

Web.xml详解 - believejava的专栏 - 博客频道 - CSDN

这篇文章主要是综合网上关于web.xml的一些介绍，希望对大家有所帮助，也欢迎大家一起讨论。 ---题记

一、Web.xml详解：

(一)web.xml加载过程（步骤）

首先简单说一下，web.xml的加载过程。

当我们去启动一个WEB项目时，容器包括（JBoss、Tomcat等）首先会读取项目web.xml配置文件里的配置，当这一步骤没有出错并且完成之后，项目才能正常地被启动起来。

1启动WEB项目的时候，容器首先会去它的配置文件web.xml读取两个节点：

和。

1紧接着，容器创建一个ServletContext（application），这个WEB项目所有部分都将共享这个上下文。

1容器以的name作为键，value作为值，将其转化为键值对，存入ServletContext。

1容器创建中的类实例，根据配置的class类路径来创建监听，在监听中会

有contextInitialized(ServletContextEvent args)初始化方法，启动Web应用时，系统调用Listener的该方法，在这个方法中获得：

ServletContext application =ServletContextEvent.getServletContext();

context-param的值= **application.getInitParameter("context-param的键");**

得到这个context-param的值之后，你就可以做一些操作了。

1举例：你可能想在项目启动之前就打开数据库，那么这里就可以在中设置数据库的连接方式（驱

动、url、user、password），在监听类中初始化数据库的连接。这个监听是自己写的一个类，除了初始化方法，它还有销毁方法，用于关闭应用前释放资源。比如：说数据库连接的关闭，此时，调

用contextDestroyed(ServletContextEvent args)，关闭Web应用时，系统调用Listener的该方法。

1接着，容器会读取，根据指定的类路径来实例化过滤器。

1以上都是在WEB项目还没有完全启动起来的时候就已经完成了的工作。如果系统中有Servlet，则Servlet是在第一次发起请求的时候被实例化的，而且一般不会被容器销毁，它可以服务于多个用户的请求。所以，Servlet的初始化都要比上面提到的那几个要迟。

总的来说，web.xml的加载顺序是：-> -> 。其中，如果web.xml中出现了相同的元素，则按照在配置文件中出现的先后顺序来加载。

对于某类元素而言，与它们出现的顺序是有关的。以为例，web.xml中当然可以定义多个，与相关的一个元素是，注意，对于拥有相同的和元素而言，必须出现在之后，否则当解析到时，它所对应的还未定义。web容器启动初始化每个时，按照出现的顺序来初始化的，当请求资源匹配多个时，拦截资源是按照元素出现的顺序来依次调用doFilter()方法的。同类似，此处不再赘述。

(二)web.xml标签详解

1. XML文档有效性检查

```
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
```

```
"http://Java.sun.com/dtd/web-app_2_3.dtd" >
```

这段代码指定文件类型定义(DTD)，可以通过它检查XML文档的有效性。下面显示的元素有几个特性，这些特性告诉我们关于DTD的信息：

- web-app定义该文档(部署描述符，不是DTD文件)的根元素
- PUBLIC意味着DTD文件可以被公开使用
- "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"意味着DTD由Sun Microsystems, Inc.维护。该信息也表示它描述的文档类型是DTD Web Application 2.3，而且DTD是用英文书写的。
- URL"http://java.sun.com/dtd/web-app_2_3.dtd"表示D文件的位置。

2.

部署描述符的根元素是。DTD文件规定元素的子元素的语法如下：

distributable?, context-param*, filter*, filter-mapping*,
listener*, servlet*, servlet-mapping*, session-config?,
mime-mapping*, welcome-file-list?,
error-page*, taglib*, resource-env-ref*, resource-ref*,
security-constraint*, login-config?, security-role*,env-entry*,
ejb-ref*, ejb-local-ref*)>

正如您所看到的，这个元素含有23个子元素，而且子元素都是可选的。问号(?)表示子元素是可选的，而且只能出现一次。星号(*)表示子元素可在部署描述符中出现零次或多次。有些子元素还可以有它们自己的子元素。
web.xml文件中元素声明的是下面每个子元素的声明。下面讲述部署描述符中可能包含的所有子元素。

注意：

在Servlet 2.3中，子元素必须按照DTD文件语法描述中指定的顺序出现。比如：如果部署描述符中的元素有和两个子元素，则子元素必须出现在子元素之前。在Servlet2.4中，顺序并不重要。

3.

test-hwp-web-application定义了web应用的名称，可以在<http://localhost:8080/manager/html>中显示。如下所示：

Applications					
Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle > 30 minutes
/docs	None specified	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle > 30 minutes
/examples	None specified	Servlet and JSP Examples	true	0	Start Stop Reload Undeploy Expire sessions with idle > 30 minutes
/host-manager	None specified	Tomcat Host Manager Application	true	0	Start Stop Reload Undeploy Expire sessions with idle > 30 minutes
/manager	None specified	Tomcat Manager Application	true	2	Start Stop Reload Undeploy Expire sessions with idle > 30 minutes
/test-hwp	None specified	test-hwp-web-application	true	0	Start Stop Reload Undeploy Expire sessions with idle > 30 minutes

4.

可以使用distributable元素来告诉servlet/JSP容器，Web容器中部署的应用程序适合在分布式环境下运行。

5.

[html]view plaincopy


在CODE上查看代码片
派生到我的代码片

1. <context-param>
2. <param-name>webAppRootKeyparam-name>
3. <param-value>business.rootparam-value>
4. context-param>
5. <context-param>
6. <param-name>contextConfigLocationparam-name>
7. <param-value>/WEB-INF/spring-configuration/*.xmlparam-value>
8. context-param>

- >解释：

元素含有一对参数名和参数值，用作应用的Servlet上下文初始化参数，参数名在整个Web应用中必须是惟一的，

在web应用的整个生命周期中上下文初始化参数都存在，任意的Servlet和jsp都可以随时随地访问它。<param-name>子元素包含有参数名，而子元素包含的是参数值。作为选择，可用子元素来描述参数。

- 什么情况下使用，为什么使用：

比如：定义一个管理员email地址用来从程序发送错误，或者与你整个应用程序有关的其他设置。使用自己定义的设置文件需要额外的代码和管理；直接在你的程序中使用硬编码（Hard-coding）参数值会给你之后修改程序带来麻烦，更困难的是，要根据不同的部署使用不同的设置；通过这种办法，可以让其他开发人员更容易找到相关的参数，因为它是一个用于设置这种参数的标准位置。

- Spring配置文件：

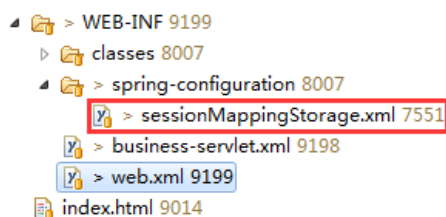
配置spring，必须需要，而可有可无，如果在web.xml中不写配置信息，默认的路径是/WEB-INF/applicationContext.xml，在WEB-INF目录下创建的xml文件的名称必须是applicationContext.xml。如果是要自定义文件名可以在web.xml里加入contextConfigLocation这个context参数：在里指定相应的xml文件名，如果有多个xml文件，可以写在一起并以“,”号分隔，比如在business-client工程中，我们采用了自定义配置方式，配置如下：

[html]view plaincopy

 在CODE上查看代码片
 派生到我的代码片

1. <context-param>
2. <param-name>contextConfigLocationparam-name>
3. <param-value>/WEB-INF/spring-configuration/*.xmlparam-value>
4. context-param>
5. <listener>
6. <listener-class>org.springframework.web.context.ContextLoaderListenerlistener-class>
7. listener>

对应工程目录结构如下所示：



- webAppRootKey配置：

部署在同一容器中的多个Web项目，要配置不同的webAppRootKey，web.xml文件中最好定义webAppRootKey参数，如果不定义，将会缺省为“webapp.root”，如下：

[html]view plaincopy

 在CODE上查看代码片
 派生到我的代码片

1. <context-param>
2. <param-name>webAppRootKeyparam-name>
3. <param-value>webapp.rootparam-value>
4. context-param>

当然也不能重复，否则报类似下面的错误：

```
Web app root system property already set to different value: 'webapp.root' =  
[/home/user/tomcat/webapps/project1/] instead of [/home/user/tomcat/webapps/project2/] -  
Choose unique values for the 'webAppRootKey' context-param in your web.xml files!
```

意思是“webapp.root”这个key已经指向了项目1，不可以再指向项目2。多个项目要对webAppRootKey进行配置，我们工程主要是让log4j能将日志写到对应项目根目录下，比如：我们的项目的webAppRootKey为

[html]view plaincopy

 在CODE上查看代码片
 派生到我的代码片

1. >
2. <context-param>
3. <param-name>webAppRootKeyparam-name>
4. <param-value> business.root param-value>
5. context-param>
6. >
7. <context-param>
8. <param-name>webAppRootKeyparam-name>
9. <param-value> pubbase.rootparam-value>
10. context-param>

这样就不会出现冲突了。就可以在运行时动态地找到项目路径，在log4j.properties配置文件中可以按下面的方式使用\${webapp.root}：

```
log4j.appender.file.File=${webapp.root}/WEB-INF/logs/sample.log
```

就可以在运行时动态地找出项目的路径。

- 多个配置文件交叉引用处理：

如果web.xml中有contextConfigLocation参数指定的Spring配置文件，则会去加载相应的配置文件，而不会去加载/WEB-INF/下的applicationContext.xml。但是如果没有指定的话，默认会去/WEB-INF/下加载applicationContext.xml。

在一个团队使用Spring的实际项目中，应该需要多个Spring的配置文件，如何使用和交叉引用的问题：

多个配置文件可以在web.xml里用空格分隔写入，如：

[html]view plaincopy

 在CODE上查看代码片
 派生到我的代码片

1. <context-param>
2. <param-name>contextConfigLocation param-name>
3. <param-value> applicationContext-database.xml,applicationContext.xmlparam-value>
4. <context-param>

多个配置文件里的交叉引用可以用ref的external或bean解决，例如：

```
applicationContext.xml
```

[html]view plaincopy

 在CODE上查看代码片
 派生到我的代码片

1. `<beanid="userService"class="domain.user.service.impl.UserServiceImpl">`
2. `<propertyname="dbbean">`
3. `<refbean="dbBean"/>`
4. `property>`
5. `bean>`

dbBean在applicationContext-database.xml中。

- 在不同环境下如何获取：

范例：

[html]view plaincopy

 在CODE上查看代码片
 派生到我的代码片

1. `<context-param>`
2. `<param-name>param_nameparam-name>`
3. `<param-value>param_valueparam-value>`
4. `context-param>`

此所设定的参数，在JSP网页中可以使用下列方法来取得：

`${initParam.param_name}`

若在Servlet可以使用下列方法来获得：

`String param_name=getServletContext().getInitParameter("param_name");`

Servlet的ServletConfig对象拥有该Servlet的ServletContext的一个引用，所以可这样取得上下文初始化参数：`getServletConfig().getServletContext().getInitParameter()`也可以在Servlet中直接调用`getServletContext().getInitParameter()`，两者是等价的。

6. `<session-config>``session-config>`

[html]view plaincopy

 在CODE上查看代码片
 派生到我的代码片

1. `<session-config>`
2. `<session-timeout>120session-timeout>`
3. `session-config>`

`<session-config>` 用于设置容器的session参数，比如：`<session-timeout>`用于指定http session的失效时间。默认时间设置在/conf/web.xml (30 minutes)。`<session-timeout>`用来指定默认的会话超时时间间隔，以分钟为单位。该元素值必须为整数。如果session-timeout元素的值为零或负数，则表示会话将永远不会超时。

7.

[html]view plaincopy

 在CODE上查看代码片
 派生到我的代码片

1. `<listener>`
2. `<listener-class>org.springframework.web.util.Log4jConfigListenerlistener-class>`
3. `listener>`

4. <listener>

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

6. <listener>

7. <listener>

8. <listener-class>com.yonyou.mcloud.cas.client.session.SingleSignOutHttpSessionListener</listener-class>

9. <listener>

1. Listener介绍:

为web应用程序定义监听器，监听器用来监听各种事件，比如：application和session事件，所有的监听器按照相同的方式定义，功能取决于它们各自实现的接口，常用的Web事件接口有如下几个：

- **ServletContextListener**：用于监听Web应用的启动和关闭；
- **ServletContextAttributeListener**：用于监听**ServletContext**范围（**application**）内属性的改变；
- **ServletRequestListener**：用于监听用户的请求；
- **ServletRequestAttributeListener**：用于监听**ServletRequest**范围（**request**）内属性的改变；
- **HttpSessionListener**：用于监听用户session的开始和结束；
- **HttpSessionAttributeListener**：用于监听**HttpSession**范围（**session**）内属性的改变。

主要用于监听Web应用事件，其中有两个比较重要的WEB应用事件：**应用的启动和停止**（starting up or shutting down）和**Session的创建和失效**（created or destroyed）。应用启动事件发生在应用第一次被Servlet容器装载和启动的时候；停止事件发生在Web应用停止的时候。Session创建事件发生在每次一个新的session创建的时候，类似地Session失效事件发生在每次一个Session失效的时候。为了使用这些Web应用事件做些有用的事情，我们必须创建和使用一些特殊的“监听类”。它们是实现了以下两个接口中任何一个接口的简单java类：**javax.servlet.ServletContextListener**或**javax.servlet.http.HttpSessionListener**，如果你想让你的类监听应用的启动和停止事件，你就得实现ServletContextListener接口；想让你的类去监听Session的创建和失效事件，那你就得实现HttpSessionListener接口。

2. Listener配置:

配置Listener只要向Web应用注册Listener实现类即可，无需配置参数之类的东西，因为Listener获取的是Web应用**ServletContext**（**application**）的配置参数。为Web应用配置Listener的两种方式：

1使用@WebListener修饰Listener实现类即可。

1在web.xml文档中使用进行配置。

我们选择web.xml这种配置方式，只有一个元素指定Listener的实现类，如下所示：

`org.springframework.web.context.ContextLoaderListener`

这里的用于Spring的加载，Spring加载可以利用**ServletContextListener**实现，也可以采用**load-on-startup Servlet**实现，但是当需要用到bean时，但加载顺序是：先加载后加载，则中初始化操作中的bean为null；所以，如果过滤器中要使用到bean，此时就可以根据加载顺序 -> ，将spring的加载改成Listener的方式。

1)利用ServletContextListener实现:

[html]view plain copy


在CODE上查看代码片

派生到我的代码片

1. <servlet>

2. <servlet-name>contextServlet-name</servlet-name>

3. <servlet-class>org.springframework.web.context.ContextLoaderServlet</servlet-class>

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

5. </servlet>

2)采用load-on-startup Servlet 实现:

[html]view plaincopy


在CODE上查看代码片

派生到我的代码片

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

我们选择了第二种方式，在J2EE工程中web服务器启动的时候最先调用web.xml，上面这段配置的意思是加载spring的监听器，其中ContextLoaderListener的作用就是启动Web容器时，自动装配applicationContext.xml的配置信息，执行它所实现的方法。

8.

[html]view plaincopy


在CODE上查看代码片

派生到我的代码片

1. <filter>
2. <filter-name>CharacterEncodingFilter</filter-name>
3. <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
4. </filter>
5. <param-name>encoding</param-name>
6. <param-value>UTF-8</param-value>
7. </filter>
8. <filter>
9. <filter-name>forceEncodingFilter</filter-name>
10. <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
11. </filter>
12. <filter>
13. <filter-name>CAS Single Sign Out Filter</filter-name>
14. <filter-class>com.yonyou.mcloud.cas.client.session.SingleSignOutFilter</filter-class>
15. </filter>
16. <filter>
17. <filter-name>CAS Authentication Filter</filter-name>
18. <filter-class>com.yonyou.mcloud.cas.client.authentication.ExpandAuthenticationFilter</filter-class>
19. </filter>
20. <filter>
21. <filter-name>casServerLoginUrl</filter-name>
22. <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
23. </filter>
24. <filter>
25. <filter-name>serverName</filter-name>
26. <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
27. </filter>
28. <filter>
29. <filter-name>CAS Validation Filter</filter-name>
30. <filter-class>org.jasig.cas.client.validation.Cas20ProxyReceivingTicketValidationFilter</filter-class>
31. </filter>

32. <init-param>
33. <param-name>casServerUrlPrefixparam-name>
34. <param-value>https://dev.yonyou.com:443/sso-serverparam-value>
35. init-param>
36. <init-param>
37. <param-name>serverNameparam-name>
38. <param-value>http://10.1.215.40:80param-value>
39. init-param>
40. <init-param>
41. <param-name>proxyCallbackUrlparam-name>
42. <param-value>https://dev.yonyou.com:443/business/proxyCallbackparam-value>
43. init-param>
44. <init-param>
45. <param-name>proxyReceptorUrlparam-name>
46. <param-value>/proxyCallbackparam-value>
47. init-param>
48. <init-param>
49. <param-name>proxyGrantingTicketStorageClassparam-name>
50. <param-
value>com.yonyou.mcloud.cas.client.proxy.MemcachedBackedProxyGrantingTicketStorageImplparam-
value>
51. init-param>
52. <init-param>
53. <param-name>encodingparam-name>
54. <param-value>UTF-8param-value>
55. init-param>
56. filter>
57. <filter>
58. <filter-name>CAS HttpServletRequest Wrapper Filterfilter-name>
59. <filter-class>org.jasig.cas.client.util.HttpServletRequestWrapperFilterfilter-class>
60. filter>
61. <filter>
62. <filter-name>CAS Assertion Thread Local Filterfilter-name>
63. <filter-class>org.jasig.cas.client.util.AssertionThreadLocalFilterfilter-class>
64. filter>
65. <filter>
66. <filter-name>NoCache Filterfilter-name>
67. <filter-class>com.yonyou.mcloud.cas.client.authentication.NoCacheFilterfilter-class>
68. filter>
69. <filter-mapping>
70. <filter-name>CharacterEncodingFilterfilter-name>
71. <url-pattern>/*url-pattern>
72. filter-mapping>
73. <filter-mapping>
74. <filter-name>NoCache Filterfilter-name>
75. <url-pattern>/*url-pattern>
76. filter-mapping>

77. <filter-mapping>
 78. <filter-name>CAS Single Sign Out Filter<filter-name>
 79. <url-pattern>/*<url-pattern>
 80. filter-mapping>
 81. <filter-mapping>
 82. <filter-name>CAS Validation Filter<filter-name>
 83. <url-pattern>/proxyCallback<url-pattern>
 84. filter-mapping>
 85. <filter-mapping>
 86. <filter-name>CAS Authentication Filter<filter-name>
 87. <url-pattern>/*<url-pattern>
 88. filter-mapping>
 89. <filter-mapping>
 90. <filter-name>CAS Validation Filter<filter-name>
 91. <url-pattern>/*<url-pattern>
 92. filter-mapping>
 93. <filter-mapping>
 94. <filter-name>CAS HttpServletRequest Wrapper Filter<filter-name>
 95. <url-pattern>/*<url-pattern>
 96. filter-mapping>
 97. <filter-mapping>
 98. <filter-name>CAS Assertion Thread Local Filter<filter-name>
 99. <url-pattern>/*<url-pattern>
 100. filter-mapping>

1. Filter介绍:

Filter可认为是Servlet的一种“加强版”，主要用于对用户请求request进行预处理，也可以对Response进行后处理，是个典型的处理链。使用Filter的完整流程是：Filter对用户请求进行预处理，接着将请求HttpServletRequest交给Servlet进行处理并生成响应，最后Filter再对服务器响应HttpServletResponse进行后处理。Filter与Servlet具有完全相同的生命周期，且Filter也可以通过来配置初始化参数，获取Filter的初始化参数则使用FilterConfig的getInitParameter()。

换种说法，Servlet里有request和response两个对象，Filter能够在一个request到达Servlet之前预处理request，也可以在离开Servlet时处理response，Filter其实是一个Servlet链。以下是Filter的一些常见应用场合，

- (1) 认证Filter
- (2) 日志和审核Filter
- (3) 图片转换Filter
- (4) 数据压缩Filter
- (5) 密码Filter
- (6) 令牌Filter
- (7) 触发资源访问事件的Filter
- (8) XSLT Filter
- (9) 媒体类型链Filter

Filter可负责拦截多个请求或响应；一个请求或响应也可被多个Filter拦截。创建一个Filter只需两步：

- 创建Filter处理类
- Web.xml文件中配置Filter

Filter必须实现javax.servlet.Filter接口，在该接口中定义了三个方法：

- void init(FilterConfig config)：用于完成Filter的初始化。FilterConfig用于访问Filter的配置信息。
- void destroy()：用于Filter销毁前，完成某些资源的回收。
- void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)：实现过滤功能的核心方法，该方法就是对每个请求及响应增加额外的处理。该方法实现对用户请求request进行预处理，也可以实现对服务器响应response进行后处理---它们的分界线为是否调用了chain.doFilter(request, response)，执行该方法之前，即对用户请求request进行预处理，执行该方法之后，即对服务器响应response进行后处理。

2. Filter配置：

Filter可认为是Servlet的“增强版”，因此Filter配置与Servlet的配置非常相似，需要配置两部分：配置Filter名称和Filter拦截器URL模式。区别在于Servlet通常只配置一个URL，而Filter可以同时配置多个请求的URL。配置Filter有两种方式：

1在Filter类中通过Annotation进行配置。

1在web.xml文件中通过配置文件进行配置。

我们使用的是web.xml这种配置方式，下面重点介绍内包含的一些元素。

用于指定Web容器中的过滤器，可包含、、、、。

1-name>用来定义过滤器的名称，该名称在整个程序中都必须唯一。

1元素指定过滤器类的完全限定的名称，即Filter的实现类。

1为Filter配置参数，与具有相同的元素描述符和。

1元素用来声明Web应用中的过滤器映射，过滤器被映射到一个servlet或一个URL模式。这个过滤器的和必须具有相同的，指定该Filter所拦截的URL。过滤是按照部署描述符的出现的顺序执行的。

1) 字符集过滤器

[html]view plaincopy


在CODE上查看代码片

派生到我的代码片

1. <filter>
2. <filter-name>CharacterEncodingFilterfilter-name>
3. <filter-class>org.springframework.web.filter.CharacterEncodingFilterfilter-class>
4. <init-param>
5. <param-name>encodingparam-name>
6. <param-value>UTF-8param-value>
7. init-param>
8. <init-param>
9. <param-name>forceEncodingparam-name>
10. <param-value>trueparam-value>
11. init-param>
12. filter>
13. <filter-mapping>
14. <filter-name>CharacterEncodingFilterfilter-name>
15. <url-pattern>/*url-pattern>
16. filter-mapping>

CharacterEncodingFilter类可以通过简单配置来帮我们实现字符集转换的功能，参数encoding用于指定编码类型，参数forceEncoding设为true时，强制执行request.setCharacterEncoding(this.encoding)和response.setCharacterEncoding(this.encoding)中的方法。

2) 缓存控制

[html]view plaincopy


在CODE上查看代码片

派生到我的代码片

1. **<filter>**
2. **<filter-name>NoCache Filterfilter-name>**
3. **<filter-class>com.yonyou.mcloud.cas.client.authentication.NoCacheFilterfilter-class>**
4. **filter>**
5. **<filter-mapping>**
6. **<filter-name>NoCache Filterfilter-name>**
7. **>**
8. **<url-pattern>/*url-pattern>**
9. **filter-mapping>**

3) 登录认证

[html]view plaincopy


在CODE上查看代码片

派生到我的代码片

1. **<filter>**
2. **<filter-name>CAS Authentication Filterfilter-name>**
3. **<filter-class>com.yonyou.mcloud.cas.client.authentication.ExpandAuthenticationFilterfilter-class>**
4. **<init-param>**
5. **<param-name>casServerLoginUrlparam-name>**
6. **<param-value>https://dev.yonyou.com:443/sso-server/loginparam-value>**
7. **init-param>**
8. **<init-param>**
9. **<param-name>serverNameparam-name>**
10. **<param-value>http://10.1.215.40:80param-value>**
11. **init-param>**
12. **filter>**
13. **<filter-mapping>**
14. **<filter-name>CAS Authentication Filterfilter-name>**
15. **<url-pattern>/*url-pattern>**
16. **filter-mapping>**

登录认证，未登录用户导向CAS Server进行认证。

4) 单点登出

[html]view plaincopy


在CODE上查看代码片

派生到我的代码片

1. **<filter>**
2. **<filter-name>CAS Single Sign Out Filterfilter-name>**
3. **<filter-class>org.jasig.cas.client.session.SingleSignOutFilterfilter-class>**
4. **filter>**

5. **<filter-mapping>**
6. **<filter-name>CAS Single Sign Out Filterfilter-name>**
7. **<url-pattern>/*url-pattern>**
8. **filter-mapping>**
9. **<listener>**
10. **<listener-class>org.jasig.cas.client.session.SingleSignOutHttpSessionListenerlistener-class>**
11. **listener>**

CAS Server通知CAS Client，删除session，注销登录信息。

5) 封装request

[html]view plaincopy


在CODE上查看代码片

派生到我的代码片

1. **<filter>**
2. **<filter-name>CAS HttpServletRequest Wrapper Filterfilter-name>**
3. **<filter-class>org.jasig.cas.client.util.HttpServletRequestWrapperFilterfilter-class>**
4. **filter>**
5. **<filter-mapping>**
6. **<filter-name>CAS HttpServletRequest Wrapper Filterfilter-name>**
7. **<url-pattern>/*url-pattern>**
8. **filter-mapping>**

封装request, 支持getUserPrincipal等方法。

6) 存放Assertion到ThreadLocal中

[html]view plaincopy


在CODE上查看代码片

派生到我的代码片

1. **<filter>**
2. **<filter-name>CAS Assertion Thread Local Filterfilter-name>**
3. **<filter-class>org.jasig.cas.client.util.AssertionThreadLocalFilterfilter-class>**
4. **filter>**
5. **<filter-mapping>**
6. **<filter-name>CAS Assertion Thread Local Filterfilter-name>**
7. **<url-pattern>/*url-pattern>**
8. **filter-mapping>**

7) 禁用浏览器缓存

[html]view plaincopy


在CODE上查看代码片

派生到我的代码片

1. **<filter>**
2. **<filter-name>NoCache Filterfilter-name>**
3. **<filter-class>com.yonyou.mcloud.cas.client.authentication.NoCacheFilterfilter-class>**
4. **filter>**
5. **<filter-mapping>**

6. <filter-name>NoCache Filterfilter-name>

7. <url-pattern>/*url-pattern>

8. filter-mapping>

8)CAS Client向CAS Server进行ticket验证

[html]view plaincopy

 在CODE上查看代码片
 派生到我的代码片

1. <filter>

2. <filter-name>CAS Validation Filterfilter-name>

3. <filter-class>org.jasig.cas.client.validation.Cas20ProxyReceivingTicketValidationFilterfilter-class>

4. <init-param>

5. <param-name>casServerUrlPrefixparam-name>

6. <param-value>https://dev.yonyou.com:443/sso-serverparam-value>

7. init-param>

8. <init-param>

9. <param-name>serverNameparam-name>

10. <param-value>http://10.1.215.40:80param-value>

11. init-param>

12. <init-param>

13. <param-name>proxyCallbackUrlparam-name>

14. <param-value>https://dev.yonyou.com:443/business/proxyCallbackparam-value>

15. init-param>

16. <init-param>

17. <param-name>proxyReceptorUrlparam-name>

18. <param-value>/proxyCallbackparam-value>

19. init-param>

20. <init-param>

21. <param-name>proxyGrantingTicketStorageClassparam-name>

22. <param-

value>com.yonyou.mcloud.cas.client.proxy.MemcachedBackedProxyGrantingTicketStorageImplparam-value>

23. init-param>

24. <init-param>

25. <param-name>encodingparam-name>

26. <param-value>UTF-8param-value>

27. init-param>

28. filter>

29. <filter-mapping>

30. <filter-name>CAS Validation Filterfilter-name>

31. <url-pattern>/proxyCallbackurl-pattern>

32. filter-mapping>

33. <filter-mapping>

34. <filter-name>CAS Validation Filterfilter-name>

35. <url-pattern>/*url-pattern>

36. filter-mapping>

9.

[html]view plaincopy


在CODE上查看代码片

派生到我的代码片

```
1. <servlet>
2. <servlet-name>businessservlet-name>
3. <servlet-class>org.springframework.web.servlet.DispatcherServletservlet-class>
4. <init-param>
5. <param-name>publishContextparam-name>
6. <param-value>>falseparam-value>
7. </init-param>
8. <load-on-startup>1load-on-startup>
9. </servlet>
10. <servlet>
11. <servlet-name>LogOutServletservlet-name>
12. <servlet-class>com.yonyou.mcloud.cas.web.servlet.LogOutServletservlet-class>
13. <init-param>
14. <param-name>serverLogoutUrlparam-name>
15. <param-value>https://dev.yonyou.com:443/sso-server/logoutparam-value>
16. </init-param>
17. <init-param>
18. <param-name>serverNameparam-name>
19. <param-value>http://10.1.215.40:80/business/param-value>
20. </init-param>
21. </servlet>
22. <servlet-mapping>
23. <servlet-name>LogOutServletservlet-name>
24. <url-pattern>/logouturl-pattern>
25. </servlet-mapping>
26. <servlet-mapping>
27. <servlet-name>businessservlet-name>
28. <url-pattern>/url-pattern>
29. </servlet-mapping>
```

1. Servlet介绍:

Servlet通常称为服务器端小程序，是运行在服务器端的程序，用于处理及响应客户的请求。Servlet是个特殊的java类，继承于**HttpServlet**。客户端通常只有GET和POST两种请求方式，Servlet为了响应这两种请求，必须重写doGet()和doPost()方法。大部分时候，Servlet对于所有的请求响应都是完全一样的，此时**只需要重写service()方法即可响应客户端的所有请求**。

另外HttpServlet有两个方法

- init(ServletConfig config): 创建Servlet实例时，调用该方法的**初始化Servlet资源**。
- destroy(): 销毁Servlet实例时，自动调用该方法的回收资源。

通常无需重写init()和destroy()两个方法，除非需要在初始化Servlet时，完成某些**资源初始化的方法**，才考虑重写init()方法，**如果重写了init()方法，应在重写该方法的第一行调用super.init(config)**，该方法将调

用HttpServlet的init()方法。如果需要在销毁Servlet之前，先完成某些资源的回收，比如关闭数据库连接，才需要重写destory方法()。

Servlet的生命周期：

创建Servlet实例有两个时机：

- 客户端第一次请求某个Servlet时，系统创建该Servlet的实例，大部分Servlet都是这种Servlet。
- Web应用启动时立即创建Servlet实例，即load-on-start Servlet。

每个Servlet的运行都遵循如下生命周期：

1. 创建Servlet实例。
2. Web容器调用Servlet的init()方法，对Servlet进行初始化。
3. Servlet初始化后，将一直存在于容器中，用于响应客户端请求，如果客户端发送GET请求，容器调用Servlet的doGet()方法处理并响应请求；如果客户端发送POST请求，容器调用Servlet的doPost()方法处理并响应请求。或者统一使用service()方法处理来响应用户请求。
4. Web容器决定销毁Servlet时，先调用Servlet的destory()方法，通常在关闭Web应用时销毁Servlet实例。

1. Servlet配置：

为了让Servlet能响应用户请求，还必须将Servlet配置在web应用中，配置Servlet需要修改web.xml文件。

从Servlet3.0开始，配置Servlet有两种方式：

- 在Servlet类中使用@WebServlet Annotation进行配置。
- 在web.xml文件中进行配置。

我们用web.xml文件来配置Servlet，需要配置和。

用来声明一个Servlet。、和元素的用法和的用法相同。元素与元素具有相同的元素描述符，可以使用子元素将初始化参数名和参数值传递给Servlet，访问Servlet配置参数通过ServletConfig对象来完成，ServletConfig提供如下方法：

java.lang.String.getInitParameter(java.lang.String name)：用于获取初始化参数

ServletConfig获取配置参数的方法和ServletContext获取配置参数的方法完全一样，只是ServletConfig是取得当前Servlet的配置参数，而ServletContext是获取整个Web应用的配置参数。

1. 、和

- ：为Servlet指定一个文本描述。
- ：为Servlet提供一个简短的名字被某些工具显示。
- ：为Servlet指定一个图标，在图形管理工具中表示该Servlet。

1. 、和元素

必须含有和，或者和。描述如下：

- 用来定义servlet的名称，该名称在整个应用中必须是惟一的
- 用来指定servlet的完全限定的名称。
- 用来指定应用中JSP文件的完整路径。这个完整路径必须由/开始。

如果load-on-startup元素存在，而且也指定了jsp-file元素，则JSP文件会被重新编译成Servlet，同时产生的Servlet也被载入内存。的内容可以为空，或者是一个整数。这个值表示由Web容器载入内存的顺序。

举个例子：如果有两个Servlet元素都含有子元素，则子元素值较小的Servlet将先被加载。如果子元素值为空或

负值，则由Web容器决定什么时候加载Servlet。如果两个Servlet的子元素值相同，则由Web容器决定先加载哪一个Servlet。

1表示启动容器时，初始化Servlet。

含有和

- `name`：Servlet的名字，唯一性和一致性，与元素中声明的名字一致。
- `url-pattern`：指定相对于Servlet的URL的路径。该路径相对于web应用程序上下文的根路径。将URL模式映射到某个Servlet，即该Servlet处理的URL。

1. 加载Servlet的过程

容器的Context对象对请求路径(URL)做出处理，去掉请求URL的上下文路径后，按路径映射规则和Servlet映射路径（）做匹配，如果匹配成功，则调用这个Servlet处理请求。

1. DispatcherServlet在web.xml中的配置：

[html]view plaincopy


在CODE上查看代码片

派生到我的代码片

1. `<servlet>`
2. `<servlet-name>businessservlet-name</servlet-name>`
3. `<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>`
4. `<init-param>`
5. `<param-name>publishContext</param-name>`
6. `<param-value>false</param-value>`
7. `<init-param>`
8. `<load-on-startup>1</load-on-startup>`
9. `<servlet>`

配置Spring MVC，指定处理请求的Servlet，有两种方式：

1默认查找MVC配置文件的地址是：/WEB-INF/\${servletName}-servlet.xml

1可以通过配置修改MVC配置文件的位置，需要在配置DispatcherServlet时指定MVC配置文件的位置。

我们在平台项目两个工程中分别使用了不同的配置方式，介绍如下：

我们在business-client工程中按照默认方式查找MVC的配置文件，配置文件目录为：/WEB-INF/business-servlet.xml。工程目录结构如下所示：

```
└─ WEB-INF 9199
  └─ classes 8007
    └─ spring-configuration 8007
      └─ sessionMappingStorage.xml 7551
      └─ business-servlet.xml 9198
      └─ web.xml 9199
    └─ index.html 9014
```

我们在public-base-server工程中，通过第2种方式进行配置，把spring-servlet.xml放

到src/main/resources/config/spring-servlet.xml，则需要在配置DispatcherServlet时指定标签。具体代码如下：

[html]view plaincopy

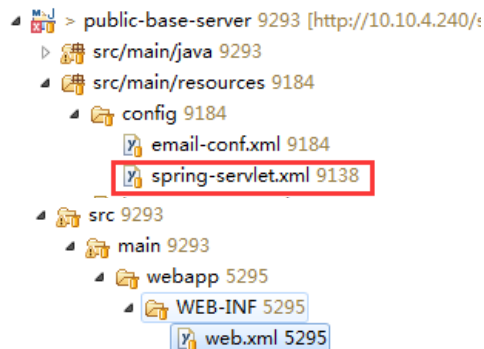

在CODE上查看代码片

派生到我的代码片

1. `<servlet>`

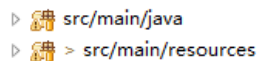
2. <servlet-name>spring</servlet-name>
3. <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
4. <init-param>
5. <param-name>publishContext</param-name>
6. <param-value>false</param-value>
7. </init-param>
8. </init-param>
9. <param-name>contextConfigLocation</param-name>
10. <param-value>classpath:config/spring-servlet.xml</param-value>
11. </init-param>
12. <load-on-startup>1</load-on-startup>
13. </servlet>

工程目录结构如下：



其中，**classpath**是web项目的类路径，可以理解为classes下面。因为无论这些配置文件放在哪，编译之后如果没有特殊情况的话都直接在classes下面。jar包的话虽然放在lib文件夹里，但实际上那些类可以直接引用，比如：**com.test.ABC**，仿佛也在classes下面一样。

在我们的工程里，经过验证，maven工程这两个

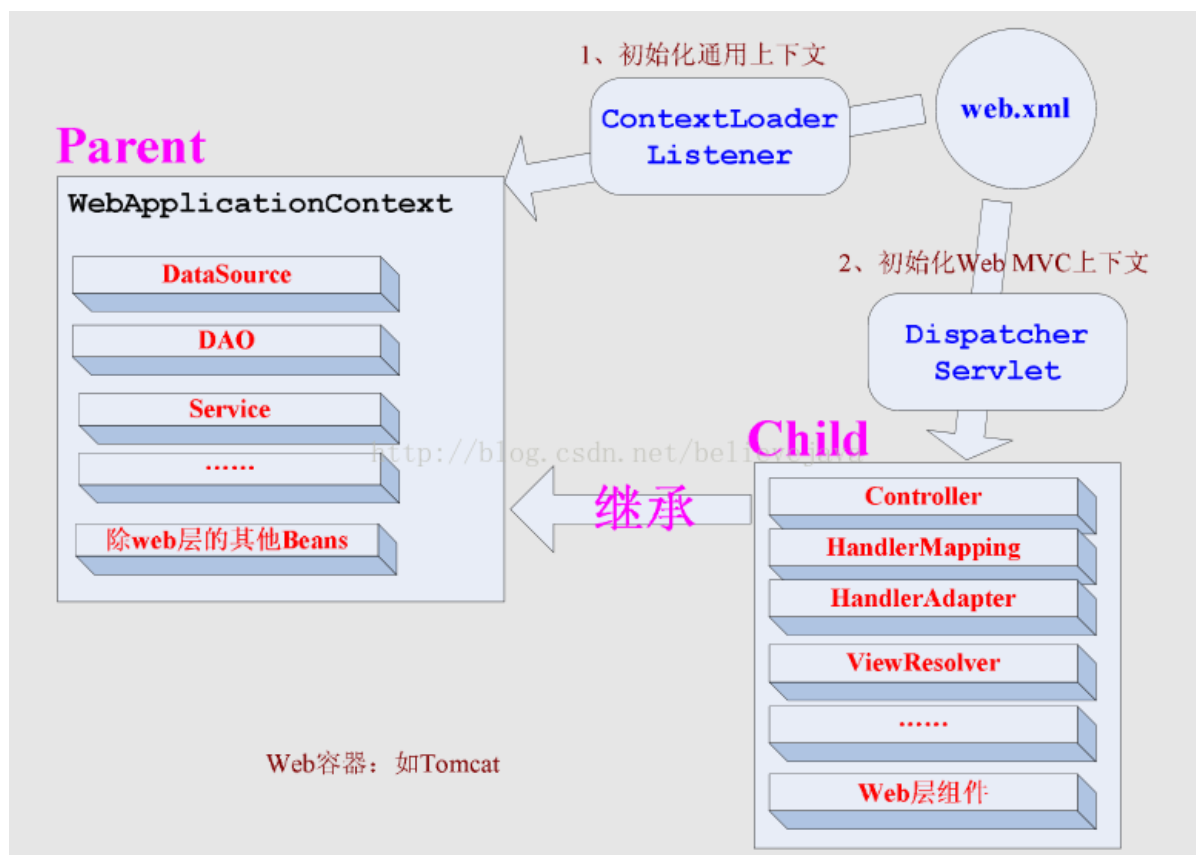


路径经过编译后生成的文件都位于classes目录下，即这两个路径相当于类路径，在下面创建config文件夹（folder），然后创建自定义的xml配置文件即可。

classpath和**classpath***区别：

同名资源存在时，**classpath**只从第一个符合条件的**classpath**中加载资源，而**classpath***会从所有的**classpath**中加载符合条件的资源。**classpath***，需要遍历所有的**classpath**，效率肯定比不上**classpath**，因此在项目设计的初期就尽量规划好资源文件所在的路径，避免使用**classpath***来加载。

1. ContextLoaderListener和DispatcherServlet初始化上下文关系和区别：



从上图可以看出，ContextLoaderListener初始化的上下文加载的Bean是对于整个应用程序共享的，一般如：DAO层、Service层Bean；DispatcherServlet初始化的上下文加载的Bean是只对Spring MVC有效的Bean，如：Controller、HandlerMapping、HandlerAdapter等，该初始化上下文只加载Web相关组件。注意：用户可以配置多个DispatcherServlet来分别处理不同的url请求，每个DispatcherServlet上下文都对应一个自己的子Spring容器，他们都拥有相同的父Spring容器（业务层，持久（dao）bean所在的容器）。

10.

[html]view plaincopy


 在CODE上查看代码片

 派生到我的代码片

1. <welcome-file-list>

2. <welcome-file>index.htmlwelcome-file>

3. welcome-file-list>

包含一个子元素，用来指定首页文件名称。元素可以包含一个或多个子元素。如果在第一个元素中没有找到指定的文件，Web容器就会尝试显示第二个，以此类推。

参考文献：

<http://wiki.metawerx.NET/wiki/Web.xml>

<http://www.cnblogs.com/konbluesky/articles/1925295.html>

http://blog.csdn.Net/sapphire_aling/article/details/6069764

<http://blog.csdn.net/zndx1xm/article/details/8711626>

<http://blog.csdn.net/zhangliao613/article/details/6289114>

<http://www.cnblogs.com/bukudekong/archive/2011/12/26/2302081.html>

http://blog.sina.com.cn/s/blog_92b93d6f0100ypp9.html

<http://blog.csdn.net/heidan2006/article/details/3075730>

[http://zhidao.baidu.com/link?url=vBOBj5f2D1Zx3wSUJo-](http://zhidao.baidu.com/link?url=vBOBj5f2D1Zx3wSUJo-XphWrG6f7QPmfzk0UtS9Xk7p1SG_0deCkiH6dT6eyH0-Pa6p4hLTefvY709d_OM0Gua)

[XphWrG6f7QPmfzk0UtS9Xk7p1SG_0deCkiH6dT6eyH0-Pa6p4hLTefvY709d_OM0Gua](http://zhidao.baidu.com/link?url=vBOBj5f2D1Zx3wSUJo-XphWrG6f7QPmfzk0UtS9Xk7p1SG_0deCkiH6dT6eyH0-Pa6p4hLTefvY709d_OM0Gua)

<http://www.blogjava.net/dashi99/archive/2008/12/30/249207.html>
<http://uule.iteye.com/blog/2051817>
<http://blog.csdn.net/javaer617/article/details/6432654>
<http://blog.csdn.net/seng3018/article/details/6758860>
[http://groups.tianya.cn/tribe/showArticle.jsp?
groupId=185385&articleId=2704257273118260804105385](http://groups.tianya.cn/tribe/showArticle.jsp?groupId=185385&articleId=2704257273118260804105385)
http://blog.csdn.net/qfs_v/article/details/2557128
http://www.blogjava.net/fancydeepin/archive/2013/03/30/java-ee_web-xml.html
[http://wenku.baidu.com/link?url=P30DokIynD5zzRU2dtdkQhEwsHi-
REKuBiHa_dK60bA6pQwggvX2mo9y9Mbb1tkYcsiRCaHBf-c
4ZgIG5P0mbbcR0_0xDJUaW15n300xJrq](http://wenku.baidu.com/link?url=P30DokIynD5zzRU2dtdkQhEwsHi-REKuBiHa_dK60bA6pQwggvX2mo9y9Mbb1tkYcsiRCaHBf-c4ZgIG5P0mbbcR0_0xDJUaW15n300xJrq)
<http://fyq891014.blog.163.com/blog/static/200740191201233052531278/>
http://blog.163.com/sir_876/blog/static/11705223201111544523333/
<http://www.guoweimei.com/archives/797>
<http://www.open-open.com/lib/view/open1402751642806.html>
<http://sishuok.com/forum/blogPost/list/5188.html;jsessionId=EBC2151611BEB99BDF390C5CADBA693A>
<http://www.micmiu.com/j2ee/spring/spring-classpath-start/>
<http://elf8848.iteye.com/blog/2008595>
http://blog.csdn.net/arvin_qx/article/details/6829873
轻量级javaEE企业应用实战（第3版） ---李刚