链滴

Servlet 3.0 的新特性

作者：xunxiake

原文链接：https://ld246.com/article/1482381939781

来源网站：链滴

Servlet 3.0 作为 Java EE 6 规范体系中一员，随着 Java EE 6 规范一起发布。该版本在前一版本 Servlet 2.5）的基础上提供了若干新特性用于简化 Web 应用的开发和部署。
TOMCAT7 支持 Servlet 3.0
新增的注解支持
Servlet 3.0 的部署描述文件 web.xml 的顶层标签 <web-app> 一个 metadata-complete 属性，该属性指定当前的部署描述文件是否是完全的。如果设置为 true，容器在部署时将只依赖部署描述文件，忽略所有的注解（同时也会跳过 web-fragment.xml 的扫描亦即禁用可插性支持，具体请看后文关于 可插性支持 的讲解）；如果不配置该属性，或者将其设置为 alse，则表示启用注解支持（和可插性支持）。

@WebServlet
@WebServlet 用于将一个类声明为 Servlet，该注解将会在部署时被容器处理，容器将根据具的属性配置将相应的类部署为 Servlet。该注解具有下表给出的一些常用属性（以下所有属性均为可属性，但是 vlaue 或者 urlPatterns 通常是必需的，且二者不能共存，如果同时指定，通常是忽略 val e 的取值）：
表 1. @WebServlet 主要属性列表
属性名 类型 描述
name String 指定 Servl t 的 name 属性，等价于 <servlet-name>。如果没有显式指定，则该 Servlet 的取值即为类的限定名。
value String[] 该属性等价于 urlPatterns 属性。两个属性不同时使用。
urlPatterns String[] 指定一组 Servlet 的 URL 匹配模式。价于 <url-pattern> 标签。
loadOnStartup int 指定 Servlet 的载顺序，等价于 <load-on-startup> 标签。
initParams WebInitParam[指定一组 Servlet 初始化参数，等价于 <init-param> 标签。
asyncSup orted boolean 声明 Servlet 是否支持异步操作模式，等价于 <async-supported gt; 标签。
description String 该 Servlet 的描述信息，等价于 <des ription> 标签。
displayName String 该 Servlet 的显示名，通常合工具使用，等价于 <display-name> 标签。
下面是一个简单的示例：
<pre class="brush: java">@WebServlet(urlPatterns = {"/simple"}, asyncSupported = true,
loadOnStartup = -1, name = "SimpleServlet", displayName = "ss",
initParams = {@WebInitParam(name = "username", value = "tom")}
)
public class SimpleServlet extends HttpServlet{ … }
 </pre>
如此配置之后，就可以不必在 web.xml 中配置相应的 <servlet> 和 <servlet-mapping gt; 元素了，容器会在部署时根据指定的属性将该类发布为 Servlet。它的等价的 web.xml 配置形式下：
<pre class="brush: xml"><servlet>
    <display-name>ss</display-name>
    <servlet-name>SimpleServlet</servlet-name>
    <servlet-class>footmark.servlet.SimpleServlet</servlet-class>
    <load-on-startup>-1</load-on-startup>
    <async-supported>true</async-supported>
    <init-param>
        <param-name>username</param-name>
        <param-value>tom</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>SimpleServlet</servlet-name>
    <url-pattern>/simple</url-pattern>
</servlet-mapping></pre>

@WebInitParam

该注解通常不单独使用，而是配合 @WebServlet 或者 @WebFilter 使用。它的作用是为 Servlet 或者过滤器指定初始化参数，这等价于 web.xml 中 <servlet> 和 <filter> 的 <init-param> 子标签。@WebInitParam 具有下表给出的一些常用属性：
表 2. @WebInitParam 的常用属性
属性名 类型 是否可选 描述
name String 否 指定参数的名字，等价于 <param-name>。
value String 否 指定参数的值，等价于 <param-value>。
description String 是 关于参数的描述，等价于 <description>

@WebFilter
@WebFilter 用于将一个类声明为过滤器，该注解将会在部署时被容器处理，容器将根据具体的性配置将相应的类部署为过滤器。该注解具有下表给出的一些常用属性 (以下所有属性均为可选属性但是 value、urlPatterns、servletNames 三者必需至少包含一个，且 value 和 urlPatterns 不能共，如果同时指定，通常忽略 value 的取值)：
表 3. @WebFilter 的常用属性
属性名 类型 描述
filterName String 指定滤器的 name 属性，等价于 <filter-name>
value String[] 该属性等价于 urlPatterns 属性。但是两者不应该同时使用。
urlPatterns String[] 指定一组过滤器的 URL 匹配模式。等价于 <url-pattern> 标签。
servletNames String[] 指定过滤器将应用于哪些 Servlet。取值是 @WebServlet 中的 name 属性的取值，或者是 web.xml 中 <servlet-name> 的取值。
dispatcherTypes DispatcherType 指定过滤器的转发模式。具体取值包括：
ASYNC、ERROR FORWARD、INCLUDE、REQUEST。
initParams WebInitParam[] 定一组过滤器初始化参数，等价于 <init-param> 标签。
asyncSupported boolean 声明过滤器是否支持异步操作模式，等价于 <async-supported> 标签。
description String 该过滤器的描述信息，等价于 <description> 标。
displayName String 该过滤器的显示名，通常配合工具使用，等价 <display-name> 标签。
下面是一个简单的示例：
<pre class="brush: java">@WebFilter(servletNames = {"SimpleServlet"},filterName="SimpleFiler")
public class LessThanSixFilter implements Filter{...} </pre>

如此配置之后，就可以不必在 web.xml 中配置相应的 <filter> 和 <filter-mapping> 元素了，容器会在部署时根据指定的属性将该类发布为过滤器。它等价的 web.xml 中的配置式为：
<pre class="brush: java"><filter>
    <filter-name>SimpleFilter</filter-name>
    <filter-class>xxx</filter-class>
</filter>
<filter-mapping>
    <filter-name>SimpleFilter</filter-name>
    <servlet-name>SimpleServlet</servlet-name>
</filter-mapping></pre>

@WebListener
该注解用于将类声明为监听器，被 @WebListener 标注的类必须实现以下至少一个接口：
ServletContextListener
ServletContextAttributeListener
ServletequestListener
ServletRequestAttributeListener
HttpSessionListene
HttpSessionAttributeListener
该注解使用非常简单，其属性如下：</>
表 4. @WebListener 的常用属性
属性名 类型 是否可选 描述
value String 是 该监听器的描述信息。
一个简单示例如下：

<pre class="brush: java">@WebListener("This is only a demo listener")
public class SimpleListener implements ServletContextListener{...} </pre>
如此，则不需要在 web.xml 中配置 <listener> 标签了。它等价的 web.xml 中的配置形式下：
<pre class="brush: xml"><listener>
    <listener-class>footmark.servlet.SimpleListener</listener-class>
</listener> </pre>

@MultipartConfig
该注解主要是为了辅助 Servlet 3.0 中 HttpServletRequest 提供的对上传文件的支持。该注解注在 Servlet 上面，以表示该 Servlet 希望处理的请求的 MIME 类型是 multipart/form-data。另外它还提供了若干属性用于简化对上传文件的处理。具体如下：
表 5. @MultipartConfig 的常用属性
属性名 类型 是否可选 描述
fileSizeThreshold int 是 当数据量大该值时，内容将被写入文件。
location String 是 存放生成的文件地址
maxFileSize long 是 允许上传的文件最大值。默认值为 -1，表示没限制。
maxRequestSize long 是 针对该 multipart/form-data 请求最大数量，默认值为 -1，表示没有限制。

异步处理支持
配置方式如下所示：
<pre class="brush: xml"><servlet>
    <servlet-name>DemoServlet</servlet-name>
    <servlet-class>footmark.servlet.Demo Servlet</servlet-class>
    <async-supported>true</async-supported>
</servlet> </pre>

Servlet 3.0 提供的 @WebServlet 和 @WebFilter 进行 Servlet 或过滤器配置的情况，这两个解都提供了 asyncSupported 属性，默认该属性的取值为 false，要启用异步处理支持，只需将该属设置为 true 即可。以 @WebFilter 为例，其配置方式如下所示：
<pre class="brush: java">@WebFilter(urlPatterns = "/demo",asyncSupported = true
)
public class DemoFilter implements Filter{...}
 </pre>
一个简单的模拟异步处理的 Servlet 示例如下：
<pre class="brush: java">@WebServlet(urlPatterns = "/demo", asyncSupported = true)
public class AsyncDemoServlet extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws IOException, ServletException {
        resp.setContentType("text/html;charset=UTF-8");
        PrintWriter out = resp.getWriter();
        out.println("进入Servlet的时间：" + new Date() + ".");
        out.flush();

    //在子线程中执行业务调用，并由其负责输出响应，主线程退出
    AsyncContext ctx = req.startAsync();
    new Thread(new Executor(ctx)).start();

    out.println("结束Servlet的时间：" + new Date() + ".");
    out.flush();
}
</pre>

```
}

public class Executor implements Runnable {
private AsyncContext ctx = null;
public Executor(AsyncContext ctx){
this.ctx = ctx;
}


public void run(){
   try {
      //等待十秒钟，以模拟业务方法的执行
      Thread.sleep(10000);
      PrintWriter out = ctx.getResponse().getWriter();
      out.println("业务处理完毕的时间：" + new Date() + ".");
      out.flush();
      ctx.complete();
   } catch (Exception e) {
      e.printStackTrace();
   }
}

}</pre>

<p> </p>
<p>Servlet 3.0 还为异步处理提供了一个监听器，使用 AsyncListener 接口表示。它可以监控如下种事件：</p>
<p>异步线程开始时，调用 AsyncListener 的 onStartAsync(AsyncEvent event) 方法； <br >异步线程出错时，调用 AsyncListener 的 onError(AsyncEvent event) 方法； <br />异步程执行超时，则调用 AsyncListener 的 onTimeout(AsyncEvent event) 方法； <br />异步行完毕时，调用 AsyncListener 的 onComplete(AsyncEvent event) 方法； <br />要注册个 AsyncListener，只需将准备好的 AsyncListener 对象传递给 AsyncContext 对象的 addListener() 方法即可，如下所示：</p>
<pre class="brush: java">AsyncContext ctx = req.startAsync();
ctx.addListener(new AsyncListener() {
   public void onComplete(AsyncEvent asyncEvent) throws IOException {
      // 做一些清理工作或者其他
   }
   ...
}); </pre>
<p> </p>
<p><span style="font-size: medium;"><strong>可插性支持</strong></span></p>
<p><span>使用该特性，现在我们可以在不修改已有 Web 应用的前提下，只需将按照一定格式打的 JAR 包放到 WEB-INF/lib 目录下，即可实现新功能的扩充，不需要额外的配置。</span></p>
<p>Servlet 3.0 引入了称之为&ldquo;Web 模块部署描述符片段&rdquo;的 web-fragment.xml 部描述文件，该文件必须存放在 JAR 文件的 META-INF 目录下，该部署描述文件可以包含一切可以在eb.xml 中定义的内容。JAR 包通常放在 WEB-INF/lib 目录下，除此之外，所有该模块使用的资源，括 class 文件、配置文件等，只需要能够被容器的类加载器链加载的路径上，比如 classes 目录等。<p>
<p>现在，为一个 Web 应用增加一个 Servlet 配置有如下三种方式（过滤器、监听器与 Servlet 三的配置都是等价的，故在此以 Servlet 配置为例进行讲述，过滤器和监听器具有与之非常类似的特性：</p>
<p>编写一个类继承自 HttpServlet，将该类放在 classes 目录下的对应包结构中，修改 web.xml，
```

其中增加一个 Servlet 声明。这是最原始的方式；
编写一个类继承自 HttpServlet，并在该类上使用 @WebServlet 注解将该类声明为 Servlet，将该类放在 classes 目录下的对应包结构，无需修改 web.xml 文件。
编写一个类继承自 HttpServlet，将该类打成 JAR 包，并在 JAR 包的 META-INF 目录下放置一个 web-fragment.xml 文件，该文件中声明了相应的 Servlet 置。web-fragment.xml 文件示例如下：
<pre class="brush: xml"><?xml version="1.0" encoding="UTF-8"?>
<web-fragment
    xmlns=http://java.sun.com/xml/ns/javaee
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="3.0"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-fragment_3_0.xsd"
    metadata-complete="true">
    <servlet>
        <servlet-name>fragment</servlet-name>
        <servlet-class>footmark.servlet.FragmentServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>fragment</servlet-name>
        <url-pattern>/fragment</url-pattern>
    </servlet-mapping>
</web-fragment></pre>

从上面的示例可以看出，web-fragment.xml 与 web.xml 除了在头部声明的 XSD 引用不同之外其主体配置与 web.xml 是完全一致的。
由于一个 Web 应用中可以出现多个 web-fragment.xml 声明文件，加上一个 web.xml 文件，载顺序问题便成了不得不面对的问题。Servlet 规范的专家组在设计的时候已经考虑到了这个问题，定义了加载顺序的规则。
web-fragment.xml 包含了两个可选的顶层标签，<name> 和 <ordering>，如果希为当前的文件指定明确的加载顺序，通常需要使用这两个标签，<name> 主要用于标识当前的件，而 <ordering> 则用于指定先后顺序。一个简单的示例如下：
<pre class="brush: xml"><web-fragment...>
    <name>FragmentA</name>
    <ordering>
        <after>
            <name>FragmentB</name>
            <name>FragmentC</name>
        </after>
    <before>
        <others/>
    </before>
    </ordering>
    ...
</web-fragment></pre>

如上所示， <name> 标签的取值通常是被其它 web-fragment.xml 文件在定义先后顺序引用的，在当前文件中一般用不着，它起着标识当前文件的作用。
在 <ordering> 标签内部，我们可以定义当前 web-fragment.xml 文件与其他文件的相对置关系，这主要通过 <ordering> 的 <after> 和 <before> 子标签来实现的。在这个子标签内部可以通过 <name> 标签来指定相对应的文件。比如：
<pre class="brush: xml"><after>
    <name>FragmentB</name>
    <name>FragmentC</name>
</after> </pre>

以上片段则表示当前文件必须在 FragmentB 和 FragmentC 之后解析。<before> 的使用此相同，它所表示的是当前文件必须早于 <before> 标签里所列出的 web-fragment.xml 文件
除了将所比较的文件通过 <name> 在 <after> 和 <begin> 中列出之外，Servlet 还提供了一个简化的标签 <others/>。它表示除了当前文件之外的其他所有的 web-fragment.xml 文件。该标签的优先级要低于使用 <name> 明确指定的相对位置关系。

ServletContext 的性能增强
除了以上的新特性之外，ServletContext 对象的功能在新版本中也得到了增强。现在，该对象持在运行时动态部署 Servlet、过滤器、监听器，以及为 Servlet 和过滤器增加 URL 映射等。以 Servlet 为例，过滤器与监听器与之类似。ServletContext 为动态配置 Servlet 增加了如下方法：
<pre class="brush: java">ServletRegistration.Dynamic addServlet(String servletName,Class&lt? extends Servlet> servletClass)
ServletRegistration.Dynamic addServlet(String servletName, Servlet servlet)
ServletRegistration.Dynamic addServlet(String servletName, String className)
<T extends Servlet> T createServlet(Class<T> clazz)
ServletRegistration getServletRegistration(String servletName)
Map<String,? extends ServletRegistration> getServletRegistrations() </pre>

其中前三个方法的作用是相同的，只是参数类型不同而已；通过 createServlet() 方法创的 Servlet，通常需要做一些自定义的配置，然后使用 addServlet() 方法来将其动态注册为一个可以于服务的 Servlet。两个 getServletRegistration() 方法主要用于动态为 Servlet 增加映射信息，这等于在 web.xml(抑或 web-fragment.xml) 中使用 <servlet-mapping> 标签为存在的 Servlet 加映射信息。
以上 ServletContext 新增的方法要么是在 ServletContextListener 的 contexInitialized 方法调用，要么是在 ServletContainerInitializer 的 onStartup() 方法中调用。
ServletContainerInitializer 也是 Servlet 3.0 新增的一个接口，容器在启动时使用 JAR 服务 API(AR Service API) 来发现 ServletContainerInitializer 的实现类，并且容器将 WEB-INF/lib 目录下 JAR 包中的类都交给该类的 onStartup() 方法处理，我们通常需要在该实现类上使用 @HandlesTypes 注来指定希望被处理的类，过滤掉不希望给 onStartup() 处理的类。

HttpServletRequest 对文件上传的支持
此前，对于处理上传文件的操作一直是让开发者头疼的问题，因为 Servlet 本身没有对此提供直的支持，需要使用第三方框架来实现，而且使用起来也不够简单。如今这都成为了历史，Servlet 3.0 经提供了这个功能，而且使用也非常简单。为此，HttpServletRequest 提供了两个方法用于从请求解析出上传的文件：
Part getPart(String name)
Collection<Part> getParts()

前者用于获取请求中给定 name 的文件，后者用于获取所有的文件。每一个文件用一个 javax.servlet.http.Part 对象来表示。该接口提供了处理文件的简易方法，比如 write()、delete() 等。此，结合 HttpServletRequest 和 Part 来保存上传的文件变得非常简单，如下所示：
<pre class="brush: java">Part photo = request.getPart("photo");
photo.write("/tmp/photo.jpg");
// 可以将两行代码简化为 request.getPart("photo").write("/tmp/photo.jpg") 一行。</pre>

另外，开发者可以配合前面提到的 @MultipartConfig 注解来对上传操作进行一些自定义的配置比如限制上传文件的大小，以及保存文件的路径等。其用法非常简单，故不在此赘述了。
需要注意的是，如果请求的 MIME 类型不是 multipart/form-data，则不能使用上面的两个方，否则将抛异常。

总结
Servlet 3.0 的众多新特性使得 Servlet 开发变得更加简单，尤其是异步处理特性和可插性支持的现，必将对现有的 MVC 框架产生深远影响。虽然我们通常不会自己去用 Servlet 编写控制层代码，是也许在下一个版本的 Struts 中，您就能切实感受到这些新特性带来的实质性改变。

<a href="http://www.ibm.com/developerworks/cn/java/j-lo-servlet30/index.html?ca=drs
cn-0423" title="【原文地址】" target="_blank">【原文地址】

原文链接：Servlet 3.0 的新特性