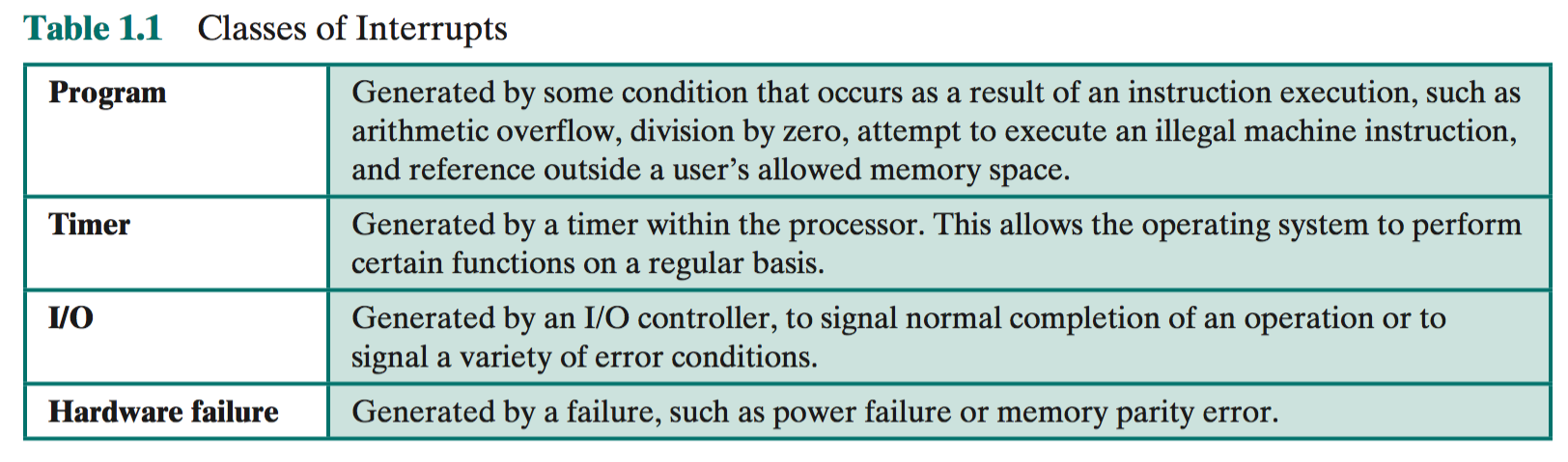
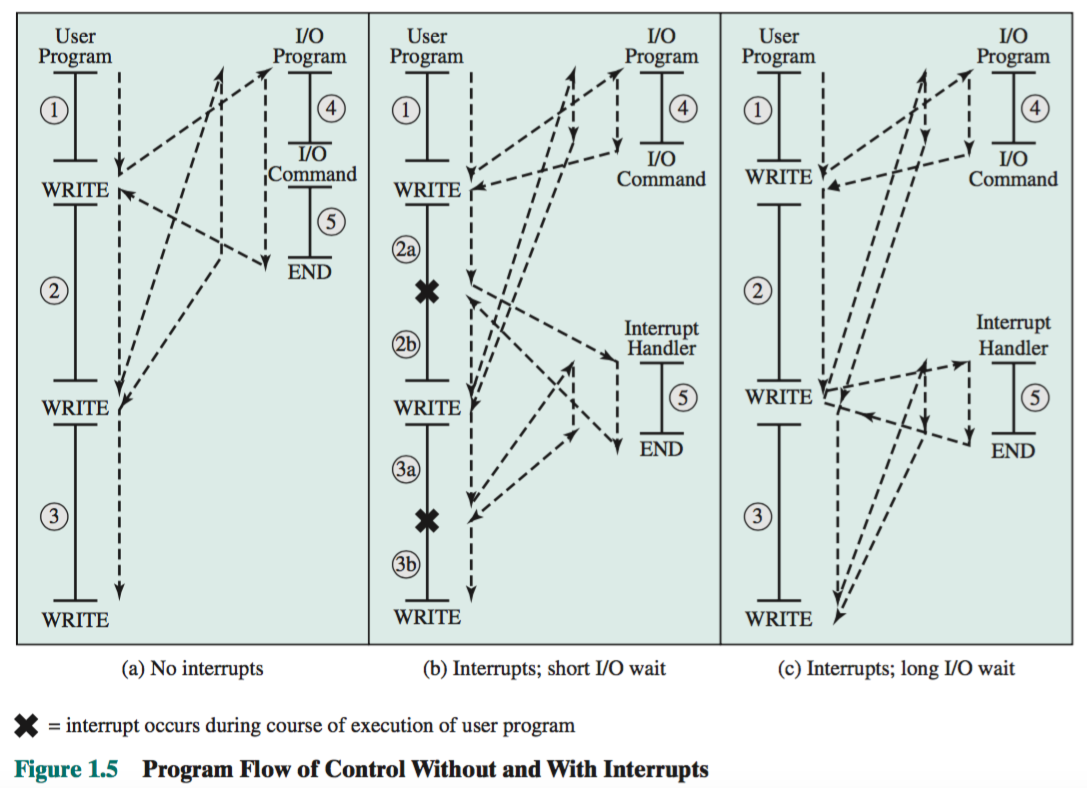
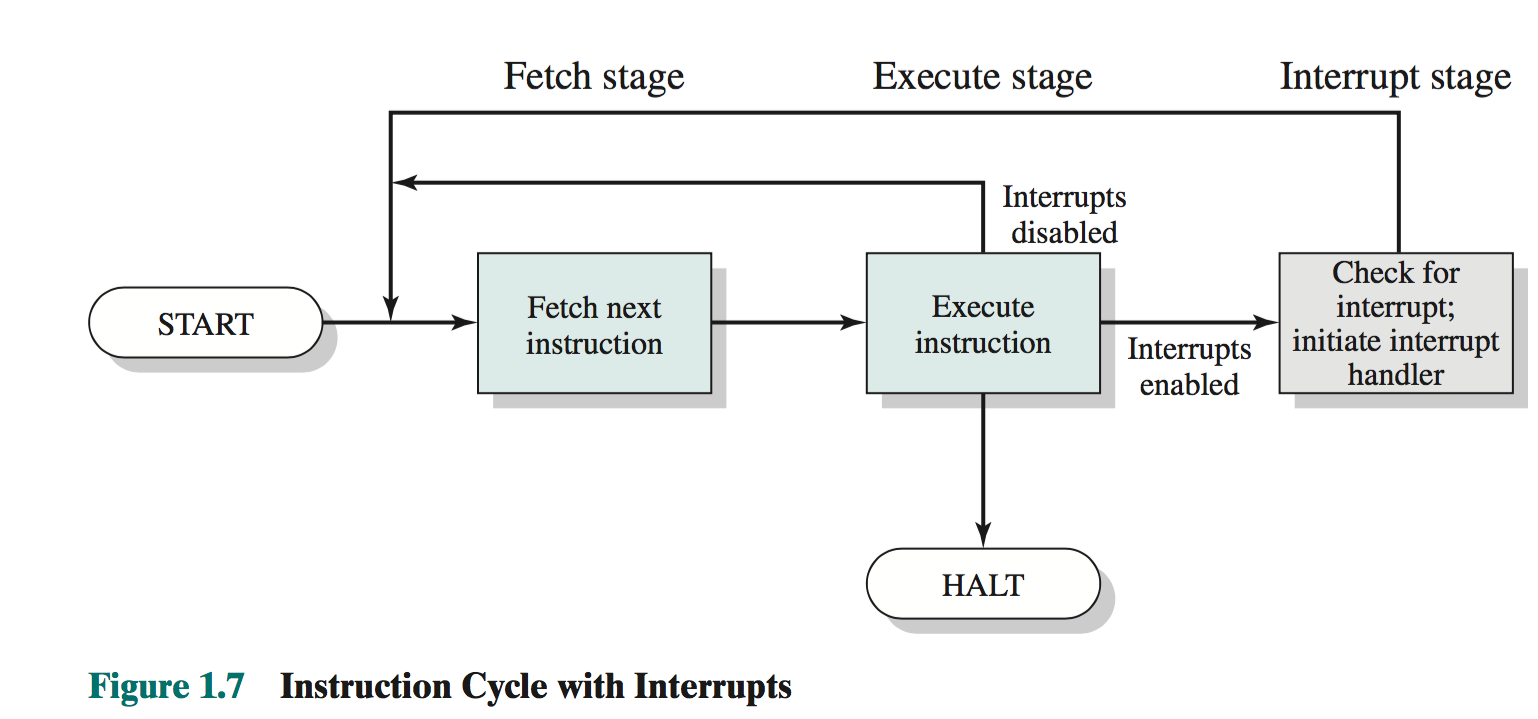
# 操作系统

## 中断





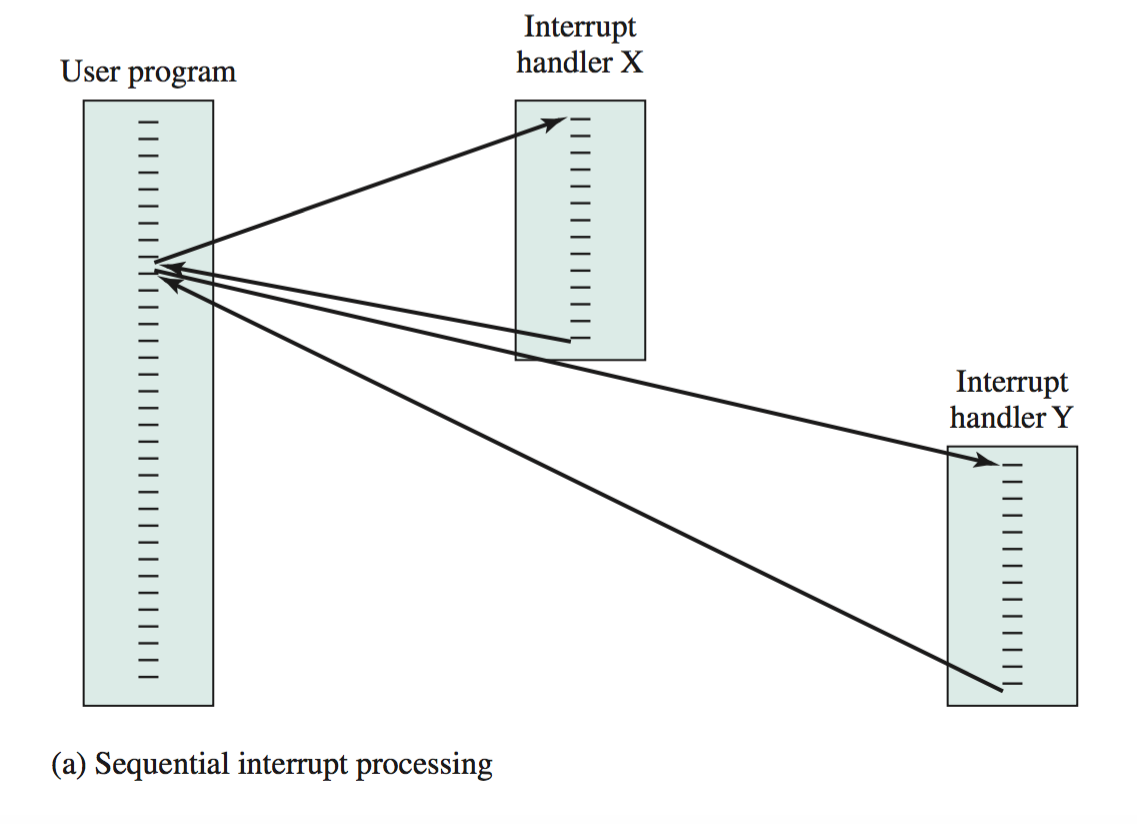


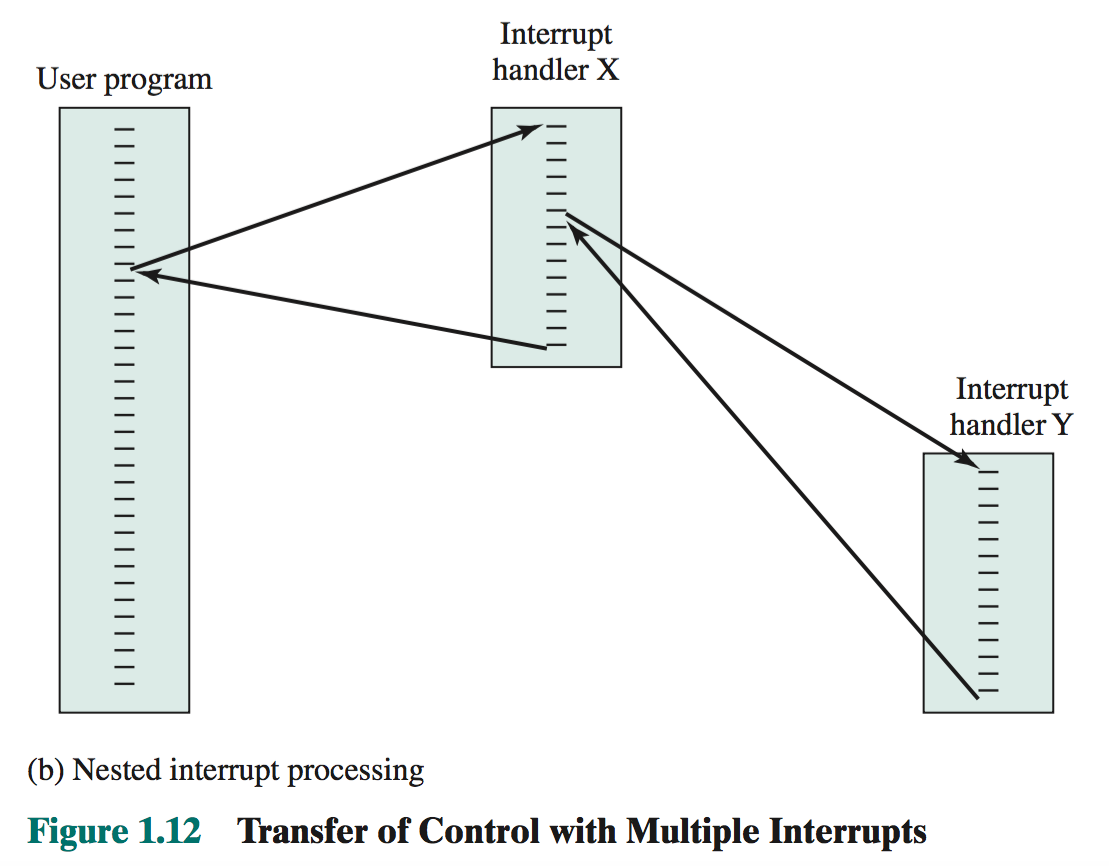


### 多中断处理

Two approaches can be taken to dealing with multiple interrupts. The first is to disable interrupts while an interrupt is being processed. A *disabled interrupt* simply means that the processor ignores any new interrupt request signal. If an interrupt occurs during this time, it generally remains pending and will be checked by the processor after the processor has reenabled interrupts. Thus, if an interrupt occurs when a user program is executing, then interrupts are disabled immediately. After the interrupt-handler routine completes, interrupts are reenabled before re- suming the user program, and the processor checks to see if additional interrupts have occurred. This approach is simple, as interrupts are handled in strict sequen- tial order (Figure 1.12a).

The drawback to the preceding approach is that it does not take into account relative priority or time-critical needs. For example, when input arrives from the com- munications line, it may need to be absorbed rapidly to make room for more input. If the first batch of input has not been processed before the second batch arrives, data may be lost because the buffer on the I/O device may fill and overflow

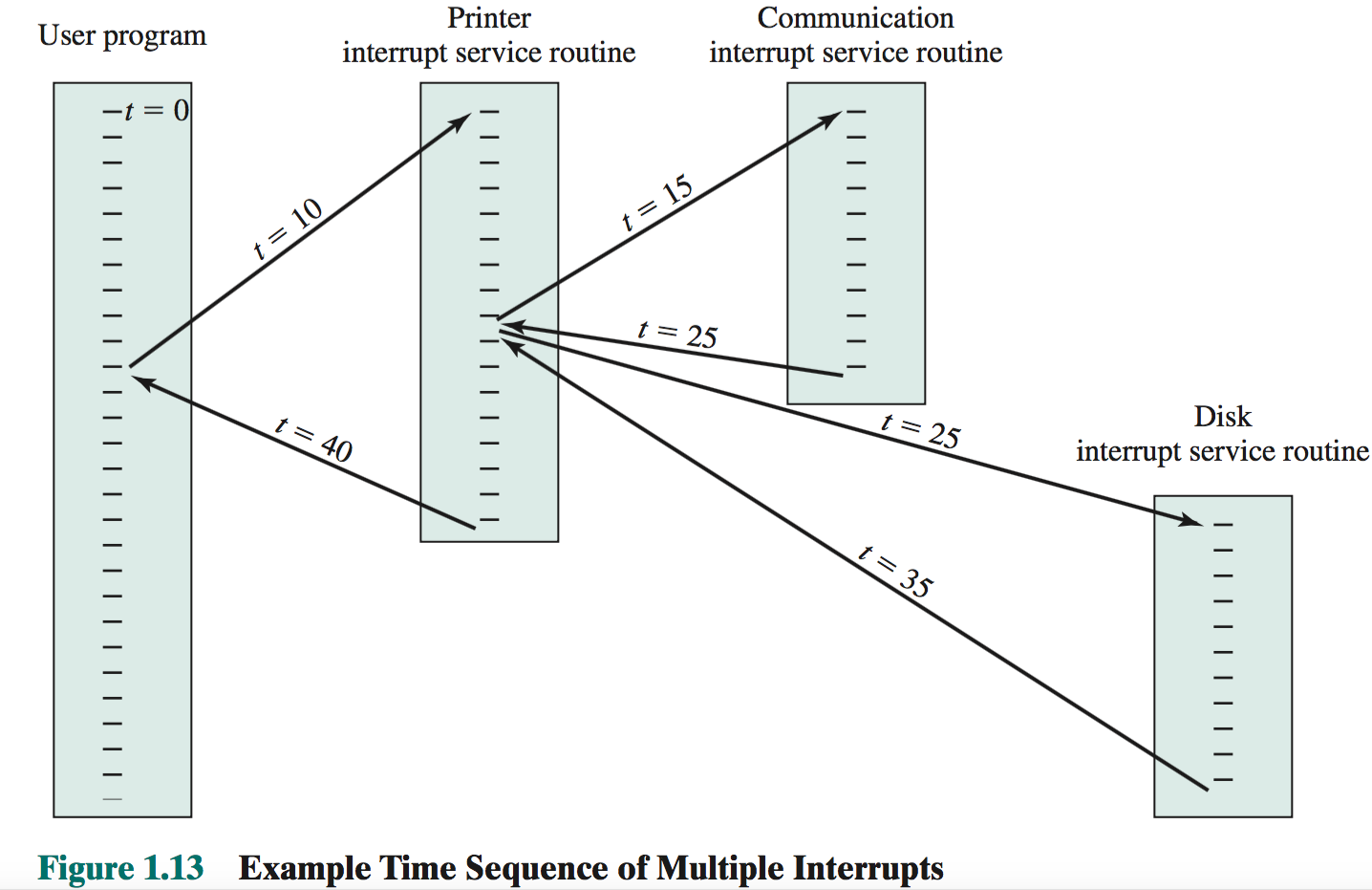




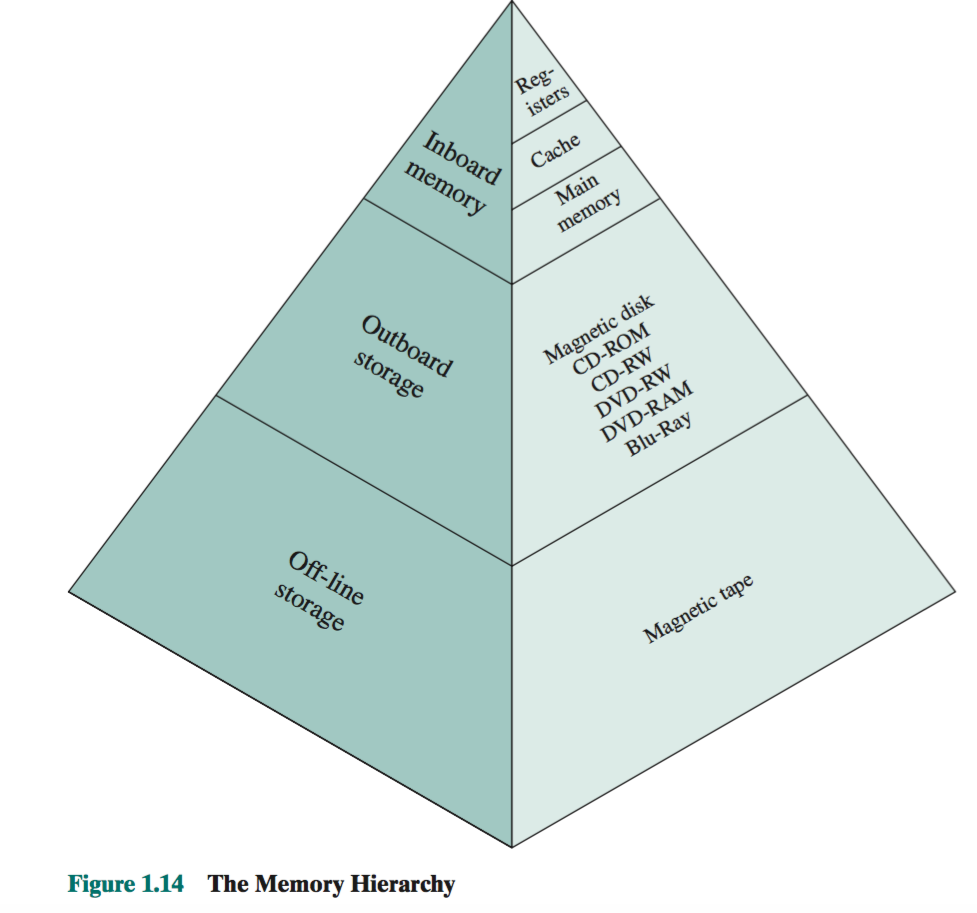
### 有优先级的中断

括号中的数字代表优先级

communication(5) > disk(4) >printer(3)



## 内存



The cache is not usually visible to the programmer or, indeed, to the processor. It is a device for staging the movement of data between main memory and processor registers to improve performance

## 缓存

To express the average time to access an item, we must consider not only the speeds of the two levels of memory but also the probability that a given reference can be found in M1. We have

where

*T* = *H* \* *T* + 11 - *H*2 \* 1*T* + *T* 2 *s*112

*Ts* = average (system) access time *T*1 = access time of M1 (e.g., cache, disk cache) *T*2 = access time of M2 (e.g., main memory, disk) *H* = hit ratio (fraction of time reference is found in M1)

**Performance**

Let us look at some of the parameters relevant to an assessment of a two-level mem- ory mechanism. First consider cost. We have

*CS* = *C*1*S*1 + *C*2*S*2 *S*1 + *S*2

**(1.2)**



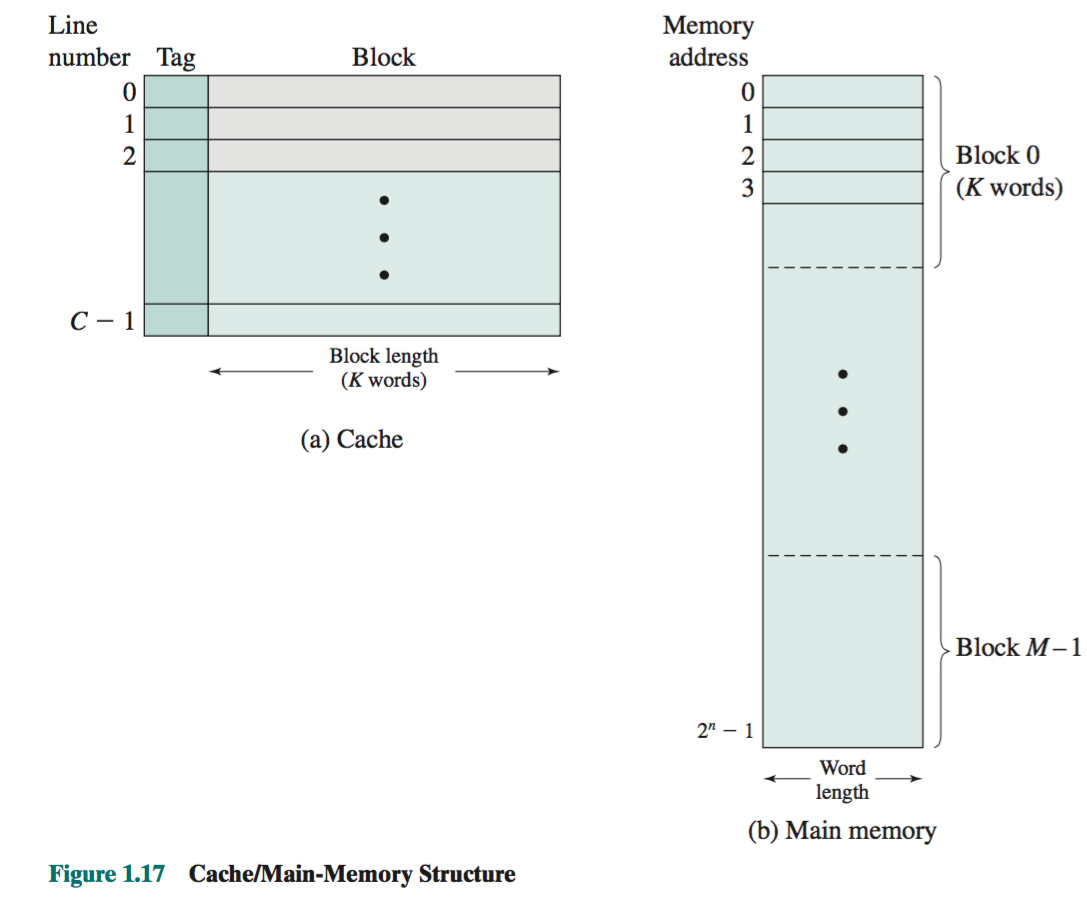
where

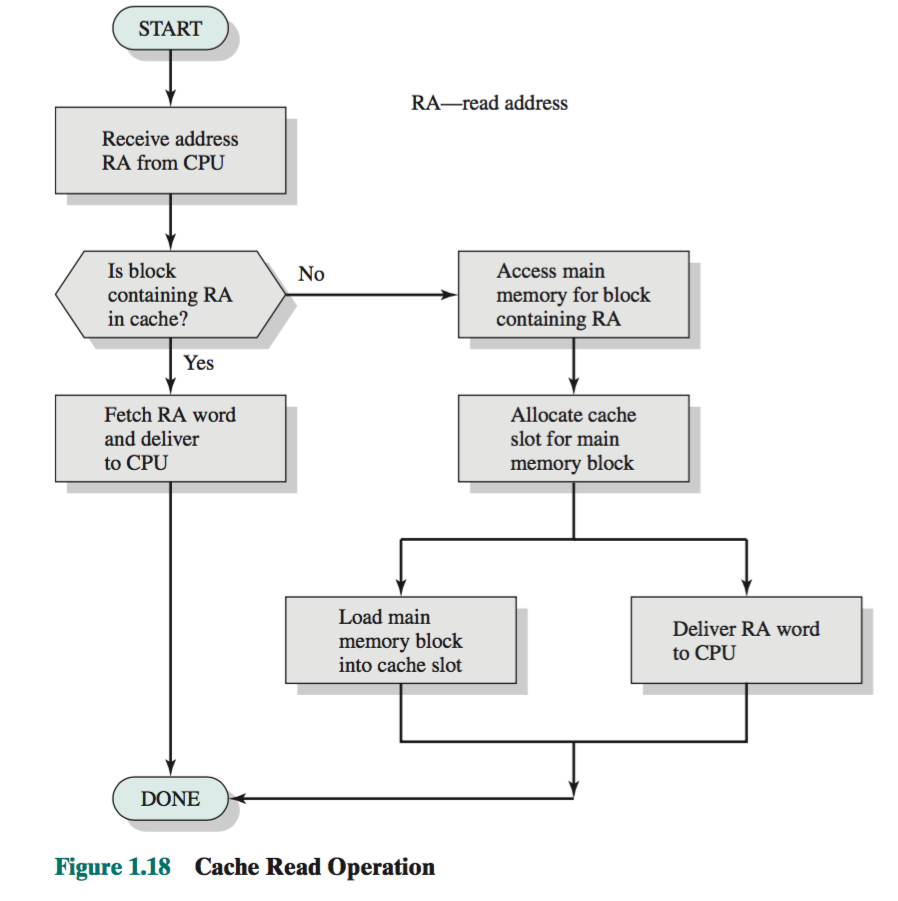
*Cs* = average cost per bit for the combined two-level memory *C*1 = average cost per bit of upper-level memory M1 *C*2 = average cost per bit of lower-level memory M2 *S*1 = size of M1

*S*2 = size of M2 We would like *Cs* ≈ *C*2. Given that *C*1 7 7 *C*2, this requires *S*1 6 6 *S*2. Figure 1.22

shows the relationship.7

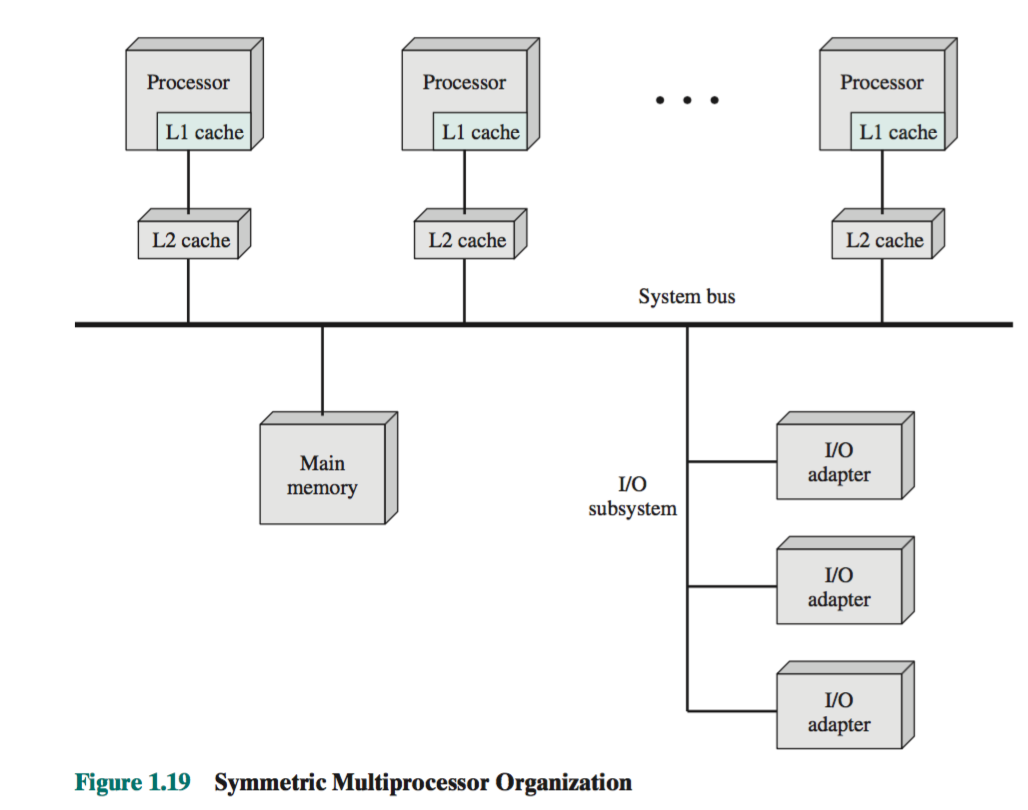




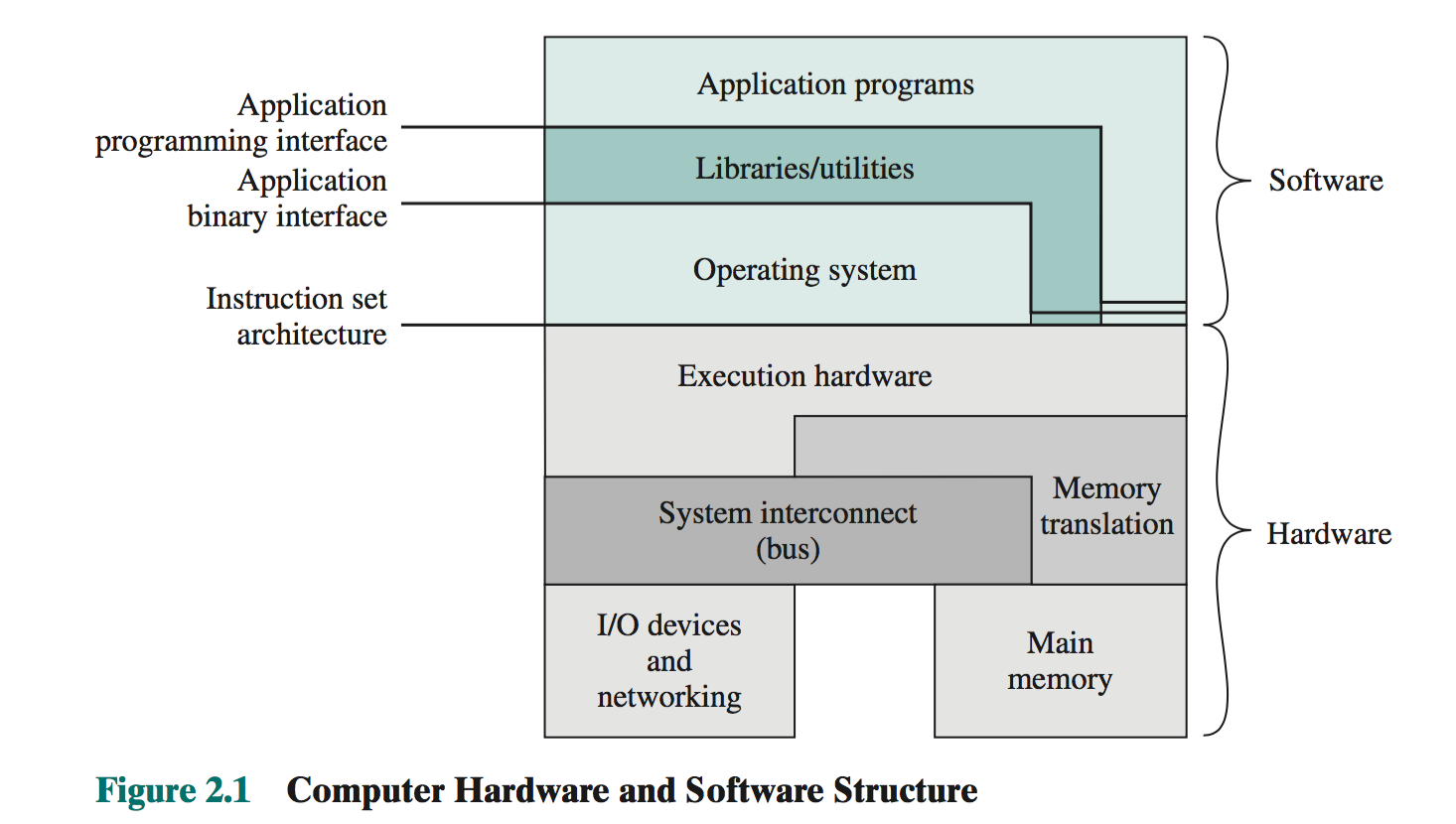


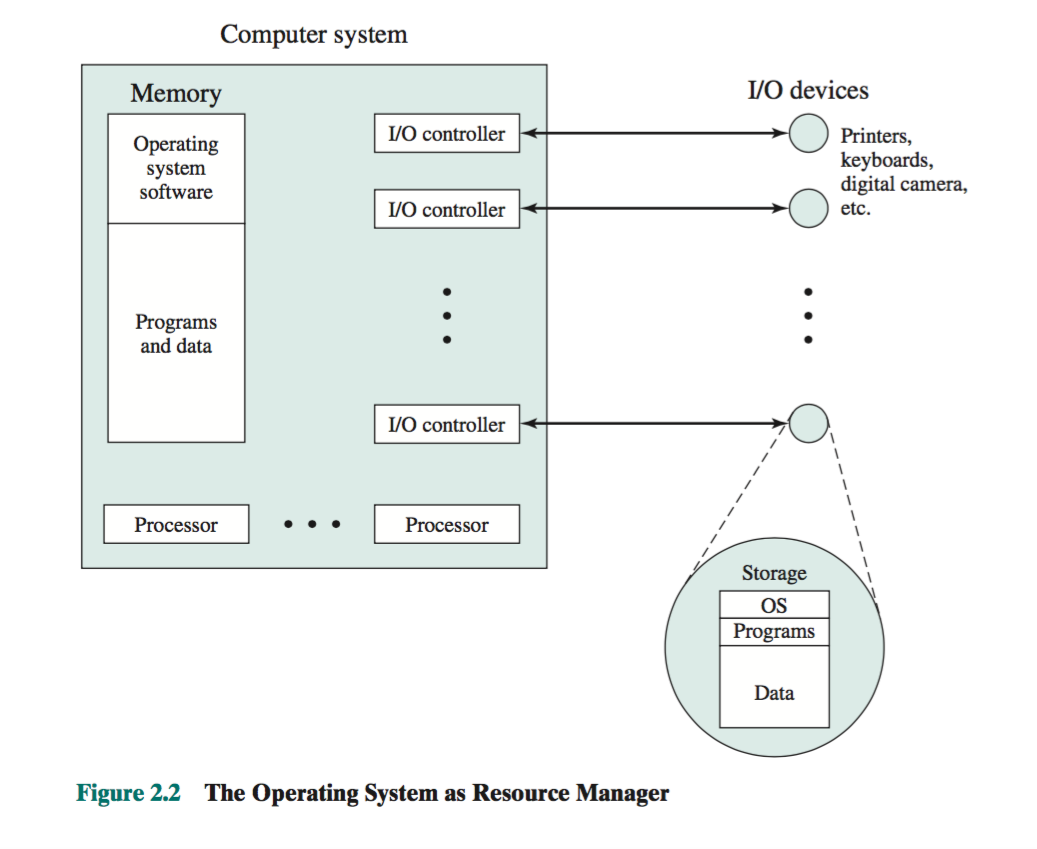
## 直接内存

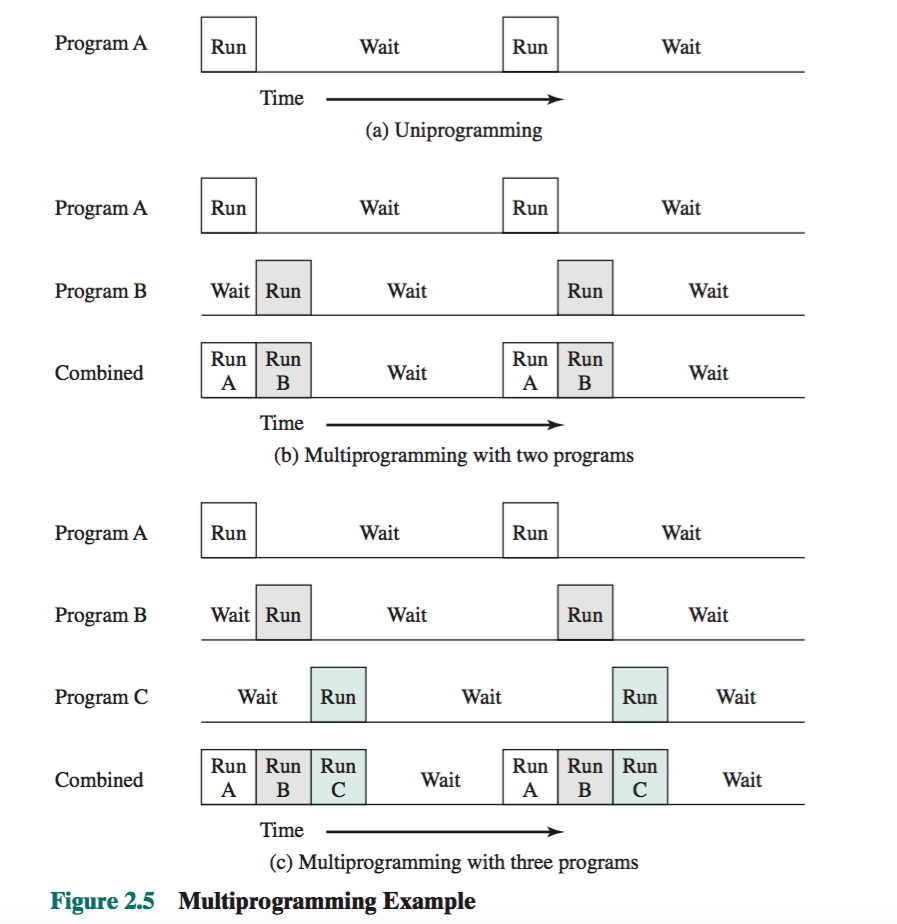
## Symmetric Multiprocessor

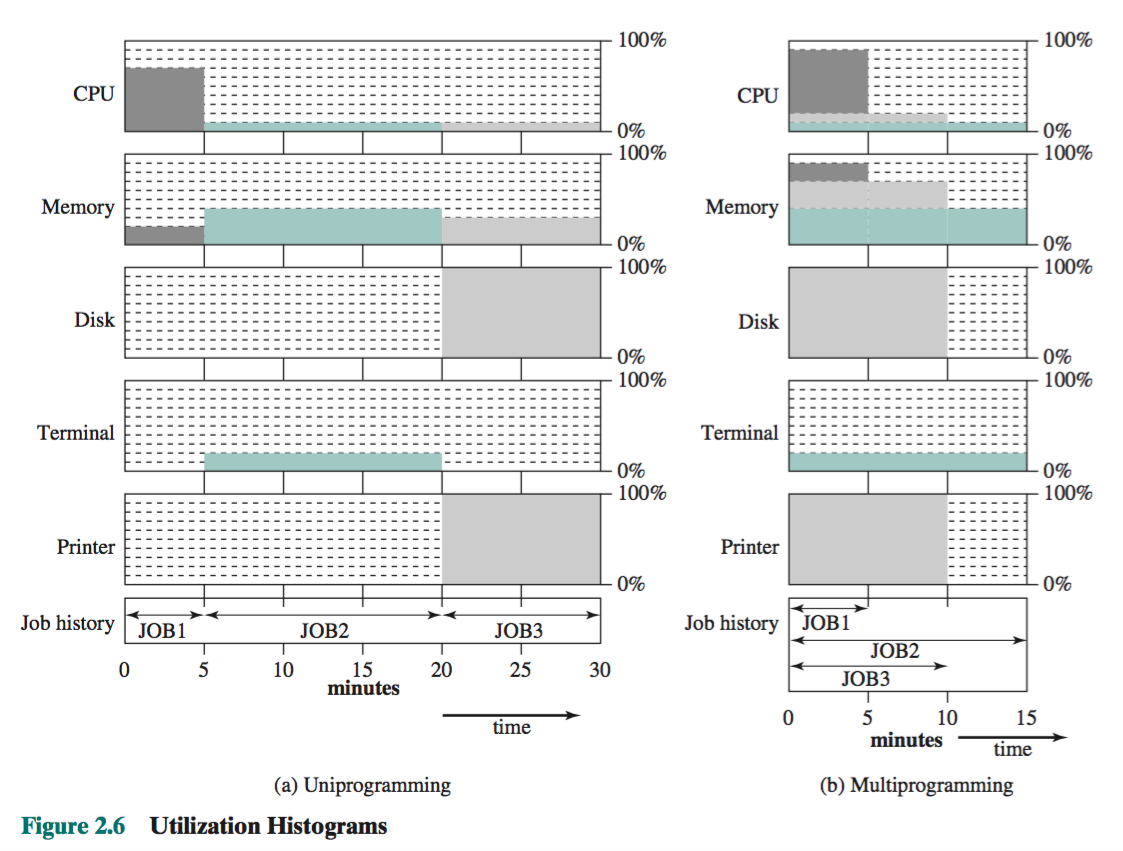


## 计算机硬件和软件结构









# Linux

## 自动启动脚本

http://www.linuxidc.com/Linux/2015-04/116648.htm

http://www.centoscn.com/CentOS/config/2015/0507/5374.html

cat /lib/systemd/system/nginx.service

## CPU使用

**vmstat 1**

the vmstat output has two columns (si, for swap in, and so, for swap out) that alert us if the system is swapping 5 1 \* \* \* source /etc/profile; /opt/cndw/shell/getmerge.py >> /opt/cndw/shell/result.log 2>&1 & result.log

## 磁盘使用

yum install sysstat

The basic I/O monitors on some systems are better than on others. Here is some partial output of iostat on a Linux system:

% **iostat -xm 5**

avg-cpu: %user %nice %system %iowait %steal %idle

23.45 0.00 37.89 0.10 0.00 38.56

Device: rrqm/s wrqm/s r/s w/s rMB/s

sda 0.00 11.60 0.60 24.20 0.02

wMB/s avgrq-sz avgqu-sz await r\_await w\_await svctm %util

0.14 13.35 0.15 6.06 5.33 6.08 0.42 1.04

The application here is writing data to disk sda. At first glance, the disk statistics look good. The w\_await—the time to service each I/O write—is fairly low (6.08 ms), and the disk is only 1.04% utilized. (The acceptable values for that depend on the physical disk, but the 5200 RPM disk in my desktop system behaves well when the service time is under 15 ms.) But there is a clue here that something is wrong: the system is spending 37.89% of its time in the kernel. If the system is doing other I/O (in other programs), that’s one thing; if all that system time is from the application being tested, then some‐ thing inefficient is happening.

The fact that the system is doing 24.2 writes per second is another clue here: that is a lot when writing only 0.14 MB per second (MBps). I/O has become a bottleneck, and the next step would be to look into how the application is performing its writes.

% **iostat -xm 5**avg-cpu: %user %nice %system %iowait %steal %idle

35.05 0.00 7.85 47.89 0.00 9.20

Device: rrqm/s wrqm/s r/s w/s rMB/s

sda 0.00 0.20 1.00 163.40 0.00

wMB/s avgrq-sz avgqu-sz await r\_await w\_await svctm %util

81.09 1010.19 142.74 866.47 97.60 871.17 6.08 100.00

The nice thing about Linux is that it tells us immediately that the disk is 100% utilized; it also tells us that processes are spending 47.89% of their time in iowait (that is, waiting for the disk).

Even on other systems where only raw data is available, that data will tell us something is amiss: the time to complete the I/O (w\_await) is 871 ms, the queue size is quite large, and the disk is writing 81 MB of data per second. This all points to disk I/O as a problem, and that the amount of I/O in the application (or, possibly, elsewhere in the system) must be reduced.

## 网络监控

netstat

开源工具 nicstat

http://angus717.blog.51cto.com/1593644/1208937/

## 查看端口占用情况

netstat -lnp | grep 4141

lsof -i tcp:4141

## 设置开机启动

chkconfig mysqld on

## yum

yum install psmisc

yum install net-tools

## RMP包安装

rpm -ivh mysql-community-release-el7-5.noarch.rpm

## 修改linux swap空间的swappiness，降低对硬盘的缓存

linux 会使用硬盘的一部分做为SWAP分区，用来进行进程调度--进程是正在运行的程序--把当前不用的进程调成‘等待（standby）‘，甚至‘睡眠 （sleep）’，一旦要用，再调成‘活动（active）’，睡眠的进程就躺到SWAP分区睡大觉，把内存空出来让给‘活动’的进程。 　　如果内存够大，应当告诉 linux 不必太多的使用 SWAP 分区， 可以通过修改 swappiness 的数值。swappiness=0的时候表示最大限度使用物理内存，然后才是 swap空间，swappiness＝100的时候表示积极的使用swap分区，并且把内存上的数据及时的搬运到swap空间里面。 　　在ubuntu 里面，默认设置swappiness这个值等于60。 　　 　　!!!! 如果内存较小，而进程调度频繁，硬盘的响动就会大了 !!!! 　　 　　现在一般1个G的内存可修改为10， 2个G的可改为5， 甚至是0。具体这样做： 　　1.查看你的系统里面的swappiness 　　$ cat /proc/sys/vm/swappiness 　　不出意外的话，你应该看到是 60 　　2.修改swappiness值为10 　　$ sudo sysctl vm.swappiness=10 　　但是这只是临时性的修改，在你重启系统后会恢复默认的60，为长治久安，还要更进一步： 　　$ sudo gedit /etc/sysctl.conf 　　在这个文档的最后加上这样一行: 　　vm.swappiness=10 　　然后保存，重启。ok，你的设置就生效了。

## shell bash判断文件或文件夹是否存在

#shell判断文件夹是否存在

#如果文件夹不存在，创建文件夹

if [ ! -d "/myfolder" ]; then

mkdir /myfolder

fi

#shell判断文件,目录是否存在或者具有权限

folder="/var/www/"

file="/var/www/log"

# -x 参数判断 $folder 是否存在并且是否具有可执行权限

if [ ! -x "$folder"]; then

mkdir "$folder"

fi

# -d 参数判断 $folder 是否存在

if [ ! -d "$folder"]; then

mkdir "$folder"

fi

# -f 参数判断 $file 是否存在

if [ ! -f "$file" ]; then

touch "$file"

fi

# -n 判断一个变量是否有值

if [ ! -n "$var" ]; then

echo "$var is empty"

exit 0

fi

# 判断两个变量是否相等

if [ "$var1" = "$var2" ]; then

echo '$var1 eq $var2'

else

echo '$var1 not eq $var2'

fi

## [Linux shell 之 提取文件名和目录名的一些方法](http://blog.csdn.net/ljianhui/article/details/43128465)

很多时候在使用Linux的shell时，我们都需要对文件名或目录名进行处理，通常的操作是由路径中提取出文件名，从路径中提取出目录名，提取文件后缀名等等。例如，从路径/dir1/dir2/file.txt中提取也文件名file.txt，提取出目录/dir1/dir2，提取出文件后缀txt等。

下面介绍两种常用的方法来进行相关的操作。

一、使用${}

1、${var##\*/}

该命令的作用是去掉变量var从左边算起的最后一个'/'字符及其左边的内容，返回从左边算起的最后一个'/'（不含该字符）的右边的内容。使用例子及结果如下：



从运行结果可以看到，使用该命令，可以提取出我们需要的文件名file.txt。

若使用时在shell程序文件中，可以使用变量来保存这个结果，再加以利用，如file=${var##\*/}

2、${var##\*.}

该命令的作用是去掉变量var从左边算起的最后一个'.'字符及其左边的内容，返回从左边算起的最后一个'.'（不含该字符）的右边的内容。使用例子及结果如下：



从运行结果可以看到，使用该命令，可以提取出我们需要的文件后缀。

如果文件的后缀不仅有一个，例如，file.tar.gz，命令${var##\*.}仅能提取最后一个后缀，而我想提取tar.gz时该怎么办？那么就要用下面所说的${var#\*.}命令了。

3、${var#\*.}

该命令的作用是去掉变量var从左边算起的第一个'.'字符及其左边的内容，返回从左边算起第一个'.'（不含该字符）的右边部分的内容。使用例子及结果如下：



从运行结果可以看到，使用该命令，可以提取出文件的多个后缀。

4、${var%/\*}

该命令的使用是去掉变量var从右边算起的第一个'/'字符及其右边的内容，返回从右边算起的第一个'/'（不含该字符）的左边的内容。使用例子及结果如下：



从运行的结果可以看到，使用该命令，可以提取出我们需要的文件所在的目录

5、${var%%.\*}

该命令的使用是去掉变量var从右边算起的最后一个'.'字符及其右边的内容，返回从右边算起的最后一个'.'（不含该字符）的左边的内容。使用例子及结果如下：



当我们需要建立一个与文件名相同名字（没有后缀）的目录与对应的文件相对应时，就可以使用该命令来进行操作。例如，解压文件的情况就与此类似，我们压缩文件file.zip时，会在与file.zip同级目录下建立一个名为file的目录。

6、${}总结

其实${}并不是专门为提取文件名或目录名的，它的使用是变量的提取和替换等等操作，它可以提取非常多的内容，并不一定是上面五个例子中的'/'或'.'。也就是说，上面的使用方法只是它使用的一个特例。

看到上面的这些命令，可能会让人感到非常难以理解和记忆，其实不然，它们都是有规律的。

#：表示从左边算起第一个

%：表示从右边算起第一个

##：表示从左边算起最后一个

%%：表示从右边算起最后一个

换句话来说，＃总是表示左边算起，％总是表示右边算起。

＊：表示要删除的内容，对于#和##的情况，它位于指定的字符（例子中的'/'和'.'）的左边，表于删除指定字符及其左边的内容；对于%和%%的情况，它位于指定的字符（例子中的'/'和'.'）的右边，表示删除指定字符及其右边的内容。这里的'\*'的位置不能互换，即不能把\*号放在#或##的右边，反之亦然。

例如：${var%%x\*}表示找出从右边算起最后一个字符x，并删除字符x及其右边的字符。

看到这里，就可以知道，其实该命令的用途非常广泛，上面只是指针文件名和目录名的命名特性来进行提取的一些特例而已。

二、basename和dirname

${}并不是专门为提取文件名和目录名设计的命令，那么basename和dirname命令就是专门为做这一件事而已准备的了。

1、basename

该命令的作用是从路径中提取出文件名，使用方法为basename NAME [SUFFIX]。

1）从路径中提出出文件名（带后缀），例子如下：



2）从上面命令的用法中可以看到，后缀（SUFFIX）是一个可选项。所以，若只想提取出文件名file，而不带有后缀，还可以在变量的后面加上后缀名，例子如下：



2、dirname

该命令的作用是从路径中提取出目录名，使用方法为 dirname NAME

使用例子如下：



这样就提取出了file.txt文件所在的目录。

注：该命令不仅能提取出普通文件所的目录，它能提取出任何文件所在的目录，例如目录所在的目录，如下：



它提取出了目录dir2所在的目录dir1.

## linux下logrotate 配置和理解

<http://blog.csdn.net/cjwid/article/details/1690101>

## 删掉多少天前日志命令

find /opt/var/log/impalad/profiles -mtime +2 -name "impala\_profile\_log\_\*" -exec rm -rf {} \;

## 时间函数

a=`date "+%Y-%m-%d"`

b=`date -d "$a - 10 days"` #前10天的日期

得到上一个小时

a=`date "+%H"`

b=`date -d "$a-1 hour" "+%H"`

b=`date -d "$a-65 minutes" "+%H"`

cur\_time=`date "+%s"` #时间戳

### pre\_hour=`date -d "- 65 minutes" "+%H"`

### linux在shell中获取时间

<http://www.cnblogs.com/mfryf/archive/2012/03/23/2413362.html>

## Firewall

检查防火墙状态

firewall-cmd --stat

临时开放ftp服务

firewall-cmd --add-service=ftp

永久开放ftp服务

firewall-cmd --add-service=ftp --permanent

关闭ftp服务

firewall-cmd --remove-service=ftp --permanent

配置防火墙在public区域永久开放http服务

firewall-cmd --permanent --zone=public --add-service=http

加入指定开放端口

firewall-cmd --add-port=1324/tcp

为了让之前的设定生效当然要重启服务咯

systemctl restart firewalld

或者使用下面的命令免去重启服务(防火墙策略配置后重新载入)

firewall-cmd --complete-reload

firewall-cmd --reload   (这两句功能相同)

检查ftp服务的21端口是否开放

iptables -L -n | grep 21 ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0 tcp dpt:21 ctstate NEW

查询ftp服务启用状态

firewall-cmd --query-service ftp

查看当前规则

firewall-cmd --list-all

仅允许部分IP访问本机服务配置

firewall-cmd --permanent --zone=public --add-rich-rule="rule family="ipv4" source address="192.168.0.4/24" service name="http" accept"

仅允许部分IP访问本机端口配置

firewall-cmd --permanent --zone=public --add-rich-rule="rule family="ipv4" source address="192.168.0.4/24" port protocol="tcp" port="8080" accept"

firewall-cmd --zone=public --add-port=80/tcp –permanent

firewall-cmd --permanent --zone=public --add-rich-rule="rule family="ipv4" source address="127.0.0.1" port protocol="tcp" port="19090" accept"

firewall-cmd --remove-rich-rule="rule family="ipv4" source address="127.0.0.1" port protocol="tcp" port="19090" accept"

firewall-cmd --permanent --add-rich-rule="rule family="ipv4" source address="123.207.85.215" port protocol="tcp" port="8080" drop"

firewall-cmd --permanent --add-rich-rule="rule family="ipv4" source address="183.60.84.174" port protocol="tcp" port="8080" drop"

## 理解Centos7防火墙firewalld

http://www.th7.cn/system/lin/201601/150565.shtml

## Linux NFS服务器的安装与配置

<http://www.cnblogs.com/mchina/archive/2013/01/03/2840040.html>

## [Linux下如何查看分区文件系统类型](http://freearth.blog.chinaunix.net/uid-20769015-id-3478861.html)

<http://freearth.blog.chinaunix.net/uid-20769015-id-3478861.html>

mount 也可查看到

## linux作业管理（ctrl+z,fg,jobs,kill等）

<http://blog.csdn.net/oopdesigner/article/details/5357337>

## 安装人类能看得懂的转化工具:yum install setroubleshoot

echo >audit.log来清空这个日志,再刷新浏览器,访问一下这个url,让只生成一个出错日志

把出错内容转成可以看得懂的:sealert -a ./audit.log >qq.txt

## 修改编码

vim /etc/locale.conf

LANG="zh\_CN.UTF-8"

## Selinux

### SELinux深入理解

<http://blog.csdn.net/myarrow/article/details/9856095/>

### 政策查阅

CentOS 6.x 预设使使用 targeted 政策,那么这个政策提供多少相关的规则呢? 此时可以透过 seinfo 来查询喔!

yum install setools-console

# 范例一:列出 SELinux 在此政策下的统计状态

[root@www ~]# seinfo tatistics for policy file: /etc/selinux/targeted/policy/policy.24 Policy Version & Type: v.24 (binary, mls) <==列出政策所在档与版本

Classes: 77 Permissions: 229

Sensitivities: 1 Categories: 1024

Types: 3076 Attributes: 251

Users: 9 Roles: 13

Booleans: 173 Cond. Expr.: 208

Allow: 271307 Neverallow: 0

Auditallow: 44 Dontaudit: 163738

Type\_trans: 10941 Type\_change: 38

Type\_member: 44 Role allow: 20

Role\_trans: 241 Range\_trans: 2590

....(底下省略)....

# 从上面我们可以看到这个政策是 targeted ,此政策的 SELinux type 有 3076 个;

# 而针对网络服务的规则 (Booleans) 共制订了 173 条规则!

# 范例二:列出与 httpd 有关的规则 (booleans) 有哪些? [root@www ~]# seinfo -b | grep httpd Conditional Booleans: 173

allow\_httpd\_mod\_auth\_pam

httpd\_setrlimit

httpd\_enable\_ftp\_server

....(底下省略).... # 你可以看到,有非常多的与 httpd 有关的规则订定呢!



从上面我们可以看到与 httpd 有关的布尔值,同样的,如果你想要找到有 httpd 字样 的安全性本文类别时, 就可以使用『 seinfo -t | grep httpd 』来查询了!如果查 询到相关的类别或者是布尔值后,想要知道详细的规则时, 就得要使用 sesearch 这 个指令了!



[root@www ~]# sesearch [--all] [-s 主体类别] [-t 目标类别] [-b 布尔值] 选项与参数:

--all :列出该类别或布尔值的所有相关信息 -t :后面还要接类别,例如 -t httpd\_t -b :后面还要接布尔值的规则,例如 -b httpd\_enable\_ftp\_server

# 范例一:找出目标档案资源类别为 httpd\_sys\_content\_t 的有关信息 [root@www ~]# sesearch --all -t httpd\_sys\_content\_t Found 683 semantic av rules:

allow avahi\_t file\_type : filesystem getattr ;

allow corosync\_t file\_type : filesystem getattr ;

allow munin\_system\_plugin\_t file\_type : filesystem getattr ;

....(底下省略).... # 『 allow 主体程序安全性本文类别 目标档案安全性本文类别 』 # 如上,说明这个类别可以被那个主题程序的类别所读取,以及目标档案资源

的格式。

你可以很轻易的查询到某个主体程序 (subject) 可以读取的目标档案资源 (Object)。 那如果是布尔值呢?里面又规范了什么?让我们来看看先:

# 范例三:我知道有个布尔值为 httpd\_enable\_homedirs ,请问该布尔值规范多 少规则?

[root@www ~]# sesearch -b httpd\_enable\_homedirs --all

Found 43 semantic av rules:

allow httpd\_user\_script\_t user\_home\_dir\_t : dir { getattr search

open } ;

allow httpd\_sys\_script\_t user\_home\_dir\_t : dir { ioctl read

getattr } ;

....(后面省略)....

从这个布尔值的设定我们可以看到里面规范了非常多的主体程序与目标档案资源的放 行与否! 所以你知道了,实际规范这些规则的,就是布尔值的项目啦!那也就是我们 之前所说的一堆规则是也! 你的主体程序能否对某些目标档案进行存取,与这个布尔 值非常有关系喔!因为布尔值可以将规则设定为启动 (1) 或者是关闭 (0) 啦!

## 数据备份同步：

### RSync实现文件备份同步

http://www.cnblogs.com/itech/archive/2009/08/10/1542945.html

## 网络:

### 网络监听的端口口分析

netstat -tulnp

### 如何限制IP，通过SSH登陆linux服务器

<http://www.t086.com/article/4828>

### 查看是否安装软件，及查看软件配置文件

rpm -qa | grep -i samba 是否安装samba

yum search samba <==先查一下有没有相关的软件

yum install samba <==找到之后,那就安装吧!

rpm -qc samba samba-common 查看配置文件的位置

### linux mtu查看&&设置

http://www.jb51.net/LINUXjishu/58083.html

MTU是Maximum Transmission Unit的缩写。意思是网络上传送的最大数据包。MTU的单位是字节

大部分网络设备都是1500。如果本机的MTU比网关的MTU大，大的数据包就会被拆开来传送，这样会产生很多数据包碎片，增加丢包率，降低网络速度。把本机的MTU设成比网关的MTU小或相同，就可以减少丢包

测试 (1472 + 8(ICMP回显示请求和回显应答报文格式长度) + 20(IP首部) = 1500)

$ ping -c 2 -M do -s 1472 192.168.0.6输出

复制代码

代码如下:

PING 192.168.0.6 (192.168.0.6) 1472(1500) bytes of data.

bytes from 192.168.0.6: icmp\_seq=1 ttl=127 time=1.06 ms

bytes from 192.168.0.6: icmp\_seq=2 ttl=127 time=1.06 ms

--- 192.168.0.6 ping statistics ---

packets transmitted, 2 received, 0% packet loss, time 1002ms

rtt min/avg/max/mdev = 1.060/1.062/1.064/0.002 ms

查看本机的mtu

$ netstat -i输出

Kernel Interface table

Iface MTU Met RX-OK RX-ERR RX-DRP RX-OVR TX-OK TX-ERR TX-DRP TX-OVR Flg

eth0 1500 0 82879 0 0 0 69000 0 0 0 BMRU

lo 16436 0 84 0 0 0 84 0 0 0 LRU

设置

# 设置本机的mtu

sudo ifconfig eth0 mtu 1472

# 查看设置后的结果

$ netstat -i

### Traceroute的工作原理：

Traceroute最简单的基本用法是：traceroute hostname

Traceroute程序的设计是利用ICMP及IP header的TTL（Time To Live）栏位（field）。首先，traceroute送出一个TTL是1的IP datagram（其实，每次送出的为3个40字节的包，包括源地址，目的地址和包发出的时间标签）到目的地，当路径上的第一个路由器（router）收到这个datagram时，它将TTL减1。此时，TTL变为0了，所以该路由器会将此datagram丢掉，并送回一个「ICMP time exceeded」消息（包括发IP包的源地址，IP包的所有内容及路由器的IP地址），traceroute 收到这个消息后，便知道这个路由器存在于这个路径上，接着traceroute 再送出另一个TTL是2 的datagram，发现第2 个路由器...... traceroute 每次将送出的datagram的TTL 加1来发现另一个路由器，这个重复的动作一直持续到某个datagram 抵达目的地。当datagram到达目的地后，该主机并不会送回ICMP time exceeded消息，因为它已是目的地了，那么traceroute如何得知目的地到达了呢？

Traceroute在送出UDP datagrams到目的地时，它所选择送达的port number 是一个一般应用程序都不会用的号码（30000 以上），所以当此UDP datagram 到达目的地后该主机会送回一个「ICMP port unreachable」的消息，而当traceroute 收到这个消息时，便知道目的地已经到达了。所以traceroute 在Server端也是没有所谓的Daemon 程式。

Traceroute提取发 ICMP TTL到期消息设备的IP地址并作域名解析。每次 ，Traceroute都打印出一系列数据,包括所经过的路由设备的域名及 IP地址,三个包每次来回所花时间。

### 列出目前主机上面记载的 IP/MAC 对应的 ARP (Adress Resolution Protocol) 表格

arp -n

### 将 192.168.1.100 那部主机的网卡卡号直接写入 ARP 表格中

arp -s 192.168.1.100 01:00:2D:23:A1:0E

### 手动设置IP地址

#### 1. IP/Netmask/Gateway 的设定、启动与观察

设定网络参数得要修改 /etc/sysconfig/network-scripts/ifcfg-eth0,请记得, 这个 ifcfg-eth0 与档案内的 DEVICE 名称设定需相同,并且,在这个档案内的 所有设定,基本上就是 bash 的变量设定规则啦 (注意大小写)!

[root@www ~]# vim /etc/sysconfig/network-scripts/ifcfg-eth0

DEVICE="eth0"

HWADDR="08:00:27:71:85:BD" 略此项目

NM\_CONTROLLED="no"

ONBOOT="yes"

BOOTPROTO=none

手动可输入 none IPADDR=192.168.1.100 NETMASK=255.255.255.0 GATEWAY=192.168.1.254 # 重点是上面这几个设定项目,底下的则可以省略的啰! NETWORK=192.168.1.0 <==就是该网段的第一个 IP,可省略 BROADCAST=192.168.1.255 <==就是广播地址啰,可省略 MTU=1500 <==就是最大传输单元的设定值,若不更改则

可省略

<==网络卡代号,必须要 ifcfg-eth0 相对应 <==就是网络卡地址,若只有一张网卡,可省

<==不要受到其他软件的网络管理! <==是否默认启动此接口的意思 <==取得 IP 的方式,其实关键词只有 dhcp,



<==就是 IP 啊 <==就是子网掩码 <==就是预设路由

上面的资料很好理解吧!请注意每个变量(左边的英文)都应该要大写! 否则我 们的 script 会误判!事实上鸟哥的设定值只有最上面的 8 个而已,其他的 NETWORK, BROADCAST, MTU 鸟哥都没有设定喔! 至于参数的说明方面,IPADDR, NETMASK, NETWORK, BROADCAST 鸟哥在这里就不再多说,要谈的是几个重要的设 定值:

o o

o

DEVICE:这个设定值后面接的装置代号需要与文件名 (ifcfg-eth0) 那个装置代号相同才行!否则可能会造成一些装置名称找不到的困扰。

BOOTPROTO:启动该网络接口时,使用何种协议?如果是手动给予IP 的环境,请输入 static 或 none ,如果是自动取得 IP 的时候, 请输 入 dhcp (不要写错字,因为这是最重要的关键词!)

GATEWAY:代表的是『整个主机系统的 default gateway』, 所以, 设定这个项目时,请特别留意!不要有重复设定的情况发生喔!也就是当 你有 ifcfg-eth0, ifcfg-eth1.... 等多个档案,只要在其中一个档案设 定 GATEWAY 即可

o

o

GATEWAYDEV:如果你不是使用固定的 IP 作为 Gateway , 而是使用 网络装置作为 Gateway (通常 Router 最常有这样的设定),那也可以使 用 GATEWAYDEV 来设定通讯闸装置呢!不过这个设定项目很少使用就是了!

HWADDR:这个东西就是网络卡的卡号了!在仅有一张网卡的情况下, 这个设定值没有啥功能, 可以忽略他。但如果你的主机上面有两张一模 一样的网卡,使用的模块是相同的。此时,你的 Linux 很可能会将 eth0, eth1 搞混,而造成你网络设定的困扰。如何解决呢? 由于 MAC 是直接 写在网卡上的,因此指定 HWADDR 到这个配置文件中,就可以解决网卡对 应代号的问题了!很方便吧!

设定完毕之后,现在让我们来重新启动网络接口吧!这样才能更新整个网络参数 嘛!

[root@www ~]# /etc/init.d/network restart

Shutting down interface eth0:

Shutting down loopback interface:

Bringing up loopback interface:

Bringing up interface eth0:

[ OK ] <== 先关闭界面 [ OK ] [ OK ] <== 再开启界面 [ OK ]

# 针对这部主机的所有网络接口 (包含 lo) 与通讯闸进行重新启动,所以网 络会停顿再开

这样就处理完毕啰,那接下来当然就是观察看看啰!

# 检查一:当然是要先察看 IP 参数对否,重点是 IP 与 Netmask 啦! [root@www ~]# ifconfig eth0 eth0 Link encap:Ethernet HWaddr 08:00:27:71:85:BD

inet addr:192.168.1.100 Bcast:192.168.1.255

Mask:255.255.255.0

inet6 addr: fe80::a00:27ff:fe71:85bd/64 Scope:Link

UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

RX packets:655 errors:0 dropped:0 overruns:0 frame:0

TX packets:468 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:1000

RX bytes:61350 (59.9 KiB) TX bytes:68722 (67.1 KiB) # 有出现上头那个 IP 的数据才是正确的启动;特别注意 inet addr 与 Mask

项目 # 这里如果没有成功,得回去看看配置文件有没有错误,然后再重新 network

restart !

# 检查二:检查一下你的路由设定是否正确 [root@www ~]# route -n

Kernel IP routing table

Destination Gateway Genmask Flags Metric Ref

Use Iface

192.168.1.0 0.0.0.0 255.255.255.0 U 0 0

0 eth0

169.254.0.0 0.0.0.0 255.255.0.0 U 1002 0

0 eth0

0.0.0.0 192.168.1.254 0.0.0.0 UG 0 0

0 eth0

# 重点就是上面的特殊字体!前面的 0.0.0.0 代表预设路由的设定值!

# 检查三:测试看看与路由器之间是否能够联机成功呢! [root@www ~]# ping -c 3 192.168.1.254 PING 192.168.1.254 (192.168.1.254) 56(84) bytes of data. 64 bytes from 192.168.1.254: icmp\_seq=1 ttl=64 time=2.08 ms 64 bytes from 192.168.1.254: icmp\_seq=2 ttl=64 time=0.309 ms 64 bytes from 192.168.1.254: icmp\_seq=3 ttl=64 time=0.216 ms

--- 192.168.1.254 ping statistics --- 3 packets transmitted, 3 received, 0% packet loss, time 2004ms rtt min/avg/max/mdev = 0.216/0.871/2.088/0.861 ms # 注意啊!有出现 ttl 才是正确的响应!如果出现『 Destination Host

Unreachable 』 # 表示没有成功的联机到你的 GATEWAY 那表示出问题啦!赶紧检查有无设定

错误。

要注意,第三个检查如果失败,可能要看你的路由器是否已经关闭?或者是你的 switch/hub 是否有问题,或者是你的网络线是否错误,还是说你的或路由器的 防火墙设定错误了?要记得去解决喔! 这三个检查做完而且都成功之后,那么 你的 TCP/IP 参数设定已经完毕了!这表示你可以使用 IP 上网啦! 只是还不 能够使用主机名上网就是了。接下来就是要设定 DNS 啰!

#### 2.DNS 服务器的 IP 设定与观察

这个 /etc/resolv.conf 很重要啦!他会影响到你是否可以查询到主机名与 IP 的对应喔!通常如下的设定就 OK 了!

我们以中华电信与 SeedNet 在南部的 DNS 服务器之 IP 作为设定的方式!请注 意一下,如果你不知道你的最接近的 DNS 服务器的 IP ,那么直接输入

nameserver 168.95.1.1 这个中华电信的 DNS 主机即可!不过如果你公司内部 有设定防止 DNS 的要求封包的防火墙规则时,那么你就得要请教贵公司的网管 单位告知你的 DNS IP 设定啦!然后赶紧测试看看:

# 检查四:看看 DNS 是否顺利运作了呢?很重要的测试喔! [root@www ~]# dig www.google.com ....(前面省略).... ;; QUESTION SECTION:

;www.google.com.

;; ANSWER SECTION: www.google.com. www.l.google.com. ....(中间省略)....

428539 IN

122 IN

IN A

CNAME www.l.google.com.

A 74.125.71.106

;; Query time: 30 msec ;; SERVER: 168.95.1.1#53(168.95.1.1) <==这里的项目也很重要! ;; WHEN: Mon Jul 18 01:26:50 2011 ;; MSG SIZE rcvd: 284

上面的输出有两个重点,一个是问题查询的是 www.google.com 的 A (Address) 参数,并且从回答 (Answer) 里面得到我们所需的 IP 参数。最后面一段的 Server 项目非常重要!你得要看是否与你的设定相同的那部 DNS 服务器 IP 才 行! 以上面输出为例,鸟哥使用中华电信的 DNS 服务器,所以就出现 168.95.1.1 的 IP 地址啰

#### 3. 主机名的修改、启动与观察

修改主机名就得要改 /etc/sysconfig/network 以及 /etc/hosts 这两个档案, 这两个档案的内容又简单的要命喔!

[root@www ~]# vim /etc/sysconfig/network

NETWORKING=yes

HOSTNAME=www.centos.vbird

[root@www ~]# vim /etc/hosts 192.168.1.100 www.centos.vbird # 特别注意,这个档案的原本内容不要删除!只要新增额外的数据即可!

修改完毕之后要顺利启动的话,得要重新启动才可以。为什么需要重新启动呢? 因为系统已经有非常多的服务启动了, 这些服务如果需要主机名,都是到这个 档案去读取的。而我们知道配置文件更新过后,服务都得要重新启动才行。 因 此,已经启动而且有读到这个档案的服务,就得要重新启动啊!真麻烦~因此,

最简单的方法,就是重新启动。 但重开机之前还需要进行一项工作,否则,你 的系统开机会花掉很多时间喔!

[root@www ~]# hostname localhost.localdomain # 还是默认值,尚未更新成功!我们还得要进行底下的动作!

#检查五:看看你的主机名有没有对应的IP呢?没有的话,开机流程会很慢! [root@www ~]# ping -c 2 www.centos.vbird PING www.centos.vbird (192.168.1.100) 56(84) bytes of data. 64 bytes from www.centos.vbird (192.168.1.100): icmp\_seq=1 ttl=64

time=0.015 ms

64 bytes from www.centos.vbird (192.168.1.100): icmp\_seq=2 ttl=64

time=0.028 ms

--- www.centos.vbird ping statistics --- 2 packets transmitted, 2 received, 0% packet loss, time 1000ms rtt min/avg/max/mdev = 0.015/0.021/0.028/0.008 ms # 因为我们有设定 /etc/hosts 规定 www.centos.vbird 的 IP , # 所以才找的到主机主机名对应的正确 IP!这时才能够 reboot 喔!重要重

要!

上面的信息中,检查的内容总共有五个步骤,这五个步骤每一步都要成功后才能够继续 往下处理喔! 至于最重要的一点,当你修改过 /etc/sysconfig/network 里面的 HOSTNAME 后, 务必要重新启动 (reboot)。但是重新启动之前,请务必『 ping 主机 名』且得到 time 的响应才行!

### 抓包

tcpdump -A -i eth0 -nn 'port 8080'

在网络接口eth0 端口8080截包 内容用ascii显示

### 任意启动 TCP/UDP 封包的埠口联机: nc, netcat

nc 指令可以用来作为某些服务的检测,因为他可以连接到某个 port 来进行 沟通,此外,还可以自行启动一个 port 来倾听其他用户的联机

### 一张网卡设置多个IP

开启

ifconfig eth0:0 192.168.21.42 netmask 255.255.255.0 up

关闭

ifconfig eth0:0 192.168.21.42 netmask 255.255.255.0 down

### NTP

#### 国内常用NTP服务器地址及IP

210.72.145.44 (国家授时中心服务器IP地址)

133.100.11.8 日本 福冈大学

time-a.nist.gov 129.6.15.28 NIST, Gaithersburg, Maryland

time-b.nist.gov 129.6.15.29 NIST, Gaithersburg, Maryland

time-a.timefreq.bldrdoc.gov 132.163.4.101 NIST, Boulder, Colorado

time-b.timefreq.bldrdoc.gov 132.163.4.102 NIST, Boulder, Colorado

time-c.timefreq.bldrdoc.gov 132.163.4.103 NIST, Boulder, Colorado

utcnist.colorado.edu 128.138.140.44 University of Colorado, Boulder

time.nist.gov 192.43.244.18 NCAR, Boulder, Colorado

time-nw.nist.gov 131.107.1.10 Microsoft, Redmond, Washington

nist1.symmetricom.com 69.25.96.13 Symmetricom, San Jose, California

nist1-dc.glassey.com 216.200.93.8 Abovenet, Virginia

nist1-ny.glassey.com 208.184.49.9 Abovenet, New York City

nist1-sj.glassey.com 207.126.98.204 Abovenet, San Jose, California

nist1.aol-ca.truetime.com 207.200.81.113 TrueTime, AOL facility, Sunnyvale, California

nist1.aol-va.truetime.com 64.236.96.53 TrueTime, AOL facility, Virginia

————————————————————————————————————

ntp.sjtu.edu.cn 202.120.2.101 (上海交通大学网络中心NTP服务器地址）

s1a.time.edu.cn 北京邮电大学

s1b.time.edu.cn 清华大学

s1c.time.edu.cn 北京大学

s1d.time.edu.cn 东南大学

s1e.time.edu.cn 清华大学

s2a.time.edu.cn 清华大学

s2b.time.edu.cn 清华大学

s2c.time.edu.cn 北京邮电大学

s2d.time.edu.cn 西南地区网络中心

s2e.time.edu.cn 西北地区网络中心

s2f.time.edu.cn 东北地区网络中心

s2g.time.edu.cn 华东南地区网络中心

s2h.time.edu.cn 四川大学网络管理中心

s2j.time.edu.cn 大连理工大学网络中心

s2k.time.edu.cn CERNET桂林主节点

s2m.time.edu.cn 北京大学

## 打开核心的封包转递 (IP forward) 功能

就如同路由表是由 Linux 的核心功能所提供的,这个转递封包的能力也是 Linux 核心所提供, 那如何观察核心是否已经有启动封包转递呢?很简单啊,观察核心功能 的显示档案即可,如下所示:

[root@www ~]# cat /proc/sys/net/ipv4/ip\_forward

0 <== 0 代表没有启动, 1 代表启动了

要让该档案的内容变成启动值 1 最简单的方是就是使用:『echo 1 > /proc/sys/net/ipv4/ip\_forward』即可。 不过,这个设定结果在下次重新启动后就会 失效。因此,鸟哥建议您直接修改系统配置文件的内容,那就是 /etc/sysctl.conf 来 达成开机启动封包转递的功能喔。

sysctl 这个指令是在核心工作时用来直接修改核心参数的一个指令,更多的功能可以 参考 man sysctl 查询。 不要怀疑!只要这个动作,你的 Linux 就具有最简单的路由 器功能了。而由于 Linux 路由器的路由表设定方法的不同,通常路由器规划其路由的 方式就有两种:



[root@www ~]# vim /etc/sysctl.conf # 将底下这个设定值修改正确即可! (本来值为 0 ,将它改为 1 即可) net.ipv4.ip\_forward = 1

[root@www ~]# sysctl -p <==立刻让该设定生效



静态路由:直接以类似 route 这个指令来直接设定路由表到核心功能当中, 设定值只要与网域环境相符即可。 不过,当你的网域有变化时,路由器就得要 重新设定;

动态路由:透过类似 Quagga 或 zebra 软件的功能,这些软件可以安装在 Linux 路由器上, 而这些软件可以动态的侦测网域的变化,并直接修改 Linux 核心的路由表信息, 你无须手动以 route 来修改你的路由表信息喔!

## 允许某台主机通过ssh登入服务器

1. 修改 /etc/hosts.allow

sshd:192.168.1.4,192.168.2.0/255.255.255.0

1. 修改 /etc/hosts.deny

sshd:all

## SSH客户端软件

XShell、 Secure CRT、pietty 、putty

## FTP客户端软件

Filezilla

# JAVA

## 基准测试 JMH

<https://my.oschina.net/u/2561483/blog/755660>

http://openjdk.java.net/projects/code-tools/jmh/

## JAVA进程空间

http://blog.csdn.net/youyou1543724847/article/details/52738413

<http://docs.oracle.com/javase/tutorial/>

<http://docs.oracle.com/javase/8/docs/api/>

## Apache Commons工具集简介

<http://zhoualine.iteye.com/blog/1770014>

<http://www.oracle.com/technetwork/java/index.html>

<http://docs.oracle.com/javase/tutorial/tutorialLearningPaths.html>

<http://docs.oracle.com/javase/specs/jls/se8/html/jls-4.html#jls-4.3.4>

http://docs.oracle.com/javase/8/

<http://www.informit.com>

http://JavaWorld.com

http://DevSource.com

## Java SE 6 新特性: Instrumentation 新功能

<http://www.ibm.com/developerworks/cn/java/j-lo-jse61/>

java 库介绍

==========================Guava===================

Guava工程包含了若干被Google的 Java项目广泛依赖 的核心库，例如：集合 [collections] 、缓存 [caching] 、原生类型支持 [primitives support] 、并发库 [concurrency libraries] 、通用注解 [common annotations] 、字符串处理 [string processing] 、I/O 等等。 所有这些工具每天都在被Google的工程师应用在产品服务中。

==========================Jackson==================

Jackson可以轻松的将Java对象转换成json对象和xml文档，同样也可以将json、xml转换成Java对象。

前面有介绍过json-lib这个框架，在线博文：http://www.cnblogs.com/hoojo/archive/2011/04/21/2023805.html

相比json-lib框架，Jackson所依赖的jar包较少，简单易用并且性能也要相对高些。而且Jackson社区相对比较活跃，更新速度也比较快。

Gson

解析和生成jsxon工具类

snappy

Snappy 是一个 C++ 的用来压缩和解压缩的开发包。其目标不是最大限度压缩或者兼容其他压缩格式，而是旨在提供高速压缩速度和合理的压缩率。Snappy 比 zlib 更快，但文件相对要大 20% 到 100%。在 64位模式的 Core i7 处理器上，可达每秒 250~500兆的压缩速度。

Curator

Curator是Netflix开源的一套ZooKeeper客户端框架. Netflix在使用ZooKeeper的过程中发现ZooKeeper自带的客户端太底层, 应用方在使用的时候需要自己处理很多事情, 于是在它的基础上包装了一下, 提供了一套更好用的客户端框架.

xmlenc

The xmlenc library is a fast stream-based XML output library for Java. Main design goals are performance, simplicitity and pureness. As far as known, xmlenc is the fastest XML output library for Java.

Paranamer

Paranamer是一个Java开源类库能够在运行期读取Java类中非私有方法与构造函数中包含的参数名。

Jersey

Jersey是JAX-RS（JSR311）开源参考实现用于构建RESTful Web service，它包含三个部分：

　　核心服务器（Core Server） 通过提供JSR 311中标准化的注释和API标准化，可以用直观的方式开发RESTful Web服务。

　　核心客户端（Core Client） Jersey客户端API能够帮助开发者与RESTful服务轻松通信；

　　集成（Integration） Jersey还提供可以轻松继承Spring、Guice、Apache Abdera的库

<http://www.cnblogs.com/xinsheng/p/3897289.html>

asn1

该工具可根据ASN.1协议描述文件,生成对应的java类

JAXB

JAXB（Java API for XML Binding），提供了一个快速便捷的方式将Java对象与XML进行转换。在JAX-WS（Java的WebService规范之一）中，JDK1.6 自带的版本JAX-WS2.1，其底层支持就是JAXB。

commons-math3

Section 1 linear 线性代数（矩阵为主）

1） Vector 向量

2） Matrix 矩阵

3） Matrix Decomposition 矩阵分解

Section 2 analysis 数学分析（函数为主）

1） Function 函数

2） Polynomial 多项式函数

3） Interpolation 插值

4） Integration 积分

5） Solver 求解

Section 3 Probabilityand Statistics 概率和统计

1）distribution 分布

2）fraction and complex 分数和复数

3）random and statistics 随机生成和统计初步

4）cluster and regression聚类和回归

Mockito：

一个强大的用于 Java 开发的模拟测试框架

<http://www.oschina.net/translate/mockito-a-great-mock-framework-for-java-development>

Xalan

xalan是一套xslt处理器（有C和JAVA语言两种版本），用来将XML文件转换为HTML,TEXT和XML等其他类型文件格式。支持XSLT1.0和XPATH 1.0版。开发人员可以通过命令行方式或在JAVA APPLET和SERVLET中使用，并可以作为自己开发的应用程序的类库使用。xalan-java实现的是transformation API for XML(TRaX)接口，此接口为jaxp1.2标准中的一部分。

spark 编译

build/mvn -T 8 -Pyarn -Phadoop-2.6 -Dhadoop.version=2.6.2 -DskipTests clean package

./make-distribution.sh —name spark1.5.2-hadoop-2.7-bin —tgz -Phadoop-2.7 –Pyarn

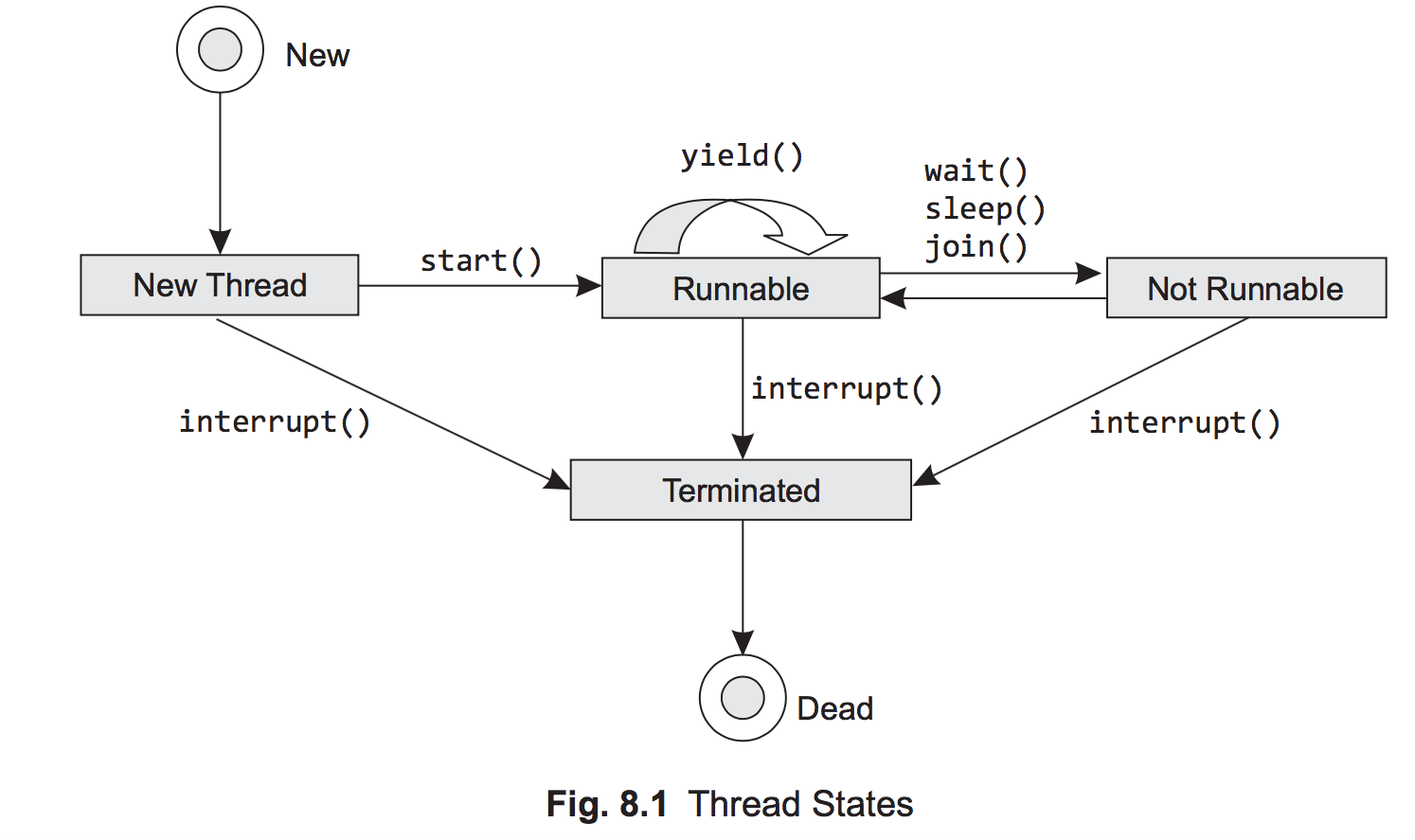
g++ --std=c++11 -o client \*.cpp –lthrift

## Restful

### SpringMVC简单构造restful, 并返回json

http://my.oschina.net/u/2272916/blog/352297

## 线程



The above statement is responsible for creating a new Thread object. In ‘New’ state, no system resource (such as CPU) is allotted to the newly born Thread object. From this state, the thread can either be started (by using start() of Thread class) or stopped (by using interrupt() of Thread class), thus moving to ‘Runnable’ or ‘Terminated’ state, respectively. No other method apart from start() and interrupt() can be called from this state and if tried to do so, it would cause an exception, IllegalThreadStateException.

**Runnable** In this state, a thread is ready for execution by the JVM. It represents the running state of the thread, as well. Ready state of a thread can be defined as it is ready for execution but it might be in the queue, waiting for the operating system to provide it the required resource, like processor. Once a thread is actually being executed by the processor then it is termed as “Running”. From ‘New’ state the thread might move to the ‘Runnable’ state on execution of the following statements:

Thread threadObj = new ThreadDemo();

threadObj.start();

As soon as start() is called, the thread is allotted the system resource as per the scheduling done by the Java Runtime Environment. Now the thread has entered into the runnable state. In Fig. 8.1, no differentiation is made between a running thread and a runnable thread. Even the running threads are made a part of the runnable state. But there is a difference between the two. A running thread is the one which is being executed by the processor. Such a thread can be called as the *current thread*. Runnable threads are those which are not actually running, but are scheduled in queue to get the processor. The scheduling scheme, under which all the

runnable threads are prioritized for sharing the processor, is implemented by the Java Runtime system. However, when a thread moves to ‘Running’ from ‘Runnable’, the instructions of the run() method are being executed sequentially. During this phase the processor can be forced to relinquish its control over the thread, thus forcing it to be a part of the queue again by the use of yield() method as shown in Fig. 8.1.

**Not Runnable** From runnable state, a thread might move to the not runnable state, as shown in Fig. 8.1. This state is just a hypothetical state used by us to categorize the three valid states of Java. A thread which is in any of these three states can be assumed to be in ‘not runnable’ state. These three states are WAITING, TIMED\_WAITING, and BLOCKED.

***Waiting*** In this state, a thread is waiting indefinitely for another thread to perform a particular action (i.e., notify). Threads can move into this state either by calling the methods Object.wait() (without time out) or Thread.join() (without time out).

***Timed\_Waiting*** Inthisstate,thethreadiswaitingforanotherthreadtoperformanaction(notify) up to a specified waiting time. A thread can get into this state by calling either of these methods: Thread.sleep(), Object.wait(), and Thread.join() (all these methods should be called with time out specified).

***Blocked*** In this state, a resource cannot be accessed because it is being used by another thread. A thread can get into this state by calling Object.wait() method.

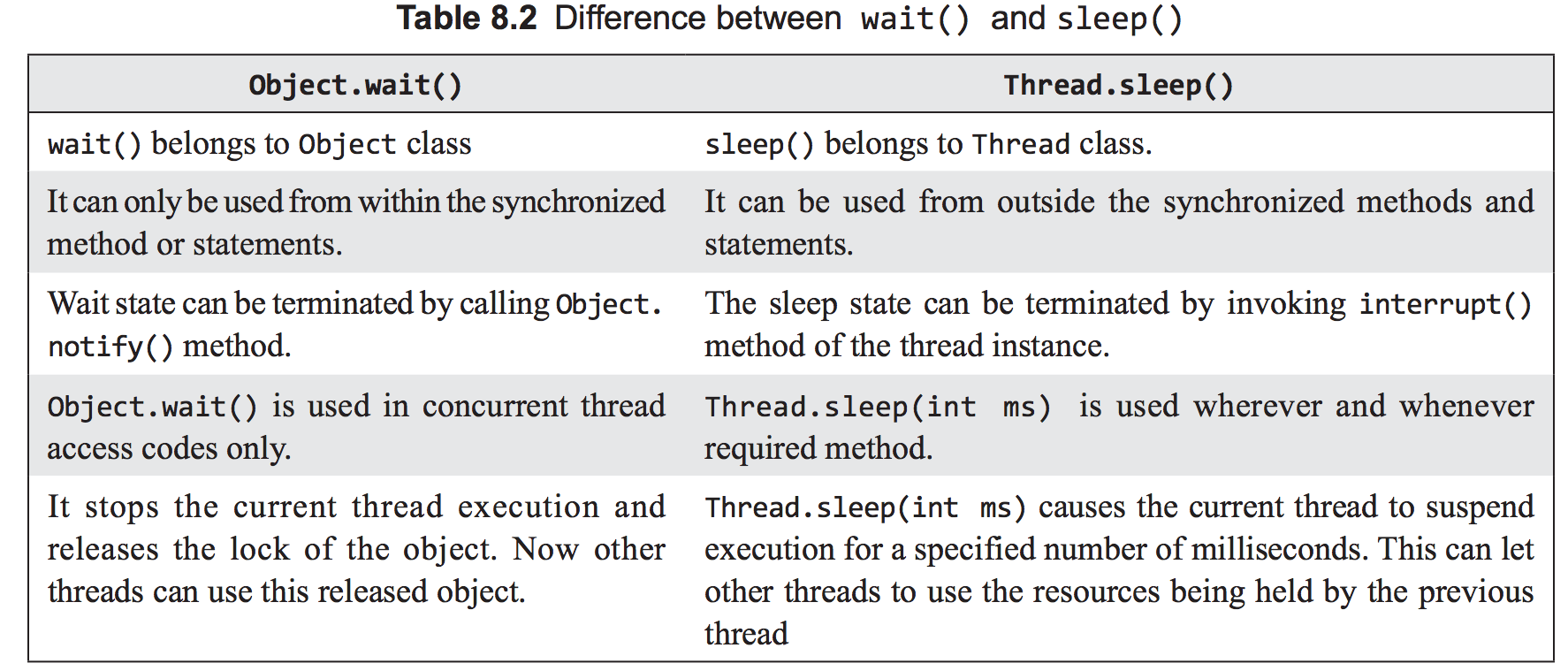
Before proceeding further, we must discuss the concept of monitors in Java. This is taken up in greater detail in Section 8.8. Monitor is an object that is a mutually exclusive lock on the resource to be accessed. A monitor can be owned by only one thread at a time. When a thread calls Object.wait() method, it releases all the acquired monitors and is put into WAITING state, until some other thread enters the same monitor and calls notify()/notifyAll(). When notify() is called, it wakes up a thread that called wait() on the same object. The method notifyAll() will wake up all the threads that called wait() on the same object. The difference between two methods is that, if notify() is used, then only one thread (selected by the JVM scheduler) is granted the monitor and all other threads are put into BLOCKED state, whereas if you use notifyAll(), it wakes up all the threads and puts them into ready state. The threads that can execute, start executing, and the rest move into the waiting state.The three methods mentioned above are final methods of the ObjectClass, so all classes have them.

final void wait() throws InterruptedException;

final void notify()

final void notifyAll()

Additional form of wait() where time can be specified for the thread to wait for that period, is also available. It puts the thread in TIMED\_WAITING state. We can easily figure out that a WAITING state thread will always be dependent on an action performed by some other thread, whereas a thread in TIMED\_WAITING is not completely dependent on an action performed by some other thread, as in this case, the wait ends automatically after the completion of the time out period. Similarly, if a thread has put itself into WAITING state by calling Thread.join() method, then it will keep waiting until the specified thread terminates or the specified time elapses. There seems to be no difference between sleep() and wait() as both of them do the same job of making a thread wait for a specified time. The differences between the start methods have been specified in Table 8.2.



## java7

### 锁：

#### java 偏向锁

http://blog.163.com/silver9886@126/blog/static/35971862201472274958280/

### 资源释放

try (resources to be used and automatically released)

{

// statements within the block

}

For example

try (abc a=new abc(); pqr p=new pqr())

{

// statements within the block

}

try 语句执行结束后，会自动调用 AutoCloseable 接口的 close方法,无论 statements block 是否抛出异常

More than one AutoCloseable resources can be used in try-with-resources statement separated by semicolon. Hence it is mandatory for abc and pqr objects to implement the AutoCloseable interface as shown below in the example. The resources created in the try-with-resources statement are closed in the reverse order of creation

## memcache

https://github.com/gwhalin/Memcached-Java-Client/downloads

## Zero Copy

<http://www.ibm.com/developerworks/linux/library/j-zerocopy>

## Lambda

<http://ifeve.com/lambda/>

## Stream

<http://ifeve.com/stream/>

## HttpClient

<http://hc.apache.org/httpcomponents-client-4.3.x/tutorial/html/connmgmt.html>

## 弱引用WeakReference

java.io ObjectOutputStream.java

http://www.thinksaas.cn/topics/0/102/102618.html

## JIT

### Optimizing Startup

1. The client compiler is most useful when the startup of an appli‐ cation is the overriding performance concern.

2. Tiered compilation can achieve startup times very close to those obtained from the client compiler.

### Optimizing Batch Operations

1. For jobs that run in a fixed amount of time, choose the compil‐ er based on which one is the fastest at executing the actual job.

2. Tieredcompilationprovidesareasonabledefaultchoiceforbatch jobs.

### Optimizing Long-Running Applications

For long-running applications, always choose the server compiler, preferably in conjunction with tiered compilation.

### Java and JIT Compiler Versions

• A 32-bit client version (-client)

• A 32-bit server version (-server)

• A 64-bit server version (-d64)

#### 32-Bit or 64-Bit?

If you have a 32-bit operating system, then you must use a 32-bit version of the JVM. If you have a 64-bit operating system, then you can choose to use either the 32- or 64-bit version of Java. Don’t assume that just because you have a 64-bit operating system, you must also use a 64-bit version of Java.

If the size of your heap will be less than about 3 GB, the 32-bit version of Java will be faster and have a smaller footprint. This is because the memory references within the JVM will be only 32 bits, and manipulating those memory references is less expensive than manipulating 64-bit references (even if you have a 64-bit CPU). The 32-bit refer‐ ences also use less memory.

Chapter 8 discusses compressed *ordinary object pointers* (oops), which is a way that the JVM can use 32-bit addresses even within the 64-bit JVM. However, even with that optimization, the 64-bit JVM will have a larger footprint because the native code it uses will still have 64-bit addresses.

The downside to the 32-bit JVM is that the total process size must be less than 4 GB (3 GB on some versions of Windows, and 3.5 GB on some old versions of Linux). That includes the heap, permgen, and the native code and native memory the JVM uses. Programs that make extensive use of long or double variables will be slower on a 32- bit JVM because they cannot use the CPU’s 64-bit registers, though that is a very ex‐ ceptional case.

Programs that fit within a 32-bit address space will run anywhere between 5% and 20% faster in a 32-bit JVM than a similarly configured 64-bit JVM. The stock batching pro‐ gram discussed earlier in this chapter, for example, is 20% faster when run on a 32-bit JVM on my desktop.

One last complication: for the sake of compatibility, the argument specifying which compiler to use is not rigorously followed. If you have a 64-bit JVM and specify -client, the application will use the 64-bit server compiler anyway. If you have a 32- bit JVM and you specify -d64, you will get an error that the given instance does not support a 64-bit JVM.

OSR(OnStackReplace) trigger = (CompileThreshold \*

((OnStackReplacePercentage - InterpreterProfilePercentage)/100))

In all compilers, the default value of the -XX:InterpreterProfilePercentage=*N* flag is 33. In the client compiler, the default value of the -XX:OnStackReplacePercentage=*N* flag is 933, and so the client compiler requires that the back-edge counter hit 13,500 before it will begin OSR compilation. In the server compiler, OSR compilation begins when the back-edge counter hits 10,700, since the default value for OnStackReplace Percentage is 140.

查看编译器信息

Seeing the compilation log requires that the program be started with the -XX:+PrintCompilation flag. If the program was started without that flag, you can get some limited visibility into the working of the compiler by using jstat.

jstat has two options to provide information about the compiler. The -compiler option supplies summary information about how many methods have been compiled (here 5003 is the process ID of the program to be inspected):

% **jstat -compiler 5003**

Compiled Failed Invalid Time FailedType FailedMethod

206 0 0 1.97 0

Note this also lists the number of methods that failed to compile and the name of the last method that failed to compile; if profiles or other information lead you to suspect that a method is slow because it hasn’t been compiled, this is an easy way to verify that hypothesis.

Alternately, you can use the -printcompilation option to get information about the last method that is compiled. Because jstat takes an optional argument to repeat its operation, you can see over time which methods are being compiled. In this example, jstat repeats the information for process ID 5003 every second (1,000 ms):

% **jstat -printcompilation 5003 1000**

Compiled Size Type Method

207 64 1 java/lang/CharacterDataLatin1 toUpperCase

208 5 1 java/math/BigDecimal$StringBuilderHelper getCharArray

The number of compiler threads (for all three compiler options) can be adjusted by setting the -XX:CICompilerCount=*N* flag (with a default value given in the previous table). That is the total number of threads the JVM will use to process the queue(s); for tiered compilation, one-third of them (but at least one) will be used to process the client compiler queue, and the remaining threads (and also at least one) will be used to process the server compiler queue.

One other setting that applies to the compilation threads is the value of the -XX:+BackgroundCompilation flag, which by default is true. That setting means that the queue is processed asynchronously as just described. But that flag can be set to false, in which case when a method is eligible for compilation, code that wants to execute it will wait until it is in fact compiled (rather than continuing to execute in the interpreter). Background compilation is also disabled when -Xbatch is specified.

### Escape Analysis

The server compiler performs some very aggressive optimizations if escape analysis is enabled (-XX:+DoEscapeAnalysis, which is true by default). For example, consider this class to work with factorials:

**public class Factorial** { **private** BigInteger factorial; **private int** n; **public** Factorial(**int** n) {

**this**.n = n; }

**public synchronized** BigInteger getFactorial() { **if** (factorial == **null**)

factorial = ...; **return** factorial;

} }

To store the first 100 factorial values in an array, this code would be used:

ArrayList<BigInteger> list = **new** ArrayList<BigInteger>(); **for**(**int**i=0;i<100;i++){

Factorial factorial = **new** Factorial(i);

list.add(factorial.getFactorial());

}

The factorial object is referenced only inside that loop; no other code can ever access that object. Hence, the JVM is free to perform a number of optimizations on that object:

* It needn’t get a synchronization lock when calling the getFactorial() method.
* It needn’t store the field n in memory; it can keep that value in a register. Similarly  it can store the factorial object reference in a register.
* In fact, it needn’t allocate an actual factorial object at all; it can just keep track of  the individual fields of the object.
* Escape analysis is enabled by default. In rare cases, it will get things wrong, in which case disabling it will lead to faster and/or more stable code. If you find this to be the case, then simplifying the code in question is the best course of action: simpler code will compile better. (It is a bug, however, and should be reported.)
* Escapeanalysisisthemostsophisticatedoftheoptimizationsthe compiler can perform. This is the kind of optimization that fre‐ quently causes microbenchmarks to go awry.
* Escape analysis can often introduce “bugs” into improperly synchronized code.

## Garbage Collector

-server -XX:-TieredCompilation -Xmx70m -Xms50m -XX:-PrintStringTableStatistics -XX:+PrintGCDetails -XX:+PrintReferenceGC -XX:-PrintCompilation -Dabc=abc -XX:-UseG1GC -XX:+PrintTenuringDistribution -XX:+PrintAdaptiveSizePolicy -Xss256k -Xmn2500m

### minor GC

Objects are first allocated in the young generation, which is some subset of the entire heap. When the young generation fills up, the garbage collector will stop all the application threads and empty out the young generation. Objects that are no longer in use are discarded, and objects that are still in use are moved elsewhere. This operation is called a minor GC.

### 垃圾分代的优势

There are two performance advantages to this design. First, because the young gener‐ ation is only a portion of the entire heap, processing it is faster than processing the entire heap. This means that the application threads are stopped for a much shorter period of time than if the entire heap were processed at once. You probably see a trade-off there, since it also means that the application threads are stopped more frequently than they would be if the JVM waited to perform GC until the entire heap were full; that trade- off will be explored in more detail later in this chapter. For now, though, it is almost always a big advantage to have the shorter pauses even though they will be more frequent.

The second advantage arises from the way objects are allocated in the young generation. Objects are allocated in eden (which comprises the vast majority of the young genera‐ tion). When the young generation is cleared during a collection, all objects in eden are either moved or discarded: all live objects are moved either to one of the survivor spaces or to the old generation. Since all objects are moved, the young generation is automat‐ ically compacted when it is collected.

All GC algorithms have stop-the-world pauses during collection of the young genera‐ tion.

### Full GC

As objects are moved to the old generation, eventually it too will fill up, and the JVM will need to find any objects within the old generation that are no longer in use and discard them. This is where GC algorithms have their biggest differences. The simpler algorithms stop all application threads, find the unused objects and free their memory, and then compact the heap. This process is called a full GC, and it generally causes a long pause for the application threads.

On the other hand, it is possible—though more computationally complex—to find un‐ used objects while application threads are running; CMS and G1 both take that ap‐ proach. Because the phase where they scan for unused objects can occur without stop‐ ping application threads, CMS and G1 are called concurrent collectors. They are also called low-pause (and sometimes—incorrectly—pauseless) collectors, since they min‐ imize the need to stop all the application threads. Concurrent collectors also take dif‐ ferent approaches to compacting the old generation.

When using the CMS or G1 collector, an application will typically experience fewer (and much shorter) pauses. The trade-off is that the application will use more CPU overall.

CMS and G1 may also perform a long, full GC pause (and avoiding those is one of the key factors to consider when tuning those algorithms).

### 例子

As you consider which garbage collector is appropriate for your situation, think about the overall performance goals that must be met. There are trade-offs in every situa‐ tion. In an application (such as a Java EE server) measuring the response time of indi‐ vidual requests, consider these points:

* The individual requests will be impacted by pause times—and more importantly by long pause times for full GCs. If minimizing the effect of pauses on response times is the goal, a concurrent collector will be more appropriate.
* If the average response time is more important than the outliers (i.e., the 90th% response time), the throughput collector will usually yield better results.
* The benefit of avoiding long pause times with a concurrent collector comes at the expense of extra CPU usage.
* Similarly, the choice of garbage collector in a batch application is guided by the following trade-off:
  1. If enough CPU is available, using the concurrent collector to avoid full GC pauses will allow the job to finish faster.
  2. If CPU is limited, then the extra CPU consumption of the concurrent collector will cause the batch job to take more time.

### The serial garbage collector

The serial collector uses a single thread to process the heap. It will stop all application threads as the heap is processed (for either a minor or full GC). During a full GC, it will fully compact the old generation.

The serial collector is enabled by using the -XX:+UseSerialGC flag (though usually it is the default in those cases where it might be used). Note that unlike with most JVM flags, the serial collector is not disabled by changing the plus sign to a minus sign (i.e., by specifying -XX:-UseSerialGC). On systems where the serial collector is the default, it is disabled by specifying a different GC algorithm.

### The throughput collector

This is the default collector for server-class machines (multi-CPU Unix machines, and any 64-bit JVM).

The throughput collector uses multiple threads to collect the young generation, which makes minor GCs much faster than when the serial collector is used. The throughput collector can use multiple threads to process the old generation as well. That is the default behavior in JDK 7u4 and later releases, and that behavior can be enabled in earlier JDK 7 JVMs by specifying the -XX:+UseParallelOldGC flag. Because it uses multiple threads, the throughput collector is often called the parallel collector.

The throughput collector stops all application threads during both minor and full GCs, and it fully compacts the old generation during a full GC. Since it is the default in most situations where it would be used, it needn’t be explicitly enabled. To enable it where necessary, use the flags -XX:+UseParallelGC -XX:+UseParallelOldGC.

### The CMS collector

The CMS collector is designed to eliminate the long pauses associated with the full GC cycles of the throughput and serial collectors. CMS stops all application threads during a minor GC, which it also performs with multiple threads. Notably, though, CMS uses a different algorithm to collect the young generation (-XX:+UseParNewGC) than the throughput collector uses (-XX:+UseParallelGC).

Instead of stopping the application threads during a full GC, CMS uses one or more background threads to periodically scan through the old generation and discard unused objects. This makes CMS a low-pause collector: application threads are only paused during minor collections, and for some very short periods of time at certain points as the background threads scan the old generation. The overall amount of time that ap‐ plication threads are stopped is much less than with the throughput collector.

The trade-off here comes with increased CPU usage: there must be adequate CPU available for the background GC thread(s) to scan the heap at the same time the appli‐ cation threads are running. In addition, the background threads do not perform any compaction, which means that the heap can become fragmented. If the CMS back‐ ground threads don’t get enough CPU to complete their tasks, or if the heap becomes too fragmented to allocate an object, CMS reverts to the behavior of the serial collector: it stops all application threads in order to clean and compact the old generation using a single thread. Then it begins its concurrent, background processing again (until, pos‐ sibly, the next time the heap becomes too fragmented).

CMS is enabled by specifying the flags -XX:+UseConcMarkSweepGC -XX:+UseParNewGC (both of which are false by default).

### The G1 collector

The G1 (or Garbage First) collector is designed to process large heaps (greater than about 4 GB) with minimal pauses. It divides the heap into a number of regions, but it is still a generational collector. Some number of those regions comprise the young gen‐ eration, and the young generation is still collected by stopping all application threads and moving all objects that are alive into the old generation or the survivor spaces. As in the other algorithms, this occurs using multiple threads.

G1 is a concurrent collector: the old generation is processed by background threads that don’t need to stop the application threads to perform most of their work. Because the old generation is divided into regions, G1 can clean up objects from the old generation by copying from one region into another, which means that it (at least partially) com‐ pacts the heap during normal processing. Hence, a G1 heap is much less likely to be subject to fragmentation—though that is still possible.

Like CMS, the trade-off for avoiding the full GC cycles is CPU time: the multiple back‐ ground threads must have CPU cycles available at the same time the application threads are running. G1 is enabled by specifying the flag -XX:+UseG1GC (which by default is false).

#### Causing and Disabling Explicit Garbage Collection

GC is typically caused when the JVM decides GC is necessary: a minor GC will be triggered when the new generation is full, a full GC will be triggered when the old generation is full, or a concurrent GC (if applicable) will be triggered when the heap starts to fill up.

##### 强制GC

Java provides a mechanism for applications to force a GC to occur: the System.gc() method. Calling that method is almost always a bad idea. This call always triggers a full GC (even if the JVM is running with CMS or G1), so application threads will be stopped for a relatively long period of time. And calling this method will not make the application any more efficient; it will cause a GC to occur sooner than might have happened other‐ wise, but that is really just shifting the performance impact.

There are exceptions to every rule, particularly when doing performance monitoring or benchmarking. For small benchmarks that run a bunch of code to properly warm up the JVM, forcing a GC before the measurement cycle may make sense. Similarly when doing heap analysis, it is usually a good idea to force a full GC before taking the heap dump. Most techniques to obtain a heap dump will perform a full GC anyway, but there are also other ways you can force a full GC: you can execute **jcmd <process id> GC.run**, or you can connect to the JVM using jconsole and click the Perform GC button in the Memory panel.

Another exception is RMI, which calls System.gc() every hour as part of its distributed garbage collector. That timing can be changed by setting a different value for these two system properties: -Dsun.rmi.dgc.server.gcInterval=*N* and -Dsun.rmi.dgc.cli ent.gcInterval=*N*. The values for N are in milliseconds, and the default value in Java 7 (which is changed from earlier releases) is 3600000 (one hour).

##### 阻止强制GC

If you end up running third-party code that incorrectly calls the System.gc() method, those GCs can be prevented entirely by including -XX:+DisableExplicitGC in the JVM arguments; by default that flag is false.

* The four available GC algorithms take different approaches to‐ ward minimizing the effect of GC on an application.
* The serial collector makes sense (and is the default) when only one CPU is available and extra GC threads would interfere with the application.
* The throughput collector is the default on other machines; it maximizes the total throughput of an application but may sub‐ ject individual operations to long pauses.
* The CMS collector can concurrently collect the old generation while application threads are running. If enough CPU is avail‐ able for its background processing, this can avoid full GC cy‐ cles for the application.
* The G1 collector also concurrently collects the old generation while application threads are running, potentially avoiding full GCs. Its design makes it less likely to experience full GCs than CMS.

##### GC 算法选择

These are the sort of trade-offs to consider when deciding which GC algorithm suits your performance goals. If the average time is all you care about, then the throughput collector will likely look similar to a concurrent collector, and you can consider the CPU usage instead (in which case, the throughput collector will be the better choice). If you’re interested in the 90th% or other percentile-based response times, then only testing can see where those line up with the number of full GC cycles the application needs to perform its job.

* When measuring response time or throughput, the choice be‐ tween throughput and concurrent collectors is dependent on the amount of CPU available for the background concurrent threads to run.
* The throughput collector will frequently have a lower average response time than a concurrent collector, but the concurrent collector will often have a lower 90th% or 99th% response time.
* When the throughput collector performs excessive full GCs, a concurrent collector will often have lower average response times.

##### Choosing between CMS and G1

CMS is expected to outperform G1 for heaps that are smaller than 4 GB

当堆小于4G时CMS预期超过G1

When large heaps are used, G1 will usually be better than CMS because of the manner in which it can divide work.

###### CMS 工作流程

CMS background thread(s) must scan the entire old generation before any objects can be freed. The time to scan the heap is obviously dependent on the size of the heap. If the CMS background thread does not finish scanning the heap and freeing objects before the heap fills up, CMS will experience a concurrent mode failure: at that point, CMS has to revert to doing a full GC with all application threads stopped. That full GC is done only with a single thread, making it a very severe performance penalty. CMS can be tuned to utilize multiple background threads to minimize that change, but the larger the heap grows, the more work those CMS threads have to do. (The chance that CMS experiences a concurrent mode failure also depends on the amount of allocation that the application does.)

###### G1工作流程

G1, on the other hand, segments the old generation into regions, so it is easier for multiple background threads to divide the necessary work of scanning the old genera‐ tion. G1 can still experience concurrent mode failures if the background threads can’t keep up, but the G1 algorithm makes that less likely to occur.

CMS can also revert to a full GC because of heap fragmentation, since CMS does not compact the heap (except during the lengthy full GCs). G1 compacts the heap as it goes.

CMS因为堆碎片问题，恢复full GC,，此后CMS没有压缩堆。而G1可以继续压缩。

G1 can still experience heap fragmentation, but its design again reduces the chance of that compared to CMS.

1. CMS is the better of the concurrent collectors when the heap is small.

2. G1isdesignedtoprocesstheheapinregions,soitwillscalebetter than CMS on large heaps.

### Avoiding concurrent mode failures

* Avoiding concurrent mode failures is the key to achieving the best possible performance with CMS.
* The simplest way to avoid those failures (when possible) is to increase the size of the heap.
* Otherwise, the next step is to start the concurrent background threads sooner by adjusting the CMSInitiatingOccupancyFrac tion.
* Tuning the number of background threads can also help.

**Running the background thread more often**

* One way to let CMS win the race is to start the concurrent cycle sooner. If the concurrent cycle starts when 60% of the old generation is filled, CMS has a better chance of finishing than if the cycle starts when 70% of the old generation is filled. The easiest way to achieve that is to set both these flags: -XX:CMSInitiatingOccupancyFraction=*N* and -XX:+UseCMSInitiatingOccupancyOnly.

**Adjusting the CMS background threads**

Each CMS background thread will consume 100% of a CPU on a machine. If an appli‐ cation experiences a concurrent mode failure and there are extra CPU cycles available, the number of those background threads can be increased by setting the -XX:ConcGCThreads=*N* flag. By default, that value is set based on the value of the ParallelGCThreads flag:

ConcGCThreads = (3 + ParallelGCThreads) / 4

This calculation is performed using integer arithmetic, which means there will be one ConcGCThread for up to four ParallelGCThreads, two ConcGCThreads for between five and eight ParallelGCThreads, and so on.

### Tuning G1

These are the options to prevent a full GC:

* Increase the size of the old generation either by increasing the heap space overall or by adjusting the ratio between the generations.
* Increase the number of background threads (assuming there is sufficient CPU).
* Perform G1 background activities more frequently.
* Increase the amount of work done in mixed GC cycles.

There are a lot of tunings that can be applied here, but one of the goals of G1 is that it shouldn’t have to be tuned that much. To that end, G1 is primarily tuned via a single flag: the same -XX:MaxGCPauseMillis=*N* flag that was used to tune the throughput collector.

#### Tuning the G1 background threads

* To have G1 win its race, try increasing the number of background marking threads (assuming there is sufficient CPU available on the machine).  Tuning the G1 threads is similar to tuning the CMS threads: the ParallelGCThreads option affects the number of threads used for phases when application threads are
* stopped, and the ConcGCThreads flag affects the number of threads used for concurrent phases. The default value for ConcGCThreads is different in G1, however. It is defined as:
* ConcGCThreads = (ParallelGCThreads + 2) / 4
* The arithmetic here is still integer-based; G1 simply increases that value one step later than CMS.

#### Tuning G1 to run more (or less) frequently

G1 can also win its race if it starts collecting earlier. The G1 cycle begins when the heap hits the occupancy ratio specified by -XX:InitiatingHeapOccupancyPercent=*N*, which has a default value of 45. Note that unlike CMS, that setting is based on the usage of the entire heap, not just the old generation.

1. G1tuningshouldbeginbysettingareasonablepausetimetarget.

2. If full GCs are still an issue after that and the heap size cannot be increased, specific tunings can be applied for specific failures.

a. To make the background threads run more frequently, ad‐ just the InitiatingHeapOccupancyPercent.

b. If additional CPU is available, adjust the number of threads via the ConcGCThreads flag.

c. To prevent promotion failures, decrease the size of the G1MixedGCCountTarget.

#### Tuning G1 mixed GC cycles

After a concurrent cycle, G1 cannot begin a new concurrent cycle until all previously marked regions in the old generation have been collected. So another way to make G1 start a marking cycle earlier is to process more regions in a mixed GC cycle (so that there will end up being fewer mixed GC cycles).

The amount of work a mixed GC does is dependent on three factors. The first is how many regions were found to be mostly garbage in the first place. There is no way to directly affect that: a region is declared eligible for collection during a mixed GC if it is 35% garbage. (It is likely this value will become a tunable parameter at some point; the experimental name for the parameter (available in experiment builds of the open source code) is -XX:G1MixedGCLiveThresholdPercent=*N*.)

The second factor is the maximum number of mixed GC cycles over which G1 will process those regions, which is specified by the value of the flag -XX:G1MixedGCCount Target=*N*. The default value for that is 8; reducing that value can help overcome pro‐ motion failures (at the expense of longer pause times during the mixed GC cycle).

On the other hand, if mixed GC pause times are too long, that value can be increased so that less work is done during the mixed GC. Just be sure that increasing that number does not delay the next G1 concurrent cycle too long, or a concurrent mode failure may result.

Finally, the third factor is the maximum desired length of a GC pause (i.e., the value specified by MaxGCPauseMillis). The number of mixed cycles specified by the G1MixedGCCountTarget flag is an upper bound; if time is available within the pause target, G1 will collect more than one-eighth (or whatever value has been specified) of the marked old generation regions. Increasing the value of the MaxGCPauseMillis flag allows more old generation regions to be collected during each mixed GC, which in turn can allow G1 to begin the concurrent cycle sooner.

#### G1调整总结

1. G1tuningshouldbeginbysettingareasonablepausetimetarget.

2. If full GCs are still an issue after that and the heap size cannot be increased, specific tunings can be applied for specific failures.

a. To make the background threads run more frequently, ad‐ just the InitiatingHeapOccupancyPercent.

b. If additional CPU is available, adjust the number of threads via the ConcGCThreads flag.

c. To prevent promotion failures, decrease the size of the G1MixedGCCountTarget.

**Always and Never Tenure**

The tenuring threshold will always take on some range between 1 and MaxTenuringThreshold. Even if the JVM is started with an initial tenuring threshold equal to the maximum tenuring threshold, the JVM may decrease that value.

There are two flags that can circumvent that behavior at either extreme. If you know that objects that survive a young collection will always be around for a long time, you can specify -XX:+AlwaysTenure (by default, false), which is essentially the same as setting the MaxTenuringThreshold to 0. This is a very, very rare situation; it means that objects will always be promoted to the old generation rather than stored in a survivor space.

The second flag is -XX:+NeverTenure (also false by default). This flag affects two things: it behaves as if the initial and max tenuring thresholds are infinity, and it prevents the JVM from adjusting that threshold down. In other words, as long as there is room in the survivor space, no object will ever be promoted to the old generation.

### Out of Memory Errors

The JVM throws an out of memory error under these circumstances:

* No native memory is available for the JVM.
* The permgen (in Java 7 and earlier) or metaspace (in Java 8) is out of memory.
* TheJavaheapitselfisoutofmemory:theapplicationhastoomanyliveobjectsfor the given heap size.
* The JVM is spending too much time performing GC.

### Automatic Heap Dumps

Out of memory errors can occur unpredictably, making it difficult to know when to get a heap dump. There are several JVM flags that can help.

-XX:+HeapDumpOnOutOfMemoryError Turning on this flag (which is false by default) will cause the JVM to create a heap dump whenever an out of memory error is thrown.

-XX:HeapDumpPath=<path>

This specifies the location where the heap dump will be written; the default is *java\_pid<pid>.hprof* in the application’s current working directory. The path can specify either a directory (in which case the default file name is used), or the name of the actual file to produce.

-XX:+HeapDumpAfterFullGC

This generates a heap dump after running a full GC.

-XX:+HeapDumpBeforeFullGC

This generates a heap dump before running a full GC.

In the case where multiple heap dumps are generated (e.g., because multiple full GCs occur), a sequence number is appended to the heap dump filename.

Try turning on these flags if the application unpredictably throws an out of memory error due to the heap space, and you need the heap dump at that point to analyze why the failure occurred.

## 堆转储Heap Dump

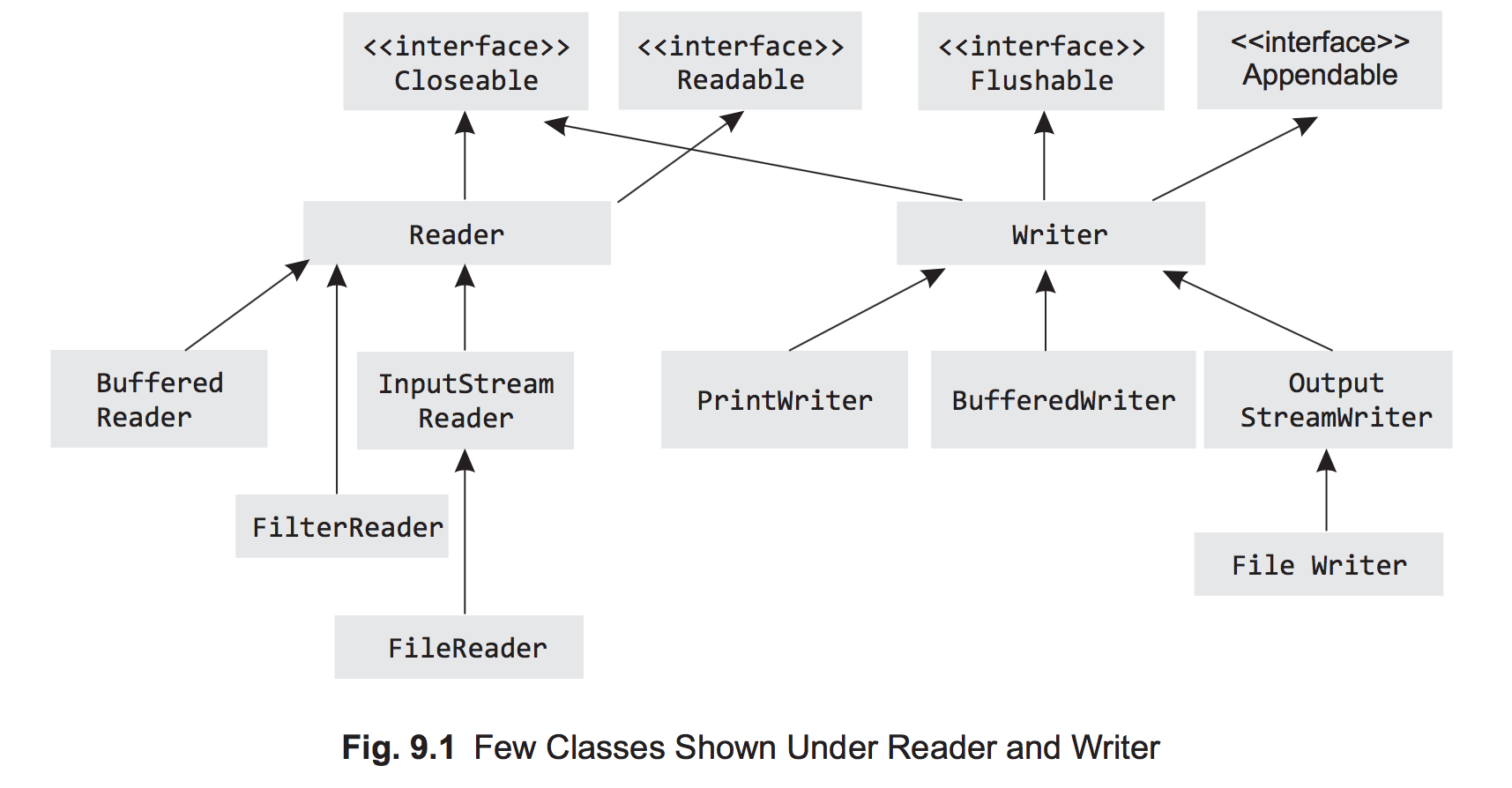
jcmd 21149 GC.heap\_dump /home/hdfs/heap\_dump.hprof

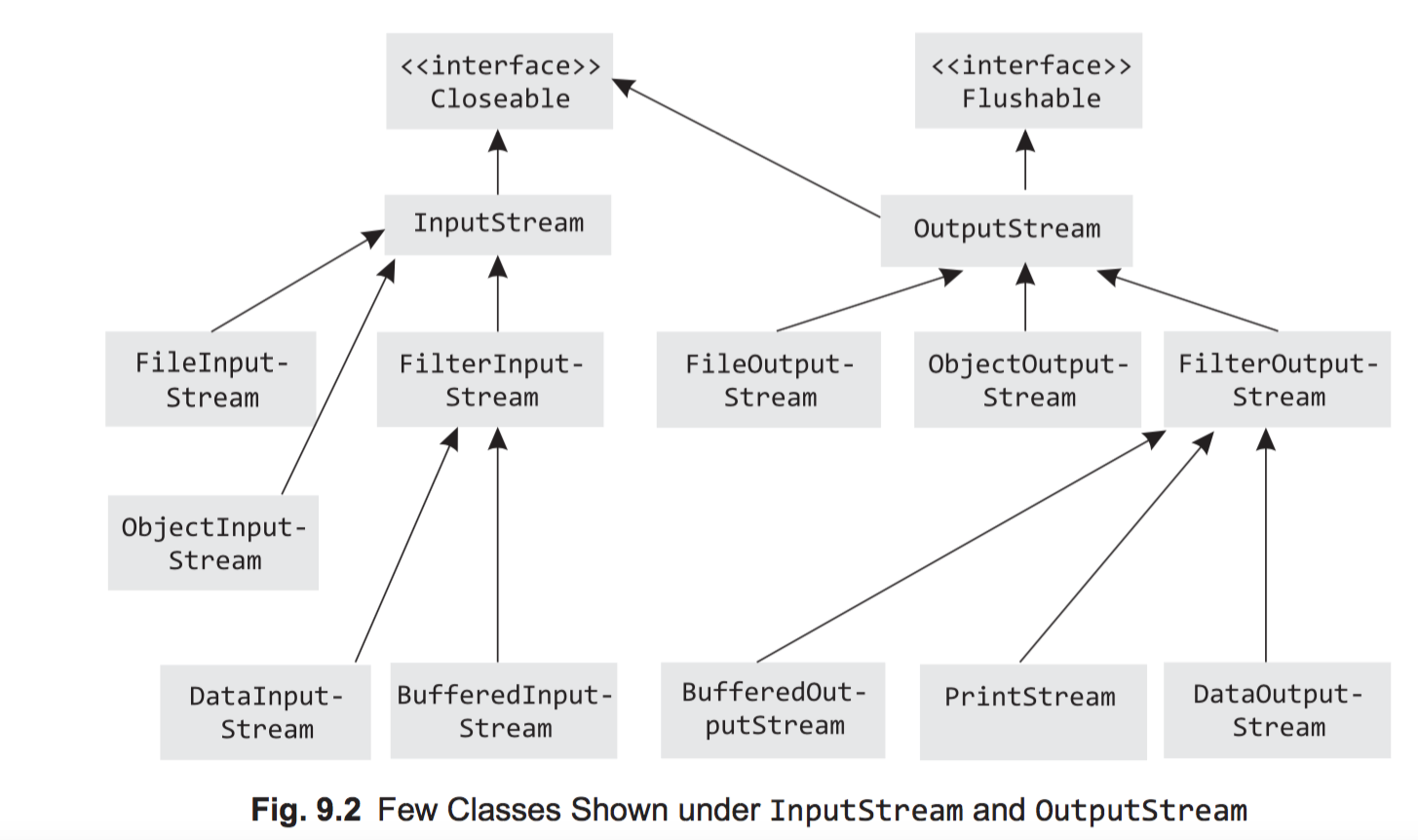
## I/O

### Java中使用内存映射文件

<http://itindex.net/detail/49906-java-内存-映射>

The java.io package contains two top level byte stream abstract classes: java.io.InputStream (for reading bytes) and java.io.OutputStream (for writing bytes). It also contains two other top level character stream abstract classes: java.io.Reader (for reading characters) and java. io.Writer (for writing characters)





The java.nio package is used to perform advanced I/O operations like memory mapping of files, file locking, buffer classes have been provided for all primitive types, channels representing connections to files. Memory mapping is a concept used in virtual memory. The entire file or region of a file (for large files) is mapped byte to byte between the file and the virtual memory. The mapped file is treated as it is actually present in the primary memory, thereby increasing the performance of I/O.

A mapped buffer is obtained using the map method of the FileChannel object

**public** **class** ReadWriteUsingNIO {

**public** **static** **void** main(String args[]) {

**try** {

// Use a mapped file to read a text file

FileInputStream fis = **new** FileInputStream("Sample.txt");

FileChannel fc = fis.getChannel();

**long** fs = fc.size();

MappedByteBuffer mBuf = fc.map(FileChannel.MapMode.***READ\_ONLY***, 0, fs);

**for** (**int** i = 0; i < fs; i++)

System.***out***.print((**char**) mBuf.get());

fc.close();

fis.close();

// write to a file using nio

String str = "welcome, writing to a file using nio package";

FileOutputStream fos = **new** FileOutputStream("samplenio.txt");

FileChannel fc1 = fos.getChannel();

ByteBuffer buffer = ByteBuffer.*allocate*(str.length());

**byte**[] b = str.getBytes();

buffer.put(b);

buffer.flip();

fc1.write(buffer);

fc1.close();

fos.close();

} **catch** (Exception e) {

System.***out***.println(e);

}

}

}

## 集合

LinkedList can act as a list, stack, queue as well as a double-ended queue

**import** java.util.\*;

**class** LinkedListDemo {

**public** **static** **void** main(String args[]) {

LinkedList<String> lis = **new** LinkedList<String>();

lis.add("Hello");

lis.add("Linked List");

lis.add("Demo");

lis.add(**null**);

**for** (String s : lis)

System.***out***.println(s);

// as a Stack (LIFO order)

LinkedList<Integer> st = **new** LinkedList<Integer>();

st.push(**new** Integer(1));

st.push(**new** Integer(2));

st.push(**new** Integer(3));

st.add(**new** Integer(4));

System.***out***.println("Object popped: " + st.pop());

System.***out***.println("Object popped: " + st.pop());

System.***out***.println("Object popped: " + st.pop());

System.***out***.println("Object popped: " + st.pop());

LinkedList<Long> l = **new** LinkedList<Long>();

// as queue (FIFO order)

l.add(**new** Long(1));

l.add(**new** Long(2));

l.add(**new** Long(3));

l.add(**new** Long(4));

System.***out***.println("Queue : " + l);

System.***out***.println("head of queue: " + l.peek());

System.***out***.println("head of queue removed and returned: " + l.poll());

System.***out***.println("Queue : " + l);

// as a double ended queue

// insertion and deleltion at both ends

l.addFirst(**new** Long(0));

System.***out***.println("Double ended Queue : " + l);

l.addLast(**new** Long(5));

System.***out***.println("Double ended Queue : " + l);

System.***out***.println("head of queue removed and returned:" + l.removeFirst());

System.***out***.println("tail of Queue removed and returned:" + l.removeLast());

System.***out***.println("Double ended Queue : " + l);

}

}

## JAVA 性能监控

### 配置Java远程监控授权（Java Mission Control）

http://www.icoolxue.com/blog/show/9

netbeans profiler

Oracle Solaris Studio profiling tool

jcmd

Prints basic class, thread, and VM information for a Java process. This is suitable for use in scripts; it is executed like this:

% **jcmd process\_id command optional\_arguments**Supplying the command help will list all possible commands, and supplying help

<command> will give the syntax for a particular command.

jconsole

Provides a graphical view of JVM activities, including thread usage, class usage, and GC activities.

jhat

Reads and helps analyze memory heap dumps. This is a postprocessing utility.

jmap

Provides heap dumps and other information about JVM memory usage. Suitable for scripting, though the heap dumps must be used in a postprocessing tool.

jinfo

Provides visibility into the system properties of the JVM, and allows some system properties to be set dynamically. Suitable for scripting.

jstack

Dumps the stacks of a Java process. Suitable for scripting.

jstat

Provides information about GC and class-loading activities. Suitable for scripting.

jvisualvm

A GUI tool to monitor a JVM, profile a running application, and analyze JVM heap dumps (which is a postprocessing activity, though jvisualvm can also take the heap dump from a live program).

These tools fits into these broad areas:

* Basic VM information
* Thread information
* Class information
* Live GC analysis
* Heap dump postprocessing
* Profiling a JVM

### 打印平台可用JVM参数

A useful way to determine what the flags are set to on a particular platform is to execute this command:

% **java other\_options -XX:+PrintFlagsFinal -version**...Hundreds of lines of output, including... uintx InitialHeapSize := 4169431040 {product} intx InlineSmallCode = 2000 {pd product}

### 打印可由jinfo在线管理的JVM参数

java -XX:+PrintFlagsFinal -version | grep manageable

% **jinfo -flag PrintGCDetails process\_id** -XX:+PrintGCDetails

Although jinfo does not itself indicate whether a flag is manageable or not, flags that are manageable (as identified when using the PrintFlagsFinal argument) can be turned on or off via jinfo:

% **jinfo -flag -PrintGCDetails process\_id** # turns off PrintGCDetails % **jinfo -flag PrintGCDetails process\_id**-XX:-PrintGCDetails

Be aware that jinfo can change the value of any flag, but that doesn’t mean that the JVM will respond to that change. For example, most flags that affect the behavior of a GC algorithm are used at startup time to determine various ways that the collector will behave. Altering a flag later via jinfo does not cause the JVM to change its behavior; it will continue executing based on how the algorithm was initialized. So this technique only works for those flags marked manageable in the output of the PrintFlagsFinal command.

### Jconsole

监控远程JAVA程序，比如在远程主机运行以下java程序,就可以在本地用jconsole 来跟踪了 在远程进程中输入 远程主机IP:60001 用户名密码默认为空

java -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.port=60001 -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false -jar logtest.jar

### VirsualVM

### HSDB

在 mac 下启动 hsdb

java -classpath /Library/Java/JavaVirtualMachines/jdk1.8.0\_45.jdk/Contents/Home/lib/sa-jdi.jar sun.jvm.hotspot.HSDB

ERROR: attach: task\_for\_pid(2635) failed: '(os/kern) failure' (5)

<http://comments.gmane.org/gmane.comp.java.openjdk.hotspot.runtime.devel/6993>

<http://bugs.java.com/bugdatabase/view_bug.do?bug_id=8013364>

查看java实例数量和占用字节数

jmap -histo 2477 | grep Connection

num instances bytes class name

54: 2068 148896 com.cloudera.impala.jdbc41.ImpalaJDBC4**Connection**

### JDK8中JVM堆内存划分

http://blog.csdn.net/jia20003/article/details/50703944

## 垃圾收集器

### Parallel GC

Parallel GC is a parallel stop-the-world collector, which means that when a GC occurs, it stops all application threads and performs the GC work using multiple threads

新生代使用 -XX:+UseParallelGC

JAVA 6

The enhanced Parallel GC was delivered in a Java 6 update release. It was enabled by a new command-line option called -XX:+UseParallelOldGC. When -XX:+UseParallelOldGC is enabled, parallel young generation collection is also enabled

a multithreaded stop-the-world young generation collector combined with a multithreaded stop-the- world old generation collector.

In Java 7 update release 4 (also referred to as Java 7u4, or JDK 7u4), - XX:+UseParallelOldGC was made the default GC and the normal mode of operation for Parallel GC. As of Java 7u4, specifying -XX:+UseParallelGC also enables -XX:+UseParallelOldGC, and likewise specifying -XX:+UseParallelOldGC also enables -XX:+UseParallelGC.

### Serial GC

-XX:+UseSerialGC

Serial GC is very similar to Parallel GC except that it does all its work in a single thread

The challenges with Serial GC are similar to those for Parallel GC. Pause times can be long, and they grow more or less linearly with the heap size and amount of live data. In addition, with Serial GC the long pauses are more pronounced, since the GC work is done in a single thread.

### Concurrent Mark Sweep (CMS) GC

CMS GC was developed in response to an increasing number of applications that demand a GC with lower worst-case pause times than Serial or Parallel GC and where it is acceptable to sacrifice some application throughput to eliminate or greatly reduce the number of lengthy GC pauses.

The major difference between Parallel GC and CMS GC is the old generation collection. For CMS GC, the old generation collections attempt to avoid long pauses in application threads. To achieve this, the CMS old generation collector does most of its work concurrently with application thread execution, except for a few relatively short GC synchronization pauses

CMS is often referred to as mostly concurrent, since there are some phases of old generation collection that pause application threads. Exam- ples are the initial-mark and remark phases. In CMS’s initial implementation, both the initial-mark and remark phases were single-threaded, but they have since been enhanced to be multithreaded. The HotSpot command-line options to support mul- tithreaded initial-mark and remark phases are -XX:+CMSParallelInitialMark Enabled and -XX:CMSParallelRemarkEnabled. These are automatically enabled by default when CMS GC is enabled by the -XX:+UseConcurrent MarkSweepGC command-line option.

### Garbage First (G1) GC

The G1 garbage collector addresses many of the shortcomings of Parallel, Serial, and CMS GC by taking a somewhat different approach. G1 divides the heap into a set of regions. Most GC operations can then be performed a region at a time rather than on the entire Java heap or an entire generation.

The term to describe the collection of a subset of old generation regions in conjunction with a young collection is *mixed GC*. Hence, a mixed GC is a GC event in which all young generation regions are collected in addition to a subset of old generation regions. In other words, a mixed GC is a mix of young and old generation regions that are being collected.

Similar to CMS GC, there is a fail-safe to collect and compact the entire old generation in dire situations such as when old generation space is exhausted.

A G1 old generation collection, ignoring the fail-safe type of collection, is a set of phases, some of which are parallel stop-the-world and some of which are parallel con- current. That is, some phases are multithreaded and stop all application threads, and others are multithreaded and execute at the same time as the application threads.

G1 initiates an old generation collection when a Java heap occupancy threshold is exceeded.

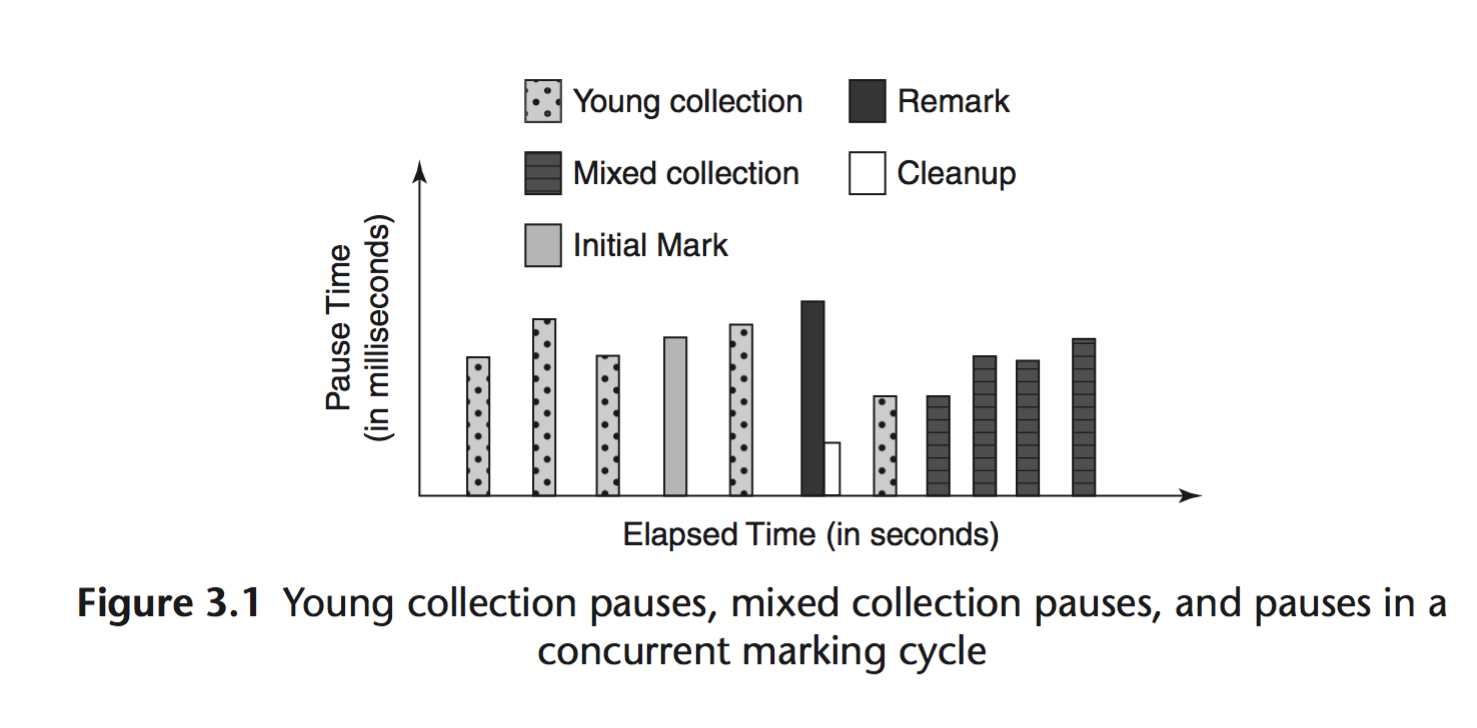
It is important to note that the heap occupancy threshold in G1 measures the old generation occupancy compared to the entire Java heap. Readers who are familiar with CMS GC remember that CMS initiates an old generation collection using an occupancy threshold applied against the old generation space only. In G1, once the heap occupancy threshold is reached or exceeded, a parallel stop-the-world initial-mark phase is scheduled to execute.

G1 primarily targets the use case of large Java heaps with reasonably low pauses, and also those applications that are using CMS GC. There are plans to use G1 to also target the throughput use case, but for applications looking for high throughput that can tolerate longer GC pauses, Parallel GC is currently the better choice.

As mentioned earlier, G1 divides the Java heap into regions. The region size can vary depending on the size of the heap but must be a power of 2 and at least 1MB and at most 32MB. Possible region sizes are therefore 1, 2, 4, 8, 16, and 32MB. All regions are the same size, and their size does not change during execution of the JVM. The region size calculation is based on the average of the initial and maximum Java heap sizes such that there are about 2000 regions for that average heap size. As an example, for a 16GB Java heap with -Xmx16g -Xms16g command-line options, G1 will choose a region size of 16GB/2000 = 8MB.

G1 may increase the Java heap size for several reasons:

* An increase in size can occur based on heap size calculations during a full GC.
* When a young or mixed GC occurs, G1 calculates the time spent to perform the GC compared to the time spent executing the Java application. If too much time is spent in GC according to the command-line setting -XX:GCTimeRatio, the Java heap size is increased. The idea behind growing the Java heap size in this situation is to allow GCs to happen less frequently so that the time spent in GC compared to the time spent executing the application is reduced.  The default value for -XX:GCTimeRatio in G1 is 9. All other HotSpot garbage collectors default to a value of 99. The larger the value for GCTimeRatio, the more aggressive the increase in Java heap size. The other HotSpot collectors are thus more aggressive in their decision to increase Java heap size and by default are targeted to spend less time in GC relative to the time spent executing the application.
* If an object allocation fails, even after having done a GC, rather than immediately falling back to doing a full GC, G1 will attempt to increase the heap size to satisfy the object allocation.
* If a humongous object allocation fails to find enough consecutive free regions to allocate the object, G1 will try to expand the Java heap to obtain more available regions rather than doing a full GC.
* When a GC requests a new region into which to evacuate objects, G1 will prefer to increase the size of the Java heap to obtain a new region rather than failing the GC and falling back to a full GC in an attempt to find an available region.



If any of the concurrent marking tasks and hence the entire cycle take too long to complete, a mixed collection pause is delayed, which could eventually lead to an evacuation failure. An evacuation failure will show up as a to-space exhausted message on the GC log, and the total time attributed to the failure will be shown in the Other section of the pause

The -XX:+PrintAdaptiveSizePolicy option dumps details of G1’s ergonomics heuristic decisions. An example follows.

First the command line:

JAVA\_OPTS="-XX:+UseG1GC -XX:+PrintGCDetails -XX:+PrintAdaptiveSizePolicy

-Xloggc:jdk8u45\_h2.log

* -Xms2g -Xmx4g -XX:MaxGCPauseMillis=100
* or   -Xms2g -Xmx4g -XX:MaxGCPauseMillis=100 -
* XX:InitiatingHeapOccupancyPercent=55

### Java 对象布局工具

<http://openjdk.java.net/projects/code-tools/jol/>

### 垃圾收集相关术语(terms)

generational garbage collection, the Java heap, young generation space, old generation space, eden space, survivor space, parallel garbage collection, stop-the-world garbage collection, concurrent garbage collection, incremental garbage collection, marking, and compaction

During a young collection, G1 GC pauses the application threads to move live objects from the young regions into survivor regions or promote them into old regions or both. For a mixed collection, G1 GC additionally moves live objects from the most (for lack of a better term) “efficient” old regions into free region(s), which become a part of the old generation.

G1 GC has three main types of garbage collection cycles: a young collection cycle, a multistage concurrent marking cycle, and a mixed collection cycle

Prior to JDK 8u40, if any humongous region was completely free, it could be collected only during the cleanup pause of the concur- rent collection cycle. In an effort to optimize the collection of short-lived humongous objects, JDK 8u40 made a noteworthy change such that if the humongous regions are determined to have no incoming references, they can be reclaimed and returned to the list of free regions during a young collection. A full garbage collection pause will also collect completely free humongous regions.

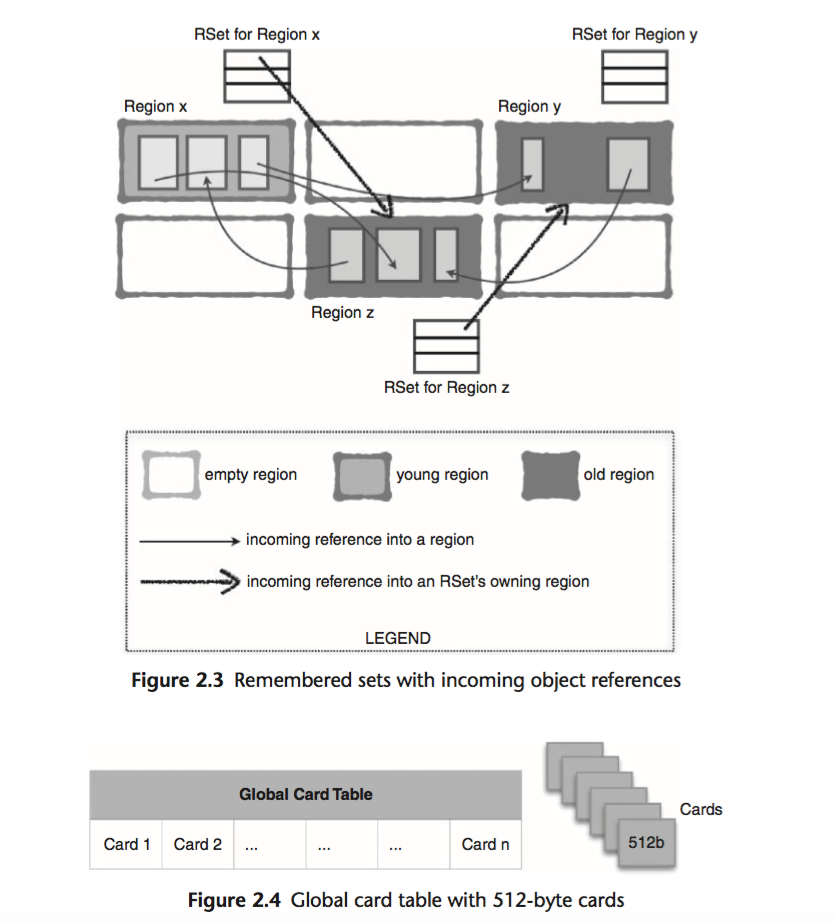
-XX:InitiatingHeapOccupancyPercent

In G1, the IHOP threshold defaults to 45 percent of the total Java heap. It is important to note that this heap occupancy percentage applies to the entire Java heap, unlike the heap occupancy command-line option used with CMS GC where it applies only to the old generation. In G1 GC, there is no physically separate old generation—there is a single pool of free regions that can be allocated as eden, survivor, old, or humongous. Also, the number of regions that are allocated, for say the eden, can vary over time. Hence having an old generation percentage didn’t really make sense.

There are two important parameters that help with the selection of candidate old regions for the CSet of a mixed collection: -XX:G1MixedGCLiveThresholdPercent and -XX:G1OldCSetRegionThresholdPercent.

-XX:G1MixedGCLiveThresholdPercent, which defaults to 85 percent (JDK 8u45) of a G1 GC region, is a liveness threshold and a set limit to exclude the most expensive of old regions from the CSet of mixed collections. G1 GC sets a limit such that any old region that falls below this liveness threshold is included in the CSet of a mixed collection.

-XX:G1OldCSetRegionThresholdPercent, which defaults to 10 percent (JDK 8u45) of the total Java heap, sets the maximum limit on the number of old regions that can be collected per mixed collection pause. The threshold is dependent on the total Java heap available to the JVM process and is expressed as a percentage of the total Java heap. Chapter 3 covers a few examples to highlight the functioning of these thresholds.



In order to limit the time spent updating RSets, G1 sets a target time as a percentage of the pause time goal (-XX:MaxGCPauseMillis) default 10

If after looking at the logs you realize that spending 10 percent of your pause time goal in updating RSets is undesirable, you can change the percentage by updating the -XX:G1RSetUpdatingPauseTime Percent command-line option to reflect your desired value

As discussed in Chapter 2, there is a command-line option called –XX:G1Conc RefinementThreads. By default it is set to the same value as –XX:ParallelGCThreads, which means that any change in XX:ParallelGCThreads will change the –XX:G1ConcRefinementThreads value as well.

The option -XX:+G1SummarizeRSetStats can be used to provide a window into the total number of RSet coarsenings (fine-grained PRT or coarsened bitmap) to help determine if concurrent refinement threads are able to handle the updated buffers and to gather more information on nmethods. This option summarizes RSet statistics every *n*th GC pause, where *n* is set by -XX:G1SummarizeRSetStatsPeriod=n.

–XX:+G1SummarizeRSetStats is a diagnostic option and hence must be enabled by adding –XX:+UnlockDiagnosticVMOptions to the command line, for example,

JAVA\_OPTS=”-XX:+UseG1GC -XX:+UnlockDiagnosticVMOptions -XX: +PrintGCDetails -XX:+G1SummarizeRSetStats -XX:G1Summarize RSetStatsPeriod=1 -Xloggc:jdk8u45\_h2.log”.

## Groovy

<http://www.groovy-lang.org/ides.html>

### Groovy在Spring中的简单使用实例

http://my.oschina.net/u/2494018/blog/611298

## Spring

### [SpringBoot集成mybatis](https://segmentfault.com/a/1190000004275305)

https://segmentfault.com/a/1190000004275305?\_ea=556953

### [Spring、Spring MVC整合的时候如何存在两次bean的加载吗](https://www.oschina.net/question/1791398_2154055)

<http://www.oschina.net/question/1791398_2154055>

http://git.oschina.net/wangkang/llsfw

### Spring MVC防御CSRF、XSS和SQL注入攻击

http://itindex.net/blog/2013/10/25/1382688300000.html

<https://github.com/spring-projects/>

<http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/>

<http://docs.spring.io/spring-framework/>

### Jackson JSONP Support

In order to enable JSONP support for @ResponseBody and ResponseEntity methods, declare an @ControllerAdvice bean that extends AbstractJsonpResponseBodyAdvice as shown below where the constructor argument indicates the JSONP query parameter name(s):

@ControllerAdvice

**public class** JsonpAdvice **extends** AbstractJsonpResponseBodyAdvice {

**public** JsonpAdvice() { **super**(***"callback"***);

} }

For controllers relying on view resolution, JSONP is automatically enabled when the request has a query parameter named jsonp or callback. Those names can be customized through jsonpParameterNames property.

### spring3mvc框架开发中resin和tomcat默认servelt配置处理静态资源

<http://blog.csdn.net/5iasp/article/details/8017918>

### Spring Web Flow 2.0 入门详解

http://docs.spring.io/spring-webflow/docs/2.3.x/reference/htmlsingle/

http://www.cnblogs.com/xwdreamer/archive/2011/11/10/2296939.html

### API

<http://www.java2s.com/Tutorials/Java_Lambda/java.util.function/Function/Function_andThen_example.htm>

<http://docs.oracle.com/javase/8/docs/api/>

<https://segmentfault.com/a/1190000000664023>

IBM

<http://www.ibm.com/developerworks/cn/java/>

### concurrent 包使用介绍

<http://my.oschina.net/u/1185331/blog/502350>

线程池

<http://zy116494718.iteye.com/blog/1704344>

<http://wenku.baidu.com/link?url=uN_WeM4pMmha6kVytYsxRiQpjPvQcUqcxrDnjWK3DccTE6a89Bxl3TMF2jUhUau5eygb3ywZYYUOFGENzYdosYbu9PBCqrz-1ya7YzKgDkW>

### java 分析工具

<http://www.cnblogs.com/nexiyi/p/java_thread_jstack.html>

强引用

<http://www.cnblogs.com/mengdd/archive/2013/09/03/3298852.html>

### jvm调优

<http://www.360doc.com/content/12/0113/08/1073512_179088229.shtml>

<http://www.cnblogs.com/dartagnan/archive/2011/09/25/2190507.html>

### 详细分析Java中断机制

<http://www.infoq.com/cn/articles/java-interrupt-mechanism>

### Java Platform, Standard Edition Tools Reference

<http://docs.oracle.com/javase/8/docs/technotes/tools/unix/index.html>

### 将 java 做成服务

<http://wrapper.tanukisoftware.com/doc/english/download.jsp>

### Spring 源码分析

#### Bean 定义，加载Bean的过程中处理

AbstractBeanDefinition

public void applyDefaults(BeanDefinitionDefaults defaults)

#### 基于注解的Bean 定义加载

AnnotatedGenericBeanDefinition

#### 创建Bean instance的地方

AbstractAutowireCapableBeanFactory

**protected** Object createBean(String beanName, RootBeanDefinition mbd, Object[] args) **throws** BeanCreationException

#### 调用@PostConstruct

InitDestroyAnnotationBeanPostProcessor

#### Lifecycle callbacks

To interact with the container’s management of the bean lifecycle, you can implement the Spring InitializingBean and DisposableBean interfaces. The container calls afterPropertiesSet() for the former and destroy() for the latter to allow the bean to perform certain actions upon initialization and destruction of your beans.

**Tip**

The JSR-250 @PostConstruct and @PreDestroy annotations are generally considered best practice for receiving lifecycle callbacks in a modern Spring application. Using these annotations means that your beans are not coupled to Spring specific interfaces. For details see the section called “@PostConstruct and @PreDestroy”.

If you don’t want to use the JSR-250 annotations but you are still looking to remove coupling consider the use of init-method and destroy-method object definition metadata.

Internally, the Spring Framework uses BeanPostProcessor implementations to process any callback interfaces it can find and call the appropriate methods

#### Initialization callbacks

The org.springframework.beans.factory.InitializingBean interface allows a bean to perform initialization work after all necessary properties on the bean have been set by the container. The InitializingBean interface specifies a single method:

**void** afterPropertiesSet() **throws** Exception;

It is recommended that you do not use the InitializingBean interface because it unnecessarily couples the code to Spring. Alternatively, use the @PostConstruct annotation or specify a POJO initialization method

#### Destruction callbacks

Implementing the org.springframework.beans.factory.DisposableBean interface allows a bean to get a callback when the container containing it is destroyed. The DisposableBean interface specifies a single method:

**void** destroy() **throws** Exception;

It is recommended that you do not use the DisposableBean callback interface because it unnecessarily couples the code to Spring. Alternatively, use the @PreDestroy annotation or specify a generic method that is supported by bean definitions.

#### Startup and shutdown callbacks

The Lifecycle interface defines the essential methods for any object that has its own lifecycle requirements (e.g. starts and stops some background process):

**public interface** Lifecycle { **void** start();

**void** stop();

**boolean** isRunning();

}

Any Spring-managed object may implement that interface. Then, when the ApplicationContext itself receives start and stop signals, e.g. for a stop/restart scenario at runtime, it will cascade those calls to all Lifecycle implementations defined within that context. It does this by delegating to a LifecycleProcessor:

**public interface** LifecycleProcessor **extends** Lifecycle { **void** onRefresh(); **void** onClose();

}

**Tip**

Note that the regular org.springframework.context.Lifecycle interface is just a plain contract for explicit start/stop notifications and does NOT imply auto-startup at context refresh time. Consider implementing org.springframework.context.SmartLifecycle instead for fine-grained control over auto-startup of a specific bean (including startup phases). Also, please note that stop notifications are not guaranteed to come before destruction: On regular shutdown,

all Lifecycle beans will first receive a stop notification before the general destruction callbacks are being propagated; however, on hot refresh during a context’s lifetime or on aborted refresh attempts, only destroy methods will be called.

The order of startup and shutdown invocations can be important. If a "depends-on" relationship exists between any two objects, the dependent side will start *after* its dependency, and it will stop *before* its dependency. However, at times the direct dependencies are unknown. You may only know that objects of a certain type should start prior to objects of another type. In those cases, the SmartLifecycle interface defines another option, namely the getPhase() method as defined on its super-interface, Phased.

**public interface** Phased { **int** getPhase();

}

**public interface** SmartLifecycle **extends** Lifecycle, Phased { **boolean** isAutoStartup(); **void** stop(Runnable callback);

}

#### Shutting down the Spring IoC container gracefully in non-web applications

**Note**

This section applies only to non-web applications. Spring’s web-based ApplicationContext implementations already have code in place to shut down the Spring IoC container gracefully when the relevant web application is shut down.

If you are using Spring’s IoC container in a non-web application environment; for example, in a rich client desktop environment; you register a shutdown hook with the JVM. Doing so ensures a graceful shutdown and calls the relevant destroy methods on your singleton beans so that all resources are released. Of course, you must still configure and implement these destroy callbacks correctly.

To register a shutdown hook, you call the registerShutdownHook() method that is declared on the ConfigurableApplicationContext interface:

**import** org.springframework.context.ConfigurableApplicationContext; **import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public final class** Boot { **public static void** main(**final** String[] args) **throws** Exception {

ConfigurableApplicationContext ctx = **new** ClassPathXmlApplicationContext( **new** String []{***"beans.xml"***});

*// add a shutdown hook for the above context...*

ctx.registerShutdownHook();

*// app runs here...*

*// main method exits, hook is called prior to the app shutting down...*

} }

#### ApplicationContextAware and BeanNameAware

When an ApplicationContext creates an object instance that implements the org.springframework.context.ApplicationContextAware interface, the instance is provided with a reference to that ApplicationContext.

**public interface** ApplicationContextAware {

**void** setApplicationContext(ApplicationContext applicationContext) **throws** BeansException;

}

When an ApplicationContext creates a class that implements the org.springframework.beans.factory.BeanNameAware interface, the class is provided with a reference to the name defined in its associated object definition.

**public interface** BeanNameAware {

**void** setBeanName(String name) **throws** BeansException;

}

The callback is invoked after population of normal bean properties but before an initialization callback such as InitializingBean *afterPropertiesSet* or a custom init-method.

#### BeanFactoryPostProcessor

To change the actual bean definition (i.e., the *blueprint* that defines the bean), you instead need to use a BeanFactoryPostProcessor as described in the section called “Customizing configuration metadata with a BeanFactoryPostProcessor”.

#### Customizing configuration metadata with a BeanFactoryPostProcessor

The next extension point that we will look at is the org.springframework.beans.factory.config.BeanFactoryPostProcessor. The semantics of this interface are similar to those of the BeanPostProcessor, with one major difference: BeanFactoryPostProcessor operates on the *bean configuration metadata*; that is, the Spring IoC container allows a BeanFactoryPostProcessor to read the configuration metadata and potentially change it *before* the container instantiates any beans other than BeanFactoryPostProcessors.

Spring includes a number of predefined bean factory post-processors, such as PropertyOverrideConfigurer and PropertyPlaceholderConfigurer

#### JDBC 配置

**<bean class**=**"org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"> <property name**=**"locations" value**=**"classpath:com/foo/jdbc.properties"/>**

**</bean>**

**<bean id**=**"dataSource" destroy-method**=**"close" class**=**"org.apache.commons.dbcp.BasicDataSource">**

**<property name**=**"driverClassName" value**=**"${jdbc.driverClassName}"/> <property name**=**"url" value**=**"${jdbc.url}"/> <property name**=**"username" value**=**"${jdbc.username}"/> <property name**=**"password" value**=**"${jdbc.password}"/>**

**</bean>**

The actual values come from another file in the standard Java Properties format: **jdbc.properties**

jdbc.driverClassName=org.hsqldb.jdbcDriver

jdbc.url=jdbc:hsqldb:hsql://production:9002

jdbc.username=sa

jdbc.password=root

#### 运行时配置

You can use the PropertyPlaceholderConfigurer to substitute class names, which is sometimes useful when you have to pick a particular implementation class at runtime. For example:

**<bean class**=**"org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"> <property name**=**"locations">**

**<value>**classpath:com/foo/strategy.properties**</value> </property>**

**<property name**=**"properties"> <value>**custom.strategy.class=com.foo.DefaultStrategy**</value>**

**</property>**

**</bean>**

**<bean id**=**"serviceStrategy" class**=**"${custom.strategy.class}"/>**

If the class cannot be resolved at runtime to a valid class, resolution of the bean fails when it is about to be created, which is during the preInstantiateSingletons() phase of an ApplicationContext for a non-lazy-init bean.

#### Customizing instantiation logic with a FactoryBean

Implement the org.springframework.beans.factory.FactoryBean interface for objects that *are themselves factories*.

The FactoryBean interface is a point of pluggability into the Spring IoC container’s instantiation logic. If you have complex initialization code that is better expressed in Java as opposed to a (potentially) verbose amount of XML, you can create your own FactoryBean, write the complex initialization inside that class, and then plug your custom FactoryBean into the container.

The FactoryBean interface provides three methods: • Object getObject(): returns an instance of the object this factory creates. The instance can

possibly be shared, depending on whether this factory returns singletons or prototypes.

• booleanisSingleton():returnstrueifthisFactoryBeanreturnssingletons,falseotherwise.

• Class getObjectType(): returns the object type returned by the getObject() method or null if the type is not known in advance.

@Autowired, @Inject, @Resource, and @Value annotations are handled by Spring BeanPostProcessor implementations which in turn means that you *cannot* apply these annotations within your own BeanPostProcessor or BeanFactoryPostProcessor types (if any). These types must be 'wired up' explicitly via XML or using a Spring @Bean method.

#### @Configuration

@Configuration

**public class** AppConfig {

@Bean

**public** MyService myService() { **return new** MyServiceImpl();

}

}

The AppConfig class above would be equivalent to the following Spring <beans/> XML:

**<beans>**

**<bean id**=**"myService" class**=**"com.acme.services.MyServiceImpl"/>**

**</beans>**

#### Full @Configuration vs 'lite' @Beans mode?

When @Bean methods are declared within classes that are *not* annotated with @Configuration they are referred to as being processed in a 'lite' mode. For example, bean methods declared in a @Component or even in a *plain old class* will be considered 'lite'.

Unlike full @Configuration, lite @Bean methods cannot easily declare inter-bean dependencies. Usually one @Bean method should not invoke another @Bean method when operating in 'lite' mode.

Only using @Bean methods within @Configuration classes is a recommended approach of ensuring that 'full' mode is always used. This will prevent the same @Bean method from accidentally being invoked multiple times and helps to reduce subtle bugs that can be hard to track down when operating in 'lite' mode.

**public static void** main(String[] args) { AnnotationConfigApplicationContext ctx = **new** AnnotationConfigApplicationContext(); ctx.register(AppConfig.**class**, OtherConfig.**class**); ctx.register(AdditionalConfig.**class**); ctx.refresh(); MyService myService = ctx.getBean(MyService.**class**); myService.doStuff();

}

#### Enabling component scanning with scan(String...)

To enable component scanning, just annotate your @Configuration class as follows:

@Configuration

@ComponentScan(basePackages = "com.acme")

**public class** AppConfig {

｝

**Tip**

Experienced Spring users will be familiar with the XML declaration equivalent from Spring’s context: namespace

**<beans>**

**<context:component-scan base-package**=**"com.acme"/>**

**</beans>**

In the example above, the com.acme package will be scanned, looking for any @Component- annotated classes, and those classes will be registered as Spring bean definitions within the container. AnnotationConfigApplicationContext exposes the scan(String...) method to allow for the same component-scanning functionality:

**public static void** main(String[] args) {

 AnnotationConfigApplicationContext ctx = **new** AnnotationConfigApplicationContext();

ctx.scan(***"com.acme"***);

ctx.refresh();

 MyService myService = ctx.getBean(MyService.**class**);

}

#### Injecting inter-bean dependencies

@Configuration

**public class** AppConfig {

@Bean

**public** Foo foo() { **return new** Foo(bar());

}

@Bean

**public** Bar bar() { **return new** Bar();

} }

**Note**

This method of declaring inter-bean dependencies only works when the @Bean method is declared within a @Configuration class. You cannot declare inter-bean dependencies using plain @Component classes.

#### Using the @Import annotation

@Configuration

**public class** ConfigA {

@Bean

**public** A a() { **return new** A();

} }

@Configuration @Import(ConfigA.class) **public class** ConfigB {

@Bean

**public** B b() { **return new** B();

} }

Now, rather than needing to specify both ConfigA.class and ConfigB.class when instantiating the context, only ConfigB needs to be supplied explicitly:

**public static void** main(String[] args) {

 ApplicationContext ctx = **new** AnnotationConfigApplicationContext(ConfigB.**class**);

*// now both beans A and B will be available...*

A a = ctx.getBean(A.**class**);

B b = ctx.getBean(B.**class**);

}

#### Combining Java and XML configuration

##### XML-centric use of @Configuration classes

@Configuration

**public class** AppConfig { @Autowired

**private** DataSource dataSource;

@Bean

**public** AccountRepository accountRepository() { **return new** JdbcAccountRepository(dataSource);

}

@Bean

**public** TransferService transferService() { **return new** TransferService(accountRepository());

} }

**system-test-config.xml**:

**<beans>**

*<!-- enable processing of annotations such as @Autowired and @Configuration -->*

**<context:annotation-config/> <context:property-placeholder location**=**"classpath:/com/acme/jdbc.properties"/>**

**<bean class**=**"com.acme.AppConfig"/>**

**<bean class**=**"org.springframework.jdbc.datasource.DriverManagerDataSource"> <property name**=**"url" value**=**"${jdbc.url}"/> <property name**=**"username" value**=**"${jdbc.username}"/> <property name**=**"password" value**=**"${jdbc.password}"/>**

**</bean>**

**</beans>**

**jdbc.properties**:

jdbc.url=jdbc:hsqldb:hsql://localhost/xdb

jdbc.username=sa

jdbc.password=123456

**public static void** main(String[] args) {

ApplicationContext ctx = **new** ClassPathXmlApplicationContext(***"classpath:/com/acme/system-test-***

***config.xml"***); TransferService transferService = ctx.getBean(TransferService.**class**); *// ...*

}

##### @Configuration class-centric use of XML with @ImportResource

@Configuration @ImportResource("classpath:/com/acme/properties-config.xml") **public class** AppConfig {

@Value("${jdbc.url}")

**private** String url; @Value("${jdbc.username}")

**private** String username; @Value("${jdbc.password}")

**private** String password;

@Bean

**public** DataSource dataSource() { **return new** DriverManagerDataSource(url, username, password);

} }

properties-config.xml

**<beans> <context:property-placeholder location**=**"classpath:/com/acme/jdbc.properties"/>**

**</beans>**

jdbc.properties

jdbc.url=jdbc:hsqldb:hsql://localhost/xdb

jdbc.username=sa

jdbc.password=

**public static void** main(String[] args) {

 ApplicationContext ctx = **new** AnnotationConfigApplicationContext(AppConfig.**class**); TransferService transferService = ctx.getBean(TransferService.**class**); *// ...*

}

### PDF EPUB

<http://www.siegmann.nl/epublib>

https://pdfbox.apache.org/2.0/examples.html

<http://epub.exirel.me/epub/index.html#api-du-module>

<http://pythonhosted.org/PyPDF2/PdfFileReader.html#PyPDF2.PdfFileReader.getDocumentInfo>

<http://www.manuel-strehl.de/dev/simple_epub_ebooks_with_python.en.html>

## J2EE

J2EE architecture is a multi-tier architecture with four tiers: client tier, web tier, enterprise tier, and information system tier The client tier basically consists of presentation logic. Web tier consists of components that respond to clients’ request over the Internet, i.e., accepting HTTP request and generating responses for them. Enterprise tier consists of business logic like the EJB, and information system tier consists of databases. J2EE consists of the following technologies: Java server pages (JSP), servlets, Java beans, Java database connectivity (JDBC), Java naming and directory interface (JNDI), enterprise Java beans (EJB), remote method invocation (RMI), Java mail API, Java messaging service, Java transaction API, and Java IDL/CORBA. We will discuss few of the technologies in this chapter.

# MAC

chflags的详解

用法：chflags [参数] 关键词 档案

常用关键词：

schg 档案完全无法被修改（super-user only）。

sappend 档案內容只能用附加（append）的方式修改（super-user only）。

sunlink 档案无法被刪除或者重命名（super-user only）。

uchg 档案完全无法被修改（super-user only 或者一般使用者）。

uappend 档案內容只能用附加（append）的方式修改（super-user only 或者一般使用者）。

uunlink 档案无法被刪除或者重命名（super-user only 或者一般使用者）。

uchg 用于文件时，owner 不能修改文件，也不能删除文件或者对文件重命名；uappend 用于文件时，owner 只能以 append 的方式修改文件，不能删除文件或者对文件重命名；uunlink 用于文件时，owner 可以对文件进行修改，但不能删除文件或者对文件重命名。

uchg 用于目录时，owner 不能删除目录或者重命名目录，可以修改目录中已有的文件，但不能在目录中添加文件，也不能删除目录中的文件或者对其重命名；uappend 用于目录时，owner 不能删除目录或者重命名目录，可以修改目录中已有的文件和在目录中添加文件，不能删除目录中的文件或者对其重命名；uunlink 用于目录时，owner 不能删除目录或者重命名目录，可以修改目录中已有的文件和在目录中添加文件，也可以删除目录中的文件或者对其重命名。

结论：uchg>uappend>uunlink。

示例：要在文件 file1 上应用系统禁删标志， 应使用下述命令：

# chflags sunlink file1

　　要禁用系统禁删标志， 只需在前述命令中的

sunlink

标志前加 “no”。 例如：

# chflags nosunlink file1

　　要显示文件上的标志， 应使用命令 [ls(1)](http://www.freebsd.org/cgi/man.cgi?query=ls&sektion=1) 的

-lo

参数：

# ls -lo file1

　　输出结果应类似于：

-rw-r--r-- 1 trhodes trhodes sunlnk 0 Mar 1 05:54 file1

# MAVEN

## 创建可执行jar

<build>

<finalName>image</finalName>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-jar-plugin</artifactId>

<version>2.4</version>

<configuration>

<archive>

<manifest>

<addClasspath>true</addClasspath>

<classpathPrefix>lib/</classpathPrefix>

<mainClass>com.cmz.test1.App</mainClass>

</manifest>

</archive>

</configuration>

</plugin>

</plugins>

</build>

http://maven.apache.org/plugins/maven-jar-plugin/jar-mojo.html

eclipse导出Maven依赖的Jar

1、 从Maven项目中导出项目依赖的jar包：进入工程pom.xml 所在的目录下，执行如下命令：

|  |  |
| --- | --- |
|  | mvn dependency:copy-dependencies |

2、从Maven仓库中导出jar包：进入工程pom.xml 所在的目录下，

输入以下命令：mvn dependency:copy-dependencies -DoutputDirectory=lib

更简单的  mvn dependency:copy-dependencies    会导出到target\dependency 下面

mvn clean dependency:copy-dependencies package  未试过 复制依赖的jar 非常有用

mvn dependency:copy-dependencies -DoutputDirectory=lib   -DincludeScope=compile 这样jar包都会copy到工程目录下的lib里面  mvn package 然后从被打包好的文件里找出jar来。

3、maven打包的时候有些jar包是不在mavencenter的。需要在pom中引入相关的本地jar包，那么相关的depency应该像如下更改

http://my.oschina.net/zimingforever/blog/266191

<dependency>

<groupId>org.apache.flueme</groupId>

<artifactId>flume-tools</artifactId>

<version>1.6.0</version>

<scope>system</scope>

<systemPath>${project.basedir}/libs/flume-tools-1.6.0.jar</systemPath>

</dependency>

<http://mvnrepository.com>

<http://mvnrepository.com/open-source/core-utilities>

<http://maven.apache.org/shared/maven-archiver/examples/classpath.html>

## maven惯用插件-groovy-maven-plugin

<http://www.myexception.cn/software/1486195.html>

<http://groovy.github.io/gmaven/groovy-maven-plugin/execute.html>

## Maven系列--"maven-compiler-plugin"的使用

http://my.oschina.net/poorzerg/blog/206856

# 大数据

<http://www.cloudera.com/content/www/en-us/documentation.html>

<http://archive.cloudera.com/cdh5/cdh/5/>

http://archive.cloudera.com/cm5/cm

<http://www.cloudera.com/documentation/enterprise/latest/topics/introduction.html>

## cloudera cdh所有的包都在这里

/opt/cloudera/parcels/CDH/

## 例如 hadoop的目录在

/opt/cloudera/parcels/CDH/lib/hadoop

## hadoop库文件在

/opt/cloudera/parcels/CDH/lib/hadoop/lib

## 配置文件目录

/opt/cloudera/parcels/CDH/etc/

## Spark

### 读取HIVE数据库

To connect Spark SQL to an existing Hive installation, you need to provide a Hive configuration. You do so by copying your *hive-site.xml* file to Spark’s *./conf/* direc‐ tory. If you just want to explore, a local Hive metastore will be used if no *hive-site.xml* is set, and we can easily load data into a Hive table to query later on.

cp /opt/cloudera/parcels/CDH/lib/hive/conf/hive-site.xml /opt/cloudera/parcels/CDH/lib/spark/conf/

执行shell

spark-shell --master yarn --driver-class-path /opt/cloudera/parcels/CDH/jars/spark-sql\_2.10-1.5.0-cdh5.6.0.jar:/opt/cloudera/parcels/CDH/lib/hive/lib/mysql-connector-java-5.1.38-bin.jar

scala>import rg.apache.spark.sql.hive.\_

scala>val hiveContext = new HiveContext(sc)

显示hive中的数据库

scala>hiveContext.sql("show databases").show()

hiveContext.refreshTable("cm.user")

val log = hiveContext.sql("select \* from cm.user")

log.count()

注意，当hive数据变动，需要重新刷新表再执行 sql及之后的统计例如count,否则会出现错误。

#### 多字段排序 count升序 pdate降序

log.groupBy("pdate").count().orderBy(asc("count"),desc("pdate")).show(10000)

#### 多字段排序 count升序 pdate降序

log.groupBy("pdate").count().sort($"count",$"pdate".desc).show(10000)

#### 连表查询 条件查询

var player = hiveContext.sql("select \* from cm.player")

var task = hiveContext.sql("select \* from cn\_cm.task")

player.join(task).where($"player.playerid"===17031).show()

#### 分组统计

task.groupBy($"playerid").count().show(10000)

pay.groupBy($"playerid").agg(avg($"amt"),sum($"amt")).show()

##### countDistinct

pay.groupBy($"playerid").agg(avg($"amt"),sum($"amt"),countDistinct($"pdate")).show()

#### 条件过滤

pay.filter($"playerid"===38937031).show()

pay.where($"playerid"===38937031).show()

### 刷新metadata

<http://lioo.iteye.com/blog/2036978>

AnalysisException: Failed to load metadata for table: CAUSED BY: TableLoadingException: Failed to load metadata for table: cn\_gamexyphs.dt\_log\_login CAUSED BY: CatalogException: Failed to create partition: CAUSED BY: FileNotFoundException: File Not

invalidate metadata my\_table;

<https://github.com/apache/spark/>

Spark cdh

### Stream Programming

http://spark.apache.org/docs/latest/streaming-programming-guide.html

**Deploying Applications**

This section discusses the steps to deploy a Spark Streaming application.

**Requirements**

To run a Spark Streaming applications, you need to have the following.

* *Cluster with a cluster manager* - This is the general requirement of any Spark application, and discussed in detail in the [deployment guide](http://spark.apache.org/docs/latest/cluster-overview.html).
* *Package the application JAR* - You have to compile your streaming application into a JAR. If you are using [spark-submit](http://spark.apache.org/docs/latest/submitting-applications.html) to start the application, then you will not need to provide Spark and Spark Streaming in the JAR. However, if your application uses advanced sources (e.g. Kafka, Flume, Twitter), then you will have to package the extra artifact they link to, along with their dependencies, in the JAR that is used to deploy the application. For example, an application using TwitterUtils will have to include spark-streaming-twitter\_2.10 and all its transitive dependencies in the application JAR.
* *Configuring sufficient memory for the executors* - Since the received data must be stored in memory, the executors must be configured with sufficient memory to hold the received data. Note that if you are doing 10 minute window operations, the system has to keep at least last 10 minutes of data in memory. So the memory requirements for the application depends on the operations used in it.
* *Configuring checkpointing* - If the stream application requires it, then a directory in the Hadoop API compatible fault-tolerant storage (e.g. HDFS, S3, etc.) must be configured as the checkpoint directory and the streaming application written in a way that checkpoint information can be used for failure recovery. See the checkpointing section for more details.
* *Configuring automatic restart of the application driver* - To automatically recover from a driver failure, the deployment infrastructure that is used to run the streaming application must monitor the driver process and relaunch the driver if it fails. Different [cluster managers](http://spark.apache.org/docs/latest/cluster-overview.html#cluster-manager-types) have different tools to achieve this.
  1. *Spark Standalone* - A Spark application driver can be submitted to run within the Spark Standalone cluster (see [cluster deploy mode](http://spark.apache.org/docs/latest/spark-standalone.html#launching-spark-applications)), that is, the application driver itself runs on one of the worker nodes. Furthermore, the Standalone cluster manager can be instructed to *supervise* the driver, and relaunch it if the driver fails either due to non-zero exit code, or due to failure of the node running the driver. See *cluster mode* and *supervise* in the [Spark Standalone guide](http://spark.apache.org/docs/latest/spark-standalone.html) for more details.
  2. *YARN* - Yarn supports a similar mechanism for automatically restarting an application. Please refer to YARN documentation for more details.
  3. *Mesos* - [Marathon](https://github.com/mesosphere/marathon) has been used to achieve this with Mesos.
* *Configuring write ahead logs* - Since Spark 1.2, we have introduced *write ahead logs* for achieving strong fault-tolerance guarantees. If enabled, all the data received from a receiver gets written into a write ahead log in the configuration checkpoint directory. This prevents data loss on driver recovery, thus ensuring zero data loss (discussed in detail in the Fault-tolerance Semantics section). This can be enabled by setting the [configuration parameter](http://spark.apache.org/docs/latest/configuration.html#spark-streaming) spark.streaming.receiver.writeAheadLog.enable to true. However, these stronger semantics may come at the cost of the receiving throughput of individual receivers. This can be corrected by running more receivers in parallel to increase aggregate throughput. Additionally, it is recommended that the replication of the received data within Spark be disabled when the write ahead log is enabled as the log is already stored in a replicated storage system. This can be done by setting the storage level for the input stream to StorageLevel.MEMORY\_AND\_DISK\_SER. While using S3 (or any file system that does not support flushing) for *write ahead logs*, please remember to enable spark.streaming.driver.writeAheadLog.closeFileAfterWrite and spark.streaming.receiver.writeAheadLog.closeFileAfterWrite. See [Spark Streaming Configuration](http://spark.apache.org/docs/latest/configuration.html#spark-streaming) for more details.

*Setting the max receiving rate* - If the cluster resources is not large enough for the streaming application to process data as fast as it is being received, the receivers can be rate limited by setting a maximum rate limit in terms of records / sec. See the [configuration parameters](http://spark.apache.org/docs/latest/configuration.html#spark-streaming) spark.streaming.receiver.maxRate for receivers and spark.streaming.kafka.maxRatePerPartition for Direct Kafka approach. In Spark 1.5, we have introduced a feature called *backpressure* that eliminate the need to set this rate limit, as Spark Streaming automatically figures out the rate limits and dynamically adjusts them if the processing conditions change. This backpressure can be enabled by setting the [configuration parameter](http://spark.apache.org/docs/latest/configuration.html#spark-streaming) spark.streaming.backpressure.enabled to true.

### Configuration and tuning

<http://spark.apache.org/docs/latest/configuration.html>

<http://spark.apache.org/docs/latest/tuning.html>

#### Garbage Collection Tuning

JVM garbage collection can be a problem when you have large “churn” in terms of the RDDs stored by your program. (It is usually not a problem in programs that just read an RDD once and then run many operations on it.) When Java needs to evict old objects to make room for new ones, it will need to trace through all your Java objects and find the unused ones. The main point to remember here is that *the cost of garbage collection is proportional to the number of Java objects*, so using data structures with fewer objects (e.g. an array of Ints instead of a LinkedList) greatly lowers this cost. An even better method is to persist objects in serialized form, as described above: now there will be only *one* object (a byte array) per RDD partition. Before trying other techniques, the first thing to try if GC is a problem is to use serialized caching.

GC can also be a problem due to interference between your tasks’ working memory (the amount of space needed to run the task) and the RDDs cached on your nodes. We will discuss how to control the space allocated to the RDD cache to mitigate this.

**Measuring the Impact of GC**

The first step in GC tuning is to collect statistics on how frequently garbage collection occurs and the amount of time spent GC. This can be done by adding -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCTimeStamps to the Java options. (See the [configuration guide](http://spark.apache.org/docs/latest/configuration.html#Dynamically-Loading-Spark-Properties) for info on passing Java options to Spark jobs.) Next time your Spark job is run, you will see messages printed in the worker’s logs each time a garbage collection occurs. Note these logs will be on your cluster’s worker nodes (in the stdout files in their work directories), *not* on your driver program.

**Advanced GC Tuning**

To further tune garbage collection, we first need to understand some basic information about memory management in the JVM:

* Java Heap space is divided in to two regions Young and Old. The Young generation is meant to hold short-lived objects while the Old generation is intended for objects with longer lifetimes.
* The Young generation is further divided into three regions [Eden, Survivor1, Survivor2].
* A simplified description of the garbage collection procedure: When Eden is full, a minor GC is run on Eden and objects that are alive from Eden and Survivor1 are copied to Survivor2. The Survivor regions are swapped. If an object is old enough or Survivor2 is full, it is moved to Old. Finally when Old is close to full, a full GC is invoked.

The goal of GC tuning in Spark is to ensure that only long-lived RDDs are stored in the Old generation and that the Young generation is sufficiently sized to store short-lived objects. This will help avoid full GCs to collect temporary objects created during task execution. Some steps which may be useful are:

* Check if there are too many garbage collections by collecting GC stats. If a full GC is invoked multiple times for before a task completes, it means that there isn’t enough memory available for executing tasks.
* In the GC stats that are printed, if the OldGen is close to being full, reduce the amount of memory used for caching by lowering spark.memory.storageFraction; it is better to cache fewer objects than to slow down task execution!
* If there are too many minor collections but not many major GCs, allocating more memory for Eden would help. You can set the size of the Eden to be an over-estimate of how much memory each task will need. If the size of Eden is determined to be E, then you can set the size of the Young generation using the option -Xmn=4/3\*E. (The scaling up by 4/3 is to account for space used by survivor regions as well.)
* As an example, if your task is reading data from HDFS, the amount of memory used by the task can be estimated using the size of the data block read from HDFS. Note that the size of a decompressed block is often 2 or 3 times the size of the block. So if we wish to have 3 or 4 tasks’ worth of working space, and the HDFS block size is 64 MB, we can estimate size of Eden to be 4\*3\*64MB.
* Monitor how the frequency and time taken by garbage collection changes with the new settings.

Our experience suggests that the effect of GC tuning depends on your application and the amount of memory available. There are [many more tuning options](http://www.oracle.com/technetwork/java/javase/gc-tuning-6-140523.html) described online, but at a high level, managing how frequently full GC takes place can help in reducing the overhead.

### Local vs. cluster modes

The behavior of the above code is undefined, and may not work as intended. To execute jobs, Spark breaks up the processing of RDD operations into tasks, each of which is executed by an executor. Prior to execution, Spark computes the task’s **closure**. The closure is those variables and methods which must be visible for the executor to perform its computations on the RDD (in this case foreach()). This closure is serialized and sent to each executor.

The variables within the closure sent to each executor are now copies and thus, when **counter** is referenced within the foreach function, it’s no longer the **counter** on the driver node. There is still a **counter** in the memory of the driver node but this is no longer visible to the executors! The executors only see the copy from the serialized closure. Thus, the final value of **counter** will still be zero since all operations on **counter** were referencing the value within the serialized closure.

In local mode, in some circumstances the foreach function will actually execute within the same JVM as the driver and will reference the same original **counter**, and may actually update it.

To ensure well-defined behavior in these sorts of scenarios one should use an Accumulator. Accumulators in Spark are used specifically to provide a mechanism for safely updating a variable when execution is split up across worker nodes in a cluster. The Accumulators section of this guide discusses these in more detail.

In general, closures - constructs like loops or locally defined methods, should not be used to mutate some global state. Spark does not define or guarantee the behavior of mutations to objects referenced from outside of closures. Some code that does this may work in local mode, but that’s just by accident and such code will not behave as expected in distributed mode. Use an Accumulator instead if some global aggregation is needed.

### Printing elements of an RDD

### Another common idiom is attempting to print out the elements of an RDD using rdd.foreach(println) or rdd.map(println). On a single machine, this will generate the expected output and print all the RDD’s elements. However, in cluster mode, the output to stdout being called by the executors is now writing to the executor’s stdout instead, not the one on the driver, so stdout on the driver won’t show these! To print all elements on the driver, one can use the collect() method to first bring the RDD to the driver node thus: rdd.collect().foreach(println). This can cause the driver to run out of memory, though, because collect() fetches the entire RDD to a single machine; if you only need to print a few elements of the RDD, a safer approach is to use the take(): rdd.take(100).foreach(println).

### Shuffle operations

Certain operations within Spark trigger an event known as the shuffle. The shuffle is Spark’s mechanism for re-distributing data so that it’s grouped differently across partitions. This typically involves copying data across executors and machines, making the shuffle a complex and costly operation.

**Background**

To understand what happens during the shuffle we can consider the example of the reduceByKey operation. The reduceByKey operation generates a new RDD where all values for a single key are combined into a tuple - the key and the result of executing a reduce function against all values associated with that key. The challenge is that not all values for a single key necessarily reside on the same partition, or even the same machine, but they must be co-located to compute the result.

In Spark, data is generally not distributed across partitions to be in the necessary place for a specific operation. During computations, a single task will operate on a single partition - thus, to organize all the data for a single reduceByKey reduce task to execute, Spark needs to perform an all-to-all operation. It must read from all partitions to find all the values for all keys, and then bring together values across partitions to compute the final result for each key - this is called the **shuffle**.

Although the set of elements in each partition of newly shuffled data will be deterministic, and so is the ordering of partitions themselves, the ordering of these elements is not. If one desires predictably ordered data following shuffle then it’s possible to use:

* mapPartitions to sort each partition using, for example, .sorted
* repartitionAndSortWithinPartitions to efficiently sort partitions while simultaneously repartitioning
* sortBy to make a globally ordered RDD

Operations which can cause a shuffle include **repartition** operations like repartition and coalesce, **‘ByKey** operations (except for counting) like groupByKey and reduceByKey, and **join** operations like cogroup and join.

**Performance Impact**

The **Shuffle** is an expensive operation since it involves disk I/O, data serialization, and network I/O. To organize data for the shuffle, Spark generates sets of tasks - *map* tasks to organize the data, and a set of *reduce* tasks to aggregate it. This nomenclature comes from MapReduce and does not directly relate to Spark’s map and reduce operations.

Internally, results from individual map tasks are kept in memory until they can’t fit. Then, these are sorted based on the target partition and written to a single file. On the reduce side, tasks read the relevant sorted blocks.

Certain shuffle operations can consume significant amounts of heap memory since they employ in-memory data structures to organize records before or after transferring them. Specifically, reduceByKey and aggregateByKey create these structures on the map side, and 'ByKey operations generate these on the reduce side. When data does not fit in memory Spark will spill these tables to disk, incurring the additional overhead of disk I/O and increased garbage collection.

Shuffle also generates a large number of intermediate files on disk. As of Spark 1.3, these files are preserved until the corresponding RDDs are no longer used and are garbage collected. This is done so the shuffle files don’t need to be re-created if the lineage is re-computed. Garbage collection may happen only after a long period time, if the application retains references to these RDDs or if GC does not kick in frequently. This means that long-running Spark jobs may consume a large amount of disk space. The temporary storage directory is specified by the spark.local.dir configuration parameter when configuring the Spark context.

Shuffle behavior can be tuned by adjusting a variety of configuration parameters. See the ‘Shuffle Behavior’ section within the [Spark Configuration Guide](http://spark.apache.org/docs/latest/configuration.html).

### 出现找不到类解决办法

Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/hadoop/fs/FSDataInputStream

at org.apache.spark.deploy.SparkSubmitArguments$$anonfun$mergeDefaultSparkProperties$1.apply(SparkSubmitArguments.scala:117)

at org.apache.spark.deploy.SparkSubmitArguments$$anonfun$mergeDefaultSparkProperties$1.apply(SparkSubmitArguments.scala:117)

at scala.Option.getOrElse(Option.scala:120)

at org.apache.spark.deploy.SparkSubmitArguments.mergeDefaultSparkProperties(SparkSubmitArguments.scala:117)

at org.apache.spark.deploy.SparkSubmitArguments.<init>(SparkSubmitArguments.scala:103)

at org.apache.spark.deploy.SparkSubmit$.main(SparkSubmit.scala:113)

at org.apache.spark.deploy.SparkSubmit.main(SparkSubmit.scala)

### 解决：

#### spark-env 环境变量

find / -name spark-env.sh

vim /opt/cloudera/parcels/CDH-5.6.0-1.cdh5.6.0.p0.45/etc/spark/conf.dist/spark-env.sh

将/usr/lib/ 替换成 /opt/cloudera/parcels/CDH/lib/

SPARK\_DIST\_CLASSPATH="$SPARK\_DIST\_CLASSPATH:/usr/lib/hadoop/lib/\*"

SPARK\_DIST\_CLASSPATH="$SPARK\_DIST\_CLASSPATH:/usr/lib/hadoop/\*"

SPARK\_DIST\_CLASSPATH="$SPARK\_DIST\_CLASSPATH:/usr/lib/hadoop-hdfs/lib/\*"

SPARK\_DIST\_CLASSPATH="$SPARK\_DIST\_CLASSPATH:/usr/lib/hadoop-hdfs/\*"

SPARK\_DIST\_CLASSPATH="$SPARK\_DIST\_CLASSPATH:/usr/lib/hadoop-mapreduce/lib/\*"

SPARK\_DIST\_CLASSPATH="$SPARK\_DIST\_CLASSPATH:/usr/lib/hadoop-mapreduce/\*"

SPARK\_DIST\_CLASSPATH="$SPARK\_DIST\_CLASSPATH:/usr/lib/hadoop-yarn/lib/\*"

SPARK\_DIST\_CLASSPATH="$SPARK\_DIST\_CLASSPATH:/usr/lib/hadoop-yarn/\*"

SPARK\_DIST\_CLASSPATH="$SPARK\_DIST\_CLASSPATH:/usr/lib/hive/lib/\*"

SPARK\_DIST\_CLASSPATH="$SPARK\_DIST\_CLASSPATH:/usr/lib/flume-ng/lib/\*"

SPARK\_DIST\_CLASSPATH="$SPARK\_DIST\_CLASSPATH:/usr/lib/paquet/lib/\*"

SPARK\_DIST\_CLASSPATH="$SPARK\_DIST\_CLASSPATH:/usr/lib/avro/lib/\*"

spark-shell --driver-class-path $(echo /opt/cloudera/parcels/CDH/jars/\*.jar |sed 's/ /:/g') --master yarn

### 提交spark应用程序到集群

java版本

spark-submit --class com.c.spark\_demo.SimpleApp --master yarn spark-demo-0.0.1-SNAPSHOT.jar

spark-submit --class com.cloudera.datascience.recommender.RunRecommender --master yarn-cluster --driver-class-path $(echo /opt/cloudera/parcels/CDH/jars/\*.jar |sed 's/ /:/g') ch03-recommender-2.0.0.jar

python版本

spark-submit --master local[4] SimpleApp.py

### Spark shell

spark-shell --master yarn --driver-class-path /opt/cloudera/parcels/CDH/jars/spark-sql\_2.10-1.5.0-cdh5.6.0.jar:/opt/cloudera/parcels/CDH/jars/spark-mllib\_2.10-1.5.0-cdh5.6.0.jar

### 监视:

http://<driver-node>:4040

### [Spark算子：RDD基本转换操作(3)–randomSplit、glom](http://lxw1234.com/archives/2015/07/343.htm)

## randomSplit

def randomSplit(weights: Array[Double], seed: Long = Utils.random.nextLong): Array[RDD[T]]

该函数根据weights权重，将一个RDD切分成多个RDD。

该权重参数为一个Double数组

第二个参数为random的种子，基本可忽略。

1. scala> var rdd = sc.makeRDD(1 to 10,10)
2. rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[16] at makeRDD at :21
4. scala> rdd.collect
5. res6: Array[Int] = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
7. scala> var splitRDD = rdd.randomSplit(Array(1.0,2.0,3.0,4.0))
8. splitRDD: **Array[org.apache.spark.rdd.RDD[Int]]** = Array(MapPartitionsRDD[17] at randomSplit at :23,
9. MapPartitionsRDD[18] at randomSplit at :23,
10. MapPartitionsRDD[19] at randomSplit at :23,
11. MapPartitionsRDD[20] at randomSplit at :23)
13. //这里注意：randomSplit的结果是一个RDD数组
14. scala> splitRDD.size
15. res8: Int = 4
16. //由于randomSplit的第一个参数weights中传入的值有4个，因此，就会切分成4个RDD,
17. //把原来的rdd按照权重1.0,2.0,3.0,4.0，随机划分到这4个RDD中，权重高的RDD，划分到//的几率就大一些。
18. //注意，权重的总和加起来为1，否则会不正常
20. scala> splitRDD(0).collect
21. res10: Array[Int] = Array(1, 4)
23. scala> splitRDD(1).collect
24. res11: Array[Int] = Array(3)
26. scala> splitRDD(2).collect
27. res12: Array[Int] = Array(5, 9)
29. scala> splitRDD(3).collect
30. res13: Array[Int] = Array(2, 6, 7, 8, 10)

## glom

def glom(): RDD[Array[T]]

该函数是将RDD中每一个分区中类型为T的元素转换成Array[T]，这样每一个分区就只有一个数组元素。

1. scala> var rdd = sc.makeRDD(1 to 10,3)
2. rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[38] at makeRDD at :21
3. scala> rdd.partitions.size
4. res33: Int = 3 //该RDD有3个分区
5. scala> rdd.glom().collect
6. res35: Array[Array[Int]] = Array(Array(1, 2, 3), Array(4, 5, 6), Array(7, 8, 9, 10))
7. //glom将每个分区中的元素放到一个数组中，这样，结果就变成了3个数组

转载请注明：[lxw的大数据田地](http://lxw1234.com) » [Spark算子：RDD基本转换操作(3)–randomSplit、glom](http://lxw1234.com/archives/2015/07/343.htm)

## SPARK 机器学习

### Spark On YARN内存分配

<http://blog.csdn.net/javastart/article/details/48877209>

<http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html>

### 美团推荐算法实践：机器学习重排序模型成亮点

http://www.csdn.net/article/2015-01-30/2823783

### spark单机部署问题

http://apache.fayea.com/spark/spark-1.6.1/spark-1.6.1-bin-hadoop2.6.tgz

**tar xfvz spark-1.6.1-bin-hadoop2.6.tgz**

**./bin/run-example org.apache.spark.examples.SparkPi**

1.端口不能绑定

15/02/27 16:14:36 INFO Remoting: Starting remoting

15/02/27 16:14:36 ERROR NettyTransport: failed to bind to bt-199-037.bta.net.cn/202.106.199.37:0, shutting down Netty transport

15/02/27 16:14:36 WARN Utils: Service 'sparkDriver' could not bind on port 0. Attempting port 1.

15/02/27 16:14:36 INFO RemoteActorRefProvider$RemotingTerminator: Shutting down remote daemon.

15/02/27 16:14:36 INFO RemoteActorRefProvider$RemotingTerminator: Remote daemon shut down; proceeding with flushing remote transports.

15/02/27 16:14:36 INFO RemoteActorRefProvider$RemotingTerminator: Remoting shut down.

15/02/27 16:14:36 INFO Slf4jLogger: Slf4jLogger started

15/02/27 16:14:36 INFO Remoting: Starting remoting

15/02/27 16:14:36 ERROR NettyTransport: failed to bind to bt-199-037.bta.net.cn/202.106.199.37:0, shutting down Netty transport

Exception in thread "main" java.net.BindException: Failed to bind to: bt-199-037.bta.net.cn/202.106.199.37:0: Service 'sparkDriver' failed after 16 retries!

        at org.jboss.netty.bootstrap.ServerBootstrap.bind(ServerBootstrap.java:272)

        at akka.remote.transport.netty.NettyTransport$$anonfun$listen$1.apply(NettyTransport.scala:393)

。。。

处理方法

从现象来看应该akka不能绑定到ip或者端口，从http://mail-archives.apache.org/mod\_mbox/spark-user/201402.mbox/<9A13072E9AA64A9B846FACA846FCA7C8@gmail.com>中找到方法。

spark-env.sh中添加配置:

export  SPARK\_MASTER\_IP=127.0.0.1

export  SPARK\_LOCAL\_IP=127.0.0.1

### Accessing publicly available datasets

* Fortunately, while commercially-sensitive data can be hard to come by, there are still a number of useful datasets available publicly. Many of these are often used as benchmark datasets for speci c types of machine learning problems. Examples of common data sources include:
* **UCI Machine Learning Repository**: This is a collection of almost 300 datasets of various types and sizes for tasks including classi cation, regression, clustering, and recommender systems. The list is available at http://archive.ics.uci.edu/ml/.
* **Amazon AWS public datasets**: This is a set of often very large datasets that can be accessed via Amazon S3. These datasets include the Human Genome Project, the Common Crawl web corpus, Wikipedia data, and Google Books Ngrams. Information on these datasets can be found at http://aws.amazon. com/publicdatasets/.
* **Kaggle**: This is a collection of datasets used in machine learning competitions run by Kaggle. Areas include classi cation, regression, ranking, recommender systems, and image analysis. These datasets can be found under the *Competitions* section at http://www.kaggle.com/competitions.
* **KDnuggets**: This has a detailed list of public datasets, including some of those mentioned earlier. The list is available at http://www.kdnuggets.com/ datasets/index.html.

### model serving for large-scale models, including recommenders based on matrix factorization.

Projects such as Oryx (https://github.com/OryxProject/ oryx) and Prediction.io (https://github.com/PredictionIO/ PredictionIO) focus on model serving for large-scale models, including recommenders based on matrix factorization.

### Clustering models have many use cases that are the same as classi cation; these include the following:

* Segmenting users or customers into different groups based on behavior characteristics and metadata
* Grouping content on a website or products in a retail business
* Finding clusters of similar genes
* Segmenting communities in ecology
* Creating image segments for use in image analysis applications such as object detection

### 安装IPython

<http://ipython.org/install.html>

http://ipython.org/ipython-doc/stable/interactive/ tutorial.html.

### 安装Anaconda 或者Enthought

http://continuum. io/downloads

https://store.enthought.com/ downloads/

#### anaconda 科学计算文档

<https://docs.continuum.io/anaconda/index>

#### API

<http://docs.scipy.org/doc/numpy/reference/generated/numpy.mean.html>

http://matplotlib.org/faq/howto\_faq.html#generate-images-without-having-a-window-appear

http://www.numpy.org

**IPYTHON=1 IPYTHON\_OPTS="--pylab" ./bin/pyspark**

**错误**

**ImportError**: No module named matplotlib

解决

pip install matplotlib

Command "python setup.py egg\_info" failed with error code 1 in /tmp/pip-build-x6dmj2/matplotlib/

yum install python-matplotlib

**RuntimeError: could not open display**

**错误**

### python cannot connect to X server

**解决：安装vnc 并启动服务**

**Linux学习之路--启动VNC服务**

**http://apluck.iteye.com/blog/830188**

### Scala

#### IDE

<http://scala-ide.org>

#### scala maven eclipse

http://jingpin.jikexueyuan.com/article/47043.html

## Hadoop

### namenode gc导致的故障一例

http://caiguangguang.blog.51cto.com/1652935/1418895

### webHDFS设置和使用说明

<http://blog.csdn.net/iloveyin/article/details/28264027>

curl -i <http://192.168.21.34:50070/webhdfs/v1/user/hive?op=LISTSTATUS>

curl -i -L <http://192.168.21.34:50070/webhdfs/v1/user/flume/Games/tzsg/pay/2016-01-16/01/event_2016-01-16_01.1453694419464.log?op=OPEN>

### hadoop 权威指南

http://shop.oreilly.com/product/0636920021773.do

### HDFS

https://issues.apache.org/jira/secure/attachment/12679296/HDFS-7314-2.patch

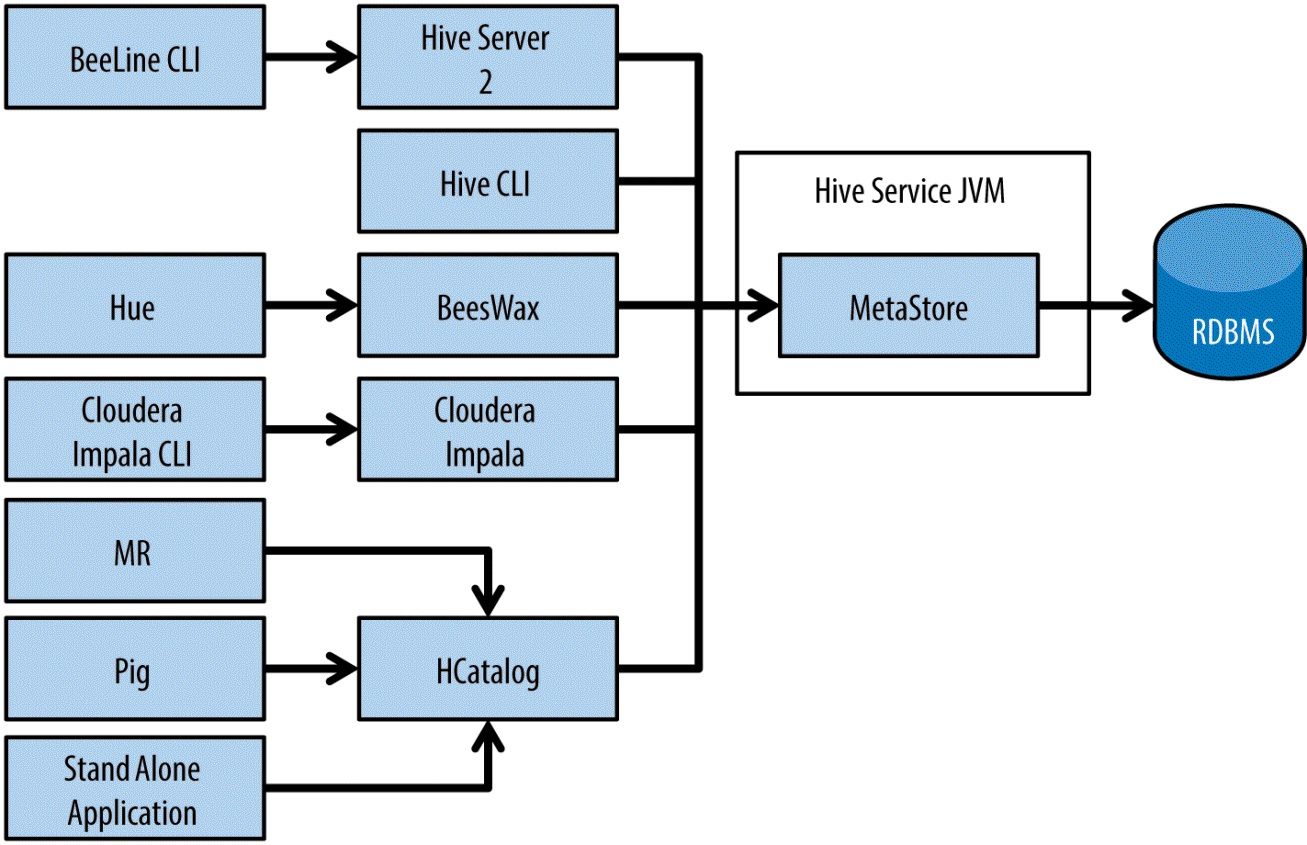
### 利用JavaAPI访问HDFS的文件

<http://blog.csdn.net/zhangzhaokun/article/details/5597433>

<http://www.cnblogs.com/xuqiang/archive/2011/06/03/2042526.html>

### Hadoop应用程序架构

#### Hive 元数据储存



#### 数据移动 Data Moving

hadoop fs –put command will copy a file over and do a full checksum to confirm that the data is copied over correctly.

One consideration when using the hdfs fs -put command or Sqoop is that the data will land on HDFS in a format that might not be optimal for long-term storage and processing,

so using these tools might require an additional batch process to get the data into the desired format

An example of where such an additional batch process would be required

is loading Gzip files into HDFS. Although Gzip files can easily be stored in HDFS and processed with MapReduce or other processing frameworks on Hadoop, as we discussed in the previous chapter Gzip files are not splittable. This will greatly impact the efficiency of processing these files, particularly as the files get larger. In this case a good solution is to store the files in Hadoop using a container format that supports splittable compression, such as SequenceFiles or Avro.

As your requirements move from batch processing to more frequent updates, you should consider tools like Flume or Kafka

Sqoop and file transfers are not going to be a good selection, as the delivery requirements get shorter than two minutes

Further, as the requirements become shorter than two minutes, the storage layer may need to change to HBase or Solr for more granular insertions and read operations

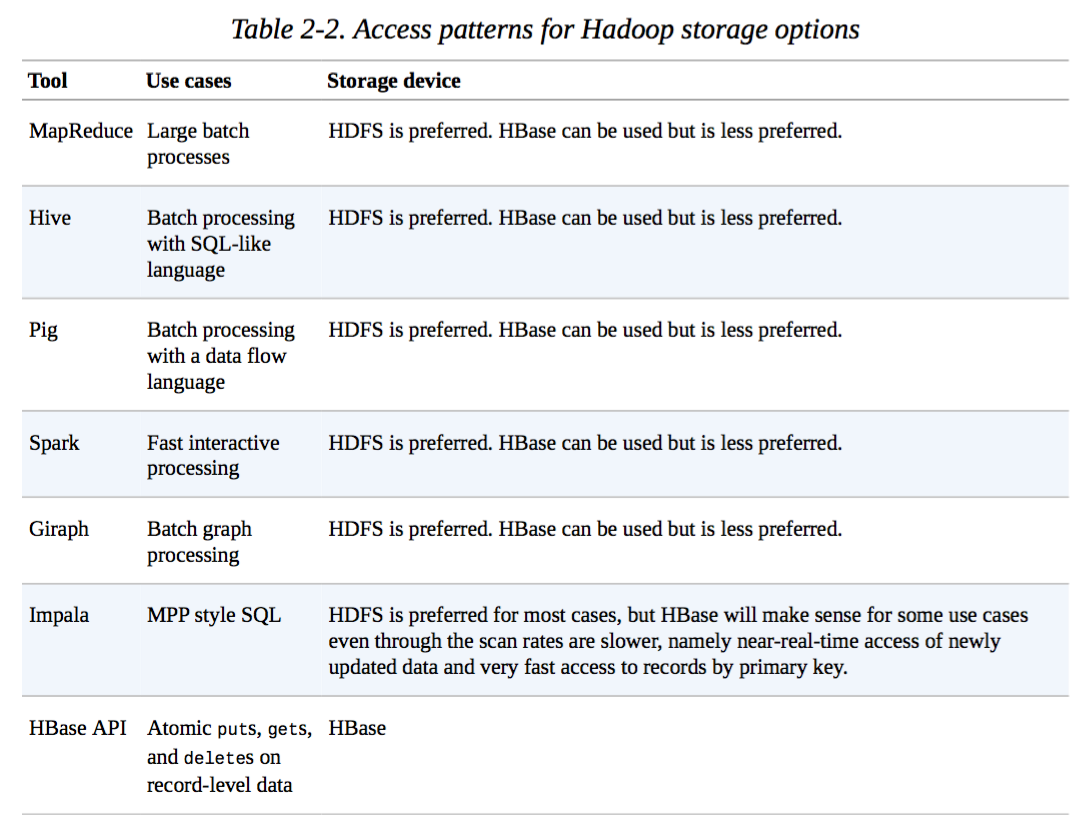
There are a number of reasons to prefer large files, one of the major reasons being how data is read from disk. Using a long consecutive scan to read a single file is faster than performing many seeks to read the same information from multiple files.

We briefly discuss methods to manage small files later in this chapter, but a full discussion is beyond the scope of this book. See Hadoop: The Definitive Guide or Hadoop in Practice for detailed discussions on techniques to manage small files.

##### HBase 跟HDFS的不同

HBase has much different access patterns than HDFS that should be considered — for example, scan rates. HBase scan rates are about 8–10 times slower than HDFS. Another difference is random access; HBase can access a single record in milliseconds, whereas HDFS doesn’t support random access other than file seeking, which is expensive and often complex.

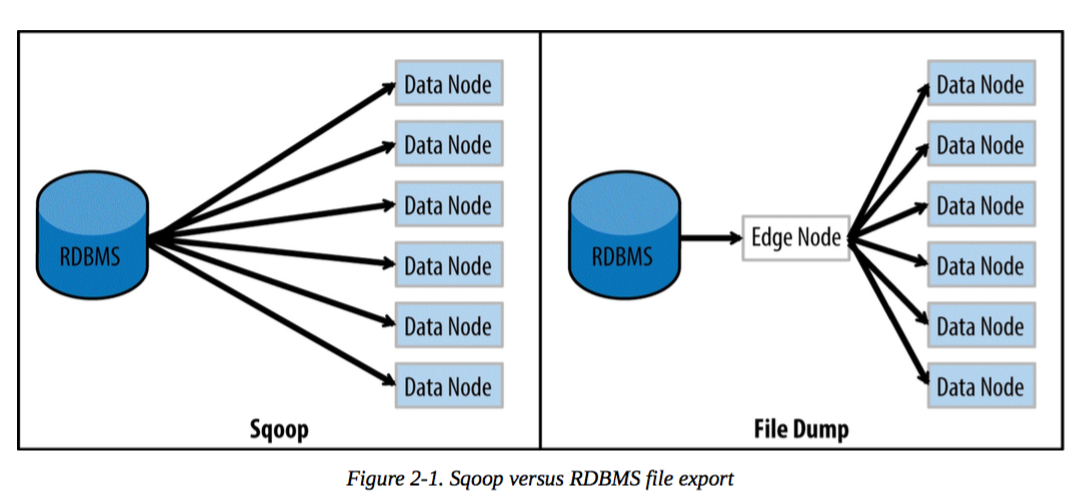
How is the data going to be used once it is in Hadoop? For example: if the requirements call for random row access, HDFS may not be the best fit, and HBase might be a better choice. Conversely, if scans and data transformations are required, HBase may not be a good selection



a splittable container format like SequenceFiles, Parquet files, or Avro files

Sqoop is a batch process, so if the timeliness of the data load into the cluster needs to be faster than batch, you’ll likely have to find an alternate method

##### Sqoop 从关系数据库读取数据到HDFS的两种方法



##### Push or Pull

All the tools discussed in this chapter can be classified as either pushing or pulling tools. The important thing to note is the actor in the architecture because in the end that actor will have additional requirements to consider, such as:

Keeping track of what has been sent Handling retries or failover options in case of failure Being respectful of the source system that data is being ingested from Access and security

We’ll cover these requirements in more detail in this chapter, but first we’ll discuss two common Hadoop tools — Sqoop and Flume — to help clarify the distinction between push and pull in the context of data ingestion and extraction with Hadoop.

a hadoop fs -put <filename> command. When the put command has finished, the command will have validated that the file is in HDFS, replicated three times, and passed a checksum check

##### the characteristics of file transfers with Hadoop:

1、It’s an all-or-nothing batch processing approach, so if an error occurs during file transfer no data will be written or read. This should be contrasted with ingestion methods such as Flume or Kafka, which provide some level of failure handling and guaranteed delivery.

2、By default file transfers are single-threaded; it’s not possible to parallize file transfers. File transfers are from a traditional filesystem to HDFS.

3、Applying transformations to the data is not supported; data is ingested into HDFS as is. Any processing of the data needs to be done after it lands in HDFS, as opposed to in- flight transformations that are supported with a system like Flume.

4、It is a byte-by-byte load, so any types of file can be transferred (text, binary, images, etc.).



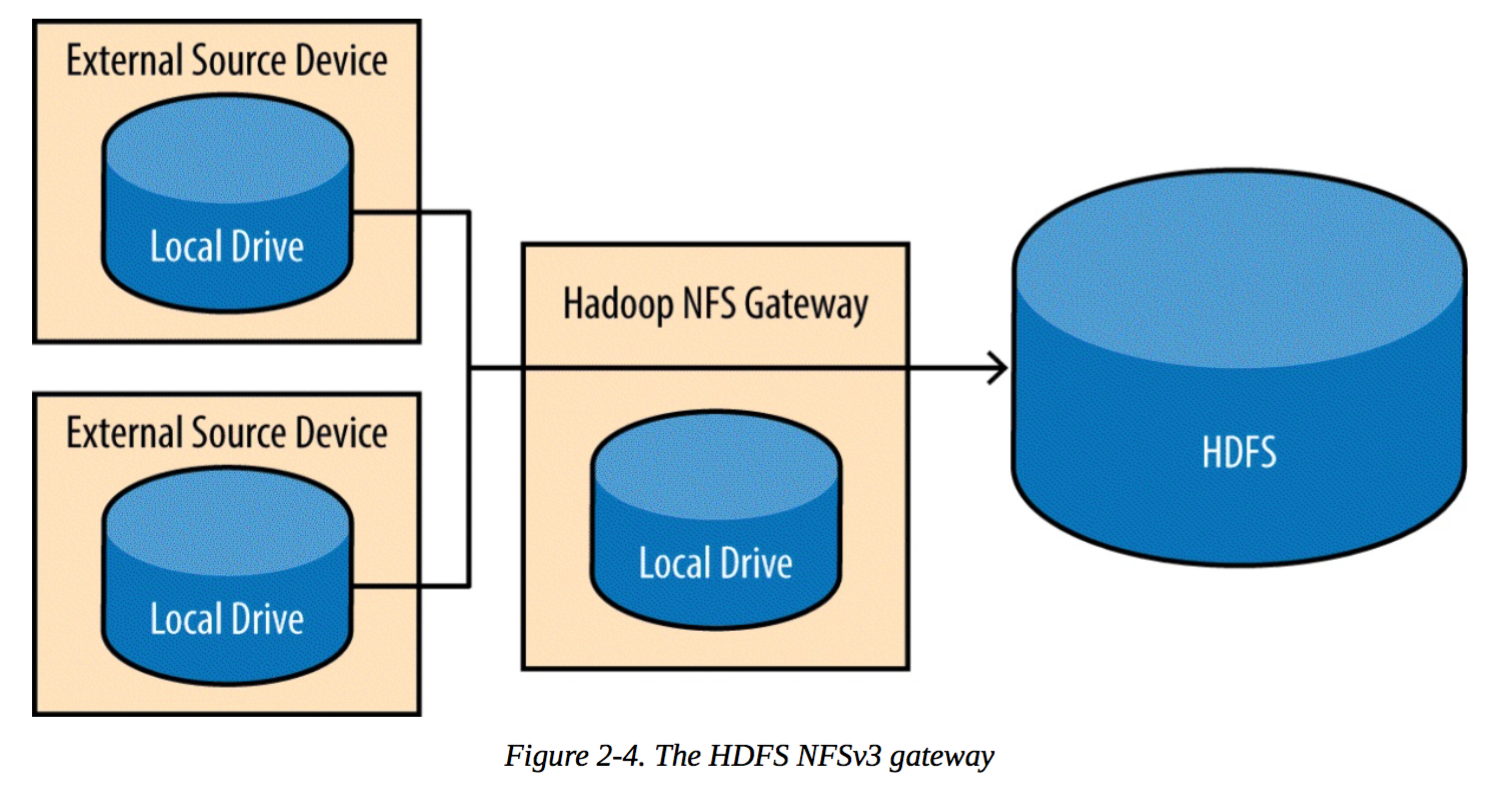
##### 小文件储存策略

Note that “many small files” could mean millions of files in a reasonably sized cluster. Still, for storage and processing efficiency it’s better to store fewer large files in Hadoop. If there is a need to ingest many small files into Hadoop, there are several approaches that can be used to mitigate this:

Use Solr for storing and indexing the small files. We’ll discuss Solr in more detail in Chapter 7.

Use HBase to store the small files, using the path and filename as the key. We’ll also discuss HBase in more detail in Chapter 7.

Use a container format such as SequenceFiles or Avro to consolidate small files.



##### Considerations for File Transfers versus Other Ingest Methods

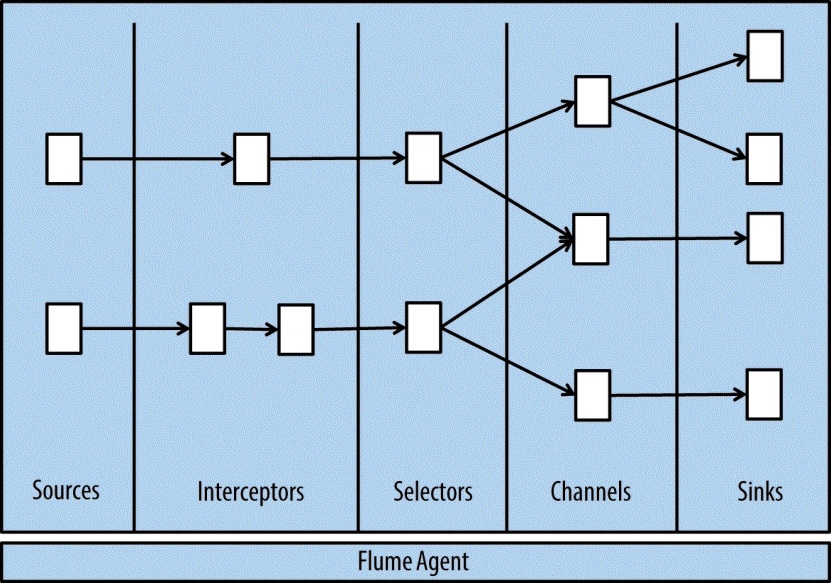
·Do you need to ingest data into multiple locations? For example, do you need to ingest data into both HDFS and Solr, or into HDFS and HBase? In this case using a file transfer will require additional work after the files are ingested, and using Flume is likely more suitable.

·Is reliability important? If so, remember that an error mid-transfer will require a restart of the file transfer process. Here again, Flume is likely a more appropriate solution.

·Is transformation of the data required before ingestion? In that case, Flume is almost certainly the correct tool.

As we’ll discuss shortly, one reason to specify a parameter for split-by is to avoid data skew. Note that each mapper will have its own connection to the database, and each will retrieve its portion of the table when we specify its portion limits in a where clause. It is important to choose a split column that has an index or is a partition key to avoid each mapper having to scan the entire table. If no such key exists, specifying only one mapper is the preferred solution.

Flume architecture



· Flume sources consume events from external sources and forward to channels. These external sources could be any system that generates events, such as a social media feed like Twitter, machine logs, or message queues. Flume sources are implemented to consume events from specific external sources, and many sources come bundled with Flume, including AvroSource, SpoolDirectorySource, HTTPSource, and JMSSource.

·Flume interceptors allow events to be intercepted and modified in flight. This can be transforming the event, enriching the event, or basically anything that can be implemented in a Java class. Some common uses of interceptors are formatting,

partitioning, filtering, splitting, validating, or applying metadata to events.

· Selectors provide routing for events. Selectors can be used to send events down zero or more paths, so they are useful if you need to fork to multiple channels, or send to a specific channel based on the event.

·Flume channels store events until they’re consumed by a sink. The most commonly used channels are the memory channel and the file channel. The memory channel stores events in memory, which provides the best performance among the channels, but also makes it the least reliable, because events will be lost if the process or host goes down. More commonly used is the disk channel, which provides more durable storage of events by persisting to disk. Choosing the right channel is an important architectural decision that requires balancing performance with durability.

·Sinks remove events from a channel and deliver to a destination. The destination could be the final target system for events, or it could feed into further Flume processing. An example of a common Flume sink is the HDFS sink, which, as its name implies, writes events into HDFS files.

·The Flume agent is a container for these components. This is a JVM process hosting a set of Flume sources, sinks, channels, and so on.

### Hadoop 文件系统多次关闭的问题

http://os.51cto.com/art/201305/394782.htm

## NOSQL

### ArangoDB，MongoDB，Neo4j 和 OrientDB 性能比较

http://www.oschina.net/news/63333/performance-comparison-between-arangodb

### Neo4j

Mac 安装

brew install neo4j

启动 neo4j start

<http://localhost:7474/>

### [neo4j使用指南](http://blog.csdn.net/gtuu0123/article/details/6384375)

<http://blog.csdn.net/gtuu0123/article/details/6384375>

### GemFire

#### [GemFire 入门篇2：GemFire 8.2 在CentOS & Mac OS X的安装步骤](http://www.cnblogs.com/doubletree/p/5185395.html)

<http://www.cnblogs.com/doubletree/archive/2016/02/09/5185395.html>

#### [Pivotal Gemfire gfsh shell的使用（1）](http://blog.csdn.net/cars_star/article/details/46832409)

http://blog.csdn.net/cars\_star/article/details/46832409

<http://baike.baidu.com/link?url=O3pT6n1QXrbKc9RgbA2TVGjNcQQEWdJwizE4Aa0LDvC0hYj3CYJ8CrgVK4xlKBFdSebrSxWntWmDSpuFQxql6K>

cdh5

<http://archive.cloudera.com/cdh5/cdh/5/>

<http://www.cloudera.com/content/www/en-us/downloads.html>

## flume

tier1.sources = r1

tier1.channels = c1 c2 c3

tier1.sinks = k1 k2 k3 k4

tier1.sources.r1.channels = c1 c2 c3

#c1:hdfs c2:kafka c3:hbase

tier1.sources.r1.selector.type = multiplexing

tier1.sources.r1.selector.header = selectorHeader

tier1.sources.r1.selector.default = c3

tier1.sources.r1.selector.mapping.HDFS = c1

tier1.sources.r1.selector.mapping.KAFKA = c2

tier1.sources.r1.selector.mapping.HBASE = c3

tier1.sources.r1.selector.mapping.KAHB = c2 c3

tier1.sources.r1.selector.mapping.HDHB = c1 c3

tier1.sources.r1.type = http

tier1.sources.r1.bind = slave1

#修改为每台服务器的主机名称

tier1.sources.r1.port = 9922

#define the sink k1

tier1.sinks.k1.channel = c1

tier1.sinks.k1.type = hdfs

tier1.sinks.k1.hdfs.path = hdfs://cndwservice1:8020/user/flume/Games/%{gameId}/%{logType}/%Y-%m-%d

tier1.sinks.k1.hdfs.filePrefix = event\_%Y-%m-%d\_%H\_%M

tier1.sinks.k1.hdfs.fileSuffix = .log

#单个文件128M

tier1.sinks.k1.hdfs.rollSize = 134217728

tier1.sinks.k1.hdfs.rollCount = 0

#tier1.sinks.k1.hdfs.batchSize = 10000

tier1.sinks.k1.hdfs.round = true

tier1.sinks.k1.hdfs.roundValue=60

tier1.sinks.k1.hdfs.roundUnit = minute

#tier1.sinks.k1.serializer = TEXT

#tier1.sinks.k1.hdfs.threadsPoolSize = 25

tier1.sinks.k1.hdfs.useLocalTimeStamp = false

tier1.sinks.k1.hdfs.minBlockReplicas = 1

tier1.sinks.k1.hdfs.fileType = DataStream

tier1.sinks.k1.hdfs.writeFormat = Text

tier1.sinks.k1.hdfs.rollInterval = 60

#define the sink k2

tier1.sinks.k2.channel=c2

tier1.sinks.k2.type=org.apache.flume.sink.kafka.KafkaCndwSink

tier1.sinks.k2.brokerList=slave1:9092,slave2:9092,slave3:9092

tier1.sinks.k2.requiredAcks=1

tier1.sinks.k2.topic=mytopic

tier1.sinks.k2.batchSize=500000

#define the sink k3

tier1.sinks.k3.channel = c3

tier1.sinks.k3.type = org.apache.flume.sink.hbase.DvAsyncHBaseSink

tier1.sinks.k3.zookeeperQuorum = master1:2181,master2:2181,slave1:2181,slave2:2181,slave3:2181

tier1.sinks.k3.znodeParent = /hbase

tier1.sinks.k3.table = %table

tier1.sinks.k3.columnFamily = info

tier1.sinks.k3.serializer = org.apache.flume.sink.hbase.DvSplittingAsyncHbaseEventSerializer

tier1.sinks.k3.batchSize = 500000

tier1.sinks.k3.requiredAcks=1

tier1.sinks.k4.channel = c3

tier1.sinks.k4.type = org.apache.flume.sink.hbase.DvAsyncHBaseSink

tier1.sinks.k4.zookeeperQuorum = master1:2181,master2:2181,slave1:2181,slave2:2181,slave3:2181

tier1.sinks.k4.znodeParent = /hbase

tier1.sinks.k4.table = %table

tier1.sinks.k4.columnFamily = info

tier1.sinks.k4.serializer = org.apache.flume.sink.hbase.DvSplittingAsyncHbaseEventSerializer

tier1.sinks.k4.batchSize = 500000

tier1.sinks.k4.requiredAcks=1

tier1.sinkgroups = g1

tier1.sinkgroups.g1.sinks = k3 k4

tier1.sinkgroups.g1.processor.type = failover

tier1.sinkgroups.g1.processor.priority.k3 = 5

tier1.sinkgroups.g1.processor.priority.k4 = 10

tier1.sinkgroups.g1.processor.maxpenalty = 10000

#define the channel c1

tier1.channels.c1.type = memory

tier1.channels.c1.capacity=10000000

tier1.channels.c1.transactionCapacity=10000000

tier1.channels.c1.keep-alive=20

#define the channel c2

tier1.channels.c2.type = memory

tier1.channels.c2.capacity=10000000

tier1.channels.c2.transactionCapacity=10000000

tier1.channels.c2.keep-alive=20

#define the channel c3

tier1.channels.c3.type = memory

tier1.channels.c3.capacity=10000000

tier1.channels.c3.transactionCapacity=10000000

tier1.channels.c3.keep-alive=20

tier1.channels.memoryChannel.capacity = 30000000

https://cwiki.apache.org/confluence/display/Hive/GettingStarted

启动

bin/flume-ng agent --conf conf --conf-file conf/thrift-logger.conf --name agent1 -Dflume.root.logger=INFO,console

curl -X POST -d ‘[{ "headers" : { "timestamp" : "434324343", "host" :"random\_host.example.com", "field1" : "val1" }, "body" : "random\_body" }]‘ localhost:9000

### thrift 到 hdfs

tier1.sources = r1

tier1.channels = c1

tier1.sinks = k1

#tier1.sources.r1.interceptors = i1

#tier1.sources.r1.interceptors.i1.type = org.apache.flume.interceptor.TimestampInterceptor$Builder

# For each one of the sources, the type is defined

tier1.sources.r1.type = thrift

tier1.sources.r1.channels = c1

tier1.sources.r1.bind = slave1

tier1.sources.r1.port = 9922

# Each sink's type must be defined

tier1.sinks.k1.channel = c1

tier1.sinks.k1.type = hdfs

tier1.sinks.k1.hdfs.path = hdfs://master1:8020/user/flume

tier1.sinks.k1.hdfs.filePrefix = event\_%y-%m-%d

tier1.sinks.k1.hdfs.fileSuffix = .log

tier1.sinks.k1.hdfs.rollSize = 0

tier1.sinks.k1.hdfs.rollCount = 0

#tier1.sinks.k1.hdfs.batchSize = 1

tier1.sinks.k1.hdfs.round = true

tier1.sinks.k1.hdfs.roundValue=24

tier1.sinks.k1.hdfs.roundUnit = hour

#tier1.sinks.k1.hdfs.threadsPoolSize = 25

tier1.sinks.k1.hdfs.useLocalTimeStamp = true

tier1.sinks.k1.hdfs.minBlockReplicas = 1

tier1.sinks.k1.fileType = DataStream

tier1.sinks.k1.writeFormat = TEXT

tier1.sinks.k1.rollInterval = 60

# Each channel's type is defined.

tier1.channels.c1.type = memory

# Other config values specific to each type of channel(sink or source)

# can be defined as well

# In this case, it specifies the capacity of the memory channel

tier1.channels.memoryChannel.capacity = 100

### flume-hdfs 按照时间关闭并新开文件

<http://www.bubuko.com/infodetail-1086730.html>

### flume-ng依据源文件名输出到HDFS指定文件名

<http://www.dzend.com/forum.php?mod=viewthread&tid=611>

### 【Flume】flume中sink到hdfs，文件系统频繁产生文件，文件滚动配置不起作用？

<http://doc.okbase.net/chiweitree/archive/126197.html>

### flume学习（八）：自定义source

<http://blog.csdn.net/xiao_jun_0820/article/details/38312091>

### FLUME-HBase-SINK

错误BUG

#### Could not write events to Hbase. Transaction failed, and rolled back.

<https://issues.cloudera.org/browse/DISTRO-684?page=com.atlassian.jira.plugin.system.issuetabpanels:all-tabpanel>

https://github.com/OpenTSDB/opentsdb/issues/62

## Hive

<https://cwiki.apache.org/confluence/display/Hive/GettingStarted>

这里罗列常用操作，更多参考 <https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-Create%2FDrop%2FTruncateTable>

### 简单的创建表

create table table\_name (

id int,

dtDontQuery string,

name string

)

### 创建有分区的表

create table table\_name (

id int,

dtDontQuery string,

name string

)

partitioned by (date string)

一个表可以拥有一个或者多个分区，每个分区以文件夹的形式单独存在表文件夹的目录下。

分区是以字段的形式在表结构中存在，通过describe table命令可以查看到字段存在，但是该字段不存放实际的数据内容，仅仅是分区的表示。

在Hive Select查询中一般会扫描整个表内容，会消耗很多时间做没必要的工作。有时候只需要扫描表中关心的一部分数据，因此建表时引入了partition概念。表中的一个 Partition 对应于表下的一个目录,Partition 就是辅助查询，缩小查询范围，加快数据的检索速度和对数据按照一定的规格和条件进行管理。

### 典型的默认创建表

[](javascript:void(0);)

CREATE TABLE page\_view(

viewTime INT,

userid BIGINT,

page\_url STRING,

referrer\_url STRING,

ip STRING COMMENT 'IP Address of the User')

COMMENT 'This is the page view table'

PARTITIONED BY(dt STRING, country STRING)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\001'

COLLECTION ITEMS TERMINATED BY '\002'

MAP KEYS TERMINATED BY '\003'

STORED AS TEXTFILE;

[](javascript:void(0);)

这里创建了表page\_view,有表的注释，一个字段ip的注释，分区有两列,分别是dt和country。

[ROW FORMAT DELIMITED]关键字，是用来设置创建的表在加载数据的时候，支持的列分隔符。不同列之间用一个'\001'分割,集合(例如array,map)的元素之间以'\002'隔开,map中key和value用'\003'分割。

[STORED AS file\_format]关键字是用来设置加载数据的数据类型,默认是TEXTFILE，如果文件数据是纯文本，就是使用 [STORED AS TEXTFILE]，然后从本地直接拷贝到HDFS上，hive直接可以识别数据。

### 常用的创建表

[](javascript:void(0);)

CREATE TABLE login(

userid BIGINT,

ip STRING,

time BIGINT)

PARTITIONED BY(dt STRING)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t'

STORED AS TEXTFILE;

[](javascript:void(0);)

 LOAD DATA LOCAL INPATH './examples/files/kv1.txt' OVERWRITE INTO TABLE pokes;

### 创建外部表

如果数据已经存在HDFS的'/user/hadoop/warehouse/page\_view'上了，如果想创建表，指向这个路径，就需要创建外部表:

[](javascript:void(0);)

CREATE EXTERNAL TABLE page\_view(

viewTime INT,

userid BIGINT,

page\_url STRING,

referrer\_url STRING,

ip STRING COMMENT 'IP Address of the User',

country STRING COMMENT 'country of origination')

COMMENT 'This is the staging page view table'

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\054'

STORED AS TEXTFILE

LOCATION '/user/hadoop/warehouse/page\_view';

[](javascript:void(0);)

创建表，有指定EXTERNAL就是外部表，没有指定就是内部表，内部表在drop的时候会从HDFS上删除数据，而外部表不会删除。

外部表和内部表一样，都可以有分区，如果指定了分区，那外部表建了之后，还要修改表添加分区。

外部表如果有分区，还可以加载数据，覆盖分区数据，但是外部表删除分区，对应分区的数据不会从HDFS上删除，而内部表会删除分区数据。

### 指定数据库创建表

如果不指定数据库，hive会把表创建在default数据库下，假设有一个hive的数据库mydb,要创建表到mydb,如下:

CREATE TABLE mydb.pokes(foo INT,bar STRING);

或者是

use mydb; --把当前数据库指向mydb

CREATE TABLE pokes(foo INT,bar STRING);

### 复制表结构

CREATE TABLE empty\_table\_name LIKE table\_name;

根据table\_name创建一个空表empty\_table\_name,empty\_table\_name没有任何数据。

**create-table-as-selectt (CTAS)**

CTAS创建的表是原子性的，这意味着，该表直到所有的查询结果完成后，其他用户才可以看到完整的查询结果表。

CTAS唯一的限制是目标表，不能是一个有分区的表，也不能是外部表。

简单的方式

CREATE TABLE new\_key\_value\_store

AS

SELECT (key % 1024) new\_key, concat(key, value) key\_value\_pair FROM key\_value\_store;

复杂的方式

CREATE TABLE new\_key\_value\_store

ROW FORMAT SERDE "org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe"

STORED AS RCFile AS

SELECT (key % 1024) new\_key, concat(key, value) key\_value\_pair

FROM key\_value\_store

SORT BY new\_key, key\_value\_pair;

### 删除表

DROP TABLE table\_name;

DROP TABLE IF EXISTS table\_name;

删除表会移除表的元数据和数据，而HDFS上的数据，如果配置了Trash，会移到.Trash/Current目录下。

删除外部表时，表中的数据不会被删除。

### 截断表

TRUNCATE TABLE table\_name;

TRUNCATE TABLE table\_name PARTITION (dt='20080808');

从表或者表分区删除所有行，不指定分区，将截断表中的所有分区，也可以一次指定多个分区，截断多个分区。

### Hive之partition

时间 2014-06-25 11:01:32 [ITeye-博客](http://www.tuicool.com/sites/jaeeAn)

原文

[http://snv.iteye.com/blog/2085030](http://snv.iteye.com/blog/2085030?utm_source=tuicool&utm_medium=referral)

主题 [Hive](http://www.tuicool.com/topics/11020065) [HDFS](http://www.tuicool.com/topics/11030091) [Nosql](http://www.tuicool.com/topics/11000065)

概述

hive的 partition可以认为是 RMDB中的分区，目的是query时减少全表扫描。

使用：

创建分区：

create EXTERNAL TABLE IF NOT EXISTS p(

id STRING COMMENT 'id'

) partitioned by (seq int)

STORED AS SEQUENCEFILE

LOCATION 'hdfs:///hive/hdfs/p/';

当数据插入的时候会以hdfs:///hive/hdfs/p/为基础构建以分区=值为名的目录

使用分区：

insert into table p partition(seq=1) select count(1) from dual;

此时会形成hdfs:///hive/hdfs/p/seq=1/数据 的文件目录格式，在insert into的时候需要指定分区否则会报如下错误：

Need to specify partition columns because the destination table is partitioned

查询操作：

select \* from p where seq=1

可以使用多个分区：

create EXTERNAL TABLE IF NOT EXISTS pp(

id STRING COMMENT 'id'

) partitioned by (seq int,num int)

STORED AS SEQUENCEFILE

LOCATION 'hdfs:///hive/hdfs/pp/';

此时如果写入数据指定了分区那么就会形成 hdfs:///hive/hdfs/pp/seq=值/num=值/数据 的目录结构

查询分区：

可以查询某表具有的分区：

show partitions pp;

 可以基于分区删除数据：

ALTER TABLE pp DROP PARTITION (seq=1,num=1);

pp为外部表那么其元数据被删除，真实数据扔存在，可以通过hadoop fs查看到

ps：

1.执行insert overwrite 的时候指定分区中的数据会被删除，其他分区数据不受影响

2.一个表可以有多个分区，分区列是伪劣不是真实存在的列以目录的形式存在

### 创建索引

create index user\_index on table user(id)

alter index user\_index on user rebuild;

### 在HIVE中使用AVRO

create EXTERNAL table avrotest(gameId string, gameName string)

ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'

STORED AS INPUTFORMAT 'org.apache.hadoop.hive.ql.io.avro.AvroContainerInputFormat'

OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.avro.AvroContainerOutputFormat'

LOCATION '/user/hive/avro/'

TBLPROPERTIES ('avro.schema.literal'='{

"name": "my\_record",

"type": "record",

"fields": [

{"name":"gameId", "type":"string"},

{"name":"gameName", "type":"string"}]}');

CREATE TABLE test2(name1 STRING,name2 STRING)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' COLLECTION ITEMS TERMINATED BY ':' MAP KEYS TERMINATED BY '#' LINES TERMINATED BY '\n'

STORED AS TEXTFILE;

LOCATION '/user/hive/avro/'

## Impala

<http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/impala_tutorial.html>

<http://www.cloudera.com/content/www/en-us/documentation/archive/impala/2-x/2-1-x/topics/impala_jdbc.html>

http://impala.io/doc/html/Planner\_8java\_source.html

impala-shell -i localhost –quiet

导入数据直接通过hive来导入比较好，现在hive中建张表，制定分隔符， 然后再 load data local inpath ......overwrite into table .......

然后impala在 refresh 那张表 或者 invalidate metadata 后。 查询。

### 导入数据

<http://blog.itpub.net/10037372/viewspace-1411981/>

### 性能调优

http://my.oschina.net/weiqingbin/blog/192612#OSC\_h3\_12

采集表的统计信息用于查询优化

在impala 在使用

compute stats tablename;

在hive中使用

为整个表或特定分区在 Hive 中执行 ANALYZE TABLE 语句：

ANALYZE TABLEtablename[PARTITION(partcol1[=val1],partcol2[=val2], ...)] COMPUTE STATISTICS [NOSCAN];

例如，为非分区表采集统计信息：

ANALYZE TABLE customer COMPUTE STATISTICS;

为以 state 和 city 分区列的分区表 store 表采集所有分区的统计信息：

ANALYZE TABLE store PARTITION(s\_state, s\_county) COMPUTE STATISTICS;

只采集分区表 store 中 California 分区的统计信息：

ANALYZE TABLE store PARTITION(s\_state='CA', s\_county) COMPUTE STATISTICS;

### Impala SQL 语言元素

http://my.oschina.net/weiqingbin/blog/189413?fromerr=E9WBdcGh#OSC\_h2\_28

### Impala 表使用 Avro 文件格式（翻译）

<http://my.oschina.net/weiqingbin/blog/194411?fromerr=HVOsFIBf>

### CM配置impala使用本地时间

**Clusters** > **Services** > **Impala** > **Configuration** > **Impala Daemon > Impala Daemon 命令行参数高级配置代码段（安全阀）**

-use\_local\_tz\_for\_unix\_timestamp\_conversions=true

### Cloudera Impala 常见问题（翻译）

<http://my.oschina.net/weiqingbin/blog/196143#OSC_h3_29>

### 删除数据库

drop database if exists debug\_dxqq cascade

## HBase

<http://javacrazyer.iteye.com/blog/1186881>

### Impala与HBase整合实践

<http://shiyanjun.cn/archives/526.html>

### Hbase 存储格式

<http://blog.itpub.net/15711267/viewspace-1063571/>

### Hive 表映射到 Hbase

创建Hbase 表

>hbase shell

创建表

>create 'test','f1'

> create 'cn\_games',{NAME => 'dxqq', VERSIONS => 1, TTL => 86400}

alter 'cn\_games',{NAME=>'dxqq\_missioninfo',VERSIONS => 1, TTL => 86400}

alter 'cn\_games',{NAME=>'dxqq\_payrequestinfo', VERSIONS => 1, TTL => 86400}

alter 'cn\_games',{NAME=>'dxqq\_paysucinfo', VERSIONS => 1, TTL => 86400}

alter 'cn\_games',{NAME=>'dxqq\_rewardinfo', VERSIONS => 1, TTL => 86400}

alter 'cn\_games',{NAME=>'dxqq\_rolecancelinfo', VERSIONS => 1, TTL => 86400}

alter 'cn\_games',{NAME=>'dxqq\_rolecreateinfo', VERSIONS => 1, TTL => 86400}

alter 'cn\_games',{NAME=>'dxqq\_rolelogininfo', VERSIONS => 1, TTL => 86400}

alter 'cn\_games',{NAME=>'dxqq\_rolelvupinfo', VERSIONS => 1, TTL => 86400}

alter 'cn\_games',{NAME=>'dxqq\_serverselinfo', VERSIONS => 1, TTL => 86400}

alter 'cn\_games',{NAME=>'dxqq\_userbrowserinfo', VERSIONS => 1, TTL => 86400}

alter 'cn\_games',{NAME=>'dxqq\_userlogininfo', VERSIONS => 1, TTL => 86400}

alter 'cn\_games',{NAME=>'dxqq\_userlogoutinfo', VERSIONS => 1, TTL => 86400}

alter 'cn\_games',{NAME=>'dxqq\_userreginfo', VERSIONS => 1, TTL => 86400}

在原表增加新的列簇

> alter "test",{NAME=>'f3'}

将表hbase test 表 f3 映射到 hive中

CREATE EXTERNAL TABLE IF NOT EXISTS test.user2(rkey string,id int,name string)ROW FORMAT SERDE 'org.apache.hadoop.hive.hbase.HBaseSerDe'

STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'

WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,f3:id,f3:name")

TBLPROPERTIES("hbase.table.name" = "test")

## Kafka

### Kafka+Storm+HDFS整合实践

<http://shiyanjun.cn/archives/934.html>

## AVRO

<https://github.com/sidshetye/HelloAvro/tree/master/Avro/Lib>

### 在Hive中使用Avro

http://www.iteblog.com/archives/1007

### Hue

<http://demo.gethue.com>

### Sqoop

#### Sqoop中文手册

<http://www.zihou.me/html/2014/01/28/9114.html>

## Storm

### 基于HBase做Storm 实时计算指标存储

<http://ifeve.com/getting-started-with-stom-index/>

### 《Storm入门》中文版

<http://ifeve.com/getting-started-with-stom-index/>

### 使用Storm实现实时大数据分析

<http://qq85609655.iteye.com/blog/2035717>

### 基于HBase做Storm 实时计算指标存储

<http://developer.51cto.com/art/201510/493422.htm>

### RealTimeTraffic

<https://github.com/whughchen/RealTimeTraffic/tree/master/src/storm/realTraffic/spout>

### Storm 常见模式

#### 批处理

<http://www.cnblogs.com/panfeng412/archive/2012/06/19/storm-common-patterns-of-batching.html>

## Solr

http://lucene.apache.org/solr/

### Solr’s features:

*Inverted index*: Lucene builds an inverted index of the documents that you add to Solr, and at query time it searches the index for matching documents. You can think of an inverted index as similar to the index at the end of this book.

*Vector space model*: By default, Lucene uses the vector space model (VSM) along with the Boolean model to determine the relevance of a document with respect to a user query. In a nutshell, the Boolean model approves, and the VSM ranks.

*Config-based*: solrconfig.xml and schema.xml are the two primary configuration files of Solr. The schema.xml file primarily defines the fields of your schema and the behavior of those fields (how the text will be tokenized while indexing and querying). Almost everything else goes in solrconfig.xml. You can also go schemaless and let Solr create fields automatically while indexing the data. The configurations can be edited manually or modified dynamically by calling the respective APIs. As of Solr 5.0, you can even upload JAR files through an API call.

*Analysis chain*: Your search query and the documents being indexed go through a chain of analyzers and tokenizers (the output of one tokenizer is fed to another in the chain). The output of the last tokenizer is the term that is indexed and matched against.

*Java*: Solr and Lucene are written in Java. Solr 5.0 requires Java 7+ to run. To customize any feature of Solr, you need to extend the appropriate Java class.

*SolrJ*: Solr bundles a Java client library that can be used for indexing documents and querying for results. Libraries are also available for other languages such as Perl and Python.

### What Makes Apache Solr So Popular

*Lucene*: Solr uses the Lucene search library at its core and wraps it to add features and expose it as a RESTful service that can be accessed over HTTP. Development of Solr and Lucene merged in March 2010, and both the code bases reside in the same trunk in Apache Subversion (SVN); hence you are sure to get all the latest Lucene features in your latest Solr release.

*Highly scalable and fault-tolerant*: You can add or remove computing capacity to Solr, just by adding or removing replicas of your instance as needed. SolrCloud even abstracts your application from knowing how the data is distributed and saves you from getting into further nuances such as load balancing and sharding. Data indexed to Solr can be replicated among multiple instances; so even if one instance goes down, the data will still be accessible.

*Enterprise ready*: Solr is well proven and trusted by many leading organizations for their search requirements and for handling extensive loads. It can be deployed in stand-alone, traditional distributed architecture or in cloud mode based on the need of the organization, big or small.

*Full-text search*: As Solr is built on top of Lucene, it provides all the matching capabilities needed including token, phrases, fuzzy, wildcard, spell-check, and autocomplete.

*RESTful XML/JSON over HTTP*: Solr is exposed as a RESTful web service and can be accessed over HTTP. Data can be exchanged in XML, JSON, CSV, and binary format.

*Flexible and extensible*: Solr has a versatile range of features available out of the box. Still, if it doesn’t fit your needs, no worries; Solr is flexible and extensible. You can modify and customize the behavior of components by extending the Java classes and adding the configuration for the created class in the appropriate file.

*Easy configuration*: If you know your schema, you can predefine it; otherwise, you can choose to go schemaless and let Solr define the fields. Also, you can modify your configuration through API calls.

*Comprehensive admin interface*: This is another feature that makes Solr fairly easy to use. Solr provides a feature-powered and user-friendly interface, accessible over HTTP right in your browser. It provides almost all the required insights, enabling you to view configuration files; inspect the text analysis; add/delete shards; manage logs in real time; search, add, delete, and update documents; trigger the data-import process; view threads and system properties; and a lot more.

*Vibrant community*: Solr, a project by the Apache Software Foundation (ASF), has a vibrant and growing community of more than 100 developers and committers. With every release, many features are added and improvements are made. The project has matured to such a level that it has an incredibly versatile set of features available out of the box.

### Major Building Blocks

*Request Handler*: All the requests that you make to Solr are processed by one of the classes implementing SolrRequestHandler. You configure the handler to map to a specific URI endpoint, and the requests made to this endpoint start getting served by it.

*Search Component*: A search component defines the logic to implement the features provided by the search handler. These components should be registered in a SearchHandler, a request handler which serves the user query. For example, features such as query, spell-checking, faceting, and hit-highlighting are implemented as components and are registered to SearchHandler. Multiple components can be registered to a search handler.

*Query Parser*: This translates the user query into instructions that Lucene understands. They are generally registered in the SearchComponent, a component that defines the logic for performing a search.

*Similarity*: This class determines how Lucene weights terms and scores a document. If you want to change the scoring behavior, this is the class to extend.

*Response Writer*: This decides how the response to the user query should be formatted. For each response type such as XML, JSON, or Velocity, there exists a separate response writer.

*Analyzer/tokenizer*: The smallest unit of data that Lucene understands is a token. The implementations of Analyzer, tokenizer or TokenFilter decide how to break the text into tokens.

*Update Request Processor*: While indexing your document, you can invoke a set of UpdateRequestProcessors as part of your chain to perform custom actions upon your data.

***Table 3-1.*** *File Formats and Open Source Tools*

Chapter 3 ■ InformatIon retrIeval

**File Formats Open Source Tools**

PDF Apache POI Apache PDFBox

 Microsoft Word/Excel/PowerPoint Apache POI Apache OpenOffice

HTML jsoup HTMLCleaner

 XML Apache Xerces Java Architecture for XML Binding (JAXB) Dom4j Many other SAX and DOM parsers are available Solr DataImportHandler

OCR Tesseract

Geospatial Geospatial Data Abstraction Library (GDAL)

E-mail  JavaMail API

Web crawling Apache Nutch

Apache Tika supports parsing of content in a wide variety of file formats such as package formats (.tar, .jar, .zip), text document (.doc, .xml, .ppt, .pdf, .html), image (.jpeg, .png, .bmp), audio (.mp3, .wav), and more.

***Table 3-2.*** *Tools and Resources for Processing of Text*

**Preprocess**  **Tools / Resources**

Natural language processing Apache OpenNLP Stanford CoreNLP

Clustering Carrot 2 Apache Mahout Mallet Weka

Classification  Apache Mahout Mallet Weka

Controlled vocabulary WordNet Library of Congress Subject Headings (LCSH) Medical Subject Headings (MeSH)

Text-processing framework Apache UIMA GATE Apache Stanbol Apache cTAKES (Clinical Research)

Knowledge base DBpedia YAGO

### 创建一个新core

solr create -c name

创建一个叫dataimport的croe

bin/solr create -c dataimport

### 导入mysql数据

修改 solrconfig.xml

添加

<lib dir="${solr.install.dir:../../../..}/dist/" regex="solr-dataimporthandler-.\*\.jar" />

<lib dir="${solr.install.dir:../../../..}/dist/" regex="mysql-connector-java-.\*\.jar" />

<requestHandler name="/dataimport"

class="org.apache.solr.handler.dataimport.DataImportHandler">

<lst name="defaults">

<str name="config">data-config.xml</str>

</lst>

</requestHandler>

创建数据表

CREATE TABLE `temp\_user` (

`id` int(10) unsigned NOT NULL AUTO\_INCREMENT,

`playerid` varchar(32) NOT NULL,

`playername` varchar(32) NOT NULL DEFAULT '',

`pdate` timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=5 DEFAULT CHARSET=utf8;

在solrconfig.xml同目录下建立data-config.xml

<dataConfig>

<dataSource type="JdbcDataSource" driver="com.mysql.jdbc.Driver" url="jdbc:mysql://127.0.0.1:3306/cm?useUnicode=true&amp;characterEncoding=utf-8" user="root" password="" />

<document name="users">

<entity name="item" query="select id,playerid,playername from temp\_user" deltaImportQuery="select id,playerid,playername from temp\_user where id='${dih.delta.id}'" deltaQuery="select id from temp\_user where pdate>'${dataimporter.last\_index\_time}'">

<field column="id" name="id" />

<field column="playerid" name="playerid" />

<field column="playername" name="playername" />

</entity>

</document>

</dataConfig>

在managed-schema 中添加

<field name="playerid" type="string" indexed="true" stored="true"/>

<field name="playername" type="string" default="test" indexed="true" stored="true"/>

修改

<uniqueKey>id</uniqueKey>

# MYSQL

## centos7下使用yum安装mysql

CentOS7的yum源中默认好像是没有mysql的。为了解决这个问题，我们要先下载mysql的repo源。

**1. 下载mysql的repo源**

$ wget http://repo.mysql.com/mysql-community-release-el7-5.noarch.rpm

**2. 安装mysql-community-release-el7-5.noarch.rpm包**

$ sudo rpm -ivh mysql-community-release-el7-5.noarch.rpm

安装这个包后，会获得两个mysql的yum repo源：/etc/yum.repos.d/mysql-community.repo，/etc/yum.repos.d/mysql-community-source.repo。

**3. 安装mysql**

$ sudo yum install mysql-server

根据步骤安装就可以了，不过安装完成后，没有密码，需要重置密码。

**4. 重置密码**

重置密码前，首先要登录

$ mysql -u root

登录时有可能报这样的错：ERROR 2002 (HY000): Can‘t connect to local MySQL server through socket ‘/var/lib/mysql/mysql.sock‘ (2)，原因是/var/lib/mysql的访问权限问题。下面的命令把/var/lib/mysql的拥有者改为当前用户：

$ sudo chown -R openscanner:openscanner /var/lib/mysql

然后，重启服务：

$ service mysqld restart

接下来登录重置密码：

$ mysql -u root

mysql > use mysql;

mysql > update user set password=password(‘123456‘) where user=‘root‘;

mysql > exit;

**5. 开放3306端口**

$ sudo vim /etc/sysconfig/iptables

添加以下内容：

-A INPUT -p tcp -m state --state NEW -m tcp --dport 3306 -j ACCEPT

保存后重启防火墙：

$ sudo service iptables restart

这样从其它客户机也可以连接上mysql服务了。

## [MySQL高效分页解决方案集（转）](http://www.cnblogs.com/fslnet/p/3784103.html)

http://www.cnblogs.com/fslnet/p/3784103.html

## Mysql在mac中得位置

/usr/local/mysql-5.7.15-osx10.11-x86\_64/bin/mysql

## 客户端工具

sequel

## 修改密码

http://blog.csdn.net/u013378306/article/details/50466073

## 修改编码

my.cnf

[mysqld]

character\_set\_server=utf8

## 函数

### 返回第一个非空值COALESCE(*value*,...)

 Returns the first non-NULL value in the list, or NULL if there are no non-NULL values.

mysql> **SELECT COALESCE(NULL,1);** -> 1

mysql> **SELECT COALESCE(NULL,NULL,NULL);** -> NULL

### 返回当前日期 CURDATE()

mysql> SELECT CURDATE() -> 2016-09-08

### 返回索引处的字符串ELT(*N*,*str1*,*str2*,*str3*,...)

mysql> **SELECT ELT(1, 'ej', 'Heja', 'hej', 'foo');** -> 'ej'

mysql> **SELECT ELT(4, 'ej', 'Heja', 'hej', 'foo');** -> 'foo'

### 加密 • ENCODE(*str*,*pass\_str*)

ENCODE('cleartext', CONCAT('my\_random\_salt','my\_secret\_password'))

### 返回指定字符串的索引FIELD(*str*,*str1*,*str2*,*str3*,...)

Returns the index (position) of *str* in the *str1*, *str2*, *str3*, ... list. Returns 0 if *str* is not found.

mysql> **SELECT FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo');** -> 2

mysql> **SELECT FIELD('fo', 'Hej', 'ej', 'Heja', 'hej', 'foo');** -> 0

### Group By

Id playerid playername pdate mn

5 1001 hello1 2016-09-08 11:57:47 1

6 1001 hello2 1

7 1002 hello3 2016-09-08 14:09:39 2

8 1003 hello4 2016-09-08 11:58:01 3

9 1003 hello5 2016-09-08 11:58:05 2

10 66 66 2016-09-08 11:58:09 3

#### GROUP\_CONCAT

This function returns a string result with the concatenated non-NULL values from a group. It returns NULL

if there are no non-NULL values. The full syntax is as follows:



GROUP\_CONCAT([DISTINCT] *expr* [,*expr* ...] [ORDER BY {*unsigned\_integer* | *col\_name* | *expr*}

[ASC | DESC] [,*col\_name* ...]] [SEPARATOR *str\_val*])

select playerid,GROUP\_CONCAT(distinct mn order by mn desc) as mn1,pdate,sum(mn) from temp\_user GROUP BY playerid

playerid mn1 pdate sum(mn)

1001 1 2016-09-08 11:57:47 2

1002 2 2016-09-08 14:09:39 2

1003 3,2 2016-09-08 11:58:01 5

66 3 2016-09-08 11:58:09 3

#### ONLY\_FULL\_GROUP\_BY

开启 set sql\_mode=' ONLY\_FULL\_GROUP\_BY'

关闭 set sql\_mode=' '

5.7默认是开启的

必须关闭 ONLY\_FULL\_GROUP\_BY，因为开启的情况下,select 中的列必须出现在group by 子句中

select playerid,GROUP\_CONCAT(distinct mn order by mn desc) as mn1,pdate,sum(mn) from temp\_user GROUP BY playerid,pdate

否则会报错

'cm.temp\_user.pdate' isn't in GROUP BY

# SVN

## SVN使用import导入数据到版本库 (2010-11-09 14:34:52)

|  |  |
| --- | --- |
| [转载](javascript:;)  [▼](javascript:;)  标签： [拷贝](http://search.sina.com.cn/?c=blog&q=%BF%BD%B1%B4&by=tag) [版本](http://search.sina.com.cn/?c=blog&q=%B0%E6%B1%BE&by=tag) [目录](http://search.sina.com.cn/?c=blog&q=%C4%BF%C2%BC&by=tag) [服务器](http://search.sina.com.cn/?c=blog&q=%B7%FE%CE%F1%C6%F7&by=tag) [组文件](http://search.sina.com.cn/?c=blog&q=%D7%E9%CE%C4%BC%FE&by=tag) [it](http://search.sina.com.cn/?c=blog&q=it&by=tag) | 分类： [c/linux/server](http://blog.sina.com.cn/s/articlelist_1267930890_8_1.html) |

正文开始

有两种方法可以将新文件引入Subversion版本库：svn import和svn add，现在讨论svn import

svn import是将未版本化文件导入版本库的最快方法，会根据需要创建中介目录。svn import不需要一个工作拷贝，你的文件会直接提交到版本库，这通常用在你希望将一组文件加入到Subversion版本库时，例如：

1)首先，服务器管理员在SVN服务器上创建项目版本库newrepos

$ svnadmin create /usr/local/svn/newrepos

2)然后，在客户端使用svn命令把本地文件导入到服务器上的项目版本库

$ svn import mytree <file:///usr/local/svn/newrepos/some/project>  -m "Initial import"

Adding         mytree/foo.c

Adding         mytree/bar.c

Adding         mytree/subdir

Adding         mytree/subdir/quux.h

Committed revision 1.

在上一个例子里，将会拷贝目录mytree到版本库的some/project下：

$ svn list file:///usr/local/svn/newrepos/some/project

bar.c

foo.c

subdir/

注意，在导入之后，原来的目录树并没有转化成工作拷贝，为了开始工作，你还是需要运行svn checkout导出一个工作拷贝。

推荐的版本库布局

尽管Subversion的灵活性允许你自由布局版本库，但我们有一套推荐的方式，创建一个trunk目录来保存开发的“主线”，一个branches目录存放分支拷贝，tags目录保存标签拷贝，例如：

$ svn list file:///usr/local/svn/repos

/trunk

/branches

/tags

svn import cloudbase svn://192.168.21.8/hadoop/cloud/cloudbase -m "导入项目"

导出不含.svn文件

svn export svn://host/project ./ --username 用户名

# Git

## 解决Git 每次提交时都要输入用户名和密码的缓存机制的设置

在使用 Git 的时候，经常会遇到需要频繁输入密码的情况，在每次 commit 的时候都要求输入你的用户名和密码，如果提交频繁的话甚是不便。

GNU Linux 用户可以使用 ‘cache’ 认证助手包来缓存认证信息，运行下面的命令来启用凭据缓存，启用后每次输入密码将保存一小时（3600秒）

git config --global  credential.helper  'cache --timeout 3600'

查看 ‘cache’ 认证助手的所有配置:

git helpcredential-cache

## 获取

git clone git://\*\*\*\*.git

## 添加

git add dir file

## 删除

git rm -rf dir

## 提交

git commit dir file -m”comment”

## push 到仓库

git push

## 将本地项目提交到git

echo "# newweb" >> README.md

git init

git add README.md

git commit -m "first commit"

git remote add origin https://github.com/\*.git

git push -u origin master

# 消息队列协议

## RabbitMQ基础概念详细介绍

<http://blog.csdn.net/whycold/article/details/41119807>

## RabbitMq、ActiveMq、ZeroMq、kafka之间的比较,资料汇总

<http://blog.csdn.net/linsongbin1/article/details/47781187>

**java 消息机制 ActiveMQ入门实例**

http://heisetoufa.iteye.com/blog/1908335

## Rabbit

启用WEBUI 管理页面

sudo sbin/rabbitmq-plugins enable rabbitmq\_management

重启服务

sbin/rabbitmq-server

打开网址

http://localhost:15672/

## MQTT

<http://jingyan.baidu.com/article/d45ad148b8efb769542b8066.html?st=2&net_type=&bd_page_type=1&os=0&rst=&word=怎样租服务器>

<http://community.particle.io/t/submission-mqtt-library-and-sample/2111?redirected=true>

### MQTT消息协议、服务器及其客户端

<http://my.oschina.net/u/2306127/blog/370059>

### MQTT——java简单测试（二）

<http://www.tuicool.com/articles/RjyuEbj>

服务端代码：

1 package bsit.mqtt.demo.one\_way;

2

3 import org.eclipse.paho.client.mqttv3.MqttClient;

4 import org.eclipse.paho.client.mqttv3.MqttConnectOptions;

5 import org.eclipse.paho.client.mqttv3.MqttDeliveryToken;

6 import org.eclipse.paho.client.mqttv3.MqttException;

7 import org.eclipse.paho.client.mqttv3.MqttMessage;

8 import org.eclipse.paho.client.mqttv3.MqttPersistenceException;

9 import org.eclipse.paho.client.mqttv3.MqttTopic;

10 import org.eclipse.paho.client.mqttv3.persist.MemoryPersistence;

11 /\*\*

12 \*

13 \* Title:Server

14 \* Description: 服务器向多个客户端推送主题，即不同客户端可向服务器订阅相同主题

15 \* @author chenrl

16 \* 2016年1月6日下午3:29:28

17 \*/

18 public class Server {

19

20 public static final String HOST = "tcp://192.168.1.3:61613";

21 public static final String TOPIC = "toclient/124";

22 public static final String TOPIC125 = "toclient/125";

23 private static final String clientid = "server";

24

25 private MqttClient client;

26 private MqttTopic topic;

27 private MqttTopic topic125;

28 private String userName = "admin";

29 private String passWord = "password";

30

31 private MqttMessage message;

32

33 public Server() throws MqttException {

34 // MemoryPersistence设置clientid的保存形式，默认为以内存保存

35 client = new MqttClient(HOST, clientid, new MemoryPersistence());

36 connect();

37 }

38

39 private void connect() {

40 MqttConnectOptions options = new MqttConnectOptions();

41 options.setCleanSession(false);

42 options.setUserName(userName);

43 options.setPassword(passWord.toCharArray());

44 // 设置超时时间

45 options.setConnectionTimeout(10);

46 // 设置会话心跳时间

47 options.setKeepAliveInterval(20);

48 try {

49 client.setCallback(new PushCallback());

50 client.connect(options);

51 topic = client.getTopic(TOPIC);

52 topic125 = client.getTopic(TOPIC125);

53 } catch (Exception e) {

54 e.printStackTrace();

55 }

56 }

57

58 public void publish(MqttTopic topic , MqttMessage message) throws MqttPersistenceException,

59 MqttException {

60 MqttDeliveryToken token = topic.publish(message);

61 token.waitForCompletion();

62 System.out.println("message is published completely! "

63 + token.isComplete());

64 }

65

66 public static void main(String[] args) throws MqttException {

67 Server server = new Server();

68

69 server.message = new MqttMessage();

70 server.message.setQos(2);

71 server.message.setRetained(true);

72 server.message.setPayload("给客户端124推送的信息".getBytes());

73 server.publish(server.topic , server.message);

74

75 server.message = new MqttMessage();

76 server.message.setQos(2);

77 server.message.setRetained(true);

78 server.message.setPayload("给客户端125推送的信息".getBytes());

79 server.publish(server.topic125 , server.message);

80

81 System.out.println(server.message.isRetained() + "------ratained状态");

82 }

83 }

客户端代码：

1 package bsit.mqtt.demo.one\_way;

2

3 import java.util.concurrent.Executors;

4 import java.util.concurrent.ScheduledExecutorService;

5 import java.util.concurrent.TimeUnit;

6

7 import org.eclipse.paho.client.mqttv3.MqttClient;

8 import org.eclipse.paho.client.mqttv3.MqttConnectOptions;

9 import org.eclipse.paho.client.mqttv3.MqttException;

10 import org.eclipse.paho.client.mqttv3.MqttSecurityException;

11 import org.eclipse.paho.client.mqttv3.MqttTopic;

12 import org.eclipse.paho.client.mqttv3.persist.MemoryPersistence;

13

14 public class Client {

15

16 public static final String HOST = "tcp://192.168.1.3:61613";

17 public static final String TOPIC = "toclient/124";

18 private static final String clientid = "client124";

19 private MqttClient client;

20 private MqttConnectOptions options;

21 private String userName = "admin";

22 private String passWord = "password";

23

24 private ScheduledExecutorService scheduler;

25

26 private void start() {

27 try {

28 // host为主机名，clientid即连接MQTT的客户端ID，一般以唯一标识符表示，MemoryPersistence设置clientid的保存形式，默认为以内存保存

29 client = new MqttClient(HOST, clientid, new MemoryPersistence());

30 // MQTT的连接设置

31 options = new MqttConnectOptions();

32 // 设置是否清空session,这里如果设置为false表示服务器会保留客户端的连接记录，这里设置为true表示每次连接到服务器都以新的身份连接

33 options.setCleanSession(true);

34 // 设置连接的用户名

35 options.setUserName(userName);

36 // 设置连接的密码

37 options.setPassword(passWord.toCharArray());

38 // 设置超时时间 单位为秒

39 options.setConnectionTimeout(10);

40 // 设置会话心跳时间 单位为秒 服务器会每隔1.5\*20秒的时间向客户端发送个消息判断客户端是否在线，但这个方法并没有重连的机制

41 options.setKeepAliveInterval(20);

42 // 设置回调

43 client.setCallback(new PushCallback());

44 MqttTopic topic = client.getTopic(TOPIC);

45 //setWill方法，如果项目中需要知道客户端是否掉线可以调用该方法。设置最终端口的通知消息

46 options.setWill(topic, "close".getBytes(), 2, true);

47

48 client.connect(options);

49 //订阅消息

50 int[] Qos = {1};

51 String[] topic1 = {TOPIC};

52 client.subscribe(topic1, Qos);

53

54 } catch (Exception e) {

55 e.printStackTrace();

56 }

57 }

58

59 public static void main(String[] args) throws MqttException {

60 Client client = new Client();

61 client.start();

62 }

63 }

MQTT订阅回调类：

1 package bsit.mqtt.demo.one\_way;

2

3 import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;

4 import org.eclipse.paho.client.mqttv3.MqttCallback;

5 import org.eclipse.paho.client.mqttv3.MqttMessage;

6

7 /\*\*

8 \* 发布消息的回调类

9 \*

10 \* 必须实现MqttCallback的接口并实现对应的相关接口方法CallBack 类将实现 MqttCallBack。

11 \* 每个客户机标识都需要一个回调实例。在此示例中，构造函数传递客户机标识以另存为实例数据。

12 \* 在回调中，将它用来标识已经启动了该回调的哪个实例。

13 \* 必须在回调类中实现三个方法：

14 \*

15 \* public void messageArrived(MqttTopic topic, MqttMessage message)接收已经预订的发布。

16 \*

17 \* public void connectionLost(Throwable cause)在断开连接时调用。

18 \*

19 \* public void deliveryComplete(MqttDeliveryToken token))

20 \* 接收到已经发布的 QoS 1 或 QoS 2 消息的传递令牌时调用。

21 \* 由 MqttClient.connect 激活此回调。

22 \*

23 \*/

24 public class PushCallback implements MqttCallback {

25

26 public void connectionLost(Throwable cause) {

27 // 连接丢失后，一般在这里面进行重连

28 System.out.println("连接断开，可以做重连");

29 }

30

31 public void deliveryComplete(IMqttDeliveryToken token) {

32 System.out.println("deliveryComplete---------" + token.isComplete());

33 }

34

35 public void messageArrived(String topic, MqttMessage message) throws Exception {

36 // subscribe后得到的消息会执行到这里面

37 System.out.println("接收消息主题 : " + topic);

38 System.out.println("接收消息Qos : " + message.getQos());

39 System.out.println("接收消息内容 : " + new String(message.getPayload()));

40 }

41 }

## Rabbitmq

创建用户

rabbitmqctl add\_user cmz pwd

设置权限

rabbitmqctl set\_permissions -p "/" cmz ".\*" ".\*" ".\*"

## 4.1 问题1 访问密码错误

=INFO REPORT==== 25-Oct-2016::22:35:26 ===   
accepting AMQP connection <0.7926.0> (192.168.2.99:62969 -> 192.168.2.59:5672)   
ndshake\_error,starting,0,   
{amqp\_error,access\_refused,   
“PLAIN login refused: user ‘sun’ - invalid credentials”,   
‘connection.start\_ok’}}

由于我在安装完rabbitMq后并创建了用户，并且连接时候使用了这个用户，但密码错误，而且我也忘记了当时创建账户的密码，导致产生这个问题。   
随后通过命令修改密码**rabbitmqctl change\_password 用户 密码**   
并在下面的配置中重新设置password后，问题1解决。

<rabbit:connection-factory id="connectionFactory" username="sun" password="123456" host="192.168.2.133" port="5672" />

* 1
* 1

## 4.2 问题2 没有权限访问vhost

=ERROR REPORT==== 25-Oct-2016::22:34:17 ===   
closing AMQP connection <0.7862.0> (192.168.2.99:62932 -> 192.168.2.59:5672):   
{handshake\_error,opening,0,   
{amqp\_error,access\_refused,   
“access to vhost ‘/’ refused for user ‘sun’”,   
‘connection.open’}}

这个问题意思是说用户sun没有vhost /的访问权限。 vhost的概念可以在**RabbitMQ in Action 2.4 Multiple tenants: virtual hosts and separation**节看到。

因此通过rabbitmqctl赋予用户权限即可   
**rabbitmqctl set\_permissions -p / sun “.” “.” “.\*”**

## 4.3 问题3 vhost下不存在队列

ERROR REPORT==== 25-Oct-2016::22:53:18 ===   
connection <0.8847.0>, channel 1 - soft error:   
{amqp\_error,not\_found,”no queue ‘myQueue’ in vhost ‘/’”,’queue.declare’}

字面意思看是说没有在vhost下找到队列。看过Rabbitmq官网6步教程的朋友都知道，queueDeclare来声明队列，不存在则创建，否则不操作。

而我最初的配置文件中是没有下面这行配置，而RabbitAdmin则是负责   
声明queues，exchanges，与bindings。

<rabbit:admin connection-factory="connectionFactory"/>

* 1
* 1

因此在最初的配置添加完这个配置后，问题解决。

## 4.4 问题4 将生产者与消费者分离

上述配置是在一个项目中，消费者和生产者在一个项目。若消费者和生产者部署在不同机器如何配置？

通过学习RabbitMq 6步教程可以看到，消费者需要与RabbitMq建立连接，   
并消费某消息队列上的消息。因此消费端同样配置好连接工厂，Template，队列，RabbitAdmin，及listener-**[Container](http://lib.csdn.net/base/docker" \o "Docker知识库" \t "_blank)**。然后通过Spring加载配置文件，创建SpringContext后，即可消费某队列上的消息。

## 4.5 官网Quick Start中消息监听器适配器的理解与使用

Quick Start中的一段代码如下，消费者通过handleMessage来消费消息。

Object listener = new Object() { public void handleMessage(String foo) { System.out.println(foo); } }; MessageListenerAdapter adapter = new MessageListenerAdapter(listener); container.setMessageListener(adapter);

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 1
* 2
* 3
* 4
* 5
* 6
* 7

在接收消息时消费端一般都会继承org.springframework.amqp.core.MessageListener这个接口，当收到消息后void onMessage(Message message);方法会被调用。但有时希望通过自己指定一个方法来处理接收到的消息，这时就会用到org.springframework.amqp.rabbit.listener.adapter.MessageListenerAdapter类。该类通过适配器模式，最终使用户可以指定消息处理的方法，其源码实现如下。

public void onMessage(Message message, Channel channel) throws Exception { Object delegate = getDelegate(); //........省略 Object convertedMessage = extractMessage(message); String methodName = getListenerMethodName(message, convertedMessage); if (methodName == null) { //抛出异常...... } Object[] listenerArguments = buildListenerArguments(convertedMessage); Object result = invokeListenerMethod(methodName, listenerArguments, message); if (result != null) { handleResult(result, message, channel); } else { // 日志 } }

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19

上面的代码截取了调用用户委派对象的主要代码。可以看到主要是获取用户委派的对象，解析参数，方法，最终调用用户指定的对象完成消息的处理。

而下面这段配置ref的作用就是用来指定要适配的目标。并且最终是通过MessageListenerAdapter来处理，具体可以在该类的onMessage方法中设置断点，然后启动项目观察。

<rabbit:listener-container connection-factory="connectionFactory"> <rabbit:listener ref="foo" method="listen" queue-names="myQueue" /> </rabbit:listener-container>

* 1
* 2
* 3
* 1
* 2
* 3

而Quickstart中之所以用handleMessage来处理消息是因为   
使用者并未实现MessageListener接口，且为指定默认方法名，所以在解析监听方法名时，会返回默认的消息处理方法名，handleMessage。

public static final String ORIGINAL\_DEFAULT\_LISTENER\_METHOD = "handleMessage"; private String defaultListenerMethod = ORIGINAL\_DEFAULT\_LISTENER\_METHOD;

# Eclipse

## Eclipse将引用了第三方jar包的Java项目打包成jar文件的两种方法

http://www.cnblogs.com/lanxuezaipiao/p/3291641.html

# Ant

## 使用ant获取文件列表

<http://blog.sina.com.cn/s/blog_605f5b4f0100xnij.html>

# Maven

<http://mvnrepository.com>

http://maven.apache.org

# Netty

## 《Netty 权威指南》—— AIO创建的TimeClient源码分析

<http://ifeve.com/netty-4-2/>

http://ifeve.com/netty5-user-guide/

## Netty Best Practices a.k.a Faster == Better

http://normanmaurer.me/presentations/2014-facebook-eng-netty/slides.html#1.0

# WEB技术

## 实时更新网页

基于[Express](http://expressjs.com/)和[Socket.io](http://socket.io/)两个框架实现了向浏览器推送内容更新

## JS图表

<http://www.open-open.com/lib/view/open1406378625726.html>

http://www.ichartjs.com

# Redis

## 安装EPEL

### CentOS6

的请编辑/etc/yum.repos.d/epel.repo输入以下内容

|  |
| --- |
| [epel]  name=Extra Packages **for** Enterprise Linux 6 - $basearch  #baseurl=http://download.fedoraproject.org/pub/epel/6/$basearch  mirrorlist=https://mirrors.fedoraproject.org/metalink?repo=epel-6&arch=$basearch  failovermethod=priority  enabled=1  gpgcheck=1  gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-6    [epel-debuginfo]  name=Extra Packages **for** Enterprise Linux 6 - $basearch - Debug  #baseurl=http://download.fedoraproject.org/pub/epel/6/$basearch/debug  mirrorlist=https://mirrors.fedoraproject.org/metalink?repo=epel-debug-6&arch=$basearch  failovermethod=priority  enabled=0  gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-6  gpgcheck=1    [epel-source]  name=Extra Packages **for** Enterprise Linux 6 - $basearch - Source  #baseurl=http://download.fedoraproject.org/pub/epel/6/SRPMS  mirrorlist=https://mirrors.fedoraproject.org/metalink?repo=epel-source-6&arch=$basearch  failovermethod=priority  enabled=0  gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-6  gpgcheck=1 |

然后执行yum clean all

yum makecache即可

### centos7的epel源

编辑/etc/yum.repos.d/epel.repo

|  |
| --- |
| [epel]  name=Extra Packages **for** Enterprise Linux 7 - $basearch  #baseurl=http://download.fedoraproject.org/pub/epel/7/$basearch  mirrorlist=https://mirrors.fedoraproject.org/metalink?repo=epel-7&arch=$basearch  failovermethod=priority  enabled=1  gpgcheck=0  gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-7  [epel-debuginfo]  name=Extra Packages **for** Enterprise Linux 7 - $basearch - Debug  #baseurl=http://download.fedoraproject.org/pub/epel/7/$basearch/debug  mirrorlist=https://mirrors.fedoraproject.org/metalink?repo=epel-debug-7&arch=$basearch  failovermethod=priority  enabled=0  gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-7  gpgcheck=1  [epel-source]  name=Extra Packages **for** Enterprise Linux 7 - $basearch - Source  #baseurl=http://download.fedoraproject.org/pub/epel/7/SRPMS  mirrorlist=https://mirrors.fedoraproject.org/metalink?repo=epel-source-7&arch=$basearch  failovermethod=priority  enabled=0  gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-7  gpgcheck=1 |

**方法一：使用命令安装（前提是已经安装了EPEL）。**

## 安装redis:

### yum -y install redis

启动/停止/重启 Redis

启动服务：

1

systemctl start redis.service

停止服务：

systemctl stop redis.service

重启服务：

systemctl restart redis.service

检查状态：

[root@idoseek ~]# systemctl status redis.service

redis.service - Redis persistent key-value database

   Loaded: loaded (/usr/lib/systemd/system/redis.service; enabled)

   Active: active (running) since 二 2014-10-21 21:37:22 EDT; 5h 26min ago

 Main PID: 30413 (redis-server)

   CGroup: /system.slice/redis.service

           └─30413 /usr/bin/redis-server 127.0.0.1:6379

10月 21 21:37:22 idoseek.com systemd[1]: Started Redis persistent key-value database.

随系统启动服务：

[root@idoseek ~]# systemctl enable redis.service

ln -s '/usr/lib/systemd/system/redis.service' '/etc/systemd/system/multi-user.target.wants/redis.service'

关闭随机启动：

[root@idoseek ~]# systemctl disable redis.service

rm '/etc/systemd/system/multi-user.target.wants/redis.service'

### 方法二：编译安装

下载安装编译:

# wget http://download.redis.io/releases/redis-2.8.17.tar.gz

# tar xzf redis-2.8.17.tar.gz

# cd redis-2.8.17

# make

# make install

设置配置文件路径:

# mkdir -p /etc/redis && cp redis.conf /etc/redis

修改配置文件：

# vim /etc/redis/redis.conf

修改为： daemonize yes

启动Redis:

# /usr/local/bin/redis-server /etc/redis/redis.conf

#关闭服务

redis-cli shutdown

或者在cli中执行shutdown

redis 127.0.0.1:6379> shutdown

清除缓存

redis-cli flushall

更多文档请参考软件包内的“README”文件。

查看状态 ：

# ss -nlp|grep redis

或者

# [ps](http://www.111cn.net/fw/photo.html) -ef | grep redis

### 下面介绍为PHP添加redis插件。

从官网下载最新的拓展，地址：http://pecl.php.net/package/redis或者https://github.com/phpredis/phpredis

#wget http://pecl.php.net/get/redis-2.2.5.tgz

#phpize

#./configure --prefix=/opt/redis --enable-redis --with-php-config=/opt/php/bin/php-config

#make && make install

把拓展添加至php.ini，重启php-fpm：

service php-fpm restart

# CentOS7安装配置redis-3.0.0

**一.安装必要包**

yum install gcc

**二.linux下安装**

#下载

wget http://download.redis.io/releases/redis-3.0.0.tar.gz

tar zxvf redis-3.0.0.tar.gz

cd redis-3.0.0

#如果不加参数,linux下会报错

make MALLOC=libc

**安装好之后,启动文件**

#启动redis

src/redis-server &

#关闭redis

src/redis-cli shutdown

**测试redis**

$ src/redis-cli

127.0.0.1:6379> set foo bar

OK

127.0.0.1:6379> get foo

"bar"

$

测试成功

**3.redis cluster集群搭建**

**建立本机测试环境**

建立运行目录

#建立redis运行目录

mkdir -p redis-server/7000/

#复制默认的配置文档

cp redis-3.0.0/redis.conf redis-server/redis.default.conf

#把编译好的server复制到运行目录

cp redis-3.0.0/src/redis-server redis-server/7000/

建立独立配置文件

#在7000目录下建立redis的配置文档

vim redis-server/7000/redis.conf

文件内容

[](javascript:void(0);)

#redis-server/7000/redis.conf

include /root/redis-server/redis.default.conf

pidfile /var/run/redis-7000.pid

port 7000

cluster-enabled yes

cluster-config-file redis-node-7000.conf

cluster-node-timeout 5000

appendonly yes

[](javascript:void(0);)

复制运行目录(模拟集群环境)

#复制目录

cp -R 7000/ 7001/

cp -R 7000/ 7002/

cp -R 7000/ 7003/

cp -R 7000/ 7004/

cp -R 7000/ 7005/

修改相应配置文件的端口和文件名

建立启动脚本 redis-server/redis-start.sh

[](javascript:void(0);)

#!/bin/sh

/root/redis-server/7000/redis-server /root/redis-server/7000/redis.conf &

/root/redis-server/7001/redis-server /root/redis-server/7001/redis.conf &

/root/redis-server/7002/redis-server /root/redis-server/7002/redis.conf &

/root/redis-server/7003/redis-server /root/redis-server/7003/redis.conf &

/root/redis-server/7004/redis-server /root/redis-server/7004/redis.conf &

/root/redis-server/7005/redis-server /root/redis-server/7005/redis.conf &

[](javascript:void(0);)

**配置集群**

安装ruby

yum install ruby-devel.x86\_64

安装redis gem

# gem install redis

Fetching: redis-3.2.1.gem (100%)

Successfully installed redis-3.2.1

Parsing documentation for redis-3.2.1

Installing ri documentation for redis-3.2.1

1 gem installed

使用脚本建立集群机制

*在create的时候,加上参数--replicas 1 表示为每个master分配一个salve,如例子,则是3个master 3个salve*

[](javascript:void(0);)

# ./redis-trib.rb create 127.0.0.1:7000 127.0.0.1:7001 127.0.0.1:7002 127.0.0.1:7003 127.0.0.1:7004 127.0.0.1:7005

>>> Creating cluster

Connecting to node 127.0.0.1:7000: OK

Connecting to node 127.0.0.1:7001: OK

Connecting to node 127.0.0.1:7002: OK

Connecting to node 127.0.0.1:7003: OK

Connecting to node 127.0.0.1:7004: OK

Connecting to node 127.0.0.1:7005: OK

>>> Performing hash slots allocation on 6 nodes...

Using 6 masters:

127.0.0.1:7000

127.0.0.1:7001

127.0.0.1:7002

127.0.0.1:7003

127.0.0.1:7004

127.0.0.1:7005

M: f3dd250e4bc145c8b9f864e82f65e00d1ba627be 127.0.0.1:7000

slots:0-2730 (2731 slots) master

M: 1ba602ade59e0770a15128b193f2ac29c251ab5e 127.0.0.1:7001

slots:2731-5460 (2730 slots) master

M: 4f840a70520563c8ef0d7d1cc9d5eaff6a1547a2 127.0.0.1:7002

slots:5461-8191 (2731 slots) master

M: 702adc7ae9caf1f6702987604548c6fc1d22e813 127.0.0.1:7003

slots:8192-10922 (2731 slots) master

M: 4f87a11d2ea6ebe9caf02c9dbd827a3dba8a53cf 127.0.0.1:7004

slots:10923-13652 (2730 slots) master

M: 216bbb7da50bd130da16a327c76dc6d285f731b3 127.0.0.1:7005

slots:13653-16383 (2731 slots) master

Can I set the above configuration? (type 'yes' to accept): yes

>>> Nodes configuration updated

>>> Assign a different config epoch to each node

>>> Sending CLUSTER MEET messages to join the cluster

Waiting for the cluster to join...

>>> Performing Cluster Check (using node 127.0.0.1:7000)

M: f3dd250e4bc145c8b9f864e82f65e00d1ba627be 127.0.0.1:7000

slots:0-2730 (2731 slots) master

M: 1ba602ade59e0770a15128b193f2ac29c251ab5e 127.0.0.1:7001

slots:2731-5460 (2730 slots) master

M: 4f840a70520563c8ef0d7d1cc9d5eaff6a1547a2 127.0.0.1:7002

slots:5461-8191 (2731 slots) master

M: 702adc7ae9caf1f6702987604548c6fc1d22e813 127.0.0.1:7003

slots:8192-10922 (2731 slots) master

M: 4f87a11d2ea6ebe9caf02c9dbd827a3dba8a53cf 127.0.0.1:7004

slots:10923-13652 (2730 slots) master

M: 216bbb7da50bd130da16a327c76dc6d285f731b3 127.0.0.1:7005

slots:13653-16383 (2731 slots) master

[OK] All nodes agree about slots configuration.

>>> Check for open slots...

>>> Check slots coverage...

[OK] All 16384 slots covered.

[](javascript:void(0);)

 如果需要全部重新自动配置,则删除所有的配置好的cluster-config-file,重新启动所有的redis-server,然后重新执行配置命令即可

# Centos开机自启动redis

一：原理介绍

先看下当前系统已设置的开机自启动程序有哪些：

[root@localhost init.d]# chkconfig --list

Note: This output shows SysV services only and does not include native

      systemd services. SysV configuration data might be overridden by native

      systemd configuration.

      If you want to list systemd services use 'systemctl list-unit-files'.

      To see services enabled on particular target use

      'systemctl list-dependencies [target]'.

netconsole     0:off1:off

2:off3:off

4:off5:off

6:off

network         0:off1:off

2:on3:on

4:on5:on

6:off

[root@localhost init.d]# ls /etc/init.d/ -lrt

total 40

-rwxr-xr-x. 1 root root  6470 Jan 15  2015 network

-rwxr-xr-x. 1 root root  2989 Jan 15  2015 netconsole

-rw-r--r--. 1 root root 13430 Jan 15  2015 functions

-rw-r--r--. 1 root root  1160 Mar  6  2015 README

-rw-r--r--. 1 root root    37 Nov 30 05:43 dump.rdb

从上面可以看出系统已设置的开机自启动程序有network和netconsole，新增开机自启动程序，基本原理为：

系统开机启动时会去加载/etc/init.d/下面的脚本，通常而言每个脚本文件会自定义实现程序的启动；若想将新的程序开机自启动，只需在该目录下添加一个自定义启动程序的脚本，然后设置相应规则即可(如我在/etc/init.d/下新建一个redis的脚本，开机启动时会去加载执行该脚本；当然这只是开机设置自启动的一种方法)

二：实践

废话少说，直接来步骤：

1、设置redis.conf中daemonize为yes,确保守护进程开启。

2、编写开机自启动脚本

vi /etc/init.d/redis

脚本内容如下：

# chkconfig: 2345 10 90

# description: Start and Stop redis

PATH=/usr/local/bin:/sbin:/usr/bin:/bin

REDISPORT=6379

EXEC=/usr/local/redis-3.0.0-beta5/src/redis-server

REDIS\_CLI=/usr/local/redis-3.0.0-beta5/src/redis-cli

PIDFILE=/var/run/redis.pid

CONF="/usr/local/redis-3.0.0-beta5/redis.conf"

AUTH="1234"

case "$1" in

start)

if [ -f $PIDFILE ]

then

echo "$PIDFILE exists, process is already running or crashed."

else

echo "Starting Redis server..."

$EXEC $CONF

fi

if [ "$?"="0" ]

then

echo "Redis is running..."

fi

;;

stop)

if [ ! -f $PIDFILE ]

then

echo "$PIDFILE exists, process is not running."

else

PID=$(cat $PIDFILE)

echo "Stopping..."

$REDIS\_CLI -p $REDISPORT SHUTDOWN

sleep 2

while [ -x $PIDFILE ]

do

echo "Waiting for Redis to shutdown..."

sleep 1

done

echo "Redis stopped"

fi

;;

restart|force-reload)

${0} stop

${0} start

;;

\*)

echo "Usage: /etc/init.d/redis {start|stop|restart|force-reload}" >&2

exit 1

esac

3、写完后保存退出VI

4、设置权限

chmod 755 redis

5、启动测试

/etc/init.d/redis start

启动成功会提示如下信息：

Starting Redis server...

2358:M 30 Nov 05:34:16.991 \* Increased maximum number of open files to 10032 (it was originally set to 1024).

                \_.\_

           \_.-``\_\_ ''-.\_

      \_.-``    `.  `\_.  ''-.\_           Redis 2.9.54 (00000000/0) 64 bit

  .-`` .-```.  ```\/    \_.,\_ ''-.\_

 (    '      ,       .-`  | `,    )     Running in stand alone mode

 |`-.\_`-...-` \_\_...-.``-.\_|'` \_.-'|     Port: 6379

 |    `-.\_   `.\_    /     \_.-'    |     PID: 2358

  `-.\_    `-.\_  `-./  \_.-'    \_.-'

 |`-.\_`-.\_    `-.\_\_.-'    \_.-'\_.-'|

 |    `-.\_`-.\_        \_.-'\_.-'    |           http://redis.io

  `-.\_    `-.\_`-.\_\_.-'\_.-'    \_.-'

 |`-.\_`-.\_    `-.\_\_.-'    \_.-'\_.-'|

 |    `-.\_`-.\_        \_.-'\_.-'    |

  `-.\_    `-.\_`-.\_\_.-'\_.-'    \_.-'

      `-.\_    `-.\_\_.-'    \_.-'

          `-.\_        \_.-'

              `-.\_\_.-'

2358:M 30 Nov 05:34:16.993 # Server started, Redis version 2.9.54

2358:M 30 Nov 05:34:16.993 # WARNING overcommit\_memory is set to 0! Background save may fail under low memory condition. To fix this issue add 'vm.overcommit\_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit\_memory=1' for this to take effect.

2358:M 30 Nov 05:34:16.993 \* The server is now ready to accept connections on port 6379

使用redis-cli测试：

[best\_husband@localhost src]$ /usr/local/redis-3.0.0-beta5/src/redis-cli

127.0.0.1:6379> set name usbdrivers

OK

127.0.0.1:6379> get name

"usbdrivers"

127.0.0.1:6379> quit

[best\_husband@localhost src]$

6、设置开机自启动

chkconfig redis on

7、关机重启测试

reboot

然后在用redis-cli测试即可。

# Python

## python 安装easy\_install和pip

easy\_install和pip都是用来下载安装Python一个公共资源库PyPI的相关资源包的

首先安装easy\_install

下载地址:https://pypi.python.org/pypi/ez\_setup

解压,安装.

python ez\_setup.py

安装好easy\_install 之后 再安装pip

下载地址:https://pypi.python.org/pypi/pip

解压,安装.

python setup.py install

## 开发工具

<http://www.jetbrains.com/pycharm/>

<https://pypi.python.org/pypi/python-crontab/>

## 比较详细Python正则表达式操作指南(re使用)

<http://www.jb51.net/article/15707.htm>

## 在python中实现动态导入模块importlib.import\_module

http://www.ddvip.com/tech/1000147013.html

## Web 开发框架

<https://www.djangoproject.com>

## python 查询impala

https://github.com/cloudera/impyla

## [crontab调度Python脚本](http://blog.itpub.net/16582684/viewspace-756872/)

<http://blog.itpub.net/16582684/viewspace-756872/>

1.  在python脚本中增加#!/usr/bin/env python来声明执行程序在系统环境变量中的名字

2.  在crontab中增加. $HOME/.bash\_profile来调用bash\_profile来声明环境变量，例： 19 16 \* \* \* . $HOME/.bash\_profile;/home/oradev/wget\_test.py

注:  文件wget\_test.py为可执行状态 -- chmod +x wget\_test.py

        该文件的目录必须在环境变量PATH 内。

还遇到一问题就是脚本不执行也不报错，最终发现python有两个版本，一个版本在/usr/bin/python是2.4.3，将其删除并指定软链接 ln -s /usr/local/bin/python2.7 /usr/bin/python

在crontab中增加source /etc/profile来调用 /etc/profile里声明的环境变量

5 1 \* \* \* source /etc/profile; /opt/cndw/shell/getmerge.py >> /opt/cndw/shell/result.log 2>&1 &

## python在linux(anaconda)的图形界面（snack）

<http://blog.csdn.net/taiyang1987912/article/details/49273601>

## 图形

http://matplotlib.org/faq/howto\_faq.html#generate-images-without-having-a-window-appear

### [RHEL7 单独安装图形 X11](http://blog.itpub.net/27771627/viewspace-1223155/)

<http://blog.itpub.net/27771627/viewspace-1223155/>

### 如何在linux下通过ssh运行X图形软件

服务器端：编辑/etc/ssh/sshd\_config中的以下内容

启用AllowTcpForwarding

启用X11Forwarding

将X11DisplayOffset设定为10.

启用X11UseLocalhost

客户机端：编辑/etc/ssh/ssh\_config中的以下内容

启用ForwardX11

连接时ssh -X或者ssh -Y就可以了

# Tomcat

## 乱码解决

在server.xml 中找到Connector

添加 uRIEncoding=*"utf-8"* useBodyEncodingForURI=*"true"*

例如:

<Connector connectionTimeout=*"20000"* port=*"8080"* protocol=*"HTTP/1.1"* redirectPort=*"8443"* uRIEncoding=*"utf-8"* useBodyEncodingForURI=*"true"*/>

# Resin

<http://caucho.com/resin-4.0/>

## OneAPM 为您提供端到端的 Java 应用性能解决方案

<http://www.oneapm.com/ai/java/resin.html>

Eclipse 配置 resin

服务器启动默认会加上项目名作为二级目录，用以下方法去掉二级目录

<!-- the default host, matching any host name -->

<host id=*""* root-directory=*"."*>

<!--

- webapps can be overridden/extended in the resin.xml

-->

<web-app id=*"/"* root-directory=*"webapps/datacenter"*>

<character-encoding>UTF-8</character-encoding>

</web-app>

</host>

## resin的log、安全性ip限制、内存jvm、keepalive、代码的更新检测等

<http://sling2007.blog.163.com/blog/static/8473271320127821117148/>

# Nginx

## centos 7 上配置SELinux允许nginx指定/home/www作为网站根目录

http://blog.csdn.net/qidizi/article/details/41291397

# Disruptor

<http://ifeve.com/disruptor-getting-started/>

# 高可用

## haproxy+keepalived实现高可用负载均衡

<http://www.cnblogs.com/dkblog/archive/2011/07/06/2098949.html>

# 缓存

Ignite

# 前端技术

## bootstrap

<http://www.runoob.com/bootstrap/bootstrap-forms.html>

<http://www.bootcss.com>

<http://chartjs.cn>

http://wrapbootstrap.com/preview/WB0901P31

## javascript

npm 包管理

<https://www.npmjs.com/search?q=io>

## 图表

http://www.jchartfx.com/

http://support.softwarefx.com/jChartFX/article/2601325#!2501015

jChartFX

<http://www.hcharts.cn/demo/highcharts>

## CSS

### 文本超出部分用…代替

width:50px;overflow:hidden;white-space:nowrap;text-overflow:ellipsis

## jquery

### class选择器

#### [JQuery 如何选择带有多个class的元素](http://blog.csdn.net/lifeng_beijing/article/details/42173895)

|  |  |
| --- | --- |
| 1 | <div class="modal fade in"></div> |

A:

1. 依次过滤

|  |  |
| --- | --- |
| 1 | $(".modal.fade").filter("in") |

2. 属性选择

|  |  |
| --- | --- |
| 1 | $("[class='modal fade in']") //此处顺序必须一致才行 |

3. 直接选择

|  |  |
| --- | --- |
| 1 | $(".modal.fade.in") |

# 序列化框架

## MessagePack, Protocol Buffers和Thrift序列化框架原理和比较说明

<http://jimmee.iteye.com/blog/2042420>

# PHP

## 框 架

Yii doit initphp

<http://www.initphp.com>

# Cisco

Softpedia

GNS3 模拟器

<http://mac.softpedia.com/get/Network-Admin/GNS3.shtml>

<http://www.cisco.com/c/en/us/support/ios-nx-os-software/ios-software-release-15-4-3-s/model.html#~tab-downloads>

<http://www.mamicode.com/info-detail-605879.html>

# VIM

## Vim 常用快捷键

http://www.cnblogs.com/sld666666/archive/2010/04/05/1704462.html

1.:sy on 语法高亮

2.:set go =  （set gui option = nothing）

3.:colo evening

4. tabnew :d\test.txt (Open a new file)

5.set go =e6. :w (save)

光标移动指令

 1. h 左

2. l 右

3. k 上

4. j 下

5. Ctrl+f 向前翻页

6.Ctrl+b 向后翻页

7. 0  行首

8. 移至行首第一个非空白字

9. $ 行尾

10.G 文档尾部

11.gg 文档首部

12.w 移至一个word的首部

13.e 移至一个word的字尾

15.b 移至前一个自首

16.L 光标移动至屏幕中间的第一个非空字元

17.：n 移至第n行行首

18.) 移至下一个句子首部

19.( 移至上一个句子首部

20.} 移至下一个段落首部

21.} 移至上一个段落首部

进入插入命令

1. i 在光标的字之前

2. a 在光标的字之后

3. o 光标的下一行

4. I 在光标的行首

5. A 在光标的行尾

6. O 在光标的上一行

7. J 将下一行整行接至本行

 删除指令

1. x 删除光标所在的字元

2. X 删除光标之前的字元

3. dd 删除整行

4. dw 删除一个word

5. dG 删除至文档尾

6. dgg 删除至档首

7. D 删除至行尾

8.d0 删除至行首

 取代及还原

1. r 取代光标所在的字元

2. R 进入取代模式

3. cc 取代整行

4. cw 取代一个word

5. ~ 光标所在的字元大小写互换

6. C 取代至行尾

7. c0 取代至行首

8. s 替换一个字元为字符串

9. u undo

10. ctrl+r redo

文字排布

 1.:ce(nter) 文字居中

2,:right  文字靠右

3.le（ft）文字靠左

复制指令（yank）

1. yy 复制整行

2. 2yy 复制

2行3. y^ 复制至行首

4. y$ 复制至行尾

5. yw 复制一个word

6. y2w 复制2个word

7. yG 复制至档尾

8. y1G 复制至档首

9. p 粘贴至游标后

10.P 粘贴至光标前

搜寻

 1. / +要找的单词+enter,n向下，N向上

替换

[range]s/pattern/string/[c,e,g,i]

　　range 表示的是范围，1，7第一行到第七行，$整篇文章 %目前编辑的，#前一次编辑的pattern 要被替换的字串string 取代之的字串c 每次替换前询问e 不提示errorg 不询问，整行替换i 不区分大小写

书签功能

  在文章某处marks ，然后跑到其他地方去了，这是呼叫mark 又回到了原处1. mx x为26个小写字母，表示光标这被mark了2.`x 回到x标志的位置3. 'x 回到x标志的行首

多档案编辑

1. :n 编辑下一个档案

2. :2n 编辑下二个档案

3. :N 编辑前一个档案

4. :e 档案名，tab自动补全

5. :bn 编辑次一个

6. :bp 编辑前一个

7. :bl 编辑最后一个

离开

 1. :q

2. :q! 强制离开

3. :wq 存档离开

4. :x 存档离开

5. :w 名字 另存为

6. :qa 离开整个vim a= all

视窗操作

 1. ctrl+w n（new） 开一个新窗口 w=window

2. ctrl+w s (spli) 开一个新窗口，原档案分离为两个

3. ctrl+w f 开一个新窗口，并编辑光标所在的word为新档案名字

4. ctlr+w q 结束分离出来的窗口

5. ctrl+w o(only)使光标所在的窗口显示其他窗口隐藏

6. ctrl+w j 移至下窗口

7. ctrl+w k 移至上窗口

8. :sp 另外开一个窗口来编辑档案

# 云盘资料下载

<http://www.panduoduo.net/r/13083062>

搜索关键字 例如 oreilly spark

# 杂七杂八

## [多个进程间共享动态链接库的原理](http://blog.chinaunix.net/uid-26983585-id-3364514.html)

http://blog.chinaunix.net/uid-26983585-id-3364514.html

sphinx 全文搜索

## 数学

### 线性代数库

**Jblas**

## 机器学习

<https://en.wikipedia.org/wiki/Naive_Bayes_classifier>

### 线性分类 Linear Models

loss function 损失函数

1、logistic loss

2、hinge loss 连接损失

link function 连接函数

<http://spark.apache.org/docs/latest/mllib-linear-methods.html#binary-classification>

<https://en.wikipedia.org/wiki/Generalized_linear_model>

<http://scikit-learn.org/stable/auto_examples/linear_model/plot_sgd_loss_functions.html>

支持向量机SVM(Suppoted Vector Machine)

<https://en.wikipedia.org/wiki/Support_vector_machine>

<http://www.support-vector-machines.org>

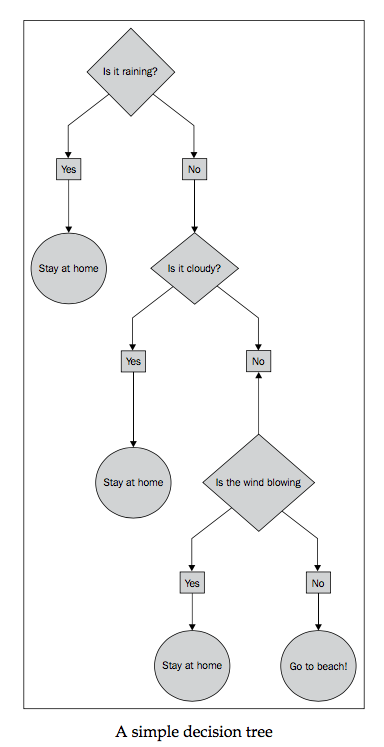
[Logistic regression (逻辑回归)](http://www.baidu.com/link?url=KRvVhtKyDXaN45Vxrk0lvWyc7WEqDibXhaD5jhM94K1LpEEeBkDA5eIrPdVDbvwxPphaACB81kocx1Iq_niosa)

### 贝叶斯模型The naïve Bayes model

<http://spark.apache.org/docs/latest/mllib-naive-bayes.html>

<https://en.wikipedia.org/wiki/Naive_Bayes_classifier>

### 决策树 Decision trees



<http://spark.apache.org/docs/latest/mllib-decision-tree.html>

<http://spark.apache.org/docs/latest/api/scala/#org.apache.spark.mllib.regression.LabeledPoint>

分类数据源

<https://www.kaggle.com/c/stumbleupon/data>

**https://en.wikipedia.org/wiki/Precision\_and\_recall**

<https://en.wikipedia.org/wiki/Information_retrieval#Average_precision>

# 制作MAC 启动盘

<http://nb.zol.com.cn/525/5253688_all.html>

sudo Install\ OS\ X\ El\ Capitan.app/Contents/Resources/createinstallmedia --volume /Volumes/OS\ X\ EI\ Caption --applicationpath Install\ OS\ X\ El\ Capitan.app –nointeraction

# 正则表达式

<http://www.runoob.com/regexp/regexp-syntax.html>

# Netty

https://github.com/normanmaurer/netty-in-action

## 关于分包那些事

Note that the message sent by the server may be received in chunks. That is, if the server sends 5 bytes, there’s no guarantee that all 5 bytes will be received at once. Even for such a small amount of data, the channelRead0() method could be called twice, first with a ByteBuf (Netty’s byte container) holding 3 bytes, and second with a ByteBuf holding 2 bytes. As a stream-oriented protocol, TCP guarantees that the bytes will be received in the order in which they were sent by the server.

## SimpleChannelInboundHandler vs. ChannelInboundHandler

You may be wondering why we used SimpleChannelInboundHandler in the client instead of the ChannelInboundHandlerAdapter used in the EchoServerHandler. This has to do with the interaction of two factors: how the business logic processes messages and how Netty manages resources.

In the client, when channelRead0() completes, you have the incoming message and you’re done with it. When the method returns, SimpleChannelInboundHandler takes care of releasing the memory reference to the ByteBuf that holds the message.

In EchoServerHandler you still have to echo the incoming message to the sender, and the write() operation, which is asynchronous, may not complete until after channelRead() returns (shown in listing 2.1). For this reason EchoServerHandler extends ChannelInboundHandlerAdapter, which doesn’t release the message at this point.

The message is released in channelReadComplete() in the EchoServerHandler when writeAndFlush() is called (listing 2.1).

## 顺序性

Think of a ChannelFuture as a placeholder for the result of an operation that’s to be executed in the future. *When* exactly it will be executed may depend on several factors and thus be impossible to predict with precision, but it is certain that it *will* be executed. Furthermore, all oper- ations belonging to the same Channel are guaranteed to be executed in the order in which they were invoked.

## ChannelHandlers

Typical uses for ChannelHandlers include:

* Transforming data from one format to another
* Providing notification of exceptions
* Providing notification of a Channel becoming active or inactive
* Providing notification when a Channel is registered with or deregistered from  an EventLoop
* Providing notification about user-defined events

## Zero-copy

Zero-copy is a feature currently available only with NIO and Epoll transport. It allows you to quickly and efficiently move data from a file system to the network without copying from kernel space to user space, which can significantly improve perfor- mance in protocols such as FTP or HTTP. This feature is not supported by all OSes. Specifically it is not usable with file systems that implement data encryption or com- pression—only the raw content of a file can be transferred. Conversely, transferring files that have already been encrypted isn’t a problem.

In our discussion of transports (see section 4.2) we mentioned the zero-copy fea- ture of NIO, which eliminates copying steps in moving the contents of a file from the file system to the network stack. All of this happens in Netty’s core, so all that’s required is that the application use an implementation of interface FileRegion, defined in the Netty API documentation as “a region of a file that is sent via a Channel that supports zero-copy file transfer.”

This listing shows how you can transmit a file’s contents using zero-copy by creat- ing a DefaultFileRegion from a FileInputStream and writing it to a Channel.

**Listing 11.11 Transferring file contents with FileRegion**

FileInputStream in = new FileInputStream(file);

FileRegion region = new DefaultFileRegion(

in.getChannel(), 0, file.length());

channel.writeAndFlush(region).addListener(

new ChannelFutureListener() {

@Override

public void operationComplete(ChannelFuture future)

}

throws Exception {

if (!future.isSuccess()) {

Throwable cause = future.cause();

// Do something

}

});

**Creates a FileInputStream**

**Creates a new Default- FileRegion for the full length of the file**

**Sends the DefaultFile- Region and registers a ChannelFutureListener**

**Handles failure**

This example applies only to the direct transmission of a file’s contents, excluding any processing of the data by the application. In cases where you need to copy the data from the file system into user memory, you can use ChunkedWriteHandler, which pro- vides support for writing a large data stream asynchronously without incurring high memory consumption.

## ByteBuf

A *derived buffer* provides a view of a ByteBuf that represents its contents in a specialized way. Such views are created by the following methods:

* duplicate()
* slice()
* slice(int, int)
* Unpooled.unmodifiableBuffer(...)
* order(ByteOrder)
* readSlice(int)  Each returns a new ByteBuf instance with its own reader, writer, and marker indices. The internal storage is shared just as in a JDK ByteBuffer. This makes a derived buffer inexpensive to create, but it also means that if you modify its contents you are modify- ing the source instance as well, so beware.
* **BYTEBUF COPYING** If you need a true copy of an existing buffer, use copy() or copy(int,int). Unlike a derived buffer, the ByteBuf returned by this call has an independent copy of the data.

# HTTP

## 基于zookeeper的分布式session实现

<http://www.open-open.com/lib/view/open1325412725125.html>

## [Nginx+resin session问题解决](http://blog.chinaunix.net/uid-14007440-id-3150269.html)

<http://blog.chinaunix.net/uid-14007440-id-3150269.html>

# 测试

## 负载生成器，压力测试。

开源工具

<http://faban.org>

# SQL

## 查看7天内 都有充值的玩家（1号到7号）

原理，先查每个玩家每天的充值数额，再过滤1号到7号当中的玩家是否连续充了7天

select a1,a2,sum(a3) from(

SELECT b.openid AS a1,

b.playerName AS a2,

SUM(amt/10.0+ISNULL(pubacct\_payamt\_coins,0)+ISNULL(payamt\_coins,0))/10.00 AS a3,

from\_unixtime(b.ts,"yyyy-MM-dd") as a4

FROM t\_callback\_pay b

WHERE b.ServerType = 'apple' and from\_unixtime(b.ts,"yyyy-MM-dd")>='2016-10-01' and from\_unixtime(b.ts,"yyyy-MM-dd")<='2016-10-07'

GROUP BY b.openid,

b.playerName,

a4

having a3>0.00 ) as t group by a1,a2 having count(a2)=7 order by sum(a3)

# Thrift

C# 编译 avpr 文件

avrogen.exe -p <.avrp file> <output dir>

avrogen.exe -p D:\avro\codegen\Release\data.avpr D:\avro\codegen\Release\

java 编译 avpr

cd inflow/resources/avro

java -jar ../tools/java/avro-tools-1.7.7.jar compile protocol data.avpr ../..//src

编译 thrift

thrift -r --gen csharp service2.thrift

thrift -r --gen java service2.thrift

thrift -r --gen php service2.thrift

# 微信开发资源

## 加解密

<http://qydev.weixin.qq.com/wiki/index.php?title=%E5%8A%A0%E8%A7%A3%E5%AF%86%E5%BA%93%E4%B8%8B%E8%BD%BD%E4%B8%8E%E8%BF%94%E5%9B%9E%E7%A0%81>

# MyBatis

## MyBatis传入多个参数的问题

http://www.tuicool.com/articles/q2mui2

# 统计

充值统计

1、注册用户

在（第一天，第三天，第一周，第二周，第三周，第一月，第二月，第三月）有多少人充值，总共充了多少钱