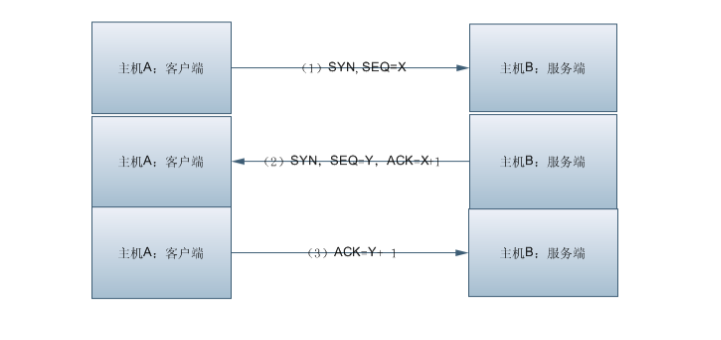
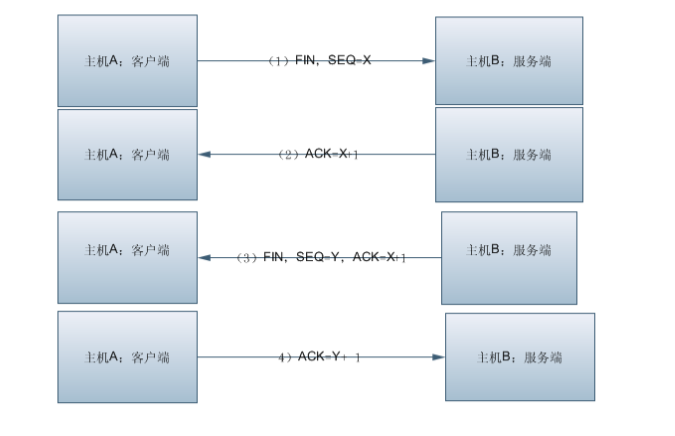
1. Java内存溢出和内存泄漏的区别
   1. Java内存溢出就是指要分配的内存超出了系统能给你的；例如你申请了一个int，但给它赋值long，就会内存溢出；
   2. Java内存泄漏是指程序的逻辑问题造成内存无法释放，这样的话内存会被占用光，例如无限死循环，会导致系统越来越慢，甚至当掉；
2. Java虚拟机的垃圾回收机制
3. Java的Error和Exception的区别和联系
4. NFS——网络文件系统（network file system）
5. 查看线程状态
   1. ps –ef | grep java
   2. jstack 进程号
6. 性能测试
   1. 并发用户数
   2. 虚拟机数
   3. 平均响应时间
   4. 请求丢包率
   5. CPU使用率
   6. 内存使用率
7. java.lang.Thread.State: RUNNABLE
   1. 代码中有循环或者递归造成程序死循环，线程的堆栈会越来越高，一会儿就会内存溢出，导致服务挂机；
   2. 死锁或者阻塞；例如静态变量的死锁或者流未关闭等等；
   3. 在while循环中加入延时函数，释放系统控制权；
   4. HashMap的使用会在多线程高并发中发生死锁问题，建议使用ConcurrentMap
8. CPU100%问题
   1. 服务器CPU一般在75%以下为正常
   2. 造成的原因可能是程序有死循环、IO没释放、静态变量、线程不安全的Java类
9. TCP/IP的三次握手
   1. 第一次握手——客户端首先给服务器端发送一个特殊的TCP数据段，该数据段不包含应用层数据，并将头部中的SYN位设置为1，所以该数据段被称为SYN数据段；另外客户端选择一个初始化序列号SEQ，设SEQ=X，并将这个编号放到初始的TCP SYN数据段的序列号字段中，该数据段被封装到一个IP数据报中，并发送给服务端
   2. 第二次握手——一旦装有TCP SYN数据段的IP数据报到达了服务器主机，服务器将以该数据报中提取出的TCP SYN，给该链接分配TCP缓冲区和变量，并给客户端TCP发送一个允许连接的数据段；这个允许连接的数据段也不包含任何应用层数据；但是，他的头部中含有3个重要信息；首先，SYN被设置为1，其次，TCP数据头部的确认字段被设置为X+1；最后服务端选择自己的初始顺序号，SEQ=y，并将该值改道TCP数据段头部的序列号字段中；
   3. 第三次握手——在接受到允许连接数据段之后，客户端也会给服务端连接分配缓冲区和变量；客户端也会给服务器发送另一个数据段，对服务器的允许连接数据段给出确认；



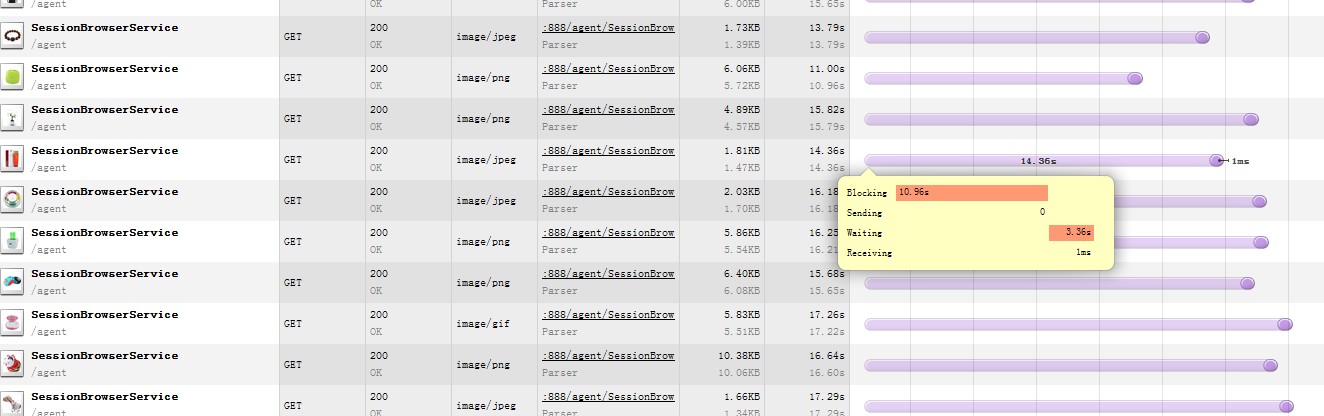
连接的释放过程：

1. 第一次握手——由进行数据通信的任意一方提出要求释放连接的请求报文段
2. 第二次握手——接收端收到请求后，会发送确认报文段，同时当接收端的所有数据数据也都已经发送完毕后，接收端会向发送端发送一个带有自己序号的报文段
3. 第三次握手：发送端收到接收端的要求释放连接的报文段后，发送反向确认



1. SYN——同步（初始）比特，当SYN=1时，表示这是一个连接请求或者连接接受报文
2. FIN——终止比特，用来释放一个连接；当FIN=1时，表示此报文段发送的数据已经发送完毕，并要求释放连接
3. ACK——确认比特，只有当ACK=1时，确认字段才有效，当ACK=0时，确定号无效
4. RST——复位比特，当RST=1时，表明TCP连接中出现严重错误（比如主机崩溃等原因）必须释放连接，通知一下对方；
5. URG——紧急比特，当URG=1时，表明紧急指针字段有效，它告诉系统此报文段中有紧急数据，应尽快传送（相当于优先级高的数据）
6. PSH——推送比特，接收端收到推送比特置1的报文段，就尽快地交付给接收应用进程，而不再等到整个缓存都填满了后再向上交付
7. 移动网络
   1. CMNET——提供了NAT服务，能够直接访问Internet终端服务器
   2. CMWAP——只提供了WAP代理和Http代理，只能用支持这两种协议的软件

1. 对于图片比较多的网站需要对图片域名进行分发，一个域名只能同时支持有限数量的请求；列如淘宝等图片较多的网站。



1. Java反射机制总结
   1. Class类
      1. Class clazz1 = String.class;
      2. String str = “abc”; Class clazz = str.getClass();//对实体类运用的方法
      3. Class clazz3 = Class.forName(“java.lang.String”);
   2. Field类
      1. Field[] fields1 = clazz1.getDeclaredFields();//获取所有申明的字段
      2. Field[] fields2 = clazz1.getFields();//获取所有的公共类字段
      3. Field fields1 = clazz1.getDeclaredFields(“value”);//获取指定字段名（所有申明的字段都可以获取）
      4. Field field2 = clazz1.getField(“CASE\_INSENSITIVE\_ORDER”);
   3. Method类
      1. Method[] method1 = clazz1.getDeclaredMethods();
      2. Method[] method2 = clazz1.getMethods();
      3. Method method1 = clazz1.getDeclaredMethod(“checkBounds”,new Class[]{byte[].class,int.class,int,class});
      4. Method method2 = clazz1.getMethod(“hashCode”);
   4. Constructor构造方法类
      1. Constructor[] cons1 = clazz1.getDeclaredConstructors();
      2. Constructor[] cons2 = class1.getConstructors();
      3. Constructor con1 = clazz1.getContructor(char[].class);
      4. Constructor con2 = clazz1.getDeclaredConstructor(new Class[]{int.class,int.class,char[].class});
   5. 由一个Class得到一个实体类
      1. String obj = (String)clazz1.newInstence();//类必须有默认构造函
   6. 由一个Constructor对象，得到类的实体
      1. String str2 = (String) con1.newInstance(new char[]{‘a’,’b’,’c’});
2. Jetty调优

（1）什么是Jetty ——HTTP服务器和Servlet/JSP容器（Http1.0,Http1.1）

协议支持：

Http、AJP（使用这种协议，具有更高的性能，因为它采用的是二进制传输。比HTTP的文本传输要有更高的效率）

IO支持：

BIO（阻塞）、NIO（非阻塞）

（2）Jetty做了什么

监听某接口，例如8080，等待用户连接请求

接收用户连接、数据

根据协议解析用户请求，例如HTTP协议，AJP协议

URL映射到具体上下文，即某个应用

转交给Servlet处理器

URL映射具体的Servlet

执行Servlet

返回响应数据

（3）Jetty使用方式

a.以内嵌容器形式运行

内嵌于其他应用程序，例如Eclipse、JBoss

需要应用程序组装Jetty的各大组件

可能需要Jetty的XML解析

EX：

Server server = new Server();//入口点

Connector connector = new SelectChannelConnector();//选择Connector

connector.setPort(8080);//设置监听端口

server.setConnectors(new Connector[]{connector});

WebAppContext webapp = new WebAppContext();//设定web上下文信息

webapp.setContextPath(“/”);

webapp.setWar(“./target/detail.war”);

server.setHandler(webapp);//加到主控类中

server.start();//启动服务器

server.join();

b.以Standalone形式运行

作为独立服务器运行

（4）Jetty性能调优点

a.线程池

线程池线程资源大小确定了服务器的服务能力

默认大小不一定能满足生产环境

线程分配方式决定了服务器的资源利用效率

固定线程处理多任务，代表：JDK的ThreadPoolExecutor

以最大线程数为限处理多任务，代表：Jetty自带的QueuedThreadPool

Work-stealing分配，Jetty目前没有实现

<Set name=”ThreadPool”>

<!—default queued blocking threadpool>

<New class=”org.eclipse.jetty.util.thread.QueuedThreadPool”>

<Set name=”minThreads”>10</Set>

<Set name=”maxThreads”>200</Set>

</New>

</Set>

b.Connectors（连接器）

选择Connector时，需要考虑

应用自身的特点、例如股票、聊天室

TCP连接数

Keep-Alive

Java BIO Connectors

SocketConnector（HTTP）

Ajp13SocketConnector（AJP）

SslSocketConnector（SSL）

Java NIO Connectors

SelectChannelConnector(HTTP)

SslSelectChannelConnector(SSL)

Acceptors

表示同时在监听read事件的线程数

默认值是1

典型值范围1~(处理器内核数+1)

对于NIO来说，设置为（）

maxIdleTime

表示连接最大空闲时间

默认值是200000，一般这个值都太大了

典型3000左右就足够了

对AJP来说，一般设置为－１，表示连接需要一直保持

LowResourcesMaxIdleTime

表示线程资源稀少时的maxIdleTime

默认值是-1，表示没有设置

一般设置一个该<=maxIdleTime

lowResourcesConnectors

只有NIO才有这个设置，表示连接空闲时的连接数，大于这个数将被shutdown

默认值是0，表示该设置没有生效

每个acceptor的连接数=(lowRescourceConnections + acceptors -1)/acceptors

AcceptQueueSize

连接被Accept前允许等待的连接数

即Socket的Backlog，默认为50

SoLingerTime

具体指定逗留时间（以毫秒为单位）

即Socket的setSoLinger,默认关闭

ResloveNames

是否反查getRemoteHost();

默认false

c.JVM

堆/栈内存大小调整

Xmx/xms 最大/最小堆大小

Xmn 新生代大小

-xx:MaxPermSize 持久代堆大小

垃圾分配回收算法

考虑暂停时间、吞吐量选择不同算法

串行/并行/并发收集

d.Content Cache

动态内容不会被Cache

静态内容才会被Cache

maxCacheSize

256,000,000

maxCachedFileSize

200,000,000

maxCachedFiles

2,048

useFileMappedBuffer

true

e.冗余组件去除

去除不需要构建的handler

例如：SessionHandler、ServletHandler

1. AJP协议
   1. AJP协议是面包协议，采用二进制代替文本形式，以提供性能。
   2. AJP协议包括四种数据类型：Byte、Boolean、Integer、String
2. SPDY协议
   1. 谷歌研发的一种新的网络协议，以最小化网络延迟，提升网络速度，优化用户的网络体验；SPDY并不是一种代替HTTP的协议，而是Http协议的增强；新协议的功能包括数据流的多路复用、请求优先级,以及HTTP[包头](http://baike.baidu.com/view/2777.htm)压缩。
   2. 该协议仍使用HTTP的方法和包头,只是对HTTP中负责连接管理和数据传输格式的部分进行了重写。
   3. SPDY的目的是减少网页加载时间.通过优先级和"复用"技术传输多个文件,而每个[客户端](http://baike.baidu.com/view/930.htm" \t "_blank)只需一个连接. 在设计上所有通讯是TLS加密的,并且gzip压缩. (而HTTP 的头部则是选择性的压缩). 此外,服务器可能提示或者甚至推送内容,而不是等待页面中的每个单独的请求.
3. 遇到特殊中文字符
   1. 例如：“尨”用GB2312以及GBK编码都显示乱码，这里使用GB18030编码，是GBK的扩充

HttpEntity entity = response.getEntity();

InputStream instream = entity.getContent();

**int** i = (**int**) entity.getContentLength();

**if** (i < 0)

i = 4096;

Reader reader = **new** InputStreamReader(instream, "GB18030");

CharArrayBuffer buffer = **new** CharArrayBuffer(i);

**try** {

**char**[] tmp = **new** **char**[1024];

**int** l;

**while** ((l = reader.read(tmp)) != -1) {

buffer.append(tmp, 0, l);

}

} **finally** {

reader.close();

}

System.*out*.println(buffer.toString());

1. 移动网络cmwap网关问题

众所周知，在国内的GPRS网络环境下，用户以cmwap接入点上网的情况居多。cmwap是中国移动的GPRS代理上网，因此在创建连接的时候，必须采用代理的方式进行连接。  
假设我连接的URL为：http://wap.uc.cn/；  
在标准代理中，是这样子的：  
Conection to 10.0.0.172:80  
GET http://wap.uc.cn/ HTTP/1.1  
Host: wap.uc.cn  
  
但是在J2ME中，程序并不能控制到TCP层，因此中国移动定义了一个私有协议字段：X-Online-Host。从移动要求的规范来看，我们只需要这样即可正常进行联网：  
Conection to 10.0.0.172:80  
GET / HTTP/1.1  
Host: 10.0.0.172  
X-Online-Host: wap.uc.cn  
实际上对于程序来说，我们只需要将原来连接的url及http header中的host替换成10.0.0.172，并添加一个http header字段：X-Online-Host，值为原来的host。然后继续采用直连的方式即可正常进行联网。  
  
这样问题就来了，很明显这是一个不标准的代理协议，不标准的地方在于：  
GET / HTTP/1.1  
因为在标准的代理协议规范中，定义了请求的必须是完整的URI。  
这个问题在平常并不会出现，但是当请求的URL比较特殊时，问题就很明显了。  
假设我连接的URL为：http://wap.baidu.com/logo.gif?img=http://wap.uc.cn/uc.png  
使用X-Online-Host协议的联网：  
Conection to 10.0.0.172:80  
GET /logo.gif?img=http://wap.uc.cn/uc.png HTTP/1.1  
Host: 10.0.0.172  
X-Online-Host: wap.baidu.com  
这样的请求到达移动网关之后，会被误发至http://wap.baidu.com/uc.png。但是实际上我们想要请求的是http://wap.baidu.com/logo.gif （?之后的表示变量）。  
因为，移动网关实际上就是一个HTTP的代理服务器，它对于X-Online-Host协议是这样处理的：  
截取请求头中的URL字段：  
如果没有http://字段的话，则将该字段作为相对URI，同X-Online-Host字段进行补全；  
如果有http://字段的话，则将该字段作为绝对URI，将host替换为X-Online-Host的值。