

JavaWeb 课程系列

尚硅谷 java 研究院

版本: V 1.0

第 1 章 HTML 基础

1. 准备

1.1 选用最新版的 STS: spring-tool-suite-3.9.2.RELEASE

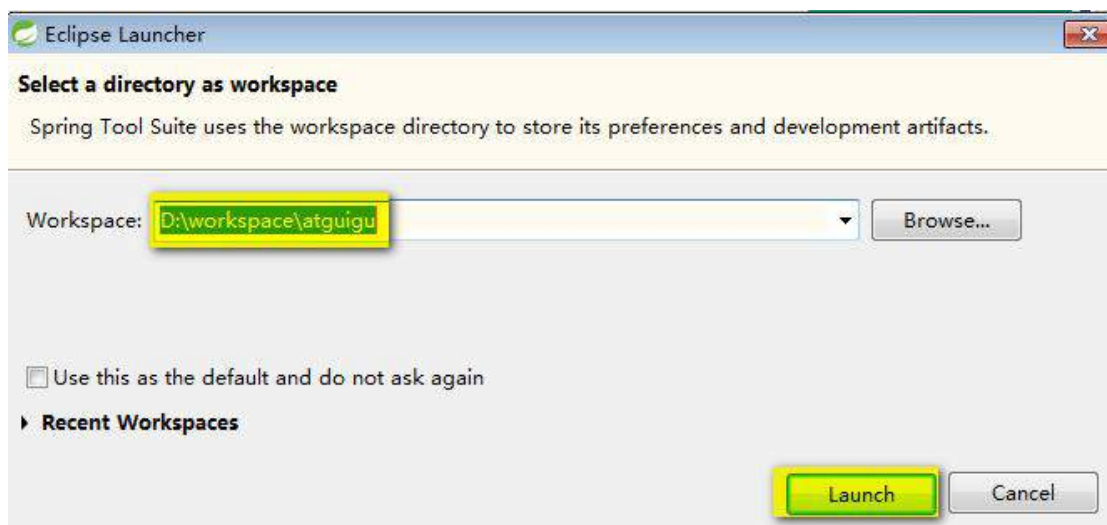
1.2 解压

解压到无中文无乱码的路径下，并修改 sts-bundle/sts-3.9.0.RELEASE/sts.ini 文件

```
-startup
plugins/org.eclipse.equinox.launcher_1.4.0.v20161219-1356.jar
--launcher.library
plugins/org.eclipse.equinox.launcher.win32.win32.x86_64_1.1.500.v20170531-1133
-product
org.springframework.ide
--launcher.defaultAction
openFile
-vmargs
-Dosgi.requiredJavaVersion=1.8
-Xms40m
-Dosgi.module.lock.timeout=10
-Xverify:none
-Dorg.eclipse.swt.browser.IEVersion=10001
-Xmx1200m
-Dfile.encoding=UTF-8
```

设置默认编码为 utf-8

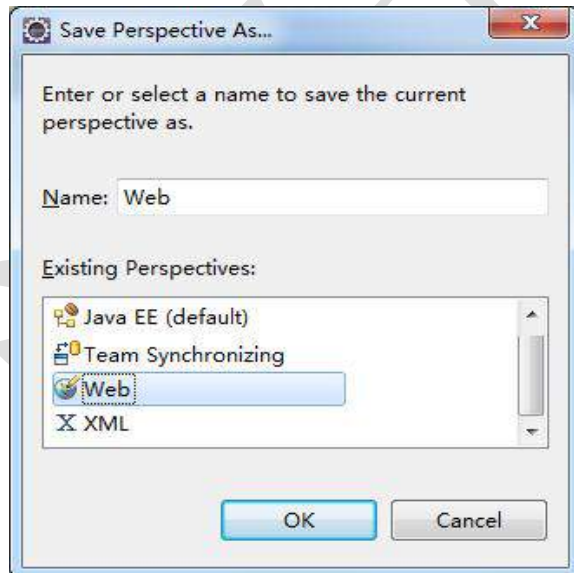
1.3 创建一个新的 Eclipse 工作空间



1.4 设置 Web 透视图并保存

1.4.1 需要保留的视窗：Package Explorer

1.4.2 保存当前设置：Window→Save Perspective As→Web



1.5 设置 JavaEE 透视图并保存

1.5.1 需要保留的视窗：Package Explorer、Outline、Console、Server、Navigator

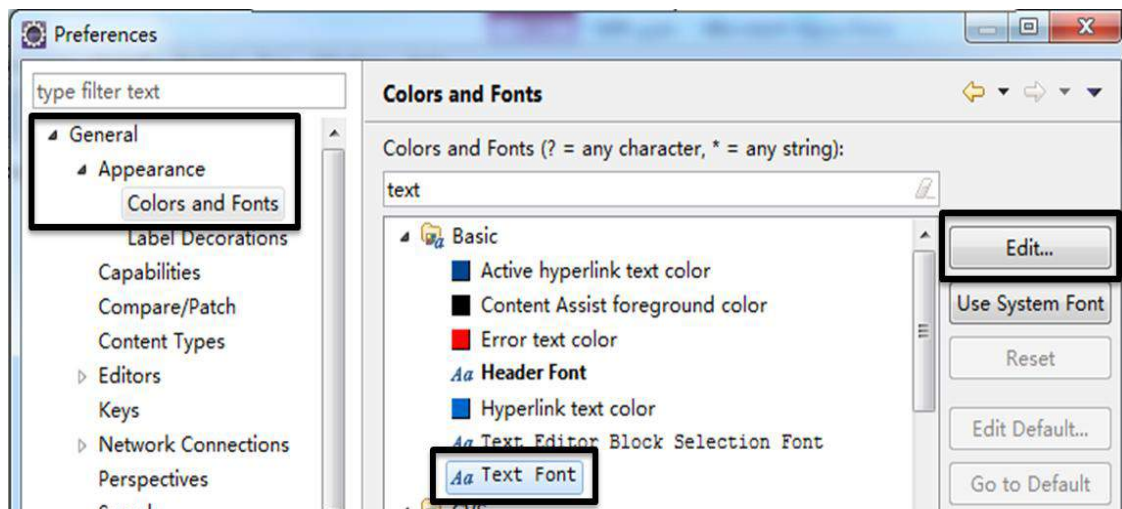
1.5.2 保存当前设置：Window→Save Perspective As→JavaEE

1.6 设置 New...菜单

操作路径：Window→Customer Perspective→Menu Visibility→只将需要的留下

1.7 设置字符显示大小

操作路径：Window→Preferences→General→Appearance→Colors and Fonts→Basic→Text Font→Edit→大小



2. 概念：b/s 与 c/s

2.1 架构了解

现在的软件开发的整体架构主要分为 B/S 架构与 C/S 架构：

2.1.1 b/s：浏览器/服务器

2.1.2 c/s：客户端/服务器

2.1.3 客户端：需要安装在系统里，才可使用

2.1.4 浏览器：浏览网页，读取 HTML 并显示

2.1.5 服务器：处理浏览器的请求

2.2 b/s 与 c/s 优劣

2.2.1 b/s 只要能上网就能使用，因为基本每台电脑都会有浏览器，维护方便。

2.2.2 c/s 必须安装在系统中，安装成功才可使用。在新的系统中没有安装不能使用，便携性差，维护成本高。

2.3 网页

浏览器中显示的内容，浏览器是网页的展示器。编写好的网页，放在浏览器中即可运行。编写网页我们使用的就是 HTML 语言

3. HTML 简介

HTML 指的是超文本**标记语言** (**Hyper Text Markup Language**)，我们也将 html 称为标签语言。他不是编译性语言，不是编程语言。只是一种标记标签，html 用定义好的标记标签来描述网页、

Html 网页文件的后缀名一般为.html。

4. HTML 基本语法

标签：由一组<>包围的关键字，表示标签的开始如:<p>。由</>包围的关键字表示标签的结束</p>，标签中间可以有内容体，如：<p>段落</p>。

标签之间不能交叉，标签如果不是自结束标签（
）必须成对出现。标签可以嵌套

HTML 不区分大小写，但是推荐使用小写。

标签中的属性必须有值，值必须使用单引号或双引号引起，如：

5. 第一个 HTML 页面

创建一个后为 helloworld.html 的文本文件，写入以下内容

```
<!-- 第一行声明 html 文档类型 -->

<!DOCTYPE html>

<!-- html 标签之间的文本描述网页 -->

<html>

<!-- head 之间的内容表示这个网页的头信息，如网页的标题，编码等 -->

  <head>

    <meta charset="UTF-8">

    <title>Insert title here</title>

  </head>
```

```
<!-- body 这个里面的内容是网页要显示出来的内容 -->

<body>

    Hello   World

</body>

</html>
```

6. HTML 的常用标签

6.1 标题标签

h1~h6 如: <h1>你好</h1>

6.2 段落标签

p 如: <p>锄禾日当午</p>

6.3 换行标签

br 如:
这是个自结束标签

6.4 无序列表

ul li 如:

```
    <li>天王盖地虎</li>
    <li>雪碧两块五</li>
</ul>
```

这是个组合标签, ul 和 li 需要一起使用

6.5 图片标签

6.5.1 img 如: src 表示图片的位置

6.5.2 相对路径:

- (1) 图片和页面在同一级目录: 图片名可以直接引用图片。
- (2) 图片在页面的下一级目录: 目录/图片名可以引用图片。
- (3) 图片在页面的上一级目录: ../图片名可以引用图片。

6.6 超链接标签

a 如：去 1.html href 表示点击后跳转去的位置

7. HTML 中的表格

HTML 中的表格使用 table 标签创建，表格由行（tr）组成，行由单元格（td）组成

```
<table>

    <tr>

        <td></td>

        <td></td>

        <td></td>

        <td rowspan="2"></td>

    </tr>

    <tr>

        <td></td>

        <td></td>

        <td></td>

    </tr>

    <tr>

        <td></td>

        <td></td>

        <td colspan="2"></td>

    </tr>

</table>
```

8. HTML 中的表单

表单类似生活中的单据，票据，申请表之类的东西，生活中我们经常会填写很多表单，比如入职申请表，入学登记表，员工信息表等。

```
<!--from 标签用来创建表单 -->

<form action="" >

<!--input 标签用来创建表单项，type 值就是收集的数据的类型 -->

    文本框:<input type="text" name="" />

    密码框:<input type="password" name="" />

    单选框:<input type="radio" name="" value="" />

    多选框:<input type="checkbox" name="" value="" />

    下拉列表:

        <select name="">

            <option value=""></option>

        </select>

<!--type 值为 submit 的按钮为提交按钮，点击时 form 表单会收集有 name 属性的表单项数据提交给 action 中的地址 -->

    提交按钮:<input type="submit" value="按钮上的文字" />

</form>
```

9. HTML 中最重要的元素 DIV 和 SPAN

- 1) Div 是 html 中最灵活最重要的元素，div 就像一个小箱子，里面可以装很多内容。他是块级元素，他会占用网页的一行。Div 可以通过调整自己的样式来完成网页的复杂布局，
- 2) span 是内联元素，只会占用自身的大小，主要用来为选中文字设置样式。并没有什么实际意义

10.HTML 中的转义字符

10.1 空格

10.2 <

<

10.3 >

>

第 2 章 CSS

1. css 简介

CSS 指层叠样式表 (Cascading Style Sheets)。主要用来设置网页中元素的样式。如边框，颜色，位置等...

CSS 即可以现在 HTML 中，也可以写在元素的 style 属性里面，还可以写在.css 外部文件里然后引入到页面

2. 基本语法

2.1 语法规范

2.1.1 写在外部文件或者 html 头标签里的时候。

```
选择器 {  
    样式名: 样式值;  
    样式名: 样式值;  
    .....  
}
```

2.1.2 写在元素的 style 属性里面的时候。“样式名: 样式值; 样式名: 样式值;”

```
<p style="color:red;font-size:16px;">你好呀! </p>
```

2.2 编写位置

2.2.1 内部

标签的 style 属性中:

```
<p style="color: red ; font-size: 12px">落霞与孤鹜齐飞</p>
```

写到 html 头的 style 标签中

```
<style type="text/css">
```

```
  p {
```



```
        color: blue;

        background-color: yellow;

    }

</style>
```

2.2.2 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" />
```

3. 选择器

3.1 标签选择器

按照标签名选中相应的元素。

```
p {
    color:red;
}
```

3.2 Id 选择器

按照元素的 id 选中相应的元素，使用 #id 值

```
<p id="abc">大家好</p>

#abc {
    color:red;
}
```

3.3 类选择器

按照元素的类名选中相应的元素，使用 .class 值

```
<p class="foot">你好</p>
<b class="foot">你也好</b>

.foot {
    color:red;
}
```

3.4 组选择器

可以同时使用多个选择器选中一组元素，使用，分隔不同的选择器
选择器 1，选择器 2，……，选择器 N{

```
    color: red;
}
```

4. 常用样式

4.1 颜色

color: red;

颜色可以写颜色名如: black, blue, red, green, white, yellow 等

颜色也可以写 rgb 值和十六进制表示值: 如 rgb(255,0,0), #00F6DE, 如果写十六进制值必须加#

4.2 宽度

width:19px;

宽度可以写像素值: 19px;

也可以写百分比值: 20%;

4.3 高度

height:20px;

同宽度一样

4.4 背景颜色

background-color:#0F2D4C

4.5 常见 CSS 样式

4.5.1 字体样式

字体颜色	color: #bbffaa;
字体大小	font-size: 20px;

4.5.2 黑色 1 像素实线边框

border: 1px solid black;

4.5.3 DIV 居中

margin-left: auto;
margin-right: auto;

4.5.4 文本居中

text-align: center;

4.5.5 超链接去下划线

text-decoration: none;

第 3 章 JavaScript

1. JavaScript 简介

1.1 起源

在 1995 年时，由 [Netscape](#) 公司的 [Brendan Eich](#)，在[网景导航者](#)浏览器上首次设计实现而成。[Netscape](#) 在最初将其脚本语言命名为 [LiveScript](#)，因为 [Netscape](#) 与 [Sun](#) 合作，网景公司管理层希望它外观看起来像 [Java](#)，因此取名为 JavaScript。

作用：

在前段页面中验证用户提交信息是否符合要求
和服务器发生交互，判断用户名是否存在[ajax]

1.2 特性

1.2.1 脚本语言。JavaScript 是一种解释型的脚本语言，C、C++、Java 等语言先编译后执行，而 JavaScript 是在程序的运行过程中逐行进行解释。

1.2.2 基于对象。JavaScript 是一种基于对象的脚本语言，它不仅可以创建对象，也能使用现有的对象。

1.2.3 简单。JavaScript 语言中采用的是弱类型的变量类型，对使用的数据类型未做出严格的要求，是基于 Java 基本语句和控制的脚本语言。

1.2.4 动态性。JavaScript 是一种采用事件驱动的脚本语言，它不需要经过 Web 服务器就可以对用户的输入做出响应。

1.2.5 跨平台性。JavaScript 脚本语言不依赖于操作系统，仅需要浏览器的支持。因此一个 JavaScript 脚本在编写后可以带到任意机器上使用，前提是机器上的浏览器支持 JavaScript 脚本语言，目前 JavaScript 已被大多数的浏览器所支持。

2. 基本语法

js 需要包括在<script>标签中，这个标签可以出现在页面的任何位置

2.1 变量

2.1.1 声明：

使用 var 如：var x=65; var y="你好";

- 1) 变量的声明不需要指定数据类型，可以接受所有的数据类型
- 2) 变量名区分大小写，abc 和 aBc 是两个不同的变量

2.1.2 赋值：

x=44;x="abc";x=new Date();

- 1) 变量可以接受任何值。
- 2) 声明和赋值也可同时进行 如 var x="abc";

2.2 函数

2.2.1 声明：

使用 function 关键字，没有指定返回值一说！参数列表也没有指定参数类型一说，因为 js 所有类型都使用 var 来声明

- 1) var abc=function(a,b){ return a+b;}

函数在 js 中也是一种对象，可以将函数的引用赋值给变量

- 2) function add(a,b){ return a+b;}

为函数起个名字叫 add

2.2.2 使用:

调用函数方法

- 1) 如果是声明方式第一种, 使用变量名+()的方式进行调用

```
abc(1,2);
```

- 2) 如果是声明方式第二种的, 直接使用函数名调用;

```
add(1,2);
```

注意: js 调用函数的时候不会检查参数列表, 所以 js 中没有重载一说, `add(1,2)`; `add(1)`; `add(1,"abc")`; `add("abc")`; `add(1,"666",true);add()`; 都是调用的同一个方法。

2.3 对象

2.3.1 对象的创建:

- 1) `var obj = new Object();`
- 2) `var obj = {};`

2.3.2 为对象添加属性方法:

js 中动态的为对象添加属性和方法。

- 1) 动态添加

```
obj.name = "张三";  
  
obj.age = 18;  
  
obj.work = function(){  
    alert("我在工作!");  
}
```

- 2) 创建时指定

```
obj = {  
    name:"张三",
```

```
        age:18,  
        work:function(){  
            alert("我在工作")  
        }  
    }  
}
```

2.3.3 使用对象的属性或方法

- 1) 使用属性: `alert(obj.name)`
- 2) 调用方法: `obj.work()`;

2.4 注释

//: 表示单行注释

/* */: 表示多行注释

2.5 其他语法

java 中的 `for`, `while`, `if-else`, `switch`, `try-catch`, `break`, `continue`, 以及各种运算符, 在 js 中也是按照同样的方式使用的。这里不再赘述。

3. js 事件

3.1 思考: js 脚本的嵌入方式

- 1) 浏览器的加载顺序。遇到 js 执行 js, 执行完成后继续加载。可能会导致文档阻塞, 加载不出内容
- 2) 文档全部加载完成后再加载 js。js 中定义了一种 `window.onload` 事件可以解决以上问题

3.2 简介：什么是事件？

事件就是浏览器或者用户交互时触发的行为。比如按钮点击，表单提交，鼠标滑动等等...

3.3 事件的分类：

- 1) 系统事件：如：文档加载完成。
- 2) 用户事件：如：鼠标移入移出，单击双击等。

3.4 事件触发：

系统事件会由系统触发，如 `window.onload` 事件，用户事件由用户行为触发如 `click` 事件。主要讲解的系统事件是 `window.onload`，用户事件主要在操作 `html` 文档的时候触发。详细事件列表可参考 `w3c` 离线文档中的 `JavaScript→dom` 事件

3.5 常见的事件

onblur	元素失去焦点。
onchange	域的内容被改变。
onclick	当用户点击某个对象时调用的事件句柄。
onfocus	元素获得焦点。
onkeydown	某个键盘按键被按下。
onload	一张页面或一幅图像完成加载

3.6 事件响应：

- 1) 我们希望某个事件发生的时候我们可以做一些事情。这个称为事件的响应，比如用户点击了一个按钮，我弹出一个框告诉用户，你成功的点击了这个按钮。
- 2) 事件触发后我们要执行的方法称为响应函数。如何将响应函数与事件关联起来。

我们常使用为事件赋值函数的方法。如 `window.onload` 事件触发时我们执行弹出对

话框

```
window.onload = function(){  
    alert("文档加载完成了！")  
}
```

我们也可以使用标签的事件属性来触发响应函数，如：

```
<a href="atguigu.com" onclick="gotoguigu()">尚硅谷</a>
```

//onclick 会触发 gotoguigu()函数

我们在<script></script>中定义这个函数

```
<script type="text/javascript">  
    function gotoguigu(){  
        alert("我要去上硅谷")  
    }  
</script>
```

3.7 取消事件的默认行为

- 1) 默认行为：某些事件触发后，系统会有默认的响应处理。如：

超链接点击后会自动跳转、表单提交点击后会发送请求

- 2) 取消默认行为的方式：

return false;即可

3.8 正确的 js 加载方式

文档加载完成后加载 js

所以我们以后写 js 的时候，请把他包在 window.onload 的响应函数里，表示文档加载完成后会执行函数里面的代码。

```
<script type="text/javascript">  
    window.onload = function(){  
        //js 代码
```



```

    }
</script>

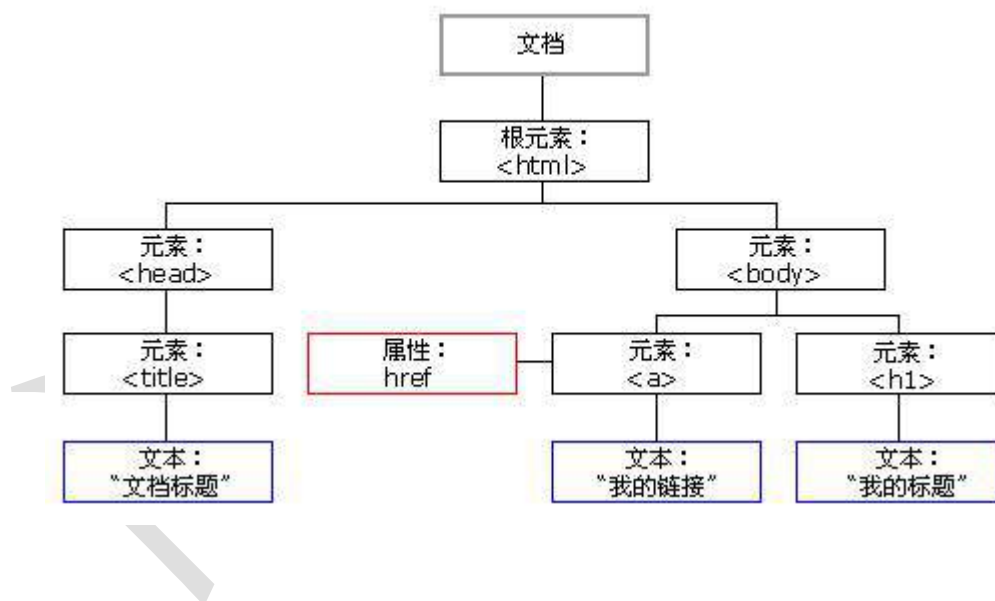
```

4. dom 操作

4.1 什么是 DOM:

Document Object Model(文档对象模型), 我们浏览器把整个网页会当成一个大的对象, 利用面向对象的方式操作网页内容。

4.2 DOM 模型:



4.3 document 对象

document 对象是 window 对象的一个属性, 代表当前整个 HTML 文档, 将这个文档抽象成了 document 对象, 包含了整个 dom 树的所有内容。

其本质是 window.document, 而 window.可以省略。直接使用 document

4.4 DOM 查询

元素查询

功能	API	返回值
●在整个文档范围内查询元素节点		
根据 id 值查询	document.getElementById("id 值")	一个具体的元素节点
根据标签名查询	document.getElementsByTagName("标签名")	元素节点数组
根据 name 属性值查询	document.getElementsByName("name 值")	元素节点数组
●在具体元素节点范围内查找子节点		
查找全部子节点	element.childNodes 【w3C 考虑换行, IE≤9 不考虑】	节点数组
查找第一个子节点	element.firstChild 【w3C 考虑换行, IE≤9 不考虑】	节点对象
查找最后一个子节点	element.lastChild 【w3C 考虑换行, IE≤9 不考虑】	节点对象
查找指定标签名的子节点	element.getElementsByTagName("标签名")	元素节点数组
●查找指定元素节点的父节点: element.parentNode		
●查找指定元素节点的兄弟节点		
查找前一个兄弟节点	node.previousSibling 【w3C 考虑换行, IE≤9 不考虑】	节点对象
查找后一个兄弟节点	node.nextSibling 【w3C 考虑换行, IE≤9 不考虑】	节点对象

4.5 属性操作

1) 读取属性值

元素对象.属性名

2) 修改属性值

元素对象.属性名=新的属性值

4.6 文本操作

1) 读取文本值: element.firstChild.nodeValue

2) 修改文本值: element.firstChild.nodeValue=新文本值

第 4 章 JQuery

1. 简介

- 1) jquery 是目前最流行的一种 JavaScript 库。
- 2) 所谓 JavaScript 库就是对 JavaScript 进行进一步封装和开发，然后将其打包为 js 文件方便重复调用。jquery 也可称为 JavaScript 框架。
- 3) jquery 的主要功能是简化 JavaScript 的开发工作。并且基本解决了浏览器的兼容问题。

2. 核心函数

2.1 \$是 jquery 的核心函数。

jquery 的大部分功能都是核心函数来完成的。

2.2 核心函数根据实参的不同，有四种不同的用法

2.2.1 传入一个函数作为参数

如：\$(function(){})

作用：和 window.onload = function(){}一样，在文档加载完成后调用

2.2.2 传入一个选择器的字符串

如：\$("#id")

作用：和 document.getElementById("id")一样，从文档查询需要的元素

2.2.3 传入一段 HTML 代码

如：\$("广州")

作用：创建一个 li 对象

2.2.4 传一个 DOM 对象

如: `var ele=document.getElementById("abc");` `var x= $(ele);`

作用: 将一个 js 获取的 dom 对象转化为一个 jquery 对象。这样就可以使用 jquery 里面的方法了

3. jquery 对象与 dom 对象

3.1 dom 对象

dom 对象是原生的网页文档对象。可以通过 js 获取到文档对象。然后进行增删改操作。

3.2 jquery 对象

jquery 对象是使用 jquery 包装后的文档对象。只有这个对象才可以调用 jquery 的方法。dom 对象不是 jquery 的对象, 所以我们不能调用 jquery 的方法, 我们需要将其包装为 jquery 对象才可

3.2.1 DOM 对象

通过原生 JS 获取的对象是 DOM 对象

3.2.2 jQuery 对象

通过 jQuery 核心函数包装过的对象叫做 jQuery 对象

3.2.3 比较

- 1) 两种对象之间不能互相调用对方的方法
- 2) 命名上的区别:
jQuery 对象命名时习惯加上 \$, 加以区分。

3.2.4 转换

- 1) DOM --> jQuery
\$(DOM 对象)

2) jQuery --> DOM

jQuery 对象[索引]

jQuery 对象的本质就是 DOM 对象的数组，所以可以通过给对象加下标的形式获取数组中的 DOM 对象

4. jquery 选择器

4.1 选择器简介

- 1) jQuery 的最厉害的地方就是它拥有众多的选择器。
- 2) jQuery 的选择器主要是集合 CSS 和 XPath 部分语法。
- 3) 选择器可以很方便的获取到页面中元素。

4.2 常见 jquery 选择器

4.2.1 基本选择器

- 1) id 选择器 `$("#id")`
- 2) 类选择器 `$(".class")`
- 3) 元素选择器 `$("标签名")`
- 4) 选择所有元素 `$("*")`
- 5) 选择器分组 `$("选择器 1, 选择器 2, 选择器 N")`

4.1.2 层次选择器

- 1) 后代元素 `$("祖先元素 后代元素")`
- 2) 子元素 `$("父元素 > 子元素")`
- 3) 下一个兄弟元素 `$("前一个 + 后一个")`
- 4) 下边所有兄弟元素 `$("前一个 ~ 后边所有")`

4.1.3 过滤选择器

参见 jqueryAPI

5. jquery 筛选

以已经查到的元素为标准，再进行查询。DOM 查询

5.1 过滤

<code>eq(index -index)</code>	获取第 N 个元素	<code>\$("p").eq(1);</code>
<code>first()</code>	获取第一个元素	<code>\$("p").first();</code>
<code>last()</code>	获取最后一个元素	<code>\$("p").last();</code>

5.2 查找

<code>children([expr])</code>	获取所有的子元素（不包括孙子<后代元素>）
<code>find(expr obj ele)</code>	找出所有的后代元素
<code>next([expr])</code>	相邻后一个的元素
<code>parent([expr])</code>	获取某元素的直接父元素
<code>parents([expr])</code>	获取某元素的祖先元素，不包含根元素
<code>prev([expr])</code>	获取元素的前一个元素

5.3 其余查看 jquery 文档

6. jquery 事件

6.1 常见事件列表

事件详情参见 jquery 文档

```
blur([[data],fn])
change([[data],fn])
click([[data],fn])
dblclick([[data],fn])
error([[data],fn])
focus([[data],fn])
focusin([[data],fn])
```

```
focusout([data],fn)
keydown([[data],fn))
keypress([[data],fn))
keyup([[data],fn))
mousedown([[data],fn))
mouseenter([[data],fn))
mouseleave([[data],fn))
mousemove([[data],fn))
mouseout([[data],fn))
mouseover([[data],fn))
mouseup([[data],fn))
resize([[data],fn))
scroll([[data],fn))
select([[data],fn))
submit([[data],fn))
unload([[data],fn))
```

6.2 事件绑定

1) 使用事件对应的函数进行绑定

```
如: $("#btn").click(function(){
    alert("我被点击啦!");
})
```

2) 使用 bind() 绑定事件: 用法: 元素.bind(事件名,[参数],回调方法)

```
如: $("p").bind("click", function(){
    alert( $(this).text() );
});
```

可以绑定多个事件, 多个事件用空格隔开

```
如: $('#foo').bind('mouseenter mouseleave', function() {
    $(this).toggleClass('entered');
});
```

3) 绑定一个一次性的事件, 事件只会触发一次。one():

```
如: $("p").one("click", function(){
    alert( $(this).text() );
});
```

```
});  
4) 为当前的对象以及以后创建的对象都绑定此事件  
如: $("p").live("click", function(){  
    alert("我是 p! ");  
});
```

6.3 解除绑定

使用 `unbind()` 方法解除事件绑定。

- 1) 不传参数, 取消当前元素的所有事件
如: `$("#p").unbind()`
- 2) 传递参数, 取消某个事件
如: `$("#p").unbind("click")`
- 3) 传递多个参数, 用空格隔开: 取消一组事件
如: `$("#p").unbind("click mouseover")`;

6.4 事件冒泡

例子:

```
<div>  
    <p>你好<p>  
</div>
```

为 `div` 和 `p` 同时绑定点击事件。当点击 `p` 的时候, `div` 的点击事件也会被触发
`$("#div").click(function(){alert("我是 div");});`
`$("#p").click(function(){alert("我是 p");});`
当我们点击 `p` 的时候, 先弹出我是 `p`, 又弹出我是 `div`。

阻止事件冒泡:

`return false;` 即可。

7. dom 增删改

7.1 文档操作

对 `dom` 对象的增删改

7.1.1 内部插入

`append(content|fn)`: 父.`append`(子): 父元素的最后插入子元素
`prepend(content|fn)`: 父.`prepend`(子): 父元素的最前面插入子元素

`appendTo(content)`: 子.`appendTo`(父): 子元素添加到父元素的最后
`prependTo(content)`: 子.`prependTo`(父): 子元素添加到父元素的最前面

7.1.2 外部插入

`after(content|fn)` : `A.after(B)`: a 的后边插入 b
`insertAfter(content)` : `A.insertAfter(B)`: 把 a 插入到 b 后边
`before(content|fn)` : `A.before(B)`: a 的前边插入 b
`insertBefore(content)` : `A.insertBefore(B)`: 把 a 插入到 b 的前边

7.1.3 替换

`replaceWith(content|fn)` : `A.replaceWith(B)`: A 被 B 替换 (A 不存在,B 存在)
`replaceAll(selector)` : `A.replaceAll(B)`: A 替换所有 B (B 不存在, A 存在)

7.1.4 删除

`empty()` : `A.empty()`: 将 A 元素下的子元素全部删除 (将 A 置空, 子元素没有, A 还在)
`remove([expr])` : `A.empty()`: 将 A 元素删除 (A 不存在)

7.1.5 复制

`clone([Even[,deepEven]])` : `A.clone()`: 克隆 A 元素并选中克隆的副本

7.2 属性操作

对 dom 对象属性的增删改

7.2.1 属性

`attr(name|pro|key,val|fn)` :
 获取属性值: `$("#img").attr("src");`
 设置属性值: `$("#img").attr("src","test.jpg");`
 设置多属性: `$("#img").attr({ src: "test.jpg", alt: "Test Image" });`
`removeAttr(name)` : 移除属性: `$("#img").removeAttr("src");`
`prop(name|pro|key,val|fn)` 1.6+ : 一般用来操作内置属性
 获取属性值: `$("#input[type='checkbox']").prop("checked");`

设置属性值: `$("#input[type='checkbox']").prop("checked",true);`
设置多属性: `$("#input[type='checkbox']").prop({ disabled: true});`
`removeProp(name)`1.6+ : 移除属性: `$('#div').removeProp('style')`

7.2.2 CSS 类

`addClass(class|fn)` : `A.addClass("selected1 selected2")`: 为 A 添加两个 class
`removeClass([class|fn])` :
 `A.removeClass("selected1 selected1")`: 删除 A 的两个 class,
 `A.removeClass()`: 删除 A 的所有 class
`toggleClass(class|fn[,sw])` :
`A.toggleClass("highlight")`: 切换 A 的 class, 如果有 highlight 移除, 如果没有添加

7.2.3 HTML 代码/文本/值

`html([val|fn])` :
 获取 html 内容: `A.html()`;
 设置 html 内容: `A.html("链接")`
`text([val|fn])`
 获取文本内容: `$('#p').text()`;
 设置文本内容: `$('#p').text("Hello world!")`;
`val([val|fn|arr])`
 获取表单元素值: `$("#input").val()`;
 设置表单元素值: `$("#input").val("hello world!")`;

7.3 css 操作

对 dom 对象样式的增删改

7.3.1 CSS

`css(name|pro|[,val|fn])` :
 获取 css 值: `$("#p").css("color")`;
 设置 css 值:
 `$("#p").css({ color: "#ff0011", background: "blue" });`
 `$("#p").css("color","red")`;

7.3.2 位置

```
offset([coordinates])  
position()  
scrollTop([val])  
scrollLeft([val])
```

7.3.3 尺寸

```
height([val|fn])  
width([val|fn])  
innerHeight()  
innerWidth()  
outerHeight([options])  
outerWidth([options])
```

8. jquery 动画

1) show([speed],[easing],[fn]);

显示隐藏的元素:

```
$("#p").show();  
$("#p").show("slow");
```

2) hide([speed],[easing],[fn]);

隐藏显示的元素:

```
$("#p").hide();  
$("#p").hide("slow");
```

3) toggle([speed],[easing],[fn]);

切换元素的显示状态

用于绑定两个或多个事件处理器函数，以响应被选元素的轮流的 click 事件。

如果元素是可见的，切换为隐藏的；如果元素是隐藏的，切换为可见的。

```
$('#p').toggle(); //显示隐藏 p
```

```
$('#p').toggle("slow");
```

```
$('#foo').toggle(showOrHide); // showOrHide 是个 true 或者 false 值，如果这个参数  
为 true ， 那么匹配的元素将显示;如果 false ， 元素将隐藏
```

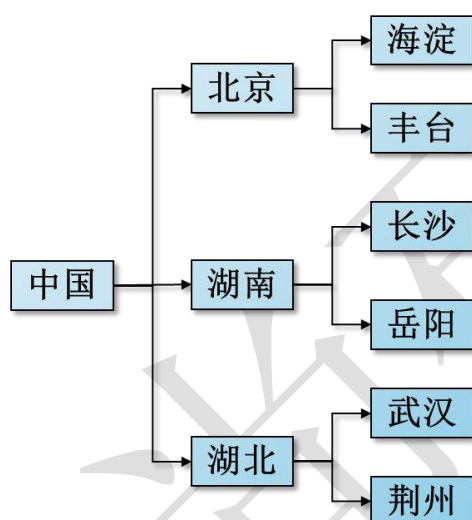
第 5 章 XML

1. xml 简介

1.1 什么是 xml

eXtensible Markup Language 可扩展标记语言——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。**



```
<?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>
```

我们不同的平台有他自己的数据格式,但是不同平台之间如果相互想传递数据,那么就应用同一种数据格式,这样大家都能读懂。就像加入 WTO 组织的各个国家一样。每个国家都有自己的语言和货币,但是如果大家都用自己的东西就很难沟通和衡量。那么我们就使用统一的方式,使用英语作为交流语言,使用美元作为

1.2 主要用途

xml 就是一种数据保存的格式而已。按照他的规则你就知道数据之间的关系。

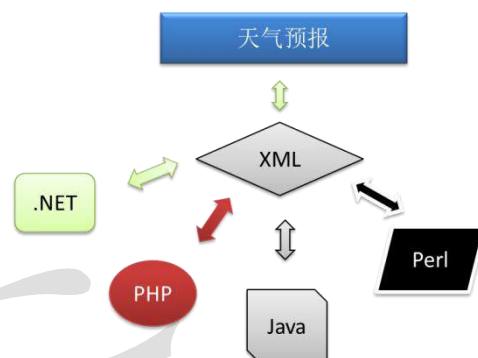
xml 经常用在下面的情况中

1.2.1 配置文件

- 1) JavaWeb 项目配置文件
- 2) c3p0 配置文件
- 3) 框架 框架都要用到很多配置文件

1.2.2 数据交换

- 1) Ajax
- 2) Webservice

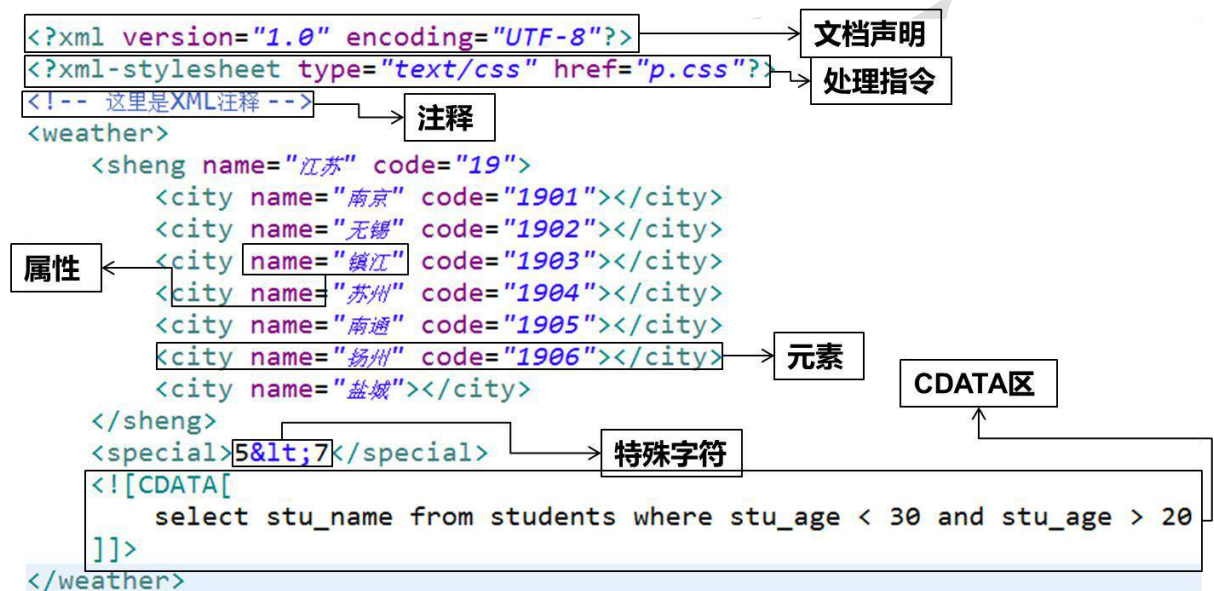


1.2.3 数据存储

1.2.4 保存关系型数据

2. xml 规范

2.1 xml 文档结构



2.2 文档声明

1) 在编写 XML 文档时，必须在文件的第一行书写文档声明。

最简单的声明语法：`<?xml version="1.0" ?>`

2) 用 encoding 属性说明读取文档所用的解码的字符集：

`<?xml version="1.0" encoding="utf-8" ?>`

这样就要求保存文件时，必须用 utf-8 编码保存。此时要求 XML 文档的作者确认当前编辑器保存文档的编码方式。

eclipse 会自动按照解码字符集进行编码保存

记事本需要另存为指定的字符集

2.3 语法规则

- (1) 第一行必须为 xml 声明，顶格写
- (2) 只能有一个根标签
- (3) 标签必须正确结束，并不能交叉嵌套
- (4) 严格区分大小，标签不能以数字开头
- (5) 属性必须有值，且必须加双引号

2.4 xml 转义字符

特殊字符	替代符号
<	<
>	>
&	&
"	"
'	'

注意：XML 实体中不允许出现 "&", "<", ">" 等特殊字符, 否则 XML 语法检查时将出错, 如果编写的 XML 文件必须包含这些字符, 则必须分别写成 "&", "<", ">" 再写入文件中。

2.5 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[...]]>

2.6 注释

xml 文件中的注释采用：“<!--注释-->” 格式。

注意：XML 声明之前不能有注释，注释不能嵌套

3. xml 解析

3.1 解析方式

1) dom: (Document Object Model, 即文档对象模型) 是 W3C 组织推荐的处理 XML 的一种方式。

它下面有两个分支：jDom 与 dom4j

它们可都可以对 xml 文件进行增删改查的操作

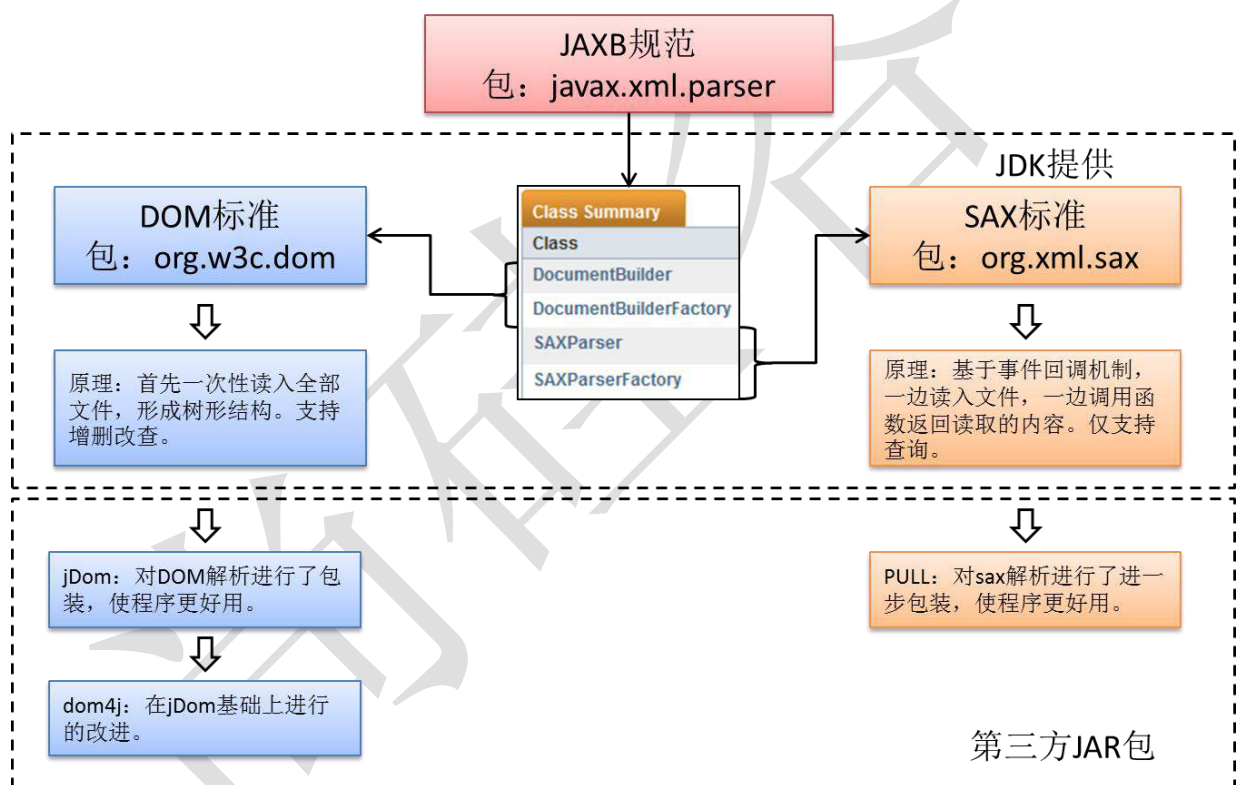
2) sax: (Simple API for XML) 不是官方标准，但它是 XML 社区事实上的标准，几乎所有的 XML 解析器都支持它。

只能进行解析（查询）

3) pull: Pull 解析和 Sax 解析很相似，都是轻量级的解析，它是一个第三方开源的 Java 项目，但在 Android 的内核中已经嵌入了 Pull 。

只能进行解析（查询）

3.2 解析技术体系



4. dom4j 解析

4.1 简介

1) Dom4j 是一个简单、灵活的开放源代码的库。Dom4j 是由早期开发 JDOM 的人分离出来而后独立开发的。与 JDOM 不同的是，dom4j 使用接口和抽象基类，虽然 Dom4j 的 API 相对要复杂一些，但它提供了比 JDOM 更好的灵活性。

2) Dom4j 是一个非常优秀的 Java XML API，具有性能优异、功能强大和极易使用的特点。现在很多软件采用的 Dom4j，例如 Hibernate。使用 Dom4j 开发，需下载 dom4j 相应的 jar 文件。

4.2 使用

dom4j 解析 xml 文件主要有以下几个步骤：

- 1) 导包，导包，导包，重要的事情说三遍！dom4j-1.6.1.jar
- 2) **创建解析器**，获取要解析的 xml 的文档对象，即 document 对象。
- 3) **获取元素**，进行**操作**。可操作属性，操作文本，获取**元素**信息等。

代码示例：

```
//1、创建saxreader对象
SAXReader reader = new SAXReader();
//2、读取xml文件,获得document对象
Document document = reader.read(new FileInputStream("web.xml"));
//3、获取到根元素，从根元素开始查找，修改
Element element = document.getRootElement();
//4、遍历所有元素
Iterator iterator = element.elementIterator();
while(iterator.hasNext()){
    Element next = (Element) iterator.next();
    System.out.println(next.getName()+"-->"+next.getText());
}

//5、获取element下的第一个子元素
/**
 * 查询都是使用根元素的element往下开始查询,一层一层的查
 * 各种元素查询的方法:
 * //获取某个元素的指定名称的第一个子节点
 *     Element element = element.element("书名");
 * //获取某个元素的指定名称的所有子元素的集合
 *     List list = element.elements("书名");
 * //添加一个指定名称的子元素
 *     Element childEle = parentEle.addElement("书名");
 * //删除某个元素指定的子元素
 *     parentEle.remove(childEle);
 */
Element element2 = element.element("servlet");
Element element3 = element2.element("servlet-name");
```

```
System.out.println(element3.getStringValue());
```

5. xpath

5.1 简介

XPath 是在 XML 文档中查找信息的语言

XPath 是通过元素和属性进行查找

XPath 简化了 Dom4j 查找节点的过程

使用 XPath 必须导入 `jaxen-1.1-beta-6.jar`

否则出现

`NoClassDefFoundError: org/jaxen/JaxenException`

5.2 语法

XPath 语法示例

<code>/students/student</code>	从根元素开始逐层找，以“/”开头
<code>//name</code>	直接获取所有 name 元素对象，以“//”开头
<code>//student/*</code>	获取所有 student 元素的所有子元素对象
<code>//student[1]</code> 或 <code>//student[last()]</code>	获取所有 student 元素的第一个或最后一个
<code>//student[@id]</code>	获取所有带 id 属性的 student 元素对象
<code>//student[@id= '002']</code>	获取 id 等于 002 的 student 元素对象

5.3 使用

1) 导包，导包，导包。谢谢。 `jaxen-1.1-beta-6.jar`

2) dom4j 怎么做就怎么做。只是在查找元素的时候可以使用 xpath 了

3) 获取所有符合条件的节点

- `document.selectNodes(String xpathExpression)` 返回 List 集合

获取符合条件的单个节点

- `document.selectSingleNode(String xpathExpression)` 返回一个 Node 对象。

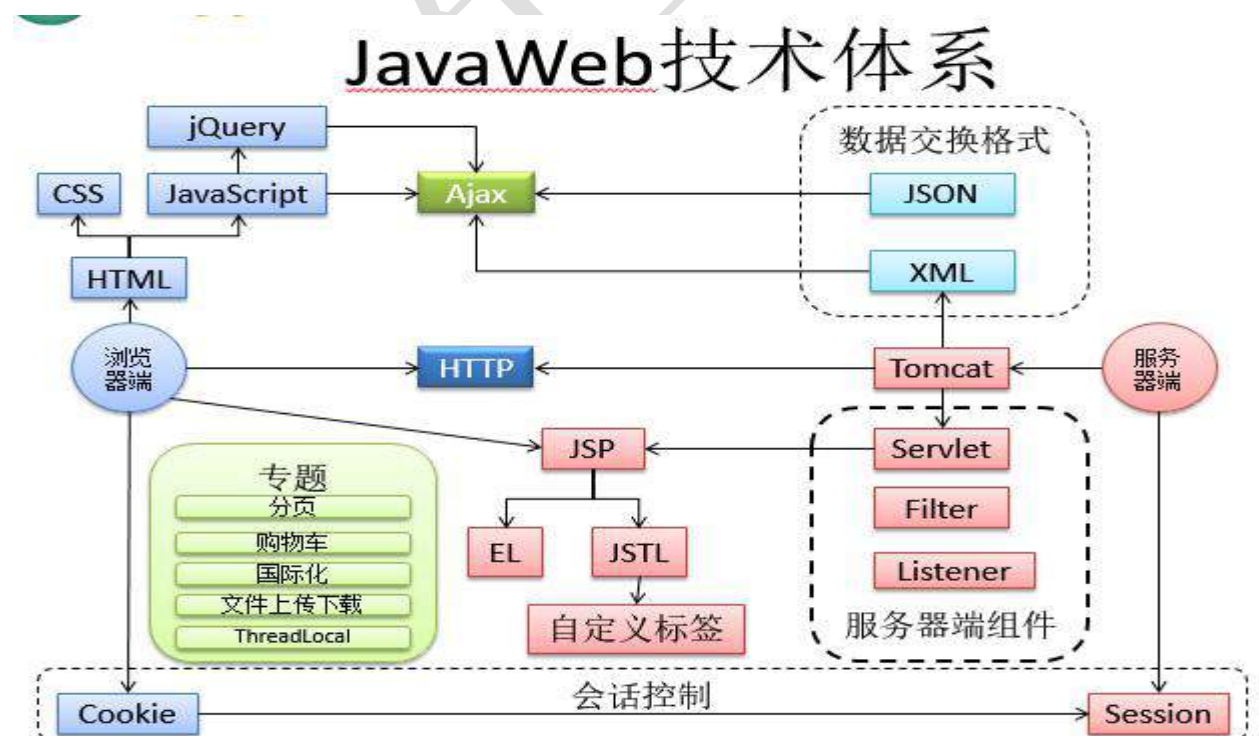
代码：

```
public void testXPath() throws Exception{  
  
    SAXReader reader = new SAXReader();  
    Document document = reader.read("stu.xml");
```

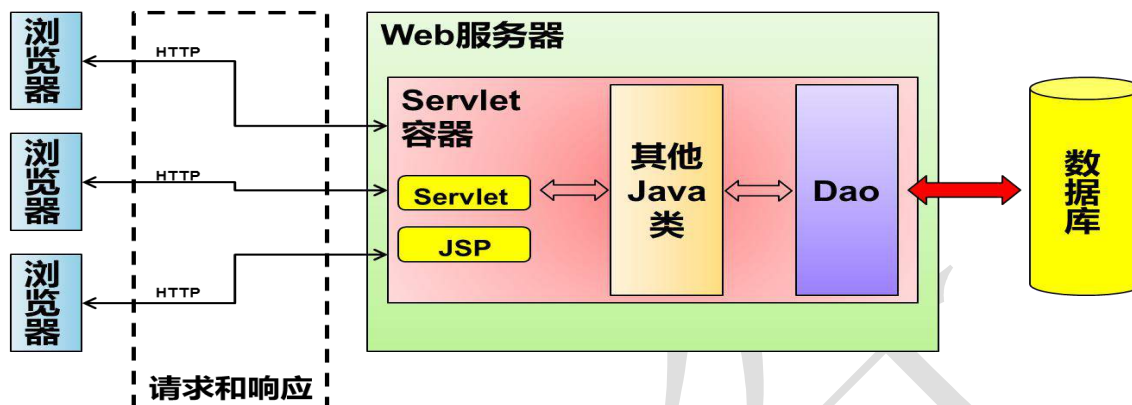
```
//我们使用selectNodes 或 selectSingleNode
//查询id为2的学生
Element stuEle = (Element) document.selectSingleNode("/students/student[@id='2']");
//获取学生的信息
String idStr = stuEle.attributeValue("id");
String name = stuEle.elementText("name");
String ageStr = stuEle.elementText("age");
String gender = stuEle.elementText("gender");
String address = stuEle.elementText("address");
//封装为一个学生对象
Student stu = new Student(Integer.parseInt(idStr), name, Integer.parseInt(ageStr),
gender, address);
System.out.println(stu);
}
```

第 6 章 Web 环境搭建

1. web 知识体系



2. web 应用运行原理



3. servlet 容器

Servlet 容器为 JavaWeb 应用提供运行时环境，它负责管理 Servlet 和 JSP(JSP 本质上是一个 Servlet)的生命周期，以及管理它们的共享数据。

Servlet 容器也称为 JavaWeb 应用容器，或者 Servlet/JSP 容器。

目前最流行的 Servlet 容器软件括：Tomcat（Apache）、WebLogic（Oracle）、WebSphere（IBM）等。

4. Tomcat

4.1 简介

Tomcat 是一个免费的开放源代码的 Servlet 容器，它是 Apache 软件基金会的一个顶级项目，由 Apache，Sun 和其他一些公司及个人共同开发而成。由于有了 Sun 的参与与支持，最新的 Servlet 和 JSP 规范总是能在 Tomcat 中的到体现。



4.2 配置

要使用 tomcat 必须正确配置。配置 tomcat 的过程主要为以下几步：

- 1) 解压 Tomcat 安装包 apache-tomcat-6.0.39-windows-x64.zip 到一个非中文目录
- 2) 检查当前系统环境中是否已经配置了 JAVA_HOME 环境变量



- 3) 进入 Tomcat 解压后根目录，运行 bin 目录下的 startup.bat 启动 tomcat 服务器。

4.3 配置服务器端口

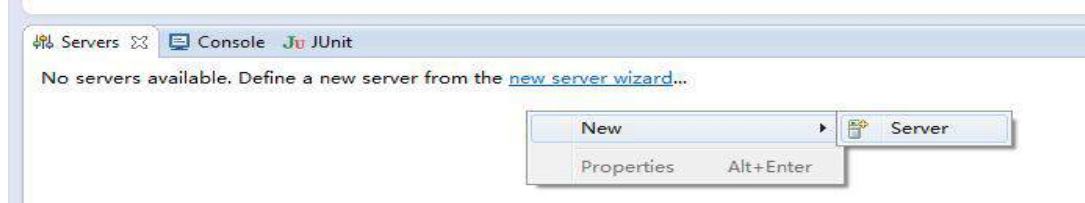
tomcat 的默认端口号是 8080；浏览器访问的时候使用 localhost:8080，但是 8080 可能会被其他程序占用，因此可能需要更换端口号。步骤如下：

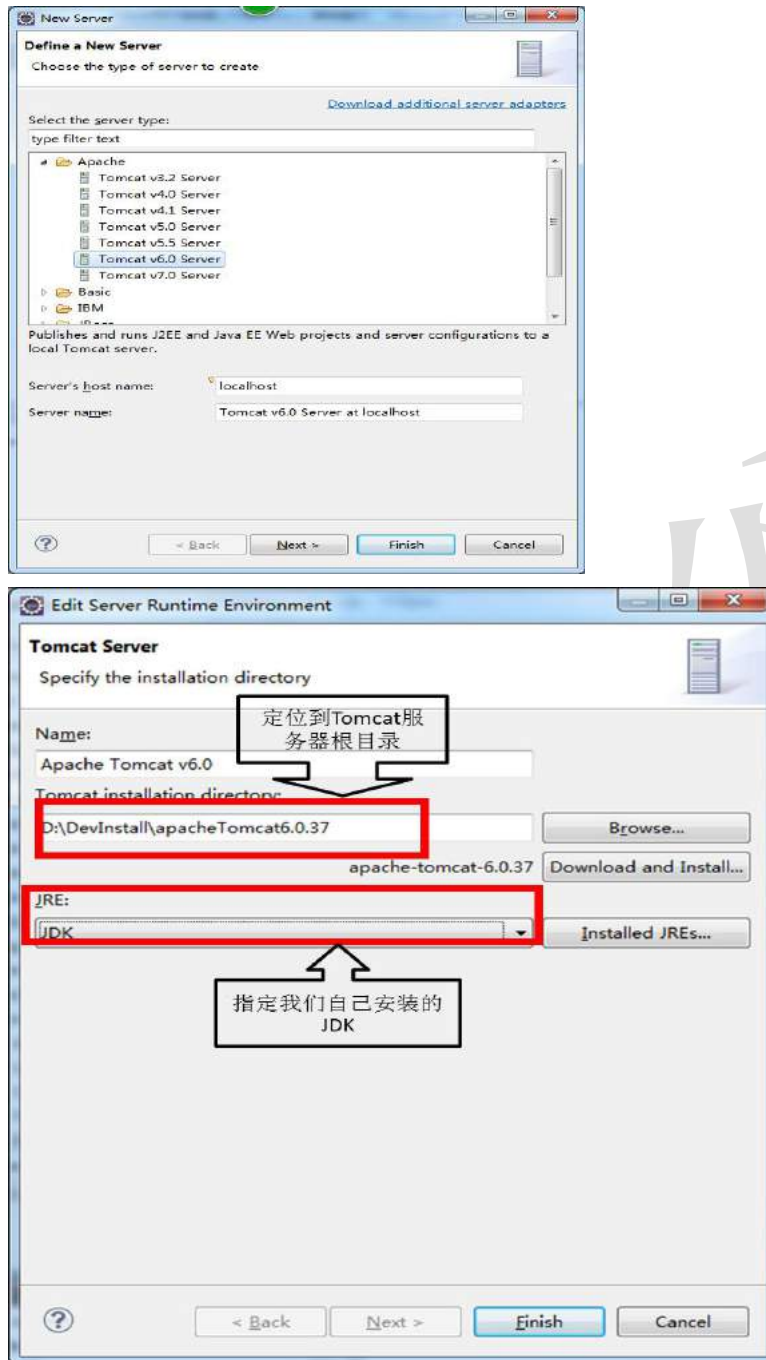
- 1) 进入 Tomcat 解压后的根目录，到 conf 目录下找到 server.xml 文件，使用文本编辑器打开
- 2) 找到 Connector 标签，将 port="8080"改为 port="8989"或其他值
- 3) 注意：Tomcat 服务器访问方式是 http://localhost:端口号/

4.4 在 eclipse 里面使用 tomcat

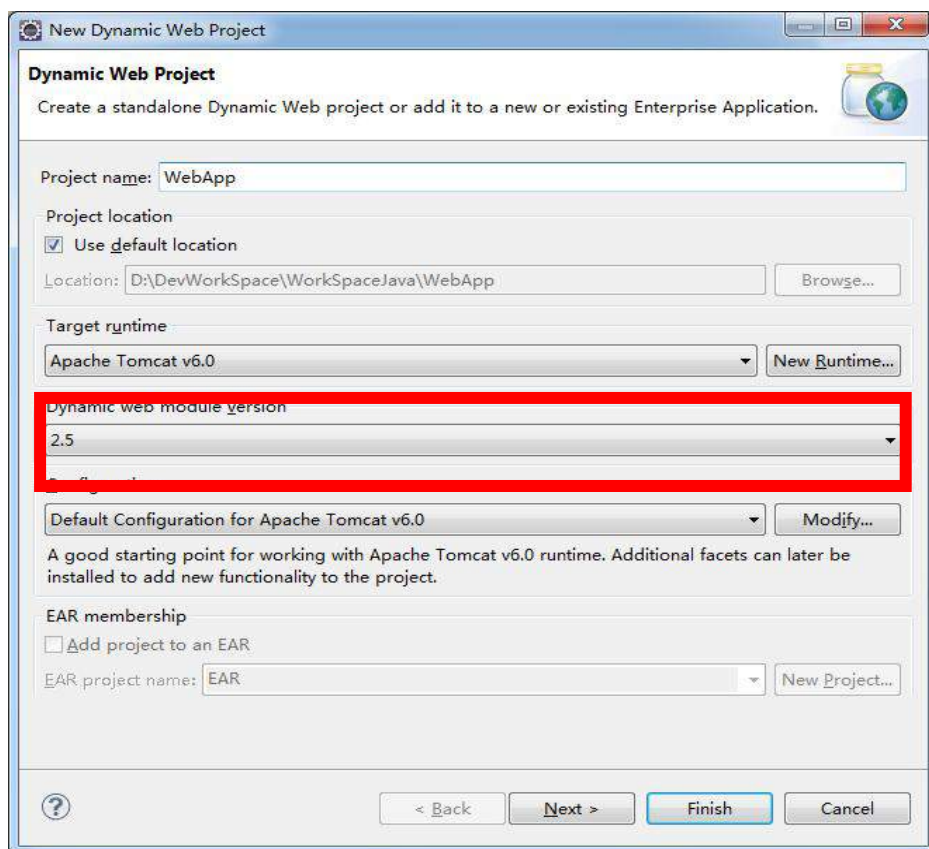
我们 web 项目的开发也是使用 eclipse 进行的，配置方法如下：

- 1) 创建 Tomcat 服务器在 eclipse 中的镜像

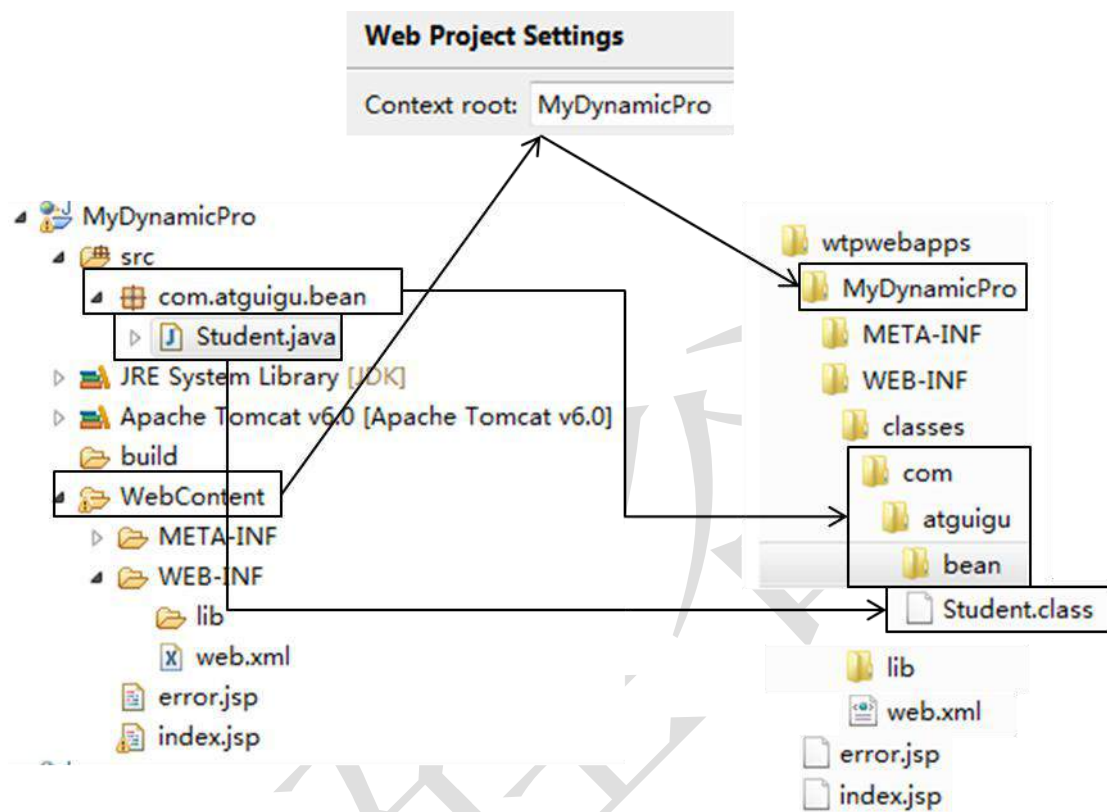




2) 在 eclipse 里创建 web 工程



4.5 eclipse 中 web 工程部署前后对比



第 7 章 HTTP 协议

1. 简介

1) HTTP 超文本传输协议 (HTTP-Hypertext transfer protocol)，是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。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。它是一种详细规定了浏览器和万维网服务器之间互相通信的规则，通过因特网传送万维网文档的数据传送协议。

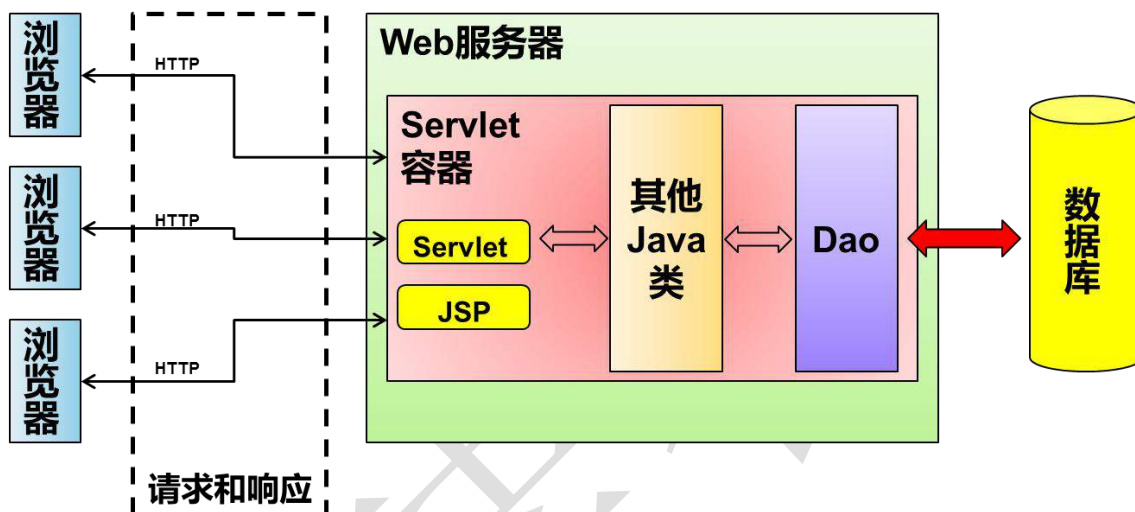
2) HTTP 协议的发展历程

超文本传输协议的前身是世外桃源(Xanadu)项目，超文本的概念是泰德·纳尔森(Ted Nelson)在 1960 年代提出的。进入哈佛大学后，纳尔森一直致力于超文本协议和该项目的研究，但他从未公开发表过资料。1989 年，蒂姆·伯纳斯·李(Tim Berners Lee)在 CERN(欧洲原子核研究委员会 = European Organization for Nuclear Research)担任软件咨询师的时候，开发了一套程序，奠定了万维网(WWW = World Wide Web)的基础。1990 年 12 月，超文本在 CERN 首次上线。1991 年夏天，继 Telnet 等协议之后，超文本转移协议成为互联网诸多协议的一分子。

当时，Telnet 协议解决了一台计算机和另外一台计算机之间一对一的控制型通信的要求。邮

件协议解决了一个发件人向少量人员发送信息的通信要求。文件传输协议解决一台计算机从另外一台计算机批量获取文件的通信要求,但是它不具备一边获取文件一边显示文件或对文件进行某种处理的功能。新闻传输协议解决了一对多新闻广播的通信要求。而超文本要解决的通信要求是:在一台计算机上获取并显示存放在多台计算机里的文本、数据、图片和其他类型的文件;它包含两大部分:超文本转移协议和超文本标记语言(HTML)。HTTP、HTML 以及浏览器的诞生给互联网的普及带来了飞跃。

3) web 运行原理

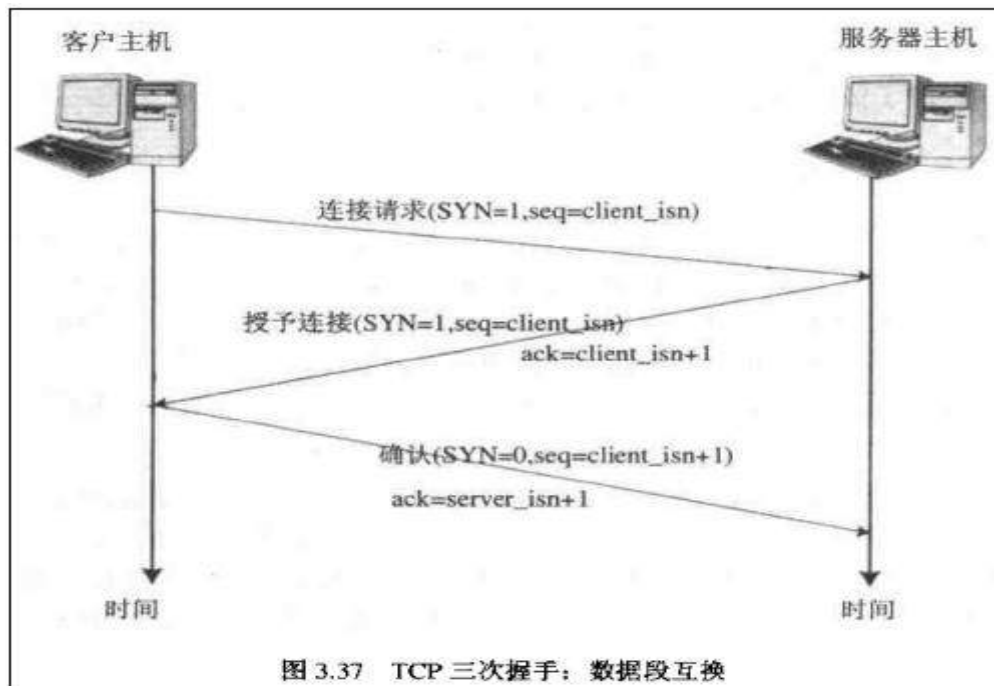


我们浏览器和服务要沟通，都是类似发信件的方式。很多的“信件”在网络之间传输。这些信件必须按照一定的格式进行书写，才能被彼此之间识别，客户端服务器本来就是两个不同的东西。

这个国家的 每个人都应学习编程，因为它教你如何思考。——乔布斯

4) tcp 三次握手:

为什么要有 tcp 的三次握手，因为我们浏览器和服务要通信，必须首先建立连接，在这个连接上进行数据发送工作。所以有三次握手。



就像两个陌生人见面要说话。

甲：你是那个 xxx 吗？

乙：哦，对，我是。你就是那个 xxx 吧。

甲：恩，是的。

（握手，开聊！）

2. 请求报文

HTTP 请求：客户端连上服务器后，向服务器发出获取某个 Web 资源的消息，称之为客户端向服务器发送了一个 HTTP 请求。一个完整的 HTTP 请求包括如下内容：

1) 请求行

URL 地址中如果包含中文，浏览器会自动对中文字符进行编码之后再发送

2) 若干消息头(请求头)

3) 实体内容(请求体) 有可能没有

```
POST /books/java.html HTTP/1.1
Accept: */*
Accept-Language: en-us
Connection: Keep-Alive
Host: localhost
Referer: http://localhost/links.jsp
User-Agent: Mozilla/4.0
Accept-Encoding: gzip, deflate
```

← 请求行

请求行用于描述客户端的请求方式、请求的资源名称，以及使用的HTTP协议版本号

← 多个消息头

消息头用于描述客户端请求哪台主机，以及客户端的一些环境信息等

← 一个空行

← 实体内容

```
name=tom&password=123
```

3. 响应报文

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。

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
- (2) 响应消息头：响应头
- (3) 实体内容：响应体

• 举例：

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Thu, 13 Jul 2000 05:46:53 GMT
Content-Length: 2291
Content-Type: text/html
Cache-control: private
```

```
<HTML>
<BODY>
.....
```

← 状态行

← 多个消息头

← 一个空行

← 实体内容

状态行用于描述服务器对请求的处理结果。

消息头用于描述服务器的基本信息，以及数据的描述，服务器通过这些数据的描述信息，可以通知客户端如何处理等一会儿它回送的数据。

代表服务器向客户端回送的数据

2) 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。

- (1) 响应状态码以 2 开头的通常表示成功。
- (2) 响应状态码以 3 开头的通常表示转移。
- (3) 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。
- (4) 响应状态码以 5 开头的通常表示服务器端程序运行出错。

3) 响应消息头：简称响应头

服务器发送给浏览器的数据，为了告诉浏览器一些情况。

代码	说明
Location: /day05/index.jsp	告诉浏览器重新定向到指定的路径
Server:apache tomcat	使用的什么 web 服务器
Content-Encoding: gzip	告诉浏览器我传给你的数据用的压缩方式
Content-Length: 80	响应体的字节数
Content-Language: zh-cn	响应体数据的语言
content-type: text/html; charset=GB2312	响应体内容的类型

Last-Modified: Tue, 11 Jul 2000 18:23:51 GMT	资源最后被修改的时间
Refresh: 1	定时刷新
Content-Disposition: attachment; filename=aaa.zip	文件下载
Set-Cookie:SS=Q0=5Lb_nQ; path=/search	将 cookie 数据回送给 ie
Expires: -1	告诉浏览器不要缓存起来
Cache-Control: no-cache	当 HTTP1.1 服务器指定 CacheControl = no-cache 时，浏览器就不会缓存该网页
Connection: close/Keep-Alive	是否保持连接
Date: Tue, 11 Jul 2000 18:23:51 GMT	响应的时间

(1) 响应体：服务器给出的响应结果的主体，通常是用来在页面上显示的。

(2) HTTP 内容类型：当前响应体的数据类型。

浏览器和服务器之间传输的数据类型并非都是文本类型，还包括图片、视频、音频等多媒体类型。这些多媒体类型是使用 MIME 类型定义的。MIME 的英文全称是"Multipurpose Internet Mail Extensions" 多功能 Internet 邮件扩充服务。MIME 类型的格式是“大类型/小类型”，并与某一种文件的扩展名相对应。

(3) 常见的 MIME 类型

文件	MIME 类型
超文本标记语言文本	.html,.html text/html
普通文本	.txt text/plain
RTF 文本	.rtf application/rtf
GIF 图形	.gif image/gif
JPEG 图形	.jpeg,.jpg image/jpeg
au 声音文件	.au audio/basic
MIDI 音乐文件	mid,.midi audio/midi,audio/x-midi
RealAudio 音乐文件	.ra, .ram audio/x-pn-realaudio
MPEG 文件	.mpg,.mpeg video/mpeg
AVI 文件	.avi video/x-msvideo
GZIP 文件	.gz application/x-gzip
TAR 文件	.tar application/x-tar

4. get、post 请求

4.1 请求方式

HTTP 中定义了 7 种请求方式：POST、GET、HEAD、OPTIONS、DELETE、TRACE、PUT。其中最常用的是 GET 和 POST

4.1.1 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 请求可以向服务器发送请求参数，在 URL 地址后面加上?，请求参数名和值用=连接，多个参数之间使用&分隔。例如：GET /mail/1.html?name=abc&password=xyz HTTP/1.1
- 4) 需要注意的是：GET 方式所能够携带的数据是由限制的，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。

4.1.2 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。

4.2 请求体

- 1) GET 请求没有请求体
- 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) get 和 post 对比

	GET 请求	POST 请求
含义	获取数据	发送数据
发送请求参数的方式	附着在 URL 地址后面	放在请求体中
传送数据大小的限制	有限制，而且能传输的数据容量很小	没有数据大小的限制
数据安全	如果没有特殊处理，会将数据明文显示在浏览器地址栏	不会将数据显示在浏览器地址栏
产生方式	点超链接、表单 method=get、直接在浏览器地址栏输入地址访问	表单 method=post 在 Ajax 应用中指定请求方式为 POST

第 8 章 Servlet

1. Servlet 简介

- 1) 从广义上来讲，Servlet 规范是 Sun 公司制定的一套技术标准，包含与 Web 应用相关的一系列接口，是 Web 应用实现方式的宏观解决方案。而具体的 Servlet 容器负责提供标准的实现。
- 2) 从狭义上来讲，Servlet 指的是 `javax.servlet.Servlet` 接口及其子接口，也可以指实现了 Servlet 接口的实现类。
- 3) Servlet 作为服务器端的一个组件，它的本意是“服务器端的小程序”。Servlet 的实例对象由 Servlet 容器负责创建；Servlet 的方法由容器在特定情况下调用；Servlet 容器会在 Web 应用卸载时销毁 Servlet 对象的实例。

2. 第一个 Servlet

2.1 编写 Servlet

- 1) 创建自己的类 `HelloServlet`，实现 Servlet 接口，并编写 `service` 方法

```
public class HelloServlet implements Servlet{
    @Override
    public void init(ServletConfig config) throws ServletException {
    }
    @Override
    public ServletConfig getServletConfig() {
        return null;
    }
    @Override
    public void service(ServletRequest req, ServletResponse res)
        throws ServletException, IOException {
        System.out.println("我是不是被执行了? ");
    }
    @Override
    public String getServletInfo() {
        return null;
    }
    @Override
```



```
public void destroy() {  
    }  
}
```

2) 在 web.xml 中配置 servlet 信息

```
<servlet>  
    <servlet-name>helloServlet</servlet-name>  
    <servlet-class>com.atguigu.listener.HelloServlet</servlet-class>  
</servlet>  
<servlet-mapping>  
    <servlet-name>helloServlet</servlet-name>  
    <url-pattern>/helloworld</url-pattern>  
</servlet-mapping>
```

配置详解:

<servlet></servlet>: 这个之间配置的是Servlet的类信息

<servlet-name>: 这是 Servlet 的别名, 一个名字对应一个 Servlet。相当于变量名

<servlet-class>: Servlet 的全类名, 服务器会根据全类名找到这个 Servlet

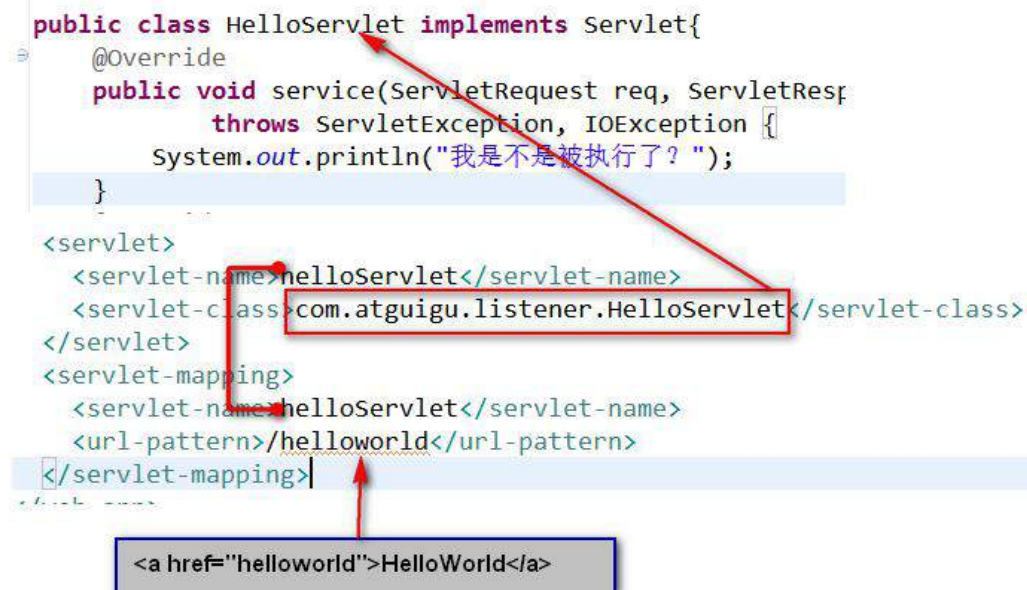
<servlet-mapping></servlet-mapping>: 这个之间配置的是 Servlet 的请求映射信息

<servlet-name>: Servlet的别名, 说明这个Servlet将会响应下面url-pattern的请求

<url-pattern>: Servlet响应的请求路径。如果访问这个路径, 这个Servlet就会响应。这个 url-pattern可以配置多个, 这时表示的就是访问这些url都会触发这个Servlet进行响应

3) 运行浏览器, 访问刚才配置的 url 路径, Servlet 的 service 方法就会被调用

2.2 运行原理:



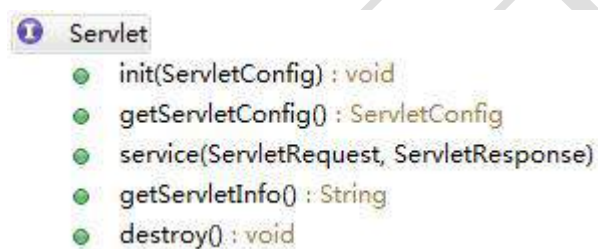
```
public class HelloServlet implements Servlet{  
    @Override  
    public void service(ServletRequest req, ServletRes  
        throws ServletException, IOException {  
        System.out.println("我是不是被执行了?");  
    }  
}  
  
<servlet>  
    <servlet-name>helloServlet</servlet-name>  
    <servlet-class>com.atguigu.listener.HelloServlet</servlet-class>  
</servlet>  
<servlet-mapping>  
    <servlet-name>helloServlet</servlet-name>  
    <url-pattern>/helloworld</url-pattern>  
</servlet-mapping>  
  
<a href="/helloworld">HelloWorld</a>
```

3. Servlet 技术体系

3.1 Servlet 的层次关系

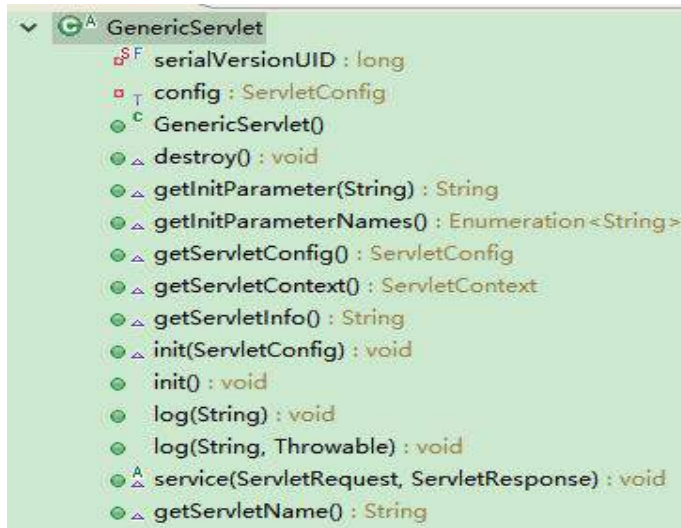


3.3 Servlet 接口



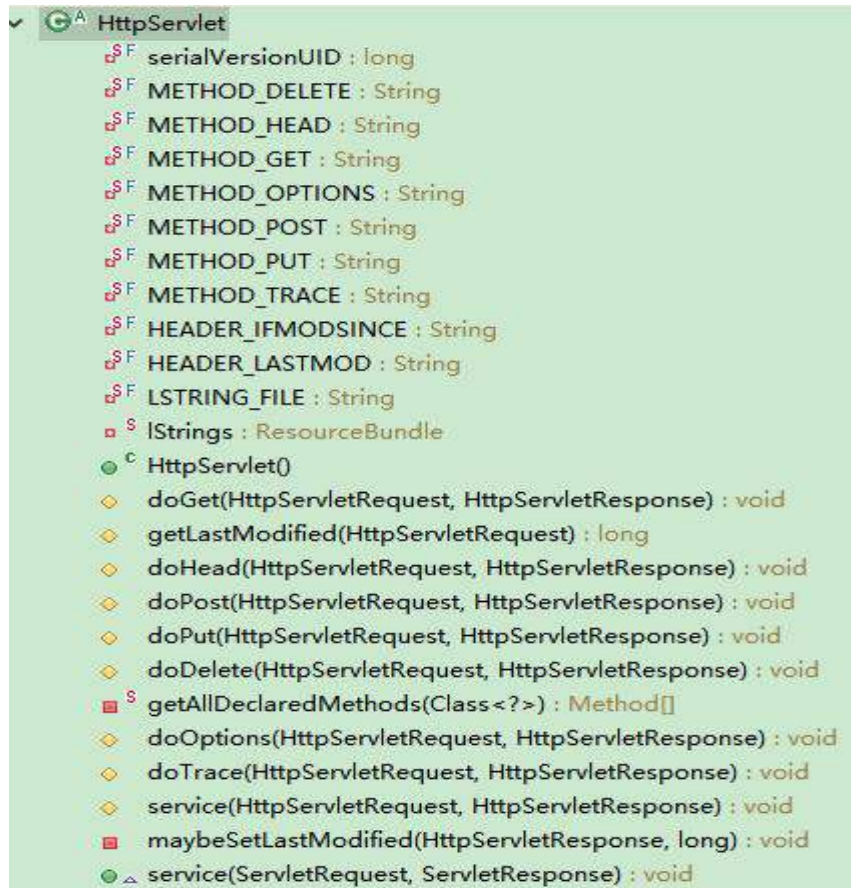
- 1) Servlet 接口是 Servlet 的规范。
- 2) init(ServletConfig config): Servlet 初始化函数。初始化时 ServletConfig 会被传入
- 3) ServletConfig getServletConfig(): 获取 ServletConfig 对象
- 4) service(ServletRequest req, ServletResponse res): 收到请求后的执行方法
- 5) String getServletInfo(): 返回此 Servlet 的描述信息
- 6) void destroy(): Servlet 的销毁方法

3.4 GenericServlet 抽象类



- 1) Servlet, ServletConfig 的实现类。里面可以实现 Servlet 的功能，可以获取 ServletConfig 的信息
- 2) String getInitParameter(String name): 获取 Servlet 初始化参数值，初始化参数在 web.xml 的 Servlet 中配置，`<init-param><param-name>user</param-name><param-value>abc</param-value></init-param>`
- 3) Enumeration<String> getInitParameterNames(): 获取 Servlet 初始化的所有参数 name 值
- 4) ServletConfig getServletConfig(): 获取 ServletConfig 对象
- 5) ServletContext getServletContext(): 获取 ServletContext 对象
- 6) String getServletInfo(): 获取 Servlet 的描述信息
- 7) void init(ServletConfig config): 从 Servlet 实现的 init 方法，这里面调用了自己的 init 方法
- 8) void init(): 自己的 init 方法
- 9) abstract void service: 抽象方法，继承此类的子类必须自己实现此方法。每次请求都会调用此方法
- 10) String getServletName(): 获取 Servlet 的名字

3.5 HttpServlet 抽象类



1) HttpServlet 继承了 GenericServlet，并实现了 service 方法。在 service 方法中，将 ServletRequest 和 ServletResponse 转换为了 HttpServletRequest 和 HttpServletResponse，用来专门处理我们的 Http 请求。

2) `service(ServletRequest, ServletResponse) : void` 方法在完成对请求和响应的强转之后调用了 `service(HttpServletRequest, HttpServletResponse) : void` 方法，在被调用的方法中对请求类型进行了判断，各种请求调用自己相应的 doXXX 方法。而我们常用的就是 doGet() 和 doPost() 方法。

3) 在我们以后的使用中，都使用继承 HttpServlet 的方式，重写 doGet 和 doPost 方法即可。在浏览器发送请求的时候，如果是 get 请求，将会调用 doGet() 方法，如果是 post 请求，将会调用 doPost() 方法。

4) 继承 HttpServlet 的新的 Servlet 写法如下(web.xml 配置与之前相同):

```
public class FirstServlet extends HttpServlet {  
    private static final long serialVersionUID = 1L;  
    public FirstServlet() {  
        super();  
    }  
}
```

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    System.out.println("doGet().....");
}
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    System.out.println("doPost().....");
}
}
```

4. Servlet 生命周期

4.1 什么是生命周期:

应用程序中的对象不仅在空间上有层次结构的关系,在时间上也会因为处于程序运行过程中的不同阶段而表现出不同状态和不同行为——这就是对象的生命周期。
简单的叙述生命周期,就是对象在容器中从开始创建到销毁的过程。

4.2 Servlet 容器

Servlet 对象是 Servlet 容器创建的,生命周期方法都是由容器调用的。这一点和我们之前所编写的代码有很大不同。在今后的学习中我们会看到,越来越多的对象交给容器或框架来创建,越来越多的方法由容器或框架来调用,开发人员要尽可能多的将精力放在业务逻辑的实现上。

4.3 Servlet 生命周期

我们在编写 Servlet 的 HelloWorld 的时候,我们发现服务器在启动后,访问配置的 url 的时候好像调用了 Servlet 的几个方法。到底 Servlet 在容器中是如何创建,使用,消亡的。我们探究一下:

主要步骤:

4.3.1 Servlet 对象的创建

默认情况下,Servlet 容器第一次收到 HTTP 请求时创建对应 Servlet 对象。容器之所以能做到这一点是由于我们在注册 Servlet 时提供了全类名,容器使用反射技术创建了 Servlet 的对象。

4.3.2 Servlet 对象初始化

- 1) Servlet 容器创建 Servlet 对象之后，会调用 `init(ServletConfig config)` 方法，对其进行初始化。在 `javax.servlet.Servlet` 接口中，`public void init(ServletConfig config)` 方法要求容器将 `ServletConfig` 的实例对象传入，这也是我们获取 `ServletConfig` 的实例对象的根本方法。
- 2) 为了简化开发，`GenericServlet` 抽象类中实现了 `init(ServletConfig config)` 方法，将 `init(ServletConfig config)` 方法获取到的 `ServletConfig` 对象赋值给了成员变量 `ServletConfig config`，目的是使其它方法可以共享这个对象。这时有一个问题：如果子类重写了这个 `init(ServletConfig config)` 方法，有可能会造成成员变量 `config` 对象赋值失败。所以 `GenericServlet` 抽象类另外提供了一个无参的 `public void init()` 方法，并在 `init(ServletConfig config)` 方法中调用，作为子类进行初始化操作时重写使用。而这个无参的 `init()` 方法之所以没有设计成抽象方法，是为了避免子类继承时强制实现这个方法带来的麻烦，使用者可以根据需要选择是否要覆盖这个方法。

4.3.3 处理请求

- 1) 在 `javax.servlet.Servlet` 接口中，定义了 `service(ServletRequest req, ServletResponse res)` 方法处理 HTTP 请求，同时要求容器将 `ServletRequest` 对象和 `ServletResponse` 对象传入。
- 2) 在 `HttpServlet` 抽象类中，`service(ServletRequest req, ServletResponse res)` 方法将 `ServletRequest` 对象和 `ServletResponse` 对象强转为了 `HttpServletRequest`、`HttpServletResponse` 子类对象，这样更适合于 HTTP 请求的处理，所以在 `doGet()` 和 `doPost()` 方法中使用的就是 `HttpServletRequest`、`HttpServletResponse` 的实现类对象了。

4.3.4 Servlet 对象销毁

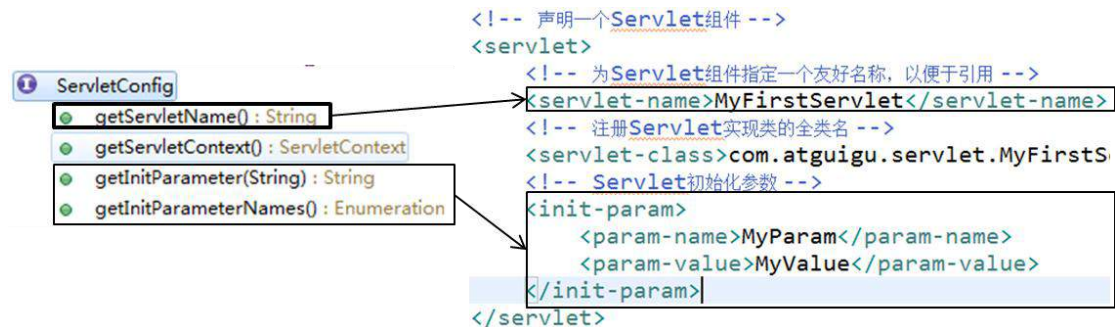
Web 应用卸载或服务器停止执行时会销毁 Servlet 对象，而销毁之前为了执行一些诸如释放缓存、关闭连接、保存数据等操作，所以设计了 `public void destroy()` 方法。

步骤总结：

- 1) Servlet 对象创建：接收到浏览器请求后，才创建对象
- 2) Servlet 初始化
创建对象之后，会调用 `init()` 方法
作用：是在 Servlet 对象创建后，执行一些初始化操作
例如，读取一些资源文件，或建立某种连接
`init()` 方法只在创建对象时执行一次，以后再接到请求时，就不执行了
- 3) Servlet 处理客户端请求
接收到请求之后调用 `service()` 方法
在每次接到请求后都会执行
- 4) Servlet 面临对象的销毁
当前 Web 应用卸载之前调用 `destroy()` 方法

在应用卸载之前，可能需要释放一些资源，关闭某些连接

5. ServletConfig 接口



5.1 ServletConfig 接口简介

封装了 Servlet 配置信息，这一点从接口的名称上就能够看出来。但同时，代表当前 Web 应用的 ServletContext 对象也封装到了 ServletConfig 对象中，使 ServletConfig 对象成为了获取 ServletContext 对象的一座桥梁。

5.2 ServletConfig 对象的主要功能

- 1) 获取 Servlet 名称
- 2) 获取 Servlet 初始化参数
- 3) 获取 ServletContext 对象

5.3 Servlet 初始化参数

在 Servlet 注册信息中，指定的一个参数，有参数名和参数值。在 servlet 标签内加入如下配置信息

```

<!-- Servlet初始化参数 -->
<init-param>
  <!-- 初始化参数的参数名 -->
  <param-name>Status</param-name>
  <!-- 初始化参数的值 -->
  <param-value>open</param-value>
</init-param>
    
```

对 Servlet 运行时如果想设置一些开关选项，就可以以配置文件的形式进行设置，在改变选项状态时，不需要修改源代码、重新编译，只需修改配置文件即可。

```
//获取初始化参数
```



```
String paramValue = servletConfig.getInitParameter("Status");
```

5.4 获取 ServletConfig 对象

- 1) 在 GenericServlet 中，由容器传入 init(ServletConfig config)方法。
- 2) 在 GenericServlet 中提供了 public ServletConfig getServletConfig()方法用来获取 ServletConfig 对象。
- 3) 如果继承了 HttpServlet，则可以直接调用 getServletConfig()方法获取 ServletConfig 对象

6. ServletContext 接口



The screenshot shows the `ServletContext` interface with the following methods:

- `getContext(String) : ServletContext`
- `getContextPath() : String`
- `getMajorVersion() : int`
- `getMinorVersion() : int`
- `getMimeType(String) : String`
- `getResourcePaths(String) : Set`
- `getResource(String) : URL`
- `getResourceAsStream(String) : InputStream`
- `getRequestDispatcher(String) : RequestDispatcher`
- `getNamedDispatcher(String) : RequestDispatcher`
- `getServlet(String) : Servlet`
- `getServlets() : Enumeration`
- `getServletNames() : Enumeration`
- `log(String) : void`
- `log(Exception, String) : void`
- `log(String, Throwable) : void`
- `getRealPath(String) : String`
- `getServerInfo() : String`
- `getInitParameter(String) : String`
- `getInitParameterNames() : Enumeration`
- `getAttribute(String) : Object`
- `getAttributeNames() : Enumeration`
- `setAttribute(String, Object) : void`
- `removeAttribute(String) : void`
- `getServletContextName() : String`

On the right, a code snippet shows the XML configuration for a context-param:

```
<!-- Web应用初始化参数 -->
<context-param>
  <param-name>ParamName</param-name>
  <param-value>ParamValue</param-value>
</context-param>
```

- 1) web 容器在启动时，他会为每个 web 应用都创建一个对应的 `ServletContext` 对象。注意：一个 web 应用对应的是一个 `ServletContext` 对象。就行每个 web 应用就像是不同的餐厅，而 `Servlet` 是餐厅服务员，可以有很多，`ServletContext` 是这个餐厅的经理只能有一个。
- 2) 由于一个 web 应用程序的所有 `Servlet` 都共享的是同一个 `ServletContext` 对象，所以 `ServletContext` 对象也被称为 `application` 对象（web 应用程序对象）
- 3) 在应用程序中能够获取运行环境或者容器信息的对象通常称之为“上下文对象”。
- 4) `ServletContext` 的主要功能

a) 获取虚拟路径所映射的本地路径

虚拟路径：浏览器访问 web 应用中资源时所使用的路径

本地路径：资源在文件系统中的实际保存路径

b) application 域范围的属性

c) 获取 web 应用程序的初始化参数

设置 Web 应用初始化参数的方式是在 web.xml 的根标签下加入如下代码

```
<!-- Web应用初始化参数 -->
<context-param>
    <param-name>ParamName</param-name>
    <param-value>ParamValue</param-value>
</context-param>
```

获取 Web 应用初始化参数

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
    //1. 获取ServletContext对象
    ServletContext context = this.getServletContext();
    //2. 获取Web应用初始化参数
    String paramValue = context.getInitParameter("ParamName");
    System.out.println("paramValue="+paramValue);
}
```

7. HttpServletRequest 接口

该接口是 `ServletRequest` 接口的子接口，封装了 HTTP 请求的相关信息，由 Servlet 容器创建其实现类对象并传入 `service(ServletRequest req, ServletResponse res)` 方法中。我们请求的详细信息都可以通过 `HttpServletRequest` 接口的实现类对象获取。这个实现类对象一般都是容器创建的，我们不需要管理。

`HttpServletRequest` 主要功能

7.1 获取请求参数

1) 什么是请求参数？

请求参数就是浏览器向服务器提交的数据

2) 浏览器向服务器如何发送数据

a) 附在 url 后面，如：<http://localhost:8989/MyServlet/MyHttpServlet?userId=20>

b) 通过表单提交

```
<form action="MyHttpServlet" method="post">
    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" />
```

```
<input type="submit" value="提交" />
</form>
```

3) 使用HttpServletRequest对象获取请求参数

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    //一个name对应一个值
    String userId = request.getParameter("userId");
    System.out.println("userId="+userId);
}

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    //一个name对应一组值
    String[] soccerTeams = request.getParameterValues("soccerTeam");
    for(int i = 0; i < soccerTeams.length; i++){
        System.out.println("team "+i+"="+soccerTeams[i]);
    }
}
```

7.2 在请求域中保存数据

数据保存在请求域中，可以转发到其他Servlet或者jsp页面，这些Servlet或者jsp页面就会从请求中再取出数据

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    //将数据保存到request对象的属性域中
    request.setAttribute("attrName", "attrValueInRequest");
    //两个Servlet要想共享request对象中的数据，必须是转发的关系
    request.getRequestDispatcher("/ReceiveServlet")
        .forward(request, response);
}

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    //从request属性域中获取数据
    Object attribute = request.getAttribute("attrName");
    System.out.println("attrValue="+attribute);
}
```


7.3、转发页面

7.4、获取请求头相关信息

8. HttpServletResponse 接口

HttpServletResponse 是 ServletResponse 接口的子接口，封装了 HTTP 响应的相关信息，由 Servlet 容器创建其实现类对象并传入 service(ServletRequest req, ServletResponse res)方法中。主要功能：

- 1) 使用 PrintWriter 对象向浏览器输出数据

```
//通过PrintWriter对象向浏览器端发送响应信息
PrintWriter writer = res.getWriter();
writer.write("Servlet response");
writer.close();
```

- 2) 实现请求重定向

9. 请求转发与重定向

请求转发和重定向是 web 应用页面跳转的主要手段，应用十分广泛，所以我们一定要搞清楚他们的区别。

9.1 请求转发：

- 1) 第一个 Servlet 接收到了浏览器端的请求，进行了一定的处理，然后没有立即对请求进行响应，而是将请求“交给下一个 Servlet”继续处理，下一个 Servlet 处理完成之后对浏览器进行了响应。在服务器内部将请求“交给”其它组件继续处理就是请求的转发。对浏览器来说，一共只发了一次请求，服务器内部进行的“转发”浏览器感觉不到，同时浏览器地址栏中的地址不会变成“下一个 Servlet”的虚拟路径。
- 2) 在转发的情况下，两个 Servlet 可以共享 Request 对象中保存的数据
- 3) 转发的情况下，可以访问 WEB-INF 下的资源
- 4) 当需要将后台获取的数据传送到 JSP 上显示的时候，就可以先将数据存放到 Request 对象中，再转发到 JSP 从属性域中获取。此时由于是“转发”，所以它们二者共享 Request 对象中的数据。

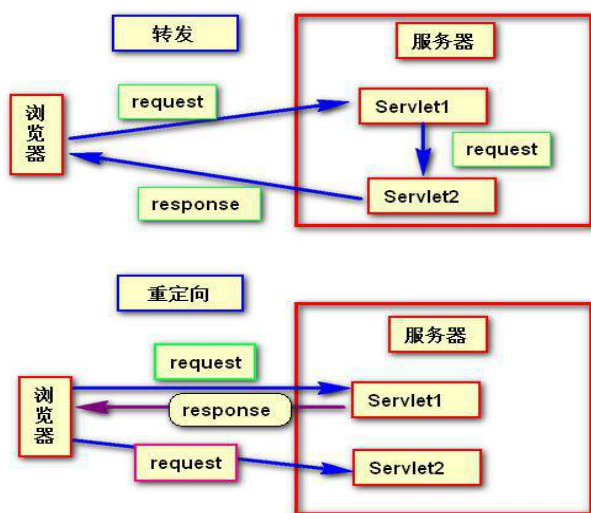
9.2 请求重定向:

- 1) 第一个 Servlet 接收到了浏览器端的请求, 进行了一定的处理, 然后给浏览器一个特殊的响应消息, 这个特殊的响应消息会通知浏览器去访问另外一个资源, 这个动作是服务器和浏览器自动完成的, 但是在浏览器地址栏里面能够看到地址的改变, 会变成下一个资源的地址。
- 2) 对浏览器来说, 一共发送两个请求, 所以用户是能够感知到变化的。
- 3) 在重定向的情况下, 不能共享 Request 对象中保存的数据。

9.3 转发与重定向的区别:

	转发	重定向
浏览器地址栏	不会变化	会变化
Request	同一个请求	两次请求
API	Request 对象	Response 对象
位置	服务器内部完成	浏览器完成
WEB-INF	可以访问	不能访问
共享请求域数据	可以共享	不可以共享
目标资源	必须是当前 Web 应用中的资源	不局限于当前 Web 应用

图解转发和重定向



10. 字符编码问题

10.1 编码简介

我们 web 程序在接收请求并处理过程中，如果不注意编码格式及解码格式，很容易导致中文乱码，引起这个问题的原因到底在哪里？如何解决？我们这个小节将会讨论此问题。

说到这个问题我们先来说一说字符集。

什么是字符集，就是各种字符的集合，包括汉字，英文，标点符号等等。各国都有不同的文字、符号。这些文字符号的集合就叫字符集。

现有的字符集 ASCII、GB2312、BIG5、GB18030、Unicode 等

这些字符集，集合了很多的字符，然而，字符要以二进制的形式存储在计算机中，我们就需要对其进行编码，将编码后的二进制存入。取出时我们就要对其解码，将二进制解码成我们之前的字符。这个时候我们就需要制定一套编码解码标准。否则就会导致出现混乱，也就是我们的乱码。

10.2 编码与解码

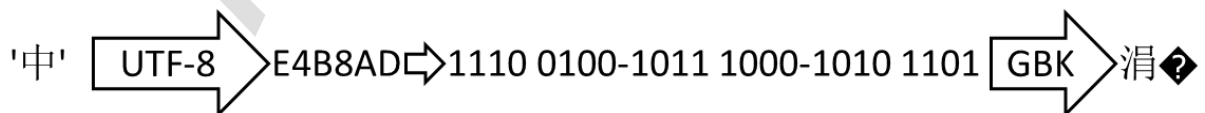
- ◆ 编码：将字符转换为二进制数

汉字	编码方式	编码	二进制
‘中’	GB2312	D6D0	1101 0110-1101 0000
	UTF-16	4E2D	0100 1110-0010 1101
	UTF-8	E4B8AD	1110 0100-1011 1000-1010 1101

- ◆ 解码：将二进制数转换为字符

1110 0100-1011 1000-1010 1101 → E4B8AD → ‘中’

- ◆ 乱码：一段文本，使用 A 字符集编码，使用 B 字符集解码，就会产生乱码。
所以解决乱码问题的根本方法就是统一编码和解码的字符集。

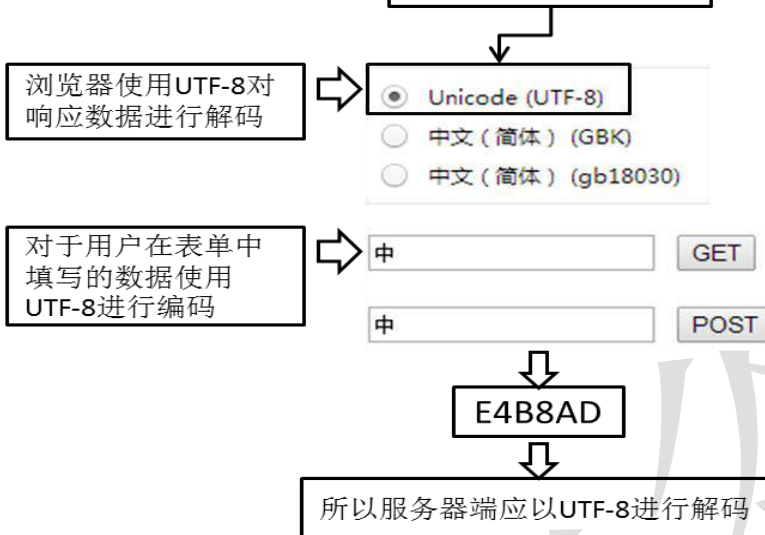


10.3 解决乱码：

解决方法就是统一字符编码。

HTML代码

```
<meta http-equiv="Content-Type"
content="text/html; charset=UTF-8">
```



乱码情况:

10.3.1 接收中文请求参数

1) POST 请求: post 请求提交了中文。在获取参数值之前, 设置请求的解码格式, 使其和页面保持一致。我们以后都使用 utf-8 编码, 它也称作万国码, 集合了基本所有的字符。

解决方法: request.setCharacterEncoding("utf-8");

2) GET 请求: GET 请求参数是在地址后面的。不能使用上述方法。我们需要修改 tomcat 的配置文件。需要在 Server.xml 文件修改 Connector 标签, 添加 URIEncoding="utf-8"属性。

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

10.3.2 向浏览器发送有中文字符的响应

向浏览器发送响应的时候, 要告诉浏览器, 我使用的字符集是哪个, 浏览器就会按照这种方式来解码。如何告诉浏览器响应内容的字符编码方案。很简单。

解决方法: response.setContentType("text/html;charset=utf-8");

或者使用这两句组合:

```
response.setCharacterEncoding("utf-8");
response.setHeader("Content-Type", "text/html;charset=utf-8");
```

我们一般使用 `response.setContentType("text/html;charset=utf-8");`

11. 路径问题

11.1 提出问题:

①创建 Web 应用 Path，目录结构如图所示



②在 a.html 中有超链接 `To b.html`

③如果先直接访问 a.html，再通过超链接转到 b.html 没有问题。

④如果先通过 TestServlet 转发到 a.html，则浏览器地址栏会变成：
`http://localhost:8989/Path/TestServlet`

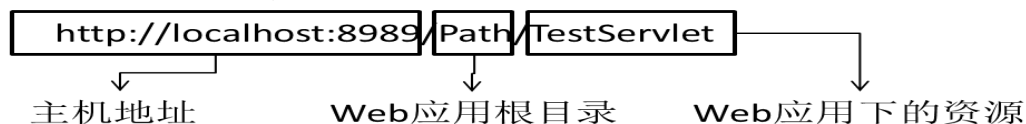
此时再点击超链接 `To b.html` 就会发生问题，找不到 b.html。

⑤原因是超链接 `To b.html` 使用的是相对路径，浏览器进行解析时，只能以当前浏览器地址栏里的路径为基准。例如，当前浏览器地址栏里的内容是：

`http://localhost:8989/Path/TestServlet`

那么经过浏览器解析后 b.html 的访问地址就成了：`http://localhost:8989/Path/TestServlet b.html` 这显然无法访问到 b.html。

11.2 完整的 url 构成



11.3 相对路径和绝对路径

11.3.1 相对路径:

虚拟路径如果不以“/”开始,就是相对路径,浏览器会以当前资源所在的虚拟路径为基准对相对路径进行解析,从而生成最终的访问路径。此时如果通过转发进入其他目录,再使用相对路径访问资源就会出错。

11.3.2 绝对路径:

虚拟路径以“/”开始,就是绝对路径。

1) 在服务器端:虚拟路径最开始的“/”表示当前 Web 应用的根目录。

例如:服务器端虚拟路径“/TestServlet”使用浏览器访问时,地址为

<http://localhost:8989/Path>

所以由服务器解析的路径,以“/”开头的都是以当前 web 路径为基准的。

由服务器解析的路径

web.xml 文件中 url-pattern 中配置的 URL,以“/”开头的

转发操作: `request.getRequestDispatcher("/xxx").forward(request,response);`

Jsp 动作标签: `<jsp:forward page="/xxx">`

这些最后的访问路径都是

<http://localhost:8989/Path/xxx>

2) 在浏览器端:虚拟路径最开始的“/”表示当前主机地址。

例如:链接地址“/ Path/ dir/ b.html”经过浏览器解析后为:

相当于 <http://localhost:8989/ Path/ dir/ b.html>

所以由浏览器解析的路径,以“/”开头的都是以当前的主机地址为基准的

由浏览器解析的路径

重定向操作: `response.sendRedirect("/xxx")`

所有 HTML 标签: ``; `<form action="/xxx"> ...`

这些最后的访问路径都是

<http://localhost:8989/xxx>

所以我们可以看出,如果是浏览器解析的路径,我们必须加上项目名称才可以正确的指向资源。<http://localhost:8989/Path/xxx>

11.3.3 jsp 页面中获取项目名

在由浏览器解析的地址中动态添加当前 web 应用根目录路径

1) JSP 表达式 `<%= request.getContextPath() %> = /LessonPractice040`

例: `<a href="<%= request.getContextPath() %>/target.jsp">Target Page`

2) EL 表达式 `${pageContext.request.contextPath } = /LessonPractice040`

例: `Target Page`

第 9 章 JSP

1. 背景

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

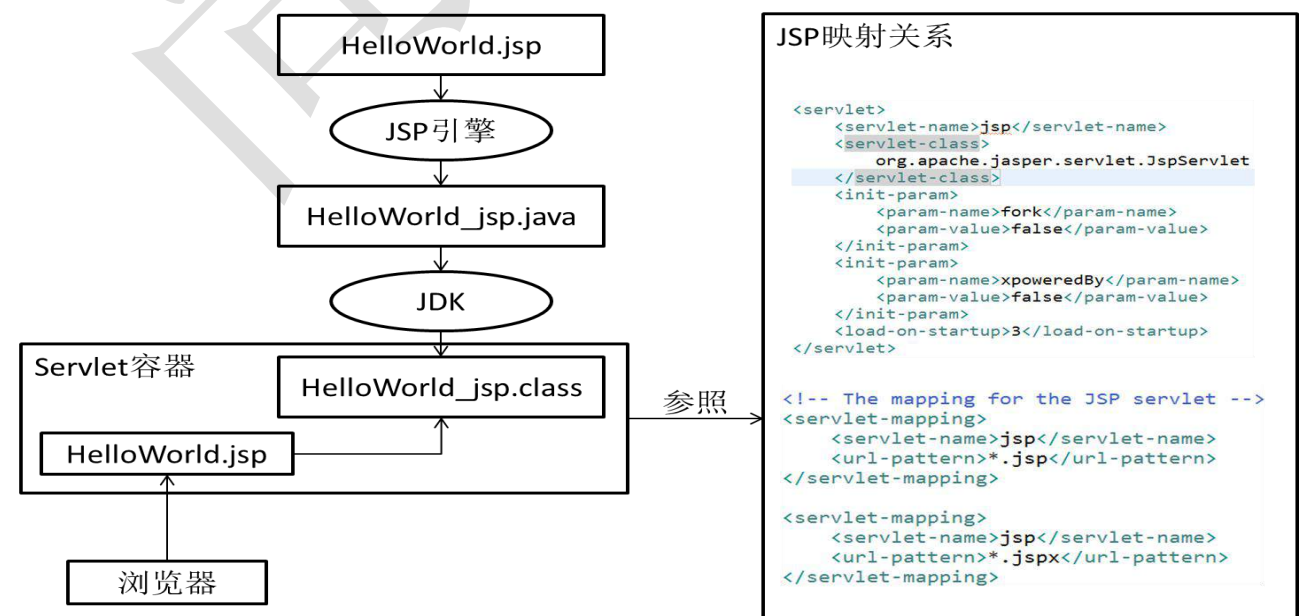
Servlet 和 HTML 二者的长处结合起来就诞生了 JSP 技术。

2. jsp 简介

Java Server Page

- 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。

3. jsp 运行原理



注意：JSP 仅在第一次访问时执行“翻译”和“编译”，之后再请求时就直接运行.class 文件了。

4. jsp 基本语法

接下来看一下 jsp 的基本语法及规则

4.1 jsp 模板元素

- 1) JSP 页面中的静态 HTML 内容称之为 JSP 模版元素，在静态的 HTML 内容之中可以嵌套 JSP 的其他各种元素来产生动态内容和执行业务逻辑。
- 2) JSP 模版元素定义了网页的基本骨架，即定义了页面的结构和外观

4.2 jsp 表达式

- 1) JSP 表达式 (expression) 提供了将一个 Java 变量或表达式的计算结果输出到客户端的简化方式，它将要输出的变量或表达式直接封装在 `<%=` 和 `%>` 之中
举例：Current time: `<%= new java.util.Date() %>`
- 2) JSP 表达式中的变量或表达式的计算结果将被转换成一个字符串，然后被插入到整个 JSP 页面输出结果的相应位置处。
- 3) JSP 表达式中的变量或表达式后面不能有分号 (;)，JSP 表达式被翻译成 Servlet 程序中的一条 `out.print(...)` 语句。

4.3 jsp 脚本片段

- 1) JSP 脚本片断 (scriptlet) 是指嵌套在 `<%` 和 `%>` 之中的一条或多条 Java 程序代码。
- 2) 在 JSP 脚本片断中，可以定义变量、执行基本的程序运算、调用其他 Java 类、访问数据库、访问文件系统等普通 Java 程序所能实现的功能。
- 3) 在 JSP 脚本片断可以直接使用 JSP 提供的隐含对象来完成 WEB 应用程序特有的功能。
- 4) JSP 脚本片断中的 Java 代码将被原封不动地搬移进由 JSP 页面所翻译成的 Servlet 的 `_jspService()` 方法中。所以，JSP 脚本片断之中只能是符合 Java 语法要求的程序代码，除此之外的任何文本、HTML 标记、其他 JSP 元素都必须在脚本片断之外编写。
- 5) JSP 脚本片断中的 Java 代码必须严格遵循 Java 语法，例如，每条命令执行语句后面必须用分号 (;) 结束。
- 6) 在一个 JSP 页面中可以有多条脚本片断（每个脚本片断代码嵌套在各自独立的一对 `<%` 和 `%>` 之间），在两个或多个脚本片断之间可以嵌入文本、HTML 标记和其他 JSP 元素。

举例：

```
<%
```

```
int x = 3;

%>
<p>这是一个 HTML 段落</p>
<%

    out.println(x);

%>
```

7) 多个脚本片断中的代码可以相互访问, 犹如将所有的代码放在一对<%%>之中的情况。
举例: 上面的 JSP 内容与下面的 JSP 内容具有同样的运行效果

```
<p>这是一个 HTML 段落</p>
<%

    int x = 3;
    out.println(x);

%>
```

8) 单个脚本片断中的 Java 语句可以是不完整的, 但是, 多个脚本片断组合后的结果必须是完整的 Java 语句, 例如, 涉及条件和循环处理时, 多个脚本片断及其他元素组合的结果必须能形成完整的条件和循环控制语句。

9) 由于脚本片断中的 Java 代码将被原封不动地搬移进由 JSP 页面所翻译成的 Servlet 的 `_jspService()` 方法中, 脚本片断之外的任何文本、HTML 标记以及其他 JSP 元素也都会被转换成相应的 Java 程序代码插入进 `_jspService()` 方法中, 且脚本片断和其他 JSP 元素的插入位置与它们在 JSP 页面中的原始位置相对应。

10) 在脚本片断中可以使用条件、循环、选择等流程控制语句来创建其周围的其他元素的执行逻辑, 因此, 在编写 JSP 页面时应考虑各个元素之间的先后顺序和相互关系, 特别是将循环、条件判断等语句分布在若干个脚本片断中编写时对其邻近的其他 JSP 元素产生的影响。

举例 1:

JSP 代码	<code>_jspService()</code> 方法中的代码
<pre><%for (int i=1; i<5; i++) {%> <h<%=i%>>www.atguigu.com</h<%=i%>> <%}%></pre>	<pre>for (int i=1; i<5; i++) { out.write("\r\n"); out.write("\t\t<h"); out.print(i); out.write(">www.atguigu.com</h"); out.print(i); out.write(">\r\n"); out.write("\t"); }</pre>

举例 2:

JSP 代码	<code>_jspService()</code> 方法中的代码
<pre><% if(age>65){ %> 可以退休 <%}else{ %> 不能退休 <%} %></pre>	<pre>if(age>65){ out.write("\r\n"); out.write("\t\t 可以退休\r\n"); out.write("\t"); }else{ out.write("\r\n");</pre>

	<pre>out.write("\t\t 不能退休\r\n"); out.write("\t"); }</pre>
--	---

4.4 jsp 声明

- 1) 由于 Java 语法的限制, 有很多语法成分不能在方法中使用, 例如: 其他方法、成员变量、静态代码块等等, 所以这些成分在 JSP 脚本中同样不能使用。
- 2) 如果希望 JSP 脚本中的代码出现在 `_jspService()` 方法外面, 可以使用 JSP 声明。
- 3) JSP 声明的格式是: `<%! 代码代码代码... %>`
- 4) JSP 隐含对象的作用域范围仅限于 `_jspService()` 方法, 所以在 JSP 声明中不能使用这些隐含对象。
- 5) 我们一般不使用 jsp 声明来写代码。

4.5 jsp 注释

- 1) JSP 注释格式: `<!-- 注释信息 -->`
- 2) JSP 注释生效的时间: JSP 引擎在将 JSP 页面翻译成 Servlet 程序时, 忽略 JSP 页面中被注释的内容。
- 3) 与 HTML 注释、Java 注释对比
 - JSP 的注释: jsp 生成 Java 源文件时被忽略
 - Java 的注释: 运行 class 文件时被忽略
 - HTML 的注释: 浏览器解析时被忽略

4.6 jsp 指令

4.6.1 指令简介

- 1) JSP 指令 (directive) 是为 JSP 引擎而设计的, 它们并不直接产生任何可见输出, 而只是告诉引擎如何处理 JSP 页面中的其余部分。
- 2) JSP 指令的基本语法格式:
 - `<%@ 指令名 属性名="值" %>`
 - 举例: `<%@ page contentType="text/html;charset=gb2312"%>`
 - 注意: 属性名部分是大小写敏感的
- 3) 在目前的 JSP2.0 中, 定义了 `page`、`include` 和 `taglib` 这三种指令, 每种指令中又都定义了一些各自的属性。如果要在一个 JSP 页面中设置同一条指令的多个属性, 可以使用多条指令语句单独设置每个属性, 也可以使用同一条指令语句设置该指令的多个属性。
 - 第一种方式:
`<%@ page contentType="text/html;charset=gb2312"%>`

```
<%@ page import="java.util.Date"%>
```

第二种方式:

```
<%@ page contentType="text/html;charset=gb2312" import="java.util.Date"%>
```

4.6.2 page 指令和 include 指令

1) page 指令

page 指令用于定义 JSP 页面的各种属性,无论 page 指令出现在 JSP 页面中的什么地方,它作用的都是整个 JSP 页面。为了保持程序的可读性和遵循良好的编程习惯,page 指令最好是放在整个 JSP 页面的起始位置

JSP 2.0 规范中定义的 page 指令的完整语法:

```
<%@ page
    [ language="java" ]
    [ extends="package.class" ]
    [ import="{package.class | package.*}, ..." ]
    [ session="true | false" ]
    [ buffer="none | 8kb | sizekb" ]
    [ autoFlush="true | false" ]
    [ isThreadSafe="true | false" ]
    [ info="text" ]
    [ errorPage="relative_url" ]
    [ isErrorPage="true | false" ]
    [ contentType="mimeType [ ;charset=characterSet ]" | "text/html ; charset=ISO-8859-1" ]
    [ pageEncoding="characterSet | ISO-8859-1" ]
    [ isELIgnored="true | false" ]
%>
```

[1]import 属性: 指定 JSP 页面转换成 Servlet 时应该导入的包。

[2]pageEncoding 属性: 设置 JSP 页面翻译成 Servlet 源文件时使用的字符集。

[3]contentType 属性: 设置 Content-Type 响应报头,标明即将发送到客户程序的文档的 MIME 类型以及浏览器对响应内容的解码字符集。

[4]errorPage 属性: 指定当前 JSP 抛出异常时的转发页面。

[5]isErrorPage 属性: 指定当前页面是不是一个显示错误消息的页面,如果是,则会自动创建 exception 对象,否则就不会创建 exception 对象。

[6]session 属性: 控制页面是否参与 HTTP 会话,其本质是要不要自动创建 session 隐含对象以供使用。

[7]isELIgnored 属性: 指定当前页面是否忽略 EL 表达式,如果忽略,EL 表达式的内容将会原封不动的输出到浏览器端。

2) include 指令

a) include 指令用于通知 JSP 引擎在翻译当前 JSP 页面时将其他文件中的内容合并进当前 JSP 页面转换成的 Servlet 源文件中,这种在源文件级别进行引入的方式称之为静态引入,当前 JSP 页面与静态引入的页面紧密结合为一个 Servlet。

b) 语法:

```
<%@ include file="relativeURL"%>
```

其中的 file 属性用于指定被引入文件的**相对路径**。

c) 细节:

- ✓ 被引入的文件必须遵循 JSP 语法，其中的内容可以包含静态 HTML、JSP 脚本元素、JSP 指令和 JSP 行为元素等普通 JSP 页面所具有的一切内容。

- ✓ 被引入的文件可以使用任意的扩展名，即使其扩展名是 html，JSP 引擎也会按照处理 JSP 页面的方式处理它里面的内容，为了见明知意，JSP 规范建议使用 .jspx (JSP fragments) 作为静态引入文件的扩展名。

- ✓ 在将 JSP 文件翻译成 Servlet 源文件时，JSP 引擎将合并被引入的文件与当前 JSP 页面中的指令元素（设置 pageEncoding 属性的 page 指令除外），所以，除了 import 和 pageEncoding 属性之外，page 指令的其他属性不能在这两个页面中有不同的设置值。

4.7 jsp 标签

4.7.1 概述

1) JSP 还提供了一种称之为 Action 的元素，在 JSP 页面中使用 Action 元素可以完成各种通用的 JSP 页面功能，也可以实现一些处理复杂业务逻辑的专用功能。

2) Action 元素采用 XML 元素的语法格式，即每个 Action 元素在 JSP 页面中都以 XML 标签的形式出现。

3) JSP 规范中定义了一些标准的 Action 元素，这些元素的标签名都以 jsp 作为前缀，并且全部采用小写，例如，<jsp:include>、<jsp:forward>等等。

4.7.2 <jsp:include>标签

1) <jsp:include>标签用于把另外一个资源的**输出内容**插入进当前 JSP 页面的输出内容之中，这种在 JSP 页面执行时的引入方式称之为动态引入。

2) 语法:

```
<jsp:include page="relativeURL | <%=expression%>" flush="true|false" />
```

- page 属性用于指定被引入资源的相对路径，它也可以通过执行一个表达式来获得。

- flush 属性指定在插入其他资源的输出内容时，是否先将当前 JSP 页面的已输出的内容刷新到客户端。

3) <jsp:include>标签与 include 指令的比较

- @include 指令在翻译“主体”代码时起作用。相当于把“被包含”的 JSP 代码复制到“主体”JSP 文件中，生成一个合并之后的 Servlet 源文件，所以二者在代码中不能使用相同的变量名等——凡是放在一起会冲突的内容都不被允许。

- <jsp:include>标签会被翻译为 JspRuntimeLibrary 类的 include()方法，“被包含”

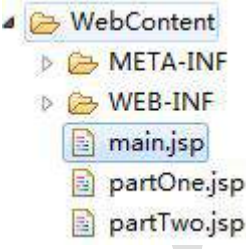
的 JSP 页面会单独翻译、编译；每次“主体”JSP 被请求时，都会先执行“被包含”的 JSP，将执行结果合并到 HTML 代码中，一起发送到浏览器端。所以使用<jsp:include>标签时，“被包含”的 JSP 中的代码不会和“主体”JSP 冲突。

○对比

	@include 指令	<jsp:include>标签
特点	静态包含	动态包含
语法的基本形式	<%@ include file="..." %>	<jsp:include page="..." />
包含动作发生的时机	翻译期间	请求期间
生成 Servlet 源文件	一个	多个
合并方式	代码复制	合并运行结果
包含的内容	文件实际内容	页面输出结果
代码冲突	有可能	不可能
被包含页面中可否设置影响主页面的响应报头	可以	不可以

○举例

[1]创建三个 JSP 页面结构如下



[2]编辑 main.jsp

```
<%@ include file="partOne.jsp" %>
<br />
<jsp:include page="partTwo.jsp"></jsp:include>
```

[3]编辑 partOne.jsp

```
<%= "I am part one!" %>
```

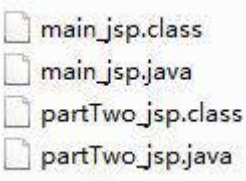
[4]编辑 partTwo.jsp

```
<%= "I am part two!" %>
```

[5]运行 main.jsp

```
I am part one!
I am part two!
```

[6]翻译和编译的结果



[7] main_jsp.java 中的内容


```
out.write("<!DOCTYPE.....>\r\n");
out.write("<html>\r\n");
out.write("<head>\r\n");
out.write("<meta .....>\r\n");
out.write("<title>Insert title here</title>\r\n");
out.write("</head>\r\n");
out.write("<body>\r\n");
out.write("\r\n");
out.write("\t");
out.print("I am part one! ");
out.write("\r\n");
out.write("\t<br />\r\n");
out.write("\t");
org.apache.jasper.runtime.JspRuntimeLibrary
    .include(request, response, "partTwo.jsp", out, false);
out.write("\r\n");
out.write("\r\n");
out.write("</body>\r\n");
out.write("</html>");
```

[8] partTwo.jsp.java 中的内容

```
out.print("I am part two! ");
```

4.7.3 <jsp:forward>标签

1) <jsp:forward>标签用于把请求转发给另外一个资源

2) 语法:

```
<jsp:forward page="relativeURL|<%=expression%>" />
```

page 属性用于指定请求转发到的资源的相对路径,它也可以通过执行一个表达式来获得。

4.7.4 <jsp:param>标签

1) 当使用<jsp:include>和<jsp:forward>标签引入或将请求转发给的资源是一个能动态执行的程序时,例如 Servlet 和 JSP 页面,那么,还可以使用<jsp:param>标签向这个程序传递请求参数。

语法 1:

```
<jsp:include page="relativeURL | <%=expression%>">
    <jsp:param name="parameterName" value="parameterValue|<%= expression %>" />
</jsp:include>
```

语法 2:

```
<jsp:forward page="relativeURL | <%=expression%>">
    <jsp:param name="parameterName" value="parameterValue|<%= expression %>" />
</jsp:include>
```

2) `<jsp:param>` 标签的 `name` 属性用于指定参数名, `value` 属性用于指定参数值。在 `<jsp:include>` 和 `<jsp:forward>` 标签中可以使用多个 `<jsp:param>` 标签来传递多个参数。

5. jsp 九大隐含对象

在 JSP 页面上编写 Java 代码时, 有九个可以直接使用的内置对象。

PageContext pageContext
HttpServletRequest request
HttpSession session
ServletContext application
HttpServletResponse response
ServletConfig config
Throwable exception
JspWriter out
Object page

为什么可以在页面使用它们, 因为我们发现, 页面是在 `service` 方法中进行解析的。而 `service` 方法在解析页面之前申明了。在页面设置为 `isErrorPage="true"` 的时候, `exception` 对象就会显示

```
public void _jspService(final javax.servlet.http.HttpServletRequest request, ①
                        final javax.servlet.http.HttpServletResponse response) ②
    throws java.io.IOException, javax.servlet.ServletException {

    final javax.servlet.jsp.PageContext pageContext; ③
    javax.servlet.http.HttpSession session = null; ④
    java.lang.Throwable exception = org.apache.jasper.runtime.JspRuntimeLibrary.getThrowable(request);
    if (exception != null) { ⑤
        response.setStatus(javax.servlet.http.HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
    }
    final javax.servlet.ServletContext application; ⑥
    final javax.servlet.ServletConfig config; ⑦
    javax.servlet.jsp.JspWriter out = null; ⑧
    final java.lang.Object page = this; ⑨
    javax.servlet.jsp.JspWriter _jspx_out = null;
    javax.servlet.jsp.PageContext _jspx_page_context = null;
```

5.1 PageContext pageContext

pageContext 主要有以下功能

1) 获取其它隐含对象


```
getException 方法返回 exception 隐式对象
getPage 方法返回 page 隐式对象
getRequest 方法返回 request 隐式对象
getResponse 方法返回 response 隐式对象
getServletConfig 方法返回 config 隐式对象
getServletContext 方法返回 application 隐式对象
getSession 方法返回 session 隐式对象
getOut 方法返回 out 隐式对象
```

2) 作为域对象

可以设置、获取属性值

```
public void setAttribute(java.lang.String name,java.lang.Object value)
public java.lang.Object getAttribute(java.lang.String name)
public void removeAttribute(java.lang.String name)
```

3) 访问其它属性域

```
1 public java.lang.Object getAttribute(java.lang.String name,int scope)
2 public void setAttribute(java.lang.String name, java.lang.Object value,int
scope)
3 public void removeAttribute(java.lang.String name,int scope)
```

int scope 代表各个域的常量，可取值如下

```
1 PageContext.APPLICATION_SCOPE
2 PageContext.SESSION_SCOPE
3 PageContext.REQUEST_SCOPE
4 PageContext.PAGE_SCOPE
```

5.2 HttpServletRequest request

域对象，可以存取属性值，用来在域中共享。

```
public void setAttribute(java.lang.String name,java.lang.Object value)
public java.lang.Object getAttribute(java.lang.String name)
public void removeAttribute(java.lang.String name)
```

5.3 HttpSession session

域对象，可以存取属性值，用来在域中共享。

5.4 ServletContext application

域对象，可以存取属性值，用来在域中共享。

5.5 四个域对象的比较

作用范围：

域对象	作用范围	起始时间	结束时间
pageContext	当前 JSP 页面	页面加载	离开页面
request	同一个请求	收到请求	响应
session	同一个会话	开始会话	结束会话
application	当前 Web 应用	Web 应用加载	Web 应用卸载

5.6 HttpServletResponse response

response 对象：代表 HTTP 响应

5.7 ServletConfig config

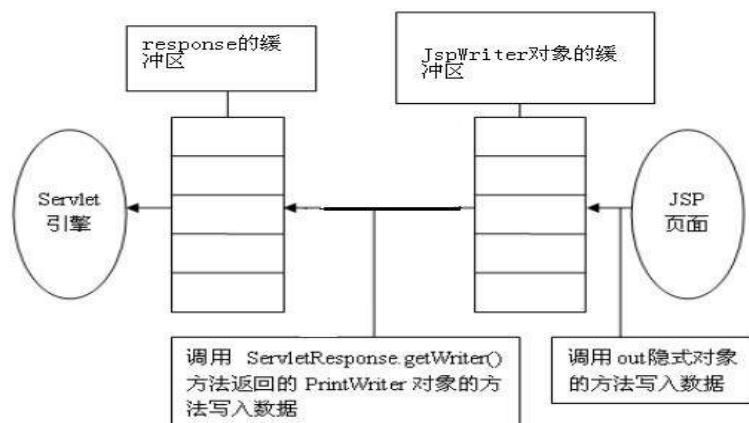
config 对象：ServletConfig 实例，代表 Servlet 配置信息

5.8 Throwable exception

exception 对象：封装了当前 JSP 页面捕获到的异常信息

5.9 JspWriter out

- 1) out 对象用于向客户端发送文本数据。
- 2) out 对象是通过调用 pageContext 对象的 getOut() 方法返回的，其作用和用法与 ServletResponse.getWriter 方法返回的 PrintWriter 对象非常相似。
- 3) JSP 页面中的 out 对象的类型为 JspWriter，JspWriter 相当于一种带缓存功能的 PrintWriter，设置 JSP 页面的 page 指令的 buffer 属性可以调整它的缓存大小，甚至关闭它的缓存。
- 4) 只有向 out 对象中写入了内容，且满足如下任何一个条件时，out 对象才去调用 ServletResponse.getWriter 方法，并通过该方法返回的 PrintWriter 对象将 out 对象的缓冲区中的内容真正写入到 Servlet 引擎提供的缓冲区中：
- 5) out 对象的工作原理图



5.10 Object page

this 的一个引用，但却是 Object 类型的，导致能用的方法仅限于 Object 类的方法，还不如 this 本身实用。

第 10 章 EL 表达式

1. 提出问题

在 JSP 页面上获取域对象中保存的数据和获取请求参数数据是非常常用的操作。

获取请求域中的数据

```
<%=request.getAttribute("message") == null ? "" : request.getAttribute("message") %>
```

获取请求参数

```
<%=request.getParameter("userName")==null? "": request.getParameter("userName")%>
```

有没有什么办法能够让上述代码简洁一些呢？有！

获取请求域中的数据

```
${requestScope.message }
```

获取请求参数

```
${param.userName }
```

这就是 EL 表达式，它能够极大的简化 JSP 页面上数据的显示。

2. el 简介

1) EL 全名为 Expression Language，它可以在 JSP 页面上可以直接使用

格式：\${表达式内容 }

例如：

```
value="${requestScope.emp.empName}"
```

相当于

```
<% Employee emp = (Employee)request.getAttribute("emp"); %>
    员工姓名<input type="text" name="empName"
        value="<%=emp.getEmpName() %>" class="ipt" />
```

2) EL 表达式的功能

获取请求参数并显示

●当前请求参数没有获取到时返回空字符串，而不是 null。这样做的好处是空字符串在网页上是没有任何显示的，不必特殊处理。

读取 4 个域对象属性值

●读取不到时返回空字符串而不是 null。

3. el 隐含对象

el 能获取域的值，到底能获取那些域的值，我们来探索一下。

EL 中有 7 个常用的对象可以直接使用

隐含对象	类型	说明
pageContext	javax.servlet.jsp.PageContext	就是 JSP 页面上的 pageContext
pageScope	java.util.Map<String,Object>	Page 范围属性名和属性值
requestScope	java.util.Map<String,Object>	Request 范围属性名和属性值
sessionScope	java.util.Map<String,Object>	Session 范围属性名和属性值
applicationScope	java.util.Map<String,Object>	Web 应用范围属性名和属性值
param	java.util.Map<String,String>	对应一个请求参数
paramValues	java.util.Map<String,String[]>	对应一组请求参数

4. el 隐含对象解析

1) pageContext

它是代表当前页面的 PageContext 对象,通过它的 get 方法可以得到 jsp 中的其它八大隐含对象

2) 四个域对象所对应的隐含对象:

- ①Map<String, Object> pageScope——对应 pageContext 域 `${ pageScope . username }`
- ②Map<String, Object> requestScope——对应 request 域 `${ requestScope . username }`
- ③Map<String, Object> sessionScope——对应 session 域 `${ sessionScope . username }`
- ④ Map<String, Object> applicationScope —— 对应 application 域 `${ applicationScope . username }`

3) 请求参数数据 param

Map<String, String> param:

保存的是请求参数的 key--value(value 只有一个),input `${ param . username }`

Map<String, String[]> paramValues:

保存的是请求参数的 key-value(value 有多个),checkbox `${ paramValues.username }`

5. el 取值方式

1) 使用“点”

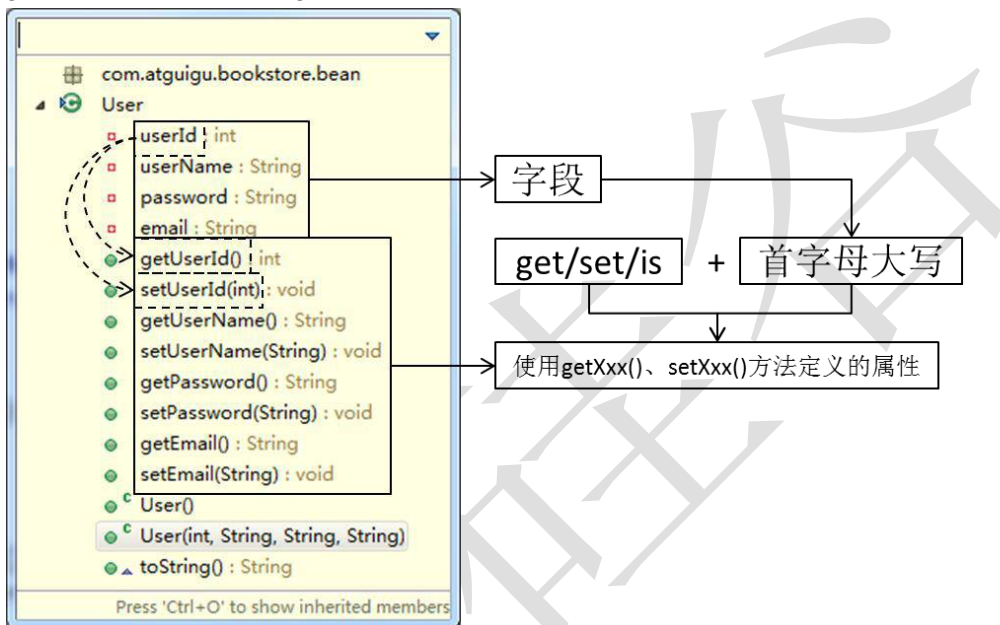
如果对象是 map 类型对象: `map.keyName` --> 得到对应的 value 值

实质: `map.get(keyName)`

如果对象是一般对象: `object.propertyName` ---> 得到对就的属性值

实质是: `object.getXxx()`

[特别强调: 字段和属性]



EL 表达式访问的是使用 `getXxx()`、`setXxx()` 方法定义的属性，而不是字段本身。例如：`${user.userName}` 会去调用 `getUserName()` 方法，而不是 `userName` 字段。所以如果没有声明 `getUserName()` 方法或权限不允许则无法显示属性值。

`getUserName()` 方法未定义或权限不足时会抛出异常: `javax.el.PropertyNotFoundException: Property 'userName' not readable on type java.lang.String`

2) 使用“[]”

map 的键有时包含“点”，这时就不能再使用“点”取值了。

例如：

```
<input type="text" name="name.x" value="aaa">
${param['name.x']}
```

如果对象是 map 类型对象: `map["keyName"]` --> 得到对应的 value 值

实质: `map.get(key)`

如果对象是一般对象: `object["propertyName"]` ---> 得到对就的属性值

实质是: `map.getXxx()`

如果是取出 list 的元素，也使用[]：

如：

```
<input type="checkbox" name="username" value="1">
<input type="checkbox" name="username" value="2">
<input type="checkbox" name="username" value="3">
<input type="checkbox" name="username" value="4">
```

取值：<h1>多选：\${paramValues.username[0]}</h1>

简化格式

域范围对象可以不用写，系统会自动按范围从小到大的顺序查找：

pageScope->requestScope-->sessionScope-->applicationScope

找到就不会继续找了，如果没有找到显示空白(不是 null，而是一个空串)不会报错

6. el 表达式运算

- 1) 算术运算：\${5+3}
- 2) 关系运算： \${5>3}
- 3) 逻辑运算： \${true&&false}
- 4) empty 运算：\${empty requestScope.emp }
 - ①null
 - [1]变量的值是 null
 - [2]域对象中不存在这个变量
 - ②空集合
 - ③空数组
 - ④ 空字符串
 - ⑤ 空字符
- 5) 三目条件运算： \${16<5?'a':"big" }

7. 获取项目虚拟路径

- 1) 获取 request 对象\${pageContext.request }
- 2) 获取 contextPath\${pageContext.request.contextPath }

第 11 章 JSTL

1. 提出问题

展示图书分类信息数据列表，需要 JSP 脚本、JSP 表达式、HTML 标签混合使用，代码可读性差、容易出错。

```
<table>
<%
    List<Category> cateList = (List<Category>)request.getAttribute("cateList");
    if(cateList.isEmpty() || cateList == null) {
        %> <tr><td>没有分类信息</td></tr> <%
    }else{
        for(int i = 0; i < cateList.size(); i++){
            Category cate = cateList.get(i);
            %>
            <tr>
                <td>
                    <form action="${pageContext.request.contextPath }/CateServlet?method=update"
                        <%=cate.getCateName() %>
                        <input type="hidden" name="cateId" value="<%=cate.getCateId() %>" />
                        <input type="text" name="cateName" value="<%=cate.getCateName() %>" />
                        <input type="submit" value="更新" />
                        <input type="button" value="删除" class="del" />
                    </form>
                </td>
            </tr>
        }
    }
%>
```

有没有一种更简洁、和 HTML 标签风格更搭配的技术解决上述问题呢？JSTL 可以解决！

2. jstl 简介

- 1) Java Server Pages Standard Tag Library——JSP 标准标签库。
- 2) JSTL 是一个标准的、已制定好的标签库，可以应用于各种领域，如：**基本输入输出**、**流程控制（分支、迭代）**、XML 文件剖析、数据库查询及**国际化**和文字格式标准化的应用等。
- 3) JSTL 所提供的标签函数库主要分为五大类：
 - ①**核心标签库**（Core tag library）
 - ②**l18N 格式标签库**（l18N-capable formatting tag library）
 - ③SQL 标签库（SQL tag library）
 - ④XML 标签库（XML tag library）
 - ⑤**函数标签库**（Functions tag library）

JSTL	前缀	URI	范例
核心标签库	c	http://java.sun.com/jsp/jstl/core	<c:out value="Hello"></c:out>
l18N 格式标签库	fmt	http://java.sun.com/jsp/jstl/fmt	<fmt:formatDate value="{date }"/>
SQL 标签库	sql	http://java.sun.com/jsp/jstl/sql	<sql:query ... ></sql:query>

XML 标签库	x	http://java.sun.com/jsp/jstl/xml	<x:parse varDom="..."></x:parse>
函数标签库	fn	http://java.sun.com/jsp/jstl/functions	\${fn:split(customerNames, ";")}

3. jstl 标签库使用

1) 导入 JSTL 所需 JAR 包

taglibs-standard-spec-1.2.1.jar

taglibs-standard-impl-1.2.1.jar

2) 在 JSP 文件中导入所需要使用的 JSTL 标签库

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

3) 使用标签库中的各个标签

<c:set value="\${pageScope.attrName}" var="newName"></c:set>

4. 核心标签库 (<c>标签库)

功能分类	标签名称
表达式操作	out
	set
	remove
流程控制	if
	choose
	when
	otherwise
	forEach
	forTokens
URL 操作	param
	url
	redirect

c:catch
c:choose
c:forEach
c:forTokens
c:if
c:import
c:otherwise
c:out
c:param
c:redirect
c:remove
c:set
c:url
c:when

Press 'Alt+/'

4.1 核心标签库：表达式操作

1) c:out

输出域对象指定属性名所对应的属性值, 如果为 null, 显示默认值

<c:out value="\${name}" default="defaultValue"/>

2) c:set

①向域对象中保存指定的属性名和属性值

```
<c:set value="${person.name}" var="myName"/>
```

②修改域对象中 JavaBean 的属性值

```
<c:set target="${person}" property="name" value="Jack"/>
```

c:remove

删除域对象中的指定属性

```
<c:remove var="myName"/>
```

4.2 核心标签库：分支

1) 一重条件判断[c:if]

多个 **c:if** 之间是没有任何关系的

```
<% pageContext.setAttribute("age", "25"); %>
<c:if test="${age < 18}">祖国的花骨朵</c:if>
<c:if test="${age >= 18 && age < 25}">2B 青年</c:if>
<c:if test="${age >= 25 && age < 40}">苦 B 青年</c:if>
<c:if test="${age >= 40 && age < 50}">年富力强</c:if>
<c:if test="${age >= 50 && age < 65}">享受成功</c:if>
<c:if test="${age >= 65}">退休生活</c:if>
```

2) 多重条件判断[c:choose/c:when/c:otherwise]

程序执行时，按照从上往下的顺序执行 **c:when**，一旦找到满足条件的 **c:when** 就不再继续执行了，如果没有任何满足的 **c:when** 则执行 **c:otherwise**。

```
<c:choose>
  <c:when test="${age < 18}">祖国的花骨朵</c:when>
  <c:when test="${age < 25}">2B 青年</c:when>
  <c:when test="${age < 40}">苦 B 青年</c:when>
  <c:when test="${age < 50}">年富力强</c:when>
  <c:when test="${age < 65}">享受成功</c:when>
  <c:otherwise>退休生活</c:otherwise>
</c:choose>
```

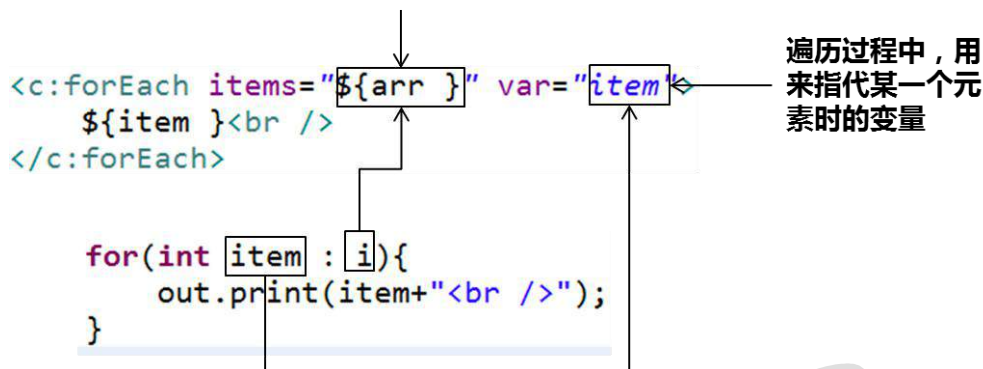
4.3 核心标签库：迭代

c:forEach

1) 遍历对象为索引集合[List/Set/Array]

```
<c:forEach items="${listAttr}" var="person">
  ${person.name }:${person.age }<br />
</c:forEach>
```

要遍历的集合或数组



2) 遍历对象为键值对集合

```
<c:forEach items=\"\${personMap}\" var=\"entry\">
    ${entry.key }:${entry.value.name }:${entry.value.age }<br />
</c:forEach>
```

3) 遍历过程中的元素状态

[1] 使用 **varStatus** 属性获取 status 元素状态对象，这个对象的类型是：
javax.servlet.jsp.jstl.core.LoopTagSupport\$1Status

```
<c:forEach items=\"\${listAttr}\" var=\"person\" varStatus=\"status\">
    ${person.name }:${person.age }:${status.class }<br />
</c:forEach>
```

[2] Status 实现了 javax.servlet.jsp.jstl.core.LoopTagStatus 接口

LoopTagStatus	
getCurrent() : Object	→ 当前本次迭代的集合项
getIndex() : int	→ 当前这次迭代从0开始的迭代索引
getCount() : int	→ 当前这次迭代从1开始的迭代计数
isFirst() : boolean	→ 当前迭代是否为第一次迭代，boolean类型
isLast() : boolean	→ 当前迭代是否为最后一次迭代，boolean类型
getBegin() : Integer	
getEnd() : Integer	
getStep() : Integer	→ 获取当前迭代过程的步长

[3] 获取遍历状态

```
<c:forEach items=\"\${arr}\" var=\"item\" varStatus=\"status\" step=\"2\">
    <tr>
        <td>${item }</td>
        <td>${status.index }</td>
        <td>${status.current }</td>
```

```

        <td>${status.count }</td>
        <td>${status.first }</td>
        <td>${status.last }</td>
        <td>${status.step }</td>
    </tr>
</c:forEach>

```

数据	Index	Current	Count	isFirst	isLast	step
00	0	00	1	true	false	2
22	2	22	2	false	false	2
44	4	44	3	false	false	2
66	6	66	4	false	false	2
88	8	88	5	false	true	2

4) c:forTokens

根据指定字符拆分字符串，并遍历得到的集合

```

<% pageContext.setAttribute("str", "aa,bb,ee,mm,tt"); %>
<c:forTokens items="${str}" delims="," var="item">
    ${item }<br />
</c:forTokens>

```

5) 练习：表格隔行变色

姓名	年龄
AAA	101
BBB	102
CCC	103
DDD	104
EEE	105

4.4 核心标签库：URL 操作

重定向

```

<%-- 使用 context 属性指定项目虚拟路径，使用 url 属性指定重定向的目标位置 --%>
<c:redirect
    context="${pageContext.request.contextPath}"
    url="/target2.jsp" />

```

传递请求参数

```
<%-- 使用 context 属性指定项目虚拟路径，使用 url 属性指定重定向的目标位置 --%>
<c:redirect
  context="${pageContext.request.contextPath}"
  url="/target2.jsp">
  <%-- 使用 name 属性指定请求参数名，使用 value 属性指定请求参数值 --%>
  <c:param name="sayHi" value="hi..."></c:param>
</c:redirect>
```

5. 函数标签库 (fn)

1) 简介

在 JSTL 中专门定义了一些函数给 EL 表达式使用，来完成一些简单通用的功能，大部分都一些字符串相关操作。

2) 格式

`${fn:函数名(函数参数)}`

3) 功能

执行对应的函数并显示

4) 使用

- ① 在 JSP 中导入 JSTL 中的 EL 函数标签库

```
<%@taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn"%>
```

- ② 使用 el 函数

```
${fn:length(person1.name)}
```

```
${fn:substring(s1,1,3)}
```

5) 标签库提供的功能

函数名	举例	功能
contains	<code><c:if test="\${fn:contains(name, searchString)}"></code>	是否包含
containsIgnoreCase	<code><c:if test="\${fn:containsIgnoreCase (name, searchString)}"></code>	是否包含，不区分大小写
endsWith	<code><c:if test="\${fn:endsWith(filename, ".txt")}"></code>	是否以.txt 结束
startsWith	<code><c:if test="\${fn:startsWith(product.id, "100-")}"></code>	是否以 100- 开始
indexOf	<code>\${fn:indexOf(name, "-")}</code>	返回-的索引
join	<code>\${fn:join(array, ";")}</code>	将数组元素使用;拼接
length	<code>\${fn:length(shoppingCart.products)}</code>	返回集合中元素个数
replace	<code>\${fn:replace(text, "a", "?")}</code>	使用? 替换所有 a
split	<code>\${fn:split(customerNames, ";")}</code>	使用;将字符串分割
substring	<code>\${fn:substring(zip, index, end)}</code>	字符串截取, [index,end) End=-1,表示截取剩下所有
substringAfter	<code>\${fn:substringAfter(zip, "-")}</code>	截取-之后的所有元素, 不包括 -

substringBefore	<code>\${fn.substringBefore(zip, "-")}</code>	截取-之前的所有元素，不包括-
toLowerCase	<code>\${fn.toLowerCase(product.name)}</code>	全部转换为小写
UpperCase	<code>\${fn.UpperCase(product.name)}</code>	全部转化为大写
trim	<code>\${fn.trim(name)}</code>	去除字符串两头的空串

第 12 章 Cookie

1、cookie 简介

1.1 什么是 cookie

cookie, 有时我们也用其复数形式 cookies, 是服务端保存在浏览器端的数据片段。以 key/value 的形式进行保存。每次请求的时候, 请求头会自动包含本网站此目录下的 cookie 数据。网站经常使用这个技术来识别用户是否登陆等功能。

简单的说, cookie 就是服务端留给计算机用户浏览器端的小文件。

- HTTP 是无状态协议, 服务器不能记录浏览器的访问状态, 也就是说服务器不能区分中两次请求是否由一个客户端发出。这样的设计严重阻碍的 Web 程序的设计。如: 在我们进行网购时, 买了一条裤子, 又买了一个手机。由于 http 协议是无状态的, 如果不通过其他手段, 服务器是不能知道用户到底买了什么。而 Cookie 就是解决方案之一。
- Cookie 实际上就是服务器保存在浏览器上的一段信息。浏览器有了 Cookie 之后, 每次向服务器发送请求时都会同时将该信息发送给服务器, 服务器收到请求后, 就可以根据该信息处理请求。
- 例如: 我们上文说的网上商城, 当用户向购物车中添加一个商品时, 服务器会将这个条信息封装成一个 Cookie 发送给浏览器, 浏览器收到 Cookie, 会将它保存在内存中(注意这里的内存是本机内存, 而不是服务器内存), 那之后每次向服务器发送请求, 浏览器都会携带该 Cookie, 而服务器就可以通过读取 Cookie 来判断用户到底买了哪些商品。当用户进行结账操作时, 服务器就可以根据 Cookie 的信息来做结算。

- Cookie 的用途:

网上商城的购物车
保持用户登录状态

- Cookie 的缺点

Cookie 做为请求或响应报文发送, 无形中增加了网络流量。

Cookie 是明文传送的安全性差。

Cookie 中保存数据是不稳定的, 用户可以随时清理 cookie

各个浏览器对 Cookie 有限制, 使用上有局限

1.2 庐山真面目

chrome 的 cookie 位置：

C:\Users\lfy\AppData\Local\Google\Chrome\User Data\Default\Cookies

ie 中 cookie 位置：

C:\Users\lfy\AppData\Local\Microsoft\Windows\InetCache

点击设置->查看对象即可



chrome 中查看 cookie

Name	Value	Domain	Path	Expires / Max-Age	Size
BAIDUID	6A744EBF82AA8BD36F1997037766FAB6:FG=1	.baidu.com	/	2084-03-24T10:44:...	44
BD_HOME	0	www.baid...	/	Session	8
BD_UPN	12314753	www.baid...	/	2016-03-26T01:07:...	14
BIDUPSID	0C392635BBA70069488A5776842B48EE	.baidu.com	/	2048-02-26T02:07:...	40
H_PS_PSSID	19140_1456_18240_17000_15234_11575	.baidu.com	/	Session	44
PSTM	1457249449	.baidu.com	/	2084-03-24T10:44:...	14
pgv_pvi	8968344576	.baidu.com	/	2038-01-18T00:00:...	17

cookie 如上图所示

从上图可以看出 cookie 是键值对的形式，有过期时间（Max-Age，session 表示在这个会话期内有效）。

1.3 cookie 原理

1) 总的来看 Cookie 像是服务器发给浏览器的一张“会员卡”，浏览器每次向服务器发送请求时都会带着这张“会员卡”，当服务器看到这张“会员卡”时就可以识别浏览器的身份。

实际上这个所谓的“会员卡”就是服务器发送的一个响应头：

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=95A92EC1D7CCB4ADFC24584CB316382E; Path=/Test_cookie
Content-Type: text/html; charset=UTF-8
Content-Length: 270
Date: Thu, 14 May 2015 03:34:00 GMT

<!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>

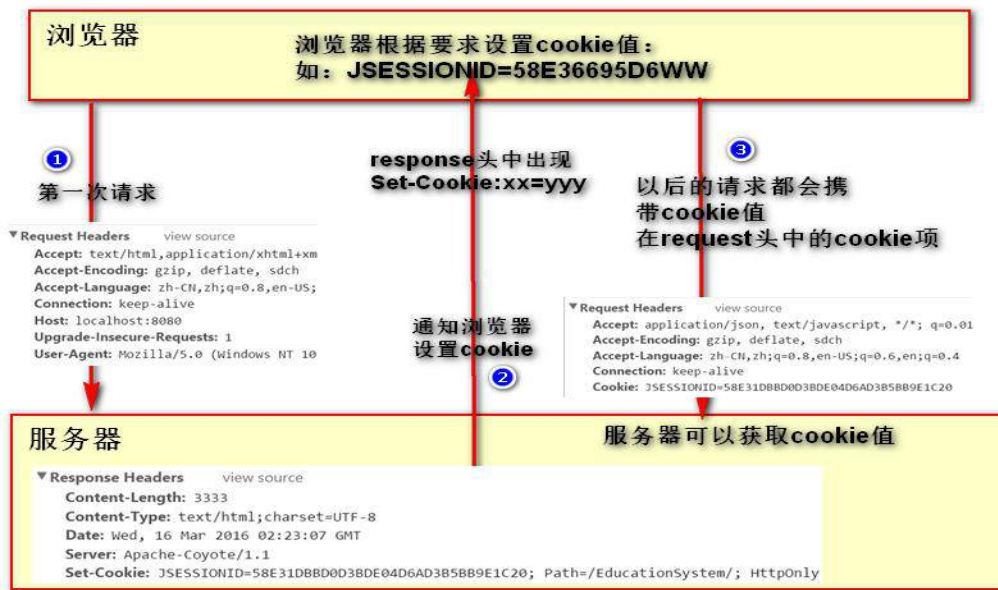
</body>
</html>
```

2) 如图 Set-Cookie 这个响应头就是服务器在向服务器发“会员卡”，这个响应头的名字是 Set-Cookie，后边 JSESSIONID=95A92EC1D7CCB4ADFC24584CB316382E 和 Path=/Test_cookie，是两组键值对的结构就是服务器为这个“会员卡”设置的信息。浏览器收到该信息后就会将它保存到内存或硬盘中。

3) 当浏览器再次向服务器发送请求时就会携带这个 Cookie 信息：

```
GET /Test_cookie/b.jsp HTTP/1.1
Accept: */*
Accept-Language: zh-CN
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR
Accept-Encoding: gzip, deflate
Host: localhost:8080
Connection: Keep-Alive
Cookie: JSESSIONID=7863BF36A7668B98257C2001FCF2F8
```

这是浏览器发送的请求报文，中间画红框的就是 Cookie 信息，这里可以理解为浏览器这次带着“会员卡”再次访问服务器。于是服务器就可以根据 Cookie 信息来判断浏览器的状态。原理图如下：



2、Cookie 的使用

2.1 创建对象

cookie 是由服务端创建的，由浏览器端保存的。所以创建对象我们应该在服务端创建 cookie 的创建方法：

1) 创建一个 CookieServlet

在 Servlet 的 doPost()方法中编写如下代码：

```
//创建一个Cookie对象
Cookie cookie = new Cookie("username", "zhangsan");
//将Cookie对象放入response对象中
response.addCookie(cookie);
```

2) 在浏览器中访问该 Servlet，会发现响应头中出现如下内容：

Set-Cookie: username=zhangsan

如此就成功的向浏览器设置了一个 Cookie，当我们在刷新页面时会发现浏览器的请求头中出现如下代码：

Cookie: username=zhangsan

3) 同样我们还可以同时设置多个 Cookie：

```
//创建一个Cookie对象
Cookie cookie1 = new Cookie("username", "zhangsan");
Cookie cookie2 = new Cookie("password", "123456");
Cookie cookie3 = new Cookie("age", "20");
//将Cookie对象放入response对象中
response.addCookie(cookie1);
response.addCookie(cookie2);
```



```
response.addCookie(cookie3);
```

浏览器会按以下形式发送 Cookie:

Cookie: username=zhangsan; password=123456; age=20

4) 设置 Cookie 就是两个步骤:

创建 Cookie 对象

将 Cookie 对象加入到 response 中

2.2 设置 cookie

2.2.1 cookie 的有效时间

- 1) 经过上边的介绍我们已经知道 Cookie 是存储在浏览器中的,但是可想而知一般情况下浏览器不可能永远保存一个 Cookie,一来是占用硬盘空间,再来一个 Cookie 可能只在某一时刻有用没必要长久保存。
- 2) 所以我们还需要为 Cookie 设置一个有效时间。
- 3) 通过 Cookie 对象的 `setMaxAge()` 可以设置 Cookie 的有效时间。
其中 `setMaxAge()` 接收一个 `int` 型的参数,来设置有效时间。参数主要有一下四种情况:
 - 设置为 0, `setMaxAge(0)`
Cookie 立即失效,下次浏览器发送请求将不会在携带该 Cookie
 - 设置大于 0, `setMaxAge(60)`
表示有效的秒数 60 就代表 60 秒即 1 分钟,也就是 Cookie 在 1 分钟后失效。
 - 设置小于 0, `setMaxAge(-1)`
设置为负数表示当前会话有效。也就是关闭浏览器后 Cookie 失效
 - 不设置
如果不设置失效时间,则默认当前会话有效。

2.2.2 cookie 的路径

- 1) Cookie 的路径指告诉浏览器访问那些地址时该携带该 Cookie,我们知道浏览器会保存很多不同网站的 Cookie,比如百度的 Cookie,新浪的 Cookie,腾讯的 Cookie 等等。那我们不可能访问百度的时候携带新浪的 Cookie,也不可能访问每个网站时都带上所有的 Cookie 这是不现实的,所以往往我们还需要为 Cookie 设置一个 Path 属性,来告诉浏览器何时携带该 Cookie。
- 2) 我们同过 Cookie 的 `setPath()` 来设置路径,这个路径是由浏览器来解析的所以/代表服务器的根目录。

如:

- 设置为 /项目名/路径 → `cookie.setPath("/项目名/路径")`
这样设置只有访问 “/项目名/路径” 下的的资源才会携带 Cookie

如：/项目名/路径/1.jsp 、 /项目名/路径/hello/2.jsp 等

- 如果不设置，默认会在访问“/项目名”下的资源时携带
如：“/项目名/index.jsp” 、 “/项目名/hello/index.jsp”

```
Cookie cookie = new Cookie("username", "abc");
cookie.setMaxAge(60*60*24); //秒为单位,一天后过期
cookie.setPath(getServletContext().getContextPath()+"/");
resp.addCookie(cookie);
resp.sendRedirect(getServletContext().getContextPath()+"/index.jsp");
```

2.3 读取 cookie

通过以上步骤，我们将 cookie 保存到了浏览器端。那么我们如何读取 cookie 中的值呢。

分析：

cookie 被设置进入浏览器后，每次请求都会携带 cookie 的值，所以我们需要从 request 中取出 cookie 进行解析。

```
//从request中获取所有cookie
Cookie[] cookies = request.getCookies();
//遍历cookie
for(Cookie c:cookies){
    String cName = c.getName();//获取cookie名
    String cValue = c.getValue();//获取cookie值
    System.out.println("cookie: " + cName + "=" + cValue);
}
```

第 13 章 Session

1. session 简介

session 是我们 jsp 九大隐含对象的一个对象。

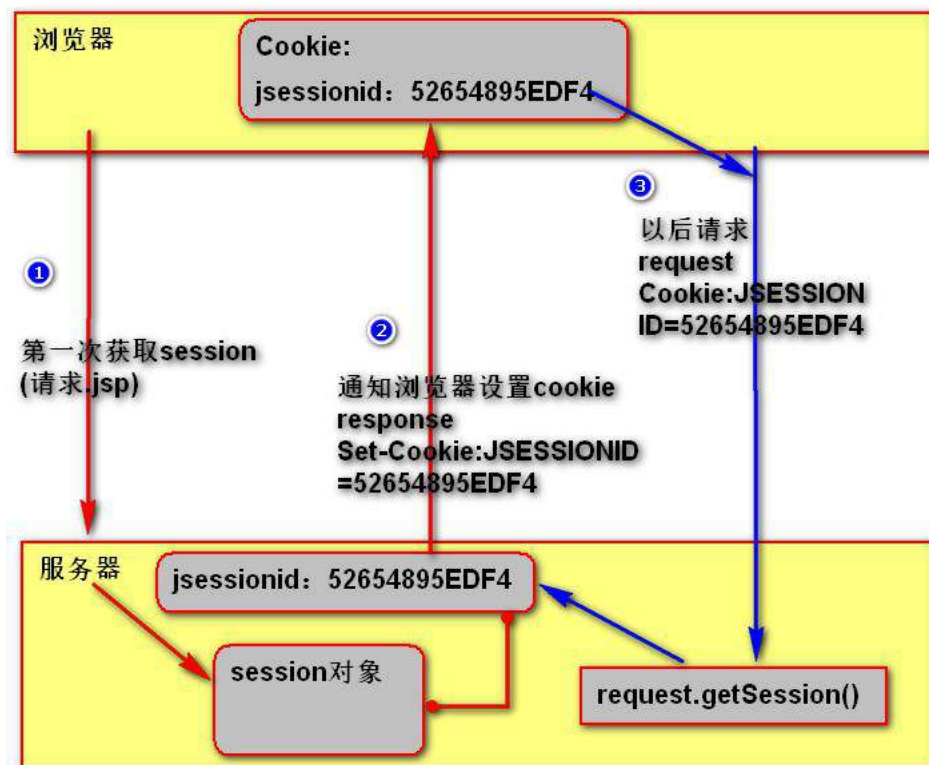
session 称作域对象，他的作用是保存一些信息，而 session 这个域对象是一次会话期间使用同一个对象。所以这个对象可以用来保存共享数据。

- 使用 Cookie 有一个非常大的局限，就是如果 Cookie 很多，则无形的增加了客户端与服务端的数据传输量。而且由于浏览器对 Cookie 数量的限制，注定我们不能再 Cookie 中保存过多的信息，于是 Session 出现。
- Session 的作用就是在服务器端保存一些用户的数据，然后传递给用户一个名字为 JSESSIONID 的 Cookie，这个 JSESSIONID 对应这个服务器中的一个 Session 对象，通过它可以获取到保存用户信息的 Session。

session 是基于 cookie 的。

在用户第一次使用 session 的时候（访问 jsp 页面会获取 session，所以一般访问 index.jsp 就算是第一次使用 session 了），服务器会为用户创建一个 session 域对象。使用 jsessionId 和这个对象关联，这个对象在整个用户会话期间使用。响应体增加 set-cookie:jsessionid=xxx 的项。用户下次以后的请求都会携带 jsessionId 这个参数，我们使用 request.getSession() 的时候，就会使用 jsessionId 取出 session 对象。

session 原理图：



2. session 使用

获取 session 对象

```
HttpSession session = request.getSession();
```

session 是我们的四大域对象之一。用来保存数据。常用的方法

```
session.setAttribute("user", new Object());
session.getAttribute("user");
session.setMaxInactiveInterval(60*60*24); //秒为单位
session.invalidate(); //使 session 不可用
```

2.1 Session 时效

1) 基本原则

Session 对象在服务器端不能长期保存，它是有时间限制的，超过一定时间没有被访问过的 Session 对象就应该释放掉，以节约内存。所以 Session 的有效时间并不是从创建对象开始计时，到指定时间后释放——而是从最后一次被访问开始计时，统计其“空闲”的时间。

2) 默认设置

在全局 web.xml 中能够找到如下配置：

```
<!-- ===== Default Session Configuration ===== -->
<!-- You can set the default session timeout (in minutes) for all newly -->
<!-- created sessions by modifying the value below. -->

<session-config>
    <session-timeout>30</session-timeout>
</session-config>
```

说明 Session 对象默认的最长有效时间为 30 分钟。

3) 手工设置

```
session.setMaxInactiveInterval(int seconds)
session.getMaxInactiveInterval()
```

4) 强制失效

```
session.invalidate()
```

5) 可以使 Session 对象释放的情况

Session 对象空闲时间达到了目标设置的最大值，自动释放

Session 对象被强制失效

Web 应用卸载

服务器进程停止

2.2 URL 重写

在整个会话控制技术体系中，保持 JSESSIONID 的值主要通过 Cookie 实现。但 Cookie 在浏览器端可能会被禁用，所以我们还需要一些备用的技术手段，例如：URL 重写。

1) URL 重写其实就是将 JSESSIONID 的值以固定格式附着在 URL 地址后面，以实现保持 JSESSIONID，进而保持会话状态。这个固定格式是：URL;jsessionid=xxxxxxxxx

例如：

```
targetServlet;jsessionid=F9C893D3E77E3E8329FF6BD9B7A09957
```

2) 实现方式：

```
response.encodeURL(String)
response.encodeRedirectURL(String)
```

例如：

```
//1.获取Session对象
HttpSession session = request.getSession();

//2.创建目标URL地址字符串
```

```
String url = "targetServlet";

//3.在目标URL地址字符串后面附加JSESSIONID的值
url = response.encodeURL(url);

//4.重定向到目标资源
response.sendRedirect(url);
```

2.3 Session 的活化和钝化

Session 机制很好的解决了 Cookie 的不足，但是当访问应用的用户很多时，服务器上就会创建非常多的 Session 对象，如果不对这些 Session 对象进行处理，那么在 Session 失效之前，这些 Session 一直都会在服务器的内存中存在。那么就，就出现了 Session 活化和钝化的机制。

1) Session 钝化:

Session 在一段时间内没有被使用时，会将当前存在的 Session 对象序列化到磁盘上，而不再占用内存空间。

2) Session 活化:

Session 被钝化后，服务器再次调用 Session 对象时，将 Session 对象由磁盘中加载到内存中使用。

如果希望 Session 域中的对象也能够随 Session 钝化过程一起序列化到磁盘上，则对象的实现类也必须实现 `java.io.Serializable` 接口。不仅如此，如果对象中还包含其他对象的引用，则被关联的对象也必须支持序列化，否则会抛出异常：`java.io.NotSerializableException`

3. 表单重复提交问题

3.1 什么是表单重复提交?

同一个表单中的数据内容多次提交到服务器。

危害:

服务器重复处理信息，负担加重。

如果是保存数据可能导致保存多份相同数据。

3.2 几种重复提交

1) 提交完表单后，直接刷新页面，会再次提交。

- 根本原因: Servlet 处理完请求以后，直接转发到目标页面。

- 这样整个业务，只发送了一次请求，那么当你在浏览器中点击刷新按钮或者狂按 f5 会一直都会刷新之前的请求

解决方案：使用重定向跳转到目标页面

2) 提交表单后，由于网速差等原因，服务器还未返回结果，连续点击提交按钮，会重复提交。

- 根本原因：按钮可以多次点击
- 解决方案：通过 js，使得按钮只能提交一次。

```
$("#form1").submit(function(){
    $("#sub_btn").prop("disabled",true);
})
```

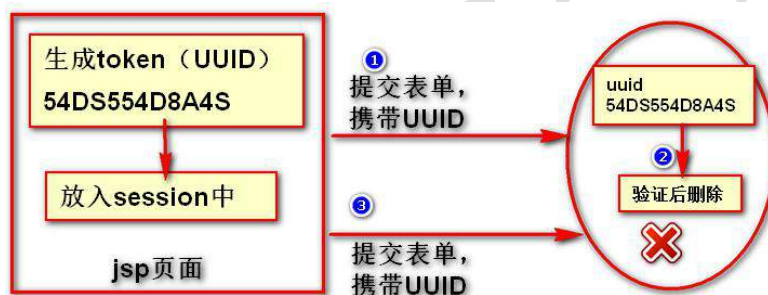
3) 表单提交后，点击浏览器回退按钮，不刷新页面，点击提交按钮再次提交表单

- 根本原因：服务器并不能识别请求是否重复。
- 解决方案：使用 token 机制。

- 1、页面生成时，产生一个唯一的 token 值。将此值放入 session
- 2、表单提交时，带上这个 token 值。
- 3、服务端验证 token 值存在，则提交表单，然后移除此值。

验证 token 不存在，说明是之前验证过一次被移除了，所以是重复请求。不予处理

原理：



代码：

➤ jsp 页面

```
<%
    String token = System.currentTimeMillis() + "";
    request.getSession().setAttribute(token, "");
%>
<div>
    <h1>测试表单重复提交</h1>
    <form action="/login" method="get">
        用户名: <input name="username" type="text"/>
        密码: <input name="password" type="password">
        <input name="token" value="<%=token%>" />
        <input type="submit">
    </form>
    <hr>
</div>
```

➤ Servlet

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    HttpSession session = request.getSession();
    String token = request.getParameter("token");
    Object attribute = session.getAttribute(token);
    response.setContentType("text/html;charset=UTF-8");
    if(attribute!=null){
        session.removeAttribute(token);
        response.getWriter().write("请求成功！");
    }else{
        response.getWriter().write("请不要重复请求！");
    }
}
```

其实防止重复提交的核心就是让服务器有一个字段能来识别此次请求是否已经执行。这个字段需要页面传递过来，因为只要回退回去的页面，字段都是一致的。不会变化，通过这个特性我们想到了 **token** 机制来防止重复提交

第 14 章 Filter

1. 提出问题

1、我们在访问后台很多页面时都需要登录，只有登录的用户才能查看这些页面，我们需要在每次请求的时候都检查用户是否登陆，这样做很麻烦，有没有一种方法可以在我们请求之前就帮我们做这些事情。有！

2、我们 web 应用经常会接收中文字符，由于可能导致中文乱码，我们每次都需要在方法的开始使用 `request.setCharacterEncoding("utf-8")`；能不能在我们要获取参数值直接就可以自己设置好编码呀。能！

这种问题的解决方法我们想到了一种办法。那就是在每次请求之前我们先将它拦截起来，当我们设置好一切东西的时候，再将请求放行。类似与我们地铁站的检票系统。每个人进站的时候必须刷卡，扣完钱后才可以进站坐车。

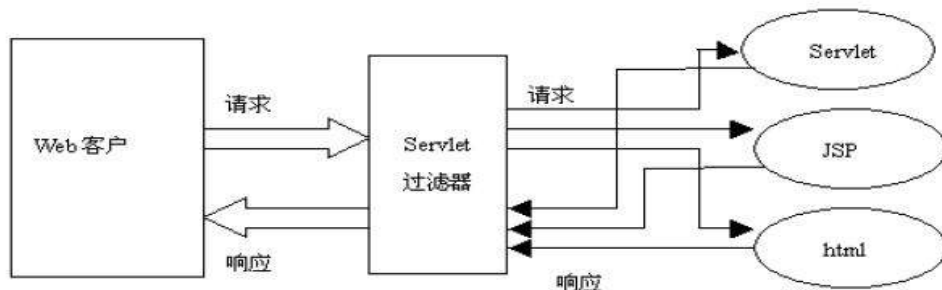
web 中也有这个机制，我们叫做过滤器。
就是我们接下来学习的 filter

2. Filter 简介

2.1 什么是 filter

- 1) Filter（过滤器）的基本功能是对 Servlet 容器调用 Servlet (JSP)的过程进行拦截，从而在 Servlet 处理请求前和 Servlet 响应请求后实现一些特殊的功能。
- 2) 在 Servlet API 中定义了三个接口类来供开发人员编写 Filter 程序：Filter, FilterChain, FilterConfig
- 3) Filter 程序是一个实现了 Filter 接口的 Java 类，与 Servlet 程序相似，它由 Servlet 容器进行调用和执行
- 4) Filter 程序需要在 web.xml 文件中进行注册和设置它所能拦截的资源：Filter 程序可以拦截 Jsp, Servlet, 静态图片文件和静态 html 文件

2.2 filter 的运行原理是什么



这个 Servlet 过滤器就是我们的 filter

- 1) 当在 web.xml 中注册了一个 Filter 来对某个 Servlet 程序进行拦截处理时, 这个 Filter 就成了 Tomcat 与该 Servlet 程序的通信线路上的一道关卡, 该 Filter 可以对 Servlet 容器发送给 Servlet 程序的请求和 Servlet 程序回送给 Servlet 容器的响应进行拦截, 可以决定是否将请求继续传递给 Servlet 程序, 以及对请求和相应信息是否进行修改
- 2) 在一个 web 应用程序中可以注册多个 Filter 程序, 每个 Filter 程序都可以对一个或一组 Servlet 程序进行拦截。
- 3) 若有多个 Filter 程序对某个 Servlet 程序的访问过程进行拦截, 当针对该 Servlet 的访问请求到达时, web 容器将把这多个 Filter 程序组合成一个 Filter 链(过滤器链)。Filter 链中各个 Filter 的拦截顺序与它们在应用程序的 web.xml 中映射的顺序一致

3. Filter-helloworld

3.1 Hello-World

filter 编写三步骤:

- 1、创建 filter 实现类, 实现 filter 接口
- 2、编写 web.xml 配置文件, 配置 filter 的信息
- 3、运行项目, 可以看到 filter 起作用了

代码:

```
//1、filter 实现类
public class MyFirstFilter implements Filter{
    @Override
```

```
public void init(FilterConfig filterConfig) throws ServletException {
    System.out.println("初始化方法");
}

@Override
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws
IOException, ServletException {
    System.out.println("doFilter方法");
}

@Override
public void destroy() {
    System.out.println("销毁方法...");
}
}

//2、web.xml 配置
<filter>
    <filter-name>MyFirstFilter</filter-name>
    <filter-class>com.atguigu.filter.MyFirstFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>MyFirstFilter</filter-name>
    <url-pattern>/index.jsp</url-pattern>
</filter-mapping>
//3、运行程序，发现 index.jsp 页面不显示了，后台输出“doFilter 方法”，说明我们写的 filter 执行了。
```

3.2 filter 的生命周期

- 1) 在服务器启动时，filter 被创建并初始化，执行 init()方法。
- 2) 请求通过 filter 时执行 doFilter 方法。
- 3) 服务器停止时，调用 destroy 方法。

3.3 filter 放行请求

我们发现，刚才的 filter 配置好后，index.jsp 页面没法访问了，访问这个页面的时候 filter 的 dofilter 方法被调用了。说明 dofilter 这个方法拦截了我们的请求。

我们如何显示页面呢。也就是如何将请求放行呢。我们观察发现有个 filterChain 被传入到这个方法里面了。filterChain 里面有个 doFilter()方法。

放行请求只需要调用 `filterChain` 的 `doFilter` 方法。

```
public void doFilter(ServletRequest request, ServletResponse response,
    FilterChain chain) throws IOException, ServletException {
    System.out.println("doFilter 方法");
    chain.doFilter(request, response); //放行请求
}
```

3.4 filter 拦截原理

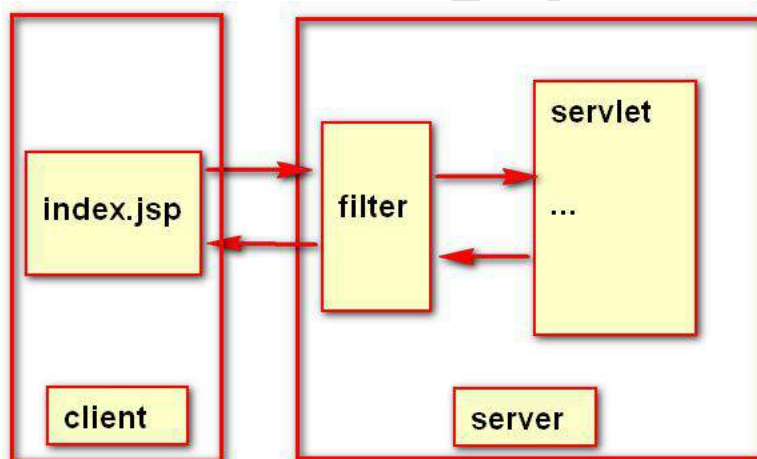
我们在 `chain.doFilter(request, response);` 方法后也写一句话，`System.out.println("doFilter 方法执行后...")`，在 `index.jsp` 页面也写上 `jsp` 脚本片段，输出我是 `jsp` 页面。运行程序发现控制台输出了这几句话：

doFilter 方法...

我是 jsp 页面

doFilter 方法后...

我们不难发现 `filter` 的运行流程



4. FilterChain

`doFilter(ServletRequest request, ServletResponse response, FilterChain chain)`

在 `doFilter` 执行之前，由容器将 `filterChain` 对象传入方法。调用此对象的 `doFilter()` 方法可以将请求放行，实际上是执行过滤器链中的下一个 `doFilter` 方法，但是如果只有一个过滤器，则为放行。

5. FilterConfig

FilterConfig 类似 ServletConfig，是 filter 的配置信息对象。FilterConfig 对象具有以下方法。

- `getFilterName() : String` - FilterConfig
- `getInitParameter(String name) : String` - FilterConfig
- `getInitParameterNames() : Enumeration<String>` - FilterConfig
- `getServletContext() : ServletContext` - FilterConfig

getFilterName(): 获取当前 filter 的名字。获取的是在 web.xml 中配置的 filter-name 的值

getInitParameter(String name): 获取 filter 的初始化参数。在 web.xml 中配置

```
<filter>
  <filter-name>BBSFilter</filter-name>
  <filter-class>com.atguigu.filter.BBSFilter</filter-class>
  <init-param>
    <param-name>user</param-name>
    <param-value>admin</param-value>
  </init-param>
</filter>
```

getInitParameterNames(): 获取 filter 初始化参数名的集合。

getServletContext(): 获取当前 web 工程的 ServletContext 对象。

6. Filter 的 url-pattern

url-pattern 是配置 filter 过滤哪些请求的。主要有以下几种配置：

web.xml 中配置的/都是以当前项目路径为根路径的

1) 精确匹配：

 /index.jsp/user/login 会在请求/index.jsp、/user/login 的时候执行过滤方法

2) 路径匹配：

 /user/* /* 凡是路径为/user/下的所有请求都会被拦截，/*表示拦截系统的所有请求，包括静态资源文件。

3) 扩展匹配：

 .jsp.action 凡是后缀名为.jsp .action 的请求都会被拦截。

注意：/login/*.jsp 这种写法是错误的，只能是上述三种的任意一种形式。不能组合新形式。

*jsp 也是错误的，扩展匹配必须是后缀名

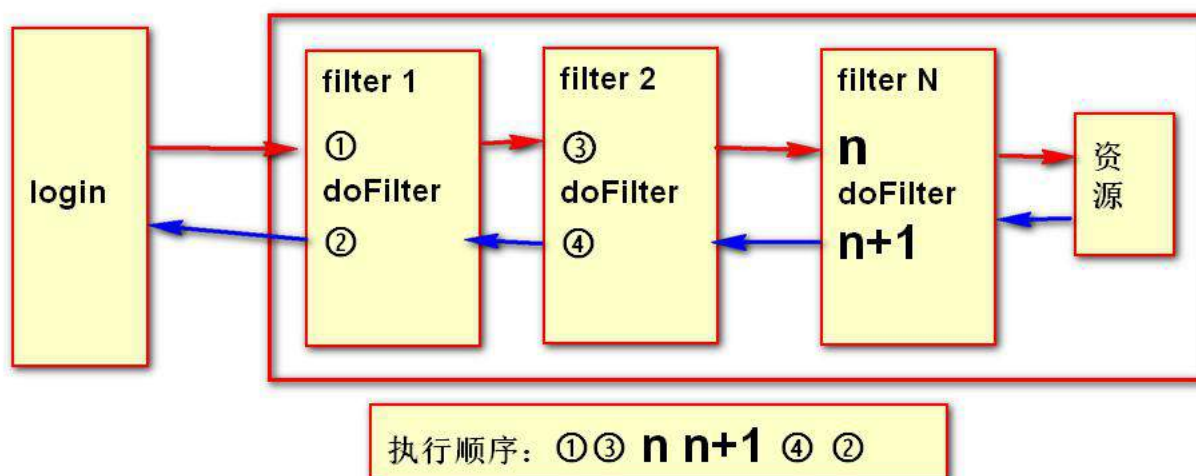
4) 多重 url-pattern 配置

上面的三种形式比较有局限性，但是 url-pattern 可以配置多个，这样这三种组合基本就能解决所有问题了

7. 多 Filter 执行顺序

如果同一个资源有多个 filter 都对其拦截，则拦截的顺序是按照 web.xml 中配置的顺序进行的

执行流程图如下



请求总是在处理之后再回来执行 doFilter 之后的方法。

第 15 章 Listener

1. Listener 简介

1) 什么是监听器:

专门用来对其他对象身上发生的事情或状态改变进行监听和相应处理的对象,当被监视对象发生情况时,立即采取相应的行动

2) 作用:

监听 web 变化:Servlet 规范中定义的一种特殊类,用于监听 web'应用程序中的 ServletContext, HttpSession, ServletRequest 等域对象的创建与销毁,以及属性变化

2. 八大 listener

我们 Servlet 规范中定义了很多种类型的监听器,他们有着各自不同的功能。我们来看一下这些监听器。

2.1 ServletContextListener:

ServletContext 生命周期监听器,创建和销毁时被触发。包含两个方法:

contextInitialized(ServletContextEvent): ServletContext 创建事件监听

contextDestroyed(ServletContextEvent): ServletContext 销毁事件监听

2.2 ServletContextAttributeListener:

ServletContext 属性监听器，发生属性变化时触发。

attributeAdded(ServletContextAttributeEvent): 增加属性时调用

attributeRemoved(ServletContextAttributeEvent): 移除属性时调用

attributeReplaced(ServletContextAttributeEvent): 属性替换时调用

2.3 HttpSessionListener:

HttpSession 生命周期监听器，创建和销毁 session 时被触发。

sessionCreated(HttpSessionEvent): session 创建时触发

sessionDestroyed(HttpSessionEvent): session 销毁时触发，invalid，session 过期。

2.4 HttpSessionAttributeListener:

HttpSession 属性监听器，session 中属性变化时触发

attributeAdded(HttpSessionBindingEvent): 属性添加时调用

attributeRemoved(HttpSessionBindingEvent): 属性移除时调用

attributeReplaced(HttpSessionBindingEvent): 属性替换时使用

2.5 HttpSessionActivationListener:

HttpSession 活化与钝化监听，不需要在 web.xml 中配置，必须是类实现此接口

sessionWillPassivate(HttpSessionEvent): 某个类和 session 一起钝化时调用

sessionDidActivate(HttpSessionEvent): 某个类和 session 一起活化时调用

2.6 HttpSessionBindingListener:

HttpSession 属性绑定时调用，不需要在 web.xml 中配置，必须是类实现此接口。当此类被当做属性值绑定到 session 中时触发

valueBound(HttpSessionBindingEvent): 属性被设置到 session 域中时触发

valueUnbound(HttpSessionBindingEvent): 属性从 session 域中移除时触发

2.7 ServletRequestListener

ServletRequest 生命周期监听器。ServletRequest 创建和销毁时触发

requestInitialized(ServletRequestEvent): 请求初始化时触发

requestDestroyed(ServletRequestEvent): 请求完成时触发

2.8 ServletRequestAttributeListener

ServletRequest 属性监听器。ServletRequest 中属性变化时触发

attributeAdded(ServletRequestAttributeEvent): request 中增加属性调用

attributeRemoved(ServletRequestAttributeEvent): request 中移除属性调用

attributeReplaced(ServletRequestAttributeEvent): request 中属性替换调用

3. Listener 分类

以上的八个监听器，我们发现很类似，除过两个特殊的外，都是属性监听器，和生命周期监听器。所以我们将以上的监听器划分为两种类型。

1) 生命周期监听器:

ServletContextListener、HttpSessionListener、ServletRequestListener

2) 属性监听器:

ServletContextAttributeListener、HttpSessionAttributeListener、ServletRequestAttributeListener

3) 特殊的两个监听器:

HttpSessionActivationListener、HttpSessionBindingListener

4. Listener 使用

1) 生命周期、属性监听器的使用

```
//创建相应的实现类:
public class MyFirstListener implements ServletRequestListener {
    public MyFirstListener() {
    }
    public void requestDestroyed(ServletRequestEvent sre) {
        System.out.println("requestDestroyed");
    }
    public void requestInitialized(ServletRequestEvent sre) {
        System.out.println("requestInitialized");
    }
}

//配置web.xml
<listener>
    <listener-class>com.atguigu.MyFirstListener</listener-class>
</listener>
```

2) 两个特殊监听器的使用

创建相应的监听器实现类。


```
public class Student implements HttpSessionBindingListener{
    @Override
    public void valueBound(HttpSessionBindingEvent event) {
        System.out.println("valueBound");
    }
    @Override
    public void valueUnbound(HttpSessionBindingEvent event) {
        System.out.println("valueUnbound");
    }
}
```

5. 生命周期监听器

ServletContextListener、HttpSessionListener、ServletRequestListener

1) ServletContextListener:

web 容器启动的时候 ServletContext 就会被创建，当 web 服务器关闭的时候这个对象被销毁

2) HttpSessionListener:

在第一次使用 session 的时候，session 会被创建，服务器关闭，session 并不会被销毁，而是钝化了。只有显式的调用 invalid 方法，或者是 session 过期，session 才会被销毁。

3) ServletRequestListener:

每个新的请求都会创建 request 对象，触发初始化方法。当一次请求完成时，request 对象被销毁，触发销毁方法。

6. 属性监听器

ServletContextAttributeListener、HttpSessionAttributeListener、ServletRequestAttributeListener 都是域属性变化时触发的。

7. 特殊的两个监听器

HttpSessionActivationListener、HttpSessionBindingListener

这两个监听器需要具体的实现类，只要实现这两个监听器即可，不用在 web.xml 中配置。

HttpSessionActivationListener 监听对象随 session 钝化活化的过程，这个对象要被钝化必须实现 serializable 接口。否则不能钝化和活化

第 16 章 AJAX

1. 什么是 ajax

- 1) AJAX 是 Asynchronous JavaScript And XML 的简称。直译为，异步的 JS 和 XML。
- 2) 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。
- 3) AJAX 也可以简单的理解为通过 JS 向服务器发送请求。
- 4) 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。
- 5) 异步处理：
 - a) 同步处理：
 - i. 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。
 - ii. 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。
 - b) 异步处理：
 - i. 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。
 - ii. 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。

2. 使用 XMLHttpRequest 对象发送请求

js 中定义了一个可以发送异步请求的对象 XMLHttpRequest。我们在网页中使用这个对象发送请求。我们来学习一下 XMLHttpRequest 对象。

2.1 创建 XMLHttpRequest 对象

由于这个对象并不是标准，但是基本所有浏览器都支持，所以针对不同的浏览器我们需要创建其对应的对象。

- 1) w3c 标准浏览器，火狐、谷歌等

```
var xhr = new XMLHttpRequest();
```

2) ie6 创建方式

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

3) ie5.5 创建方式

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

所以为了兼容所有浏览器，我们可以创建一个公共的获取此对象的方法

//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;  
}
```

2.2 使用 XMLHttpRequest

1) open(method,url,async)

open()用于设置请求的基本信息，接收三个参数。

- ◆ method
请求的方法：get 或 post
接收一个字符串
- ◆ url
请求的地址，接收一个字符串
- ◆ Async
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

- 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

请求体参数使用键值对的形式，多个参数用&分割

如：“username=lll&password=lll”

3) `setRequestHeader(header,value)`

用于设置请求头

`header` 参数

字符串类型，要设置的请求头的名字

`value` 参数

字符串类型，要设置的请求头的值

post 请求的时候需要设置

```
xhr.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
```

2.3 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 页面未找到
- 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。

2.4 使用 js 发送 ajax 请求示例

1) 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);
        }
    }
};
```

2) 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);
        }
    }
};
```

3. json 数据

js 对 xml 数据的操控性不是很好。例如，返回这段 xml

```
<student><name>tomcat</name><age>20</age></student>
```

js 的处理如下：

```
var ele = xhr.responseXml;
```

```
var nodeVal = ele.getElementsByTagName("name").firstChild.nodeValue;
```

这种处理方式很麻烦。而且 xml 虽然看起来很清晰，但是标签占据了基本 1/3 内容，比较浪费流量。

json 就能解决上述问题。js 也是原生支持 json。操作很方便。

3.1 什么是 json

json (JavaScript Object Notation (javascript 对象表示法))，是一种轻量级的数据交换格式。有着自己独立简单的语法。

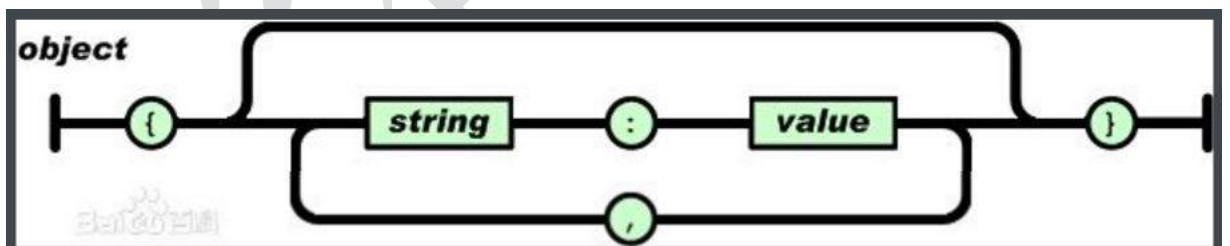
json 经常被用来传递数据，他的数据信息集中，占用空间少，解析方便。

json 的本质就是 js 对象。js 中我们创建对象可以直接进行属性赋值操作,stu.name="xx";

3.2 json 的语法规则

用键值对表示数据、数据由逗号分隔、花括号保存对象、方括号保存数组。

键值对的写法是： 键:值



如：{"firstName":"Brett","lastName":"McLaughlin","email":"aaaa"}

3.3 json 的值可取范围

json 的值可以是：

数字、字符串（使用双引号包裹）、逻辑值、数组（方括号中）、对象（在花括号中）、null.

```
{
  "people":[
    {"firstName":"Brett","lastName":"McLaughlin","email":"aaaa"},

```

```
        {"firstName":"Jason","lastName":"Hunter","email":"bbbb"},
        {"firstName":"Elliott","lastName":"Harold","email":"cccc"}
    ]
}
```

上面这个 json 表示的意思是。people 的值是一个对象数组，里面有三个对象。每个对象里面有三个不同的属性。

4. js 中的 json 使用

1) 创建 json 对象

```
var json1 = {"name1":"value1","name2":"value2" , "name3":[1,"str",true]};
var json2 = [{"name1":"value1"}, {"name2":"value2"}];
```

注意：创建的是 json 对象，不是字符串，不能在最前面和最后面加双引号

2) 获取/设置 json 对象属性值。

获取：

```
alert(json1.name1)//value1
alert(json2[0].name1)//value1
```

设置：

```
json1.name1="你好";
json2[0].name1="Hello";
```

3) js 中 json 对象与字符串的互转

将 json 对象转为字符串

JSON.stringify(JSON 对象)

将字符串转换为 json 对象

JSON.parse(JSON 字符串)

5. java 中操作 json

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。

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。

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

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。

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);
```

我们在以后的使用中，会经常使用 json 来传递数据，xml 这种方式基本上已经不用来传递数据了。所以我们以后的 ajax 请求获取到的数据多为 json 格式，我们要熟练掌握 js 中 json 的解析操作。

6. jquery 中的 ajax

jquery 使得 js 变得更加简单易用。那么在 jquery 中是如何发送 ajax 请求的呢？

主要讲解 jquery 以下的几个方法。

\$.get()、\$.post()、\$.getJSON()、\$.ajax();

6.1 \$.get();

jquery 中发送 get 请求的方法

方法签名:

`$.get(url, [data], [callback], [type])`

[]表示参数可选

参数解析:

url:发送的请求地址

data:待发送 Key/value 参数。

callback:请求成功时回调函数。

type:返回内容格式, xml, html, script, json, text, _default。

发送示例:

```
$.get("user?method=login",{username:"lll",password:"lll"},function(data){  
    alert(data.errCode);  
},"json");
```

6.2 \$.post();

jquery 中发送 post 请求的方法。

方法签名:

`$.post(url, [data], [callback], [type])`

[]表示可选参数

参数解析:

url:发送请求地址。

data:待发送 Key/value 参数。

callback:发送成功时回调函数。

type:返回内容格式, xml, html, script, json, text, _default。

发送示例:

```
$.post("user?method=login",{username:"lll",password:"lll"},function(data){  
    alert(data.errCode);  
},"json");
```

6.3 \$.getJSON();

jquery 中返回 json 数据的 get 请求

方法签名:

`$.getJSON(url, [data], [callback])`

[]表示可选参数

参数解析:

url:发送请求地址。

data:待发送 Key/value 参数。

callback:载入成功时回调函数。

发送示例:

```
$.getJSON("user?method=login",{username:"lll",password:"lll"},function(data){
    alert(data.errCode);
})
```

6.4 \$.ajax()

jquery 中底层的 ajax 请求方法，可以设置详细的参数

方法签名:

`$.ajax(url,[settings])`

[]表示可选参数

参数解析:

url:发送请求地址。

settings:其他详细设置。可设置项参加 jquery 文档

发送示例:

```
$.ajax({
    type: "POST",
    url: "some.php",
    data: "name=John&location=Boston",
    success: function(msg){
        alert( "Data Saved: " + msg );
    };
    error:function(){
        alert("请求失败")
    }
})
```

```
});
```

第 17 章文件上传下载

1. 文件上传

1.1 提出问题

客户需要将自己的文件上传给服务器怎么处理？
文件上传的时候是怎样提交给服务器的？

答：我们可以使用 `multipart/form-data` 表单处理文件上传
文件的上传是以流的形式提交给服务器的

1.2 文件上传步骤

文件上传我们需要用到两个包。`commons-fileupload-1.3.1.jar`、`commons-io-2.4.jar`。
所以先导入包。然后进行文件上传测试
文件上传步骤：

1) 修改表单 `enctype` 值为 `multipart/form-data`，并且 `method=post`

```
<form action="upload" method="post" enctype="multipart/form-data">  
    <input type="file" name="file">  
    <input type="submit">  
</form>
```

2) 创建文件上传请求解析器

在 `doPost` 方法中创建 `ServletFileUpload` 对象，并传入 `DiskFileItemFactory` 工厂对象

```
DiskFileItemFactory factory = new DiskFileItemFactory();  
ServletFileUpload fileUpload = new ServletFileUpload(factory);
```

3) 解析并上传文件

解析上传文件请求

```
List<FileItem> list = fileUpload.parseRequest(request);  
for(FileItem f : list){  
    if(f.isFormField()){//普通表单项
```

```
        }else{ //文件表单项
            String name = f.getName();
            System.out.println(name);
            f.write(new File("D:"+name));
        }
    }
}
```

注意事项:

①、当文件上传表单的文件项未选择文件时，文件大小为 0，应该做判断，当文件大小为 0 则不进行上传

②、使用 `fileItem.getName()` 获取文件名。在火狐谷歌下，为文件的名字。在 ie 下为文件的路径，因此要获取文件名需要进行处理

③、使用 `fileItem.getString("utf-8")`，获取文件普通表单项的值

完整的文件上传代码:

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    // 1、创建fileItem工厂
    DiskFileItemFactory factory = new DiskFileItemFactory();
    // 2、创建文件上传对象
    ServletFileUpload fileUpload = new ServletFileUpload(factory);
    // 3、解析文件上传请求
    try {
        List<FileItem> list = fileUpload.parseRequest(request);
        // 4、遍历解析出的每个FileItem文件
        for (FileItem f : list) {
            // 这是普通表单项
            if (f.isFormField()) {
                // 表单项的name值
                String fieldName = f.getFieldName();
                // 表单项value，使用编码取得中文
                String fieldValue = f.getString("utf-8");
                System.out.println(fieldName + ":" + fieldValue);

                // 表示文件表单项
            } else {
                //获取文件大小，有可能并没有上传文件
                long size = f.getSize();
                if(size == 0){
                    continue;
                }
            }
        }
    }
}
```

```
// 获取文件名字
String name = f.getName();
// ie下是全路径，火狐谷歌为文件名，需要不同处理
if (name.contains("\\")) {
    name = name.substring(name.lastIndexOf("\\") + 1);
}

//获取文件保存的服务端路径
String realPath = request.getServletContext().getRealPath("/upload");
File file = new File(realPath);
if(!file.exists()){
    file.mkdirs();
}
//生成一个唯一不重复的文件名
name = UUID.randomUUID().toString().replace("-", "")+"-"+name;
//将文件内容写入
f.write(new File(realPath+"/"+name));
}
}
} catch (FileUploadException e) {
    e.printStackTrace();
} catch (Exception e) {
    e.printStackTrace();
}
response.setContentType("text/html;charset=UTF-8");
response.getWriter().write("上传完成！");
}
```

其他注意：

fileUpload 可以来限制文件的大小

```
//设置当文件的大小为 50KB
```

```
//fileUpload.setFileSizeMax(1024*50);
```

```
//设置多个文件的总大小为 150KB
```

```
fileUpload.setSizeMax(1024*150);
```

当文件超出大小限制的时候会抛出异常，增加 catch 子句来处理

```
catch (SizeLimitExceededException e) {
    response.setContentType("text/html;charset=UTF-8");
    response.getWriter().write("文件大小超出限制");
    return;
}
```

原理：

分析文件上传的请求我们会发现，文件上传的时候 request 的 body 会增加一个分界符，分界符用来分割正常的表单域和文件表单域，文件的内容在分界符的中间，以流的形式传递，文件上传的 request 请求如下：

```
▼ Request Headers    view parsed
POST /07_Listener/upload HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Content-Length: 22757
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Origin: http://localhost:8080
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/48.0.2564.116 Safari/537.36
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryc1ITildLK1M7OQfk
Referer: http://localhost:8080/07_Listener/index.jsp
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.8
Cookie: JSESSIONID=3AEE76312C6A75EEF75F13D5CDB344AD
```

以上是请求头：

下面是请求体携带的数据：

```
-----WebKitFormBoundary0svVE7HD8NeS9aUZ
Content-Disposition: form-data; name="username"

fdsf
-----WebKitFormBoundary0svVE7HD8NeS9aUZ
Content-Disposition: form-data; name="password"

fdsfds
-----WebKitFormBoundary0svVE7HD8NeS9aUZ
Content-Disposition: form-data; name="file"; filename="web讲义 - 复习.docx"
Content-Type: application/vnd.openxmlformats-officedocument.wordprocessingml.document

-----WebKitFormBoundary0svVE7HD8NeS9aUZ--
```

2. 文件下载

2.1 提出问题

如何把服务器上的文件传输给客户端。href 可以吗？href 可以传送某些资源，当客户端不支持此资源的显示或者打开时，就会启动下载。但是如果客户端支持显示或者打开此资源。如：图片，视频，音频，文本文件等。浏览器会直接打开他们。我们需要浏览器进行下载操作。所以单纯的 href 已经不能操作了。

如果资源文件是比较重要的文件，我们需要放到 WEB-INF 下将其保护，那么 href 是链接不到的。

2.2 文件下载步骤

2.2.1 下载步骤:

1) 设置响应体

```
response.setContentType(this.getServletContext().getMimeType("abc.txt"));  
response.setHeader("Content-Disposition", "attachment;filename="+ "abc.txt");
```

2) 输出文件流

```
FileInputStream fis = new  
FileInputStream(this.getServletContext().getRealPath("/abc.txt"));  
ServletOutputStream stream = response.getOutputStream();  
IOUtils.copy(fis, stream); //使用commons.io工具输出流  
fis.close();
```

2.2.2 存在的问题:

1) 文件名为中文

➤ 使用 `URLEncoder.encode` 方法将文件名进行编码

```
String fileName = URLEncoder.encode("中文.txt", "utf-8");  
response.setHeader("Content-Disposition", "attachment;filename="+fileName);  
火狐不支持, 显示格式不正确
```

火狐需要使用 base64 编码, 加上 utf-8 标注。

```
String fileName = "";  
String header = request.getHeader("User-Agent");  
if(header.contains("Firefox")){  
    fileName = "?utf-8?b?" + new  
        BASE64Encoder().encode(name.getBytes("utf-8")) + "?=";  
}else{  
    fileName = URLEncoder.encode("中文.txt", "utf-8");  
}
```

➤ 先用 GBK 解码, 再使用 iso8859-1 编码

```
String fileName = new String(name.getBytes("gbk"), "iso8859-1");  
response.setHeader("Content-Disposition", "attachment;filename="+fileName);
```

尚硅谷