# 1：linux

## 1.1：基础

### 1.1.1：档案权限与目录配置

chmod：改变权限

各权限分数 r:4 ； w:2 x:1

方法1:chmod [-R] xyz 档案名 #-R是递归整个文件夹里的内容使其一并更改权限

方法2:chmod u=rwx,g=rwx,o=rwx,a=rwx 档案名 # a包括了u,g,o

su 切换用户 exit #离开用户

chgrp #改变档案所属群组 chgrp groupName xxx

chown #改变档案拥有着 chown root xxx

目录：

/etc/passwd #保存所有账户信息

/etc/shadow #保存密码信息

/etc/保存组信息

######################关于预设权限######################

umask 查看预售权限 输出0022

umask -S 输出u=rwx,g=rwx,o=rx

0022的意思我们只关注后三位，意思是被拿掉的权限分，例如022意思就是

用户有全部权限，组被拿掉写入权限，其它用户被拿掉写入权限

注意：档案默认不拥有X权限；

umask 000 #改变预设权限,默认新建的目录用有所有权限

######################特殊权限######################

SUID ,SGID,SBIT ；

SUID：一个用户在执行有此权限的二进制文件时，暂时拥有该文件所有者权限

SGID：...暂时拥有该文件所属群组权限

权限分： SUID:4 SGID:2 SBIT:1

增加特殊权限:chmod u=rwxs,g=rwxs ...

### 1.1.2：防火墙

检查是否安装防火墙 rpm -qa

如果没有安装改组件，可以通过yum install iptables进行安装(必须联网)

查看防火墙状态 service iptables status

立即停止防火墙 service iptables stop

立即启用防火墙 service iptables start

查看端口开放情况 Iptables -L -n

### 1.1.3：其它

复制文件到其它机器 scp ./uc-jsw.zip root@192.168.11.211:/usr/local/

查看进程 ps -ef|grep tomcat

实时输出 tailf xxx.log

查看整个文本 cat xxx

man ls #man用于查看某个命令的用法

ldd /usr/local/nginx/sbin/nginx) #ldd检测某条命令需要哪些共享库文件的支持

ldconfig -p | grep libpcre #查看libpcre动态库是否存在

/sbin/ldconfig #刷新动态库缓存

##########ifconfig 为同一ip配置多个别名##########

ifconfig eth0:0 192.168.1.90 netmask 255.255.255.0 up

netstat -lnp 8080 查看端口对应进程

#### 1. 查看时间和日期

命令 ： date +'%Y-%m-%d %H:%M'

#### 2.设置时间和日期

例如：将系统日期设定成2009年11月3日的命令

命令 ： "date -s 11/03/2009"

将系统时间设定成下午5点55分55秒的命令

命令 ： "date -s 17:55:55"

#### 3. 将当前时间和日期写入BIOS，避免重启后失效

命令 ： "hwclock -w"

### 1.1.4：linux档案与目录管理

. #此层目录

.. #上层目录

- #前一个工作目录

~ #当前用户家目录

~root #root用户家目录

chgrp:改变文件或档案所属组 chgrp 组名 档案名

chown：改变文件或档案所属用户; chown 用户名 档案名

cp [

-r #递归复制目录

-a #复制全部档案信息] 源路径 新路径

-i #若目标文件已经存在，在覆盖前进行提示

-p #连同档案属性一起复制过去

]

rm [

-r #递归删除

-f #忽略警告信息

-i #二次确认

]

mv[

-f #若目标文件存在，则强制覆盖

-i #若目标文件存在，则询问是否覆盖

-u #若目标文件存在且source比较新，则进行覆盖

]

mget 下载路径 //下载文件至当前目录

cd #变换路径（change directory）

pwd #显示当前路径(print working directory)

mkdir #[p可建立多层路径][m 后面可加权限分数] 建立文件夹

rmdir #[r递归删除，若不加只能删除当前目录，且必须为空目录] 删除目录

echo #打印，列出

$PATH #$表示后面接的是变量

echo $PATH #打印的是可执行文件(命令)的搜索路径，PATH="$PATH":/root 意思是将root目##录加入path中；

ls [

-a #包括隐藏档案全部档案

-l #显示完整档案内容

-h #显示档案容量

-r #将排序结果反向输出

-t #按时间排序

-i #显示完整的档案名，显示inode

-S #按容量大小排序

--full-time #时间格式化显示

--time=mtime|ctime|atime #显示的时间类型mtime是modification time缩写，当内容数据变更时，此时间将会更新，默认显示此类型时间； ctime是status time，当档案状态比如权限等改变时，此时间将改变；atime是access time，内容被读取时，时间改变

]

输入 basename|dirname  /etc/yum/version-groups.conf

basename #取得最后档案名 输出：version-groups.conf

dirname #取得路径名 输出： /etc/yum

############文档内容检索####################

cat #由第一行开始显示档案内容

[

-A #显示特殊符号

-n #显示行号

]

tac #由最后一行开始显示

nl #输出行号显示

head # head -2 xxx 从头开始显示两行

tail # tail -2 xxx 从尾开始显示2行

od #以二进制显示文件内容

more #页一页显示档案内容

[

空格键:向下翻一页

回车键：向下翻一行

:f 立即显示当前行

q 立即离开More

b 往回翻页

/ 输入/后再输入要搜索的字符串再按回车

n 继续往下搜索同字符串

]

less#一页一页显示档案内容

touch #建立新的档案

file #查看文档类型

####################文档检索####################

连续按两次tab键可以列举可执行指令

which #搜索可执行指令路径 ，例如 which redis-cli

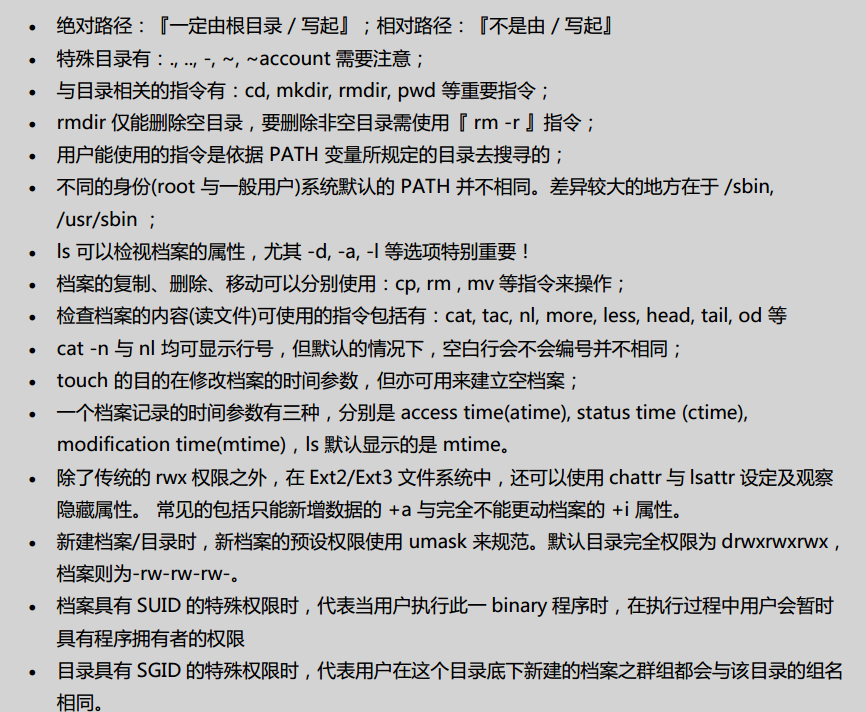
[

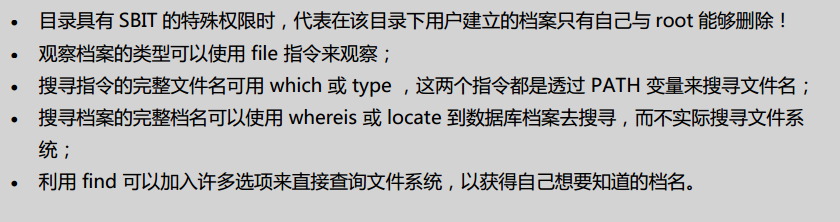
-a #列出path下所有能匹配到该指令的路径。默认只列出首次匹配的

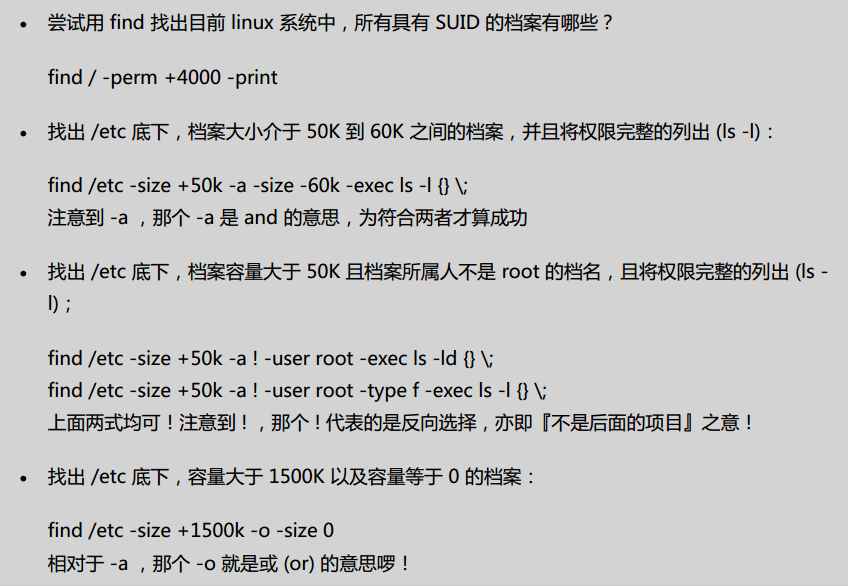
]

whereis #在数据库中搜寻档案 ,例如 whereis aaa

find :在硬盘中搜索







### 1.1.5：linux磁盘与文件系统

df [指定目录] #列出文件系统使用状况

[

-k #以kByte容量显示文件系统

-m #以mByte容量显示文件系统

-h #以G/M/K等能读懂的单位显示容量

-T #显示文件系统类型，比如EXT3...

-i #不用硬盘容量，而用inode的数量来显示

]

du -sh #列出此目录下总使用容量

ln 源档案名 链接档案名 #建立实体链接，例如 ： ln test/cy testLn

ln -s 源路径或者档案名 链接路径 #建立快捷方式

fdisk -l ：显示磁盘信息

####################挂载ISO文件####################

1>建立挂载点

mount -o loop /tmp/isoloop xxx.iso

这是用df就可以看到效果了，可以进入/tmp/isoloop查看iso文件内容

####################建立大档案进行挂载####################

1>首先建立一个大型档案

dd if=/dev/zero of=/tmp/swaploop bs=1M count=512

bs是block大小，count是总共几个bs

2>格式化档案(就像是格式化磁盘分区)

mkfs -t ext3 /tmp/swaploop

3>建立挂载点

mount -o loop /tmp/swaploop /tmploop/ #将/tmp/swaploo这个档案(磁盘分区)挂载到/tmploop

umount /tmploop #卸载挂载点

########################配置swap内存置换空间######################

1>用dd建立一个大型档案

dd if=/dev/zero of=/tmp/swaploop bs=1M count=5000

2>将档案格式化成swap文件格式

mkswap /tmp/swaploop

3>启动swap

swapon /tmp/swaploop

swapoff /tmp/swaploop #关闭swap

接下来可以使用free查看内存信息看到效果！

### 1.1.6：档案与文件系统的压缩与打包

常见压缩文件格式

.gz:gzip程序压缩的档案

.gz2:bzip2程序压缩的档案

.tar:tar程序打包的档案，未压缩

.tar.gz:tar打包且经过gzip压缩

.tar.bz2：tar打包且经过bzip2压缩

gzip|bzip2

[

-v #压缩文件指令

-d #解压缩指令

-number #压缩等级，默认为6，最高为9，-1最快压缩最差，9最慢压缩最好

]

zcat|bzcat: 如果压缩的是文本文件，可以用zcat直接查看； 例如：zcat cy.gz

tar #打包指令  
[

-c #建立打包档案

-v #在压缩/解压过程中，输出正在处理的文件名

-t #查看档案内目录数据

-x #解压缩

-z #通过gzip压缩或解压缩

-j #通过bzip2压缩或解压缩

-C #解压缩到特定目录

-f #要处理的档案名，如果是压缩，则是压缩后的文件，比如xxx.tar.gz

-p #保留备份数据的原本权限与属性

-P #

--exclude=FILE #在压缩过程中，排除指定文件

]

打包：tar -cv -f result.tar sourceName

打包并压缩：tar -cvz -f 打包并压缩：tar -cvz -f result.tar.gz sourceName

.tar.gz sourceName

打包并压缩：tar -cvj -f result.tar.bz2 sourceName

解包到当前目录：tar -xvf result.tar

解包到指定目录：tar -xvf result.tar -C xxx

如果是打包并压缩的就可以使用:tar -xvz|j f...

查看文档内容：tar -tvf result.tar

##############只解压指定文件：###############

1>先找到要解压的文件

tar -tvzf result.tar.gz |grep 文件名

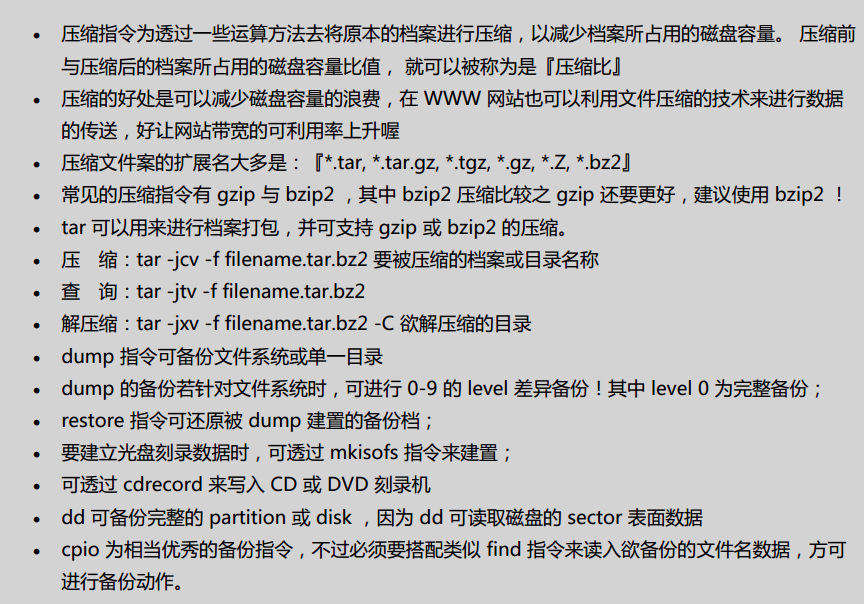
假设要解压testgzip/t1这个文件到result文件下

接下来：tar -xzv -C result -f testgzip.tar.gz testgzip/t1

###########一边压缩一遍解压缩到当前目录########

cd /usr/cy

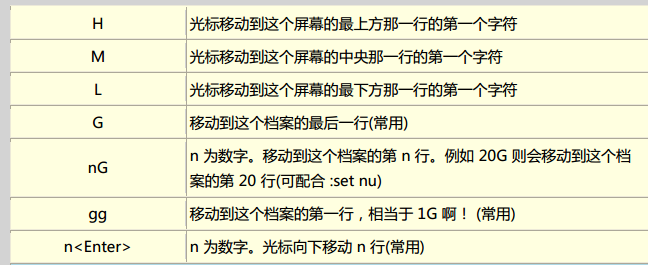
tar -cvzf - ../testgzip|tar -xvzf -



### 1.1.7：VIM程序编辑器

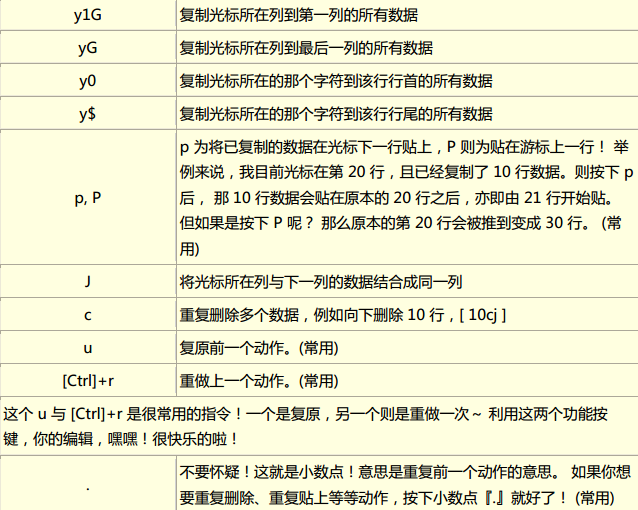
**vi命令大全：**

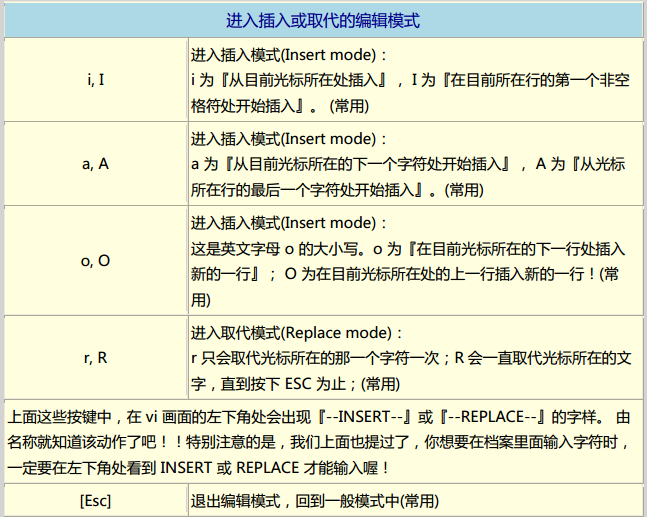






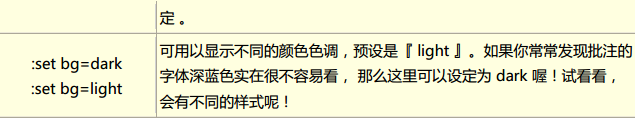






vim编辑器





### 1.1.8：shell编程

alias cy=”ls -al” #指令别名

#################################变量#################################

变量：命令约定:大写为系统变量，小写为程序变量，

查看某变量：echo ${PATH}

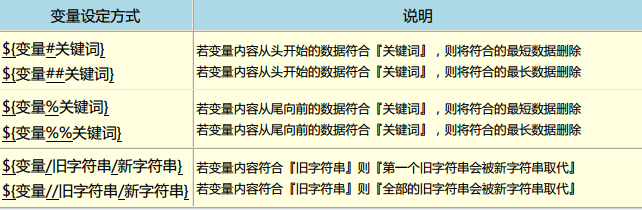
新增变量：mypath='/utc/tmp'，也可以：mypath=${PATH}

往变量追加内容：mypath=${mypath}'xxxx'

将这个自定义变量设为环境变量(全局变量)，让其它子程序能使用:export mypath

删除变量:unset mypath

注意:设置变量时，如果有特殊字符，需要用\进行转义



一个使用列子，假设一个变量值为tmp=/a/b/c

则可以用 cd /test${tmp}/d进入这个目录！

env:列出系统默认环境变量

set：列出全部变量

PS1变量比较有意义

$的意义是：当前shell的PID，可以用 echo $$查看

?号的意义是：返回上一个执行指令的结果，如果非0，则上一个指令有问题！

read:读取来自键盘输入的变量

实例：read -p "please input you are name:" -t 20 youName

解释：-p是提示信息，-t是等待20秒， youName是变量，用来保存输入的值

declare:用来声明变量，可以指定变量类型

declare -i sum=1+2+3 ,echo ${sum} ===>>>6 :i是声明为Integer类型

declare -p sum ;===>>>declare -i sum="6";显示变量详情

declare -x sum ;===>>>相当于export功能，将变量声明为全局变量

declare +x sum ;===>>>重置为局部变量

declare -r sum; ===>>>设置此变量为只读

declare -a sum ===>>>声明数组

sum[1]="cyyy" ===>>>给数组赋值

echo ${sum[1]} ===>>>输出cyyy

echo "ccc" ===>>>直接输出ccc

echo $? 是拿到上一个命令执行后的返回值

newEle=${oldEle-caiyyy} ===>>>给newEle变量赋值，如果oldEle为空，则赋值caiyyy,否则是oldEle值

newEle=${oldEle:-caiyyy} ===>>>如果oldEle是空字符串，也会赋值caiyyy,

clear:清除屏幕信息

alias:查看当前所有别名

使用：alias vi='vim'

移除别名 unalias vi

history -20:显示最近20条命令记录

history -c:清除记录

!6 ； 执行第6个命令

#################################数据流重导向#################################

1> 表示写入正确输出

1>> 表示追加写入正确输出

2> 表示写入错误输出

2>>表示追加写入错误输出

实例：

ll /usr > /test/out ===>>>将usr目录打印信息写入 test/out里，如果out不存在，则自动创建，如果out有内容，则先清空再写入

ll /aaa 2> /test/out2 ===>>>

ll /usr > /test/out3 2> test/out4 ===>>>如果执行正确则写入 out3,否则写入out4

ll /usr &>> out4 ===>>>将错误或者正确的信息都往out4里面追加

cat xxx 2> /dev/null ===>>>将可能会出现的错误信息丢掉，/dev/null 这个文件是个黑洞

cat >out ===>>>创建一个out文件，并等待写入键盘输入的数据

cat >>out <out2 ===>>>将out2文件内容追加到out里

cat >>out <<"cy" ===>>>等待键盘输入，如果输入cy，则结束输入

#################################逻辑符号#################################

ls /etc/out &> /dev/null && echo "exist" || echo "not exist"

解释：检查/etc是否有out这个档案或目录，且忽略掉输出信息， 如果有，则打印esist，否则打印not exist

#################################管线命令#################################

last | cut -d ':' -f 1 #last是显示登录信息，|是管线命令的标记，cut是用来处理每一行数据，

#-d是将每行数据按照:进行分割，然后-f取出数组里第一条数据

grep【 #用来取出符合条件的数据

-c 计算匹配数据的次数

-i 忽略大小写

-n 顺便输出行号

-v 反向选择

】

wc【

-l 列出行数

-m 多少字符

】

tee#双重导向，既可以写入文件也同时可以输出到屏幕

ls -l /etc |tee ./out1 |tee ./out2 #将打印内容写入out1和out2

tr #删除或者替换字符

last |tr '[a-z]' '[A-Z]' #将打印last的信息小写字母换成大写

last | tr -d 'ops' #将打印信息的ops去掉

split【 #分隔档案

-b 以大小分隔

-l 以行数分隔

】

split -file1 -b 1k

#################################正则表达式#################################

grep -n 'ttt' mytxt #从mytxt中找出有ttt的行，并显示行号

grep -vn 'ttt' mytxt #反向选择

grep -in 'ttt' mytxt #忽略大小写

grep --color='auto' -in 'ttt' mytxt #给关键字附上颜色

grep 't[ae]st mytxt #tast或者test都能被搜出来

grep '[^g]oo' mytxt #搜寻oo字符，但前面不能有g

grep '[^a-z]oo' mytxt #搜寻oo字符，但前面必须是大写

[]：是字符区间

^ :如果放到[]中，则是反向选择的意思，如果是字符前面，则是开头

grep '^[a-z]' mytxt #查找以小写字母开头的

grep 'o$' mytxt #查找以o结尾的

.是一个任意字符

\*是0个或者多个字符

grep 'g.\*g' mytxt #搜索已g开头g结尾，中间任意字符的数据

grep 'g..g' mytxt #只会搜出 goog gxxg这种字符串

grep 'go\*g' mytxt #意思是o这个字符可以出现也可以不出现 那么 gg可以搜出，gog也可以搜出

grep 'o\{1,3\}'mytxt #找出有o字符1-3个

grep 'o\{1,\}'mytxt #找出有o字符1个以上

grep 'o\{1}'mytxt #找出有o字符1个

sed工具：

nl mytxt | sed '2,5d' #不显示2-5行

sed -i '2,5d' mytxt #直接删除2-5行，会修改档案内容

diff -bBi mytxt mytxt2 #比较这两个档案内容

【

-b #忽略一行当中的多个空白差异

-B #忽略空白行差异

-i #忽略大小写

】

nl mytxt |grep 'o' 也可以写成 grep 'o' mytxt

sh 【 #执行脚本

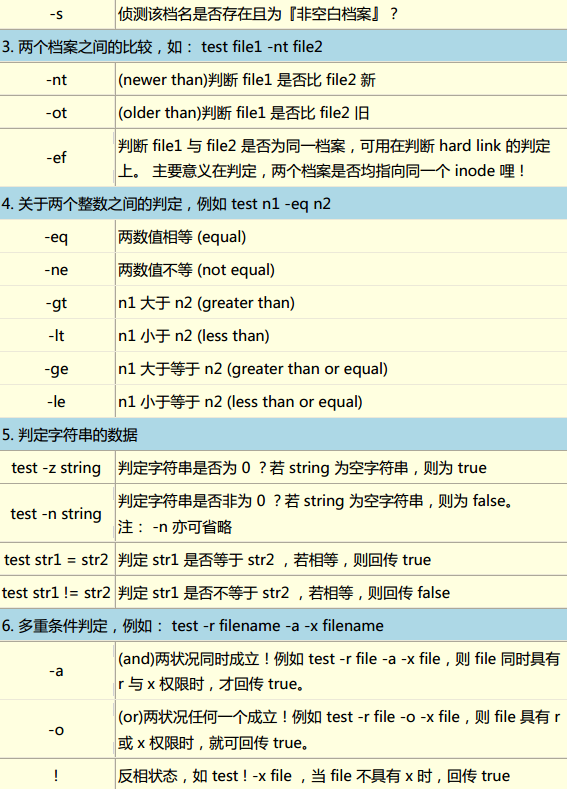
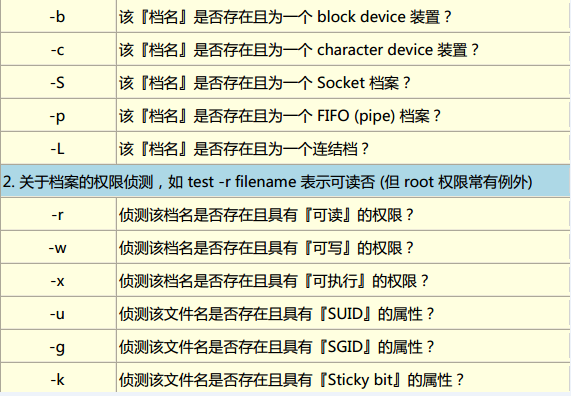
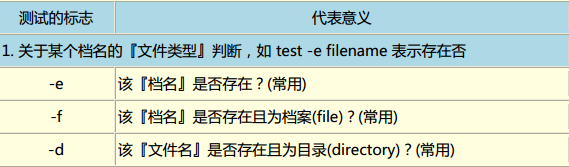
-n #不执行script，仅检测语法是否有问题

-v #执行前，先打印内容到屏幕

-x #打印脚本执行过程

】





## 1.2：服务安装

### 1.2.1：第三方包通用命令

用于linux源码安装软件，一般下载源码包得到文件：xxxx.tgz

解包软件 tar zxf xxxx.tgz

配置 cd xxxx ./configure

编译 make

安装 make install

卸载 make uninstall

### 1.2.3：mysql的安装

1

查找以前是否安装有mysql，使用下面命令：

rpm -qa|grep -i mysql

如果显示有如下包则说明已安装mysql

mysql-4.1.12-3.RHEL4.1

mysqlclient10-3.23.58-4.RHEL4.1

2

如果已安装，则需要删除已安装的数据库，使用以下命令来删除数据库

删除命令：rpm -e --nodeps 包名

( rpm -ev mysql-4.1.12-3.RHEL4.1 )

删除老版本mysql的开发头文件和库

命令：rm -fr /usr/lib/mysql

rm -fr /usr/include/mysql

注意：卸载后/var/lib/mysql中的数据及/etc/my.cnf不会删除，如果确定没用后就手工删除

rm -f /etc/my.cnf

rm -fr /var/lib/mysql

3：查看当前系统版本，用来选择下载对应的mysql安装包  
cat /proc/version print:Linux version 2.6.32-431.el6.i686 ，这代表是32位

进入mysql官网：http://downloads.mysql.com/archives/community/

找到以下三个文件且用wget下载下来rhel5.是企业版

1. MySQL-server-5.6.10-1.rhel5.x86\_64.rpm：MySQL服务器；
2. MySQL-client-5.6.10-1.rhel5.x86\_64.rpm：MySQL客户端；
3. MySQL-devel-5.6.10-1.rhel5.x86\_64.rpm：Mysql开发依赖包。

4：rpm -ivh ... （安装下载好的3个文件）

（要移除安装可以使用 rpm -e MySQL-server 、 rpm -e MySQL-devel、MySQL-client即可；）

如果万一安装mysql-server失败，可能是包冲突了。建议先rpm -qa|grep -i mysql

找到冲突的libs包，再 yum -y remove mysql-libs-xxx\* 移除冲突的libs包，在重新安装MYSQL

5：初始化mysql数据库

/usr/bin/mysql\_install\_db

6:开始启动mysql，service mysql start 或者/etc/init.d/mysql start ，如果启动报错，比如：Starting MySQL. ERROR! The server quit without updating PID file ，需要看usr/local/mysql下的某些文件有没有写入权限！

7:安装后，root默认是没有密码，设置密码是：update user set password=password('123') where user='root' and host='localhost';

 flush privileges;

如果忘记了root密码，则要去/usr/my.conf或者/etc/my.conf这配置文件中增加skip-grant-tables，意思是启动MySQL服务的时候跳过权限表认证。 这样登录的时候就不需要输入用户名和密码，登录进去后，再修改root密码；

8:登录进去muql，修改连接域，让外部机器可以访问到linux下mysql服务

再次设置密码（我也不知道意思，根据提示做的）:SET PASSWORD = PASSWORD(‘root’)  
登录:mysql -u root --password=root

修改连接域：UPDATE mysql.user set host =’%’ where user=’root’

### 1.2.4：jdk安装

解压jdk

设置环境变量：export JAVA\_HOME=路径

PATH=${PATH}:${JAVA\_HOME}/bin

注意：上面方法只会在当前登录bash环境中有效，一旦机器重启，则需要重新设置

#将变量设置为永久变量

编辑 /etc/profile文件，将以上变量写入脚本中

source /etc/profile 重新加载

## 1.3：网络相关

重启网卡 service network restart

## 1.4：系统监控

# 2：windows

## 2.1：查看并杀死进程(用于端口占用)

netstat -aon | findstr "8081" #查看端口是否被占用

tasklist | findstr "3096" 找到PID为3096的进程

# 3：mysql

## 3.1：随记

删除索引：DROP INDEX indexName ON dbName.tableName

查看索引：SHOW INDEX FROM tableName

创建索引：

ALTER TABLE dbName.tableName ADD INDEX indexName(columnName);

创建联合索引：

CREATE INDEX column1\_column2\_column3 ON commodity(column1,column2,column3)

USE prms\_db;

DROP PROCEDURE IF EXISTS index\_add;

DELIMITER ;;

CREATE PROCEDURE index\_add()

BEGIN

DECLARE CurrentDatabase VARCHAR(100);

SELECT DATABASE() INTO CurrentDatabase;

IF NOT EXISTS (SELECT \* FROM information\_schema.statistics WHERE table\_schema=CurrentDatabase AND table\_name = 'panicbuy\_remind' AND index\_name = 'index\_commodity\_id') THEN

ALTER TABLE prms\_db.panicbuy\_remind ADD INDEX index\_commodity\_id(commodity\_id);

ELSE

SELECT 'exist';

END IF;

END;;

DELIMITER ;

CALL index\_add();

删除表：DROP TABLE IF EXISTS `tableName`;  
插入数据：INSERT IGNORE INTO tableName (columnName1,columnName2) values (‘’,’’);

新增字段：

-- 创建一个存储过程判断字段是否存在

drop PROCEDURE if EXISTS add\_column;

DELIMITER ;;

create PROCEDURE add\_column()

BEGIN

-- TABLE\_SCHEMA 数据库名

-- table\_name 表名

-- COLUMN\_NAME 列名

if not EXISTS(select 1 FROM information\_schema.COLUMNS WHERE TABLE\_SCHEMA='cims\_db' AND table\_name='on\_shelf' AND COLUMN\_NAME='status\_type') then

-- 要执行的字段新增

ALTER TABLE `cims\_db`.`on\_shelf`

ADD COLUMN `status\_type` VARCHAR(50) NULL DEFAULT '0' COMMENT '状态类型' AFTER `status`;

ELSE

select 'exist';

end if;

end

;;

DELIMITER ;

CALL add\_column();

-- 结束存储过程创建

**创建表：**

CREATE TABLE IF NOT EXISTS test\_proc(

id VARCHAR(32) NOT NULL COMMENT '主键id',

default\_name VARCHAR(100) NULL DEFAULT NULL COMMENT '测试名',

PRIMARY KEY(id),

INDEX name\_index (default\_name)

)

ENGINE = INNODB

DEFAULT CHARSET = utf8

COMMENT = '测试存储过程';

**删除表**：DROP TABLE IF EXISTS test\_proc;

查看所有存储过程： SHOW PROCEDURE STATUS

删除存储过程：drop PROCEDURE if EXISTS name

查看存储过程创建代码:SHOW CREATE PROCEDURE name;

删除表：DROP TABLE IF EXISTS tableName

**2 .数据操纵语言DML**  
数据操纵语言DML主要有三种形式：  
1) 插入：INSERT  
2) 更新：UPDATE  
3) 删除：DELETE

1. **数据定义语言DDL**  
   数据定义语言DDL用来创建数据库中的各种对象-----表、视图、  
   索引、同义词、聚簇等如：  
   CREATE TABLE/VIEW/INDEX/SYN/CLUSTER

## 3.2：函数以及存储过程记录

==============通用创建索引存储过程==========

DELIMITER $$

USE `prms\_db`$$

DROP PROCEDURE IF EXISTS `add\_index\_proc`$$

CREATE DEFINER=`root`@`%` PROCEDURE `add\_index\_proc`(t\_name VARCHAR(100),\_index\_name VARCHAR(100),\_index\_type VARCHAR(100),\_index\_method VARCHAR(100),col\_names VARCHAR(100))

SQL SECURITY INVOKER

BEGIN

DECLARE CurrentDatabase VARCHAR(100);

DECLARE sql\_str VARCHAR(1000);

DECLARE default\_index\_type VARCHAR(100) DEFAULT 'index';

DECLARE flag INT;

SELECT DATABASE() INTO CurrentDatabase;

SELECT 1 FROM information\_schema.statistics WHERE table\_schema=CurrentDatabase AND table\_name = t\_name AND index\_name = \_index\_name ;

SELECT CurrentDatabase,t\_name,\_index\_name,flag;

IF NOT EXISTS(SELECT 1 FROM information\_schema.statistics WHERE table\_schema=CurrentDatabase AND table\_name = t\_name AND index\_name = \_index\_name ) THEN

SET @default\_index\_type='index';

SET @sql\_str=CONCAT('ALTER TABLE ',t\_name);

IF \_index\_type!= '' THEN

SET @default\_index\_type=CONCAT(\_index\_type,' ',@default\_index\_type);

END IF;

SET @sql\_str=CONCAT(@sql\_str,' add ',@default\_index\_type,' ',\_index\_name,'(',col\_names,')');

SELECT @sql\_str,\_index\_type,default\_index\_type,col\_names;

IF \_index\_method!= '' THEN

SET @sql\_str=CONCAT(@sql\_str,' using ',\_index\_method);

END IF;

PREPARE st FROM @sql\_str;

EXECUTE st;

DEALLOCATE PREPARE st;

END IF;

END$$

DELIMITER ;

**调用示例:**

**CALL add\_index\_proc('表名,'索引名','索引类型','',’字段名');**

==============通用添加字段存储过程==============

DELIMITER $$

USE `prms\_db`$$

DROP PROCEDURE IF EXISTS `add\_column\_proc`$$

CREATE DEFINER=`root`@`%` PROCEDURE `add\_column\_proc`(t\_name VARCHAR(100),col\_name VARCHAR(100),datatype VARCHAR(100),remark VARCHAR(255))

SQL SECURITY INVOKER

BEGIN

DECLARE CurrentDatabase VARCHAR(100);

DECLARE sql\_str VARCHAR(1000);

SELECT DATABASE() INTO CurrentDatabase;

IF NOT EXISTS(SELECT 1 FROM information\_schema.COLUMNS WHERE TABLE\_SCHEMA=CurrentDatabase AND table\_name=t\_name AND COLUMN\_NAME=col\_name) THEN

SET @sql\_str=CONCAT('ALTER TABLE ',t\_name ,' ADD COLUMN ' ,col\_name,' ', datatype,' DEFAULT NULL NULL COMMENT \'',remark,'\'');

PREPARE st FROM @sql\_str;

EXECUTE st;

DEALLOCATE PREPARE st;

END IF;

END$$

DELIMITER ;

**调用示例：**

**CALL add\_column\_proc('表名','列名','数据类型','备注');**

## 3.3：性能优化

## 3.4：触发器记录

-- 新增抢购数据

DELIMITER $

CREATE TRIGGER t\_tmp\_snappingitem\_trigger AFTER INSERT

ON t\_ebsite\_snappingitem FOR EACH ROW

BEGIN

DECLARE FID VARCHAR(50);

DECLARE flimited INT(11);

DECLARE fkgoodsid VARCHAR(50);

DECLARE FTITLE VARCHAR(500);

DECLARE FTYPE VARCHAR(50);

DECLARE fidlimeted INT(11);

DECLARE fprice DECIMAL(11,2);

SET FID = new.FID;

SET flimited = new.flimited;

SET fkgoodsid = new.fkgoodsid;

SET FTITLE = new.FTITLE;

SET FTYPE = "ROB";

SET fidlimeted = new.fidlimited;

SET fprice = new.ffavorablePrice;

INSERT INTO t\_tmp\_presaleitem(FID,flimited,fkgoodsid,FTITLE,FTYPE,fidlimeted,fprice) VALUES(FID,flimited,fkgoodsid,FTITLE,FTYPE,fidlimeted,fprice);

END$

DELIMITER ;

-- 修改抢购数据

DELIMITER $

CREATE TRIGGER t\_tmp\_snappingitem\_trigger\_update AFTER UPDATE

ON t\_ebsite\_snappingitem FOR EACH ROW

BEGIN

DECLARE FID VARCHAR(50);

DECLARE fkgoodsid VARCHAR(50);

DECLARE FTITLE VARCHAR(500);

DECLARE fnewlimeted INT(11);

DECLARE fnewidlimeted INT(11);

DECLARE fnewprice DECIMAL(11,2);

DECLARE frecord TEXT(100000);

DECLARE fnewRecord TEXT(100000);

SET FID = new.FID;

SET fkgoodsid = new.fkgoodsid;

SET FTITLE = new.FTITLE;

SET fnewlimeted = new.flimited;

SET fnewidlimeted = new.fidlimited;

SET fnewprice = new.ffavorablePrice;

SET frecord = (SELECT IFNULL(TMP.frecord,"") FROM t\_tmp\_presaleitem TMP WHERE TMP.fid = new.fid);

SET fnewRecord = CONCAT(";",NOW(),":限购量旧值",old.flimited,"变更为",new.flimited,",ID限购旧值",old.fidlimited,"变更为",new.fidlimited,",价格旧值",

old.ffavorablePrice,"变更为",new.ffavorablePrice,",开始时间旧值",old.fStartSaleTime,"变更为",new.fStartSaleTime,",结束时间旧值",old.fEndSaleTime,"变更为",new.fEndSaleTime);

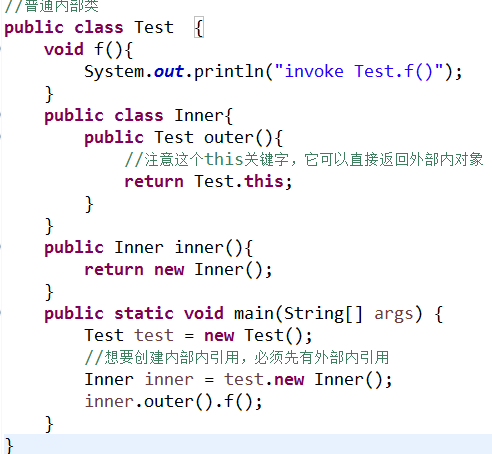
UPDATE t\_tmp\_presaleitem D SET D.fkgoodsid=fkgoodsid,D.FTITLE =FTITLE,D.fnewlimeted = fnewlimeted,D.fnewidlimeted =fnewidlimeted,D.fnewprice = fnewprice,D.FRECORD=CONCAT(frecord,",",fnewRecord) WHERE D.fid = FID;

END$

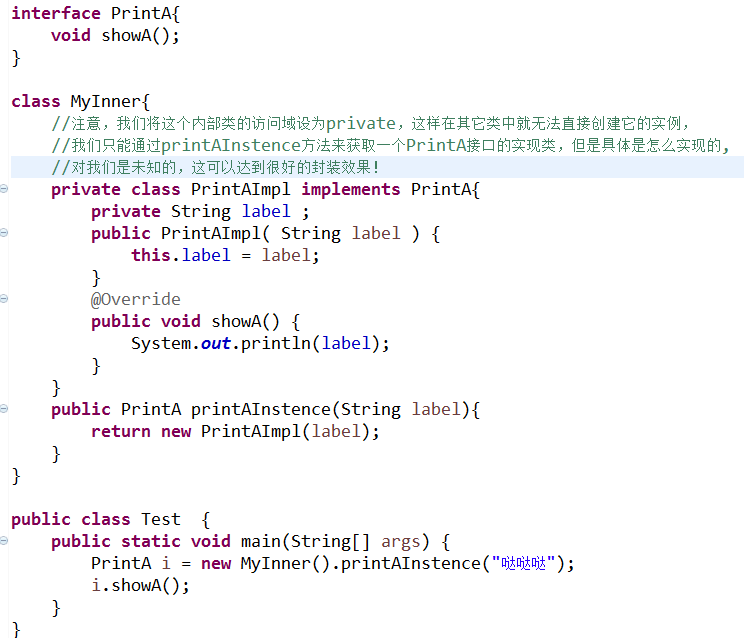
DELIMITER ;

# 4：java

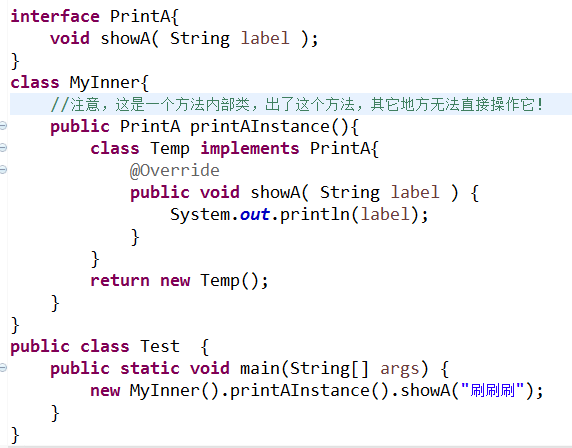
## 4.1：内部类



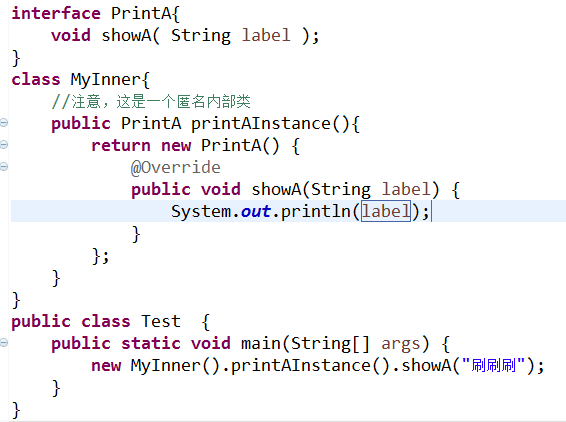
//下面是一个使用private修饰符的列子



//方法内部类



//匿名内部类



## 4.2：API记录

System.*identityHashCode*(obj) ；

org.springframework.util.Assert

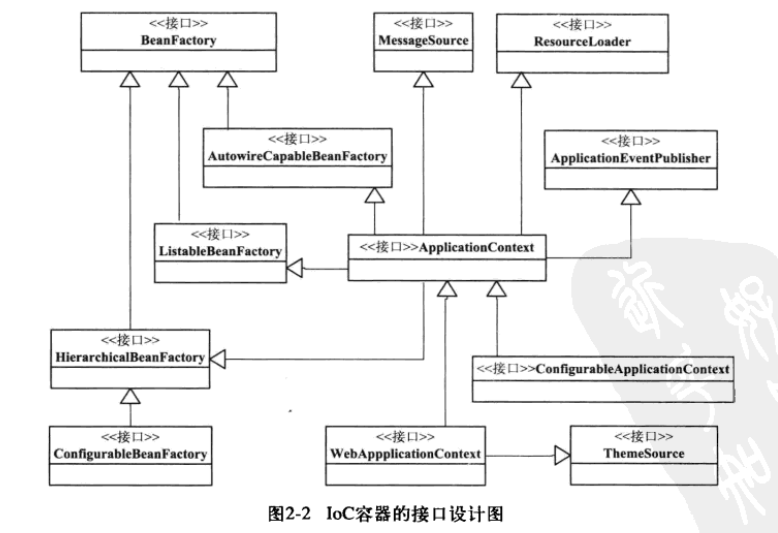
org.springframework.util.StringUtils

# 5：spring

## 5.1：总览

spring呢主要有ioc,aop,事务处理，jdbc，orm，与hibernate，mybatis等的集成，远程调用，还有众多实用的模板，比如sql，redis，mq等等的模板

## 5.2：ioc容器



# 6：redis

## 6.1：redis安装

cd usr/local/down/

wget http://download.redis.io/releases/redis-3.0.7.tar.gz #下载tar.gz文件到当目录

tar xzf redis-3.0.7.tar.gz

yum install gcc

make PREFIX=/uer/local/redis install &&　make　#安装到指定目录

# /etc/rc.d/init.d/目录下的脚本在系统启动的时候某些指定脚本将被执行

cd /usr/local/redis

cp /usr/local/down/redis-3.0.7/utils/redis\_init\_script /etc/rc.d/init.d/redis #复制脚本到注册表

mkdir /etc/redis

cp /usr/local/down/redis-3.0.7 /etc/redis/6379.conf

#设置密码

vi /etc/redis/6379.conf

#requirepass foobared去掉注释，foobared改为自己的密码

#启动服务

service redis start

或者redis-server #后面可以加上指定配置文件的路径，如：redis-server /etc/redis/6379.conf

开启端口 -A INPUT -m state --state NEW -m tcp -p tcp --dport 6379 -j ACCEPT

重新加载防火墙规则 service iptables restart

#链接redis redis-cli

## 6.2：redis集群

可参见博客：<http://www.cnblogs.com/LiZhiW/p/4851631.html#_labelStart>

配置slave1:

slaveof 192.168.1.200 6380 #masterip ,master端口

masterauth 123456 #master的密码

配置slave2:

slaveof 192.168.1.200 6380 #masterip ,master端口

masterauth 123456 #master的密码

* requirepass 在master里，设置身份验证密码，并确定实例不处理没有身份验证的客户端请求。
* masterauth 在slave里，为了slave验证master以便于从master正确的复制数据。

当使用Sentinel时，没有单独的master，因为在故障转移之后slave可以改变master的角色，并且老master可以被重新配置为slave，所以你要做的是在master和slaves的所有实例里都配置上上面的指令。

配置sentinel进行监听

sentinel会进行监听，通知(当监听的节点有问题，会通过api通知连接的客户端)，自动故障转移。

接下来配置两个sentinel

touch sentinel001.conf

touch sentinel002.conf

#################################配置开始######################################

port 26379 #当前Sentinel服务运行的端口

dir /usr/data/redis #Sentinel服务运行时使用的临时文件夹

##############master001监控配置#################

#2代表将这个主实例判断为失效至少需要2个 Sentinel进程的同意

sentinel monitor master001 192.168.1.200 6380 2

# sentinel auth-pass <master-name> <password>

#

# 指定了Sentinel认为Redis实例已经失效所需的毫秒数

sentinel down-after-milliseconds master001 30000

#

#指定了在执行故障转移时，最多可以有多少个从Redis实例在同步新的主实例

#在从Redis实例较多的情况下这个数字越小，同步的时间越长，完成故障转移所需的时间就越长

sentinel parallel-syncs master001 1

#

#如果在该时间（ms）内未能完成故障转移操作，则认为该failover失败

sentinel failover-timeout master001 180000

#指定sentinel检测到该监控的redis实例指向的实例异常时，

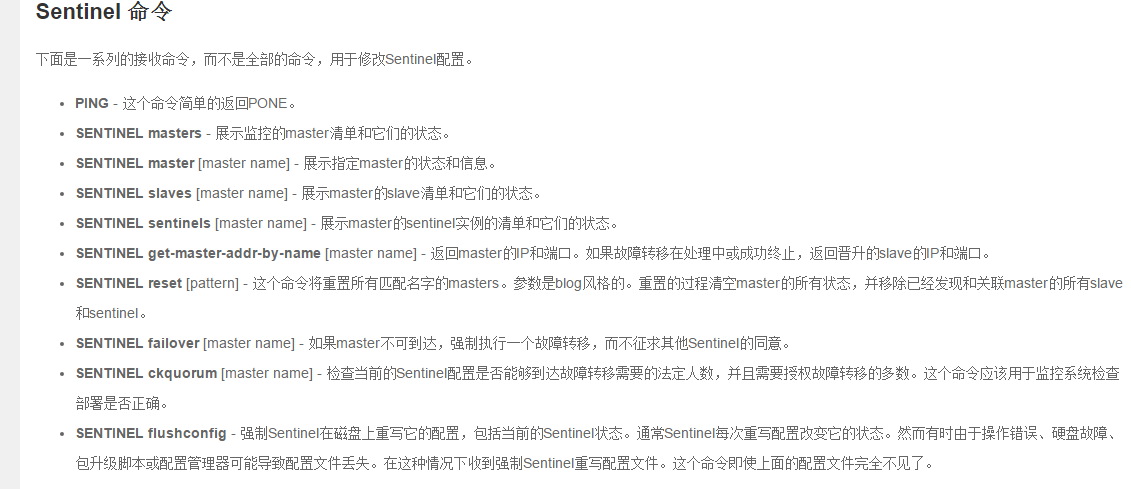
#调用的报警脚本。该配置项可选，但是很常用

# sentinel notification-script <master-name> <script-path>

#

# sentinel client-reconfig-script <master-name> <script-path>

好了。现在可以先启动两个哨兵sentinel，再启动master和两个slave进行测试观察了！

更多配置参考：http://redis.majunwei.com/topics/sentinel.html

## 6.3：应用链接redis

请看详细配置

## 6.4：redis分布式锁实现

# 7：nginx

## 7.1：安装nginx:

官网路径：nginx.org/download/nginx-1.10.2.tar.gz

cd /usr/local/download

#安装c++环境

yum install -y gcc gcc-c++

#安装pcre库

wget http://nchc.dl.sourceforge.net/project/pcre/pcre/8.39/pcre-8.39.tar.gz

tar -xvf ...

./configure

make && make install

#安装zlib库

wget <http://zlib.net/zlib-1.2.8.tar.gz>

tar -xvf ...

./configure

make && make install

#安装openSSL

wget https://www.openssl.org/source/openssl-1.1.0c.tar.gz

...

#依赖库安装完毕，开始安装nginx

wget nginx.org/download/nginx-1.10.2.tar.gz

tar -xvf nginx-1.10.2.tar.gz

cd ./nginx-1.10.2

./configure

默认会安装在 /usr/local/nginx

## **7.2：启动，停止，重启ng**

cd /usr/local/nginx

目录下有4个文件夹，其中sbin里存放的就是ng服务器的主程序

1>启动服务

./sib/nginx

2>停止服务

kill -TERM|INT|QUIT|HUP port #TERM或INT是快速停止ng服务，丢失所有处理中请求，QUIT是平缓停止服务，处理完正在处理的请求，但不接受新的请求。HUP是使用新的配置文件启动进程，之后平缓停止原有进程。也就是所谓的平缓重启；

## 7.3：nginx.conf配置

1：全局指令

user :指定可以运行nginx服务的用户 列：user root;

worker\_processes：指定同时允许创建的工作进程; 列 worker\_processes 3|auto;

pid :指定nginx主进程号存放路径

error\_log:异常日志Log保存配置 ,此指令可在http块，server和location块中配置

例如error\_log logs/error.log debug|info|notice|warn|error|crit|alert|emerg

include :引入其它配置文件,此指令可放任意地方 ；列：include /usr/xxx.conf

2：events配置

acept\_mutex on|off ；默认为On，对process进行序列号，防止多个进程对连接争抢

multi\_accept on|off ;设置是否允许同时接受多个网络请求，默认为off ，off是每个process一次只能接受一个新到达的网络连接

worker\_connections number:每一个process同时开启的最大连接数

use epoll: 配置事件驱动模型

3:http配置

access\_log filePath [formatName]：访问日志,filePath是日志保存路径，formatName是日志格式化字符串名称

sendfile\_max\_chunk 128k;设置每个process调用sendfile()传输的数据量不能超过这个值，默认为0不限制

keepalive\_timeout outTime [keep-alive]：设置服务端对连接的保持时间，keep-alive是设置客户端保持连接时间 ； 例如：keepalive\_timeout 120s 120s ; 该指令可在http，server,location中配置，第二个时间参数是客户端主动与服务器断开连接的事件

keepalive\_requests number配置单连接请求连接上线；默认为100

nginx是一种模块化的结果，基本可以分为核心模块，标准http模块，可选http模块，邮件服务模块和其它第三方模块

send\_timeout #服务器等待时间，也就是会话超时时间 ，例如 send\_timeout 10s;

client\_header\_buffer\_size #允许接受客户端请求头的缓冲区大小,

如： client\_header\_buffer\_size 4k;

## 7.4：配置示范

需求：

访问192.168.1.33:9998/xxx1/xx2/xxx3/x.html需要输出/home/xx1/xss/img/xxx3/x.html文件

配置：

service{

listen 9998;

server\_name localhost;

location ~ ^/xxx1/xx2/(\.+)$ {

alias /home/xx1/xss/img/$1;

}

error\_page 500 502 503 504 /50x.html;

}

~ 表示接下来是用正则进行匹配，^表示要用/xxx1开头，\.+表示任意字符出现一次或者多次,.是特殊字符，需要\进行转义，用$表示已某某结尾，使用(\.+)捕捉字符然后放入$1变量

## 7.5：nginx的模块化

查看ng当前支持的模块：

cd目录到下载的ng编译后文件所在地，cd ./objs ; cat ngx\_modules.c

## 7.6：rewrite功能

rewrite功能依赖于PCRE库

设置一组后台服务器

upstream testServer{ #testServer是服务组的组名

ip\_hash|least\_coon;

server [www.fyrj.com](http://www.fyrj.com) weight=5;

server 127.0.0.1:8080 max \_fails=0 fail\_timeout=30s;

server 127.0.0.1:8081 backup;

server 127.0.0.1:8082 down;

}

#ip\_hash 不能与weight一起用，如果用这个，最好ng是最前方的服务器，否则它得到的ip是前面服务器的ip，就会有问题

# weight是权重，采用加权轮叫策略

#max \_fails请求失败次数

#fail\_timeout 当被检测无效时，这个时间内该服务器都会被标识为无效

#backup 只有当组内其它服务都无效，才会访问此服务，这是备用的标识

#down 永久标识某台服务以及宕机

以下等用到的时候再记录：

重定向的各种指令，if,break,rewirte,rewirte\_log,set ,等等，以及rewirte的三种模式【last,break,permanent】，以及ng的常用内置变量

## 7.7：正向代理与反向代理功能

## 7.8：负载均衡功能

# 8：tomcat

## 8.1：开启远程调试

在catalina.sh中的上面部分加入

CATALINA\_OPTS="-Xdebug  -Xrunjdwp:transport=dt\_socket,address=8000,server=y,suspend=n"

然后在Eclipse中开始配置Remote java application，指定运行tomcat的ip和上面的address端口号，就可以了！

## 8.2：虚拟主机的配置

在Engine节点下添加host节点

配置如：

**<Host** name="www.abc.net" appBase="/var/JavaServer/test" unpackWARs="true" autoDeplo y="true"**>**

**<Alias>**abc.com**</Alias>**

**<Context** path="/hello" docBase="/var/img/hello" debug="0" privileged="true "**/>**

**</Host>**

host节点代表一个虚拟主机，name表示要访问的域名，alias是域名别名，方法abc.com和www.abc.net效果是一样的；appBase表示项目所在路径，当前路径下可以存放多个项目，unpackWARs为true，表示该路径下如果有war包，就会自动解压。autoDeploy表示开启热部署。

Context表示一个访问，当访问 /xxx/hello/a.jpg时，会转换成/xxx/var/img/hello/a.jpg

## 8.3：tomcat的访问日志开启

<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"

prefix="localhost\_access\_log." suffix=".txt"

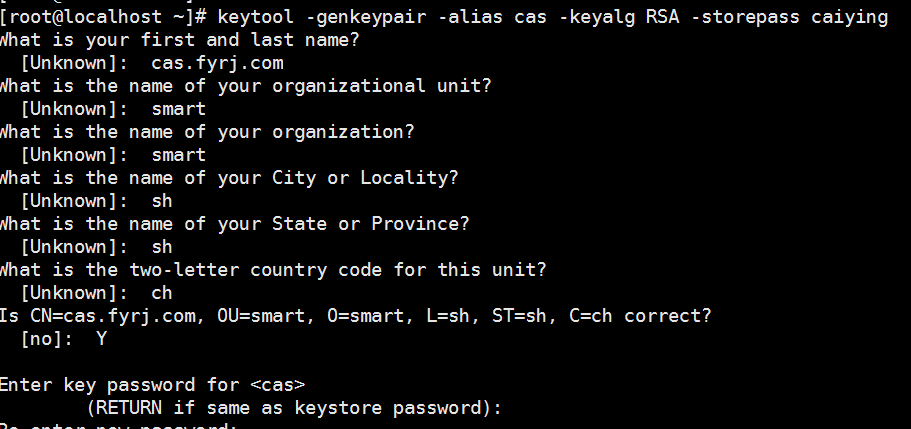
pattern="%h %l %u %{yyyy-MM-dd'T'HH:mm:ss.SSSZ}t &quot;%r&quot; %s %b %D" />

# 9：系统运维

# 10：cas

## 10.1：安装cas

1：利用jdk的keytool生成密钥对



keytool -genkeypair -alias cas -keyalg RSA -storepass caiying

What is your first and last name?

[Unknown]: cas.fyrj.com

What is the name of your organizational unit?

[Unknown]: smart

What is the name of your organization?

[Unknown]: smart

What is the name of your City or Locality?

[Unknown]: sh

What is the name of your State or Province?

[Unknown]: sh

What is the two-letter country code for this unit?

[Unknown]: ch

Is CN=cas.fyrj.com, OU=smart, O=smart, L=sh, ST=sh, C=ch correct?

[no]: Y

Enter key password for <cas>

(RETURN if same as keystore password):

Re-enter new password:

keytool还有几条命令可以用：

查看 keypair：

**keytool -list -storepass caiying**

删除 keypair：

**keytool -delete -alias cas<别名> -storepass caiying**

如果不需要用https，则上面步骤不用做，好，现在先去除https认证

前往官网下载cas3.5.2 ; <https://www.apereo.org/projects/cas/download-cas>

确保jdk,tomcat，等安装好后，配置好tomcat，运行官网下载的包中解压出来的

cas-server-uber-webapp-3.5.2.war

## 10.2：去除https认证

**1、进入WEB-INF\spring-configuration目录**

　　打开warnCookieGenerator.xml文件 修改p:cookieSecure的值为false

**2、打开ticketGrantingTicketCookieGenerator.xml文件**  
　　同样修改p:cookieSecure的值为false

**3、打开WEB-INF\deployerConfigContext.xml文件**  
　　查找org.jasig.cas.authentication.handler.support.HttpBasedServiceCredentialsAuthenticationHandler   
　　把这代码块修改为如下：

<bean class="org.jasig.cas.authentication.handler.support.HttpBasedServiceCredentialsAuthenticationHandler"

p:httpClient-ref="httpClient" p:requireSecure="false" />

# 11：ftp

# 12：rabbit Mq

## 12.1：linux安装rabbit Mq

前往http://www.erlang.org/downloads下载erlang最新版

前往http://www.erlang.org/downloads 下载rabbitMq 的最新版

[http://www.erlang.org/download/otp\_src\_19.2.tar.gz](http://erlang.org/download/otp_src_19.2.tar.gz) #erlang

<http://www.rabbitmq.com/releases/rabbitmq-server/v3.6.6/rabbitmq-server-generic-unix-3.6.6.tar.xz> #rabbitMq

Rabbitmq 是用 erlang 语言写的，所以需要安装 Erlang，安装 erlang 又需要安装 python 与 simplejson,所以从python开始

1、安装 python：

#wget http://www.python.org/ftp/python/2.5.2/Python-2.5.2.tar.bz2

#tar -jxvf Python-2.5.2.tar.bz2

#cd Python-2.5.2

#./configure

#make && make install

#测试：在命令行下输入python，出现python解释器即表示已经正确安装。

2、安装 simplejson：

#wget http://pypi.python.org/packages/source/s/simplejson/simplejson-2.0.9.tar.gz

#tar zxvf simplejson-2.0.9.tar.gz

#cd simplejson-2.0.9

#python setup.py build

#python setup.py install

3、安装erlang

先安装以下依赖包

#yum -y install make gcc gcc-c++ kernel-devel m4 ncurses-devel openssl-devel

#yum install libX\* java\* tk unixODBC unixODBC-devel

#yum -y install ncurses-devel

#yum install ncurses-devel

#yum -y install xmlto

#wget [http://www.erlang.org/download/otp\_src\_19.2.tar.gz](http://erlang.org/download/otp_src_19.2.tar.gz)

#mv tp\_src\_R16B.tar erlang\_R16B #重命名解压后的文件

#cd erlang\_R16B/

#./configure --prefix=/usr/local/erlang --with-ssl --enable-threads --enable-smp-support --enable-kernel-poll --enable-hipe --without-javac

#make && make install

vi /etc/profile #设置环境变量

ERL\_HOME=/usr/local/erlang

PATH=$ERL\_HOME/bin:$PATH

export ERL\_HOME PATH

source /etc/profile #更新环境变量

#执行erl，进入erlang的shell成功安装 测试一下是否安装成功,在控制台输入命令erl

1. 进入解压后的rabbitmq-server

5、安装web管理控制台

cd /opt/mq/rabbitmq/sbin

mkdir /etc/rabbitmq/

rabbitmq-plugins enable rabbitmq\_management

<http://192.168.1.199:15672/> #进入控制台

6、运行 ./sbin/rabbitmq-server

启动rabbitmq服务:

前台运行：./rabbitmq-server start (用户关闭连接后,自动结束进程)

后台运行：./rabbitmq-server -detached

关闭(关闭mq服务和erlang节点):rabbitmqctl stop

关闭(只关闭mq) rabbitmqctl stop\_app

## 12.2：基本概念和命令

通道 ：

队列：

交换机：

路由键：

绑定：

vhost:

虚拟rabbitmq服务，它就相当于一个完整的服务，就像我们的虚拟机和物理机一样；如果用vhost，那么可以用一台机器起一个mq服务，这个mq服务里面再用多个vhost来对不同的应用提供服务，那么每一个vhost里面的通道，队列等等都是独立的，且每个的权限控制也都是独立的，那么就解决了多个应用用同一个mq然后数据混乱命名冲突等问题；

./rabbitmqctl list\_vhosts #查看当前有的vhost

./rabbitmqctl add\_vhost [name] #增加一个vhost

./rabbitmqctl delete\_vhost [name] #删除一个vhost

## 12.3：对消息的处理

1：如果消息发给A，而A收到但是没来得及确认就网络异常断开，则mq会认为这条信息还没有分发，则会发给下一个订阅者B

2：消息若无人订阅，则一直存在队列中，等待有人订阅

3：如果A收到消息不进行确认，那么mq将认为A没有做好处理下一条消息的能力，将不再发送消息给A

4：如果A和B同时订阅队列 queue\_tmp ，那么只会有一个

5：可以直接拒绝消息

## 12.4：消息事务

为什么要事务呢？想想，生产者生产一条消息，发送给mq，但mq并不会返回任何处理信息，到底这条消息有没有成功通过交换机发给各个队列，如果设置了持久化，那它到底有没有真正被写入硬盘（可能写入的时候磁盘挂了），这个时候我们可以用confirm机制来做，如果处理成功，那么服务端将会回调客户端。

## 12.5：权限与用户管理

1. 用户管理

用户管理包括增加用户，删除用户，查看用户列表，修改用户密码。

相应的命令

(1) 新增一个用户

rabbitmqctl add\_user Username Password

(2) 删除一个用户

rabbitmqctl delete\_user Username

(3) 修改用户的密码

rabbitmqctl change\_password Username Newpassword

(4) 查看当前用户列表

rabbitmqctl list\_users

2. 用户角色

按照个人理解，用户角色可分为五类，超级管理员, 监控者, 策略制定者, 普通管理者以及其他。

(1) 超级管理员(administrator)

可登陆管理控制台(启用management plugin的情况下)，可查看所有的信息，并且可以对用户，策略(policy)进行操作。

(2) 监控者(monitoring)

可登陆管理控制台(启用management plugin的情况下)，同时可以查看rabbitmq节点的相关信息(进程数，内存使用情况，磁盘使用情况等)

(3) 策略制定者(policymaker)

可登陆管理控制台(启用management plugin的情况下), 同时可以对policy进行管理。但无法查看节点的相关信息(上图红框标识的部分)。

与administrator的对比，administrator能看到这些内容

(4) 普通管理者(management)

仅可登陆管理控制台(启用management plugin的情况下)，无法看到节点信息，也无法对策略进行管理。

(5) 其他

无法登陆管理控制台，通常就是普通的生产者和消费者。

了解了这些后，就可以根据需要给不同的用户设置不同的角色，以便按需管理。

设置用户角色的命令为：

rabbitmqctl set\_user\_tags User Tag

User为用户名， Tag为角色名(对应于上面的administrator，monitoring，policymaker，management，或其他自定义名称)。

也可以给同一用户设置多个角色，例如

rabbitmqctl set\_user\_tags hncscwc monitoring policymaker

3. 用户权限

用户权限指的是用户对exchange，queue的操作权限，包括配置权限，读写权限。配置权限会影响到exchange，queue的声明和删除。读写权限影响到从queue里取消息，向exchange发送消息以及queue和exchange的绑定(bind)操作。

例如： 将queue绑定到某exchange上，需要具有queue的可写权限，以及exchange的可读权限；向exchange发送消息需要具有exchange的可写权限；从queue里取数据需要具有queue的可读权限。详细请参考官方文档中"How permissions work"部分。

相关命令为：

(1) 设置用户权限

rabbitmqctl set\_permissions -p VHostPath User ConfP WriteP ReadP (VHostPath默认是/ )

(2) 查看(指定hostpath)所有用户的权限信息

rabbitmqctl list\_permissions [-p VHostPath]

(3) 查看指定用户的权限信息

rabbitmqctl list\_user\_permissions User

(4) 清除用户的权限信息

rabbitmqctl clear\_permissions [-p VHostPath] User

rabbitmqctl change\_password user pwd #更改用户密码

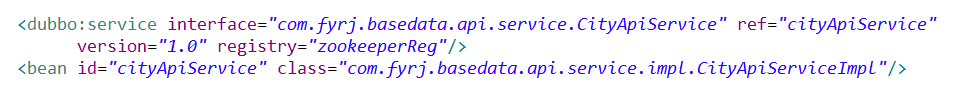
# 13：dubbo

## 13.1：dubbo配置记录

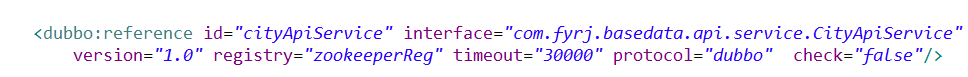
基础中心配置：



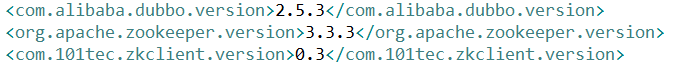
服务提供者配置：

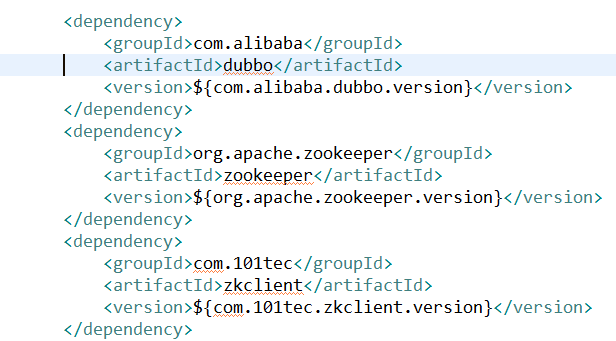


服务消费者配置：



基础依赖包





## 13.2：dubbo服务治理

下载安装dubbo-admin-2.5.3.war

解压到tomcat下，编辑WEB-INF/dubbo.properties配置相应信息

# 14：email

# 15：zookeeper

## 15.1：单机，集群，伪集群等安装和配置

下载zookeeper-3.3.6

#########################单机配置#######################

进入：zookeeper-3.3.6/conf

touch zoo.cfg #增加配置文件

#配置如下内容

tickTime=2000 #基本时间单位，会话超时间是此时间\*2，也就是4秒

dataDir=/usr/data/zookeeper/data #pid文件存放路径，必须先建立该目录

dataLogDir=/usr/data/zookeeper/logs

clientPort=21001 #客户端连接端口

#########################伪集群配置#######################

#########################集群配置#######################

## 15.2：相关命令

ZooKeeper服务命令:

     在准备好相应的配置之后，可以直接通过zkServer.sh 这个脚本进行服务的相关操作

* 1. 启动ZK服务:       sh bin/zkServer.sh start
* 2. 查看ZK服务状态: sh bin/zkServer.sh status
* 3. 停止ZK服务:       sh bin/zkServer.sh stop
* 4. 重启ZK服务:       sh bin/zkServer.sh restart

zk客户端命令

ZooKeeper命令行工具类似于**[Linux](http://lib.csdn.net/base/linux" \o "Linux知识库" \t "http://blog.csdn.net/xiaolang85/article/details/13021339/_blank)**的shell环境，不过功能肯定不及shell啦，但是使用它我们可以简单的对ZooKeeper进行访问，数据创建，数据修改等操作.  使用 zkCli.sh -server 127.0.0.1:2181 连接到 ZooKeeper 服务，连接成功后，系统会输出 ZooKeeper 的相关环境以及配置信息。

**命令行工具的一些简单操作如下：**

* 1. 显示根目录下、文件： ls / 使用 ls 命令来查看当前 ZooKeeper 中所包含的内容
* 2. 显示根目录下、文件： ls2 / 查看当前节点数据并能看到更新次数等数据
* 3. 创建文件，并设置初始内容： create /zk "test" 创建一个新的 znode节点“ zk ”以及与它关联的字符串
* 4. 获取文件内容： get /zk 确认 znode 是否包含我们所创建的字符串
* 5. 修改文件内容： set /zk "zkbak" 对 zk 所关联的字符串进行设置
* 6. 删除文件： delete /zk 将刚才创建的 znode 删除
* 7. 退出客户端： quit
* 8. 帮助命令： help

ZooKeeper 常用四字命令：

      ZooKeeper 支持某些特定的四字命令字母与其的交互。它们大多是查询命令，用来获取 ZooKeeper 服务的当前状态及相关信息。用户在客户端可以通过 telnet 或 nc 向 ZooKeeper 提交相应的命令

* 1. 可以通过命令：echo stat|nc 127.0.0.1 2181 来查看哪个节点被选择作为follower或者leader
* 2. 使用echo ruok|nc 127.0.0.1 2181 测试是否启动了该Server，若回复imok表示已经启动。
* 3. echo dump| nc 127.0.0.1 2181 ,列出未经处理的会话和临时节点。
* 4. echo kill | nc 127.0.0.1 2181 ,关掉server
* 5. echo conf | nc 127.0.0.1 2181 ,输出相关服务配置的详细信息。
* 6. echo cons | nc 127.0.0.1 2181 ,列出所有连接到服务器的客户端的完全的连接 / 会话的详细信息。
* 7. echo envi |nc 127.0.0.1 2181 ,输出关于服务环境的详细信息（区别于 conf 命令）。
* 8. echo reqs | nc 127.0.0.1 2181 ,列出未经处理的请求。
* 9. echo wchs | nc 127.0.0.1 2181 ,列出服务器 watch 的详细信息。
* 10. echo wchc | nc 127.0.0.1 2181 ,通过 session 列出服务器 watch 的详细信息，它的输出是一个与 watch 相关的会话的列表。
* 11. echo wchp | nc 127.0.0.1 2181 ,通过路径列出服务器 watch 的详细信息。它输出一个与 session 相关的路径。

# 16：maven

## 16.1：安装与介绍

windows安装：

官网下载zip文件，解压，设置path环境变量，cmd用mvn -v进行测试就可以了！

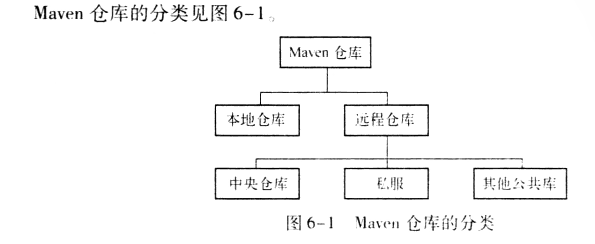
linux安装：

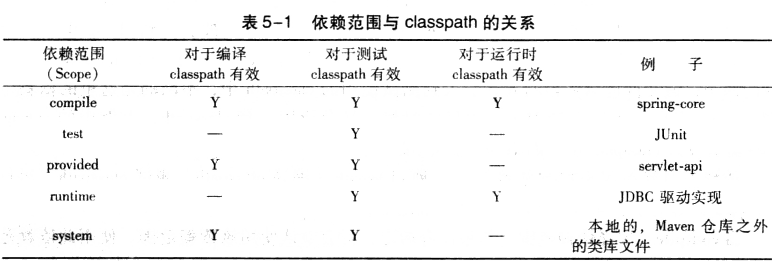
1:确定先安装好了jdk

2: export M2\_HOME=/usr/local/down/apache-maven-3.3.9

PATH=${PATH}:${M2\_HOME}/bin

安装后，可以将conf下的settings.xml复制到家目录下的.m2下，这样修改配置文件就只会针对自己有效，而不会造成一改就改了全局的配置文件，影响所有人！





记住：依赖传递，依赖调解，可选依赖，排除依赖这几个特性，

可选依赖：采用<optional>true</optional>声明，如果A依赖B，B依赖两个可选依赖，那么A是不会传递B的这两个依赖的

排除依赖：



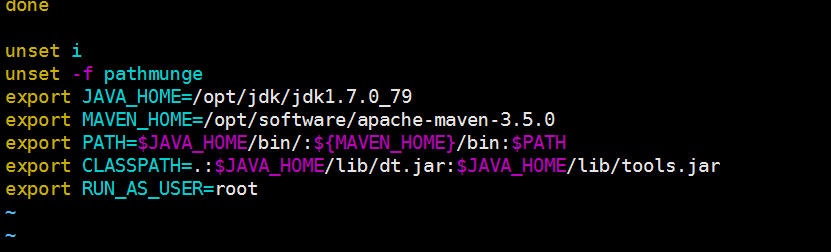
## 16.1：linux安装

前往http://maven.apache.org/download.cgi下载最新包apache-maven-3.5.0-bin.tar.gz

解压后编辑/etc/profile文件，在后面加入以下两行配置

export MAVEN\_HOME=/opt/software/apache-maven-3.5.0

export PATH=$JAVA\_HOME/bin/:${MAVEN\_HOME}/bin:$PATH



使上面配置生效：source /etc/profile

## 16.2：命令大全

mvn dependency:list # 查看当前项目所有已解析依赖

mvn dependency:tree #查到当前项目已解析依赖的树

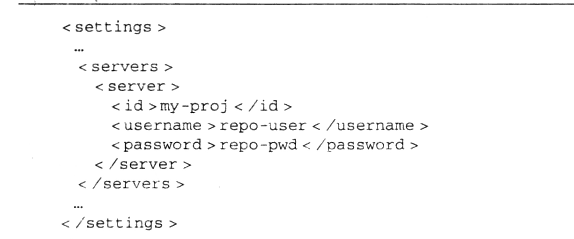
mvn clean deploy #发布版本到仓库

## 16.3：settings.xml详解

#将本地仓库路径设为这里

<localRepository>/path/to/local/repo</localRepository>

**配置认证信息！**



**镜像仓库配置：**

**<mirror>**

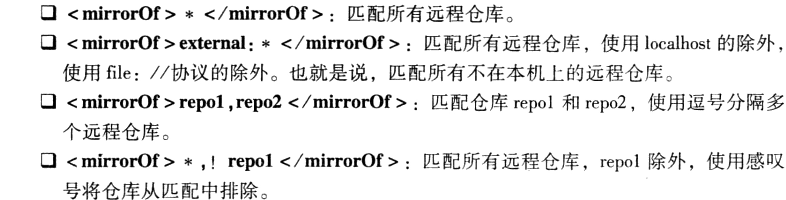
**<id>nexus-public</id>**

**<name>nexus-public</name>**

**<mirrorOf>\*</mirrorOf>**

**<url>http://dev:9999/nexus/content/groups/public/</url>**

**</mirror>**



## 16.4：pom配置详解



**如果用了配置了认证信息的私服或其它要认证的仓库，或者要往仓库发布版本，则需要在settings.xml中配置认证信息！**

**#######################################################################**

**部署到远程仓库的配置，这些配置都应该配置到父pom中，所有子模块项目继承！**

**<distributionManagement>**

**<repository><!--发布版本-->**

**<id>nexus-snapshots</id>**

**<name>Sonatype Nexus Releases</name>**

**<url>${nexus.content}/repositories/releases</url>**

**</repository>**

**<snapshotRepository><!--快照版本-->**

**<id>nexus-snapshots</id>**

**<name>Sonatype Nexus Snapshots</name>**

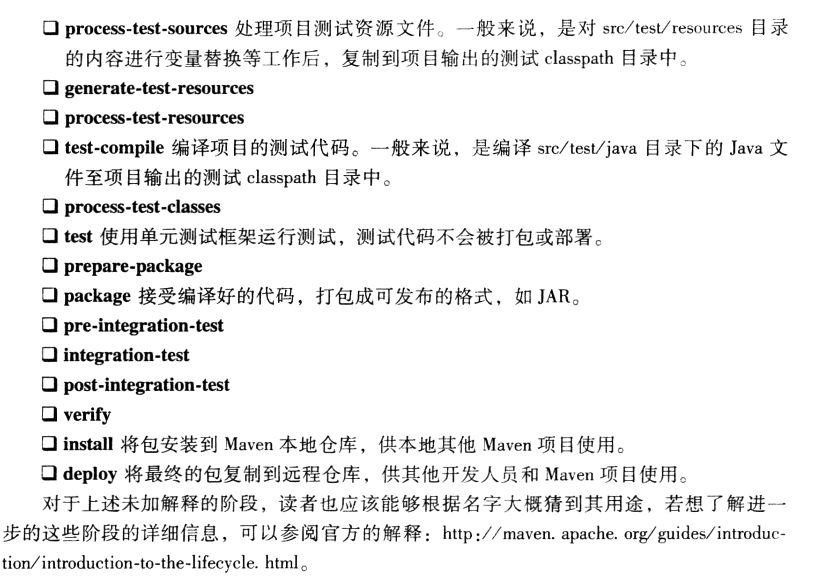
**<url>${nexus.content}/repositories/snapshots</url>**

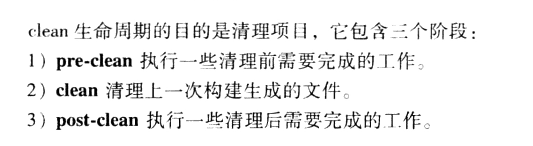
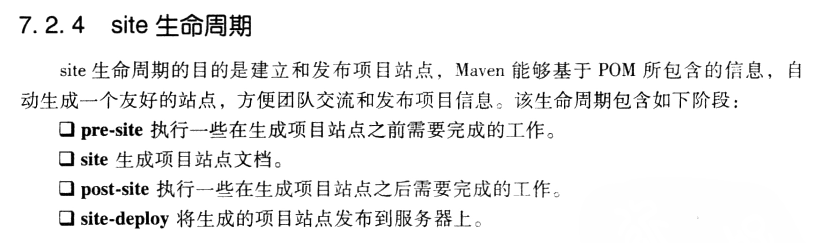
**<!--${nexus.content}是配置了一个properties变量-->**

**</snapshotRepository>**

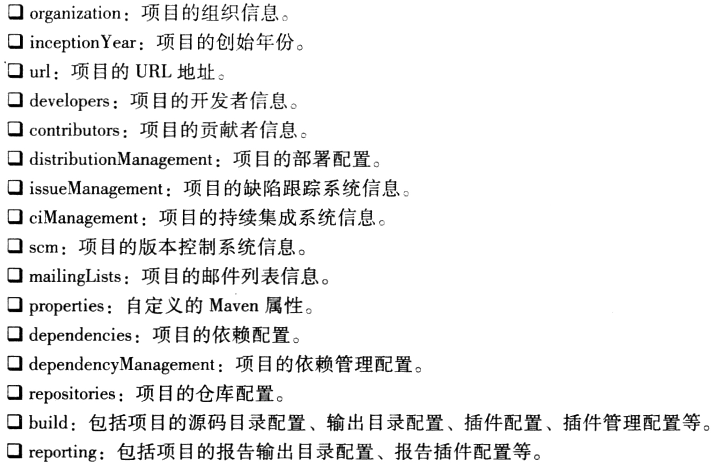
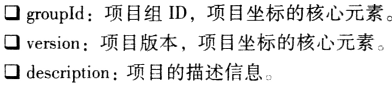
**</distributionManagement>**

## 16.5：生命周期\插件\聚合\继承





**可继承的pom元素**



## 16.6：灵活构建

## 16.7：创建nexus私服

去官网下载Linux或windows的包

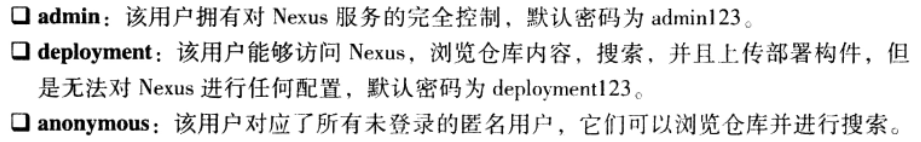
linux下：直接启动bin/nexus start

若报错，则需要去etc/profile中新增对应环境变量

使用 netstat -lnp查看端口开放情况。默认是8081

用ip:8081/nexus访问试试看！

点击又上角Log In用admin登录，默认密码是admin123



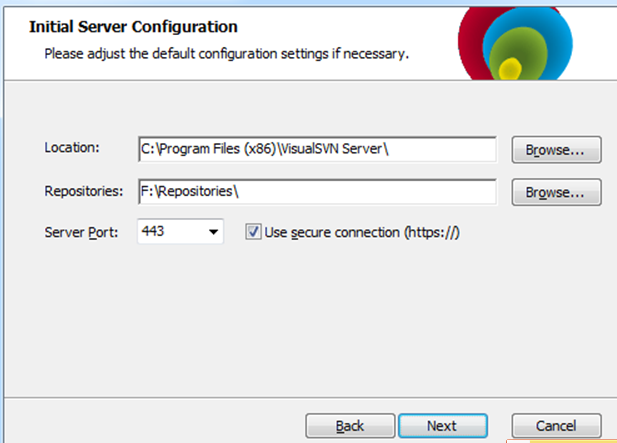
# 17：svn

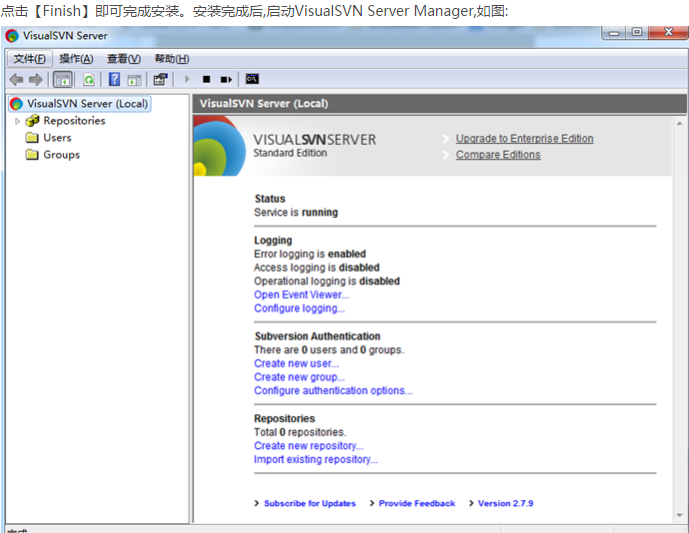
## 17.1：下载与安装

指导网址：<http://blog.sina.com.cn/s/blog_9ab140ab0102wzpb.html>

首先来下载和搭建SVN服务器,下载地址如下: http://subversion.apache.org/packages.html，进入网址后，滚动到浏览器最底部看到如下截图：



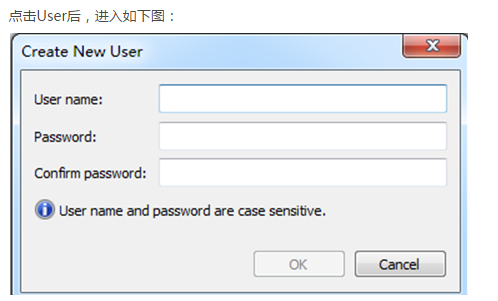


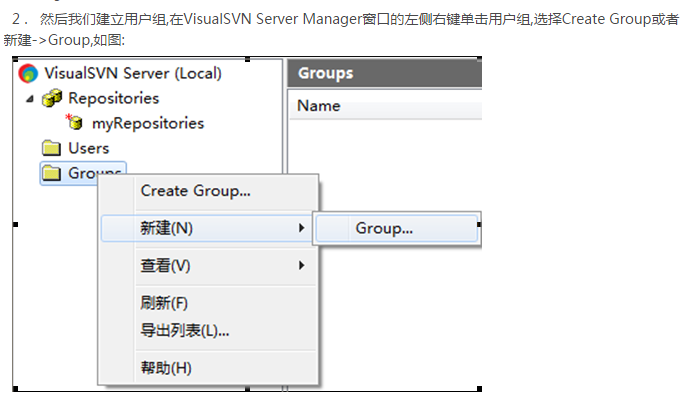


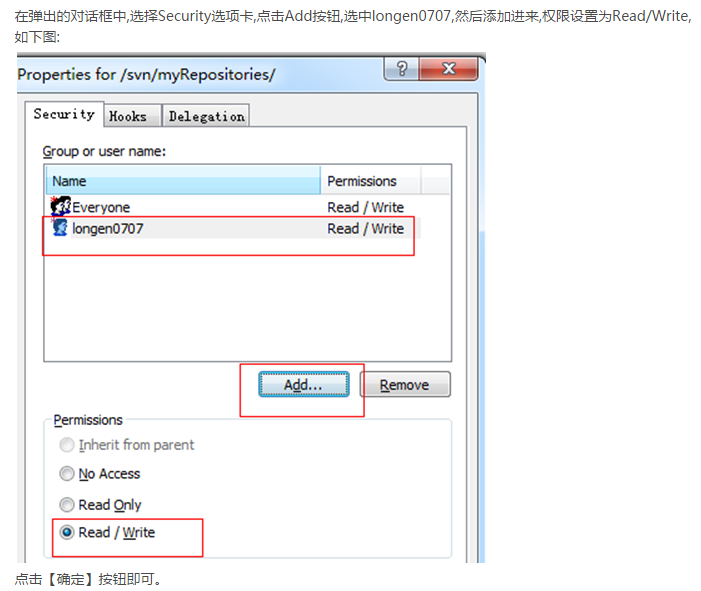
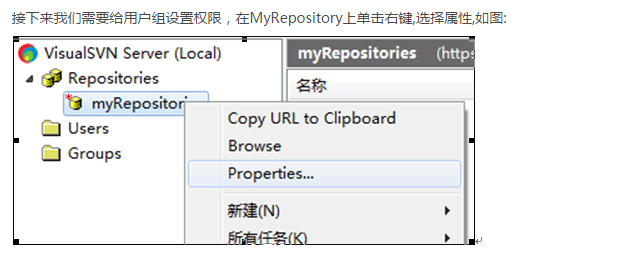


## 17.2：账号与权限设定



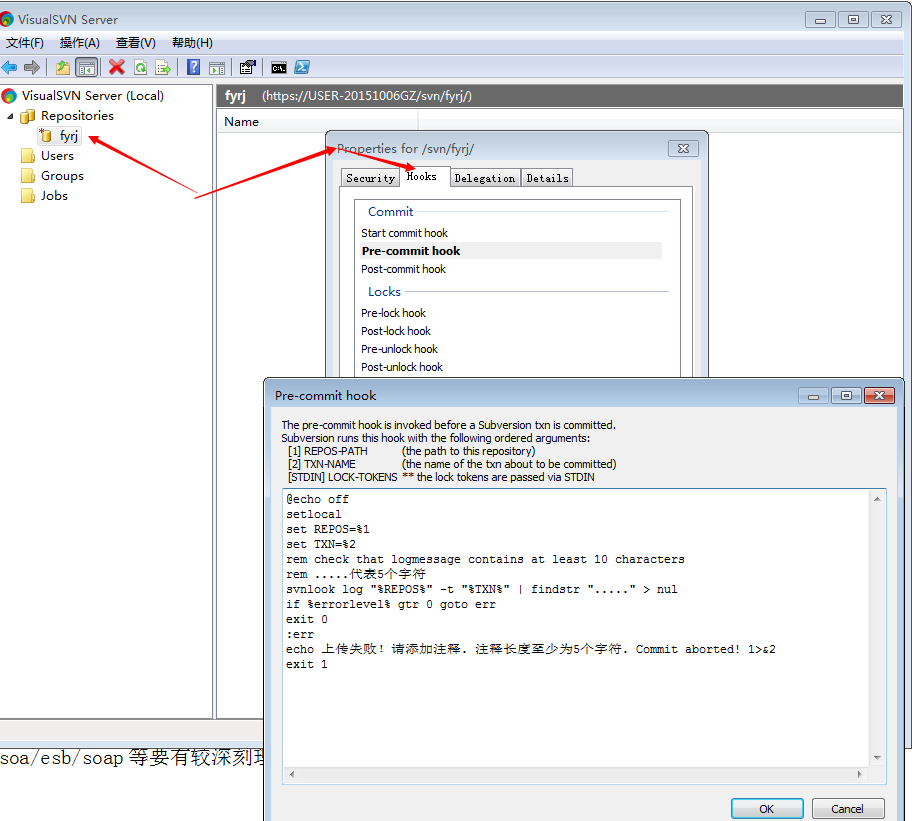






## 17.3：特殊设定

设置必须填写注释才允许提交代码



脚本为====>>>  
@echo off

setlocal

set REPOS=%1

set TXN=%2

rem check that logmessage contains at least 10 characters

rem .....代表5个字符

svnlook log "%REPOS%" -t "%TXN%" | findstr "....." > nul

if %errorlevel% gtr 0 goto err

exit 0

:err

echo 上传失败！请添加注释. 注释长度至少为5个字符. Commit aborted! 1>&2

exit 1

# 18：需要解决的问题记录

1：同步，异步，阻塞，非阻塞等具体概念的理解

2：liunx内核参数和对nginx 的优化

3：squid缓存服务器

4、熟悉spring/spring mvc/mybatis/redis/ dubbo相关技术，对webservice/restful/soa/esb/soap等要有较深刻理解与认识

# 19：网址收录

<http://ifeve.com/read-write-locks/> 并发读写锁详解

<http://doc.redisfans.com/> redis命令大全

<http://www.yiibai.com/redis/redis_commands.html> redis各种技巧与lua等等介绍

<http://www.jfox.info/25-ge-jing-dian-de-spring-mian-shi-wen-da> java面试题

<http://ifeve.com/spring-interview-questions-and-answers/> 69道spring面试题

# 20：自编面试笔试题汇总

## 1：redis分布式锁与zookeeper分布式对比

好的，首先我们来谈一下原理，redis实现采用的是一个单进程单线程的模式，它将多个并发访问进行了队列串行化，所以它对同一个数据的访问是可以保证其原子性的，其次redis提供了一些命令操作，setNx，getSet等命令能够让我们非常方便的实现分布式锁。那么zookeeper就像是我们的linux的文件系统，实际上它的设计就是一个精简的文件系统，它的数据结构原型是一颗znode树，那么它有提供一些文件操作命令，包括增加目录节点删除目录节点等等，且在zookeeper集群中，它能够保证并发情况下都能拿到正确的值。那么通过使用这些命令，我们则可以实现分布式锁。

实现redis分布式锁思路：

我们可以用一个循环来不断的尝试获取锁，当然，我们需要设定一个超时时间和一个等待时间，每隔一段等待时间就去尝试获取锁，如果过了此超时时间还没有获取到锁，则获取锁失败。怎么获取锁呢？我们可以借用redis的setNx命令，如果某个锁key不存在，那么则成功获取锁，且写入一个过期时间做为此锁的值。如果setNx返回false，那么我们则拿到此key的过期时间值，和当前时间对比判断此锁是否已过期，如果过期，则使用getSet命令将新的过期时间写入；成功获取到锁。至于释放锁则就是将lock key 移除则可。

实现zookeeper分布式锁思路：

它的每个锁都需要一个路径来指定（如：/zook/lock1）

1:根据指定的路径查找zookeeper集群中这个路径是否存在

2：如果这个目录存在，则根据查询者的一些特征来判断锁是否为查询者的，如果不是，则获取锁失败，返回null，反之则把该锁返回给查询者。

3：如果目录不存在，则说明没有锁，则创建一个目录节点，且将查询者的特征信息写入该目录节点的数据中，然后返回该锁。

两者的对比：

redis锁需要使用者自己不断的去轮询尝试获取锁，每次轮询去获取锁的时候都有可能拿不到锁，性能较差，如果锁失效，存在多线程去争抢锁；

zookeeper是首先创建加锁标识文件，如果需要等待锁，则添加监听后等待通知或者超时，当锁被释放后，按照节点顺序，依次通知使用者。

## 2：java创建对象的几种方式是什么？

1>直接使用new调用构造器进行创建

2>使用Class的newInstance()调用无参构造器进行创建  
3>使用序列号和反序列号进行创建

4>使用clone进行创建

## 3：字符串的陷阱

String s = "hello.java";

String s2 = "hello."+"java";

System.out.println(s==s2); 1===>>> true

String s3 = "hello.";

String s4 = "java";

String s5 = s3+s4;

System.out.println(s==s5); 2===>>>false

final String s6 = "hello.";

final String s7 = "java";

String s8 = s6+s7;

System.out.println(s==s8); 3===>>>true

String s9 = “hello”;

String s10 = new String(“hello”);

System.out.println(s9==s10); 4===>>>false

final String s11 = “hello.”;  
 String s12 = getStr(); //假设返回值为java

String s13 = s11+s12;

System.out.println(s==s13); 5===>>>false

首先分析一下内存图：

直接使用=号创建： 在栈中开辟一块空间，命名s，因为常量池中只会允许存在一份相同的字符串常量，所以常量池中如果存在”hello.java”，则不再创建，直接将栈空间存放的值指向常量池中”hello.java”的内存地址，否则先往常量池中存放”hello.java”字符串内容再指向。

使用new创建： 先在栈中开辟一块空间，命名s，在编译时，常量池中会被放入”hello.java”字符串，此时栈空间未指向任何地方，在运行new String(“hello.java”) 的new时，会将常量池中的”hello.java”字符串复制一份放入堆中，且此时栈空间的值指向该堆内存地址。

问：结果1为什么是true？因为直接使用=号，编译器在编译的时候就确定了s2的值，实际上String s2=”hello.”+”java”在编译器看来就是String s2 = “hello.java”；根据内存图我们知道，s和s2都指向常量池中的同一个字符串的内存地址，所以为true。

问：结果2为什么是false？

问：结果3为什么是true？加了final修饰，对于final修饰的变量，它的值在编译器期间就能够被确定下来，所以它等同于String s8 = “hello.”+”java” 等同于 “hello.java”;

问：结果4为什么是false？不言而喻，一个指向对堆中内存地址，一个指向常量池中字符串的内存地址，肯定是false了！

问：结果5为什么是false？s12的值在编译期间无法确定下来，只有运行时调用方法后才能知道。所以尽管它用了final修饰，但没用。

问：String s = new String (“a”+”b”);总共创建了几个对象？

答：总共创建了两个，由于”a”+”b”会被直接编译成”ab”，编译后会直接将”ab”放入常量池中，所以这里创建了一个对象，运行时会将常量池中的值拷贝一份放入堆中，这里又是一个对象，所以总共创建了两个！

问：使用stringBuffer和stringBuilder的concat和String的直接+号累加有什么区别？

答：string的本质是一个字符数组,查看源码发现，该char[]使用了final修饰，由此我们知道string是不可变得，如果使用+号，则实际上是在不断的重新生成字符串对象。而stringBuffer和stringBuilder内部使用的是可变char数组，所以它只会生成一个对象，两者区别是 buffer是线程安全的，而builder是不安全的

## 4：int和Integer的陷阱，自动装箱和拆箱

思考这样一个题目：

Integer i01 = 59;

int i02 = 59;

Integer i03 = Integer.valueOf(59);

Integer i04= new Integer(59);

System.out.println(i01==i02); //===>>>true; //由于自动拆箱机制，实际上是两个值为59的int变量在做对比。

System.out.println(i01==i03); //===>>>true; //由于缓存机制，两者其实都是引用的同一个缓存对象；

System.out.println(i03==i04); //===>>>false; //由于用了new，所以是两个不同对象在比较

System.out.println(i02==i04); //===>>>true; //由于自动拆箱机制，实际上是两个值为59的int变量在做对比。

其实某些代码被编译器处理后会变成：

Integer i01 = 59; ===>>> Integer i01 = Integer.valueOf(59);

System.out.println( i01.intValue()==i02 );

自动装箱：我们直接声明一个Integer对象，用int值为其赋值时，会有一个自动装箱的过程，也就是valueOf方法，为了优化性能，该方法又用了缓存的机制，当值为+-128时，这段代码Integer i = 1;返回的实际上是cache对象；

自动拆箱：当我们拿一个Integer对象去与一个int变量做比对时，会自动调用intValue()方法，将该对象转换为int基本类型后再与后者做比较，这是自动拆箱过程

由此：我们可以解释上面代码运行现象！

## 5：概括一下java虚拟机的类加载机制

首先我们说什么是类加载机制？类加载机制就是说jvm把描述一个类的数据从class文件中加载到内存后，对其进行验证，转换解析初始化等等操作后，最终形成可以被jvm直接使用的java类型的这么一个过程！

那么它具体包括那些步骤呢？我认为它包括以下步骤：加载，验证，准备，解析，初始化，使用，卸载等7个步骤，其中验证，准备，解析又常被称之为 ”链接” 阶段。好，接下来具体说下这些步骤我们的虚拟机到底做了些什么!

加载: 此阶段，jvm需要做2个操作。

1>通过一个类的全限定名获取这个类的二进制流，注意，该流它可以从jar包，war包，动态代理生成的类等等之中获取。

2>在java堆内存中生成一个代表这个类的java.lang.Class对象。

验证：由于我们的class文件不一定是有编译器进行编译得来的，它可以由各种工具直接生成，所以jvm必须对其进行校验用来保证自身的运行安全。那么它只要会进行3种验证。

1>文件格式的验证；验证字节流是否符合class的规范，语法等等。

2>元数据的验证；对字节流描述的信息，也就是类的元数据信息进行语义分析，用来保证它所描述的信息符合java语言的规范，比如是否继承了不能继承的父类等等。

3>符号引用验证；对类自身以外的信息进行匹配性验证，比如，引用的外部包是否存在，引用的外部类的某个字段或者方法是否存在等等。

准备：该阶段会有两个操作，将类中的静态变量，常量等数据放入方法区(也称之为永久代)，对变量赋初始值，比如将int类型赋值为0；假设有 static int tmp = 100 ; 那么在准备阶段后，tmp将被赋值为0且放入方法区中！

解析：将常量池中的符号引用替换为直接引用；

初始化：这里参考问题6,7；准备阶段只是赋初始值，此阶段才会真正赋值!注意；对于静态字段，只有用到的这个类才会被初始化，假设A类有变量a，B继承A，在main中打印B.a；则只有A会被初始化，因为B哪怕继承了A，但它不满足4大初始化时机的任意一点；对于常量更不同，假设A中有常量final int a = 100 ; 在Test类中的main中打印A.a，则A也不会进行初始化，因为在编译阶段，常量a会被放入Test类的常量池中；对常量A.a的引用实际上转换成为了Test类对自身常量池的引用；

## 6：java类初始化顺序

我们只需要记住以下几点

1>先加载父类的静态区域块，再加载在子类的静态区域块；

2>如果我们用new 子类() ; 则会开始初始化父类以及子类；

3>先按顺序初始化父类的成员变量，再调用构造函数；

4>按顺序初始化子类的成员变量，再调用构造函数；

## 7：类的初始化时机

有且只有四种情况会触发jvm对类进行初始化，注意，对类进行初始化和调用构造器是两回事。

1>使用new关键字创建对象，或者对static变量赋值，或者获取static变量的值，或者调用static方法；

2>使用反射调用类的方法；

3>如果初始化子类时，父类还没有被初始化，则必须对父类进行初始化；

4>jvm启动时，需要指定一个主类(main方法所在的类)，jvm会首先初始化该类；

## 8：说说java的内存模型

说到java的内存，首先，我们先要了解什么是java虚拟机，因为内存是位于虚拟机之中的。

那么jvm我认为它就是一个虚拟的微型计算机，它有自己独立的运算器，寄存器，内存等等东西。jvm它向我们屏蔽了操作系统底层的差异，使得我们只需要让java程序提供给虚拟机能够识别执行的字节码，那么java程序就能在各种不同平台上运行。

那么，jvm它主要有4个大块。

类加载器，它负责载入我们的二进制流，导入我们的class文件，为我们的类变量分配内存初始化变量等等操作。然后是GC垃圾收集器，它负责回收我们的堆内存。再然后是执行引擎，它负责执行我们的类中的方法中的指令代码。最后是内存区域。

好，我们说到内存区域，那它到底有些什么东西呢？我认为，java内存主要分为5大区域

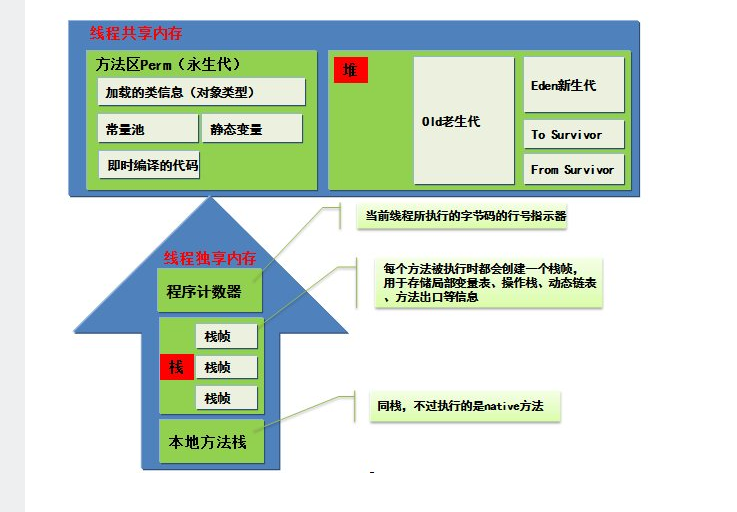
1：程序计数器：java是多线程的，每个线程的开启都会为它分配一个程序计数器内存区域，这个区域就是用来指明当前线程执行到了那一步，如果执行一个方法，那么PC寄存器将指向这个方法指令。

2：本地方法栈：用来存放native方法运行的一些内存数据。

3：栈:每个线程都会开启一个栈空间，每一个方法的执行都会在此栈空间上压入一帧，这被称之为压榨，方法运行完毕，则进行出栈。帧中存放着当前方法的局部变量，返回值，等等内容。

4：堆：存放着所有的对象。

5：方法区：当一个类被jvm加载时，静态变量，类信息，常量等数据都会被载入方法区内存中。方法区中有一块特殊区域，那就是常说的常量池。方法区在垃圾回收算法 分类回收算法 中也被称之为永久代。



## 9：finally与return执行时机的一道题

## 10：简要描述一下spring以及它有些什么模块

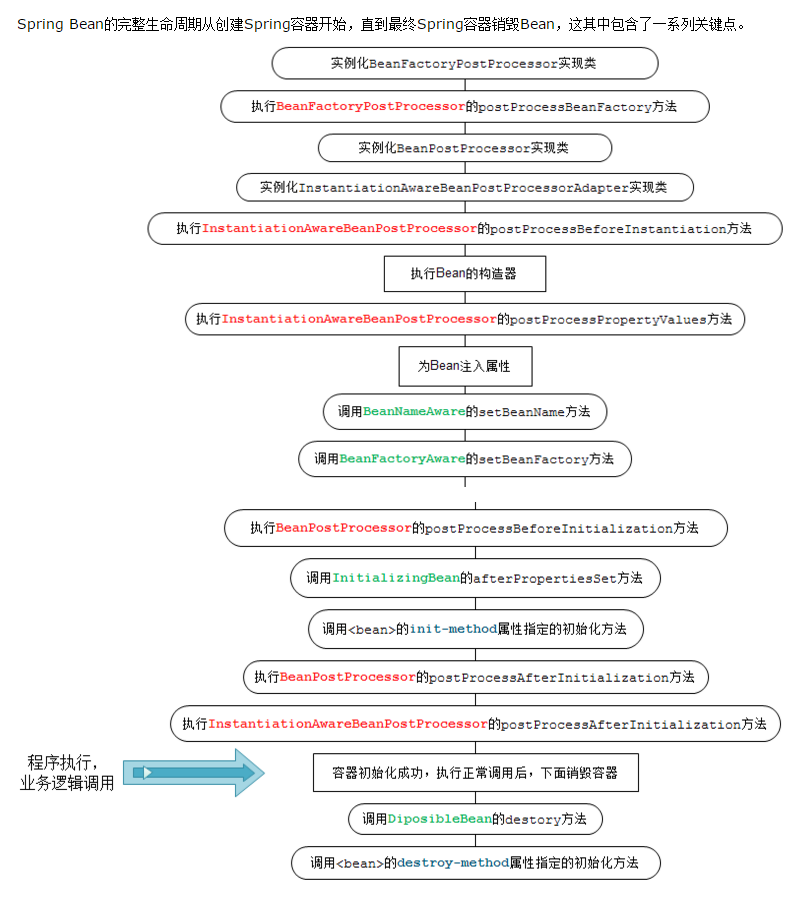
spring是专门为简化java企业级应用开发而打造的框架，它通过将POJO抽象成bean，并提供对bean的管理来降低我们的开发工作；那么在spring中，主要有Data access，web , core container , Test 以及util 等模块，其中又可以细分为 jdbc , orm , websocket,servlet , bean , core , context , aop 等等模块；

## 11：ioc是什么?有什么用?有哪几种注入方式?

ioc指的是控制反转，它倡导面向接口编程，遵循面向对象的依赖倒置原则；在以前，我们需要自己手动去创建所需要的对象，而IOC则帮助我们将我们需要的对象自行创建好且帮我们组建好；这样就大量减少了我们的代码量，且降低了耦合度。IOC也支持懒汉式注入和饿汉式注入，这样让我们的代码更加的灵活。一般来说，IOC有构造器注入和setter方法注入和接口注入三种方式。IOC核心功能位于spring-context中，它通过beanFactory以及它的各种子接口和实现类来实现对POJO类的bean转换，也就是实现对我们java类的实例化，定位，对应用程序中的对象进行配置，对对象间的依赖关系进行维护等等。

## 12：aop是什么?有什么用?实现原理？

## 13：简要说说spring的bean的生命周期



## 14：说说一个bean的作用域有哪些？

singleton:在整个ioc容器中只存在一个实例；

protorype:每次请求多会获得新的实例；

以下三个是在spring 2.0后专门针对web环境而设计新增的，它们只在applicationContext上下文中有效！

session:request:每个http session会话对应一个实例；

request:每次http请求生成一个新实例；

global session :相当于servlet的application作用域；

## 15：BeanFactory和applicationContext之间有什么关系呢？

beanFactory是IOC容器实现的基础接口，它的职责是实例化对象，定位，配置对象数据，维护对象关系等，而applicationContext是一个抽象类，它是从BeanFactory继承而来，在原有的功能基础上，它更是对我们企业级应用的功能开发提供了扩展，包括资源国际化，事件传递机制及各种不同应用层的context实现，比如webApplicationContext等等。

## 16：spring的事务传播和隔离级别以及数据库锁机制！

spring声明式事务开启需要我们配置<tx:annotation-driven>

我们在类或者public方法上使用@transaction来告知spring我们需要事务的参与，transaction可以用在接口上，这样它的实现类都将继承事务配置；这里有一点我们要注意下，spring必须要基于接口的代理模式才能对接口注解起作用，基于类的代理是无法起到作用的！

如果方法不是public，事务注解也将不会起到作用！

传播行为有6个：

如果当前有事务，则加入当前事务，否则开启新事物

@transaction(propagation=Propagation.REQUIRED)

容器不为这个方法开启事务

@transaction(propagation=Propagation.NOT\_SUPPORTED)

无论如何都开启新的事务，如果当前有事务则将老事务挂起，新事物执行完毕再执行老事物

@transaction(propagation=Propagation.REQUIRES\_NEW)

如果其它bean调用这个方法，如果其它bean中有事务，则沿用其它bean中的事务，否则不开启事务

@transaction(propagation=Propagation.SUPPORTS)

必须在没有事务的环境下执行，否则抛出异常

@transaction(propagation=Propagation.NEVER)

必须在有事务的环境下执行，否则抛出异常

@transaction(propagation=Propagation.MANDATORY)

隔离级别有4个：

1>读取未提交数据，会出现脏读和幻读

**事务B更新row1数据瞬间，row1加入共享锁，此时事务A允许读取共享锁行数据，所以造成脏读！**

@transaction(ioslation=Isolation.READ\_UNCOMMITED)

2>读取已提交数据，会出现不可重复读和幻读

**事务B更新row1数据瞬间，row1加入排它锁，此时事务A不允许读取排它锁锁定的数据行，所以不会造成脏读**

@transaction(ioslation=Isolation.READ\_COMMITED)

3>可重复读，会出现幻读

**事务A读取row1数据瞬间，row1数据被加入共享锁，此时其它事务只允许读取而不能修改，所以可重复读**

@transaction(ioslation=Isolation.REPEATABLE\_READ)

4>串行化

**最高锁机制，给整个表加入排它锁，其它事务无法读取和更新！**

@transaction(ioslation=Isolation.SERIALIZABLE)

我再解释下脏读，不可重复读，可重复读，幻读，共享锁，排他锁的意思；

脏读：事务A读到了事务B更新但未提交事务的数据，假设B执行了update语句，这时A就会读到，A读完后B遇到了问题进行事务回滚，这是A读到的数据就是脏的！

不可重复读：同一个事物中，同一条sql每次读到的数据不一致，也就是说，在事务A第一次select后，此时事务B update一条数据，A再执行sql，这时读到的数据内容就变了！

可重复读：同一条sql每次读到的数据是一样的！也就是说后续的读取跟第一次的读取结果是一致的，当然，如果事务B新增了数据，还是会不同！

幻读：事务A读取了15条数据，此时事务B写入一条，事务A再次读取将读取16条数据，这就是幻读！

共享锁：其它事务只能读取加锁行数据，无法修改，想要修改必须等事务结束

排它锁：其它事务不能修改也不能读取加锁行数据

## 17：对比一下hibernate和mybatis的区别

首先，从功能上来讲，hibernate是一个重量级的完整的ORM映射框架，它的着重点是建立POJO与数据表之间的关系，通过它强大的映射功能，让我们可以以面向对象的方式来操作数据行，它让我们不再关注数据行的变化，而是转为关注实体类的变化。只要实体类改变，那么它将自动生成对应的sql让数据行也随之改变。且它的HQL向我们屏蔽了数据库底层的差异，让我们扩展切换数据库毫无压力。

而mybatis的着重点则是POJO与sql之间的关系，它并不能让我们面向对象的方式来操作数据行。

其次，就学习成本而言，hibernate显然学习成本更高，想要精通掌握更是困难，而mybatis则是随学随用，学习曲线非常低。

然后，就开发社区而言，hibernate社区非常活跃，mybatis则比较冷清。

然后，就代码量而言，hibernate只要建立好了关系，sql则完全自动生成，我们可以花更多的精力去关注业务流程实现，而mybatis则需要我们手动编写大量sql和resultMap，当然，通过我们好的架构，这些语句都可以自动生成。相对来说，我个人更喜欢mybatis，可能是hibernate不太精通，用起来总感觉不太灵活。而mybatis则由于手动写sql，所以更加灵活。而代价则是当我们万一需要切换数据库的时候，mybatis会有巨大的难度。还有一点，hibernate自动生成的sql会查询出所有字段，这点对性能有所影响。

再然后，就性能调优方面而言，hibernate有几点我们需要关注，

1>尽量采用延迟加载策略

2>使用批量抓取，制订批量抓取数目

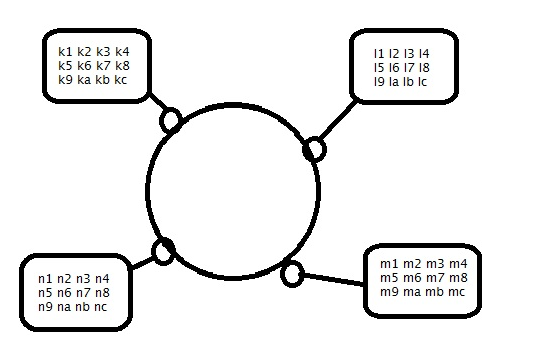
3>制订合理的会话管理方式

4>使用合理的缓存策略

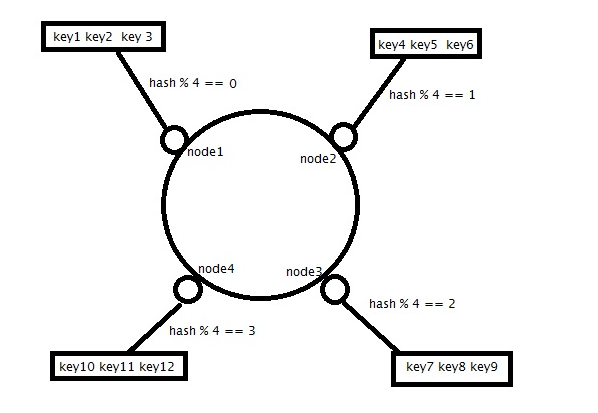
5建立良好的ORM映射关系

## 18：hash算法与一致性hash理解

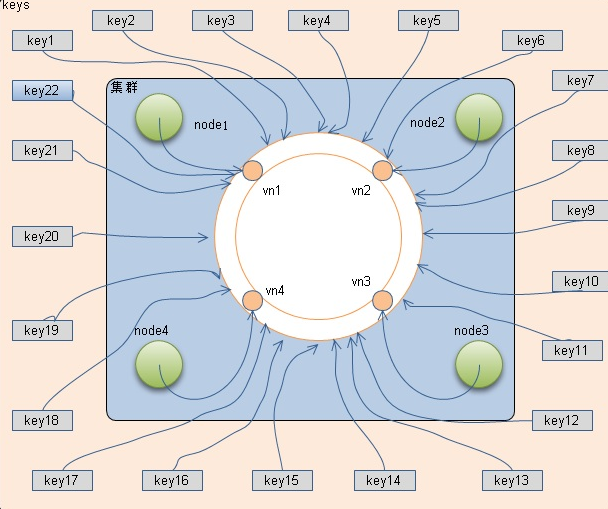
如图，假设在集群中，有4台机器，我们需要将数据缓存在机器上，此时，每个机器存放固定的key，而我们客户端也需要知道每个key应该存放到哪台机器，也就是说，我们需要维护好key与机器的关系，那么如果有一台机器挂掉，或者新增了一台机器，此时怎么办？



好，我们开始用hash算法，简单的取模运算操作，我们拿到key的hashCode ， 对机器数取模，假设有4台机器，那么结果无非是0,1,2,3 ；此时我们不需要去维护key与机器的关系，也不关心哪台机器存放了哪些key，一切交由算法决定，但是呢，如果挂掉机器1，那么想想，新的获取或者存放数据的操作，都将由原来的 hashCode%4变成hashCode%3，原先去机器1上取数据都将变成在在机器2上取，依次类推，那么我们需要将整个机器的数据都迁移一遍，工程量是不是有点大呢！？

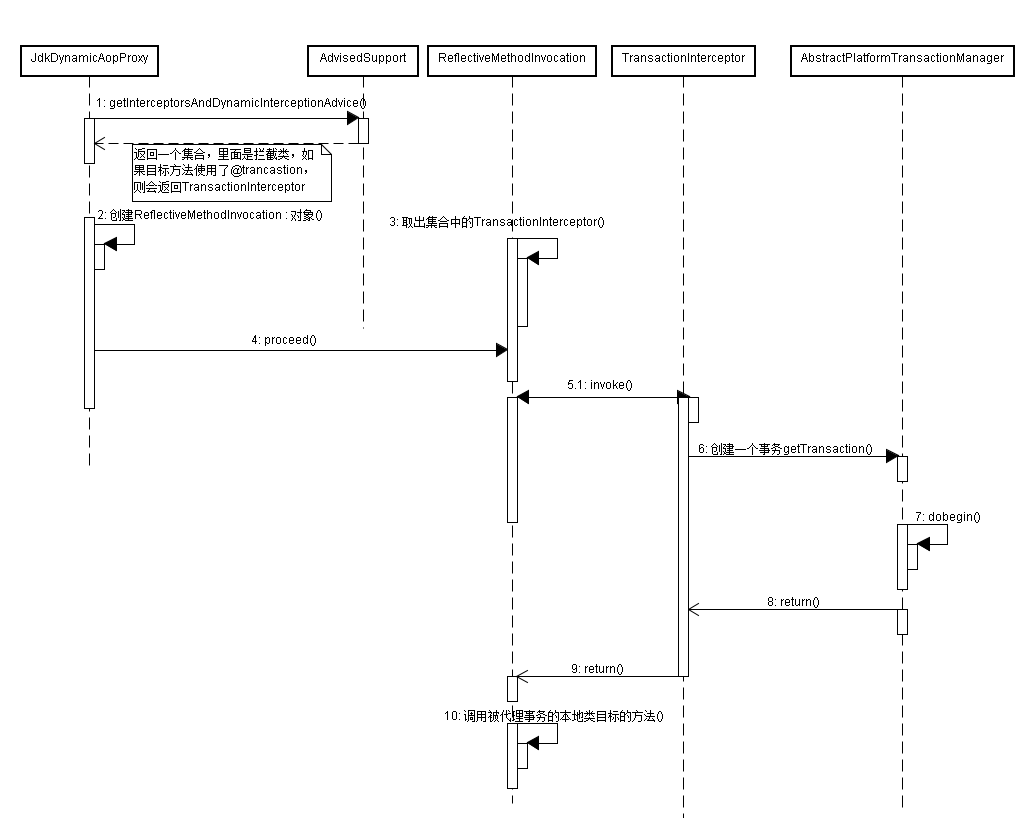


于是，我们开始沿用一致性hash算！它可以更加均匀的分散我们的请求或者数据存储，将我们的数据迁移工作量降到最低。什么是一致性hash呢，如图：



我们构建一个大圆，然后我们构建2的32次方个虚拟节点，真实机器节点也分布在这个园上，此时一个key决定存放在哪里，算法就会变成 key的hashCode%2的32次方，此时将会得到一个虚拟节点，那么我们接下来顺时针往下寻找，一旦找到一个真实节点，就将数据存放在此节点上！如果我们挂掉一个节点，那么就只要将这个节点到上一个节点之间的数据迁移到下一个节点上就可以了!

## 19：谈谈对spring事务的理解



## 20：集群，分布式的概念

**集群是个物理形态，分布式是个工作方式。**

* 分布式：一个业务分拆多个子业务，部署在不同的服务器上
* 集群：同一个业务，部署在多个服务器上

1：**分布式是指将不同的业务分布在不同的地方。而集群指的是将几台服务器集中在一起，实现同一业务。**

分布式中的每一个节点，都可以做集群。而集群并不一定就是分布式的。

举例：就比如新浪网，访问的人多了，他可以做一个群集，前面放一个响应服务器，后面几台服务器完成同一业务，如果有业务访问的时候，响应服务器看哪台服务器的负载不是很重，就将给哪一台去完成。

而分布式，从窄意上理解，也跟集群差不多，但是它的组织比较松散，不像集群，有一个组织性，**一台服务器垮了，其它的服务器可以顶上来。**

**分布式的每一个节点，都完成不同的业务，一个节点垮了，那这个业务就不可访问了。**

2：简单说，**分布式是以缩短单个任务的执行时间来提升效率的，而集群则是通过提高单位时间内执行的任务数来提升效率。**

例如：如果一个任务由 10 个子任务组成，每个子任务单独执行需 1 小时，则在一台服务器上执行该任务需 10 小时。

采用分布式方案，提供 10 台服务器，每台服务器只负责处理一个子任务，不考虑子任务间的依赖关系，执行完这个任务只需一个小时。(这种工作模式的一个典型代表就是 Hadoop 的 Map/Reduce 分布式计算模型）

而采用集群方案，同样提供 10 台服务器，每台服务器都能独立处理这个任务。假设有 10 个任务同时到达，10 个服务器将同时工作，1 小时后，10 个任务同时完成，这样，整身来看，还是 1 小时内完成一个任务！

好的设计应该是分布式和集群的结合，先分布式再集群，具体实现就是业务拆分成很多子业务，然后针对每个子业务进行集群部署，这样每个子业务如果出了问题，整个系统完全不会受影响。

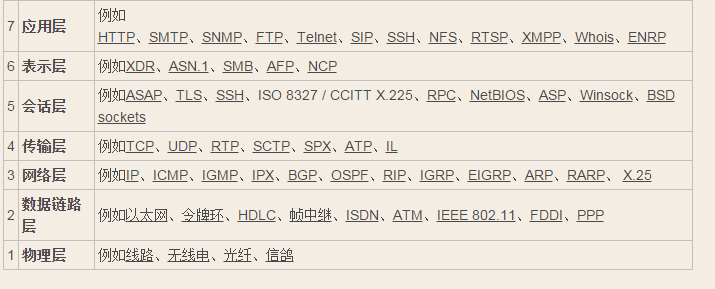
以上摘自：[分布式与集群的区别是什么？](https://www.zhihu.com/question/20004877) 分布式概念参考：[分布式系统的特点以及设计理念](http://www.infoq.com/cn/articles/features-and-design-concept-of-distributed-system)

另外，还有一个概念和分布式比较相似，那就是**微服务。**

**微服务是一种架构风格，一个大型复杂软件应用由一个或多个微服务组成。系统中的各个微服务可被独立部署，各个微服务之间是松耦合的。每个微服务仅关注于完成一件任务并很好地完成该任务。在所有情况下，每个任务代表着一个小的业务能力。**

## 21：说说你对TCP、IP、UDP、HTTP、HTTPS的理解

OK，首先回答这个问题前，我想先说说我们著名的七层网络工作模型，自底向上分别是：物理层，数据链路层，网络层，传输层，会话层，表示层，应用层



那么，我们的IP是网络层，TCP和UDP则是传输层，HTTP以及HTTPS则是我们的应用层；

然后，我们分别说说他们的一些特点。IP就是internet protocol,网络传输协议，它就好像是一条高速公路，它允许其他协议在上面奔跑，TCP Transfer controller protocol 传输控制协议与UDP user data page protocol用户数据包协议则像是一辆奔跑在IP协议上的大卡车，他们装载着HTTP协议所描述的数据包，将数据送达给目的地。

那么TCP有UDP有什么区别呢？TCP有著名的三次握手概念，首先，客户端发送数据包给服务端，说hello 在线不，老铁？,接下来服务端返回数据包给客户端，说”老弟，哥在呢！”，然后客户端继续发送数据包给服务端，说”老铁，给我个看了喷鼻血的图片撒！” ，接下来服务端与客户端将正式进行数据传输。那么他们是基于长链接的，除非一方主动断开，否则不会断开连接。而我们的HTTP则不同。

HTTP就是我们常说的超文本传输协议，见名知意，它就是用来描述我们的数据包，那么和它一个级别的有我们常见的FTP file Transfer protocol 文件传输协议,SMTP simple mail Transfer  protocol 简单邮件传输协议等等。HTTP它是短链接的，它一般运用在浏览器、服务器之间，浏览器发起一次HTTP请求，服务器响应后，则断开连接，它是无状态的。

好，HTTPS则是对我们HTTP的扩展，它基于网景公司的SSL来进行加密传输。它和HTTP的断开是不同的，HTTP是40，HTTPS是443。

## 22：一道经典sql题记录

已知有表：

course(c\_id主键ID，name 科目名称) 课程表

student(s\_id 主键ID ，name 学生姓名) 学生表

score (sc\_id 主键ID，c\_id 课程ID，s\_id 学生ID ，grade 分数) 成绩表

-- 1.计算每个人的总成绩并排名(要求显示字段：姓名，总成绩)

SELECT stu.name,SUM(s.grade) AS sumScore FROM

score s

LEFT JOIN student stu ON s.s\_id = stu.`s\_id`

GROUP BY s.s\_id ORDER BY sumScore DESC

-- 2.计算每个人单科的最高成绩(要求显示字段:姓名，课程，最高成绩)

SELECT stu.name,c.name,MAX(s.grade) FROM

score s

LEFT JOIN student stu ON s.s\_id = stu.s\_id

LEFT JOIN course c ON c.c\_id = s.c\_id

GROUP BY s.s\_id

-- 3.计算每个人的平均成绩（要求显示字段: 姓名，平均成绩）

SELECT stu.name,AVG(s.`grade`)

FROM score s LEFT JOIN student stu ON s.s\_id = stu.`s\_id`

LEFT JOIN course c ON c.c\_id = s.c\_id

GROUP BY s.s\_id

-- 4.列出各门课程成绩最好的学生(要求显示字段:姓名,科目，成绩)

SELECT stu.name,c.name,MAX(s.grade)

FROM score s LEFT JOIN student stu ON s.s\_id = stu.`s\_id`

LEFT JOIN course c ON c.c\_id = s.c\_id

GROUP BY s.c\_id

-- 5.列出各门课程成绩最好的两位学生(要求显示字段:姓名,科目，成绩)

-- 6．列出各门课程的平均成绩（要求显示字段：课程，平均成绩）

SELECT c.name,AVG(s.grade)

FROM score s LEFT JOIN student stu ON s.s\_id = stu.`s\_id`

LEFT JOIN course c ON c.c\_id = s.c\_id

GROUP BY s.c\_id

-- 7．列出数学成绩的排名（要求显示字段：姓名，成绩，排名）

SELECT

stu.name,s.grade,

( SELECT COUNT(1) FROM score s1 LEFT JOIN course c1 ON s1.c\_id = c1.c\_id WHERE c1.name = '数学' AND s1.grade > s.grade)+1 AS 'cj'

FROM score s

LEFT JOIN student stu ON s.s\_id = stu.s\_id

LEFT JOIN course c ON c.c\_id = s.c\_id

WHERE c.name = '数学'

ORDER BY cj

-- 8．列出数学成绩在2-3名的学生（要求显示字段:姓名,科目，成绩）

SELECT \* FROM (

SELECT

stu.name AS 'stuName',c.name,s.grade,

(SELECT COUNT(1) FROM score s1 LEFT JOIN course c1 ON s1.c\_id = c1.c\_id WHERE c1.name ='数学' AND s1.grade > s.grade)+1 AS 'pm'

FROM score s LEFT JOIN student stu ON s.s\_id = stu.s\_id

LEFT JOIN course c ON s.c\_id = c.c\_id

WHERE c.name ='数学'

)TMP WHERE TMP.pm BETWEEN 2 AND 3

-- 9．求出李四的数学成绩的排名

SELECT

stu.name,c.name,s.grade,

(SELECT COUNT(1) FROM score s1 LEFT JOIN course c1 ON s1.c\_id = c1.c\_id WHERE c1.name = '数学' AND s1.grade > s.grade )+1 AS '名次'

FROM score s LEFT JOIN student stu ON s.s\_id = stu.`s\_id`

LEFT JOIN course c ON c.c\_id = s.c\_id

WHERE c.name = '数学' AND stu.`name`='李四'

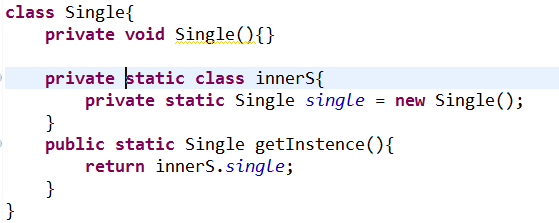
10统计如下：



11：统计如下：



## 23：使用静态内部类实现单列



# 21：单词收录

|  |  |  |
| --- | --- | --- |
| slot | [slɒt] | 位置；狭槽；水沟；硬币投币口 |
| atomic | [ə'tɒmɪk] | 原子的，原子能的；微粒子的 |
| resolver | [riː'zɒlvə] | 解析器 |
| delegate | [ˈdɛlɪˌɡeɪt;] | 委派…为代表 |
| **loader** | **['ləʊdə]** | 装货的人；载入程序；装货设备；装填器 |
| Standard | ['stændəd] | 标准；水准；旗；度量衡标准 |
| parse | [pɑːz] | 解析；从语法上分析 |
| Eagerly | ['i:gəlɪ] | 急切地；渴望地；热心地 |
| Early | ['ɜːlɪ] | 早期的；早熟的；提早；在初期 |
| actually | ['æktjʊəlɪ] | 实际上；事实上 |
| apply | [ə'plaɪ] | **申请；涂，敷；应用** |
| monitor | ['mɒnɪtə] | **监视器；监听器；监控器；显示屏；班长** |
| Handler | ['hændlə] | 处理者；管理者；拳击教练 |
| resolve | [rɪ'zɒlv] | 决定；溶解；使…分解；决心要做… |
| helper | ['helpə(r)] | 助手，帮手 |
| Candidate | ['kændɪdeɪt] | 候选人，候补者；应试者 |
| generic | [dʒɪ'nerɪk] | 类的；一般的；属的；非商标的 |
| necessary | [ˈnɛsɪsərɪ] | 必要的；必需的；必然的 |
| already | [ɔːl'redɪ] | 已经，早已；先前 |
| determine | [dɪ'tɜːmɪn] | 决定，确定；判定，判决；限定 |
| Exposure | [ɪk'spəʊʒə] | 暴露；曝光；揭露；陈列 |
| exposed | [ɪk'spəʊzd] | 暴露的，无掩蔽的 |
| populate | ['pɒpjʊleɪt] | 填充,填入,居住于,构成人口 |
| Enhanced | [ɪn'hɑːnst] | 加强的；增大的 |
| proceed | [prə'siːd] | 开始；继续进行；发生；行进 |
| expose | [ɪk'spəʊz; ek-] | 揭露，揭发；使曝光；显示 |
| Introduction | [,ɪntrə'dʌkʃ(ə)n] | 介绍；引进；采用；入门；传入 |
| calculate | ['kælkjʊleɪt] | 计算；以为；作打算 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# 22：jenkins

第一步：安装

前往http://mirrors.jenkins.io/war/下载 2.0版本

下载好tomcat7.0 ，配置好linux中的JDK, maven

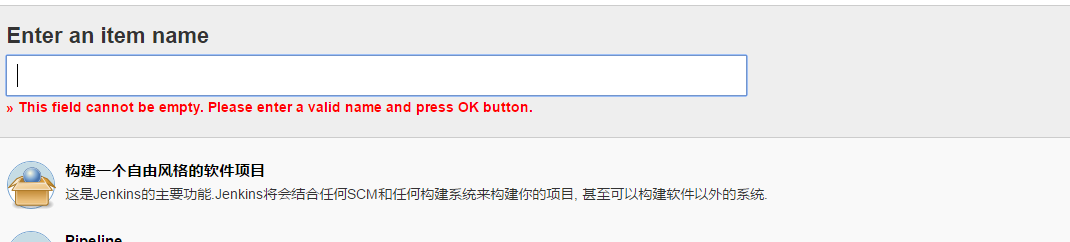
将war防止tomcat下。覆盖ROOT路径，启动，默认端口为8080

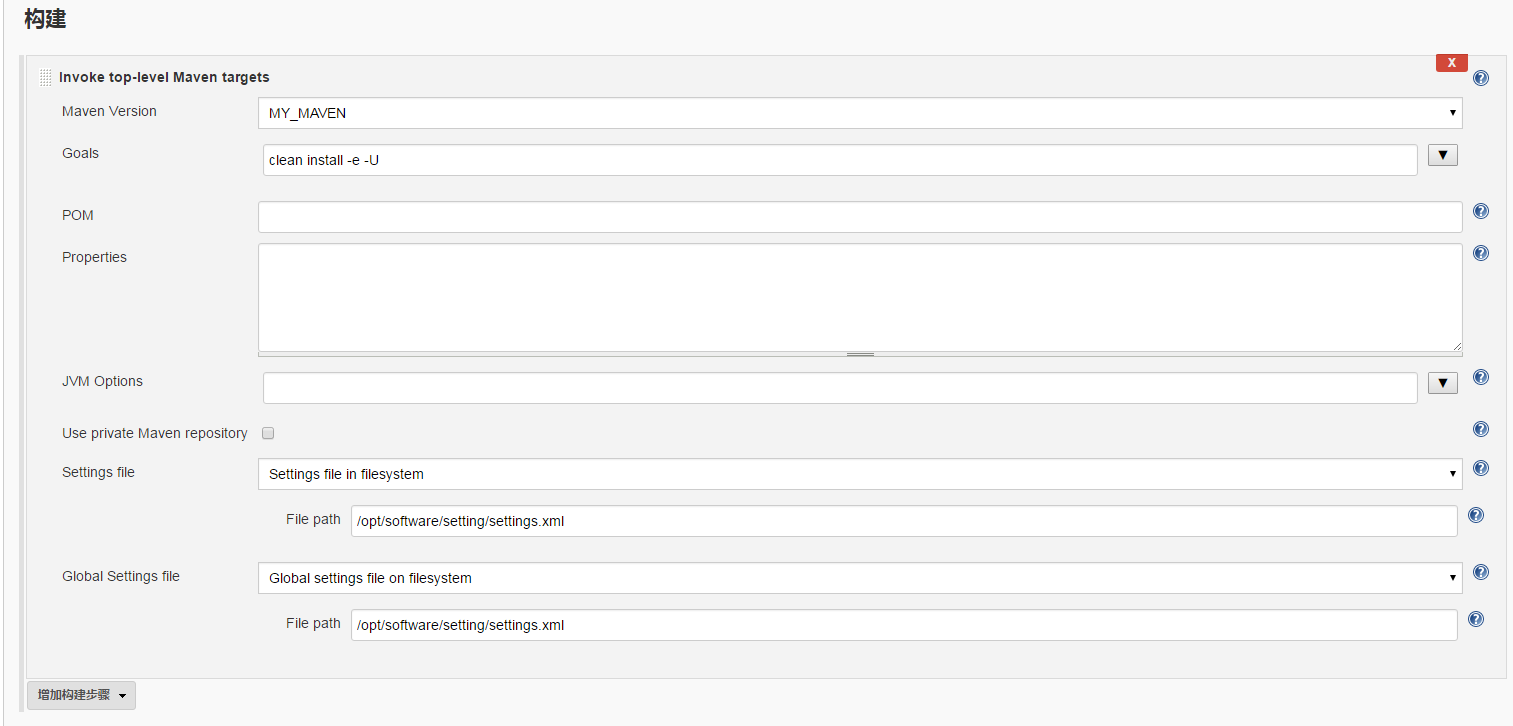
启动完毕后访问xxx:8080 ，根据提示进行初始化。

第二步：配置maven插件 **[系统管理](http://192.168.1.199:8080/manage)**--->>>[Global Tool Configuration](http://192.168.1.199:8080/configureTools)--->>>Maven



第三步：新建maven工程，关键配置在下面两个截图中



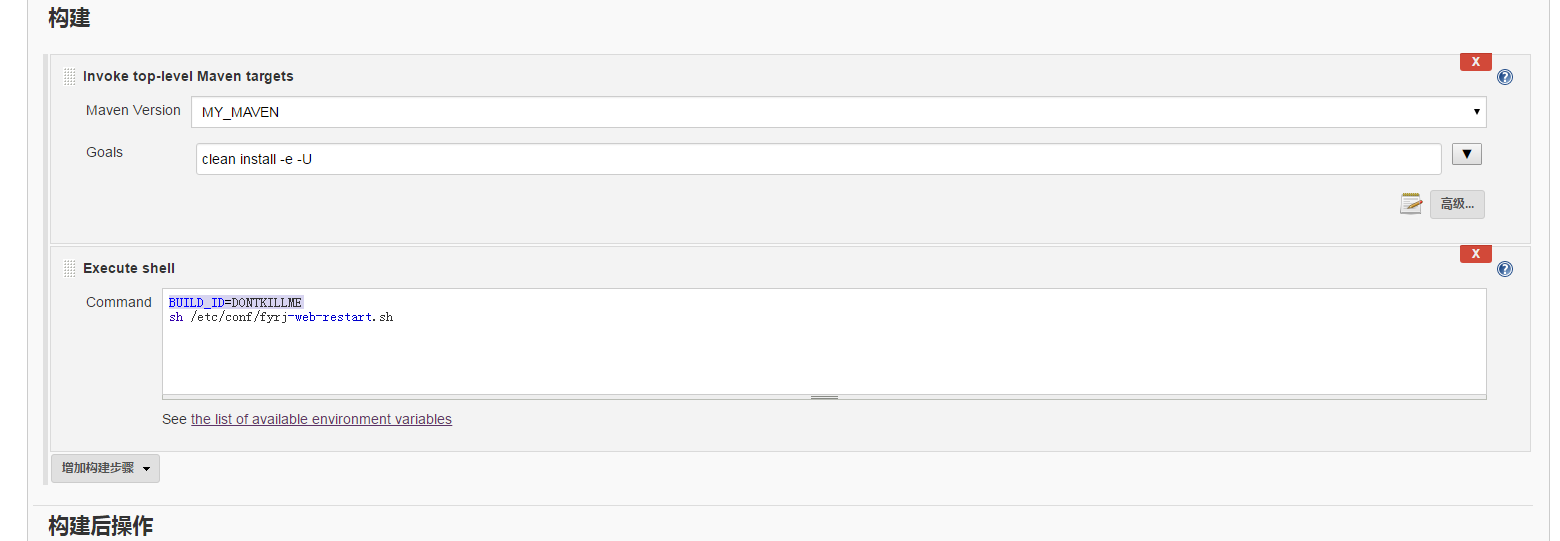


至此，点击构建就可以构建一个工程了，该工程代码将会出现在root家目录的

/root/.jenkins/workspace 下

构建完工程后我们希望可以自动部署，此时则需要执行一些自定义的脚本；

回到工程配置页面，在这里加入Execute shell



**在这里我是重新部署war到tomcat，具体脚本如下：**

**#!/bin/sh**

**WAR\_PATH=/root/.jenkins/workspace/fyrj-web/target/web.war;**

**TOMCAT\_PATH=/opt/app/fyrj-web;**

**PID="";**

**getPID(){**

**echo `ps -ef | grep $1 | grep -v grep | awk '{print $2}'`;**

**}**

**deployTomcat(){**

**echo "start deploy $WAR\_PATH";**

**bash "$TOMCAT\_PATH/bin/shutdown.sh";**

**sleep 5;**

**rm -rf "$TOMCAT\_PATH/work";**

**rm -rf "$TOMCAT\_PATH/webapps/web.war";**

**rm -rf "$TOMCAT\_PATH/webapps/web";**

**echo "delete $TOMCAT\_PATH/work";**

**echo "delete $TOMCAT\_PATH/webapps/web.war";**

**echo "delete $TOMCAT\_PATH/webapps/web";**

**mv "$WAR\_PATH" "$TOMCAT\_PATH/webapps";**

**echo "waiting for cp $WAR\_PATH to $TOMCAT\_PATH ...";**

**sleep 5;**

**PID=`getPID $TOMCAT\_PATH`;**

**if [ "$PID" != "" ]; then**

**echo "kill tomcat $PID";**

**kill -9 $PID;**

**fi**

**bash "$TOMCAT\_PATH/bin/startup.sh";**

**echo "waiting for start...";**

**sleep 5;**

**echo "end deploy $WAR\_PATH";**

**}**

**deployTomcat;**

**OK,完成！**

**解决jenkins打包不能及时更新SVN的问题**

**在使用Jenkins做持续集成时，遇到Jenkins的SVN插件没有更新到最新的代码的情况。**

**例如，在代码提交之后就立即使用Jenkins更新代码，结果刚提交的代码没有被更新到，更新到的代码是旧版本的。**

**查阅网上相关内容，有一种说法为：**

**Jenkins服务器时间与SVN服务器时间不一致，Jenkins的SVN插件是使用时间标签下载，而不是取HEAD，**

**因此如果svn服务器的提交代码时间比Jenkins的当前时间晚，该代码就不会被更新。**

**所解决问题的方法是：**

**只要将Jenkins服务器时间与SVN服务器时间设置成一样的就可以。**

**没错，上面是解决了问题，但Jenkins的SVN插件是与时间戳相关的SVN revision吗？**

**查看某个Jenkins Job的构建日志，在使用SVN插件更新代码时，日志如下：**

**Updating svn://repository\_path at revision '2015-08-06T08:48:12.490 +0800'**

**从上面可以看出来，该次构建相应的revision确实是构建时间戳。**

**那么，可以让Jenkins的SVN插件更新代码时，设置revision为HEAD吗？**

**答案是可以的，在SVN URL加@HEAD后缀即可，Jenkins的SVN插件是支持这个的。**

**在SVN URL加@HEAD后缀后，构建Jenkins Job后日志输出如下：**

**Updating svn://repository\_path@HEAD**

**而且这样确保更新的代码是最新的，不会因为Jenkins服务器与SVN服务器之间的时间差受到影响。**

**注：HEAD是SVN revision关键字，表示版本库中的最新版本。**

**原文：<http://my.oschina.net/donhui/blog/490125>**

# 23：ELK

## 23.1：安装

参考链接：<http://m.blog.csdn.net/article/details?id=51280058>

<http://www.cnblogs.com/qindongliang/p/4897710.html>

**下载：****elasticsearch：**

**wget**

<https://download.elastic.co/elasticsearch/elasticsearch/elasticsearch-1.7.3.tar.gz>

**Logstash**

**wget**

<https://download.elastic.co/logstash/logstash/logstash-1.5.4.tar.gz>

**Kibana**

**wget** <https://download.elastic.co/kibana/kibana/kibana-4.1.2-linux-x64.tar.gz>

**1：首先安装JDK1.8**

**2：然后解压elasticsearch，**

如果机器其它地方需要依赖当前JDK其它版本，

则需要修改./bin/elasticsearch 文件，

加上JAVA\_HOME="/opt/software/ELK/jdk1.8.0\_11";

**后台启动es ：nohup ./bin/elasticsearch &**

访问地址为：ip:9200

可在config/elasticsearch.yml 中修改端口和其它信息

**然后在线安装head和bigdisk插件**   
直接在elasticsearch的根目录下，

分别输入：

安装head 访问地址：<http://10.224.200.27:9200/_plugin/head/>   
bin/plugin --install mobz/elasticsearch-head

安装bigdesk

访问地址：http://10.224.200.27:9200/\_plugin/bigdesk/#nodes  
bin/plugin --install lukas-vlcek/bigdesk

head 是简单索引管理界面，bigdesk是服务器监控界面

**3：安装kibana**

1. 解压到指定目录   
   （2）在根目录执行bin/kibana直接启动 也可以后台启动；  
   （3）访问http://ip:5601/ 配置一个ElasticSearch索引（配置索引前，需要该索引存在，所以，需要我们先用logstash收集日志生成索引）

**4：安装logstash**

在需要收集日志的服务器分别安装logstash

解压后在根目录下建立conf文件夹，里面存放配置文件，比如存放一个my\_conf.conf文件

内容为：

input{

file {

codec => plain{

charset => "GB2312"

}

path => ["/srv/www/detail.log","/srv/www/simple.log"]

type => "fxckh-dzt"

start\_position => "beginning"

}

file {

path => ["/xx/xxx/\*.log"]

type => "fxckh-dzt-beta"

start\_position => "beginning"

}

}

output {

elasticsearch {

codec => plain{

charset => "GB2312"

}

host => "10.224.200.27"

index => "logstash-dzt-%{type}-%{+YYYY.MM.dd}"

document\_type => "%{type}"

cluster => elasticsearch

}

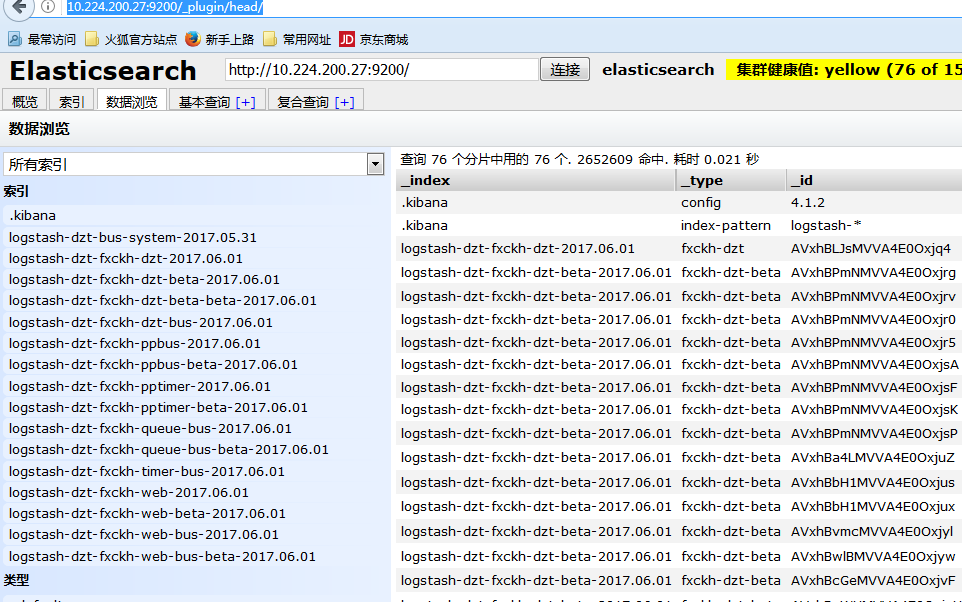
stdout { codec => json\_lines }

}

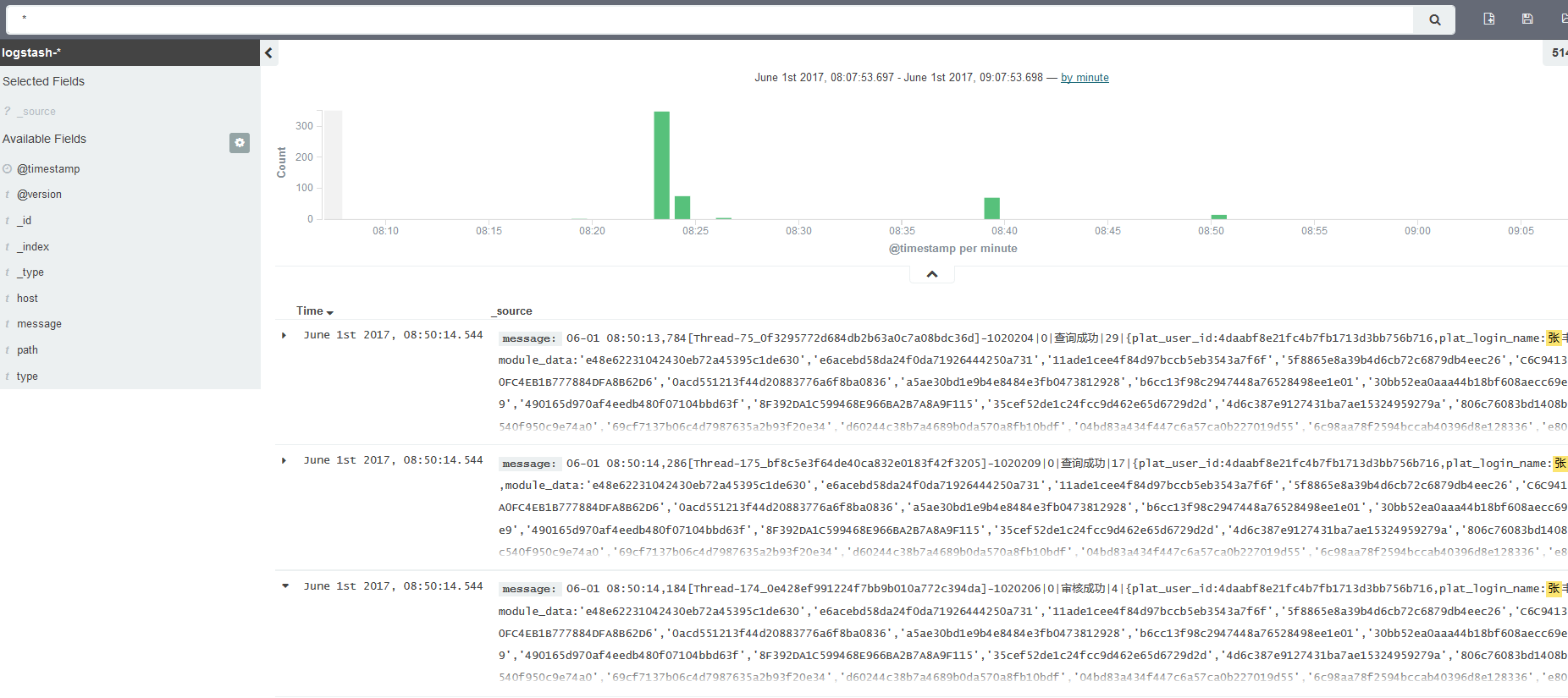
**OK，接下来指定配置文件后台启动logstash：**

nohup ./bin/logstash -f ./conf/my\_conf.conf &

接下来可以访问各页面看看效果了



这里应该有了索引



这里应该有了日志内容

## 23.2：API

删除索引数据：curl -XDELETE 'http://ip:9200/logstash-2013.03.\*'

具体参考：<https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-delete.html>