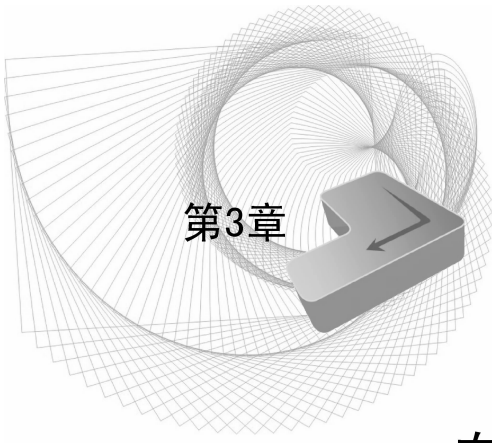


第3章

chapter 3

在 JSP 中调用 Java 代码

JSP 网页以 Java 为脚本，直接在 JSP 网页中调用 Java 代码是 JSP 技术最基本的技术点。从本章开始将正式进入对 JSP 技术的学习，同时，随着学习的深入，本书也将逐渐介绍如何编写结构良好、易于开发与维护的 JSP 页面。

3.1 基本动态技术元素

JSP 有三种基本动态技术元素，即表达式、声明、脚本。通常普通的 JSP 页面就由这三类基本元素组成。

1. 表达式

JSP 表达式的目的是将一些动态信息显示到页面的最终输出。

表达式的语法有两种，最为常用的传统语法如下：

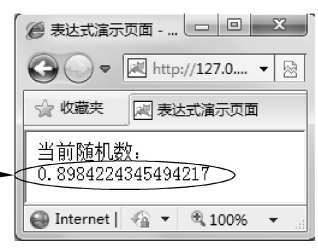
```
<% = 变量或表达式 %>
```

如果表达式中包含的是变量，则该变量的值会在最终页面适当的位置显示。而如果表达式中包含的是 Java 表达式，则会首先计算 Java 表达式的值，然后 JSP 会将该值显示于页面之中。

例如，以下 JSP 页面使用类 Math 的静态方法 random()得到一个随机数，然后利用 JSP 表达式将该随机数显示在网页之上。

```
<%@ page contentType = "text/html; charset = gb2312" language = "java" %>
<!DOCTYPE html PUBLIC " - //W3C//DTD XHTML 1.0 Transitional//EN" " http://www. w3. org/TR/
xhtml1/DTD/xhtml1 - transitional. dtd">
<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
<meta http - equiv = "Content - Type" content = "text/html; charset = gb2312" />
<title>表达式演示页面</title>
</head>
<body>
当前随机数: </br>
<% = Math.random() %>
</body>
</html>
```

JSP表达式结果显示在最终网页之中



从页面显示的结果来看，文本“当前随机数：”被正常显示，其后的 HTML 换行标记“
”也使得其后显示的内容换到新的一行中开始显示，只有“<%= Math.random()%>”一行被替换成了一个随机数。

以下是上述 JSP 页面生成的最终代码，与 JSP 页面代码进行比较可以发现，只有第一行和“<%=Math.random()%>”这一行不同，其他部分的代码完全相同。

```
                                        ← 第一行的<%@ page…%>被空白行代替
<!DOCTYPE html PUBLIC " -//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>表达式演示页面</title>
</head>
<body>
当前随机数：</br>
0.8984224345494217  ← 表达式行由一个随机数代替
</body>
</html>
```

可以看到 JSP 页面中的 HTML 元素将在生成的最终结果中被原样保留，只有 JSP 本身的代码会发生一些变化。因此，通常 JSP 页面中的静态文本（通常就是 HTML 元素）被称为“模板”。当然，模板中的内容也并非总是一成不变，例如可以通过脚本屏蔽部分模板中的代码，或者将模板中的部分代码根据需要重复多次。

注意：动态页面最终生成的 HTML 代码可以通过浏览器进行查看。对于 IE 浏览器，可以在待查看的页面上单击鼠标右键，然后从弹出的快捷菜单中选择“查看源文件”命令，或者执行“页面”→“查看源文件”命令即可。

除了上面介绍的 JSP 表达式语法形式外，JSP 的表达式元素还支持一种 XML 语法格式。JSP 表达式的 XML 语法格式如下：

```
<jsp:expression> 变量或表达式 </jsp:expression>
```

使用“<jsp:expression>Math.random()</jsp:expression>”替换上述示例 JSP 页面中的“<%=Math.random()%>”并查看页面执行效果，可以看到，两者执行的效果是完全一致的。使用 XML 语法格式在形式上令 JSP 页面更接近纯标签的页面，JSP 在使用一些第三方的库时，通常也是通过 XML 标签形式实现的。

注意：建议在同一 JSP 页面（项目）中只使用上述两种语法中的一种，而不要混合使用两种语法。这一方面是因为在同一页面（项目）中保持语法风格一致是一个良好的习惯，另一方面则是因为并非所有的服务器都支持同一页面中混合使用两种形式的语法。不过，本书所使用的 Tomcat 是支持两者在同一页面中混合使用的。

2. 声明

声明用于定义变量、方法甚至类或者静态初始块，但通常仅在声明部分定义变量及方法。在声明部分定义的变量及方法可以在 JSP 页面的表达式或脚本中使用。

声明的语法也有两种，其格式分别如下：

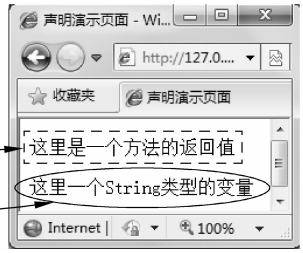
```
<%!  变量定义/方法定义/类定义  %>
```

或

```
<jsp:declaration> 变量定义/方法定义/类定义 </jsp:declaration>
```

以下代码在 JSP 的声明标记中声明了两个 String 类型的变量并为其赋值，随后则利用 JSP 表达式将其显示在网页之上。

```
<%@ page contentType="text/html; charset=gb2312" language="java" %>
<!DOCTYPE html PUBLIC " -//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>声明演示页面</title>
</head>
<body>
<%!
String s="这里一个 String 类型的变量";
String t()
{
    return "这里是一个方法的返回值";
}
%>
<p><%=t()%></p>
<p><%=s%></p>
</body>
</html>
```



与表达式类似，如果使用 XML 语法格式替换上述例程并再次查看页面效果，可以看到两者的效果是完全一致的。

在实际应用中，如果不是有特殊需要，其实变量的声明并不需要一定在声明中进行，在随后将介绍的脚本中也可以进行变量的声明。对于 JSP 而言，通常在这两处声明变量并无区别，但是在底层两者还是有本质差别的。

在运行时，一个 JSP 页面实际上会被编译成一个类，JSP 声明部分会成为这个类的成员，即在 JSP 声明部分声明的变量会成为这个类的域成员，而在 JSP 声明部分声明的方法和类则分别成为这个类的成员方法和内嵌类。而在 JSP 脚本中声明的变量则会成为这个类中名为_jspService 方法的局部变量。

3. 脚本

脚本是初学 JSP 技术时要掌握的最重要的技术，它是 JSP 动态生成网页的核心部分。在脚本中，利用 Java 的循环结构，可以生成需要反复出现的页面元素；利用 Java 的分支结构，则可以根据当时的不同条件生成不同的网页构成元素。

脚本同样也支持两种语法格式：

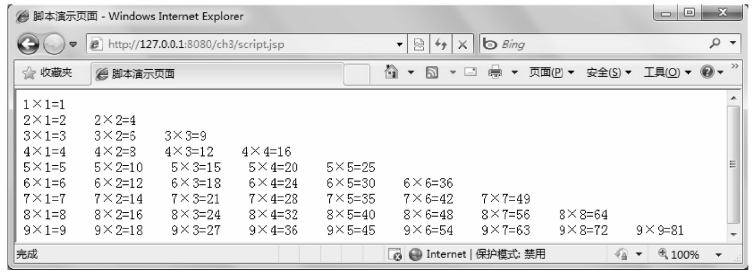
```
<% 脚本内容 %>
```

或

```
<jsp:scriptlet> 脚本内容 </jsp:scriptlet>
```

以下 JSP 页面通过脚本在页面上显示了一张乘法九九表：

```
<%@ page contentType = "text/html; charset = gb2312" language = "java" %>
<!DOCTYPE html PUBLIC " - //W3C//DTD XHTML 1. 0 Transitional//EN" " http://www. w3. org/TR/
xhtml1/DTD/xhtml1 - transitional. dtd">
<html xmlns = "http://www. w3. org/1999/xhtml">
<head>
<meta http - equiv = "Content - Type" content = "text/html; charset = gb2312" />
<title>脚本演示页面</title>
</head>
<body>
<%
for(int i = 1; i <= 9; i++)                          ← 用于控制行数的循环
{
    for(int j = 1; j <= 9; j++)                      ← 用于控制列数的循环
    {                                                  } 第1段脚本
%>
        <% = i %> x <% = j %> = <% = i * j %>       ← 显示一个乘法
<%
        if( i == j)
            break;                                   } 第2段脚本
    }
%>
    <br/>
<%
}                                                    } 第3段脚本
%>
</body>
</html>
```



注意：在实际应用中，动态生成网页往往需要多段脚本的配合。因此在编写脚本时，代码结构不应局限于一段脚本内，而应着眼于整个页面中相关的部分。

4. 注释

几乎在任何编程语言中添加足够清晰的注释已经成为一项基本的要求。在 JSP 页面中的注释与其他语言相同，它仅仅用于编程者注明代码的逻辑功能，或者说明其他事项，它对页面的生成及程序执行结果没有任何影响。

JSP 页面的注释语法如下：

```
<% -- 注释信息 -- %>
```

除了上述注释形式以外，在 JSP 的声明和脚本中也可以使用 Java 语法中的注释。实际上，在实际应用中，通常也有必要混合使用两种形式的注释，因为 JSP 页面注释语法适用于声明和脚本之外，而 Java 注释语法适用于声明和脚本之中。

以下示例 JSP 页面展示了各种不同注释的用法，由于页面中未包含任何可视元素，因

此如果查看该页面的运行效果，看到的将是一个没有任何内容的空白页。

```
<%@ page contentType="text/html; charset=gb2312" language="java" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>注释示例</title>
</head>
<body>
<%-- 此处是 JSP 注释 --%>
<%!
// 此处是声明中的 Java 注释
/* 此处也是声明中的 Java 注释 */
%>
<%
// 此处是脚本中的 Java 注释
/* 此处也是脚本中的 Java 注释 */
%>
</body>
</html>
```

注意：在 JSP 页面之中也可以使用 HTML 的注释“<!--……-->”，它与 JSP 注释的差别在于 JSP 注释最终不会被传送到客户端的浏览器上，而 HTML 注释会被原封不动地传递到客户端的浏览器中。

3.2　Java 类

类是面向对象编程中最为重要的概念，通常在 JSP 页面中很少直接定义类，但是 JSP 技术支持在页面中定义并使用类。

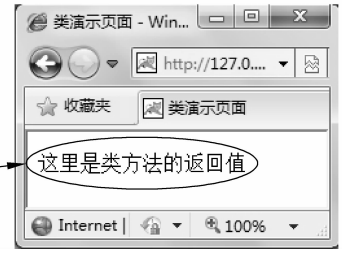
在 JSP 的基本脚本元素中，声明和脚本均支持类的定义，并且可以在任何适当的地方对所定义的类产生实例并加以引用。

以下例程在声明中定义了一个名为 Demo 的类，然后在脚本中产生了该类的一个实例，调用该实例的 sayHello 方法并将其返回值赋予一个变量，最后在表达式中将该变量的值输出到了网页之上。其完整代码如下：

```
<%@ page contentType="text/html; charset=gb2312" language="java" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>类演示页面</title>
</head>
<body>
```

```
<% -- 声明 -- %>
<%!                          ← 从这里开始是声明
class Demo                   ← -- 一个普通类的定义
{
    public String sayHello()
    {
        return "这里是类方法的返回值";
    }
}
 %>
<% -- 脚本 -- %>
<%
Demo demo = new Demo();
String s = demo.sayHello();
 %>
<% -- 利用表达式进行输出 -- %>
<p><% = s %></p>
</body>
</html>
```

在脚本中产生类的实例并调用实例的方法



如果将上述例程在声明中定义的类改为在脚本中定义，页面执行效果并不会有所不同。但从本质上说，定义在声明部分的类是一个定义在类之中的内嵌类，而定义在脚本部分的类是一个定义在方法之中的内嵌类。

注意：面向对象的编程强调以类为程序中的基本元素，但是在 JSP 页面中定义类并不是被推荐的形式，在结构良好的程序中，通常会在 Java 文件中定义类，然后在 JSP 页面中使用它。其具体实现方法将在本书后继章节中进行介绍。

3.3 JSP 页面的底层工作机制

JSP 是基于 Java 的动态网页技术，而一个 JSP 页面本质上就是一个特殊的 Java 类。一个 JSP 页面在服务器中被运行时，它首先会被服务器编译，其过程简略而言就是 JSP 页面将先被转化成为一个保存于 Java 文件中的类，然后服务器再将这个类编译成 class 文件。

例如，以下是一个包含有声明、脚本、表达式的 JSP 页面，其中也涵括了变量、方法和类的应用。

```
<% @ page contentType = "text/html; charset = gb2312" language = "java" %>
<!DOCTYPE html PUBLIC " - //W3C//DTD XHTML 1. 0 Transitional//EN" " http://www. w3. org/TR/
xhtml1/DTD/xhtml1 - transitional. dtd">
<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
<meta http - equiv = "Content - Type" content = "text/html; charset = gb2312" />
<title>综合演示页面</title>
</head>
<body>
<% -- 声明 -- %>
<%!
String s = "这里一个 String 类型的变量";
```

```
String t()
{
    return "这里是一个方法的返回值";
}
class Demo
{
    public String sayHello()
    {
        return "这里是类方法的返回值";
    }
}
%>
<% -- 脚本 -- %>
<%
String s1 = s;
String s2 = t();
Demo demo = new Demo();
String s3 = demo.sayHello();
%>
<% -- 利用表达式进行输出 -- %>
<p><% = s1 %></p>
<p><% = s2 %></p>
<p><% = s3 %></p>
</body>
</html>
```

如果将以上 JSP 页面保存于名为 all.jsp 的文件之中，它经 Tomcat 服务器转化后的完整的 Java 代码如下：

```
package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;

public final class all_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {
String s = "这里一个 String 类型的变量";
String t()
{
    return "这里是一个方法的返回值";
}
class Demo
{
    public String sayHello()
    {
        return "这里是类方法的返回值";
    }
}
```

JSP页面转化后的类名是JSP页面名加上“_jsp”

JSP页面转化后的类扩展于HttpJspBase，同时实现

同时实现JspSourceDependent接口

JSP页面中声明部分的代码被原样保留在类中

```
private static final JspFactory _jspxFactory = JspFactory.getDefaultFactory();

private static java.util.List<String> _jspx_dependants;

private javax.el.ExpressionFactory _el_expressionfactory;
private org.apache.tomcat.InstanceManager _jsp_instancemanager;

public java.util.List<String> getDependants() {
  return _jspx_dependants;
}

public void _jspInit() {
  _el_expressionfactory =
          _jspxFactory.getJspApplicationContext(getServletConfig().
                  getServletContext()).getExpressionFactory();
  _jsp_instancemanager =
          org.apache.jasper.runtime.InstanceManagerFactory.
                  getInstanceManager(getServletConfig());
}

public void _jspDestroy() {
}

public void _jspService(HttpServletRequest request, HttpServletResponse response)
      throws java.io.IOException, ServletException {

  PageContext pageContext = null;
  HttpSession session = null;
  ServletContext application = null;
  ServletConfig config = null;
  JspWriter out = null;
  Object page = this;
  JspWriter _jspx_out = null;
  PageContext _jspx_page_context = null;


  try {
    response.setContentType("text/html; charset=gb2312");
    pageContext = _jspxFactory.getPageContext(this, request, response,
                null, true, 8192, true);
    _jspx_page_context = pageContext;
    application = pageContext.getServletContext();
    config = pageContext.getServletConfig();
    session = pageContext.getSession();
    out = pageContext.getOut();
    _jspx_out = out;
```

```
      out.write("\r\n");
      out.write("<!DOCTYPE html PUBLIC \" - //W3C//DTD XHTML 1.0 Transitional//EN\" \"http://
www.w3.org/TR/xhtml1/DTD/xhtml1 - transitional.dtd\">\r\n");
      out.write("< html xmlns = \"http://www.w3.org/1999/xhtml\">\r\n");
      out.write("< head>\r\n");
       out. write( "< meta http - equiv = \" Content - Type\" content = \" text/html; charset =
gb2312\" />\r\n");
      out.write("< title>混合演示页面</title>\r\n");
      out.write("</head>\r\n");
      out.write("< body>\r\n");
      out.write('\r');
      out.write('\n');
      out.write('\r');
      out.write('\n');
      out.write('\r');
      out.write('\n');
```

```
String s1 = s;
String s2 = t();
Demo demo = new Demo();
String s3 = demo.sayHello();
```

脚本中的代码也被原样保留在方法_jspService之中

```
out.write('\r');
out.write('\n');
out.write("\r\n");
out.write("< p>");
out.print(s1);
out.write("</p>\r\n");
out.write("< p>");
out.print(s2);
out.write("</p>\r\n");
out.write("< p>");
out.print(s3);
out.write("</p>\r\n");
out.write("</body>\r\n");
out.write("</html>\r\n");
```

页面最终HTML代码由out对象生成

JSP表达式被转化成out对象的print方法的参数

```
    } catch (Throwable t) {
      if (!(t instanceof SkipPageException)){
        out = _jspx_out;
        if (out != null && out.getBufferSize() != 0)
          try { out.clearBuffer(); } catch (java.io.IOException e) {}
        if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
      }
    } finally {
      _jspxFactory.releasePageContext(_jspx_page_context);
    }
  }
}
```

注意：JSP 页面转换而成的 Java 文件可以在 Tomcat 的主目录下的 work 目录之下适当的包目录中找到。

JSP 页面几乎总能成功地被转换成相应的 Java 文件。当然，如果 JSP 页面文件中存在有语法错误，这些错误也会被转化到 Java 文件之中，而当服务器尝试编译该 Java 文件时才会报告这些错误。

JSP 页面被转换成 Java 文件并被编译的时机是在该 JSP 被访问的时候，但是并非每次访问都会执行转换与编译过程。通常一个 JSP 页面被访问时，服务器首先会查找此前是否已经为该 JSP 页面生成了 class 文件，如果已经存在对应的 class 文件，则继续比较该 JSP 页面文件的修改时间与对应的 class 文件的修改时间，如果 JSP 页面修改时间不晚于 class 文件的修改时间，则该 JSP 页面将不会被再次转换与编译；而如果不存在与该 JSP 页面对应的 class 文件或者 JSP 页面修改时间晚于 class 文件的修改时间，则该 JSP 页面将会被转换与编译。最终服务器将调用生成的 class 文件生成对该页面请求的响应信息。所以，如果理解了上述过程，同时也就理解了 JSP 页面第一次被访问时通常比较慢的原因。

3.4 小结

本章介绍了 JSP 开发技术中最基础的内容，通过本章的学习，应当可以通过声明、表达式和脚本来构建一些简单的 JSP 动态网页。同时，本章也介绍了在 JSP 页面中直接定义并使用类的方法。在最后则介绍了 JSP 页面的底层工作机制，了解 JSP 在底层的运作情况，对于写好上层代码及调试 JSP 页面都是有帮助的。

3.5 练习

1. JSP 页面中最基本的三类动态技术元素是什么？
2. JSP 页面的三类动态技术元素的作用分别是什么？
3. 编写一个能够在一次 Tomcat 运行期间统计该页面访问次数的 JSP 页面。