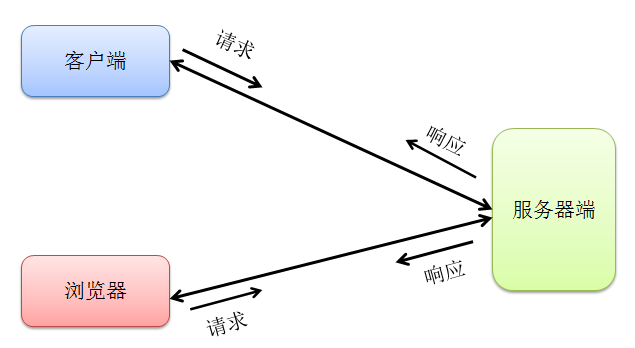
# Java Web 技术栈

**讲师：刘峰吉**

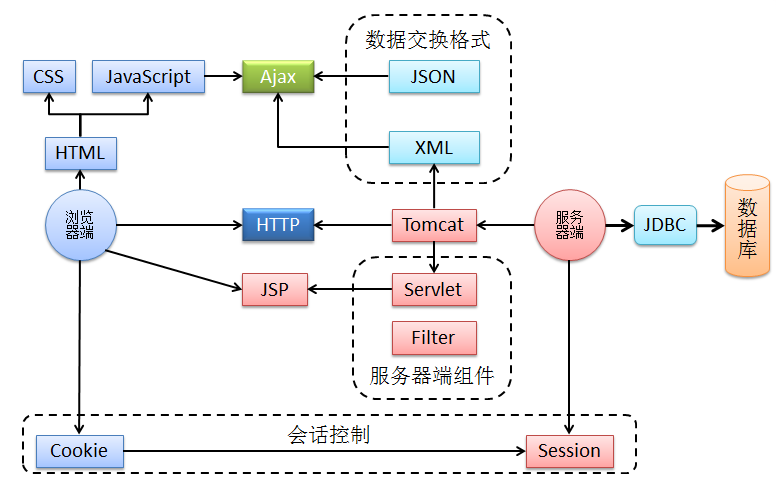
# 一、Java Web 概述

## 1、软件架构：B/S、C/S



## 2、Java Web 相关技术

Java Web，是用Java技术来解决相关web互联网领域的技术总和。包括：web客户端 和 web浏览器端。

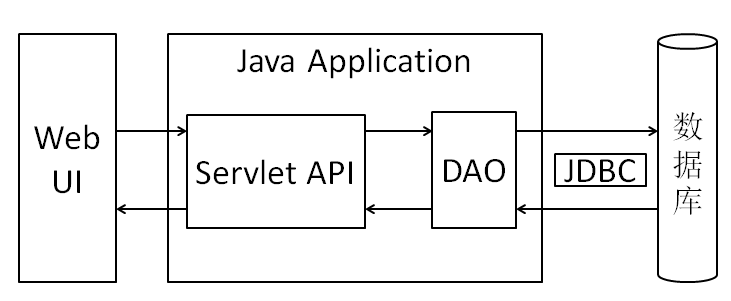


Tomcat是可以运行JSP 、Servlet、Filter、 这样的程序的，因为Tomcat已经实现了JavaEE 里面的一些规范，Java EE 的工程是主要就是写JSP 、Servlet、Filter这些，tomcat符合JavaEE的规范，所以可以运行JSP 、Servlet、Filter，这些都是Java代码，Java代码必须放到虚拟机里面运行，所以Tomcat必须依赖于虚拟机，才能启动，才能运行我们的程序

异步通信，以流的形式返回数据到页面，现在基本都是使用json

同步通信，以转发或者重定向的方式跳转到页面

## 3、项目体系结构



# 二、开发工具

## 1、HBuilder

HBuilder 官网

https://www.dcloud.io/hbuilderx.html

HBuilderX，简称HX，是轻如编辑器、强如IDE的合体版本

它令我们爱不释手，希望你也会喜欢上这款免费工具。

HBuilder是DCloud（数字天堂）推出的一款支持HTML5的Web开发IDE。 [1] HBuilder的编写用到了Java、C、Web和Ruby。HBuilder本身主体是由Java编写。

它基于Eclipse，所以顺其自然地兼容了Eclipse的插件。 [2]

软件名称 HBuilder 开发商 DCloud（数字天堂） 软件平台 windows,mac 软件语言 中文

特点快，是HBuilder的最大优势，通过完整的语法提示和代码输入法、代码块等，大幅提升HTML、js、css的开发效率。

## 2、EditPlus

EditPlus是一款由韩国 Sangil Kim （ES-Computing）出品的Windows端小巧但是功能强大的可处理文本、HTML和程序语言的文本编辑器，具有内置FTP、FTPS和sftp功能.虽然它可以作为一个很好的记事本替代品，但它也为网页作者和程序员提供了许多强大的特性。EditPlus也提供网页作家及程序设计师许多强悍的功能。EditPlus默认支持HTML，CSS，PHP，ASP，Perl，C / C ++，Java，javascript和VBScript的强大且可自定义的语法高亮显示。程序内嵌网页浏览器，其它功能还包含 FTP 功能、HTML 编辑、URL 突显、自动完成、剪贴文本、行列选择、强大的搜索与替换、多重撤销/重做、拼写检查、自定义快捷键，等等。你甚至可以通过设置用户工具将其作为C,Java,Php等等语言的一个简单的IDE。HTML编辑器和程序员编辑器。虽然它可以作为一个很好的记事本替代品，但它也为网页作者和程序员提供了许多强大的功能。 此外，您可以创建自己的语法文件以支持其他编程语言。因此，它支持用户定义的工具，帮助文件和按键记录文件

## 3、Eclipse

Eclipse 是一个开放源代码的、基于Java的可扩展开发平台。就其本身而言，它只是一个框架和一组服务，用于通过插件组件构建开发环境。幸运的是，Eclipse 附带了一个标准的插件集，包括Java开发工具（Java Development Kit，JDK）

# 三、UI（Web01\_UI）

version:2.5

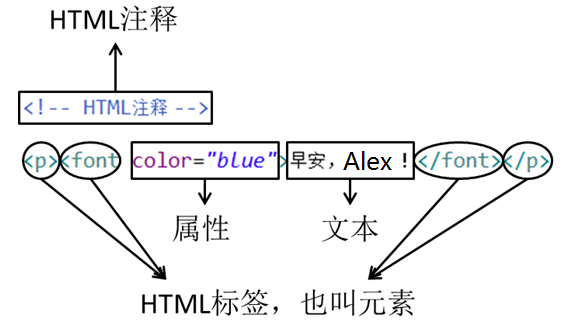
## 1、HTML

### 1）HTML概述

（1）超文本标记语言：Hyper Text Markup Language

（2）HTML文件由一组标签构成，经过浏览器解析之后可以显示网页内容。

（3）HTML文档的构成



### 2）基本语法规范等

基本语法规范及具体的使用见工程：模板 -> Web01\_UI



### 3）HTML 语法

grammar.html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>Insert title here</title>  </head>  <body>  <!-- ①标签不能交叉嵌套 -->  正确：<p><i>早安，Alex</i></p>  错误：<p><i>早安，Alex</p></i>    <!-- ②标签必须正确关闭 -->  <!-- i.有文本内容的标签： -->  正确：<p>早安，Alex</p>  错误：<p>早安，Alex    <!-- ii.没有文本内容的标签： -->  正确：aaaaaa<br />bbbbbbbb  错误：aaaaaa<br>bbbbbbbb    <!-- ③属性必须有值，属性值必须加引号 -->  正确：<font color="blue">早安，Alex</font>  错误：<font color>早安，Alex</font>  错误：<font color=red>早安，Alex</font>    <!-- ④注释不能嵌套 -->  正确：<!-- 注释内容 -->  错误：<!-- 注释内容<!-- 嵌套的注释 --> 溢出的注释 -->    </body>  </html> |

### 4）表格

table.html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>Insert title here</title>  </head>  <body>  <table border="1" align="center" width="80%">  <tr><th>姓名</th><th>位置</th><th>公司</th><th>财富</th></tr>  <tr>  <td>王健林</td>  <td>北京</td>  <td>万达</td>  <td>2211</td>  </tr>  <tr>  <td>马云</td>  <td>杭州</td>  <td>阿里巴巴</td>  <td>1889</td>  </tr>  <tr>  <td>马化腾</td>  <td>深圳</td>  <td>腾讯</td>  <td>1642</td>  </tr>  <tr>  <td>王卫</td>  <td>深圳</td>  <td>顺丰速运</td>  <td>1240</td>  </tr>  </table>  </body>  </html> |

### 5）超链接

#### （1）a.html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>Insert title here</title>  </head>  <body>  <a href="pages/target.html">跳转到目标页面，目标页面有美女！</a>  </body>  </html> |

#### （2）target.html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>Insert title here</title>  </head>  <body>  美女！  </body>  </html> |

### 6）表单

#### （1）form标签

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>Insert title here</title>  </head>  <body>  <!-- 创建表单 -->  <!-- 使用form标签定义表单  使用action属性指定提交数据的位置  使用method属性指定提交数据的方式  GET：会将提交的数据附着在URL地址后面，在浏览器地址栏可以看到  POST：不会提交的数据附着在URL地址后面，在浏览器地址栏看不到  -->  <form action="target.html" method="post">    用户名：<input type="text" name="userName" /><br /><br />  密码：<input type="password" name="password" /><br /><br />    <!-- 提交按钮：type="submit"，使用value属性指定它是页面上显示的文字 -->  <input type="submit" value="登录" />    </form>  </body>  </html> |

#### （2）文本框

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>Insert title here</title>  </head>  <body>  <!-- file:///D:/workspace/test/Web02\_UI/WebContent/04.表单/target.html?username=Tom&email=tom@126.com&password=abc123&other=%E6%97%A0-->  <form action="target.html" method="get">  用户名：<input type="text" name="username" /><br /><br />  邮箱：<input type="text" name="email" /><br /><br />  密码：<input type="text" name="password" /><br /><br />  其他：<input type="text" name="other" /><br /><br />  <input type="submit" value="登录" />  <br /><br />  </form>  </body>  </html> |

#### （3）密码框

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>Insert title here</title>  </head>  <body>  <form action="target.html" method="get">    密码：<input type="password" name="password" /><br /><br />    <input type="submit" value="登录" />    <br /><br />    </form>  </body>  </html> |

#### （4）单选框

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>Insert title here</title>  </head>  <body>  <form action="target.html" method="get">    性别：  <input type="radio" name="gender" value="male" />男生  <input type="radio" name="gender" checked="checked" value="female" />女生  <input type="radio" name="gender" value="eastSuccess" />东方不败<br /><br /><br /><br />    <input type="submit" value="提交" />    <br /><br />    </form>  </body>  </html> |

#### （5）多选框

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>Insert title here</title>  </head>  <body>  <form action="target.html" method="get">    你喜欢的足球队<br /><br />    巴西<input type="checkbox" name="soccerTeam" value="Brazil" />  德国<input type="checkbox" name="soccerTeam" value="German" />  荷兰<input type="checkbox" name="soccerTeam" value="Holland" /><br /><br /><br /><br />    <input type="submit" value="提交" />    <br /><br />    </form>  </body>  </html> |

target.html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>Insert title here</title>  </head>  <body>  Target Page...  </body>  </html> |

### 7）插入图片

image.html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>Insert title here</title>  </head>  <body>  <img alt="加载图片失败" src="img/a.jpg">  </body>  </html> |

### 8）列表

List.html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>Insert title here</title>  </head>  <body>  <!-- 在这里创建无序列表 -->  <ul>  <li>火影忍者</li>  <li>死神</li>  <li>海贼王</li>  <li>名侦探柯南</li>  <li>网球王子</li>  </ul>    <br /><br />    <!-- 在这里创建有序列表 -->  <ol>  <li>漩涡鸣人</li>  <li>黑崎一护</li>  <li>路飞</li>  <li>柯南</li>  <li>越前龙马</li>  </ol>  </body>  </html> |

### 9）标题段落

article.html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>Insert title here</title>  </head>  <body>  <!-- 在这里创建标题 -->  <h1>江畔独步寻花</h1>  <h3>唐·杜甫</h3>  <!-- 在这里创建段落 -->  <p>黄四娘家花满蹊</p>  <p>千朵万朵压枝低</p>  <p>留连戏蝶时时舞</p>  <p>自在娇莺恰恰啼</p>  </body>  </html> |

**尖叫提示：还有许多没有讲到，大家可以看W3School，讲到的这些都是最常用的**

## 2、CSS

### 1）CSS概述

在CSS技术诞生之前，页面上定义结构和显示效果的代码耦合在一起，开发和维护都非常繁琐。

为了将显示效果设置从HTML代码中抽离出来，并提供更强大的表现力，W3C组织制定了CSS标准——Cascading Style Sheet 层叠样式表。

### 2）CSS编写位置

#### （1）内部

标签的style属性中

<p style="color: red ; font-size: 12px">落霞与孤鹜齐飞，秋水共长天一色</p>

写到style标签中

<style type="text/css">

p {

color: blue;

background-color: yellow;

}

</style>

#### （2）外部

写在外部的css文件中，然后通过link标签引入外部的css文件

<link rel="stylesheet" type="text/css" href="style.css" />

**尖叫提示：注意优先级**

### 3）基本语法

选择器 {声明1;声明2}

选择器 {

样式名1:样式值1;

样式名2:样式值2;

}

### 4）选择器

（1）元素选择器

标签名

例子：

p{}

a{}

h1{}

（2） id选择器

#id

（3）类选择器

.class

（4）选择器分组

可以同时使用多个选择器

选择器1,选择器2,选择器N{}

**注意优先级**

### 5）声明

声明格式：样式名1:样式值1;

### 6）常用样式

字体颜色 color

宽度 width

高度 height

背景颜色 background-color

### 7）颜色

在CSS中我们使用RGB值来表示颜色。

RGB值可以写成一个三组两位十六进制数字，这个数字要求以#号开头

#红色绿色蓝色

000000 - ffffff

### 8）CSS 定义样式位置的案例

#### （1）css1.html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>Insert title here</title>  </head>  <body>  <p style="color: red ; font-size: 12px">落霞与孤鹜齐飞，秋水共长天一色</p>  </body>  </html> |

#### （2）css2.html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>Insert title here</title>  <style type="text/css">  p {  color: blue;  background-color: yellow;  }  </style>  </head>  <body>  <p>落霞与孤鹜齐飞，秋水共长天一色</p>  </body>  </html> |

#### （3）css3.html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>Insert title here</title>  <link rel="stylesheet" type="text/css" href="style.css" />  </head>  <body>  <p>落霞与孤鹜齐飞，秋水共长天一色</p>  </body>  </html> |

style.css

|  |
| --- |
| p {  color: blue;  background-color: yellow;  } |

### 9）CSS 选择器案例

#### （1）css1.html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>Insert title here</title>  <style type="text/css">  p {  color: blue;  background-color: yellow;  }  </style>  </head>  <body>  <p>落霞与孤鹜齐飞，秋水共长天一色</p>  <p>落霞与孤鹜齐飞，秋水共长天一色</p>  <p>落霞与孤鹜齐飞，秋水共长天一色</p>  <p>落霞与孤鹜齐飞，秋水共长天一色</p>  </body>  </html> |

#### （2）css2.html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>Insert title here</title>  <style type="text/css">  #p1 {  color: blue;  background-color: yellow;  }  </style>  </head>  <body>  <p id="p1">落霞与孤鹜齐飞，秋水共长天一色</p>  <p>落霞与孤鹜齐飞，秋水共长天一色</p>  <p>落霞与孤鹜齐飞，秋水共长天一色</p>  <p>落霞与孤鹜齐飞，秋水共长天一色</p>  </body>  </html> |

#### （3）css3.html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>Insert title here</title>  <style type="text/css">  .selectp {  color: blue;  background-color: yellow;  }  </style>  </head>  <body>  <p class="selectp">落霞与孤鹜齐飞，秋水共长天一色</p>  <p>落霞与孤鹜齐飞，秋水共长天一色</p>  <p class="selectp">落霞与孤鹜齐飞，秋水共长天一色</p>  <p>落霞与孤鹜齐飞，秋水共长天一色</p>  </body>  </html> |

#### （4）css4.html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>Insert title here</title>  <style type="text/css">  #p1,.selectp {  color: blue;  background-color: yellow;  }  </style>  </head>  <body>  <p id="p1">落霞与孤鹜齐飞，秋水共长天一色</p>  <p>落霞与孤鹜齐飞，秋水共长天一色</p>  <p class="selectp">落霞与孤鹜齐飞，秋水共长天一色</p>  <p class="selectp">落霞与孤鹜齐飞，秋水共长天一色</p>  </body>  </html> |

### 10）CSS颜色案例

#### （1）css1.html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>Insert title here</title>  <style type="text/css">  #p1,.selectp {  color: #000000;  background-color: #ff0000;  }  </style>  </head>  <body>  <p id="p1">落霞与孤鹜齐飞，秋水共长天一色</p>  <p>落霞与孤鹜齐飞，秋水共长天一色</p>  <p class="selectp">落霞与孤鹜齐飞，秋水共长天一色</p>  <p class="selectp">落霞与孤鹜齐飞，秋水共长天一色</p>  </body>  </html> |

**尖叫提示：还有许多没有讲到，大家可以看W3School，讲到的这些都是最常用的**

## 3、JavaScript

### 1）Java Script 概述

（1）Why?

拨号上网Time:

网速很慢，数据提交到服务器端验证，体验很差

JavaScript横空出世，担当了浏览器端数据验证的重任，从此一发不可收拾

五彩缤纷的现在:

时至今日JavaScript已经不仅仅局限于表单验证，她还能为网页添加各式各样的动态功能,为用户提供更流畅美观的浏览效果。

（2）What?

JavaScript是一门客户端脚本语言，主要运行在浏览器中，浏览器中负责运行JavaScript脚本代码的程序叫JavaScript引擎。

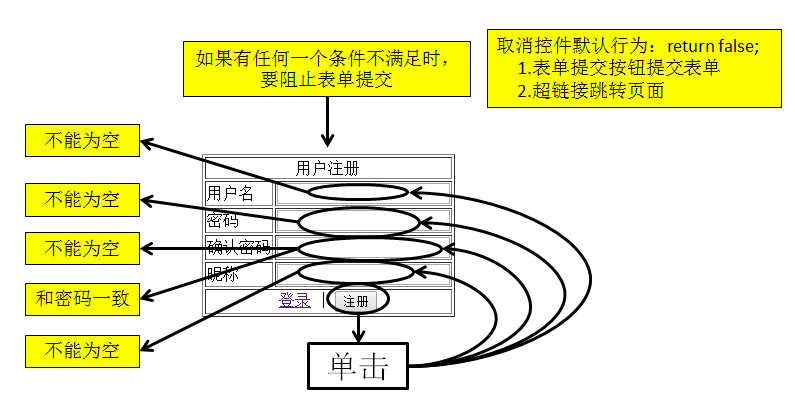
只有当浏览器读取javaScript代码时，才进行解释、执行。所以查看HTML源文件就能查看JavaScript源代码。

### 2）Java Script 案例





### 3）Java Script 练习



### 4）Java Script Hello World 案例

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  <!-- 1.JavaScript代码写在HTML中的script标签中 -->  <script type="text/javascript">    //2.借助window.onload事件在文档加载完成后执行程序  window.onload = function() {    //3.从文档中获取<button id="btnId">SayHello</button>标签对应的元素节点对象  var btnEle = document.getElementById("btnId");    //4.绑定单击事件响应函数  //给按钮对象的单击事件属性赋值一个函数的引用  btnEle.onclick = function() {    //5.在事件的响应函数中弹出警告框  window.alert("大数据专业");  };  };  </script>  </head>  <body>  <button id="btnId">SayHello</button>  </body>  </html> |

### 5）Java Script 基本语法案例

#### （1）变量

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  <script type="text/javascript">  //01.使用var关键字声明变量，声明时不需要指定类型  var a = 20;  var b = 'abc';  var c = true;    alert(a+" "+b+" "+c);    //02.变量在使用过程中可以接受不同类型的值  a = "Hello";  b = 230.53;  c = ["good",true,20];    alert(a+" "+b+" "+c);      </script>  </head>  <body>  </body>  </html> |

#### （2）函数

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  <script type="text/javascript">    //01.声明函数使用function关键字，不需要指定返回值类型，形参也不需要指定类型  function sum(a,b) {  return a+b;  }    //02.调用函数：不检查实参的类型，个数  var result = sum(5,10);  alert(result);    result = sum("Hello ","Tom");  alert(result);    result = sum("Hello ");  alert(result);  </script>  </head>  <body>  </body>  </html> |

#### （3）函数是对象

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>Insert title here</title>  <script type="text/javascript">  //在JavaScript中，函数也作为一种数据类型存在，而且是引用数据类型，函数名就是指向其内存空间地址的引用  function sum(a,b) {  return a+b;  }    alert(sum);    var ref01 = sum;  alert("ref01="+ref01);    var ref02 = sum;  alert("ref02="+ref01);  </script>  </head>  <body>  </body>  </html> |

#### （4）回调函数

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>Insert title here</title>  <script type="text/javascript">  //回调函数：声明后，不由自己调用，而是交给系统或其他函数执行的函数  function callBack() {  window.alert("I am call back!");  }    function call(fun) {  fun();  }    window.callBack();    call(callBack);  </script>  </head>  <body>  </body>  </html> |

#### （5）this关键字

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>Insert title here</title>  <script type="text/javascript">  //在JavaScript中，任何一个函数都是由对象调用的，在函数中可以通过this关键字引用调用这个函数的对象    var obj01 = {  name : "obj01 name",  getName : showName  };    var obj02 = {  name : "obj02 name",  getName : showName  };    function showName() {  alert(this.name);  }    //对象的属性指向了一个函数，所以也是函数的一个引用  //函数的引用加上()就是执行引用的函数  obj01.getName();  obj02.getName();    </script>  </head>  <body>  </body>  </html> |

#### （6）Object

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>Insert title here</title>  <script type="text/javascript">    //在JavaScript中创建对象  //1.new Object()  var objByNew = new Object();  objByNew.userName = "Tom2014";  objByNew.password = "123456";    alert(objByNew.userName+" "+objByNew.password);    //2.JSON格式  //{属性名01:属性值01,属性名02:属性值02,...}  var objByJSON = {  userName : "Jerry",  password : "654321"  };  alert(objByJSON.userName+" "+objByJSON.password);  </script>  </head>  <body>  </body>  </html> |

### 6）Java Script 事件案例

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  <script type="text/javascript">  /\*  地雷 事件监听和响应  生产 声明一个事件响应函数  找一个埋设的位置 从文档中获取一个按钮对象  埋设 将响应函数绑定到按钮对象上  触发引信爆炸 在浏览器捕获到用户对按钮的单击事件时执行响应函数  \*/  window.onload = function(){  //1.从文档中获取按钮对象  var btn = document.getElementById("btnId");  //2.声明一个事件响应函数  /\* function myClick() {  alert("Bom!");  } \*/    //3.将响应函数的引用赋值给按钮对象的onclick属性——绑定  /\* btn.onclick = myClick; \*/    btn.onclick = function () {  alert("Bom!~~~");  };    }  </script>  </head>  <body>  <button id="btnId">SayHello</button>  </body>  </html> |

### 7）Java Script 加载方式案例

#### （1）load1.html

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  </head>  <body>  <!-- 写在HTML标签内，结构与行为耦合，不推荐 -->  <button onclick="alert('hello~~~')">SayHello</button>  </body>  </html> |

#### （2）load2.html

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  <!-- 写在script标签内：无法获取button -->  <script type="text/javascript">  /\*  浏览器加载HTML文件时，是按照从上到下的顺序加载，如果遇到script标签，就停下来先执行script标签中的程序，  执行完程序以后，再回过头来继续加载后面的HTML标签  \*/  //3.使用DOM API从文档中获取id="btnId"按钮对象  var btn = document.getElementById("btnId");    alert(btn);    //4.给按钮对象绑定单击响应函数  btn.onclick = function() {    //5.在响应函数中弹出Hello  alert("Hello!~~~");    };  </script>  </head>  <body>  <button id="btnId">SayHello</button>  </body>  </html> |

#### （3）load3.html

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  </head>  <body>  <button id="btnId">SayHello</button>  </body>  <!-- 写在body标签后面，能获取到button，但不符合习惯 -->  <!--  类名：首字母大写，后面驼峰式命名  变量和方法：首字母小写，后面驼峰式命名  包：全部小写，单词用.分隔  -->  <script type="text/javascript">  //3.使用DOM API从文档中获取id="btnId"按钮对象  var btn = document.getElementById("btnId");  alert(btn);  //4.给按钮对象绑定单击响应函数  btn.onclick = function() {  //5.在响应函数中弹出Hello  alert("Hello!~~~");  };  </script>  </html> |

#### （4）load4.html

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  <!-- 使用window.onload -->  <script type="text/javascript">  //在整个文档加载完成后执行这个函数  function myPrograme() {  var btn = document.getElementById("btnId");    alert(btn);    //4.给按钮对象绑定单击响应函数  btn.onclick = function() {    //5.在响应函数中弹出Hello  alert("Hello!Success!~~~");    };  }    //将myPrograme绑定到“文档加载完成”这个事件上  //window代表当前浏览器窗口，它加载HTML文档完成后触发“文档加载完成”这个事件  window.onload = myPrograme;  </script>  </head>  <body>  <button id="btnId">SayHello</button>  </body>  </html> |

### 8）DOM

#### （1）dom.html

|  |
| --- |
| <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Untitled Document</title>  <link rel="stylesheet" type="text/css" href="style/css.css" />  <script type="text/javascript">    window.onload = function() {    var btn01Ele = document.getElementById("btn01");  btn01Ele.onclick = function() {    //查找北京这个城市  //<li id="bj">北京</li>  //查询元素时通常都是根据元素的特点进行获取  //在整个HTML文档中，id属性的值不能重复，否则只有第一个有效  //给元素节点对象的变量命名时，最好以Ele或Element结尾  var bjEle = document.getElementById("bj");  alert(bjEle.firstChild.nodeValue);  //<p><b>tt</b></p>    };    var btn02Ele = document.getElementById("btn02");  btn02Ele.onclick = function() {  //查找全部城市、游戏和手机  var liEleArr = document.getElementsByTagName("li");  //alert(liEleArr.length);  for(var i = 0; i < liEleArr.length; i++) {  alert(liEleArr[i].firstChild.nodeValue);  }  };    var btn03Ele = document.getElementById("btn03");  btn03Ele.onclick = function() {  //查找所有单选框  /\* gender:  <input type="radio" name="gender" value="male"/>  Male  <input type="radio" name="gender" value="female"/> \*/  var genderEleArr = document.getElementsByName("gender");  alert(genderEleArr.length);  };    var btn04Ele = document.getElementById("btn04");  btn04Ele.onclick = function(){  //读取文本框中的输入内容  var usernameEle = document.getElementById("username");  //读取属性值——元素对象.属性名  //文本框的输入内容可以通过value属性进行读写  alert(usernameEle.value);    };    var btn05Ele = document.getElementById("btn05");  btn05Ele.onclick = function(){  //设置文本框中的输入内容  var usernameEle = document.getElementById("username");  //设置属性值——元素对象.属性名=新值  //文本框的输入内容可以通过value属性进行读写  usernameEle.value = "new value";    };    var btn06Ele = document.getElementById("btn06");  btn06Ele.onclick = function() {  //读取北京这个城市的文本内容方式一  //文本节点对象和文本节点中的文本内容  //使用DOM标准中定义的API获取文本值  //1.获取文本节点父节点  //<li id="bj">北京</li>  var bjEle = document.getElementById("bj");    //2.调用父节点的 firstChild属性得到它的第一个子节点  //alert(bjEle.firstChild.nodeName);#text  var textNode = bjEle.firstChild;    //3.得到文本节点后再获取节点的值  var textContent = textNode.nodeValue;  alert("full:"+textContent);    //简化写法  alert("simple:"+bjEle.firstChild.nodeValue);    };      var btn07Ele = document.getElementById("btn07");  btn07Ele.onclick = function() {  //读取北京这个城市的文本内容方式二  //1.获取文本节点父节点  //<li id="bj">北京</li>  var bjEle = document.getElementById("bj");    //使用时注意：这是一个属性，不是方法，不要在后面加()  //alert("innerHTML="+bjEle.innerHTML);    var cityEle = document.getElementById("city");  alert(cityEle.innerHTML);  };    };      </script>  </head>  <body>  <div id="total">  <div class="inner">  <p>  你喜欢哪个城市?  </p>  <ul id="city">  <li id="bj">北京</li>  <li>上海</li>  <li>东京</li>  <li>首尔</li>  </ul>  <br>  <br>  <p>  你喜欢哪款单机游戏?  </p>  <ul id="game">  <li id="rl">红警</li>  <li>实况</li>  <li>极品飞车</li>  <li>魔兽</li>  </ul>  <br />  <br />  <p>  你手机的操作系统是?  </p>  <ul id="phone"><li>IOS</li><li id="android">Android</li><li>Windows Phone</li></ul>  </div>  <div class="inner">  gender:  <input type="radio" name="gender" value="male"/>  Male  <input type="radio" name="gender" value="female"/>  Female  <br>  <br>  name:  <input type="text" name="name" id="username" value="abcde"/>  </div>  </div>  <div id="btnList">  <div><button id="btn01">查找北京这个城市</button></div>  <div><button id="btn02">查找全部城市、游戏和手机</button></div>  <div><button id="btn03">查找所有单选框</button></div>  <div><button id="btn04">读取文本框中的输入内容</button></div>  <div><button id="btn05">设置文本框中的输入内容</button></div>  <div><button id="btn06">读取北京这个城市的文本内容方式一</button></div>  <div><button id="btn07">读取北京这个城市的文本内容方式二</button></div>  </div>  </body>  </html> |

#### （2）style.css

|  |
| --- |
| @CHARSET "UTF-8";  body {  width: 800px;  margin-left: auto;  margin-right: auto;  }  button {  width: 300px;  margin-bottom: 10px;  }  #btnList {  float:left;  }  #total{  width: 450px;  float:left;  }  ul{  list-style-type: none;  margin: 0px;  padding: 0px;  }  .inner li{  border-style: solid;  border-width: 1px;  padding: 5px;  margin: 5px;  background-color: #99ff99;  float:left;  }  .inner{  width:400px;  border-style: solid;  border-width: 1px;  margin-bottom: 10px;  padding: 10px;  float: left;  } |

# 四、Tomcat 服务器搭建（web02\_tomcat）

## 1、JDK

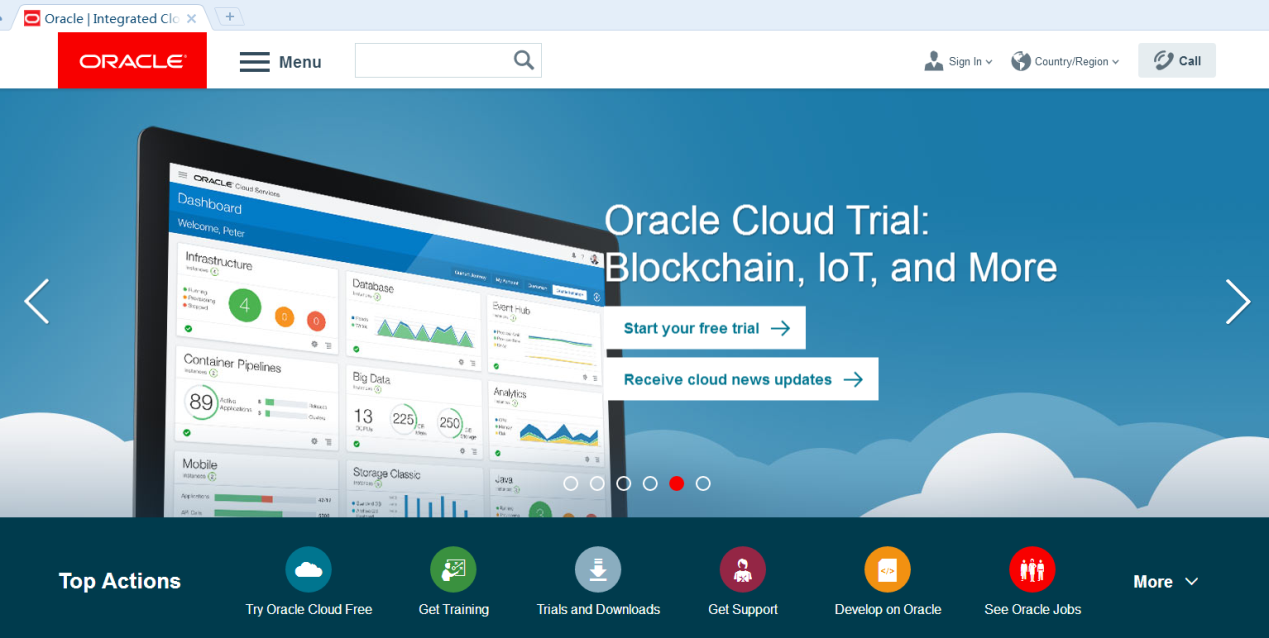
Tomcat是可以运行JSP 、Servlet、Filter、 这样的程序的，因为Tomcat已经实现了JavaEE 里面的一些规范，Java EE 的工程是主要就是写JSP 、Servlet、Filter这些，tomcat符合JavaEE的规范，所以可以运行JSP 、Servlet、Filter，这些都是Java代码，Java代码必须放到虚拟机里面运行，所以Tomcat必须依赖于虚拟机，才能启动，才能运行我们的程序

### 1）网址

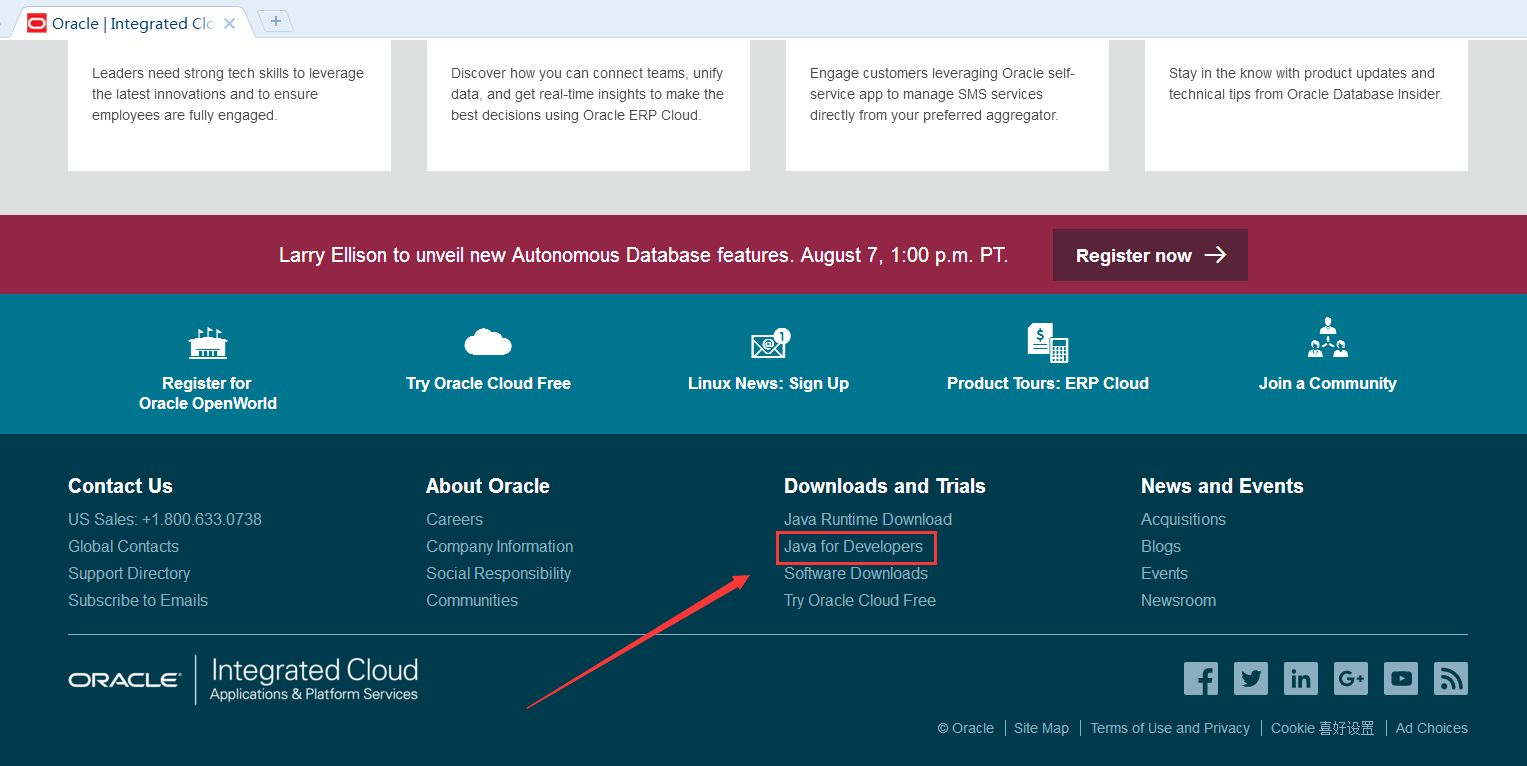
|  |
| --- |
| https://www.oracle.com/index.html |

### 2）下载

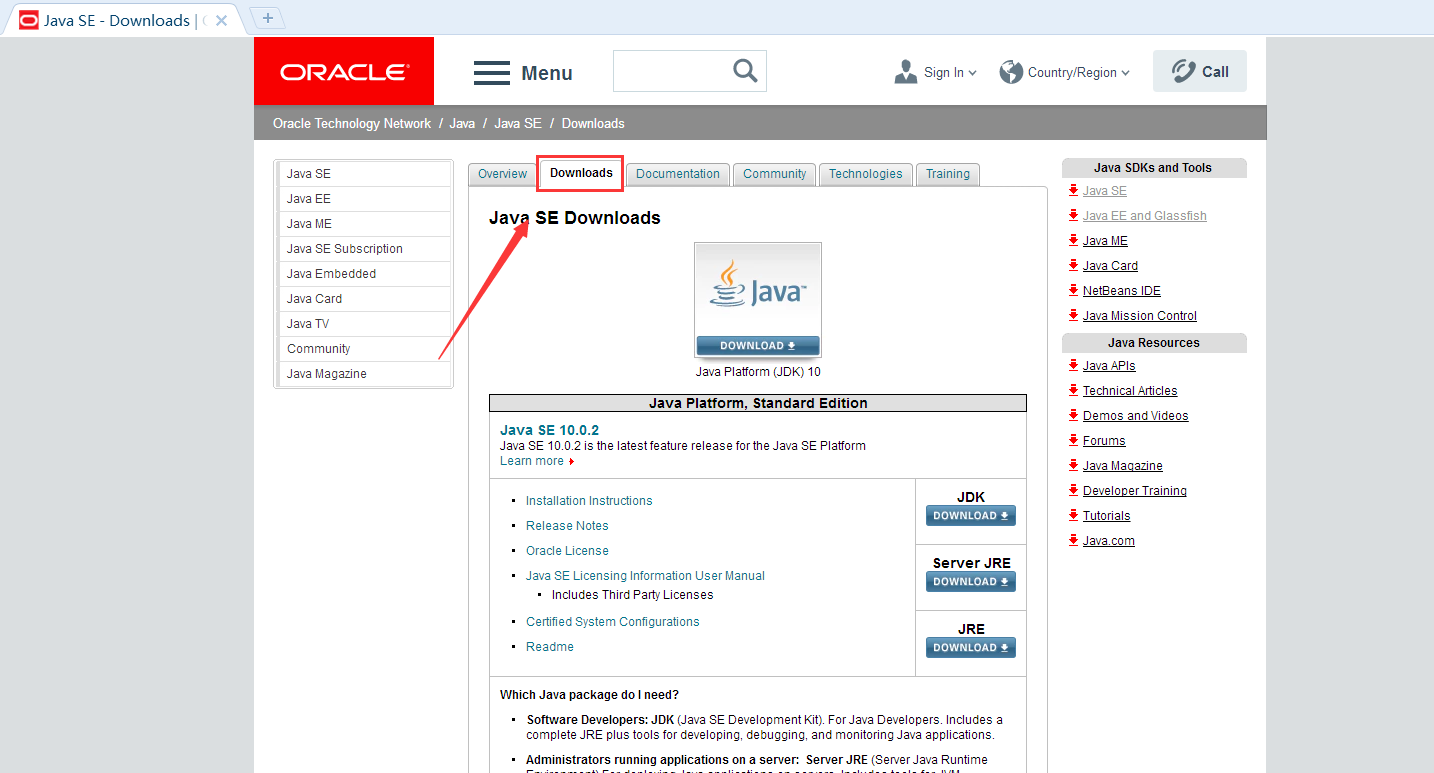
#### （1）首页



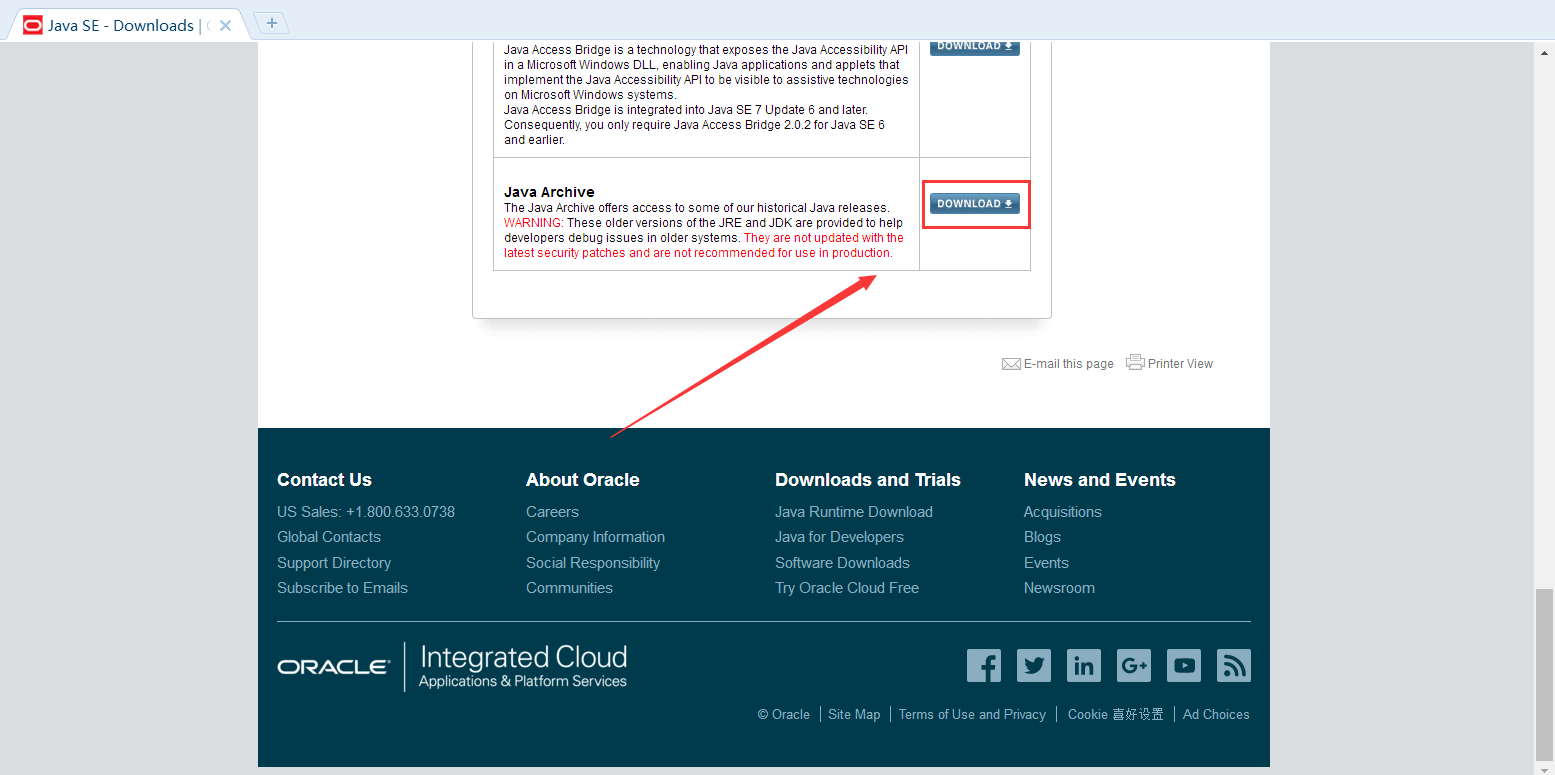
#### （2）点击Java for Developers



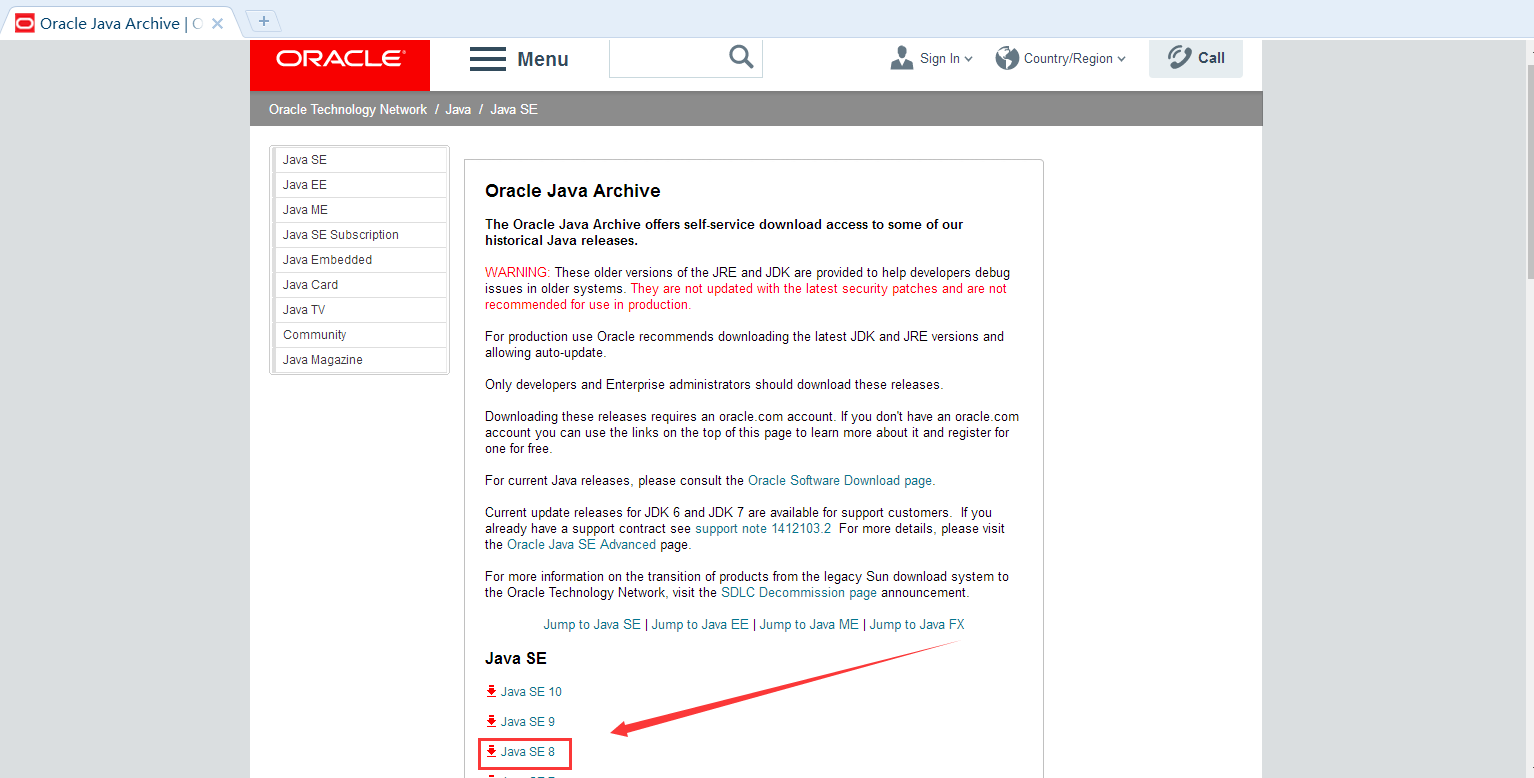
#### （3）点击Downloads



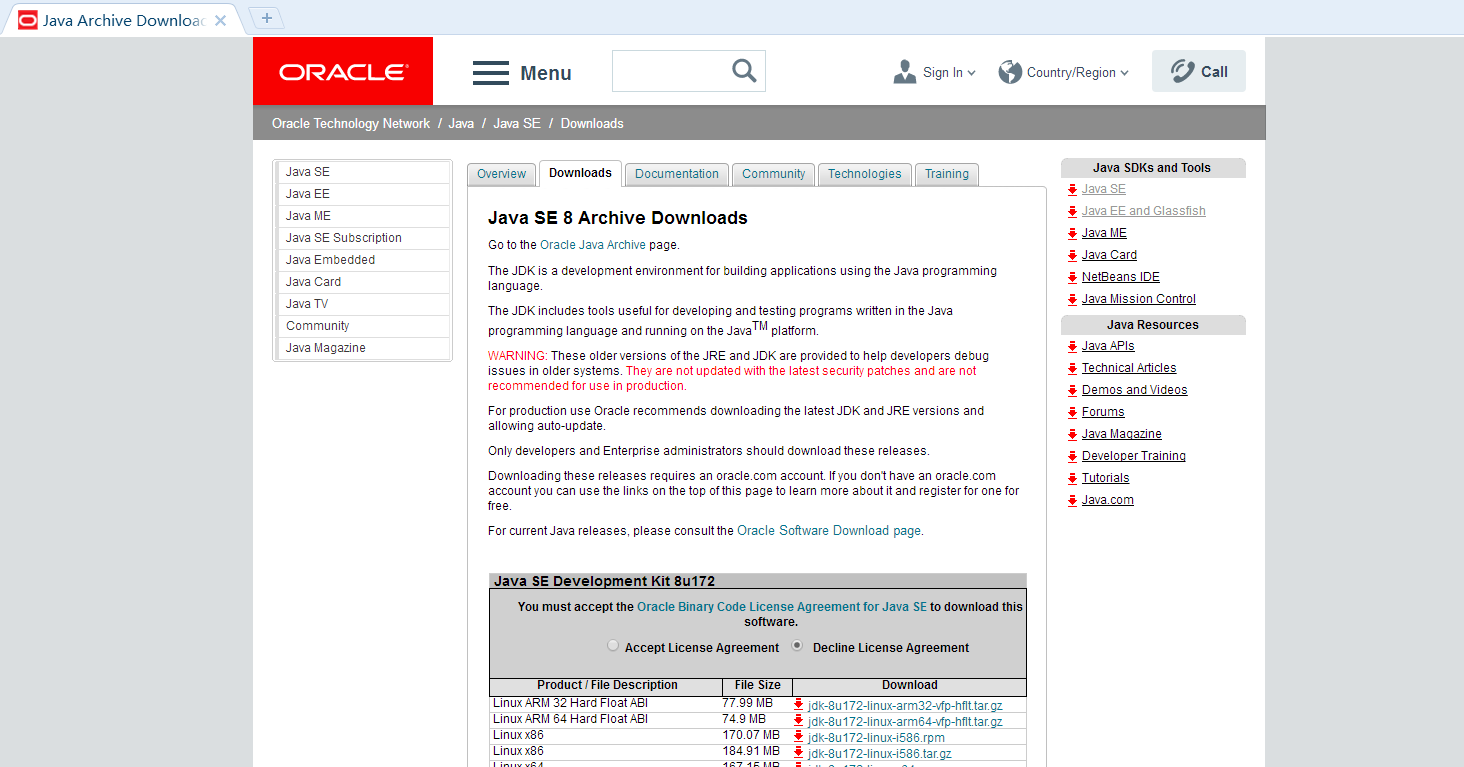
#### （4）Java Archive -> Download



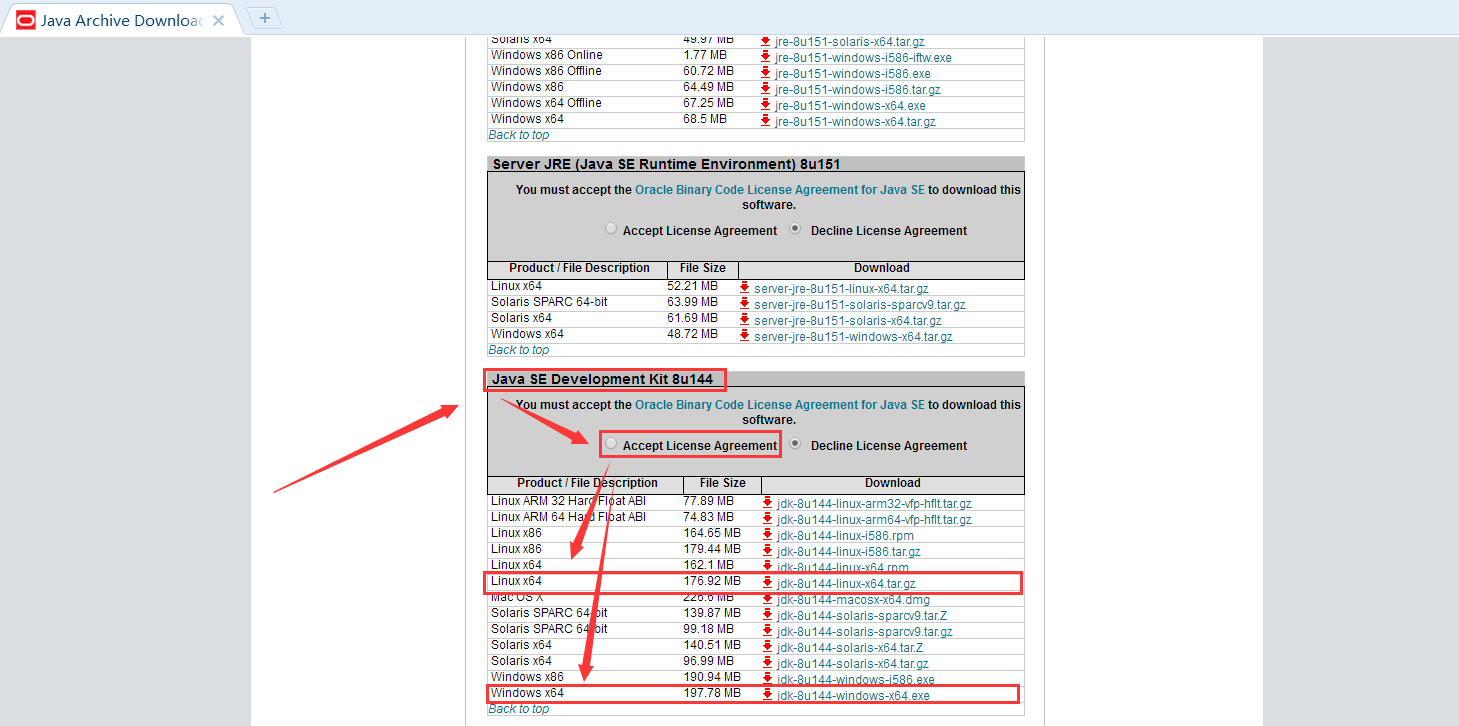
#### （5）Java SE 8



#### （6）Downloads



#### （7） Java SE Development Kit 8u144



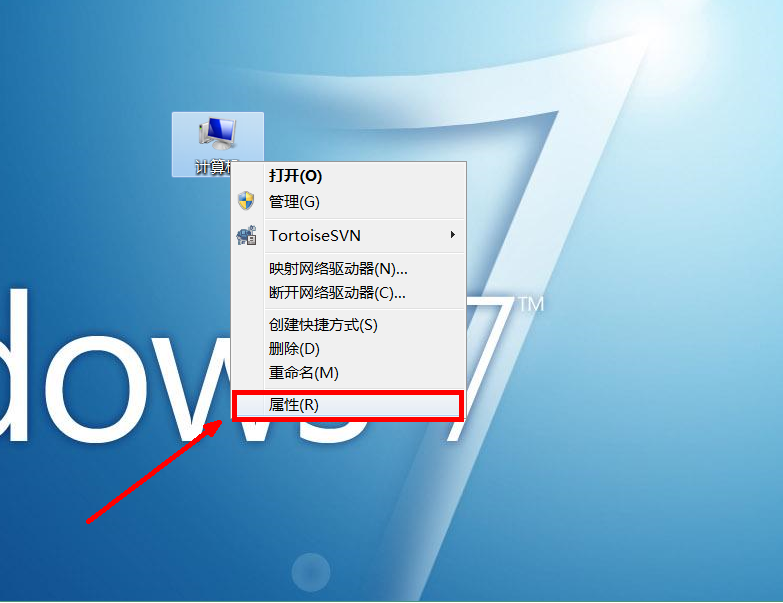
### 3）window Jdk1.8 环境变量配置

#### （1）思路

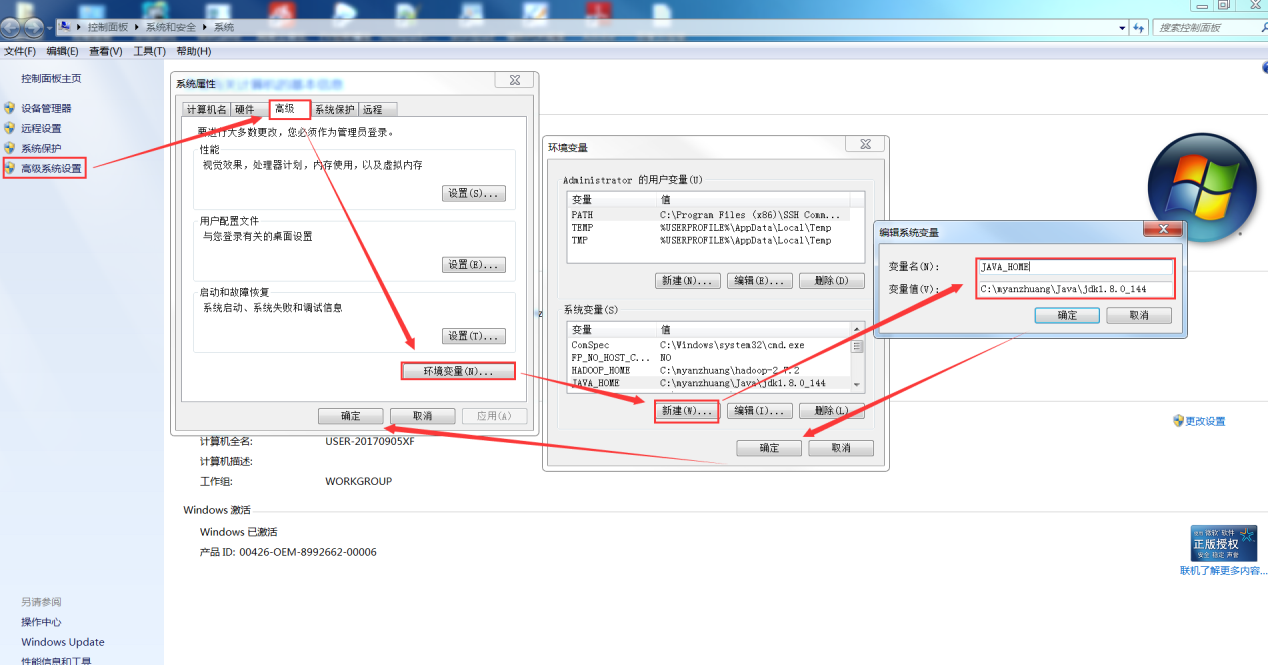
安装完JDK后，配置环境变量

计算机→属性→高级系统设置→高级→环境变量

#### （2）计算机右键，左键点击属性



#### （3）配置JAVA\_HOME

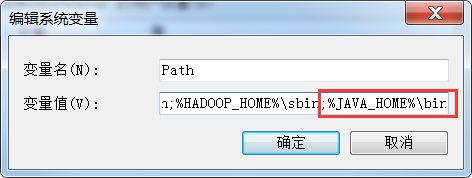


|  |
| --- |
| JAVA\_HOME  C:\myanzhuang\Java\jdk1.8.0\_144 |

#### （4）系统变量→寻找 Path 变量→编辑

在变量值最后输入 %JAVA\_HOME%\bin;%JAVA\_HOME%\jre\bin;

（注意原来Path的变量值末尾有没有;号，如果没有，先输入；号再输入上面的代码）

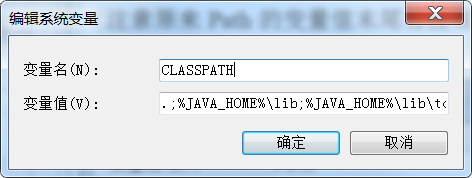


|  |
| --- |
| ;% JAVA\_HOME%\bin; |

#### （5）系统变量→新建 CLASSPATH 变量

变量值填写   .;%JAVA\_HOME%\lib;%JAVA\_HOME%\lib\tools.jar（注意最前面有一点）

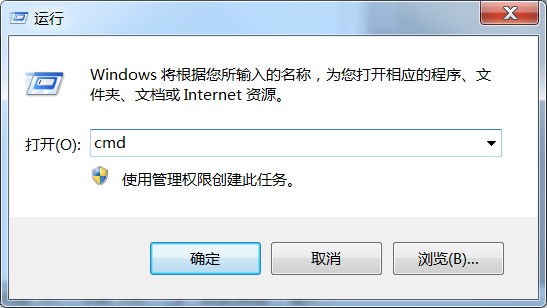
系统变量配置完毕

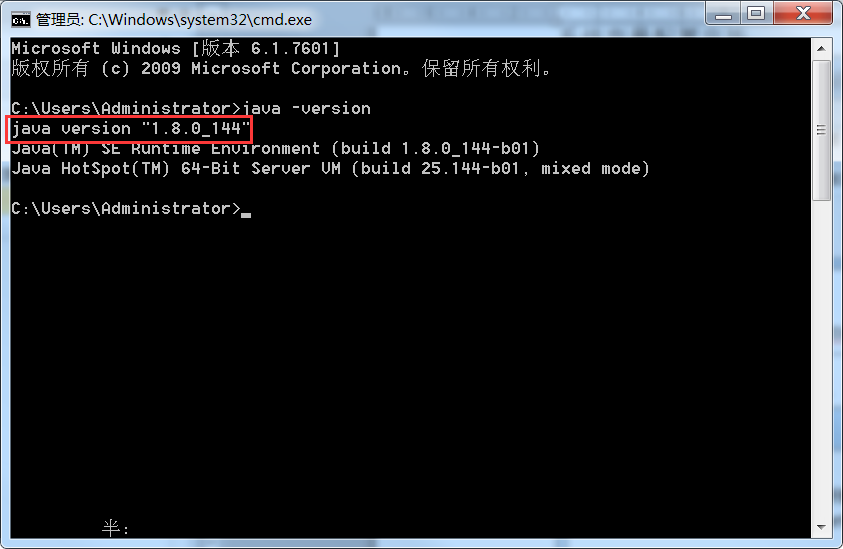


|  |
| --- |
| CLASSPATH  **.**;%JAVA\_HOME%\lib;%JAVA\_HOME%\lib\tools.jar |

#### （6）检验是否配置成功 运行cmd 输入 java -version （java 和 -version 之间有空格）

若如图所示 显示版本信息 则说明安装和配置成功。

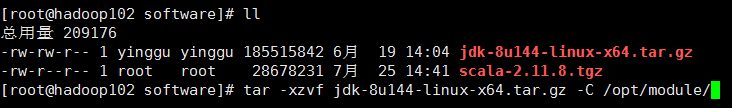




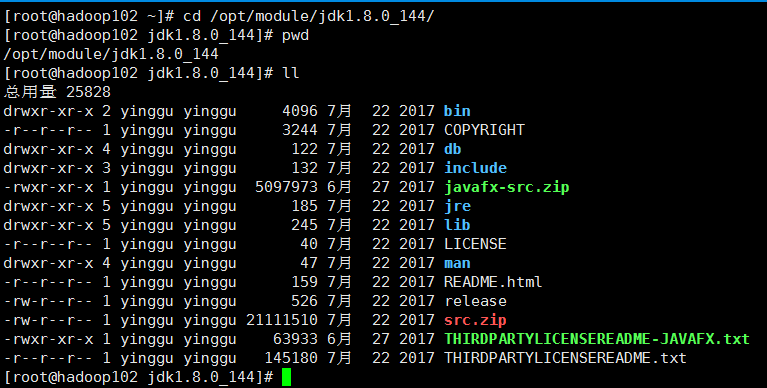
### 4）Linux Jdk1.8 环境变量配置

#### （1）上传到linux,使用tar命令解压jdk包

|  |
| --- |
| [root@hadoop102 software]# tar -xzvf jdk-8u144-linux-x64.tar.gz -C /opt/module/ |

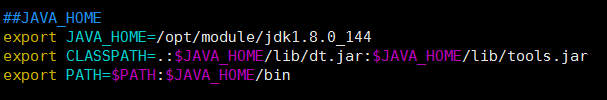


#### （2）查看解压后jdk



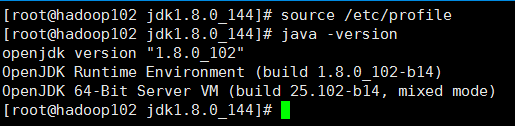
#### （3）编辑/etc/profile,配置jdk环境变量

|  |
| --- |
| [root@hadoop102 jdk1.8.0\_144]# vim /etc/profile  ##JAVA\_HOME  export JAVA\_HOME=/opt/module/jdk1.8.0\_144  export CLASSPATH=.:$JAVA\_HOME/lib/dt.jar:$JAVA\_HOME/lib/tools.jar  export PATH=$PATH:$JAVA\_HOME/bin |



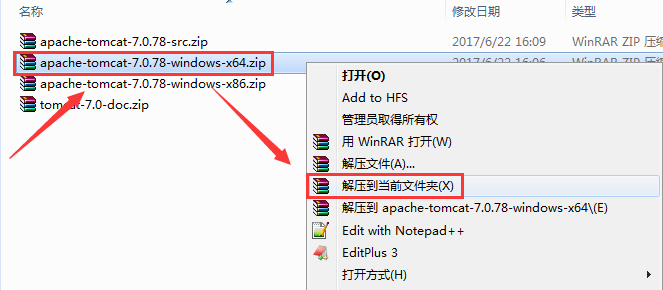
#### （4）查看jdk环境变量是否配置成功

|  |
| --- |
| [root@hadoop102 jdk1.8.0\_144]# source /etc/profile  [root@hadoop102 jdk1.8.0\_144]# java -version |

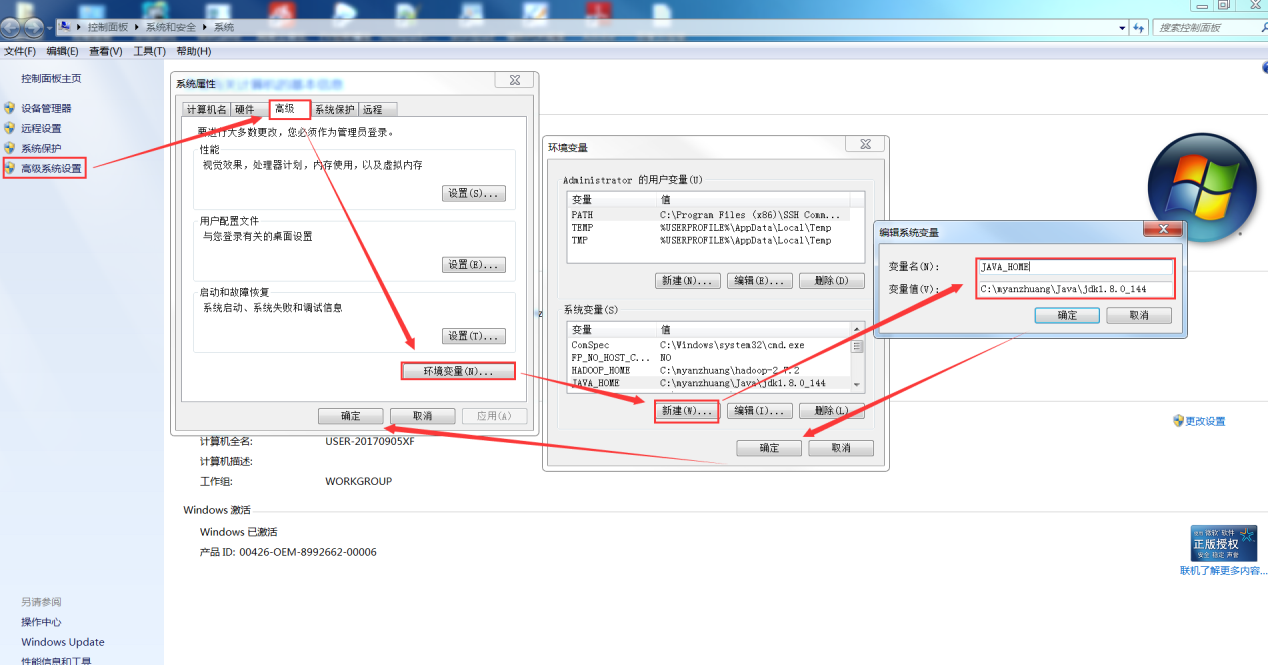


## 2、Tomcat

### 1）右键解压



### 2）配置CATALINA\_HOME



|  |
| --- |
| CATALINA\_HOME  C:\alexsoftware\apache-tomcat-8.5.23 |

### 3）系统变量→寻找 Path 变量→编辑

在变量值最后输入 %CATALINA\_HOME%\bin;

（注意原来Path的变量值末尾有没有;号，如果没有，先输入；号再输入上面的代码）

|  |
| --- |
| ;%CATALINA\_HOME%\bin; |

### 4）启动Tomcat （一定要配置环境变量的）

#### （1）方式一

Window + R -> cmd –> catalina run

#### （2）方式二

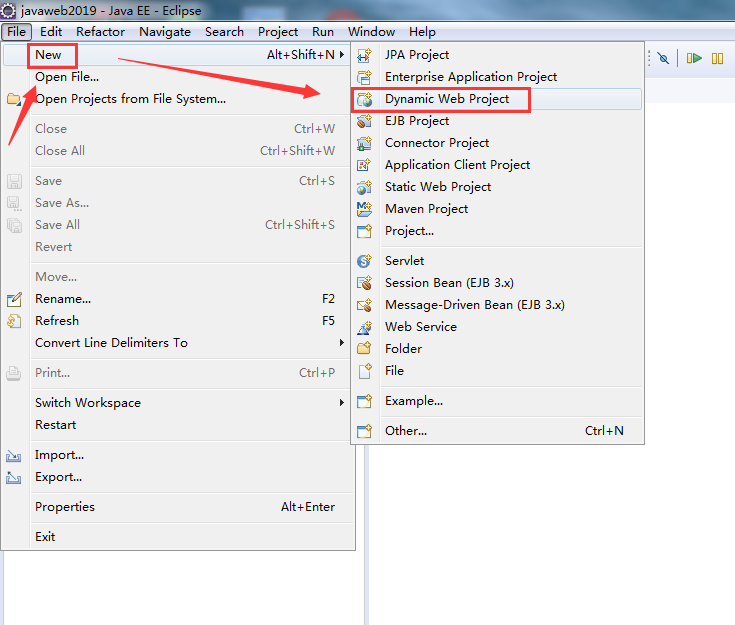
Window + R -> cmd -> startup.bat

### 5）浏览器访问

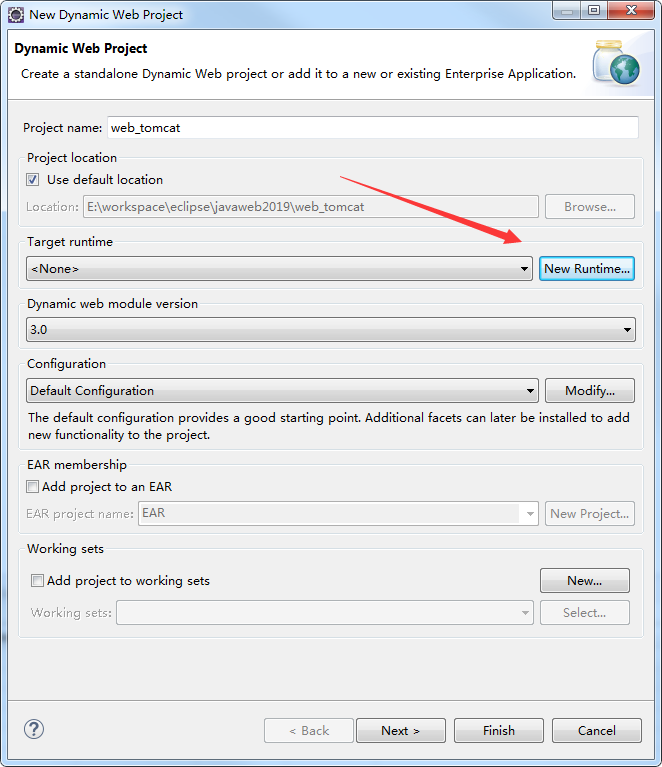
http://localhost:8080

## 3、Eclipse中配置Tomcat，并创建web项目

### 1）New -> Dynamic Web Project

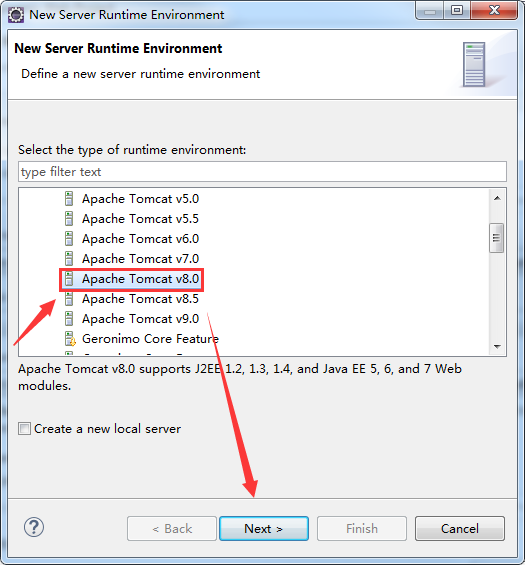


### 2）New Runtime…

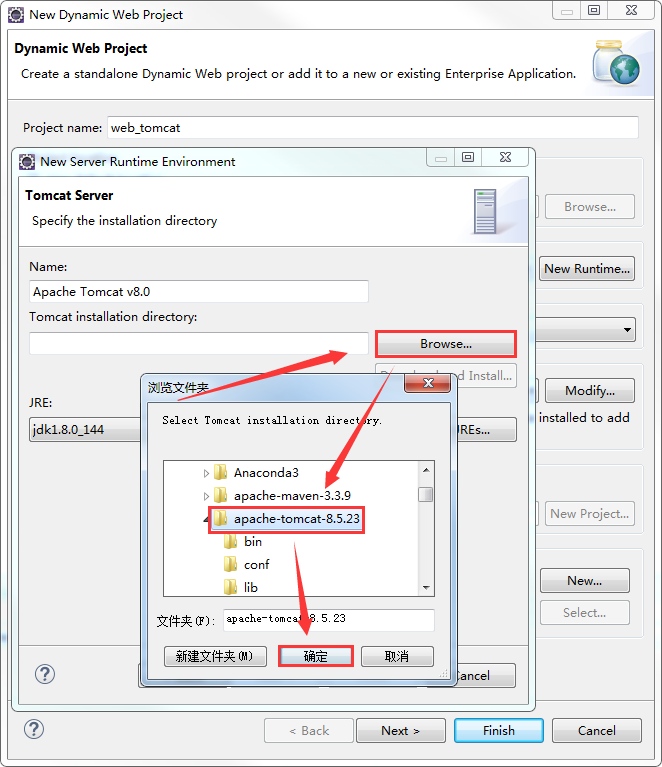


**尖叫提示：3.0更多的是基于注解的开发，2.5更多的是基于web.xml的配置**

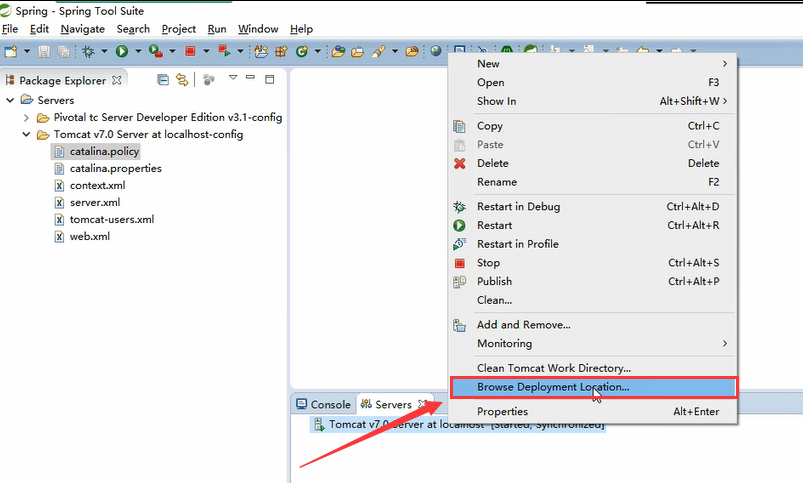
### 3）Apache Tomcat 8 -> Next



### 4）Browse -> Tomcat 8 -> 确定



### 5）查看Eclipse 集成的Tomcat 工作空间

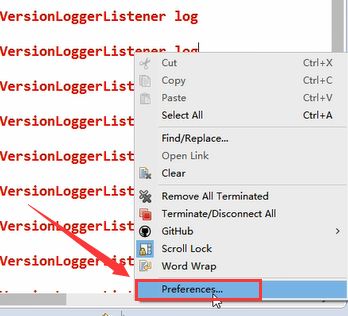




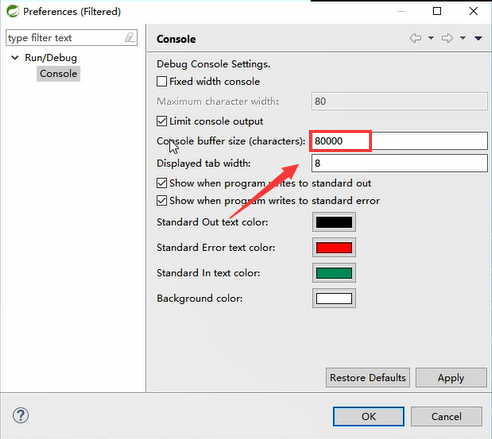
**尖叫提示：切换工作空间需要重新配置，因为集成的Tomcat是在工作空间里面的**

### 6）加长控制台显式log日志数量

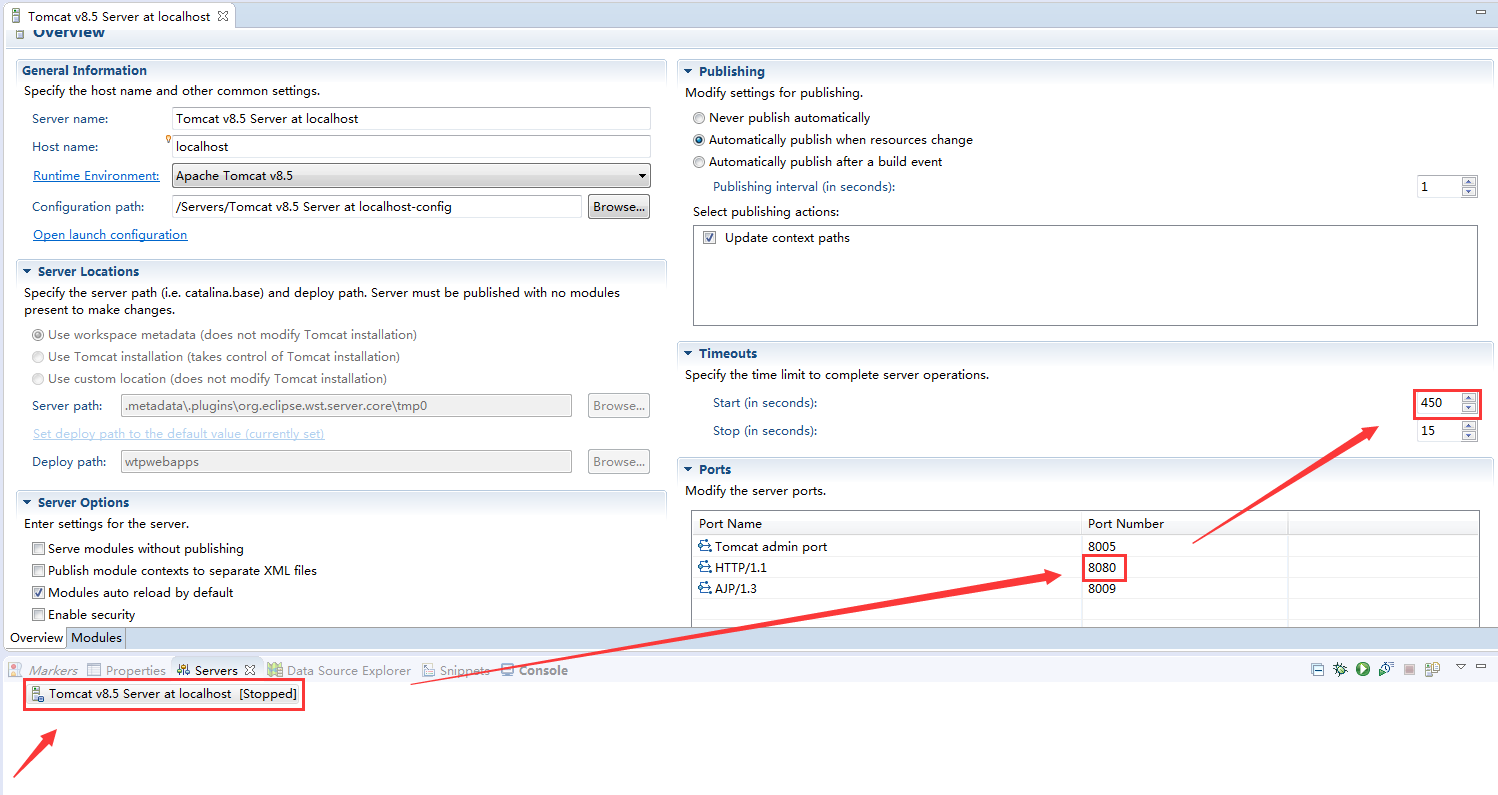
#### （1）右键 -> Preferences ..



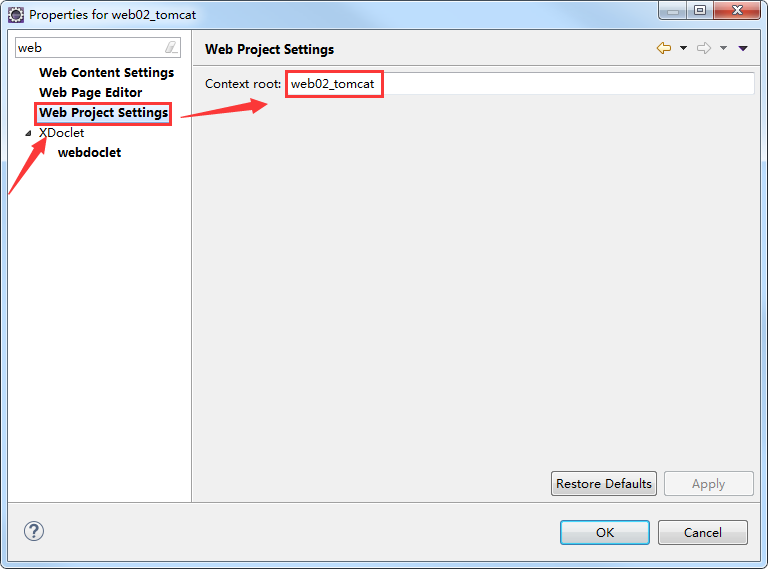
#### （2）Console buffer size



### 7）修改Tomcat 配置参数



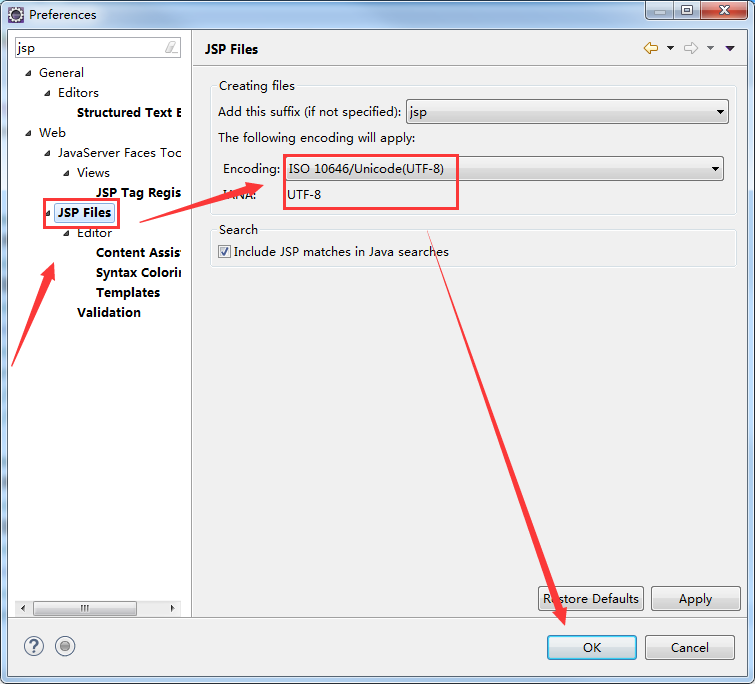
### 8）修改发布项目名称



### 9）Tomcat 编码和端口问题

|  |
| --- |
| <Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000"  redirectPort="8443" URIEncoding="UTF-8"/> |

### 10）jsp 修改UTF-8编码

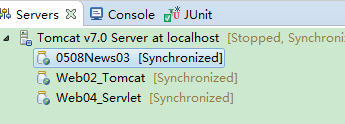


## 4、清理Tomcat服务缓存

清理服务器缓存的步骤：

1）将服务器停了

2）将所有在服务器上跑的web项目：remove。



3）服务器上右键：clear一下

4）如果想更彻底的clean：就在eclipse-project-clean

此章节着重演示一下Eclipse 与 Java 、Tomcat 的各种操作，主要看课堂演示，jsp和html内容尽量写在webcontent里面，如果写在web-inf里面，会受安全保护，访问报404

# 五、XML（Web03\_XML）

## 1、XML 介绍

### 1）XML是什么？

可扩展标记语言 eXtensible Markup Language——由W3C组织发布，目前推荐遵守的是W3C组织于2000年发布的XML1.0规范。XML的使命，就是以一个统一的格式，组织有关系的数据，为不同平台下的应用程序服务。

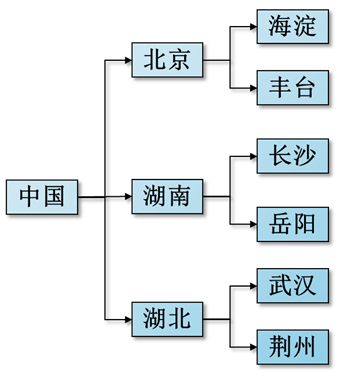
### 2）XML vs HTML

HTML先于XML产生，XML是在HTML的基础上改进而来的，但他们的作用不太相同。

HTML是用来显示网页数据的，而XML是用来组织和保存的数据的。

HTML文档中可以使用的标签是固定的一整套，XML允许用户自定义标签。

### 3）Xml案例



|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <中国>  <北京>  <海淀></海淀>  <丰台></丰台>  </北京>  <湖南>  <长沙></长沙>  <岳阳></岳阳>  </湖南>  <湖北>  <武汉></武汉>  <荆州></荆州>  </湖北>  </中国> |

## 2、XML用途

### 1）配置文件

C3P0中的c3p0-config.xml

Tomcat的server.xml

Android项目中的AndroidManifest.xml

JavaWeb工程中的web.xml

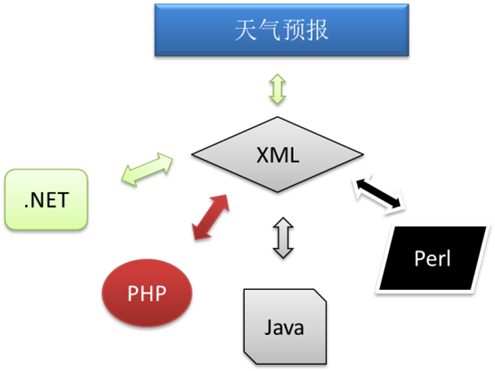
### 2）布局文件

Android中的布局文件

### 3）数据交换格式

Ajax

WebService



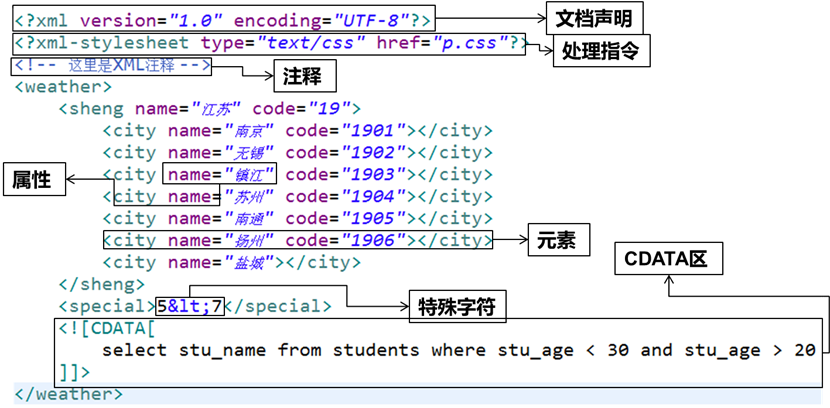
### 4）数据存储

保存关系型数据

举例

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <class>  <student id="007">  <name>王五</name>  <age>18</age>  </student>  <student id="008">  <name>张三</name>  <age>23</age>  </student>  </class> |

## 3、XML文档构成



说明

### 1）XML声明

version属性指定XML版本，固定值是1.0

encoding指定的字符集是告诉解析器使用什么字符集进行解码，而编码是由文本编辑器决定的。

### 2）转义字符



### 3）CDATA区

[1]当XML文档中需要写一些程序代码、SQL语句或其他不希望XML解析器进行解析的内容时，就可以写在CDATA区中

[2]XML解析器会将CDATA区中的内容原封不动的输出

[3]CDATA区的定义格式：**<![CDATA[…]]>**

## 4、XML语法规范

1）第一行是XML声明，必须顶格写

2）只能有一个根标签

3）标签必须正确结束

4）标签不能交叉嵌套

5）注释不能嵌套

6）属性必须有值，且必须加引号

7）标签不能以数字开头

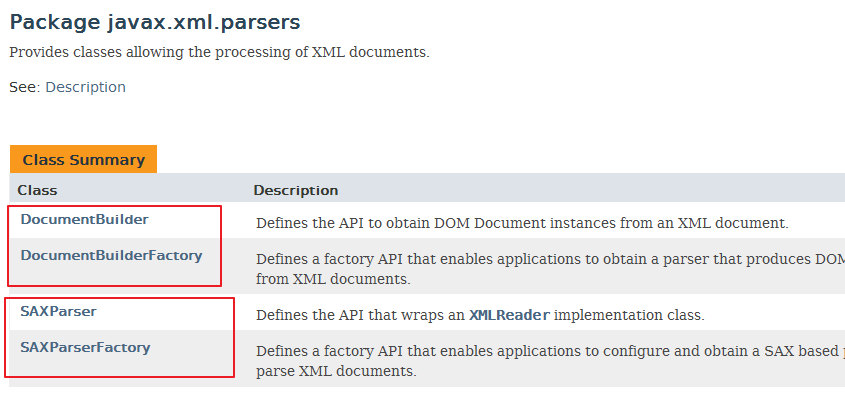
## 5、XML 解析

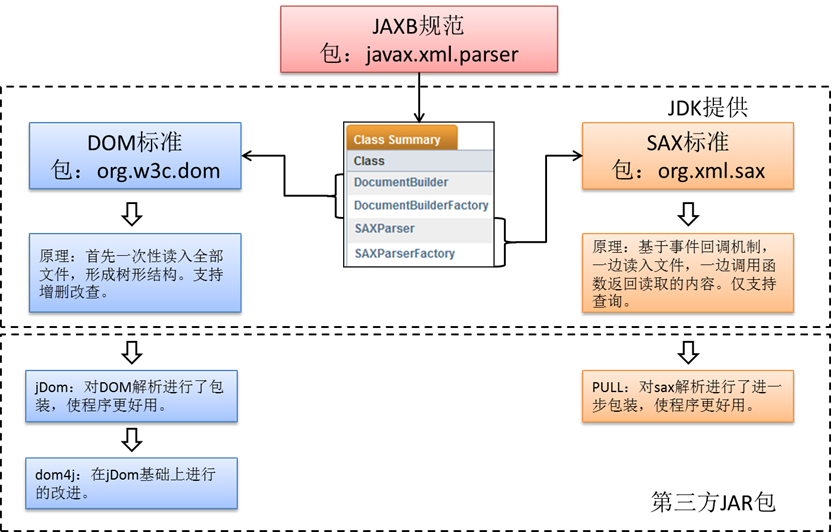
rt.jar

org.w3c.dom

org.xml.sax

两种解析方式





## 6、两种解析方式的对比

### 1）DOM体系

特点：将XML文档一次性全部加装到内存中，生成树形结构。

优点：API直观，支持增删改查

缺点：文档如果很大则会非常消耗内存

### 2）SAX体系

特点：不将XML文档一次性全部加载到内存中，每一次只加载一小部分。基于事件回调。

优点：不需要一次性加载整个文档，有利于节约内存资源

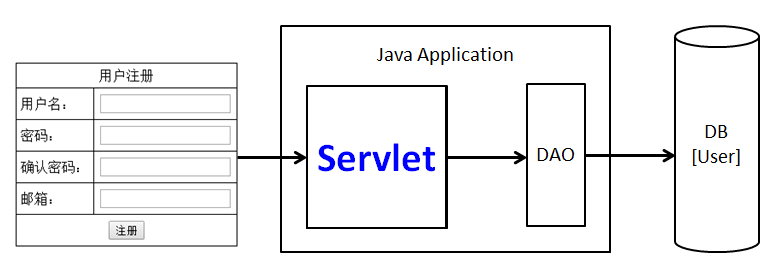
缺点：不支持增删改，只能查询

# 六、Servlet

## 1、Servlet 概述

Why ?

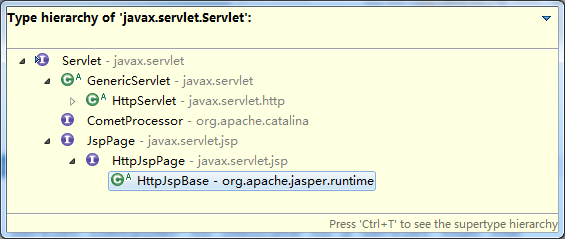
注册信息如何插入数据库表？



What ?

Servlet :服务器端小程序 Server + Applet

具体：javax.servlet.Servlet接口及其子接口，也可以指实现了Servlet接口的实现类



## 2、Servlet 使用案例一（web\_servlet01）

通过web\_servlet01 初体验 Servlet web项目

Hello World

Servlet接口扩展

为什么要扩展Servlet接口？封装不常用方法

实现类体系

GenericServlet实现Servlet接口

HttpServlet继承GenericServlet

创建Servlet的最终方式

继承HttpServlet

### 1）Servlet的使用三步曲

#### （1）创建一个类

实现接口:Servlet或继承HttpServlet

#### （2）在web.xml中“注册”这个实现类

|  |
| --- |
| <servlet>  <servlet-name>LoginServlet</servlet-name>  <servlet-class>com.alex.web.servlet.LoginServlet</servlet-class>  <!-- load-on-startup 值大于等于0的整数值,表示服务器启动时创建Servlet,值越小优先级越高 -->  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>LoginServlet</servlet-name>  <url-pattern>/LoginServlet</url-pattern>  </servlet-mapping> |

#### （3）Tomcat

（Servlet容器）会创建实现类对象，并调用相应的生命周期方法

### 2）生命周期方法

构造器

init()方法

service()方法

destroy()方法

### 3）执行过程

第一次请求

创建对象

执行init()方法

执行service()方法

后面请求

执行service()方法 …. …..

对象销毁前

执行destroy()方法

### 4）Servlet对象创建特殊情况

<!-- load-on-startup 值大于等于0的整数值,表示服务器启动时创建Servlet,值越小优先级越高 -->

<load-on-startup>1</load-on-startup>

### 5）code

#### （1）LoginServlet

|  |
| --- |
| package com.alex.web.servlet;  import java.io.IOException;  import javax.servlet.Servlet;  import javax.servlet.ServletConfig;  import javax.servlet.ServletException;  import javax.servlet.ServletRequest;  import javax.servlet.ServletResponse;  import javax.servlet.http.HttpServlet;  //Servlet 对象时单例的，伪单例  //public class LoginServlet implements Servlet{  public class LoginServlet extends HttpServlet{    //声明周期方法  //生命周期:表示对象从创建、初始化、使用、销毁过程    //创建对象  public LoginServlet(){  System.out.println("LoginServlet.....");  }  //销毁对象时使用  @Override  public void destroy() {  System.out.println("destroy ......");  }  //初始化对象  //初始化方法只执行一次  @Override  public void init(ServletConfig servletconfig) throws ServletException {  System.out.println("init ... ...");  }  //处理请求使用的  @Override  public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException {  System.out.println("service ... ...");  response.getWriter().println("Hello World... ...");  }  } |

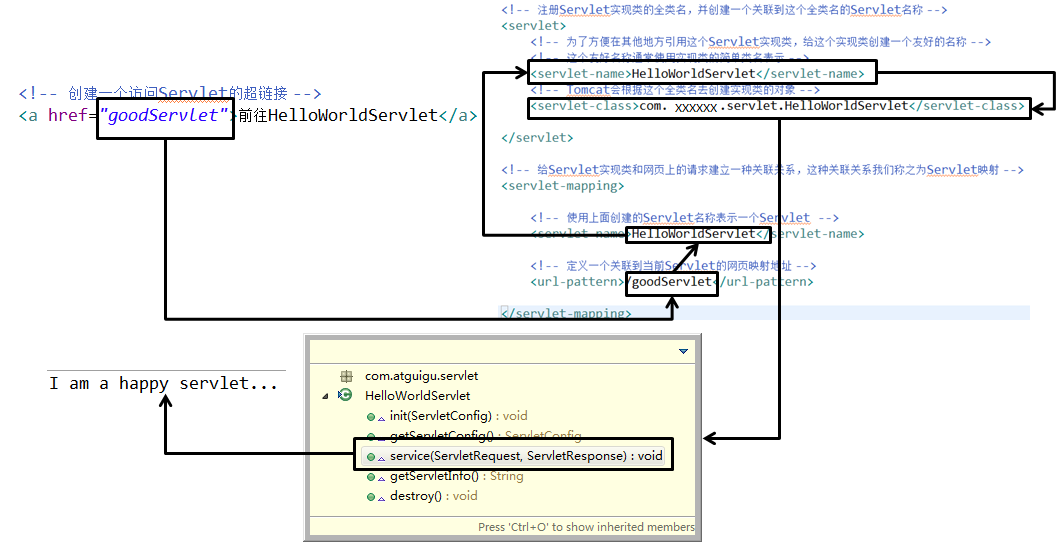
#### （2）web.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns="http://xmlns.jcp.org/xml/ns/javaee"  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"  id="WebApp\_ID" version="3.1">  <display-name>web\_servlet01</display-name>  <servlet>  <servlet-name>LoginServlet</servlet-name>  <servlet-class>com.alex.web.servlet.LoginServlet</servlet-class>  <!-- load-on-startup 值大于等于0的整数值,表示服务器启动时创建Servlet,值越小优先级越高 -->  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>LoginServlet</servlet-name>  <url-pattern>/LoginServlet</url-pattern>  </servlet-mapping>  <welcome-file-list>  <welcome-file>index.html</welcome-file>  <welcome-file>index.htm</welcome-file>  <welcome-file>index.jsp</welcome-file>  <welcome-file>default.html</welcome-file>  <welcome-file>default.htm</welcome-file>  <welcome-file>default.jsp</welcome-file>  </welcome-file-list>    </web-app> |

#### （3）index.jsp

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Hello World</title>  </head>  <body>  <a href="LoginServlet">LoginServlet</a>  </body>  </html> |

## 3、Servlet 执行过程分析



## 4、Servlet 使用案例二 login（web\_servlet02）

Form

通过web\_servlet02，学习get 和post 提交流程，编码等等

**编码**

### 1）code

#### （1）web.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns="http://xmlns.jcp.org/xml/ns/javaee"  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"  id="WebApp\_ID" version="3.1">    <display-name>web\_servlet02</display-name>  <servlet>  <servlet-name>LoginServlet</servlet-name>  <servlet-class>com.alex.web.servlet.LoginServlet</servlet-class>  <!-- load-on-startup 值大于等于0的整数值,表示服务器启动时创建Servlet,值越小优先级越高 -->  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>LoginServlet</servlet-name>  <url-pattern>/LoginServlet</url-pattern>  </servlet-mapping>  <welcome-file-list>  <welcome-file>index.html</welcome-file>  <welcome-file>index.htm</welcome-file>  <welcome-file>index.jsp</welcome-file>  <welcome-file>default.html</welcome-file>  <welcome-file>default.htm</welcome-file>  <welcome-file>default.jsp</welcome-file>  </welcome-file-list>    </web-app> |

#### （2）login.jsp

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  </head>  <body>  Get  <form action="LoginServlet" method="get">  用户名：<input type="text" name="username" />  <br/>  密码：<input type="text" name="passwd" />  <br/>  <!-- 访问AutoByEclipseServlet中的doGet方法 -->  <input type="submit" value="发送GET请求" />  </form>  <br />  <br />  POST  <form action="LoginServlet" method="post">  用户名<input type="text" name="username" />  <br/>  密码：<input type="text" name="passwd" />  <br/>  <!-- 访问AutoByEclipseServlet中的doPost方法 -->  <input type="submit" value="发送POST请求" />  </form>  </body>  </html> |

#### （3）LoginServlet

|  |
| --- |
| package com.alex.web.servlet;  import java.io.IOException;  import java.util.Map;  import javax.servlet.Servlet;  import javax.servlet.ServletConfig;  import javax.servlet.ServletException;  import javax.servlet.ServletRequest;  import javax.servlet.ServletResponse;  import javax.servlet.http.HttpServlet;  //Servlet 对象时单例的，伪单例  //public class LoginServlet implements Servlet{  public class LoginServlet extends HttpServlet{    //声明周期方法  //生命周期:表示对象从创建、初始化、使用、销毁过程    //创建对象  public LoginServlet(){  System.out.println("LoginServlet.....");  }  //销毁对象时使用  @Override  public void destroy() {  System.out.println("destroy ......");  }  //初始化对象  //初始化方法只执行一次  @Override  public void init(ServletConfig servletconfig) throws ServletException {  System.out.println("init ... ...");  }  //处理请求使用的  @Override  public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException {    //设置请求字符编码，解决客户端提交数据是中文乱码问题  //设置必须在request.getParameter(" ") 方法前调用，否则不起作用  //只针对post请求方式有效，对get请求方式无效  //如果解决Get请求乱码问题，需要在Tomcat/conf/server.xml中设置字符编码  //<Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="8443" URIEncoding="UTF-8"/>  request.setCharacterEncoding("UTF-8");    //如何获取客户端浏览器的数据？  //一个参数名称带一个值，例如：文本框、密码框  String username = request.getParameter("username");  String passwd = request.getParameter("passwd");    //一个参数名称带多个值，例如：复选框  //String[] values = request.getParameterValues("");    //一次性获取所有请求参数，key为参数名称，值为参数名称对应的值  //Map<String, String[]> parameterMap = request.getParameterMap();    System.out.println("username:" + username);  System.out.println("passwd:" + passwd);    //服务器端告诉客户端浏览器，返回的是text/html类型的数据  //设置响应内容类型及编码，需要在response.getWriter().println("")方法调用前设置，否则不起作用  response.setContentType("text/html;charset=UTF-8");    if("admin".equals(username) && "123".equals(passwd)){  response.getWriter().println("登录成功...");  }else{  response.getWriter().println("登录失败，用户名或者密码不正确");  }  }  } |

### 2）编码问题

（1）从页面获取值的时候post 提交方式

|  |
| --- |
| request.setCharacterEncoding("UTF-8"); |

（2）从页面获取值的时候get 提交方式

|  |
| --- |
| <Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="8443" URIEncoding="UTF-8"/> |

（3）Servlet 返回到页面

|  |
| --- |
| response.setContentType("text/html;charset=UTF-8"); |

（4）jsp页面设置utf-8 编码

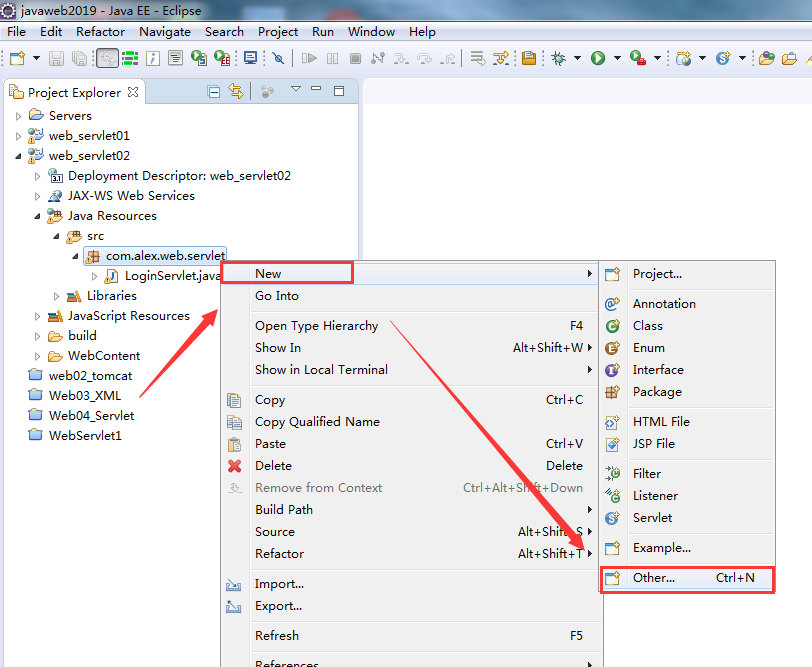
|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"> |

## 5、Eclipse中创建Servlet（web\_servlet03）

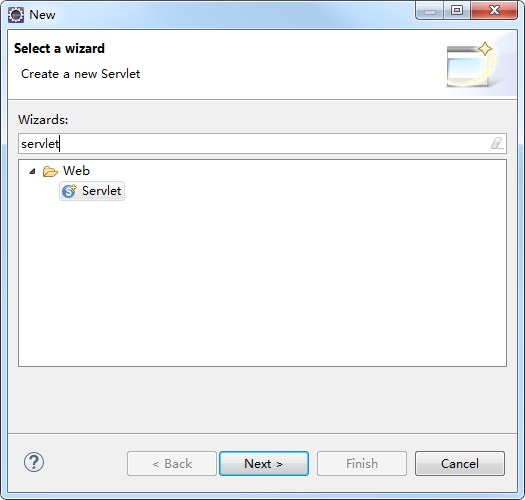
**实现自动配置**

通过 web\_servlet03 学习Eclipse 自动创建Servlet

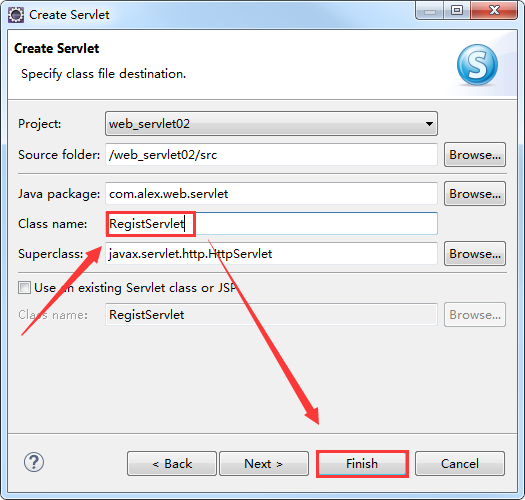
### 1）New -> Other



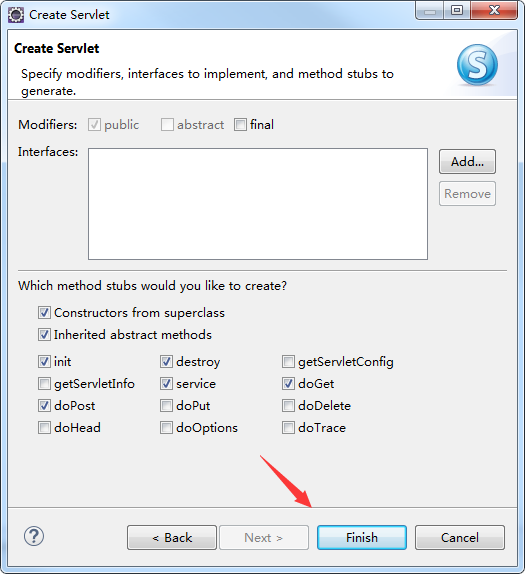
### 2）Servlet -> Next



### 3）Class Name -> Finish



### 4）Finish



### 5）web.xml

web.xml 里面自动生成

|  |
| --- |
| <servlet>  <description>完成注册请求处理</description>  <display-name>RegistServlet</display-name>  <servlet-name>RegistServlet</servlet-name>  <servlet-class>com.alex.web.servlet.RegistServlet</servlet-class>  <init-param>  <param-name>filename</param-name>  <param-value>d:/tmp/log.txt</param-value>  </init-param>  <load-on-startup>2</load-on-startup></servlet>  <servlet-mapping>  <servlet-name>RegistServlet</servlet-name>  <url-pattern>/RegistServlet</url-pattern>  </servlet-mapping> |

### 6）RegistServlet

|  |
| --- |
| package com.alex.web.servlet;  import java.io.IOException;  import javax.servlet.ServletConfig;  import javax.servlet.ServletException;  import javax.servlet.annotation.WebServlet;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  /\*\*  \* Servlet implementation class RegistServlet  \*/  public class RegistServlet extends HttpServlet {    private static final long serialVersionUID = 1L;    /\*\*  \* @see HttpServlet#HttpServlet()  \*/  public RegistServlet() {  super();  // TODO Auto-generated constructor stub  }  /\*\*  \* @see Servlet#init(ServletConfig)  \*/  public void init(ServletConfig config) throws ServletException {  //在配置文件里面传递的参数，使用这种方式获取值  String filename = config.getInitParameter("filename");  }  /\*\*  \* @see Servlet#destroy()  \*/  public void destroy() {  // TODO Auto-generated method stub  }  /\*\*  \* @see HttpServlet#service(HttpServletRequest request, HttpServletResponse response)  \*/  protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  // TODO Auto-generated method stub  }  /\*\*  \* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)  \*/  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  // TODO Auto-generated method stub  response.getWriter().append("Served at: ").append(request.getContextPath());  }  /\*\*  \* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)  \*/  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  // TODO Auto-generated method stub  doGet(request, response);  }  } |

## 6、Servlet 转发与重定向（web\_servlet04）

通过web\_servlet04 学习转发和重定向

### 1）请求的转发

（1）实现

request.getRequestDispatcher("/regist\_error.html").forward(request, response);

（2）特点

在服务器内部完成，用户感知不到

浏览器地址栏不变

整个过程浏览器只发出了一个请求

目标资源可以在WEB-INF目录下

### 2）请求的重定向

（1）实现

response.sendRedirect("./regist\_success.html");

response.sendRedirect("/MyNews/regist\_success.html");

response.sendRedirect(request.getContextPath() + "/MyNews/regist\_success.html");

（2）特点

服务器以302状态码通知浏览器访问新地址

浏览器地址栏改变

整个过程浏览器发出两次请求

目标资源不可以在WEB-INF目录下

### 3）案例实操

#### （1）LoginServlet

|  |
| --- |
| package com.alex.web.servlet;  import java.io.IOException;  import javax.servlet.ServletConfig;  import javax.servlet.ServletException;  import javax.servlet.ServletRequest;  import javax.servlet.ServletResponse;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  public class LoginServlet extends HttpServlet {  private static final long serialVersionUID = 1L;  // 声明周期方法  // 生命周期:表示对象从创建、初始化、使用、销毁过程  // 创建对象  public LoginServlet() {  System.out.println("LoginServlet.....");  }  // 销毁对象时使用  @Override  public void destroy() {  System.out.println("destroy ......");  }  // 初始化对象  // 初始化方法只执行一次  @Override  public void init(ServletConfig servletconfig) throws ServletException {  System.out.println("init ... ...");  }  // 处理请求使用的  @Override  public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException {  // 设置请求字符编码，解决客户端提交数据是中文乱码问题  // 设置必须在request.getParameter(" ") 方法前调用，否则不起作用  // 只针对post请求方式有效，对get请求方式无效  // 如果解决Get请求乱码问题，需要在Tomcat/conf/server.xml中设置字符编码  // <Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000"  // redirectPort="8443" URIEncoding="UTF-8"/>  request.setCharacterEncoding("UTF-8");  // 如何获取客户端浏览器的数据？  // 一个参数名称带一个值，例如：文本框、密码框  String username = request.getParameter("username");  String passwd = request.getParameter("passwd");  System.out.println("username:" + username);  System.out.println("passwd:" + passwd);  // 服务器端告诉客户端浏览器，返回的是text/html类型的数据  // 设置响应内容类型及编码，需要在response.getWriter().println("")方法调用前设置，否则不起作用  response.setContentType("text/html;charset=UTF-8");  if ("admin".equals(username) && "123".equals(passwd)) {  // 转发请求  // 转发请求资源的路径，以斜杠开头，表示从当前应用程序的根路径来查找资源  // XML配置文件中以斜杠开头，也表示从当前应用程序的根路径来查找资源  request.getRequestDispatcher("/success.jsp").forward(request, response);  } else {  // 转换为子接口，子接口内容更丰富一些  HttpServletRequest httpServletRequest = (HttpServletRequest)request;  HttpServletResponse httpServletResponse = (HttpServletResponse) response;  //httpServletRequest.getContextPath() 方法获取当前应用程序的上下文路径  // URL : http://localhost:8080/web\_servlet04/login.jsp  // URI : /web\_servlet04/login.jsp  // getContextPath() : /web\_servlet04  httpServletResponse.sendRedirect(httpServletRequest.getContextPath() + "/failure.jsp");  }  }  } |

#### （2）web.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns="http://xmlns.jcp.org/xml/ns/javaee"  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"  id="WebApp\_ID" version="3.1">  <display-name>web\_servlet04</display-name>  <servlet>  <servlet-name>LoginServlet</servlet-name>  <servlet-class>com.alex.web.servlet.LoginServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>LoginServlet</servlet-name>  <url-pattern>/LoginServlet</url-pattern>  </servlet-mapping>  <welcome-file-list>  <welcome-file>index.html</welcome-file>  <welcome-file>index.htm</welcome-file>  <welcome-file>index.jsp</welcome-file>  <welcome-file>default.html</welcome-file>  <welcome-file>default.htm</welcome-file>  <welcome-file>default.jsp</welcome-file>  </welcome-file-list>  </web-app> |

#### （3）login.jsp

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  </head>  <body>  Get  <form action="LoginServlet" method="get">  用户名：<input type="text" name="username" />  <br/>  密码：<input type="text" name="passwd" />  <br/>  <!-- 访问AutoByEclipseServlet中的doGet方法 -->  <input type="submit" value="发送GET请求" />  </form>  <br />  <br />  POST  <form action="LoginServlet" method="post">  用户名<input type="text" name="username" />  <br/>  密码：<input type="text" name="passwd" />  <br/>  <!-- 访问AutoByEclipseServlet中的doPost方法 -->  <input type="submit" value="发送POST请求" />  </form>  </body>  </html> |

#### （4）failure.jsp

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  </head>  <body>  failure  </body>  </html> |

#### （5）success.jsp

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  </head>  <body>  success  </body>  </html> |

## 7、Servlet 常见问题

### 1）HttpServlet cannot be resolved to a type

在工程中，Servlet显示：HttpServlet cannot be resolved to a type。

原因：Eclipse中使用的HttpServlet类之所以识别不到的原因是没有导入Servlet-api.jar包

解决方案一

1-右键项目名称，进入properties进行配置

2-点击左边菜单里面的targeted Runtimes就可以看到tomcat没有进行勾选，这就是出现这个情况的根本原因。

3-勾选上tomcat，然后点击apply应用，然后点击OK

解决方案二

在需要导入的工程上右键进入properties，选择Java build path ，library选项卡中add external jars，选择你所安装tomcat目录下lib文件中的Servlet-api.jar包即可

### 2）Web应用中的字符乱码问题

①请求

[1]GET请求

(1)由于GET请求的请求参数是附着在URL地址后面的，所以需要通过在服务器端指定URL地址的解码字符集解决。方法：在Tomcat解压目录/conf目录/server.xml中，找到第一个connector标签，加入URIEncoding="UTF-8"

[2]POST请求

由于POST请求的请求参数是在请求体中，所以需要在request对象获取请求参数之前，设置解码字符集：request.setCharacterEncoding("UTF-8");

②响应

指定响应数据的编码字符集，然后告诉浏览器使用什么字符集进行解码。

实现方式：response.setContentType("text/html;charset=UTF-8");

注意：不要在获取PrintWriter对象后再设置

## 8、项目阶段（web\_servlet05）

通过web\_servlet05 ，打通任督二脉，完成登陆和注册功能，这里面分了两个项目，功能一样，其中一个使用了才c3p0连接池和dbutil框架

Jsp -> view

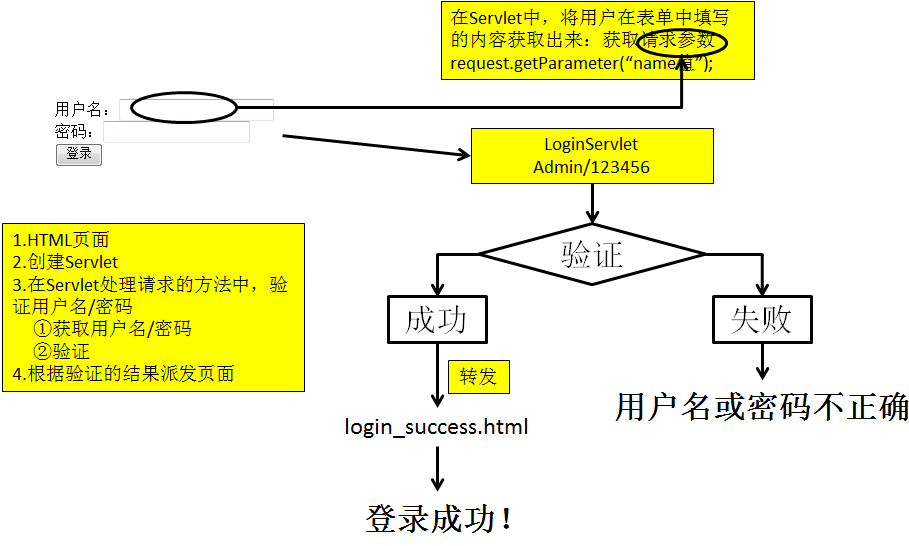
Servlet -> Ctroller

dao、entity .. -> model

mvc -> view 提交到Ctroller，Ctroller访问model，model把数据返回给Ctroller, Ctroller再放回到jsp页面

### 1）需求

实现用户的登陆和注册功能



### 2）code

#### （1）UserDao

|  |
| --- |
| package com.alex.web.dao;  import com.alex.web.entity.User;  public interface UserDao {  public User login(String username,String userpswd) throws Exception;    } |

#### （2）UserDaoImpl

|  |
| --- |
| package com.alex.web.dao;  import java.sql.Connection;  import java.sql.PreparedStatement;  import java.sql.ResultSet;  import com.alex.web.entity.User;  import com.alex.web.util.DBUtil;  public class UserDaoImpl implements UserDao {    /\*\*  CREATE TABLE t\_user(  id INT PRIMARY KEY AUTO\_INCREMENT,  username VARCHAR(32),  password VARCHAR(64),  email VARCHAR(128),  age INT  );  \*/  @Override  public User login(String username, String userpswd) throws Exception {  Connection connection = null ;  ResultSet resultSet = null ;  PreparedStatement prepareStatement = null;  User user = null ;  try {  connection = DBUtil.getConnection();  String sql = "select \* from t\_user where username=? and password=?";  prepareStatement = connection.prepareStatement(sql);  prepareStatement.setObject(1, username);  prepareStatement.setObject(2, userpswd);  resultSet = prepareStatement.executeQuery();    if (resultSet.next()) {  user = new User();  user.setId(resultSet.getInt("id"));  user.setUsername(resultSet.getString("username"));  user.setPassword(resultSet.getString("password"));  user.setEmail(resultSet.getString("email"));  user.setAge(resultSet.getInt("age"));  }  } catch (Exception e) {  e.printStackTrace();  throw e ;  } finally{  DBUtil.closeConnection(resultSet,prepareStatement,connection);  }  return user;  }  } |

#### （3）User

|  |
| --- |
| package com.alex.web.entity;  public class User {    private Integer id;  private String username;  private String password;  private String email;  private int age;  public Integer getId() {  return id;  }  public void setId(Integer id) {  this.id = id;  }  public String getUsername() {  return username;  }  public void setUsername(String username) {  this.username = username;  }  public String getPassword() {  return password;  }  public void setPassword(String password) {  this.password = password;  }  public String getEmail() {  return email;  }  public void setEmail(String email) {  this.email = email;  }  public int getAge() {  return age;  }  public void setAge(int age) {  this.age = age;  }  @Override  public String toString() {  return "User [id=" + id + ", username=" + username + ", password=" + password + ", email=" + email + ", age="  + age + "]";  }  } |

#### （4）LoginServlet

|  |
| --- |
| package com.alex.web.servlet;  import java.io.IOException;  import javax.servlet.ServletException;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  import com.alex.web.dao.UserDao;  import com.alex.web.dao.UserDaoImpl;  import com.alex.web.entity.User;    /\*\*  \* Servlet作用:处理请求  \* 步骤:  \* ①获取请求数据.  \* ②调用业务层或DAO层方法,完成业务逻辑处理或数据处理.  \* ③根据调用的方法获取返回结果,根据结果来进行判断,然后跳转(转发,重定向)到不同页面.  \*/  public class LoginServlet extends HttpServlet {  @Override  public void service(HttpServletRequest req, HttpServletResponse resp)  throws ServletException, IOException{  try {  String username = req.getParameter("username");  String passwd = req.getParameter("passwd");  resp.setContentType("text/html;charset=UTF-8");  UserDao userDao = new UserDaoImpl();  User loginUser = userDao.login(username, passwd);  if(loginUser!=null){  req.getRequestDispatcher("/success.jsp").forward(req, resp);  }else{  resp.sendRedirect(req.getContextPath()+"/failure.jsp"); // http://localhost:8080/WebServlet1/failure.html  }  } catch (Exception e) {  e.printStackTrace();  resp.sendRedirect(req.getContextPath()+"/error.jsp");  }    }  } |

#### （5）DBUtil

|  |
| --- |
| package com.alex.web.util;  import java.io.FileInputStream;  import java.sql.Connection;  import java.sql.DriverManager;  import java.sql.ResultSet;  import java.sql.SQLException;  import java.sql.Statement;  import java.util.Properties;  public class DBUtil {  static String driverClass;  static String url;  static String user;  static String password;  static {  try {  // 加载配置文件  Properties pro = new Properties();  //pro.load(new FileInputStream("src/db.properties"));  // 反射加载读取项目配置文件  pro.load(DBUtil.class.getClassLoader().getResourceAsStream("db.properties"));  // 读取配置文件  driverClass = pro.getProperty("driverClass");  url = pro.getProperty("url");  user = pro.getProperty("username");  password = pro.getProperty("password");  // 1.加载驱动  Class.forName(driverClass);  } catch (Exception e) {  e.printStackTrace();  }  }  /\*\*  \* 功能：获取连接 异常抛就行，可根据自己的方式处理异常  \*  \* @return 可用的连接对象  \* @throws IOException  \* @throws FileNotFoundException  \*/  public static Connection getConnection() throws Exception {  return DriverManager.getConnection(url, user, password);  }  /\*\*  \* 功能：关闭资源  \*  \* @param set  \* @param statement  \* @param connection  \* @throws SQLException  \*/  public static void closeConnection(ResultSet set, Statement statement, Connection connection) throws SQLException {  if (set != null)  set.close();  if (statement != null)  statement.close();  if (connection != null)  connection.close();  }  } |

#### （6）db.properties

|  |
| --- |
| driverClass=com.mysql.jdbc.Driver  url=jdbc:mysql:///test  #url=jdbc:mysql://127.0.0.1:3306/test  #url=jdbc:mysql:///test?rewriteBatchedStatements=true&useServerPrepStmts=false  username=root  password=root  initialSize=5  maxActive=10 |

### 3）调试

浏览器端使用F12

服务器端使用小虫子进行断点调试

## 9、Servlet Filter（web\_filter）

request.setCharacterEncoding("UTF-8");// 重复性代码

response.setContentType("text/html;charset=UTF-8");

像字符编码这种代码，每个Servlet都需要编码，就属于冗余代码，可以抽取出来放到拦截器里面

### 1）Filter简介

> Filter翻译为中文是过滤器的意思。

> Filter是JavaWeb的三大web组件之一Servlet、Filter、Listener

> Filter的作用是在请求到达WEB资源（HTML、CSS、Servlet、JSP）之前进行拦截。

> Filter需要交给Tomcat访问器来管理

> Filter的作用：

1.Filter可以在请求到达目标资源之前进行拦截

2.Filter也可以放行请求

3.Filter可以在响应到达浏览器之前做一个预处理

### 2）Filter的HelloWorld

> 步骤：

1.创建一个类实现Filter接口

2.在web.xml文件中对Filter进行配置

<filter>

<filter-name>Filter的别名</filter-name>

<filter-class>Filter的全类名</filter-class>

</filter>

<filter-mapping>

<filter-name>别名</filter-name>

<url-pattern>需要Filter拦截的资源地址</url-pattern>

</filter-mapping>

### 3）Filter的生命周期

> 一个请求的发送，到响应返回是一个线程。

> Filter的生命周期，指的是Filter对象由被创建到被销毁的过程。

> Filter的生命周期方法：

- 构造器：Filter的构造器在服务器启动时调用。 构造器只会调用一次，说明Filter也是单例多线程的。

- init()：在构造器被调用后，紧接着被调用。作用：用来初始化Filter。

- doFilter()：每一次拦截请求时都会调用。

参数

ServletRequest request

> 请求报文

ServletResponse response

> 响应报文

FilterChain chain

> 该对象的作用就是放行请求：

可以通过 chain.doFilter(request,response) 放行请求。

当调用chain.doFilter()就相当于调用目标的资源的service方法。

- destroy方法在项目停止时调用，用来在对象被销毁前做一些收尾工作。

> url-pattern的设置规则：

1.精确匹配：只有当目标资源的地址和url-pattern的地址一模一样时，Filter才会拦截资源

例子：/2.jsp , 只有当浏览器访问项目根目录下的2.jsp时，才会调用Filter

2.路径匹配：当访问的资源在url-pattern配置的路径下时，Filter就会拦截资源。

例子1：/hello/\* ， 只要访问项目根目录下hello下的资源，就会调用Filter。

例子2：/\* , 会拦截项目根目录下所有资源

3.后缀匹配：只要访问的资源地址和url-pattern的后缀一样就会调用Filter

例子：\*.jsp 只要是以jsp结尾的请求都会拦截

- 以上规则适用于Servlet的url-pattern配置

- 当使用Filter拦截一个Servlet时，可以将Filter的url-pattern和Servlet的url-pattern一样

这样Filter就会拦截发送到Servlet的请求。

- 也可以通过在filter-mapping中添加一个servlet-name标签来设置要拦截的Servlet的name

### 4）Filter的执行顺序

> 我们可以为一个资源设置多个过滤器，

当我们为一个资源设置了多个过滤器时，这些过滤器就组成一个Filter链的结构。

当我们去掉用chain.doFilter(),如果该过滤器后边还有其他的过滤，

相当于调用了下一个过滤器doFilter()方法

如果过滤器后没有其他的过滤器，我们调用chain.doFilter()

相当于调用了目标资源的service()方法。

> 多个Filter的执行顺序，有filter-mapping标签的配置顺序决定，

filter-mappint靠前，则filter先执行，靠后则后执行。

> 我们可以通过修改filter-mapping的顺序来修改Filter的执行顺序，

但是注意，filter-mapping不能写在filter标签的前边。

### 5）code

#### （1）CharacterEncodingFilter

|  |
| --- |
| package com.alex.web.filter;  import java.io.IOException;  import javax.servlet.Filter;  import javax.servlet.FilterChain;  import javax.servlet.FilterConfig;  import javax.servlet.ServletException;  import javax.servlet.ServletRequest;  import javax.servlet.ServletResponse;  /\*\*  \* 来解决 POST 请求中文乱码问题  \* 拦截器的作用  \* 解决那些问题：  \* 密码加密  \* 对请求参数值进行非法字符过滤  \* 权限控制  \* 登陆验证  \*/  public class CharacterEncodingFilter implements Filter{  String filename;    @Override  public void destroy() {  System.out.println("destroy ... ...");    }  @Override  public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)  throws IOException, ServletException {  System.out.println("doFilter 前... ...");    request.setCharacterEncoding("UTF-8");// 重复性代码    //这个编码设置，一定是获取输出流之前，在获取输出流之后在设置就不管用了  response.setContentType("text/html;charset=UTF-8");    chain.doFilter(request, response);//放行- 执行后续过滤器，执行目标程序    //狸猫换太子  String name = (String)request.getAttribute("name");  request.setAttribute("name", name + "22");    System.out.println("doFilter 后 ... ...");  }  //获取web.xml中filter配置的初始化参数  @Override  public void init(FilterConfig filterConfig) throws ServletException {  System.out.println("init ... ...");  filename = filterConfig.getInitParameter("filename");    }  } |

#### （2）LoginServlet

|  |
| --- |
| package com.alex.web.servlet;  import java.io.IOException;  import javax.servlet.ServletException;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  public class LoginServlet extends HttpServlet {  private static final long serialVersionUID = 1L;  // 处理请求使用的  @Override  public void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  request.setCharacterEncoding("UTF-8");// 重复性代码  response.setContentType("text/html;charset=UTF-8");  String username = request.getParameter("username");  String passwd = request.getParameter("passwd");  if ("admin".equals(username) && "123".equals(passwd)) {  request.setAttribute("name", "admin");  request.setAttribute("age", 20);  request.getSession().setAttribute("name", "Tom");  request.getSession().getServletContext().setAttribute("name", "alex");  request.getRequestDispatcher("/success.jsp").forward(request, response);  } else {  response.sendRedirect(request.getContextPath() + "/failure.jsp");  }  }  } |

#### （3）web.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns="http://xmlns.jcp.org/xml/ns/javaee"  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"  id="WebApp\_ID" version="3.1">  <display-name>web\_filter</display-name>  <!-- 过滤器：是单例的，服务器启动时被创建，别初始化 -->  <filter>  <filter-name>CharacterEncodingFilter</filter-name>  <filter-class>com.alex.web.filter.CharacterEncodingFilter</filter-class>  <!-- 可以设置多个参数，后期用框架的时候，像这种字符编码的过滤器，根本不需要我们自己写，只需要配置上就可以了 -->  <init-param>  <param-name>filename</param-name>  <param-value>D:/temp/log.txt</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>CharacterEncodingFilter</filter-name>  <!-- 路径匹配规则：  精确匹配：  <url-pattern>/LoginServlet</url-pattern>  路径匹配：  <url-pattern>/yyy/\*</url-pattern>  扩展匹配：  <url-pattern>\*.jsp</url-pattern>  <url-pattern>\*.action</url-pattern>  -->  <url-pattern>/\*</url-pattern>  </filter-mapping>  <servlet>  <servlet-name>LoginServlet</servlet-name>  <servlet-class>com.alex.web.servlet.LoginServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>LoginServlet</servlet-name>  <url-pattern>/LoginServlet</url-pattern>  </servlet-mapping>  <welcome-file-list>  <welcome-file>index.html</welcome-file>  <welcome-file>index.htm</welcome-file>  <welcome-file>index.jsp</welcome-file>  <welcome-file>default.html</welcome-file>  <welcome-file>default.htm</welcome-file>  <welcome-file>default.jsp</welcome-file>  </welcome-file-list>  </web-app> |

### 6、多个拦截器的拦截顺序

#### 1）拦截的路径一样

|  |
| --- |
| <filter-mapping>  <filter-name>CharacterEncodingFilter1</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>  <filter-mapping>  <filter-name>CharacterEncodingFilter2</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |

拦截路径一样，拦截器的顺序，就看web.xml的配置顺序，从上到下依次

#### 2）拦截的路径不一样

|  |
| --- |
| <filter-mapping>  <filter-name>CharacterEncodingFilter3</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping>  <filter-mapping>  <filter-name>CharacterEncodingFilter4</filter-name>  <url-pattern>\*.jsp</url-pattern>  </filter-mapping> |

如何都能拦截，那么这两个拦截器都会拦截一下，那么谁是第一个拦截器呢？？ 最精准匹配的，就是第一个拦截器

## 10、Servlet Listener（web\_listener）

### 1）监听器简介

Listener是JavaWeb中三大组件之一。Servlet、Filter、Listener

三大组件都有的共同特点，都需要实现一个接口，并在web.xml文件配置。

JavaWeb中的监听器的监听对象是谁？

ServletContext

HttpSession

ServletRequest

### 2）监听器分类

> JavaWeb中的监听器共有三种，共8个监听器

- 生命周期监听器，监听三个对象的创建和销毁的事件。

- 属性监听器，监听三个对象中属性的变化。

- session对象监听器，将它session中的属性，以及session的活化和钝化。

### 3）生命周期监听器

ServletContextListener

- ServletContext生命周期监听器

- void contextDestroyed(ServletContextEvent sce)

- 该方法在ServletContext对象销毁前调用

- void contextInitialized(ServletContextEvent sce)

- 该方法在ServletContext对象创建之前调用。

- ServletContextEvent 对象可以获取到ServletContext对象

ServletContextEvent.getServletContext();

HttpSessionListener

- HttpSession生命周期监听器

- void sessionCreated(HttpSessionEvent se)

- 该方法在Session创建时调用

- void sessionDestroyed(HttpSessionEvent se)

- 在session销毁时调用

- HttpSessionEvent

- 可以获取HttpSession对象

ServletRequestListener

- void requestDestroyed(ServletRequestEvent sre)

- 在request对象销毁时调用

- void requestInitialized(ServletRequestEvent sre)

- 在request对象创建时调用

- ServletRequestEvent

可以获取ServletContext对象和ServletRequest

编写一个监听器的步骤：

1.创建一个类并实现一个接口。

2.在web.xml文件中注册监听器。

### 4）属性监听器

> 属性监听器，监听三个域中的属性的变化：添加一个属性，替换一个属性，移除一个属性

> ServletContextAttributeListener

- 监听ServletContext中的属性的变化

void attributeAdded(ServletContextAttributeEvent scab)

- 当向ServletContext中添加属性时调用

attributeRemoved(ServletContextAttributeEvent scab)

- 移除属性时调用

attributeReplaced(ServletContextAttributeEvent scab)

- 替换一个属性时调用

> HttpSessionAttributeListener

void attributeAdded(HttpSessionBindingEvent se)

void attributeRemoved(HttpSessionBindingEvent se)

void attributeReplaced(HttpSessionBindingEvent se)

- HttpSessionBindingEvent

1.可以获取到属性名 getName()

2.可以获取属性的旧值 getValue()

3.可以获取到HttpSession对象 getSession();

> ServletRequestAttributeListener

void attributeAdded(ServletRequestAttributeEvent srae)

void attributeRemoved(ServletRequestAttributeEvent srae)

void attributeReplaced(ServletRequestAttributeEvent srae)

> 以下两个监听器监听session域中的属性的。

### 5）HttpSessionBindingListener

> 监听session域中某一个类的实例的添加和移除。

> 该接口由JavaBean来实现，不需要再web.xml文件配置

valueBound(HttpSessionBindingEvent event)

- 当该类的实例，作为属性设置进session域中时调用

valueUnbound(HttpSessionBindingEvent event)

- 当该类的实例，从session域中被移除时调用

### 6）HttpSessionActivationListener

> 监听session域中的某类属性，和session一起活化和钝化的事件。

> 该接口由JavaBean来实现，同样不需要再web.xml中配置

void sessionDidActivate(HttpSessionEvent se)

- 当前对象和session一起被活化到内存时调用

void sessionWillPassivate(HttpSessionEvent se)

当前对象和session一起钝化到硬盘时调用

### 7）code

#### （1）InitDBDataListener

|  |
| --- |
| package com.alex.web.listener;  import javax.servlet.ServletContextEvent;  import javax.servlet.ServletContextListener;  /\*\*  \* ServletContextListener : 用于监听ServletContext 对象（application）的创建和销毁  \*  \*/  public class InitDBDataListener implements ServletContextListener {    public InitDBDataListener(){  System.out.println("InitDBDataListener 对象被创建了");  }  // 用于监听application对象销毁后，需要执行的业务逻辑  @Override  public void contextDestroyed(ServletContextEvent sce) {  System.out.println(InitDBDataListener.class.getName() + "- contextDestroyed 执行了 - 删除数据库模拟数据");  }  // 用于监听application对象创建后需要执行的业务逻辑  // application 对象在服务器启动时，加载当前项目创建  @Override  public void contextInitialized(ServletContextEvent sce) {  System.out.println(InitDBDataListener.class.getName() + "- contextInitialized 执行了 - 往数据库插入了模拟数据");  }  } |

#### （2）web.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns="http://xmlns.jcp.org/xml/ns/javaee"  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"  id="WebApp\_ID" version="3.1">  <display-name>web\_listener</display-name>  <!--  web 开发三大组件： listener、filter、servlet  同时存在的时候，在web.xml创建的先后顺序  listener > filter > servlet  -->  <!-- 监听器对象，服务器启动时 -->  <listener>  <listener-class>com.alex.web.listener.InitDBDataListener</listener-class>  </listener>  <welcome-file-list>  <welcome-file>index.html</welcome-file>  <welcome-file>index.htm</welcome-file>  <welcome-file>index.jsp</welcome-file>  <welcome-file>default.html</welcome-file>  <welcome-file>default.htm</welcome-file>  <welcome-file>default.jsp</welcome-file>  </welcome-file-list>  </web-app> |

# 七、HTTP协议

## 1、HTTP概述

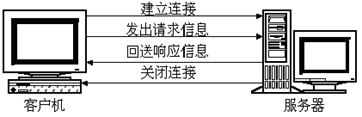
1）HTTP 是什么 ？？what?

HTTP超文本传输协议 (HTTP-Hyper text transfer protocol)，是一个属于应用层的面向对象的协议。

它于1990年提出，经过几年的使用与发展，得到不断地完善和扩展。

它定义了浏览器和服务器之间进行数据交换时的数据格式：请求和响应的数据格式。

2）浏览器与服务器之间的通信过程要经历四个步骤：



说明：

本次响应完成后，就关闭连接。下次再想请求时候，就建立另外的一个连接。多个连接间没有任何的关系。所以Http协议，又称为无状态的协议。

## 2、GET/POST请求格式

以Web04\_Servlet中的form.xml的get/post请求Servlet为例说明：（要求Servlet中提供请求转发和重定向）

### 1）请求分为三部分

请求行、请求消息头、请求体

### 2）请求行

GET请求

[1]从字面意思来说，GET请求是用来向服务器端获取信息而发送的请求。

[2]没有特殊设置，默认情况下浏览器发送的都是GET请求，例如点击超链接、在浏览器地址栏直接输入地址访问。

[3]GET请求也可以向服务器端发送请求参数，形式是在URL地址后面加上?，请求参数名和值用=连接，多个请求参数之间使用&分隔。例如：GET /mail/1.html?name=abc&password=xyz HTTP/1.1

需要注意的是：GET方式所能够携带的数据是有限制的，其数据大小通常不能超过4K，不适于提交大量表单数据，故而在表单的提交方式中首选POST方式。

POST请求

[1]POST请求的字面含义是向服务器端发送数据，仅在表单中设置method=”post”时，请求方式为POST方式[另外在Ajax应用中，可以指定请求方式为POST。

[2]POST请求会将请求参数放在请求体中，而不是URL地址后面，并且发送数据的大小是没有限制的。

### 3）请求消息头：（简称请求头）

|  |  |
| --- | --- |
| 代码 | 说明 |
| Accept: text/html,image/\* | 我支持的数据类型 |
| Accept-Charset: utf-8 | 支持的数据的编码字符集 |
| Accept-Encoding: gzip | 支持的压缩方式 |
| Accept-Language: en-us,zh-cn | 支持的语言 |
| Host: localhost:8888 | 请求的主机名 |
| Referer: http://www.it315.org/index.jsp | 发送请求的界面对应的url 防盗链 |
| User-Agent: Mozilla/4.0 | 浏览器的相关信息 |
| Connection: keep-Alive | 请求完成后，到是断开呢还是连接着 |
| Date: Tue, 11 Jul 2000 18:23:51 GMT | 发送请求的时间 |
| Cookie: tt=123 | 对具体客户端所做的标记 |

### 4）请求体

[1]GET请求没有请求体 GET最大传输4K大小的数据

[2]POST请求：如果form表单提交的方式为post,则表单项的数据以请求体的形式发送给服务器，没有大小限制

## 3、响应格式

1）所谓响应其实就是服务器对请求处理的结果，或者如果浏览器请求的直接就是一个静态资源的话，响应的就是这个资源本身。

2）HTTP响应的组成

（1）响应状态行：包括协议版本、响应状态码、响应状态信息

HTTP/1.1 200 OK

协议版本号 响应状态码 响应状态说明

（2）响应消息头：响应头。封装了与当前响应相关的信息

（3）响应体。用来在浏览器窗口中显示的内容。例如：html中的内容。

3）最常见的响应状态码

|  |  |  |
| --- | --- | --- |
| 代码 | 表示 | 说明 |
| 200 | 成功 | 服务器已成功处理了请求。通常，这表示服务器提供了请求的网页 |
| 302 | 重定向 | 代表让浏览器重新请求另一个资源 |
| 404 | 找不到 | 找不到请求的资源，但有时请求路径正确也返回404往往是由于Web应用有配置方面的问题，例如按照配置文件中指定的组件的全类名找不到指定的类 |
| 500 | 错误 | 服务器内部错误，例如服务器端程序运行时抛出异常 |

响应状态码以2开头的通常表示成功。

响应状态码以3开头的通常表示转移。

响应状态码以4开头的通常表示无法访问，其中包括找不到资源或没有权限等。

响应状态码以5开头的通常表示服务器端程序运行出错。

# 八、JSP（web\_jsp01）

## 1、JSP 概述

1）Why? 为什么要有jsp

Servlet可以通过转发或重定向跳转到某个HTML文档。但HTML文档中的内容不受Servlet的控制。比如登录失败时，跳转回登录表单页面无法显示诸如“用户名或密码不正确”的错误消息，所以我们目前采用的办法是跳转到一个错误信息页面。如果通过Servlet逐行输出响应信息则会非常繁琐。

|  |  |  |
| --- | --- | --- |
|  | Servlet | HTML |
| 长处 | 接收请求参数，访问域对象，转发页面 | 以友好方式显示数据 |
| 短处 | 以友好方式显示数据 | 动态显示数据 |

那能否将Servlet和HTML二者的长处结合起来呢？

2）What? 什么是jsp

Java Server Page

JSP = HTML + Servlet

（1）JSP的本质是一个Servlet，Servlet能做的事情JSP都能做。

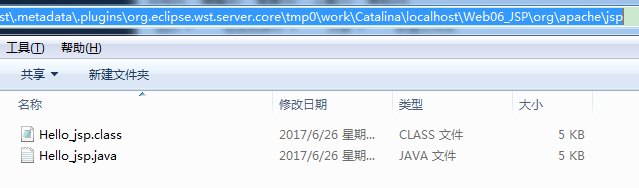
（2）JSP能够以HTML页面的方式呈现数据，是一个可以嵌入Java代码的HTML。

（3）JSP不同于HTML，不能使用浏览器直接打开，而必须运行在Servlet容器中。

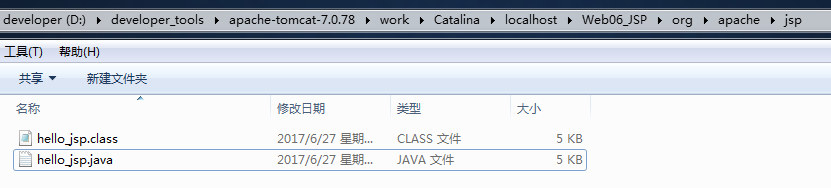
## 2、JSP的本质

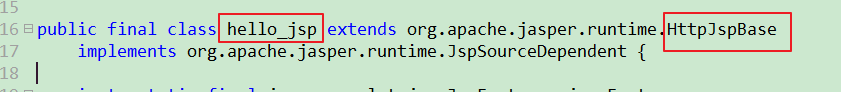
1）jsp声明在

Java 虚拟机要想运行一个类，需要先把java源代码变异成java字节码，变成.class才能运行，一个jsp是根本无法直接运行的，那么怎么办呢？ Tomcat提供了jsp的解析器，把jsp解析成java代码，再把jsp编译成class文件，解析出来的代码放到哪里去了呢？放到work目录下

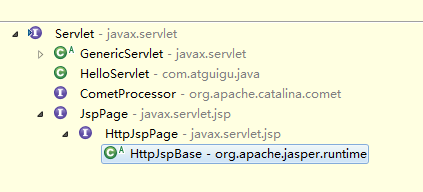


2）或者声明在





以及



## 3、code

### 1）LoginServlet

|  |
| --- |
| package com.alex.web.servlet;  import java.io.IOException;  import javax.servlet.ServletException;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  public class LoginServlet extends HttpServlet {  private static final long serialVersionUID = 1L;  // 处理请求使用的  @Override  public void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  request.setCharacterEncoding("UTF-8");  response.setContentType("text/html;charset=UTF-8");  String username = request.getParameter("username");  String passwd = request.getParameter("passwd");  if ("admin".equals(username) && "123".equals(passwd)) {  request.setAttribute("name", "admin");  request.setAttribute("age", 20);  request.getSession().setAttribute("name", "Tom");  request.getSession().getServletContext().setAttribute("name", "alex");  request.getRequestDispatcher("/success.jsp").forward(request, response);  } else {  response.sendRedirect(request.getContextPath() + "/failure.jsp");  }  }  } |

### 2）failure.jsp

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8" isErrorPage="true"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  </head>  <body>  failure  <% exception.equals("e"); %>  </body>  </html> |

### 3）login.jsp

|  |
| --- |
| <!-- HTML注释 -->  <%-- JSP 注释 --%>  <%-- page指令：用于设置当前页面 --%>  <%-- taglib指令：用于加载标签库 --%>  <%-- include指令：用于包含其它页面 --%>  <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  </head>  <body>  Get  <form action="LoginServlet" method="get">  用户名：<input type="text" name="username" />  <br/>  密码：<input type="text" name="passwd" />  <br/>  <!-- 访问AutoByEclipseServlet中的doGet方法 -->  <input type="submit" value="发送GET请求" />  </form>  <br />  <br />  POST  <form action="LoginServlet" method="post">  用户名<input type="text" name="username" />  <br/>  密码：<input type="text" name="passwd" />  <br/>  <!-- 访问AutoByEclipseServlet中的doPost方法 -->  <input type="submit" value="发送POST请求" />  </form>  </body>  </html> |

### 4）success.jsp

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>  <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  <link rel="stylesheet" href="css/success.css">  <script src="${pageContext.request.contextPath}/js/jquery.min.js"></script>  <script type="text/javascript">  </script>  <style type="text/css">  body {  padding: 0px;  margin: 0px;  }  </style>  </head>  <body>  success  <%  //不推荐在jsp中写Java代码，不易于维护  //页面数据处理，推荐使用JSTL 和 EL 表达式  String name = (String)request.getAttribute("name");  out.println("欢迎，" + name);  %>  <br /><br />  <c:out value="${requestScope.name}"></c:out><br /><br />  <c:out value="${sessionScope.name}"></c:out><br /><br />  <c:out value="${applicationScope.name}"></c:out><br /><br />  <!-- param 是从原始请求哪里取数据 -->  ${param.username}<br /><br />  ${requestScope.name}<br /><br />  ${sessionScope.name}<br /><br />  ${applicationScope.name}<br /><br />  <!-- 只写name 默认是从request里面取，从小到大的范围去搜索数据 -->  <!-- 从小到大的范围搜索数据  范围：  pageContext 当前页面有效  request 当前请求有效，一次请求可能会经过多个servlet或者jsp  session 当前客户端有效，同一个浏览器，同一个用户，不管发起多少次请求，数据都是有效的  application 所有的页面，所有的客户端，都能看到的数据，就放到application里面  查找数据顺序：  pageContext -> request -> session -> application  -->  ${name}  <c:choose>  <c:when test="${ age > 30 }"> 30 啦</c:when>  <c:when test="${ name eq 'admin' }">opop</c:when>  <c:otherwise>123</c:otherwise>  </c:choose>  </body>  </html> |

### 5）testSession.jsp

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  </head>  <body>  ${sessionScope.name}  </body>  </html> |

## 4、Servlet 传参数到jsp页面

|  |
| --- |
| request.setAttribute("name", "admin");  request.getSession().setAttribute("name", "Tom");  request.getSession().getServletContext().setAttribute("name", "alex"); |

## 5、jsp 获取Servlet 传递过来的值

### 1）方式一

在jsp页面中写java代码

|  |
| --- |
| <%  //不推荐在jsp中写Java代码，不易于维护  //页面数据处理，推荐使用JSTL 和 EL 表达式  String name = (String)request.getAttribute("name");  out.println("欢迎，" + name);  %> |

### 2）方式二

在jsp页面中使用EL表达式

|  |
| --- |
| <!-- param 是从原始请求哪里取数据 -->  ${param.username}<br /><br />  ${requestScope.name}<br /><br />  ${sessionScope.name}<br /><br />  ${applicationScope.name}<br /><br /> |

<!-- 只写name 默认是从request里面取，从小到大的范围去搜索数据 -->

<!-- 从小到大的范围搜索数据

范围：

pageContext 当前页面有效

request 当前请求有效，一次请求可能会经过多个servlet或者jsp

session 当前客户端有效，同一个浏览器，同一个用户，不管发起多少次请求，数据都是有效的

application 所有的页面，所有的客户端，都能看到的数据，就放到application里面

查找数据顺序：

pageContext -> request -> session -> application

-->

## 6、jsp页面写css代码

### 1）方式一

|  |
| --- |
| <p style="width: 5rem;"> 哈哈哈 </p> |

### 2）方式二

|  |
| --- |
| <p align="center" width="80%"> </p> |

### 3）方式三

|  |
| --- |
| <style type="text/css">  body {  padding: 0px;  margin: 0px;  }  </style> |

### 4）方式四

|  |
| --- |
| <link rel="stylesheet" href="css/success.css"> |

## 7、jsp页面写js代码

### 1）方式一

|  |
| --- |
| <script type="text/javascript">  </script> |

### 2）方式二

|  |
| --- |
| <script type="text/javascript"  src="${pageContext.request.contextPath}/js/jquery.min.js"></script> |

## 8、jsp页面写JSTL语法

在jsp页面中，不推荐使用java代码，不方便维护

页面数据处理，推荐使用JSTL 和 EL 表达式

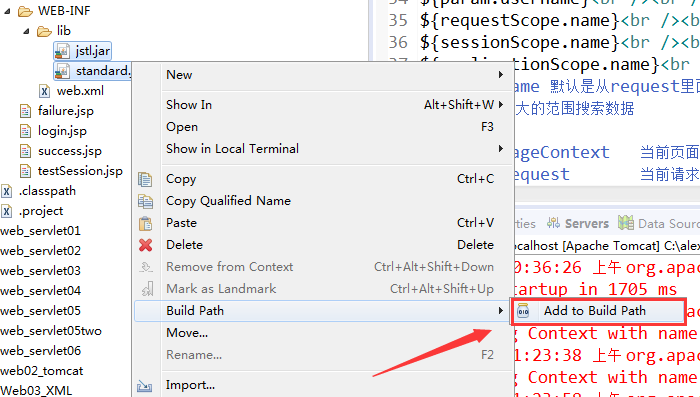
### 1）添加jar包

jstl.jar

standard.jar

复制到lib文件夹下，Build Path

jstl-1.2.jar



### 2）jsp页面添加taglib指令

|  |
| --- |
| <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %> |

### 3）JSTL写法

|  |
| --- |
| <c:choose>  <c:when test="${ age > 30 }"> 30 啦</c:when>  <c:when test="${ name eq 'admin' }">opop</c:when>  <c:otherwise>123</c:otherwise>  </c:choose> |

## 9、jsp 页面 session

课堂演示

（1）第一步

Login.jsp 数据提交到 LoginServlet

（2）第二步

LoginServlet 把数据存到session里面，转发到success.jsp页面

request.getSession().setAttribute("name", "Tom");

（3）第三步

直接访问testsession.jsp页面，看看能不能从session中取出数据来

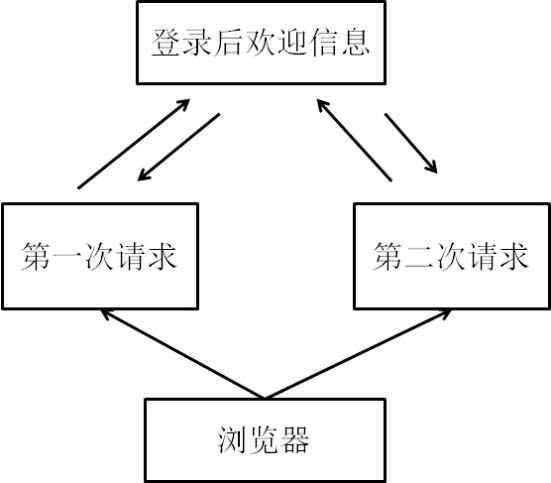
${sessionScope.name}

# 九、cookie（web\_cookie）

## 1、cookie的出现

### 1）当前的问题

在浏览器发送不同请求时，服务器端基于HTTP协议无法识别多次请求是否来自同一个浏览器



### 2）解决方法情境举例

一个咖啡厅促销活动：每位顾客喝5杯咖啡赠送一杯

方法一：让服务员把喝过咖啡的顾客都记住——不现实。

方法二：给每位顾客办一张会员卡，在卡片上记录他消费的次数

将消费的次数保存在顾客手中——Cookie：在浏览器端缓存数据的一种技术

0001：3→咖啡厅

0002：4→咖啡厅

0003：1→咖啡厅

方法三：给每位顾客办一张会员卡，卡片上只有一个卡号，咖啡厅根据卡号保存顾客消费的次数

将消费的次数保存在咖啡厅——Session：在服务器端缓存数据的技术

0001→咖啡厅[0001：3]

0002→咖啡厅[0002：5]

0003→咖啡厅[0003：1]

## 2、cookie的运行机制

### 1）Cookie运行机制

①“办卡”：创建Cookie对象

Cookie cookie = new Cookie("myCookie", UUID.randomUUID().toString());

②“将卡片交到顾客手中”：将Cookie对象中封装的信息回送到浏览器

API：response.addCookie(cookie);

以响应消息头的形式发送给浏览器

Set-Cookie:myCookie=5691a6b4-7441-4916-b319-1a04c31ec504

③“顾客带着卡片来咖啡厅”：浏览器请求服务器时携带Cookie数据

以请求消息头的形式携带Cookie数据

Cookie:myCookie=5691a6b4-7441-4916-b319-1a04c31ec504

④“咖啡厅检查顾客卡片”：服务器从请求对象中获取Cookie数据

//从request对象中能够获取到Cookie信息，返回的是Cookie的数组

response.setContentType("text/html;charset=UTF-8");

PrintWriter writer = response.getWriter();

Cookie[] cookies = request.getCookies();

//cookies数组不一定有值，如果浏览器没有提交Cookie数据，则cookies的值就等于null

if(cookies != null) {

for(int i = 0; i < cookies.length; i++) {

String name = cookies[i].getName();

String value = cookies[i].getValue();

writer.write("Cookie中的数据是："+name+"="+value);

writer.write("<br />");

}

}

### 2）不同浏览器

不同的浏览器访问同一个服务器，服务器也会根据浏览器的不同，返回不同的Cookie

### 3）Cookie有效性

①分类

[1]会话级Cookie：保存在浏览器内存中，随着浏览器关闭的关闭而释放，默认情况

[2]持久化级Cookie：保存在文件系统中，不会随着浏览器关闭而释放，但到达预先指定的时间就会自动释放

②设置

cookie.setMaxAge(int seconds);

[1]seconds正整数：表示将Cookie设置为持久化Cookie，保持seconds秒

[2]seconds=0：表示删除Cookie

[3]seconds负整数：表示将Cookie设置为会话级Cookie

注意：设置完成以后，必须使用response的addCookie()添加才可生效

## 3、code

### （1）CreateCookieServlet

|  |
| --- |
| package com.alex.web.servlet.cookie;  import java.io.IOException;  import javax.servlet.ServletException;  import javax.servlet.http.Cookie;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  //创建Cookie  public class CreateCookieServlet extends HttpServlet {  private static final long serialVersionUID = 1L;  // 写cookie  @Override  protected void service(HttpServletRequest request, HttpServletResponse response)  throws ServletException, IOException {  Cookie cookie = new Cookie("username", "alex");  response.addCookie(cookie);  }  } |

### （2）DeleteCookieServlet

|  |
| --- |
| package com.alex.web.servlet.cookie;  import java.io.IOException;  import javax.servlet.ServletException;  import javax.servlet.http.Cookie;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  public class DeleteCookieServlet extends HttpServlet {  private static final long serialVersionUID = 1L;  // 删除cookie, cookie没有delete 或者remove 等方法，  // 要想删除，只能重新赋值或者设置失效时间  @Override  protected void service(HttpServletRequest request, HttpServletResponse response)  throws ServletException, IOException {  Cookie cookie = new Cookie("username", null);  // 设置cookie的失效时间  cookie.setMaxAge(0);  response.addCookie(cookie);  }  } |

### （3）GetCookieServlet

|  |
| --- |
| package com.alex.web.servlet.cookie;  import java.io.IOException;  import javax.servlet.ServletException;  import javax.servlet.http.Cookie;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  public class GetCookieServlet extends HttpServlet {    private static final long serialVersionUID = 1L;  // 读取cookie  @Override  protected void service(HttpServletRequest request, HttpServletResponse response)  throws ServletException, IOException {  Cookie[] cookies = request.getCookies();  if (cookies != null) {  for (Cookie cookie : cookies) {  String name = cookie.getName();  if ("username".equals(name)) {  String value = cookie.getValue();  System.out.println(name + " -> " + value);  }  }  }  }  } |

### （4）PersistCookieServlet

|  |
| --- |
| package com.alex.web.servlet.cookie;  import java.io.IOException;  import javax.servlet.ServletException;  import javax.servlet.http.Cookie;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  //创建持久性cookie  public class PersistCookieServlet extends HttpServlet {    private static final long serialVersionUID = 1L;    //写cookie  @Override  protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  Cookie cookie = new Cookie("username","alex");  //设置cookie的失效时间  cookie.setMaxAge(60\*60);  //如果添加路径，那么只有访问某某路径下的时候，cookie才会跟着走  //cookie.setPath(request.getContextPath() + "/xxx");  response.addCookie(cookie);  }  } |

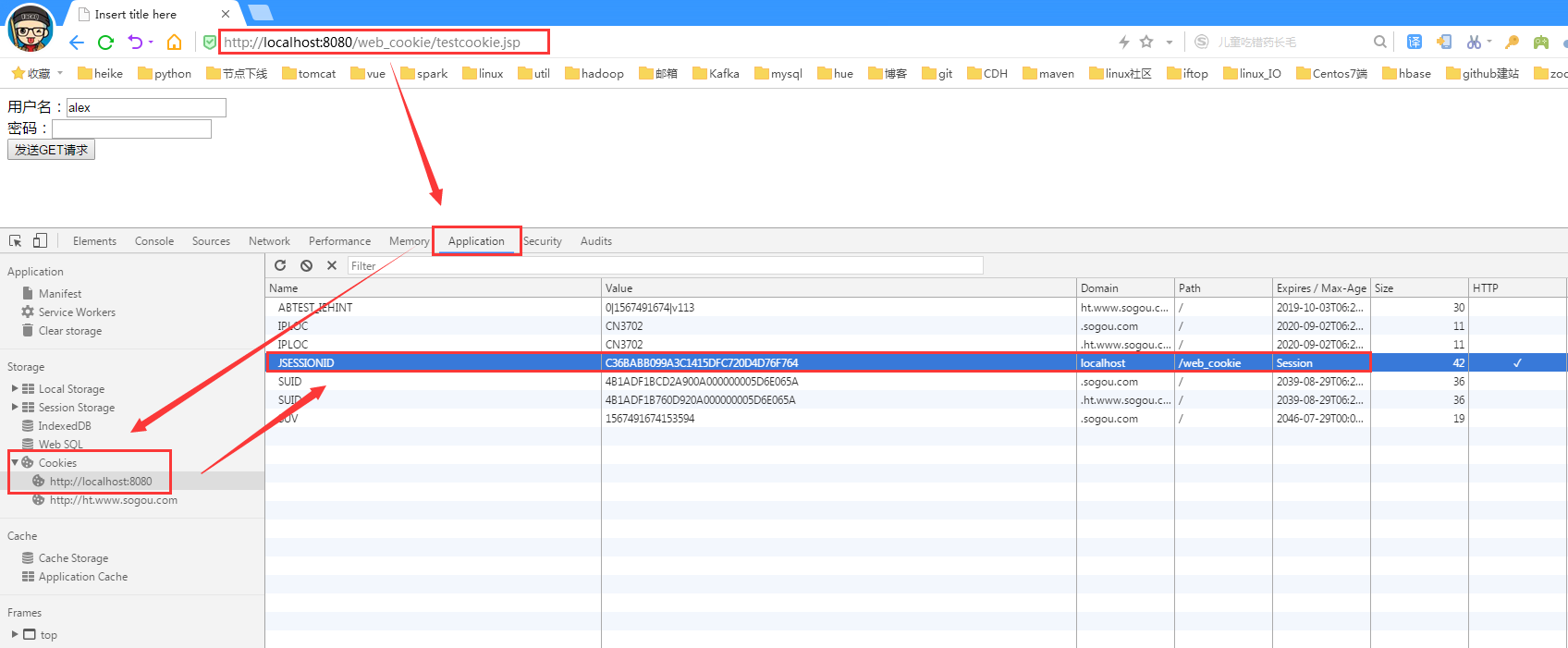
### （5）web.xml

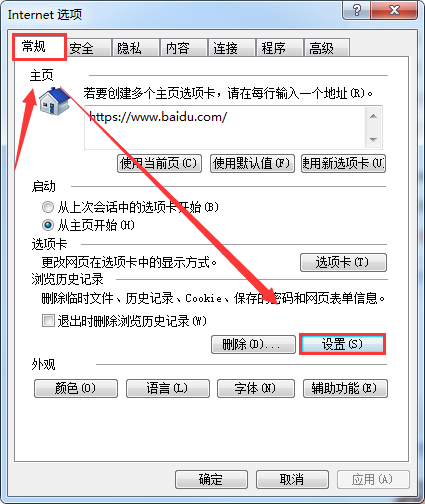
|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns="http://xmlns.jcp.org/xml/ns/javaee"  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"  id="WebApp\_ID" version="3.1">  <display-name>web\_cookie</display-name>  <servlet>  <servlet-name>CreateCookieServlet</servlet-name>  <servlet-class>com.alex.web.servlet.cookie.CreateCookieServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>CreateCookieServlet</servlet-name>  <url-pattern>/CreateCookieServlet</url-pattern>  </servlet-mapping>  <servlet>  <servlet-name>GetCookieServlet</servlet-name>  <servlet-class>com.alex.web.servlet.cookie.GetCookieServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>GetCookieServlet</servlet-name>  <url-pattern>/GetCookieServlet</url-pattern>  </servlet-mapping>  <servlet>  <servlet-name>PersistCookieServlet</servlet-name>  <servlet-class>com.alex.web.servlet.cookie.PersistCookieServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>PersistCookieServlet</servlet-name>  <url-pattern>/PersistCookieServlet</url-pattern>  </servlet-mapping>  <servlet>  <servlet-name>DeleteCookieServlet</servlet-name>  <servlet-class>com.alex.web.servlet.cookie.DeleteCookieServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>DeleteCookieServlet</servlet-name>  <url-pattern>/DeleteCookieServlet</url-pattern>  </servlet-mapping>  <welcome-file-list>  <welcome-file>index.html</welcome-file>  <welcome-file>index.htm</welcome-file>  <welcome-file>index.jsp</welcome-file>  <welcome-file>default.html</welcome-file>  <welcome-file>default.htm</welcome-file>  <welcome-file>default.jsp</welcome-file>  </welcome-file-list>    </web-app> |

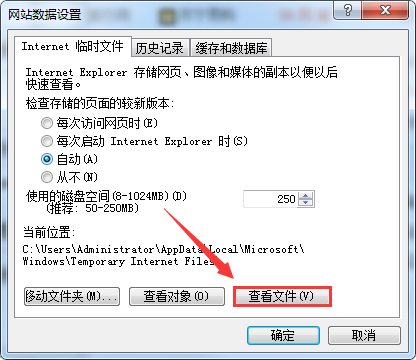
### （6）testcookie.jsp

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  </head>  <body>  <h3>  <a href="CreateCookieServlet">创建Cookie</a>  </h3>  <h3>  <a href="GetCookieServlet">访问Cookie</a>  </h3>  <h3>  <a href="PersistCookieServlet">设置持久化Cookie</a>  </h3>  <h3>  <a href="DeleteCookieServlet">删除指定的Cookie</a>  </h3>  </body>  </html> |

## 4、cookie本地存放位置







# 十、session（web\_session）

## 1、Session 概述

Session也是一个域对象，可以在自身的属性域中保存数据，在一定范围内共享。

## 2、Session的工作机制

1）并不是浏览器一访问服务器就创建Session对象，而是只有在服务器端调用request.getSession()方法时，Tomcat服务器才会“创建”Session对象。

2）request.getSession()方法

（1）请求中没有携带任何有关Session的标识,往往就是浏览器第一次访问的时候。

（2）请求中没有携带任何有关Session的标识：Tomcat会创建一个Session对象，并创建一个特殊的Cookie，这个特殊Cookie的name属性是JSESSIONID，值是一个唯一值,而且是随机生成的。将这个JSESSIONID Cookie作为关联这个Session对象的标识写回浏览器。

（3）请求中携带了JSESSIONID时，Tomcat会在已经创建好的Session对象的集合中，查找匹配的Session对象。实现多个请求识别浏览器的身份的目的。

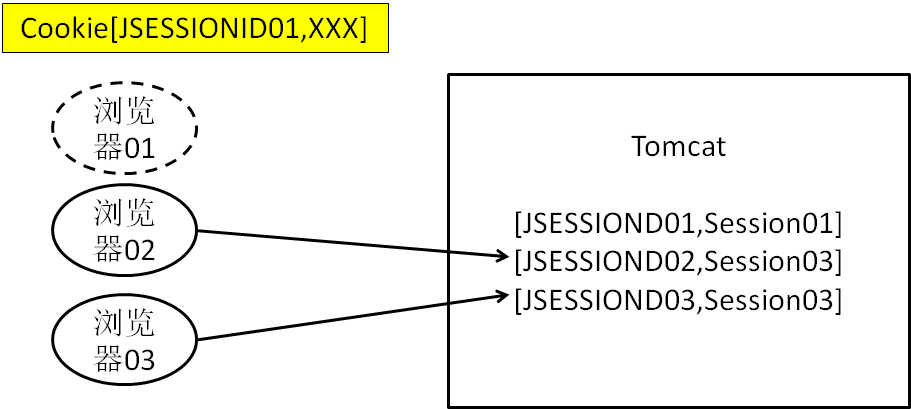
①能找到：返回找到的Session对象

②找不到：创建一个新的Session对象返回.通常由于服务器超时释放。

即：我们是通过JSESSIONID的值找到的session对象。

3）isNew()返回boolean标识当前Session对象是新创建的，还是原有的

4）getId()返回当前Session对象的JSESSIONID值



## 3、Session的时效管理

Session代表浏览器和服务器之间的一次会话，本来应该在会话开始时创建，会话结束时释放。但是浏览器关闭时，服务器根本检测不到，那就只能由服务器自己设置一个时间了。

[1]自然超时

<1>Tomcat配置文件中web.xml中有默认的超时时间的配置

|  |
| --- |
| <!-- 单位是分钟 -->  <session-config>  <session-timeout>30</session-timeout>  </session-config> |

Session对象默认的有效时间是30分钟，指的并不是Session对象从创建开始30分钟后过期，而是从最后一次访问开始，30分钟后失效。

<2>调用Session对象的方法手动设置超时时间

session.setMaxInactiveInterval(10)。以秒为单位

[2]手动调用invalidate()方法直接失效

## 4、项目阶段

登录成功，记录浏览器信息

### 1）显示登录用户

//将用户信息保存到Session对象中，到JSP页面中就可以通过Session隐含对象将数据读取出来

HttpSession session = request.getSession();

session.setAttribute("loginUser", userName);

### 2）退出操作

String method = request.getParameter("method");

if("logout".equals(method)){

HttpSession session = request.getSession();

session.invalidate();

response.sendRedirect(request.getContextPath()+"/index.jsp");

return;

}

## 5、code

### 1）SessionTestServlet

|  |
| --- |
| package com.alex.web.session;  import java.io.IOException;  import javax.servlet.ServletException;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  import javax.servlet.http.HttpSession;  public class SessionTestServlet extends HttpServlet{  private static final long serialVersionUID = 1L;    @Override  protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  String username = request.getParameter("username");  //获取并返回之前分配的session  //如果之前没分配过，就创建一个  HttpSession session = request.getSession(true);  //服务器将JSessionId写到客户端Cookie中了  System.out.println("JSESSIONID" + session.getId());  session.setAttribute("username", username);    //一般注销时，需要销毁session  //表示销毁session  //session.invalidate();    //注销系统，也可以不杀死session,而是将session数据清理掉  //删除session存的数据  //session.removeAttribute("username");  }  } |

### 2）web.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns="http://xmlns.jcp.org/xml/ns/javaee"  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"  id="WebApp\_ID" version="3.1">  <display-name>web\_session</display-name>  <servlet>  <servlet-name>SessionTestServlet</servlet-name>  <servlet-class>com.alex.web.session.SessionTestServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>SessionTestServlet</servlet-name>  <url-pattern>/SessionTestServlet</url-pattern>  </servlet-mapping>    <!-- 单位是分钟 -->  <session-config>  <session-timeout>30</session-timeout>  </session-config>  <welcome-file-list>  <welcome-file>index.html</welcome-file>  <welcome-file>index.htm</welcome-file>  <welcome-file>index.jsp</welcome-file>  <welcome-file>default.html</welcome-file>  <welcome-file>default.htm</welcome-file>  <welcome-file>default.jsp</welcome-file>  </welcome-file-list>  </web-app> |

### 3）testsession.jsp

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  </head>  <body>  <form action="SessionTestServlet" method="post">  用户名<input type="text" name="username" value="alex"/>  <input type="submit" value="发送POST请求" />  </form>  </body>  </html> |

### 4）a.jsp

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  </head>  <body>  ${sessionScope.username}  </body>  </html> |

# 十一、Ajax（web\_ajax）

## 1、Ajax简介

AJAX 是 Asynchronous JavaScript And XML 的简称。直译为，异步的JS和XML。

AJAX的实际意义是，不发生页面跳转、异步载入内容并改写页面内容的技术。

AJAX也可以简单的理解为通过JS向服务器发送请求。

AJAX这门技术很早就被发明，但是直到2005年被谷歌的大量使用，才在市场中流行起来，可以说Google为AJAX的推广起到到推波助澜的作用。

## 2、Ajax同步处理

 AJAX出现之前，我们访问互联网时一般都是同步请求，也就是当我们通过一个页面向服务器发送一个请求时，在服务器响应结束之前，我们的整个页面是不能操作的，也就是直观上来看他是卡主不动的。

 这就带来了非常糟糕的用户体验。首先，同步请求时，用户只能等待服务器的响应，而不能做任何操作。其次，如果请求时间过长可能会给用户一个卡死的感觉。最后，同步请求的最大缺点就是即使整个页面中只有一小部分内容发生改变我们也要刷新整个页面。

## 3、Ajax异步处理

 而异步处理指的是我们在浏览网页的同时，通过AJAX向服务器发送请求，发送请求的过程中我们浏览网页的行为并不会收到任何影响，甚至主观上感知不到在向服务器发送请求。当服务器正常响应请求后，响应信息会直接发送到AJAX中，AJAX可以根据服务器响应的内容做一些操作。

 使用AJAX的异步请求基本上完美的解决了同步请求带来的问题。首先，发送请求时不会影响到用户的正常访问。其次，即使请求时间过长，用户不会有任何感知。最后，AJAX可以根据服务器的响应信息局部的修改页面，而不需要整个页面刷新。

## 4、请求对象

向服务器发送请求，毫无疑问需要使用Http协议，也就是我们需要通过JS来向服务器发送一个请求报文。这里我们来回顾一下请求报文的格式：

|  |
| --- |
| 请求首行  请求头  空行  请求体 |

这是一个请求报文的格式，那我们如果手动的创建这么一个报文格式来发送给服务器想必是非常麻烦呢，于是浏览器为我们提供了一个XMLHttpRequest对象。

## 5、XMLHttpRequest

XMLHttpRequest对象是AJAX中非常重要的对象，所有的AJAX操作都是基于该对象的。

XMLHttpRequest对象用来封装请求报文，我们向服务器发送的请求信息全部都需要封装到该对象中。

这里需要稍微注意一下，XMLHttpRequest对象并没有成为标准，但是现在的主流浏览器都支持该对象，而一些如IE6的老版本浏览器中的创建方式有一些区别，但是问题不大。

获取XMLHttpRequest对象：

由于浏览器之间的差异，不同浏览器中获取XMLHttpRequest的方式不同，但总的来说一共有三种方式：

目前主流浏览器都支持

var xhr = new XMLHttpRequest()

IE6支持的方式

var xhr = new ActiveXObject("Msxml2.XMLHTTP")



IE5.5以下支持的方式

var xhr = new ActiveXObject("Microsoft.XMLHTTP")



根据三种不同的方式编写通用方法来获取XMLHttpRequest对象：

|  |
| --- |
| //获取XMLHttpRequest的通用方法  **function** getXMLHttpRequest(){  **var** xhr;  **try**{  //大部分浏览器都支持  xhr = **new** XMLHttpRequest();  }**catch**(e){  **try**{  //如果不支持，在这里捕获异常并且采用IE6支持的方式  xhr = **new** ActiveXObject("Msxml2.XMLHTTP");  }**catch**(e){  //如果还不支持，在这里捕获异常并采用IE5支持的方式  xhr = **new** ActiveXObject("Microsoft.XMLHTTP");  }  }  **return** xhr;  } |

## 6、XMLHttpRequest对象的方法

### 1）open(method,url,async)

 open()用于设置请求的基本信息，接收三个参数。

（1）method

 请求的方法：get或post

 接收一个字符串

（2）url

 请求的地址，接收一个字符串

（3）Assync

 发送的请求是否为异步请求，接收一个布尔值。

 true 是异步请求

 false 不是异步请求（同步请求）

### 2）send(string)

 send()用于将请求发送给服务器，可以接收一个参数

（1）string参数

 该参数只在发送post请求时需要。

 string参数用于设置请求体

### 3）setRequestHeader(header,value)

 用于设置请求头

（1）header参数

 字符串类型，要设置的请求头的名字

（2）value参数

 字符串类型，要设置的请求头的值



## 7、XMLHttpRequest对象的属性

### 1）readyState

 描述XMLHttpRequest的状态

 一共有五种状态分别对应了五个数字：

 0 ：请求尚未初始化，open()尚未被调用

 1 ：服务器连接已建立，send()尚未被调用

 2 ：请求已接收，服务器尚未响应

 3 ：请求已处理，正在接收服务器发送的响应

 4 ：请求已处理完毕，且响应已就绪。

### 2）status

 请求的响应码

 200 响应成功

 404 页面为找到

 500 服务器内部错误

… … …

### 3）onreadystatechange

 该属性需要指向一个函数

 该函数会在readyState属性发生改变时被调用

### 4）responseText

 获得字符串形式的响应数据。

### 5）responseXML（用的比较少）

 获得 XML 形式的响应数据。

## 8、示例代码

### 1）使用Ajax发送GET请求

|  |
| --- |
| //获取xhr对象  **var** xhr = getXMLHttpRequest();  //设置请求信息  xhr.open("get","AjaxServlet?&t="+Math.random(),**true**);  //发送请求  xhr.send();  //监听请求状态  xhr.onreadystatechange = **function**(){  //当响应完成  **if**(xhr.readyState == 4){  //且状态码为200时  **if**(xhr.status == 200){  //接收响应信息（文本形式）  **var** text = xhr.responseText;  //弹出消息  alert(text);  }  };  }; |

这是一个最简单的AJAX代码，向AjaxServlet发送了一个get请求，并且在页面中输出响应的内容

### 2）使用Ajax发送Post请求

|  |
| --- |
| //获取xhr对象  **var** xhr = getXMLHttpRequest();  //设置请求信息  xhr.open("post","2.jsp",**true**);  //设置请求头  xhr.setRequestHeader("Content-Type","application/x-www-form-urlencoded");  //发送请求  xhr.send("hello=123456");  //监听请求状态  xhr.onreadystatechange = **function**(){  //当响应完成  **if**(xhr.readyState == 4){  //且状态码为200时  **if**(xhr.status == 200){  //接收响应信息（文本形式）  **var** text = xhr.responseText;  //弹出消息  alert(text);  }  }; |

# 十二、json（web\_json、JsonUtils）

http://www.json.org/

## 1）Json 简述

（1）上边两个示例返回都是一个字符串，那当我们需要返回一个复杂的数据时，比如说需要返回一个对象时，就需要定义一下数据的格式。

（2）AJAX一开始使用的时XML的数据格式，XML的数据格式非常简单清晰，容易编写，但是由于XML中包含了过多的标签，以及十分复杂的结构，解析起来也相对复杂，所以目前来讲，AJAX中已经几乎不使用XML来发送数据了。取而代之的是一项新的技术JSON。

（3）JSON是JavaScript Object Notation 的缩写，是JS提供的一种数据交换格式。

（4）JSON对象本质上就是一个JS对象，但是这个对象比较特殊，它可以直接转换为字符串，在不同语言中进行传递，通过工具又可以转换为其他语言中的对象。

## 2）Json 例子

例，有如下一个JSON对象

 {“name”:”sunwukong” , ”age”:18 , ”address”:”beijing” }

 这个对象中有三个属性name、age和address

 如果将该对象使用单引号引起了，那么他就变成了一个字符串

 ‘{“name”:”sunwukong” , ”age”:18 , ”address”:”beijing” }’

 变成字符串后有一个好处，就是可以在不同语言之间传递。

 比如，将JSON作为一个字符串发送给Servlet，在Java中就可以把JSON字符串转换为一个Java对象。

## 3）JSON通过6种数据类型来表示

 （1）字符串

 例子：”字符串”

 注意：不能使用单引号

 （2）数字：

 例子：123.4



（3）布尔值：

 例子：true、false



（4）null值:

 例子：null



（5）对象

 例子：{“name”:”sunwukong”, ”age”:18}



（6）数组

 例子：[1,”str”,true]

## 4）在JS中操作JSON

 创建JSON对象

 var json = {“name1”:”value1”,”name2”:”value2” , “name3”:[1,”str”,true]};

 var json = [{“name1”:”value1”},{“name2”:”value2”}];

 JSON对象转换为JSON字符串

 JSON.stringify(JSON对象)

 JSON字符串转换为JSON对象

 JSON.parse(JSON字符串)

## 5）在Java中操作JSON

（1）在Java中可以从文件中读取JSON字符串，也可以是客户端发送的JSON字符串，所以第一个问题，我们先来看如何将一个JSON字符串转换成一个Java对象。

（2）首先解析JSON字符串我们需要导入第三方的工具，目前主流的解析JSON的工具大概有三种json-lib、jackson、gson。三种解析工具相比较json-lib的使用复杂，且效率较差。而Jackson和gson解析效率较高。使用简单，这里我们以gson为例讲解。

（3）Gson是Google公司出品的解析JSON工具，使用简单，解析性能好。

（4）Gson中解析JSON的核心是Gson的类，解析操作都是通过该类实例进行。

 废话不多说直接上代码。

## 6）废话不多说直接上代码

解析外部文本文件中的JSON

（1）JSON字符串转换为对象

|  |
| --- |
| String json = "{\"name\":\"张三\",\"age\":18}";  Gson gson = **new** Gson();  //转换为集合  Map<String,Object> stuMap = gson.fromJson(json, Map.**class**);  //如果编写了相应的类也可以转换为指定对象  Student fromJson = gson.fromJson(json, Student.**class**); |

（2）对象转换为JSON字符串

|  |
| --- |
| Student stu = **new** Student("李四", 23);  Gson gson = **new** Gson();  //{"name":"李四","age":23}  String json = gson.toJson(stu);    Map<String , Object> map = **new** HashMap<String, Object>();  map.put("name", "孙悟空");  map.put("age", 30);  //{"age":30,"name":"孙悟空"}  String json2 = gson.toJson(map);    List<Student> list = **new** ArrayList<Student>();  list.add(**new** Student("八戒", 18));  list.add(**new** Student("沙僧", 28));  list.add(**new** Student("唐僧", 38));  //[{"name":"八戒","age":18},{"name":"沙僧","age":28},{"name":"唐僧","age":38}]  String json3 = gson.toJson(list); |

## 7）code（web\_json）

### （1）Student

|  |
| --- |
| package com.alex.j.web.entity;  public class Student {    private int id;  private String name;  private int age;    public Student() {    }    public Student(int id, String name, int age) {  super();  this.id = id;  this.name = name;  this.age = age;  }    public int getId() {  return id;  }    public void setId(int id) {  this.id = id;  }    public String getName() {  return name;  }    public void setName(String name) {  this.name = name;  }    public int getAge() {  return age;  }    public void setAge(int age) {  this.age = age;  }  @Override  public String toString() {  return "Student [id=" + id + ", name=" + name + ", age=" + age + "]";  }    } |

### （2）GsonTestJson

|  |
| --- |
| package com.alex.j.web.json;  import java.util.ArrayList;  import java.util.List;  import org.junit.Test;  import com.alex.j.web.entity.Student;  import com.google.gson.Gson;  public class GsonTestJson {  @Test  public void testJavaToJson() {  Student student = new Student(1001, "Tom", 22);  Gson gson = new Gson();  String stujson = gson.toJson(student);  System.out.println(stujson);  }    @Test  public void testJavaListToJson() {  List<Student> stulist = new ArrayList<Student>();    Student student1 = new Student(1001, "Tom", 22);  Student student2 = new Student(1002, "Alex", 18);  stulist.add(student1);  stulist.add(student2);    Gson gson = new Gson();  String stu\_list\_json = gson.toJson(stulist);  System.out.println(stu\_list\_json);  }    @Test  public void testJsonToJava(){  String jsonstr = "{\"id\":1001,\"name\":\"Tom\",\"age\":22}";  Gson gson = new Gson();  Student student = gson.fromJson(jsonstr, Student.class);  System.out.println(student);  }  } |

### （3）jar 包

gson-2.2.4.jar

## 8、Json 的Java常用转换方式

### 1）Gson

#### （1）添加依赖

|  |
| --- |
| <dependency>  <groupId>com.google.code.gson</groupId>  <artifactId>gson</artifactId>  <version>2.2.4</version>  </dependency> |

#### （2）转化方法

① 对象转Json:

|  |
| --- |
| Gson gson = new Gson();  String json = gson.toJson(Object object); |

② Json转对象

|  |
| --- |
| gson.fromJson(String json, Class<T> classOfT) |

③.集合转Json

|  |
| --- |
| Gson gson = new Gson();  String json = gson.toJson(Object object); |

④.Json转集合

|  |
| --- |
| TypeToken<T> typeOfT = new TypeToken<T>(){};  T fromJson = (T)gson.fromJson(json, typeOfT.getType()); |

### 2）Json-Lib

#### （1）添加依赖

|  |
| --- |
| <dependency>  <groupId>net.sf.json-lib</groupId>  <artifactId>json-lib</artifactId>  <version>2.4</version>  <classifier>jdk15</classifier>  </dependency> |

#### （2）转化方法

① 对象转Json

|  |
| --- |
| JSONObject fromObject = JSONObject.fromObject(Object object);  String string = fromObject.toString(); |

② Json转对象

|  |
| --- |
| JSONObject fromObject2 = JSONObject.fromObject(string);  Object bean =JSONObject.toBean(JSONObject jsonObject, Class beanClass) |

③ 集合转Json

|  |
| --- |
| JSONArray fromObject = JSONArray.fromObject(Object object);  String string = fromObject.toString(); |

④ Json转集合

|  |
| --- |
| JSONArray fromObject2 = JSONArray.fromObject(string);  Collection collection = JSONArray.toCollection  (JSONArray jsonArray, Class objectClass) |

### 3）Fastjson

Fastjson是阿里巴巴公司开发的，Java语言编写的，JSON的处理器。

#### 添加依赖

|  |
| --- |
| <dependency>  <groupId>com.alibaba</groupId>  <artifactId>fastjson</artifactId>  <version>1.2.31</version>  </dependency> |

#### （2）转化方法

① 对象转Json

|  |
| --- |
| JSON.toJSONString(Object object); |

② Json转对象

|  |
| --- |
| JSON.parseObject(String text,Class<T> Class); |

③ 集合转Json

|  |
| --- |
| JSON.toJSONString(Object object) |

④ Json转集合:

|  |
| --- |
| JSON.parseArray(String text,Class<T> Class); |

### 4）Jackson

#### （1）添加依赖

|  |
| --- |
| <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-databind</artifactId>  <version>2.8.8</version>  </dependency> |

#### （2）转换方法

|  |
| --- |
| public static String object\_to\_json(Object object) throws JsonProcessingException {  ObjectMapper mapper = new ObjectMapper();  return mapper.writeValueAsString(object);  }  public static <T> T json\_to\_object(String json, Class<T> mainClass, Class<?>... parametricClasses) throws JsonParseException, JsonMappingException, IOException {  ObjectMapper mapper = new ObjectMapper();  JavaType parametricType = mapper.getTypeFactory().constructParametricType(mainClass, parametricClasses);  T readValue = mapper.readValue(json, parametricType);  return readValue;  } |

## 9、js 常用转换json方式（web\_js\_json）

### 1）Json 概述

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式，采用完全独立于语言的文本格式，是理想的数据交换格式。同时，JSON是 JavaScript 原生格式，这意味着在 JavaScript 中处理 JSON数据不须要任何特殊的 API 或工具包。

本文主要是对JS操作JSON的要领做下总结。

在JSON中，有两种结构：对象和数组。

1. 一个对象以“{”（左括号）开始，“}”（右括号）结束。每个“名称”后跟一个“:”（冒号）；“‘名称/值’ 对”之间运用 “,”（逗号）分隔。 名称用引号括起来；值如果是字符串则必须用括号，数值型则不须要。例如：

var o={"xlid":"cxh","xldigitid":123456,"topscore":2000,"topplaytime":"2009-08-20"}；

2. 数组是值（value）的有序集合。一个数组以“[”（左中括号）开始，“]”（右中括号）结束。值之间运用 “,”（逗号）分隔。

例如：

var jsonranklist=[{"xlid":"cxh","xldigitid":123456,"topscore":2000,"topplaytime":"2009-08-20"},{"xlid":"zd","xldigitid":123456,"topscore":1500,"topplaytime":"2009-11-20"}];

为了方便地处理JSON数据，JSON提供了json.js包，下载地址：http://www.json.org/json.js

在数据传输流程中，json是以文本，即字符串的形式传递的，而JS操作的是JSON对象，所以，JSON对象和JSON字符串之间的相互转换是关键。例如：

JSON字符串:

var str1 = '{ "name": "cxh", "sex": "man" }';

JSON对象:

var str2 = { "name": "cxh", "sex": "man" };

### 2）Json字符串转为Json对象

三种方式

|  |
| --- |
| var obj = eval('(' + str + ')');  var obj = str.parseJSON();  var obj = JSON.parse(str); |

### 3）JSON对象转化为JSON字符串

两种方式

|  |
| --- |
| var last=obj.toJSONString();  var last=JSON.stringify(obj); |

### 4）Json读取

|  |
| --- |
| var str = '{ "name": "cxh", "sex": "man" }';  var obj = str.parseJSON();  Alert(obj.name);  数据组  var str='[{"name":"cxh","sex":"man"},{"name":"cxh1","sex":"man1"}]';  var obj = str.parseJSON();  alert(obj[0].name) |

### 5）注意事项

上面的多个要领中，除了eval()函数是js自带的之外，其他的多个要领都来自json.js包。新版本的 JSON 修改了 API，将 JSON.stringify() 和 JSON.parse() 两个要领都注入到了 Javascript 的内建对象里面，前者变成了 Object.toJSONString()，而后者变成了 String.parseJSON()。如果提示找不到toJSONString()和parseJSON()要领，则说明您的json包版本太低

https://bestiejs.github.io/json3/#section\_1

https://www.cnblogs.com/liuqiyun/p/8594898.html

https://blog.csdn.net/chenchunlin526/article/details/78850996

# 十三、Jquery（web\_jquery）

## 1、JQuery对Ajax的支持（web\_ajax）

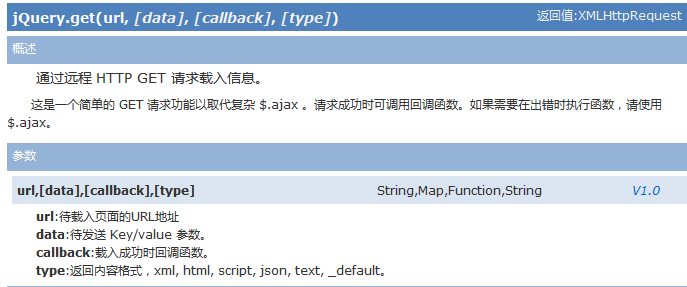
JQuery是当前比较主流的 JavaScript 库，封装了很多预定义的对象和实现函数，帮助使用者建立有高难度交互的页面，并且兼容大部分主流的浏览器.



JQuery同样提供了对Ajax的支持，可以更加方便快速的进行Ajax的开发，相关的方法有$.get $.post $.ajax等.



## 2、$.get方法



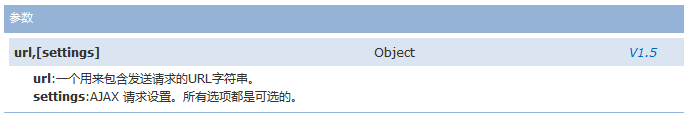
## 3、$.post方法



## 4、$.ajax方法

 jQuery 底层 AJAX 实现。简单易用的高层实现见 $.get, $.post 等。$.ajax() 返回其创建的 XMLHttpRequest 对象。大多数情况下你无需直接操作该函数，除非你需要操作不常用的选项，以获得更多的灵活性。最简单的情况下，$.ajax()可以不带任何参数直接使用。

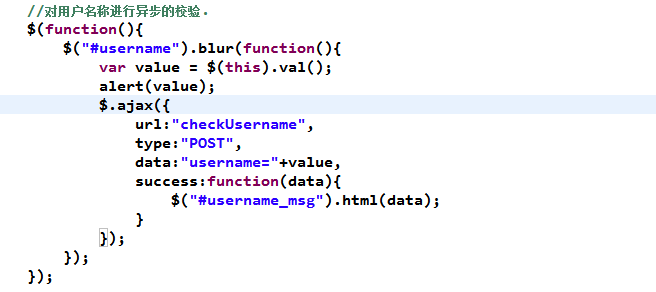
 $.ajax方法的参数



对于settings请求设置来说，所有选项都是可选的，详见jQuery手册

 具体的示例代码

 登录表单对信息的异步校验



 结合JSON,完成数据的动态展示



===========



# 十四、Book Project项目

## 1、Index.jsp

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  <%@ page import="com.alex.web.entity.BaseUser"%>  <%String basePath = request.getScheme() + "://" + request.getServerName() + ":" + request.getServerPort() + request.getContextPath();%>  <%String contextPath = request.getContextPath();%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Book Project</title>  <script type="text/javascript" src="style/jquery/jquery-1.8.2.min.js"></script>  <script type="text/javascript" src="<%=contextPath%>/style/js/json3.min.js"></script>  <link rel="stylesheet" href="${pageContext.request.contextPath}/style/css/index.css">  <script type="text/javascript">  </script>  <style type="text/css">    </style>  <%--  静态引入  <%@ include file="" %>  动态引入  <jsp:include page="inc.jsp" />  --%>  <jsp:include page="inc.jsp"></jsp:include>  </head>  <body>  <%  BaseUser baseUser = (BaseUser) session.getAttribute("baseUser");  if (baseUser != null) {  request.getRequestDispatcher("main.jsp").forward(request, response);  } else {  request.getRequestDispatcher("main.jsp").forward(request, response);  }  %>  </body>  </html> |

## 2、inc.jsp

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  <%@ page import="java.util.Map"%>  <%@ page import="java.util.HashMap"%>  <%@ page import="org.apache.commons.lang3.StringUtils"%>  <%@page isELIgnored="false" %>  <%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>  <%@taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>  <%String basePath = request.getScheme()+"://"+request.getServerName()+":"+request.getServerPort()+request.getContextPath()+"/";%>  <%String contextPath = request.getContextPath();%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title></title>  <script type="text/javascript" src="style/jquery/jquery-1.8.2.min.js"></script>  <script type="text/javascript" src="style/bootstrap/js/bootstrap.min.js"></script>  <script type="text/javascript" src="<%=contextPath%>/style/js/json3.min.js"></script>  <link rel="stylesheet" type="text/css" href="style/bootstrap/css/bootstrap.css"/>  <link rel="stylesheet" type="text/css" href="style/bootstrap/css/bootstrap-responsive.css" />  <link rel="stylesheet" type="text/css" href="style/css/index.css" />  <link rel="stylesheet" type="text/css" href="${pageContext.request.contextPath}/style/css/index.css"/>  <link rel="stylesheet" type="text/css" href="${pageContext.request.contextPath}style/foot/foot.css"/>  <link rel="stylesheet" type="text/css" href="${pageContext.request.contextPath}style/head/head.css"/>  <script type="text/javascript">  </script>  <style type="text/css">  </style>  </head>  <body>  </body>  </html> |

## 3、logo(icon)

如何在网页标题栏title加入logo(icon)图标？

打开某一个网页会在浏览器的标签栏处显示该网页的标题和图标，当网页被添加到收藏夹或者书签中时也会出现网页的图标，怎么在网页title左边显示网页的logo图标呢？

### 1）方法一（被动式）

制作一个ico格式的图片，将图片命名为favicon.ico，像素大小为16\*16，所使用的颜色不得超过16色，然后再把favicon.ico放到网站的根目录下就行了。这样的话浏览器会不停的搜索您的网站的根目录，只要它一发现了名字叫做favicon.ico 这个文件，就会将该图标显示在访问者的地址栏和收藏夹列表中了。

### 2）方法二（自动式）

在网页HEAD标记中添加如下代码：

标题栏：

<link rel="icon" href="ico地址" type="image/x-icon">

收藏夹：

<link rel="shortcut icon" href="ico地址" type="image/x-icon">

注意：图标要用 16\*16 色的。。。(保证了兼容性，无论在哪个地方都可以显示)

在这里要注意，如果将 16\*16 的 bmp 文件改为 ico ，在WINDOWS 支持，但放在浏览器或别的软件就会出错 。因此图片的格式要通过软件来调整，毕竟内部的信息是不一样的。

同时在使用方法二的时候，就不一定要把图形文件放在网站的根目录下面了，你可以放在任何位置，甚至直接连结到其他网站上面的图形文件也可以，而且图形文件的文件 名也不一定要叫做 favicon.ico，可以自己随便取。 不过要注意的是使用这个做法的话，只有在把有加入上面 HTML 语法的页面加入书签的时候才会有作用。

获取网站的浏览器上的icon图标

http://www.im286.com/favicon.ico

把前面的域名换成你要获取的网站

https://github.com/favicon.ico