**WEB**

# 简介

## 相关知识

Web即网页，提供给人们访问的。

**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的

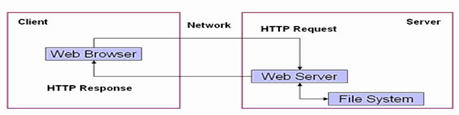
动态web资源： 指web页面中供人们浏览的数据是由程序产生的,不同的时间点访问web页面看到的内容各不相同。

## Web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。例如有：a.html、b.html

Web应用开发好之后，若想提供外界访问，需要把web应用程序所在的目录交给web服务器管理，这个过程称之为虚拟目录的映射。

### 静态资源

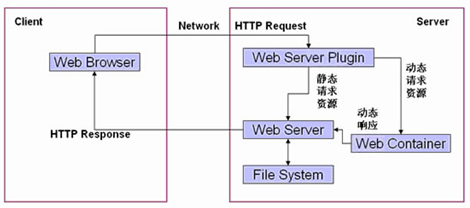


静态资源是用户通过浏览器请求web服务器，web服务器根据用户需求取出内容后返回给浏览器，浏览器经过渲染展现给用户

**缺点：**

* **Web页面内容无法更新**
* **静态页面无法访问数据库**

### 动态资源

****

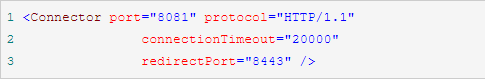
动态资源是指页面可以和用户进行交互，一般情况下用户请求页面，web服务器会根据用户的需求查询数据库的数据，将特定的数据返回给用户，结合JS实现动态的效果。

## Web服务器

Web服务器是一个计算机中的一个应用程序，用于用户发送的请求

# Tomcat服务器

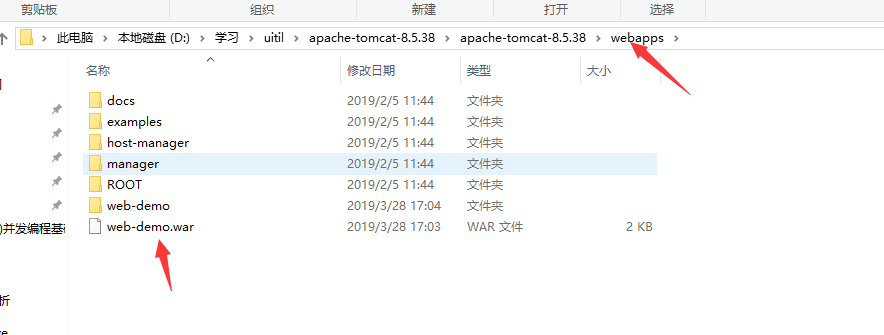
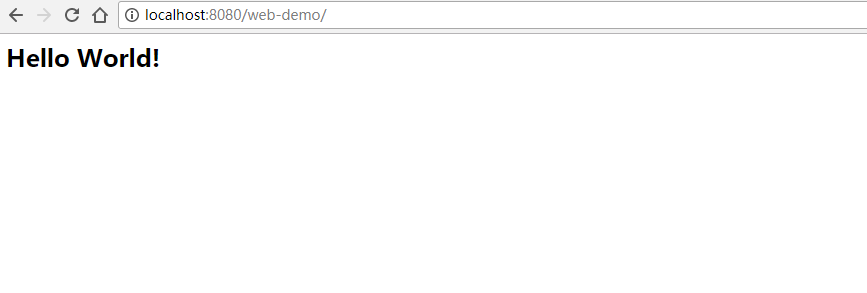
## 服务器端口配置

**将Tomcat服务器启动端口修改成8081端口**

## 服务器虚拟目录映射

**Web应用开发好后，若想供外界访问，需要把web应用所在的目录交给web服务器管理，这个过程称之为虚拟目录的映射。**

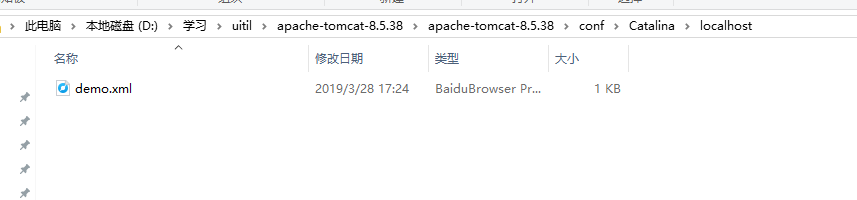
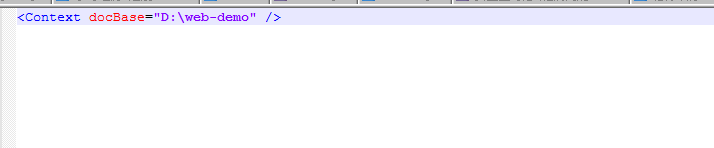
### tomcat服务器自动映射

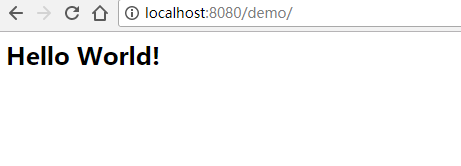
　tomcat服务器会自动管理webapps目录下的所有web应用，并把它映射成虚似目录。换句话说，tomcat服务器webapps目录中的web应用，外界可以直接访问。

### 方法二

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，注意这一句话"**The context path and version will be derived from the base name of the file**"，这一句话的意思翻译过来就是"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。

　　$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

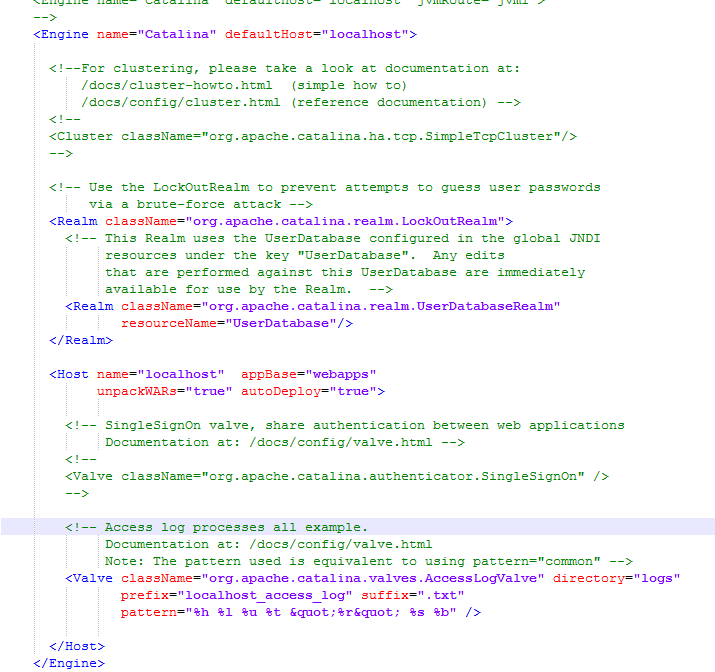


 Xml的文件名就是应用的虚拟目录。

## 配置虚拟主机

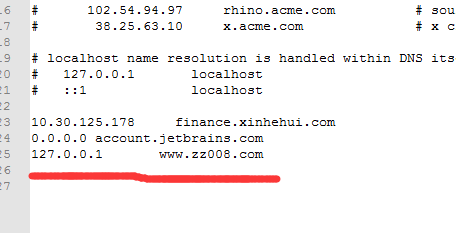
配置虚似主机就是配置一个网站。  
　　在Tomcat服务器配置一个虚拟主机(网站)，需要修改conf文件夹下的server.xml这个配置文件，使用Host元素进行配置，打开server.xml，可以看到Tomcat服务器自带的一个名称为localhost的虚拟主机(网站)

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/demo**"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。

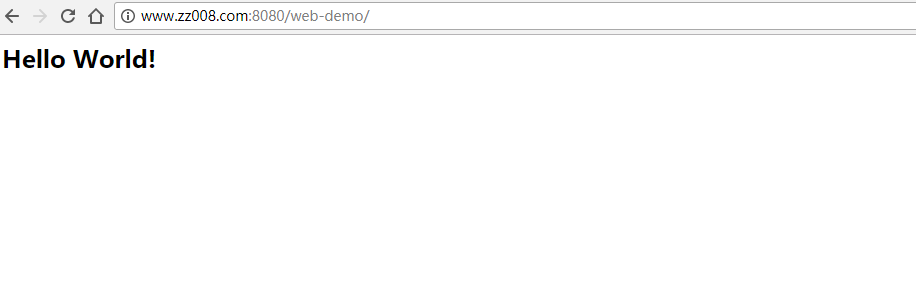
新建一个虚拟主机：

　　 这里我们新配置一个虚拟主机，虚拟主机的name是"www.zz008.com"，虚拟主机" www.zz008.com "现在管理着webapps文件夹下的所有web应用，平时我们在互联网上使用域名"www.baidu.com"访问百度的网站时，其实就是在访问一个名称是"www.baidu.com"的虚拟主机，所以当我们要访问name是" www.zz008.com "的这个虚拟主机时，就可以使用"域名(www.zz008.com)"去访问，注意一下appBase="D:\learn\webapps"，这里的webapps文件夹代表的不是一个项目的根目录，而是一个存放了一个或者多个JavaWeb应用的文件夹

不过要想访问[www.zz008.com这个域名还得修改host](http://www.zz008.com这个域名还得修改host)文件:

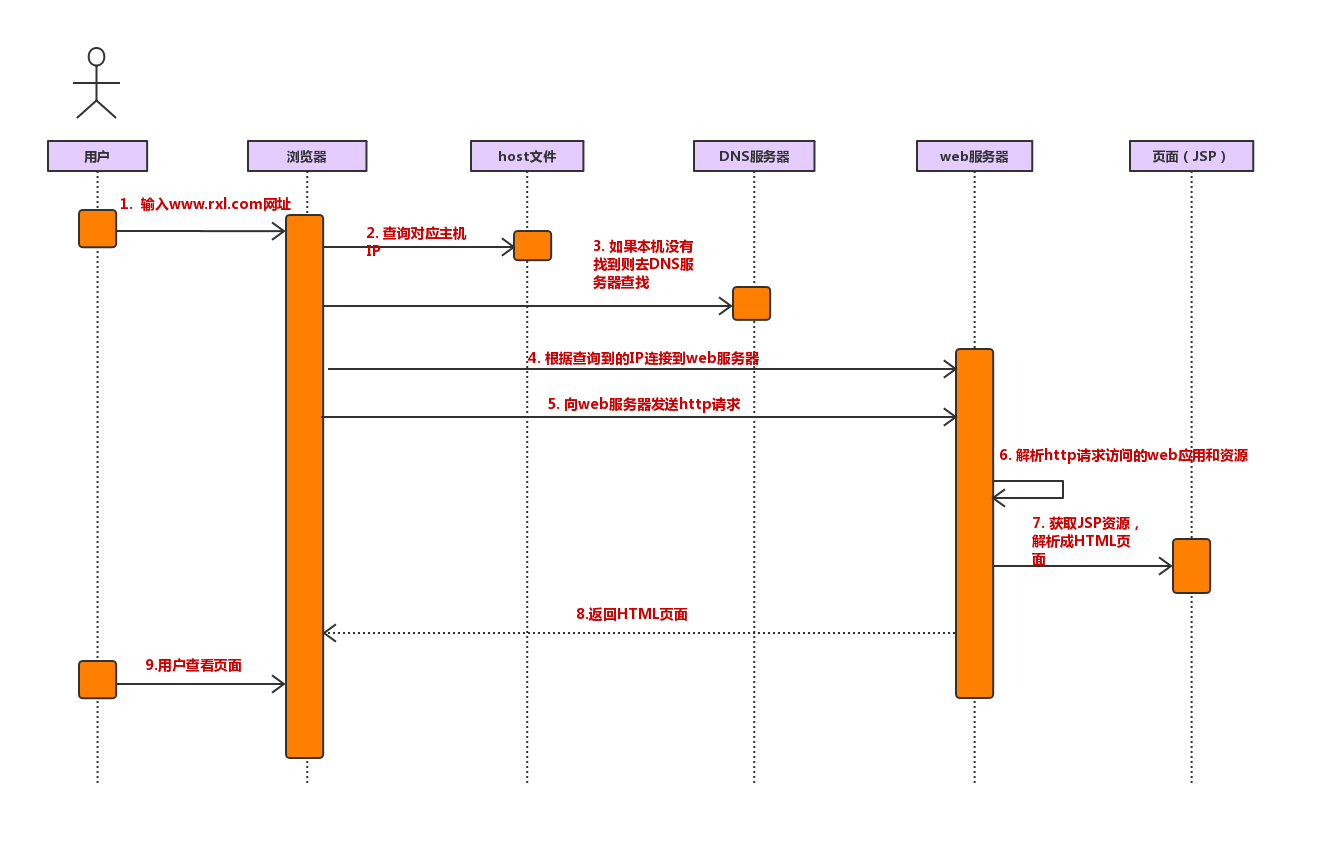


最后就可以访问了:

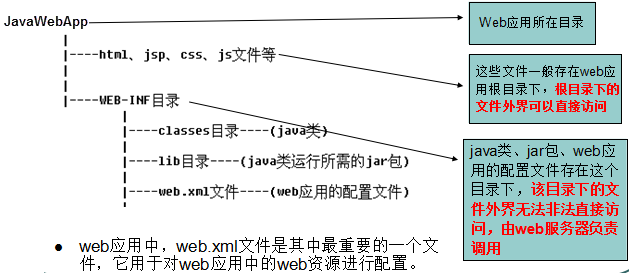


## 浏览器与服务器交互过程

**如图：**



## JavaWeb应用的组成结构



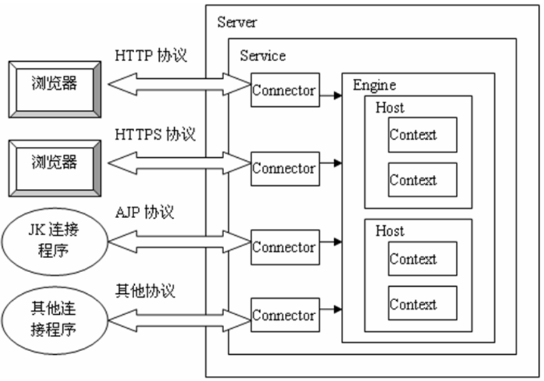
WebRoot →Web应用所在目录，一般情况下虚拟目录要配置到此文件夹当中。

　　　　┝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。**

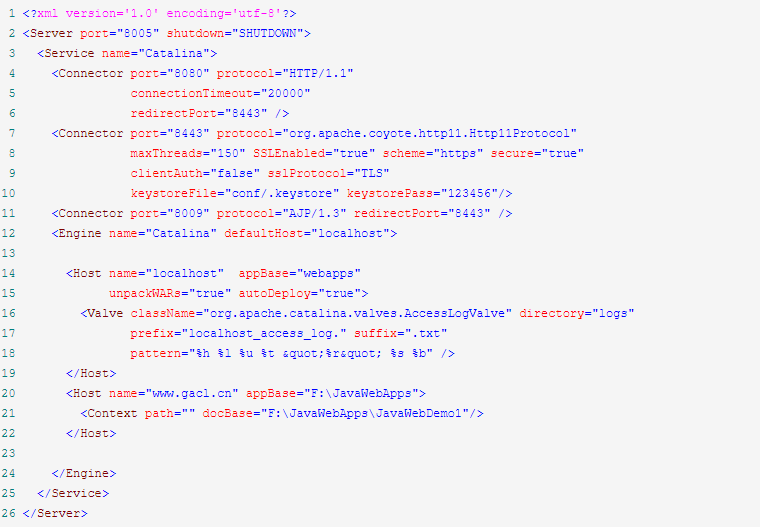
　　web.xml的格式可以直接从Tomcat中参考得到：找到Tomcat目录下的webapps\ROOT\WEB-INF这个目录下的web.xml文件，把这个文件拷贝到我们新建的WEB-INF文件夹中，并修改这个web.xml文件，把里面的注释删除掉，只留下如下所示的代码即可：

## Tomcat体系结构



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)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。

## 互联网上的加密原理

Tomcat服务器启动时候会启动多个Connector(连接器)，而Tomcat服务器的连接器又分为加密连接器和非加密连机器，比如：

### 1. 对称加密

采用单钥密码系统的加密方法，同一个密钥可以同时用作信息的加密和解密，这种加密方法称为对称加密，也称为单密钥加密。

　　需要对加密和解密使用相同密钥的加密算法。由于其速度快，对称性加密通常在消息发送方需要加密大量数据时使用。对称性加密也称为密钥加密。

　　所谓对称，就是采用这种加密方法的双方使用方式用同样的密钥进行加密和解密。密钥是控制加密及解密过程的指令。算法是一组规则，规定如何进行加密和解密。

　　加密的安全性不仅取决于加密算法本身，密钥管理的安全性更是重要。**因为加密和解密都使用同一个密钥，如何把密钥安全地传递到解密者手上就成了必须要解决的问题**。

　　常用的对称加密有：DES、IDEA、RC2、RC4、SKIPJACK、RC5、AES算法等

### 2. 非对称加密

非对称加密算法需要两个密钥：**公开密钥（publickey）**和**私有密钥（privatekey）**。公开密钥与私有密钥是一对，如果用公开密钥对数据进行加密，只有用对应的私有密钥才能解密；如果用私有密钥对数据进行加密，那么只有用对应的公开密钥才能解密。因为加密和解密使用的是两个不同的密钥，所以这种算法叫作非对称加密算法。 非对称加密算法实现机密信息交换的基本过程是：甲方生成一对密钥并将其中的一把作为公用密钥向其它方公开；得到该公用密钥的乙方使用该密钥对机密信息进行加密后再发送给甲方；甲方再用自己保存的另一把专用密钥对加密后的信息进行解密。另一方面，甲方可以使用乙方的公钥对机密信息进行签名后再发送给乙方；乙方再用自己的私匙对数据进行验签。

#### 非对称加密工作原理

　　1.A要向B发送信息，A和B都要产生一对用于加密和解密的公钥和私钥。

　　2.A的私钥保密，A的公钥告诉B；B的私钥保密，B的公钥告诉A。

　　3.A要给B发送信息时，A用B的公钥加密信息，因为A知道B的公钥。

　　4.A将这个消息发给B（已经用B的公钥加密消息）。

　　5.B收到这个消息后，B用自己的私钥解密A的消息。其他所有收到这个报文的人都无法解密，因为只有B才有B的私钥

　　发送方使用接收方的公钥对数据加密，而接收方则使用自己的私钥解密，这样，信息就可以安全无误地到达目的地了，即使被第三方截获，由于没有相应的私钥，也无法进行解密。通过数字的手段保证加密过程是一个不可逆过程，即只有用私有密钥才能解密。

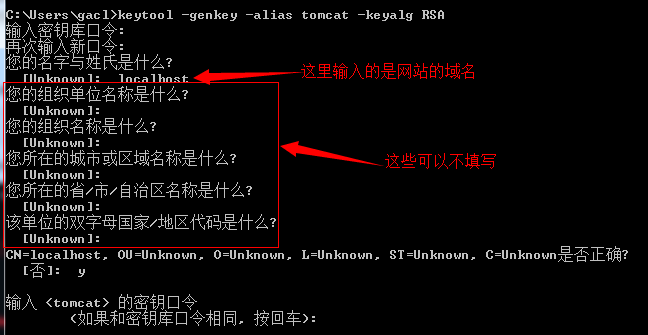
　　非对称性加密依然没有解决数据传输的安全性问题，比如A想向B发数据，B首先生成一对密钥(公钥和私钥)，然后将公钥发给A，A拿到B发给他的公钥有就可以使用公钥加密数据后发给B，然而在B公钥发送给A的这个过程中，很有可能会被第三方C截获，C截获到B的公钥后，也使用B的公钥加密数据，然后发给B，B接收到数据后就晕了，因为搞不清楚接收到的数据到底是A发的还是C发的，这是其中一个问题，另一个问题就是，C截获到B发的公钥后，C可以自己生成一对密钥(公钥和私钥)，然后发给A，A拿到公钥后就以为是B发给他的，然后就使用公钥加密数据发给B，发送给B的过程中被C截获下来，由于A是用C发给他的公钥加密数据的，而C有私钥，因此就可以解密A加密过后的内容了，而B接收到A发给他的数据后反而解不开了，因为数据是用C的公钥加密的，B没有C的私钥，所以就无法解密。所以，**非对称性加密存在一个问题：A想向B发数据，A如何确定拿到的公钥一定是B发的呢？那么如何解决这个问题呢？只能靠一个第三方机构(CA机构，即证书授权中心(Certificate Authority )，或称证书授权机构)来担保。A想向B发数据，B首先将公钥发给CA机构，CA机构拿到B的公钥后跑到B的家里问：这是你发的公钥吗？B确认过后说是：没错，是我发的！那么此时CA机构就会为B的公钥做担保，生成一份数字证书给B，数字证书包含了CA的担保认证签名和B的公钥，B拿到CA的这份数字证书后，就发给A，A拿到数字证书后，看到上面有CA的签名，就可以确定当前拿到的公钥是B发的，那么就可以放心大胆地使用公钥加密数据，然后发给B了。**

## https连接器

### 生成Tomcat服务器的数字证书

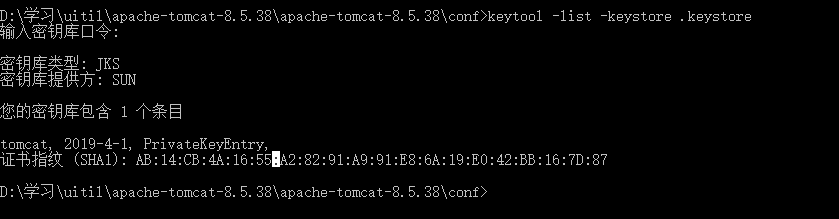
　　SUN公司提供了制作证书的工具keytool， 在JDK 1.4以后的版本中都包含了这一工具，它的位置为<JAVA\_HOME>\bin\keytool.exe

**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

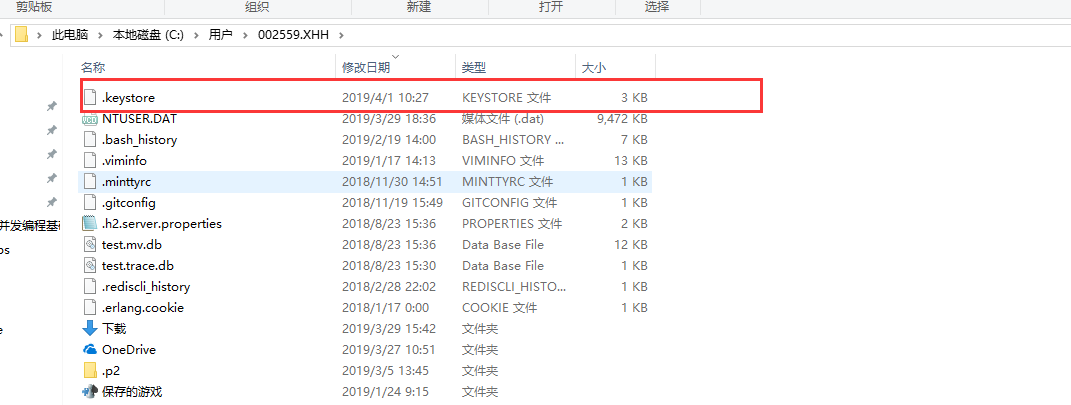
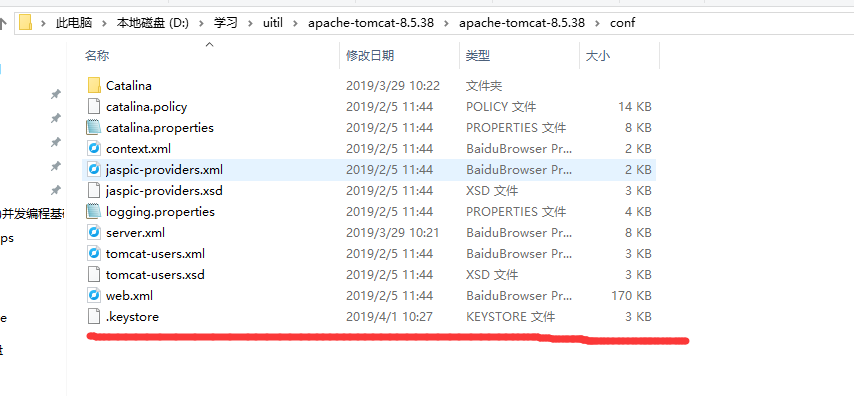
命令执行完之后，操作系统的用户文件夹下面就会生成一个.keystore文件：

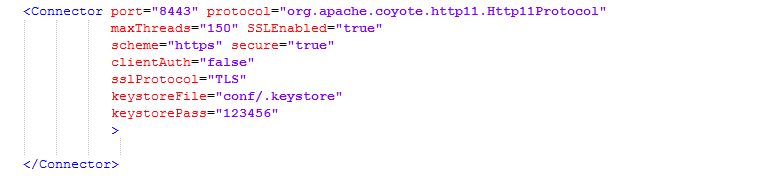
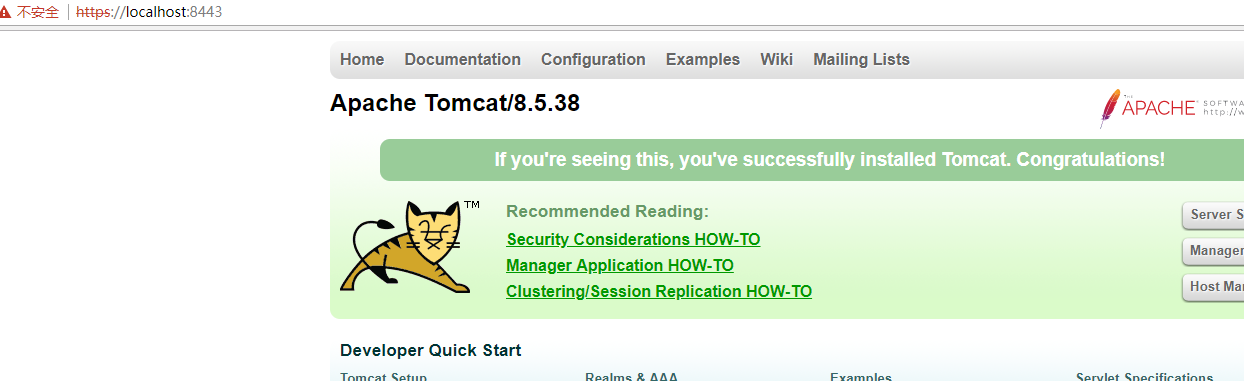
使用命令：**keytool -list -keystore .keystore**

查看.keystore密钥库里面的所有证书

### 配置https连接器

1. **将生成的.keystore密钥库文件拷贝到Tomcat服务器的conf目录下：**



1. **修改server.xml文件，配置https连接器：**
2. **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。**
3. **使用"https://localhost:8443/"访问8443的加密连接器**

### 安装数字证书

# 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。**

## HTTP协议的版本

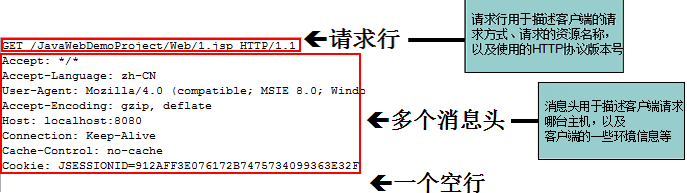
**HTTP/1.0 :** 客户端与web服务器建立连接后，只能获得一个web资源

**HTTP/1.1：**允许客户端与web服务器建立连接后，在一个连接上获取多个web资源。

## HTTP请求

**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**



### 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  
　　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，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，多个数据之间以&进行分隔，例如：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。**

### 消息头

HTTP请求中的常用消息头:

**Accept:** 浏览器通过这个头告诉服务器，它所支持的数据类型。

**Accept-Charset：**浏览器告诉服务器，它支持哪种字符集

**Accept-Encoding:** 浏览器告诉服务器，它的语言环境

**Host:** 告诉服务器，想访问哪台主机

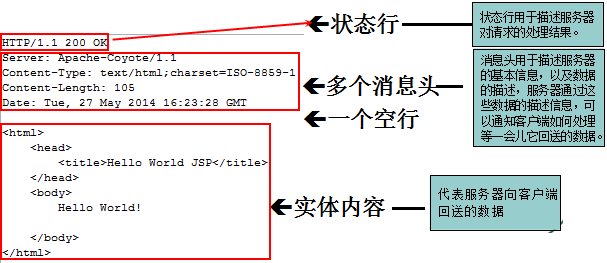
**If-Modified-Since：**浏览器告诉服务器，缓存数据的时间

**Referer:** 告诉服务器，客户机是哪个页面来的，防盗链

**Connection**: 告诉服务器,请求完后是断开链接还是保持链接

## HTTP响应

**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 。



### 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，



### 响应头

**HTTP响应中的常用响应头(消息头):**

**Location: 告诉浏览器跳转到哪里。**

**Server: 告诉浏览器服务器的型号。**

**Content-Encoding: 告诉浏览器，数据的压缩格式**

**Content-Language: 告诉浏览器，语言环境**

**Content-Type: 告诉浏览器回送数据的类型**

**Refresh： 告诉浏览器定时刷新**

**Content-Disposition: 告诉浏览器以下载方式打开数据**

**Transfer-Encoding：告诉浏览器数据是以分块方式回送的**

**Expires: -1 控制浏览器不要缓存**

**Cache-Control: no-cache**

**Pragma: no-cache**

## 服务端实例

### 设置Location响应头，实现请求重定向

**package** servlet;  
  
**import** javax.servlet.ServletException;  
**import** javax.servlet.http.HttpServlet;  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpServletResponse;  
**import** java.io.IOException;  
  
  
**public class** LocationDemo **extends** HttpServlet {  
  
 @Override  
 **public void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  
 *// 设置状态码为302* resp.setStatus(302);  
 resp.setHeader(**"Location"**, **"/index.jsp"**);  
 }  
  
 @Override  
 **public void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  
 **this**.doPost(req, resp);  
 }  
}

### 设置Content-Encoding响应头,指定压缩格式

**package** servlet;  
  
**import** javax.servlet.ServletException;  
**import** javax.servlet.http.HttpServlet;  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpServletResponse;  
**import** java.io.ByteArrayOutputStream;  
**import** java.io.IOException;  
**import** java.util.zip.GZIPOutputStream;  
  
**public class** ContentEncodingDemo **extends** HttpServlet {  
  
 @Override  
 **public void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  
 String data = **"abcdefgabcdefgabcdefgabcdefgabcdefgabcdefg"** +  
 **"abcdefgabcdefgabcdefgabcdefgabcdefgabcdefgabcde"** +  
 **"fgabcdefgabcdefgabcdefgabcdefgabcdefgab"**;  
  
 ByteArrayOutputStream out = **new** ByteArrayOutputStream();  
 GZIPOutputStream gout = **new** GZIPOutputStream(out);  
 gout.write(data.getBytes());  
 gout.close();  
  
 **byte**[] da = out.toByteArray();  
 resp.setHeader(**"Content-Encoding"**, **"gzip"**);  
 resp.setHeader(**"Content-Length"**, da.**length** + **""**);  
  
 resp.getOutputStream().write(da);  
 }  
  
 @Override  
 **public void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  
 **this**.doPost(req, resp);  
 }  
}

### 设置Content-Type响应头，指定回送数据类型

**package** servlet;  
  
**import** javax.servlet.ServletException;  
**import** javax.servlet.http.HttpServlet;  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpServletResponse;  
**import** java.io.IOException;  
**import** java.io.InputStream;  
  
**public class** ContentTypeDemo **extends** HttpServlet {  
  
 @Override  
 **public void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  
 resp.setHeader(**"Content-Type"**, **"image/jpeg"**);  
 InputStream in = **this**.getServletContext().getResourceAsStream(**"/static/b.jpg"**);  
 **byte**[] buff = **new byte**[1024];  
 **int** len = 0;  
 **while** ((len = in.read(buff)) != -1){  
 resp.getOutputStream().write(buff, 0, len);  
 }  
 }  
  
 @Override  
 **public void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  
 **this**.doPost(req, resp);  
 }  
  
}

### 设置refresh响应头，让浏览器定时刷新

**package** servlet;  
  
**import** javax.servlet.ServletException;  
**import** javax.servlet.http.HttpServlet;  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpServletResponse;  
**import** java.io.IOException;  
**import** java.util.concurrent.atomic.AtomicInteger;  
  
**public class** RefreshDemo **extends** HttpServlet {  
  
 **private** AtomicInteger **count** = **new** AtomicInteger(0);  
  
 @Override  
 **public void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  
 *// 设置浏览器3秒刷新一次  
 ///resp.setHeader("refresh", "3");  
  
 //3秒后跳转到http://baidu.com* resp.setHeader(**"refresh"**, **"3;url=http://baidu.com"**);  
 resp.getWriter().write(**"wo shi zhu!+ "** + **count**.incrementAndGet());  
 }  
  
 @Override  
 **public void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  
 **this**.doPost(req, resp);  
 }  
  
}

### 设置content-disponstion响应头，下载文件

**package** servlet;  
  
**import** javax.servlet.ServletException;  
**import** javax.servlet.http.HttpServlet;  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpServletResponse;  
**import** java.io.IOException;  
**import** java.io.InputStream;  
  
**public class** DispositionDemo **extends** HttpServlet {  
  
 @Override  
 **public void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  
 resp.setHeader(**"Content-Disposition"**, **"attachment;filename=aa.jpg"**);  
  
 InputStream in = **this**.getServletContext().getResourceAsStream(**"/static/b.jpg"**);  
 **byte**[] buff = **new byte**[1024];  
 **int** len = 0;  
 **while** ((len = in.read(buff)) != -1){  
 resp.getOutputStream().write(buff, 0, len);  
 }  
 }  
  
 @Override  
 **public void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  
 **this**.doPost(req, resp);  
 }  
  
}

# Servlet开发

## 简介

Servlet是sun公司提供的一门用于开发动态web资源的技术。  
　　Sun公司在其API中提供了一个servlet接口，用户若想用发一个动态web资源(即开发一个Java程序向浏览器输出数据)，需要完成以下2个步骤：  
　　1、编写一个Java类，实现servlet接口。  
　　2、把开发好的Java类部署到web服务器中。  
　　按照一种约定俗成的称呼习惯，通常我们也把实现了servlet接口的java程序，称之为Servlet

## C:\Users\002559.XHH\Downloads\未命名文件 (3).jpgServlet运行流程

**Servlet程序是由web服务器进行调用，web服务器收到Servlet的访问请求后：**

1. Web服务器首先会检查web服务器容器中是否已装载并创建了该Servlet的实例,如果装直接执行第4步
2. 装载该Servlet并创建Servlet的实例
3. 调用Servlet对象的init()方法
4. 创建一个用于封装HTTP请求消息的HttpServletRequest对象和一个代表Http响应消息的HtttpServletReponse对象，然后调用Servlet的Servie()方法并将请求和响应对象作为参数传递进去
5. WEB应用程序被停止或重新启动之前，Servlet引擎将卸载Servlet，并在卸载之前调用Servlet的destroy()方法.

**注意：如果再web.xml中配置load-on-startup=1，（默认为0）则在应用启动时就会创建一个servlet对象**

## Servlet重点对象

### ServletConfig对象

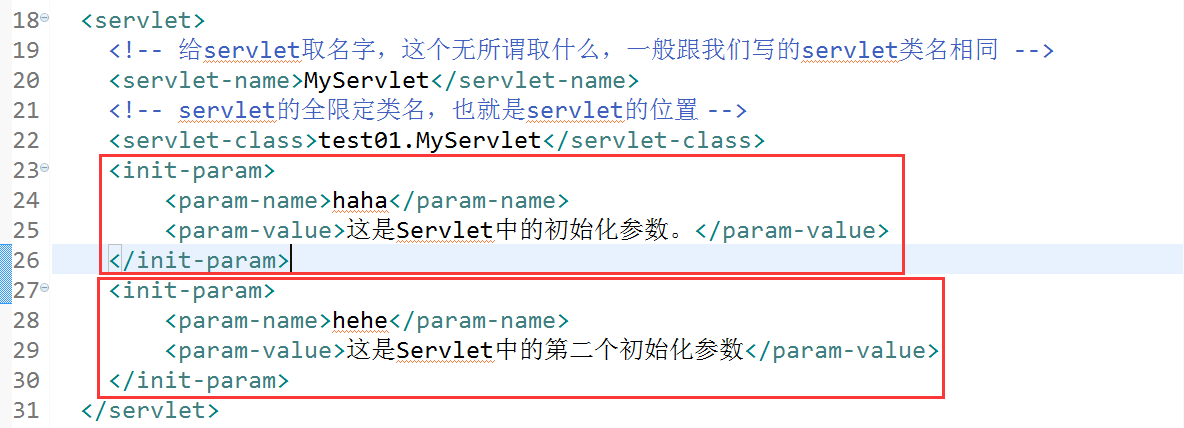
#### 获取途径

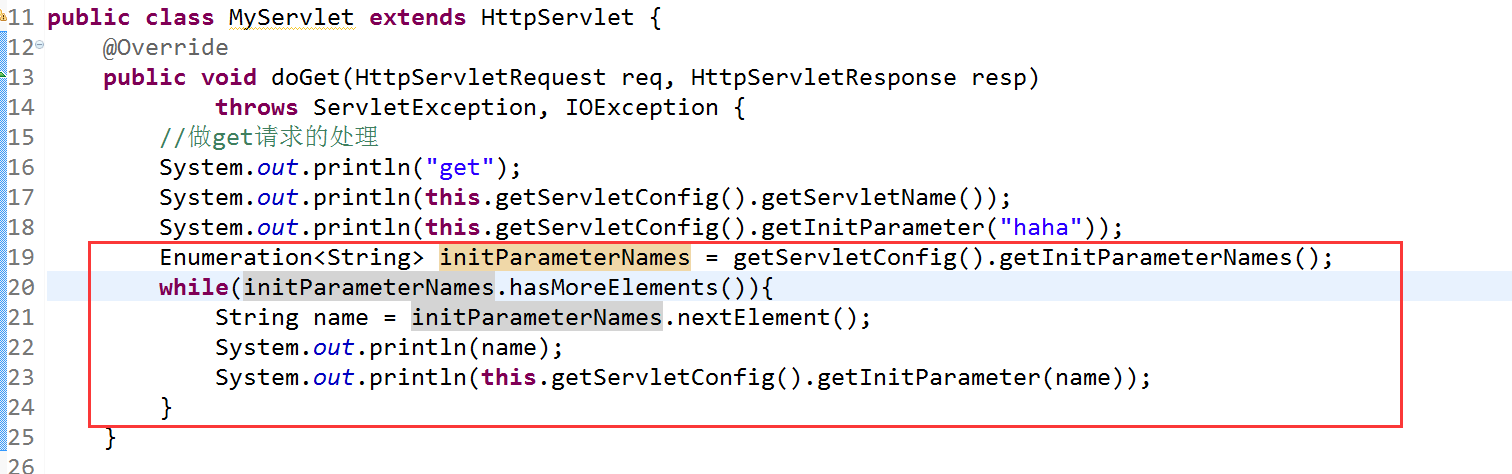
**getServletConfig();**

#### 方法

* **getServletName();** ：获取Servlet的名称，也就是我们在web.xml中配置的servlet-name
* **getServletContext();**
* **getInitParameter(String) :** 获取在servlet中初始化参数的值。这里注意与全局初始化参数的区分。这个获取的只是在该servlet下的初始化参数

**getInitParameterNames();**  : 获取在Servlet中所有初始化参数的名字，也就是key值，可以通过key值，来找到各个初始化参数的value值。注意返回的是枚举类型





### ServletContext对象

　 WEB容器在启动时，它会为每个WEB应用程序都创建一个对应的ServletContext对象，它代表当前web应用。  
　　ServletConfig对象中维护了ServletContext对象的引用，开发人员在编写servlet时，可以通过ServletConfig.getServletContext方法获得ServletContext对象。  
　　由于一个WEB应用中的所有Servlet共享同一个ServletContext对象，因此Servlet对象之间可以通过ServletContext对象来实现通讯。ServletContext对象通常也被称之为context域对象。

#### 获取途径

* **getServletContext();**
* **getServletConfig().getServletContext();**

#### 方法

1. **web项目中共享数据**

**setAttribute(String name, Object obj) :** 在web项目范围内存放内容，以便让在web项目中所有的servlet读能访问到

**getAttribute(String name) :** 通过指定名称获得内容

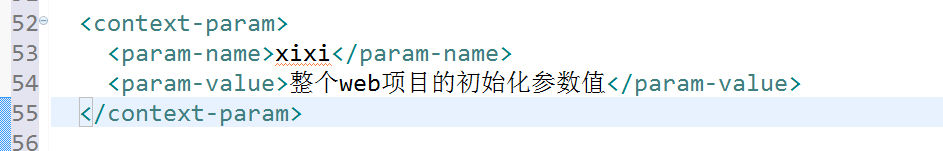
**removeAttribute(String name) :** 通过指定名称移除内容

1. **整个web项目初始化参数 //这个就是全局初始化参数，每个Servlet中都能获取到该初始化值**

**getInitPatameter(String name)　:** 通过指定名称获取初始化值

**getInitParameterNames() :** 获得枚举类型

web.xml 配置 整个web项目的初始化：



1. **获取web项目资源**

**getServletContext().getRealPath("/WEB-INF/web.xml") :** 获取web项目下指定资源的路径：

**InputStream getResourceAsStream(java.lang.String path) ：**获取web项目下指定资源的内容，返回的是字节输入流。

### 3. 实例

#### 1. 通过ServletConfig获取Servlet初始化实例

**package** servlet;  
  
**import** javax.servlet.ServletConfig;  
**import** javax.servlet.ServletException;  
**import** javax.servlet.http.HttpServlet;  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpServletResponse;  
**import** java.io.IOException;  
**import** java.util.Enumeration;  
  
*/\*\*  
 \* 获取Servlet初始化参数  
 \*/***public class** ServletConfigDemo **extends** HttpServlet{  
  
 @Override  
 **public void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  
 ServletConfig config = **this**.getServletConfig();  
 *// 获取单个参数* String str = config.getInitParameter(**"drive"**);  
 System.***out***.println(str);  
 *// 获取全部参数* Enumeration<String> enumeration = config.getInitParameterNames();  
 **while** (enumeration.hasMoreElements()){  
 String paramName = enumeration.nextElement();  
 System.***out***.println(paramName + **" : "** + config.getInitParameter(paramName));  
 }  
 }  
  
 @Override  
 **public void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  
 **this**.doPost(req, resp);  
 }  
}

#### 2. 通过ServletContext实现多个Servlet之间数据共享

#### 3. 通过ServletContext实现请求转发

#### 4. 使用ServletContext获取资源文件

**package** servlet;  
  
**import** javax.servlet.ServletException;  
**import** javax.servlet.http.HttpServlet;  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpServletResponse;  
**import** java.io.IOException;  
**import** java.io.InputStream;  
**import** java.util.Properties;  
  
**public class** ServletContextDemo3 **extends** HttpServlet {  
  
 @Override  
 **public void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  
 *// 读取静态资源, 同时像页面输出三个图片只会显示一张  
 /\*resp.setHeader("Content-Type", "images/jpeg");  
 readStaticFile(resp, "a.jpg");  
 readStaticFile(resp, "b.jpg");  
 readStaticFile(resp, "d.jpg");\*/  
 // 读取属性文件* readResources(resp, **"context.properties"**);  
 }  
  
 @Override  
 **public void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  
 **this**.doPost(req, resp);  
 }  
  
  
 */\*\*  
 \* 读取Resources包下面的资源文件  
 \*/* **private void** readResources(HttpServletResponse response, String resName) **throws** IOException {  
 InputStream in = **this**.getServletContext().getClassLoader().getResourceAsStream(**"/WEB-INF/classes/resources/"** + resName);  
 Properties prop = **new** Properties();  
 prop.load(in);  
 String str = prop.getProperty(**"raoxianlong"**);  
 response.getWriter().write(str);  
 }  
  
 */\*\*  
 \* 读取WebRoot目录下的static静态文件  
 \** ***@param* response** *\** ***@param* resName** *\*/* **private void** readStaticFile(HttpServletResponse response, String resName) **throws** IOException {  
 InputStream in = **this**.getServletContext().getResourceAsStream(**"/static/"** + resName);  
  
 **byte**[] buffer = **new byte**[1024];  
 **int** len;  
 **while** ((len = in.read(buffer)) != -1){  
 response.getOutputStream().write(buffer, 0, len);  
 }  
 }  
  
  
}

# Response & Requeset

## Response

Web服务器收到客户端的http请求，会针对每一次请求，分别创建一个用于代表请求的request对象、和代表响应的response对象。  
request和response对象即然代表请求和响应，那我们要获取客户机提交过来的数据，只需要找request对象就行了。要向客户机输出数据，只需要找response对象就行了。

### 常用方法

#### 负责向客户端发送数据的相关方法

* **getOutputStream() :** 返回一个ServletOutputStream，负责向客户端写数据
* **getWriter() :** 返回一个PrintWriter输出字符数据，对中文支持比较好

#### 负责向客户端发送响应头的相关方法

* **addHeader() :** 添加一个响应头
* **setHeader()**

#### 负责向客户端发送响应状态码

* **setStatus();**

#### 状态码常量

**404 :**

**C:\Users\002559.XHH\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\FEFC190C.tmp200:**

**C:\Users\002559.XHH\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\F0AAD118.tmp500:**

**C:\Users\002559.XHH\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\29420EE6.tmp**

#### 设置编码的相关方法

**setHeader(”Content-Type”, “text/html;charset=UTF-8”) : 告诉浏览器数据已哪种展示**

**setCharacterEncoding(“UTF-8”) ：告诉浏览器**

### 实例应用

#### 使用PrintWriter向页面输出中文数据

#### 文件下载

**spackage** servlet;  
  
**import** javax.servlet.ServletException;  
**import** javax.servlet.http.HttpServlet;  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpServletResponse;  
**import** java.io.FileInputStream;  
**import** java.io.IOException;  
**import** java.io.InputStream;  
**import** java.io.PrintWriter;  
**import** java.net.URLEncoder;  
  
**public class** ResponseDemo **extends** HttpServlet {  
  
 @Override  
 **public void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
 *// 1. 设置编码传输中文  
 /\*response.setCharacterEncoding("UTF-8");  
 response.setHeader("content-type", "text/html;charset=UTF-8");  
 PrintWriter writer = response.getWriter();  
 String dataStr = "饶先龙是猪!";  
 writer.write(dataStr);\*/  
 // 2. 文件下载* String filePath = **this**.getServletContext().getRealPath(**"/static/学习目标.txt"**);  
 *// 获取文件名* String fileName = filePath.substring(filePath.lastIndexOf(**"\\"**) + 1);  
 System.***out***.println(**"文件名称:"** + fileName);  
 *// 设置下载文件以及文件名称* response.setHeader(**"content-disposition"**, **"attachment;filename="** + URLEncoder.*encode*(fileName, **"UTF-8"**));  
  
 InputStream in = **new** FileInputStream(filePath);  
 **byte**[] buffer = **new byte**[1024];  
 **int** len;  
 **while** ((len = in.read(buffer)) != -1){  
 response.getOutputStream().write(buffer, 0 , len);  
 }  
 }  
  
 @Override  
 **public void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  
 **this**.doPost(req, resp);  
 }  
  
}

#### 生成验证码

1. **package** servlet;  
     
   **import** javax.imageio.ImageIO;  
   **import** javax.servlet.ServletException;  
   **import** javax.servlet.http.HttpServlet;  
   **import** javax.servlet.http.HttpServletRequest;  
   **import** javax.servlet.http.HttpServletResponse;  
   **import** java.awt.\*;  
   **import** java.awt.image.BufferedImage;  
   **import** java.io.IOException;  
   **import** java.util.Random;  
     
   */\*\*  
    \* 验证码  
    \*/***public class** VerificationCode **extends** HttpServlet {  
     
    **private final** String **codes** = **"qwertyuiopasdfghjklzxcvbnm123456789"**;  
    **private final int size** = 4;  
     
    @Override  
    **public void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
    BufferedImage image = **new** BufferedImage(200, 60, BufferedImage.***TYPE\_INT\_RGB***);  
    Graphics2D g = (Graphics2D) image.getGraphics();  
    g.setColor(Color.***CYAN***); *// 设置背景颜色* g.fillRect(0, 0, 200, 60);  
    g.setColor(Color.***BLACK***); *// 设置字体颜色* g.setFont(**new** Font(**null**, Font.***BOLD***, 40));  
    g.drawString(makeCode(), 5, 40);  
     
    response.setHeader(**"refresh"**, **"5"**);  
    response.setContentType(**"image/jpeg"**);  
    *//5.设置响应头控制浏览器不缓存图片数据* response.setDateHeader(**"expries"**, -1);  
    response.setHeader(**"Cache-Control"**, **"no-cache"**);  
    response.setHeader(**"Pragma"**, **"no-cache"**);  
    ImageIO.*write*(image,**"jpg"**, response.getOutputStream());  
    }  
     
    @Override  
    **public void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  
    **this**.doPost(req, resp);  
    }  
     
    **private** String makeCode(){  
    StringBuffer buffer = **new** StringBuffer();  
    Random random = **new** Random();  
    **int** length = **codes**.length();  
    **for** (**int** i = 0; i < **size**; i++) {  
    buffer.append(**" "** + **codes**.charAt(random.nextInt(length)));  
    }  
    **return** buffer.toString();  
    }  
     
     
     
   }

## Request

HttpServletRequest对象代表客户端的请求，当客户端通过HTTP协议访问服务器时，HTTP请求头中的所有信息都封装在这个对象中，通过这个对象提供的方法，可以获得客户端请求的所有信息。

### 常用方法

#### 获取请求基本信息

**getRequestURL : 方法返回客户端发出请求时的完整URL。**  
**getRequestURI ： 方法返回请求行中的资源名部分。**  
**getQueryString ： 方法返回请求行中的参数部分。**  
　　getPathInfo ： 方法返回请求URL中的额外路径信息。额外路径信息是请求URL中的位于Servlet的路径之后和查询参数之前的内容，它以“/”开头。  
　　**getRemoteAddr：方法返回发出请求的客户机的IP地址。**  
　　getRemoteHost : 方法返回发出请求的客户机的完整主机名。  
　　getRemotePort ： 方法返回客户机所使用的网络端口号。  
　　getLocalAddr : 方法返回WEB服务器的IP地址。  
　　getLocalName : 方法返回WEB服务器的主机名。

#### 获取请求头信息

getHeader(string name)方法:String   
　　getHeaders(String name)方法:Enumeration   
　　getHeaderNames()方法

#### 获取客户端提交的参数(请求参数)

getParameter(String)方法**(常用)**

getParameterValues(String name)方法**(常用)**

getParameterNames()方法(**不常用**)

getParameterMap()方法**(编写框架时常用)**

### 实例应用

#### 获取请求信息

**package** request;  
  
**import** javax.servlet.ServletException;  
**import** javax.servlet.http.HttpServlet;  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpServletResponse;  
**import** java.io.IOException;  
**import** java.io.PrintWriter;  
  
**public class** RequestDemo **extends** HttpServlet {  
  
 @Override  
 **public void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
 response.setCharacterEncoding(**"UTF-8"**);  
 response.setHeader(**"content-type"**, **"text/html;charset=UTF-8"**);  
 PrintWriter print = response.getWriter();  
 print.write(printRequestInfo(request.getRequestURL().toString(), **"客户端发出请求时的完整URL"**));  
 print.write(printRequestInfo(request.getRequestURI(), **"请求行中资源名部分"**));  
 print.write(printRequestInfo(request.getQueryString(), **"请求参数"**));  
 print.write(printRequestInfo(request.getPathInfo(), **"额外参数信息"**));  
 print.write(printRequestInfo(request.getRemoteAddr(), **"客户端IP"**));  
 print.write(printRequestInfo(request.getRemoteHost(), **"客户端主机"**));  
 print.write(printRequestInfo(request.getRemotePort() + **""**, **"客户端的端口"**));  
 print.write(printRequestInfo(request.getLocalAddr() + **""**, **"WEB服务器IP地址"**));  
 print.write(printRequestInfo(request.getLocalPort() + **""**, **"WEB服务器端口"**));  
 print.write(printRequestInfo(request.getAuthType() + **""**, **"AuthType"**));  
 print.write(printRequestInfo(request.getRemoteUser() + **""**, **"RemoteUser"**));  
 }  
  
 @Override  
 **public void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  
 **this**.doPost(req, resp);  
 }  
  
 **private** String printRequestInfo(String info, String remark){  
 **return** remark+ **": "** + info + **"<br>"**;  
 }  
  
}

#### 获取请求头信息

**package** request;  
  
**import** javax.servlet.ServletException;  
**import** javax.servlet.http.HttpServlet;  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpServletResponse;  
**import** java.io.IOException;  
**import** java.io.PrintWriter;  
**import** java.util.Enumeration;  
  
*/\*\*  
 \* 获取请求头信息  
 \*/***public class** RequestDemo1 **extends** HttpServlet {  
  
 @Override  
 **public void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
 response.setCharacterEncoding(**"UTF-8"**);  
 response.setHeader(**"content-type"**, **"text/html;charset=UTF-8"**);  
 PrintWriter print = response.getWriter();  
  
 Enumeration<String> headerNames = request.getHeaderNames();  
 **while** (headerNames.hasMoreElements()){  
 String headName = headerNames.nextElement();  
 print.write(printRequestInfo(request.getHeader(headName), headName));  
 }  
  
 }  
  
 @Override  
 **public void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  
 **this**.doPost(req, resp);  
 }  
  
 **private** String printRequestInfo(String info, String remark){  
 **return** remark+ **": "** + info + **"<br>"**;  
 }  
  
}

## 乱码解决方案

　 1、如果提交方式为post，想不乱码，只需要在服务器端设置request对象的编码即可，客户端以哪种编码提交的，服务器端的request对象就以对应的编码接收，比如客户端是以UTF-8编码提交的，那么服务器端request对象就以UTF-8编码接收(**request.setCharacterEncoding("UTF-8")**)

2、如果提交方式为get，设置request对象的编码是无效的，request对象还是以默认的ISO8859-1编码接收数据，因此要想不乱码，只能在接收到数据后再手工转换，步骤如下：

1).获取获取客户端提交上来的数据，得到的是乱码字符串,data="???è?????"

String data = request.getParameter("paramName");

2).查找ISO8859-1码表，得到客户机提交的原始数据的字节数组

byte[] source = data.getBytes("ISO8859-1");

3).通过字节数组以指定的编码构建字符串，解决乱码

data = new String(source, "UTF-8");

## 转发和重定向

**请求转发：指一个web资源收到客户端请求后，通知服务器去调用另外一个web资源进行处理。**  
　　请求转发的应用场景：MVC设计模式

　　在Servlet中实现请求转发的两种方式：

**1、通过ServletContext的getRequestDispatcher(String path)方法，该方法返回一个RequestDispatcher对象，调用这个对象的forward方法可以实现请求转发。**

**例如：将请求转发的test.jsp页面**

RequestDispatcher reqDispatcher =this.getServletContext().**getRequestDispatcher**("/test.jsp");

reqDispatcher.forward(request, response);

**2、通过request对象提供的getRequestDispatche(String path)方法，该方法返回一个RequestDispatcher对象，调用这个对象的forward方法可以实现请求转发。**

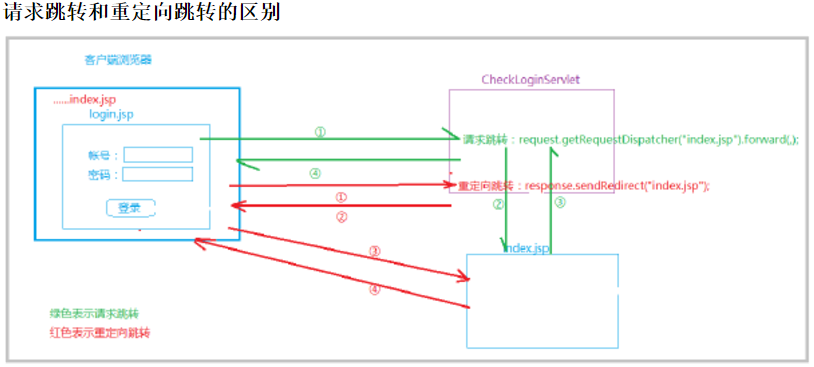
**例如：将请求转发的test.jsp页面**

request.**getRequestDispatcher**("/test.jsp").forward(request, response);

request对象同时也是一个域对象(Map容器)，开发人员通过request对象在实现转发时，把数据通过request对象带给其它web资源处理。

**请求转发：**

HttpServletResponse.sendRedirect方法对浏览器的请求直接作出响应，响应的结果就是告诉浏览器去重新发出对另外一个URL的 访问请求，这个过程好比有个绰号叫“浏览器”的人写信找张三借钱，张三回信说没有钱，让“浏览器”去找李四借，并将李四现在的通信地址告诉给了“浏览器”。于是，“浏览器”又按张三提供通信地址给李四写信借钱，李四收到信后就把钱汇给了“浏览器”。可见，“浏览器”一共发出了两封信和收到了两次回复， “浏览器”也知道他借到的钱出自李四之手。RequestDispatcher.forward方法在服务器端内部将请求转发给另外一个资源，浏览器只知道发出了请求并得到了响应结果，并不知道在服务器程序内部发生了转发行为。这个过程好比绰号叫“浏览器”的人写信找张三借钱，张三没有钱，于是张三找李四借了一些钱，甚至还可以加上自己的一些钱，然后再将这些钱汇给了“浏览器”。可见，“浏览器”只发 出了一封信和收到了一次回复，他只知道从张三那里借到了钱，并不知道有一部分钱出自李四之手。

一个web资源收到客户端请求后，**通知服务器去调用另外一个web资源**进行处理，称之为请求转发/307。  
　　一个web资源收到客户端请求后，**通知浏览器去访问另外一个web资源**进行处理，称之为请求重定向/302。

# Cookie

## 概念

会话可简单理解为：用户开一个浏览器，点击多个超链接，访问服务器多个web资源，然后关闭浏览器，整个过程称之为一个会话。  
　　有状态会话：一个同学来过教室，下次再来教室，我们会知道这个同学曾经来过，这称之为有状态会话。

## JavaWeb保存会话数据的两种技术

### Cookie

服务器端将用户的数据以cookie的形式写给用户各自的浏览器。当用户再次访问服务器中的web资源时，就会带着各自的数据区。

### Session

Session是服务端技术，服务器在运行时可以为每一个用户创建一个其独享的session对象，关于某个用户的数据就会存在这个它单独的session对象中，当用户访问服务器的web资源时，就可以取到各自的session对象中的数据

## Java提供的操作Cookie的API

　Java中的javax.servlet.http.Cookie类用于创建一个Cookie

　response接口也中定义了一个addCookie方法，它用于在其响应头中增加一个相应的Set-Cookie头字段。 同样，request接口中也定义了一个getCookies方法，它用于获取客户端提交的Cookie。



## Cookie注意细节

* 一个Cookie只能标识一种信息，它至少含有一个标识该信息的名称（NAME）和设置值（VALUE）。
* 浏览器一般只允许存放300个Cookie，每个站点最多存放20个Cookie，每个Cookie的大小限制为4KB。
* **如果创建了一个cookie，并将他发送到浏览器，默认情况下它是一个会话级别的cookie（即存储在浏览器的内存中），用户退出浏览器之后即被删除。若希望浏览器将该cookie存储在磁盘上，则需要使用maxAge，并给出一个以秒为单位的时间。将最大时效设为0则是命令浏览器删除该cookie。**
* **删除cookie时，path必须一致，否则不会删除**
* cookie中存取中文：

要想在cookie中存储中文，那么必须使用URLEncoder类里面的encode([String](eclipse-javadoc:%E2%98%82=JavaWeb_Cookie_Study_20140715/D:%5C/MyEclipse10%5C/Common%5C/binary%5C/com.sun.java.jdk.win32.x86_1.6.0.013%5C/jre%5C/lib%5C/rt.jar%3Cjava.net(URLEncoder.class%E2%98%83URLEncoder~encode~Ljava.lang.String;~Ljava.lang.String;%E2%98%82String) s, [String](eclipse-javadoc:%E2%98%82=JavaWeb_Cookie_Study_20140715/D:%5C/MyEclipse10%5C/Common%5C/binary%5C/com.sun.java.jdk.win32.x86_1.6.0.013%5C/jre%5C/lib%5C/rt.jar%3Cjava.net(URLEncoder.class%E2%98%83URLEncoder~encode~Ljava.lang.String;~Ljava.lang.String;%E2%98%82String) enc)方法进行中文转码，例如：

Cookie cookie = new Cookie("userName", URLEncoder.encode("龙凤呈祥", "UTF-8"));

response.addCookie(cookie);

在获取cookie中的中文数据时，再使用URLDecoder类里面的decode([String](eclipse-javadoc:%E2%98%82=JavaWeb_Cookie_Study_20140715/D:%5C/MyEclipse10%5C/Common%5C/binary%5C/com.sun.java.jdk.win32.x86_1.6.0.013%5C/jre%5C/lib%5C/rt.jar%3Cjava.net(URLDecoder.class%E2%98%83URLDecoder~decode~Ljava.lang.String;~Ljava.lang.String;%E2%98%82String) s, [String](eclipse-javadoc:%E2%98%82=JavaWeb_Cookie_Study_20140715/D:%5C/MyEclipse10%5C/Common%5C/binary%5C/com.sun.java.jdk.win32.x86_1.6.0.013%5C/jre%5C/lib%5C/rt.jar%3Cjava.net(URLDecoder.class%E2%98%83URLDecoder~decode~Ljava.lang.String;~Ljava.lang.String;%E2%98%82String) enc)进行解码，例如：

URLDecoder.decode(cookies[i].getValue(), "UTF-8")

## 实例

### 记录上一次用户访问时间