用 web.xml 控制 Web 应用的行为

目录

1 定义头和根元素

部署描述符文件就像所有 XML 文件一样，必须以一个 XML 头开始。这个头声明可以使用的 XML 版本并给出文件的字符编码。
DOCYTPE 声明必须立即出现在此头之后。这个声明告诉服务器适用的 servlet 规范的版本（如 2.2 或 2.3）并指定管理此文件其余部分内容的语法的 DTD(Document Type Definition，文档类型定义)。

所有部署描述符文件的顶层（根）元素为 web-app。请注意，XML 元素不像 HTML，他们是大小写敏感的。因此，web-App 和 WEB-APP 都是不合法的，web-app 必须用小写。

2 部署描述符文件内的元素次序

XML 元素不仅是大小写敏感的，而且它们还对出现在其他元素中的次序敏感。例如，XML 头必须是文件中的第一项，DOCTYPE 声明必须是第二项，而 web-app 元素必须是第三项。在 web-app 元素内，元素的次序也很重要。服务器不一定强制要求这种次序，但它们允许（实际上有些服务器就是这样做的）完全拒绝执行含有次序不正确的元素的 Web 应用。这表示使用非标准元素次序的 web.xml 文件是不可移植的。

下面的列表给出了所有可直接出现在 web-app 元素内的合法元素所必需的次序。例如，此列表说明 servlet 元素必须出现在所有 servlet-mapping 元素之前。请注意，所有这些元素都是可选的。因此，可以省略掉某一元素，但不能把它放于不正确的位置。

- icon icon 元素指出 IDE 和 GUI 工具用来表示 Web 应用的一个和两个图像文件的位置。
- display-name display-name 元素提供 GUI 工具可能会用来标记这个特定的 Web 应用的一个名称。
- description description 元素给出与此有关的说明性文本。
- context-param context-param 元素声明应用范围内的初始化参数。
- filter 过滤器元素将一个名字与一个实现 javax.servlet.Filter 接口的类相关联。
- filter-mapping 一旦命名了一个过滤器，就要利用 filter-mapping 元素把它与一个或多个 servlet 或 JSP 页面相关联。
- listener servlet API 的版本 2.3 增加了对事件监听程序的支持，事件监听程序在建立、修改和删除会话或 servlet 环境时得到通知。Listener 元素指出事件监听程序类。
- servlet 在向 servlet 或 JSP 页面制定初始化参数或定制 URL 时，必须首先命名 servlet 或 JSP 页面。Servlet 元素就是用来完成此项任务的。
- servlet-mapping 服务器一般为 servlet 提供一个缺省的 URL：http://host/webAppPrefix/servlet/ServletName。但是，常常会更改这个 URL，以便 servlet 可以访问初始化参数或更容易地处理相对 URL。在更改缺省 URL 时，使用 servlet-mapping 元素。
- session-config 如果某个会话在一定时间内未被访问，服务器可以抛弃它以节省内存。可通过使用 HttpSession 的 setMaxInactiveInterval 方法明确设置单个会话对象

的超时值，或者可利用 session-config 元素制定缺省超时值。

- mime-mapping 如果 Web 应用具有想到特殊的文件，希望能保证给他们分配特定的 MIME 类型，则 mime-mapping 元素提供这种保证。
- welcom-file-list welcome-file-list 元素指示服务器在收到引用一个目录名而不是文件名的 URL 时，使用哪个文件。
- error-page error-page 元素使得在返回特定 HTTP 状态代码时，或者特定类型的异常被抛出时，能够制定将要显示的页面。
- taglib taglib 元素对标记库描述符文件（Tag Libraryu Descriptor file）指定别名。此功能使你能够更改 TLD 文件的位置，而不用编辑使用这些文件的 JSP 页面。
- resource-env-ref resource-env-ref 元素声明与资源相关的一个管理对象。
- resource-ref resource-ref 元素声明一个资源工厂使用的外部资源。
- security-constraint security-constraint 元素制定应该保护的 URL。它与 login-config 元素联合使用
- login-config 用 login-config 元素来指定服务器应该怎样给试图访问受保护页面的用户授权。它与 sercurity-constraint 元素联合使用。
- security-role security-role 元素给出安全角色的一个列表,这些角色将出现在 servlet 元素内的 security-role-ref 元素的 role-name 子元素中。分别地声明角色可使高级 IDE 处理安全信息更为容易。
- env-entry env-entry 元素声明 Web 应用的环境项。
- ejb-ref ejb-ref 元素声明一个 EJB 的主目录的引用。
- ejb-local-ref ejb-local-ref 元素声明一个 EJB 的本地主目录的应用。

3 分配名称和定制的 UL

在 web.xml 中完成的一个最常见的任务是对 servlet 或 JSP 页面给出名称和定制的 URL。用 servlet 元素分配名称，使用 servlet-mapping 元素将定制的 URL 与刚分配的名称相关联。

3.1 分配名称

为了提供初始化参数，对 servlet 或 JSP 页面定义一个定制 URL 或分配一个安全角色，必须首先给 servlet 或 JSP 页面一个名称。可通过 servlet 元素分配一个名称。最常见的格式包括 servlet-name 和 servlet-class 子元素（在 web-app 元素内），如下所示：

```
<servlet>
  <servlet-name>Test</servlet-name>
  <servlet-class>moreservlets.TestServlet</servlet-class>
</servlet>
```

这表示位于 WEB-INF/classes/moreservlets/TestServlet 的 servlet 已经得到了注册名 Test。给 servlet 一个名称具有两个主要的含义。首先，初始化参数、定制的 URL 模式以及其他定制通过此注册名而不是类名引用此 servlet。其次,可在 URL 而不是类名中使用此名称。因此，

利用刚才给出的定义，URL http://host/webAppPrefix/servlet/Test 可用于 http://host/webAppPrefix/servlet/moreservlets.TestServlet 的场所。

请记住：XML 元素不仅是大小写敏感的，而且定义它们的次序也很重要。例如，web-app 元素内所有 servlet 元素必须位于所有 servlet-mapping 元素（下一小节介绍）之前，而且还要位于 5.6 节和 5.11 节讨论的与过滤器或文档相关的元素（如果有的话）之前。类似地，servlet 的 servlet-name 子元素也必须出现在 servlet-class 之前。5.2 节“部署描述符文件内的元素次序”将详细介绍这种必需的次序。

例如，程序清单 5-1 给出了一个名为 TestServlet 的简单 servlet，它驻留在 moreservlets 程序包中。因为此 servlet 是扎根在一个名为 deployDemo 的目录中的 Web 应用的组成部分，所以 TestServlet.class 放在 deployDemo/WEB-INF/classes/moreservlets 中。程序清单 5-2 给出将放置在 deployDemo/WEB-INF/ 内的 web.xml 文件的一部分。此 web.xml 文件使用 servlet-name 和 servlet-class 元素将名称 Test 与 TestServlet.class 相关联。图 5-1 和图 5-2 分别显示利用缺省 URL 和注册名调用 TestServlet 时的结果。

程序清单 5-1 TestServlet.java

```
package moreservlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/** Simple servlet used to illustrate servlet naming
 *  and custom URLs.
 *  <P>
 *  Taken from More Servlets and JavaServer Pages
 *  from Prentice Hall and Sun Microsystems Press,
 *  http://www.moreservlets.com/.
 *  &copy; 2002 Marty Hall; may be freely used or adapted.
 */

public class TestServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String uri = request.getRequestURI();
    out.println(ServletUtilities.headWithTitle("Test Servlet") +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H2>URI: " + uri + "</H2>\n" +
                "</BODY></HTML>");
  }
```

```
}
```

程序清单 5-2 web.xml（说明 servlet 名称的摘录）

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC  "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
     "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <!-- … -->
  <servlet>
    <servlet-name>Test</servlet-name>
    <servlet-class>moreservlets.TestServlet</servlet-class>
  </servlet>
  <!-- … -->
</web-app>
```

3.2 定义定制的 URL

大多数服务器具有一个缺省的 serlvet URL：

http://host/webAppPrefix/servlet/packageName.ServletName。**虽然在开发中使用这个** URL **很方便，但是我们常常会希望另一个** URL **用于部署。例如，可能会需要一个出现在** Web **应用顶层的** URL（**如，**http://host/webAppPrefix/Anyname），**并且在此** URL **中没有** servlet **项。位于顶层的** URL **简化了相对** URL **的使用。此外，对许多开发人员来说，顶层** URL **看上去比更长更麻烦的缺省** URL **更简短。**

事实上，有时需要使用定制的 URL。**比如，你可能想关闭缺省** URL **映射，以便更好地强制实施安全限制或防止用户意外地访问无初始化参数的** servlet。**如果你禁止了缺省的** URL，**那么你怎样访问** servlet **呢？这时只有使用定制的** URL **了。**

为了分配一个定制的 URL，**可使用** servlet-mapping **元素及其** servlet-name **和** url-pattern **子元素。**Servlet-name **元素提供了一个任意名称，可利用此名称引用相应的** servlet **；**url-pattern **描述了相对于** Web **应用的根目录的** URL。url-pattern **元素的值必须以斜杠（/）起始。**

下面给出一个简单的 web.xml **摘录，它允许使用** URL http://host/webAppPrefix/UrlTest **而不是**http://host/webAppPrefix/servlet/Test**或** http://host/webAppPrefix/servlet/moreservlets.TestServlet。**请注意，仍然需要** XML **头、**DOCTYPE **声明以及** web-app **封闭元素。此外，可回忆一下，**XML **元素出现地次序不是随意的。特别是，需要把所有** servlet **元素放在所有** servlet-mapping **元素之前。**

```
<servlet>
  <servlet-name>Test</servlet-name>
  <servlet-class>moreservlets.TestServlet</servlet-class>
</servlet>
<!-- ... -->
```

```
<servlet-mapping>
  <servlet-name>Test</servlet-name>
  <url-pattern>/UrlTest</url-pattern>
</servlet-mapping>
```

URL 模式还可以包含通配符。例如，下面的小程序指示服务器发送所有以 Web 应用的 URL 前缀开始，以..asp 结束的请求到名为 BashMS 的 servlet。

```
<servlet>
  <servlet-name>BashMS</servlet-name>
  <servlet-class>msUtils.ASPTranslator</servlet-class>
</servlet>
<!-- ... -->
<servlet-mapping>
  <servlet-name>BashMS</servlet-name>
  <url-pattern>/*.asp</url-pattern>
</servlet-mapping>
```

3.3 命名 JSP 页面

因为 JSP 页面要转换成 sevlet，自然希望就像命名 servlet 一样命名 JSP 页面。毕竟，JSP 页面可能会从初始化参数、安全设置或定制的 URL 中受益，正如普通的 serlvet 那样。虽然 JSP 页面的后台实际上是 servlet 这句话是正确的，但存在一个关键的猜疑：即，你不知道 JSP 页面的实际类名（因为系统自己挑选这个名字）。因此，为了命名 JSP 页面，可将 jsp-file 元素替换为 servlet-calss 元素，如下所示：

```
<servlet>
  <servlet-name>Test</servlet-name>
  <jsp-file>/TestPage.jsp</jsp-file>
</servlet>
```

命名 JSP 页面的原因与命名 servlet 的原因完全相同：即为了提供一个与定制设置（如，初始化参数和安全设置）一起使用的名称，并且，以便能更改激活 JSP 页面的 URL（比方说，以便多个 URL 通过相同页面得以处理，或者从 URL 中去掉.jsp 扩展名）。但是，在设置初始化参数时，应该注意，JSP 页面是利用 jspInit 方法，而不是 init 方法读取初始化参数的。

例如，程序清单 5-3 给出一个名为 TestPage.jsp 的简单 JSP 页面，它的工作只是打印出用来激活它的 URL 的本地部分。TestPage.jsp 放置在 deployDemo 应用的顶层。程序清单 5-4 给出了用来分配一个注册名 PageName，然后将此注册名与 http://host/webAppPrefix/UrlTest2/anything 形式的 URL 相关联的 web.xml 文件（即，deployDemo/WEB-INF/web.xml）的一部分。

程序清单 5-3	TestPage.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
```

```
<HEAD>
   <TITLE>
  JSP Test Page
   </TITLE>
</HEAD>
<BODY BGCOLOR="#FDF5E6">
<H2>URI:   <%=   request.getRequestURI()   %></H2>
</BODY>
</HTML>
```

程序清单 5-4 web.xml（说明 JSP 页命名的摘录）

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
      PUBLIC  "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
       "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
   <!-- ... -->
   <servlet>
      <servlet-name>PageName</servlet-name>
      <jsp-file>/TestPage.jsp</jsp-file>
   </servlet>
   <!-- ... -->
   <servlet-mapping>
      <servlet-name> PageName </servlet-name>
      <url-pattern>/UrlTest2/*</url-pattern>
   </servlet-mapping>
   <!-- ... -->
</web-app>
```

4 禁止激活器 servlet

对 servlet **或** JSP **页面建立定制** URL **的一个原因是，这样做可以注册从** init（servlet）**或** jspInit（JSP **页面）方法中读取得初始化参数。但是，初始化参数只在是利用定制** URL **模式或注册名访问** servlet **或** JSP **页面时可以使用，用缺省** URL http://host/webAppPrefix/servlet/ServletName **访问时不能使用。因此，你可能会希望关闭缺省** URL**，这样就不会有人意外地调用初始化** servlet **了。这个过程有时称为禁止激活器** servlet**，因为多数服务器具有一个用缺省的** servlet URL **注册的标准** servlet**，并激活缺省的** URL **应用的实际** servlet**。**

有两种禁止此缺省 URL 的主要方法：

- 在每个 Web 应用中重新映射/servlet/模式。
- 全局关闭激活器 servlet。

重要的是应该注意到，虽然重新映射每个 Web 应用中的/servlet/模式比彻底禁止激活 servlet 所做的工作更多，但重新映射可以用一种完全可移植的方式来完成。相反，全局禁止激活器 servlet 完全是针对具体机器的，事实上有的服务器（如 ServletExec）没有这样的选择。下面的讨论对每个 Web 应用重新映射/servlet/ URL 模式的策略。后面提供在 Tomcat 中全局禁止激活器 servlet 的详细内容。

4.1 重新映射/servlet/URL 模式

在一个特定的 Web 应用中禁止以http://host/webAppPrefix/servlet/ 开始的 URL 的处理非常简单。所需做的事情就是建立一个错误消息 servlet，并使用前一节讨论的 url-pattern 元素将所有匹配请求转向该 servlet。只要简单地使用：

```
<url-pattern>/servlet/*</url-pattern>
```

作为 servlet-mapping 元素中的模式即可。

例如，程序清单 5-5 给出了将 SorryServlet servlet（程序清单 5-6）与所有以 http://host/webAppPrefix/servlet/ 开头的 URL 相关联的部署描述符文件的一部分。

程序清单 5-5 web.xml（说明 JSP 页命名的摘录）

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC  "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
     "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <!-- ... -->
  <servlet>
    <servlet-name>Sorry</servlet-name>
    <servlet-class>moreservlets.SorryServlet</servlet-class>
  </servlet>
  <!-- ... -->
  <servlet-mapping>
    <servlet-name> Sorry </servlet-name>
    <url-pattern>/servlet/*</url-pattern>
  </servlet-mapping>
  <!-- ... -->
</web-app>
```

程序清单 5-6 SorryServlet.java

```
package moreservlets;
```

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/** Simple servlet used to give error messages to
 *  users who try to access default servlet URLs
 *  (i.e., http://host/webAppPrefix/servlet/ServletName)
 *  in Web applications that have disabled this
 *  behavior.
 *  <P>
 *  Taken from More Servlets and JavaServer Pages
 *  from Prentice Hall and Sun Microsystems Press,
 *  http://www.moreservlets.com/.
 *  &copy; 2002 Marty Hall; may be freely used or adapted.
 */

public class SorryServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Invoker Servlet Disabled.";
    out.println(ServletUtilities.headWithTitle(title) +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H2>" + title + "</H2>\n" +
                "Sorry, access to servlets by means of\n" +
                "URLs that begin with\n" +
                "http://host/webAppPrefix/servlet/\n" +
                "has been disabled.\n" +
                "</BODY></HTML>");
  }

  public void doPost(HttpServletRequest request,
                     HttpServletResponse response)
      throws ServletException, IOException {
    doGet(request, response);
  }
}
```

4.2 全局禁止激活器：Tomcat

Tomcat 4 中用来关闭缺省 URL 的方法与 Tomcat 3 中所用的很不相同。下面介绍这两种方法：

1．禁止激活器： Tomcat 4

Tomcat 4 用与前面相同的方法关闭激活器 servlet，即利用 web.xml 中的 url-mapping 元素进行关闭。不同之处在于 Tomcat 使用了放在 install_dir/conf 中的一个服务器专用的全局 web.xml 文件，而前面使用的是存放在每个 Web 应用的 WEB-INF 目录中的标准 web.xml 文件。

因此，为了在 Tomcat 4中关闭激活器 servlet，只需在 install_dir/conf/web.xml 中简单地注释出/servlet/* URL 映射项即可，如下所示：

```
<!--
   <servlet-mapping>
      <servlet-name>invoker</servlet-name>
      <url-pattern>/servlet/*</url-pattern>
   </servlet-mapping>
 -->
```

再次提醒，应该注意这个项是位于存放在 install_dir/conf 的 Tomcat 专用的 web.xml 文件中的，此文件不是存放在每个 Web 应用的 WEB-INF 目录中的标准 web.xml。

2．禁止激活器：Tomcat3

在 Apache Tomcat 的版本 3 中，通过在 install_dir/conf/server.xml 中注释出 InvokerInterceptor 项全局禁止缺省 servlet URL。例如，下面是禁止使用缺省 servlet URL 的 server.xml 文件的一部分。

```
<!--
<RequsetInterceptor
   className="org.apache.tomcat.request.InvokerInterceptor"
  debug="0"  prefix="/servlet/"  />
-->
```

5 初始化和预装载 servlet 与 JSP 页面

这里讨论控制 servlet 和 JSP 页面的启动行为的方法。特别是，说明了怎样分配初始化参数以及怎样更改服务器生存期中装载 servlet 和 JSP 页面的时刻。

5.1 分配 servlet 初始化参数

利用 init-param 元素向 servlet 提供初始化参数，init-param 元素具有 param-name 和 param-value 子元素。例如，在下面的例子中，如果 initServlet servlet 是利用它的注册名

（InitTest）访问的，它将能够从其方法中调用 getServletConfig().getInitParameter("param1") 获得“ Value 1”，调用 getServletConfig().getInitParameter("param2")获得“ 2 ”。

```
<servlet>
  <servlet-name>InitTest</servlet-name>
  <servlet-class>moreservlets.InitServlet</servlet-class>
  <init-param>
    <param-name>param1</param-name>
    <param-value>value1</param-value>
  </init-param>
  <init-param>
    <param-name>param2</param-name>
    <param-value>2</param-value>
  </init-param>
</servlet>
```

在涉及初始化参数时，有几点需要注意：

- **返回值。**GetInitParameter **的返回值总是一个** String**。因此，在前一个例子中，可对** param2 **使用** Integer.parseInt **获得一个** int**。**
- JSP **中的初始化。**JSP **页面使用** jspInit **而不是** init**。**JSP **页面还需要使用** jsp-file **元素代替** servlet-class**。**
- **缺省** URL**。初始化参数只在通过它们的注册名或与它们注册名相关的定制** URL **模式访问** Servlet **时可以使用。因此，在这个例子中，**param1 **和** param2 **初始化参数将能够在使用** URL http://host/webAppPrefix/servlet/InitTest **时可用，但在使用** URL http://host/webAppPrefix/servlet/myPackage.InitServlet **时不能使用。**

例如，程序清单 5-7 **给出一个名为** InitServlet **的简单** servlet**，它使用** init **方法设置** firstName **和** emailAddress **字段。程序清单** 5-8 **给出分配名称** InitTest **给** servlet **的** web.xml **文件。**

程序清单 5-7 InitServlet.java

```
package moreservlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/** Simple servlet used to illustrate servlet
 *  initialization parameters.
 *  <P>
 *  Taken from More Servlets and JavaServer Pages
 *  from Prentice Hall and Sun Microsystems Press,
 *  http://www.moreservlets.com/.
 *  &copy; 2002 Marty Hall; may be freely used or adapted.
 */
```

```
public class InitServlet extends HttpServlet {
  private String firstName, emailAddress;

  public void init() {
    ServletConfig config = getServletConfig();
     firstName = config.getInitParameter("firstName");
    emailAddress = config.getInitParameter("emailAddress");
   }

  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
     response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String uri = request.getRequestURI();
    out.println(ServletUtilities.headWithTitle("Init Servlet") +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H2>Init Parameters:</H2>\n" +
                 "<UL>\n" +
               "<LI>First name: " + firstName + "\n" +
               "<LI>Email address: " + emailAddress + "\n" +
                 "</UL>\n" +
                 "</BODY></HTML>");
   }
}
```

程序清单 5-8 web.xml（说明初始化参数的摘录）

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
     PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
      "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <!-- ... -->
  <servlet>
    <servlet-name>InitTest</servlet-name>
    <servlet-class>moreservlets.InitServlet</servlet-class>
    <init-param>
      <param-name>firstName</param-name>
      <param-value>Larry</param-value>
    </init-param>
    <init-param>
```

```
        <param-name>emailAddress</param-name>
        <param-value>Ellison@Microsoft.com</param-value>
      </init-param>
    </servlet>
    <!-- ... -->
</web-app>
```

5.2 分配 JSP 初始化参数

给 JSP 页面提供初始化参数在三个方面不同于给 servlet 提供初始化参数。

1）使用 jsp-file 而不是 servlet-class。因此，WEB-INF/web.xml 文件的 servlet 元素如下所示：

```
<servlet>
  <servlet-name>PageName</servlet-name>
  <jsp-file>/RealPage.jsp</jsp-file>
  <init-param>
    <param-name>...</param-name>
    <param-value>...</param-value>
  </init-param>
  ...
</servlet>
```

2) 几乎总是分配一个明确的 URL 模式。对 servlet，一般相应地使用以 http://host/webAppPrefix/servlet/ 开始的缺省 URL。只需记住，使用注册名而不是原名称即可。这对于 JSP 页面在技术上也是合法的。例如，在上面给出的例子中，可用 URL http://host/webAppPrefix/servlet/PageName 访问 RealPage.jsp 的对初始化参数具有访问权的版本。但在用于 JSP 页面时，许多用户似乎不喜欢应用常规的 servlet 的 URL。此外，如果 JSP 页面位于服务器为其提供了目录清单的目录中（如，一个既没有 index.html 也没有 index.jsp 文件的目录），则用户可能会连接到此 JSP 页面，单击它，从而意外地激活未初始化的页面。因此，好的办法是使用 url-pattern（5.3 节）将 JSP 页面的原 URL 与注册的 servlet 名相关联。这样，客户机可使用 JSP 页面的普通名称，但仍然激活定制的版本。例如，给定来自项目 1 的 servlet 定义，可使用下面的 servlet-mapping 定义：

```
<servlet-mapping>
  <servlet-name>PageName</servlet-name>
  <url-pattern>/RealPage.jsp</url-pattern>
</servlet-mapping>
```

3）JSP 页使用 jspInit 而不是 init。自动从 JSP 页面建立的 servlet 或许已经使用了 inti 方法。因此，使用 JSP 声明提供一个 init 方法是不合法的，必须制定 jspInit 方法。

为了说明初始化 JSP 页面的过程，程序清单 5-9 给出了一个名为 InitPage.jsp 的 JSP 页面，它包含一个 jspInit 方法且放置于 deployDemo Web 应用层次结构的顶层。一般，http://host/deployDemo/InitPage.jsp 形式的 URL 将激活此页面的不具有初始化参数访问权的版本，从而将对 firstName 和 emailAddress 变量显示 null。但是，web.xml 文件（程序清单

5-10）分配了一个注册名，然后将该注册名与 URL 模式/InitPage.jsp 相关联。

程序清单 5-9 InitPage.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>JSP Init Test</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<H2>Init Parameters:</H2>
<UL>
  <LI>First name: <%= firstName %>
  <LI>Email address: <%= emailAddress %>
</UL>
</BODY></HTML>
<%!
private String firstName, emailAddress;

public void jspInit() {
  ServletConfig config = getServletConfig();
   firstName = config.getInitParameter("firstName");
  emailAddress = config.getInitParameter("emailAddress");
}
%>
```

程序清单 5-10 web.xml（说明 JSP 页面的 init 参数的摘录）

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
      PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
       "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
   <!-- ... -->
   <servlet>
      <servlet-name>InitPage</servlet-name>
      <jsp-file>/InitPage.jsp</jsp-file>
      <init-param>
         <param-name>firstName</param-name>
         <param-value>Bill</param-value>
      </init-param>
      <init-param>
         <param-name>emailAddress</param-name>
         <param-value>gates@oracle.com</param-value>
      </init-param>
```

```
  </servlet>
  <!-- ... -->
  <servlet-mapping>
    <servlet-name> InitPage</servlet-name>
    <url-pattern>/InitPage.jsp</url-pattern>
  </servlet-mapping>
  <!-- ... -->
</web-app>
```

5.3 提供应用范围内的初始化参数

一般，对单个地 servlet 或 JSP 页面分配初始化参数。指定的 servlet 或 JSP 页面利用 ServletConfig 的 getInitParameter 方法读取这些参数。但是，在某些情形下，希望提供可由任意 servlet 或 JSP 页面借助 ServletContext 的 getInitParameter 方法读取的系统范围内的初始化参数。

可利用 context-param 元素声明这些系统范围内的初始化值。context-param 元素应该包含 param-name、param-value 以及可选的 description 子元素，如下所示：

```
<context-param>
  <param-name>support-email</param-name>
  <param-value>blackhole@mycompany.com</param-value>
</context-param>
```

可回忆一下，为了保证可移植性，web.xml 内的元素必须以正确的次序声明。但这里应该注意，context-param 元素必须出现任意与文档有关的元素（icon、display-name 或 description）之后及 filter、filter-mapping、listener 或 servlet 元素之前。

5.4 在服务器启动时装载 servlet

假如 servlet 或 JSP 页面有一个要花很长时间执行的 init（servlet）或 jspInit（JSP）方法。例如，假如 init 或 jspInit 方法从某个数据库或 ResourceBundle 查找产量。这种情况下，在第一个客户机请求时装载 servlet 的缺省行为将对第一个客户机产生较长时间的延迟。因此，可利用 servlet 的 load-on-startup 元素规定服务器在第一次启动时装载 servlet。下面是一个例子。

```
<servlet>
  <servlet-name> … </servlet-name>
  <servlet-class> … </servlet-class>   <!-- Or jsp-file -->
  <load-on-startup/>
</servlet>
```

可以为此元素体提供一个整数而不是使用一个空的 load-on-startup。想法是服务器应该在装载较大数目的 servlet 或 JSP 页面之前装载较少数目的 servlet 或 JSP 页面。例如，下面

的 servlet 项（放置在 Web 应用的 WEB-INF 目录下的 web.xml 文件中的 web-app 元素内）将指示服务器首先装载和初始化 SearchServlet，然后装载和初始化由位于 Web 应用的 result 目录中的 index.jsp 文件产生的 servlet。

```
<servlet>
  <servlet-name>Search</servlet-name>
  <servlet-class>myPackage.SearchServlet</servlet-class> <!-- Or jsp-file -->
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet>
  <servlet-name>Results</servlet-name>
  <servlet-class>/results/index.jsp</servlet-class> <!-- Or jsp-file -->
  <load-on-startup>2</load-on-startup>
</servlet>
```

6 声明过滤器

servlet 版本 2.3 引入了过滤器的概念。虽然所有支持 servlet API 版本 2.3 的服务器都支持过滤器，但为了使用与过滤器有关的元素，必须在 web.xml 中使用版本 2.3 的 DTD。

过滤器可截取和修改进入一个 servlet 或 JSP 页面的请求或从一个 servlet 或 JSP 页面发出的相应。在执行一个 servlet 或 JSP 页面之前，必须执行第一个相关的过滤器的 doFilter 方法。在该过滤器对其 FilterChain 对象调用 doFilter 时，执行链中的下一个过滤器。如果没有其他过滤器，servlet 或 JSP 页面被执行。过滤器具有对到来的 ServletRequest 对象的全部访问权，因此，它们可以查看客户机名、查找到来的 cookie 等。为了访问 servlet 或 JSP 页面的输出，过滤器可将响应对象包裹在一个替身对象（stand-in object）中，比方说把输出累加到一个缓冲区。在调用 FilterChain 对象的 doFilter 方法之后，过滤器可检查缓冲区，如有必要，就对它进行修改，然后传送到客户机。

例如，程序清单 5-11 帝国难以了一个简单的过滤器，只要访问相关的 servlet 或 JSP 页面，它就截取请求并在标准输出上打印一个报告（开发过程中在桌面系统上运行时，大多数服务器都可以使用这个过滤器）。

程序清单 5-11 ReportFilter.java

```
package moreservlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

/** Simple filter that prints a report on the standard output
 *  whenever the associated servlet or JSP page is accessed.
 *  <P>
```

```
 *    Taken from More Servlets and JavaServer Pages
 *   from Prentice Hall and Sun Microsystems Press,
 *    http://www.moreservlets.com/.
 *   &copy; 2002 Marty Hall; may be freely used or adapted.
 */

public class ReportFilter implements Filter {
  public void doFilter(ServletRequest request,
                       ServletResponse response,
                       FilterChain chain)
      throws ServletException, IOException {
    HttpServletRequest req = (HttpServletRequest)request;
    System.out.println(req.getRemoteHost() +
                       " tried to access " +
                       req.getRequestURL() +
                       " on " + new Date() + ".");
    chain.doFilter(request,response);
  }

  public void init(FilterConfig config)
      throws ServletException {
  }

  public void destroy() {}
}
```

一旦建立了一个过滤器，可以在 web.xml **中利用** filter **元素以及** filter-name**（任意名称）、**file-class**（完全限定的类名）和（可选的）**init-params **子元素声明它。请注意，元素在** web.xml **的** web-app **元素中出现的次序不是任意的；允许服务器（但不是必需的）强制所需的次序，并且实际中有些服务器也是这样做的。但这里要注意，所有** filter **元素必须出现在任意** filter-mapping **元素之前，**filter-mapping **元素又必须出现在所有** servlet **或** servlet-mapping **元素之前。**

例如，给定上述的 ReportFilter **类，可在** web.xml **中作出下面的** filter **声明。它把名称** Reporter **与实际的类** ReportFilter **（位于** moreservlets **程序包中）相关联。**

```
<filter>
  <filter-name>Reporter</filter-name>
  <filter-class>moresevlets.ReportFilter</filter-class>
</filter>
```

一旦命名了一个过滤器，可利用 filter-mapping **元素把它与一个或多个** servlet **或** JSP **页面相关联。关于此项工作有两种选择。**

首先，可使用 filter-name **和** servlet-name **子元素把此过滤器与一个特定的** servlet **名（此** servlet **名必须稍后在相同的** web.xml **文件中使用** servlet **元素声明）关联。例如，下面的程序片断指示系统只要利用一个定制的** URL **访问名为** SomeServletName **的** servlet **或** JSP **页面，**

就运行名为 Reporter 的过滤器。

```
<filter-mapping>
  <filter-name>Reporter</filter-name>
  <servlet-name>SomeServletName</servlet-name>
</filter-mapping>
```

其次，可利用 filter-name 和 url-pattern 子元素将过滤器与一组 servlet、JSP 页面或静态内容相关联。例如，相面的程序片段指示系统只要访问 Web 应用中的任意 URL，就运行名为 Reporter 的过滤器。

```
<filter-mapping>
  <filter-name>Reporter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

例如，程序清单 5-12 给出了将 ReportFilter 过滤器与名为 PageName 的 servlet 相关联的 web.xml 文件的一部分。名字 PageName 依次又与一个名为 TestPage.jsp 的 JSP 页面以及以模式http://host/webAppPrefix/UrlTest2/ 开头的 URL 相关联。TestPage.jsp 的源代码已经 JSP 页面命名的谈论在前面的 3 节“分配名称和定制的 URL”中给出。事实上，程序清单 5-12 中的 servlet 和 servlet-name 项从该节原封不动地拿过来的。给定这些 web.xml 项，可看到下面的标准输出形式的调试报告（换行是为了容易阅读）。

```
audit.irs.gov tried to access
http://mycompany.com/deployDemo/UrlTest2/business/tax-plan.html
on Tue Dec 25 13:12:29 EDT 2001.
```

程序清单 5-12 Web.xml（说明 filter 用法的摘录）

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC  "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
     "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <filter>
    <filter-name>Reporter</filter-name>
    <filter-class>moresevlets.ReportFilter</filter-class>
  </filter>
  <!-- ... -->
  <filter-mapping>
    <filter-name>Reporter</filter-name>
    <servlet-name>PageName</servlet-name>
  </filter-mapping>
  <!-- ... -->
  <servlet>
    <servlet-name>PageName</servlet-name>
    <jsp-file>/RealPage.jsp</jsp-file>
```

```
    </servlet>
    <!-- ... -->
    <servlet-mapping>
        <servlet-name> PageName </servlet-name>
        <url-pattern>/UrlTest2/*</url-pattern>
    </servlet-mapping>
<!-- ... -->
</web-app>
```

7 指定欢迎页

假如用户提供了一个像http://host/webAppPrefix/directoryName/ 这样的包含一个目录名但没有包含文件名的 URL，会发生什么事情呢？用户能得到一个目录表？一个错误？还是标准文件的内容？如果得到标准文件内容，是 index.html、index.jsp、default.html、default.htm 或别的什么东西呢？

Welcome-file-list 元素及其辅助的 welcome-file 元素解决了这个模糊的问题。例如，下面的 web.xml 项指出，如果一个 URL 给出一个目录名但未给出文件名，服务器应该首先试用 index.jsp，然后再试用 index.html。如果两者都没有找到，则结果有赖于所用的服务器（如一个目录列表）。

```
<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>index.html</welcome-file>
</welcome-file-list>
```

虽然许多服务器缺省遵循这种行为，但不一定必须这样。因此，明确地使用 welcom-file-list 保证可移植性是一种良好的习惯。

8 指定处理错误的页面

现在我了解到，你在开发 servlet 和 JSP 页面时从不会犯错误，而且你的所有页面是那样的清晰，一般的程序员都不会被它们的搞糊涂。但是，是人总会犯错误的，用户可能会提供不合规定的参数，使用不正确的 URL 或者不能提供必需的表单字段值。除此之外，其它开发人员可能不那么细心，他们应该有些工具来克服自己的不足。

error-page 元素就是用来克服这些问题的。它有两个可能的子元素，分别是：error-code 和 exception-type。第一个子元素 error-code 指出在给定的 HTTP 错误代码出现时使用的 URL。第二个子元素 excpetion-type 指出在出现某个给定的 Java 异常但未捕捉到时使用的 URL。error-code 和 exception-type 都利用 location 元素指出相应的 URL。此 URL 必须以/开始。location所指出的位置处的页面可通过查找 HttpServletRequest 对象的两个专门的属性来访问关于错误的信息，这两个属性分别是：javax.servlet.error.status_code 和

javax.servlet.error.message。

可回忆一下，在 web.xml 内以正确的次序声明 web-app 的子元素很重要。这里只要记住，error-page 出现在 web.xml 文件的末尾附近，servlet、servlet-name 和 welcome-file-list 之后即可。

8.1 error-code 元素

为了更好地了解 error-code 元素的值，可考虑一下如果不正确地输入文件名，大多数站点会作出什么反映。这样做一般会出现一个 404 错误信息，它表示不能找到该文件，但几乎没提供更多有用的信息。另一方面，可以试一下在www.microsoft.com、www.ibm.com 处或者特别是在www.bea.com 处输出未知的文件名。这是会得出有用的消息，这些消息提供可选择的位置，以便查找感兴趣的页面。提供这样有用的错误页面对于 Web 应用来说是很有价值得。事实上，http://www.plinko.net/404/ 就是把整个站点专门用于 404 错误页面这个内容。这个站点包含来自全世界最好、最糟和最搞笑的 404 页面。

程序清单 5-13 给出一个 JSP 页面，此页面可返回给提供位置程序名的客户机。程序清单 5-14 给出指定程序清单 5-13 作为返回 404 错误代码时显示的页面的 web.xml。请注意，浏览器中显示的 URL 仍然是客户机所提供的。错误页面是一种后台实现技术。

最后一点，请记住 IE5 的缺省配置显然不符合 HTTP 规范，它忽略了服务器生成的错误消息，而是显示自己的标准出错信息。可转到其 Tools 菜单，选择 Internet Options，单击 Advanced，取消 Show Friendly HTTP Error Message 来解决此问题。

程序清单 5-13 NotFound.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>404: Not Found</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<H2>Error!</H2>
I'm sorry, but I cannot find a page that matches
<%= request.getRequestURI() %> on the system. Maybe you should
try one of the following:
<UL>
  <LI>Go to the server's <A HREF="/">home page</A>.
   <LI>Search for relevant pages.<BR>
        <FORM ACTION="http://www.google.com/search">
        <CENTER>
       Keywords: <INPUT TYPE="TEXT" NAME="q"><BR>
        <INPUT TYPE="SUBMIT" VALUE="Search">
        </CENTER>
        </FORM>
  <LI>Admire a random multiple of 404:
       <%= 404*((int)(1000*Math.random())) %>.
```

```
  <LI>Try a <A HREF="http://www.plinko.net/404/rndindex.asp"
              TARGET="_blank">
       random 404 error message</A>. From the amazing and
       amusing plinko.net <A HREF="http://www.plinko.net/404/">
       404 archive</A>.
</UL>
</BODY></HTML>
```

程序清单 5-14 web.xml（指出 HTTP 错误代码的错误页面的摘录）

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
     "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <error-page>
      <error-code>404</error-code>
      <location>/NotFound.jsp</location>
  </error-page>
  <!-- ... -->
</web-app>
```

8.2 exception-type 元素

error-code 元素处理某个请求产生一个特定的 HTTP 状态代码时的情况。然而，对于 servlet 或 JSP 页面返回 200 但产生运行时异常这种同样是常见的情况怎么办呢？这正是 exception-type 元素要处理的情况。只需提供两样东西即可：即提供如下的一个完全限定的异常类和一个位置：

```
<error-page>
  <exception-type>packageName.className</exception-type>
  <location>/SomeURL</location>
</error-page>
```

这样，如果 Web 应用中的任何 servlet 或 JSP 页面产生一个特定类型的未捕捉到的异常，则使用指定的 URL。此异常类型可以是一个标准类型，如 javax.ServletException 或 java.lang.OutOfMemoryError，或者是一个专门针对你的应用的异常。

例如，程序清单 5-15 给出了一个名为 DumbDeveloperException 的异常类，可用它来特别标记经验较少的程序员（不是说你的开发组中一定有这种人）所犯的错误。这个类还包含一个名为 dangerousComputation 的静态方法，它时不时地生成这种类型的异常。程序清单 5-16 给出对随机整数值调用 dangerousCompution 的一个 JSP 页面。在抛出此异常时，如程序清单 5-18 的 web.xml 版本中所给出的 exception-type 所指出的那样，对客户机显示 DDE.jsp（程

序清单 5-17）。图 5-16 和图 5-17 分别给出幸运和不幸的结果。

程序清单 5-15 DumbDeveloperException.java

```
package moreservlets;

/** Exception used to flag particularly onerous
    programmer blunders. Used to illustrate the
    exception-type web.xml element.
 *  <P>
 *  Taken from More Servlets and JavaServer Pages
 *  from Prentice Hall and Sun Microsystems Press,
 *  http://www.moreservlets.com/.
 *  &copy; 2002 Marty Hall; may be freely used or adapted.
 */

public class DumbDeveloperException extends Exception {
  public DumbDeveloperException() {
    super("Duh. What was I *thinking*?");
  }

  public static int dangerousComputation(int n)
      throws DumbDeveloperException {
    if (n < 5) {
      return(n + 10);
    } else {
      throw(new DumbDeveloperException());
    }
  }
}
```

程序清单 5-16 RiskyPage.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>Risky JSP Page</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<H2>Risky Calculations</H2>
<%@ page import="moreservlets.*" %>
<% int n = ((int)(10 * Math.random())); %>
<UL>
  <LI>n: <%= n %>
  <LI>dangerousComputation(n):
      <%= DumbDeveloperException.dangerousComputation(n) %>
```

```
</UL>
</BODY></HTML>
```

程序清单 5-17 DDE.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD><TITLE>Dumb</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<H2>Dumb Developer</H2>
We're brain dead. Consider using our competitors.
</BODY></HTML>
```

程序清单 5-18 web.xml（为异常指定错误页面的摘录）

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
      PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
      "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
   <!-- ... -->
   <servlet>  … </servlet>
   <!-- ... -->
   <error-page>
         <exception-type>
            moreservlets.DumbDeveloperException
         </exception-type>
         <location>/DDE.jsp</location>
   </error-page>
   <!-- ... -->
</web-app>
```

9 提供安全性

利用 web.xml 中的相关元素为服务器的内建功能提供安全性。

9.1 指定验证的方法

使用 login-confgi 元素规定服务器应该怎样验证试图访问受保护页面的用户。它包含三个可能的子元素，分别是：auth-method、realm-name 和 form-login-config。login-config 元素

应该出现在 web.xml 部署描述符文件的结尾附近，紧跟在 security-constraint 元素之后。

- auth-method

login-config 的这个子元素列出服务器将要使用的特定验证机制。有效值为 BASIC、DIGEST、FORM 和 CLIENT-CERT。服务器只需要支持 BASIC 和 FORM。

BASIC 指出应该使用标准的 HTTP 验证，在此验证中服务器检查 Authorization 头。如果缺少这个头则返回一个 401 状态代码和一个 WWW-Authenticate 头。这导致客户机弹出一个用来填写 Authorization 头的对话框。此机制很少或不提供对攻击者的防范，这些攻击者在 Internet 连接上进行窥探（如通过在客户机的子网上执行一个信息包探测装置），因为用户名和口令是用简单的可逆 base64 编码发送的，他们很容易得手。所有兼容的服务器都需要支持 BASIC 验证。

DIGEST 指出客户机应该利用加密 Digest Authentication 形式传输用户名和口令。这提供了比 BASIC 验证更高的防范网络截取得的安全性，但这种加密比 SSL（HTTPS）所用的方法更容易破解。不过，此结论有时没有意义，因为当前很少有浏览器支持 Digest Authentication，所以 servlet 容器不需要支持它。

FORM 指出服务器应该检查保留的会话 cookie 并且把不具有它的用户重定向到一个指定的登陆页。此登陆页应该包含一个收集用户名和口令的常规 HTML 表单。在登陆之后，利用保留会话级的 cookie 跟踪用户。虽然很复杂，但 FORM 验证防范网络窥探并不比 BASIC 验证更安全，如果有必要可以在顶层安排诸如 SSL 或网络层安全（如 IPSEC 或 VPN）等额外的保护。所有兼容的服务器都需要支持 FORM 验证。

CLIENT-CERT 规定服务器必须使用 HTTPS（SSL 之上的 HTTP）并利用用户的公开密钥证书（Pulic Key Certificat）对用户进行验证。这提供了防范网络截取的很强的安全性，但只有兼容 J2EE 的服务器需要支持它。

- realm-name

此元素只在 auth-method 为 BASIC 时使用。它指出浏览器在相应对话框标题使用的、并作为 Authorization 头组成部分的安全域的名称。

- form-login-config

此元素只在 auth-method 为 FORM 时适用。它指定两个页面，分别是：包含收集用户名及口令的 HTML 表单的页面（利用 form-login-page 子元素），用来指示验证失败的页面（利用 form-error-page 子元素）。由 form-login-page 给出的 HTML 表单必须具有一个 j_security_check 的 ACTION 属性、一个名为 j_username 的用户名文本字段以及一个名为 j_password 的口令字段。

例如，程序清单 5-19 指示服务器使用基于表单的验证。Web 应用的顶层目录中的一个名为 login.jsp 的页面将收集用户名和口令，并且失败的登陆将由相同目录中名为 login-error.jsp 的页面报告。

程序清单 5-19 web.xml（说明 login-config 的摘录）

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
     "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
```

```
  <!-- ... -->
  <security-constraint> ... </security-constraint>
  <login-config>
    <auth-method> FORM </auth-method>
    <form-login-config>
      <form-login-page>/login.jsp</form-login-page>
      <form-error-page>/login-error.jsp</form-error-page>
    </form-login-config>
  </login-config>
  <!-- ... -->
</web-app>
```

9.2 限制对 Web 资源的访问

现在，可以指示服务器使用何种验证方法了。“了不起，”你说道，“除非我能指定一个来收到保护的 URL，否则没有多大用处。”没错。指出这些 URL 并说明他们应该得到何种保护正是 security-constriaint 元素的用途。此元素在 web.xml 中应该出现在 login-config 的紧前面。它包含是个可能的子元素，分别是：web-resource-collection、 auth-constraint、user-data-constraint 和 display-name。下面各小节对它们进行介绍。

- web-resource-collection

此元素确定应该保护的资源。所有 security-constraint 元素都必须包含至少一个 web-resource-collection 项。此元素由一个给出任意标识名称的 web-resource-name 元素、一个确定应该保护的 URL 的 url-pattern 元素、一个指出此保护所适用的 HTTP 命令（GET、POST 等,缺省为所有方法）的 http-method 元素和一个提供资料的可选 description 元素组成。例如，下面的 Web-resource-collection 项（在 security-constratint 元素内）指出 Web 应用的 proprietary 目录中所有文档应该受到保护。

```
<security-constraint>
  <web-resource-coolection>
    <web-resource-name>Proprietary</web-resource-name>
    <url-pattern>/propritary/*</url-pattern>
  </web-resource-coolection>
<!-- ... -->
</security-constraint>
```

重要的是应该注意到，url-pattern 仅适用于直接访问这些资源的客户机。特别是，它不适合于通过 MVC 体系结构利用 RequestDispatcher 来访问的页面，或者不适合于利用类似 jsp:forward 的手段来访问的页面。这种不匀称如果利用得当的话很有好处。例如，servlet 可利用 MVC 体系结构查找数据，把它放到 bean 中，发送请求到从 bean 中提取数据的 JSP 页面并显示它。我们希望保证决不直接访问受保护的 JSP 页面，而只是通过建立该页面将使用的 bean 的 servlet 来访问它。url-pattern 和 auth-contraint 元素可通过声明不允许任何用户直接访问 JSP 页面来提供这种保证。但是，这种不匀称的行为可能让开发人员放松警惕，使他

们偶然对应受保护的资源提供不受限制的访问。

- auth-constraint

尽管 web-resource-collention 元素质出了哪些 URL 应该受到保护，但是 auth-constraint 元素却指出哪些用户应该具有受保护资源的访问权。此元素应该包含一个或多个标识具有访问权限的用户类别 role-name 元素，以及包含（可选）一个描述角色的 description 元素。例如，下面 web.xml 中的 security-constraint 元素部门规定只有指定为 Administrator 或 Big Kahuna（或两者）的用户具有指定资源的访问权。

```
<security-constraint>
  <web-resource-coolection> ... </web-resource-coolection>
  <auth-constraint>
    <role-name>administrator</role-name>
    <role-name>kahuna</role-name>
  </auth-constraint>
</security-constraint>
```

重要的是认识到，到此为止，这个过程的可移植部分结束了。服务器怎样确定哪些用户处于任何角色以及它怎样存放用户的口令，完全有赖于具体的系统。

例如，Tomcat 使用 install_dir/conf/tomcat-users.xml 将用户名与角色名和口令相关联，正如下面例子中所示，它指出用户 joe（口令 bigshot）和 jane（口令 enaj）属于 administrator 和 kahuna 角色。

```
<tomcat-users>
 <user name="joe" password="bigshot" roles="administrator,kahuna" />
 <user name="jane" password="enaj" roles="kahuna" />
</tomcat-users>
```

- user-data-constraint

这个可选的元素指出在访问相关资源时使用任何传输层保护。它必须包含一个 transport-guarantee 子元素（合法值为 NONE、INTEGRAL 或 CONFIDENTIAL），并且可选地包含一个 description 元素。transport-guarantee 为 NONE 值将对所用的通讯协议不加限制。INTEGRAL 值表示数据必须以一种防止截取它的人阅读它的方式传送。虽然原理上（并且在未来的 HTTP 版本中），在 INTEGRAL 和 CONFIDENTIAL 之间可能会有差别，但在当前实践中，他们都只是简单地要求用 SSL。例如，下面指示服务器只允许对相关资源做 HTTPS 连接：

```
<security-constraint>
<!-- ... -->
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

- display-name

security-constraint 的这个很少使用的子元素给予可能由 GUI 工具使用的安全约束项一个名称。

9.3 分配角色名

迄今为止，讨论已经集中到完全由容器（服务器）处理的安全问题之上了。但 servlet 以及 JSP 页面也能够处理它们自己的安全问题。

例如，容器可能允许用户从 bigwig 或 bigcheese 角色访问一个显示主管人员额外紧贴的页面，但只允许 bigwig 用户修改此页面的参数。完成这种更细致的控制的一种常见方法是调用 HttpServletRequset 的 isUserInRole 方法，并据此修改访问。

Servlet 的 security-role-ref 子元素提供出现在服务器专用口令文件中的安全角色名的一个别名。例如，假如编写了一个调用 request.isUserInRole（" boss "）的 servlet，但后来该 servlet 被用在了一个其口令文件调用角色 manager 而不是 boss 的服务器中。下面的程序段使该 servlet 能够使用这两个名称中的任何一个。

```
<servlet>
<!-- ... -->
  <security-role-ref>
    <role-name>boss</role-name>          <!-- New alias -->
    <role-link>manager</role-link>      <!-- Real name -->
  </security-role-ref>
</servlet>
```

也可以在 web-app 内利用 security-role 元素提供将出现在 role-name 元素中的所有安全角色的一个全局列表。分别地生命角色使高级 IDE 容易处理安全信息。

10 控制会话超时

如果某个会话在一定的时间内未被访问，服务器可把它扔掉以节约内存。可利用 HttpSession 的 setMaxInactiveInterval 方法直接设置个别会话对象的超时值。如果不采用这种方法，则缺省的超时值由具体的服务器决定。但可利用 session-config 和 session-timeout 元素来给出一个适用于所有服务器的明确的超时值。超时值的单位为分钟，因此，下面的例子设置缺省会话超时值为三个小时（180 分钟）。

```
<session-config>
  <session-timeout>180</session-timeout>
</session-config>
```

11 Web 应用的文档化

越来越多的开发环境开始提供 servlet 和 JSP 的直接支持。例子有 Borland Jbuilder Enterprise Edition、Macromedia UltraDev、Allaire JRun Studio（写此文时，已被 Macromedia 收购）以及 IBM VisuaAge for Java 等。

大量的 web.xml 元素不仅是为服务器设计的，而且还是为可视开发环境设计的。它们包

括 icon、display-name 和 discription 等。

可回忆一下，在 web.xml 内以适当地次序声明 web-app 子元素很重要。不过，这里只要记住 icon、display-name 和 description 是 web.xml 的 web-app 元素内的前三个合法元素即可。

- icon

icon 元素指出 GUI 工具可用来代表 Web 应用的一个和两个图像文件。可利用 small-icon 元素指定一幅 16 x 16 的 GIF 或 JPEG 图像，用 large-icon 元素指定一幅 32 x 32 的图像。下面举一个例子：

```
<icon>
  <small-icon>/images/small-book.gif</small-icon>
  <large-icon>/images/tome.jpg</large-icon>
</icon>
```

- display-name

display-name 元素提供 GUI 工具可能会用来标记此 Web 应用的一个名称。下面是个例子。

```
<display-name>Rare Books</display-name>
```

- description

description 元素提供解释性文本，如下所示：

```
<description>
This Web application represents the store developed for
rare-books.com, an online bookstore specializing in rare
and limited-edition books.
</description>
```

12 关联文件与 MIME 类型

服务器一般都具有一种让 Web 站点管理员将文件扩展名与媒体相关联的方法。例如，将会自动给予名为 mom.jpg 的文件一个 image/jpeg 的 MIME 类型。但是，假如你的 Web 应用具有几个不寻常的文件，你希望保证它们在发送到客户机时分配为某种 MIME 类型。mime-mapping 元素（具有 extension 和 mime-type 子元素）可提供这种保证。例如，下面的代码指示服务器将 application/x-fubar 的 MIME 类型分配给所有以.foo 结尾的文件。

```
<mime-mapping>
  <extension>foo</extension>
  <mime-type>application/x-fubar</mime-type>
</mime-mapping>
```

或许，你的 Web 应用希望重载（override）标准的映射。例如，下面的代码将告诉服务器在发送到客户机时指定.ps 文件作为纯文本（text/plain）而不是作为 PostScript（application/postscript）。

```
<mime-mapping>
  <extension>ps</extension>
  <mime-type>application/postscript</mime-type>
```

```
</mime-mapping>
```

关于 MIME 类型的更多信息，请参阅网址

13 定位 TLD

JSP taglib 元素具有一个必要的 uri 属性，它给出一个 TLD（Tag Library Descriptor）文件相对于 Web 应用的根的位置。TLD 文件的实际名称在发布新的标签库版本时可能会改变，但我们希望避免更改所有现有 JSP 页面。此外，可能还希望使用保持 taglib 元素的简练性的一个简短的 uri。这就是部署描述符文件的 taglib 元素派用场的所在了。Taglib 包含两个子元素：taglib-uri 和 taglib-location。taglib-uri 元素应该与用于 JSP taglib 元素的 uri 属性的东西相匹配。Taglib-location 元素给出 TLD 文件的实际位置。例如，假如你将文件 chart-tags-1.3beta.tld 放在 WebApp/WEB-INF/tlds 中。现在，假如 web.xml 在 web-app 元素内包含下列内容。

```
<taglib>
  <taglib-uri>/charts.tld</taglib-uri>
  <taglib-location>
    /WEB-INF/tlds/chart-tags-1.3beta.tld
  </taglib-location>
</taglib>
```

给出这个说明后，JSP 页面可通过下面的简化形式使用标签库。

```
<%@ taglib uri="/charts.tld" prefix="somePrefix" %>
```

14 指定应用事件监听程序

应用事件监听器程序是建立或修改 servlet 环境或会话对象时通知的类。它们是 servlet 规范的版本 2.3 中的新内容。这里只简单地说明用来向 Web 应用注册一个监听程序的 web.xml 的用法。

注册一个监听程序涉及在 web.xml 的 web-app 元素内放置一个 listener 元素。在 listener 元素内，listener-class 元素列出监听程序的完整的限定类名，如下所示：

```
<listener>
  <listener-class>package.ListenerClass</listener-class>
</listener>
```

虽然 listener 元素的结构很简单，但请不要忘记，必须正确地给出 web-app 元素内的子元素的次序。listener 元素位于所有的 servlet 元素之前以及所有 filter-mapping 元素之后。此外，因为应用生存期监听程序是 serlvet 规范的 2.3 版本中的新内容，所以必须使用 web.xml DTD 的 2.3 版本，而不是 2.2 版本。

例如，程序清单 5-20 给出一个名为 ContextReporter 的简单的监听程序，只要 Web 应用的 Servlet-Context 建立（如装载 Web 应用）或消除（如服务器关闭）时，它就在标准输出上显示一条消息。程序清单 5-21 给出此监听程序注册所需要的 web.xml 文件的一部分。

程序清单 5-20 ContextReporterjava

```
package moreservlets;

import javax.servlet.*;
import java.util.*;

/** Simple listener that prints a report on the standard output
 *  when the ServletContext is created or destroyed.
 *  <P>
 *  Taken from More Servlets and JavaServer Pages
 *  from Prentice Hall and Sun Microsystems Press,
 *  http://www.moreservlets.com/.
 *  &copy; 2002 Marty Hall; may be freely used or adapted.
 */

public class ContextReporter implements ServletContextListener {
  public void contextInitialized(ServletContextEvent event) {
    System.out.println("Context created on " +
                       new Date() + ".");
  }

  public void contextDestroyed(ServletContextEvent event) {
    System.out.println("Context destroyed on " +
                       new Date() + ".");
  }
}
```

程序清单 5-21 web.xml（声明一个监听程序的摘录）

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
      PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
       "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
  <!-- ... -->
  <filter-mapping> … </filter-mapping>
  <listener>
    <listener-class>package.ListenerClass</listener-class>
  </listener>
 <servlet> ... </servlet>
  <!-- ... -->
```

```
</web-app>
```

15 J2EE 元素

本节描述用作 J2EE 环境组成部分的 Web 应用的 web.xml 元素。这里将提供一个简明的介绍，详细内容可以参阅 http://java.sun.com/j2ee/j2ee-1_3-fr-spec.pdf 的 Java 2 Plantform Enterprise Edition 版本 1.3 规范的第 5 章。

- distributable

distributable 元素指出，Web 应用是以这样的方式编程的：即，支持集群的服务器可安全地在多个服务器上分布 Web 应用。例如，一个可分布的应用必须只使用 Serializable 对象作为其 HttpSession 对象的属性，而且必须避免用实例变量（字段）来实现持续性。distributable 元素直接出现在 discription 元素之后，并且不包含子元素或数据，它只是一个如下的标志。

```
<distributable />
```

- resource-env-ref

resource-env-ref 元素声明一个与某个资源有关的管理对象。此元素由一个可选的 description 元素、一个 resource-env-ref-name 元素（一个相对于 java:comp/env 环境的 JNDI 名）以及一个 resource-env-type 元素（指定资源类型的完全限定的类），如下所示：

```
<resource-env-ref>
  <resource-env-ref-name>
    jms/StockQueue
  </resource-env-ref-name>
  <resource-env-ref-type>
    javax.jms.Queue
  </resource-env-ref-type>
</resource-env-ref>
```

- env-entry

env-entry 元素声明 Web 应用的环境项。它由一个可选的 description 元素、一个 env-entry-name 元素（一个相对于 java:comp/env 环境 JNDI 名）、一个 env-entry-value 元素（项值）以及一个 env-entry-type 元素（java.lang 程序包中一个类型的完全限定类名，java.lang.Boolean、java.lang.String 等）组成。下面是一个例子：

```
<env-entry>
  <env-entry-name>minAmout</env-entry-name>
  <env-entry-value>100.00</env-entry-value>
  <env-entry-type>minAmout</env-entry-type>
</env-entry>
```

- ejb-ref

ejb-ref 元素声明对一个 EJB 的主目录的应用。它由一个可选的 description 元素、一个 ejb-ref-name 元素（相对于 java:comp/env 的 EJB 应用）、一个 ejb-ref-type 元素（bean 的类型，Entity 或 Session）、一个 home 元素（bean 的主目录接口的完全限定名）、一个 remote 元素

（bean 的远程接口的完全限定名）以及一个可选的 ejb-link 元素（当前 bean 链接的另一个 bean 的名称）组成。

- ejb-local-ref

ejb-local-ref 元素声明一个 EJB 的本地主目录的引用。除了用 local-home 代替 home 外，此元素具有与 ejb-ref 元素相同的属性并以相同的方式使用。

题外话：

制作此文档只是为了帮助网友学习和认识 web.xml 在 Web 应用中的作用，并没有任何赢利的意图。如果本书中文版出版社机械工业出版社或本书译者钟鸣、石永平对此行为表示反对，可以留言，我将删掉此贴。
贴子地址是：http://www.chinajavaworld.net/forum/topic.cgi?forum=43&topic=1188