



第2章 JSP应用基础

♥2.1 HTML语言

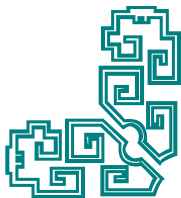
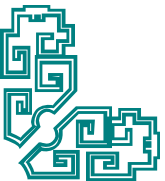
♥2.2 JSP语法

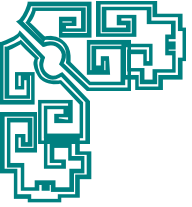
♥2.3 JSP内置对象

♥2.4 JavaBean及其应用

♥2.5 Servlet应用

♥2.6 JSP综合应用实例——开发一个简单的留言系统





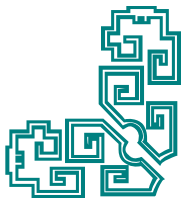
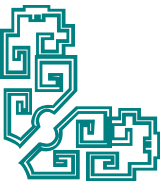
第2章 JSP应用基础

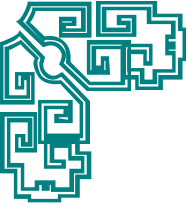
Java Web是原Sun公司（已被Oracle公司收购）在Java Servlet规范中提出的通用技术，指的是仅用HTML/JSP、JavaBean和Servlet等开发互联网Web应用的一系列技术的总和。Java Web是Java EE 程序的传统开发方式，也是Java EE的基础。

回顾：

Java EE的两种开发方式是？

Web开发
框架开发





2.1 HTML语言



2.1.1 HTML文件的基本结构

HTML文件的基本结构如图2.3所示。

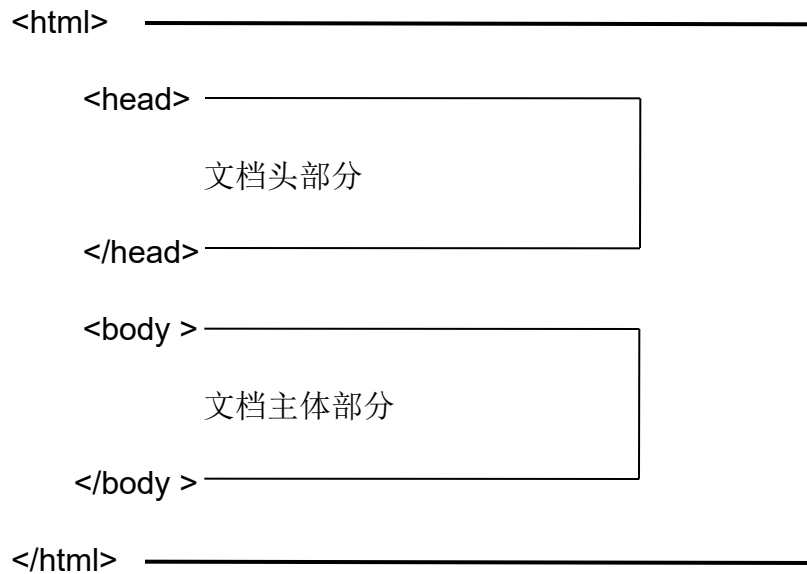
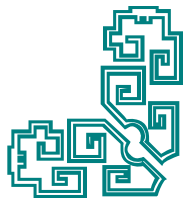
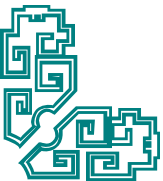


图2.3 HTML文件的基本结构



2.1.1 HTML文件的基本结构

大家可以看下面的例子，文档头部分有<title>网页标题标记，文档主体部分有图片标记、<a>...超链接标记及一些文字。把这段代码命名为a.html，保存在想要保存的路径。双击该文件就可以运行了。只要该路径下有b.html文件，单击超链接就可以跳转到b.html。

```
<html>
<head>
  <title>第一个html网页</title>
</head>
<body>
  
  <br>
  <a href="b.html">超链接</a>
  <hr align=center>
  这是我的第一个网页
</body>
</html>
```

注意：文件编码格式！！！！



2.1.2 HTML文件的语言标记

◇ 1. HTML标记

`<html>...</html>`

HTML标记表示文档内容的开始和结束。`<html>`是开始标记，`</html>`是结束标记，其他所有的HTML代码都位于这两个标记之间。浏览器将该标记中的内容视为一个Web文档，按照HTML语言规则对文档内的标记进行解释。

`<html>...</html>`标记是可选的，但是最好不要省略这两个标记，以保持Web文档结构的完整性。

◇ 2. 首部标记

`<head>...</head>`

首部标记中提供与网页有关的各种信息。在首部标记中，一般使用下列标记。

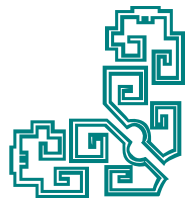
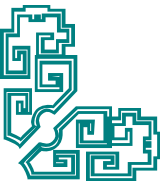
`<title>...</title>`：制定网页的标题。

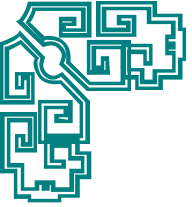
`<style>...</style>`：定义文档内容样式表。

`<script>... </script>`：插入脚本语言程序。（body调用的，出现在head里）

`<meta>`：描述网页信息。

这些信息首先向浏览器提供，但不作为文档内容提交。





2.1.2 HTML文件的语言标记

◇ 3. 标题栏标记

`<title>...</title>`

标题栏标记的内容是在浏览器标题栏中显示的文本。通常，Web搜索工具用它作为索引。

◇ 4. 描述标记

`<meta 属性="值"...>`

描述文档属性参数。

◇ 5. 正文标记

正文标记的格式如下：

`<body 属性="值"...事件="执行的程序"...>...</body>`

正文标记中包含文档的内容。其常用属性如下：

background: 文档背景图像的URL 地址。

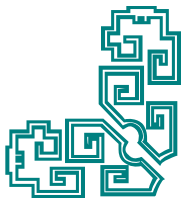
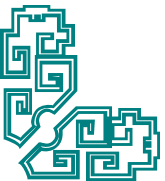
bgscolor: 文档的背景颜色。

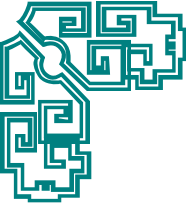
text: 文档中文本的颜色。

link: 文档中链接的颜色。

vlink: 文档中已被访问过的链接的颜色。

alink: 文档中正被选中的链接的颜色。





2.1.2 HTML文件的语言标记

此外，HTML文件还有很多用来设置文本格式的标记，下面举几个常用的标记。

（1）分段标记

`<p 属性="值"...></p>`

段落是文档的基本信息单位。利用分段标记，可以忽略文档中原有的回车和换行。定义一个新段落，就是换行并插入一个空行。

单独使用`<p>`标记时会空一行，使后续内容隔一行显示。同时使用`<p></p>`，则将段落包围起来，表示一个分段的块。其最常用的属性如下。

- **align: 段落的水平对齐方式**，其值如下：

left: 左对齐（默认值）。

center: 居中对齐。

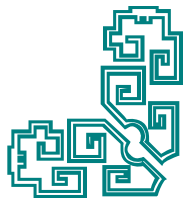
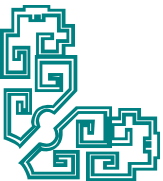
right: 右对齐。

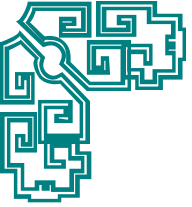
justify: 两边对齐。

```
<p align=left color=red>
```

```
XXXXXXXXXX
```

```
</p>
```





2.1.2 HTML文件的语言标记

(2) 换行标记

`
`

该标记强行中断当前行，使后续内容在下一行显示，这个标记很简单，也很常用。

(3) 标题标记

标题标记的格式如下：

`<h1 属性="值"...>...</h1>`

`<h2 属性="值"...>...</h2>`

`<h3 属性="值"...>...</h3>`

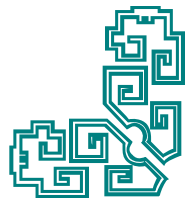
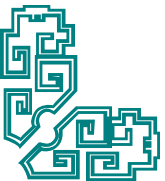
`<h4 属性="值"...>...</h4>`

`<h5 属性="值"...>...</h5>`

`<h6 属性="值"...>...</h6>`

其常用属性如下。

- **align**：段落的水平对齐方式，其值如下：
left：左对齐（默认值）。
center：居中对齐。
right：右对齐。
justify：两边对齐。





2.1.2 HTML文件的语言标记

（4）对中标记

`<center>...</center>`

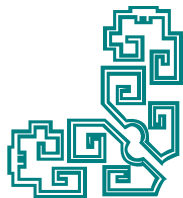
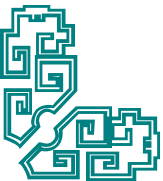
该标记中间的内容全部居中。

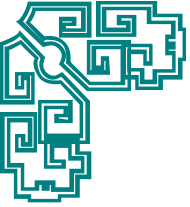
（5）块标记(逻辑块，不是程序块)

`<div 属性="值">...</div>`

块标记的作用是定义文档块。常用属性如下。

- **align:** 段落的水平对齐方式。其值如下：
left: 左对齐（默认值）。
center: 居中对齐。
right: 右对齐。





2.1.2 HTML文件的语言标记

(6) 水平线标记

`<hr 属性="值"...>`

在`<hr>`标记位置画一条线。常用属性如下。

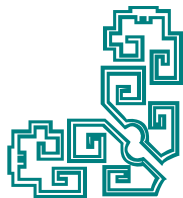
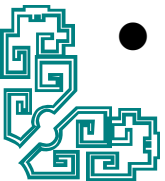
- **align**: 段落的水平对齐方式。其值如下:
left: 左对齐（默认值）。
center: 居中对齐。
right: 右对齐。
- **color**: 线的颜色。
- **size**: 线的宽度（以像素为单位）。
- **width**: 线的长度（像素或占页面宽度的百分数）。
- **noshade**: 显示一条无阴影的实线。

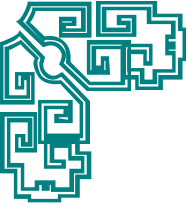
(7) 字体标记

`...`

字体标记用来设置文本的字符格式，主要包括字体、字号和颜色等。常用属性如下。

- **face**: 字体名表。。
- **size**: 字号值。
- **color**: 颜色值。设置字体的颜色。





2.1.2 HTML文件的语言标记

（8）图像标记

``

图像标记的常用属性如下。

- **src**: 图像文件的URL 地址。
- **alt**: 图像的简单文本说明，在浏览器下不能显示图像或图像加载时间过长时显示该文本。
- **height**: 显示图像的高度（像素或百分比）。
- **width**: 显示图像的宽度（像素或百分比）。
- **align**: 图像大小小于显示区域大小时的对齐方式。

使用**align**属性设置图像与文本在垂直方向的对齐方式，此时**align**属性的取值如下：

top: 图像与文本顶部对齐。

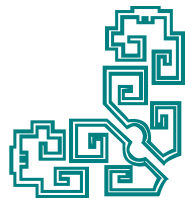
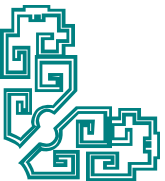
middle: 图像与文本中央对齐。

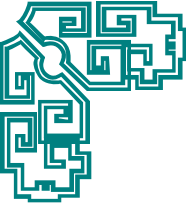
bottom: 图像与文本底部对齐。

当图像在左右绕排文本时，**align**属性的取值如下：

left: 图像居左，文本居右。

right: 图像居右，文本居左。





2.1.2 HTML文件的语言标记

（9）超链接标记

`<a 属性: "值"...>`超链接内容``

超链接的常用属性如下。

- **href:** 目标端点的URL 地址（可以包含一个或多个参数）。

如前面的例子中：

``超链接``

单击此超链接，就会跳转到名为b.html的页面。该属性是必选项。

- **target:** 窗口或框架的名称。

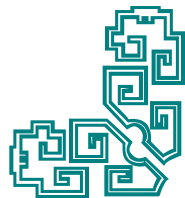
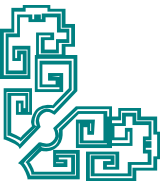
target属性的取值既可以是窗口或框架的名称，也可以是如下保留字：

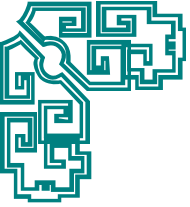
_blank: 未命名的新浏览器窗口。

_parent: 父框架页或窗口。如果包含链接的框架不是嵌套的，则链接的目标文件加载到整个浏览器窗口中。

_self: 所在的同一框架或窗口。

_top: 整个浏览器的窗口，并删除所有框架。





2.1.3 表单与表格的使用

◆ 1. 表单及其控件标记

表单用来从用户（站点访问者）处收集信息，然后将这些信息提交给服务器进行处理。表单中可以包含允许用户进行交互的各种控件，例如，文本框、列表框、复选框和单选按钮等。用户在表单中输入或选择数据后提交，该数据就会提交到相应的表单处理程序，以各种不同的方式进行处理。表单结构如下：

`<form 定义>`

`[<input 定义>]`

`[<textarea定义>]`

`[<select定义>]`

`[<button定义>]`

`</form>`

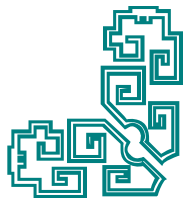
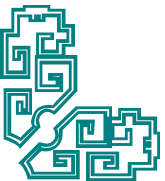
（1）表单标记

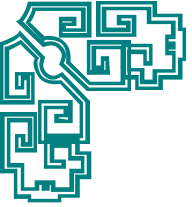
该标记的格式如下：

`<form 属性="值"...事件="代码">`

`...`

`</form>`





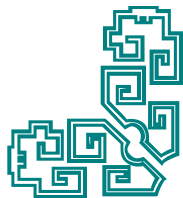
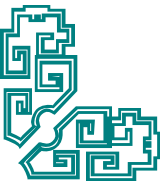
2.1.3 表单与表格的使用

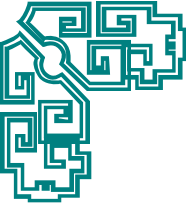
form标记的属性如下。

- **name**: 表单的名称。
- **method**: 表单数据传输到服务器的方法。其属性值如下:
 - post**: 在HTTP请求中嵌入表单数据。
 - get**: 将表单数据附加到请求该页的URL中。（可以在浏览器的地址目录中查看）
- **action**: 接收表单数据的服务器端程序或动态网页的URL地址。（数据提交后交给谁处理）
- **target**: 目标窗口。其属性值如下:
 - _blank**: 在未命名的新窗口中打开目标文档。
 - _parent**: 在显示当前文档的窗口的父窗口中打开目标文档。
 - _self**: 在提交表单所使用的窗口中打开目标文档。
 - _top**: 在当前窗口内打开目标文档，确保目标文档占用整个窗口。

form标记有以下事件。

- **onsubmit**: 提交表单时调用的时间处理程序。
- **onreset**: 重置表单时调用的处理程序。





2.1.3 表单与表格的使用

下面具体介绍表单中的控件。

（2）表单输入控件标记

`<input 属性="值" ... 事件="代码"...>`

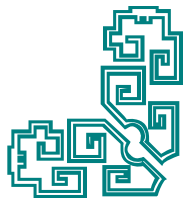
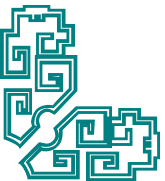
① 单行文本框。

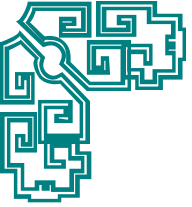
创建单行文本框方法如下：

`<input type="text" 属性="值" ... 事件="代码"...>`

（a）单行文本框的属性如下：

- **name**：单行文本框的名称，通过它可以在脚本中引用该文本框控件。
- **value**：文本框的值。
- **default value**：文本框的初始值。
- **size**：文本框的宽度（字符数）。
- **maxlength**：允许在文本框内输入的最大字符数。
- **form**：所属的表单（只读）。





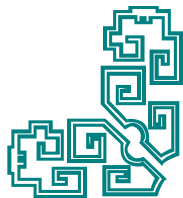
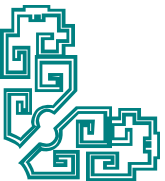
2.1.3 表单与表格的使用

(b) 单行文本框的方法如下。

- `click()`: 单击该文本框。
- `focus()`: 得到焦点。
- `blur()`: 失去焦点。
- `select()`: 选择文本框的内容。

(c) 单行文本框的事件如下。

- `onclick`: 单击该文本框时执行的代码。
- `onblur`: 失去焦点时执行的代码。
- `onchange`: 内容变化时执行的代码。
- `onfocus`: 得到焦点时执行的代码。
- `onselect`: 选择内容时执行的代码。





2.1.3 表单与表格的使用

② 密码文本框。

创建密码文本框方法如下：

`<input type="password" 属性 = "值"...事件="代码"...>`

密码文本框的属性、方法和事件与单行文本框的设置基本相同，只是密码文本框没有onclick事件。

③ 隐藏域。

创建隐藏域方法如下：

`<input type="hidden" 属性 = "值" ...>`

隐藏域的属性、方法和事件与单行文本框的设置基本相同，只是没有default value属性。

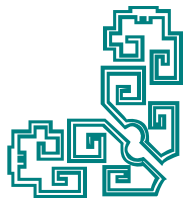
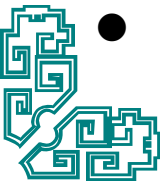
④ 复选框。

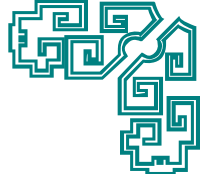
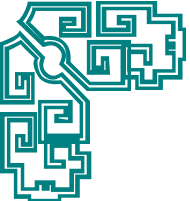
创建复选框方法如下：

`<input type="checkbox" 属性 = "值" ... 事件="代码" ...>选项文本`

（a）复选框的属性如下。

- name: 复选框的名称。
- value: 选中时提交的值。
- checked: 当第一次打开表时该复选框处于选中状态。
- defaultchecked: 判断复选框是否定义了checked属性。





2.1.3 表单与表格的使用

(b) 复选框的方法如下。

- **focus()**: 得到焦点。
- **blur()**: 失去焦点。
- **click()**: 单击该复选框。

(c) 复选框的事件如下。

- **onfocus**: 得到焦点时执行的代码。
- **onblur**: 失去焦点时执行的代码。
- **onclick**: 单击该文本框时执行的代码。

但值为空白。例如，要创建以下复选框：
应在body体内设置代码如下：

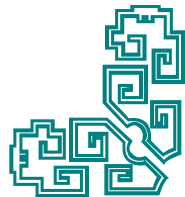
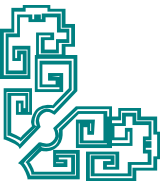
☒ 苹果 ☐ 香蕉 ☐ 橘子

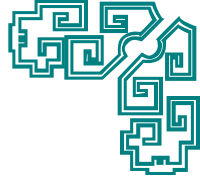
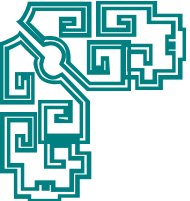
`<input name="fruit" type="checkbox" checked="">苹果`

`<input name="fruit" type="checkbox">香蕉`

`<input name="fruit" type="checkbox">橘子`

通过name来区分不同的复选框，做不同的组合。





2.1.3 表单与表格的使用

⑤ 单选按钮。

创建单选按钮方法如下：

`<input type="radio" 属性 = "值"...事件="代码"...>`选项文本

单选按钮的属性如下。

- **name:** 单选按钮的名称，若干个名称相同的单选按钮构成一个控件组，在该组中只能选中一个选项。
- **value:** 提交时的值。
- **checked:** 当第一次打开表单时该单选按钮处于选中状态。该属性是可选的。

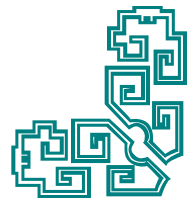
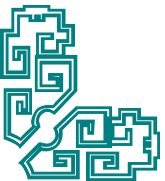
例如，创建以下单选按钮：

性别： ☒男 ☐女

应在body中设置代码如下：

性别： `<input name="sex" type="radio" checked>`男

`<input name="sex" type="radio">`女



2.1.3 表单与表格的使用

⑥ 按钮。

使用input标记可以在表单中添加3种类型的按钮：**提交按钮**、**重置按钮**和**自定义按钮**。创建按钮的方法如下：

`<input 属性="值"...onclick="代码">`

（a）按钮的属性如下。

- **type**：按钮种类。具体如下。
submit：创建一个提交按钮。
reset：创建一个重置按钮。
button：创建一个自定义按钮。

- **name**：按钮的名称。
- **value**：显示在按钮上的标题文本。

（b）按钮的事件如下。

- **onclick**：单击按钮执行的脚本代码。

⑦ 文件域。

创建文件域方法如下：

`<input type="file" 属性="值"...>`

其中，“属性：'值'”部分可以进行如下设置：

- **name**：文件域的名称。
- **value**：初始文件名。
- **size**：文件名输入框的宽度。



2.1.3 表单与表格的使用

（3）其他表单控件

下面分别介绍这两种控件的用法。

① 滚动文本框。

`<textarea 属性="值"...事件="代码"...>初始值</textarea>`

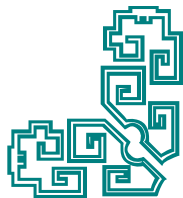
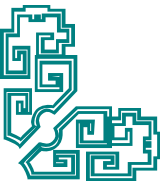
其属性如下：

- **name**：滚动文本框控件的名称。
 - **rows**：控件的高度。
 - **cols**：控件的宽度。
 - **readonly**：表示文本框中的内容是只读的，不能被修改。
- 该标记的其他属性、方法和相关事件与单行文本框基本相同。

② 选项选单。

创建选项选单方法如下：

```
<select name= "值" size="值" [multiple]>  
    <option[selected] value="值">选项1</option>  
    <option[selected] value="值">选项2</option>  
    ...  
</select>
```



2.1.3 表单与表格的使用

其属性如下：

- **name**：选项选单控件的名称。
- **size**：在列表中一次可以看到的选项数目，默认值为1。
- **multiple**：允许做多项选择。
- **selected**：该选项的初始状态为选中。

下面就这些控件做一个综合的例子来简单体验这些控件的用法，界面如图2.4所示。

综合展现HTML标记

下面展示表单的应用

姓名:

密码:

性别: ☒ 男 ☐ 女

水果: ☒ 苹果 ☐ 香蕉 ☐ 橘子

滚动文本框

备注:

专业:

课程:

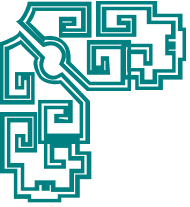
图2.4 综合展现HTML表单标记实例

2.1.3 表单与表格的使用

◇ 2. 表格的使用

表格由表头、行和单元格组成，常用于组织和显示信息，还可以用于安排页面布局。表格的表头、行和单元格分别用不同的标记来定义。可以用**table**标记定义表格；表格中的每一行用**tr**标记来表示；行中的单元格用**td**或**th**标记定义。其中**th**标记定义表格的列标题单元格，表格的标题说明则用**caption**标记来定义。定义表格的格式如下：

```
<table>
  <caption>表格标题文字</caption>
  <tr>
    <th>第1列表头</th><th>第2列表头</th>...<th>第n列表头</th>
  </tr>
  <tr>
    <td>第1行第1列值</td><td>第1行第2列值</td>...<td>第1行第n列值</td>
  </tr>
  <tr>
    ...
    <td>第n行第1列值</td><td>第n行第2列值</td>...<td>第n行第n列值</td>
  </tr>
  ...
</table>
```

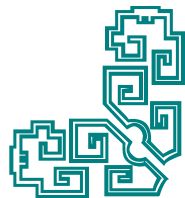
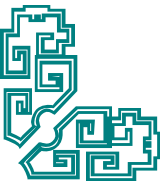


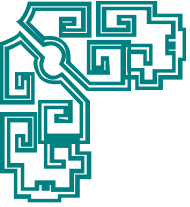
2.1.3 表单与表格的使用

（1）设置表格的属性

用**table**标记创建表格时，可以设置以下属性。

- **align**: 表格的对齐方式。其属性值如下：
left: 左对齐（默认值）。
center: 居中对齐。
right: 右对齐。
- **background**: 表格背景图片的URL地址。
- **bgcolor**: 表格的背景颜色。
- **border**: 表格边框的宽度（像素），默认值为0。
- **bordercolor**: 表格边框的颜色，**border**≠0时起作用。
- **bordercolordark**: 三维边框的阴影颜色，**border**≠0时起作用。
- **bordercolorlight**: 三维边框的高亮显示颜色，**border**≠0时起作用。
- **cellpadding**: 单元格内数据与单元格边框之间的间距（像素）。
- **width**: 表格的宽度（像素或百分比）。



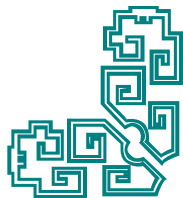
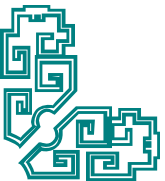


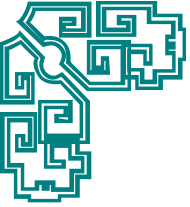
2.1.3 表单与表格的使用

（2）设置行的属性

表格中的每一行是用|标记来定义的，可以设置下列属性。
| |

- **align**: 行中单元格的水平对齐方式。其属性值如下：
left: 左对齐（默认值）。
center: 居中对齐。
right: 右对齐。
- **background**: 行的背景图片的URL地址。
- **bgcolor**: 行的背景颜色。
- **bordercolor**: 行的边框颜色，只有当table标记的border≠0时起作用。
- **bordercolordark**: 行的三维边框的阴影颜色，只有当table标记的border≠0时起作用。
- **bordercolorlight**: 行的三维边框的高亮显示颜色，只有当table标记的border≠0时起作用。
- **valign**: 行中单元格内容的垂直对齐方式。其属性值如下：
top: 顶端对齐。
middle: 居中对齐。
bottom: 底端对齐。
baseline: 基线对齐。



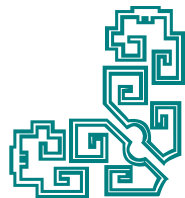
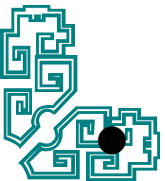


2.1.3 表单与表格的使用

（3）设置单元格的属性

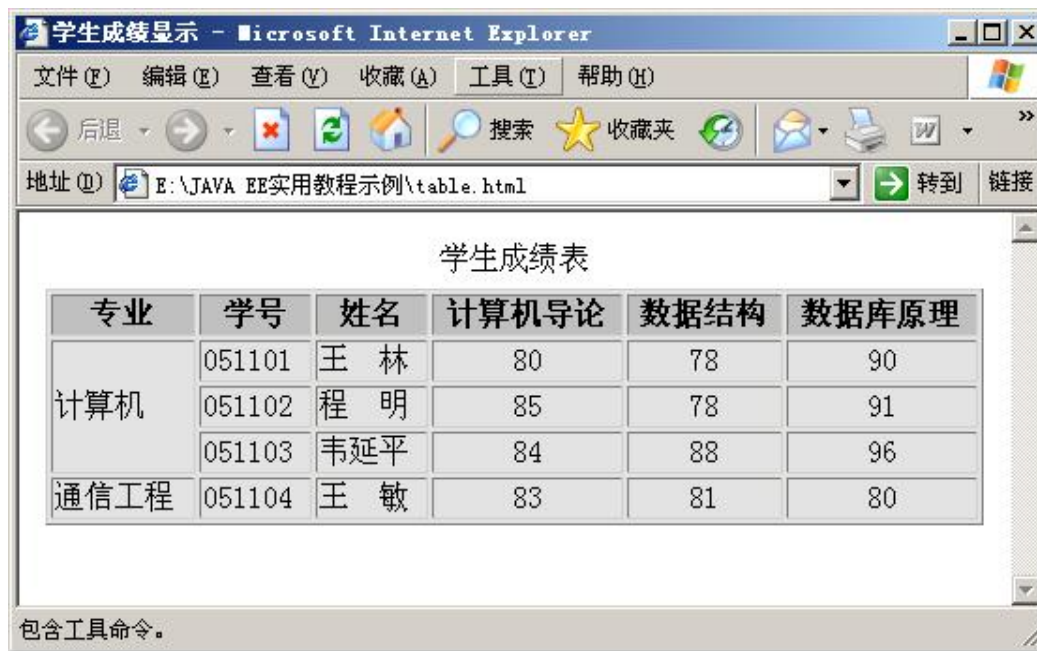
td标记和th标记的属性如下。

- **align**: 行中单元格的水平对齐方式。其属性值如下：
left: 左对齐（默认值）。
center: 居中对齐。
right: 右对齐。
- **background**: 单元格的背景图片的URL 地址。
- **bgcolor**: 单元格的背景颜色。
- **bordercolor**: 单元格的边框颜色，只有当table标记的border≠0时起作用。
- **bordercolordark**: 单元格的三维边框的阴影颜色，只有当table标记的border≠0时起作用。
- **bordercolorlight**: 单元格的三维边框的高亮显示颜色，只有当table标记的border≠0时起作用。
- **colspan**: 合并单元格时一个单元格跨越的表格列数。
- **rowspan**: 合并单元格时一个单元格跨越的表格行数。
- **valign**: 单元格中文本的垂直对齐方式。其属性值如下：
top: 顶端对齐。
middle: 居中对齐。
bottom: 底端对齐。
baseline: 基线对齐。
- **nowrap**: 若指定该属性，则要避免Web浏览器将单元格里文本换行。



2.1.3 表单与表格的使用

表格示例：



The screenshot shows a Microsoft Internet Explorer window titled "学生成绩显示 - Microsoft Internet Explorer". The address bar displays "E:\JAVA EE实用教程示例\table.html". The main content area shows a table titled "学生成绩表". The table has 6 columns: "专业" (Major), "学号" (Student ID), "姓名" (Name), "计算机导论" (Introduction to Computer Science), "数据结构" (Data Structures), and "数据库原理" (Database Principles). The data is as follows:

专业	学号	姓名	计算机导论	数据结构	数据库原理
计算机	051101	王 林	80	78	90
	051102	程 明	85	78	91
	051103	韦延平	84	88	96
通信工程	051104	王 敏	83	81	80

At the bottom of the window, there is a status bar that says "包含工具命令。"

图2.5 表格的展示

2.1.3 表单与表格的使用

利用

综合实例 - Microsoft Internet Explorer

文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)

后退 前进 搜索 收藏夹 转送到 链接

地址(A) E:\JAVA EE实用教程示例\form_table.html

综合展现HTML标记

下面展示表单的应用

姓名:

密码:

性别: ☒ 男 ☐ 女

水果: ☒ 苹果 ☐ 香蕉 ☐ 橘子

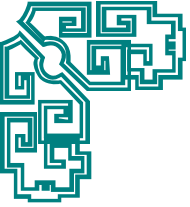
备注:

专业:

课程:

包含工具命令。

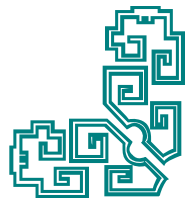
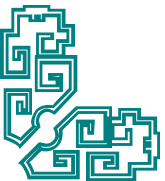
图2.6 使用表格后的表单

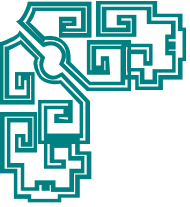


2.1.4 多框架文档

框架网页通过一个**frameset**（框架集）标记和多个**frame**（框架）标记来定义。在框架网页中，将**frameset**标记置于**head**之后，以取代**body**的位置，还可以使用**noframes**标记生成不能被浏览器显示时的替换内容。框架网页的基本结构如下：

```
<html>
<head>
  <title>框架网页的基本结构</title>
</head>
  <frameset属性="值"...>
    <frame属性="值"...>
    <frame属性="值"...>
    <frame属性="值"...>
    ...
  </frameset>
</html>
```





2.1.4 多框架文档

◇ 1. 框架集

`<frameset属性="值"...>`

...

`</frameset>`

可以使用**frameset**标记的下列属性对框架的结构进行设置。

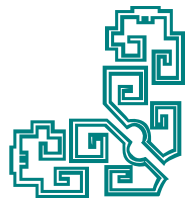
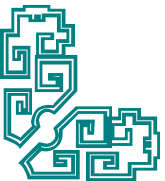
cols: 创建纵向分隔框架时指定各个框架的列宽。取值有3种形式，即像素、百分比（%）和相对尺寸（*）。例如：

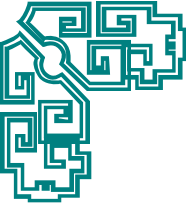
`cols="*, *, *"`：表示将窗口划分成3个等宽的框架；

`cols="30%, 200, *"`：表示将浏览器窗口划分为3个框架，其中第1个占窗口宽度的30%，第2个为200像素，第3个为窗口的剩余部分；

`cols="*, 3*, 2*"`：表示左边的框架占窗口宽度的1/6，中间的框架占窗口宽度的1/2，右边的框架占窗口宽度的1/3。

- **rows**: 横向分隔框架时各个框架的行高。
- **frameborder**: 框架周围是否显示三维边框。
- **framespacing**: 框架之间的间隔（以像素为单位，默认值为0）。

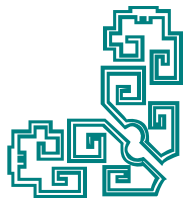
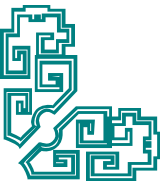


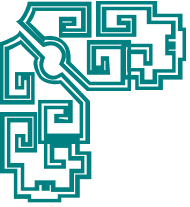


2.1.4 多框架文档

例如：创建一个嵌套框架集。

```
<html>
<head>
  <title>创建框架网页</title>
</head>
  <frameset rows="20%,400,* "> //把框架分为3个部分（行分），分别是20%、400及剩余部分
    <frame>
      <frameset cols="300,* "> //将第一行部分分为2列，300及剩余部分
        <frame>
        <frame>
      </frameset>
    </frame>
  <noframes>
    <body>
      <p>此网页使用了框架，但您的浏览器不支持框架。</p>
    </body>
  </noframes>
</frameset>
</html>
```





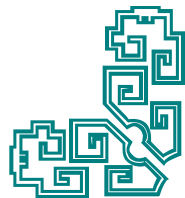
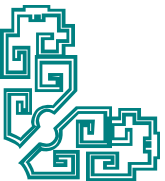
2.1.4 多框架文档

◇ 2. 框架

`<frame 属性="值"...>`

`frame`标记具有下列属性。

- **name**: 框架的名称。
- **frameboder**: 框架周围是否显示三维边框。
- **marginheight**: 框架的高度（以像素为单位）。
- **marginwidth**: 框架的宽度（以像素为单位）。
- **noresize**: 不能调整框架的大小。
- **scrolling**: 指定框架是否可以滚动。其属性值如下：
 - yes**: 框架可以滚动。
 - no**: 框架不能滚动。
 - auto**: 框架在需要时添加滚动条。
- **src**: 在框架中显示的HTML文件。



2.1.4 多框架文档

下面结合表格及表单的特性综合展现它们的应用。首先看实现主界面，如图2.7所示。

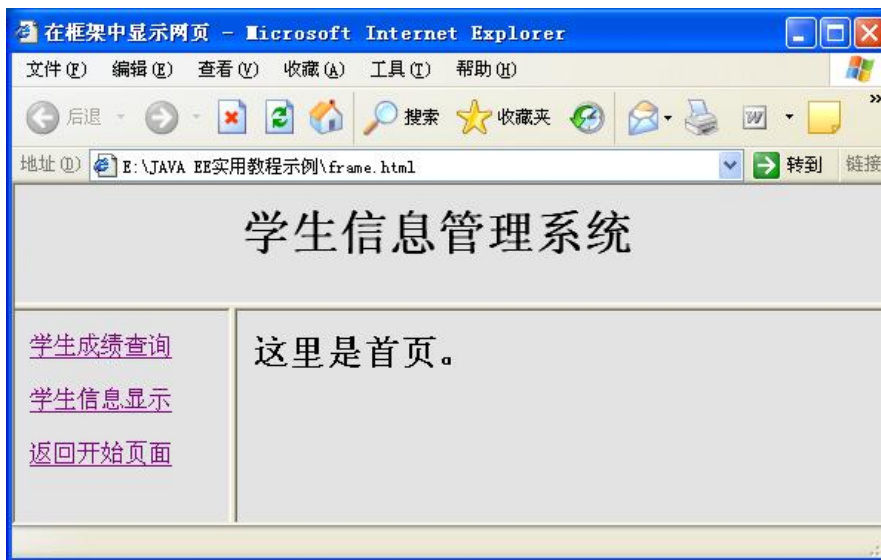


图2.7 学生信息管理系统主界面

2.1.4 多框架文档

单击【学生成绩查询】超链接，出现如图2.8所示的界面。



图2.8 学生成绩查询界面

2.1.4 多框架文档

单击【学生信息显示】超链接，出现如图2.9所示的界面。

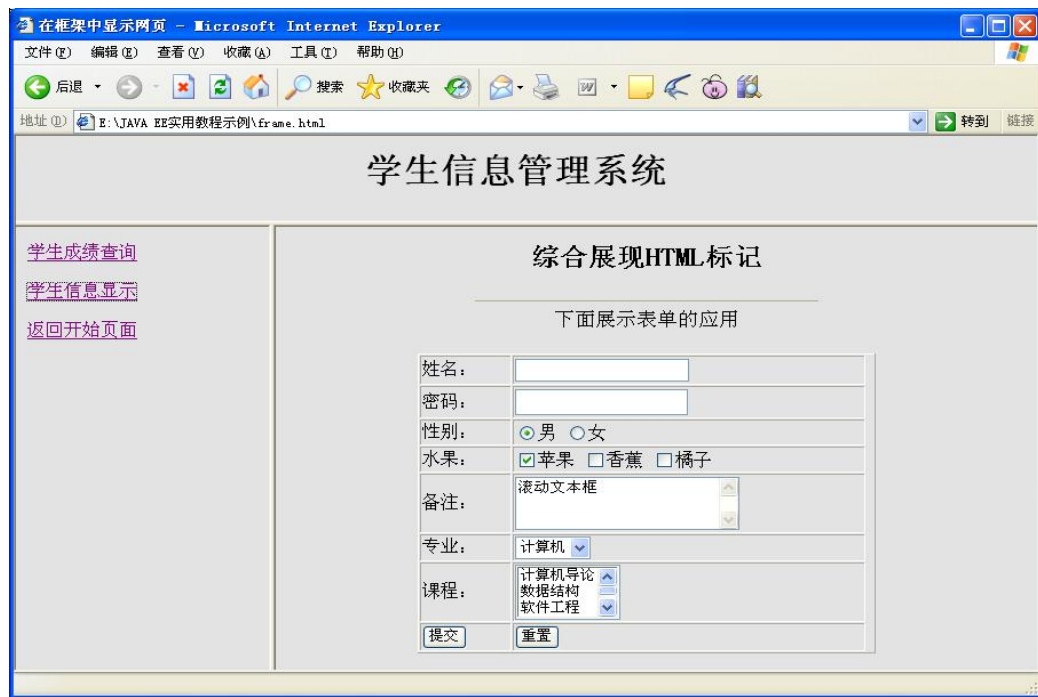
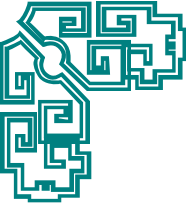


图2.9 学生信息显示界面



2.1.4 多框架文档

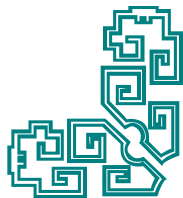
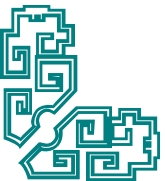
代码实现如下：

(1) head.html

```
<html>
<head>
</head>
<body bgcolor="#E3E3E3">
    <center><h1>学生信息管理系统</h1></center>
</body>
</html>
```

(2) left.html

```
<html>
<head>
</head>
<body bgcolor="#e3e3e3">
    <a href="table.html" target="right">学生成绩查询<br><br>
    <a href="form_table.html" target="right">学生信息显示<br><br>
    <a href="right.html" target="right">返回开始页面<br><br>
</body>
</html>
```

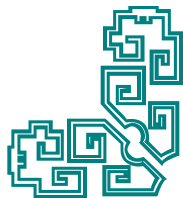
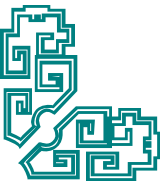


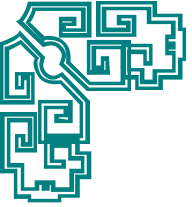


2.1.4 多框架文档

(3) right.html

```
<html>
<head>
</head>
<body bgcolor="#e3e3e3">
    <h2 align="center">这里是首页。</h2>
</body>
</html>
```





2.1.4 多框架文档

(4) frame.html

<html>

<head><title>在框架中显示网页</title></head>

 <frameset rows="80,*">

 <frame src ="head.html" name ="head">

 <frameset cols ="25%,*">

 <frame src ="left.html" name ="left">

 <frame src ="right.html" name ="right">

 </frameset>

 <noframes>

 <body>

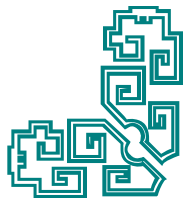
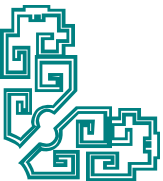
 此网页使用了框架，但您的浏览器不支持框架

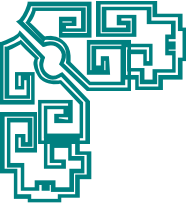
 </body>

 </noframes>

 </frameset>

</html>





2.2 JSP语法

JSP示例：

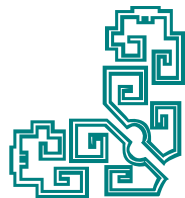
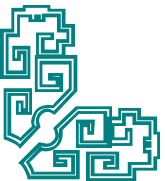
启动Tomcat服务器，在IE浏览器中输入“<http://localhost:8080/input.jsp>”，系统会显示input.jsp页面，如图2.1所示。当在文本框中输入10后，单击【计算】按钮会跳转到另外一个页面，也就是result.jsp页面，并且输出结果314.0，如图2.2所示。

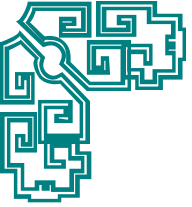


图2.1 input.jsp页面



图2.2 result.jsp页面

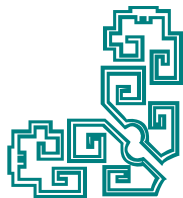
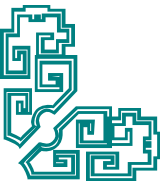




2.2 JSP语法

JSP（Java Server Pages）是由原Sun公司倡导、许多家公司参与一起建立的一种动态网页技术标准。

它是在传统的网页HTML文件（*.htm， *.html）中插入Java程序段（Scriptlet）和JSP标记（tag），从而形成的JSP文件（*.jsp）。





2.2 JSP语法

✧ 2.2.1 JSP数据定义

在JSP中可以用`<%! 和 %>`定义一个或多个变量。在其中定义的变量为该页面级别的共享变量，可以被访问此页面的所有用户访问。其语法格式如下：

`<%! 变量声明 %>`//全局代码块

如下面的代码片段：

```
<%!  
    String name="liu";  
    int i=0;
```

```
%>
```

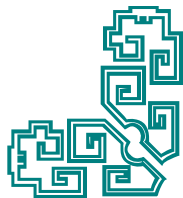
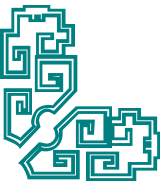
此外，这种声明方式还可以定义一个方法或类，定义方法的格式如下：

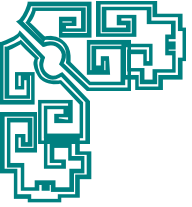
```
<%!  
    返回值数据类型 函数名(数据类型, 参数, ...){  
        语句;  
        return (返回值);  
    }
```

```
%>
```

定义一个类，如下面的代码片段：

```
<%!  
    public class A{...}  
%>
```



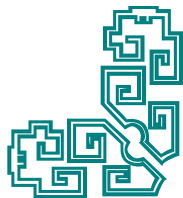
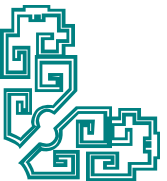


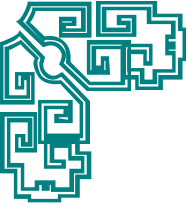
2.2.2 JSP程序块

来看下面这段JSP代码，命名为circle.jsp:

```
<%@ page language="java" pageEncoding="ISO-8859-1"%>
<html>
<body>
    <%      double r=10.0, s;
           s=3.14 * r * r;
           out.print(s);
    %> //局部代码块
</body>
</html>
```

将上面的circle.jsp文件存放到webapps\ROOT文件夹下。启动Tomcat服务器，在浏览器中输入地址“<http://localhost:8080/circle.jsp>”，将在窗口中显示圆面积的值“314.0”。





2.2.3 JSP表达式

2.2.2节的例子中可以发现，要输出面积 s 的值，先计算 s 的值，然后输出结果。JSP中提供了一种表达式，可以很方便地输出运算结果，其格式如下：

`<%=Java表达式 %>`

于是，2.2.2节circle.jsp文件的代码可以修改如下：

```
<%@ page language="java" pageEncoding="ISO-8859-1"%>
```

```
<html>
```

```
<body>
```

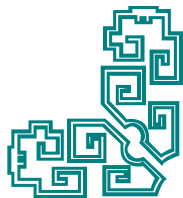
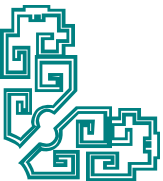
```
    <%double r=10.0,s;
```

```
    %>
```

```
    <%=3.14*r*r %>脚本段语句<%变量名或方法%>
```

```
</body>
```

```
</html>
```



2.2.4 JSP指令

常用的有3条指令：page、include和taglib。

◆ 1. page指令

page指令主要用来设定整个JSP文件的属性和相关功能，如前面写的JSP文件的头：

```
<%@ page contentType="text/html, charset=gb2312"%>
```

一般用到的page指令还有导入需要的包，用法如下：

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

◆ 2. include指令

重复性功能代码每个文件都写一遍很繁琐。include指令用来解决这个问题，其用来**导入包含静态**的文件，如JSP网页文件、HTML网页文件，但不能包含用“<%=”和“%>”表示的代表表达式的文件。其语法格式如下：

```
<%@ include file="被包含文件url" %>
```

如有head.jsp文件，其内容如下：

```
<%@page language="java" contentType="text/html; charset=gb2312"%>
```

```
<%@page import="java.sql.ResultSet"%>
```

现在在另一个文件中调用它：

```
%@include file="head.jsp"%//不能使用Java代码块声明同名变量
```

```
<html>
```

```
<head><title>输出页面</title></head>
```

```
<body>这句话是我想输出的</body>
```

```
</html>
```



2.2.4 JSP指令

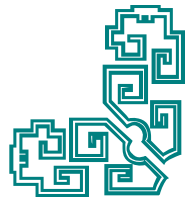
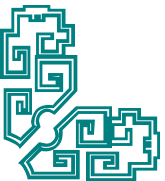
◇ 3. taglib指令

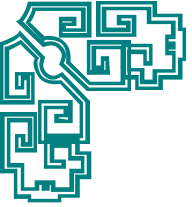
taglib指令语法格式如下：

```
<%@ taglib uri="tagLibraryURI" prefix="tagPrefix" %>
```

其中uri= " tagLibraryURI " 指明标签库文件的存放位置。而prefix= " tagPrefix " 则表示该标签使用时的前缀。例如，在Struts 2中用到标签：

```
<%@ taglib uri="/struts-tags" prefix="s"%>
```





2.2.5 JSP动作

◇ 1. <jsp:param>

<jsp:param>的语法规则如下：

```
<jsp:param name="paramName" value="paramValue"/>
```

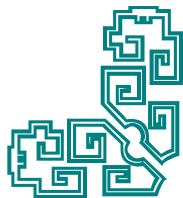
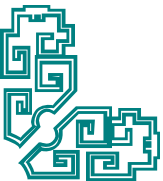
例如：

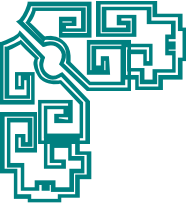
```
<jsp:param name="username" value="liu"/>
```

<jsp:param>通常与<jsp:include>、<jsp:forward>或<jsp:plugin>等一起使用。

在独立于其他操作使用时，<jsp:param>动作没有作用。

会将数据以？的形式拼接在转发路径的后面





2.2.5 JSP动作

◇ 2. **<jsp:include>**可以使用Java代码块声明同名变量（动态引入）

<jsp:include>的语法规则如下：

```
<jsp:include page=" { relativeurl | <%= expression %> } " flush="true" />
```

或者为：

```
<jsp:include page=" { relativeurl | <%= expression %> } " flush="true" >
```

```
    <jsp:param name="paramName" value="{ paramValue | <%= expression %>}" />
```

```
</jsp:include>
```

<jsp:include>可以向一个对象提出请求，并可以将结果包含在一个JSP文件中。

<jsp:include>可以将静态的HTML、服务器程序的输出结果及来自其他JSP的输出结果包括到当前页面中。使用的是相对的URL来调用资源。

例如，包含普通的HTML文件：

```
<jsp:include page=" hello.html " />
```

使用相对路径：

```
<jsp:include page=" /index.html " />
```

包含动态JSP文件：

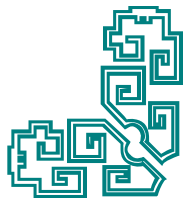
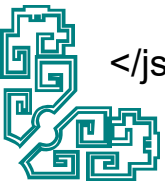
```
<jsp:include page=" scripts/login.jsp " />
```

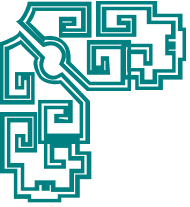
向被包含的程序传递参数：

```
<jsp:include page=" scripts/login.jsp " >
```

```
    <jsp:param name="username" value="zheng" />
```

```
</jsp:include>
```





2.2.5 JSP动作

◇ 3. <jsp:useBean>

<jsp:useBean>的语法规则如下：

```
<jsp:useBean id="name" class="classname" scope="page | request |  
session | application" typeSpec />
```

语法参数说明如下：

id： 设置JavaBean的名称，利用此id，可以识别在同一个JSP程序中使用不同的JavaBean组件实例。

class： 指定JavaBean对应的Java类名查找该JavaBean的路径。

scope： 指定JavaBean对象的作用域。scope的值可能是page、request、session和application。

typeSpec： 可能是如下的4种形式之一：

class="className"

//仅指明应用的类名

class="className" type="typeName"

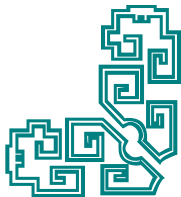
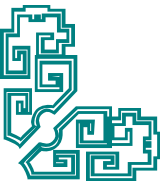
//指明应用的类名及类型

beanName="beanName" type="typeName"

//指明应用的其他Bean的名称及类型

type="typeName"

//仅指明类型



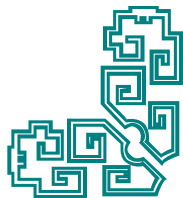
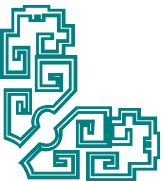


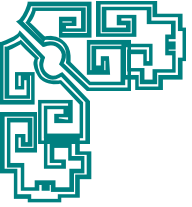
2.2.5 JSP动作

【例2.2】 useBean动作元素的应用。

创建Web项目，命名为“JSP”，在WebRoot文件夹下创建JSP文件，命名为bean.jsp，其代码如下：

```
<%@ page contentType="text/html;charset=GB2312" %>
<html>
<head>
<title>useBean动作元素的应用</title>
</head>
<body>
<jsp:useBean id="test" scope="page" class="test.TestBean" />
<%
    test.setString("南京师范大学");
    String str=test.getStringValue();
    out.print(str);
%>
</body>
</html>
```





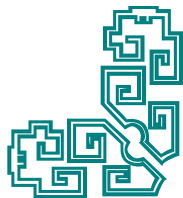
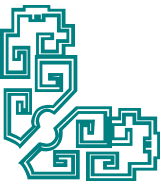
2.2.5 JSP动作

在src文件夹下创建包test，在包test下创建TestBean.java，其代码如下：

```
package test;

public class TestBean{
    private String str=null;
    public TestBean(){ }
    public void setString(String value){
        str=value;
    }
    public String getStringValue(){
        return str;
    }
}
```

部署运行项目，在浏览器中输入“<http://localhost:8080/JSP/bean.jsp>”，页面就会输出“南京师范大学”。

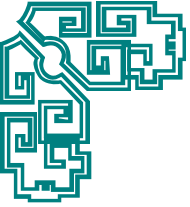


2.2.5 JSP动作

◇ 4. <jsp:setProperty>

<jsp:setProperty>的语法规则如下:

```
<jsp:setProperty
    name= "BeanName " //某个Bean的名称
    {   property= " * " | //应用的Bean对应类中的属性名
        property= "propertyName " [ param= "parameterName " ] |
        property= "propertyName " value= "propertyValue "
    }
/>
```



2.2.5 JSP动作

语法参数说明如下：

- **name**：指定目标Bean对象。
- **property**：指定要设置Bean的属性名。
- **value**：指定Bean属性的值。

`<jsp:setProperty>`将字符串类型转换为其他类型的方法如下：

boolean（或**Boolean**）：`java.lang.Boolean.valueOf(String)`;

byte（或**Byte**）：`java.lang.Byte.valueOf(String)`;

char（或**Character**）：`java.lang.Character.valueOf(String)`;

double（或**Double**）：`java.lang.Double.valueOf(String)`;

float（或**Float**）：`java.lang.Float.valueOf(String)`;

int（或**Integer**）：`java.lang.Integer.valueOf(String)`;

long（或**Long**）：`java.lang.Long.valueOf(String)`;

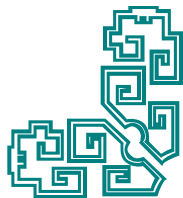
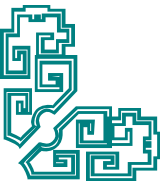
- **param**：指定从request对象的某一参数取值以设置Bean的同名属性，即要将其值赋给一个Bean属性的HTTP请求的参数名称。

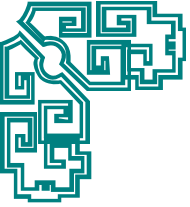
根据JSP规范，如下代码都是合法的。

```
<jsp:setProperty name= "TestBean" property= " * " />
```

```
<jsp:setProperty name= "TestBean" property= "username" />
```

```
<jsp:setProperty name= "TestBean" property= "username" value= "jack" />
```





2.2.5 JSP动作

◇ 5. <jsp:getProperty>

<jsp:getProperty>的语法规则如下：

```
<jsp:getProperty name= "BeanName" property= "PropertyName" />
```

其中属性name是JavaBean实例的名称，property是要显示的属性的名称。

根据语法规则，如下代码是合法的。

```
<jsp:useBean id="test" scope="page" class="test.TestBean" />
```

```
<h1>Get of string :<jsp:getProperty name= "test" property= "StringValue" /></h1>
```

<jsp:getProperty>可以获取Bean的属性值。

◇ 6. <jsp:forward>一次请求，地址栏信息不变，相当于请求转发

<jsp:forward>的语法规则如下：

```
<jsp:forward page=" { relativeurl | <%= expression %> } " />
```

或者为：

```
<jsp:forward page=" { relativeurl | <%= expression %> } " >
```

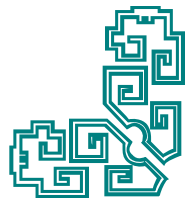
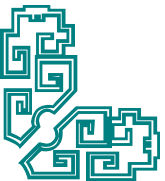
```
    <jsp:param name="paramName" value="{ paramValue | <%= expression %>}" />
```

```
</jsp:forward>
```

<jsp:forward>标记只有一个属性page。page属性指定要转发资源的相对URL。page的值既可以给出，也可以在请求时动态计算。例如：

```
<jsp:forward page="/utils/errorReporter.jsp" />
```

```
<jsp:forward page="<%=someJavaExpression %>" />
```



2.2.5 JSP动作

◇ 7. <jsp:plugin>

<jsp:plugin>的语法规则如下:

<jsp:plugin

type="bean | applet"

code="className"

codebase="classFileDirectoryName"

[name="instanceName"]

[archive="URIToArchive ,..."]

[align="bottom | top | middle | left | right"]

[height="displayPixels"]

[width="displayPixels"]

[hspace="leftRightPixels"]

[vspace="topBottomPixels"]

[jreversion="JREVersionNumber | 1.2 "]

[nspluginurl ="url ToPlugin"]

[iepluginurl ="url ToPlugin"]>

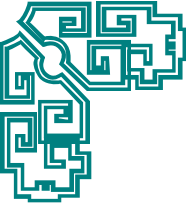
[<jsp:params>

[<jsp:params name="paramName" value="{ parameterValue | <%= expression %>}" />]+

</jsp:params>]

[<jsp:fallback> text message for user </jsp:fallback>]

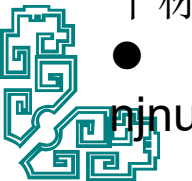
</jsp:plugin>

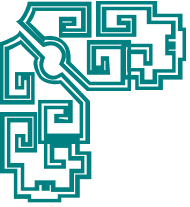


2.2.5 JSP动作

语法参数说明如下：

- **type**: 指定被执行的Java程序的类型是JavaBean还是Java Applet。
- **code**: 指定会被JVM执行的Java Class的名字，必须以.class结尾命名。
- **codebase**: 指定会被执行的Java Class文件所在的目录或路径，默认值为调用</jsp:plugin>指令的JSP文件的目录。
- **name**: 确定这个JavaBean或者Java Applet程序的名字，它可以在JSP程序的其他地方被调用。
- **archive**: 表示包含对象Java类的.jar文件。
- **align**: 对图形、对象、Applet等进行定位，可以选择的值为bottom、top、middle、left和right五种。
- **height**: JavaBean或者Java Applet将要显示出来的高度、宽度的值，此值为数字，单位为像素。
- **hspace**和**vspace**: JavaBean或者Java Applet显示时在浏览器显示区左、右、上、下所需留下的空间，单位为像素。
- **jreversion**: JavaBean或者Java Applet被正确运行所需要的Java运行时环境的版本，默认值是1.2。
- **nspluginurl**: 可以为Netscape Navigator用户下载JRE插件的地址。此值为一个标准的URL，如<http://www.njnu.edu.cn>。
- **iepluginurl**: IE用户下载JRE的地址。此值为一个标准的URL，如<http://www.njnu.edu.cn>。

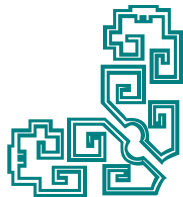
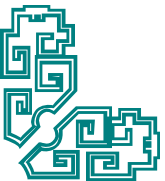


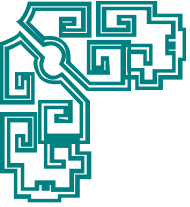


2.2.5 JSP动作

- `<jsp:params>`和`</jsp:params>`：使用`<jsp:params>`操作指令，可以向JavaBean或者Java Applet传送参数和参数值。
- `<jsp:fallback>`和`</jsp:fallback>`：该指令中间的一段文字用于Java插件不能启动时显示给用户；如果插件能够正确启动，而JavaBean或者Java Applet的程序代码不能找到并被执行，那么浏览器将会显示这个出错信息。例如：

```
<jsp:plugin
  type="applet"
  code="Test.class"
  codebase="/example/jsp/applet "
  height="180"
  width="160"
  jreversion="1.2">
  <jsp:params>
  <jsp:params name="test" value="TsetPlugin" />
  </jsp:params>
  <jsp:fallback>
  <p> To load apple is unsuccessful </p>
  </jsp:fallback>
</jsp:plugin>
```





2.2.6 JSP注释

◇ 1. 输出注释

输出注释的语法规则如下：

`<!-- 注释内容[<%=表达式%>]-->`

这种注释和HTML文件中的注释很相似，唯一不同的是，前者可以在这个注释中用表达式，以便动态生成不同内容的注释。这些注释的内容在客户端是可见的，也就是可以在HTML文件的源代码中看到。如下面一段注释：

`<!-- 现在是时间是：<%=(new java.util.Date()).toLocaleString() %> -->`

把上面代码放在一个JSP文件的body体中运行后，可以在其源代码中看到：

`<!-- 现在是时间是：2009-2-19 14:08:46 > -->`

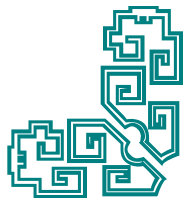
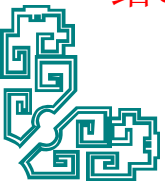
◇ 2. 隐藏注释

隐藏注释的语法规则如下：

`<%-- 注释内容--%>`

隐藏注释与输出注释不同的是，这个注释虽然写在JSP程序中，但是不会发送给用户。

网页发起请求，Tomcat服务器接收到请求后，先把请求数据解析并封装到请求对象request中，然后调用对应的servlet处理请求，把request对象作为实参传递给servlet



2.3 JSP内置对象（**jspService**方法中声明）

✧ 2.3.1 page对象

page对象代表JSP页面本身，是this引用的一个代名词。对JSP页面创建者通常不可访问，所以一般很少用到该对象。

✧ 2.3.2 config对象

config对象是ServletConfig类的一个对象，存放着一些Servlet初始化信息，且只有在JSP页面范围内才有效。其常用方法如下：

- getInitParameter(name)：取得指定名字的Servlet初始化参数值。
- getInitParameterNames()：取得Servlet初始化参数列表，返回一个枚举实例。
- getServletContext()：取得Servlet上下文（ServletContext）。
- getServletName()：取得生成的Servlet的名字。

✧ 2.3.3 out对象

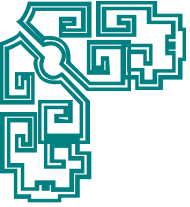
JSP页面的主要目的是动态产生客户端需要的HTML结果，前面已经用过out.print()和out.println()来输出结果。此外out对象还提供了一些其他方法来控制管理输出缓冲区和输出流。例如，要获得当前缓存区大小可以用下面的语句：

```
out.getBufferSize();
```

要获得剩余缓存区大小应为：

```
out.getRemaining();
```

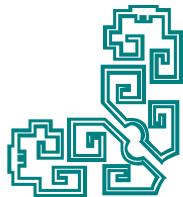
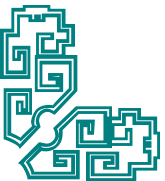
内置对象在jsp页面中使用，（局部代码块或者脚本语句），不能够在全局代码块中使用

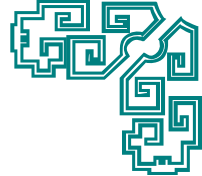
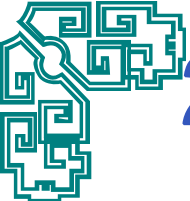


2.3.4 response对象

response对象实现**HttpServletResponse**接口，可以对客户的请求做出动态的响应，向客户端发送数据，如**Cookies**、**HTTP**文件的头信息等，一般是**HttpServletResponse**类或其子类的一个对象。以下是**response**对象的主要方法：

- **addHeader(String name,String value)**: 添加**HTTP**头文件，该头文件将会传到客户端去，如果有同名的头文件存在，那么原来的头文件会被覆盖。
- **setHeader(String name,String value)**: 设定指定名字的**HTTP**文件头的值，如果该值存在，那么它将会被新的值覆盖。
- **containsHeader(String name)**: 判断指定名字的**HTTP**文件头是否存在，并返回布尔值。
- **flushBuffer()**: 强制将当前缓冲区的内容发送到客户端。
- **addCookie(Cookie cookie)**: 添加一个**Cookie**对象，用来保存客户端的用户信息，可以用**request**对象的**getCookies()**方法获得这个**Cookie**。
- **sendError(int sc)**: 向客户端发送错误信息。例如，“505”指示服务器内部错误，“404”指示网页找不到的错误。
- **setRedirect(url)**: 把响应发送到另一个指定的页面（**URL**）进行处理。

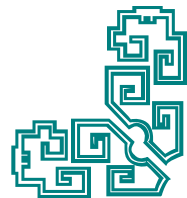
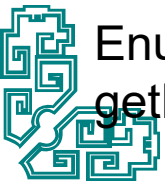




2.3.5 request对象 (封存要处理的数据, tomcat服务器创建)

下面介绍其主要的方法:

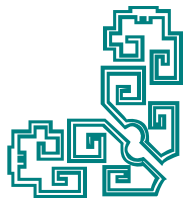
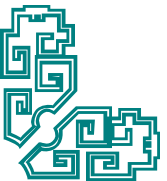
- **getParameter(String name):** 以字符串的形式返回客户端传来的某一个请求参数的值, 该参数由name指定。
 - **getParameterValue(String name):** 以字符串数组的形式返回指定参数所有值。
 - **getParameterNames():** 返回客户端传送给服务器端的所有的参数名, 结果集是一个Enumeration (枚举) 类的实例。
 - **getAttribute(String name):** 返回name指定的属性值, 若不存在指定的属性, 则返回null。
 - **setAttribute(String name, java.lang.Object obj):** 设置名字为name的request参数的值为obj。
 - **getCookies():** 返回客户端的Cookie对象, 结果是一个Cookie数组。
 - **getHeader(String name):** 获得HTTP协议定义的传送文件头信息, 例如, `request.getHeader("User-Agent")` 含义为: 返回客户端浏览器的版本号、类型。
 - **getDateHeader():** 返回一个Long类型的数据, 表示客户端发送到服务器的头信息中的时间信息。
 - **getHeaderName():** 返回所有request Header的名字, 结果集是一个Enumeration (枚举) 类的实例。得到名称后就可以使用**getHeader**、**getDateHeader**等得到具体的头信息。
-

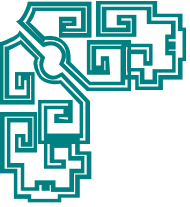




2.3.5 request对象

- `getServerPort()`: 获得服务器的端口号。
- `getServerName()`: 获得服务器的名称。
- `getRemoteAddr()`: 获得客户端的IP地址。
- `getRemoteHost()`: 获得客户端的主机名, 如果该方法失败, 则返回客户端的IP地址。
- `getProtocol()`: 获得客户端向服务器端传送数据所依据的协议名称。
- `getMethod()`: 获得客户端向服务器端传送数据的方法。
- `getServletPath()`: 获得客户端所请求的脚本文件的文件路径。
- `getCharacterEncoding ()`: 获得请求中的字符编码方式。
- `getSession(Boolean create)`: 返回和当前客户端请求相关联的`HttpSession`对象。
- `getQueryString()`: 返回查询字符串, 该字符串由客户端以`get`方法向服务器端传送。
- `getRequestURI()`: 获得发出请求字符串的客户端地址。
- `getContentType()`: 获取客户端请求的MIME类型。如果无法得到该请求的MIME类型, 则返回-1。





2.3.6 session对象

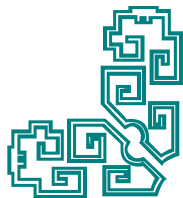
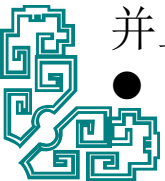
session对象的主要方法如下：

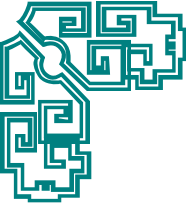
- **getAttribute(String name)**: 获得指定名字的属性，如果该属性不存在，将会返回null。
- **getAttributeNames()**: 返回session对象存储的每一个属性对象，结果集是一个Enumeration类的实例。
- **getCreationTime()**: 返回session对象被创建的时间，单位为毫秒。
- **getId()**: 返回session对象在服务器端的编号。
- **getLastAccessedTime()**: 返回当前session对象最后一次被操作的时间，单位为毫秒。
- **getMaxInactiveInterval ()**: 获取session对象的生存时间，单位为秒。
- **setMaxInactiveInterval (int interval)**: 设置session对象的有效时间（超时时间），单位为秒。在网站的实际应用中，30分钟的有效时间对某些网站来说有些太短，但对有些网站来说又有些太长。

例如：设置有效时间为200s。

```
<%session.setMaxInactiveInterval (200);%>
```

- **removeAttribute(String name)**: 删除指定属性的属性名和属性值。
- **setAttribute(String name,Java.lang.Object value)**: 设定指定名字的属性，并且把它存储在session对象中。
- **invalidate()**: 注销当前的session对象。

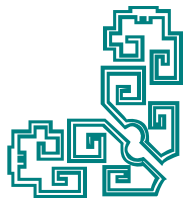
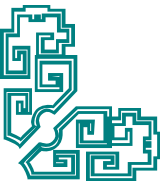




2.3.7 application对象

application对象的主要方法如下：

- `getAttribute(String name)`: 返回由name指定名字的application对象的属性值。
- `getAttributeNames()`: 返回所有application对象属性的名字，结果集是一个Enumeration类型的实例。
- `getInitParameter(String name)`: 返回由name指定名字的application对象的某个属性的初始值，如果没有参数，就返回null。
- `getServerInfo()`: 返回Servlet编译器当前版本信息。
- `setAttribute(String name, Object obj)`: 将参数Object指定的对象obj添加到application对象中，并为添加的对象指定一个属性。
- `removeAttribute(String name)`: 删除一个指定的属性。



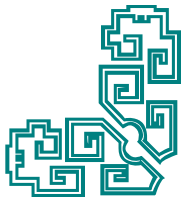
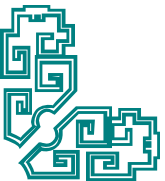


2.3.7 application对象

下面用实例说明它们三者之间的区别。

首先，建立项目Application_Session_Request。在项目中建立一个JSP页面first.jsp，用于这三个对象保存数据。

```
<%@ page language="java" pageEncoding="gb2312"%>
<html>
<body>
    <%
        request.setAttribute("request","保存在Request中的内容");
        session.setAttribute("session","保存在Session中的内容");
        application.setAttribute("application","保存在Application中的内容");
    %>
    <jsp:forward page="second.jsp"></jsp:forward>
</body>
</html>
```

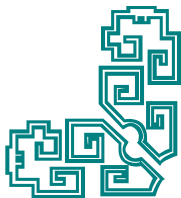
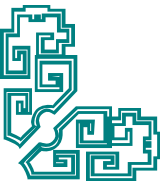




2.3.7 application对象

然后，再建立另一个JSP页面second.jsp，用于获取这三个对象保存的值。

```
<%@ page language="java" pageEncoding="gb2312"%>
<html>
<head>
</head>
<body>
  <%
    out.println("request:"+(String)request.getAttribute("request")+"<br>");
    out.println("session:"+(String)session.getAttribute("session")+"<br>");
    out.print("application:"+(String)application.getAttribute("application")+"<br>");
  %>
</body>
</html>
```



2.3.7 application对象

部署运行，打开IE，输入
“http://localhost:8080/Application_Session_Request/first.jsp”，会发现这三个对象保存的内容都能取出，如图2.16所示。

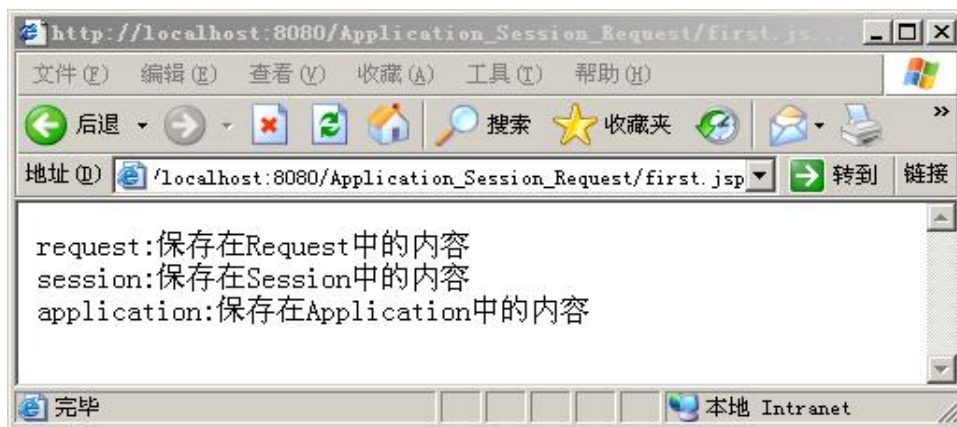


图2.16 运行界面1

2.3.7 application对象

由于在first.jsp中运用了<jsp:forward page="second.jsp"></jsp:forward>，页面跳转到second.jsp，但是在浏览器中的地址也就是请求并没有改变，属于同一请求。这时这三个对象保存的内容都可以取到，也就是说在同一请求范围内，这三个对象都有效，在该IE浏览器中输入

“http://localhost:8080/Application_Session_Request/second.jsp”，结果如图2.17所示。



图2.17 运行界面2

2.3.7 application对象

在不同请求中，request对象就失效了，但是由于用的是同一个IE，也就是同一会话，session和application仍然有效。如果再重新打开一个IE，然后直接输入“http://localhost:8080/Application_Session_Request/second.jsp”，结果如图2.18所示。

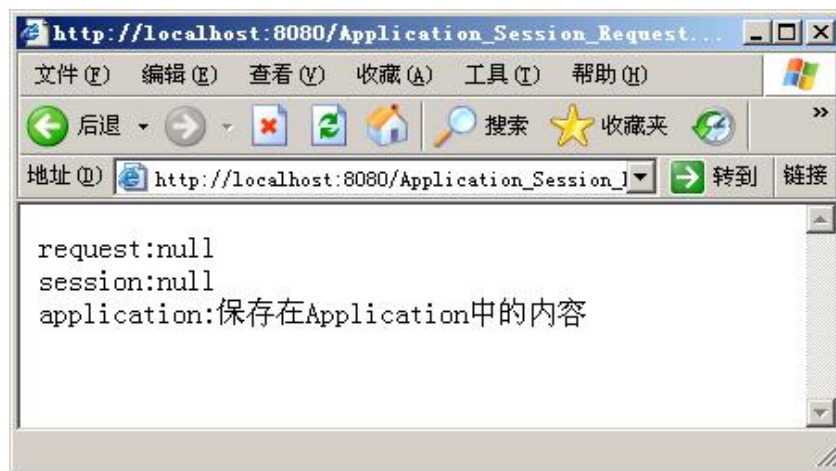


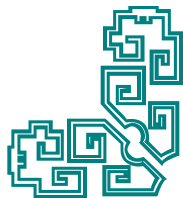
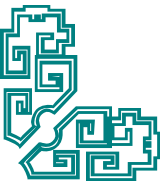
图2.18 运行界面3

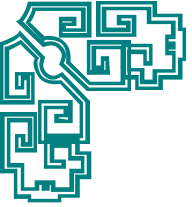


2.3.8 pageContext对象 (封存了当前jsp的运行信息)

pageContext对象的主要方法如下（页面上下文对象，封存了其他内置对象）

- **getAttribute():** 返回与指定范围内名称有关的变量或null，例如：
`CustomContext MyContext=(CustomContext)`
`pageContext.getAttribute("Large Bird", PageContext.SESSION_SCOPE);`
这段代码在作用域中获得一个对象。
- **forward(String relativeurl Path):** 把页面重定向到另一个页面或Servlet组件上。
- **findAttribute():** 按照页面请求、会话及应用程序范围的顺序实现对某个已经命名属性的搜索。
- **getException():** 返回当前的exception对象。
- **setAttribute():** 设置默认页面的范围或指定范围中的已命名对象。例如：
`CustomContext MyContext=new CustomContext("Penguin");`
`pageContext.setAttribute("Large Bird", MyContext.PageContext.SESSION_SCOPE);`
这段代码在作用域中设置一个对象。
- **removeAttribute():** 删除默认页面范围或指定范围中已命名的对象。



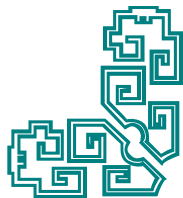
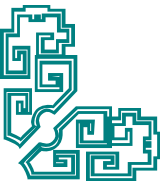


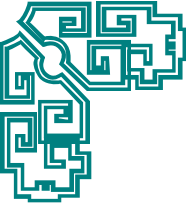
2.3.9 exception对象

exception对象用来处理JSP文件在执行时所发生的错误和异常。它可以配合**page**指令一起使用，通过指定某一页面为错误处理页面，把所有的错误都集中到那个页面去处理。这样使得整个系统更加健壮，程序的流程更加清晰，这也是JSP比ASP、PHP先进的地方。

exception对象的主要方法如下：

- **getMessage()**：返回错误信息。
- **printStackTrace()**：以标准错误的形式输出一个错误和错误堆栈。
- **toString()**：以字符串的形式返回一个对异常的描述。





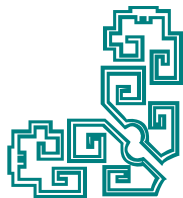
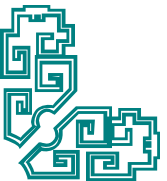
2.4 JavaBean及其应用

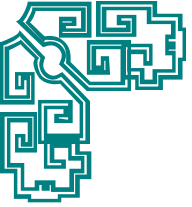
JavaBean是使用Java语言描述的软件组件模型，是一个可以重复使用的Java类。

JavaBean形式和要素

只需要在类的方法命名上遵守以下规则：

- 如果类的成员变量名字是XXX，那么为了更改或获取成员变量的值，在类中可以使用两种方法，setXXX()和getXXX()。
- 对于boolean类型的成员变量，允许使用is代替上面的get和set。
- 类中方法的访问属性都必须是public的。
- 类中如果有构造方法，那么这个构造方法也是public的，且无参数。





2.5 Servlet应用

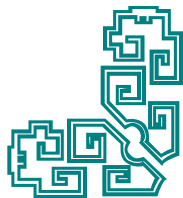
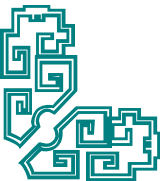
Servlet是Java语言处理Web请求的一种机制，前面所讲的JSP其实就是建立在Servlet之上的。Servlet是Java Web技术的核心基础，是进一步学习Java EE 系列架构的前提。

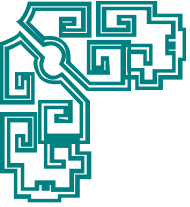
Servlet是一种服务器端的Java程序，由服务器进行加载，具有独立于平台和协议的特性，可以生产动态的Web页面。

Servlet是位于Web服务器内部的服务器端的Java应用程序，它担任客户（Web浏览器）请求与服务器（Web服务器上的应用程序）响应的中间层。

课后作业：

查找Servlet API，画出Servlet API中主要类和接口之间的关系。





2.5 Servlet应用

Servlet的变成方式:

1.实现Servlet接口

这种情况Servlet不是独立的应用程序，没有main()方法，而是生存在容器中，由容器来管理。

2.继承GenericServlet类

由java.servlet包提供的一个抽象类，它给出了Servlet接口中除了service()方法外的4个方法的实现，并且还是实现了ServletConfig接口，可以简化代码。

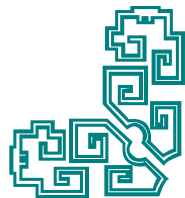
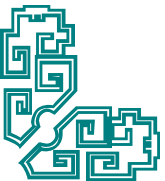
3.继承HttpServlet类、覆盖doXXX()方法

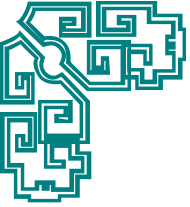
由java.servlet.http包提供的一个抽象类，它继承了GenericServlet类。通常情况下都是覆盖其doGet()和doPost()方法，然后在其中一个方法里调用另一个，做到合二为一。

4.继承HttpServlet类、重写service()方法

本质就是扩展HttpServlet 类，用户只需要重写service()方法，Servlet模块执行service()方法时，会自动调用doGet()和doPost()方法，实现Servlet的逻辑处理功能。

其中，最常用的就是方式3。





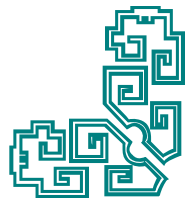
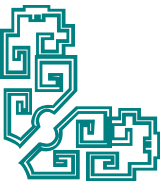
2.5 Servlet应用

✧ 2.5.1 Servlet的主要接口和类

◆ 1. Servlet接口

下面就来开发一个Servlet项目。

首先打开安装了MyEclipse插件的Eclipse，然后建一个Web项目。选择【File】→【New】→【Project...】菜单项，弹出新建命令对话框，选择【MyEclipse】→【Java Enterprise Projects】→【Web Project】菜单项，单击【Next】按钮进入Web应用详细信息设置，在【Project Name】文本框中输入Web应用名称，命名为“Servlet”，在【J2EE Specification Level】一栏中选择【Java EE 5.0】菜单项，其他为默认值，如图2.10所示，单击【Finish】按钮完成。



2.5.1 Servlet的主要接口和类

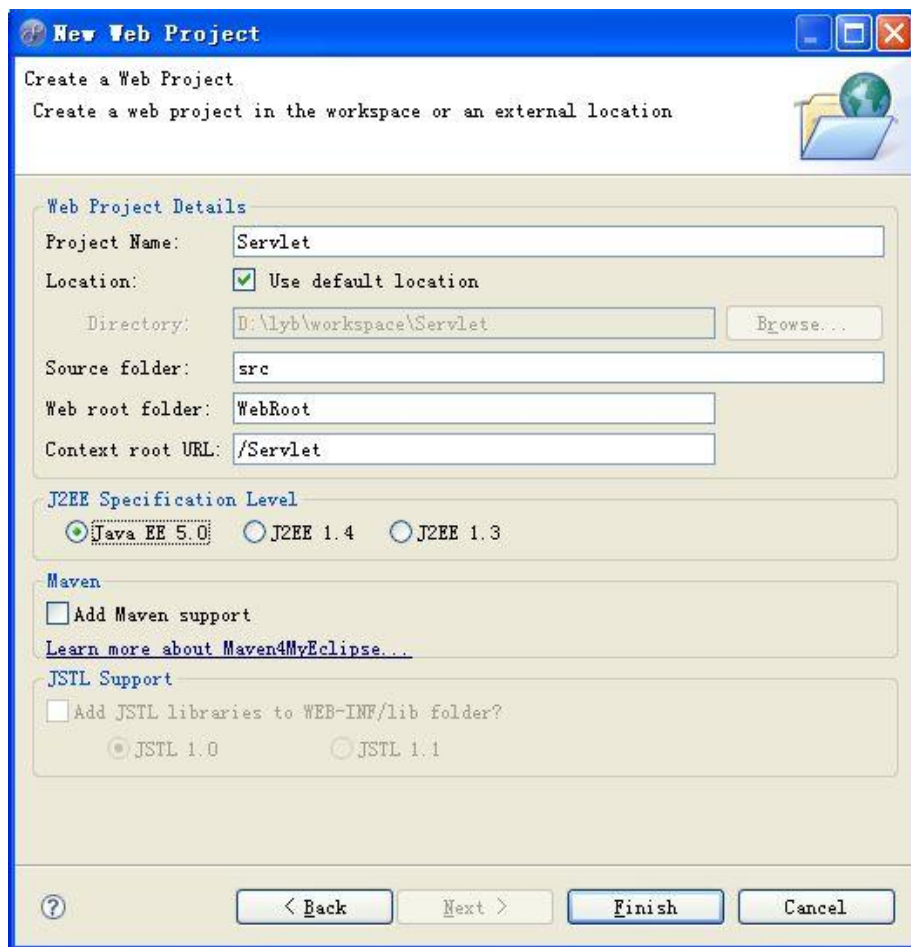


图2.10 建立Web项目

2.5.1 Servlet的主要接口和类

项目建立完成后，在左边的视图中可以看到刚才新建项目的内容，如图2.11所示。右击src文件夹，选择【new】→【class】菜单项，弹出新建类对话框，在【name】一栏中输入类名，命名为“HelloWorld”，其他为默认值，单击【Finish】按钮完成。

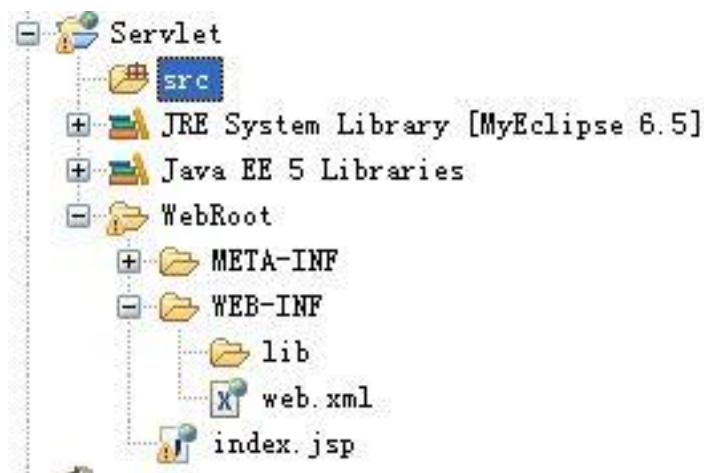


图2.11 项目视图

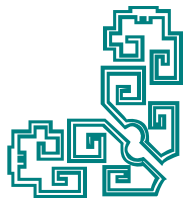
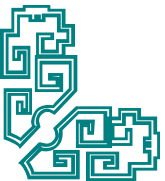


2.5.1 Servlet的主要接口和类


下面编辑HelloWorld类，让它实现Servlet接口。

选择【WebRoot】→【WEB-INF】菜单项，双击打开web.xml文件。修改web.xml文件代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/JavaEE/web-app_2_5.xsd">
  <servlet>
    <servlet-name>HelloWorld</servlet-name>
    <servlet-class>HelloWorld</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>HelloWorld</servlet-name>
    <url-pattern>/helloWorld</url-pattern>
  </servlet-mapping>
</web-app>
```



2.5.1 Servlet的主要接口和类

首先单击工具栏中的  按钮，出现Web应用部署的对话框，单击【Add】按钮，出现“New Deployment”对话框。在【Server】一栏中选择“Tomcat 6.x”，【Deploy type】栏中选择“Exploded Archive”（开发模式），单击【Finish】按钮完成。此时在【Deployments】栏中显示Tomcat 6.x服务器信息，如图2.12所示。单击【OK】按钮完成部署。

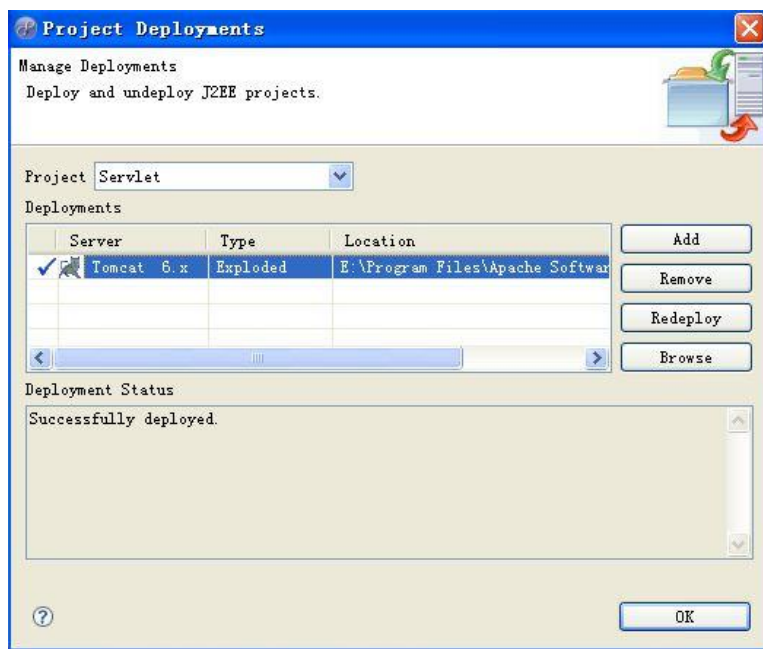
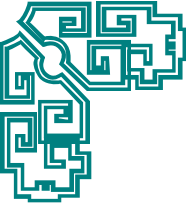


图2.12 部署项目

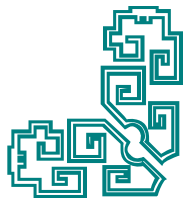
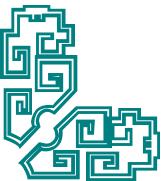


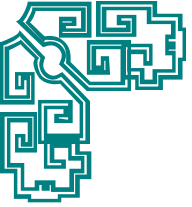
2.5.1 Servlet的主要接口和类

在运行之前，首先要启动Tomcat服务器，单击工具栏中  按钮的下三角，选择【Tomcat 6.x】→【Start】菜单项，启动Tomcat服务器。启动完成后，打开IE浏览器，输入“<http://localhost:8080/Servlet/helloWorld>”，就会在页面中显示HelloWorld，如图2.13所示。



图2.13 运行界面





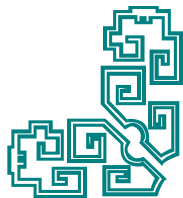
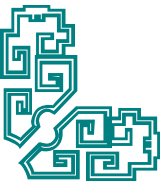
2.5.1 Servlet的主要接口和类

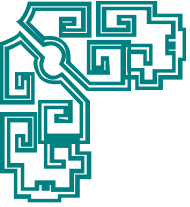
下面介绍这5个方法的作用。

- **init():** 在Servlet实例化之后，Servlet容器会调用init()方法，来初始化该对象。
- **service():** 容器调用service()方法来处理客户端的请求
- **destroy():** 当容器检测到一个Servlet对象应该从服务中被移除时，容器会调用该对象的destroy()方法，来释放Servlet对象所使用的资源，保存数据到持久存储设备中。
- **getServletConfig():** 返回容器调用init()方法时传递给Servlet对象的ServletConfig对象，ServletConfig对象包含了Servlet的初始化参数。
- **getServletInfo():** 返回一个String类型的字符串，其中包括关于Servlet的信息，例如，作者、版本和版权。

下面再来看看访问的地址。以上例为例（见图2.13），

“http://localhost:8080/”是服务器URL，而后面的“Servlet”是项目名，再后面的“helloWorld”是在web.xml文件中配置的<url-pattern>的值。



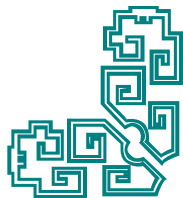
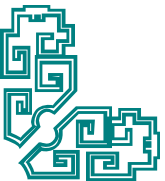


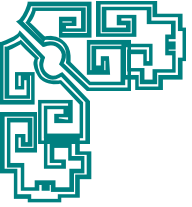
2.5.1 Servlet的主要接口和类

◇ 2. GenericServlet类

为了简化Servlet的编写，在javax.servlet包中提供了一个抽象的类GenericServlet。它给出了除service()方法外的其他4个方法的简单实现。GenericServlet类实现了Servlet接口和ServletConfig接口。所以上例的HelloWorld类如果继承这个类，代码会简化很多。其代码如下：

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.GenericServlet;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
public class HelloWorld extends GenericServlet{
    public void service(ServletRequest arg0, ServletResponse arg1)
        throws ServletException, IOException {
        // TODO Auto-generated method stub
        PrintWriter pw=arg1.getWriter();
        pw.println("HelloWorld");
    }
}
```





2.5.1 Servlet的主要接口和类

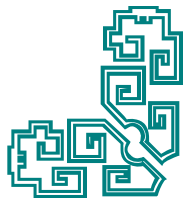
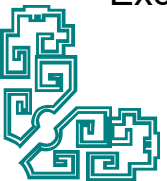
◇ 3. HttpServlet类

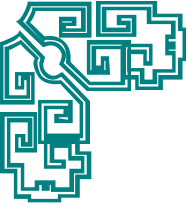
HttpServlet类中重载了GenericServlet的service()方法:

- public void service(ServletRequest req, ServletResponse res) throws ServletException, java.io.IOException
- protected void service(HttpServletRequest req, HttpServletResponse res) throws ServletException, java.io.IOException

根据不同的请求方法, HttpServlet提供了7个处理方法:

- protected void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, java.io.IOException
- protected void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException, java.io.IOException
- protected void doHead(HttpServletRequest req, HttpServletResponse res) throws ServletException, java.io.IOException
- protected void doPut(HttpServletRequest req, HttpServletResponse res) throws ServletException, java.io.IOException
- protected void doDelete(HttpServletRequest req, HttpServletResponse res) throws ServletException, java.io.IOException
- protected void doTrace(HttpServletRequest req, HttpServletResponse res) throws ServletException, java.io.IOException
- protected void doOptions(HttpServletRequest req, HttpServletResponse res) throws ServletException, java.io.IOException

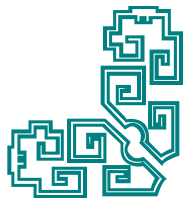
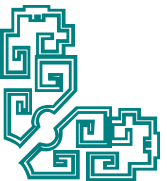


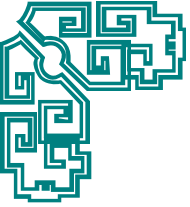


2.5.1 Servlet的主要接口和类

通常情况下，都是覆盖其doGet()和doPost()方法。然后在其中的一个方法中调用另一个方法，这样就可以做到合二为一。如上例可以改成：

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class HelloWorld extends HttpServlet{
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        PrintWriter pw=res.getWriter();
        pw.println("HelloWorld");
    }
    protected void doPost(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        doPost(req, res);
    }
}
```



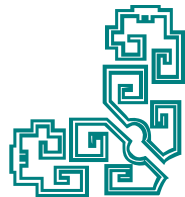
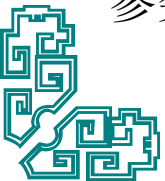


2.5.1 Servlet的主要接口和类

❖ 4. HttpServletRequest和HttpServletResponse接口

下面列几个HttpServletRequest中常用的方法：

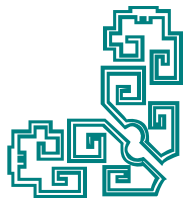
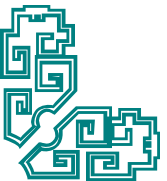
- `setAttribute(String name, Object)`: 设置名字为name的request的参数值；
- `getAttribute(String name)`: 返回由name指定的属性值；
- `getAttributeNames()`: 返回request对象所有属性的名字集合，结果是一个枚举的实例；
- `getCookies()`: 返回客户端的所有Cookie对象，结果是一个Cookie数组；
- `getCharacterEncoding()`: 返回请求中的字符编码方式；
- `getHeader(String name)`: 获得HTTP协议定义的文件头信息；
- `getHeaders(String name)`: 返回指定名字的request Header的所有值，结果是一个枚举的实例；
- `getHeaderNames()`: 返回所有request Header的名字，结果是一个枚举的实例；
- `getInputStream()`: 返回请求的输入流，用于获得请求中的数据；
- `getMethod()`: 获得客户端向服务器端传送数据的方法；
- `getParameter(String name)`: 获得客户端传送给服务器端的由name指定的参数值；





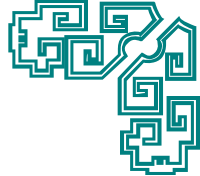
2.5.1 Servlet的主要接口和类

- `getParameterNames()`: 获得客户端传送给服务器端的所有参数名称, 结果是一个枚举的实例;
- `getParameterValues(String name)`: 获得由`name`指定的参数的所有值, 一般用于checkbox;
- `getRequestURI()`: 获取发出请求字符串的客户端地址;
- `getRemoteAddr()`: 获取客户端的IP地址;
- `getRemoteHost()`: 获取客户端的名字;
- `getSession([Boolean create])`: 返回和请求相关session;
- `getServerName()`: 获取服务器的名字;
- `getServletPath()`: 获取客户端所请求的脚本文件的路径;
- `getServerPort()`: 获取服务器的端口号;
- `removeAttribute(String name)`: 删除请求中的一个属性;





2.5.2 Servlet生命周期及实际应用 举例



◆ 1. Servlet生命周期

当Servlet被装载到容器后，生命周期开始。首先调用init()方法进行初始化，初始化后，调用service()方法，根据请求的不同调用不同的doXxx()方法处理客户请求，并将处理结果封装到HttpServletResponse中返回给客户端。当Servlet实例从容器中移除时调用其destroy()方法，这就是Servlet运行的整个过程。

◆ 2. Servlet实际应用举例

【例2.1】 Servlet应用。

这个实例要达到这样的目的，首先在一个HTML文件中建立一个表单，里面有一个输入框，当客户输入内容后，提交到一个Servlet类，而这个Servlet类取出客户输入的信息，并在一个HTML页面上显示该内容。其效果如图2.14、图2.15所示。

http://localhost:8080/ServletExample/input.html

请输入你想显示的内容: 你好!

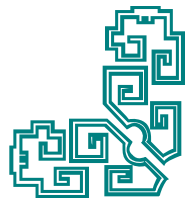
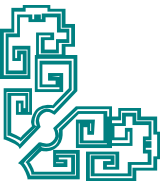
提交 重置

图2.14 输入页面

http://localhost:8080/ServletExample/inputServlet

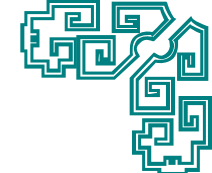
你好!

图2.15 响应页面





2.5.2 Servlet生命周期及实际应用 举例

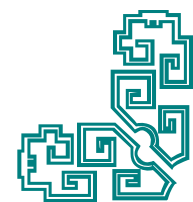
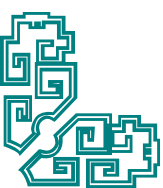


开发这个Servlet应用的步骤如下：

① 建立一个Web项目，命名为“ServletExample”。

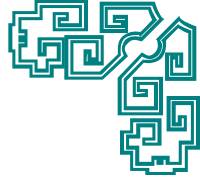
② 在WebRoot文件夹下创建一个HTML文件，操作方法与在src文件下建立一个Class文件差不多。右击WebRoot文件夹，新建一个HTML文件，命名为input.html。其代码如下：

```
<html>
<head>
    <title>Servlet实例</title>
</head>
<body>
    <form action="inputServlet" method="post">
        请输入你想显示的内容: <input type="text" name="input"/><br>
        <input type="submit" value="提交"/>
        <input type="reset" value="重置"/>
    </form>
</body>
</html>
```





2.5.2 Servlet生命周期及实际应用 举例

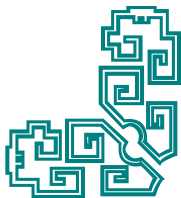
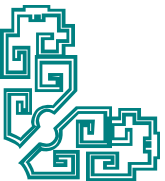


③ 在项目的src文件夹下建立一个包。

编写Servlet类代码。

④ 布局web.xml文件，一定要记住，有一个Servlet文件就要在web.xml中布置一个<servlet>和<servlet-mapping>。这里的布局代码。

⑤ 部署运行，得到以上结果。



2.6 JSP综合应用实例——开发一个简单的留言系统

【例2.3】 用JSP、Servlet和JavaBean开发一个简单的留言系统。这里要达到这样的目的，首先是一个用户登录界面，如图2.19所示。



图2.19 用户登录界面

2.6 JSP综合应用实例——开发一个简单的留言系统

登录成功后会显示所有的留言信息，如图2.20所示。



图2.20 用户登录成功后的主界面

2.6 JSP综合应用实例——开发一个简单的留言系统

单击【留言】按钮，跳转到图2.21所示的界面。



The screenshot shows a Microsoft Internet Explorer window titled "留言板 - Microsoft Internet Explorer". The address bar displays "http://localhost:8080/JSPExample/liuyan.jsp". The main content area is titled "填写留言信息" (Fill in message information). It contains a form with two fields: "留言标题:" (Message Title) with the value "航空" (Aviation) and "留言内容" (Message Content) with the value "中国的航空技术比较先进!" (China's aviation technology is relatively advanced!). At the bottom of the form are two buttons: "提交" (Submit) and "重置" (Reset).

图2.21 留言界面

2.6 JSP综合应用实例——开发一个简单的留言系统

填写好要留言的标题及内容后单击【提交】按钮，跳转到如图2.22所示的留言成功界面。



图2.22 留言成功界面

2.6 JSP综合应用实例——开发一个简单的留言系统

单击该页面的【这里】超链接，回到主界面，可以发现主界面的信息多了刚才添加的留言，如图2.23所示。



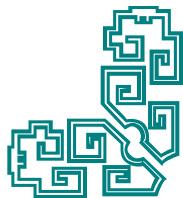
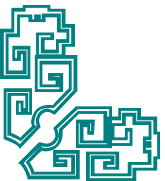
图2.23 留言信息主界面

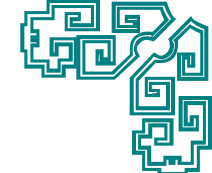


2.6 JSP综合应用实例——开发一个简单的留言系统

◆ 1. 需求分析

既然是留言系统，肯定要有用户登录，所以需要有一个用户表（**userTable**）。字段包括：**id**、**username**和**password**。其中**id**设为自动增长的**int**型，并设为主键。**username**和**password**都设为**varchar**型。登录成功后要有个主界面，显示别人和自己的留言信息，那就应该有个留言表（**lyTable**）。字段包括：**id**、**userId**、**date**、**title**、**content**。其中**id**设为自动增长的**int**型，并设为主键。**userId**是**user**表中的**id**，表明该条留言是该用户留的。





2.6 JSP综合应用实例——开发一个简单的留言系统

2. 建立数据库与表

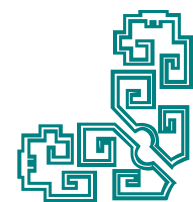
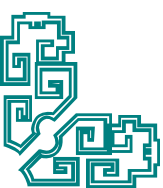
建立数据库，命名为“JSP”，这里我们用的是SQL Server 2005（本书以后的例子中一律采用该数据库系统）。然后在该数据库中建立上面提到的两个表：userTable表、lyTable表，如表2.1、表2.2所示。

表2.1 userTable表

字段名称	数据类型	主 键	自 增	允许为空	描 述
id	int	是	增1		ID号
username	varchar(20)				用户名
password	varchar(20)				密码

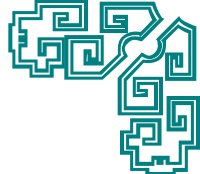
表2.2 lyTable表

字段名称	数据类型	主 键	自 增	允许为空	描 述
id	int	是	增1		ID号
userId	int				用户ID号
date	datetime				发布时间
title	varchar(20)				标题
content	varchar(500)				留言内容





2.6 JSP综合应用实例——开发一个简单的留言系统



◆ 3. 新建项目

打开MyEclipse，新建项目，命名为“JSPEXample”。

◆ 4. 建立表对应的标准JavaBean

userTable表对应的JavaBean[如下](#)。

lyTable表对应的标准JavaBean[如下](#)。

◆ 5. 创建登录页面

首先还是先创建登录界面login.jsp，其[代码](#)。

◆ 6. 建立MainServlet类

当输入登录名和密码后，单击【登录】按钮，提交给了一个Servlet页面，且其URL为“mainServlet”。在src文件夹下建立包，命名为“servlet”，表示该包下存放的都是Servlet类，如果文件多，可方便我们查询。在servlet包下建立一个Servlet类，命名为“MainServlet”，其[代码](#)。

◆ 7. 建立DB类

在src文件夹下建立包，命名为“db”，在db包中建立DB类。如下是DB类的[代码](#)。

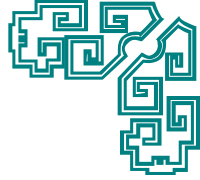
◆ 8. 建立main.jsp

MainServlet中验证成功后会去main.jsp页面，而且在main.jsp页面中会显示所有留言信息，下面是main.jsp文件的[内容](#)。





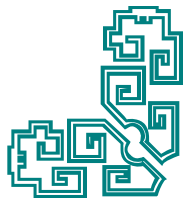
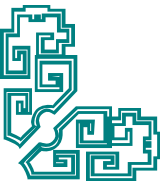
2.6 JSP综合应用实例——开发一个简单的留言系统

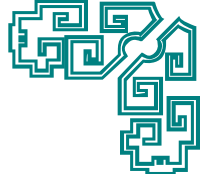


所以在DB类中就要添加这样一个方法，其代码为：

```
public String getUserName(int id){
    String username=null;
    try{
        pstmt=ct.prepareStatement("select username from [userTable] where
id=?");

        pstmt.setInt(1, id);
        ResultSet rs=pstmt.executeQuery();
        while(rs.next()){
            username=rs.getString(1);
        }
        return username;
    }catch(Exception e){
        e.printStackTrace();
        return null;
    }
}
```



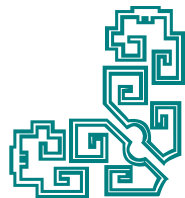
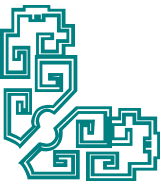


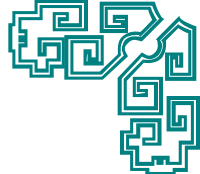
2.6 JSP综合应用实例——开发一个简单的留言系统

◇ 9. 建立liuyan.jsp

当前用户对象已经保存在session中，所以留言时，不用填写；而留言时间就应该是当时的时间，也可以直接获得，所以用户需要填写的就是留言的标题及内容。其代码为：

```
<%@ page language="java" pageEncoding="gb2312"%>
<html>
<head>
    <title>留言板</title>
</head>
<body bgcolor="#E3E3E3">
    <center>
        <form action="addServlet" method="post">
            <table border="1">
                <caption>填写留言信息</caption>
                <tr><td>留言标题</td>
                    <td><input type="text" name="title"/></td></tr>
                <tr><td>留言内容</td>
                    <td><textarea name="content" rows="5"
cols="35"></textarea></td>
                </tr>
            </table>
            <input type="submit" value="提交"/>
            <input type="reset" value="重置"/>
        </form>
    </center>
</body>
```





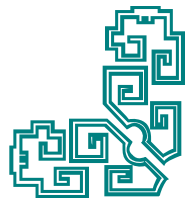
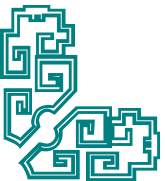
2.6 JSP综合应用实例——开发一个简单的留言系统

◆ 10. 建立AddServlet类

在servlet包下建立AddServlet来操作这些内容，其代码。

在这个Servlet类中，调用了DB类的插入留言信息方法，所以要在DB类中再添加上这个方法。其实如果大家已经很清楚整个过程，就可以一次性在DB类中把用到的方法全部写完，以便以后调用，这样会方便很多，而不是在用到时才去写。该方法代码如下：

```
public boolean addInfo(LyTable ly){
    try{
        pstmt=ct.prepareStatement("insert into [lyTable] values(?,?,?,?)");
        pstmt.setInt(1, ly.getUserId());
        pstmt.setDate(2, ly.getDate());
        pstmt.setString(3, ly.getTitle());
        pstmt.setString(4, ly.getContent());
        pstmt.executeUpdate();
        return true;
    }catch(Exception e){
        e.printStackTrace();
        return false;
    }
}
```





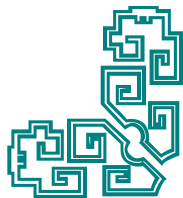
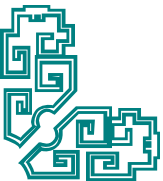
2.6 JSP综合应用实例——开发一个简单的留言系统

◆ 11. 创建成功页面

在AddServlet中我们发现，成功后去了一个告诉用户成功的success.jsp页面，该页面代码为：

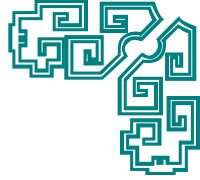
```
<%@ page language="java" pageEncoding="gb2312"%>
<html>
<head>
    <title>成功界面 </title>
</head>
<body bgcolor="#E3E3E3">
    留言成功，单击<a href="mainServlet">这里</a>返回主界面。
</body>
</html>
```

在该页面中有一个超链接，该链接是去mainServlet，这里应该是第二次访问这个Servlet了，所以在该Servlet中可以从session取到当前用户的对象，不用再一次查询数据库，而只是把留言表中的信息查出保存，然后跳转到main.jsp。





2.6 JSP综合应用实例——开发一个简单的留言系统



◆ 12. 配置web.xml

在讲Servlet时我们说过，有一个Servlet，就要有其配置文件与其对应，而这里有两个Servlet，所以在web.xml中就应该为它们配置，其[代码](#)。

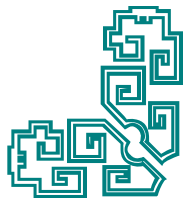
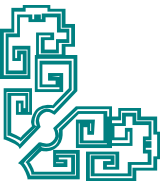
◆ 13. 创建注册页面

注册页面register.jsp[代码](#)。

◆ 14. 创建RegisterServlet类

代码实现和登录差不多，只是实现的功能不同而已，这里不多做解释，[代码](#)。当然，需要在DB类中加上插入用户名和密码的方法。该方法代码如下：

```
public boolean insertUser(String username,String pwd){
    try{
        pstmt=ct.prepareStatement("insert into [userTable] values(?,?)");
        pstmt.setString(1, username);
        pstmt.setString(2, pwd);
        pstmt.executeUpdate();
        return true;
    }catch(Exception e){
        e.printStackTrace();
        return false;
    }
}
```

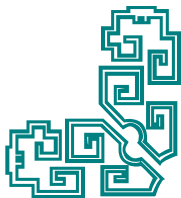
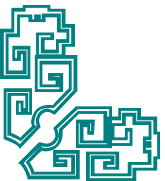


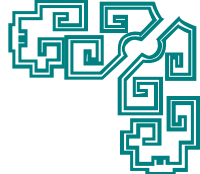


2.6 JSP综合应用实例——开发一个简单的留言系统

既然有一个Servlet类就要在web.xml中配置，这个Servlet类肯定也不能避免，所以在web.xml中应该加上如下的代码：

```
<servlet>
    <servlet-name>registerServlet</servlet-name>
    <servlet-class>servlet.RegisterServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>registerServlet</servlet-name>
    <url-pattern>/registerServlet</url-pattern>
</servlet-mapping>
```





2.6 JSP综合应用实例——开发一个简单的留言系统

◆ 15. 部署运行

部署项目JSPExample，启动Tomcat服务器，在浏览器地址栏中输入“<http://localhost:8080/JSPExample/login.jsp>”。

但当输入用户名和密码时（大家可以在数据库中插入一条记录做数据准备），服务器端报了这样的错误，如图2.24所示。

```
java.lang.ClassNotFoundException: com.microsoft.sqlserver.jdbc.SQLServerDriver
    at org.apache.catalina.loader.WebappClassLoader.loadClass(WebappClassLoader.java:1601)
    at org.apache.catalina.loader.WebappClassLoader.loadClass(WebappClassLoader.java:1526)
    at java.lang.ClassLoader.loadClassInternal(Unknown Source)
    at java.lang.Class.forName0(Native Method)
    at java.lang.Class.forName(Unknown Source)
    at db.DB.<init>(DB.java:11)
    at servlet.MainServlet.doGet(MainServlet.java:25)
    at servlet.MainServlet.doPost(MainServlet.java:54)
```

图2.24 服务器端报错信息

