# JSP 动作元素

与JSP指令元素不同的是，JSP动作元素在请求处理阶段起作用。JSP动作元素是用XML语法写成的。

利用JSP动作可以动态地插入文件、重用JavaBean组件、把用户重定向到另外的页面、为Java插件生成HTML代码。

动作元素基本上都是预定义的函数，JSP规范定义了一系列的标准动作，它用JSP作为前缀，可用的标准动作元素如下：

|  |  |
| --- | --- |
| **语法** | **描述** |
| jsp:include | 在页面被请求的时候引入一个文件。 |
| jsp:useBean | 寻找或者实例化一个JavaBean。 |
| jsp:setProperty | 设置JavaBean的属性。 |
| jsp:getProperty | 输出某个JavaBean的属性。 |
| jsp:forward | 把请求转到一个新的页面。 |
| jsp:plugin | 根据浏览器类型为Java插件生成OBJECT或EMBED标记。 |
| jsp:element | 定义动态XML元素 |
| jsp:attribute | 设置动态定义的XML元素属性。 |
| jsp:body | 设置动态定义的XML元素内容。 |
| jsp:text | 在JSP页面和文档中使用写入文本的模板 |

## 常见的属性

所有的动作要素都有两个属性：id属性和scope属性。

* **id属性：**

id属性是动作元素的唯一标识，可以在JSP页面中引用。动作元素创建的id值可以通过PageContext来调用。

* **scope属性：**

该属性用于识别动作元素的生命周期。 id属性和scope属性有直接关系，scope属性定义了相关联id对象的寿命。 scope属性有四个可能的值： (a) page, (b)request, (c)session, 和 (d) application。

## <jsp:include>动作元素

<jsp:include>动作元素用来包含静态和动态的文件。该动作把指定文件插入正在生成的页面。语法格式如下：

<jsp:include page="相对 URL 地址" flush="true" />

　前面已经介绍过include指令，它是在JSP文件被转换成Servlet的时候引入文件，而这里的jsp:include动作不同，插入文件的时间是在页面被请求的时候。

以下是include动作相关的属性列表。

|  |  |
| --- | --- |
| **属性** | **描述** |
| page | 包含在页面中的相对URL地址。 |
| flush | 布尔属性，定义在包含资源前是否刷新缓存区。 |

### **实例**

以下我们定义了两个文件 **date.jsp** 和 **main.jsp**，代码如下所示：

date.jsp文件代码：

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%><p>

今天的日期是: <%= (new java.util.Date()).toLocaleString()%></p>

main.jsp文件代码：

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%><!DOCTYPE html><html><head><meta charset="utf-8"><title>菜鸟教程(runoob.com)</title></head><body>

<h2>include 动作实例</h2><jsp:include page="date.jsp" flush="true" />

</body></html>

现在将以上两个文件放在服务器的根目录下，访问main.jsp文件。显示结果如下：

include 动作实例

今天的日期是: 2016-6-25 14:08:17

JSP 教程

[JSP 教程](https://www.runoob.com/jsp/jsp-tutorial.html" \o "JSP 教程" \t "https://www.runoob.com/jsp/_top)[JSP 简介](https://www.runoob.com/jsp/jsp-intro.html" \o "JSP 简介" \t "https://www.runoob.com/jsp/_top)[JSP 开发环境搭建](https://www.runoob.com/jsp/jsp-setup.html" \o "JSP 开发环境搭建" \t "https://www.runoob.com/jsp/_top)[Eclipse JSP/Servlet](https://www.runoob.com/jsp/eclipse-jsp.html" \o "Eclipse JSP/Servlet" \t "https://www.runoob.com/jsp/_top)[JSP 结构](https://www.runoob.com/jsp/jsp-architecture.html" \o "JSP 结构" \t "https://www.runoob.com/jsp/_top)[JSP 生命周期](https://www.runoob.com/jsp/jsp-life-cycle.html" \o "JSP 生命周期" \t "https://www.runoob.com/jsp/_top)[JSP 语法](https://www.runoob.com/jsp/jsp-syntax.html" \o "JSP 语法" \t "https://www.runoob.com/jsp/_top)[JSP 指令](https://www.runoob.com/jsp/jsp-directives.html" \o "JSP 指令" \t "https://www.runoob.com/jsp/_top)**[JSP 动作元素](https://www.runoob.com/jsp/jsp-actions.html" \o "JSP 动作元素" \t "https://www.runoob.com/jsp/_top)**[JSP 隐式对象](https://www.runoob.com/jsp/jsp-implicit-objects.html" \o "JSP 隐式对象" \t "https://www.runoob.com/jsp/_top)[JSP 客户端请求](https://www.runoob.com/jsp/jsp-client-request.html" \o "JSP 客户端请求" \t "https://www.runoob.com/jsp/_top)[JSP 服务器响应](https://www.runoob.com/jsp/jsp-server-response.html" \o "JSP 服务器响应" \t "https://www.runoob.com/jsp/_top)[JSP HTTP 状态码](https://www.runoob.com/jsp/jsp-http-status-codes.html" \o "JSP HTTP 状态码" \t "https://www.runoob.com/jsp/_top)[JSP 表单处理](https://www.runoob.com/jsp/jsp-form-processing.html" \o "JSP 表单处理" \t "https://www.runoob.com/jsp/_top)[JSP 过滤器](https://www.runoob.com/jsp/jsp-writing-filters.html" \o "JSP 过滤器" \t "https://www.runoob.com/jsp/_top)[JSP Cookie 处理](https://www.runoob.com/jsp/jsp-cookies.html" \o "JSP Cookie 处理" \t "https://www.runoob.com/jsp/_top)[JSP Session](https://www.runoob.com/jsp/jsp-session.html" \o "JSP Session" \t "https://www.runoob.com/jsp/_top)[JSP 文件上传](https://www.runoob.com/jsp/jsp-file-uploading.html" \o "JSP 文件上传" \t "https://www.runoob.com/jsp/_top)[JSP 日期处理](https://www.runoob.com/jsp/jsp-handling-date.html" \o "JSP 日期处理" \t "https://www.runoob.com/jsp/_top)[JSP 页面重定向](https://www.runoob.com/jsp/jsp-page-redirect.html" \o "JSP 页面重定向" \t "https://www.runoob.com/jsp/_top)[JSP 点击量统计](https://www.runoob.com/jsp/jsp-hits-counter.html" \o "JSP 点击量统计" \t "https://www.runoob.com/jsp/_top)[JSP 自动刷新](https://www.runoob.com/jsp/jsp-auto-refresh.html" \o "JSP 自动刷新" \t "https://www.runoob.com/jsp/_top)[JSP 发送邮件](https://www.runoob.com/jsp/jsp-sending-email.html" \o "JSP 发送邮件" \t "https://www.runoob.com/jsp/_top)

## **JSP 高级教程**

[JSP 标准标签库（JSTL）](https://www.runoob.com/jsp/jsp-jstl.html" \o "JSP 标准标签库（JSTL）" \t "https://www.runoob.com/jsp/_top)[JSP 连接数据库](https://www.runoob.com/jsp/jsp-database-access.html" \o "JSP 连接数据库" \t "https://www.runoob.com/jsp/_top)[JSP XML 数据处理](https://www.runoob.com/jsp/jsp-xml-data.html" \o "JSP XML 数据处理" \t "https://www.runoob.com/jsp/_top)[JSP JavaBean](https://www.runoob.com/jsp/jsp-javabean.html" \o "JSP JavaBean" \t "https://www.runoob.com/jsp/_top)[JSP 自定义标签](https://www.runoob.com/jsp/jsp-custom-tags.html" \o "JSP 自定义标签" \t "https://www.runoob.com/jsp/_top)[JSP 表达式语言](https://www.runoob.com/jsp/jsp-expression-language.html" \o "JSP 表达式语言" \t "https://www.runoob.com/jsp/_top)[JSP 异常处理](https://www.runoob.com/jsp/jsp-exception-handling.html" \o "JSP 异常处理" \t "https://www.runoob.com/jsp/_top)[JSP 调试](https://www.runoob.com/jsp/jsp-debugging.html" \o "JSP 调试" \t "https://www.runoob.com/jsp/_top)[JSP 国际化](https://www.runoob.com/jsp/jsp-internationalization.html" \o "JSP 国际化" \t "https://www.runoob.com/jsp/_top)

[JSP 指令](https://www.runoob.com/jsp/jsp-directives.html" \o "JSP 指令)

[JSP 隐式对象](https://www.runoob.com/jsp/jsp-implicit-objects.html" \o "JSP 隐式对象)

# JSP 动作元素

与JSP指令元素不同的是，JSP动作元素在请求处理阶段起作用。JSP动作元素是用XML语法写成的。

利用JSP动作可以动态地插入文件、重用JavaBean组件、把用户重定向到另外的页面、为Java插件生成HTML代码。

动作元素只有一种语法，它符合XML标准：

<jsp:action\_name attribute="value" />

动作元素基本上都是预定义的函数，JSP规范定义了一系列的标准动作，它用JSP作为前缀，可用的标准动作元素如下：

|  |  |
| --- | --- |
| **语法** | **描述** |
| jsp:include | 在页面被请求的时候引入一个文件。 |
| jsp:useBean | 寻找或者实例化一个JavaBean。 |
| jsp:setProperty | 设置JavaBean的属性。 |
| jsp:getProperty | 输出某个JavaBean的属性。 |
| jsp:forward | 把请求转到一个新的页面。 |
| jsp:plugin | 根据浏览器类型为Java插件生成OBJECT或EMBED标记。 |
| jsp:element | 定义动态XML元素 |
| jsp:attribute | 设置动态定义的XML元素属性。 |
| jsp:body | 设置动态定义的XML元素内容。 |
| jsp:text | 在JSP页面和文档中使用写入文本的模板 |

## 常见的属性

所有的动作要素都有两个属性：id属性和scope属性。

* **id属性：**

id属性是动作元素的唯一标识，可以在JSP页面中引用。动作元素创建的id值可以通过PageContext来调用。

* **scope属性：**

该属性用于识别动作元素的生命周期。 id属性和scope属性有直接关系，scope属性定义了相关联id对象的寿命。 scope属性有四个可能的值： (a) page, (b)request, (c)session, 和 (d) application。

## <jsp:include>动作元素

<jsp:include>动作元素用来包含静态和动态的文件。该动作把指定文件插入正在生成的页面。语法格式如下：

<jsp:include page="相对 URL 地址" flush="true" />

　前面已经介绍过include指令，它是在JSP文件被转换成Servlet的时候引入文件，而这里的jsp:include动作不同，插入文件的时间是在页面被请求的时候。

以下是include动作相关的属性列表。

|  |  |
| --- | --- |
| **属性** | **描述** |
| page | 包含在页面中的相对URL地址。 |
| flush | 布尔属性，定义在包含资源前是否刷新缓存区。 |

### **实例**

以下我们定义了两个文件 **date.jsp** 和 **main.jsp**，代码如下所示：

date.jsp文件代码：

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%><p>

今天的日期是: <%= (new java.util.Date()).toLocaleString()%></p>

main.jsp文件代码：

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%><!DOCTYPE html><html><head><meta charset="utf-8"><title>菜鸟教程(runoob.com)</title></head><body>

<h2>include 动作实例</h2><jsp:include page="date.jsp" flush="true" />

</body></html>

现在将以上两个文件放在服务器的根目录下，访问main.jsp文件。显示结果如下：

include 动作实例

今天的日期是: 2016-6-25 14:08:17

## <jsp:useBean>动作元素

**jsp:useBean** 动作用来加载一个将在JSP页面中使用的JavaBean。

这个功能非常有用，因为它使得我们可以发挥 Java 组件复用的优势。

以下是useBean动作相关的属性列表。

|  |  |
| --- | --- |
| **属性** | **描述** |
| class | 指定Bean的完整包名。 |
| type | 指定将引用该对象变量的类型。 |
| beanName | 通过 java.beans.Beans 的 instantiate() 方法指定Bean的名字。 |

## <jsp:setProperty>动作元素

jsp:setProperty用来设置已经实例化的Bean对象的属性，有两种用法。首先，你可以在jsp:useBean元素的外面（后面）使用jsp:setProperty，如下所示：

<jsp:useBean id="myName" ... />

...<jsp:setProperty name="myName" property="someProperty" .../>

此时，不管jsp:useBean是找到了一个现有的Bean，还是新创建了一个Bean实例，jsp:setProperty都会执行。第二种用法是把jsp:setProperty放入jsp:useBean元素的内部，如下所示：

<jsp:useBean id="myName" ... >

...

<jsp:setProperty name="myName" property="someProperty" .../></jsp:useBean>

此时，jsp:setProperty只有在新建Bean实例时才会执行，如果是使用现有实例则不执行jsp:setProperty。

jsp:setProperty动作有下面四个属性,如下表：

|  |  |
| --- | --- |
| **属性** | **描述** |
| name | name属性是必需的。它表示要设置属性的是哪个Bean。 |
| property | property属性是必需的。它表示要设置哪个属性。有一个特殊用法：如果property的值是"\*"，表示所有名字和Bean属性名字匹配的请求参数都将被传递给相应的属性set方法。 |
| value | value 属性是可选的。该属性用来指定Bean属性的值。字符串数据会在目标类中通过标准的valueOf方法自动转换成数字、boolean、Boolean、 byte、Byte、char、Character。例如，boolean和Boolean类型的属性值（比如"true"）通过 Boolean.valueOf转换，int和Integer类型的属性值（比如"42"）通过Integer.valueOf转换。 　　value和param不能同时使用，但可以使用其中任意一个。 |
| param | param 是可选的。它指定用哪个请求参数作为Bean属性的值。如果当前请求没有参数，则什么事情也不做，系统不会把null传递给Bean属性的set方法。因此，你可以让Bean自己提供默认属性值，只有当请求参数明确指定了新值时才修改默认属性值。 |

## <jsp:getProperty>动作元素

jsp:getProperty动作提取指定Bean属性的值，转换成字符串，然后输出。

下表是与getProperty相关联的属性：

|  |  |
| --- | --- |
| **属性** | **描述** |
| name | 要检索的Bean属性名称。Bean必须已定义。 |
| property | 表示要提取Bean属性的值 |

编译完成后会在当前目录下生成一个 **TestBean.class** 文件， 将该文件拷贝至当前 JSP 项目的 **WebContent/WEB-INF/classes/com/runoob/main** 下( com/runoob/main 包路径，没有需要手动创建)。

## <jsp:forward> 动作元素

　jsp:forward动作把请求转到另外的页面。jsp:forward标记只有一个属性page。

以下是forward相关联的属性：

|  |  |
| --- | --- |
| **属性** | **描述** |
| page | page属性包含的是一个相对URL。page的值既可以直接给出，也可以在请求的时候动态计算，可以是一个JSP页面或者一个 Java Servlet. |

## <jsp:plugin>动作元素

jsp:plugin动作用来根据浏览器的类型，插入通过Java插件 运行Java Applet所必需的OBJECT或EMBED元素。

如果需要的插件不存在，它会下载插件，然后执行Java组件。 Java组件可以是一个applet或一个JavaBean。

plugin动作有多个对应HTML元素的属性用于格式化Java 组件。param元素可用于向Applet 或 Bean 传递参数。

以下是使用plugin 动作元素的典型实例:

如果你有兴趣可以尝试使用applet来测试jsp:plugin动作元素，<fallback>元素是一个新元素，在组件出现故障的错误时发送给用户错误信息。

## <jsp:element> 、 <jsp:attribute>、 <jsp:body>动作元素

<jsp:element> 、 <jsp:attribute>、 <jsp:body>动作元素动态定义XML元素。动态是非常重要的，这就意味着XML元素在编译时是动态生成的而非静态。

## <jsp:text>动作元素

<jsp:text>动作元素允许在JSP页面和文档中使用写入文本的模板，语法格式如下：

<jsp:text>模板数据</jsp:text>

以上文本模板不能包含重复元素，只能包含文本和EL表达式（注：EL表达式将在后续章节中介绍）。请注意，在XML文件中，您不能使用表达式如 ${whatever > 0}，因为>符号是非法的。 你可以使用 ${whatever gt 0}表达式或者嵌入在一个CDATA部分的值。

# JSP 隐式对象

JSP隐式对象是JSP容器为每个页面提供的Java对象，开发者可以直接使用它们而不用显式声明。JSP隐式对象也被称为预定义变量。

JSP所支持的九大隐式对象：

|  |  |
| --- | --- |
| **对象** | **描述** |
| request | **HttpServletRequest** 接口的实例 |
| response | **HttpServletResponse** 接口的实例 |
| out | **JspWriter**类的实例，用于把结果输出至网页上 |
| session | **HttpSession**类的实例 |
| application | **ServletContext**类的实例，与应用上下文有关 |
| config | **ServletConfig**类的实例 |
| pageContext | **PageContext**类的实例，提供对JSP页面所有对象以及命名空间的访问 |
| page | 类似于Java类中的this关键字 |
| Exception | **Exception**类的对象，代表发生错误的JSP页面中对应的异常对象 |

## request对象

request对象是javax.servlet.http.HttpServletRequest 类的实例。每当客户端请求一个JSP页面时，JSP引擎就会制造一个新的request对象来代表这个请求。

request对象提供了一系列方法来获取HTTP头信息，cookies，HTTP方法等等。

## response对象

response对象是javax.servlet.http.HttpServletResponse类的实例。当服务器创建request对象时会同时创建用于响应这个客户端的response对象。

response对象也定义了处理HTTP头模块的接口。通过这个对象，开发者们可以添加新的cookies，时间戳，HTTP状态码等等。

## out对象

out对象是 javax.servlet.jsp.JspWriter 类的实例，用来在response对象中写入内容。

最初的JspWriter类对象根据页面是否有缓存来进行不同的实例化操作。可以在page指令中使用buffered='false'属性来轻松关闭缓存。

JspWriter类包含了大部分java.io.PrintWriter类中的方法。不过，JspWriter新增了一些专为处理缓存而设计的方法。还有就是，JspWriter类会抛出IOExceptions异常，而PrintWriter不会。

下表列出了我们将会用来输出boolean，char，int，double，String，object等类型数据的重要方法：

|  |  |
| --- | --- |
| **方法** | **描述** |
| **out.print(dataType dt)** | 输出Type类型的值 |
| **out.println(dataType dt)** | 输出Type类型的值然后换行 |
| **out.flush()** | 刷新输出流 |

## session对象

session对象是 javax.servlet.http.HttpSession 类的实例。和Java Servlets中的session对象有一样的行为。

session对象用来跟踪在各个客户端请求间的会话。

## application对象

application对象直接包装了servlet的ServletContext类的对象，是javax.servlet.ServletContext 类的实例。

这个对象在JSP页面的整个生命周期中都代表着这个JSP页面。这个对象在JSP页面初始化时被创建，随着jspDestroy()方法的调用而被移除。

通过向application中添加属性，则所有组成您web应用的JSP文件都能访问到这些属性。

## config对象

config对象是 javax.servlet.ServletConfig 类的实例，直接包装了servlet的ServletConfig类的对象。

这个对象允许开发者访问Servlet或者JSP引擎的初始化参数，比如文件路径等。

以下是config对象的使用方法，不是很重要，所以不常用：

config.getServletName();

它返回包含在<servlet-name>元素中的servlet名字，注意，<servlet-name>元素在 WEB-INF\web.xml 文件中定义。

## pageContext 对象

pageContext对象是javax.servlet.jsp.PageContext 类的实例，用来代表整个JSP页面。

这个对象主要用来访问页面信息，同时过滤掉大部分实现细节。

这个对象存储了request对象和response对象的引用。application对象，config对象，session对象，out对象可以通过访问这个对象的属性来导出。

pageContext对象也包含了传给JSP页面的指令信息，包括缓存信息，ErrorPage URL,页面scope等。

PageContext类定义了一些字段，包括PAGE\_SCOPE，REQUEST\_SCOPE，SESSION\_SCOPE， APPLICATION\_SCOPE。它也提供了40余种方法，有一半继承自javax.servlet.jsp.JspContext 类。

其中一个重要的方法就是removeArribute()，它可接受一个或两个参数。比如，pageContext.removeArribute("attrName")移除四个scope中相关属性，但是下面这种方法只移除特定scope中的相关属性：

pageContext.removeAttribute("attrName", PAGE\_SCOPE);

## page 对象

这个对象就是页面实例的引用。它可以被看做是整个JSP页面的代表。

page 对象就是this对象的同义词。

## exception 对象

exception 对象包装了从先前页面中抛出的异常信息。它通常被用来产生对出错条件的适当响应。

# JSP 客户端请求

当浏览器请求一个网页时，它会向网络服务器发送一系列不能被直接读取的信息，因为这些信息是作为HTTP信息头的一部分来传送的。您可以查阅HTTP协议来获得更多的信息。

下表列出了浏览器端信息头的一些重要内容，在以后的网络编程中将会经常见到这些信息：

|  |  |
| --- | --- |
| **信息** | **描述** |
| Accept | 指定浏览器或其他客户端可以处理的MIME类型。它的值通常为 **image/png** 或 **image/jpeg** |
| Accept-Charset | 指定浏览器要使用的字符集。比如 ISO-8859-1 |
| Accept-Encoding | 指定编码类型。它的值通常为 **gzip** 或**compress** |
| Accept-Language | 指定客户端首选语言，servlet会优先返回以当前语言构成的结果集，如果servlet支持这种语言的话。比如 en，en-us，ru等等 |
| Authorization | 在访问受密码保护的网页时识别不同的用户 |
| Connection | 表明客户端是否可以处理HTTP持久连接。持久连接允许客户端或浏览器在一个请求中获取多个文件。**Keep-Alive** 表示启用持久连接 |
| Content-Length | 仅适用于POST请求，表示 POST 数据的字节数 |
| Cookie | 返回先前发送给浏览器的cookies至服务器 |
| Host | 指出原始URL中的主机名和端口号 |
| If-Modified-Since | 表明只有当网页在指定的日期被修改后客户端才需要这个网页。 服务器发送304码给客户端，表示没有更新的资源 |
| If-Unmodified-Since | 与If-Modified-Since相反， 只有文档在指定日期后仍未被修改过，操作才会成功 |
| Referer | 标志着所引用页面的URL。比如，如果你在页面1，然后点了个链接至页面2，那么页面1的URL就会包含在浏览器请求页面2的信息头中 |
| User-Agent | 用来区分不同浏览器或客户端发送的请求，并对不同类型的浏览器返回不同的内容 |

## HttpServletRequest类

request对象是javax.servlet.http.HttpServletRequest类的实例。每当客户端请求一个页面时，JSP引擎就会产生一个新的对象来代表这个请求。

request对象提供了一系列方法来获取HTTP信息头，包括表单数据，cookies，HTTP方法等等。

接下来将会介绍一些在JSP编程中常用的获取HTTP信息头的方法。详细内容请见下表：

|  |  |
| --- | --- |
| **序号** | **方法& 描述** |
| 1 | **Cookie[] getCookies()**  返回客户端所有的Cookie的数组 |
| 2 | **Enumeration getAttributeNames()**  返回request对象的所有属性名称的集合 |
| 3 | **Enumeration getHeaderNames()**  返回所有HTTP头的名称集合 |
| 4 | **Enumeration getParameterNames()**  返回请求中所有参数的集合 |
| 5 | **HttpSession getSession()**  返回request对应的session对象，如果没有，则创建一个 |
| 6 | **HttpSession getSession(boolean create)**  返回request对应的session对象，如果没有并且参数create为true，则返回一个新的session对象 |
| 7 | **Locale getLocale()**  返回当前页的Locale对象，可以在response中设置 |
| 8 | **Object getAttribute(String name)**  返回名称为name的属性值，如果不存在则返回null。 |
| 9 | **ServletInputStream getInputStream()**  返回请求的输入流 |
| 10 | **String getAuthType()**  返回认证方案的名称，用来保护servlet，比如 "BASIC" 或者 "SSL" 或 null 如果 JSP没设置保护措施 |
| 11 | **String getCharacterEncoding()**  返回request的字符编码集名称 |
| 12 | **String getContentType()**  返回request主体的MIME类型，若未知则返回null |
| 13 | **String getContextPath()**  返回request URI中指明的上下文路径 |
| 14 | **String getHeader(String name)**  返回name指定的信息头 |
| 15 | **String getMethod()**  返回此request中的HTTP方法，比如 GET,，POST，或PUT |
| 16 | **String getParameter(String name)**  返回此request中name指定的参数，若不存在则返回null |
| 17 | **String getPathInfo()**  返回任何额外的与此request URL相关的路径 |
| 18 | **String getProtocol()**  返回此request所使用的协议名和版本 |
| 19 | **String getQueryString()**  返回此 request URL包含的查询字符串 |
| 20 | **String getRemoteAddr()**  返回客户端的IP地址 |
| 21 | **String getRemoteHost()**  返回客户端的完整名称 |
| 22 | **String getRemoteUser()**  返回客户端通过登录认证的用户，若用户未认证则返回null |
| 23 | **String getRequestURI()**  返回request的URI |
| 24 | **String getRequestedSessionId()**  返回request指定的session ID |
| 25 | **String getServletPath()**  返回所请求的servlet路径 |
| 26 | **String[] getParameterValues(String name)**  返回指定名称的参数的所有值，若不存在则返回null |
| 27 | **boolean isSecure()**  返回request是否使用了加密通道，比如HTTPS |
| 28 | **int getContentLength()**  返回request主体所包含的字节数，若未知的返回-1 |
| 29 | **int getIntHeader(String name)**  返回指定名称的request信息头的值 |
| 30 | **int getServerPort()**  返回服务器端口号 |

## HTTP信息头示例

在这个例子中，我们会使用HttpServletRequest类的getHeaderNames()方法来读取HTTP信息头。这个方法以枚举的形式返回当前HTTP请求的头信息。

获取Enumeration对象后，用标准的方式来遍历Enumeration对象，用hasMoreElements()方法来确定什么时候停止，用nextElement()方法来获得每个参数的名字。

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%><%@ page import="java.io.\*,java.util.\*" %><!DOCTYPE html><html><head><meta charset="utf-8"><title>菜鸟教程(runoob.com)</title></head><body><h2>HTTP 头部请求实例</h2><table width="100%" border="1" align="center"><tr bgcolor="#949494"><th>Header Name</th><th>Header Value(s)</th></tr><%

Enumeration headerNames = request.getHeaderNames();

while(headerNames.hasMoreElements()) {

String paramName = (String)headerNames.nextElement();

out.print("<tr><td>" + paramName + "</td>\n");

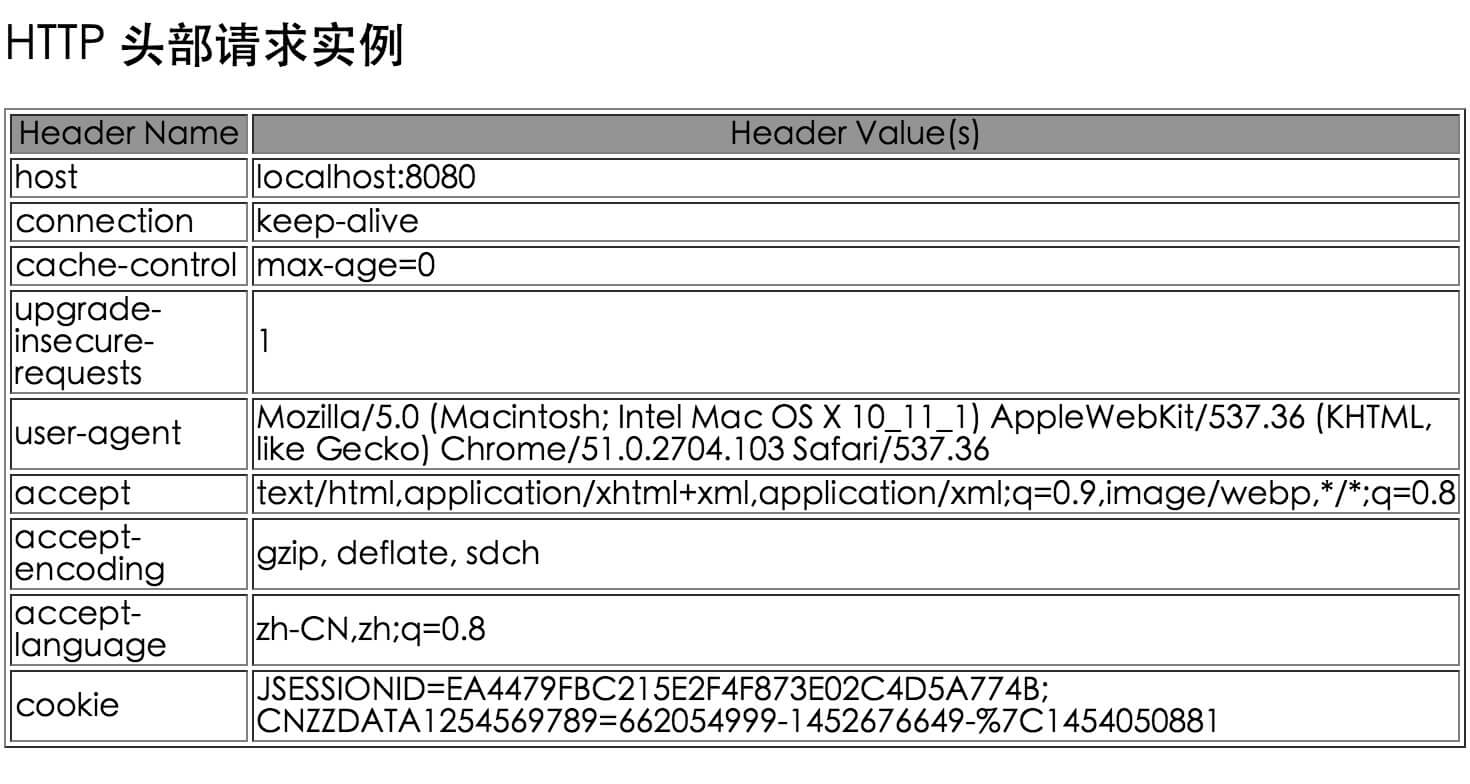
String paramValue = request.getHeader(paramName);

out.println("<td> " + paramValue + "</td></tr>\n");

}

%></table></body></html>

访问main.jsp，将会得到以下结果：



# JSP 服务器响应

Response响应对象主要将JSP容器处理后的结果传回到客户端。可以通过response变量设置HTTP的状态和向客户端发送数据，如Cookie、HTTP文件头信息等。

# JSP 服务器响应

Response响应对象主要将JSP容器处理后的结果传回到客户端。可以通过response变量设置HTTP的状态和向客户端发送数据，如Cookie、HTTP文件头信息等。

一个典型的响应看起来就像下面这样：

HTTP/1.1 200 OKContent-Type: text/htmlHeader2: ......HeaderN: ...

(空行)<!doctype ...><html><head>...</head><body>...</body></html>

状态行包含HTTP版本信息，比如HTTP/1.1，一个状态码，比如200，还有一个非常短的信息对应着状态码，比如OK。

下表摘要出了HTTP1.1响应头中最有用的部分，在网络编程中您将会经常见到它们：

|  |  |
| --- | --- |
| **响应头** | **描述** |
| Allow | 指定服务器支持的request方法（GET，POST等等） |
| Cache-Control | 指定响应文档能够被安全缓存的情况。通常取值为 **public，private** 或**no-cache** 等等。 Public意味着文档可缓存，Private意味着文档只为单用户服务并且只能使用私有缓存。No-cache 意味着文档不被缓存。 |
| Connection | 命令浏览器是否要使用持久的HTTP连接。**close值** 命令浏览器不使用持久HTTP连接，而keep-alive 意味着使用持久化连接。 |
| Content-Disposition | 让浏览器要求用户将响应以给定的名称存储在磁盘中 |
| Content-Encoding | 指定传输时页面的编码规则 |
| Content-Language | 表述文档所使用的语言，比如en， en-us,，ru等等 |
| Content-Length | 表明响应的字节数。只有在浏览器使用持久化 (keep-alive) HTTP 连接时才有用 |
| Content-Type | 表明文档使用的MIME类型 |
| Expires | 指明啥时候过期并从缓存中移除 |
| Last-Modified | 指明文档最后修改时间。客户端可以 缓存文档并且在后续的请求中提供一个 **If-Modified-Since**请求头 |
| Location | 在300秒内，包含所有的有一个状态码的响应地址，浏览器会自动重连然后检索新文档 |
| Refresh | 指明浏览器每隔多久请求更新一次页面。 |
| Retry-After | 与503 (Service Unavailable)一起使用来告诉用户多久后请求将会得到响应 |
| Set-Cookie | 指明当前页面对应的cookie |

## HttpServletResponse类

response 对象是 javax.servlet.http.HttpServletResponse 类的一个实例。就像服务器会创建request对象一样，它也会创建一个客户端响应。

response对象定义了处理创建HTTP信息头的接口。通过使用这个对象，开发者们可以添加新的cookie或时间戳，还有HTTP状态码等等。

下表列出了用来设置HTTP响应头的方法，这些方法由HttpServletResponse 类提供：

|  |  |
| --- | --- |
| **S.N.** | **方法 & 描述** |
| 1 | **String encodeRedirectURL(String url)**  对sendRedirect()方法使用的URL进行编码 |
| 2 | **String encodeURL(String url)**  将URL编码，回传包含Session ID的URL |
| 3 | **boolean containsHeader(String name)**  返回指定的响应头是否存在 |
| 4 | **boolean isCommitted()**  返回响应是否已经提交到客户端 |
| 5 | **void addCookie(Cookie cookie)**  添加指定的cookie至响应中 |
| 6 | **void addDateHeader(String name, long date)**  添加指定名称的响应头和日期值 |
| 7 | **void addHeader(String name, String value)**  添加指定名称的响应头和值 |
| 8 | **void addIntHeader(String name, int value)**  添加指定名称的响应头和int值 |
| 9 | **void flushBuffer()**  将任何缓存中的内容写入客户端 |
| 10 | **void reset()**  清除任何缓存中的任何数据，包括状态码和各种响应头 |
| 11 | **void resetBuffer()**  清除基本的缓存数据，不包括响应头和状态码 |
| 12 | **void sendError(int sc)**  使用指定的状态码向客户端发送一个出错响应，然后清除缓存 |
| 13 | **void sendError(int sc, String msg)**  使用指定的状态码和消息向客户端发送一个出错响应 |
| 14 | **void sendRedirect(String location)**  使用指定的URL向客户端发送一个临时的间接响应 |
| 15 | **void setBufferSize(int size)**  设置响应体的缓存区大小 |
| 16 | **void setCharacterEncoding(String charset)**  指定响应的编码集（MIME字符集），例如UTF-8 |
| 17 | **void setContentLength(int len)**  指定HTTP servlets中响应的内容的长度，此方法用来设置 HTTP Content-Length 信息头 |
| 18 | **void setContentType(String type)**  设置响应的内容的类型，如果响应还未被提交的话 |
| 19 | **void setDateHeader(String name, long date)**  使用指定名称和值设置响应头的名称和内容 |
| 20 | **void setHeader(String name, String value)**  使用指定名称和值设置响应头的名称和内容 |
| 21 | **void setIntHeader(String name, int value)**  指定 int 类型的值到 name 标头 |
| 22 | **void setLocale(Locale loc)**  设置响应的语言环境，如果响应尚未被提交的话 |
| 23 | **void setStatus(int sc)**  设置响应的状态码 |

# JSP HTTP 状态码

HTTP请求与HTTP响应的格式相近，都有着如下结构：

* 以状态行+CRLF（回车换行）开始
* 零行或多行头模块+CRLF
* 一个空行，比如CRLF
* 可选的消息体比如文件，查询数据，查询输出

下表列出了可能会从服务器返回的HTTP状态码和与之关联的消息：

|  |  |  |
| --- | --- | --- |
| **状态码** | **消息** | **描述** |
| 100 | Continue | 只有一部分请求被服务器接收，但只要没被服务器拒绝，客户端就会延续这个请求 |
| 101 | Switching Protocols | 服务器交换机协议 |
| 200 | OK | 请求被确认 |
| 201 | Created | 请求时完整的，新的资源被创建 |
| 202 | Accepted | 请求被接受，但未处理完 |
| 203 | Non-authoritative Information |  |
| 204 | No Content |  |
| 205 | Reset Content |  |
| 206 | Partial Content |  |
| 300 | Multiple Choices | 一个超链接表，用户可以选择一个超链接并访问，最大支持5个超链接 |
| 301 | Moved Permanently | 被请求的页面已经移动到了新的URL下 |
| 302 | Found | 被请求的页面暂时性地移动到了新的URL下 |
| 303 | See Other | 被请求的页面可以在一个不同的URL下找到 |
| 304 | Not Modified |  |
| 305 | Use Proxy |  |
| 306 | Unused | 已经不再使用此状态码，但状态码被保留 |
| 307 | Temporary Redirect | 被请求的页面暂时性地移动到了新的URL下 |
| 400 | Bad Request | 服务器无法识别请求 |
| 401 | Unauthorized | 被请求的页面需要用户名和密码 |
| 402 | Payment Required | 目前还不能使用此状态码 |
| 403 | Forbidden | 禁止访问所请求的页面 |
| 404 | Not Found | 服务器无法找到所请求的页面 |
| 405 | Method Not Allowed | 请求中所指定的方法不被允许 |
| 406 | Not Acceptable | 服务器只能创建一个客户端无法接受的响应 |
| 407 | Proxy Authentication Required | 在请求被服务前必须认证一个代理服务器 |
| 408 | Request Timeout | 请求时间超过了服务器所能等待的时间，连接被断开 |
| 409 | Conflict | 请求有矛盾的地方 |
| 410 | Gone | 被请求的页面不再可用 |
| 411 | Length Required | "Content-Length"没有被定义，服务器拒绝接受请求 |
| 412 | Precondition Failed | 请求的前提条件被服务器评估为false |
| 413 | Request Entity Too Large | 因为请求的实体太大，服务器拒绝接受请求 |
| 414 | Request-url Too Long | 服务器拒绝接受请求，因为URL太长。多出现在把"POST"请求转换为"GET"请求时所附带的大量查询信息 |
| 415 | Unsupported Media Type | 服务器拒绝接受请求，因为媒体类型不被支持 |
| 417 | Expectation Failed |  |
| 500 | Internal Server Error | 请求不完整，服务器遇见了出乎意料的状况 |
| 501 | Not Implemented | 请求不完整，服务器不提供所需要的功能 |
| 502 | Bad Gateway | 请求不完整，服务器从上游服务器接受了一个无效的响应 |
| 503 | Service Unavailable | 请求不完整，服务器暂时重启或关闭 |
| 504 | Gateway Timeout | 网关超时 |
| 505 | HTTP Version Not Supported | 服务器不支持所指定的HTTP版本 |

## 设置HTTP状态码的方法

下表列出了HttpServletResponse 类中用来设置状态码的方法：

|  |  |
| --- | --- |
| **S.N.** | **方法 & 描述** |
| 1 | **public void setStatus ( int statusCode )**  此方法可以设置任意的状态码。如果您的响应包含一个特殊的状态码和一个文档，请确保在用PrintWriter返回任何内容前调用setStatus方法 |
| 2 | **public void sendRedirect(String url)**  此方法产生302响应，同时产生一个 Location 头告诉URL 一个新的文档 |
| 3 | **public void sendError(int code, String message)**  此方法将一个状态码(通常为 404)和一个短消息，自动插入HTML文档中并发回给客户端 |

## HTTP状态码程序示例

接下来的例子将会发送407错误码给浏览器，然后浏览器将会告诉您"Need authentication!!!"。

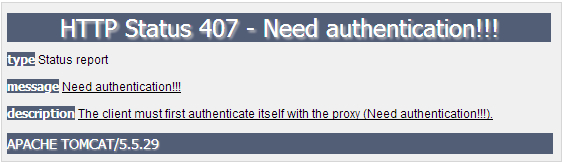
<html><head><title>Setting HTTP Status Code</title></head><body><%

// 设置错误代码，并说明原因

response.sendError(407, "Need authentication!!!" );

%></body></html>

访问以上JSP页面，将会得到以下结果：



您也可以试试使用其他的状态码，看会不会得到什么意想不到结果。

# JSP 表单处理

我们在浏览网页的时候，经常需要向服务器提交信息，并让后台程序处理。浏览器中使用 GET 和 POST 方法向服务器提交数据。

## GET 方法

GET方法将请求的编码信息添加在网址后面，网址与编码信息通过"?"号分隔。如下所示：

http://www.runoob.com/hello?key1=value1&key2=value2

GET方法是浏览器默认传递参数的方法，一些敏感信息，如密码等建议不使用GET方法。

用get时，传输数据的大小有限制 （注意不是参数的个数有限制），最大为1024字节。

## POST 方法

一些敏感信息，如密码等我们可以通过POST方法传递，POST提交数据是隐式的。

POST提交数据是不可见的，GET是通过在url里面传递的（可以看一下你浏览器的地址栏）。

JSP使用getParameter()来获得传递的参数，getInputStream()方法用来处理客户端的二进制数据流的请求。

## JSP 读取表单数据

**getParameter():** 使用 request.getParameter() 方法来获取表单参数的值。

**getParameterValues():** 获得如checkbox类（名字相同，但值有多个）的数据。 接收数组变量 ，如checkbox类型

**getParameterNames():**该方法可以取得所有变量的名称，该方法返回一个 Enumeration。

**getInputStream():**调用此方法来读取来自客户端的二进制数据流。

## 使用URL的 GET 方法实例

以下是一个简单的URL,并使用GET方法来传递URL中的参数：

http://localhost:8080/testjsp/main.jsp?name=菜鸟教程&url=http://ww.runoob.com

## 读取所有表单参数

以下我们将使用 **HttpServletRequest** 的 **getParameterNames()** 来读取所有表单参数,该方法可以取得所有变量的名称，该方法返回一个枚举。

一旦我们有了一个 Enumeration（枚举），我们就可以调用 hasMoreElements() 方法来确定是否还有元素，以及使用nextElement（）方法来获得每个参数的名称。

# JSP 过滤器

JSP 和 Servlet 中的过滤器都是 Java 类。

过滤器可以动态地拦截请求和响应，以变换或使用包含在请求或响应中的信息。

可以将一个或多个过滤器附加到一个 Servlet 或一组 Servlet。过滤器也可以附加到 JavaServer Pages (JSP) 文件和 HTML 页面。

过滤器是可用于 Servlet 编程的 Java 类，可以实现以下目的：

* 在客户端的请求访问后端资源之前，拦截这些请求。
* 在服务器的响应发送回客户端之前，处理这些响应。

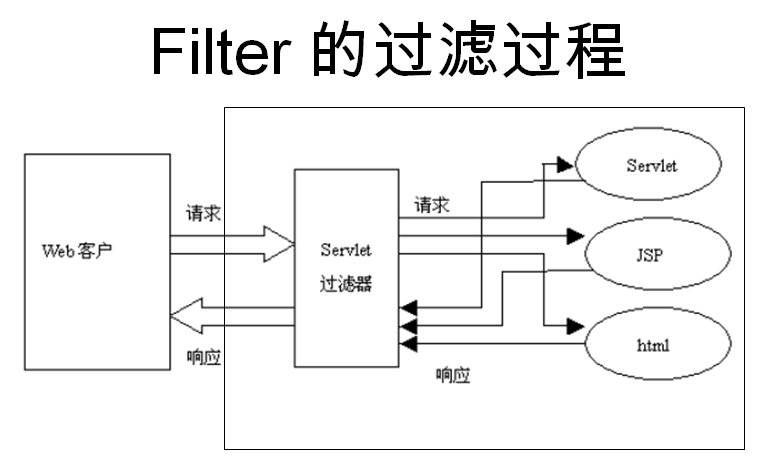
根据规范建议的各种类型的过滤器：

* 身份验证过滤器（Authentication Filters）。
* 数据压缩过滤器（Data compression Filters）。
* 加密过滤器（Encryption Filters）。
* 触发资源访问事件过滤器。
* 图像转换过滤器（Image Conversion Filters）。
* 日志记录和审核过滤器（Logging and Auditing Filters）。
* MIME-TYPE 链过滤器（MIME-TYPE Chain Filters）。
* 标记化过滤器（Tokenizing Filters）。
* XSL/T 过滤器（XSL/T Filters），转换 XML 内容。

过滤器通过 Web 部署描述符（web.xml）中的 XML 标签来声明，然后映射到您的应用程序的部署描述符中的 Servlet 名称或 URL 模式。

当 Web 容器启动 Web 应用程序时，它会为您在部署描述符中声明的每一个过滤器创建一个实例。

Filter 的执行顺序与在 web.xml 配置文件中的配置顺序一致，一般把 Filter 配置在所有的 Servlet 之前。



## Servlet 过滤器方法

过滤器是一个实现了 javax.servlet.Filter 接口的 Java 类。javax.servlet.Filter 接口定义了三个方法：

|  |  |
| --- | --- |
| **序号** | **方法 & 描述** |
| 1 | **public void doFilter (ServletRequest, ServletResponse, FilterChain)** 该方法完成实际的过滤操作，当客户端请求方法与过滤器设置匹配的URL时，Servlet容器将先调用过滤器的doFilter方法。FilterChain用户访问后续过滤器。 |
| 2 | **public void init(FilterConfig filterConfig)** web 应用程序启动时，web 服务器将创建Filter 的实例对象，并调用其init方法，读取web.xml配置，完成对象的初始化功能，从而为后续的用户请求作好拦截的准备工作（filter对象只会创建一次，init方法也只会执行一次）。开发人员通过init方法的参数，可获得代表当前filter配置信息的FilterConfig对象。 |
| 3 | **public void destroy()** Servlet容器在销毁过滤器实例前调用该方法，在该方法中释放Servlet过滤器占用的资源。 |

### **FilterConfig 使用**

Filter 的 init 方法中提供了一个 FilterConfig 对象。

在 init 方法使用 FilterConfig 对象获取参数：

### **Web.xml 中的 Servlet 过滤器映射（Servlet Filter Mapping）**

定义过滤器，然后映射到一个 URL 或 Servlet，这与定义 Servlet，然后映射到一个 URL 模式方式大致相同。

## 过滤器的应用顺序

web.xml 中的 filter-mapping 元素的顺序决定了 Web 容器应用过滤器到 Servlet 的顺序。若要反转过滤器的顺序，您只需要在 web.xml 文件中反转 filter-mapping 元素即可。

例如，上面的实例将先应用 LogFilter，然后再应用 AuthenFilter，但是下面的实例将颠倒这个顺序：

## web.xml配置各节点说明

* <filter>指定一个过滤器。
  + <filter-name>用于为过滤器指定一个名字，该元素的内容不能为空。
  + <filter-class>元素用于指定过滤器的完整的限定类名。
  + <init-param>元素用于为过滤器指定初始化参数，它的子元素<param-name>指定参数的名字，<param-value>指定参数的值。
  + 在过滤器中，可以使用FilterConfig接口对象来访问初始化参数。
* <filter-mapping>元素用于设置一个 Filter 所负责拦截的资源。一个Filter拦截的资源可通过两种方式来指定：Servlet 名称和资源访问的请求路径
  + <filter-name>子元素用于设置filter的注册名称。该值必须是在<filter>元素中声明过的过滤器的名字
  + <url-pattern>设置 filter 所拦截的请求路径(过滤器关联的URL样式)
* <servlet-name>指定过滤器所拦截的Servlet名称。
* <dispatcher>指定过滤器所拦截的资源被 Servlet 容器调用的方式，可以是REQUEST,INCLUDE,FORWARD和ERROR之一，默认REQUEST。用户可以设置多个<dispatcher>子元素用来指定 Filter 对资源的多种调用方式进行拦截。
* <dispatcher>子元素可以设置的值及其意义
  + REQUEST：当用户直接访问页面时，Web容器将会调用过滤器。如果目标资源是通过RequestDispatcher的include()或forward()方法访问时，那么该过滤器就不会被调用。
  + INCLUDE：如果目标资源是通过RequestDispatcher的include()方法访问时，那么该过滤器将被调用。除此之外，该过滤器不会被调用。
  + FORWARD：如果目标资源是通过RequestDispatcher的forward()方法访问时，那么该过滤器将被调用，除此之外，该过滤器不会被调用。
  + ERROR：如果目标资源是通过声明式异常处理机制调用时，那么该过滤器将被调用。除此之外，过滤器不会被调用。

## 一、Filter 的基本工作原理

* 1、Filter 程序是一个实现了特殊接口的 Java 类，与 Servlet 类似，也是由 Servlet 容器进行调用和执行的。
* 2、当在 web.xml 注册了一个 Filter 来对某个 Servlet 程序进行拦截处理时，它可以决定是否将请求继续传递给 Servlet 程序，以及对请求和响应消息是否进行修改。
* 3、当 Servlet 容器开始调用某个 Servlet 程序时，如果发现已经注册了一个 Filter 程序来对该 Servlet 进行拦截，那么容器不再直接调用 Servlet 的 service 方法，而是调用 Filter 的 doFilter 方法，再由 doFilter 方法决定是否去激活 service 方法。
* 4、但在 Filter.doFilter 方法中不能直接调用 Servlet 的 service 方法，而是调用 FilterChain.doFilter 方法来激活目标 Servlet 的 service 方法，FilterChain 对象时通过 Filter.doFilter 方法的参数传递进来的。
* 5、只要在 Filter.doFilter 方法中调用 FilterChain.doFilter 方法的语句前后增加某些程序代码，这样就可以在 Servlet 进行响应前后实现某些特殊功能。
* 6、如果在 Filter.doFilter 方法中没有调用 FilterChain.doFilter 方法，则目标 Servlet 的 service 方法不会被执行，这样通过 Filter 就可以阻止某些非法的访问请求。

# JSP Cookie 处理

Cookie是存储在客户机的文本文件，它们保存了大量轨迹信息。在servlet技术基础上，JSP显然能够提供对HTTP cookie的支持。

通常有三个步骤来识别回头客：

* 服务器脚本发送一系列cookie至浏览器。比如名字，年龄，ID号码等等。
* 浏览器在本地机中存储这些信息，以备不时之需。
* 当下一次浏览器发送任何请求至服务器时，它会同时将这些cookie信息发送给服务器，然后服务器使用这些信息来识别用户或者干些其它事情。

本章节将会传授您如何去设置或重设cookie的方法，还有如何访问它们及如何删除它们。

*JSP Cookie 处理需要对中文进行编码与解码，方法如下：*

String str = java.net.URLEncoder.encode("中文"，"UTF-8"); //编码String str = java.net.URLDecoder.decode("编码后的字符串","UTF-8"); // 解码

Set-Cookie: name=runoob; expires=Friday, 04-Feb-07 22:03:38 GMT;

path=/; domain=runoob.comConnection: closeContent-Type: text/html

正如您所见，Set-Cookie信息头包含一个键值对，一个GMT（格林尼治标准）时间，一个路径，一个域名。键值对会被编码为URL。有效期域是个指令，告诉浏览器在什么时候之后就可以清除这个cookie。

如果浏览器被配置成可存储cookie，那么它将会保存这些信息直到过期。如果用户访问的任何页面匹配了cookie中的路径和域名，那么浏览器将会重新将这个cookie发回给服务器。浏览器端的信息头长得就像下面这样：

JSP脚本通过request对象中的getCookies()方法来访问这些cookie，这个方法会返回一个Cookie对象的数组。

## Servlet Cookie 方法

下表列出了Cookie对象中常用的方法：

|  |  |
| --- | --- |
| **序号** | **方法 & 描述** |
| 1 | **public void setDomain(String pattern)**  设置cookie的域名，比如 runoob.com |
| 2 | **public String getDomain()**  获取cookie的域名，比如 runoob.com |
| 3 | **public void setMaxAge(int expiry)**  设置cookie有效期，以秒为单位，默认有效期为当前session的存活时间 |
| 4 | **public int getMaxAge()**  获取cookie有效期，以秒为单位，默认为-1 ，表明cookie会活到浏览器关闭为止 |
| 5 | **public String getName()**  返回 cookie的名称，名称创建后将不能被修改 |
| 6 | **public void setValue(String newValue)**  设置 cookie的值 |
| 7 | **public String getValue()**  获取cookie的值 |
| 8 | **public void setPath(String uri)**  设置cookie 的路径，默认为当前页面目录下的所有URL，还有此目录下的所有子目录 |
| 9 | **public String getPath()**  获取cookie 的路径 |
| 10 | **public void setSecure(boolean flag)**  指明cookie是否要加密传输 |
| 11 | **public void setComment(String purpose)**  设置注释描述 cookie的目的。当浏览器将cookie展现给用户时，注释将会变得非常有用 |
| 12 | **public String getComment()**  返回描述cookie目的的注释，若没有则返回null |

## 使用JSP设置Cookie

使用JSP设置cookie包含三个步骤：

**(1)创建一个Cookie对象：**调用Cookie的构造函数，使用一个cookie名称和值做参数，它们都是字符串。

Cookie cookie = new Cookie("key","value");

请务必牢记，名称和值中都不能包含空格或者如下的字符：

[ ] ( ) = , " / ? @ : ;

**(2) 设置有效期：**调用setMaxAge()函数表明cookie在多长时间（以秒为单位）内有效。下面的操作将有效期设为了24小时。

cookie.setMaxAge(60\*60\*24);

**(3) 将cookie发送至HTTP响应头中：**调用response.addCookie()函数来向HTTP响应头中添加cookie。

response.addCookie(cookie);

## 使用 JSP 读取 Cookie

想要读取cookie，您就需要调用request.getCookies()方法来获得一个javax.servlet.http.Cookie对象的数组，然后遍历这个数组，使用getName()方法和getValue()方法来获取每一个cookie的名称和值。

<h3实例演示< h3="" style="color: rgb(51, 51, 51); font-family: "Helvetica Neue", Helvetica, "PingFang SC", "Hiragino Sans GB", "Microsoft YaHei", "Noto Sans CJK SC", "WenQuanYi Micro Hei", Arial, sans-serif; font-size: 12px; font-style: normal; font-variant-ligatures: normal; font-variant-caps: normal; font-weight: 400; letter-spacing: normal; orphans: 2; text-align: start; text-indent: 0px; text-transform: none; white-space: normal; widows: 2; word-spacing: 0px; -webkit-text-stroke-width: 0px; background-color: rgb(255, 255, 255); text-decoration-style: initial; text-decoration-color: initial;">

让我们来读取上个例子中的cookie,

## 使用JSP删除Cookie

删除cookie非常简单。如果您想要删除一个cookie，按照下面给的步骤来做就行了：

* 获取一个已经存在的cookie然后存储在Cookie对象中。
* 将cookie的有效期设置为0。
* 将这个cookie重新添加进响应头中。

# JSP Session

HTTP是无状态协议，这意味着每次客户端检索网页时，都要单独打开一个服务器连接，因此服务器不会记录下先前客户端请求的任何信息。

有三种方法来维持客户端与服务器的会话：

## Cookies

网络服务器可以指定一个唯一的session ID作为cookie来代表每个客户端，用来识别这个客户端接下来的请求。

这可能不是一种有效的方式，因为很多时候浏览器并不一定支持cookie，所以我们不建议使用这种方法来维持会话。

## 隐藏表单域

一个网络服务器可以发送一个隐藏的HTML表单域和一个唯一的session ID，就像下面这样：

<input type="hidden" name="sessionid" value="12345">

这个条目意味着，当表单被提交时，指定的名称和值将会自动包含在GET或POST数据中。每当浏览器发送一个请求，session\_id的值就可以用来保存不同浏览器的轨迹。

这种方式可能是一种有效的方式，但点击<A HREF>标签中的超链接时不会产生表单提交事件，因此隐藏表单域也不支持通用会话跟踪。

## 重写URL

您可以在每个URL后面添加一些额外的数据来区分会话，服务器能够根据这些数据来关联session标识符。

举例来说，http://w3cschool.cc/file.htm;sessionid=12345， session标识符为sessionid=12345，服务器可以用这个数据来识别客户端。

相比而言，重写URL是更好的方式来，就算浏览器不支持cookies也能工作，但缺点是您必须为每个URL动态指定session ID，就算这是个简单的HTML页面。

## session对象

除了以上几种方法外，JSP利用servlet提供的HttpSession接口来识别一个用户，存储这个用户的所有访问信息。

默认情况下，JSP允许会话跟踪，一个新的HttpSession对象将会自动地为新的客户端实例化。禁止会话跟踪需要显式地关掉它，通过将page指令中session属性值设为false来实现，就像下面这样：

<%@ page session="false" %>

JSP引擎将隐含的session对象暴露给开发者。由于提供了session对象，开发者就可以方便地存储或检索数据。

下表列出了session对象的一些重要方法：

|  |  |
| --- | --- |
| **S.N.** | **方法 & 描述** |
| 1 | **public Object getAttribute(String name)**  返回session对象中与指定名称绑定的对象，如果不存在则返回null |
| 2 | **public Enumeration getAttributeNames()**  返回session对象中所有的对象名称 |
| 3 | **public long getCreationTime()**  返回session对象被创建的时间， 以毫秒为单位，从1970年1月1号凌晨开始算起 |
| 4 | **public String getId()**  返回session对象的ID |
| 5 | **public long getLastAccessedTime()**  返回客户端最后访问的时间，以毫秒为单位，从1970年1月1号凌晨开始算起 |
| 6 | **public int getMaxInactiveInterval()**  返回最大时间间隔，以秒为单位，servlet 容器将会在这段时间内保持会话打开 |
| 7 | **public void invalidate()**  将session无效化，解绑任何与该session绑定的对象 |
| 8 | **public boolean isNew()**  返回是否为一个新的客户端，或者客户端是否拒绝加入session |
| 9 | **public void removeAttribute(String name)**  移除session中指定名称的对象 |
| 10 | **public void setAttribute(String name, Object value)**  使用指定的名称和值来产生一个对象并绑定到session中 |
| 11 | **public void setMaxInactiveInterval(int interval)**  用来指定时间，以秒为单位，servlet容器将会在这段时间内保持会话有效 |

## JSP Session应用

这个例子描述了如何使用HttpSession对象来获取创建时间和最后一次访问时间。我们将会为request对象关联一个新的session对象，如果这个对象尚未存在的话。

## 删除Session数据

当处理完一个用户的会话数据后，您可以有如下选择：

* **移除一个特定的属性：**

调用public void removeAttribute(String name)  方法来移除指定的属性。

* **删除整个会话：**

调用public void invalidate() 方法来使整个session无效。

* **设置会话有效期：**

调用 public void setMaxInactiveInterval(int interval)  方法来设置session超时。

* **登出用户：**

支持servlet2.4版本的服务器，可以调用 logout()方法来登出用户，并且使所有相关的session无效。

* **配置web.xml文件：**

如果使用的是Tomcat，可以向下面这样配置web.xml文件：

<session-config>

<session-timeout>15</session-timeout>

</session-config>

超时以分钟为单位，Tomcat中的默认的超时时间是30分钟。

Servlet中的getMaxInactiveInterval( ) 方法以秒为单位返回超时时间。如果在web.xml中配置的是15分钟，则getMaxInactiveInterval( ) 方法将会返回900。

JSP 可以与 HTML form 标签一起使用，来允许用户上传文件到服务器。上传的文件可以是文本文件或图像文件或任何文档。

# JSP 日期处理

使用JSP最重要的优势之一，就是可以使用所有Java  API。本章将会详细地讲述Java中的Date类，它在java.util包下，封装了当前日期和时间。

Date类有两个构造函数。第一个构造函数使用当前日期和时间来初始化对象。

Date( )

第二个构造函数接受一个参数，这个参数表示从1970年1月1日凌晨至所要表示时间的毫秒数。

Date(long millisec)

获取Date对象后，您就能够使用下表列出的所有方法：

|  |  |
| --- | --- |
| **序号** | **方法 & 描述** |
| 1 | **boolean after(Date date)**  如果比给定的日期晚，则返回true，否则返回false |
| 2 | **boolean before(Date date)**  如果比给定的日期早，则返回true，否则返回false |
| 3 | **Object clone( )**  获取当前对象的一个副本 |
| 4 | **int compareTo(Date date)**  如果与给定日期相等，则返回0，如果比给定日期早，则返回一个负数，如果比给定日期晚，则返回一个正数 |
| 5 | **int compareTo(Object obj)**  与 compareTo(Date) 方法相同，如果 obj 不是Date类或其子类的对象，抛出ClassCastException异常 |
| 6 | **boolean equals(Object date)**  如果与给定日期相同，则返回true，否则返回false |
| 7 | **long getTime( )**  返回从1970年1月1日凌晨至此对象所表示时间的毫秒数 |
| 8 | **int hashCode( )**  返回此对象的哈希码 |
| 9 | **void setTime(long time)**  使用给定参数设置时间和日期，参数time表示从1970年1月1日凌晨至time所经过的毫秒数 |
| 10 | **String toString( )**  将此对象转换为字符串并返回这个字符串 |

## 获取当前日期和时间

使用JSP编程可以很容易的获取当前日期和时间，只要使用Date对象的toString()方法就行了，就像下面这样：

## 日期比较

就像我在开头所提到的，您可以在JSP脚本中使用任何Java方法。如果您想要比较两个日期，

可以参照下面的方法来做：

* 使用getTime()方法得到毫秒数，然后比较毫秒数就行了。
* 使用before()，after()，equals()方法。比如，new Date(99,2,12).before(new Date(99,2,18))返回true。
* 使用compareTo()方法，这个方法在Comparable接口中定义，在Date中实现。

## 使用SimpleDateFormat格式化日期

SimpleDateFormat使用一种地区敏感的方式来格式化和解析日期，它允许您使用自定义的模式来格式化日期和时间。

## SimpleDateFormat格式码

要指定模式字符串，需要使用下表列出的格式码：

|  |  |  |
| --- | --- | --- |
| **字符** | **描述** | **示例** |
| G | 时代标识符 | AD |
| y | 4位数年份 | 2001 |
| M | 月 | July or 07 |
| d | 日 | 10 |
| h | 12小时制， A.M./P.M. (1~12) | 12 |
| H | 24小时制 | 22 |
| m | 分钟 | 30 |
| s | 秒 | 55 |
| S | 毫秒 | 234 |
| E | 星期 | Tuesday |
| D | 一年中的某天 | 360 |
| F | 一个月中某星期的某天 | 2 (second Wed. in July) |
| w | 一年中的某星期 | 40 |
| W | 一个月中的某星期 | 1 |
| a | A.M./P.M. 标记 | PM |
| k | 一天中的某个小时 (1~24) | 24 |
| K | 一天中的某个小时，A.M./P.M. (0~11) | 10 |
| z | 时区 | Eastern Standard Time |
| ' | 文本分隔 | Delimiter |
| " | 单引号 | ` |

# JSP 页面重定向

当需要将文档移动到一个新的位置时，就需要使用JSP重定向了。

最简单的重定向方式就是使用response对象的sendRedirect()方法。这个方法的签名如下：

public void response.sendRedirect(String location)throws IOException

这个方法将状态码和新的页面位置作为响应发回给浏览器。您也可以使用setStatus()和setHeader()方法来得到同样的效果：

....String site = "http://www.runoob.com" ;

response.setStatus(response.SC\_MOVED\_TEMPORARILY);

response.setHeader("Location", site); ....

# JSP 点击量统计

有时候我们需要知道某个页面被访问的次数，这时我们就需要在页面上添加页面统计器，页面访问的统计一般在用户第一次载入时累加该页面的访问数上。

要实现一个计数器，您可以利用应用程序隐式对象和相关方法getAttribute()和setAttribute()来实现。

这个对象表示JSP页面的整个生命周期中。当JSP页面初始化时创建此对象，当JSP页面调用jspDestroy()时删除该对象。

以下是在应用中创建变量的语法：

application.setAttribute(String Key, Object Value);

您可以使用上述方法来设置一个计数器变量及更新该变量的值。读取该变量的方法如下：

application.getAttribute(String Key);

在页面每次被访问时，你可以读取计数器的当前值，并递增1，然后重新设置，在下一个用户访问时就将新的值显示在页面上。

## 复位计数器

使用以上方法，在 web 服务器重启后，计数器会被复位为 0，即前面保留的数据都会消失，你可以使用以下几种方式解决该问题:

* 在数据库中定义一个用于统计网页访问量的数据表 count，字段为 hitcount，hitcount 默认值为0，将统计数据写入到数据表中。
* 在每次访问时我们读取表中 hitcount 字段。
* 每次访问时让 hitcount 自增 1。
* 在页面上显示新的 hitcount 值作为页面的访问量。
* 如果你需要统计每个页面的访问量，你可以使用以上逻辑将代码添加到所有页面上。

# JSP 自动刷新

想象一下，如果要直播比赛的比分，或股票市场的实时状态，或当前的外汇配给，该怎么实现呢？显然，要实现这种实时功能，您就不得不规律性地刷新页面。

JSP提供了一种机制来使这种工作变得简单，它能够定时地自动刷新页面。

刷新一个页面最简单的方式就是使用response对象的setIntHeader()方法。这个方法的签名如下：

public void setIntHeader(String header, int headerValue)

这个方法通知浏览器在给定的时间后刷新，时间以秒为单位。

# JSP 发送邮件

虽然使用JSP实现邮件发送功能很简单，但是需要有JavaMail API，并且需要安装JavaBean Activation Framework。

* 您可以从 Java 网站下载最新版本的 [JavaMail](http://www.oracle.com/technetwork/java/javamail/index.html" \t "https://www.runoob.com/jsp/_blank)，打开网页右侧有个 **Downloads** 链接，点击它下载。
* 您可以从 Java 网站下载最新版本的 [JAF（版本 1.1.1）](http://www.oracle.com/technetwork/articles/java/index-135046.html" \t "https://www.runoob.com/jsp/_blank)。

你也可以使用本站提供的下载链接：

* [JavaMail mail.jar 1.4.5](http://static.runoob.com/download/mail.jar" \t "https://www.runoob.com/jsp/_blank)
* [JAF（版本 1.1.1） activation.jar](http://static.runoob.com/download/activation.jar" \t "https://www.runoob.com/jsp/_blank)

下载并解压这些文件，在根目录下，您将会看到一系列jar包。将mail.jar包和activation.jar包加入CLASSPATH变量中。

## 发送一封简单的邮件

这个例子展示了如何从您的机器发送一封简单的邮件。它假定localhost已经连接至网络并且有能力发送一封邮件。与此同时，请再一次确认mail.jar包和activation.jar包已经添加进CLASSPATH变量中。

# JSP 标准标签库（JSTL）

JSP标准标签库（JSTL）是一个JSP标签集合，它封装了JSP应用的通用核心功能。

JSTL支持通用的、结构化的任务，比如迭代，条件判断，XML文档操作，国际化标签，SQL标签。 除了这些，它还提供了一个框架来使用集成JSTL的自定义标签。

根据JSTL标签所提供的功能，可以将其分为5个类别。

* **核心标签**
* **格式化标签**
* **SQL 标签**
* **XML 标签**
* **JSTL 函数**

使用任何库，你必须在每个 JSP 文件中的头部包含 **<taglib>** 标签。

## 核心标签

核心标签是最常用的 JSTL标签。引用核心标签库的语法如下：

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

|  |  |
| --- | --- |
| **标签** | **描述** |
| [<c:out>](https://www.runoob.com/jsp/jstl-core-out-tag.html" \t "https://www.runoob.com/jsp/_blank) | 用于在JSP中显示数据，就像<%= ... > |
| [<c:set>](https://www.runoob.com/jsp/jstl-core-set-tag.html" \t "https://www.runoob.com/jsp/_blank) | 用于保存数据 |
| [<c:remove>](https://www.runoob.com/jsp/jstl-core-remove-tag.html" \t "https://www.runoob.com/jsp/_blank) | 用于删除数据 |
| [<c:catch>](https://www.runoob.com/jsp/jstl-core-catch-tag.html" \t "https://www.runoob.com/jsp/_blank) | 用来处理产生错误的异常状况，并且将错误信息储存起来 |
| [<c:if>](https://www.runoob.com/jsp/jstl-core-if-tag.html" \t "https://www.runoob.com/jsp/_blank) | 与我们在一般程序中用的if一样 |
| [<c:choose>](https://www.runoob.com/jsp/jstl-core-choose-tag.html" \t "https://www.runoob.com/jsp/_blank) | 本身只当做<c:when>和<c:otherwise>的父标签 |
| [<c:when>](https://www.runoob.com/jsp/jstl-core-choose-tag.html" \t "https://www.runoob.com/jsp/_blank) | <c:choose>的子标签，用来判断条件是否成立 |
| [<c:otherwise>](https://www.runoob.com/jsp/jstl-core-choose-tag.html" \t "https://www.runoob.com/jsp/_blank) | <c:choose>的子标签，接在<c:when>标签后，当<c:when>标签判断为false时被执行 |
| [<c:import>](https://www.runoob.com/jsp/jstl-core-import-tag.html" \t "https://www.runoob.com/jsp/_blank) | 检索一个绝对或相对 URL，然后将其内容暴露给页面 |
| [<c:forEach>](https://www.runoob.com/jsp/jstl-core-foreach-tag.html" \t "https://www.runoob.com/jsp/_blank) | 基础迭代标签，接受多种集合类型 |
| [<c:forTokens>](https://www.runoob.com/jsp/jstl-core-foreach-tag.html" \t "https://www.runoob.com/jsp/_blank) | 根据指定的分隔符来分隔内容并迭代输出 |
| [<c:param>](https://www.runoob.com/jsp/jstl-core-param-tag.html" \t "https://www.runoob.com/jsp/_blank) | 用来给包含或重定向的页面传递参数 |
| [<c:redirect>](https://www.runoob.com/jsp/jstl-core-redirect-tag.html" \t "https://www.runoob.com/jsp/_blank) | 重定向至一个新的URL. |
| [<c:url>](https://www.runoob.com/jsp/jstl-core-url-tag.html" \t "https://www.runoob.com/jsp/_blank) | 使用可选的查询参数来创造一个URL |

## 格式化标签

JSTL格式化标签用来格式化并输出文本、日期、时间、数字。引用格式化标签库的语法如下：

<%@ taglib prefix="fmt"

uri="http://java.sun.com/jsp/jstl/fmt" %>

|  |  |
| --- | --- |
| **标签** | **描述** |
| [<fmt:formatNumber>](https://www.runoob.com/jsp/jstl-format-formatnumber-tag.html" \t "https://www.runoob.com/jsp/_blank) | 使用指定的格式或精度格式化数字 |
| [<fmt:parseNumber>](https://www.runoob.com/jsp/jstl-format-parsenumber-tag.html" \t "https://www.runoob.com/jsp/_blank) | 解析一个代表着数字，货币或百分比的字符串 |
| [<fmt:formatDate>](https://www.runoob.com/jsp/jstl-format-formatdate-tag.html" \t "https://www.runoob.com/jsp/_blank) | 使用指定的风格或模式格式化日期和时间 |
| [<fmt:parseDate>](https://www.runoob.com/jsp/jstl-format-parsedate-tag.html" \t "https://www.runoob.com/jsp/_blank) | 解析一个代表着日期或时间的字符串 |
| [<fmt:bundle>](https://www.runoob.com/jsp/jstl-format-bundle-tag.html" \t "https://www.runoob.com/jsp/_blank) | 绑定资源 |
| [<fmt:setLocale>](https://www.runoob.com/jsp/jstl-format-setlocale-tag.html" \t "https://www.runoob.com/jsp/_blank) | 指定地区 |
| [<fmt:setBundle>](https://www.runoob.com/jsp/jstl-format-setbundle-tag.html" \t "https://www.runoob.com/jsp/_blank) | 绑定资源 |
| [<fmt:timeZone>](https://www.runoob.com/jsp/jstl-format-timezone-tag.html" \t "https://www.runoob.com/jsp/_blank) | 指定时区 |
| [<fmt:setTimeZone>](https://www.runoob.com/jsp/jstl-format-settimezone-tag.html" \t "https://www.runoob.com/jsp/_blank) | 指定时区 |
| [<fmt:message>](https://www.runoob.com/jsp/jstl-format-message-tag.html" \t "https://www.runoob.com/jsp/_blank) | 显示资源配置文件信息 |
| [<fmt:requestEncoding>](https://www.runoob.com/jsp/jstl-format-requestencoding-tag.html" \t "https://www.runoob.com/jsp/_blank) | 设置request的字符编码 |

## SQL标签

JSTL SQL标签库提供了与关系型数据库（Oracle，MySQL，SQL Server等等）进行交互的标签。引用SQL标签库的语法如下：

<%@ taglib prefix="sql"

uri="http://java.sun.com/jsp/jstl/sql" %>

|  |  |
| --- | --- |
| **标签** | **描述** |
| [<sql:setDataSource>](https://www.runoob.com/jsp/jstl-sql-setdatasource-tag.html" \t "https://www.runoob.com/jsp/_blank) | 指定数据源 |
| [<sql:query>](https://www.runoob.com/jsp/jstl-sql-query-tag.html" \t "https://www.runoob.com/jsp/_blank) | 运行SQL查询语句 |
| [<sql:update>](https://www.runoob.com/jsp/jstl-sql-update-tag.html" \t "https://www.runoob.com/jsp/_blank) | 运行SQL更新语句 |
| [<sql:param>](https://www.runoob.com/jsp/jstl-sql-param-tag.html" \t "https://www.runoob.com/jsp/_blank) | 将SQL语句中的参数设为指定值 |
| [<sql:dateParam>](https://www.runoob.com/jsp/jstl-sql-dateparam-tag.html" \t "https://www.runoob.com/jsp/_blank) | 将SQL语句中的日期参数设为指定的java.util.Date 对象值 |
| [<sql:transaction>](https://www.runoob.com/jsp/jstl-sql-transaction-tag.html" \t "https://www.runoob.com/jsp/_blank) | 在共享数据库连接中提供嵌套的数据库行为元素，将所有语句以一个事务的形式来运行 |

## XML 标签

JSTL XML标签库提供了创建和操作XML文档的标签。引用XML标签库的语法如下：

<%@ taglib prefix="x"

uri="http://java.sun.com/jsp/jstl/xml" %>

在使用xml标签前，你必须将XML 和 XPath 的相关包拷贝至你的<Tomcat 安装目录>\lib下:

* **XercesImpl.jar**

下载地址： [http://www.apache.org/dist/xerces/j/](http://www.apache.org/dist/xerces/j/" \t "https://www.runoob.com/jsp/_blank)

* **xalan.jar**

下载地址： [http://xml.apache.org/xalan-j/index.html](http://xml.apache.org/xalan-j/index.html" \t "https://www.runoob.com/jsp/_blank)

|  |  |
| --- | --- |
| **标签** | **描述** |
| [<x:out>](https://www.runoob.com/jsp/jstl-xml-out-tag.html" \t "https://www.runoob.com/jsp/_blank) | 与<%= ... >,类似，不过只用于XPath表达式 |
| [<x:parse>](https://www.runoob.com/jsp/jstl-xml-parse-tag.html" \t "https://www.runoob.com/jsp/_blank) | 解析 XML 数据 |
| [<x:set>](https://www.runoob.com/jsp/jstl-xml-set-tag.html" \t "https://www.runoob.com/jsp/_blank) | 设置XPath表达式 |
| [<x:if>](https://www.runoob.com/jsp/jstl-xml-if-tag.html" \t "https://www.runoob.com/jsp/_blank) | 判断XPath表达式，若为真，则执行本体中的内容，否则跳过本体 |
| [<x:forEach>](https://www.runoob.com/jsp/jstl-xml-foreach-tag.html" \t "https://www.runoob.com/jsp/_blank) | 迭代XML文档中的节点 |
| [<x:choose>](https://www.runoob.com/jsp/jstl-xml-choose-tag.html" \t "https://www.runoob.com/jsp/_blank) | <x:when>和<x:otherwise>的父标签 |
| [<x:when>](https://www.runoob.com/jsp/jstl-xml-choose-tag.html" \t "https://www.runoob.com/jsp/_blank) | <x:choose>的子标签，用来进行条件判断 |
| [<x:otherwise>](https://www.runoob.com/jsp/jstl-xml-choose-tag.html" \t "https://www.runoob.com/jsp/_blank) | <x:choose>的子标签，当<x:when>判断为false时被执行 |
| [<x:transform>](https://www.runoob.com/jsp/jstl-xml-transform-tag.html" \t "https://www.runoob.com/jsp/_blank) | 将XSL转换应用在XML文档中 |
| [<x:param>](https://www.runoob.com/jsp/jstl-xml-param-tag.html" \t "https://www.runoob.com/jsp/_blank) | 与<x:transform>共同使用，用于设置XSL样式表 |

## JSTL函数

JSTL包含一系列标准函数，大部分是通用的字符串处理函数。引用JSTL函数库的语法如下：

<%@ taglib prefix="fn"

uri="http://java.sun.com/jsp/jstl/functions" %>

|  |  |
| --- | --- |
| **函数** | **描述** |
| [fn:contains()](https://www.runoob.com/jsp/jstl-function-contains.html" \t "https://www.runoob.com/jsp/_blank) | 测试输入的字符串是否包含指定的子串 |
| [fn:containsIgnoreCase()](https://www.runoob.com/jsp/jstl-function-containsignoreCase.html" \t "https://www.runoob.com/jsp/_blank) | 测试输入的字符串是否包含指定的子串，大小写不敏感 |
| [fn:endsWith()](https://www.runoob.com/jsp/jstl-function-endswith.html" \t "https://www.runoob.com/jsp/_blank) | 测试输入的字符串是否以指定的后缀结尾 |
| [fn:escapeXml()](https://www.runoob.com/jsp/jstl-function-escapexml.html" \t "https://www.runoob.com/jsp/_blank) | 跳过可以作为XML标记的字符 |
| [fn:indexOf()](https://www.runoob.com/jsp/jstl-function-indexof.html" \t "https://www.runoob.com/jsp/_blank) | 返回指定字符串在输入字符串中出现的位置 |
| [fn:join()](https://www.runoob.com/jsp/jstl-function-join.html" \t "https://www.runoob.com/jsp/_blank) | 将数组中的元素合成一个字符串然后输出 |
| [fn:length()](https://www.runoob.com/jsp/jstl-function-length.html" \t "https://www.runoob.com/jsp/_blank) | 返回字符串长度 |
| [fn:replace()](https://www.runoob.com/jsp/jstl-function-replace.html" \t "https://www.runoob.com/jsp/_blank) | 将输入字符串中指定的位置替换为指定的字符串然后返回 |
| [fn:split()](https://www.runoob.com/jsp/jstl-function-split.html" \t "https://www.runoob.com/jsp/_blank) | 将字符串用指定的分隔符分隔然后组成一个子字符串数组并返回 |
| [fn:startsWith()](https://www.runoob.com/jsp/jstl-function-startswith.html" \t "https://www.runoob.com/jsp/_blank) | 测试输入字符串是否以指定的前缀开始 |
| [fn:substring()](https://www.runoob.com/jsp/jstl-function-substring.html" \t "https://www.runoob.com/jsp/_blank) | 返回字符串的子集 |
| [fn:substringAfter()](https://www.runoob.com/jsp/jstl-function-substringafter.html" \t "https://www.runoob.com/jsp/_blank) | 返回字符串在指定子串之后的子集 |
| [fn:substringBefore()](https://www.runoob.com/jsp/jstl-function-substringbefore.html" \t "https://www.runoob.com/jsp/_blank) | 返回字符串在指定子串之前的子集 |
| [fn:toLowerCase()](https://www.runoob.com/jsp/jstl-function-tolowercase.html" \t "https://www.runoob.com/jsp/_blank) | 将字符串中的字符转为小写 |
| [fn:toUpperCase()](https://www.runoob.com/jsp/jstl-function-touppercase.html" \t "https://www.runoob.com/jsp/_blank) | 将字符串中的字符转为大写 |
| [fn:trim()](https://www.runoob.com/jsp/jstl-function-trim.html" \t "https://www.runoob.com/jsp/_blank) | 移除首尾的空白符 |

[JSP 发送邮件](https://www.runoob.com/jsp/jsp-sending-email.html" \o "JSP 发送邮件)

[JSP 连接数据库](https://www.runoob.com/jsp/jsp-database-access.html" \o "JSP 连接数据库)

## **1 篇笔记 写笔记**

   stackoverflow

  hgn\*\*\*@163.com

[参考地址](https://stackoverflow.com/questions/4959223/where-can-i-download-jstl-taglibs-i-e-jstl-jar-and-standard-jar" \t "https://www.runoob.com/jsp/_blank)

JSTL 1.1 与 JSTL 1.2 之间的区别？如何下载 JSTL 1.2?

JSTL 1.2 中不要求 standard.jar 包。

您可以在 Maven 中央仓库中找到它们。

[http://repo2.maven.org/maven2/javax/servlet/jstl/](http://repo2.maven.org/maven2/javax/servlet/jstl/" \t "https://www.runoob.com/jsp/_blank)

[http://repo2.maven.org/maven2/taglibs/standard/](http://repo2.maven.org/maven2/taglibs/standard/" \t "https://www.runoob.com/jsp/_blank)

由于JSTL 1.1已经过时，Apache已将其置于[存档中](http://archive.apache.org/dist/jakarta/taglibs/standard/" \t "https://www.runoob.com/jsp/_blank)。选择jakarta-taglibs-standard-current.zip文件。但是，如果您正在运行Servlet 2.5兼容容器并且web.xml声明为至少Servlet 2.5，那么您应该能够使用新的[JSTL 1.2](http://central.maven.org/maven2/javax/servlet/jstl/1.2/jstl-1.2.jar" \t "https://www.runoob.com/jsp/_blank)。需要注意的是JSTL 1.2并没有要求standard.jar。

# JSP XML 数据处理

当通过HTTP发送XML数据时，就有必要使用JSP来处理传入和流出的XML文档了，比如RSS文档。作为一个XML文档，它仅仅只是一堆文本而已，使用JSP创建XML文档并不比创建一个HTML文档难。

## 使用JSP发送XML

使用JSP发送XML内容就和发送HTML内容一样。唯一的不同就是您需要把页面的context属性设置为text/xml。要设置context属性，使用<%@page % >命令，就像这样：

<%@ page contentType="text/xml" %>

接下来这个例子向浏览器发送XML内容：

<%@ page contentType="text/xml" %>

<books>

<book>

<name>Padam History</name>

<author>ZARA</author>

<price>100</price>

</book></books>

# JSP JavaBean

JavaBean是特殊的Java类，使用J ava语言书写，并且遵守JavaBean API规范。

接下来给出的是JavaBean与其它Java类相比而言独一无二的特征：

* 提供一个默认的无参构造函数。
* 需要被序列化并且实现了Serializable接口。
* 可能有一系列可读写属性。
* 可能有一系列的"getter"或"setter"方法。

## JavaBean属性

一个JavaBean对象的属性应该是可访问的。这个属性可以是任意合法的Java数据类型，包括自定义Java类。

一个JavaBean对象的属性可以是可读写，或只读，或只写。JavaBean对象的属性通过JavaBean实现类中提供的两个方法来访问：

|  |  |
| --- | --- |
| **方法** | **描述** |
| get**PropertyName**() | 举例来说，如果属性的名称为myName，那么这个方法的名字就要写成getMyName()来读取这个属性。这个方法也称为访问器。 |
| set**PropertyName**() | 举例来说，如果属性的名称为myName，那么这个方法的名字就要写成setMyName()来写入这个属性。这个方法也称为写入器。 |

一个只读的属性只提供getPropertyName()方法，一个只写的属性只提供setPropertyName()方法。

## 访问JavaBean

<jsp:useBean> 标签可以在JSP中声明一个JavaBean，然后使用。声明后，JavaBean对象就成了脚本变量，可以通过脚本元素或其他自定义标签来访问。<jsp:useBean>标签的语法格式如下：

<jsp:useBean id="bean 的名字" scope="bean 的作用域" typeSpec/>

其中，根据具体情况，scope的值可以是page，request，session或application。id值可任意只要不和同一JSP文件中其它<jsp:useBean>中id值一样就行了。

## 访问 JavaBean 对象的属性

在 **<jsp:useBean>** 标签主体中使用 **<jsp:getProperty/>** 标签来调用 **getter** 方法，使用 **<jsp:setProperty/>** 标签来调用 **setter** 方法，语法格式如下：

# JSP 自定义标签

自定义标签是用户定义的JSP语言元素。当JSP页面包含一个自定义标签时将被转化为servlet，标签转化为对被 称为tag handler的对象的操作，即当servlet执行时Web container调用那些操作。

JSP标签扩展可以让你创建新的标签并且可以直接插入到一个JSP页面。 JSP 2.0规范中引入Simple Tag Handlers来编写这些自定义标记。

你可以继承SimpleTagSupport类并重写的doTag()方法来开发一个最简单的自定义标签。

## 创建"Hello"标签

接下来，我们想创建一个自定义标签叫作<ex:Hello>，标签格式为：

<ex:Hello />

## 自定义标签属性

你可以在自定义标准中设置各种属性，要接收属性，值自定义标签类必须实现setter方法， JavaBean 中的setter方法如下所示：

你还可以包含以下属性：

|  |  |
| --- | --- |
| **属性** | **描述** |
| name | 定义属性的名称。每个标签的是属性名称必须是唯一的。 |
| required | 指定属性是否是必须的或者可选的,如果设置为false为可选。 |
| rtexprvalue | 声明在运行表达式时，标签属性是否有效。 |
| type | 定义该属性的Java类类型 。默认指定为 **String** |
| description | 描述信息 |
| fragment | 如果声明了该属性,属性值将被视为一个 **JspFragment**。 |

# JSP 表达式语言

JSP表达式语言（EL）使得访问存储在JavaBean中的数据变得非常简单。JSP EL既可以用来创建算术表达式也可以用来创建逻辑表达式。在JSP EL表达式内可以使用整型数，浮点数，字符串，常量true、false，还有null。

## 一个简单的语法

典型的，当您需要在JSP标签中指定一个属性值时，只需要简单地使用字符串即可：

<jsp:setProperty name="box" property="perimeter" value="100"/>

JSP EL允许您指定一个表达式来表示属性值。一个简单的表达式语法如下：

${expr}

其中，expr指的是表达式。在JSP EL中通用的操作符是 **.** 和 **{}** 。这两个操作符允许您通过内嵌的JSP对象访问各种各样的JavaBean属性。

举例来说，上面的<jsp:setProperty>标签可以使用表达式语言改写成如下形式：

<jsp:setProperty name="box" property="perimeter"

value="${2\*box.width+2\*box.height}"/>

当JSP编译器在属性中见到"${}"格式后，它会产生代码来计算这个表达式，并且产生一个替代品来代替表达式的值。

您也可以在标签的模板文本中使用表达式语言。比如<jsp:text>标签简单地将其主体中的文本插入到JSP输出中：

<jsp:text><h1>Hello JSP!</h1></jsp:text>

现在，在<jsp:text>标签主体中使用表达式，就像这样：

<jsp:text>

Box Perimeter is: ${2\*box.width + 2\*box.height}</jsp:text>

在EL表达式中可以使用圆括号来组织子表达式。比如${(1 + 2) \* 3}等于9，但是${1 + (2 \* 3)} 等于7。

想要停用对EL表达式的评估的话，需要使用page指令将isELIgnored属性值设为true：

<%@ page isELIgnored ="true|false" %>

这样，EL表达式就会被忽略。若设为false，则容器将会计算EL表达式。

## EL中的基础操作符

EL表达式支持大部分Java所提供的算术和逻辑操作符：

|  |  |
| --- | --- |
| **操作符** | **描述** |
| . | 访问一个Bean属性或者一个映射条目 |
| [] | 访问一个数组或者链表的元素 |
| ( ) | 组织一个子表达式以改变优先级 |
| + | 加 |
| - | 减或负 |
| \* | 乘 |
| / or div | 除 |
| % or mod | 取模 |
| == or eq | 测试是否相等 |
| != or ne | 测试是否不等 |
| < or lt | 测试是否小于 |
| > or gt | 测试是否大于 |
| <= or le | 测试是否小于等于 |
| >= or ge | 测试是否大于等于 |
| && or and | 测试逻辑与 |
| || or or | 测试逻辑或 |
| ! or not | 测试取反 |
| empty | 测试是否空值 |

## JSP EL中的函数

JSP EL允许您在表达式中使用函数。这些函数必须被定义在自定义标签库中。函数的使用语法如下：

${ns:func(param1, param2, ...)}

ns指的是命名空间（namespace），func指的是函数的名称，param1指的是第一个参数，param2指的是第二个参数，以此类推。比如，有函数fn:length，在JSTL库中定义，可以像下面这样来获取一个字符串的长度：

${fn:length("Get my length")}

要使用任何标签库中的函数，您需要将这些库安装在服务器中，然后使用<taglib>标签在JSP文件中包含这些库。

## JSP EL隐含对象

JSP EL支持下表列出的隐含对象：

|  |  |
| --- | --- |
| **隐含对象** | **描述** |
| pageScope | page 作用域 |
| requestScope | request 作用域 |
| sessionScope | session 作用域 |
| applicationScope | application 作用域 |
| param | Request 对象的参数，字符串 |
| paramValues | Request对象的参数，字符串集合 |
| header | HTTP 信息头，字符串 |
| headerValues | HTTP 信息头，字符串集合 |
| initParam | 上下文初始化参数 |
| cookie | Cookie值 |
| pageContext | 当前页面的pageContext |

您可以在表达式中使用这些对象，就像使用变量一样。接下来会给出几个例子来更好的理解这个概念。

## pageContext对象

pageContext对象是JSP中pageContext对象的引用。通过pageContext对象，您可以访问request对象。比如，访问request对象传入的查询字符串，就像这样：

${pageContext.request.queryString}

## Scope对象

pageScope，requestScope，sessionScope，applicationScope变量用来访问存储在各个作用域层次的变量。

举例来说，如果您需要显式访问在applicationScope层的box变量，可以这样来访问：applicationScope.box。

## param和paramValues对象

param和paramValues对象用来访问参数值，通过使用request.getParameter方法和request.getParameterValues方法。

举例来说，访问一个名为order的参数，可以这样使用表达式：${param.order}，或者${param["order"]}。

接下来的例子表明了如何访问request中的username参数：

<%@ page import="java.io.\*,java.util.\*" %><%

String title = "Accessing Request Param";

%><html><head><title><% out.print(title); %></title></head><body><center><h1><% out.print(title); %></h1></center><div align="center"><p>${param["username"]}</p></div></body></html>

param对象返回单一的字符串，而paramValues对象则返回一个字符串数组。

## header和headerValues对象

header和headerValues对象用来访问信息头，通过使用 request.getHeader方法和request.getHeaders方法。

举例来说，要访问一个名为user-agent的信息头，可以这样使用表达式：${header.user-agent}，或者${header["user-agent"]}。

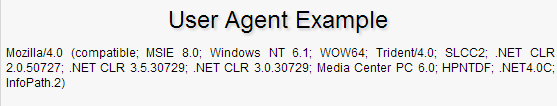
接下来的例子表明了如何访问user-agent信息头：

<%@ page import="java.io.\*,java.util.\*" %><%

String title = "User Agent Example";

%><html><head><title><% out.print(title); %></title></head><body><center><h1><% out.print(title); %></h1></center><div align="center"><p>${header["user-agent"]}</p></div></body></html>

运行结果如下：



header对象返回单一值，而headerValues则返回一个字符串数组。

# JSP 异常处理

当编写JSP程序的时候，程序员可能会遗漏一些BUG，这些BUG可能会出现在程序的任何地方。JSP代码中通常有以下几类异常:

* 检查型异常:检查型异常就是一个典型的用户错误或者一个程序员无法预见的错误。举例来说，如果一个文件将要被打开，但是无法找到这个文件，则一个异常被抛出。这些异常不能再编译期被简单地忽略。
* 运行时异常:一个运行时异常可能已经被程序员避免，这种异常在编译期将会被忽略。
* 错误:错误不是异常，但问题是它超出了用户或者程序员的控制范围。错误通常会在代码中被忽略，您几乎不能拿它怎么样。举例来说，栈溢出错误。这些错误都会在编译期被忽略。

本节将会给出几个简单而优雅的方式来处理运行时异常和错误。

## 使用Exception对象

exception对象是Throwable子类的一个实例，只在错误页面中可用。下表列出了Throwable类中一些重要的方法:

|  |  |
| --- | --- |
| **序号** | **方法&描述** |
| 1 | **public String getMessage()**  返回异常的信息。这个信息在Throwable构造函数中被初始化 |
| 2 | **public ThrowablegetCause()**  返回引起异常的原因，类型为Throwable对象 |
| 3 | **public String toString()**  返回类名 |
| 4 | **public void printStackTrace()**  将异常栈轨迹输出至System.err |
| 5 | **public StackTraceElement [] getStackTrace()**  以栈轨迹元素数组的形式返回异常栈轨迹 |
| 6 | **public ThrowablefillInStackTrace()**  使用当前栈轨迹填充Throwable对象 |

JSP提供了可选项来为每个JSP页面指定错误页面。无论何时页面抛出了异常，JSP容器都会自动地调用错误页面。

<%@page isErrorPage="true"%>指令，这个指令告诉JSP编译器需要产生一个异常实例变量。

## 在错误页面中使用JSTL标签

可以利用JSTL标签来编写错误页面ShowError.jsp。

## 使用 try…catch块

如果您想要将异常处理放在一个页面中，并且对不同的异常进行不同的处理，那么您就需要使用try…catch块了。

# JSP 调试

要测试/调试一个JSP或servlet程序总是那么的难。JSP和Servlets程序趋向于牵涉到大量客户端/服务器之间的交互，这很有可能会产生错误，并且很难重现出错的环境。

接下来将会给出一些小技巧和小建议，来帮助您调试程序。

## 使用System.out.println()

System.out.println()可以很方便地标记一段代码是否被执行。当然，我们也可以打印出各种各样的值。此外：

* 自从System对象成为Java核心对象后，它便可以使用在任何地方而不用引入额外的类。使用范围包括Servlets，JSP，RMI，EJB's，Beans，类和独立应用。
* 与在断点处停止运行相比，用System.out进行输出不会对应用程序的运行流程造成重大的影响，这个特点在定时机制非常重要的应用程序中就显得非常有用了。

接下来给出了使用System.out.println()的语法：

System.out.println("Debugging message");

## 使用JDB Logger

J2SE日志框架可为任何运行在JVM中的类提供日志记录服务。因此我们可以利用这个框架来记录任何信息。

消息可以使用各种优先级发送，通过使用sever()，warning()，info()，config()，fine()，finer()，finest()方法。finest()方法用来记录最好的信息，而sever()方法用来记录最严重的信息。

使用Log4J 框架来将消息记录在不同的文件中，这些消息基于严重程度和重要性来进行分类。

## 调试工具

NetBeans是树形结构，是开源的Java综合开发环境，支持开发独立的Java应用程序和网络应用程序，同时也支持JSP调试。

NetBeans支持如下几个基本的调试功能：

* 断点
* 单步跟踪
* 观察点

详细的信息可以查看NetBeans使用手册。

## 使用JDB Debugger

可以在JSP和servlets中使用jdb命令来进行调试，就像调试普通的应用程序一样。

通常，我们直接调试sun.servlet.http.HttpServer 对象来查看HttpServer在响应HTTP请求时执行JSP/Servlets的情况。这与调试applets非常相似。不同之处在于，applets程序实际调试的是sun.applet.AppletViewer。

大部分调试器在调试applets时都能够自动忽略掉一些细节，因为它知道如何调试applets。如果想要将调试对象转移到JSP身上，就需要做好以下两点：

* 设置调试器的classpath，让它能够找到sun.servlet.http.Http-Server  和相关的类。
* 设置调试器的classpath，让它能够找到您的JSP文件和相关的类。

设置好classpath后，开始调试sun.servlet.http.Http-Server 。您可以在JSP文件的任意地方设置断点，只要你喜欢，然后使用浏览器发送一个请求给服务器就应该可以看见程序停在了断点处。

## 使用注释

程序中的注释在很多方面都对程序的调试起到一定的帮助作用。注释可以用在调试程序的很多方面中。

JSP使用Java注释。如果一个BUG消失了，就请仔细查看您刚注释过的代码，通常都能找出原因。

## 客户端和服务器的头模块

有时候，当JSP没有按照预定的方式运行时，查看未加工的HTTP请求和响应也是很有用的。如果对HTTP的结构很熟悉的话，您可以直接观察request和response然后看看这些头模块到底怎么了。

## 重要调试技巧

这里我们再透露两个调试JSP的小技巧：

* 使用浏览器显示原始的页面内容，用来区分是否是格式问题。这个选项通常在View菜单下。
* 确保浏览器在强制重新载入页面时没有捕获先前的request输出。若使用的是Netscape Navigator浏览器，则用Shift-Reload；若使用的是IE浏览器，则用Shift-Refresh。

# JSP 国际化

在开始前，需要解释几个重要的概念：

* 国际化（i18n）：表明一个页面根据访问者的语言或国家来呈现不同的翻译版本。
* 本地化（l10n）：向网站添加资源，以使它适应不同的地区和文化。比如网站的印度语版本。
* 区域：这是一个特定的区域或文化，通常认为是一个语言标志和国家标志通过下划线连接起来。比如"en\_US"代表美国英语地区。

如果想要建立一个全球化的网站，就需要关心一系列项目。本章将会详细告诉您如何处理国际化问题，并给出了一些例子来加深理解。

JSP容器能够根据request的locale属性来提供正确地页面版本。接下来给出了如何通过request对象来获得Locale对象的语法：

java.util.Locale request.getLocale()

## 检测Locale

下表列举出了Locale对象中比较重要的方法，用于检测request对象的地区，语言，和区域。所有这些方法都会在浏览器中显示国家名称和语言名称：

|  |  |
| --- | --- |
| **序号** | **方法 & 描述** |
| 1 | **String getCountry()**  返回国家/地区码的英文大写，或 ISO 3166 2-letter 格式的区域 |
| 2 | **String getDisplayCountry()**  返回要显示给用户的国家名称 |
| 3 | **String getLanguage()**  返回语言码的英文小写，或ISO 639 格式的区域 |
| 4 | **String getDisplayLanguage()**  返回要给用户看的语言名称 |
| 5 | **String getISO3Country()**  返回国家名称的3字母缩写 |
| 6 | **String getISO3Language()**  返回语言名称的3字母缩写 |

## 语言设置

JSP可以使用西欧语言来输出一个页面，比如英语，西班牙语，德语，法语，意大利语等等。由此可见，设置Content-Language信息头来正确显示所有字符是很重要的。

第二点就是，需要使用HTML字符实体来显示特殊字符，比如"&#241;" 代表的是"?"，"&#161;"代表的是 "?" ：

## 区域特定日期

可以使用java.text.DateFormat类和它的静态方法getDateTimeInstance()来格式化日期和时间。接下来的这个例子显示了如何根据指定的区域来格式化日期和时间：

## 区域特定货币

可以使用java.text.NumberFormat类和它的静态方法getCurrencyInstance()来格式化数字。比如在区域特定货币中的long型和double型。

## 区域特定百分比

可以使用java.text.NumberFormat类和它的静态方法getPercentInstance()来格式化百分比。接下来的例子告诉我们如何根据指定的区域来格式化百分比：