## 一、软件体系结构

###### 常见软件体系结构B/S、C/S

1.1　C/S

* C/S结构即客户端/服务器（Client/Server），例如QQ；
* 需要编写服务器端程序，以及客户端程序，例如我们安装的就是QQ的客户端程序；
* 缺点：软件更新时需要同时更新客户端和服务器端两端，比较麻烦；
* 优点：安全性比较好。服务器计算压力会被客户端分担一大部分.

1.2　B/S

* B/S结构即浏览器/服务器（Browser/Server）；
* 优点：只需要编写服务器端程序；
* 缺点：安全性较差。

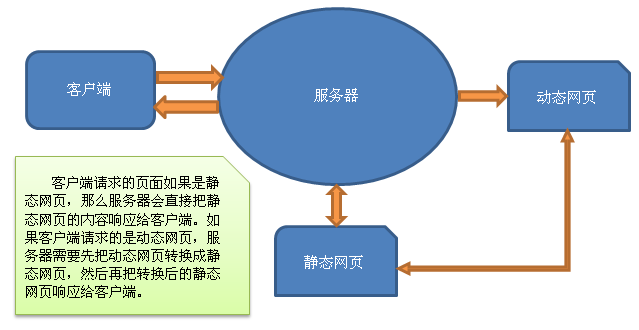
###### 2　WEB资源

2.1　Web资源介绍

* html：静态资源；
* JSP/Servlet：动态资源。

当然，除了JavaWeb程序，还有其他Web程序，例如：ASP、PHP等。

2.2　静态资源和动态资源区别



2.3　访问Web资源

打开浏览器，输入URL：

* http://www.itcast.cn:80/index.html

###### 3　Web服务器

Web服务器的作用是接收客户端的请求，给客户端作出响应。

对于JavaWeb程序而已，还需要有JSP/Servlet容器，JSP/Servlet容器的基本功能是把动态资源转换成静态资源，当然JSP/Servlet容器不只这些功能，我们会在后面一点一点学习。

我们需要使用的是Web服务器和JSP/Servlet容器，通常这两者会集于一身。下面是对JavaWeb服务器：

* Tomcat（Apache）：当前应用最广的JavaWeb服务器，；
* JBoss（Redhat红帽）：支持JavaEE，应用比较广；
* GlassFish（Orcale）：Oracle开发JavaWeb服务器，应用不是很广；
* Resin（Caucho）：支持JavaEE，应用越来越广；
* Weblogic（Orcale）：要钱的！支持JavaEE，适合大型项目；
* Websphere（IBM）：要钱的！支持JavaEE，适合大型项目；

## 二、Tomcat

###### 1　Tomcat概述

Tomcat服务器由Apache提供，开源免费。由于Sun和其他公司参与到了Tomcat的开发中，所以最新的JSP/Servlet规范总是能在Tomcat中体现出来。当前最新版本是Tomcat8，我们课程中使用Tomcat7。Tomcat7支持Servlet3.0，而Tomcat6只支持Servlet2.5！

###### 2　安装、启动、配置Tomcat

下载Tomcat可以到<http://tomcat.apache.org>下载。

Tomcat分为安装版和解压版：

* 安装版：一台电脑上只能安装一个Tomcat；
* 解压版：无需安装，解压即可用，解压多少份都可以，所以我们选择解压版。

2.1　Tomcat目录结构

安装版Tomcat的安装过程请参考day03\_res/Tomcat安装.doc文件。

把解压版Tomcat解压到一个没有中文，没有空格的路径中即可，建议路径不要太深，因为我们需要经常进入Tomcat安装目录。例如：F:\apache-tomcat-7.0.42

2.2　启动和关闭Tomcat

在启动Tomcat之前，我们必须要配置环境变量：

* JAVA\_HOME：必须先配置JAVA\_HOME，因为Tomcat启动需要使用JDK；
* CATALANA\_HOME(推荐不去配置)：如果是安装版，那么还需要配置这个变量，这个变量用来指定Tomcat的安装路径，例如：F:\apache-tomcat-7.0.42。
* 启动：进入%CATALINA\_HOME%\bin目录，找到startup.bat，双击即可；
* 关闭：进入%CATALINA\_HOME%\bin目录，找到shutdown.bat，双击即可；

startup.bat会调用catalina.bat，而catalina.bat会调用setclasspath.bat，setclasspath.bat会使用JAVA\_HOME环境变量，所以我们必须在启动Tomcat之前把JAVA\_HOME配置正确。

启动问题：

* 点击startup.bat后窗口一闪即消失：检查JAVA\_HOME环境变量配置是否正确；

2.3　进入Tomcat主页

1.访问：<http://localhost:8080>

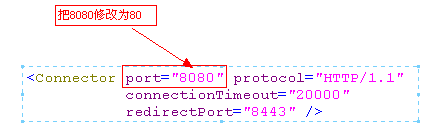
2. <http://127.0.0.1:8080>

3<http://本机IP地址:8080>

4<http://本机主机名:8080>

2.4　配置端口号

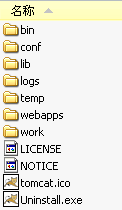
打开%CATALANA\_HOME%\conf\server.xml文件：



http默认端口号为80，也就是说在URL中不给出端口号时就表示使用80端口。当然你也可以修改为其它端口号。

当把端口号修改为80后，在浏览器中只需要输入：<http://localhost>就可以访问Tomcat主页了。

2.5　Tomcat的目录结构



* bin：该目录下存放的是二进制可执行文件，如果是安装版，那么这个目录下会有两个exe文件：tomcat6.exe、tomcat6w.exe，前者是在控制台下启动Tomcat，后者是弹出UGI窗口启动Tomcat；如果是解压版，那么会有startup.bat和shutdown.bat文件，startup.bat用来启动Tomcat，但需要先配置JAVA\_HOME环境变量才能启动，shutdawn.bat用来停止Tomcat；
* conf：这是一个非常非常重要的目录，这个目录下有四个最为重要的文件：
* server.xml：配置整个服务器信息。例如修改端口号，添加虚拟主机等；下面会详细介绍这个文件；
* tomcatusers.xml：存储tomcat用户的文件，这里保存的是tomcat的用户名及密码，以及用户的角色信息。可以按着该文件中的注释信息添加tomcat用户，然后就可以在Tomcat主页中进入Tomcat Manager页面了；
* web.xml：部署描述符文件，这个文件中注册了很多MIME类型，即文档类型。这些MIME类型是客户端与服务器之间说明文档类型的，如用户请求一个html网页，那么服务器还会告诉客户端浏览器响应的文档是text/html类型的，这就是一个MIME类型。客户端浏览器通过这个MIME类型就知道如何处理它了。当然是在浏览器中显示这个html文件了。但如果服务器响应的是一个exe文件，那么浏览器就不可能显示它，而是应该弹出下载窗口才对。MIME就是用来说明文档的内容是什么类型的！
* context.xml：对所有应用的统一配置，通常我们不会去配置它。
* lib：Tomcat的类库，里面是一大堆jar文件。如果需要添加Tomcat依赖的jar文件，可以把它放到这个目录中，当然也可以把应用依赖的jar文件放到这个目录中，这个目录中的jar所有项目都可以共享之，但这样你的应用放到其他Tomcat下时就不能再共享这个目录下的Jar包了，所以建议只把Tomcat需要的Jar包放到这个目录下；
* logs：这个目录中都是日志文件，记录了Tomcat启动和关闭的信息，如果启动Tomcat时有错误，那么异常也会记录在日志文件中。
* temp：存放Tomcat的临时文件，这个目录下的东西可以在停止Tomcat后删除！
* webapps：存放web项目的目录，其中每个文件夹都是一个项目；如果这个目录下已经存在了目录，那么都是tomcat自带的。项目。其中ROOT是一个特殊的项目，在地址栏中没有给出项目目录时，对应的就是ROOT项目。<http://localhost:8080/examples，进入示例项目。其中examples>就是项目名，即文件夹的名字。
* work：运行时生成的文件，最终运行的文件都在这里。通过webapps中的项目生成的！可以把这个目录下的内容删除，再次运行时会生再次生成work目录。当客户端用户访问一个JSP文件时，Tomcat会通过JSP生成Java文件，然后再编译Java文件生成class文件，生成的java和class文件都会存放到这个目录下。
* LICENSE：许可证。
* NOTICE：说明文件。

## 三、Web应用

###### 1　创建静态应用

* 在webapps下创建一个hello目录；
* 在webapps\hello\下创建index.html；
* 启动tomcat；
* 打开浏览器访问http://localhost:8080/hello/index.html

index.html

|  |
| --- |
| <html>  <head>  <title>hello</title>  </head>  <body>  <h1>Hello World!</h1>  </body>  </html> |

###### 2　创建动态应用

* 在webapps下创建hello1目录；
* 在webapps\hello1\下创建WEB-INF目录；
* 在webapps\hello1\WEB-INF\下创建web.xml；
* 在webapps\hello1\下创建index.html。
* 打开浏览器访问http://localhost:8080/hello/index.html

web.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <web-app version="2.5"  xmlns="http://java.sun.com/xml/ns/javaee"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee  http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd">  </web-app> |

完整的Web应用还需要在WEB-INF目录下创建：

* classes；
* lib目录；

webapps

|- hello

|-index.html

|-WEB-INF

|-web.xml

|-classes

|-lib

* hello：应用目录，hello就是应用的名称；
* index.html：应用资源。应用下可以有多个资源，例如css、js、html、jsp等，也可以把资源放到文件夹中，例如：hello\html\index.html，这时访问URL为：http://localhost:8080/hello/html/index.html；
* WEB-INF：这个目录名称必须是大写，这个目录下的东西是无法通过浏览器直接访问的，也就是说放到这里的东西是安全的；
* web.xml：应用程序的部署描述符文件，可以在该文件中对应用进行配置，例如配置应用的首页：

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

* classes：存放class文件的目录；
* lib：存放jar包的目录；

###### 3　配置外部应用

也可以把应用放到Tomcat之外，这就是外部应用了。例如我们把上面写的hello应用从webapps目录中剪切到C盘下，即C:/hello。现在hello这个Web应用已经不在Tomcat中了，这时我们需要在tomcat中配置外部应用的位置，配置的方式一共有两种：

* conf/server.xml：打开server.xml文件，找到<Host>元素，在其中添加<Context>元素，代码如下：(了解)

server.xml

|  |
| --- |
|  |

1. path：指定当前应用的名称；
2. docBase：指定应用的物理位置；
3. 浏览器访问路径：<http://localhost:8080/itcast_hello/index.html>。

* conf/catalana/localhost：在该目录下创建itcast\_hello.xml文件，在该文件中编写<Context>元素，代码如下：(了解)

|  |
| --- |
|  |

1. 文件名：指定当前应用的名称；
2. docBase：指定应用的物理位置；
3. 浏览器访问路径：<http://localhost:8080/itcast_hello/index.html>。

###### 4　理解server.xml（了解）

<Server>

<Servier>

<Connector>

<Engine>

<Host>

<Context>

* <Server>：根元素，表示整个服务器的配置信息；
* <Service>：<Server>的子元素，在<Server>中只能有一个<Service>元素，它表示服务；
* <Connector>：<Service>的子元素，在<Service>中可以有N个<Connector>元素，它表示连接。
* <Engine>：<Service>的子元素，在<Service>中只能有一<Engine>元素，该元素表示引擎，它是<Service>组件的核心。
* <Host>：<Engine>的子元素，在<Engine>中可以有N个<Host>元素，每个<Host>元素表示一个虚拟主机。所谓虚拟主机就像是真的主机一样，每个主机都有自己的主机名和项目目录。例如<Host name=”localhost” appBase=”webapps”>表示主机名为localhost，这个主机的项目存放在webapps目录中。访问这个项目下的主机时，需要使用localhost主机名，项目都存放在webapps目录下。
* <Context>：<Host>元素的子元素，在<Host>中可以有N个<Context>元素，每个<Context>元素表示一个应用。如果应用在<Host>的appBase指定的目录下，那么可以不配置<Context>元素，如果是外部应用，那么就必须配置<Context>。如果要为应用指定资源，也需要配置<Context>元素。

我们可以把<Server>看作是一个大酒店：

* <Service>：酒店的服务部门；
* <Connector>：服务员；
* <Engine>：后厨；
* <Host>：后厨中的一个区，例如川菜区是一个<Host>、粤菜区是一个<Host>；
* <Context>：后厨的一个厨师。

用户发出一个请求：http://localhost:8080/hello/index.jsp。发现是http/1.1协议，而且还是8080端口，所以就交给了处理这一请求的“服务员（处理HTTP请求的<Connector>）”，“服务员”再把请求交给了“后厨（<Engine>）”，因为请求是要一盘水煮鱼，所以由“川菜区（<Host>）”负责，因为“大老王师傅<Context>”做水煮鱼最地道，所以由它完成。

* <Connector>：关心请求中的http、和8080；
* < Host>：关心localhost；
* <Context>：关心hello

###### 5　映射虚拟主机(了解)

我们的目标是，在浏览器中输出：<http://www.itcast.cn>就可以访问我们的项目。

完成这一目标，我们需要做三件事：

* 修改端口号为80，这一点应该没有问题吧；
* 在本机上可以解析域名为127.0.0.1，这需要修改C:\WINDOWS\system32\drivers\etc\hosts文件，添加对<http://www.itcast.cn>和127.0.01的绑定关系；
* 在server.xml文件中添加一个<Host>（主机）。

1. 修改端口号为80



1. 绑定<http://www.itcast.cn>和127.0.01的绑定关系



1. server.xml文件中添加一个<Host>



* name=”www.itcast.cn”：指定虚拟主机名为[www.itcast.cn](http://www.itcast.cn)；
* appBase=”F:/itcastapps”：指定当前虚拟主机的应用程序存放目录为F:/itcastapps。
* 在itcastapps目录下创建名为ROOT的应用，因为一个主机只可以有一个名为ROOT的应用，名为ROOT的应用在浏览器中访问是可以不给出应用名称。



现在访问：<http://www.itcast.cn>看看是什么页面！

请注意，只有本机可以通过<http://www.itcast.cn>来访问，而其他电脑不可以！

###### 6　MyEclipse创建JavaWeb应用

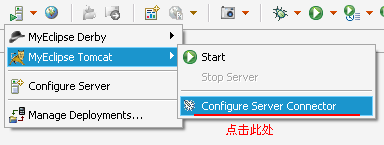
6.1　配置Tomcat

使用MyEclipse配置服务器后，就可以使用MyEclipse来启动和停止服务器了。当然，你需要先安装好服务器（Tomcat），才能配置。

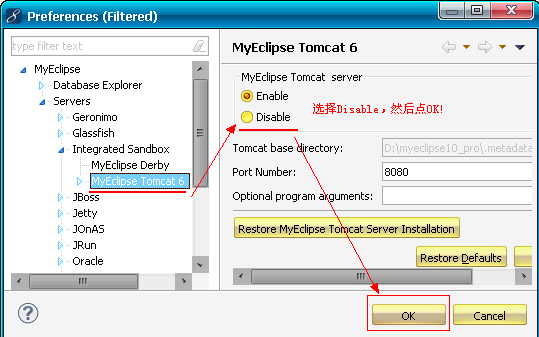
MyEclipse自带了一个Tomcat，强烈建议不要使用它。所以，我们需要先把MyEclipse自带的Tomcat关闭，然后再来配置我们自己的Tomcat。

* 关闭MyEclipse自带Tomcat。

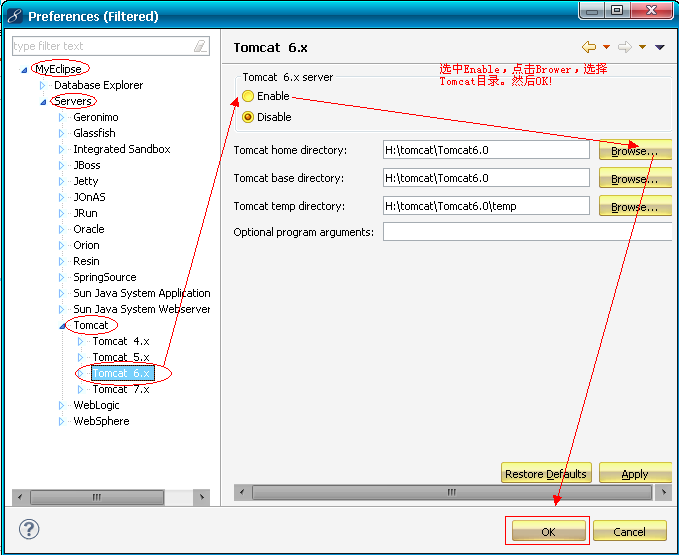
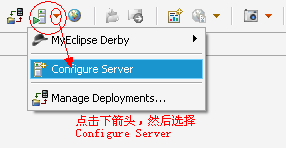
在工具栏中找到，点击下箭头，点击Configure Server Connector。



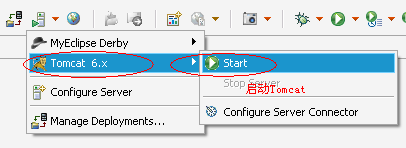
弹出对话框



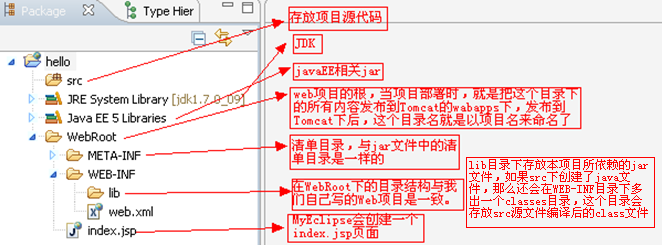
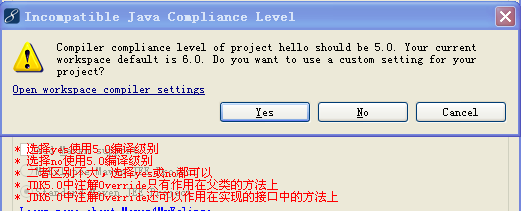
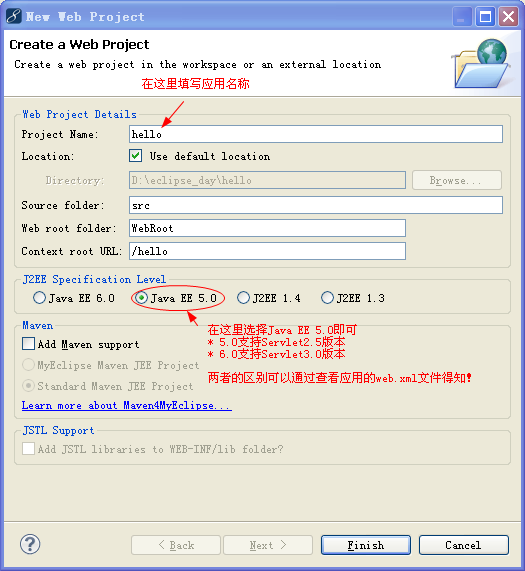
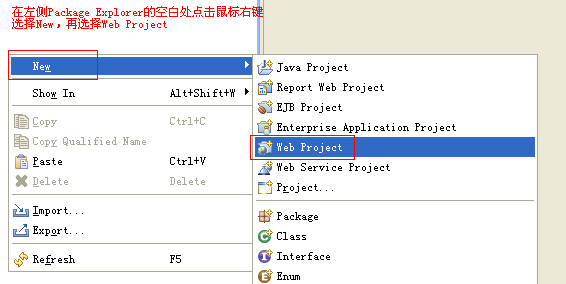
* 配置我们自己的Tomcat



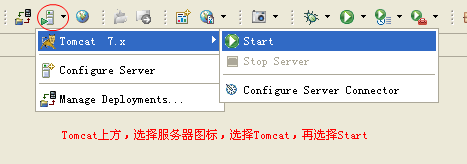
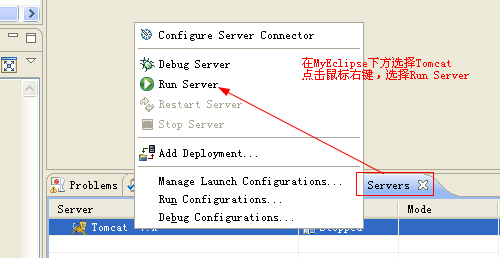
使用MyEclipse启动Tomcat



6.2　创建JavaWeb应用



6.3　启动Tomcat

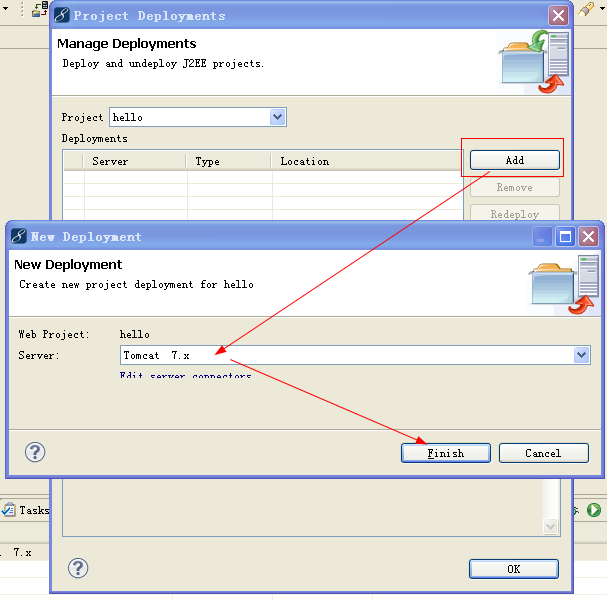
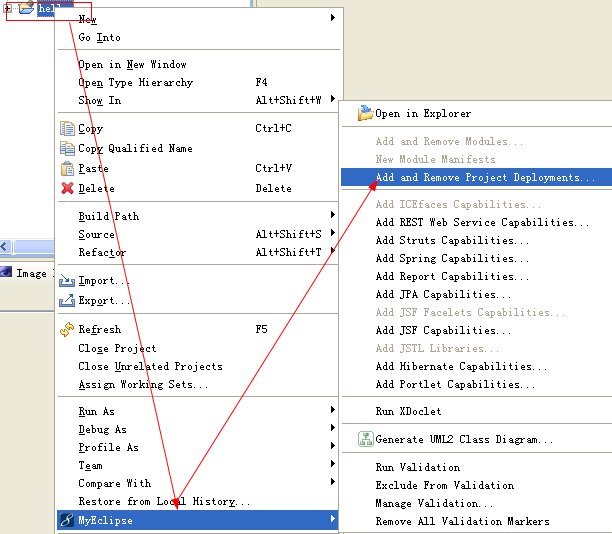


　　上面两种方式都可以启动tomcat

6.4　关闭tomcat

　　与启动Tomcat相同位置下方就是Stop Server，即可关闭tomcat了。

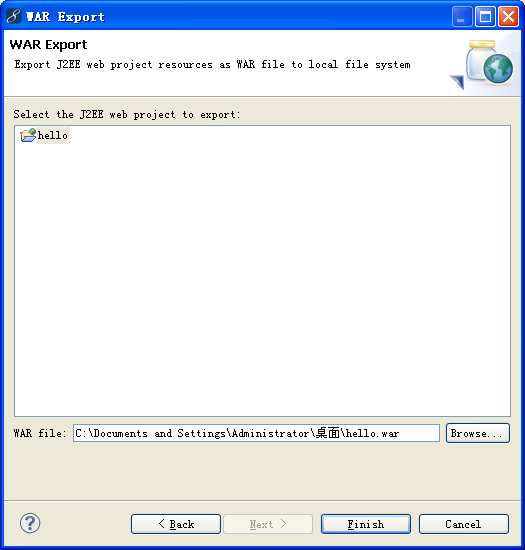
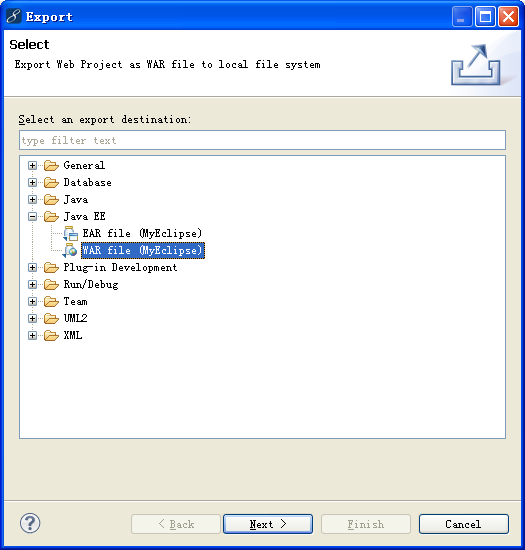
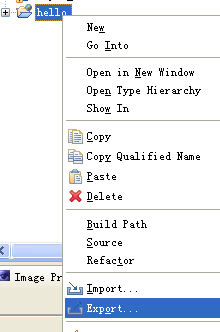
6.5　发布项目到tomcat的webapps目录



　　项目发布后，就是把项目的WebRoot目录copy到Tomcat的webapps目录，并把WebRoot重命名为项目名称，即hello。所以在Tomcat的webapps下会多出一个文件夹hello。

6.6　打war包

JavaSE程序可以打包成Jar包，而JavaWeb程序可以打包成war包。然后把war发布到Tomcat的webapps目录下，Tomcat会在启动时自动解压war包。



## 四、HTTP协议

###### 1　安装HttpWatch

HttpWatch是专门为IE浏览器提供的，用来查看HTTP请求和响应内容的工具。而FireFox上需要安装FireBug软件。如果你使用的是Chrome，那么就不用自行安装什么工具了，因为它自身就有查看请求和响应内容的功能！

HttpWatch和FireBug这些工具对浏览器而言不是必须的，但对我们开发者是很有帮助的，通过查看HTTP请求响应内容，可以使我们更好的学习HTTP协议。

###### 2　HTTP概述

HTTP（hypertext transport protocol），即超文本传输协议。这个协议详细规定了浏览器和万维网服务器之间互相通信的规则。

HTTP就是一个通信规则，通信规则规定了客户端发送给服务器的内容格式，也规定了服务器发送给客户端的内容格式。其实我们要学习的就是这个两个格式！客户端发送给服务器的格式叫“请求协议”；服务器发送给客户端的格式叫“响应协议”。

###### 3　 URL和URI

URL：统一资源定位符，就是一个网址，例如：<http://www.itcast.cn就是一个URL>。/hello/index.html也是一个URL，URL必须是一个真实存在的网址。

URI：统一资源标识符：比URI包含了URL，URI的范围更加宽泛，URI可以是一个不存在的网址。在网络上用来标签资源的都是URI，例如zhangSan@163.com也是URI。

###### 4　请求协议

请求协议的格式如下：

|  |
| --- |
| 请求首行；请求方式 请求路径 请求协议/请求版本. Get <http://xxx.com> HTTP/1.1  请求头信息；键值对  空行； 分隔符.请求体和请求头  请求体。 |

　　浏览器发送给服务器的内容就这个格式的，如果不是这个格式服务器将无法解读！在HTTP协议中，请求有很多请求方法，其中最为常用的就是GET和POST。不同的请求方法之间的区别，后面会一点一点的介绍。

4.1　GET请求

　　打开IE，在访问hello项目的index.jsp之间打开HttpWatch，并点击“Record”按钮。然后访问index.jsp页面。查看请求内容如下：

|  |
| --- |
| GET /hello/index.jsp HTTP/1.1  Host: localhost  User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:5.0) Gecko/20100101 Firefox/5.0  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8  Accept-Language: zh-cn,zh;q=0.5  Accept-Encoding: gzip, deflate  Accept-Charset: GB2312,utf-8;q=0.7,\*;q=0.7  Connection: keep-alive  Cookie: JSESSIONID=369766FDF6220F7803433C0B2DE36D98 |

* GET /hello/index.jsp HTTP/1.1：GET请求，请求服务器路径为/hello/index.jsp，协议为1.1；
* \*Host:localhost：请求的主机名为localhost；
* \*User-Agent: Mozilla/5.0 (**Windows NT 5.1**; rv:5.0) Gecko/20100101 **Firefox/5.0**：与浏览器和OS相关的信息。有些网站会显示用户的系统版本和浏览器版本信息，这都是通过获取User-Agent头信息而来的；
* Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8：告诉服务器，当前客户端可以接收的文档类型，其实这里包含了\*/\*，就表示什么都可以接收；
* Accept-Language: zh-cn,zh;q=0.5：当前客户端支持的语言，可以在浏览器的工具🡪选项中找到语言相关信息；
* Accept-Encoding: gzip, deflate：支持的压缩格式。数据在网络上传递时，可能服务器会把数据压缩后再发送；
* Accept-Charset: GB2312,utf-8;q=0.7,\*;q=0.7：客户端支持的编码；
* Connection: keep-alive：客户端支持的链接方式，保持一段时间链接，默认为3000ms；
* \*Cookie: JSESSIONID=369766FDF6220F7803433C0B2DE36D98：因为不是第一次访问这个地址，所以会在请求中把上一次服务器响应中发送过来的Cookie在请求中一并发送去过；这个Cookie的名字为JSESSIONID，然后在讲会话是讲究它！

4.2　POST请求

为了演示POST请求，我们需要修改index.jsp页面，即添加一个表单：

|  |
| --- |
| <form action=*""* method=*"post"*>  关键字：<input type=*"text"* name=*"keyword"*/>  <input type=*"submit"* value=*"提交"*/>  </form> |



打开HttpWatch，输入hello后点击提交，查看请求内容如下：

|  |
| --- |
| POST /hello/index.jsp HTTP/1.1  Accept: image/gif, image/jpeg, image/pjpeg, image/pjpeg, application/msword, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/x-ms-application, application/x-ms-xbap, application/vnd.ms-xpsdocument, application/xaml+xml, \*/\*  Referer: http://localhost:8080/hello/index.jsp  Accept-Language: zh-cn,en-US;q=0.5  User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0; InfoPath.2; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)  Content-Type: application/x-www-form-urlencoded  Accept-Encoding: gzip, deflate  Host: localhost:8080  Content-Length: 13  Connection: Keep-Alive  Cache-Control: no-cache  Cookie: JSESSIONID=E365D980343B9307023A1D271CC48E7D  keyword=hello |

POST请求是可以有体的，而GET请求有请求体(请求体是空白)。

* **Referer**: http://localhost:8080/hello/index.jsp：请求来自哪个页面，例如你在百度上点击链接到了这里，那么Referer:http://www.baidu.com；如果你是在浏览器的地址栏中直接输入的地址，那么就没有Referer这个请求头了；
* Content-Type: application/x-www-form-urlencoded：表单的数据类型，说明会使用url格式编码数据；url编码的数据都是以“%”为前缀，后面跟随两位的16进制，例如“传智”这两个字使用UTF-8的url编码用为“%E4%BC%A0%E6%99%BA”；
* Content-Length:13：请求体的长度，这里表示13个字节。
* keyword=hello：请求体内容！hello是在表单中输入的数据，keyword是表单字段的名字。

Referer请求头是比较有用的一个请求头，它可以用来做统计工作，也可以用来做防盗链。

**统计工作**：我公司网站在百度上做了广告，但不知道在百度上做广告对我们网站的访问量是否有影响，那么可以对每个请求中的Referer进行分析，如果Referer为百度的很多，那么说明用户都是通过百度找到我们公司网站的。

**防盗链**：我公司网站上有一个下载链接，而其他网站盗链了这个地址，例如在我网站上的index.html页面中有一个链接，点击即可下载JDK7.0，但有某个人的微博中盗链了这个资源，它也有一个链接指向我们网站的JDK7.0，也就是说登录它的微博，点击链接就可以从我网站上下载JDK7.0，这导致我们网站的广告没有看，但下载的却是我网站的资源。这时可以使用Referer进行防盗链，在资源被下载之前，我们对Referer进行判断，如果请求来自本网站，那么允许下载，如果非本网站，先跳转到本网站看广告，然后再允许下载。

###### 5　响应协议

5.1　响应内容

响应协议的格式如下：

|  |
| --- |
| 响应首行  响应头信息；  空行；  响应体。 |

响应内容是由服务器发送给浏览器的内容，浏览器会根据响应内容来显示。

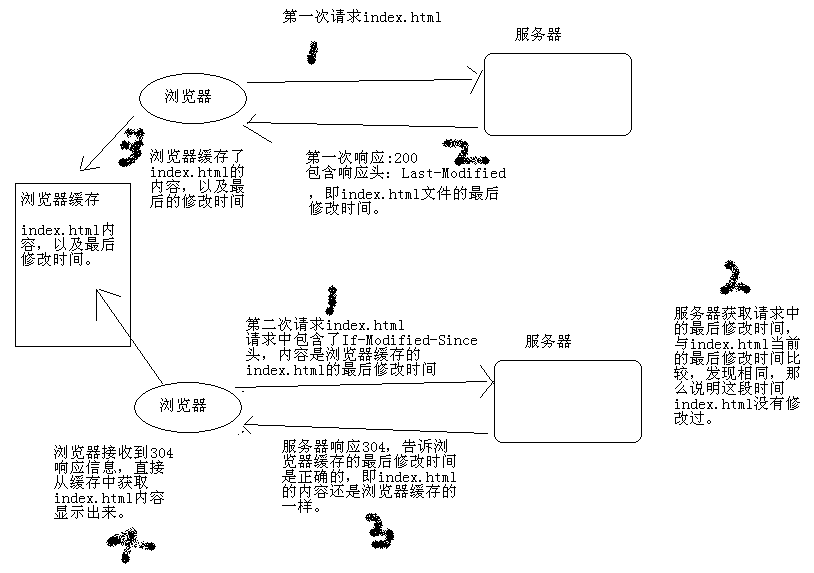
|  |
| --- |
| HTTP/1.1 200 OK  Server: Apache-Coyote/1.1  Content-Type: text/html;charset=UTF-8  Content-Length: 724  Set-Cookie: JSESSIONID=C97E2B4C55553EAB46079A4F263435A4; Path=/hello  Date: Wed, 25 Sep 2012 04:15:03 GMT  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <base href="http://localhost:8080/hello/">    <title>My JSP 'index.jsp' starting page</title>  <meta http-equiv="pragma" content="no-cache">  <meta http-equiv="cache-control" content="no-cache">  <meta http-equiv="expires" content="0">  <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">  <meta http-equiv="description" content="This is my page">  <!--  <link rel="stylesheet" type="text/css" href="styles.css">  -->  </head>    <body>  <form action="" method="post">  关键字：<input type="text" name="keyword"/>  <input type="submit" value="提交"/>  </form>  </body>  </html> |

* HTTP/1.1 200 OK：响应协议为HTTP1.1，状态码为200，表示请求成功，OK是对状态码的解释；
* Server: Apache-Coyote/1.1：服务器的版本信息；
* \* Content-Type: text/html;charset=UTF-8：响应体使用的编码为UTF-8；
* Content-Length: 724：响应体为724字节；
* Set-Cookie: JSESSIONID=C97E2B4C55553EAB46079A4F263435A4; Path=/hello：响应给客户端的Cookie；
* Date: Wed, 25 Sep 2012 04:15:03 GMT：响应的时间，这可能会有8小时的时区差；

5.2　状态码

响应头对浏览器来说很重要，它说明了响应的真正含义。例如200表示响应成功了，302表示重定向，这说明浏览器需要再发一个新的请求。

* 200：请求成功，浏览器会把响应体内容（通常是html）显示在浏览器中；
* 404：请求的资源没有找到，说明客户端错误的请求了不存在的资源；
* 500：请求资源找到了，但服务器内部出现了错误；
* 302：重定向，当响应码为302时，表示服务器要求浏览器重新再发一个请求，服务器会发送一个响应头Location，它指定了新请求的URL地址；
* 304：当用户第一次请求index.html时，服务器会添加一个名为Last-Modified响应头，这个头说明了index.html的最后修改时间，浏览器会把index.html内容，以及最后响应时间缓存下来。当用户第二次请求index.html时，在请求中包含一个名为If-Modified-Since请求头，它的值就是第一次请求时服务器通过Last-Modified响应头发送给浏览器的值，即index.html最后的修改时间，If-Modified-Since请求头就是在告诉服务器，我这里浏览器缓存的index.html最后修改时间是这个，您看看现在的index.html最后修改时间是不是这个，如果还是，那么您就不用再响应这个index.html内容了，我会把缓存的内容直接显示出来。而服务器端会获取If-Modified-Since值，与index.html的当前最后修改时间比对，如果相同，服务器会发响应码304，表示index.html与浏览器上次缓存的相同，无需再次发送，浏览器可以显示自己的缓存页面，如果比对不同，那么说明index.html已经做了修改，服务器会响应200。



5.3　其他响应头

告诉浏览器不要缓存的响应头：

* Expires: -1；
* Cache-Control: no-cache；
* Pragma: no-cache；

自动刷新响应头，浏览器会在3秒之后请求http://www.itcast.cn：

* Refresh: 3;url=http://www.itcast.cn

5.4　HTML中指定响应头

在HTMl页面中可以使用<meta http-equiv="" content="">来指定响应头，例如在index.html页面中给出<meta http-equiv="Refresh" content="3;url=http://www.itcast.cn">，表示浏览器只会显示index.html页面3秒，然后自动跳转到http://www.itcast.cn。

## 五、Session共享

### 一、抛出Session一致性问题

1、session复制

tomcat 本身带有复制session的功能。（不讲）

2、共享session

需要专门管理session的软件，memcached 缓存服务，可以和tomcat整合，帮助tomcat共享管理session。需要第三方提供的缓存数据库。如memcached,redis。

### 二、利用memcached缓存session

1.安装memcached:yum –y install memcached

2.安装telnet（通过telnet测试memcached数据库）：yum –y install telnet

3.启动：service memcached start

4.连接：telnet 127.0.0.1 11211

5.测试数据：

输入>> set demo 0 0 5

输入>> 111

输入>>get demo

6.将jar包拷贝到tomcat的lib下

7.在tomcat的server.xml中配置jvmRoute

<Engine name="Catalina" defaultHost="localhost" jvmRoute="tomcat2">

8.在context.xml的Context配置Manager

<Manager className="de.javakaffee.web.msm.MemcachedBackupSessionManager"

memcachedNodes="n1:192.168.17.4:11211"

sticky="false"

lockingMode="auto"

sessionBackupAsync="false"

requestUriIgnorePattern=".\*\.(ico|png|gif|jpg|css|js)$"

sessionBackupTimeout="1000" transcoderFactoryClass="de.javakaffee.web.msm.serializer.kryo.KryoTranscoderFactory"

/>

9.index.jsp配置

<%@page language="java" contentType="text/html;charset=UTF-8" pageEncoding="UTF-8"%>

SessionID:<%=session.getId()%>

<br/>

SessionIP:<%=request.getServerName()%>

<br/>

<h1>tomcat1</h1>

### 三、利用tomcat-redis-session-manager实现session共享

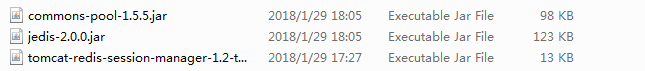
1. 安装redis缓存数据库，修改配置文件vi /etc/redis.conf

2. 将bind的127.0.0.1修改为本机地址，否则只能本机访问了



1. 导入redislib中三个jar包到tomcat中，

Tomcat-redis-session下载：<https://github.com/jcoleman/tomcat-redis-session-manager/downloads>



1. 在tomcat添加配置

单点配置：

<Valve className="com.radiadesign.catalina.session.RedisSessionHandlerValve" />

<Manager className="com.radiadesign.catalina.session.RedisSessionManager"

host="127.0.0.1"

port="6379"

database="0"

maxInactiveInterval="60" />

集群配置：

<Valve className="com.radiadesign.catalina.session.RedisSessionHandlerValve" />

<Manager className="com.radiadesign.catalina.session.RedisSessionManager"

sentinelMaster="MySentinelMaster" <!-- 可选-->

sentinels="192.168.1.1:8080,192.168.1.2:8081,.." <!-- 可选 -->

/>

4.index.jsp配置

<%@page language="java" contentType="text/html;charset=UTF-8" pageEncoding="UTF-8"%>

SessionID:<%=session.getId()%>

<br/>

SessionIP:<%=request.getServerName()%>

<br/>

<h1>tomcat1</h1>

### 四、利用spring-session-data-redis 实现session共享

1.添加依赖

<dependencies>

<dependency><!-- spring-session-data-redis -->

<groupId>org.springframework.session</groupId>

<artifactId>spring-session-data-redis</artifactId>

<version>1.2.1.RELEASE</version>

</dependency>

<dependency><!-- redis client -->

<groupId>redis.clients</groupId>

<artifactId>jedis</artifactId>

<version>2.8.1</version>

</dependency>

<dependency><!-- web依赖，为了启动项目 -->

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>4.2.5.RELEASE</version>

</dependency>

</dependencies>

1. Spring 配置

<!-- 需要开启注解，因为session.data.redis大量使用了注解的方式 -->

<context:annotation-config/>

<bean id=*"redisHttpSessionConfiguration"* class=*"org.springframework.session.data.redis.config.annotation.web.http.RedisHttpSessionConfiguration"*>

<property name=*"maxInactiveIntervalInSeconds"* value=*"600"* />

</bean>

<bean id=*"jedisPoolConfig"* class=*"redis.clients.jedis.JedisPoolConfig"*>

<property name=*"maxTotal"* value=*"100"* />

<property name=*"maxIdle"* value=*"10"* />

</bean>

<bean id=*"jedisConnectionFactory"* class=*"org.springframework.data.redis.connection.jedis.JedisConnectionFactory"*

destroy-method=*"destroy"*>

<property name=*"hostName"* value=*"127.0.0.1"* />

<property name=*"port"* value=*"6379"* />

<property name=*"password"* value=*""* />

<property name=*"timeout"* value=*"100"* />

<property name=*"poolConfig"* ref=*"jedisPoolConfig"* />

</bean>

1. web.xml 配置

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/classes/applicationContext.xml</param-value>

</context-param>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<!-- 就是这个filter -->

<filter>

<filter-name>springSessionRepositoryFilter</filter-name>

<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>

</filter>

<filter-mapping>

<filter-name>springSessionRepositoryFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

1. 创建页面

<%@page language=*"java"* contentType=*"text/html;charset=UTF-8"* pageEncoding=*"UTF-8"*%>

SessionID:<%=session.getId()%>

<br/>

SessionIP:<%=request.getServerName()%>

<br/>

<h1>tomcat1</h1>

第二页tomcat修改为tomcat2

启动后分别访问SessionId和SessionIp应该是相同的。

这个地方最好使用Ngnix做负载均衡（采用循环模式）

### 五、综上

tomcat-redis-session-manager目前只支持到tomcat7-java7，需要对tomcat进行维护，移植性不高；

Memcache数据类型少，缓存上限低，不推荐。

Spring-session-data-redis，对代码无侵入，且需要简单的配置，即可实现。

## Tomcat服务器优化

Tomcat默认的配置，尤其是内存和线程的配置，其配置都很低，容易成为性能瓶颈，所以我们需要对Tomcat服务器进行优化，提升其运行性能

### 内存优化（JVM参数优化）

说是内存的优化，其实内存知识JVM运行参数中的的一部分。

Windows 下的 catalina.bat

Linux 下的 catalina.sh

下面就是内存的设置了

JAVA\_OPTS="-server -Xms4096M -Xmx4096M -Xmn512M"

我们甚至可以设置更多的运行参数，下面是设置多类参数，最后共同组成Tomcat参数

DATE=$(date +%Y%m%d%H%M%S) #当前时间

# JVM memory options

JAVA\_OPT\_MEM="

-server -Xms4096M -Xmx4096M -Xmn512M -XX:MetaspaceSize=512M -XX:MaxMetaspaceSize=512M

-Xverify:none

-XX:SurvivorRatio=8

-XX:MaxTenuringThreshold=7

-XX:GCTimeRatio=19

-XX:+DisableExplicitGC

-XX:+UseParNewGC

-XX:+UseConcMarkSweepGC

-XX:+CMSPermGenSweepingEnabled

-XX:+UseCMSCompactAtFullCollection

-XX:CMSFullGCsBeforeCompaction=0

-XX:+CMSClassUnloadingEnabled

-XX:-CMSParallelRemarkEnabled

-XX:CMSInitiatingOccupancyFraction=70"

# JVM gc log file

JAVA\_OPT\_LOG="

-Xloggc:/usr/local/{{ projectname }}/logs/{{ projectname }}.gc.$DATE.log"

# JVM gc options

JAVA\_OPT\_GC="

-XX:+PrintGCDetails

-XX:+PrintGCTimeStamps

-XX:+PrintHeapAtGC

-XX:+PrintGCApplicationStoppedTime "

JAVA\_OPT\_HS\_ERR="-XX:CompileCommandFile=.hotspot\_compiler"

# Tomcat JVM参数

JAVA\_OPTS="$JAVA\_OPT\_MEM $JAVA\_OPT\_GC $JAVA\_OPT\_LOG $JAVA\_OPT\_HS\_ERR"

### **Tomcat Exectuor(线程池)优化**

打开tomcat的serve.xml，配置Executor，相关参数说明如下

Executor 指的是 Connector（连接器）使用的executor（执行器），也就是线程池

<Executor

name="tomcatThreadPool"

namePrefix="catalina-exec-"

maxThreads="150"

minSpareThreads="4"

maxIdLeTime = "60000"

/>

* name：给执行器（线程池）起一个名字；
* namePrefix：指定线程池中的每一个线程的 name 前缀；
* maxThreads：线程池中最大的线程数量，假设请求的数量超过了 750，这将不是意味着将 maxThreads 属性值设置为 750，它的最好解决方案是使用「Tomcat集群」。也就是说，如果有 1000 请求，两个 Tomcat 实例设置 maxThreads = 500，而不在单 Tomcat 实例的情况下设置 maxThreads=1000。
* minSpareThreads：线程池中允许空闲的线程数量（多余的线程都杀死）；
* maxIdLeTime：一个线程空闲多久算是一个空闲线程；

### **Tomcat Connector(连接器)优化**

在server.xml中配置

比如:

<Connector

port="80"

protocol="HTTP/1.1"

maxThreads="600"

maxSpareThreads="500"

minSpareThreads="100"

acceptCount="700"

connectionTimeout="20000"

/>

1、URIEncoding=“UTF-8”：设置Tomcat的字符集。

这种配置我们一般是不会设置的，因为关于乱码的转换我们会在具体项目中具体处理，直接修改Tomcat的字符集未免过于太死板。

2、maxThreads=“300”：设置当前Tomcat的最大并发数。

Tomcat默认配置的最大请求数是150个，即同时能支持150个并发。

但是在实际运用中，最大并发数与硬件性能和CPU数量都有很大关系的，更好的硬件、更高的处理器都会使Tomcat支持更多的并发数。

一般在实际开发中，当某个应用拥有 250 个以上并发的时候，都会考虑到应用服务器的集群。

3、minSpareThreads=“50”：设置当前Tomcat初始化时创建的线程数，默认值为25。

4、maxSpareThreads="X" 表示如果最多可以有X个线程，一旦超过X个,则会关闭不在需要的线程

5、acceptCount=“250”：当同时连接的人数达到maxThreads参数设置的值时，还可以接收排队的连接数量，超过这个连接的则直接返回拒绝连接。指定当任何能够使用的处理请求的线程数都被使用时，能够放到处理队列中的请求数，超过这个数的请求将不予处理。默认值为100。

在实际应用中，如果想加大Tomcat的并发数 ，应该同时加大acceptCount和maxThreads的值。

5、enableLookups=“false”：是否开启域名反查，一般设置为false来提高处理能力，它的取值还有true，一般很少使用。

6、maxKeepAliveRequests=“1”：nginx动态的转给tomcat，nginx是不能keepalive的，而tomcat端默认开启了keepalive，会等待keepalive的timeout，默认不设置就是使用connectionTimeout。

所以必须设置tomcat的超时时间，并关闭tomcat的keepalive。否则会产生大量tomcat的socket timewait。

maxKeepAliveRequests=”1”就可以避免tomcat产生大量的TIME\_WAIT连接，从而从一定程度上避免tomcat假死。

### **Tomcat IO 优化**

1：同步阻塞IO(JAVA BIO) 同步并阻塞，服务器实现模式为一个连接一个线程(one connection one thread 想想都觉得恐怖，线程可是非常宝贵的资源)，当然可以通过线程池机制改善.

2：JAVA NIO 又分为同步非阻塞IO，异步阻塞IO与BIO最大的区别one request one thread.可以复用同一个线程处理多个connection(多路复用).

3：异步非阻塞IO(Java NIO2又叫AIO) 主要与NIO的区别主要是操作系统的底层区别，可以做个比喻：比作快递，NIO就是网购后要自己到官网查下快递是否已经到了(可能是多次)，然后自己去取快递;AIO就是快递员送货上门了(不用关注快递进度)。

BIO方式适用于连接数目比较小且固定的架构，这种方式对服务器资源要求比较高，并发局限于应用中，JDK1.4以前的唯一选择，但程序直观简单易理解.Tomcat8 以下版本，默认使用的就是 BIO「阻塞式IO)」模式。

NIO方式适用于连接数目多且连接比较短(轻操作)的架构，比如聊天服务器，并发局限于应用中，编程比较复杂，JDK1.4开始支持.Tomcat8 以上版本，默认使用的就是NIO模式「非阻塞式 IO」。

AIO方式使用于连接数目多且连接比较长(重操作)的架构，比如相册服务器，充分调用OS参与并发操作，编程比较复杂，JDK7开始支持.

在server.xml中

<Connector port="80" protocol="org.apache.coyote.http11.Http11NioProtocol"

connectionTimeout="20000"

URIEncoding="UTF-8"

useBodyEncodingForURI="true"

enableLookups="false"

redirectPort="8443" />

实现对Tomcat的IO切换。

### **禁用 AJP 连接器**

AJP的全称 Apache JServer Protocol，使用 Nginx+Tomca t的架构，所以用不着 AJP 协议，所以把AJP连接器禁用。

找到这个，并将其注释掉即可。

<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />

### **大杀器APR**

全称 Apache Portable Runtime，是Tomcat生产环境运行的首选方式，如果操作系统未安装 APR 或者 APR 路径未指到 Tomcat 默认可识别的路径，则 APR 模式无法启动，自动切换启动 NIO 模式。所以必须要安装 APR 和 Native，直接启动就支持 APR，APR是从操作系统级别解决异步 IO 问题，APR 的本质就是使用 JNI 技术调用操作系统底层的 IO 接口，所以需要提前安装所需要的依赖

APR是从操作系统级别来解决异步的IO问题，大幅度的提高性能. (http://apr.apache.org/)。

APR(Apache Portable Runtime)是一个高可移植库，它是Apache HTTP Server 2.x 的核心，能更好地和其它本地web技术集成，总体上让Java更有效率作为一个高性能web服务器平台而不是简单作为后台容器;

在产品环境中，特别是直接使用Tomcat做WEB服务器的时候，应该使用Tomcat Native来提高其性能，如果不配APR，基本上300个线程狠快就会用满，以后的请求就只好等待.但是配上APR之后，并发的线程数量明显下降，从原来的300可能会马上下降到只有几十，新的请求会毫无阻塞的进来;

在局域网环境测，就算是400个并发，也是一瞬间就处理/传输完毕，但是在真实的Internet环境下，页面处理时间只占0.1%都不到，绝大部分时间都用来页面传输，如果不用APR，一个线程同一时间只能处理一个用户，势必会造成阻塞，所以生产环境下用apr是非常必要的.

### 开启gzip

gzip是http协议中使用的一种加密算法,客户端向web服务器端发出了请求后，通常情况下服务器端会将页面文件和其他资源，返回到客户

端，客户端加载后渲染呈现，这种情况文件一般都比较大，如果开启Gzip ，那么服务器端响应后，会将页面，JS,CSS等文本文件或者其他文件

通过高压缩算法将其压缩，然后传输到客户端，由客户端的浏览器负责解压缩与呈现。通常能节省40%以上的流量（一般都有60%左右），一些

PHP，JSP文件也能够进行压缩。

找到Tomcat 目录下的conf下的server.xml，并找到如下信息

<Connector port="8080" maxHttpHeaderSize="8192" maxThreads="150" minSpareThreads="25" maxSpareThreads="75"

enableLookups="false" redirectPort="8443" acceptCount="100" connectionTimeout="20000"

disableUploadTimeout="true">

将它改成如下的形式（其实在上面代码的下面已经有了，将他们打开而已。）：

<Connector connectionTimeout="20000" protocol="HTTP/1.1" port="8080" maxHttpHeaderSize="8192" maxThreads="150"

minSpareThreads="25" maxSpareThreads="75" enableLookups="false" redirectPort="8443" acceptCount="100"

disableUploadTimeout="true" compression="on" compressionMinSize="2048" noCompressionUserAgents="gozilla,

traviata" compressableMimeType="text/html,text/xml" />

这样，就能够对html和xml进行压缩了，如果要压缩css 和 js，那么需要将

<Connector port="8080" ......... compressableMimeType="text/html,text/xml,text/css,text/javascript" >

你甚至可以压缩图片：

<Connector port="8080" .........

compressableMimeType="text/html,text/xml,text/css,text/javascript,image/gif,image/jpg" >

开启后重启Tomcat ，通过浏览器查看headers信息就能看到是否开启（firebug中有），

如果开启了，那么transfer-encoding就会是Gzip，否则就是chunked。

### **Tomcat的UTF-8编码**

tomcat利用request.setCharacterEncoding("UTF-8");来设置Tomcat接收请求的编码格式，只对POST方式提交的数据有效，对GET方式提交的数据无效!

要设置GET的编码，可以修改server.xml文件中，相应的端口的Connector的属性：URIEncoding="UTF-8"，这样，GET方式提交的数据才会被正确解码。

<Connector port="8080" protocol="HTTP/1.1"

connectionTimeout="20000"

redirectPort="8443"

URIEncoding="UTF-8" />

### **tomcat启动异常**

当tomcat启动时出现闪退，或出现异常

The JRE\_HOME environment variable is not defined correctl

This environment variable is needed to run this program

的时候。解决方法

1.在window下，因为start.bat执行的时候，先执行Catalina.bat文件加载tomcat相关配置，Catalina.bat中也会执行

setclasspath.bat加载java\_home,jre\_home等。所以在setclasspath.bat中最上边增加两行

set JAVA\_HOME=C:\Program Files (x86)\Java\jdk1.7.0\_13\

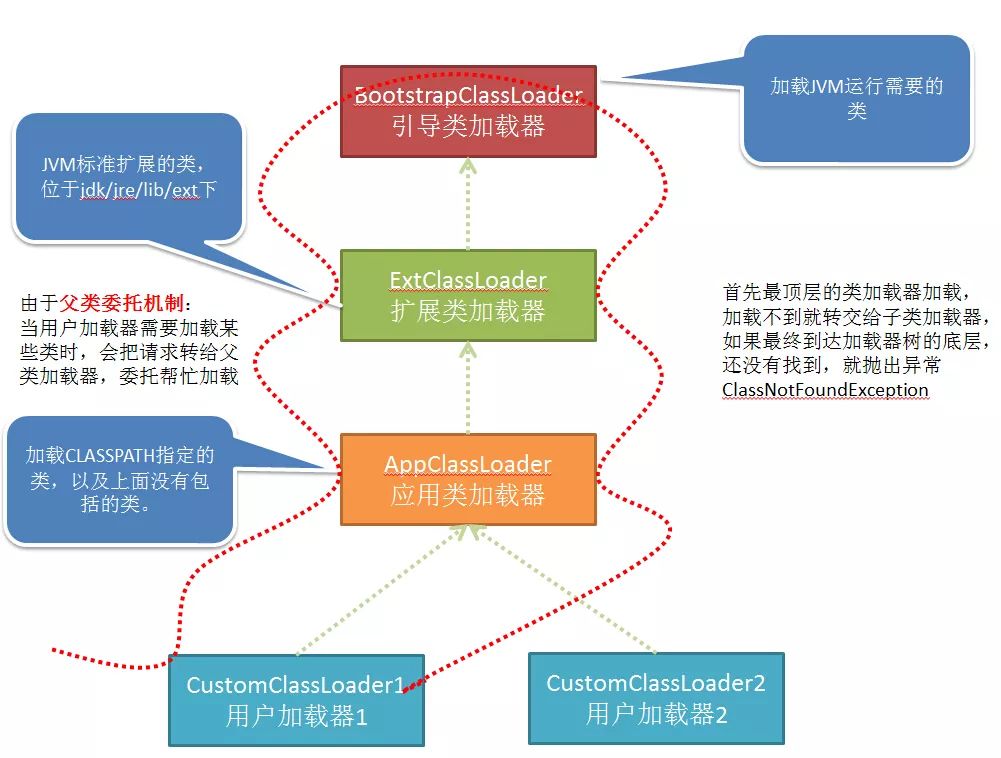
set JRE\_HOME=C:\Program Files (x86)\Java\jdk1.7.0\_13\

即可在cmd中执行start.bat启动tomcat。

# 七、类加载过程

## 1.JVM类加载

JVM类加载采用:父类委托机制，如下图所示：



JVM中包括集中类加载器：

* BootStrapClassLoader 引导类加载器
* ExtClassLoader 扩展类加载器
* AppClassLoader 应用类加载器
* CustomClassLoader 用户自定义类加载器

他们的区别上面也都有说明。需要注意的是，不同的类加载器加载的类是不同的，因此如果用户加载器1加载的某个类，其他用户并不能够使用

当JVM运行过程中，用户需要加载某些类时，会按照下面的步骤（父类委托机制）：

用户自己的类加载器，把加载请求传给父加载器，父加载器再传给其父加载器，一直到加载器树的顶层。

最顶层的类加载器首先针对其特定的位置加载，如果加载不到就转交给子类。

如果一直到底层的类加载都没有加载到，那么就会抛出异常ClassNotFoundException。

因此，按照这个过程可以想到，如果同样在CLASSPATH指定的目录中和自己工作目录中存放相同的class，会优先加载CLASSPATH目录中的文件。

## 2.Tomcat类加载

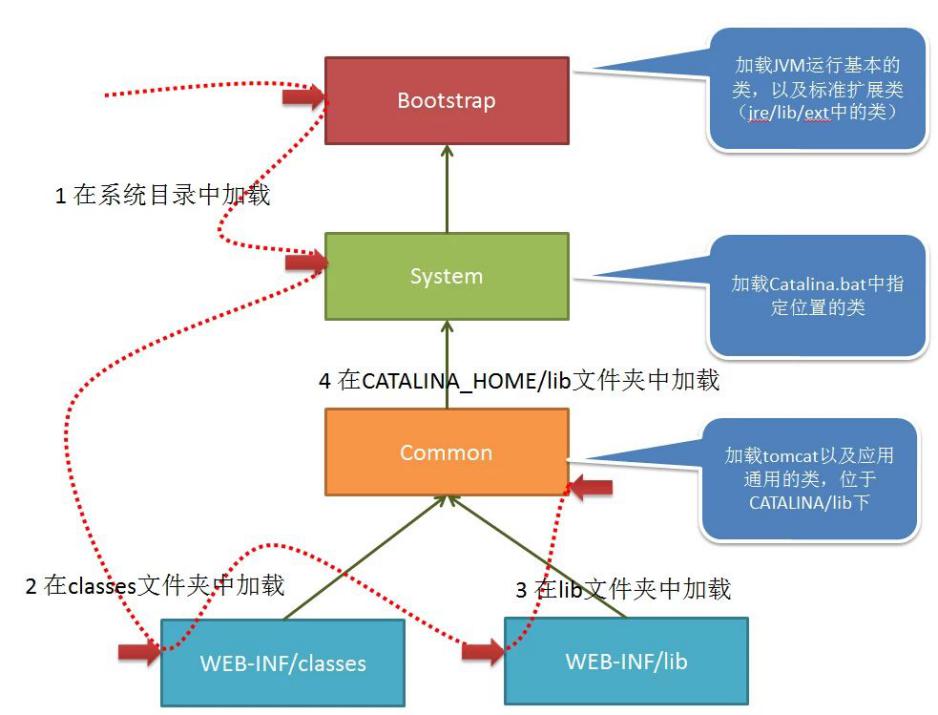
[图解Tomcat类加载机制(阿里面试题)](https://www.cnblogs.com/aspirant/p/8991830.html)：<https://www.cnblogs.com/aspirant/p/8991830.html>

【快手】Tomcat的类加载机制是什么样的？是否违反双亲委派机制？

<https://mp.weixin.qq.com/s/4Kv58XUzbZyzvZCM1qx2eA>

<https://mp.weixin.qq.com/s/YsW3MxozoxSrOeSCJOhuBQ>

为什么要违反：1.为了保证多个webAPP之间的隔离性；2.提高公共类库的性能，不用重复加载。3.提高灵活性，热加载使用的是卸载响应的classloader。不影响别人。



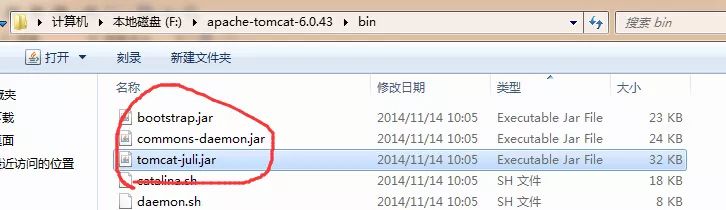
当Tomcat启动时，会创建几种类加载器：

1、Bootstrap 引导类加载器

加载JVM启动所需的类，以及标准扩展类（位于jre/lib/ext下）

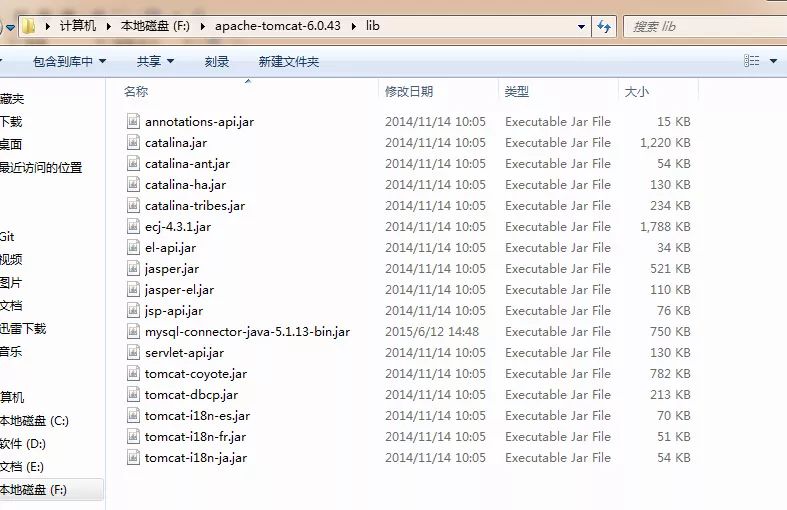
2、System 系统类加载器

加载Tomcat启动的类，比如bootstrap.jar，通常在catalina.bat或者catalina.sh中指定。位于CATALINA\_HOME/bin下



3、Common 通用类加载器

加载Tomcat使用以及应用通用的一些类，位于CATALINA\_HOME/lib下，比如servlet-api.jar



4、webapp 应用类加载器

每个应用在部署后，都会创建一个唯一的类加载器。该类加载器会加载位于 WEB-INF/lib下的jar文件中的class 和 WEB-INF/classes下的class文件

当应用需要到某个类时，则会按照下面的顺序进行类加载：

1、使用bootstrap引导类加载器加载

2、使用system系统类加载器加载

3、使用应用类加载器在WEB-INF/classes中加载

4、使用应用类加载器在WEB-INF/lib中加载

5、使用common类加载器在CATALINA\_HOME/lib中加载