# JavaWeb开发入门

Web是一种分布式应用架构，旨在共享分布在网络上的各个Web服务器中的所有相互链接信息。Web采用客户/服务器通信模式，客户与服务器之间用HTTP协议通信。Web使用超级文本技术(HTML)来链接网络上的信息。信息存放在服务器端，客户机通过浏览器(如IE、FireFox、Netscape)就可以查找网络中的各个Web服务器上的信息。

与Web相关的一个概念是WWW(World Wide Web)。WWW是指全球范围内的Web，它以Internet为网络平台，Internet是来自世界各地的众多相互连接的计算机和其他设备的集合，而WWW则是Internet上的一种分布式应用结构。

归纳起来，Web具有以下3个特征：

* 用超级文本技术HTML来表达信息，以及建立信息与信息的链接。
* 用统一资源定位技术URL来实现网络上信息的精确定位。
* 用网络应用层协议HTTP来规范浏览器与Web服务器之间的通信过程

## WEB开发的相关知识

* WEB，在英语中web即表示网页的意思，它用于表示Internet主机上供外界访问的资源。
* Internet上供外界访问的Web资源分为：
  + 静态web资源（如html 页面）：指web页面中供人们浏览的数据始终是不变。
  + 动态web资源：指web页面中供人们浏览的数据是由程序产生的，不同时间点访问web页面看到的内容各不相同。
* 静态web资源开发技术
  + Html
* 常用动态web资源开发技术：
  + JSP/Servlet、ASP、PHP等
  + 在Java中，动态web资源开发技术统称为JavaWeb，我们课程的重点也是教大家如何使用Java技术开发动态的web资源，即动态web页面。
  + javaweb：使用java技术开发web页面。供浏览器访问的项目

注：动态web资源最终要生成html，只是不同的时间访问生成的html内容不一样。

## 什么是WEB服务器

* 学习web开发，需要先安装一台web服务器，然后再在web服务器中开发相应的web资源，供用户使用浏览器访问。

如在我们本机的D盘有一个a.html，怎样让其他的电脑访问这个网页呢？答：必须有一个网络服务程序，这个网络服务程序就是服务器，这样的服务器指的是软软件服务器，并不是硬件服务器。我们写一个Java程序来监听某个端口，当其他机子访问这个端口的时候，我们就把a.html文件发给它，这样别人就可以访问了这个网页了。代码哪下：

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

ServerSocket ss = **new** ServerSocket(10086);

**int** i = 0;

**while**(i++ < 10) {

**final** Socket s = ss.accept();

**new** Thread(){

**public** **void** run() {

**try** {

BufferedWriter out = **new** BufferedWriter(**new** OutputStreamWriter(s.getOutputStream()));

BufferedReader in = **new** BufferedReader(**new** FileReader("src/demo.html"));

**int** len = -1;

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

**while**((len = in.read(cbuf)) != -1){

out.write(cbuf,0,len);

}

out.close();

in.close();

s.close();

} **catch** (Exception e) {

e.printStackTrace();

}

}

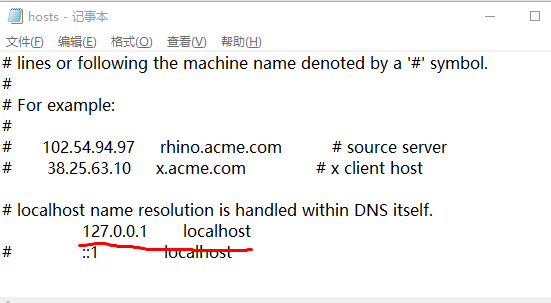
}.start();

}

ss.close();

}

代码写好后，运行，就相当于在本机开启了一台服务器，等待客户端连接。用ie输入：<http://localhost:10086/> ，注：使用localhost必须是配置了C:\Windows\System32\drivers\etc下的hosts文件，如下：



<http://localhost> 中的localhost就是一个域名，怎么知道这个域名访问是哪台电脑的呢（也就是访问哪个ip呢？），通过上面的hosts文件配置，我们告诉了电脑，localhost这个域名对应的ip地址是127.0.0.1，而这个地址是访问电脑本机的地址，也就是说去访问这个ip，不会去联网找别人的电脑，而是访问自己。

效果如下：



注：只能用IE浏览器访问，用非IE浏览器访问时访问不到，具体原因不详，如下：



1、不管什么web资源，想被远程计算机访问，都必须有一个与之对应的网络通信程序，当用户来访问时，这个网络通信程序读取web资源数据，并把数据发送给来访者。

2、WEB服务器就是这样一个程序，它用于完成底层网络通迅。使用这些服务器，用户只需要关注web资源怎么编写，而不需要关心资源如何发送到客户端手中，从而极大的减轻了开发者的开发工作量。

## 什么是JavaEE规范

JavaEE是13中Java技术的总称。

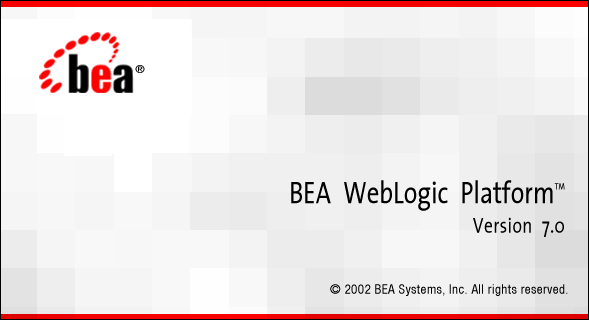
## 常见WEB服务器

WebLogic支持JavaEE规范，也就是支持Java的13种技术，那么这个服务器容量肯定很大，而我们平时的开发中只用到13种技术中的一、两种，如果我们用这个服务器来开发的话，就相当于拿着牛刀杀鸡，大才小用，而且浪费空间。如果是企业级的话，可能就需要用到它了。它是收费的，学习用途不用钱，商业用途就收钱。

WebLogic是[美国](http://baike.baidu.com/view/2398.htm)bea公司出品的一个application server确切的说是一个基于JavaEE架构的[中间件](http://baike.baidu.com/view/23710.htm)，BEA WebLogic是用于开发、集成、部署和管理大型分布式Web应用、网络应用和[数据库应用](http://baike.baidu.com/view/4195600.htm)的Java[应用服务器](http://baike.baidu.com/view/995472.htm)。将Java的动态功能和Java Enterprise标准的安全性引入大型网络应用的开发、集成、部署和管理之中。

目前WebLogic已并入Oracle公司。

* WebLogic是BEA公司的产品，是目前应用最广泛的Web服务器，支持JavaEE规范，而且不断的完善以适应新的开发要求，启动界面如图



* 另一个常用的Web服务器是IBM公司的WebSphereAS，支持JavaEE规范，启动界面如图



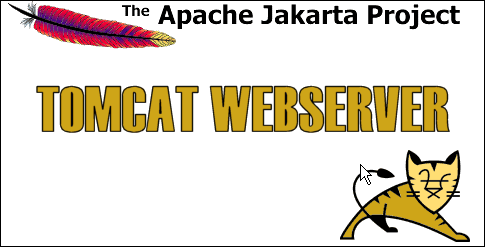
这个服务器也是相当的大，如果没有4G的内存可能运行不起来。

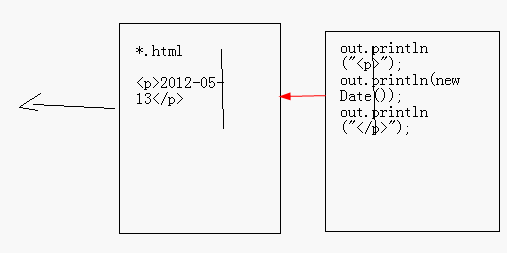
* JBoss：大名鼎鼎的红帽的产品，原来属于开源组织JbossAS，后被红帽收购。支持JavaEE规范，免费。这个用起来很方便，之前只有120多M，解压后即可用，删除即是卸载。



* 以上3个都是大型的服务器，我们学习时只学JSP和Servlet，因此只要支持这两种技术的服务器就够了，因此介绍Tomcat服务器，该服务器容量很小，只有几M。

在小型的应用系统或者有特殊需要的系统中，可以使用一个免费的Web服务器：Tomcat，该服务器支持全部JSP以及Servlet规范，启动界面如图



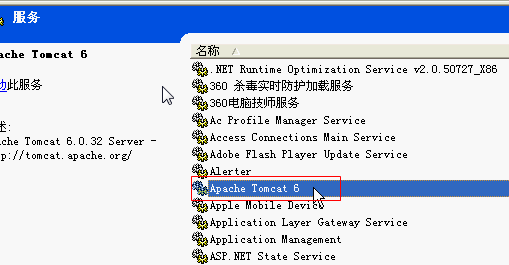


Tomcat可以负责上图右边动态的部分和左边静态的部分，当然也可以把静态的部分让其他的Web服务器来处理。我们学习时就不分了，只用Tomcat来学习。不是技术管理人员，不需要知道这个服务器分不分，我们做为开发人员，只管开发就行了。

# Tomcat简介

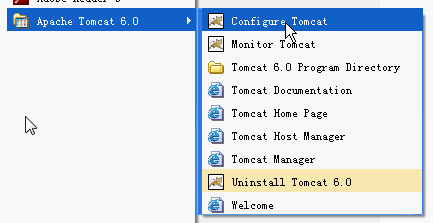
* Web访问默认的服务器端口是：80,Tomcat为了区别，一般用8080端口
* Tomcat是一个服务器，也称Servlet容器（实现了什么技术的服务器就称为什么容器，如实现了JavaEE的服务器可称为JavaEE容器）
* Tomcat是在SUN公司推出的小型Servlet/JSP调试工具的基础上发展起来的一个优秀的Servlet容器，Tomcat本身完全用Java语言编写。目前是Apache开源软件组织的一个软件项目，它的官方网址为[http://tomcat.apache.org](http://tomcat.apache.org/)。得到了广大开源代码志愿者的大力支持，且可以和目前大部分的主流Web服务器(IIS、Apache服务器)一起工作（如何集成可以在网上找张孝详老师的教程），它运行稳定、可靠且高效。已成为目前开发企业JavaWeb应用的最佳Servlet容器选择之一。
* 我们后面的JavaWeb学习都是基于Tomcat服务器的。
* **Tomcat官方站点**：http://tomcat.apache.org
* **获取Tomcat安装程序包**
  + tar.gz文件是Linux操作系统下的安装版本
  + exe文件是Windows系统下的安装版本
  + zip文件是Windows系统下的压缩版本  
    apache-tomcat-6.0.32-src.zip是源文件  
    apache-tomcat-6.0.32-windows-x86.zip 才是Tomcat软件

安装版本的会注册一个服务，如下：

****

**把这个服务启动，Tomcat就启动了。**

**开始菜单：**

****

## Tomcat的版本

Tomcat的版本随着SUN公司推出的Servlet/JSP规范不同而不同，基本上是SUN公司每推出一个版本，Tomcat也会发行新的版本以适应新的规范。

Tomcat的版本，5.0以下的版本基本就没人用了，所以5.0以下的版本可以不去掌握。

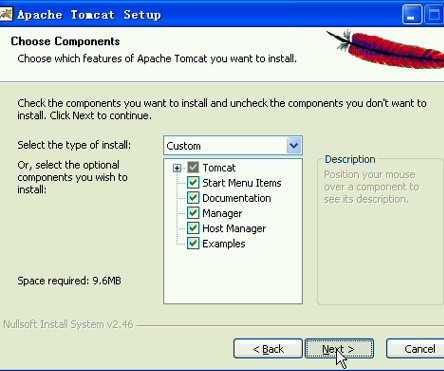
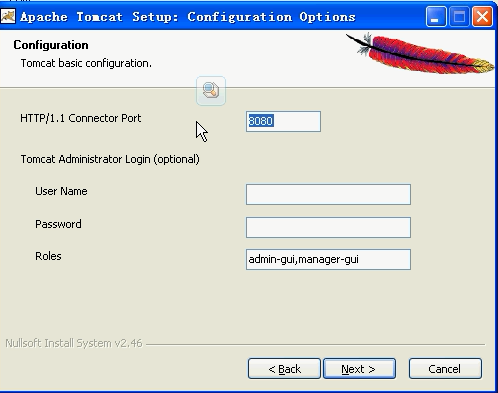
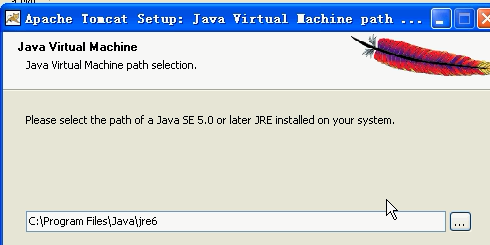
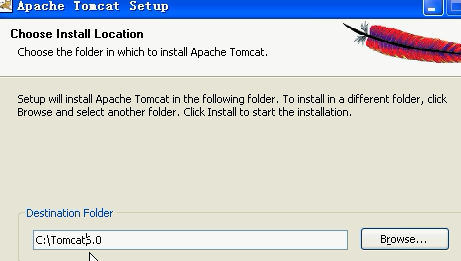
下图中，5.0版本和5.5版本有什么区别呢？它们支持的规范和需要的JDK版本一样，而5.5是比较稳定的版本。目前决定用的版本是6.0的，因为7.0的用了一些注解什么的技术。

|  |  |  |
| --- | --- | --- |
| Tomcat版本 | Servlet/JSP规范 | JDK版本 |
| 7.0 | 3.0/2.2 | 1.6 |
| 6.0.x | 2.5/2.1 | 1.5 |
| 5.5.x | 2.4/2.0 | 1.4 |
| 5.0.x | 2.4/2.0 | 1.4 |
| 4.1.x | 2.3/1.2 | 1.3 |
| 3.3.x | 2.2/1.1 | 1.1 |

Tomcat版本6.0.x对应Servlet2.0/JSP2.1规范，对应的JDK为1.5

Tomcat6.x在默认情况下采用Eclipse的JDT(Java Development Tool)来编译JSP文件，在Tomcat6的发行包中自带了JDT(ecj.jar)，而之前的Tomcat发行版本用JDK提供的Java编译器来编译JSP文件。因此，在使用Tomcat6.x时，即使不完整安装JDK，只要安装了用于运行Java程序的JRE(Java Runtime Environment,Java运行时环境)，就可以运行Tomcat6.x了。

## 安装Tomcat

* Tomcat6本身是基于JDK1.5的应用程序，因此在安装和使用之前必须先行安装1.5及以上版本的JDK(JRE)
* 解压安装：在everything搜索“apache-tomcat-6.0.36-windows-x64.zip”
* Exe安装  
    
  如果用安装版本，建议这里的选项都勾上  
    
  网页默认端口是80，如输入：http://www.baidu.com，其实完整应该是：<http://www.baidu.com:80>  
  这里用户、密码用于管理Tomcat里面的应用，最好设置一下，也可以不设置的。  
    
  指定JRE路径  
    
  这里指定Tomcat安装位置，不要有空格
* Tomcat的配置：

解压版的Tomcat需要配置JAVA\_HOME=C:\Java\jdk1.6.0 ，注意：后面不能加分号（；），JAVA\_HOME必须都大写，必须写一模一样。

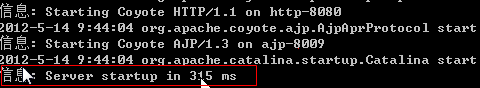
Tomcat的安装目录不要有空格，也不要有中文。在CMD命令行中输入目录名时，按Tab键可以自动补全目录名。

配置在命令行直接输startup就可以启动tomcat：

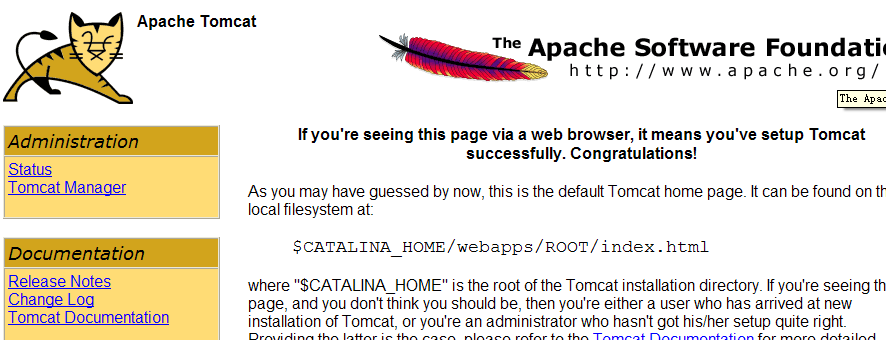
配置：CATALINA\_HOME=C:\apache-tomcat-6.0.32

配置：path=%CATALINA\_HOME%\bin;

* 验证Tomcat是否能用：在cmd中进入Tomcat的安装目录中的 bin 目录下，输入startup.bat ，这时会显示启动Tomcat用了多少时间，如下：



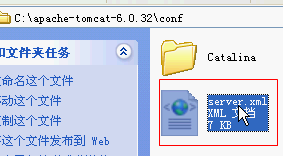
输入 http://localhost:8080/，显示如下界面代表安装成功

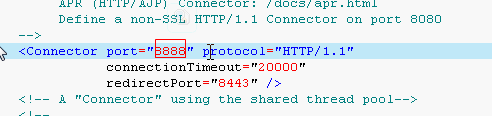


## Tomcat常见启动问题

### 1、JAVA\_HOME环境变量没设置

### 2、端口占用问题(更改默认端口)

修改端口：  


打开这个文件，搜索8080，然后改成自己想要的端口(1024-65535)，如下改为了8888：  


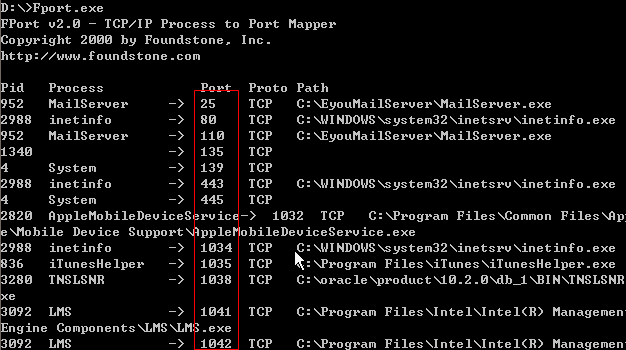
修改端口后要重新启动Tomcat

#### 通过Fport.exe查看端口占用

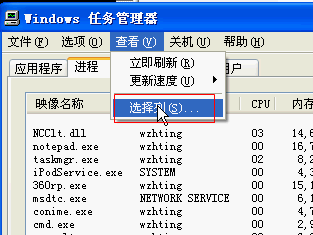
通过Fport.exe查看正在被使用的端口，然后关闭该端口让Tomcat使用：

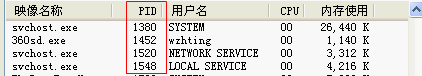
注：Fport.exe放到不带中文中不带空格的路径下

在cmd中进入到Fport.exe所在目录，然后输入Fport.exe，如下：



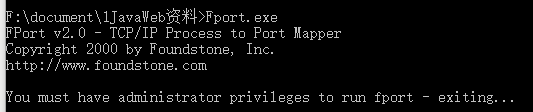
可看到左边有对应的Pid，打开任务管理器：



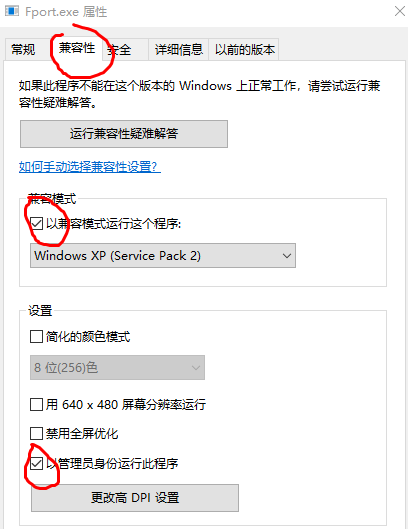


通过pid把进程结束即可。

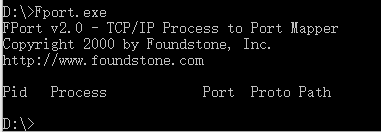
注：运行Fport.exe时有可能报如下信息:



提示信息说必须有管理员权限来运行fport，右击Fport.exe，勾选如下内容：



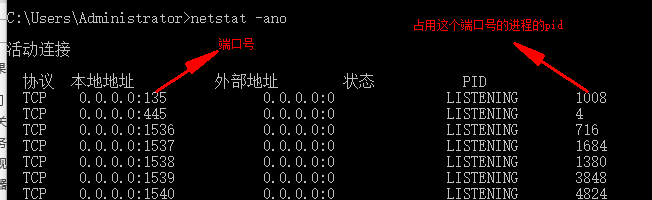
不知道是不是不支持64位电脑，还是不支持Win10，我的电脑配置后运行效果如下：



查看了Fport的文档才知道了，原来是不支持Win10，只支持这些系统：fport supports Windows NT4, Windows 2000 and Windows XP

#### 通过cmd命令查看端口占用

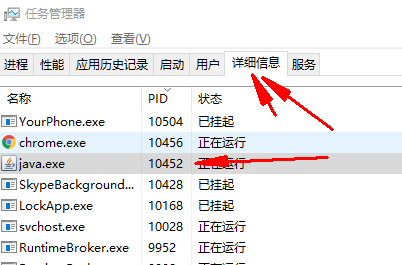
也可以用系统自带的命令查看端口：netstat -ano



这样去找端口占用太慢了，使用过滤：netstat -ano|findstr "8080"



这样就很快看到占用8080端口的进行的pid是10452，在任务管理器中找这个进程：



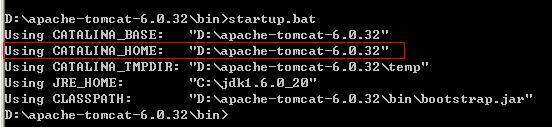
#### 通过360流量防火墙查看



点击“管理”按钮，里面有结束进程的功能

### 3、Catalina\_home环境变量的设置问题

在没有配置之前，在哪个Tomcat的安装目录\bin目录下通过startup.bat来启动Tomcat，则这时的CATALINA\_HOME则会自动设为当前目录的Tomcat目录为该目录，如下运行D盘下的Tomcat：



**不建议配置CATALINA\_HOME环境变量，这个环境变量是用来指定Tomcat的根目录的。为什么不建设呢，原因如下：**

1、在C盘中的Tomcat目录下的webapps目录下新建一个a目录，在该目录下创建一个a.html，内容为aaaaaa

2、再解压一个Tomcat服务器到D盘，把上页的a目录复制到D盘的Tomcat\webapps目录下，并把a.html内容改为bbbbbb

3、建新环境变量：CATALINA\_HOME=C:\apache-tomcat-6.0.32

4、在CMD运行D盘中的服务器

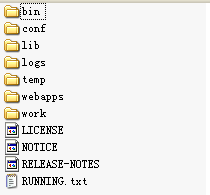
5、在IE输入：http://localhost:8080/a/a.html，可以看到网页的内容为aaaaaaa，虽然我们启动的是D盘的Tomcat，但是就是因为配置了CATALINA\_HOME，在运行时跟据CATALINA\_HOME的配置去运行Tomcat，因为配置的是C盘的，所以即便在D盘的Tomcat目录下输入：startup，但是运行的依旧是C盘的。

6、不用配置CATALINA\_HOME也可以运行Tomcat，如果没配置则会把CATALINA\_HOME配置成当前目录下的Tomcat目录。也就是说，在哪个目录下运行startup就会运行哪个目录下的Tomcat服务器

**catalina——脚本名**

脚本catalina用于启动和关闭tomcat服务器，是最关键的脚本 ，在tomcat里有此文件

## Tomcat的目录结构



存放启动和关闭 Tomcat 的脚本文件

存放 Tomcat 服务器的各种配置文件

存放 Tomcat 服务器的支撑jar 包

存放 Tomcat 的日志文件

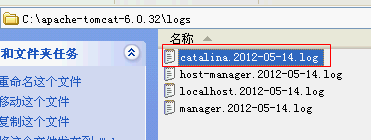
存放 Tomcat 运行时产生的临时文件

（全称是Web Applications）web应用所在目录，即供外界访问的web

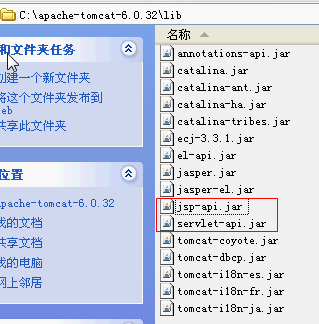
资源的存放目录

Tomcat 的工作目录

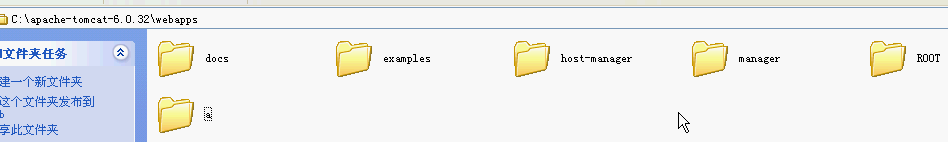
Tomcat启动的信息保存在：

  
如果启动时，Tomcat一闪而过就挂了，则可以在这里找Log看看异常信息

在lib目录中，可以看到支持servlet和jsp，用WinRAR打开servlet-api.jar，可以发现是javax包，看到有java开头的，那就是Sun公司定义的一些规范，Tomcat实现了这个规范，所以Tomcat是servlet容器。



webapps目录（该目录下有几个目录，就代表管理着几个应用）



这里面的一个目录就是一个web应用，所以这里有6个web应用。

# JavaWEB应用程序

* WEB应用程序指供浏览器访问的程序，通常也简称为web应用。
* 一个web应用由多个静态web资源和动态web资源组成，如:
  + html、css、js文件
  + Jsp文件、java程序、支持jar包
  + 配置文件
  + ……
* Web应用开发好后，若想供外界访问，需要把web应用所在目录交给web服务器管理，这个过程称之为虚似目录的映射。
* JavaWeb SUN的Servlet规范对JavaWeb应用作了这样的定义“JavaWeb应用由一组Servlet/JSP、HTML文件、相关Java类以及其他可以绑定的资源构成，它可以在由各种供应商提供的符合Servlet规范的Servlet容器中运行”。

## 在JavaWeb应用中可以包含以下内容：

* Servlet组件：标准Servlet接口的实现类，运行在服务器端，包含了被Servlet容器动态调用的程序代码。
* JSP组件：包含Java程序代码的HTML文档，运行在服务器端。当客户端请求访问JSP文件时，Servlet容器先把它编译成Servlet类，然后动态调用它的程序代码。
* 相关的Java类：开发人员自定义的与Web应用相关的Java类。
* 静态文档：存放在服务器端的文件系统中，如HTML文件、图片文件和声音文件等。当客户端请求访问这些文件时，Servlet容器会先先从服务器上的本地文件系统中读取这些文件，再把它们发送给客户端。
* 客户端类：由客户端来运行的类。Applet是典型的客户端类。当客户请求访问Applet时，Servlet容器先从服务器上的本地文件系统中读取Applet的class类文件的数据，再把它发送到客户端，由客户端来运行Applet。
* web.xml文件：JavaWeb应用的配置文件，采用XML格式。

## JavaWeb应用的目录结构（SUN公司指定的）

开发web应用时，不同类型的文件有严格的存放规则，否则不仅可能会使web应用无法访问，还会导致web服务器启动报错。

Web应用所在目录

这些文件一般存在web应

用根目录下，**根目录下的**

**文件外界可以直接访问**

java类、jar包、web应

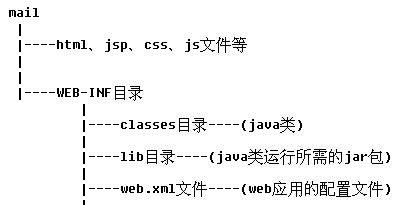
用的配置文件存在这个

目录下，**该目录下的文**

**件外界无法非法直接访**

**问，由web服务器负责**

**调用**



MyApp

html\jsp\css\js

WEB-INF:此目录必须存在，且大小写一致。（存放在该目录下的资源，用户无法直接访问）

classes:存放编译好的class文件。必须有

lib:存放当前应用需要到的jar包。必须有

web.xml:当前应用的配置文件。必须有

假如在MyApp目录下面所需的目录或文件不齐全，那么浏览器一样能访问里面的网页，只是这个目录不能叫做JavaWeb应用，只能叫做Web应用。

在WEB-INF目录的classes及lib子目录下，都可以存放java类文件。在运行时，Servlet容器的类加载器先加载classes目录下的类，再加载lib目录下的JAR文件中的类。因此，如果两个目录下存在同名的类，classes目录下的类具有优先权。

我们注意到Tomcat的安装目录下也有一个lib目录，这个与Web应用中的lib目录的区别在于：

Tomcat的lib子目录：存放的JAR文件不仅能被Tomcat访问，还能被所有在Tomcat中发布的JavaWeb应用访问。

JavaWeb应用的lib子目录：存放的JAR文件只能被当前JavaWeb应用访问。

假如Tomcat类加载器要加载一个MyClass的类，它会按照以下先后顺序到各个目录中去查找MyClass的class文件，直到找到为止，如果所有目录中都不存在MyClass.class的文件，则会抛出异常：

1、在JavaWeb应用的WEB-INF/classes中查找MyClass.class文件。

2、在JavaWeb应用的 WEB-INF/lib目录下的JAR文件中查找MyClass.class文件。

3、在Tomcat的lib子目录下直接查找MyClass.class文件。

4、在Tomcat的lib子目录下JAR的文件中查找MyClass.class文件。

Note：

Tomcat6.x与Tomcat5.x的目录结构有所区别。在Tomcat5.x版本中，Tomcat允许在server/lib目录、common/lib和shared/lib目录下存放JAR文件，这3个目录的区别在于：

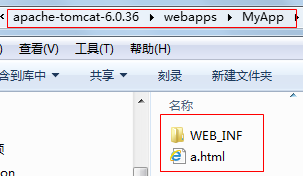
在server/lib目录下的JAR文件只可被Tomcat访问。

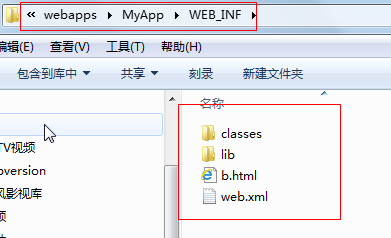
在shared/lib目录下的JAR文件可以被所有的JavaWeb应用访问，但不能被Tomcat访问。

在common/lib目录下的JAR文件可以被Tomcat和所有JavaWeb应用访问。

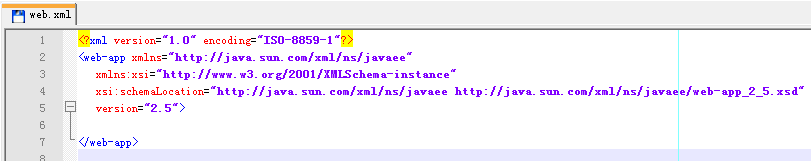
## 创建JavaWeb应用实例

根据上面的要求创建目录如下：

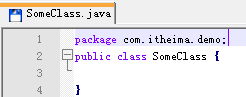
 注：图中文件夹错误，应为：WEB-INF，这中间的不是下划线，该目录写错时，启动Tomcat不会有错误信息

注：图中文件夹错误，应为：WEB-INF，这中间的不是下划线

对于web.xml中的内容，可以复制ROOT应用中的web.xml，复制后删除多余的，保留xml的根目录即可，如下：

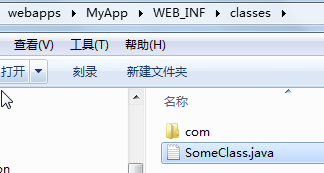


在classes目录下创建.class文件：



上面是源文件，需要的是.class文件：

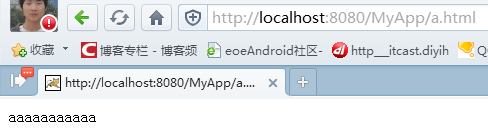


注：图中文件夹错误，应为：WEB-INF，这中间的不是下划线

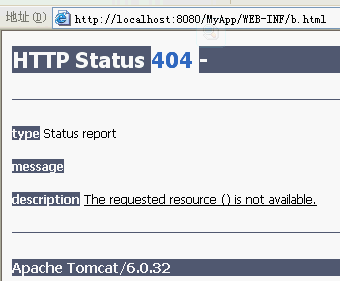
### 访问我们的JavaWeb应用

1、开启Tomcat

2、在浏览器中输入：<http://localhost:8080>   
 在Tomcat的webapps目录中有多个web应用，要访问哪一个应用要指定，所以应输入：<http://localhost:8080/MyApp>  
 在MyApp应用下有许多的资源，要访问哪个资源也需要指定，所以就输入：<http://localhost:8080/MyApp/a.html>，效果如下：

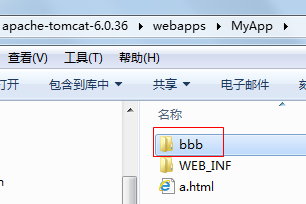


访问WEB-INF目录下的资源是无法直接访问的，如访问WEB-INF下的b.html，错误如下：



因此这个目录下可以存放一些不想被外界直接访问的资源，虽然不能被外界直接访问，可是应用内部是可以互相访问的

假如下面的bbb目录中有一个bbb.html，那么访问地址为：<http://localhost:8080/MyApp/bbb/bbb.html>

注：图中文件夹错误，应为：WEB-INF，这中间的不是下划线

### 发部

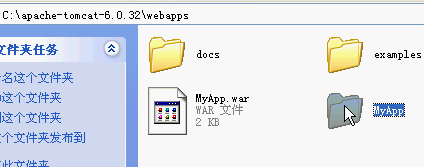
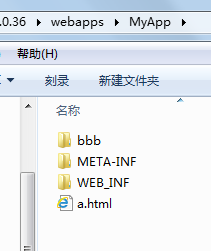
把开发好的JavaWeb应用交给Tomcat管理的过程称之为发布

发布方式：

1、开放式目录方式：把你的web应用的目录拷贝到Tomcat/webapps目录下

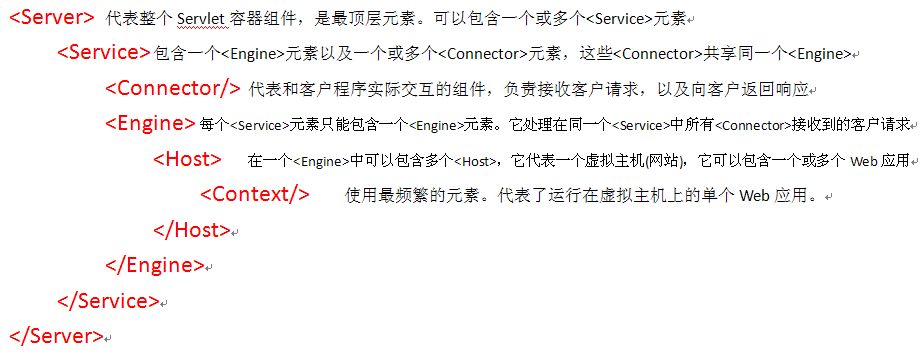
2、war包发布方式：把你的web应用打成war包，再拷贝到Tomcat/webapps目录下

方法：在命令行进入应用的根目录，执行 jar -cvf 应用名.war . （注：后面的点作用是把应用打包后放在当前目录，这个war包可以用WinARAR打开查看），这里的应用名可以随便取，不一定要与应用的文件夹名字相同

3、注意：先启动Tomcat，当把war包复制到Tomcat/webapps目录下时，Tomcat会自动把war包按照war文件名作为一个文件夹解压，如果这时删除war包，那么解压出来的文件夹也会自动被删除。  
 把war包复制到webapps目录后会自动解压，如下：  
  注：图中文件夹错误，应为：WEB-INF，这中间的不是下划线

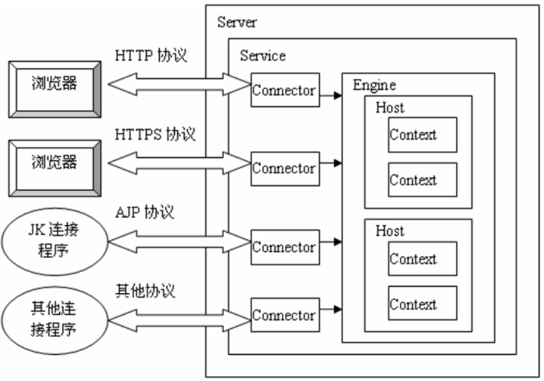
# Tomcat的组成结构

* Tomcat本身由一系列可配置的组件构成，其中核心组件是Servelt容器组件，它是所有其他Tomcat组件的顶层容器。每个组件都可以在Tomcat安装目录/conf/server.xml文件中进行配置，每个Tomcat组件在server.xml文件中对应一种配置元素。以下用XML的形式展示了各种Tomcat组件之间的关系



这里的一个标签就是封装了一个Java对象，设置标签的属性就是设置Java对象的属性

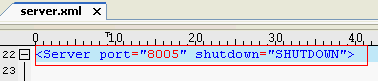
## Tomcat体系架构



不同的协议连接到不同配置的Connector，在Connector里收到了请求，要访问的主机、资源，这个请求就交给了Engine来处理，Engine就找到对应的Host里面对应的Context

## /conf/server.xml分析

如果这个文件修改不正确，则在启动Tomcat或已启动的Tomcat中会打印出错误信息

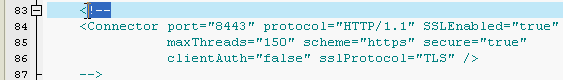
根元素：Server：

在Server里面有个服务：Service，这个服务的名字叫“Catalina”：

在这个服务里面的连接器：Connector：，像：<http://localhost:8080/MyApp/a.html>这个连接就是通过这个链接器来访问JavaWeb应用的。

1、客户端连接服务器：Connector

又一个连接器：Connector：，这个连接器是和其他服务器集成用的。

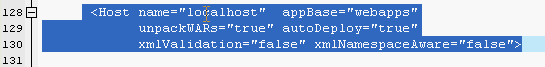
一个被注释掉的连接器：Connector： 

这个连接器多了个 SSLEnabled=”true”，这是带安全的http协议，网上用的https开头的网址就是用这个连接器，像网上银行用的都是这个链接器

一个服务里有一个引擎：，这个引擎的名字也叫“Catalina”，这里指定了默认的主机使用名为“localhost”的主机。多个连接器共用一个Engine，一个Engine管理多个主机：Host

2、虚拟主机：Host

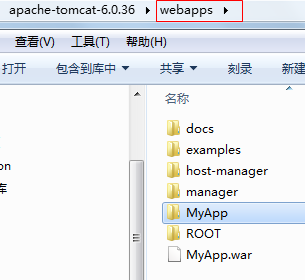
主机的概念，如：我们建了一个网站想放到网上，那么一般是买一个虚拟空间，这个空间就是一个Host

一个Engine下有多个Host：

如果我们要做多个网站，每个网站的域名都不一样，则这里需要创建多个Host与多个网站对应。

name:虚拟主机的名称

appBase:该主机所有的web应用的存放目录（可以修改，但是一般不要改，但当别人问：所有的web应用都必须放在webapps目录下吗？）

这里面的每一个文件夹就代表一个Context元素

unpackWARs=”true”,解压WAR包，为ture则自动解压。

autoDeploy=”true”，我们把应用复制到webapps目录后就能访问了，是因为这里设置了true，表示自动布署

## Web组件的URL

无论是开放式目录结构还是打包文件方式发布web应用，web应用的默认URL入口都是Web应用的根目录名。例如要访问MyApp应用，它的URL入口为/MyApp，如访问本地服务<http://localhost:8080/MyApp>（http://127.0.0.1:8080/MyApp），通过下面的“虚拟目录的映射方式”，可以了解到可以把Web应用的根目录名映射成一个别名，则访问时URL中可用应用的目录名，也可用映射的别名，如下：

## 虚拟目录的映射方式

### 虚拟目录的映射方式一

让tomcat自动映射： tomcat服务器会自动管理webapps目录下的所有web应用，并把它映射成虚似目录。换句话说，tomcat服务器webapps目录中的web应用，外界可以直接访问。

如往webapps中往往了一个MyApp的应用，则Tomcat自动把这个应用映射为”/MyApp”的路径，则可以这样访问MyApp应用：http://localhost:8080/MyApp

通过映射可以给应用起别名，虚拟目录的映射是通过Context元素来指定的，在name属性为localhost的Host元素下增加Context元素，如下：

<Host name="localhost" appBase="webapps" unpackWARs="true" autoDeploy="true"> //这里的appBase指定了应用的默认位置，当接收到请求的时候，会先在这个位置查找应用，如果在这个位置找不到，再从Context中配置的位置去查找

<Context path="/itheima" docBase="D:\apache-tomcat-6.0.36\webapps\MyApp"/> // 这里的docBase指定了我们应用的位置，而path则是为我们的应用绑定了一个虚拟路径

</Host>

path属性用于设置一个虚拟路径名，以“/”开头

此时可通过<http://localhost:8080/MyApp>是OK的，这个“/MyApp”是Tomcat帮我们映射的，因为我们把MyApp放到了webapps目录下

而我们通过Context给MyApp又映射了一个别名为“/itheima”，所以也可以使用别名访问：<http://localhost:8080/ithaima>

我们也可以新建一上Host标签来映射一个新的主机，上面的主机名为localhost，这相当于域名为localhost，如果我们需要一个新的网站，则需要一个新的主机，新建Host如下：

<Host name="www.itheima.com" appBase="E:\itheimaapps" unpackWARs="true" autoDeploy="true">

</Host>

这时，E:\itheimaapps目录的功能就与webapps目录的功能一样了，我们往itheimaapps文件夹中放入一个MyApp应用或者war包，Tomcat也会自动帮我们部署，且自动映射一个“/MyApp”路径，访问方式为：主机 + “:” + 端口号 + 路径，具体为：<http://www.itheima.com:8080/MyApp>，那电脑怎么知道www.itheima.com这个域名访问的是哪个ip地址呢？这时就需要去修改hosts文件，增加一个域名映射，把这个域名映射为127.0.0.1地址即可。

注：一个Host中可以用多个应用，如果应用并不在appBase目录下，则需要在Host标签下通过Context标签来指明应用位置，并设置别名，如下：

<Host name="www.itheima.com" appBase="E:\itheimaapps" unpackWARs="true" autoDeploy="true">

<Context path="/b" docBase="D:\MyApp"/>

</Host>

这时，MyApp并不在appBase的目录下，所以Tomcat并不会自动给我们映射一个“/MyApp”的路径，所以不能使用<http://www.itheima.com:8080/MyApp>来访问此应用，只能通过别名的方式来访问：<http://www.itheima.com/b/>

这种映射方式需要重新启动Tomcat后才能生效（实际运行过程中不建议使用），比如京东商城，重启一下则可能会损失几千万。

推荐方式：

在Tomcat安装目录/conf/[enginename]/[hostname]目录下建立一个扩展名为xml的配置文件，具体可看下面的“虚拟目录的映射方式二”

### 虚拟目录的映射方式二

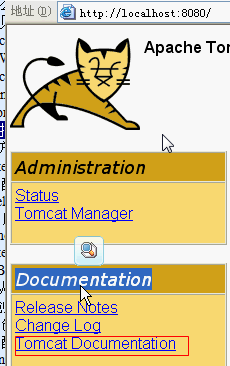
在Tomcat安装目录/conf/[enginename]/[hostname]目录下建立一个扩展名为xml的配置文件，（其实就是D:\apache-tomcat-6.0.36\conf\Catalina\localhost位置，刚解压Tomcat时是没有的，把Tomcat启动一下这个路径就有了），该文件名即为虚拟路径（path），在其中增加一个xml文件，文件名随意，如abcd.xml如下：

<?xml version="1.0" encoding="UTF-8"?>

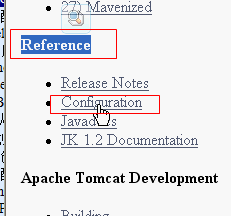
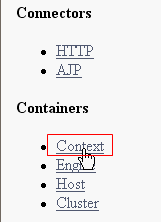
<Context docBase="D:\MyApp"/>

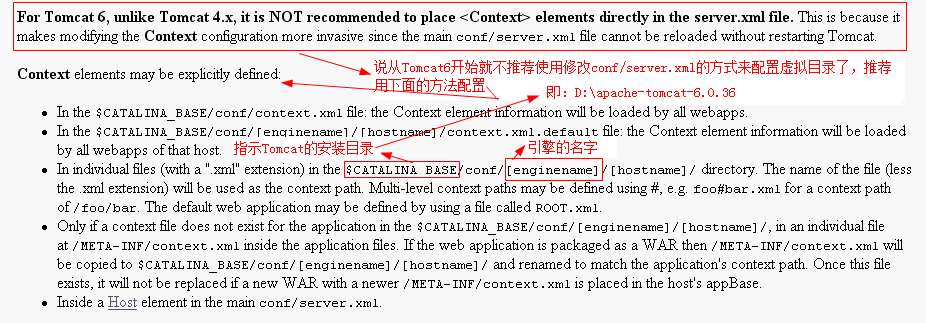
这时，abcd.xml中的文件名abcd即为MyApp的虚拟路径，访问方式为：http://localhost:8080/abcd

要想玩Tomcat很牛B的话，可以把这个文档通读一下：



通过Tomcat Documentation来查看如何配置Context（即如何配置虚拟目录）

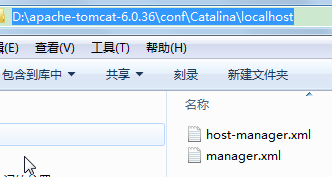
 



conf目录下的server.xml :

上面推荐的多种配置虚拟目录的方法中，我们看第三种，组合起来的路径为：D:\apache-tomcat-6.0.36\conf\Catalina\localhost\ 如下：



在这个目录下创建一个xml文件，如daizhenliang.xml，

在daizhenliang.xml文件中配置Context标签，指定要映射到的真实目录即可，这里不需要配置path属性，因为文件的名字就是虚拟目录的路径，如下：

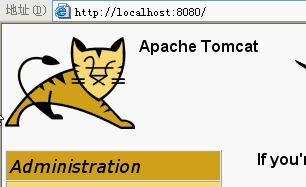
<?xml version="1.0" encoding="UTF-8"?>

<Context docBase="D:\MyApp"/>

访问的网址为：<http://localhost:8080/daizhenliang/a.html>

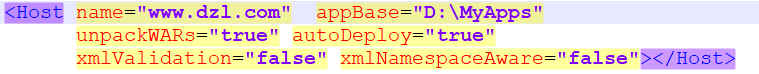
以后想要修改虚拟目录的路径把文件名改了就可以了，不需要重启Tomcat。

### 配置默认的虚拟路径



我们发现这个url并没有指定访问哪个“虚拟路径”，也没有指定要访问哪个资源，其实这里是访问了默认应用里的默认页面，那么哪个应用是默认应用呢？答：是webapps下的ROOT，哪个是默认页面呢？答：index.html。

要把我们的应用配置成默认应用，有个前提是：“我们的应用必须先部署，也就是要先能正常访问，再把它变成默认应用。”比如放到webapps下，Tomcat自动帮我们部署。或者我们自己创建一个Host也可以，如下：



此时往MyApps中放入一个MyWeb应用，此时可通过[www.dzl.com:8080/MyWeb](http://www.dzl.com:8080/MyWeb)访问

好了，配置默认应用的前提条件我们已经准备好了，我们想把MyWeb应用配置成默认应用，这样访问时就可以直接这样访问了： www.dzl.com:8080

如果想把我们自己的应用配置成默认应用，即按照前面讲的“虚拟目录的映射方式二”，把xml的文件名起为ROOT.xml，并在ROOT.xml中指定应用的位置，如下：



然后重启Tomcat即可，此时就可以通过[www.dzl.com:8080](http://www.dzl.com:8080)来访问MyWeb应用，不需要指定虚拟路径了。当然还有个前提是要去hosts中配置一下[www.dzl.com的ip](http://www.dzl.com的ip)为127.0.0.1。为什么默认应用不需要指定虚拟路径?因为这是默认应用啊！默认的意思就是如果你不指定虚拟路径来指定访问具体哪个应用的话，则默认访问“默认应用”，当改成默认应用后，访问时不能加ROOT，如果配置了默认主页，则访问方式为：http://localhost:8080/，如果发现访问时显示的还是原来的Tomcat主界面，则可能是浏览器有缓存的问题，可以清除缓存，或者换一个浏览器访问。

（当然了，我们也可以把自己的应用复制到webapps/ROOT目录下来变成默认应用，不推荐这么做，Tomcat默认应用中的文档还是很有帮助的）

### 问题：

www.dzl.com:8080访问失败

创建了新的Host且位置不是在webapps，配置了默认应用后，发现访问不了，这可用排查法，方法如下：

1、先确定应用是OK的，把应用放一个到webapps下，使用localhost的方式访问这个应用，如果可以说明应用的代码是没问题

2、是否在hosts文件中配置了域名与127.0.0.1对应，如果配置了，验证一下是否生效了。使用[www.dzl.com:8080](http://www.dzl.com:8080)看能否访问到Tomcat主页，如果可以访问说明域名映射也是OK的，最终我发现问题出在这里，然后增加了另一个映射，如下：  
  
发现使用dzl.com是ok的，使用[www.dzl.com](http://www.dzl.com)是不行的，真是见了鬼，不知道什么原因。可能hosts文件就是这样的，不能映射带www的域名。我的是Win10系统，不知道是否与系统有关

3、再后来，用着用着dzl.com也用不了，后来注销后发现[www.dzl.com](http://www.dzl.com)可以用了，dzl.com用不了，真是见鬼的。后来再用着用着，连[www.dzl.com](http://www.dzl.com)也用不了了。

4、最后发现，就是Win10的问题，Win10修改hosts时生效时无效的。

### 配置应用的默认主页

只要是修改当前应用的相关设置，一般都是在当前应用的WEB-INF/web.xml文件里进行配置

修改当前web应用的web.xml文件，在根元素中增加以下内容

<welcome-file-list>

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

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

</welcome-file-list>

当输入的URL没有指定访问哪个页面时（如：，则按上面的顺序查找应用中的页面，先找有没有a.html，如果有就显示，如果没有再找index.html有没有，依次类推，并不是按index这样的文件名来查找的，是按顺序查找的。

如果这个标签不记得怎么写，可以打开conf/web.xml，然后搜索”welcome“就可以找到这个标签，复制粘贴修改即可。

只要牵扯到对当前web应用进行配置，找web.xml文件

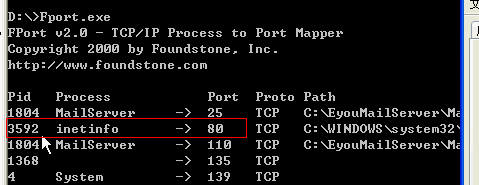
### 配置应用的默认端口

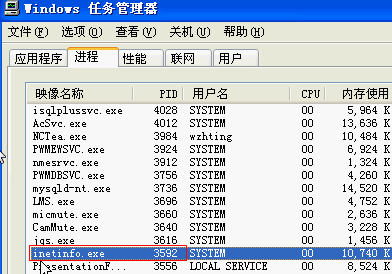
把conf/server.xml中的这个连接器中的端口修改为80即可：



**修改后要重启Tomcat**

在修改之前应该先查看一下80端口是否被占用：





可以发现这个进程结束了之后又自己开启，结束不了，其实它是注册到服务里了，可以这样来结束：

这个服务其实是IIS的服务，我们打开服务窗口：



可以看到该服务是自动启动的，停止即可。

再查看80端口是否被占用。

### <http://www.baidu.com>省略了什么

在我们输入<http://www.baidu.com>是其实省略了默认端口、默认应用、默认主页，如下：

http://www.baidu.com:80/默认应用/默认主页

### Context元素

Tomcat6.x提供了多种配置<Context>元素的途径。当其加载一个web应用时，会依次按照以下五种方式尝试查找Web应用的<Context>元素，直到找到为止：

1)到Tomcat安装目录/conf/context.xml文件中查找<Context>元素。

2)到Tomcat安装目录/conf/[enginename]/[hostname]/context.xml文件中查找<Context>元素。

[enginename]：表示<Engine>的name属性

[hostname]：表示<Host>的name属性。

3)到Tomcat安装目录/conf/[enginename]/[hostname]/[contextpath].xml文件中查找<Context>元素。

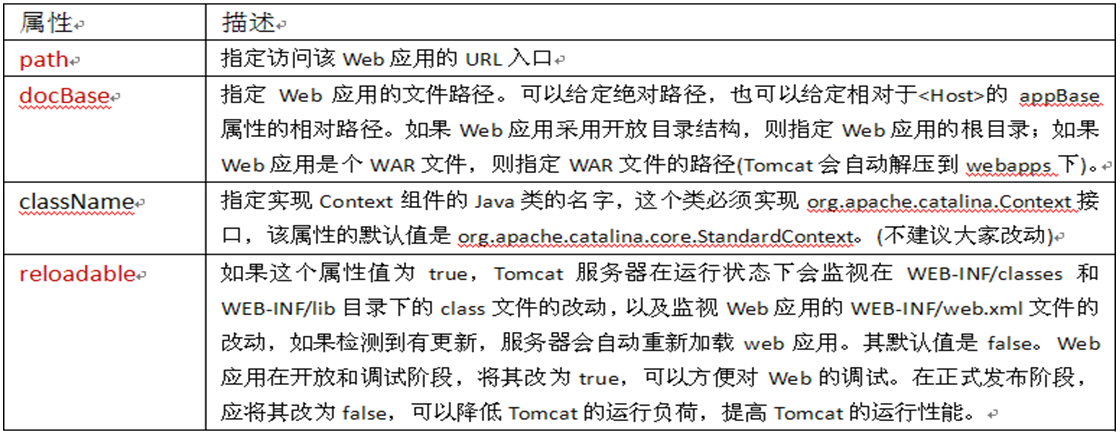
[contextpath]：表示单个Web应用的URL入口。

4)到Web应用的META-INF/context.xml文件中查找<Context>元素。

5) 到Tomcat安装目录/conf/server.xml文件中查找<Context>元素。只适用于单个Web应用。

如果仅仅为单个Web应用配置<Context>元素，可以优先采用第三种配置方式。

### Context元素的属性



在一般情况下，<Context>元素都会使用默认的标准Context组件，即className属性采用默认值org.apache.catalina.core.StandardContext，它除了拥有上面介绍到的属性外，还有自身专有的属性：

cachingAllowed：是否允许启用静态资源(HTML、图片、声音等)的缓存。默认值为true。

cacheMaxSize：设置静态资源缓存的最大值，单位为K。

workDir：指定Web应用的工作目录。

uppackWAR：如果为true，会把war文件展开为开放目录后再运行。为false，直接运行war文件。默认值为true。

### web.xml文件

* 通过web.xml文件，可以将web应用中的：
  + 某个web资源配置为网站首页
  + 将servlet程序映射到某个url地址上
  + ……
  + 但凡涉及到对web资源进行配置，都需要通过web.xml文件
* 举例：通过web.xml文件配置网站首页。
* 注意：web.xml文件必须放在web应用\WEB-INF目录下。

## 配置虚拟主机

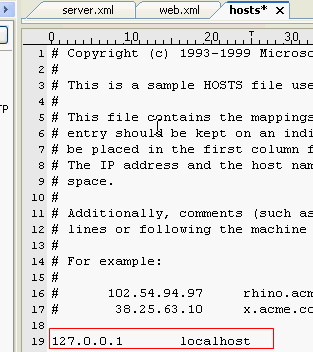
一个虚拟主机可以先理解成一个网站

* 假如两家公司Web应用都发布在同一个Tomcat服务器上，可以为每家公司分别创建一个虚拟主机

[www.itcast.cn](http://www.itcast.cn/) [www.it315.org](http://www.it315.org/)

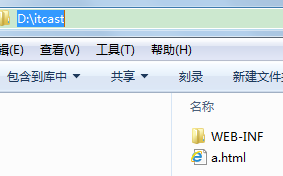
* 尽管以上两个虚拟主机位于同一个主机，但是当客户通过以上两个不同虚拟主机名访问Web应用时，会感觉到这两个应用分别拥有独立的主机
* 如需在WEB服务器中配置一个网站，需使用Host元素进行配置，例：

<Host name=”域名” appBase=”c:\app”></Host>

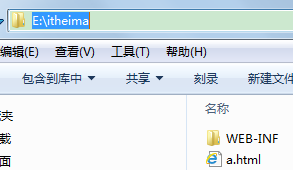
* 配置的主机(网站)要想被外部访问，必须在DNS服务器或windows系统中注册。
* 配置多个网站。
* 缺省虚似主机。
* 修改Windows系统中的C:\WINDOWS\system32\drivers\etc\hosts，如果这个文件不请允许修改，则把这个文件复制到别的目录，修改之后复制回来覆盖即可  
  从这里可以看到localhost这个域名对应的ip地址是127.0.0.1，这个ip地址是本地地址  
  域名服务器（DNS服务器，全称：Domain Naming Server）功能：就是把所有的域名与ip地址一一对应起来。因为记ip地址不好记，所以出现了域名服务器  
  全世界域名服务器最核心的只有6台，一台在欧洲，一台在日本，一台在台湾，剩下的都在美国  
  .com 公司域名  
  .gov 政府域名  
  .edu 教育域名  
  这些都属于顶级域名，顶级域名要便宜些  
    
  把我们的域名也映射进去：  
    
  这就相当于申请了两个域名。但这是保存在本机的hosts文件里的，其他机器无法知道，也就无法通过这两个域名来访问我们的站点

### 配置步骤

以配置[www.itcast.cn](http://www.itcast.cn) [www.it315.org](http://www.it315.org/) 到同一个Tomcat服务器为例：

1、建立好两个站点，如在D盘保存有itcast站点，假设这个站点已经配好了默认主页：  
 

在E盘保存有itheima站点，假设这个站点已经配好了默认主页：



2、打开conf/server.xml，增加两个Host元素，如下：

<Host name="www.itcast.cn" appBase="D:\itcastapps" unpackWARs="true" autoDeploy="true"/>

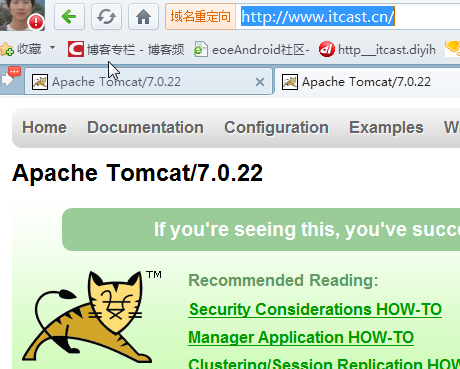
<Host name="www.itheima.com" appBase="E:\itheimaapps" unpackWARs="true" autoDeploy="true"/>

修改后需要重启Tomcat。从这一步中，我们可以知道，这里的主机名一般是域名，而我们在买别人的空间用时，这个域名一般是别人自动有默认的，当想绑定我们的域名时，估计别人也是要在这里修改成我们申请的域名，所以把我们申请的空间和域名绑定到一起可能需要些时间，要看别人什么时候给你修改上去。从这一步也可以看出一个域名只能绑定到一个主机，多个域名可以绑定到同一个主机。

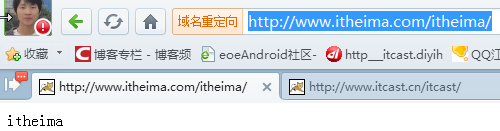
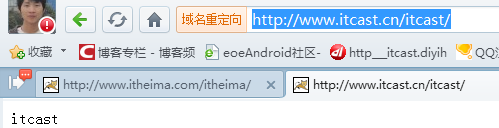
并把这个连接器的端口改为80：  
 ，修改后要重启Tomcat

3、打开C:\WINDOWS\system32\drivers\etc\hosts文件：  
 把我们的域名也映射进去：  
   
 这就相当于申请了两个域名。但这是保存在本机的hosts文件里的，其他机器无法知道，也就无法通过这两个域名来访问我们的站点

4、访问：在浏览器分别输入：<http://www.itheima.com/> 与 <http://www.itcast.cn/> 访问的结果如下：



这里访问到的是默认应用ROOT，因为我们没有配置默认应用，所以访问到了Tomcat默认的应用。  
 所以我们必须告诉电脑要访问哪个应用：通过<http://www.itheima.com/itheima/> 和 <http://www.itcast.cn/itcast/> 来访问各自己的站点，如下：

我们还可以通过Context元素给应用映射一个别名，如：

<Host name="www.itcast.cn" appBase="D:\itcastapps" unpackWARs="true" autoDeploy="true">

<Context path="/a" docBase="D:\itcastapps\itcast"/>

</Host>

<Host name="www.itheima.com" appBase="E:\itheimaapps" unpackWARs="true" autoDeploy="true">

<Context path="/b" docBase="E:\itheimaapps\itheima"/>

</Host>

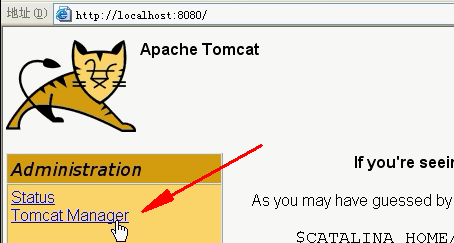
则，此时url中可用应用目录的名字或映射的名字来访问应用，如：

通过<http://www.itheima.com/itheima/> 和 <http://www.itcast.cn/itcast/> 来访问

或 通过 <http://www.itheima.com/b/> 和 <http://www.itcast.cn/a/> 来访问

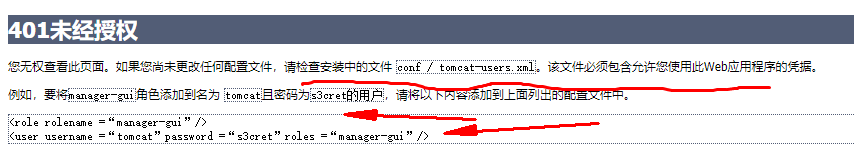
# Tomcat远程管理应用

Tomcat的管理平台是Tomcat本身的一个Web应用，管理平台对应manager应用，它位于Tomcat安装目录/webapps/manager目录



假设你的Tomcat安装在了一台远程服务器上，这里的URL应该换成真实的url外网地址。

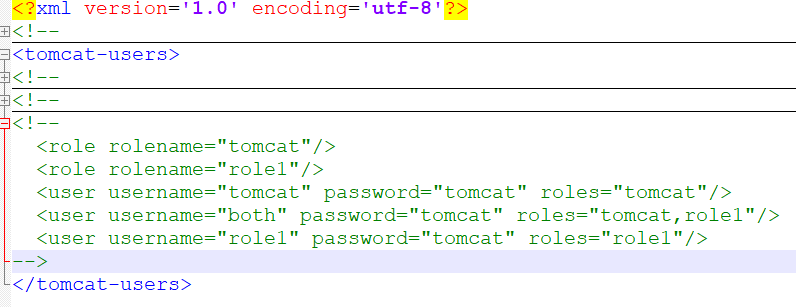
  
之里需要用户名与密码，如果是安装版的Tomcat，在安装时就会有设置用户名与密码的地方，如果是解压版本，则没有，我们点取消，这时会有提示如下：



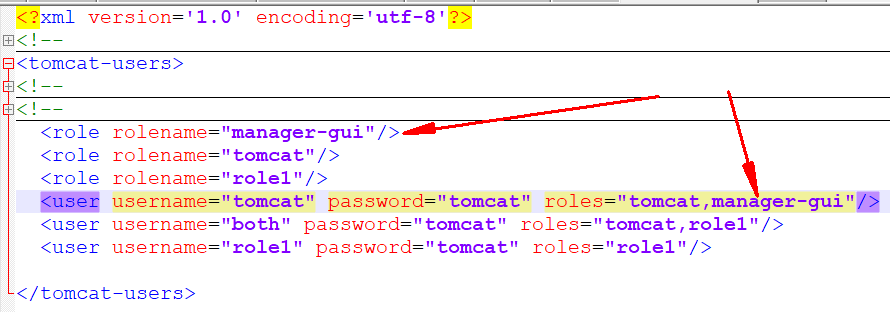
这页面是英文的，是经过Google翻译的，这个报错页面提示可以在conf/tomcat-users.xml中去配置用户名与密码

我们复制下面的第一行代码到tomcat-users.xml文件（<role rolename =“manager-gui”/>）中，在修改这个配置文件之前，先把Tomcat停下来。因为Tomcat是在启动的时候来把这些属性读进内存的。

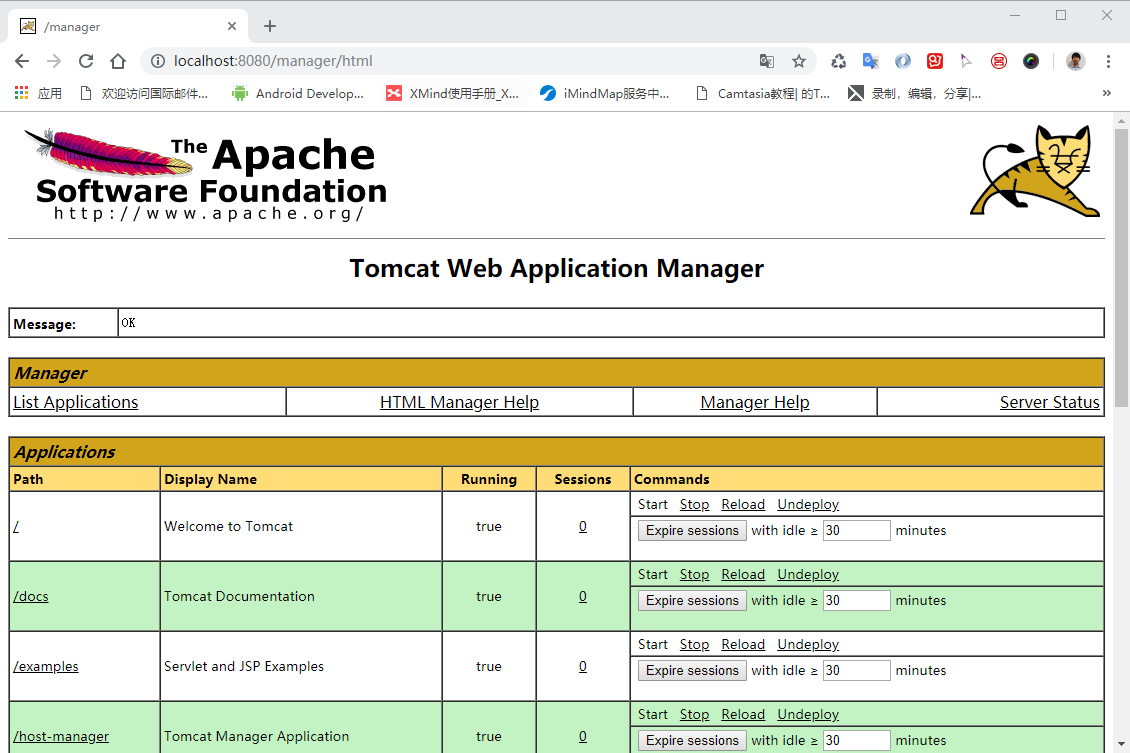
打开tomcat-users.xml文件，如下：



把这个注释打开，把我们上面复制 的第一行代码粘贴进来，并在<user username=”tomcat”这行增加”manager-gui”这个roles，如下：



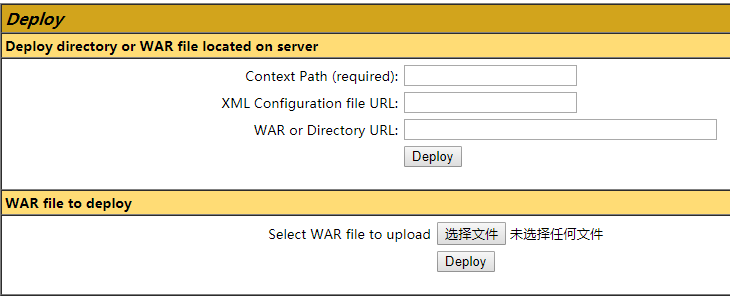
manager-gui这个角色名是Tomcat中程序中写死了的，修改完之后需要重启Tomcat。我们登录Tomcat配置时就是使用用户名为tomcat，密码也是tomcat进行登录即可，他就可以登录manager-gui这个管理界面。界面如下：



Applications中展示了Tomcat管理的应用，这里有几个应用，也可以到webapps下数一数个数对不对。其中/代表的是webapps目录下的ROOT（默认应用）

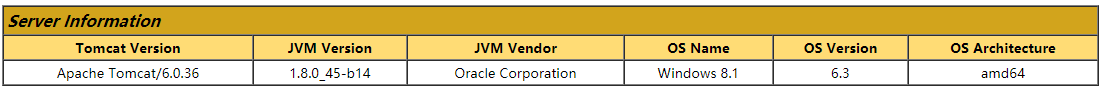
Running表示应用的运行状态，Sessions表示有几个会话，如manager就是1，因为我们当前使用浏览器访问的这个界面就是manager应用的页面，Start、Stop为开启和停止应用，Reload为重新加载应用，Undeploy为从管理的应用中删除。

再往下看这个页面：



这表示远程部署应用，即服务器不是在本地时的部署方式。

再往下有一些系统信息：



这个Manager页面就是用于远程管理应用的。如你的Tomcat在远程，你可以通过URL登录到这个管理页面来管理远程的应用或部署应用。

# 怎样通过URL地址找到网站的？

在输入<http://www.itheima.com> 时，首先找本地的hosts文件中是否有对应的ip地址，如果没有则查找DNS服务器中对应的ip地址，全球主要的DNS服务器就6台，台湾、日本、欧洲各一台，其他的在美国。为什么会这么少，因为域名服务器不能乱搞啊，你搞一个他也搞一个，如果两个人的域名服务器中把同一个域名映射成了不同的ip，这就乱了，所以域名服务器公司得有规范，不能乱开。

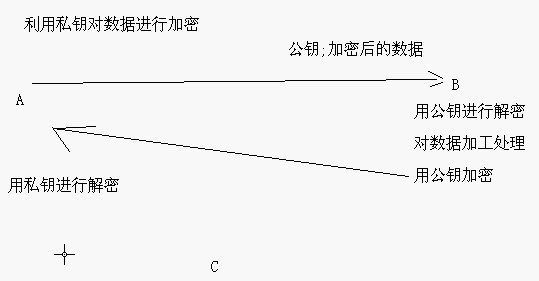
# WEB开发的前景

软件开发的两种架构：c/s和b/s

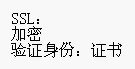
随着网络带宽的不断提升，云计算概念的提出，浏览器只要足够强大，c/s架构立马就会被淘汰，不仅c/s架构会被淘汰，软件最终都会消失、操作系统都可以没有，最终将会是b/s架构的天下，也就是浏览器+搜索引擎的天下。所有现在桌面软件提供的功能，最后都由网站提供，也就是说，将来打开电脑就是一个浏览器，想要什么服务，通过搜索引擎一找，就可以在网上找到相应的服务，用就是了。所以web开发人员是现在最流行的岗位。

# 在Tomcat中配置SSL

## 为什么需要SSL协议



上图的加密方式属于不对称加密方式，假如a和b之间要进行数据传输，a利用私钥对数据进行加密后发送给B，发送的东西为“公钥和加加密后的数据”，B接收到后用公钥进行解密并对数据加工处理后用公钥再加密发送回给A，A再用私钥进行解密。这样不安全，如C可以截取到A发给B的数据，也能拿到A发过去的公钥，就可以解密，处理后再冒充B发回给A。怎么解决这个问题呢？答：出示证书。，如下图：

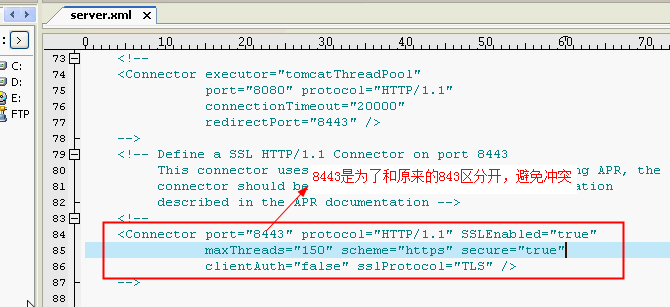


SSL协议就两样东西：加密和验证身份，验证身份是通过“证书”来验证的，这就好比人的身份证可以验证一个人叫什么名字一样。

这样当B在解密A发来的数据前，会让A出示证书，如果证明是A那么才解密处理数据，然后加密再发回给A。A在解密数据前也会验证数据是不是B发来的（通过验证证书），如果验证通过才解密处理数据。这里A发送给B的数据假如给C截取了，C能知道是A发的，可C把数据发回去给A的时候，A验证后不是由B发来的就不处理了，这样就解决了问题。

* 在网络上，信息在由源主机到目标主机的传输过程会经过其他计算机，网络上的信息有可能被非法分子监听，从而导致个人隐私泄露。随着电子商务的发展，对信息安全的要求越来越高，于是Netscape公司提出了SSL(Server Socket Layer)协议，旨在达到在开放网络(Internet)上安全、保密地传输信息的目的。
* SSL是一种保证网络上的两个节点进行安全通信的协议，建立在SSL协议上的HTTP被称为HTTPS协议。**HTTPS使用的默认端口是443**。SSL采用加密技术来实现安全通信，保证通信数据的保密性和完整性，并且保证通信双方可以验证对方的身份。

加密技术的基本原理是：数据从一端发送到另一端，发送者先对数据加密，然后再把它发送给接受者。这样，在网络上传输的是经过加密的数据，如果有人非法截获了这批数据，由于没有解密的密钥，就无法获得真正的原始数据。接收者收到加密的数据后，先对数据解密，然后再处理。  
 Tomcat是支持Https协议的，在conf/server.xml中注释的一个连接器就是用来接收该协议的连接的，如下图：



* SSL通过安全证书来证明客户或服务器的身份。当客户通过安全的连接和服务器通信时，服务器会先向客户出示它的安全证书，而每一个证书在世界范围内都是唯一的。
* 为了扩大用户群，一些服务器向客户出示证书后，并不要求客户出示安全证书，但在某些情况下，服务器也会要求客户出示安全证书，以便核实该客户的身份。如，银行，为了不失去用户数量，一般我们输入用户名密码就能查询余额了，不需要我们出示证书，而且很多人也没有证书。但在交易时，为了安全就必须出示证书了（如Ukey，这就是证书），如转账时就需要证书，这就得去银行办一个U盾了。

## 安全证书的获得有两种方式：

* 1、从权威机构获得证书

Certificate Authority,CA和VeriSign和Thawte，它们保证了证书的可信性。申请证书时，需要付费，且一个证书只针对一个IP有效。如某公司的网站用了两个服务器，一个用电信、一个用联通，两个ip那么就必须申请两个证书。

* 2、创建自我签名的证书

如果只关心数据在网络上的安全传输，而不需要对方身份验证，可以创建自我签名的证书。与权威机构颁发的证书采用的加密技术是一样的，只是人家的是权威，自创的不权威而已，其他都一样。

可以根据安全需要，为Tomcat配置SSL，它包含以下两个步骤：

1、准备安全证书

2、配置Tomcat的SSL连接器(Connector)

SUN公司提供了制作证书的工具keytool。在JDK1.4以上版本中都包含了这一工具，在JDK根目录中的bin目录下，通过这个工具生成证书的命令为:

keytool -genkey -alias tomcat -keyalg RSA ，这个命令将生成包含一对非对称密钥和自我签名的证书，这个命令中的参数含义如下：

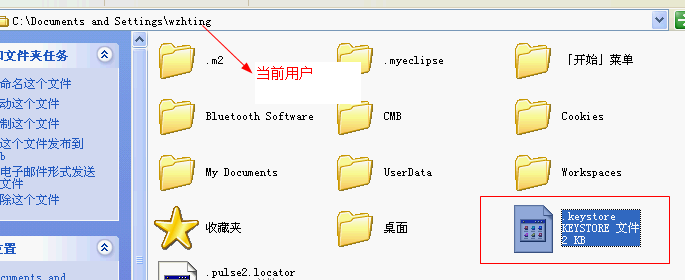
* + -genkey：生成一对非对称密钥。
  + -alias：指定密钥对的别名，该别名是公开的。上面命令的别名为”tomcat”
  + -keyalg：指定加密算法，此处采用通用的RSA算法。

这个命令将在操作系统的用户目录下生成名为”.keystore”的文件，这个文件就是证书文件

如果希望生成的keystore文件放在其他目录中，可以使用-keystore参数：keytool -genkey -alias tomcat -keyalg RSA -keystore d:\.keystore

**操作图解如下：**



  
 这里生成的.keystore与早期Adnroid开发中打包apk使用的keystore是一样的。

## 配置SSL连接器

* 在Tomcat的server.xml文件中，已经提供了现成的配置SSL连接器的代码，只要把注释去掉即可

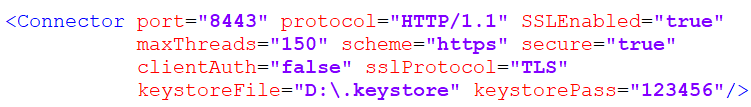
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"

maxThreads="150" scheme="https" secure="true"

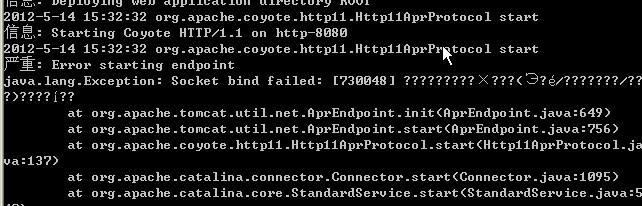
clientAuth="false" sslProtocol="TLS" />

* 在上面的连接器中增加如下两个属性：
  + keystoreFile：指定keystore文件的存放位置，可以指定绝对路经。如果此项没有设定，在默认情况下，Tomcat将从当前操作系统用户的用户名目录下读取名为”.keystore”的文件。
  + keystorePass:指定keystore的密码，如果此项没有设定，在默认情况下，Tomcat将使用”changeit”密码。

**配置后的连接器为：**

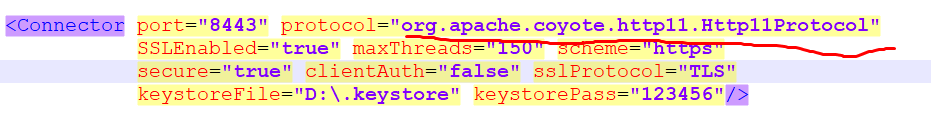


配置好后运行Tomcat，会发现有一个错误，如下：



如果Tomcat用的是安装版本的话就不会有错，在配置tomcat的SSL双向握手中，由于6.0.33版本中默认启用了APR（APR是通过JNI访问的可移植库，可以提高Tomcat的性能和伸缩性），所以采用传统的配置方式会报异常，通过下面的方式解决：

把连接器的protocal值改为：org.apache.coyote.http11.Http11Protocol，如下：



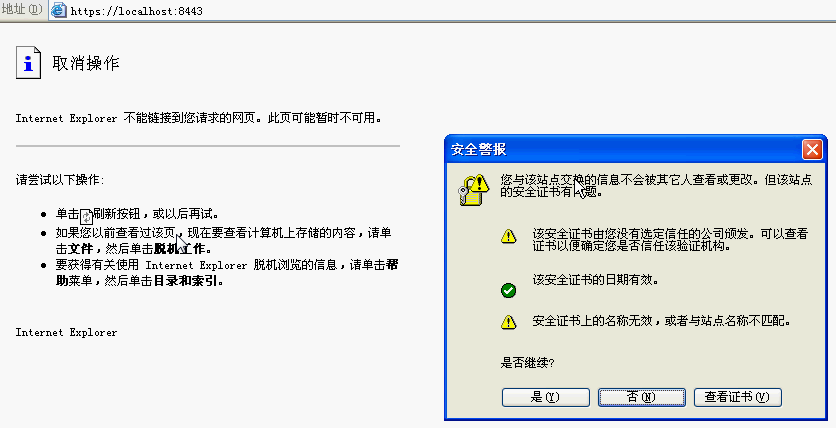
APR给Tomcat的性能提升起到很大的作用，建义配置APR方式。

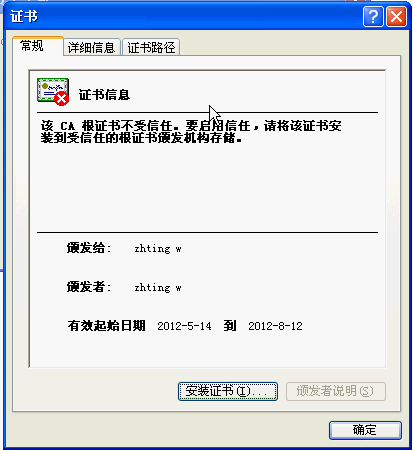
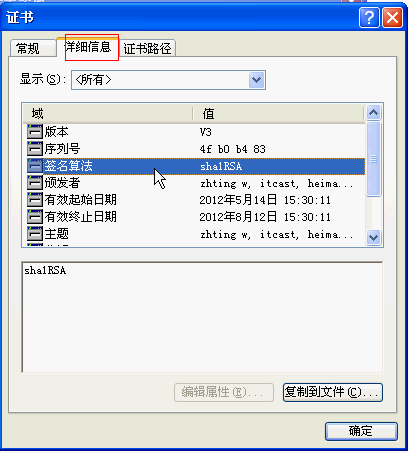
这样配置之后，Tomcat就开启了一个带安全协议的连接器。然后就可以通过下面的方式来访问Tomcat：

[https://localhost:8443](https://localhost:8443/)

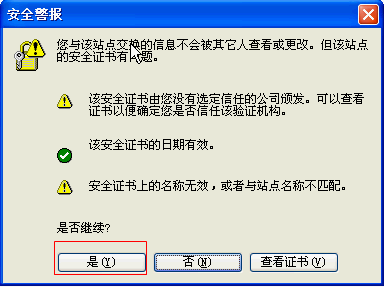
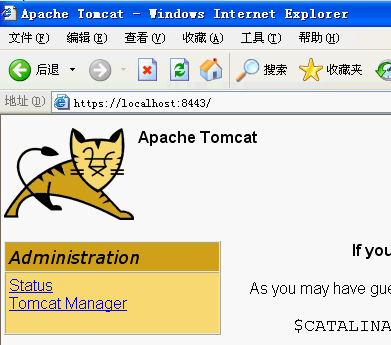
当Tomcat接收到这一HTTPS请求后，会向客户的浏览器发送服务器的安全证书，IE浏览器接收到证书后，将向客户显示安全警报。

这时会跳一个对话框出来，如下（IE6的效果图）：

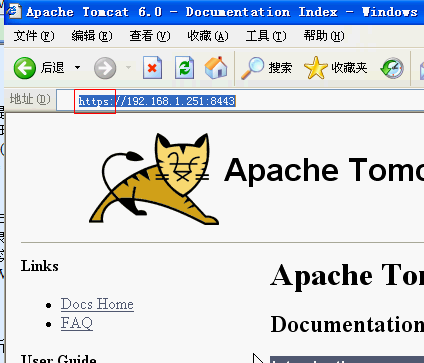
  
 这个对话框其实主要就是说这个网站的证书不是由权威机构颁发的。点击“查看证书(V)”，如下：

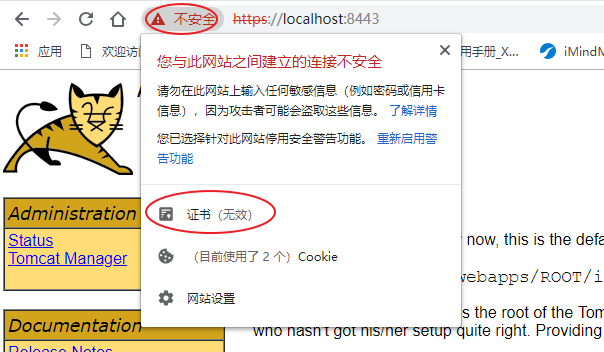
如果信任这个证书那么在和一个对话框中就点“是(Y)”，这样就可以查看网站了，如下：

点是后：

这时我们再点网站的其他链接将都是带安全协议过来的连接，如：



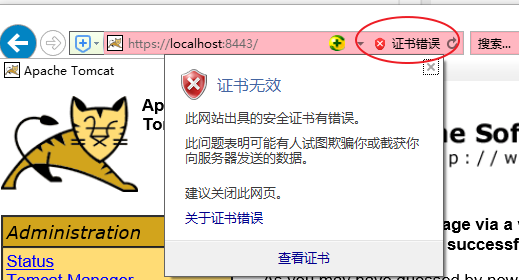
所以，如果是使用https协议访问服务器，服务器会返回一个证书给IE浏览器，浏览器能知道这个证书是否是权威机会颁发的，如果不是权威机构颁发的，则会弹一个警告框给我们进行提示。如果访问的是银行，则银行服务器发给ie浏览器的证书肯定是权威机构颁发的，这浏览器就不会弹出警告

如果用的是非ie浏览器，一般在访问时也会有个警告，有的是弹框，有的是直接显示一个提示页面，点击同意按钮则可继续方法，一般在地址栏有按钮显示证书信息，如Chrome浏览器的地址栏左边有显示证书的按钮，如下：  


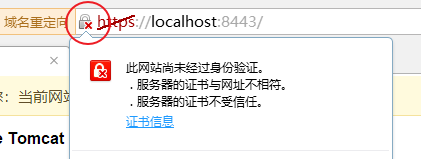
点证书就能看到证书的相关信息了，如下：



在高版本的ie显示如下：

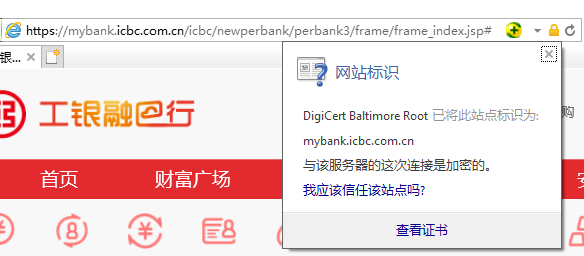


360浏览器：



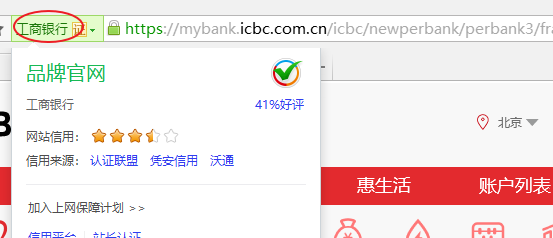
如果证书是权威机会颁发的，则一般会显示一个绿色的锁按钮，如Chrome的：



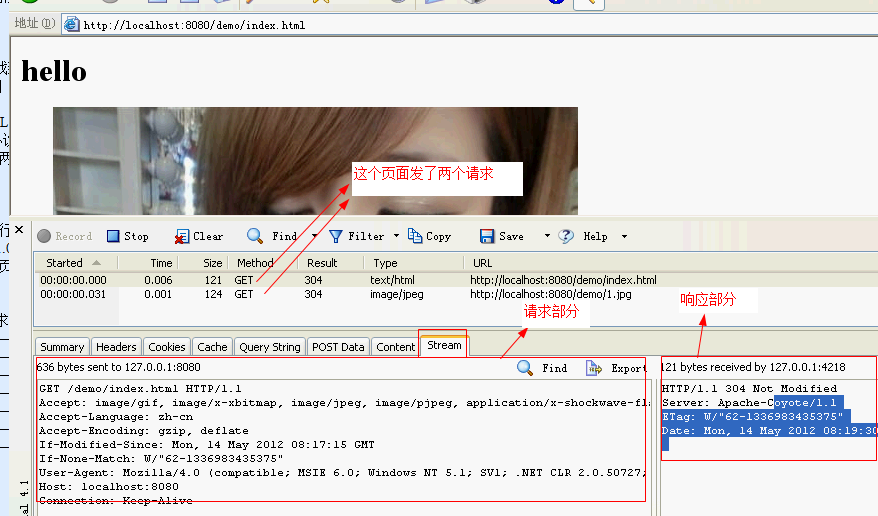
高版本ie：  


360：





# HTTP协议

HTTP协议就包含两部分内容：请求部分、响应部分  


浏览器给服务器请求资源，请求什么，要发送一些请求，这些请求用什么格式，怎么表示，还有返回的数据也用什么格式，怎么表示等，这些就是HTTP协议要规定的内容。

## HTTP简介

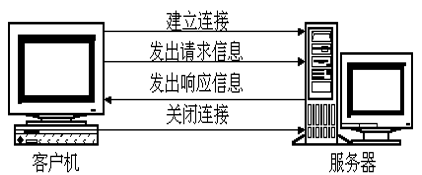
* WEB浏览器与WEB服务器之间的一问一答的交互过程必须遵循一定的规则，这个规则就是HTTP协议。
* HTTP是HyperText Transfer Protocol(超文本传输协议)的简写，它是TCP/IP协议的一个应用层协议，用于定义WEB浏览器与WEB服务器之间交换数据的过程及数据本身的格式。
* HTTP协议的版本

HTTP/1.0、HTTP/1.1

* HTTP协议是学习JavaWEB开发的基石，不深入了解HTTP协议，就不能说掌握了WEB开发，更无法管理和维护一些复杂的WEB站点。

## HTTP1.0的基本运行方式

* 基于HTTP[协议](http://www.cnpaf.net/)的客户/服务器模式的信息交换过程，如图所示，它分四个过程，建立连接、发送请求信息、发送响应信息、关闭连接。

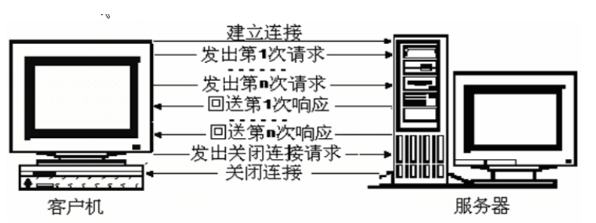
  
 这里的“建立连接”就是TCP连接了。建立连接、发出请求信息、发出响应信息、关闭连接，这样的一个过程就是一个交互过程，而发出的这些信息该怎么描述就是HTTP协议要做的事情了。

* 浏览器与WEB服务器的连接过程是短暂的，每次连接只处理一个请求和响应。对每一个页面的访问，浏览器与WEB服务器都要建立一次单独的连接。
* 浏览器到WEB服务器之间的所有通讯都是完全独立分开的请求和响应。
* HTTP1.0协议有个特点：无状态，就是不保存状态

## HTTP1.1与HTTP1.0的比较

目前一般都是用HTTP1.1协议，用HTTP1.0的比较少了。

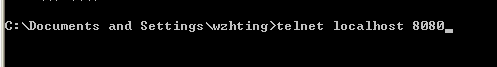
* HTTP1.1的特点
* 在一个TCP连接上可以传送多个HTTP请求和响应。
* 多个请求和响应过程可以重叠
* 增加了更多的请求头和响应头，比如Host、If-Unmodified-Since请求头等



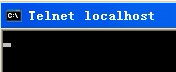
**实验：**

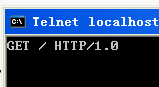
telnet命令可以连接任何的网络的东西：

我们在浏览器浏览：<http://localhost:8080>，改用命令行来访问：

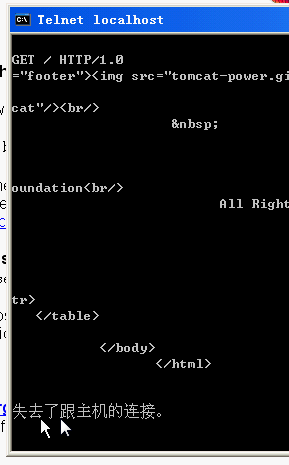


回车后如下：

  
这时打的字母都不能错，因为一按上去就发送出去了。输入如下：

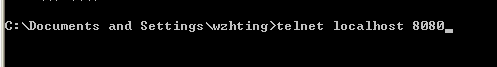


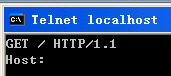
GET表示用GET请求，第一个/表示要请求localhost:8080中的默认页面，用的是HTTP/1.0协议，这里回车后如下：

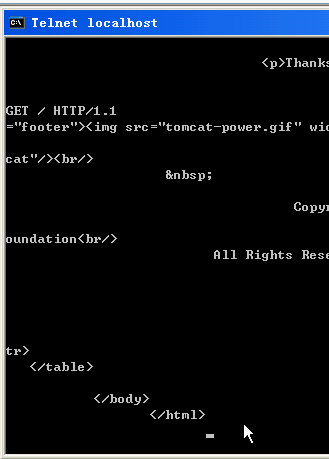


这里可以看到“失去了跟主机的连接”，就是拿到数据后就关闭了连接。这里获取的数据就是html的源代码。浏览器访问<http://localhost:8080>也是获取这些数据，只不过是浏览器获取到后经过解析后用图形化界面显示。

下面再用HTTP1.1来连接







这时可看到并没有和主机失去连接。当长时间没有交互了，服务器就会给断开，以节约资源。

## 浏览器访问多图网页

* 在一个HTML页面中如果包含<img src=”abc.jpg”>标记的话，当浏览器解析到这些标记时，还会向服务器请求访问标记中指定的文件，即再次建立连接并发出HTTP请求。html文件里只能保存文本，而图片、音乐、视频这些都是以超链接的方式访问的，会另外发请求。
* 如果HTML页面中有一个超级链接：

<a href=”http://www.itcast.cn”>itcast</a>，当点击这个链接时，也会触发浏览器与Web服务器开始一次新的HTTP通信。

## HTTP协议的请求与响应的组成

当向服务器发送请求时，会涉及如下内容：

HTTP协议的请求部分组成包含：请求行、请求头、一个空行、请求正文

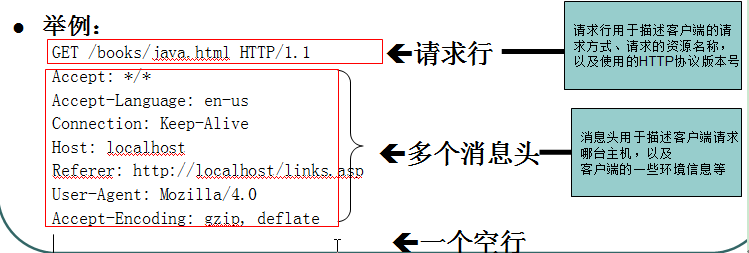
HTTP协议的响应部分组成包含：响应行、响应头、一个空行、响应正文（浏览器就显示响应正文内容）

## HTTP请求消息

客户端连上服务器后，向服务器请求某个web资源，称之为客户端向服务器发送了一个HTTP请求。

一个完整的HTTP请求包括如下内容：

一个请求行、若干消息头、以及请求正文，其中的一些消息头和正文都是可选的，消息头和正文内容之间要用空行隔开。

****

**请求头后有一个空行，空行后是正文，如表单中的数据用POST方式提交时就出现在正文位置，用GET方式提交时出现在请求行。在url地址后加?username=dzl，这种请求也是GET请求。**

**总结起来HTTP请求消息就包含4个内容：请求行、消息头、一个空行、正文**

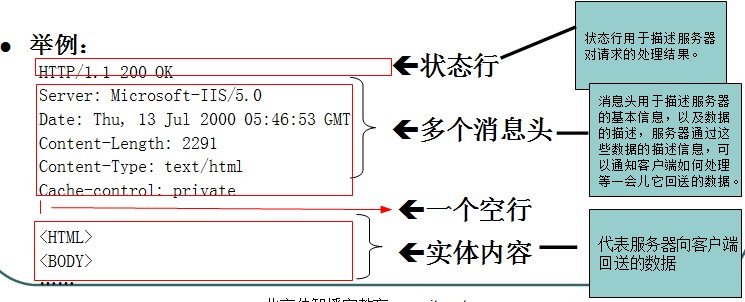
Web服务器通过客户端发送过来的这些请求信息，就可以确定向请求者回送什么资源，以及根据客户端的环境信息采用什么方式进行回送等。

## HTTP响应消息

一个HTTP响应代表服务器向客户端回送的数据。

一个完整的HTTP响应包括如下内容：

一个状态行、若干消息头、以及响应正文，其中的一些消息头和正文都是可选的，消息头和正文内容之间要用空行隔开。



**总结起来HTTP响应消息就包含4个内容：响应行、响应消息头、一个空行、响应正文（浏览器就显示正文内容）**

## HTTP消息头功能说明（请求头和响应头）

* “消息头”即表示请求头，也表示响应头
* 使用消息头，可以实现HTTP客户机与服务器之间的条件请求和应答，消息头相当于服务器和浏览器之间的一些暗号指令。
* 每个消息头包含一个头字段名称，然后依次是冒号、空格、值（多个值用逗号隔开）、回车和换行符（注：这里的空格可以没有，但是最好加上），示例如下：  
    
  注：这个空格可以没有，但是最好加上。取值多个值用逗号隔开，逗号后面也有一个空隔，也可以没有，但是最好有。
* 消息头字段名是不区分大小写的，但习惯上将单词的第一个字母大写，多个单词则每个单词开头字母都大写，单词之间用-连接，如：
* 整个消息头中的各行消息头可按任何顺序排列。
* 消息头又可分为通用信息头、请求头、响应头、实体头等四类。通用信息头的意思是，这个消息头既可用于请求头，也可用于响应头。实体头很少用到，老师也不知道干嘛用的。
* 许多请求头字段都允许客户端在值部分指定多个可接受的选项，多个选项之间以逗号分隔。
* 有些头字段可以出现多次，例如，响应消息中可以包含有多个”Warning”头字段。

## HTTP请求的细节——请求行

请求行位于请求部分的第一行

默认的请求是GET请求。如表单不指定则用GET请求，也可通过method属性指定为POST请求

请求行：GET /cache/global/img/gs.gif HTTP/1.1

如上，请求行就包括3个部分：请求方式、请求的资源路径、使用的协议

* 请求行
  + 格式：*请求方式 资源路径 HTTP版本号<CRLF>*
  + 举例：GET /index.html HTTP/1.1
  + 请求方式：GET、POST、HEAD、OPTIONS、DELETE、TRACE、PUT
  + 用户如没有设置，默认情况下浏览器向服务器发送的都是get请求，例如在浏览器直接输地址访问，点超链接访问等都是get，用户如想把请求方式改为post，可通过更改表单的提交方式实现。
* GET方式
  + 如请求方式为GET方式，则可以在请求的URL地址后以?的形式带上参数，然后提交给服务器的数据，多个数据之间以&进行分隔，例如：  
    GET /mail/login.html?name=abc&password=xyz HTTP/1.1
  + GET方式的特点：在URL地址后附带的参数是有限制的，其数据容量通常不能超过1K。
* POST方式
  + 如请求方式为POST方式，则可以在请求的正文内容中向服务器发送数据，Post方式的特点：传送的数据量无限制。

## GET请求与POST请求优缺点

GET请求缺点：

* 不安全，把请求参数显示到url上
* 长度有1K限制

POST 请求优点：

* 安全，把请求参数显示到请求的正文里。 这里说的安全也只是针对普通人。对于我们懂技术的，也没安全可言
* 没有长度限制

不管POST或GET，都用于向服务器请求某个WEB资源，这两种方式的区别主要表现在数据传递的方式不同。比如用户名与密码，可以用get请求提交，也可用Post请求提交。表单提交数据时默认也是Get方式提交的 ，这样会把数据显示到url上面，所以可以指定用post提交。示例如下：  
写一个a.html和b.html，b.html的内容随意写，a.html内容如下：  
 注：name属性用于指定一个变量，用指定的变量保存输入的值，如果不指定name，则没法接收输入的值。  
访问a.html如下：

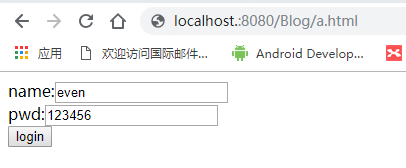
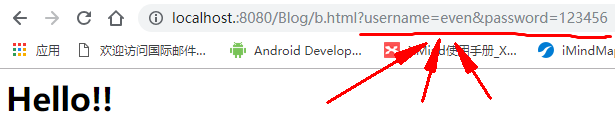
<form action="b.html">

name:<input type="text" name="username"/><br/>

pwd:<input type="text" name="password"/><br/>

<input type="submit" value="login"/>

</form>

  
点击login按钮会跳转到b.html，如下：  


用户名与密码都显示到了地址栏上，说明表单默认是get请求，修改表单的请求方式如下：

<form action="b.html" method="post">  
这时再访问a.html时就是使用POST方式请求了，注：这里说的是使用post方请求b.html页面，并不是a.html，a.html还是get请求方式的。

## HTTP响应的细节——响应行

请求行位于响应部分的第一行

响应行：HTTP/1.1 200 OK

响应行就包括3个部分：协议版本号、响应状态码、状态码描述

* **响应行**

**格式：** *HTTP版本号　状态码　原因叙述<CRLF>*

**举例：***HTTP/1.1 200 OK*

* 状态码用于表示服务器对请求的各种不同处理结果和状态，它是一个三位的十进制数。响应状态码分为5类，使用最高位为1到5来进行分类如下所示：

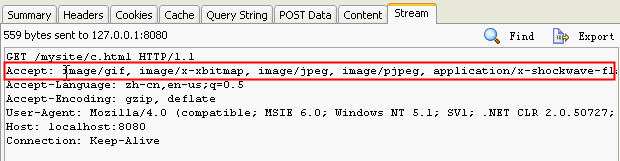
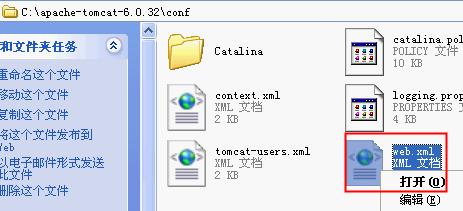
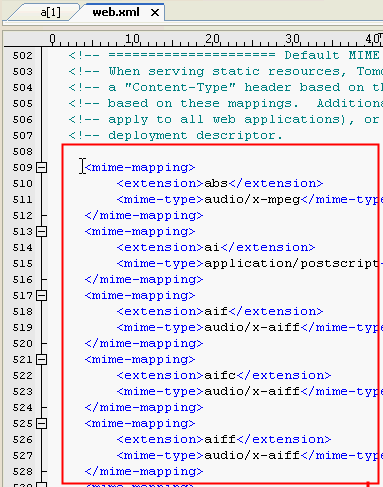
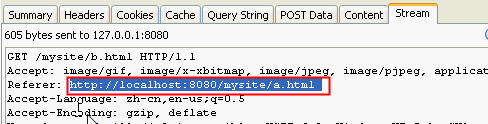
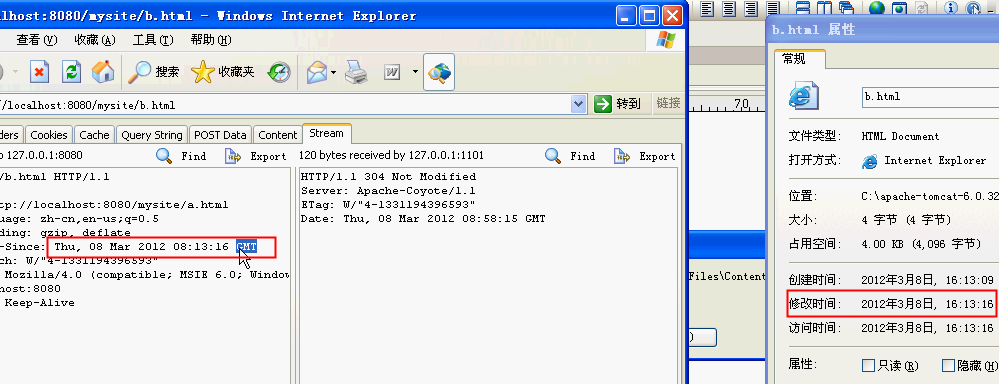
|  |  |
| --- | --- |
| **状态码** | **含义** |
| 100～199 | 表示成功接收请求，要求客户端继续提交下一次请求才能完成整个处理过程 |
| 200～299 | 表示成功接收请求并已完成整个处理过程 |
| 300～399 | 为完成请求，客户需进一步细化请求。例如，请求的资源已经移动到一个新地址 |
| 400～499 | 客户端的请求有错误 |
| 500～599 | 服务器端出现错误 |

这里一共有500个状态码，但是有470个基本没用，是保留的，能用的就二三十个。

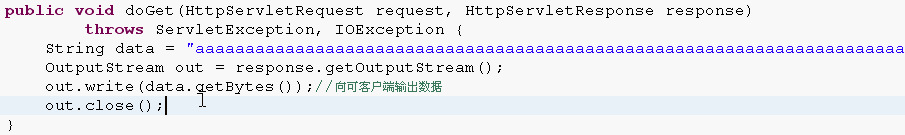
## HTTP响应的细节——常用状态码

* 200(正常): 表示一切正常，返回的是正常请求结果
* 302/307(临时重定向) : 指出被请求的文档已被临时移动到别处，此文档的新的URL在Location响应头中给出。举例：比如访问a，a说我现在不处理这个事情了，去找b吧，现在由b处理这个请求。现实生活中，张三找四李借钱，李四说没钱，王五有钱，去找王五借，于是张三就转而去找王五借钱了。
* 304(未修改) : 表示客户机缓存的版本是最新的，客户机可以继续使用它，无需到服务器请求。
* 404(找不到) : 服务器上不存在客户机所请求的资源。
* 500(服务器内部错误) : 服务器端的程序发生错误

## HTTP请求的细节——常用请求头

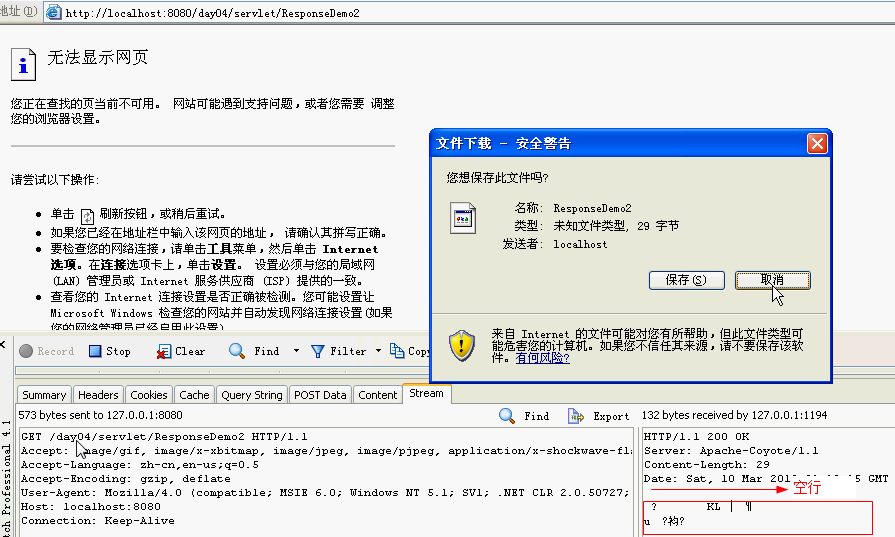
* 请求头字段用于客户端在请求消息中向服务器传递附加信息，主要包括客户端可以接受的数据类型(MIME类型)、压缩方法、语言以及发出请求的超链接所属页面的URL地址等信息。
* 常用请求头：
  + Accept：浏览器可接受的数据类型（即MIME类型 ）  
    这是告诉服务器，我可以解析这些类型的文件，如httpWatch截图如下：  
      
    注：有时候会看到有类型是：\*/\*，这表示什么类型的文件都能解析  
    如，我们的\*.txt文件，在window系统是.txt表示纯文本文件，可是在网络上就不能这么表示了，应该如下：  
    text/plain 文本/纯文本  
    text/html 文本/超级文本  
    text/css css样式文本  
    text/javascript 脚本文本  
    text/xml xml文本  
    text/json json文本  
    关于MIME类型，可以从这里查看：  
      
    
  + Accept-Charset: 浏览器通过这个头告诉服务器，它支持哪种字符集
  + Accept-Encoding:浏览器能够进行解码的数据编码方式，比如gzip ，这样服务器知道你能解压这种压缩，那服务器就可以把数据压缩成这种格式后发过来，这样就节省容量、带宽，加快速度。
  + Accept-Language:浏览器所希望的语言种类，当服务器能够提供一种以上的语言版本时要用到。 可以在浏览器中进行设置。
  + Host:显示URL中的主机和端口
  + Referer:包含一个URL，用户从该URL代表的页面出发访问当前请求的页面，图片防盗链就是通过这个请求头来实现的  
      
    如上图HttpWatch的截图，表示这个请求是从a.html网页中点击的某个链接过来的，这样能知道你是不是盗链的。  
      
    该请求头还有一个作用是：做广告时查看广告是从哪个网站连接过来的，如在百度放了广告，在CSDN放了广告，通过这个请求头就能知道用户是从哪个网站推广过来的。如看到百度转过来的很多，CSDN的很少，那么CSDN的就可以不做了，节省钱。
  + Content-Type:内容类型，这就是请求的时候要发正文，而正文的类型是什么就用这个请求头表示
  + If-Modified-Since: Wed, 02 Feb 2011 12:04:56 GMT利用这个头与服务器的文件进行比对，如果一致，则从缓存中直接读取文件。  
      
    这里可以看到请求里带上了缓存文件的修改时间，右边是服务器中文件的修改时间，这里看起来不一样，其实是一样的，请求那边用的是GMT时间，这和右边是刚好相差8个小时的（慢8小时），这个服务器自动会处理后再对比，我们不需要去转换。
  + User-Agent:浏览器类型.，即告诉服务器客户端用的是什么浏览器
  + Content-Length:表示请求消息正文的长度，如果请求没有发送正文，那么这个头也不会出现，因为没有意义
  + Connection:表示是否需要持久连接。如果服务器看到这里的值为“Keep -Alive”，或者看到请求使用的是HTTP 1.1（HTTP 1.1默认进行持久连接
  + Cookie:这是最重要的请求头信息之一，需要和Session（会话）一起讲
  + Date：Date: Mon, 22 Aug 2011 01:55:39 GMT 请求时间，即请求时的当前时间，是GMT时间，不是北京时间，这个时间比北京时间慢8小时

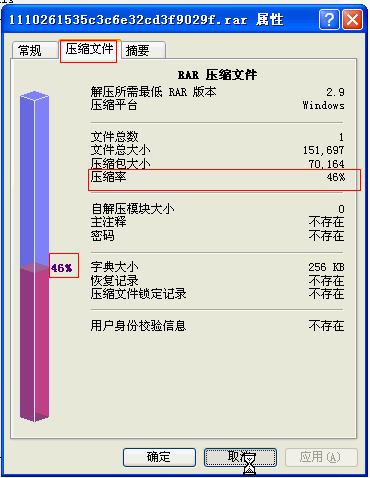
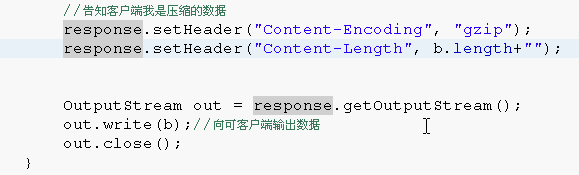
## HTTP响应的细节——常用响应头

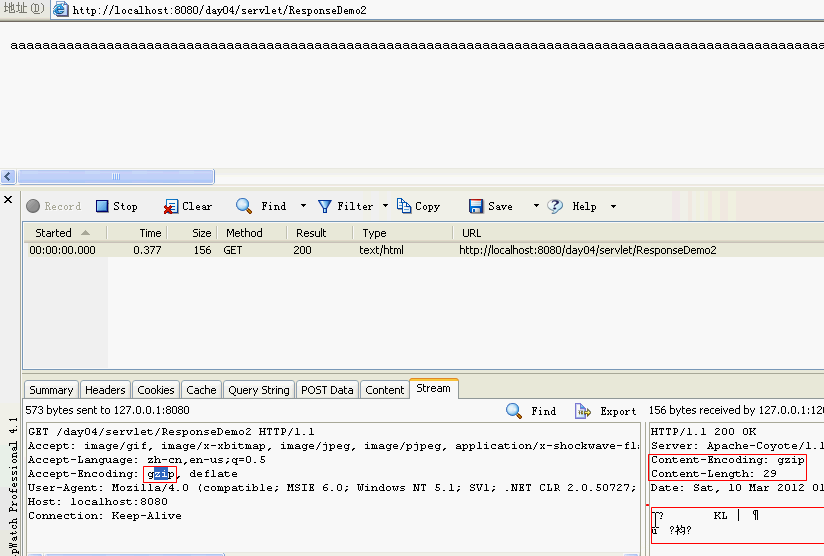
* **响应头字段用于向客户端传递附加信息**
* **常用响应头**
  + Location: http://www.it315.org/index.jsp指示新的资源的位置  
    这是要配合状态码307一起使用的，状态码307表示要把请求转到其他页面，转到哪个页面呢就通过Location响应头来指定，如下：  
    
  + Server:apache tomcat指示服务器的类型，即告诉客户端服务器用的是什么类型服务器
  + Content-Encoding: gzip服务器发送的数据采用的编码类型，如果是浏览器接收会自动解压数据。如：  
      
    这里是把字符串无压缩就输出了，通过下面方式是压缩后再输出，这样传送时容量小、速度快：  
    

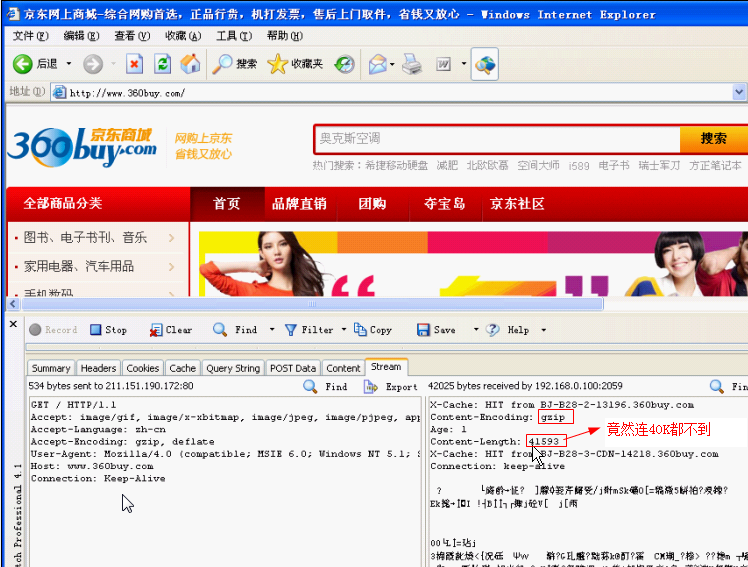
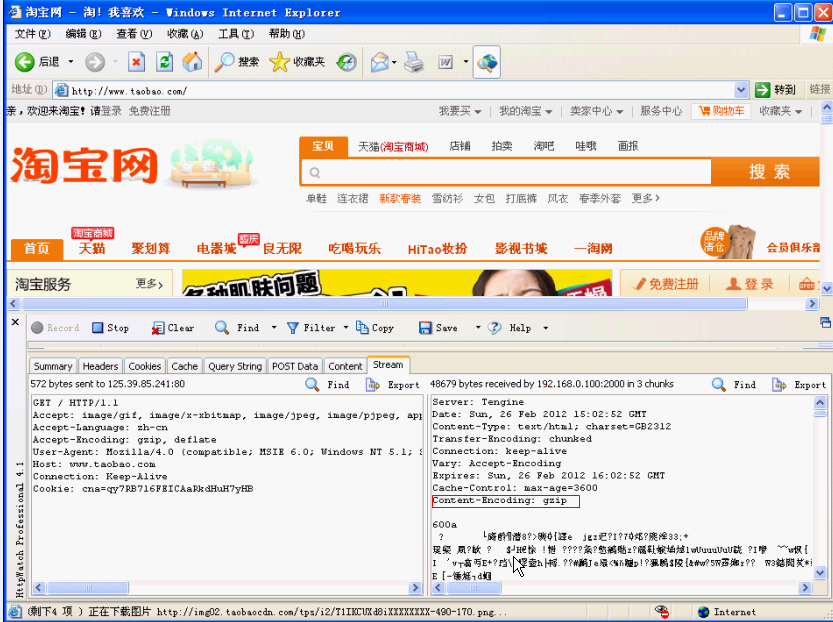
GZI压缩是通过GZIPOutputStream类来压缩的，而这个类需要接收一个输出流，GZIPOutputStream把数据压缩后存到哪去了呢？就是存到构造器接口的输出流里了，上面我们为什么要使用ByteArrayOutputStream输出流呢？因为通过这个输出流的.toByteArray，就可以把压缩后的数据获取到。

我们访问Servlet，如下：

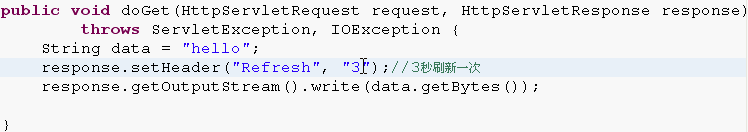
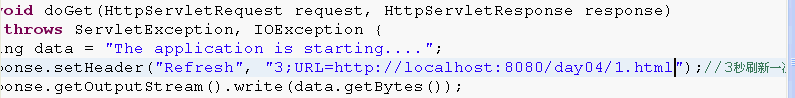
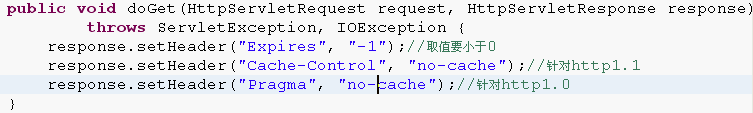


发现要下载文件，点取消即可，我们看到HttpWatch中的响应正文并不是我们的aaaaaaaa的内容，因为这是压缩后的内容，再看myeclipse输出台：  
，这里可以看到文本的压缩比率是非常高的，我们还可能通过WinRAR去压缩一个文本文件来看压缩后确实变的小很多：  
  
  
为什么上面访问Servlet时出现保存文件，而不是在页面中显示aaaaaa呢？答：因为客户端不知道这些数据是压缩后的数据，所以在数据输出之前就得先设置响应头来告诉户端这些数据是压缩后的数据，如下：  


再访问Servlet效果如下：  


注：这必须得客户端支持gzip解压才能正确显示压缩的数据。  
我们可以查看一些大型网站是否使用了gzip压缩：  
  
  


可看到淘宝网没有发长度的响应头过来，但是数据也是使用gzip压缩后的数据。  
所以一般的大型网站都会通过gzip压缩后再输出的。

* + Content-Length: 80 告诉浏览器正文的长度
  + Content-Language: zh-cn 服务器发送的文本的语言，这个语言就有中文、英文等，方便浏览器使用对应的语言来展示界面。
  + Content-Type: text/html; charset=GB2312 服务器发送的内容的MIME类型，这里告诉了客户端用的是文本文件，既然是文本那还得告诉它这个文本用的字符编码
  + Last-Modified: Tue, 11 Jul 2000 18:23:51 GMT 文件的最后修改时间，客户端会保存这个时间，在下一次访问时，客户端就会使用If-Modified-Since请求头把这个时间带上传给服务器。这是我自己猜的，具体是不是不清楚哦
  + Refresh: 1;url=http://www.it315.org 指示客户端刷新频率。单位是秒，如下：  
    **刷新自己：**  
      
      
    **刷新到其他页面去：**  
      
    这可用于注册，注册完毕后显示3秒后自动跳转
  + Content-Disposition: attachment;filename=aaa.zip 指示客户端下载文件，演示如下：  
    
  + Set-Cookie:SS=Q0=5Lb\_nQ; path=/search 服务器端发送的Cookie
  + **Expires: -1**
  + **Cache-Control: no-cache (1.1)**
  + **Pragma: no-cache (1.0)**这3个响应头（Expires、Cache-Control、Pragma）共同来完成不缓存资源，如验证码图片就不需要缓存，这里**Expires的值可以为0，也可以为-1，都能设置无缓存，因为不知道客户端用的是什么版本的协议，所以Cache-Control: no-cache**是给Http 1.1协议用，**Pragma: no-cache** 是给Http 1.0协议用，如下设置：  
    
  + Connection: close/Keep-Alive 让客户端是否一直保持连接，这个一般不用管
  + Date: Tue, 11 Jul 2000 18:23:51 GMT 服务器返回数据那一刻的当前时间，这个是GMT时间，要比北京时间晚8个小时

## 作业

* 请写一篇关于HTTP协议的笔记，要求：
  + 描述清楚HTTP请求头、响应头的格式
  + 请求头和响应头中各个头字段的含义
* 如果浏览器传递给WEB服务器的参数内容超过1K，应该使用那种方式发送请求消息？
* 请描述200、302、304、404和500等响应状态码所表示的意义。
* 请列举三种禁止浏览器缓存的头字段，并写出相应的设置值。