

## 08 | 整合IoC和MVC：如何在Web环境中启动IoC容器？

2023-03-29 郭屹 来自北京

《手把手带你写一个MiniSpring》



你好，我是郭屹。

通过上节课的工作，我们就初步实现了一个原始的 MVC 框架，并引入了 @RequestMapping 注解，还通过对指定的包进行全局扫描来简化 XML 文件配置。但是这个 MVC 框架是独立运行的，它跟我们之前实现的 IoC 容器还没有什么关系。


那么这节课，我们就把前面实现的 IoC 容器与 MVC 结合在一起，使 MVC 的 Controller 可以引用容器中的 Bean，这样整合成一个大的容器。

### Servlet 服务器启动过程

IoC 容器是一个自我实现的服务器，MVC 是要符合 Web 规范的，不能自己想怎么来就怎么来。为了融合二者，我们有必要了解一下 Web 规范的内容。在 Servlet 规范中，服务器启动的时候，会根据 web.xml 文件来配置。下面我们花点时间详细介绍一下这个配置文件。

这个 web.xml 文件是 Java 的 Servlet 规范中规定的，它里面声明了一个 Web 应用全部的配置信息。按照规定，每个 Java Web 应用都必须包含一个 web.xml 文件，且必须放在 WEB-INF 路径下。它的顶层根是 web-app，指定命名空间和 schema 规定。通常，我们会在 web.xml 中配置 context-param、Listener、Filter 和 Servlet 等元素。

下面是常见元素的说明。

 复制代码

```
1 <display-name></display-name>
2 声明WEB应用的名字
3 <description></description>
4 声明WEB应用的描述信息
5 <context-param></context-param>
6 声明应用全局的初始化参数。
7 <listener></listener>
8 声明监听器，它在建立、修改和删除会话或servlet环境时得到事件通知。
9 <filter></filter>
10 声明一个实现javax.servlet.Filter接口的类。
11 <filter-mapping></filter-mapping>
12 声明过滤器的拦截路径。
13 <servlet></servlet>
14 声明servlet类。
15 <servlet-mapping></servlet-mapping>
16 声明servlet的访问路径，试一个方便访问的URL。
17 <session-config></session-config>
18 session有关的配置，超时值。
19 <error-page></error-page>
20 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。
```

当 Servlet 服务器如 Tomcat 启动的时候，要遵守下面的时序。

1. 在启动 Web 项目时，Tomcat 会读取 web.xml 中的 context-param 节点，获取这个 Web 应用的全局参数。
2. Tomcat 创建一个 ServletContext 实例，是全局有效的。


3. 将 context-param 的参数转换为键值对，存储在 ServletContext 里。
4. 创建 listener 中定义的监听类的实例，按照规定 Listener 要继承自 ServletContextListener。监听器初始化方法是 contextInitialized(ServletContextEvent event)。初始化方法中可以通过 event.getServletContext().getInitParameter( "name" ) 方法获得上下文环境中的键值对。
5. 当 Tomcat 完成启动，也就是 contextInitialized 方法完成后，再对 Filter 过滤器进行初始化。
6. servlet 初始化：有一个参数 load-on-startup，它为正数的值越小优先级越高，会自动启动，如果为负数或未指定这个参数，会在 servlet 被调用时再进行初始化。init-param 是一个 servlet 整个范围之内有效的参数，在 servlet 类的 init() 方法中通过 this.getInitParameter("param1") 方法获得。

规范中规定的这个时序，就是我们整合两者的关键所在。

## Listener 初始化启动 IoC 容器

由上述服务器启动过程我们知道，我们把 web.xml 文件里定义的元素加载过程简单归总一下：先获取全局的参数 context-param 来创建上下文，之后如果配置文件里定义了 Listener，那服务器会先启动它们，之后是 Filter，最后是 Servlet。因此我们可以利用这个时序，把容器的启动放到 Web 应用的 Listener 中。

Spring MVC 就是这么设计的，它按照这个规范，用 ContextLoaderListener 来启动容器。我们也模仿它同样来实现这样一个 Listener。

 复制代码


```
1 package com.minis.web;
2
3 import javax.servlet.ServletContext;
4 import javax.servlet.ServletContextEvent;
5 import javax.servlet.ServletContextListener;
6
7 public class ContextLoaderListener implements ServletContextListener {
8     public static final String CONFIG_LOCATION_PARAM = "contextConfigLocation";
9     private WebApplicationContext context;
```

```

10
11 public ContextLoaderListener() {
12 }
13 public ContextLoaderListener(WebApplicationContext context) {
14     this.context = context;
15 }
16 @Override
17 public void contextDestroyed(ServletContextEvent event) {
18 }
19 @Override
20 public void contextInitialized(ServletContextEvent event) {
21     initWebApplicationContext(event.getServletContext());
22 }
23 private void initWebApplicationContext(ServletContext servletContext) {
24     String sContextLocation = servletContext.getInitParameter(CONFIG_LOCATION_PARAM);
25     WebApplicationContext wac = new AnnotationConfigWebApplicationContext(sContextLocation);
26     wac.setServletContext(servletContext);
27     this.context = wac;
28     servletContext.setAttribute(WebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE, this.context);
29 }
30 }

```

ContextLoaderListener 这个类里，先声明了一个常量 CONFIG\_LOCATION\_PARAM，它的默认值是 contextConfigLocation，这是代表配置文件路径的一个变量，也就是 IoC 容器的配置文件。这也就意味着，Listener 期望 web.xml 里有一个参数用来配置文件路径。我们可以看一下 web.xml 文件。

 复制代码


```

1 <context-param>
2     <param-name>contextConfigLocation</param-name>
3     <param-value>applicationContext.xml</param-value>
4 </context-param>
5 <listener>
6     <listener-class>
7         com.minis.web.ContextLoaderListener
8     </listener-class>
9 </listener>

```

上面这个文件，定义了这个 Listener，还定义了全局参数指定配置文件路径。

ContextLoaderListener 这个类里还定义了 WebApplicationContext 对象，目前还不存在这个类。但通过名字可以知道，WebApplicationContext 是一个上下文接口，应用在 Web 项目里。我们看看如何定义 WebApplicationContext。

 复制代码

```
1 package com.minis.web;
2
3 import javax.servlet.ServletContext;
4 import com.minis.context.ApplicationContext;
5
6 public interface WebApplicationContext extends ApplicationContext {
7     String ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE = WebApplicationContext.class.get
8
9     ServletContext getServletContext();
10    void setServletContext(ServletContext servletContext);
11 }
```

可以看出，这个上下文接口指向了 Servlet 容器本身的上下文 ServletContext。

接下来我们继续完善 ContextLoaderListener 这个类，在初始化的过程中初始化 WebApplicationContext，并把这个上下文放到 servletContext 的 Attribute 某个属性里面。


 复制代码

```
1 public void contextInitialized(ServletContextEvent event) {
2     initWebApplicationContext(event.getServletContext());
3 }
4 private void initWebApplicationContext(ServletContext servletContext) {
5     String sContextLocation =
6     servletContext.getInitParameter(CONFIG_LOCATION_PARAM);
7     WebApplicationContext wac = new
8     AnnotationConfigWebApplicationContext(sContextLocation);
9     wac.setServletContext(servletContext);
10    this.context = wac;
11    servletContext.setAttribute(WebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT
```

在这段代码中，通过配置文件参数从 web.xml 中得到配置文件路径，如 applicationContext.xml，然后用这个配置文件创建了

AnnotationConfigWebApplicationContext 这一对象，我们叫 WAC，这就成了新的上下文。然后调用 `ServletContext.setAttribute()` 方法，按照默认的属性值将 WAC 设置到 `ServletContext` 里。这样，AnnotationConfigWebApplicationContext 和 `ServletContext` 就能够互相引用了，很方便。


而这个 AnnotationConfigWebApplicationContext 又是什么呢？我们看下它的定义。

 复制代码

```
1 package com.minis.web;
2
3 import javax.servlet.ServletContext;
4 import com.minis.context.ClassPathXmlApplicationContext;
5
6 public class AnnotationConfigWebApplicationContext
7     extends ClassPathXmlApplicationContext implements WebApplicationContext
8     private ServletContext servletContext;
9
10    public AnnotationConfigWebApplicationContext(String fileName) {
11        super(fileName);
12    }
13    @Override
14    public ServletContext getServletContext() {
15        return this.servletContext;
16    }
17    @Override
18    public void setServletContext(ServletContext servletContext) {
19        this.servletContext = servletContext;
20    }
21 }
```

由 AnnotationConfigWebApplicationContext 的继承关系可看出，该类其实质就是我们 IoC 容器中的 ClassPathXmlApplicationContext，只是在此基础上增加了 `ServletContext` 的属性，这样就成了一个适用于 Web 场景的上下文。

我们在这个过程中用到了一个配置文件 `applicationContext.xml`，它是由定义在 `web.xml` 里的一个参数指明的。


 复制代码

```
1 <context-param>
```



```
2     <param-name>contextConfigLocation</param-name>
3     <param-value>applicationContext.xml</param-value>
4 </context-param>
```

这个配置文件就是我们现在的 IoC 容器的配置文件，主要作用是声明 Bean，如：


 复制代码

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans>
3     <bean id="bbs" class="com.test.service.BaseBaseService">
4         <property type="com.test.service.AServiceImpl" name="as" ref="aservice"/>
5     </bean>
6     <bean id="aservice" class="com.test.service.AServiceImpl">
7         <constructor-arg type="String" name="name" value="abc"/>
8         <constructor-arg type="int" name="level" value="3"/>
9         <property type="String" name="property1" value="Someone says"/>
10        <property type="String" name="property2" value="Hello World!"/>
11        <property type="com.test.service.BaseService" name="ref1" ref="baseservic
12    </bean>
13    <bean id="baseservice" class="com.test.service.BaseService">
14    </bean>
15 </beans>
```

回顾一下，现在完整的过程是：当 Servlet 服务器启动时，Listener 会优先启动，读配置文件路径，启动过程中初始化上下文，然后启动 IoC 容器，这个容器通过 refresh() 方法加载所管理的 Bean 对象。这样就实现了 Tomcat 启动的时候同时启动 IoC 容器。


## 改造 DispatcherServlet，关联 WAC

好了，到了这一步，IoC 容器启动了，我们回来再讨论 MVC 这边的事情。我们已经知道，在服务器启动的过程中，会注册 Web 应用上下文，也就是 WAC。这样方便我们通过属性拿到启动时的 WebApplicationContext。

 复制代码

```
1 this.webApplicationContext = (WebApplicationContext) this.getServletContext().get
```

因此我们改造一下 DispatcherServlet 这个核心类里的 init() 方法。

 复制代码  
super.i

```
1 public void init(ServletConfig config) throws ServletException {
2     this.webApplicationContext = (WebApplicationContext)
3     this.getServletContext().getAttribute(WebApplicationContext.ROOT_WEB_APPLICATION
4     sContextConfigLocation = config.getInitParameter("contextConfigLocation");
5     URL xmlPath = null;
6     try {
7         xmlPath = this.getServletContext().getResource(sContextConfigLocation);
8     } catch (MalformedURLException e) {
9         e.printStackTrace();
10    }
11    this.packageNames = XmlScanComponentHelper.getNodeValue(xmlPath);
12 }
```

Refr

首先在 Servlet 初始化的时候，从 sevletContext 里获取属性，拿到 Listener 启动的时候注册好的 WebApplicationContext，然后拿到 Servlet 配置参数 contextConfigLocation，这个参数代表的是配置文件路径，这个时候是我们的 MVC 用到的配置文件，如 minisMVC-servlet.xml，之后再扫描路径下的包，调用 refresh() 方法加载 Bean。这样，DispatcherServlet 也就初始化完毕了。

然后是改造 initMapping() 方法，按照新的办法构建 URL 和后端程序之间的映射关系：查找使用了注解 @RequestMapping 的方法，将 URL 存放到 urlMappingNames 里，再把映射的对象存放到 mappingObjs 里，映射的方法存放到 mappingMethods 里。用这个方法取代过去解析 Bean 得到的映射，省去了 XML 文件里的手工配置。你可以看一下相关代码。

 复制代码

```
1 protected void initMapping() {
2     for (String controllerName : this.controllerNames) {
3         Class<?> clazz = this.controllerClasses.get(controllerName);
4         Method[] methods = clazz.getDeclaredMethods();
5         if (methods != null) {
6             for (Method method : methods) {
7                 boolean isRequestMapping =
8                 method.isAnnotationPresent(RequestMapping.class);
9                 if (isRequestMapping) {
10                     String methodName = method.getName();
11                     String urlMapping =
```



```

12 method.getAnnotation(RequestMapping.class).value();
13         this.urlMappingNames.add(urlMapping);
14         this.mappingObjs.put(urlMapping, obj);
15         this.mappingMethods.put(urlMapping, method);
16     }
17 }
18 }
19 }
20 }

```

最后稍微调整一下 doGet() 方法内的代码，去除不再使用的结构。

 复制代码

```

1 protected void doGet(HttpServletRequest request, HttpServletResponse response) th
2     String sPath = request.getServletPath();
3     if (!this.urlMappingNames.contains(sPath)) {
4         return;
5     }
6
7     Object obj = null;
8     Object objResult = null;
9     try {
10         Method method = this.mappingMethods.get(sPath);
11         obj = this.mappingObjs.get(sPath);
12         objResult = method.invoke(obj);
13     } catch (Exception e) {
14         e.printStackTrace();
15     }
16     response.getWriter().append(objResult.toString());
17 }

```

代码里的这个 doGet() 方法从请求中获取访问路径，按照路径和后端程序的映射关系，获取到需要调用的对象和方法，调用方法后直接把结果返回给 response。

到这里，整合了 IoC 容器的 MVC 就完成了。

## 验证

下面进行测试，我们先看一下 Tomcat 使用的 web.xml 文件配置。

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:web="http://
3   <context-param>
4     <param-name>contextConfigLocation</param-name>
5     <param-value>applicationContext.xml</param-value>
6   </context-param>
7   <listener>
8     <listener-class>
9       com.minis.web.ContextLoaderListener
10    </listener-class>
11  </listener>
12  <servlet>
13    <servlet-name>minisMVC</servlet-name>
14    <servlet-class>com.minis.web.DispatcherServlet</servlet-class>
15    <init-param>
16      <param-name>contextConfigLocation</param-name>
17      <param-value> /WEB-INF/minisMVC-servlet.xml </param-value>
18    </init-param>
19    <load-on-startup>1</load-on-startup>
20  </servlet>
21  <servlet-mapping>
22    <servlet-name>minisMVC</servlet-name>
23    <url-pattern>/</url-pattern>
24  </servlet-mapping>
25 </web-app>
```

然后是 IoC 容器使用的配置文件 applicationContext.xml。

[复制代码](#)

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans>
3   <bean id="bbs" class="com.test.service.BaseBaseService">
4     <property type="com.test.service.AServiceImpl" name="as" ref="aservice"/>
5   </bean>
6   <bean id="aservice" class="com.test.service.AServiceImpl">
7     <constructor-arg type="String" name="name" value="abc"/>
8     <constructor-arg type="int" name="level" value="3"/>
9     <property type="String" name="property1" value="Someone says"/>
10    <property type="String" name="property2" value="Hello World!"/>
11    <property type="com.test.service.BaseService" name="ref1" ref="baseservice"
12  </bean>
13  <bean id="baseservice" class="com.test.service.BaseService">
14  </bean>
15 </beans>
```

MVC 扫描的配置文件 minisMVC-servlet.xml。

[复制代码](#)

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <components>
3   <component-scan base-package="com.test"/>
4 </components>
```

最后，在 com.minis.test.HelloworldBean 内的测试方法上，增加 @RequestMapping 注解。

[复制代码](#)

```
1 package com.test;
2
3 import com.minis.web.RequestMapping;
4
5 public class HelloWorldBean {
6   @RequestMapping("/test")
7   public String doTest() {
8     return "hello world for doGet!";
9   }
10 }
```

启动 Tomcat 进行测试，在浏览器输入框内键入：localhost:8080/test。

注：这个端口号可以自定义，也可依据实际情况在请求路径前增加上下文。

运行成功，学到这里，看到这个结果，你应该很开心吧。

## 小结

这节课，我们把 MVC 与 IoC 整合在了一起。具体过程是这样的：在 Tomcat 启动的过程中先拿 context-param，初始化 Listener，在初始化过程中，创建 IoC 容器构建 WAC (WebApplicationContext)，加载所管理的 Bean 对象，并把 WAC 关联到 servlet context 里。

然后在 DispatcherServlet 初始化的时候，从 sevletContext 里获取属性拿到 WAC，放到 servlet 的属性中，然后拿到 Servlet 的配置路径参数，之后再扫描路径下的包，调用 refresh() 方法加载 Bean，最后配置 url mapping。

我们之所以有办法整合这二者，核心的原因是 **Servlet 规范中规定的时序**，从 listener 到 filter 再到 servlet，每一个环节都预留了接口让我们有机会干预，写入我们需要的代码。我们在学习过程中，更重要的是要学习如何构建可扩展体系的思路，在我们自己的软件开发过程中，记住**不要将程序流程固定死**，那样没有任何扩展的余地，而应该想着预留出一些接口理清时序，让别人在关节处也可以插入自己的逻辑。

容器是一个框架，之所以叫做框架而不是应用程序，关键就在于这套可扩展的体系，留给其他程序员极大的空间。读 Rodd Johnson 这些大师的源代码，就像欣赏一本优美的世界名著，每每都会发出“春风大雅能容物，秋水文章不染尘”的赞叹。希望你可以学到其中的精髓。

完整源代码参见 [🔗 https://github.com/YaleGuo/minis](https://github.com/YaleGuo/minis)

## 课后题

学完这节课，我也给你留一道思考题。我们看到从 Dispatcher 内可访问 WebApplicationContext 里面管理的 Bean，那通过 WebApplicationContext 可以访问

Dispatcher 内管理的 Bean 吗？欢迎你在留言区和我交流讨论，也欢迎你把这节课分享给需要的朋友。我们下节课见！

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

## 精选留言 (10)



**不是早晨，就是黄昏**

2023-04-04 来自河南

最后，在 `com.minis.test.HelloworldBean` 内的测试方法上  
但是你项目代码里有新建了一个 `test` 目录，且是和 `minis` 同级，而 `minisMVC-servlet.xml` 里配置的也是 `com.test`，文章给的代码也是 `com.test` 里的，感觉写的有点过于随意了。。。。

作者回复：多谢指正。



1



**风轻扬**

2023-04-03 来自北京

思考题：Spring中有父子容器的概念。子容器：MVC容器，父容器：Spring容器。子可以访问父，反过来不行，这是由Spring的体系结构决定的，子容器继承父容器，所以子容器是知道父容器的，所以也就能得到父容器的引用，进而得到父容器中的bean。但是父容器是无法知道子容器的，所以也就无法直接获取子容器中的bean，但是可以通过 `getBeanFactory` 来得到子容器，从而获取到子容器中的bean，但java的三层模型，`controller--->service--->dao`，`controller` 注入 `service` 对象是正常的，`service` 注入 `controller` 有点奇怪，一般不这么干。不知道以上理解的对不对

作者回复：很对。



2



**睿智的仓鼠**

2023-03-29 来自湖北

文中代码实现了 `webApplicationContext` 的注入，但排版缺少了很重要的 `populateBean()` 方法，没有使用到初始化好的ioc容器，github中相关的完整的代码是：

```
```java
```

DispatcherServlet:

```
protected void initController() {
    this.controllerNames = scanPackages(this.packageNames);
    for (String controllerName : this.controllerNames) {
        Object obj = null;
        Class<?> clz = null;
        try {
            clz = Class.forName(controllerName);
            obj = clz.newInstance();
        } catch (Exception e) {
            e.printStackTrace();
        }
        this.controllerClasses.put(controllerName, clz);

        populateBean(obj);
        this.controllerObjs.put(controllerName, obj);
    }
}

// 处理controller中的@Autowired注解
private void populateBean(Object bean) {
    Class<?> clz = bean.getClass();
    Field[] fields = clz.getDeclaredFields();
    for (Field field : fields) {
        boolean isAutowired = field.isAnnotationPresent(Autowired.class);
        if (isAutowired) {
            String fieldName = field.getName();
            Object autowiredBean = this.webApplicationContext.getBean(fieldName);
            field.setAccessible(true);
            try {
                field.set(bean, autowiredBean);
            } catch (IllegalAccessException e) {
                throw new RuntimeException(e);
            }
        }
    }
}
```



```
}  
...  

```

作者回复: 感谢感谢。



👍 1



**啊良梓是我**

2023-05-18 来自广东

课后题，通过WebApplicationContext 可以访问到DispatcherServlet里面的bean吗？

我觉得是分成两种情况来讲的？

根据Servlet的时序来讲的话，那么当初始化好WebApplicationContext的时候，DispatcherServlet还没有进行初始化，所以是空的无法访问的；

但是Servlet初始化完成后呢那应该是可以的？但是怎么访问？用一个方法？

这个方法怎么表明他是IOC的bean的？有点乱了

作者回复: 你往后面看再回首的时候，有这一部分的解释。



**啊良梓是我**

2023-05-18 来自广东

黑夜模式，代码里面的标签全没了，作者可以跟后台反馈一下吗 改一下颜色啥的

作者回复: 好的



**Geek\_b71d2c**

2023-05-06 来自北京

老师，请问一下servletContext.getInitParameter(CONFIG\_LOCATION\_PARAM)一直获取到的是空，这是什么原因呀

作者回复: web.xml文件中的初始化参数没给吧？参考Github上的





杨松

2023-04-18 来自辽宁

老师，在代码分支geek\_mvc2中，ContextLoaderListener加载了资源文件applicationContext.xml中的bean对象，然后DispatcherServlet会加载/WEB-INF/minisMVC-servlet.xml并扫描包com.test，那么这个包下的对象即被ioc容器加载了，又被DispatcherServlet(mvc容器)加载了，是不是后续的分支会解决这个问题

作者回复: 分开的, mvc容器的配置文件和ioc的不同, 从mvc容器中可以访问到ioc容器。



peter

2023-03-31 来自北京

请教老师几个问题：

Q1: 我用idea2019创建的后端项目并没有web.xml，为什么？

我创建的是springboot项目，maven项目，src目录下面是main和test，main下面是java和resource，并没有WebContent目录，也没有WEB-INF目录，更没有web.xml文件。这个现象怎么解释？另外，不同的项目有不同的目录结构，目录结构的定义在哪里有官方说明？

Q2: 用idea创建的项目缺省是基于tomcat吗？

Q3: spring必须基于tomcat，不能独立工作吗？

按文中的说法，servlet必须要用tomcat这个容器，这样的话，spring并不能独立使用，必须依赖于tomcat。

作者回复: 你从Github上看目录结构吧。或者你手工建web项目，然后把github上的代码copy过来。狭义的Spring是不需要依赖Tomcat的，可以独立的。但是如果需要web功能，就要一个servlet服务器，Tomcat只是一种。

共 3 条评论 >



马儿

2023-03-30 来自四川

课后习题：目前Dispatcher可以访问到WebApplicationContext中的bean，Dispatcher中的bean目前也存在对象的属性中了，但是Dispatcher没有被WebApplicationContext引用所以不能被访问。请问老师spring在管理controller产生的bean的时候是将这些bean统一注册到WebApplicationContext吗？

作者回复: 往后看。Spring里面是用了两层application context.



**Shark**

2023-03-29 来自浙江

在tomcat启动的过程中，是先初始化IoC容器，再初始化DispatcherServlet，在初始化DispatcherServlet的过程中记录URI与负责执行的方法和方法的对象关系映射，所以这些URI对应的对象此时是由DispatcherServlet管理的，而非IoC容器，而DispatcherServlet也不是IoC容器管理的，后续是不是会统一到IoC容器中？

作者回复：慢慢看，后面会抽出两个application context来。

