Web 应用程序及结构

Web 应用程序

Web 应用程序是一种可以通过 Web 访问的应用程序。一个 Web 应用程序是由完成特定任务的各种 Web 组件构成的并通过 Web 将服务展示给外界。在实际应用中，Web 应用程序是由多个 Servlet、JSP页面、HTML 文件以及图像文件等组成。

Web 应用程序的结构

一个 Web 应用程序的所有资源被保存在一个结构化的目录中，目录结构是按照资源和文件的位置严格定义的。

Tomcat 安装目录的 webapps 目录是所有 Web 应用程序的根目录。任何新建大的 Web 应用程序都应该在该文件夹下创建一个新的文件。

1. 文档根目录

每个 Web 应用程序都有一个文档根目录，它是应用程序所在的目录。应用程序可以公开访问的文件都应该放在该目录或其子目录中。通常我们把该目录中的文件组织在多个子目录中。例如 HTML 存放在 html 文件中目录中，JSP 放在 jsp 文件目录中，这样方便对 Web 应用程序的文件进行管理。

2. WEB-INF 目录

每个 Web 应用程序在它的根目录中都必须有一个 WEB-INF 目录。该目录供存放供服务器访问的资源。尽管该目录物理上位于文档根目录中，但不将它看做文档根目录的一部分，也就是说，在 WEB-INF 目录中的文件不为客户服务。

- classes 目录

  classes 目录存放支持该 Web 应用程序的类文件，如 Servlet 类文件、JavaBeans 类文件等。在运行时，容器自动将该目录添加到 Web 应用程序的类路径中。

- lib 目录

  该目录存放 Web 应用程序使用的全部 JAR 文件，包括第三方的 JAR 文件。例如 JDBC 驱动程序 JAR 文件应该存放在该目录中。也可以把应用程序所用到的类文件打包成 JAR 文件存放到该目录中。在运行时，容器自动将该目录中的所有 JAR 文件添加到 Web 应用程序的类路径中。

- web.xml 文件

  每个 Web 应用程序都必须有一个 web.xml 文件。它包含 Servlet 容器运行 Web 应用程序所需要的信息，例如 Servlet 声明、映射、属性、授权及安全限制等。

  </>

3. Web 归档文件

一个 Web 应用程序包含许多文件，可以将这些文件打包成一个扩展名为 `.war` 的 Web 归档文件中，一般称为 WAR 文件。WAR 文件主要是为了方便 Web 应用程序在不同系统之间的移植。

创建一个 WAR 文件可通过命令 `jar -cvf 名称.war *` 实现。

4. 默认的 Web 应用程序

除用户创建的 Web 应用程序外，Tomcat 服务器还有一个默认的 Web 应用程序。`\webapps\ROOT` 目录被设置为默认的 Web 应用程序的文档根目录。它与其它的 Web 应用程序类似，只不过访问它的资源不需要制定应用程序的名称或上下文路径。

部署描述文件

Web 应用程序中包含多种组件，有些组件可以使用注解配置，有些组件需要使用部署描述文件配置。部署描述文件（Deployment Descriptor,DD）可用来初始化 Web 应用程序的组件。Web 容器在启动时读取该文件，对应用程序配置，所以有时也将该文件称为配置文件。

DD 文件是一个 XML 文件。与所有的 XML 文件一样，该文件的第一行是声明，通过 version 属性和 encoding 属性指定 XML 的版本所使用的字符集。例如：`<?xml version="1.0" encoding="ISO-8859-1">`。

下面所有元素都包含在 `<web-app>...</web-app>` 元素中，它是 DD 文件的根元素，其他所有元素都应该在这对元素内部声明。在 `<web-app>` 元素中指定了 5 个属性。`xmlns` 属性声明了 w eb.xml 文件命名空间的 XML 方案文档的位置；`xmlns:xsi` 属性指定了命名空间的实例；`xsi:schemaLocation` 属性指定了方案的位置；`version` 指定了方案的版本；`metadata-complete` 指定是否可在源程序中使用注解，true 表示注解无效。

DD 文件的定义

为了保证跨 Web 容器的可移植性，部署描述文件的文档类型定义（Document Type Definition,DTD） 的标志规定了文档的语法和标签的规则，这些规则包括一系列的元素和实体声明。下面列出了 `<web-app>` 元素的 DTD 定义，仅给出常用元素。`<!ELEMENT web-app(description?,display-name?,icon?,distributable?,context-param*,filter*,filter-mapping*,listener*,servlet*,servlet-mapping*,session-config?,mime-mapping*,welcome-file-list?,error-page*,jsp-config*,security-constraint*,login-config?,security-role*)>`

在 DTD 中，带问号（?）的元素可以出现 0 次或一次，带星号（*） 的元素可以出现 0 次或多次，带加号（+）的元素可以出现一次或多次，不带符号的元素只能出现一次。

常用元素的含义：

元素名	说明
-	
description	对应用程序的简短描述
display-name	定义应用程序的显示名称
context-param	定义应用程序的初始化参数
servlet	定义 Servlet
servlet-mapping	定义 Servlet 映射
w elcome-file-list	定义应用程序的欢迎文件
session-config	定义会话时间
listener	定义监听器类
filter	定义过滤器
filter-mapping	定义过滤器映射
error-page	定义错误处理页面
security-constraint	定义 Web 应用程序的安全约束
mime-mapping	定义常用文件扩展名的 MIME 类型

`<servlet>` 元素

`<servlet>` 元素为 Web 应用程序定义一个 Servlet，该元素的 DTD 定义如下：`<!ELEMENT servlet(description?,icon?display-name?,servlet-name,(servlet-class|jsp-file),init-param*,load-on-startup?,security-role-ref*)>`

1. `<servlet-name>` 元素

该元素用来定义 Servlet 名称，该元素是必选项。定义的名称在 DD 文件中应该唯一。可以通过 ServletConfig 的 `getServletName()` 检索 Servlet 名。

2. `<servlet-class>` 元素

该元素指定 Servlet 类的完整名称，即需要带包的名称，例如：`com.demo.helloservlet` 。容器将使用该类创建 Servlet 实例。Servlet 类以及它所依赖的所有类都应该在 Web 应用程序的路径中。WEB-INF 目录中的 classes 目录和 lib 目录中的 JAR 被自动添加到容器的类路径中，因此如果把类放到这两个地方就不需要设置类路径。也可以使用 `` 元素指定一个 JSP 文件代替 `` 元素。

3. `<init-param>` 元素

该元素定义向 Servlet 传递的初始化参数。在一个 `` 元素中可以定义任意多个 `` 元素。每个 `` 元素必须有且仅有一组 `` 和 `` 子元素。`` 定义参数名，`` 定义参数值。Servlet 可以通过 ServletConfig 接口的 `getInitParameter()` 检索初始化参数。

4. `<load-on-startup>元素`

一般情况下，Servlet 是在被请求时由容器装入内存，也可以使 Servlet 在容器启动时就装入内存。`` 元素指定是否在 Web 应用程序启动时载入该 Servlet。该元素的值是一个整数。如果没有指定该元素或其内容为一个负数，容器将根据需要决定何时装入 Servlet。如果其内容为一个正数，则在 Web 应用程序启动时载入该 Servlet。对不同的 Servlet，可以指定不同的值，这可以控制容器装入这些 Servlet 的顺序，值小的先装入。

`<servlet-mapping>` 元素

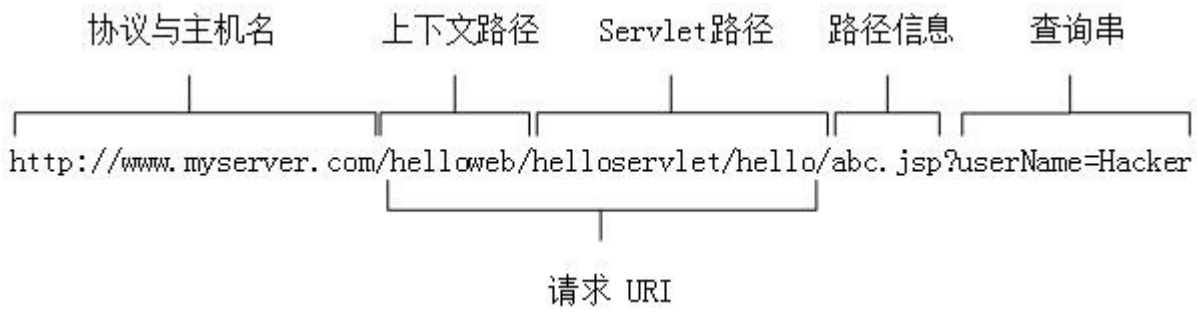
`<servlet-mapping>` 元素定义了一个映射，它指定哪个 URL 模式被该 Servlet 处理。容器使用这些映射根据实际的 URL 访问合适的 Servlet。`<servlet-mapping>` 元素的定义是：`<!ELEMENT servlet-mapping(servlet-name,url-pattern)>` 。`<servlet-name>` 元素应该是使用 `<servlet>` 元素定义的 Servlet 名，而 `<url-pattern>` 可以包含要与该 Servlet 关联的模式字符串。例如：

```
<servlet-mapping>
<servlet-name>helloservlet</servlet-name>
<url-pattern>/helloservlet/hello/*</url-pattern>
</servlet-mapping>
```

对于上面的映射定义，如果一个请求 URL 串与 "/helloservlet/hello/*" 匹配，容器将使用名为 helloservlet 的 servlet 为用户提供服务。例如：`http://www.myserver.com/helloweb/helloservlet/hello/abc.jsp`

1. URL 的组成

一个请求的 URL 可以由多个部分组成，如下图:



URL 第一部分包括协议、主机名和可选的端口号，第二部分是请求 URI，第三部分是查询串。

请求 URI 的内容可以使用 HttpServletRequest 的 `getRequestURI()` 得到，查询串的内容可以使用 `getQuerString()` 得到。一个请求 URI 又由三部分组成：上下文路径(context path)、Servlet 路径(servlet path)和路径信息(path info)。

- 上下文路径：对于上面的 URI，/helloservlet 为上下文路径。
- Servlet 路径：对于上面的 URI，/helloservlet/hello 为 Servlet 路径。
- 路径信息：它实际上是额外的路径信息，对于上面的 URI，路径信息为/abc.jsp。

要想获得上述三种路径信息，可以使用请求对象的 `getContextPath()` 、`getServletPath()` 、`getPathInfo()`

2. `<url-pattern>` 的三种形式

在 `` 中指定 URL 映射可以有三种形式。

1. 目录匹配

   以斜杠 “/” 开头，以 “/” 结尾的形式。例如，下面的映射将把任何在 Servlet 路径中以 `/helloServlet/hello/` 字符串开头的请求都发送到 `helloServlet`.

   ```
   <servlet-mapping>
     <servlet-name>helloServlet</srvlet-name>
     <url-pattern>/helloServlet/hello/\*<url-pattern>
   </servlet-mapping>
   ```

2. 扩展名匹配

   以星号 “*” 开始，后接一个扩展名（如 *.do 或 *.pdf 等）。例如，下面的映射将把所有以 .pdf 结尾的请求发送到 pdfGeneratorServlet。

   ```
   <servlet-mapping>
      <servlet-name>pdfGeneratorServlet<servlet-name>
      <url-pattern>\*.pdf<url-pattern>
   </servlet-mapping>
   ```

3. 精准匹配

   所有其他字符串都作为精准匹配。例如下面的映射：

   ```
   <servlet-mapping>
     <servlet-ame>reportServlet<servlet-name>
     <url-pattern>/report<url-pattern>
   </servlet-mapping>
   ```

   容器将把 http://w w w.myserver.com/hellow eb/report 请求送给 reportServlet。然而，并不把请求 http:w w w.myserver.com/hellow eb/report/sales 发送给 reportServlet。

3. 容器如何解析 URL

当容器接收到一个 URL 请求，它要解析该 URL，找到与该 URL 匹配的资源为用户提供服务。假设一个请求的 URL 为：`http://w w w.myserver/hellow eb/helloServlet/hello/abc.jsp`

下面说明容器如何解析该 URL，并将请求发送到匹配的 Servlet:

1. 当容器接收到该请求 URL 后，它首先解析出 URI。然后从中取出第一部分作为上下文路径，这里是 /hellow eb，接下来在容器中查找是否有名称为 hellow eb 的 Web 应用程序。
2. 如果没有名为 hellow eb 的 Web 应用程序，则上下文路径为空，请求将发送到默认的 Web 应用程序（路径名为 ROOT）。
3. 如果有名为 hellow eb 的应用程序，则继续解析下一部分。容器尝试将 Servlet 路径与 Servlet 映射匹配，如果找到一个匹配，则完整的 URI 请求（上下文部分除外）就是 Servlet 路径，在这种情况下，路径信息为 null。
4. 容器沿着请求 URI 路径树向下，每次一层目录，使用 “/” 作为路径分隔符，反复尝试最长的路径，看是否与一个 Servlet 匹配。如果有一个匹配。请求 URI 的匹配部分就是 Servlet 路径，剩余部分是路径信息。
5. 如果不能找到匹配的资源，容器将向客户发送一个 404 错误消息。

`<welcome-file-list>` 元素

通常在浏览器的地址栏中输入一个路径名称而没有指定特定文件，也能访问到一个页面，这个页面就是欢迎页面，文件名通常为index.html。

在 Tomcat 中，如果访问的 URL 是目录，并且没有特定的 Servlet 与这个 URL 模式匹配。那么它将在该目录中首先查找 index.html 文件，如果找不到将查找 index.jsp 文件，如果找到上述文件，将该文件返回给用户。如果找不到（包括目录也找不到），将向客户发送 404 错误消息。

假设有一个 Web 应用程序，它的默认的欢迎页面是 index.html，还有一些目录都有自己的欢迎页面，如 default.jsp。可以在 DD 文件 `<web-app>` 元素中使用 `<welcome-file-list>` 元素指定欢迎页面的查找列表，如下所示：

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