# Part 1 Summary

## 1.1 前端优化

* 合并css、js，减少Http请求
* 启用压缩，减少数据传输
* 使用浏览器缓存。生成新的js时，只需更新Html中引用。
* CSN加速：缓存静态资源，如图片、文件、CSS、JavaScript脚本、静态网页等。
* 反向代理

## 1.2 应用服务器

系统架构

* 缓存：加快数据读取，降低DB访问压力。网站遇到性能瓶颈，第一个想到的解决方案
* 异步操作：加快请求响应及实现削峰。
* 使用集群：使用负载均衡，有更好的响应延迟，提高吞吐能力。

代码优化

* 多线程,最大发挥CPU和处理IO阻塞。
* 使用NIO(非阻塞，基于事件驱动)
* 资源复用，比如单例和线程池。
* 合理的数据结构
* 减少垃圾回收。
* 尽量不使用锁,减小锁的力度。

## 1.4 Protocol

### HTTP 1.1

**最大并发数限制**

并发指建立的TCP连接数。可以一个连接发送多个请求。Server端按发送顺序处理。

# Part 2 性能优化

## 2.1 指标

### 2.1.1 性能指标

PV QPS TPS

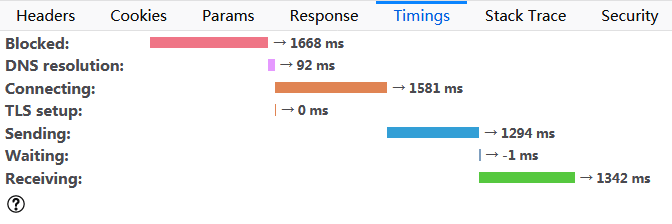
#### Concurrency

并发数。服务器能同时处理的并发TCP连接数。

#### Response

响应时间。发出请求到收到相应数据的时间

##### FireFox



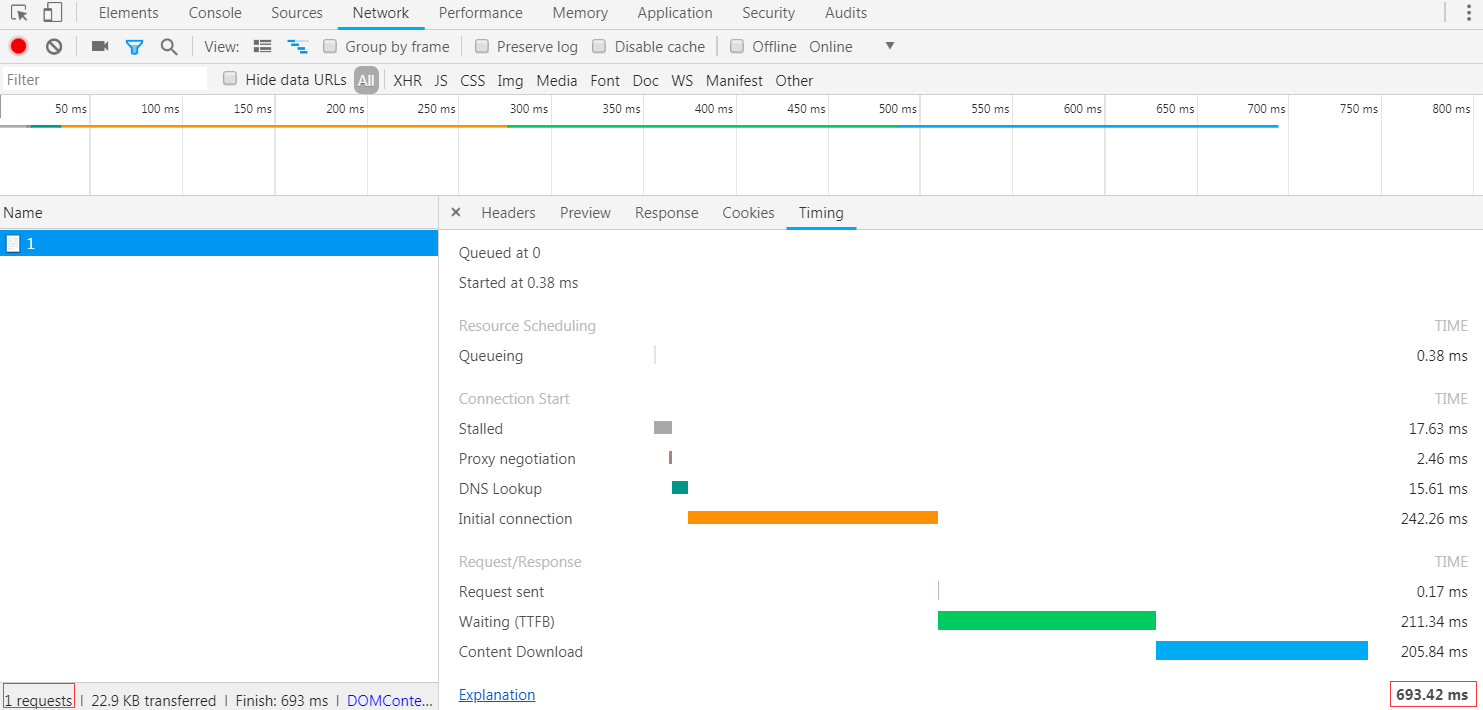
Connecting:TCP建立连接时间

Waiting : Waiting for a response from the server.

Receiving :read entire response from server

##### Chrome

[Reference](https://developers.google.com/web/tools/chrome-devtools/network-performance/understanding-resource-timing)



Queuing：阻塞的原因

Finish： timeline里所有request总时间

DOMContentLoaded ： 加载html dom内容，不包括js，css等

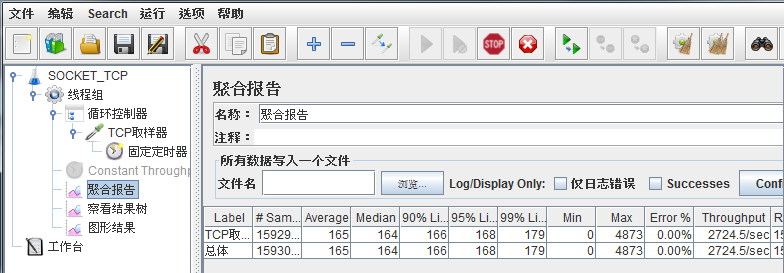
Load： 整个页面的时间

#### 吞吐量

系统单位处理的请求数（QPS）。一个TCP连接可同时发出多个请求，吞吐量是衡量服务器性能的指标。

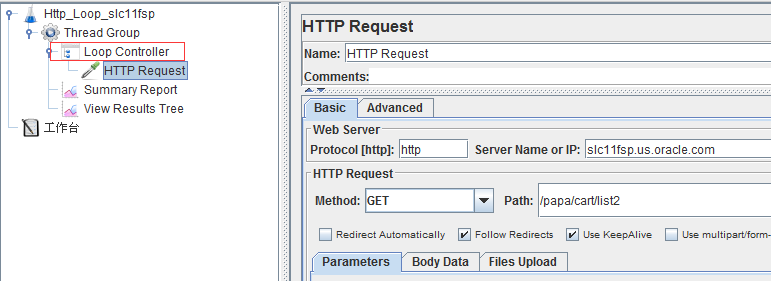
##### JMeter

TCP



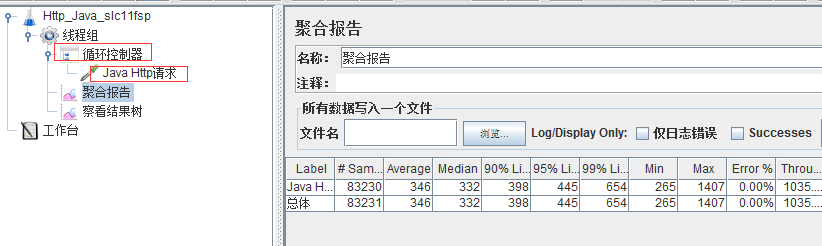
HTTP

短连接



此处是自带的HTTP Request，虽然Loop，但每个request都会重新建立TCP连接，使用完close。

长连接



java中不要close连接

性能计数器

包括System Load、对象和线程数、内存、CPU、磁盘、网络IO

### 2.1.2 性能测试方法

* 性能测试：性能预期
* 负荷测试：找到性能安全临界线
* 压力测试：超过安全符合继续施压，获得最大压力承受能力

压力测试

<https://blog.csdn.net/wyingquan/article/details/414608>

### 2.1.3 测试流程

很多时候在线上出现问题时，往往会伴随着某些指标的异常。大部分情况下，在问题发生之前，某些指标就会提前有异常显示。

step 1 概括

top查看整个系统负荷，正常情况单核负荷在0.7，四核就是3一下。

top –Hp 查看java线程占用系统资源情况，CPU。MEM。。。

ps 查看java进程，jstack单个线程，jconsole 线程的变化，jvisula线程池

## 2.2 线程

ps 查看java进程，jstack单个线程，jconsole 线程的变化，jvisula线程池

### 2.2.1 Base

理想状态CPU，内存，IO外设都满负荷工作，大部分情况瓶颈都在IO的读写部分，多线程适合计算性case。

线程切换开销

[Reference](http://ifeve.com/java-context-switch/)

上下文切换会带来直接和间接两种性能开销

**直接消耗**: CPU寄存器需要保存和加载, 系统调度器的代码需要执行, 刷新TLB(TLB：translation lookaside buffer，快表，可以理解为页表缓冲，地址变换高速缓存), CPU 的pipeline需要刷掉;

**间接消耗**：多核的cache之间得共享数据,对于程序的影响要看线程工作区操作数据的大小。

切换时间实测

线程数量

**1) Max Number of threads:**

(MaxProcessMemory - JVMMemory - ReservedOsMemory) / ThreadStackSize

* MaxProcessMemory 指的是一个进程的最大内存。在32位的 windows下是 2G
* JVMMemory         JVM内存
* ReservedOsMemory  保留的操作系统内存
* ThreadStackSize      线程栈的大小(-Xss)

2)线程数量

((CUP时间+CUP等待时间)/CUP时间)\*CUP数量

### 2.2.1 TOMCAT

tomcat的线程优化，本质上就是java 的线程池优化。结合socket特点来优化。实现线程资源和任务的一种平衡。

Server

超过最大连接数，响应时间变长

最大连接数越大，cup load约高

线程数越大，超过500，响应时间变长。线程切换时间？

吞吐量是否应该在服务端测，排除传输时间干扰。

#### Manager

[Reference](https://tomcat.apache.org/tomcat-7.0-doc/manager-howto.html)

Server Status

<http://localhost:8080/manager/status>

<http://localhost:8080/manager/status?XML=true>

**Stage**

P: Parse and prepare request

S: Service

F: Finishing

R: Ready

K: Keepalive

TODO2 linux 命令 tomcat 每次只能accept一个request？

Thread Dump

<http://localhost:8080/manager/text/threaddump>

VM Info

<http://localhost:8080/manager/text/vminfo>

<http://localhost:8080/manager/status?XML=true>

#### 设置策略

maxThreads

maximumPoolSize

最大的线程数。即最大连接数。Tomcat7默认值200。maxThreads规定的是最大的线程数目，并不是实际running的CPU数量；实际running的是maxConnections。

maxThreads应设大点。因为，处理线程真正用于计算的时间可能很少，大多数线程都在等待；大多数时间可能在阻塞，如等待数据库返回数据、等待硬盘读写数据等。当然，maxThreads的值并不是越大越好，如果maxThreads过大，那么CPU会花费大量的时间用于线程的切换，整体效率会降低。

acceptCount

所有可用的线程数都被使用时，可以放到处理队列中的请求数。默认值10。

通过前面的介绍可以知道，虽然tomcat同时可以处理的连接数目是maxConnections，但服务器中可以同时接收的连接数为maxConnections+acceptCount 。acceptCount的设置，与应用在连接过高情况下希望做出什么反应有关系。如果设置过大，后面进入的请求等待时间会很长；如果设置过小，后面进入的请求立马返回connection refused

maxThreads和acceptCount。如果要加大并发连接数，应同时加大这两个参数。

maxConnections

maxConnections的设置与Tomcat的运行模式有关。如果tomcat使用的是BIO，那么maxConnections的值应该与maxThreads一致；如果tomcat使用的是NIO，那么类似于Tomcat的默认值，maxConnections值应该远大于maxThreads。

KeepAlive

maxKeepAliveRequests： 保持请求数量，默认值100。

keepAliveTimeout 默认值60000

keepAliveTimeout VS connnectionTimeout

**connnectionTimeout**： 网络连接超时，默认值60000，单位：毫秒。通常可设置为30000毫秒。设置为0表示永不超时，这样设置有隐患的

minSpareThreads：

最小空闲线程数。Tomcat初始化时创建的线程数。默认值25。

maxSpareThreads： 最大空闲线程数。一旦创建的线程超过这个值，Tomcat就会关闭不再需要的socket线程。默认值75。

**acceptorThreadCount**： acceptor数量

bufferSize： 输入流缓冲大小，默认值2048 bytes。

compression： 压缩传输，取值on/off/force，默认值off。 其中和最大连接数相关的参数为

enableLookups： 是否反查域名，默认值为true。为了提高处理能力，应设置为false

server.xml

1. **<Connector** port="8080" protocol="HTTP/1.1"
2. connectionTimeout="20000" maxThreads="1000" minSpareThreads="60"
3. maxSpareThreads="600" acceptCount="120"  redirectPort="8443" URIEncoding="utf-8"**/>**

### 2.2.3 Tools

#### top

**Load Average**

一段时间 (1 分钟、5分钟、15分钟) 内平均 Load

系统负荷正常范围：

* 大于0.7，必须开始调查了，问题出在哪里，防止情况恶化。
* 持续大于1.0，必须动手寻找解决办法，把这个值降下来。
* 当系统负荷达到5.0，就表明系统有很严重的问题，长时间没有响应，或者接近死机了。

以上指标基于单CPU的，四核机器的负载最好保持在3(4\*0.7 = 2.8)以下。

%CPU %MEM

**命令补充**

监控最小单位是进程，看不到java线程和客户连接数，可用ps和netstat两个命令来补充top的不足。

#### netstat

1.Linux下查看并发连接数，一个浏览器可能建立几个连接，非并发用户数

netstat -na | grep ESTABLISHED | grep 8080 | wc -l

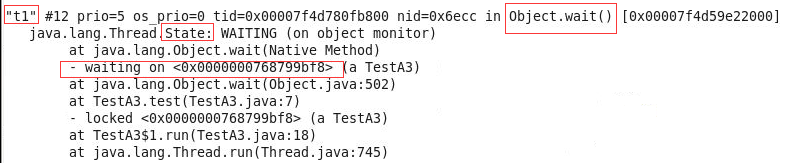
#### jstack

[Reference](http://www.hollischuang.com/archives/110)

生成线程快照。

**主要目的**：定位线程出现长时间停顿的原因，如线程间死锁、死循环、请求外部资源导致的长时间等待等。 通过jstack来查看各个线程的调用堆栈，就可以知道没有响应的线程到底在后台做什么事情，或者等待什么资源。

**用法**：jstack <pid>



线程状态

A处为线程状态。NEW

调用修饰

表示线程在方法调用时,额外的重要的操作。线程Dump分析的重要信息。修饰上方的方法调用。

locked <地址> 目标：使用synchronized申请对象锁成功,监视器的拥有者。

waiting to lock <地址> 目标：使用synchronized申请对象锁未成功,在迚入区等待。

waiting on <地址> 目标：使用synchronized申请对象锁成功后,释放锁幵在等待区等待。

parking to wait for <地址> 目标

**RUNNABLE**

**locked**进入synchronized方法, 申请对象锁成功才会出现此句



BLOCKED

**waiting to lock**申请锁失败，申请成功应该是-locked <地址>。



WATING

**waiting on ：**synchronized修饰的方法调用wait()。Thread释放锁后会**等待锁**



申请对象锁成功后,释放锁进入等待区。线程状态：WAITING

parking to wait for <地址> 目标

TIMED\_WAITING

at java.lang.Thread.sleep

**Thread State**

Wait on condition

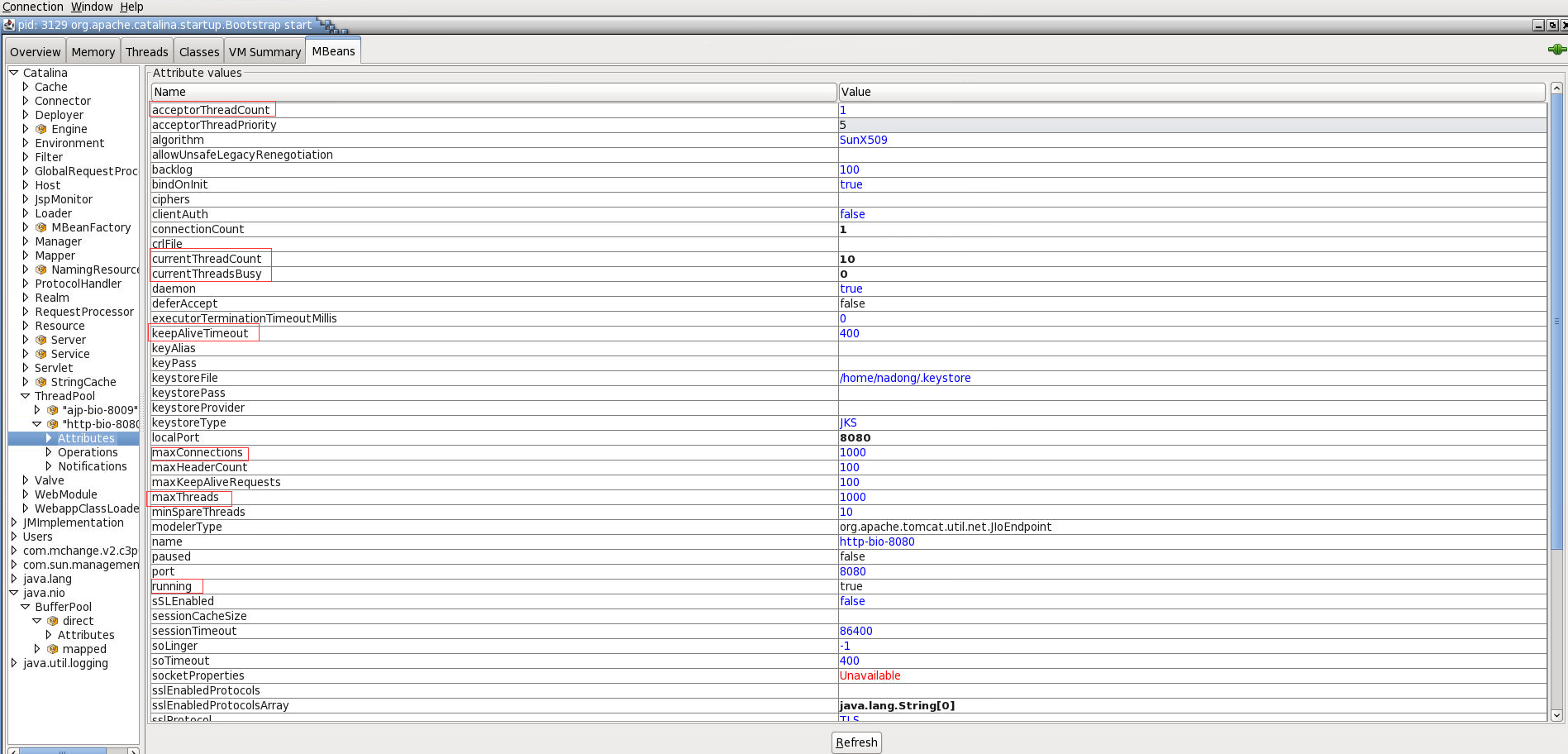
死锁

http://www.hollischuang.com/archives/2409

http://www.hollischuang.com/archives/110

#### jconsole

jconsole查询Acceptor属性



currentThreadCount

currentThreadBusy 线程池

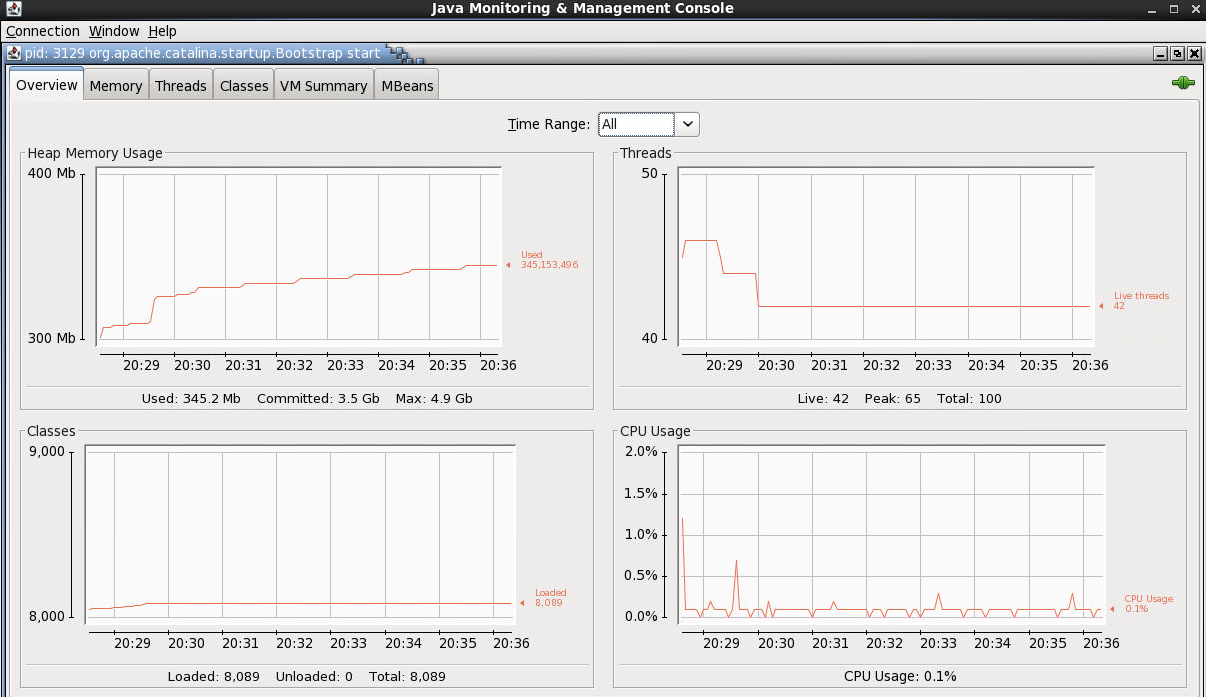
#### jconsole

配置

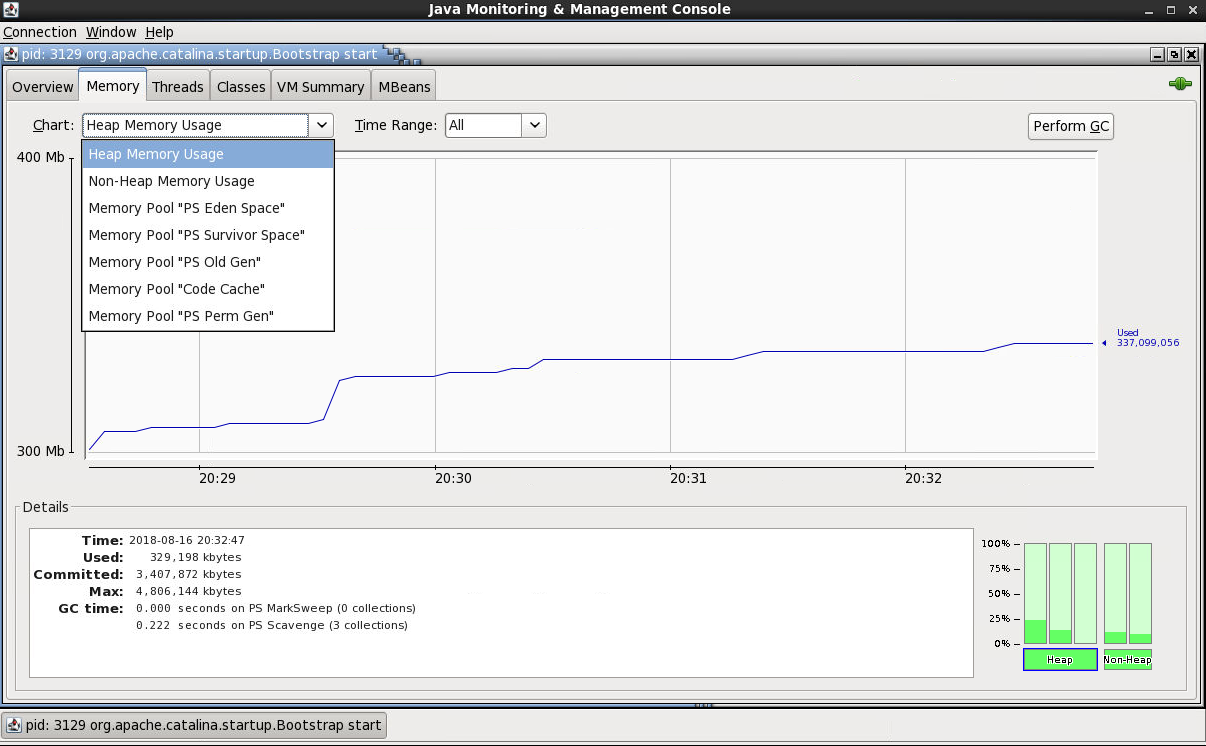
JAVA\_OPTS

-Djava.rmi.server.hostname=127.0.0.1 -Dcom.sun.management.jmxremote.port=8090 -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false

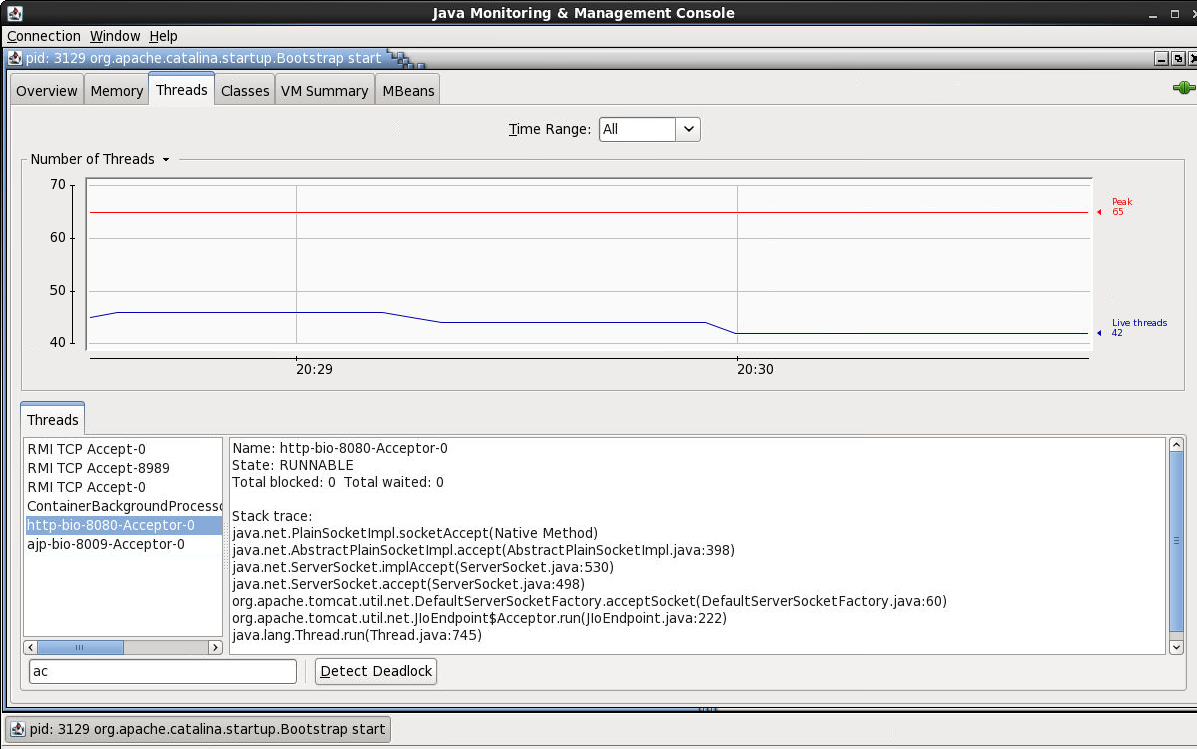
Overview



Memory



Thread



Total blocked

Total waited:

**state**

Wait on condition

#### Jvisual

JDK自带性能检测工具



**Running**: thread is still running.   
**Sleeping**: thread is sleeping (method yield () was called on the thread object)  
**Wait**: thread was blocked by a mutex or a barrier, and is waiting for another thread to release the lock

**Park**: parked threads are suspended until they are given a permit. Unparking a thread is usually done by calling method unpark () on the thread object  
**Monitor**: threads are waiting on a condition to become true to resume execution

## 2.3 内存

### Tools

#### jstat

jstat -option pic < interval >

**gcutil**



S0：年轻代中第一个survivor（幸存区）已使用的占当前容量百分比   
S1：年轻代中第二个survivor（幸存区）已使用的占当前容量百分比   
E：年轻代中Eden（伊甸园）已使用的占当前容量百分比   
O：old代已使用的占当前容量百分比

P：perm代已使用的占当前容量百分比   
YGC：从应用程序启动到采样时年轻代中gc次数   
YGCT：从应用程序启动到采样时年轻代中gc所用时间(s)

FGC：从应用程序启动到采样时old代(全gc)gc次数   
FGCT：从应用程序启动到采样时old代(全gc)gc所用时间(s)   
GCT：从应用程序启动到采样时gc用的总时间(s)   
**gc**



 S0C：年轻代中第一个survivor（幸存区）的容量 (byte)   
 S1C：年轻代中第二个survivor（幸存区）的容量 (byte)   
 S0U：年轻代中第一个survivor（幸存区）目前已使用空间 (byte)   
 S1U：年轻代中第二个survivor（幸存区）目前已使用空间 (byte)   
 EC：年轻代中Eden（伊甸园）的容量 (byte)   
 EU：年轻代中Eden（伊甸园）目前已使用空间 (byte)   
OC：Old代的容量 (byte)   
OU：Old代目前已使用空间 (byte)   
PC：Perm(持久代)的容量 (byte)   
PU：Perm(持久代)目前已使用空间 (字节)

**gcnew**

年轻代对象的信息



**gcold**

 old代对象的信息



**gcoldcapacity**

old代对象的信息及其占用量



OGCMN：old代中初始化(最小)的大小 (字节)   
OGCMX：old代的最大容量 (字节)   
OGC：old代当前新生成的容量 (字节)   
OC：Old代的容量 (byte)

**gcnewcapacity:**



 NGCMN：young初始化(最小)的大小 (字节)   
 NGCMX：年轻代(young)的最大容量 (字节)   
 NGC：年轻代(young)中当前的容量 (字节)

S0CMX：年轻代中第一个survivor（幸存区）的最大容量 (字节)   
S1CMX ：年轻代中第二个survivor（幸存区）的最大容量 (字节)

ECMX：年轻代中Eden（伊甸园）的最大容量 (字节)   
**gcpermcapacity**

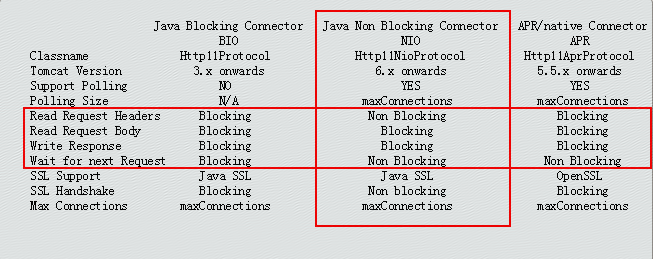
 PGCMN：perm代中初始化(最小)的大小 (字节)   
  PGCMX：perm代的最大容量 (字节)     
  PGC：perm代当前新生成的容量 (字节)   
PC：Perm(持久代)的容量 (byte)

#### jmap

## 2.4 IO优化

配置文件server.xml，略。

IO模型支持



tomcat要实现servlet规范所以不能最大发挥NIO的特性，servlet3.0之前完全是同步阻塞模型，Read http body 以及 response还是同步阻塞，因为servlet规范规定的就是这样。

### Tools

#### iostat

iostat -x 1 10

#### pidstat

查看单个线程，展示I/O统计，每秒更新一次

pidstat -d pid

pidstat |grep mysql

# Part 3 DB

## 3.1 MYSQL

连接

mysql –uroot [-h slc11fsp] –p

查看版本mysql -V;



### 概括

查看mysql占用系统资源

top –p pis

### information\_schema

show databases/tables;

**Status**：Syntax：show [global/session] status like '%xxx%';

**Variables** :Syntax：show variables like '%max\_connections%';

**Innodb** :Syntax : show [global] status like 'Innodb\_%'

### Connections

show full processlist;



每一条代表一个Thread，一个Tcp连接

Threads\_connected 当前连接数。只有数量值，show processlist有更详细的信息;

connections 试图连接mySQL服务器的次数

Max\_used\_connections

Lock

Syntax: show status like '%Innodb\_row\_lock%';



Innodb\_row\_lock\_current\_waits 当前阻塞线程数

Innodb\_row\_lock\_waits 总共wait次数

2.select \* from information\_schema.innodb\_locks;

查看lock具体数据



lock\_trx\_id找到事务

lock\_table+lock\_data找到具体锁定的数据。

Lock\_mode: X/S

lock\_type： RECORD行锁。

### Questions

Slow\_queries： 超过long\_query\_time数量。show variables like "%long\_query\_time%";

Questions ： 发往服务器的查询的数量。统计开始时间点为service start。

DDL统计

Innodb\_rows\_read/inserted/deleted/updated

Innodb\_data\_read/writes

buffer

Threads\_running 不在睡眠的线程数量。   
Uptime 服务器工作了多少秒。

### 数据大小

select data\_length,index\_length from information\_schema.tables where table\_schema='bigdata' and table\_name = 'ld\_cart';

单位是byte，优化显示

select concat(round(sum(data\_length/1024/1024),2),'MB') as data\_length\_MB,concat(round(sum(index\_length/1024/1024),2),'MB') as index\_length\_MB from information\_schema.tables where table\_schema='bigdata' and table\_name = 'fund\_history;

行数select count(1) from ld\_cart;

idex

show index from bigdata.fund;

### Transaction

开启事务 start transaction;

select \* FROM information\_schema.innodb\_trx\G;



trx\_id

trx\_state: RUNNING/LOCK WAIT

trx\_mysql\_thread\_id

trx\_stared 事务开始时间

trx\_wait\_stated:block事务wait stated

trx\_query:

trx\_rows\_locked 锁定row行数

trx\_tables\_locked 锁定table数量

Timeout

show variables like 'innodb\_lock\_wait\_timeout';

innodb\_lock\_wait\_timeout 等待获取锁

Autocommit



show variables like 'autocommit';

set autocommit = 0;仅对当前连接生效

autocommit VS start transaction

[Reference](https://blog.csdn.net/qq_26941173/article/details/77872038)

set autocommit=0 所有语句自动开启事务，commit结束事务。  
start transaction 启动一个新事务。语句"挂起"自动提交模式

ReadOnly

查看 show global variables like "%read\_only%";

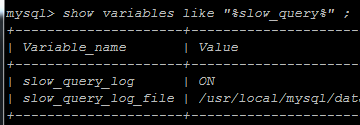
设置

### 日志

show variables like 'log%';

#### show query

show variables like "%slow\_query%" ;



开启日志

set global slow\_query\_log=ON;

分析

mysqldumpslow /usr/local/mysql/data/mysql/slc11fsp-slow.log