tomcat使用总结

# 系统需求

对于tomcat等服务器的参数调优，各个隐藏参数的了解

# 项目负责

# 项目前思考

# 项目前攻破

# 项目中使用的技术总结

## Tomcat调试

找到Bootstrap类，并在VM Options新增



## Tomcat原理



### 初始化servlet

ContextConfig

----configDefaultServlet 对应的conf 下的web.xml

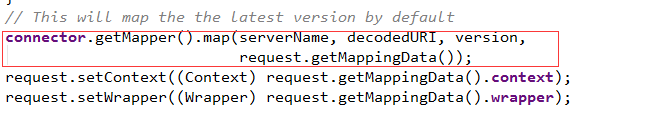
DefaultServlet、JspServlet

----configCustomServlet

context下面对应的web.xml

### 处理请求

Connector接收对应的socket连接，然后交由SocketProcessor进行处理。CoyoteAdapter搜索对应的context和servlet



通过管道式职责链，进行逐层调用处理

职责链关系：

Enginer

---Host1(localhost)

---Context ---project1

---wrapper(Servlet2)

---wrapper(Servlet2)

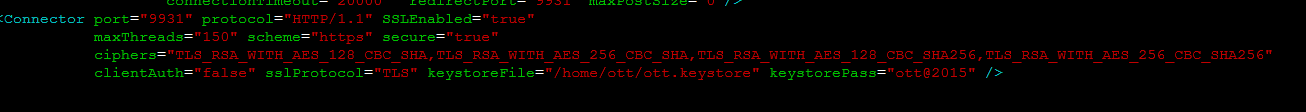
---Context ----project2

….

----Host2(127.0.0.1)



## https:



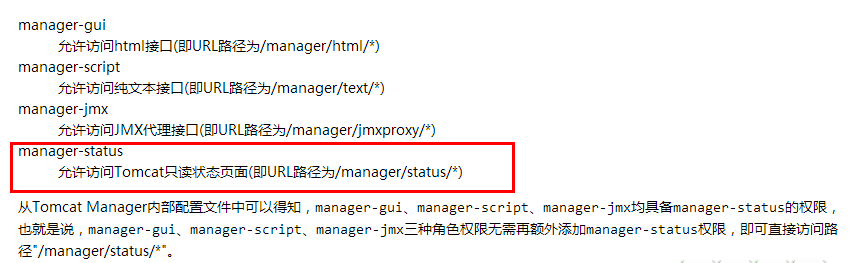
配置加密算法，可以使客户端加密加快

## 查看线程情况

jstack pid 验证可以打出对应的线程状态

但是试了linux下的Kill -3 pid没有效果

## Tomcat监控



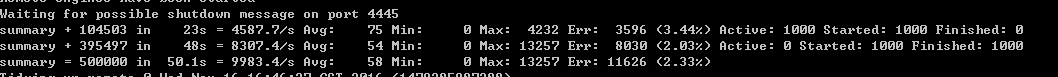
## Tomcat的运行模式

### BIO

Blocking IO，顾名思义，即阻塞式I/O操作，表示Tomcat使用的是传统的Java I/O操作(即java.io包及其子包)。Tomcat在默认情况下，就是以bio模式运行的。遗憾的是，就一般而言，bio模式是三种运行模式中性能最低的一种。

原理：通过socket.accept，阻塞等待请求，然后发起线程执行socket的数据

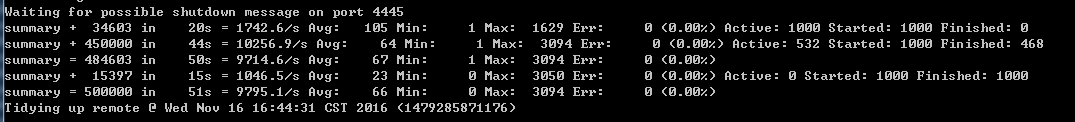
测试结果



### NIO

nio(new I/O)，是Java SE 1.4及后续版本提供的一种新的I/O操作方式(即[java](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=128&fv=18&is_app=0&jk=20b1e9adb3feb9a6&k=java&k0=java&kdi0=0&luki=6&mcpm=0&n=10&p=baidu&q=97086099_cpr&rb=0&rs=1&seller_id=1&sid=a6b9feb3ade9b120&ssp2=1&stid=9&t=tpclicked3_hc&td=1691595&tu=u1691595&u=http%3A%2F%2Fwww%2E365mini%2Ecom%2Fpage%2Ftomcat%2Dconnector%2Dmode%2Ehtm&urlid=0" \t "_blank).nio包及其子包)。Java nio是一个基于[缓冲区](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=128&fv=18&is_app=0&jk=20b1e9adb3feb9a6&k=%BB%BA%B3%E5%C7%F8&k0=%BB%BA%B3%E5%C7%F8&kdi0=0&luki=1&mcpm=0&n=10&p=baidu&q=97086099_cpr&rb=0&rs=1&seller_id=1&sid=a6b9feb3ade9b120&ssp2=1&stid=9&t=tpclicked3_hc&td=1691595&tu=u1691595&u=http%3A%2F%2Fwww%2E365mini%2Ecom%2Fpage%2Ftomcat%2Dconnector%2Dmode%2Ehtm&urlid=0" \t "_blank)、并能提供非阻塞I/O操作的Java API，因此nio也被看成是non-blocking I/O的缩写。它拥有比传统I/O操作(bio)更好的并发运行性能。要让Tomcat以nio模式来运行也比较简单

原理：采用ServerSocketChannel的accept，也是阻塞方式等待客户端连接，但是不是直接发起线程处理。而是通过往Poller中添加事情，唤起其处理。



#### 源码分析：



Acceptor在接收客户端的Socket，用了传统的ServerSocketChannel的accept的，这里也是采用Blocking的方式；接收到客户端请求后，添加PollerEvent到Poller里面，并通知

Poller消费Acceptor生产的PollerEvent，然后采用Selector的nio方式来绑定READ的事件，并交由Worker进行处理

Worker处理Socket的数据读取及解析，并提到值Executor中执行

### ARP

待研究

## Tomcat配置参数

### server.xml

<Connector port="80" protocol="HTTP/1.1" maxThreads="600"

minSpareThreads="100" maxSpareThreads="500" acceptCount="700"

connectionTimeout="20000" redirectPort="8443" />

#### maxThreads

最大处理的线程数，tomcat最多可以处理的线程数，即当前tomcat所能接受的请求，超出的话则会等待，默认值200

#### minSpareThreads

最小空闲线程数，即默认会开启一定量的空闲线程，等待请求

#### maxSpareThreads

最大空闲线程数，即默认会开启最大量的空闲线程，如果当前空闲的线程超出这个数量，则会进行回收。在tomcat 7.0后，该属性已被废除

#### acceptCount

指定当所有可以使用的处理请求的线程数都被使用时，可以放到处理队列中的请求数，超过这个数的请求将不予处理，默认值100

#### maxConnections

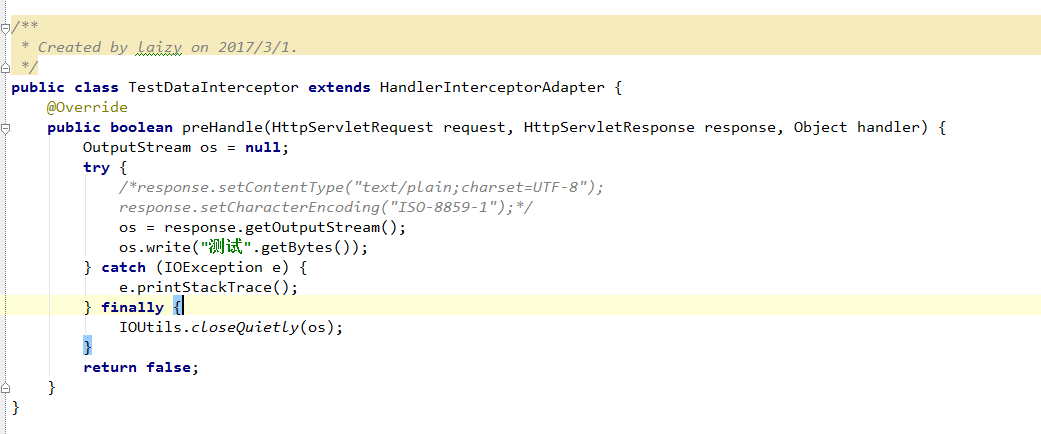
tomcat最多同时处理的请求数，当达到最大请求数后，会进入等待队列，这时候等待队列的数量就是acceptCount的数量，默认值是maxThreads

#### URIEncoding

Request的请求的编码格式，不过目前没有找到response的默认编码格式(ISO-)的修改，所以如果在response响应到前台的串，需要使用getBytes(“utf-8”)来指定编码，否则会产生中文乱码。还必须加上response.setContentType(“text/html;charset=utf-8”);默认是ISO-8859-1

# 编码规则

## Response乱码



1. 服务器输出一个普通的中文信息(测试)，chrome展示正常，firefox展示不正常。浏览器的转码不一样
2. 在response中的header添加了ContentType:text/plain;charset=UTF-8，浏览器都正常了，但是采用telnet，和curl，展示的还是乱码。因为contentType是对http协议有效

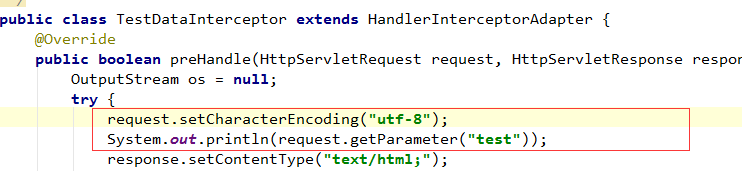
SetContentType和setCharacterEncoding都是设置响应的方式

1. 设置characterCode为ISO-8859-1，但是在html中设置了meta charset=’utf-8’，显示为乱码

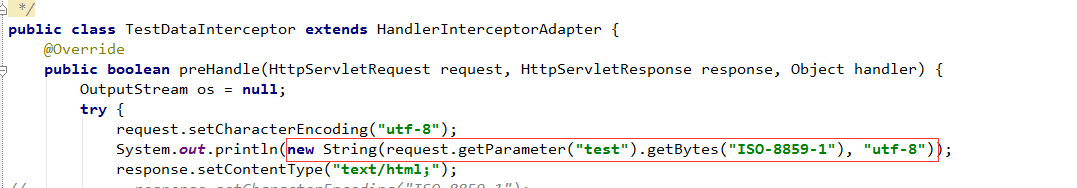


## Request乱码

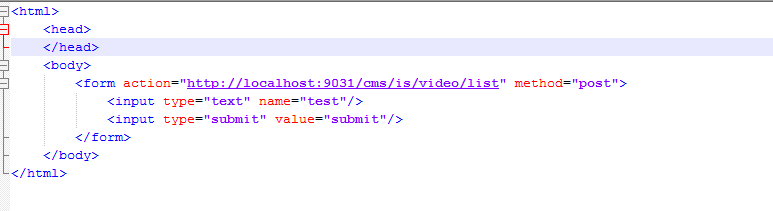
1. 采用get方式，往url塞参数，这时候后台get，没有配置UriEncoding，那么后台获取的乱码
2. 通过request.setCharactorEncoding的方式，输出还是乱码

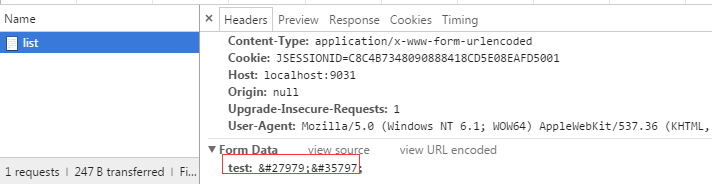


1. 配置tomcat的uriEncoding，或者getByte(“ISO-8859-1”),utf-8转码的话，输出能够正常



1. Post方式请求，没有设置meta，提交请求是用unicode编码转化了





1. 在html页面上加上<meta http-equiv="Content-Type" content="text/html;charset=utf-8">之后，提交中文，后台能够正常获取
2. 后台如果采用request.setChar..Encoding(iso-8859-1)的话，那么前后两端的参数转化不一样，就会乱码

# 项目后期思考

# 项目中觉得亮点

## 问题1：acceptCount在linux下无法正常工作？

在linux环境下，设置acceptCount=2，maxConnections=2，采用jmeter进行发送25次请求，那么应该是2个请求进行处理，2个请求等待，其他请求都应该被refuse，但是事实是请求成功的数量有5个是成功，5个是recv failed(被迫关闭连接)



## 问题2：linux配置了https，而linux到tomcat是使用http，导致tomcat的资源不能访问

问题原因：由于在后台拼接AbsolutePath的时候，获取到的协议是https，但是tomcat的获取的port是使用http，导致资源拼接起来是https://xxx:httpPort/xx。如果不使用下列配置的话，获取的都是nginx的请求，所以需要配上forward的信息；参考链接(http://feitianbenyue.iteye.com/blog/2056357)

解决方法：

<Valve className="org.apache.catalina.valves.RemoteIpValve" remoteIpHeader="X-Forwarded-For"

remoteIpProxiesHeader="X-Forwarded-By"

protocolHeader="X-Forwarded-Proto"

httpServerPort="9012" httpsServerPort="9912"

## Host、Context配置的使用

想通过项目配置一下默认项目访问

<Contenxt docBase=”project” path=”” />

问题1：没有配置host的appBase目录，导致tomcat下面的整个目录都变成webapps的项目，并被拷贝到work目录下

问题2：虽然配置Context对应的path，但是由于默认会deploy目录，所以/xxx还是照样能访问

## tomcat和nginx相比，什么效率明显小于nginx

线程和进程的差别：同一个进程的多个线程，可共用进程内的资源，线程是进程的一个实体。nginx可以设置多个进程(资源更多)，而tomcat就是一个进程。nginx应该还有本地缓存机制，而tomcat没有。