思想：

1. IOC：依赖注入

2. AOP：拦截器、Filter

Spring 也具有这两大思想。

3、Sping 的主要内容：

1）IOC(Inversion of Control，控制反转)：提供依赖注入。

2）AOP(Aspect-Oriented Programming，面向方面编程)：解决非业务逻辑功能的实现。

3）事务和数据访问

4）SSH

4、Sping 提供了一个轻量级容器：IOC 容器

1）管理 javabean 对象的生命周期

2）维护对象跟对象之间的依赖关系 ===> 依赖注入

重量级容器与轻量级容器的比较：

1. 重量级容器通常可以独立运行；轻量级容器通常不能独立运行，必须依赖其他容器。

2. 重量级容器包含的服务大而全，所有应用的服务都必须购买；轻量级容器包含的服务可以由开发人员自己选择。

3. 重量级容器直接管理线程调度；轻量级容器不关注线程调度问题。

5、为什么使用 Sping?

1）提供依赖注入(通过依赖注入的方式组装对象)，使得开发的系统中的组件之间的耦合性降低(松耦合)；

2）不使用 J2EE 中的技术，也可以通过声明的方式来获得一些服务；

3）spring 是一个扩展性非常强的框架，很容易跟其他的框架进行集成，而且可以简化被集成的框架的使用。

6、Spring 的特点：

1）在 spring 框架中，把 checked 类型的异常全部转变为 unchecked 类型；

2）spring 框架包含了很多的模块，我们可以只使用 spring 提供的一个或者几个模块；

3）采用 spring 开发的应用扩展性非常的好，而且开发出来的类都是可重用的。

7、Spring模块介绍：

1）spring Core：提供了IOC容器，是一个核心模块，是所有其他模块的基础；

2）spring AOP：提供了面向方面编程的功能；

3）spring ORM：提供了集成持久层框架的能力，用于集成其他开源的框架；

4）spring Dao：对jdbc作了一个轻量级的封装，简化了jdbc的编程；

5）spring Web：提供了集成web层框架的能力，主要用于spring跟表现层框架的集成；

6）spring Context：是spring核心模块的一个扩展模块，提供了更强大的功能；

7）spring Web MVC：spring提供的表现层框架，是一个基于mvc的web框架的实现。

第二章 Spring IOC(控制反转)

1、IOC容器：管理bean对象的生命周期，提供依赖注入并将两个对象之间的依赖关系维护起来；依赖注入是IOC的一种表现形式。

2、好莱坞原则：

A(演员) <===== B(好莱坞)

```
public class A{
	private B b;
	/*
	public A(){
		b = new B();
	}
	*/
	public void setB(B b){
		this.b = b;
	}
}
```

3、依赖注入(Dependency Injection)：

1）接口注入： spring不支持

2）构造器注入： spring支持

如果在配置一个bean(bean类是标准的javabean)时，采用的注入方式是构造器注入，那么spring容器在创建bean时调用有参的构造器，并且把它所依赖的对象注入进去。

3）setter方法注入： spring支持

如果在配置一个bean时，采用的注入方式是setter方法注入，那么spring容器在创建bean时首先调用bean的无参构造器，然后调用bean的setter方法把它所依赖的对象注入进来。

4、Spring 中注入的数据主要有：

1）基本类型的数据：

<value>...<value/>

2）集合对象：

<list/> <set/> <map/> <props/>

3）其他的 Bean：

<ref bean="..."/>

5、获得容器创建的 bean 对象的两种方式：

1）方式一：

```
// 获取 xml 资源文件
Resource res = new ClassPathResource("com/briup/ioc/setter/setter.xml");
// 创建 IOC 容器：容器的接口 BeanFactory，配置文件是用 xml 语法写的。
BeanFactory factory = new XmlBeanFactory(res);
// 通过 IOC 容器获得对象：getBean 传入的是 bean 标签中 name 的值
ExampleBean exampleBean = (ExampleBean)factory.getBean("exampleBean");
```

2）方式二：

```
ApplicationContext ctx = new
ClassPathXmlApplicationContext("com/briup/ioc/setter/setter.xml");
ExampleBean exampleBean = (ExampleBean)ctx.getBean("exampleBean");
```

6、IoC 容器支持的两个不同的 bean 作用域：singleton 和 prototype

1）singleton(单例)：IOC 容器管理的 bean 默认情况下都是单例的，可以通过 bean 元素中的 singleton 属性来控制是否为单例。

2）prototype(原型)：Prototype 作用域的 bean 会导致在每次对该 bean 请求（将其注入到另一个 bean 中，或者以程序的方式调用容器的 getBean()方法）时都会创建一个新的 bean 实例。

配置：在<bean>中设置 singleton="false"。

注意：根据经验，对有状态的 bean 应该使用 prototype 作用域，而对无状态的 bean 则应该使用 singleton 作用域。

3）XmlBeanFactory 默认是预先装载，在 xml 找到的 bean 对象都会全部装载进内存。

注意：1.属性：pre-instantiates 和 lazy-init；

2.设置 lazy-init="true"，则表示 bean 对象在调用 getBean 时才被创建；

3.lazy-init 对 BeanFactory 是不起作用的；

4.XmlBeanFactory 不会预先实例化单例的 bean。

7、在 Spring 中创建非 javabean 类型的对象有三种方式：

1）静态工厂方法

1. 创建一个工厂类，提供一个静态的工厂方法；

2. 在配置文件中配置一个产品的 bean(产品对象)，class 为工厂的类，另外还要指定工厂方法的名字；

如下:

```
<bean id="conn" class="XXX" factory-method="getConnection"></bean>
```

其中 getConnection 是 XXX 中的静态方法,返回值是一个需要的对象。

2）实例工厂方法

1. 创建一个工厂类，提供一个非静态的工厂方法；

2. 配置一个工厂的 bean(工厂对象)；再配置一个产品的 bean，并指定工厂 bean 的名字，以及工厂方法的名字；

如下:

```
<bean id="factory" class="XXX"></bean>
<bean id="conn" factory-bean="factory" factory-method="getConnection"></bean>
```

其中 getConnection 是 XXX 类中的非静态方法,返回值为需要的对象，conn 为 JNDI 的名字。

3）提供工厂类，实现 FactoryBean 接口(简化我们的配置)

配置一个产品的 bean，指定工厂的类。

如下：

```
<bean id="conn" class="com.briup.ConnectionFactory"> ... </bean>
```

该类必须实现 FactoryBean 接口和接口中的方法，返回的是 connection 对象.

通过 factory.getBean("conn")获得的是 connection 对象；通过 factory.getBean("&conn")获得的是该对象本身，这里指 ConnectionFactory。

8、自动装配(autowired)：

1）byName：

根据当前 bean 所依赖的其他 bean 的 property 的名字，查找 ioc 容器中是否存在名字相同的 bean，如果存在则通过 setter 方法注入进来。

2）byType：

根据当前 bean 所依赖的其他 bean 的类型，查找 ioc 容器中是否存在类型相同的 bean，如果存在则通过 setter 方法注入进来。

3）constructor：

根据当前 bean 的构造器参数类型，查找 ioc 容器中是否存在类型相同的 bean，如果能查找到所有构造器参数，那么通过构造器注入进来。

4）autodetec：

先进行构造器装配，而后进行类型装配。

9、依赖性检查(dependency-check)：

检查 bean 的依赖关系是否注入完毕，若没有注入完毕则报异常。

1）simple：只检测基本类型的值是否注入，而不关心对象是否注入；

2）objects：只检测对象是否注入，而不检测基本类型的值是否注入；

3）all：检测全部。

作业：

1）提供一个 spring 的配置文件，如 user.xml。将 UserAction，UserServiceImpl，UserDaoImpl 配置成 spring 的 bean，并且维护它们之间存在的依赖关系。

2）写一个测试类，通过 spring 的 ioc 容器获得 UserAction 对象，调用它的 register 方法，返回 success，则测试成功。

10、Spring 提供的一些特殊接口：

1）BeanFactoryAware：

如果一个 bean 内部需要获得 BeanFactory 对象，那么可以让 bean 类实现这个接口，BeanFactory 会自动传入进来。

2）BeanNameAware：

如果一个 bean 内部需要获得自己的名字，那么可以让 bean 类实现这个接口，bean 的名字会自动传入进来。

3）FactoryBean 接口：实现三个方法，简化配置。

```
// 工厂方法，返回产品对象
public Object getObject() throws Exception {
        ...
        return conn;
}
// 返回产品类型
public Class getObjectType() {
        return Connection.class;
}
// 返回产品是否是单例
public boolean isSingleton() {
        return true;
}
```

main()：

```
// 返回工厂对象
ConnectionFactory cf = (ConnectionFactory) factory.getBean("&conn");
// 返回产品对象
Connection conn = (Connection) factory.getBean("conn");
```

4）BeanPostProcessor：

Spring 提供的一个接口，开发人员可以提供一个类实现这个接口，只要把这个实现类配置成 spring 的一个 bean，那么这个 bean 就会对其他的 bean 作处理。

5）BeanFactoryPostProcessor：

spring 提供的一个接口，开发人员可以提供一个实现类实现这个接口，只要把这个实现类配置成 spring 的一个 bean，那么这个 bean 就会对 BeanFactory 作处理。

第三章 Spring AOP(面向切面编程)

1、关注点

1）核心关注点(OOP)：关注系统的业务逻辑

2）横切关注点(AOP)：关注系统级服务，比如事务、安全、日志

2、方面(Aspect)：将散落在各处的代码规整到一处形成一个模块，这个模块称为方面。

3、连接点(Joinpoint)：程序执行过程中的某一点，比如方法的调用、异常的抛出。

4、增强(Advice)：为某个目标对象增加额外的功能，在 spring 中增强是通过拦截器实现的。

1）前增强拦截器：在目标对象方法执行之前执行的拦截器。

实现的接口：MethodBeforeAdvice

```
<!-- 目标对象 -->
<bean name="target" class="com.briup.aop.OrderServiceImpl"></bean>
<!-- 拦截器对象 -->
<bean name="advice" class="com.briup.aop.before.LoggerBeforeAdvice"></bean>
<!-- 创建目标对象的代理对象，注意在调用时调用的是代理对象，而不是目标对象 -->
<bean name="proxy" class="org.springframework.aop.framework.ProxyFactoryBean">
    <!-- 注入目标对象 -->
    <property name="target">
        <ref bean="target"/>
    </property>
    <!-- 注入拦截器的名字，而不是拦截器对象 -->
    <property name="interceptorNames">
        <list>
            <value>advice</value>
```

```
                </list>
            </property>
            <!-- 注入代理对象需要实现的接口名称 -->
            <property name="proxyInterfaces">
                <list>
                    <value>com.briup.aop.IOrderService</value>
                </list>
            </property>
        </bean>
```

2）后增强拦截器：在目标对象方法执行之后执行的拦截器。

实现的接口：AfterReturningAdvice

3）环绕增强拦截器：在目标对象方法执行之前和之后都执行的拦截器。

实现的接口：MethodInterceptor

4）抛出增强拦截器：在目标对象方法执行时抛出异常才执行的拦截器。

实现的接口：ThrowsAdvice

5、切入点(Pointcut)：一组连接点，如果有选择性的拦截目标对象中的方法的话需要定义切入点。

为拦截器(advice)定义切入点从而产生增强器(advidor)：

org.springframework.aop.support.RegexpMethodPointcutAdvisor

```
        <!-- 拦截器：拦截目标对象的所有方法 -->
        <bean name="advice" class="com.briup.aop.before.LoggerBeforeAdvice"></bean>
        <!-- 增强器：拦截目标对象的部分方法 -->
        <bean name="advisor"
class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">
            <property name="advice">
                <ref bean="advice"/>
            </property>
            <property name="patterns">
                <list>
                    <!-- .：通配一个字符(任意字符)；*：通配前一个字符若干次 -->
                    <value>.*save</value>
                    <value>.*find</value>
                </list>
            </property>
        </bean>
```

6、目标对象(Target Object)：需要被增强功能的对象称之为目标对象，也被称为被增强或被代理对象。

7、代理对象(Proxy Object)：为目标对象增加额外功能之后产成的新的对象，主要负责调用拦截器和目标对象。

1）产生代理对象的两种机制：

1.java 中的动态代理：产生代理对象需要和目标对象实现共同接口；

2.第三方的 cglib 库：产生的代理对象是目标对象的子类对象。

2）Spring 提供的专门用来产生代理对象的类：

org.springframework.aop.framework.ProxyFactoryBean

3）自动代理：不需要写配置信息就能产生代理对象

1. org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator

特点： a. 只能注入增强器，对拦截器不起作用

b. 不灵活：代理配置文件中所有的目标对象

2. org.springframework.aop.framework.autoproxy.BeanNameAutoProxyCreator

特点： a. 对拦截器、增强器都起作用

b. 灵活：可选择性的代理配置文件中的目标对象

8、织入(Weaving)：为目标对象增加额外功能从而产生代理对象的过程。

作业：1.给 UserDao 增加写日志的功能(抛出增强拦截器) —LoggerInterceptor

2.给 UserService 增加事务处理的功能(环绕增强拦截器实现) —TransactionInterceptor

第四章 Spring 中的数据访问

1、使用 JDBC

配置 DataSource：

1）自己配置一个数据源：

工厂类：org.apache.commons.dbcp.BasicDataSource

产品：DataSource 对象

2）查找目录服务器上的数据源：

工厂类：org.springframework.jndi.JndiObjectFactoryBean

产品：目录服务器上查找出来的数据源

2、使用 hibernate

配置 SessionFactory

1）创建 hibernate 的 SessionFactory 对象：

工厂类：org.springframework.orm.hibernate3.LocalSessionFactoryBean

产品：SessionFactory 对象

2）目录服务器上的 SessionFactory 对象查找：

工厂类：org.springframework.jndi.JndiObjectFactoryBean

产品：目录服务器上查找出来的 SessionFactory 对象

3、事务

1）事务管理器：

1. 专门用来管理事务的一个对象，不同的持久化方式采用不同的事务管理器进行管理。

2. 是借助于 SessionFactory 管理事务的。

3. Hibernate：org.springframework.orm.hibernate3.HibernateTransactionManager

2）事务拦截器：

1. 给某一个目标对象增加事务管理的功能，是借助于事务管理器管理事务的。

2. org.springframework.transaction.interceptor.TransactionInterceptor(环绕增强拦截器)

4、Spring 跟 struts 的集成：

1）在 web.xml 中配置一个监听器，用来初始化 spring 上下文，它会创建 ApplicationContext 对象 ac，并且把 ac 放到 ServletContext 中；

Spring 上下文：指的是 Spring IOC 容器

监听器：ServletContextListener

真正类型：org.springframework.web.context.ContextLoaderListener

作用：创建 spring 的 ioc 容器，并且将其放到 application 范围(Servlet Context)内。

2）在 struts-config.xml 文件中配置一个 action，类型为 spring 框架提供的一个类型。主要工作是从 ServletContext 中取到 ApplicationContext 对象 ac，然后从 ac 中找到名字为 action path 的那个 bean，并且调用这个 bean 的 execute 方法；

3）在 spring 的配置文件中配置一个 bean，类型为真正的 action 的类型，bean 的名字必须跟 struts-config.xml 中配置的 action 的 path 一致。

补充：servlet 中访问 spring 容器中 bean 实例的方法

```
ICustomerService service = (IcustomerService)WebApplicationContextUtils
.getWebApplicationContext(request.getSession().getServletContext()).getBean("customerService");
```

Spring 笔记于 2008 年 9 月 25 日全部完成！！！

CVS学习笔记

==========

主讲：张海斌

第一天：cvs
两天半：plsql+junit
半天：团队分组+团队活动
18天：woss(宽带运营支撑系统)

CVS：
一、什么是CVS

CVS(Concurrent Version System)：版本并发访问控制系统，C/S架构的软件。

1.源代码的存储仓库
2.多人协同开发一个大型项目
3.备份代码

二、什么是版本

标记文件的编号，在某个时间点对文件进行标记并命名（特定对象的潜在的体现）。

三、CVS的作用(为什么使用cvs)

1.版本的并发控制
1）对同一个文件的不同版本进行控制
2）对多个人同时开发进行控制
2.源代码备份
1）对源文件进行备份
2）对文件的操作进行历史记录

四、cvs工作原理

CVS的基本工作思路是这样的：在一台服务器上建立一个仓库，仓库里可以存放许多不同项目的源程序．由仓库管理员统一管理这些源程序．这样，就好象只有一个人在修改文件一样．避免了冲突．每个用户在使用仓库之前，首先要把仓库里的项目文件下载到本地．用户做的任何修改首先都是在本地进行，然后用 cvs 命令进行提交，由 cvs 仓库管理员统一修改．这样就可以做到跟踪文件变化，冲突控制等等。

五、cvs 的历史

1.1986 年开始使用 shell 脚本编写
2.1989 年开始使用 C 语言编写
3.1993 年出现基于网络的操作（TCP/IP）

六、明白 CVSROOT

源代码的存放路径

七、cvs 的安装

八、cvs 的使用

1.telnet 到 192.168.1.200
cvs /cvs
2.根据自己的名字创建目录

mkdir cvs_xuxiang

mkdir manager_xuxiang
mkdir pro1_xuxiang
mkdir pro2_xuxiang

3.setenv CVSROOT /u2/cvs/cvs_xuxiang

4.echo $CVSROOT

5.cvs init
初始化一个 cvs

6.cd manager_xuxiang
touch a.java
touch b.java

cvs import -m "first cvs" first briup start

-m:表示要加上注释信息
"first cvs"：注释信息
first：工程的名字
briup：厂商的名字

7. cd pro1_xuxiang
cvs checkout first

cd pro2_xuxiang
cvs checkout first

九、cvs 的四种状态

cvs status a.java

1.Up-to-date:表示服务器端与本地文件是一致的，版本是一样的
2.Locally Modified：表示本地的文件比服务器端的新
3.Needs Patch：表示服务器端的文件要比本地的文件新
4.Needs Merge：表示服务器端与本地文件有冲突

十、状态之间的转化

1.Locally Modified-->commit-->Up-to-date
2.Needs Patch-->update-->Up-to-date
3.Needs Merge-->update,vi,commit-->Up-to-date

十一、增加一个文件

touch e.java
cvs add e.java
cvs commit e.java

十二、删除一个文件

rm e.java
cvs remove e.java
cvs commit e.java

十三、删除整个工程

cvs release -d first

CVS 的历史

1、1986 年，使用 shell 脚本编写
2、1989 年，使用 C 语言改写
3、1993 年，出现基于网络的使用（TCP/IP）

CVS 笔记于 2008 年 9 月 27 日全部完成！！！

PL/SQL 学习笔记

==========

主讲：张海斌

第一章 PL/SQL 概述

1、oracle 应用编程方法：

1）OCI：C 语言和数据库打交道的方法，更底层，较少用；
2）ODBC：微软在 1991 年提出的访问数据库的标准；
3）JDBC：java 访问数据库的技术；
4）SQLJ：一种很新的 java 访问数据库的技术，它允许在 java 程序中嵌入静态的 SQL 语句，会的人很少；
5）PL/SQL：运行在数据库内部的程序语言，为其他方法在数据库外访问数据库。

2、什么是 PL/SQL?为什么使用 PL/SQL?

PL/SQL: Procedural Language/SQL

1）是在标准 SQL 的基础上增加了过程化控制的语言；
2）是 Oracle 客户端访问 Oracle 服务器的操作语言；
3）oracle 对标准 SQL 的扩充。

3、PL/SQL 的优缺点:

1）PL/SQL 是结构化的编程语言，不是面向对象的；
2）良好的可维护性(保存在数据库内部,不管 Oracle 运行什么操作系统)；
3）良好的可移植性；
4）提升系统的性能。

缺点：不能移植到异构的数据库系统，只能运行在 Oracle 数据库上。

4、SQL 与 PL/SQL 的比较

1）SQL

1.第四代语言（

第一代语言：机器语言

第二代语言：汇编语言
第三代语言：高级语言
)

2.做什么，不管怎么做
3.缺少过程与控制语句
4.无算法描述能力

2）PL/SQL（procedural Language／SQL）

SQL 的扩展
1.增加了变量和类型
2.控制结构
3.过程与函数
4.对象类型与方法

第二章 PL/SQL 程序结构

1、结构组成:

```
declare

    //声明部分

begin

    //执行部分(一定有)

exception

    //异常部分

end;
```

注意：begin 和 end 部分必须有，其它两个部分可有可无。

2、pl/sql 的开发环境:

1）任何的纯文本的编辑器都可以，如:vi

设置临时环境变量：set serveroutput on;

```
declare
    v_name varchar2(20) :='Hello World';
begin
    dbms_output.put_line(v_name);
```

```
        dbms_output.put_line('Hello'||' World');
    end;
    /
```

注意：/表示执行!

2）Windows 平台下建议使用“PLSQL Developer”。

3、pl/sql 的字符集:

字母: A-Z，a-z

数字: 0-9

空白: Tab，space，回车

符号: +-*/<>=();:'@"%||&--/**/

注意：pl/sql 本身对字符的大小写是不敏感的

4、pl/sql 的标识符:

1）必须以字母开头

2）后面跟任何非空格字符,货币符号,数字,下划线以及#.

3）最大长度为 30 个字符(建议一般长度为 8 个字符)

5、注释:

1）单行注释: --

2）多行注释:

```
/*
    多行注释
*/

declare
    v_name varchar2(20) :='briup';
begin
    /*
        注释语句
        输出 briup
    */
    dbms_output.put_line(v_name);
    --dbms_output.put_line('briup1');
```

```
end;
/

create table xuxiang(
id number primary key,--主键
name varchar2(20)--姓名
);
```

6、变量的声明:

1）语法:

Var_name [CONSTANT] type [NOT NULL][:=value];

2）注意:

a.[constant][not null][:=value]可有可无

b.constant:表示常量,值不可改变

c.not null:表示非空,一定要有初始值

d.赋值使用":="

e.如果没有赋初始值,默认为 NULL

f.":="可以用 DEFAULT 代替

```
declare
        v_name constant varchar2(20) not null :='aaaa' ; -- ":="表示赋初值
begin
        dbms_output.put_line(v_name);
end;
/

declare
        v_name constant varchar2(20) :='aaaa';
begin
        v_name :='bbbb'; --错误：常量是不能改变初值的
        dbms_output.put_line(v_name);
end;
/


declare
        v1 varchar2(20) :='aaaa';
        v2 varchar2(20);
begin
        dbms_output.put_line(v1);
        dbms_output.put_line(v2); --null：nothing，什么也没有！
        dbms_output.put_line(v1);
end;
```

```
/

结果：      aaaa
            aaaa
==========================================

begin
        dbms_output.put_line(""""");
end;
/
```

结果：输出第三个和第五个单引号，第二个和第四个单引号为转义符。

注意：若为五个单引号的话则会报错。

第三章 PL/SQL 数据类型

1、pl/sql 数据类型

1）标量类型：数字型、字符型、布尔型、日期型

2）组合类型：record、table、varray

3）参考类型：ref cursor、ref object_type

4）Lob：大数据类型(BFLLE、BLOB、CLOB、NCLOB)

2、数字类型

1）NUMBER：可以存储数字值，包括整数或浮点数

2）NUMBER(p,s)：

p 是精度，s 是刻度范围。精度是数值中所有数字位的个数，而刻度范围是小数点右边数字位的个数。刻度范围可以是负数，这表示由小数点开始向左进行计算数字位的个数。

例如：
2008.08 ---> number(6,1) ---> 2008.10
2008.08 ---> number(6,-1) ---> 2010.00
2008.0354 ---> number(6,3) ---> 2008.035 ---> 错误，超出精度
0.00020080314 ---> number(7,5) ---> 0.000201
2008 ---> number(3,-1) ---> 2010

声明	赋值	实际值
NUMBER	2008.0314	2008.0314
NUMBER(4)	2000	2000
NUMBER(3)	2008	出错

NUMBER(6,3)	2008.0314	出错
NUMBER(6,2)	2008.0354	2008.04
NUMBER(7,5)	2008.031114	出错
NUMBER(4,6)	0.00020080314	0.000201

注意:

1.首先处理刻度，正数小数点向右移动，负数小数点向左移动，移动完毕做四舍五入；

2.判断精度，有效位数从左边第一位不是零的数字算起，到按照刻度移动完以后小数点所在位置；

3.如果小数点是左移，在四舍五入结束后要将小数点还原，没有的位数用零补上。

3）BINARY_INTEGER

仅存储整型，以二进制形式存储，适用于仅用来计算。通常适用于循环计数。

3、字符型

CHAR(20)：

存储可变长度的字符串，长度以字节进行指定。最大长度为 4000 字节。

CHAR：

定长字符串，最大长度是 32767 字节。

LONG：

变长字符串，最大长度是 32767 字节。

4、布尔型

BOOLEAN：

在 PL/SQL 控制结构中使用，仅可以存储 TRUE、FALSE 和 NULL。

5、日期型

DATA：

存储日期和时间信息，包括世纪、年、月、日、

6、%type 类型

```
create table tbl_xuxiang(
id number(30) primary key,
name varchar2(50)
);


begin
        for i in 1..100 loop
   insert into tbl_xuxiang values(i,'briup'||i);
        end loop;
end;
/
```

使用标量类型声明变量：

```
declare
        v_id number(30);
        v_name varchar2(50);
begin
        select id ,name into v_id,v_name from tbl_xuxiang where id=1;
        dbms_output.put_line('ID:'||v_id||'   NAME:'||v_name);
end;
/
```

%type:表示变量具有与数据库当中表的某一字段相同的类型

```
declare
        --v_id number(30);
        v_id tbl_xuxiang.id%type;
        v_name tbl_xuxiang.name%type;
begin
        select id ,name into v_id,v_name from tbl_xuxiang where id=10;
        dbms_output.put_line('ID: '||v_id||'  NAME: '||v_name);
end;
/
```

根据自己定义的变量的类型声明其他变量：

```
declare
        v_id number(30);
        v_id1 v_id%type;
        v_name tbl_xuxiang.name%type;
begin
        select id ,name into v_id1,v_name from tbl_xuxiang where id=10;
        dbms_output.put_line('ID:'||v_id1||'  NAME:'||v_name);
```

```
end;
/
```

7、Record类型

```
declare
        type t_rec is record(v_id number(30),
                        v_name varchar2(50));
        v_rec t_rec;
begin
        select * into v_rec from tbl_xuxiang where id=20;
        dbms_output.put_line('ID:'||v_rec.v_id||' NAME:'||v_rec.v_name);
end;
/


declare
        type t_rec is record (v_id tbl_xuxiang.id%type,
        v_name tbl_xuxiang.name%type);
        v_rec t_rec;
begin
        select id ,name into v_rec from tbl_xuxiang where id=30;
        dbms_output.put_line('ID:'||v_rec.v_id||'NAME:'||v_rec.v_name);
end;
/

declare
        m_id number(30);
        m_name varchar2(50);
        type t_rec is record (v_id m_id%type,
                        v_name m_name%type);
        v_rec t_rec;
begin
        select * into v_rec from tbl_xuxiang where id=2;
        dbms_output.put_line('ID:'||v_rec.v_id||'NMAE:'||v_rec.v_name);
end;
/


declare
        type t_rec1 is record(
                v_id number(30),
                v_name varchar2(50));
        type t_rec2 is record(
                v_id number(30),
                v_name varchar2(50));
        v_rec1 t_rec1;
        v_rec11 t_rec1;
        v_rec2 t_rec2;
```

```
begin
        --v_rec1 :=v_rec2;  --error，类型不相同是不能相互之间赋值的
        select * into v_rec11 from tbl_xuxiang where id=20;
        v_rec1:=v_rec11;
        v_rec2.v_id :=v_rec1.v_id;
        dbms_output.put_line('ID:'||v_rec2.v_id||'   NAME:'||v_rec1.v_name);
end;
/
```

8、%rowtype 类型

%rowtype:表示与数据库中表的类型相同

```
declare
        v_test tbl_xuxiang%rowtype;
begin
        select id ,name into v_test from tbl_xuxiang where id=5;
        dbms_output.put_line('ID:'||v_test.id||'   NAME:'||v_test.name);
end;
/
```

9、table 类型

1）语法:

type type_name is table of type index by binary_integer --表示下标用整型表示

1.与 java 中的数组类似

2.可以是标量类型，也可以是复合类型。

2）使用标量类型定义 table

```
declare
        type t_tab is table of varchar2(50) index by binary_integer;
        v_tab t_tab;
begin
        select name into v_tab(10) from tbl_xuxiang where id=51;
        dbms_output.put_line('NAME:'||v_tab(10));
end;
/
```

3）使用数据库中表的某一字段的类型来定义 table

```
declare
        type t_tab is table of tbl_xuxiang.id%type index by binary_integer;
        v_tab t_tab;
begin
        select id into v_tab(1) from tbl_xuxiang where id=1;
        dbms_output.put_line('ID:'||v_tab(1));
```

```
end;
/
```

4）使用数据库中表的类型来定义 table

```
declare
        type t_tab is table of tbl_xuxiang%rowtype index by binary_integer;
        v_tab t_tab;
begin
        select id ,name into v_tab(100) from tbl_xuxiang where id=40;
        dbms_output.put_line('ID:'||v_tab(100）id||'NAME:'||v_tab(100）name);
end;
/
```

5）通过自己定义的变量的类型定义 table

```
declare
        v_var number(30);
        type t_tab is table of v_var%type index by binary_integer;
        v_tab t_tab;
begin
        select id into v_tab(10) from tbl_xuxiang where id=5;
        dbms_output.put_line('ID:'||v_tab(10));
end;
/
```

10、变量的作用域:

1.变量的作用域是可以访问该变量的程序部分；

2.pl/sql 的变量的作用域是从变量的声明开始到该语句块的结束.；

3.内部的块中可以访问外部块中的变量；反之，外部块中不可以访问内部块的变量。

```
declare

begin   --外部语句块

   v_name varchar2(20) :='briup';

        declare

        begin   --内部语句块
                v_name varchar2(20) :='briup1';

                dbms_output.put_line(v_name);

        end;

end;
```

```
/



<<1_Outer>>
DECLARE
        v_AvailableFlag       BOOLEAN;
        v_SSN           NUMBER(9);
BEGIN
        <<2_Outer>>
        DECLARE
                v_SSN           CHAR(11);
                v_StartDate     DATE;
        BEGIN
           1_Outer.v_SSN
        END;
        ③
END;
```

第四章 PL/SQL 中的控制语句

1、条件语句(分支语句)

```
declare
        v_num1 number(10);

begin
        select count(*) into v_num1 from tbl_xuxiang;
        if v_num1 <=0 then
                dbms_output.put_line('no result');
        elsif
                v_num1>=1 and v_num1<=50 then
                dbms_output.put_line('RESULT:'||v_num1);
        else
                dbms_output.put_line('more than 50');
        end if;

end;
/
```

2、循环语句

1）loop 循环

```
        declare
                i integer :=0;
                j integer :=40;
        begin
                select count(*) into i from tbl_xuxiang;
```

```
        loop
          if j>i then --判断条件，退出 loop 循环
            exit;
          end if;
          dbms_output.put_line('j:'||j);
          j :=j+1;
        end loop;
end;
/
```

2）while 循环

```
declare
        v_num number(10) :=0;
begin
        while v_num<=20 loop
          dbms_output.put_line('v_num:'||v_num);
          v_num :=v_num+1;
        end loop;
end;
/
```

3）for 循环

```
declare
        v_num number(10) :=0;
begin
        for v_num in 1..10 loop         --闭区间
        if   mod(v_num,2)=0 then

        dbms_output.put_line('v_num:'||v_num);
        end if;
        end loop;
end;
/
```

注意：语句 for v_num in 10..1 不错，但是不行。

4）for 循环语句的逆序输出

```
declare
        v_num number(10) :=0;
begin
        for v_num in reverse 1..10 loop
        dbms_output.put_line(v_num);
        end loop;
end;
```

```
/
```

3、GOTO 语句

```
declare
        v_num number(10) :=0;
begin
        for v_num in 1..10 loop
        dbms_output.put_line('v_num:'||v_num);
        if v_num>=5 then
                goto lb_1;
        end if;
        end loop;
        <<lb_1>>
        dbms_output.put_line('loop end!!');
end;
/
```

第五章 PL/SQL 游标

1、游标(cursor)

1）作用：提取多行的数据集

2）使用步骤：

1.声明游标：只能有一条查询语句,声明游标时使用到的变量一定要在声明游标之前声明变量

2.打开游标：相当于我们执行了一条 select 语句

3.提取数据：存放数据的变量的数量，类型，顺序一定要与 Table 中的字段是一致的.

4.关闭游标：

(1)游标使用完毕要关闭.

(2)游标关闭以后不能再次提取数据或是关闭游标.

(3)相当于将内存中游标的数据清空

注意：a、游标只能打开或者关闭一次

b、游标在关闭之后不能再次 fetch 数据

```
declare
        cursor v_cur is select id,name from tbl_xuxiang ;
        v_id number(30);
        v_name varchar2(50);
begin
        --打开游标
        open v_cur;
        --提取数据
```

```
        dbms_output.put_line(v_cur%rowcount); --0
        fetch v_cur into v_id,v_name;
        dbms_output.put_line(v_cur%rowcount); --1
        --关闭游标
        close v_cur;
end;
/
```

2.游标的属性:

1）%found：查找到数据就返回 true,否则返回 false,在没有打开游标的时候引用返回错误

2）%notfound：作用与%found 恰好相反

3）%isopen：游标打开返回 true,否则返回 false

4）%rowcount：fetch 完数据返回相应的记录行数.没有 fetch 的时候返回 0.

使用方式：cursor_name 加上属性

如：if v_cur%found then ...

3、练习：

结合前面所学习的循环语句和游标的属性，将整张表中的所有记录查询出来，输出到控制台(至少使用两种不同的方式)。

1）loop 循环：

```
declare
        cursor v_cur is select id, name from tbl_xuxiang;
        v_row tbl_xuxiang%rowtype;
begin
        open v_cur;
        loop
          fetch v_cur into v_row;
          exit when v_cur%notfound;
                dbms_output.put_line('id:'||v_row.id||'   name:'||v_row.name);
        end loop;
        close v_cur;
end;
/
```

2）while 循环：

```
declare
        cursor v_cur is select id ,name from tbl_xuxiang;
        v_row tbl_xuxiang%rowtype;
```

```
begin
        open v_cur;
        fetch v_cur into v_row;
        while v_cur%found loop
        dbms_output.put_line('id:'||v_row.id||'   name:'||v_row.name);

        fetch v_cur into v_row;

        end loop;
        close v_cur;
end;
/
```

3）for 循环游标(*)

```
begin

        for v_cur in (select id ,name from tbl_xuxiang) loop
        dbms_output.put_line('ID:'||v_cur.id||'NAME:'||v_cur.name);
        end loop;

end;
/
```

4、游标 带参数传递

```
declare
        cursor v_cur(v_name varchar2) is select id,name from tbl_xuxiang where name=v_name;
        v_row tbl_xuxiang%rowtype;
begin
        open v_cur('briup10');
        fetch v_cur into v_row;
        dbms_output.put_line('ID:'||v_row.id||'NAME:'||v_row.name);
        close v_cur;
end;
/
```

注意:如果声明带参数的游标的时候,参数只能有类型,但是不能加上长度限制.如:v_name varchar2.

5、练习：创建两张结构相同的表

```
        create table xuxiang_test1(
        id      number          primary key,
        name    varchar2(30)
        );

        create table xuxiang_test2(
        id      number          primary key,
```

```
name   varchar2(30)
);
```

首先往 test1 表中插入 50 条记录，然后利用游标将 test1 中的前 30 条记录查询出来插入到 test2 表中

（在插入之前先将 test2 中的数据清空），并且将结果输出打印在控制台上（只写一个匿名语句块）。

```
begin
        dbms_output.put_line('***clear xuxiang_test1***');
        delete from xuxiang_test1;
        dbms_output.put_line('***insert 50 xuxiang_test1***');
        for i in 1..50 loop
        insert into xuxiang_test1 values(i,'briup'||i);
        end loop;
        dbms_output.put_line('***clear xuxiang_test2***');
        delete from xuxiang_test2;
        dbms_output.put_line('***insert 30 xuxiang_test2***');
        for v_cur in (select id ,name from xuxiang_test1 where rownum<31) loop
        insert into xuxiang_test2 values(v_cur.id,v_cur.name);
        dbms_output.put_line('ID: '||v_cur.id||'        Name: '||v_cur.name);
        end loop;
        commit;
        dbms_output.put_line('***success***');
end;
/
```

第六章 PL/SQL 异常

1、异常处理：

```
declare
        //声明异常
begin
        //触发异常
exception
        //处理异常
end;
```

显示错误信息：show error

2、程序范例：

```
declare
        e1 exception;
        e2 exception;
        e3 exception;
```

```
        e4 exception;
begin
        raise e4; --抛出异常 e4
exception
    when e1 then
        dbms_output.put_line('Exception e1');
    when e2 then
        dbms_output.put_line('Exception e2');
    when e3 then
        dbms_output.put_line('Exception e3');
    when others then
        dbms_output.put_line('other Exception');
end;
/


declare
        v_num number(2);
begin
        v_num :=5/0;
exception
        when others then
        dbms_output.put_line('Exception: the divided number is zero!');
end;
/
```

第七章 PL/SQL 存储过程

1、匿名语句块

1）没有名字

2）只执行一次,不会保存在数据库中

3）不能相互之间进行调用

2、命名语句块

1）有特定的名字

2）保存在数据库中,可以重复执行

3）可以在匿名语句块中进行调用

常见的命名块：procedure,function,package,trigger

3、procedure:

1）语法：

```
CREATE [OR REPLACE] PROCEDURE proc_name
[ (arg_name [ { IN | OUT | IN OUT } ] TYPE,
```

```
…
arg_name [ { IN | OUT | IN OUT } ] TYPE) ]
{ IS | AS }
procedure_body
```

1.or replace:如果存在同名的语句块,就会替换掉原来已经存在的语句块

2.PROCEDURE:命名块的类型

3.proc_name:命名块的名字

4.arg_name:参数的名字(可有可无)

```
create or replace procedure pro_xuxiang
is
        v_name varchar2(30):='briup';
begin
        dbms_output.put_line(v_name);
end;
/

begin
        pro_xuxiang;
end;
/

exec pro_xuxiang;
```

2）procedure 参数类型(模式):

1.in:这类型的参数只可以读取,不能写入,只能出现在等号的右边.(默认的参数类型)

2.out:这类型的参数只可以写入,不能读取,只能出现在等号的左边.任何实参都会被忽略,只能传形参.

3.in out :这类型的参数是前两种情况的结合体,既可以写入,也可以读取.

注意：模式为 IN OUT 或 OUT 参数对应的实际参数必须是变量，不能是常量或表达式。

```
create or replace procedure pro_xuxiang(v_name in varchar2)
is
        v_name1 varchar2(30) :='aaaa';
begin
        v_name1 :=v_name;
        dbms_output.put_line(v_name1);
end;
/

begin
        pro_xuxiang('bbbb');
end;
/


create or replace procedure pro_xuxiang (v_name out varchar2)
is
        v_name1 varchar2(30) :='aaaa';
begin
        v_name :=v_name1;
        dbms_output.put_line(v_name);
end;
/

declare
        v_name varchar2(30):='bbbb';
begin
        pro_xuxiang(v_name);
end;
/


create or replace procedure pro_xuxiang (v_name in out varchar2)
is
        v_name1 varchar2(30) :='aaaa';
begin
        v_name :=v_name1;
        v_name1 :=v_name;
        dbms_output.put_line(v_name);
end;
/
```

3）参数的传递

指定实参的模式：提高程序的可读性.

```
create or replace procedure pro_xuxiang(v_name1 in varchar2,v_name2 in varchar2)
is
```

```
        v_name11 varchar2(30) :='aaaa';
begin
        v_name11 :=v_name1;
        dbms_output.put_line(v_name11);
        dbms_output.put_line('**************');
        v_name11 :=v_name2;
        dbms_output.put_line(v_name11);
end;
/

begin
pro_xuxiang(v_name2=>'cccc',v_name1=>'bbbb');
end;
/

begin
pro_xuxiang('bbbb',v_name2=>'cccc');
end;
/
```

注意:

1. 调用存储过程的时候，传递参数可以按照位置标识，也可以按照名字标识
2. 但是两种方式混用时，第一个参数必须通过位置来指定

4、function

1）function 与 procedure 的比较：

function 与 procedure 的相同点

1.都是有名字的语句块

2.都有统一的形式:声明,执行,异常处理

3.都存储在数据库中,都可以被匿名语句块来调用

function 与 procedure 的区别:

1.procedure 本身就是一个 pl/sql 语句,而 function 只是 pl/sql 表达式的一部分.

2.procedure 没有返回值,而 function 有返回值

2）function 的声明:

```
CREATE [OR REPLACE] FUNCTION func_name
[ (arg_name [ { IN | OUT | IN OUT } ] TYPE,
…
(arg_name [ { IN | OUT | IN OUT } ] TYPE )]
RETURN TYPE
{ IS | AS }
Func_body
```

RETURN TYPE：返回值的类型,不能有长度的限制

注意:

1.在 function 的体部有一个 return 语句，来返回一个数值给调用者；

2.在 function 的体部可以有多个 return 语句，但单只能执行一个。

```
create or replace function fun_xuxiang
return varchar2
is
        v_name varchar2(30) :='ibm';
begin
        dbms_output.put_line(v_name);
        return v_name;
        dbms_output.put_line('********');
        return 'hp';
end;
/

declare
        v_name varchar2(30) :='acer';
begin
        v_name :=fun_xuxiang;
        dbms_output.put_line('NAME:'||v_name);
end;
/
```

执行结果： ibm

NAME:ibm

```
create or replace function fun_xuxiang(v_name in varchar2)
return varchar2
is
        v_name1 varchar2(30) :='ibm';
begin
        v_name1 :=v_name;
        dbms_output.put_line(v_name1);
        return v_name1;
end;
/

declare
        v_name varchar2(30) :='acer';
begin
        v_name :=fun_xuxiang('hp');
        dbms_output.put_line('NAME:'||v_name);
end;
/
```

```
create or replace function fun_xuxiang(v_name out varchar2)
return varchar2
is
        v_name1 varchar2(30) :='ibm';
begin
        v_name :=v_name1;
        dbms_output.put_line(v_name);
        return v_name;
end;
/

declare
        v_name varchar2(30) :='acer';
        v_name2 varchar2(30) :='hp';
begin
        v_name :=fun_xuxiang(v_name2);
        dbms_output.put_line('NAME:'||v_name);
end;
/


create or replace function fun_xuxiang(v_name in out varchar2)
return varchar2
is
        v_name1 varchar2(30) :='ibm';
begin
        v_name :=v_name1;
        v_name1 :=v_name;
        dbms_output.put_line('v_name:'||v_name);
        dbms_output.put_line('v_name1:'||v_name1);
        return v_name;
end;
/


declare
        v_name varchar2(30) :='acer';
        v_name2 varchar2(30) :='hp';
begin
        v_name :=fun_xuxiang(v_name2);
        dbms_output.put_line('NAME:'||v_name);
end;
/
```

5、删除过程与函数

```
DROP PROCEDURE proc_name;
DROP FUNCTION func_name;
```

第八章 PL/SQL 包

1、包:

1）将一组相关的对象存放在一起的 pl/sql 结构

2）包是保存在数据库中,不是本地.

3）包是带有名字的声明.

4）包中可以包含有所有出现在块的声明部分的内容

5）包中可以有变量,游标,类型,function,procedure

6）包分为包的头部和体部,只有在包的头部含有 function,procedure 的时候,才一定要有包体,否则可以没有包体.

7）先创建包头,在创建包的体部,且头部编译成功,包体才能编译成功.

2、包头:包含了所有包的信息,但不含有过程代码.

3、体部:包含了所有在头部声明的 function,procedure 的过程代码.

```
--创建包头：
create or replace package pac_xuxiang
is
        v_name varchar2(30) :='briup';
        procedure proc_xuxiang1;
end pac_xuxiang;
/

--创建包体：名字必须与包头的名字相同
create or replace package body pac_xuxiang
is
        procedure proc_xuxiang1
        is
        begin
                dbms_output.put_line('procedure1');
        end;

end;
/

--包的使用：
begin
        dbms_output.put_line(pac_xuxiang.v_name);
        pac_xuxiang.proc_xuxiang1;
end;
```

```
/
```

4、包中过程或函数的重载：

包中过程或者是函数的重载：名字相同，参数不同

参数的类型不同指的是参数的类型族，如 char 与 varchar 属于统一类型族。

5、练习：写一个包，包含有两个 procedure(pro_insert,pro_move)，分别实现插入 test1 表 20 条记录，然后查询出来插入到 test2 中。

```
create or replace package pac_xuxiang
is
        procedure pro_insert;
        procedure pro_move;
end pac_xuxiang;
/

create or replace package body pac_xuxiang
is
        procedure pro_insert
        is
        begin
                delete from xuxiang_test1;
                for i in 1..20 loop
                insert into xuxiang_test1 values(xuxiang_seq_common.nextval,'briup'||i);
                end loop;
        end;

        procedure pro_move
        is
        begin
                delete from xuxiang_test2;
                for v_cur in (select id,name from xuxiang_test1) loop
                insert into xuxiang_test2 values(v_cur.id,v_cur.name);
                dbms_output.put_line('ID: '||v_cur.id||'        Name: '||v_cur.name);
                end loop;
        end;
end;
/

begin
        dbms_output.put_line('***insert xuxiang_test1***');
        pac_xuxiang.pro_insert;
        dbms_output.put_line('***move xuxiang_test2***');
        pac_xuxiang.pro_move;
        dbms_output.put_line('***success***');
end;
/
```

第九章 PL/SQL 触发器

1、trigger 与 function,procedure 的比较

1）trigger 与 function,procedure 的相同点：

1.都有名字,都是命名的 pl/sql 语句块

2.都有声明部分,执行部分,异常处理部分

2）trigger 与 function,procedure 的不同点：

trigger 会自动执行,不用匿名语句块去调用

2、trigger 的触发事件：在数据库的表中执行 DML 语句.

1）什么是 DML 语句：只改变某一张表中的数据结果，而不改变表结构的语句。
SELECT, INSERT, UPDATE, DELETE

2）什么是 DDL 语句：改变表结构的语句
DROP, CREATE, ALTER, GRANT, REVOKE

3、TRIGGER 的作用

1）用于维护数据的完整性；

2）有些复杂的数据完整性约束无法在创建表的时刻通过声明性约束解决；

3）通过数据库表的记录的修改来执行审计功能；

4）当表被修改的时候，自动给其他需要执行操作的程序发信号。

4、trigger 的种类:

1）触发事件：3 种(insert, delete, update)

2）触发的时间：2 种(before, after)

3）触发的级别：2 种(行级别，语句级别)

注意：一个表最多可以定义 12 种触发器

```
create table test_xuxiang(
id      number          primary key,
name    varchar2(30)
);

create or replace trigger tri_xuxiang before
insert on test_xuxiang for each row
declare
        v_num number(10) :=0;
```

```
begin
        v_num :=fun_xuxiang +1;
        dbms_output.put_line('NUM:'||v_num);
        dbms_output.put_line('ID:'||:new.id); -- ":new.id"表示新插入纪录的 id
end;
/

create or replace function fun_xuxiang
return number
is
        v_num number(10) :=0;
begin
        select count(*) into v_num from test_xuxiang;
        return v_num;
end;
/

begin
        for i in 1..5 loop
        insert into test_xuxiang(id,name)values(i,'aaaa');
        end loop;
        commit;
end;
/
```

5、注意:

1）正在被 DML 语句操作的表是不能被触发器访问的

2）For each row 的意义是：在一次操作表的语句中，每操作成功一行就会触发一次；不写的话，表示是表级触发器，则无论操作多少行，都只触发一次；

3）:NEW 和:OLD 使用方法和意义，new 只出现在 insert 和 update 时，old 只出现在 update 和 delete 时。在 insert 时 new 表示新插入的行数据，update 时 new 表示要替换的新数据、old 表示要被更改的原来的数据行，delete 时 old 表示要被删除的数据。

4）在触发器中不能使用 commit。

第十章 动态 PL/SQL

1、使用动态 SQL 包来创建表:

```
declare
        v_cur binary_integer; --注意这里不是游标
        v_sql varchar2(200);
begin
        --为执行动态 sql 语句打开一个游标
        v_cur:=dbms_sql.open_cursor; --打开系统游标，返回 int 型的值
        v_sql:='create table mytest_xuxiang'||i||'(id number(5) primary key,name
```

```
varchar2(10))';
            --解析动态 sql 语句：
            --参数 1：打开游标
            --参数 2：执行的 sql 语句
            --参数 3：oracle 数据库类型版本，native 表示使用本地数据库的版本
            dbms_sql.parse(v_cur,v_sql,dbms_sql.native);
            --关闭游标
            dbms_sql.close_cursor(v_cur);
    end;
    /
```

注意：在匿名语句块中调用 procedure,如果此过程中使用了 DDL 语句,那么该用户必须直接赋予权限.

1.用超级用户登录，再更改到 oracle DBA 用户的命令：

```
su - ./as sysdba
```

2.oracle 用户的权限：

```
create user briup identified by briup.
grant connect,resource to briup;
```

3.帐户被锁住时的解锁命令：

```
alter user briup account unlock;
```

练习：使用动态 SQL 包来创建 5 张结构相同的表。

```
declare
    v_cur binary_integer;
    v_sql varchar2(200);
begin
    v_cur:=dbms_sql.open_cursor;
    for i in 1..5 loop
    v_sql:='create table xuxiang_mytest'||i||'(id number(5) primary key,name
varchar2(10))';
    dbms_sql.parse(v_cur,v_sql,dbms_sql.native);
    end loop;
    dbms_sql.close_cursor(v_cur);
end;
/
```

2、使用动态 SQL 包来查询数据库的记录

```
create or replace procedure pro_select_xuxiang(v_table varchar2,v_num number)
is
    v_cur binary_integer;
```

```
        v_result binary_integer;
        v_id    number(5);
        v_name          varchar2(10);
        v_sql   varchar2(200);
begin
        --打开游标
        v_cur:=dbms_sql.open_cursor;
        v_sql:='select id,name from '||v_table||' where rownum<'||v_num;

        --解析动态 sql
        dbms_sql.parse(v_cur,v_sql,dbms_sql.native);

        --指定取出数据的类型和约束
        --指定取出数据的类型，必须用变量表示（注意：字符串类型要指定约束）
        dbms_sql.define_column(v_cur,1,v_id);
        dbms_sql.define_column(v_cur,2,v_name,10);

        --执行 sql
        v_result:=dbms_sql.execute(v_cur);

        --获取数据
        while dbms_sql.fetch_rows(v_cur)>0 loop
                --定义变量存放取出的数据

                dbms_sql.column_value(v_cur,1,v_id);
                dbms_sql.column_value(v_cur,2,v_name);
                dbms_output.put_line('id:'||v_id||' name:'||v_name);
        end loop;
        --关闭游标
        dbms_sql.close_cursor(v_cur);
end;
/

begin
        pro_select_xuxiang('tbl_xuxiang',5);
end;
/
```

3、使用动态 SQL 包插入数据

```
        create or replace procedure pro_insert_xuxiang(v_talbe varchar2,v_id number,v_name
varchar2)
        is
                v_cur   binary_integer;
                v_result        binary_integer;
```

```
        v_sql   varchar2(200);
    begin
        --打开游标
        v_cur:=dbms_sql.open_cursor;
        v_sql:='insert into '||v_talbe||' values(:id,:name)';
        --解析动态 sql 语句
        dbms_sql.parse(v_cur,v_sql,dbms_sql.native);
        --绑定变量
        dbms_sql.bind_variable(v_cur,':id',v_id);
        dbms_sql.bind_variable(v_cur,':name',v_name);
        --执行动态 sql
        v_result:=dbms_sql.execute(v_cur);
        --关闭游标
        dbms_sql.close_cursor(v_cur);
        commit;
    end;
    /

    begin
        pro_insert_xuxiang('tbl_xuxiang',589,'seaside');
    end;
    /
```

4、定时调用

1）定时调用，一共四个参数：1、脚本号，2、操作(varchar)3、第一次执行时间，4、每隔 x 时间执行一次

```
create table xuxiang_job(s_date date);

declare
    code number:=121;
begin
        DBMS_JOB.SUBMIT(code,'insert into xuxiang_job
values(sysdate);',SYSDATE,'SYSDATE+1/(24*60*60)');
    commit;
end;
/
```

2）查询脚本号，在数据字典(USER_JOBS)中查询：

```
select JOB,what from USER_JOBS;
```

3）删除脚本

```
begin
DBMS_JOB.REMOVE(161);
end;
/
```

PL/SQL 笔记于 2008 年 10 月 8 日全部完成！！！

---

ant学习笔记

==========

主讲：张海斌

一、什么是 ant?

ant 是构建工具，apache 的开源项目。

什么是构建?

概念到处可查到，形象来说，你要把代码从某个地方拿来，编译，再拷贝到某个地方去等等操作。

二、为什么使用 ant?

1、服务器上部署的时候

当你的程序开发完成，部署人员要部署在服务器上的时候，总不能因为因为安装一个程序就配置一个 eclipse+myeclipse 吧，ant 在这个时候是个很好的选择，因为它小巧，容易配置，你带着你写好的 build.xml 到任何一台服务器上，只需要做简单的修改（一些设定，例如目录），然后一两个命令完成，这难道不是一件美好的事情吗。

2、优点：

跨平台 --因为 ant 是使用 java 实现的，所以它跨平台

使用简单 --与 ant 的兄弟 make 比起来

语法清晰 --同样是和 make 相比

功能强大 --ant 能做的事情很多.

三、构建 ant 环境

要使用 ant 首先要构建一个 ant 环境，步骤很简单：

1）安装 jdk，设置 JAVA_HOME ,PATH ,CLASS_PATH

2）下载 ant 地址 http://www.apache.org

3）解压 ant 你得到的是一个压缩包，解压缩它，并把它放在一个尽量简单的目录，例如 D:\ant-1.6

4）设置 ANT_HOME PATH 中添加 ANT_HOME 目录下的 bin 目录

5）测试一下你的设置,开始-->运行-->cmd 进入命令行-->键入 ant 回车,如果看到

Buildfile: build.xml does not exist!
Build failed

那么恭喜你你已经完成 ant 的设置!

---

ant 笔记于 2008 年 10 月 15 日全部完成！！！

Web Service 学习笔记

==========

主讲：邓远辉

前言

1、why?

适应异构系统的需求，适应不同语言交换的变化。

目前应用领域：系统集成

JCA：是 java 中用来进行系统集成的一种技术，但是前提是调用者必须是 java 程序。

企业系统集成：基于互联网的商务系统和商务系统(B2B)伙伴之间的集成。

2、what?

使用 URI 标识的软件应用。

第一章

1、WebService(WS)的概念：使用 URI 定位的应用程序，借助于 XML 进行描述、发现以及数据传输。

Web 服务：是 w3c 组织制定的一个行业标准。是一个软件应用，通过一个 URI 来进行定位。web 服务的描述和访问都基于 xml，具体的实现和访问跟语言和平台无关。web 服务的交互通过基于 xml 的 soap 消息完成的, 往往需要将 soap 消息架在 http 传输协议上。

1）C/S：一台服务器多态客户端；

2）B/S：多台服务器多台客户端；

3）WebService：多台服务器多台客户端，服务器既可以是服务器也可以是客户端，它只是一种补充。适合于企业之间的数据交换。是程序和程序之间的自动交互，而不是人与

程序进行交互。依赖于 xml。

2、WS 的特征：3 个

使用 XML，能让企业间的交互做到自动进行。

DAO：Data Access and Active domain Object（数据访问和活动域对象）

MVC 模式：目的是便于维护

V：接受请求，显示响应结果；(“a.jsp”不允许出现在视图层)

第二章

1、WS 中的三个对象：

1）服务提供者：提供 web 服务的个人或者公司

2）服务使用者：使用 web 服务的其他个人或者公司系统

3）服务注册中心：服务的提供者用于发布服务的地方，服务的使用者可以从注册中心查找需要的服务(语义查询)。

WS 并不能使代码更加健壮

SUN ONE：开放的网络环境

Axis：Apache 开发的引擎

2、WS 包含的三个标准：

1）SOAP(简单对象访问协议)：对数据传输内容进行封装

解决数据传递的问题（表达数据传输的格式）

制定了与 web 服务通信时发布和接收的消息的规范

2）WSDL(web 服务描述语言)：web service describe language

制定了描述 web 服务的一个规范

3）UDDL(统一发现描述集成语言)：unified discover dericte language

制定了发布和查找 web 服务的一个规范

WSDL4j：

类名：org.apache.axis.wsdl.WSDL2Java

应用参数：http://192.168.1.218:8888/axis/Hello.jws?wsdl

TcpMonitor：

类名：org.apache.axis.utils.tcpmon
监听端口：本机--->9999
目标地址：192.168.1.218
目标端口：8888

WS 笔记于 2008 年 11 月 1 日全部完成！！！