web.xml文件详解

# 引入

问题1：web工程中的**web.xml文件**有什么作用呢？它是每个web.xml工程都必须的吗？

回答：一个web中完全可以没有**web.xml文件**，也就是说，web.xml文件并不是web工程必须的。

问题2：那什么时候需要，什么时候可以不需要呢？

回答：要想回答上面的问题，得先了解**web.xml文件**使用来干什么的。**web.xml文件是用来配置：欢迎页、servlet、filter等的**。当你的web工程没用到这些时，你可以不用web.xml文件来配置你的web工程。

# web.xml的模式(Schema)文件

问题3：那么**web.xml**能做的所有事情都有那些？

回答：其实，**web.xml的模式(Schema)文件中定义了多少种标签元素，web.xml中就可以出现它的模式文件所定义的标签元素，它就能拥有定义出来的那些功能**。**web.xml的模式文件是由Sun公司定义的，每个web.xml文件的根元素<web-app>中，都必须标明这个web.xml使用的是哪个模式文件**。如：

<?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">

</web-app>

而且**web.xml的模式文件中定义的标签**并不是定死的，模式文件也是可以改变的，一般来说，随着web.mxl模式文件的版本升级，里面定义的功能会越来越复杂，也即标签元素的种类会越来越多，但有些是不常用的，我们只需记住一些常用的就可以了。

# web.xml的常用标签

下面列出web.xml常用的标签元素及这些标签元素的功能：

### 指定欢迎页面<welcome-file-list>；

### **配置servlet：<servlet>与<servlet-mapping>必须一起出现；(最重要)**

### 指定错误处理页面：<error-page>；

### 设置过滤器：<filter>；

### 设置监听器：<listener>；

### 设置会话过期时间：<session-config>；

### 等等。

## 指定欢迎页面，例如：

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

<welcome-file>index1.jsp</welcome-file>

</welcome-file-list>

上面的例子指定了2个欢迎页面，显示时按顺序从第一个找起，如果第一个存在，就显示第一个，后面的不起作用。如果第一个不存在，就找第二个，以此类推。

关于欢迎页面：

访问一个网站时，默认看到的第一个页面就叫**欢迎页**，一般情况下是由首页来充当欢迎页的。一般情况下，我们会在web.xml中指定欢迎页。但web.xml并不是一个Web的必要文件，没有web.xml，网站仍然是可以正常工作的。只不过网站的功能复杂起来后，web.xml的确有非常大用处，所以，**默认创建的动态web工程在WEB-INF文件夹下面都有一个web.xml文件**。

对于tomcat来说，当你只指定一个web的根名，没有指定具体页面，去访问时一个web时，如果web.xml文件中配置了欢迎页，那么就返回指定的那个页面作为欢迎页，而在文中没有web.xml文件，或虽然有web.xml，但web.xml也没指定欢迎页的情况下，它默认先查找index.html文件，如果找到了，就把index.html作为欢迎页还回给浏览器。如果没找到index.html，tomcat就去找index.jsp。找到index.jsp就把它作为欢迎页面返回。而如果index.html和index.jsp都没找到，又没有用web.xml文件指定欢迎页面，那此时tomcat就不知道该返回哪个文件了，它就显示The requested resource (/XXX) is not available的页面。其中XXX表示web的根名。但如果你指定了具体页面，是可以正常访问的。

## 配置Servlet：对Servlet命名、初始化、定制URL。

### 简单使用

我们可以为Servlet和JSP文件命名并定制URL,其中定制URL是依赖一命名的，命名必须在定制URL前。下面拿servlet来举例：

为Servlet命名：

<servlet>

<servlet-name>servlet1</servlet-name>

<servlet-class>net.test.TestServlet</servlet-class>

</servlet>

为Servlet定制URL

<servlet-mapping>

<servlet-name>servlet1</servlet-name>

<url-pattern>\*.do</url-pattern>

</servlet-mapping>

注意：<servlet>与<servlet-mapping>必须是成对出现的。

### 为Servlet定制初始化参数：

可以定制**servlet、JSP、Context**的初始化参数，然后可以再servlet、JSP、Context中获取这些参数值。下面哪servlet来举例：

<servlet>

<servlet-name>servlet1</servlet-name>

<servlet-class>net.test.TestServlet</servlet-class>

**<init-param>**

<param-name>userName</param-name>

<param-value>Tommy</param-value>

</init-param>

**<init-param>**

<param-name>E-mail</param-name>

<param-value>Tommy@163.com</param-value>

</init-param>

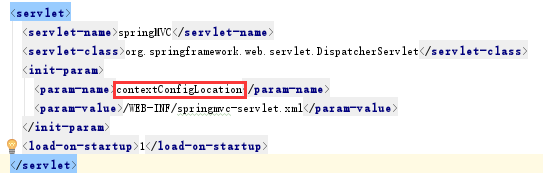
</servlet>

经过上面的配置，在servlet中能够调用**getServletConfig().getInitParameter("param1")**获得参数名对应的值。

### SpringMVC的DispatcherServlet配置

为DispatcherServlet命名并指定contextConfigLocation：名字可以任意起一般取springmvc或springMVC即可。

参数只有contextConfigLocation，该名字不能改变，因为SpringMVC源码中默认通过**getServletConfig().getInitParameter("**contextConfigLocation**")获取到springMVC配置文件的位置。**



<servlet>

<servlet-name>**springMVC**</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

**<init-param>**

<param-name>**contextConfigLocation**</param-name>

<param-value>/WEB-INF/springmvc-servlet.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

为Servlet定制URL：

<servlet-mapping>

<servlet-name>**springMVC**</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

## 指定错误处理页面，

可以通过**“异常类型”或“错误码”**来指定错误处理页面。

<error-page>

<error-code>404</error-code>

<location>/error404.jsp</location>

</error-page>

-----------------------------

<error-page>

<exception-type>java.lang.Exception<exception-type>

<location>/exception.jsp<location>

</error-page>

## 设置过滤器：

比如设置一个编码过滤器，过滤所有资源。与servlet类似，filter与filter-mapping成对出现。

<filter>

<filter-name>XXXCharaSetFilter</filter-name>

<filter-class>net.test.CharSetFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>XXXCharaSetFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

## 设置监听器：

<listener>

<listener-class>net.test.XXXLisenet</listener-class>

</listener>

**spring提供的监听器**：

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

## 设置会话(Session)过期时间

其中时间以分钟为单位，假如设置60分钟超时：

<session-config>

<session-timeout>60</session-timeout>

</session-config>

## 其他标签

除了这些标签元素之外，还可以往web.xml中添加那些标签元素呢，那些标签元素都能起什么作用呢？我们只要去查看**web.xml的模式文件**就能知道。直接看模式文件看不懂，可以找一些中文教程来看看。

# <context-param>的作用

存在着这样一种配置节：**context-param**，它用于**向 ServletContext 提供键值对**，即**应用程序上下文信息**。我们的 listener, filter 等在初始化时会用到这些上下文中的信息，那么 context-param 配置节是不是应该写在 listener 配置节前呢？**实际上context-param 配置节可写在任意位置，因此真正的加载顺序为：context-param ->listener -> filter -> servlet**。

如在配置SpringMVC的时候需要加载SpringMVC的特有的xml配置文件，可以在**context-param**中定义，这样在servlet==>**init-param**中就不需要定义了。



# web.xml的加载顺序

**web.xml 的加载顺序是：context-param -> listener -> filter-> servlet**

1、启动一个WEB项目的时候，WEB容器会去读取它的配置文件web.xml，读取**<context-param>和<listener>**两个结点。

2、紧急着，容创建一个**ServletContext（servlet上下文），**这个web项目的所有部分都将共享这个**上下文**。

3、容器将**<context-param>**转换为键值对，并交给servletContext。

4、容器创建<listener>中的类实例，创建监听器。

首先可以肯定的是，**加载顺序与它们在 web.xml 文件中的先后顺序无关**。即不会因为 filter 写在 listener 的前面而会先加载 filter。最终得出的结论是：**listener ->filter -> servlet** 。

同时还存在着这样一种**配置节：context-param**，它用于向 **ServletContext 提供键值对**，即应用程序上下文信息。我们的 listener, filter 等在初始化时会用到这些上下文中的信息，那么 context-param 配置节是不是应该写在 listener 配置节前呢？实际上context-param 配置节可写在任意位置，因此真正的加载顺序为：context-param ->listener -> filter -> servlet

对于某类配置节而言，与它们出现的顺序是有关的。以 filter 为例，web.xml 中当然可以定义多个 filter，**与 filter 相关的一个配置节是 filter-mapping**，这里一定要注意，对于拥有相同 filter-name 的 filter 和 filter-mapping 配置节而言，filter-mapping必须出现在 filter 之后，否则当解析到 filter-mapping 时，它所对应的 filter-name 还未定义。**web 容器启动时初始化每个 filter 时，是按照 filter 配置节出现的顺序来初始化的**，当请求资源匹配多个 filter-mapping 时，filter 拦截资源是按照 filter-mapping配置节出现的顺序来依次调用 doFilter() 方法的。

servlet 同 filter 类似，此处不再赘述。 servlet与servlet-mapping、filter与filter-mapping。

由此，可以看出，web.xml 的加载顺序是：context-param -> listener -> filter-> servlet ，而同个类型之间的实际程序调用的时候的顺序是根据对应的 mapping 的顺序进行调用的。

# load-on-startup理解

**load-on-startup 元素**在web应用启动的时候指定了**servlet被加载的顺序**，它的值必须是一个整数。

* 如果它的值是一个负整数或是这个元素不存在，那么容器会在该servlet被调用的时候，加载这个servlet 。
* 如果值是正整数或零，容器在配置的时候就加载并初始化这个servlet，容器必须保证值小的先被加载。如果值相等，容器可以自动选择先加载谁。

在servlet的配置当中，<load-on-startup>5</load-on-startup>的含义是：

标记容器是否在启动的时候就加载这个servlet。

* **当值为0或者大于0时，表示容器在应用启动时就加载这个servlet；**
* **当是一个负数时或者没有指定时，则指示容器在该servlet被选择时才加载。**

**正数的值越小，启动该servlet的优先级越高**;因此一般SpringMVC对应的DispatcherServlet的启动级别为设置为1。

# 配置Spring

<!-- 指定spring配置文件位置 -->

<context-param>

<param-name>**contextConfigLocation**</param-name>

<param-value>

<!--加载多个spring配置文件 -->

/WEB-INF/applicationContext.xml, /WEB-INF/action-servlet.xml

</param-value>

</context-param>

**<!-- 定义SPRING监听器，加载spring -->**

**<listener>**

**<listener-class>org.springframework.web.context.ContextLoaderListener**

**</listener-class>**

</listener>

<listener>

<listener-class> org.springframework.web.context.request.RequestContextListener

</listener-class>

</listener>

# web.xml文件详解

web.xml常用元素

<web-app>

<display-name></display-name>定义了WEB应用的名字；

<description></description> 声明WEB应用的描述信息；

<context-param></context-param> context-param元素声明应用范围内的初始化参数。

<filter></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></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></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></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></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></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></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。

<welcome-file-list></welcome-file-list> 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> 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> 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>声明与资源相关的一个管理对象。

<resource-ref></resource-ref> 声明一个资源工厂使用的外部资源。

<security-constraint></security-constraint> 制定应该保护的URL。它与

login-config元素联合使用

<login-config></login-config> 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>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>声明Web应用的环境项。

<ejb-ref></ejb-ref>声明一个EJB的主目录的引用。

< ejb-local-ref></ ejb-local-ref>声明一个EJB的本地主目录的应用。

</web-app>

# 相应元素具体配置

## 1、Web应用图标：指出IDE和GUI工具用来表示Web应用的大图标和小图标

<icon>

<small-icon>/images/app\_small.gif</small-icon>

<large-icon>/images/app\_large.gif</large-icon>

</icon>

## 2、Web 应用名称：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

<display-name>Tomcat Example</display-name>

## 3、Web 应用描述：给出于此相关的说明性文本

<disciption>Tomcat Example servlets and JSP pages.</disciption>

## 4、上下文参数：声明应用范围内的初始化参数。

<context-param>

<param-name>ContextParameter</para-name>

<param-value>test</param-value>

<description>It is a test parameter.</description>

</context-param>

在servlet里面可以通过**getServletContext().getInitParameter("context/param")**得到。

## 5、过滤器配置：将一个名字与一个实现javaxs.servlet.Filter接口的类相关联。

<filter>

<filter-name>setCharacterEncoding</filter-name>

<filter-class>com.myTest.setCharacterEncodingFilter</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>GB2312</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>setCharacterEncoding</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

## 6、监听器配置

<listener>

<listerner-class>listener.SessionListener</listener-class>

</listener>

## 7、Servlet配置

基本配置

<servlet>

<servlet-name>snoop</servlet-name>

<servlet-class>SnoopServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>snoop</servlet-name>

<url-pattern>/snoop</url-pattern>

</servlet-mapping>

高级配置

<servlet>

<servlet-name>snoop</servlet-name>

<servlet-class>SnoopServlet</servlet-class>

<init-param>

<param-name>foo</param-name>

<param-value>bar</param-value>

</init-param>

<run-as>

<description>Security role for anonymous access</description>

<role-name>tomcat</role-name>

</run-as>

</servlet>

<servlet-mapping>

<servlet-name>snoop</servlet-name>

<url-pattern>/snoop</url-pattern>

</servlet-mapping>

元素说明

<servlet></servlet> 用来声明一个servlet的数据，主要有以下子元素：

<servlet-name></servlet-name> 指定servlet的名称

<servlet-class></servlet-class> 指定servlet的类名称

<jsp-file></jsp-file> 指定web站台中的某个JSP网页的完整路径

<init-param></init-param> 用来定义参数，可有多个init-param。在servlet

类中通过**getInitParamenter(String name)**方法访问初始化参数

<load-on-startup></load-on-startup>指定当Web应用启动时，装载Servlet的次序。

当值为正数或零时：Servlet容器先加载数值小的servlet，再依次加载其他数值大的servlet. 当值为负或未定义：Servlet容器将在Web客户首次访问这个servlet时加载它。

<servlet-mapping></servlet-mapping> 用来定义servlet所对应的URL，包含

两个子元素

<servlet-name></servlet-name> 指定servlet的名称

<url-pattern></url-pattern> 指定servlet所对应的URL

## 8、会话超时配置（单位为分钟）

<session-config>

<session-timeout>120</session-timeout>

</session-config>

## 9、MIME类型配置

<mime-mapping>

<extension>htm</extension>

<mime-type>text/html</mime-type>

</mime-mapping>

## 10、指定欢迎文件页配置

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

<welcome-file>index.html</welcome-file>

<welcome-file>index.htm</welcome-file>

</welcome-file-list>

## 11、配置错误页面

一、通过错误码来配置error-page

<error-page>

<error-code>404</error-code>

<location>/NotFound.jsp</location>

</error-page>

上面配置了当系统发生404错误时，跳转到错误处理页面NotFound.jsp。

二、通过异常的类型配置error-page

<error-page>

<exception-type>java.lang.NullException</exception-type>

<location>/error.jsp</location>

</error-page>

上面配置了当系统发生java.lang.NullException（即空指针异常）时，跳转到错误处理

页面error.jsp

## 12、TLD配置

<taglib>

<taglib-uri>http://jakarta.apache.org/tomcat/debug-taglib</taglib-uri>

<taglib-location>/WEB-INF/jsp/debug-taglib.tld</taglib-location>

</taglib>

如果MyEclipse一直在报错,应该把<taglib> 放到 <jsp-config>中

<jsp-config>

<taglib>

<taglib-uri>http://jakarta.apache.org/tomcat/debug-taglib</taglib-uri>

<taglib-location>/WEB-INF/pager-taglib.tld</taglib-location>

</taglib>

</jsp-config>

## 13、资源管理对象配置

<resource-env-ref>

<resource-env-ref-name>jms/StockQueue</resource-env-ref-name>

</resource-env-ref>

## 14、资源工厂配置

<resource-ref>

<res-ref-name>mail/Session</res-ref-name>

<res-type>javax.mail.Session</res-type>

<res-auth>Container</res-auth>

</resource-ref>

配置数据库连接池就可在此配置：

<resource-ref>

<description>JNDI JDBC DataSource of shop</description>

<res-ref-name>jdbc/sample\_db</res-ref-name>

<res-type>javax.sql.DataSource</res-type>

<res-auth>Container</res-auth>

</resource-ref>

## 15、安全限制配置

<security-constraint>

<display-name>Example Security Constraint</display-name>

<web-resource-collection>

<web-resource-name>Protected Area</web-resource-name>

<url-pattern>/jsp/security/protected/\*</url-pattern>

<http-method>DELETE</http-method>

<http-method>GET</http-method>

<http-method>POST</http-method>

<http-method>PUT</http-method>

</web-resource-collection>

<auth-constraint>

<role-name>tomcat</role-name>

<role-name>role1</role-name>

</auth-constraint>

</security-constraint>

## 16、登陆验证配置

<login-config>

<auth-method>FORM</auth-method>

<realm-name>Example-Based Authentiation Area</realm-name>

<form-login-config>

<form-login-page>/jsp/security/protected/login.jsp</form-login-page>

<form-error-page>/jsp/security/protected/error.jsp</form-error-page>

</form-login-config>

</login-config>

## 17、安全角色：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。

<security-role>

<role-name>tomcat</role-name>

</security-role>

## 18、Web环境参数：env-entry元素声明Web应用的环境项

<env-entry>

<env-entry-name>minExemptions</env-entry-name>

<env-entry-value>1</env-entry-value>

<env-entry-type>java.lang.Integer</env-entry-type>

</env-entry>

## 19、EJB 声明

<ejb-ref>

<description>Example EJB reference</decription>

<ejb-ref-name>ejb/Account</ejb-ref-name>

<ejb-ref-type>Entity</ejb-ref-type>

<home>com.mycompany.mypackage.AccountHome</home>

<remote>com.mycompany.mypackage.Account</remote>

</ejb-ref>

## 20、本地EJB声明

<ejb-local-ref>

<description>Example Loacal EJB reference</decription>

<ejb-ref-name>ejb/ProcessOrder</ejb-ref-name>

<ejb-ref-type>Session</ejb-ref-type>

<local-home>com.mycompany.mypackage.ProcessOrderHome</local-home>

<local>com.mycompany.mypackage.ProcessOrder</local>

</ejb-local-ref>

## 21、配置DWR

<servlet>

<servlet-name>dwr-invoker</servlet-name>

<servlet-class>uk.ltd.getahead.dwr.DWRServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>dwr-invoker</servlet-name>

<url-pattern>/dwr/\*</url-pattern>

</servlet-mapping>

## 22、配置Struts

<display-name>Struts Blank Application</display-name>

<servlet>

<servlet-name>action</servlet-name>

<servlet-class>

org.apache.struts.action.ActionServlet

</servlet-class>

<init-param>

<param-name>detail</param-name>

<param-value>2</param-value>

</init-param>

<init-param>

<param-name>debug</param-name>

<param-value>2</param-value>

</init-param>

<init-param>

<param-name>config</param-name>

<param-value>/WEB-INF/struts-config.xml</param-value>

</init-param>

<init-param>

<param-name>application</param-name>

<param-value>ApplicationResources</param-value>

</init-param>

<load-on-startup>2</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>action</servlet-name>

<url-pattern>\*.do</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

<!-- Struts Tag Library Descriptors -->

<taglib>

<taglib-uri>struts-bean</taglib-uri>

<taglib-location>/WEB-INF/tld/struts-bean.tld</taglib-location>

</taglib>

<taglib>

<taglib-uri>struts-html</taglib-uri>

<taglib-location>/WEB-INF/tld/struts-html.tld</taglib-location>

</taglib>

<taglib>

<taglib-uri>struts-nested</taglib-uri>

<taglib-location>/WEB-INF/tld/struts-nested.tld</taglib-location>

</taglib>

<taglib>

<taglib-uri>struts-logic</taglib-uri>

<taglib-location>/WEB-INF/tld/struts-logic.tld</taglib-location>

</taglib>

<taglib>

<taglib-uri>struts-tiles</taglib-uri>

<taglib-location>/WEB-INF/tld/struts-tiles.tld</taglib-location>

</taglib>

23、