# web.xml 中的listener、 filter、servlet 加载顺序及其详解

## 一、概述

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<listener>和<context-param>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。

## 二 、 load-on-startup

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。

## 三 、加载顺序

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：ServletContext -> 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。

由此，可以看出，web.xml 的加载顺序是：ServletContext -> 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。

## 四 、 web.xml文件详解

我将自己知道的web.xml的元素整理了一下：

### 1、schema

1、web.xml首先是肯定要包含它的schema.

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd"

version="2.4">

</web-app>

其它的元素都放在<web-app></web-app>之中。

<discription></discription> 是对站台的描述

<display-name></display-name> 定义站台的名称

<distributable/> 是指定该站台是否可分布式处理

<context-param></context-param> 用来设定web站台的环境参数，它包含两个子元素：

<param-name></param-name> 用来指定参数的名称

<param-value></param-value> 用来设定参数值

比如：

<context-param>

<param-name>my\_param</param-name>

<param-value>hello</param-value>

</context-param>

在此设定的参数，可以在servlet中用 getServletContext().getInitParameter("my\_param") 来取得

### 2、listener

<listener></listener> 用来设定Listener接口，它的主要子元素为

<listener-class></listener-class> 定义Listener的类名称

比如：

<listener>

<listener-class>com.myTest.ContextListener</listener-class>

</listener>

### 3、filter

3.1、filter

<filter></filter> 是用来声明filter的相关设定，它包含以下子元素：

<filter-name></filter-name> 这当然就是指定filter的名字

<filter-class></filter-class> 这是用来定义filter的类的名称

<init-param></init-param> 用来定义参数，它有两个子元素：

<param-name></param-name> 用来指定参数的名称

<param-value></param-value> 用来设定参数值

比如：

<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>

3.2、filter-mapping

与<filter></filter>同时使用的是<filter-mapping></filter-mapping> 用来定义filter所对应的URL，它有两个子元素：

<filter-name></filter-name> 指定filter的名字

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

比如：

<filter-mapping>

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

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

</filter-mapping>

### 4、servlet

4.1、servlet

<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，和前面的<init-param>差不多

4.2、servlet-mapping

与<servlet></servlet>一起使用的是<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

比如：

<servlet>

<servlet-name>ShoppingServlet</servlet-name>

<servlet-class>com.myTest.ShoppingServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>ShoppingServlet</servlet-name>

<url-pattern>/shop/ShoppingServlet</url-pattern>

</servlet-mapping>

### 5、session-config

<session-config></session-config> 用来定义web站台中的session参数，包含一个子元素：

<session-timeout></session-timeout> 用来定义这个web站台所有session的有效期限，单位为 分钟

### 6、mime-mapping

<mime-mapping></mime-mapping> 定义某一个扩展名和某一个MIME Type做对映，包含两个子元素：

<extension></extension> 扩展名的名称

<mime-type></mime-type> MIME格式

比如：

<mime-mapping>

<extension>doc</extension>

<mime-type>application/vnd.ms-word</mime-type>

</mime-mapping>

<mime-mapping>

<extension>xls</extension>

<mime-type>application/vnd.ms-excel</mime-type>

</mime-mapping>

### 7、welcom-file-list

<welcome-file-list></welcom-file-list> 用来定义首页的列单，包含一个子元素：

<welcome-file></welcome-file> 指定首页的文件名称

比如：

<welcome-file-list>

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

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

</welcom-file-list>

### 8、error-page

<error-page></error-page> 用来处理错误代码或异常的页面，有三个子元素：

<error-code></error-code> 指定错误代码

<exception-type></exception-type> 指定一个JAVA异常类型

<location></location> 指定在web站台内的相关资源路径

比如：

<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>

### 9、taglib

<taglib></taglib> 用来设定JSP网页所用到的Tag Library路径，有两个子元素：

<taglib-uri></taglib-uri> 定义TLD文件的URI，在JSP网页中用taglib指令便可取得该URI的TLD文件

<taglib-location></taglib-location> 指定TLD文件相对于web站台的存放位置

比如：

<taglib>

<taglib-uri>myTaglib</taglib-uri>

<taglib-location>/WEB-INF/tlds/MyTaglib.tld</taglib-location>

</taglib>

### 10、resource-ref

<resource-ref></resource-ref> 定义利用JNDI取得站台可利用的资源，有五个子元素：

<description></description> 资源说明

<rec-ref-name></rec-ref-name> 资源名称

<res-type></res-type> 资源种类

<res-auth></res-auth> 资源经由Application或Container来许可

<res-sharing-scope></res-sharing-scope> 资源是否可以共享，有Shareable和Unshareable两个 值，默认为Shareable

比如，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>

### 11、jsp-config

<jsp-config> 包括<taglib> 和<jsp-property-group> 两个子元素。

其中<taglib>元素在JSP 1.2时就已经存在；而<jsp-property-group>是JSP 2.0 新增的元素。

<jsp-property-group>元素主要有八个子元素，它们分别为：

1).<description>：设定的说明；

2).<display-name>：设定名称；

3).<url-pattern>：设定值所影响的范围，如：/CH2 或 /\*.jsp；

4).<el-ignored>：若为true，表示不支持EL 语法；

5).<scripting-invalid>：若为true，表示不支持<% scripting %>语法；

6).<page-encoding>：设定JSP 网页的编码；

7).<include-prelude>：设置JSP 网页的抬头，扩展名为.jspf；

8).<include-coda>：设置JSP 网页的结尾，扩展名为.jspf。

一个简单的<jsp-config>元素完整配置：

<jsp-config>

<taglib>

<taglib-uri>Taglib</taglib-uri>

<taglib-location>/WEB-INF/tlds/MyTaglib.tld</taglib-location>

</taglib>

<jsp-property-group>

<description>Special property group for JSP Configuration JSP example.</description>

<display-name>JSPConfiguration</display-name>

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

<el-ignored>true</el-ignored>

<page-encoding>GB2312</page-encoding>

<scripting-invalid>true</scripting-invalid>

<include-prelude>/include/prelude.jspf</include-prelude>

<include-coda>/include/coda.jspf</include-coda>

</jsp-property-group>

</jsp-config>

配置web.xml来限制对某些servlet的请求

有时我们只希望通过认证的用户才能请求某些servlet的话，就可以在web.xml中来进行相应的配置，来达到此目的。

这就要用到<security-constraint></security-constraint>元素。

对于tomcat，中web.xml使用security-constraint元素需要在位于<Tomcat-installation-directory>/conf/tomcat-users.xml的XML文件中创建用户名和密码。比如下面的这个tomcat-users.xml文件：

<?xml version='1.0' encoding='utf-8'?>

<tomcat-users>

<role rolename="tomcat"/>

<role rolename="manager"/>

<role rolename="admin"/>

<user username="tomcat" password="tomcat" roles="tomcat"/>

<user username="both" password="tomcat" roles="tomcat,manager"/>

<user username="admin" password="admin" roles="admin"/>

</tomcat-users>

此XML片段包括一个tomcat-users根元素，它包含一个或多个role和user元素。

然后在Web应用程序的web.xml中创建security-constraint、login-config和security-role元素。

<security-constraint>

<web-resource-collection>

<web-resource-name>HelloServlet</web-resource-name>

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

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

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

</web-resource-collection>

<auth-constraint>

<description>This applies only to the "tomcat" security role</description>

<role-name>admin</role-name>

</auth-constraint>

<user-data-constraint>

<transport-guarantee>NONE</transport-guarantee>

</user-data-constraint>

</security-constraint>

<login-config>

<auth-method>BASIC</auth-method>

</login-config>

<security-role>

<role-name>admin</role-name>

</security-role>

其中security-constraint元素包含一个或多个web-resource-collection元素，它是描述Web应用程序中的哪些web资源受到指定安全限制的保护。http-method元素指定安全限制覆盖的HTTP方法。上面的例子中，当我们对/HelloServlet的GET或POST请求时将触发配置的安全机制。

auth-constraint元素用于描述允许访问Web组件的安全角色。此例中安全角色的例子有tomcat、manager、admin。而只有当作为admin角色的用户才可以访问HelloServlet。

Web应用程序通过login-config元素来认证用户，并确认该用户是否为正确的角色。

longin-config包含的transport-guarantee子元素用来指定认证方法，BASIC是一种常见的Web认证方式，浏览器给用户提示一个对话框，要求输入用户名和密码，随后Tomcat将给出的用户名和密码与tomcat-users.xml中的用户名和密码进行比较，然后使用前面的security-constraint配置来确定用户是否可访问受保护的servlet。

（除BASIC外，还可以是FORM、CLIENT-CERT、DIGEST等）

其实这种认证方法实际上有两个步骤：

1、检查提供的用户名和密码是否正确。

2、判断用户是否映射到特定的安全角色。例如，用户可能提供了正确的用户名和密码，但没有映射到特定的安全角色，也将被禁止访问特定的Web资源。