# 基本概念

## WEB开发的相关知识

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。

动态网页并非是那些带有动画效果的网页，而是指具有交互性、内容可以自动更新，并且内容会根据访问的时间和访问者而改变的网页。这里所说的交互性是指网页可以根据用户的要求动态改变或响应。

### 网络程序开发的体系结构

C/S 就像是桌面 QQ 等一些**运行在桌面的程序，**，在**服务端主要就是一个数据库，把所有业务逻辑以及界面的渲染操作交给客户端**去完成。而 B/S 就是我们的浏览器，**把业务逻辑交给服务端完成，客户端仅仅只做界面渲染和数据交换。**

B/S 架构带来了以下两个方面的好处：

* **客户端使用同一的浏览器（ Browser ）。**
* **服务端（ Server ）基于统一的 HTTP 。**

传统的 C/S 架构使用自定义的应用层协议不同，B/S 价格使用的都是统一的 HTTP。使用同一的 HTTP 也为服务提供商简化了开发模式，使得服务器开发者可以采用相对规范的开发模式，这样可以大大节省开发成本。

B/S 网络架构从前端到后端都得到了简化，基于统一的应用层协议 HTTP 来交互数据，与大多数传统 C/S 互联网应用程序采用的长连接的交互模式不同，**HTTP 采用无状态的短连接的通信方式。**

基于 HTTP 本身的特点，目前的 B/S 网络架构大多采用[**CDN**](https://baike.baidu.com/item/CDN/420951)的架构设计

## WEB应用程序

多个web资源用于对外提供服务，此时应把这多个web资源放在一个目录中，以组成一个web应用（或web应用程序）  
　　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、配置文件等等。

**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。**

## 静态与动态WEB

* **静态WEB中存在以下几个缺点：**

**1、Web页面中的内容无法动态更新，所有的用户每时每刻看见的内容和最终效果都是一样的。**

实现静态WEB客户端动态效果的手段：

* JavaScript
* VBScript

**2、静态WEB无法连接数据库，无法实现和用户的交互。**

* **动态WEB**中，程序依然使用客户端和服务端，客户端依然使用浏览器，使用HTTP协议发起请求（Request），现在的所有请求都先经过一个**WEB Server Plugin（服务器插件）**来处理，此插件用于区分是请求的是静态资源(\*.htm或者是\*.htm)还是动态资源。

客户端请求的是静态资源(\*.htm或者是\*.htm)，则将**请求直接转交给WEB服务器**，之后WEB服务器从文件系统中取出内容，发送回客户端浏览器进行解析执行。

客户端请求的是动态资源（\*.jsp、\*.asp/\*.aspx、\*.php），则先将请求转交给**WEB Container，在WEB Container中连接数据库，从数据库中取出数据等一系列操作后动态拼凑页面的展示内容**，拼凑页面的展示内容后，把所有的展示内容交给WEB服务器，之后通过WEB服务器将内容发送回客户端浏览器进行解析执行。

动态WEB现在的实现手段非常多：

* Microsoft ASP、ASP.NET
* PHP
* JAVA Servlet/JSP

SUN公司最早推出的WEB技术推出的是Servlet程序，Servlet程序本身使用的时候有一些问题，**所有的程序是采用JAVA代码+HTML的方式编写的**，即，要使用JAVA输出语句，一行一行地输出所有的HTML代码，之后，SUN公司受到了ASP的启发，**发展出了JSP(Java Server Page)**

## WEB服务器

Web服务器是指驻留于因特网上某种类型计算机的程序，是可以向发出请求的浏览器提供文档的**程序**。

服务器是一种**被动程序**：只有当Internet上运行在其他计算机中的浏览器发出请求时，服务器才会响应。

### 常见的Web服务器介绍

* WebLogic是美国bea公司出品的一个application server。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的Java应用服务器。
* WebSphere Application Server 是一种功能完善、开放的Web应用程序服务器，是IBM公司电子商务计划的核心部分，它是基于 Java 的应用环境，用于建立、部署和管理 Internet 和 Intranet Web 应用程序。
* TomcatTomcat是一个实现了JAVA EE标准的最小的WEB服务器，是Apache 软件基金会的Jakarta 项目中的一个核心项目，由Apache、Sun 和其他一些公司及个人共同开发而成。因为Tomcat 技术先进、性能稳定，而且开源免费。
* Microsoft的Web服务器产品为Internet Information Services （IIS），IIS 是允许在公共Intranet或Internet上发布信息的Web服务器。

### 浏览器与服务器交互的过程

当我们打开浏览器，输入URL地址"http://www.gacl.cn:8080/JavaWebDemo1/1.jsp"去访问服务器上的1.jsp这个web资源的过程中，浏览器和服务器都做了神马操作呢，我们是怎么在浏览器里面看到1.jsp这个web资源里面的内容的呢？

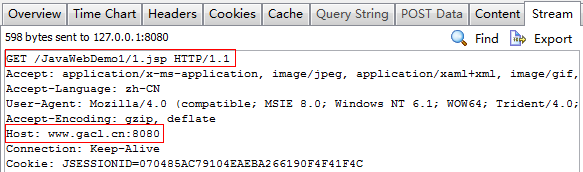
浏览器和服务器做了以下几个操作：

　　1、浏览器根据主机名"www.gacl.cn"去操作系统的Hosts文件中查找主机名对应的IP地址。

　　2、浏览器如果在操作系统的Hosts文件中没有找到对应的IP地址，就去互联网上的DNS服务器上查找"www.gacl.cn"这台主机对应的IP地址。

　　3、浏览器查找到"www.gacl.cn"这台主机对应的IP地址后，就使用IP地址连接到Web服务器。

　　4、浏览器连接到web服务器后，就使用http协议向服务器发送请求，发送请求的过程中，浏览器会向Web服务器以Stream(流)的形式传输数据，告诉Web服务器要访问服务器里面的哪个Web应用下的Web资源，如下图所示：



　　这就是浏览器向Web服务器发请求时向服务器传输的数据，解释一下"**GET /JavaWebDemo1/1.jsp HTTP/1.1**"这里面的内容，

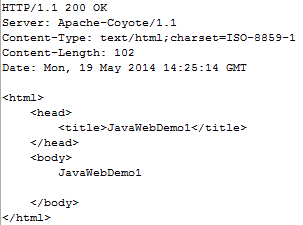
**GET**：告诉Web服务器，浏览器是以GET的方式向服务器发请求。

**/JavaWebDemo1/1.jsp**：告诉Web服务器，浏览器要访问**JavaWebDemo1**应用里面的**1.jsp**这个Web资源。

**HTTP/1.1：**告诉Web服务器，浏览器是以HTTP协议请求的，使用的是1.1的版本。

　　5、浏览器做完上面4步工作后，就开始等待，等待Web服务器把自己想要访问的**1.jsp**这个Web资源传输给它。

　　6、服务器接收到浏览器传输的数据后，开始解析接收到的数据，服务器解析"**GET /JavaWebDemo1/1.jsp HTTP/1.1**"里面的内容时知道客户端浏览器要访问的是**JavaWebDemo1**应用里面的**1.jsp**这个Web资源，然后服务器就去读取**1.jsp**这个Web资源里面的内容，将读到的内容再以Stream(流)的形式传输给浏览器，如下图所示：



　　这个就是Web服务器传输给浏览器的数据。

7、浏览器拿到服务器传输给它的数据之后，就可以把数据展现给用户看

# Tomcat

## Tomcat

在本地计算机上随便创建一个web页面，用户是无法访问到的，但是如果启动tomcat服务器，**把web页面放在tomcat服务器中**，用户就可以访问了。这说明什么问题？  
　　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，We应用的开发者只需要关注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

**Tomcat优化经验：**

* 1. 关掉对web.xml的监视，把jsp提前编辑成Servlet。有富余物理内存的情况，加大tomcat使用的jvm的内存；
  2. 服务器资源。服务器所能提供CPU、内存、硬盘的性能对处理能力有决定性影响。
  3. 利用缓存和压缩：对于静态页面最好是能够缓存起来，这样就不必每次从磁盘上读。这里我们采用了**Nginx作为缓存服务器**，将图片、css、js文件都进行了缓存，有效的减少了后端tomcat的访问。开启gzip压缩也是必不可少的。但考虑到tomcat已经需要处理很多东西了，所以把这个压缩的工作就交给前端的Nginx来完成。
  4. 采用集群：**实现横向扩展**，那么**组建tomcat集群**
  5. 优化tomcat参数：这里以tomcat7的参数配置为例，需要修改conf/server.xml文件，主要是**优化连接配置**，关闭客户端dns查询。

## Tomcat运行原理

**Tomcat依赖<CATALINA\_HOME>/conf/server.xml这个配置文件启动server（一个Tomcat实例，核心就是启动容器Catalina）。**

**server.xml文件:该文件用于配置和server相关的信息，比如tomcat启动的端口号、配置host主机、配置Context**

**web.xml文件：部署描述文件，这个web.xml中描述了一些默认的servlet，部署每个webapp时，都会调用这个文件，配置该web应用的默认servlet。**

**Tomcat部署Webapp时，依赖context.xml和web.xml（<CATALINA\_HOME>/conf/目录下的context.xml和web.xml在部署任何webapp时都会启动，他们定义一些默认行为，而具体每个webapp的  META-INF/context.xml  和  WEB-INF/web.xml  则定义了每个webapp特定的行为）两个配置文件部署web应用。**

1.Tomcat是运行在JVM中的一个进程。**它定义为【中间件】**，顾名思义，是一个在Java项目与JVM之间的中间容器。

2.Web项目的本质，是一大堆的资源文件和方法。Web项目没有入口方法(main方法)，，意味着Web项目中的方法不会自动运行起来。

3.Web项目部署进Tomcat的webapp中的目的是很明确的，那就是希望Tomcat去调用

写好的方法去为客户端返回需要的资源和数据。

4. Tomcat可以运行起来，并调用写好的方法。那么，Tomcat一定有一个main方法。

5. 对于Tomcat而言，它并不知道我们会有什么样的方法，这些都只是在项目被部署进webapp下后才确定的，由此分析，必然用到了Java的反射来实现类的动态加载、实例化、获取方法、调用方法。但是我们部署到Tomcat的中的Web项目必须是按照规定好的接口来进行编写，以便进行调用。

### Tomcat启动过程

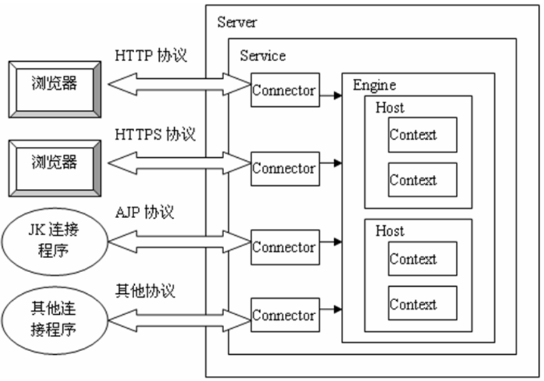
【Tomcat的启动过程】Tomcat 先根据**/conf/server.xml 下的配置启动Server**，再加载Service，对于与Engine相匹配的Host，每个Host 下面都有一个或多个Context。

　　注意：Context 既可配置在server.xml 下，也可配置成一单独的文件，放在conf\Catalina\localhost 下，简称应用配置文件。

开启Tomcat：可以在IDE中启动Tomcat的服务器，也可以手动在<CATALINA\_HOME>/bin/目录下找到startup.bat并双击

1. 引导（Bootstrap）启动：调用了org.apache.catalina.startup.Bootstrap.class中的main方法，开始启动**Tomcat容器**；
2. 调用Bootstrap中的init（），创建了Catalina对象（**核心容器**）
3. 调用Bootstrap中的load（）：实际上是通过反射调用了catalina的load方法。（找到config file（server.xml）；然后创建digester，解析server.xml，生成各组件对象（Server、Service、Container、Connector等）以及建立相互之间的关系。
4. 最后start()，同样是在Bootstrap中通过反射调用catalina对象的start方法。接着启动server.start()方法，接下来是一系列的container的start
5. 使用类CatalinaShutdownHook实现，它继承Thread，run中进行清理

## Tomcat的体系结构



**Jasper是Tomcat的JSP引擎，**它解析JSP文件，将它们编译成JAVA代码作为servlet。在运行时，Jasper允许自动检测JSP文件的更改并重新编译它们

**AJP连接器:**它以与HTTP连接器相同的方式工作，但是他们使用的是HTTP的AJP协议。

1. A Server element represents the entire Catalina servlet container. (Singleton)
2. **Service是这样一个集合：它由一个或者多个Connector组成，以及一个Engine**，负责处理所有Connector所获得的客户请求
3. 一个Connector将在某个指定端口上侦听客户请求，并将获得的请求交给Engine来处理，从Engine处获得回应并返回客户
4. 当Engine获得一个请求时，它把该请求**匹配到某个Host上**，然后把该请求交给该Host来处理  
   Engine有一个默认虚拟主机，当请求无法匹配到任何一个Host上的时候，将交给该默认Host来处理
5. 代表一个Virtual Host，虚拟主机，每个虚拟主机和某个网络域名Domain Name相匹配  
   每个虚拟主机下都可以部署(deploy)一个或者多个Web App，每个Web App对应于一个Context，有一个Context path。当Host获得一个请求时，将把该请求匹配到某个Context上，然后把该请求交给该Context来处理  
   匹配的方法是“最长匹配”，所以一个path==”"的Context将成为该Host的默认Context  
   所有无法和其它Context的路径名匹配的请求都将最终和该默认Context匹配
6. 一个Context对应于一个Web Application，一个Web Application由一个或者多个Servlet组成  
   Context在创建的时候将根据配置文件$CATALINA\_HOME/conf/web.xml和$WEBAPP\_HOME/WEB-INF/web.xml载入Servlet类。当Context获得请求时，将在自己的映射表(mapping table)中寻找相匹配的Servlet类  
   如果找到，则执行该类，获得请求的回应，并返回

Tomcat服务器的启动是基于一个**server.xml文件**的，Tomcat启动的时候首先会启动一个Server，Server里面就会启动Service，**Service里面就会启动多个"Connector(连接器)**"，每一个连接器都在等待客户机的连接，当有用户使用浏览器去访问服务器上面的web资源时，首先是连接到Connector(连接器)，Connector(连接器)是不处理用户的请求的，而是将用户的请求交给**一个Engine(引擎)、** **Container**去处理，Engine(引擎)接收到请求后就会解析用户想要访问的Host，然后将请求交给相应的Host，Host收到请求后就会解析出用户想要访问这个Host下面的哪一个Web应用,一个web应用对应一个Context。

<?xml version='1.0' encoding='utf-8'?>

<Server port="8005" shutdown="SHUTDOWN">

<Service name="Catalina">

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

connectionTimeout="20000"

redirectPort="8443" />

<Connector port="8443" protocol="org.apache.coyote.http11.Http11Protocol"

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

clientAuth="false" sslProtocol="TLS"

keystoreFile="conf/.keystore" keystorePass="123456"/>

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

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

<Host name="localhost" appBase="webapps"

unpackWARs="true" autoDeploy="true">

<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"

prefix="localhost\_access\_log." suffix=".txt"

pattern="%h %l %u %t &quot;%r&quot; %s %b" />

</Host>

<Host name="www.gacl.cn" appBase="F:\JavaWebApps">

<Context path="" docBase="F:\JavaWebApps\JavaWebDemo1"/>

</Host>

</Engine>

</Service>

</Server>

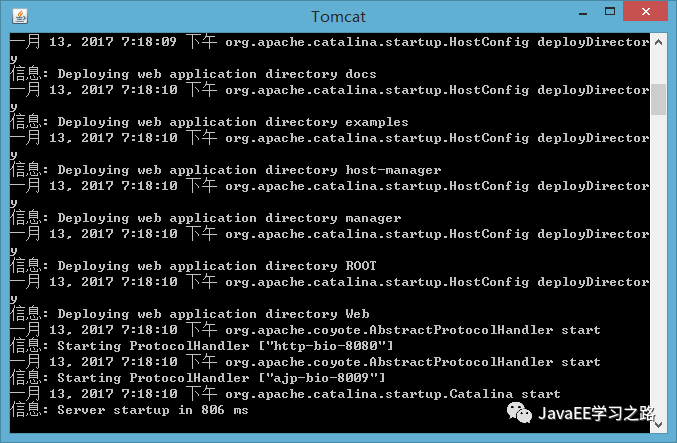
## 打包JavaWeb应用

* 在Java中，使用"jar"命令来对将JavaWeb应用打包成一个War包。
* Eclipse 直接有
* Maven package

## 配置Tomcat

运行Tomcat需要JDK的支持【Tomcat会通过JAVA\_HOME找到所需要的JDK】。

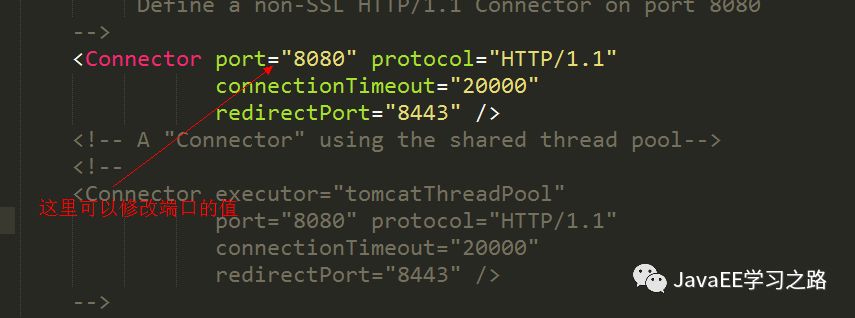
* **新建JAVA\_HOME环境变量**【路径是JDK的主目录】

****

* **进入Tomcat目录下的bin中启动startup.bat**，以下是成功启动Tomcat的页面。
* 在浏览器地址栏输入http://localhost:8080,如果**能够出现Tomcat页面，说明配置成功了**！

**注意：**如果在使用Tomcat时出现了Error deploying web application directory web222,很大原因就是jdk和tomcat的版本不匹配。目前，**jdk8不能匹配tomcat7.0**，所以**要降低jdk版本**【换成JDK7】

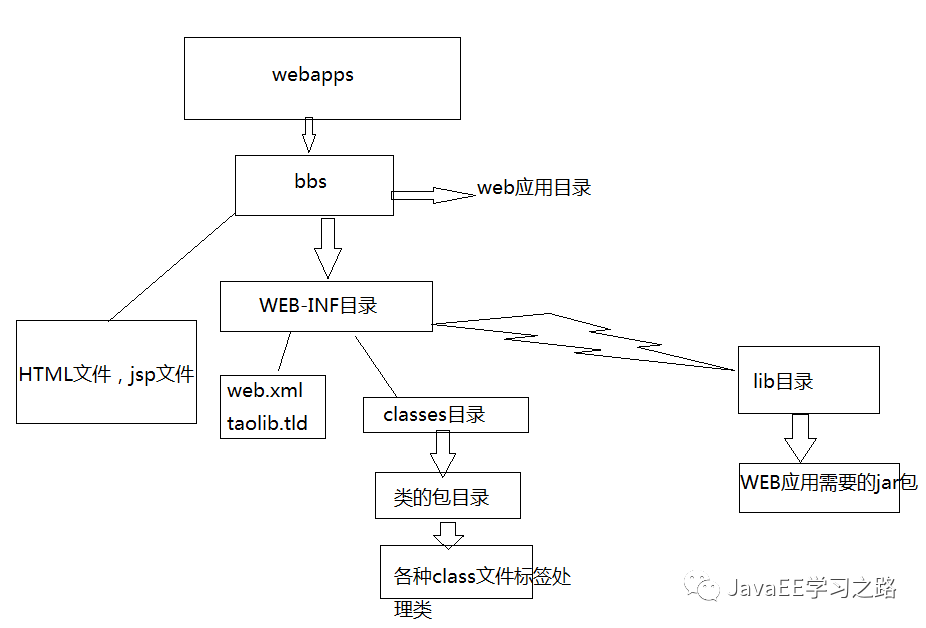
当然也有可能不是一帆风顺能够直接启动Tomcat，**不能正常启动Tomcat会有以下情况**

1. JAVA\_HOME配置错误
2. 端口被占用了
   * 第一种方法：在cmd输入netstat -anb查看谁占用了该端口【**Tomcat默认的是8080】**,在进程中关闭它
   * 第二种方法：主动改变tomcat的端口,**到tomcat主目录下的conf/server.xml文件中修改**,把8080端口改成是8088或者是其他的

**tomcat结构目录的简单介绍**

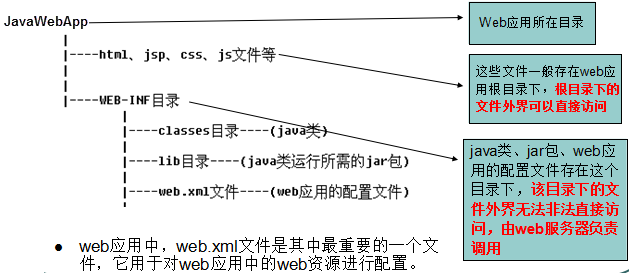
1. bin：启动和关闭tomcat的bat文件
2. conf：配置文件
   * server.xml该文件用于配置server相关的信息，比如tomcat启动的端口号，配置主机(Host)
   * web.xml文件配置与web应用（web应用相当于一个web站点）
   * tomcat-user.xml配置用户名密码和相关权限.
3. lib：该目录放置运行tomcat运行需要的jar包
4. logs：存放日志，当我们需要查看日志的时候，可以查询信息
5. webapps：放置我们的web应用
6. work工作目录：该目录用于存放**jsp被访问后生成对应的server文件和.class文件**

**在webapps中建立了web1目录**，下面放置我们的html文件，jsp文件，图片等等，**则web1就被当做web应用管理起来。**

**web站点的目录是有规范的**

为什么要这样设置web站点目录呢？

* 需求：我有多个html文件，想把其中的一个html文件作为我web站点的首页。
* 如果**没有WEB-INF目录下的web.xml文件支持，是无法解决我的需求的**
* **这个规范是约定熟成的。**



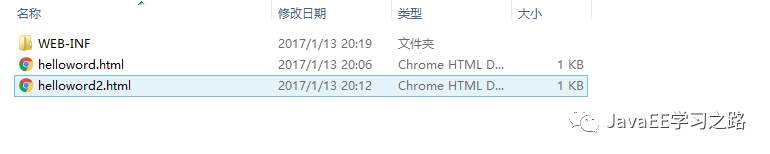
**WebRoot →Web应用所在目录，一般情况下虚拟目录要配置到此文件夹当中。**

　　　　┝WEB-INF：**此文件夹必须位于WebRoot文件夹里面，而且必须以这样的形式去命名，字母都要大写。**

　┝web.xml：**配置文件，有格式要求，此文件必须以这样的形式去命名，并且必须放置到WEB-INF文件夹中。**

**WEB-INF目录下的资源是不能直接被浏览器访问的**

**将web站点下的helloword2.xml文件作为站点的首页**

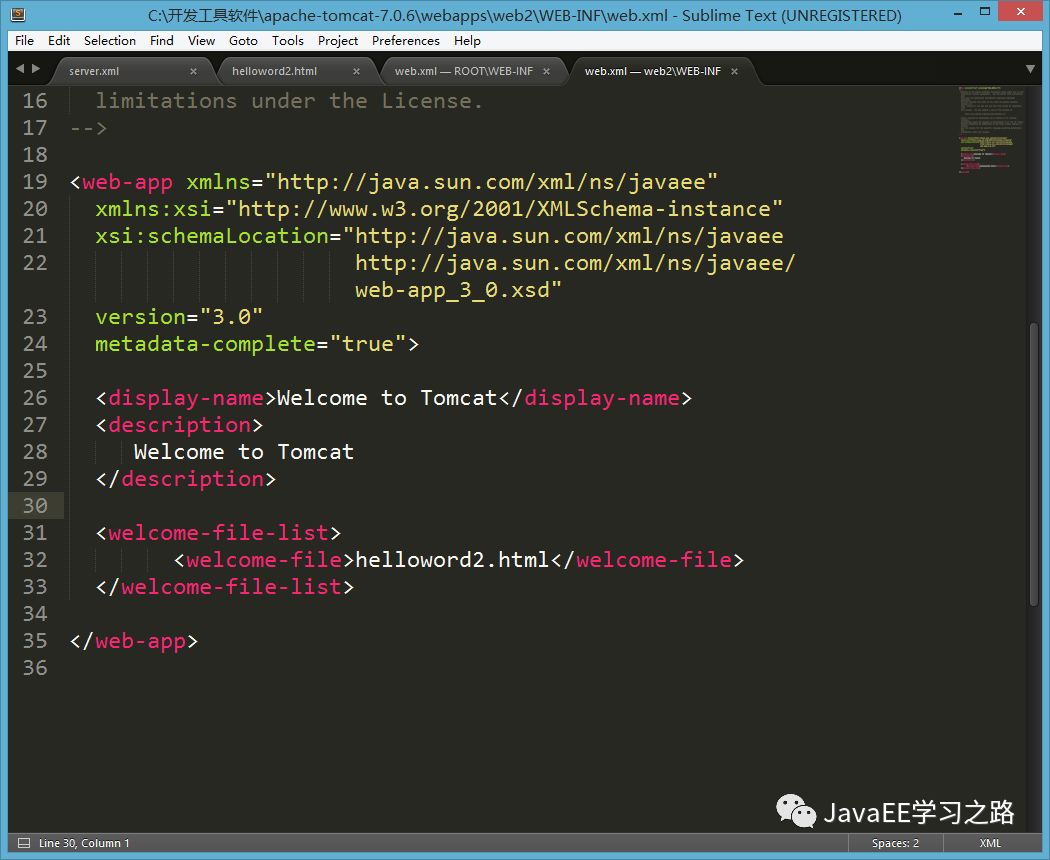
* 新建一个WEB-INF目录，在**web1目录中。**
* 在WEB-INF目录下创建一个web.xml
* web.xml我们不可能会写，所以可以在webapps目录下其他的站点中抄一份过来【复制ROOT/WEB-INF/web.xml的文件到自己的站点中】
* 在web.xml中添加以下代码：

在工程的[web.xml](https://www.baidu.com/s?wd=web.xml&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)中会有一个

<welcome-file-list>

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

</welcome-file-list>

这个代码 ，这是设置工程访问的首页

* 访问web站点【**helloword2.html已经是web站点的首页了，所以不需要指定资源访问了**】

## Tomcat 端口配置，及原理详解

tomcat服务器需**配置三个端口**才能启动，安装时默认启用了这三个端口，当要运行多个tomcat服务时需要修改这三个端口，不能相同。

**端口配置路径为tomcat\conf\service.xml**

* 修改Shutdown端口（默认为8005端口）

<Server port="8005" shutdown="SHUTDOWN">

* 修改http访问端口（默认为8080端口）   
  负责**建立HTTP连接**。在通过浏览器访问Tomcat服务器的Web应用时，使用的就是这个连接器。

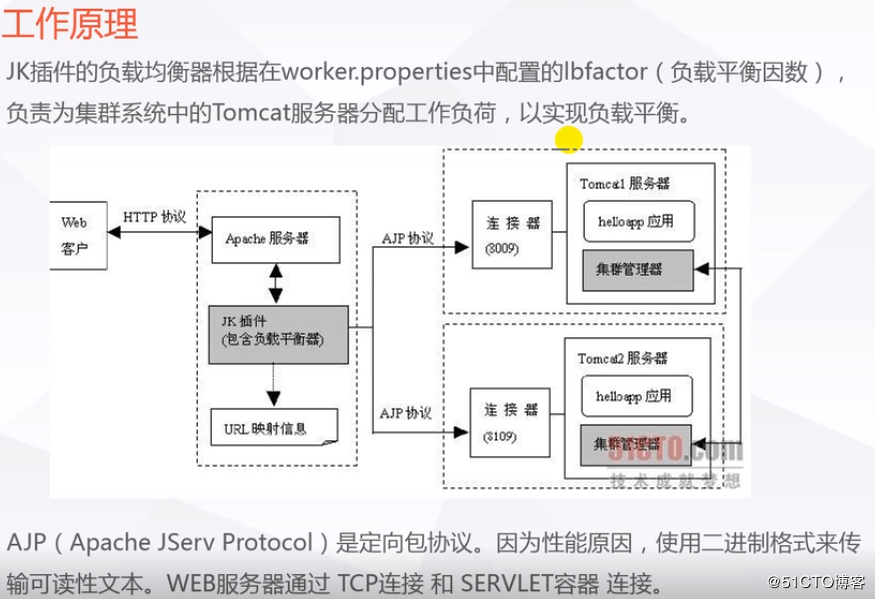
<Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="8443" />

* **AJP协议访问端口（默认为8009端口）   
  负责和其他的HTTP服务器建立连接。**在把Tomcat与其他HTTP服务器集成时，就需要用到这个连接器。

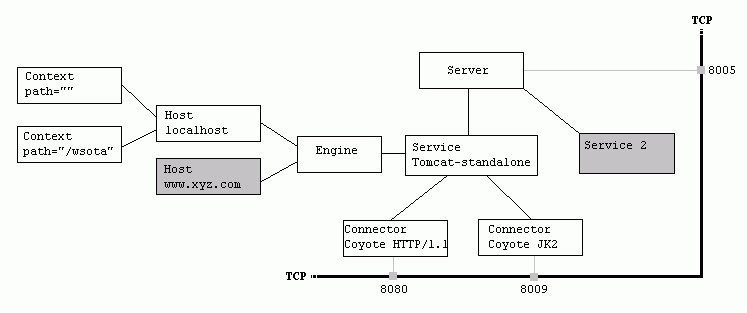
<!-- Define an AJP 1.3 Connector on port 8009 -->

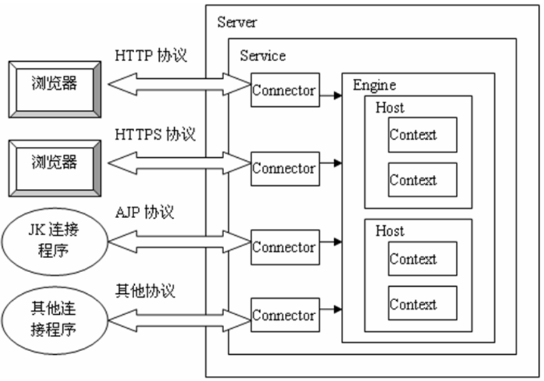
<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />

* Web客户访问Tomcat服务器上**JSP组件**的两种方式如图所示。

**Tomcat Server.xml的组成说明**





**结合图，我们研究下Tomcat Server处理一个http请求的过程**

假设来自客户的请求为：   
<http://localhost:8080/wsota/wsota_index.jsp>   
1. 请求被发送到本机端口8080，被在那里侦听的Coyote HTTP/1.1 Connector获得   
2. Connector把该请求交给它所在的**Service的Engine来处理**，并等待来自Engine的回应   
3. Engine获得请求localhost/wsota/wsota\_index.jsp，匹配它所拥有的所有虚拟主机Host   
4. Engine匹配到名为localhost的Host（即使匹配不到也把请求交给该Host处理，因为该Host被定义为该Engine的默认主机）   
5. localhost Host获得请求/wsota/wsota\_index.jsp，匹配它所拥有的所有Context   
6. Host匹配到路径为/wsota的Context（如果匹配不到就把该请求交给路径名为”“的Context去处理）   
7. path=”/wsota”的Context获得请求/wsota\_index.jsp，在它的mapping table中寻找对应的servlet   
8. Context匹配到URL PATTERN为\*.jsp的servlet，对应于JspServlet类   
9. 构造HttpServletRequest对象和HttpServletResponse对象，作为参数调用JspServlet的doGet或doPost方法   
10. Context把执行完了之后的HttpServletResponse对象**返回给Host**  
11. Host把HttpServletResponse对象**返回给Engine**  
12. Engine把HttpServletResponse对象**返回给Connector**  
13. Connector把HttpServletResponse对象**返回给客户browser**

* Server   
  A Server element represents the entire Catalina servlet container. **(Singleton)**
* Service   
  A Service element represents the combination of one or more Connector components that share a single Engine   
  Service是这样一个集合：它由**一个或者多个Connector组成，以及一个Engine**，负责处理所有Connector所获得的客户请求
* Connector   
  一个Connector将在某个指定端口上侦听客户请求，并将获得的请求交给Engine来处理，从Engine处获得回应并返回客户   
  TOMCAT有两个典型的Connector，**一个直接侦听来自browser的http请求，一个侦听来自其它WebServer的请求**Coyote Http/1.1 Connector 在端口8080处侦听来自客户browser的http请求，Coyote JK2 Connector 在端口8009处侦听来自其它WebServer(Apache)的servlet/jsp代理请求
* Engine   
  The Engine element represents the entire request processing machinery associated with a particular Service   
  It receives and processes all requests from one or more Connectors and returns the completed response to the Connector for ultimate transmission back to the client Engine下可以配置多个虚拟主机Virtual Host，每个虚拟主机都有一个域名，**当Engine获得一个请求时，它把该请求匹配到某个Host上，**然后把该请求交给该Host来处理，Engine有一个默认虚拟主机，当请求无法匹配到任何一个Host上的时候，将交给该默认Host来处理
* Host   
  代表一个Virtual Host，虚拟主机，每个虚拟主机和某个网络域名Domain Name相匹配，每个虚拟主机下都可以部署(deploy)一个或者多个Web App，每个Web App对应于一个Context，有一个Context path，当Host获得一个请求时，将把该请求匹配到某个Context上，然后把该请求交给该Context来处理，匹配的方法是“最长匹配”，所以一个path==”“的Context将成为该Host的默认Context，所有无法和其它Context的路径名匹配的请求都将最终和该默认Context匹配
* Context   
  一个Context对应于一个Web Application，一个Web Application由一个或者多个Servlet组成Context在创建的时候将根据配置文件WEBAPP\_HOME/WEB-INF/web.xml载入Servlet类当Context获得请求时，将在自己的映射表(mapping table)中寻找相匹配的Servlet类，如果找到，则执行该类，获得请求的回应，并返回。

配置文件$CATALINA\_HOME/conf/server.xml的说明该文件描述了如何启动Tomcat Server

<!-- 启动Server

\*\*\*在端口8005处等待关闭命令\*\*\*

如果接受到"SHUTDOWN"字符串则关闭服务器

-->

<Server port="8005" shutdown="SHUTDOWN" debug="0">

<!-- Listener ???

目前没有看到这里

-->

<Listener className="org.apache.catalina.mbeans.ServerLifecycleListener" debug="0"/>

<Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" debug="0"/>

<!-- Global JNDI resources ???

目前没有看到这里，先略去

-->

<GlobalNamingResources>

... ... ... ...

</GlobalNamingResources>

<!-- Tomcat的Standalone Service

**Service是一组Connector的集合**

它们共用一个Engine来处理所有Connector收到的请求

-->

<Service name="Tomcat-Standalone">

<!-- Coyote HTTP/1.1 Connector

className : 该Connector的实现类是org.apache.coyote.tomcat4.CoyoteConnector

\*\*\*port : 在端口号8080处侦听来自客户browser的HTTP1.1请求\*\*\*

minProcessors : 该Connector先创建5个线程等待客户请求，每个请求由一个线程负责

maxProcessors : 当现有的线程不够服务客户请求时，若线程总数不足75个，则创建新线程来处理请求

acceptCount : 当现有线程已经达到最大数75时，为客户请求排队

当队列中请求数超过100时，后来的请求返回Connection refused错误

redirectport : 当客户请求是https时，把该请求转发到端口8443去

其它属性略

-->

<Connector className="org.apache.coyote.tomcat4.CoyoteConnector"

port="8080"

minProcessors="5" maxProcessors="75" acceptCount="100"

enableLookups="true"

redirectPort="8443"

debug="0"

connectionTimeout="20000"

useURIValidationHack="false"

disableUploadTimeout="true" />

<!-- Engine用来处理Connector收到的Http请求

它将匹配请求和自己的虚拟主机，并把请求转交给对应的Host来处理

默认虚拟主机是localhost

-->

<Engine name="Standalone" defaultHost="localhost" debug="0">

<!-- 日志类，目前没有看到，略去先 -->

<Logger className="org.apache.catalina.logger.FileLogger" .../>

<!-- Realm，目前没有看到，略去先 -->

<Realm className="org.apache.catalina.realm.UserDatabaseRealm" .../>

<!-- 虚拟主机localhost

appBase : 该虚拟主机的根目录是webapps/

它将匹配请求和自己的Context的路径，并把请求转交给对应的Context来处理

-->

<Host name="localhost" debug="0" appBase="webapps" unpackWARs="true" autoDeploy="true">

<!-- 日志类，目前没有看到，略去先 -->

<Logger className="org.apache.catalina.logger.FileLogger" .../>

<!-- Context，对应于一个Web App

path : 该Context的路径名是""，故该Context是该Host的默认Context

docBase : 该Context的根目录是webapps/mycontext/

-->

<Context path="" docBase="mycontext" debug="0"/>

<!-- 另外一个Context，路径名是/wsota -->

<Context path="/wsota" docBase="wsotaProject" debug="0"/>

</Host>

</Engine>

</Service>

</Server>

## -Tomcat服务器虚拟目录的映射方式(content)

**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**。

**为什么需要配置虚拟目录？**

* 如果把所有web站点的目录都放在webapps下，可能导致**磁盘空间不够用**，也**不利于对web站点目录的管理**【如果存在非常多的web站点目录】
* 把**web站点的目录分散到其他磁盘管理就需要配置虚拟目录【默认情况下，只有webapps下的目录才能被Tomcat自动管理成一个web站点】**
* 把web应用所在目录交给web服务器管理，这个过程称之为虚拟目录的映射
* **配置虚拟目录方法一：在server.xml文件的host元素中配置**
* 在**其他盘符**下创建一个web站点目录，并创建WEB-INF目录和一个html文件。
* 找到Tomcat目录下/conf/server.xml文件
* 在server.xml中的host元素节点下添加如下代码。**path表示的是访问时输入的web项目名，docBase表示的是站点目录的绝对路径**

**<Context** path="/JavaWebApp" docBase="F:\JavaWebDemoProject" />

即可将在F盘下的JavaWebDemoProject这个JavaWeb应用映射到JavaWebApp这个虚拟目录上，JavaWebApp这个虚拟目录是由Tomcat服务器管理的，JavaWebApp是一个硬盘上不存在的目录，**是我们自己随便写的一个目录**，也就是虚拟的一个目录，所以称之为"虚拟目录"。

1 <Host name="localhost" appBase="webapps"

2 unpackWARs="true" autoDeploy="true"

3 xmlValidation="false" xmlNamespaceAware="false">

4

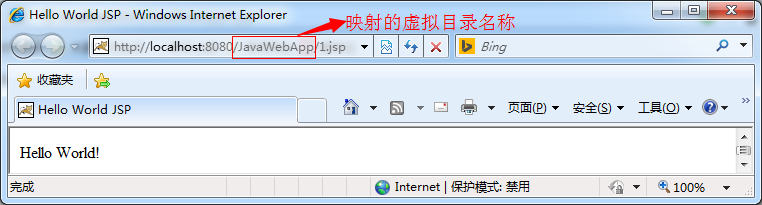
5 <Context path="/JavaWebApp" docBase="F:\JavaWebDemoProject" />

6 </Host>

其中，Context表示上下文，**代表的就是一个JavaWeb应用**，Context元素有两个属性，

* Ⅰ.path：用来配置虚似目录，**必须以"/"开头。**
* Ⅱ.docBase：配置此虚似目录对应着硬盘上的Web应用所在目录。

使用浏览器访问"/JavaWebApp"这个虚拟目录下的1.jsp这个web资源，访问结果如下：

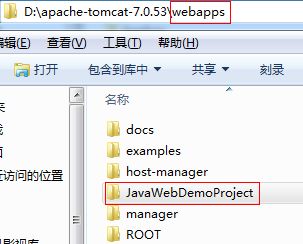


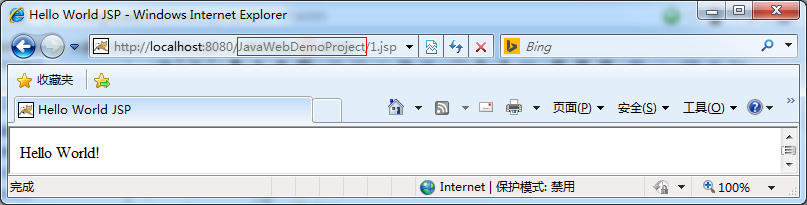
jsp可以正常访问，这说明我们已经成功地将将在**F盘下的JavaWebDemoProject**这个JavaWeb应用**映射到JavaWebApp这个虚拟目录上**了，访问"/JavaWebApp/1.jsp"就相当于访问"F:\JavaWebDemoProject\1.jsp"。、

**注意：**在Tomcat6之后中，不再建议在server.xml文件中使用配置context元素的方式来添加虚拟目录的映射，因为每次修改server.xml文件后，Tomcat服务器就必须要重新启动后才能重新加载server.xml文件。

* **虚拟目录的映射方式二：让tomcat服务器自动映射**

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。

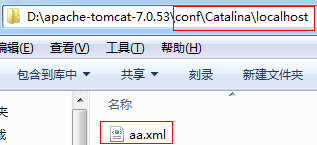
例如：把F盘下的JavaWebDemoProject这个JavaWeb应用直接copy到tomcat服务器webapps目录中，如下图所示：  
　　

此时Tomcat服务器就会自动为JavaWebDemoProject这个JavaWeb应用映射一个同名的虚拟目录"/JavaWebDemoProject"，然后就可以使用浏览器访问这个JavaWeb应用的资源了，如下图所示：

* **虚拟目录的映射方式三**

在tomcat服务器的**\conf\Catalina\localhost目录**下添加一个以**xml作为扩展名**的文件，xml文件的名字可以任意取，比如下面的aa.xml。 **"context元素的path属性源自于是这个xml文件的名字"，**上面提到过，Context元素的path属性是用来配置虚似目录的名称的，所以虚似目录的名称就是这个xml文件的名称。**该文件的名字就是站点的名字。**

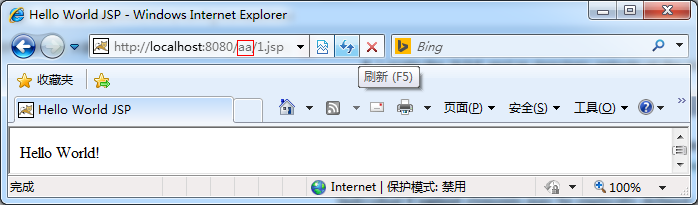
$CATALINA\_BASE指的就是tomcat服务器根目录，[enginename]指的是Tomcat服务器使用的引擎名称，Tomcat使用的引擎是Catalina



在aa.xml文件中添加**Context**元素映射JavaWeb应用，代码如下：

**<Context docBase="F:\JavaWebDemoProject" />**

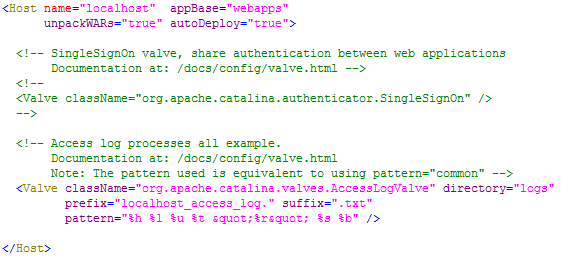
**注意**：在**Context**元素中并没有指明path属性来设置虚拟目录的名称，那么"F:\JavaWebDemoProject"映射的虚拟目录名称是神马呢，就是**当前正在编辑的这个xml文件的名称aa。**



使用这种方式映射虚拟目录的最大好处是修改了配置文件后不用重启Tomcat服务器，比如将aa.xml修改成bb.xml，Tomcat服务器会自动Undeploying context [/aa]，然后自动信息。

## Tomcat服务器配置虚似主机

配置虚似主机就是配置一个网站。**多个不同域名的网站共存于一个Tomcat中**  
　　在Tomcat服务器配置一个虚拟主机(网站)，需要修改conf文件夹下的server.xml这个配置文件，使用Host元素进行配置，打开server.xml，可以看到Tomcat服务器自带的一个名称为localhost的虚拟主机(网站)，如下图所示：

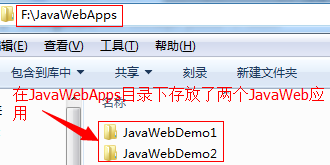


平时我们将开发好的JavaWeb应用放到webapps文件夹下，然后就可以使用"**http://localhost:端口号/JavaWebAppName**"的方式去访问了，其实访问的就是name是"localhost"的那台虚拟主机(Host)，这台虚拟主机管理webapps文件夹下的所有web应用。

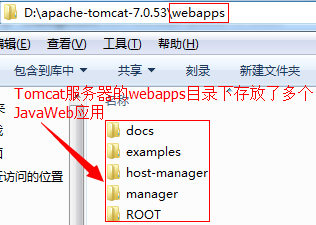
* **在tomcat的server.xml文件中添加主机名**

 <Host name="www.gacl.cn" appBase="F:\JavaWebApps">  
                   <Context path="/JavaWebApp" docBase=" F:\JavaWebApps " />  
      </Host>

这里我们新配置一个虚拟主机，虚拟主机的name是"www.gacl.cn"，虚拟主机"www.gacl.cn"现在管理着**JavaWebApps文件夹下的所有web应用**

****

**就好像是Tomcat服务器的webapps文件夹一样，里面存放了很多的JavaWeb应用**

****

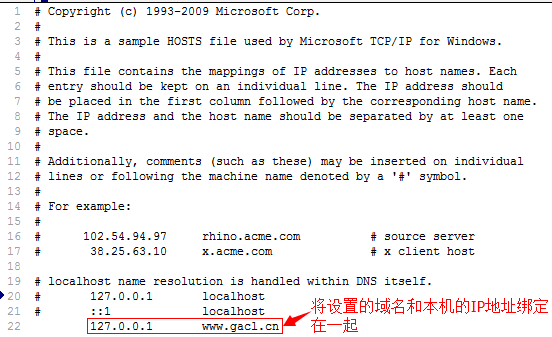
## 配置临时域名

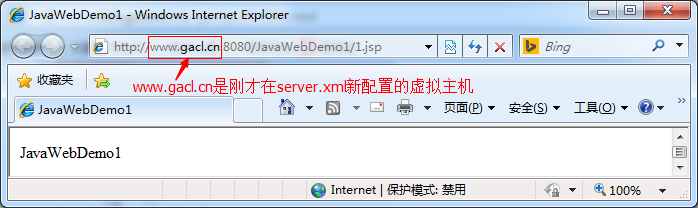
**访问Tomcat服务器有好几种方式**

* **使用localhost域名访问【localhost代表本机】**
* **使用ip地址127.0.0.1访问【该ip地址也是本机】**
* **使用机器名称访问【只限用于本机上或者局域网】**
* **使用本机IP地址访问【在cmd中输入ipconfig可以查询到本机IP地址】**
* **还可以为机器配置临时域名**

配置临时域名步骤

* 配置的主机(网站)要想通过域名被外部访问，必须在DNS服务器或windows系统中注册访问网站时使用的域名，找到"**C:\Windows\System32\drivers\etc**"**目录下的hosts文件**
* 编辑这个文件，将新添加的网站的域名和IP地址绑定在一起，这样我们就可以在浏览器中使用www.gacl.cn这个域名去访问name是www.gacl.cn那个虚拟主机里面管理的那些web应用了



使用浏览器通过域名"www.gacl.cn"访问"www.gacl.cn"这个虚拟主机下JavaWebDemo1这个web应用下的1.jsp这个web资源，**"www.gacl.cn"这个虚拟主机开放了一个8080端口，**用户只能通过8080端口去访问JavaWebDemo1这个web应用下1.jsp这个web资源

## https连接器：互联网上的加密

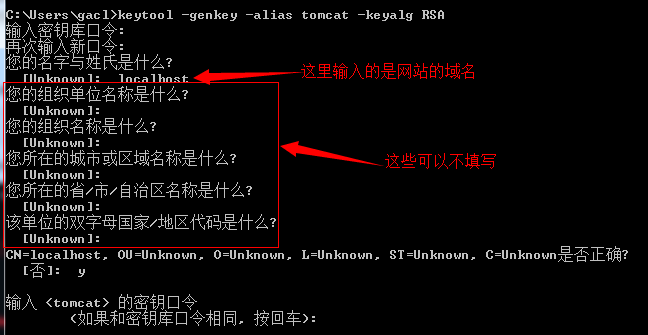
浏览器与服务器交互时，浏览器想将数据加密后再发送给服务器，那么该怎么做呢？**服务器首先要向浏览器出示一份数字证书，浏览器看到数字证书后，就可以使用数字证书里面的公钥加密数据，所以要想做浏览器和服务器的加密数据传输，那么首先得针对服务器生成一份数字证书。**然后再配置一下服务器，让服务器收到浏览器的请求之后，会向浏览器出示它的数字证书。

* 1. **生成Tomcat服务器的数字证书**

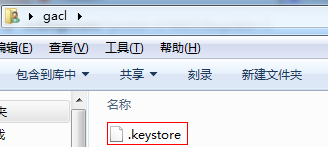
SUN公司提供了制作证书的工具keytool， 在JDK 1.4以后的版本中都包含了这一工具，它的位置为<JAVA\_HOME>\bin\keytool.exe

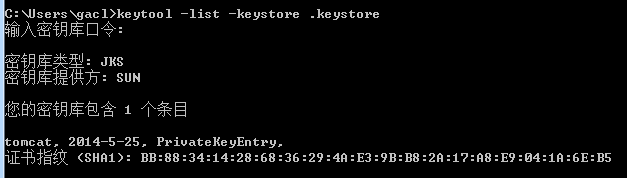
keytool -genkey -alias tomcat -keyalg RSA

 使用keytool生成一个名字为tomcat的证书，存放在.keystore这个密钥库中



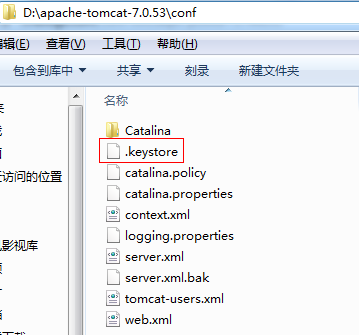
命令执行完，操作系统的用户文件夹下面就会生成一个.keystore文件，如下图所示：



使用命令：keytool -list -keystore .keystore查看.keystore密钥库里面的所有证书　　

* 1. **配置https连接器**

将生成的.keystore密钥库文件拷贝到Tomcat服务器的conf目录下，如下图所示：



　　修改server.xml文件，配置https连接器，代码如下：

1 <Connector port="8443" protocol="org.apache.coyote.http11.Http11Protocol"

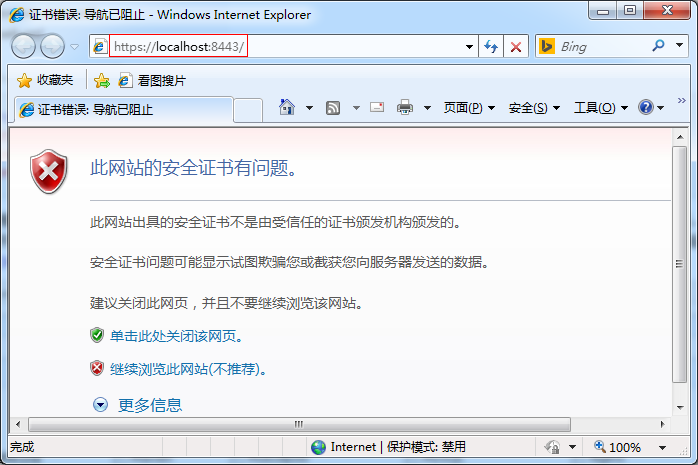
2 maxThreads="150" SSLEnabled="true" scheme="https" secure="true"

3 clientAuth="false" sslProtocol="TLS"

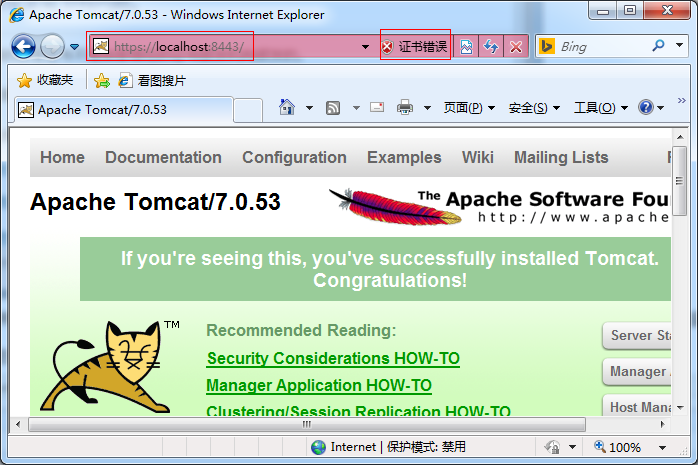
4 keystoreFile="conf/.keystore" keystorePass="123456"/>

在server.xml文件中配置了一个端口是8443的加密连接器，浏览器访问8443端口的连接器时，将会以加密的方式来访问web服务器，这个连接器收到浏览器的请求后，将会向浏览器出示一份数字证书，浏览器再用数字证书里面的公钥来加密数据，keystoreFile="conf/.keystore" 用来指明密钥库文件的所在路径，服务器从密钥库中提取证书时需要密码，keystorePass="123456"指明密钥库的访问密码。

使用"https://localhost:8443/"访问8443的加密连接器

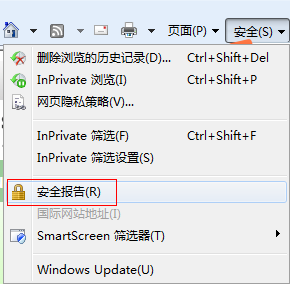


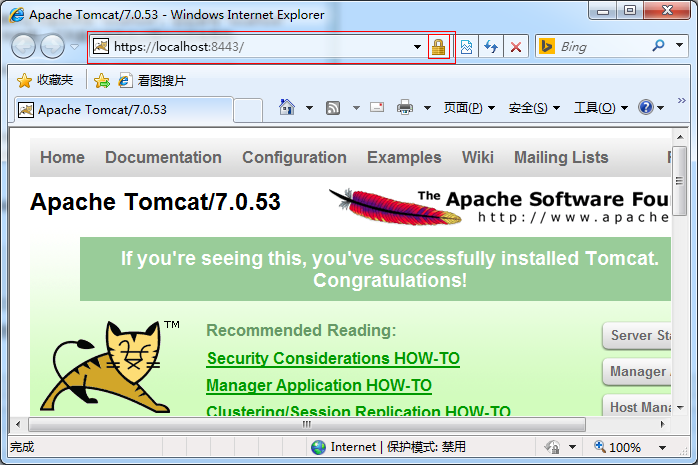
由于密钥库里面的证书是我们手工生成的，没有经过CA的认证，所以使用"https://localhost:8443/"访问8443的加密连接器，浏览器会出现"证书错误，导航已阻止"，浏览器认为当前要访问的这个主机是不安全的，不推荐继续访问，点击C:\Users\夏\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\63A4DAB1.tmp就可以继续访问了，如下图所示：



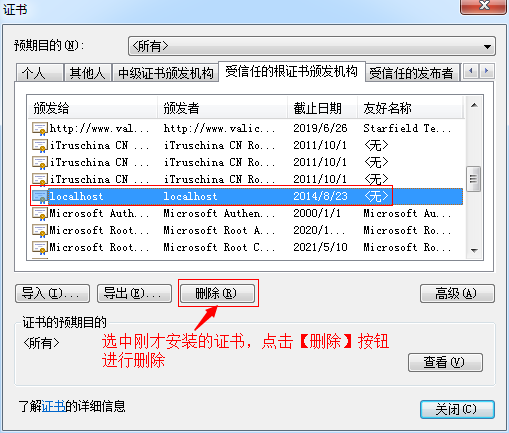
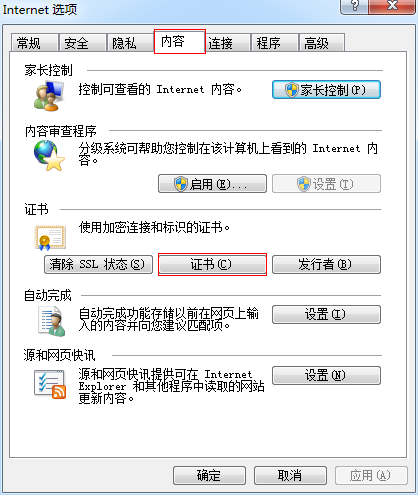
* 1. **安装数字证书**

　　为了让浏览器信任我们生成的数字证书，需要将数字证书安装到浏览器中，以IE8浏览器为例进行证书安装说明，安装步骤如下：



证书安装成功后，重启IE浏览器，使用"https://localhost:8443/"访问8443的加密连接器，此时浏览器就不再提示证书错误了，如下图所示：

* 1. **删除数字证书**

　　以IE8为例进行说明，操作步骤如下：工具----->Internet选项　

删除之后重启浏览器即可

# HTTP协议

## 什么是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。客户端连上web服务器后，若想获得web服务器中的某个web资源，需遵守一定的通讯格式，**HTTP协议用于定义客户端与web服务器通迅的格式。**

**特点：无状态，默认端口 80**

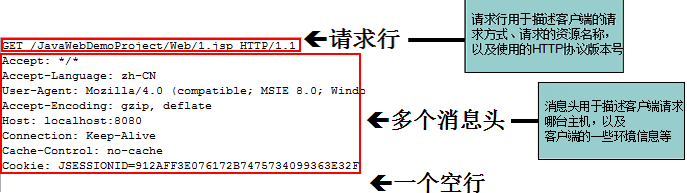
**HTTP 协议到底约束了什么？**

1. **约束了浏览器以何种格式向服务端发送数据**
2. **约束了服务器应该以何种格式来接收客户端发送的数据**
3. **约束了服务器应该以何种格式来反馈数据给浏览器**
4. **约束了浏览器应该以何种格式来接收服务器的反馈数据**

## HTTP请求

**HTTP请求包括的内容：**

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、以及实体内容**  


**HTTP请求的细节——请求行**

请求行中的GET称之为请求方式，请求方式有：POST、GET、HEAD、OPTIONS、DELETE、TRACE、PUT，常用的有： GET、 POST

**get重点在从服务器上获取资源，post重点在向服务器发送数据；**

**get方式只能支持ASCII字符，向服务器传的中文字符可能会乱码。post支持标准字符集，可以正确传递中文字符。**

用户如果没有设置，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。  
　　不管POST或GET，向服务器请求某个WEB资源，区别主要表现在数据传递上：如果请求方式为GET方式，则可以在请求的URL地址后以?的形式带上交给服务器的数据，多个数据以&进行分隔，GET /mail/1.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。  
　　如果请求方式为POST方式，则可以在请求的实体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。

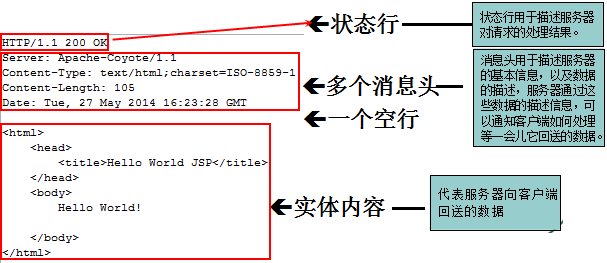
**HTTP请求的细节——消息头**

　　HTTP请求中的常用消息头

* Accept: text/html,image/\* 【浏览器告诉服务器，它支持的数据类型】
* Accept-Charset: ISO-8859-1 【浏览器告诉服务器，它支持哪种**字符集**】
* Accept-Encoding: gzip,compress 【浏览器告诉服务器，它支持的**压缩格式**】
* Accept-Language: en-us,zh-cn 【浏览器告诉服务器，它的语言环境】
* Host: www.it315.org:80【浏览器告诉服务器，它的想访问哪台主机】
* If-Modified-Since: Tue, 11 Jul 2000 18:23:51 GMT【浏览器告诉服务器，缓存数据的时间】
* Referer: http://www.it315.org/index.jsp【浏览器告诉服务器，客户机是从那个页面来的---**反盗链**】
* User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)【浏览器告诉服务器，浏览器的内核是什么】
* Cookie【浏览器告诉服务器，**带来的Cookie是什么**】
* Connection: close/Keep-Alive 【浏览器告诉服务器，请求完后是断开链接还是保持链接】
* Date: Tue, 11 Jul 2000 18:23:51 GMT【浏览器告诉服务器，请求的时间】

## HTTP响应

**HTTP响应包括的内容**

一个HTTP响应代表服务器向客户端回送的数据，它包括： 一个状态行、若干消息头、以及实体内容 

**HTTP响应的细节——状态行**

* 状态行格式： 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处理结果，它是一个三位的十进制数。响应状态码分为5类，如下所示：

**响应头**

* Location: http://www.it315.org/index.jsp 【服务器告诉浏览器**要跳转到哪个页面**】
* Server:apache tomcat【服务器告诉浏览器，服务器的型号是什么】
* Content-Encoding: gzip 【服务器告诉浏览器**数据压缩的格式**】
* Content-Length: 80 【服务器告诉浏览器回送数据的长度】
* Content-Language: zh-cn 【服务器告诉浏览器，服务器的语言环境】
* Content-Type: text/html; charset=GB2312 【服务器告诉浏览器，**回送数据的类型**】
* Last-Modified: Tue, 11 Jul 2000 18:23:51 GMT【服务器告诉浏览器该资源上次更新时间】
* Refresh: 1;url=http://www.it315.org【服务器告诉浏览器要**定时刷新**】
* Content-Disposition: attachment; filename=aaa.zip【服务器告诉浏览器**以下载方式打开数据**】
* Transfer-Encoding: chunked 【服务器告诉浏览器数据以分块方式回送】
* Set-Cookie:SS=Q0=5Lb\_nQ; path=/search【服务器告诉浏览器要**保存Cookie**】
* Expires: -1【服务器告诉浏览器**不要设置缓存**】
* Cache-Control: no-cache 【服务器告诉浏览器**不要设置缓存**】
* Pragma: no-cache 【服务器告诉浏览器**不要设置缓存**】
* Connection: close/Keep-Alive 【服务器告诉浏览器连接方式】
* Date: Tue, 11 Jul 2000 18:23:51 GMT【服务器告诉浏览器回送数据的时间】



**HTTP响应细节——常用响应头**

　响应头

* Location: http://www.it315.org/index.jsp 【服务器告诉浏览器**要跳转到哪个页面**】
* Server:apache tomcat【服务器告诉浏览器，服务器的型号是什么】
* Content-Encoding: gzip 【服务器告诉浏览器**数据压缩的格式**】
* Content-Length: 80 【服务器告诉浏览器回送数据的长度】
* Content-Language: zh-cn 【服务器告诉浏览器，服务器的语言环境】
* Content-Type: text/html; charset=GB2312 【服务器告诉浏览器，**回送数据的类型**】
* Last-Modified: Tue, 11 Jul 2000 18:23:51 GMT【服务器告诉浏览器该资源上次更新时间】
* Refresh: 1;url=http://www.it315.org【服务器告诉浏览器要**定时刷新**】
* Content-Disposition: attachment; filename=aaa.zip【服务器告诉浏览器**以下载方式打开数据**】
* Transfer-Encoding: chunked 【服务器告诉浏览器数据以分块方式回送】
* Set-Cookie:SS=Q0=5Lb\_nQ; path=/search【服务器告诉浏览器要**保存Cookie**】
* Expires: -1【服务器告诉浏览器**不要设置缓存**】
* Cache-Control: no-cache 【服务器告诉浏览器**不要设置缓存**】
* Pragma: no-cache 【服务器告诉浏览器**不要设置缓存**】
* Connection: close/Keep-Alive 【服务器告诉浏览器连接方式】
* Date: Tue, 11 Jul 2000 18:23:51 GMT【服务器告诉浏览器回送数据的时间】

## 在服务端设置响应头来控制客户端浏览器的行为

**设置Location响应头，实现请求重定向**

15 response.setStatus(302);//设置服务器的响应状态码

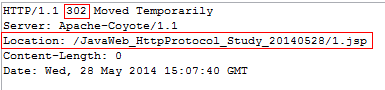
16 /\*\*

17 \*设置响应头，服务器通过 **Location这个头，来告诉浏览器跳到哪里，这就是所谓的请求重定向**

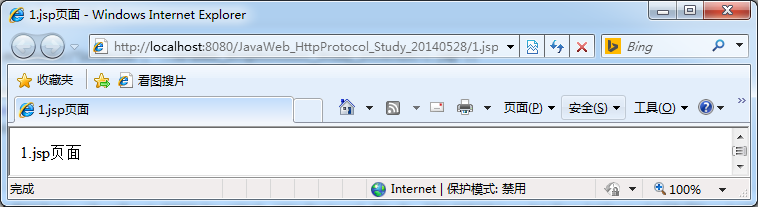
18 \*/

19 response.setHeader("Location", "/JavaWeb\_HttpProtocol\_Study\_20140528/1.jsp");

当在浏览器中使用URL地址"http://localhost:8080/JavaWeb\_HttpProtocol\_Study\_20140528/servlet/ServletDemo01"访问ServletDemo01时，就可以看到服务器作出响应后发送到浏览器的状态码和响应头信息，如下图所示：



服务器返回302状态码，你要的资源没有，我通过Location响应头告诉你哪里有，而浏览器解析响应头Location后知道要跳转到/JavaWeb\_HttpProtocol\_Study\_20140528/1.jsp页面，所以就会自动跳转到1.jsp，如下图所示：



**设置Content-Encoding响应头，告诉浏览器数据的压缩格式**

1 package gacl.http.study;

2

3 import java.io.ByteArrayOutputStream;

4 import java.io.IOException;

5 import java.util.zip.GZIPOutputStream;

6 import javax.servlet.ServletException;

7 import javax.servlet.http.HttpServlet;

8 import javax.servlet.http.HttpServletRequest;

9 import javax.servlet.http.HttpServletResponse;

10 /\*\*

11 \* @author gacl

12 \*这个小程序是用来演示以下两个小知识点

13 \*1、使用GZIPOutputStream流来压缩数据

14 \*2、设置响应头Content-Encoding来告诉浏览器，服务器发送回来的数据压缩后的格式

15 \*/

16 public class ServletDemo02 extends HttpServlet {

17

18 public void doGet(HttpServletRequest request, HttpServletResponse response)

19 throws ServletException, IOException {

20 String data = "abcdabcdabcdabcdabcdabcdab" +

21 "cdabcdabcdabcdabcdabcdabcdabcdabc" +

22 "dabcdabcdabcdabcdabcdabcdabcdabc" +

23 "dabcdabcdabcdabcdabcdabcdabcdabcdab" +

24 "cdabcdabcdabcdabcdabcdabcdabcdabcdab" +

25 "cdabcdabcdabcdabcdabcdabcdabcdabcdab" +

26 "cdabcdabcdabcdabcdabcdabcdabcdabcdab" +

27 "cdabcdabcdabcdabcdabcdabcdabcdabcdabcd";

28 System.out.println("原始数据的大小为：" + data.getBytes().length);

29

30 ByteArrayOutputStream bout = new ByteArrayOutputStream();

31 GZIPOutputStream gout = new GZIPOutputStream(bout); //buffer

32 gout.write(data.getBytes());

33 gout.close();

34 //得到压缩后的数据

35 byte g[] = bout.toByteArray();

36 response.setHeader("Content-Encoding", "gzip");

37 response.setHeader("Content-Length",g.length +"");

38 response.getOutputStream().write(g);

39 }

40

41 public void doPost(HttpServletRequest request, HttpServletResponse response)

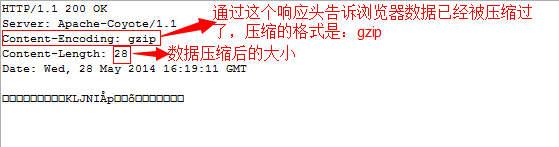
42 throws ServletException, IOException {

43 this.doGet(request, response);

44 }

45 }

服务器发给浏览器的响应信息如下：

C:\Users\夏\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\5019E33D.tmp

浏览器支持的压缩格式有：C:\Users\夏\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\5F325553.tmp

**设置content-type响应头，指定回送数据类型**

1 package gacl.http.study;

2 import java.io.IOException;

3 import java.io.InputStream;

4 import java.io.OutputStream;

5 import javax.servlet.ServletException;

6 import javax.servlet.http.HttpServlet;

7 import javax.servlet.http.HttpServletRequest;

8 import javax.servlet.http.HttpServletResponse;

9 public class ServletDemo03 extends HttpServlet {

10 public void doGet(HttpServletRequest request, HttpServletResponse response)

11 throws ServletException, IOException {

12 /\*\*

13 \* 浏览器能接收(Accept)的数据类型有:

14 \* application/x-ms-application,

15 \* image/jpeg,

16 \* application/xaml+xml,

17 \* image/gif,

18 \* image/pjpeg,

19 \* application/x-ms-xbap,

20 \* application/vnd.ms-excel,

21 \* application/vnd.ms-powerpoint,

22 \* application/msword,

23 \*/

24 response.setHeader("content-type", "image/jpeg");//使用content-type响应头指定发送给浏览器的数据类型为"image/jpeg"

25 //读取位于项目根目录下的img文件夹里面的WP\_20131005\_002.jpg这张图片，返回一个输入流

26 InputStream in = this.getServletContext().getResourceAsStream("/img/WP\_20131005\_002.jpg");

27 byte buffer[] = new byte[1024];

28 int len = 0;

29 OutputStream out = response.getOutputStream();//得到输出流

30 while ((len = in.read(buffer)) > 0) {//读取输入流(in)里面的内容存储到缓冲区(buffer)

31 out.write(buffer, 0, len);//将缓冲区里面的内容输出到浏览器

32 }

33 }

34 public void doPost(HttpServletRequest request, HttpServletResponse response)

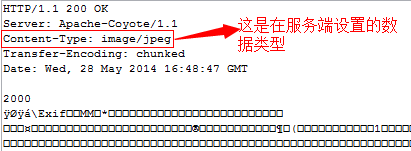
35 throws ServletException, IOException {

36 this.doGet(request, response);

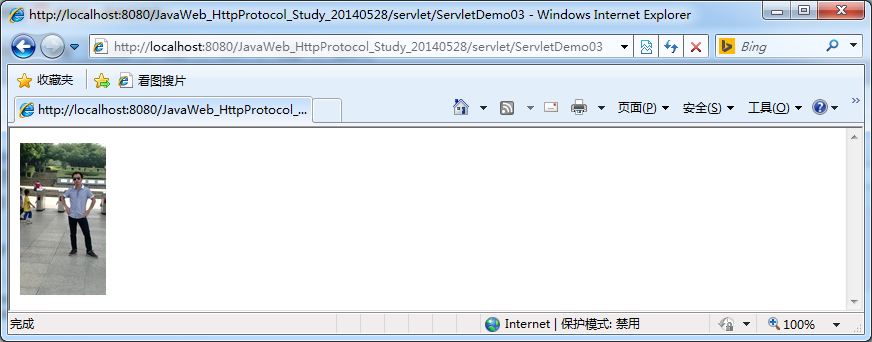
37 }

38 }

服务器发给浏览器的响应信息如下：



ServletDemo03的运行结果如下图所示：



在浏览器中显示出了图片

**设置refresh响应头，让浏览器定时刷新**

12 /\*\*

13 \* 设置refresh响应头，让浏览器每隔3秒定时刷新

14 \*/

15 // response.setHeader("refresh", "3");

16 /\*\*

17 \* **设置refresh响应头，让浏览器3秒后跳转到http://www.baidu.com**

18 \*/

19 **response.setHeader("refresh", "3;url='http://www.baidu.com'");**

20 response.getWriter().write("gacl");

**设置content-disposition响应头，让浏览器下载文件**

1 package gacl.http.study;

2

3 import java.io.IOException;

4 import java.io.InputStream;

5 import java.io.OutputStream;

6

7 import javax.servlet.ServletException;

8 import javax.servlet.http.HttpServlet;

9 import javax.servlet.http.HttpServletRequest;

10 import javax.servlet.http.HttpServletResponse;

11

12 public class ServletDemo05 extends HttpServlet {

13 public void doGet(HttpServletRequest request, HttpServletResponse response)

14 throws ServletException, IOException {

15 /\*\*

16 \* 设置content-disposition响应头，让浏览器下载文件

17 \*/

18 response.setHeader("content-disposition", "attachment;filename=xxx.jpg");

19 InputStream in = this.getServletContext().getResourceAsStream("/img/1.jpg");

20 byte buffer[] = new byte[1024];

21 int len = 0;

22 OutputStream out = response.getOutputStream();

23 while ((len = in.read(buffer)) > 0) {

24 out.write(buffer, 0, len);

25 }

26 }

27

28 public void doPost(HttpServletRequest request, HttpServletResponse response)

29 throws ServletException, IOException {

30 this.doGet(request, response);

31 }

32

33 }

在浏览器中访问ServletDemo05就会弹出文件下载框，如下图所示：

