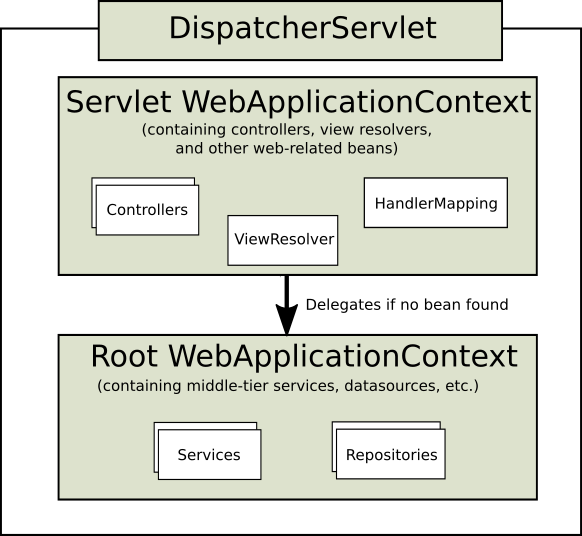
# SpringBoot启动时是如何创建IOC容器和内嵌Web容器？

**作者 王侦**

这篇文章，咱们来聊聊纯Spring和SpringBoot是怎样创建IOC容器，以及SpringBoot是怎样创建内嵌的Web容器。

### 纯Spring中IOC容器是在怎样创建的？

使用过SSM框架搭建web应用的同学都知道，如果在web.xml中配置一个 ContextLoaderListener和一个DispatcherServlet，会生成两个IOC容器，并且DispatcherServlet 创建的 IOC 容器的父容器就是ContextLoaderListener创建的 IOC 容器。



比如下面的web.xml配置

<web-app>

<listener>

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

</listener>

<context-param>

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

<param-value>/WEB-INF/app-context.xml</param-value>

</context-param>

<servlet>

<servlet-name>app</servlet-name>

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

<init-param>

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

<param-value></param-value>

</init-param>

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

</servlet>

<servlet-mapping>

<servlet-name>app</servlet-name>

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

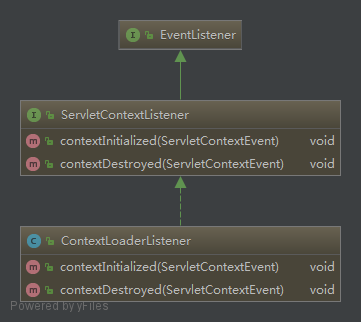
</servlet-mapping>

</web-app>

在这种父子级联容器的情况下，DispatcherServlet创建的sub context管理Controller Bean，而ContextLoaderListener创建的root context管理业务类的Bean，包括service和dao层的Bean。

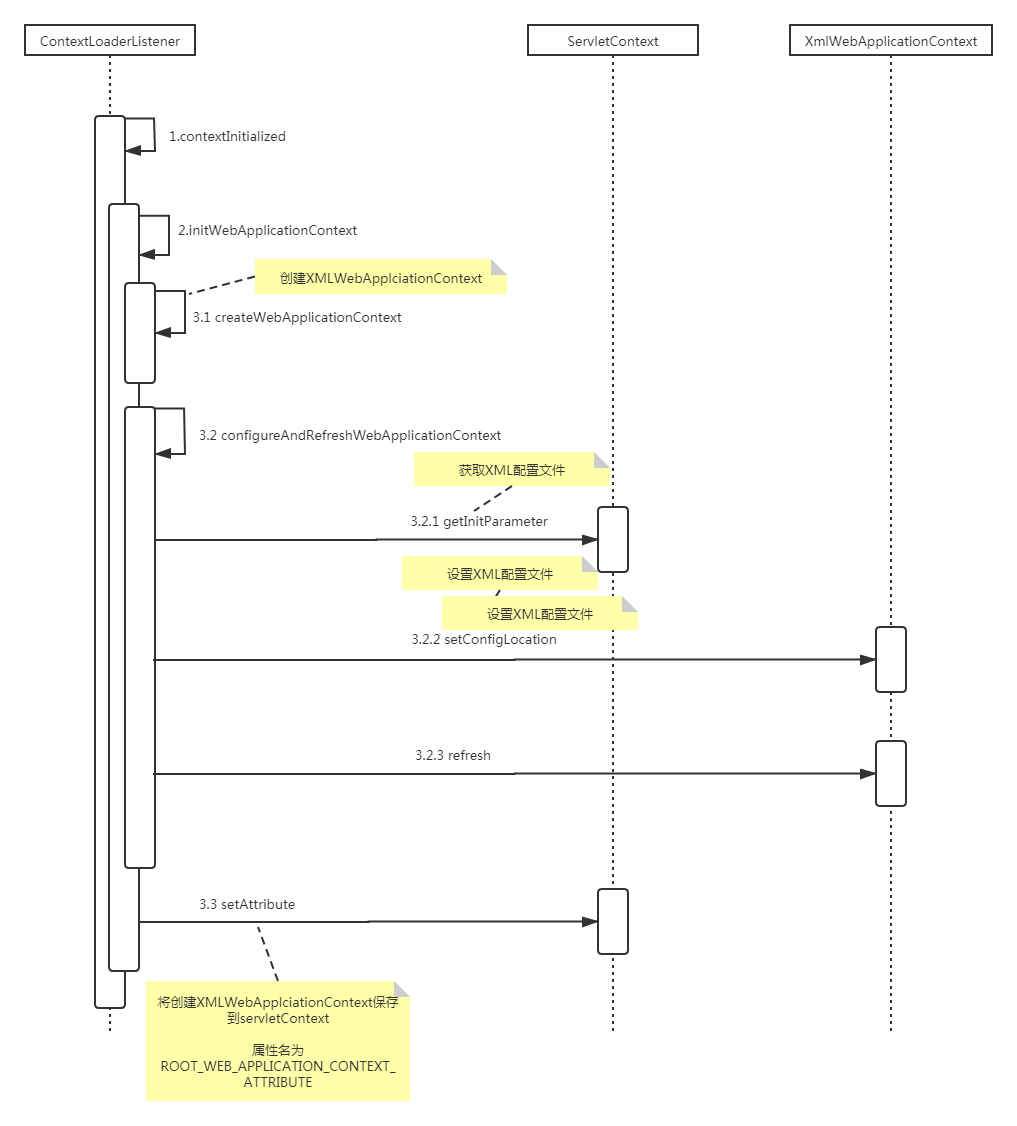
接下来，就深入了解一下这两个组件是怎么创建IOC容器的。

**ContextLoaderListener是怎样创建IOC容器的？**



ContextLoaderListener实现了ServletContextListener接口。ServletContextListener监听ServletContext的事件，ServletContextListener接口里的方法会在Web容器的生命周期特定时候被调用。

在Web服务器启动时，ServletContext被创建的时候，ContextLoaderListener的contextInitialized() 方法会被调用。下面是ContextLoaderListener的方法contextInitialized()调用时序图：



从时序图中可以看到 contextInitialized方法创建IOC容器的大致流程：

\* step1.创建XmlWebApplicationContext容器

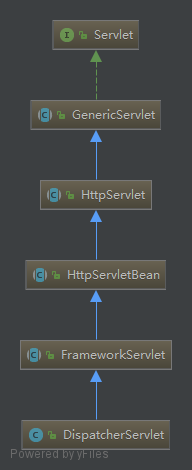
\* step2.从ServletContext中获取我们在web.xml中配置的contextConfigLocation

\* step3.将配置文件设置到XmlWebApplicationContext容器里面去

\* step4.refresh刷新IOC容器，也就是解析配置文件里面的bean并放入到IOC容器

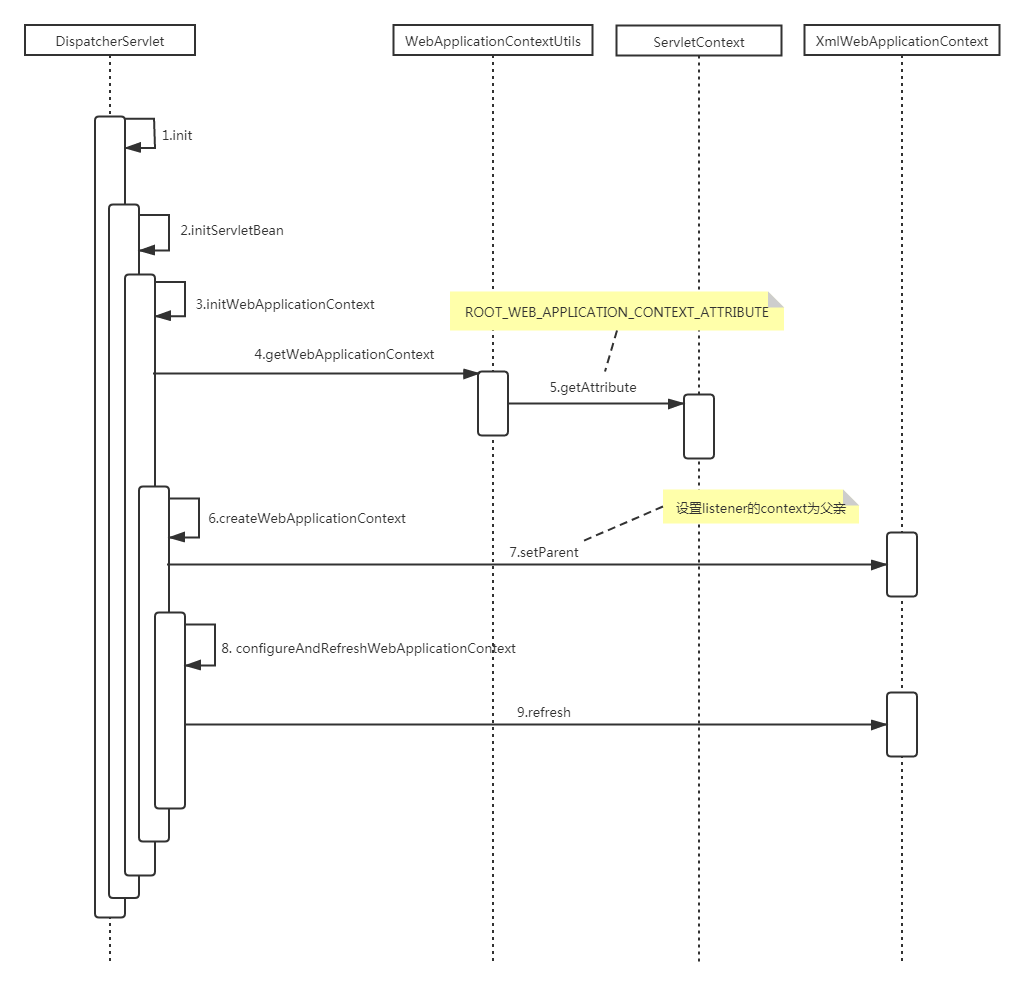
总结一下，ContextLoaderListener的作用是创建一个XmlWebApplicationContext的实例，然后解析applicationContext.xml里面的bean放入到XmlWebApplicationContext实例管理的 IOC 容器里面。

**DispatcherServlet又是怎样创建IOC容器的呢？**



DispatcherServlet类作用主要是分发请求到对应的controller，从上图可以看出其继承体系，DispatcherServlet本身是一个servlet，所以tomcat启动时候会调用其init方法。

同样，首先看下其init方法的时序图：



从时序图中可以看到init方法创建IOC容器的大致流程：

\* step1.获取前面ContextLoaderListener创建的容器，该容器放在ServletContext的ROOT\_WEB\_APPLICATION\_CONTEXT\_ATTRIBUTE属性中

\* step2.创建XmlWebApplicationContext类型的sub容器

\* step3.将sub容器的父容器设置为step1取到的父容器

\* step4.获取sub context对应的controller bean所在的配置文件路径，如果在配置 DispatcherServlet的时候配置了init-param参数contextConfigLocation，则这里会使用用户配置的 xml 文件。

\* step5.refresh刷新IOC容器，也就是解析配置文件里面的bean并放入到IOC容器

总结一下，DispatcherServlet也会创建一个XmlWebApplicationContext类型的IOC容器，并且该容器使用ContextLoaderListener创建的root context作为父容器。

用户可以通过配置 DispatcherServlet 的contextConfigLocation 参数来自定义配置文件路径。

### SpringBoot中IOC容器是在怎样创建的？

既然了解了纯Spring是怎样创建IOC容器的，那么无xml配置的SpringBoot创建IOC容器会有什么不同？

@SpringBootApplication

@EnableScheduling

@ServletComponentScan

public class Application {

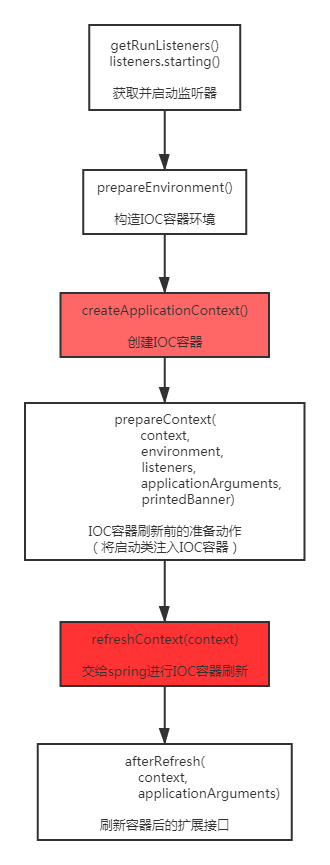
public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

}

来看一下SpringApplication的run方法的流程图：



从流程图中可以看到run方法创建IOC容器的大致流程：

\* 1）获取并启动监听器

\* 2）准备好IOC容器环境，比如解析配置文件

\* 3）创建AnnotationConfigEmbeddedWebApplicationContext类型的IOC容器

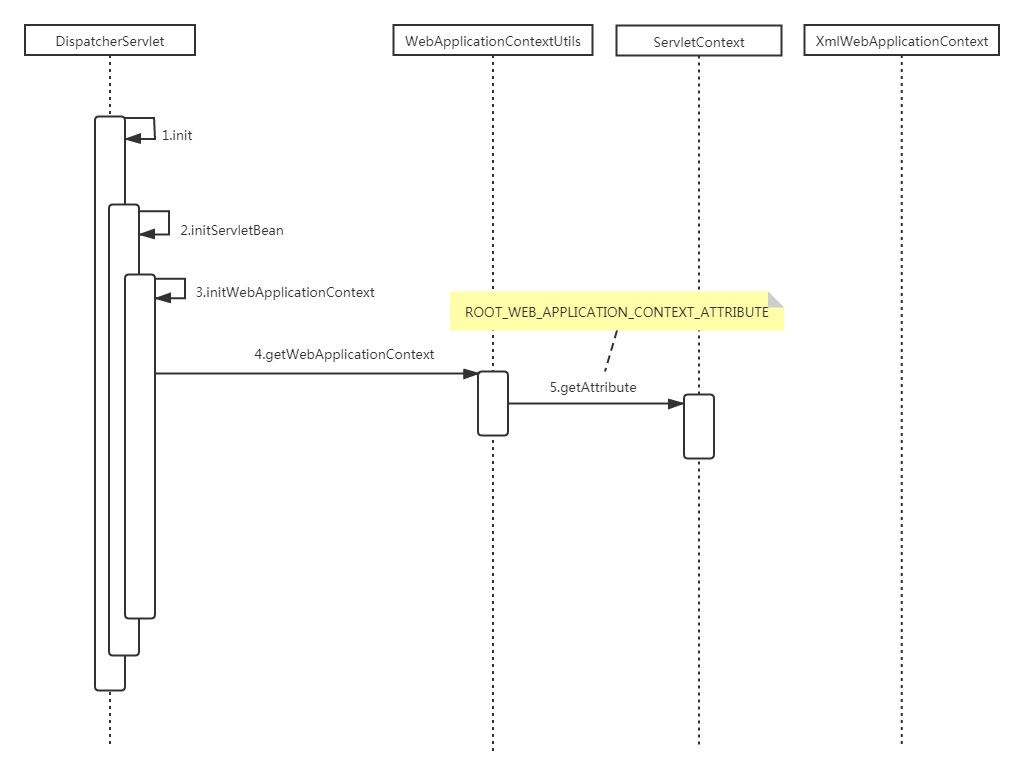
\* 4）IOC容器刷新前的准备工作，包含一个非常关键的操作：将启动类注入容器，为后续开启自动化配置奠定基础。

\* 5）接下来，SpringBoot将刷新工作交给Spring，加载实例Bean到IOC容器

\* 6）刷新容器后的扩展接口，默认为空实现。

可见SpringBoot与Spring创建IOC容器的方式很不一样， **那么SpringBoot中DispatcherServlet会不会创建IOC容器？**

这里DispatcherServlet执行的时序图与纯Spring有很大区别：



从时序图中可以看到init方法创建IOC容器的大致流程：

\* step1.这里获取的是SpringApplication的run方法创建的AnnotationConfigEmbeddedWebApplicationContext容器

\* step2.此时会发现DispatcherServlet中的webApplicationContext已经存在，就是上面创建的容器，所以后面的不会创建新的IOC容器

总结一下，SpringBoot中创建IOC容器的方式与纯Spring中创建方式有很大不同，SpringBoot默认只会在SpringApplication的run方法创建AnnotationConfigEmbeddedWebApplicationContext类型的IOC容器，并且该容器会作为DispatcherServlet的webApplicationContext。

### SpringBoot是怎样创建内嵌Web容器的？

**首先，我们要找到在哪里有Web容器的相关配置？**

@EnableAutoConfiguration注解会扫描classpath 下含有 Meta-INF/spring.factories 文件的 jar，并解析文件中名字为 org.springframework.boot.autoconfigure.EnableAutoConfiguration 的配置项。在spring-boot-autoconfigure-1.5.9.RELEASE.jar就有

# Auto Configure

org.springframework.boot.autoconfigure.EnableAutoConfiguration=\

...

org.springframework.boot.autoconfigure.web.EmbeddedServletContainerAutoConfiguration,\

org.springframework.boot.autoconfigure.web.ErrorMvcAutoConfiguration,\

org.springframework.boot.autoconfigure.web.HttpEncodingAutoConfiguration,\

org.springframework.boot.autoconfigure.web.HttpMessageConvertersAutoConfiguration,\

org.springframework.boot.autoconfigure.web.MultipartAutoConfiguration,\

...

可以看到EnableAutoConfiguration将EmbeddedServletContainerAutoConfiguration注入到 IOC 容器中。

EmbeddedServletContainerAutoConfiguration就是控制Web容器自动装配的核心类。

**接下来，看看该核心类干了啥。**

@AutoConfigureOrder(Ordered.HIGHEST\_PRECEDENCE)

@Configuration

@ConditionalOnWebApplication

@Import(BeanPostProcessorsRegistrar.class)

public class EmbeddedServletContainerAutoConfiguration {

@ConditionalOnWebApplication只有是Web类型的IOC容器时，才注入该类到IOC容器中。

@Configuration

@ConditionalOnClass({ Servlet.class, Tomcat.class })

@ConditionalOnMissingBean(value = EmbeddedServletContainerFactory.class, search = SearchStrategy.CURRENT)

public static class EmbeddedTomcat {

@Bean

public TomcatEmbeddedServletContainerFactory tomcatEmbeddedServletContainerFactory() {

return new TomcatEmbeddedServletContainerFactory();

}

}

如果当前classpath的jar里面含有Servlet.class、Tomcat.class这两个类，并且当前IOC容器中没有EmbeddedServletContainerFactory类型的Bean，则创建TomcatEmbeddedServletContainerFactory实例并放入IOC容器。

因为对于Tomcat容器来说，它的核心代码有Servlet.class、Tomcat.class这两个类。

@Configuration

@ConditionalOnClass({ Servlet.class, Server.class, Loader.class,

WebAppContext.class })

@ConditionalOnMissingBean(value = EmbeddedServletContainerFactory.class, search = SearchStrategy.CURRENT)

public static class EmbeddedJetty {

@Bean

public JettyEmbeddedServletContainerFactory jettyEmbeddedServletContainerFactory() {

return new JettyEmbeddedServletContainerFactory();

}

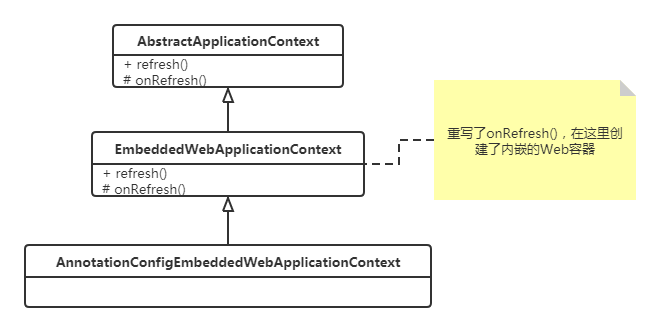
}

这里与上面Tomcat类似，只要当前classpath的jar里面含有 Servlet.class、Server.class、Loader.class、WebAppContext.class，并且IOC容器中没有EmbeddedServletContainerFactory，则创建JettyEmbeddedServletContainerFactory实例到IOC容器。

因为对于Jetty容器来说，它的核心代码有Servlet.class、Server.class、Loader.class、WebAppContext.class。

当应用依赖spring-boot-starter-web时，默认引入的是Tomcat，所以会创建一个TomcatEmbeddedServletContainerFactory实例并放入IOC容器。

**进一步来看看是什么时候创建内嵌Web容器？**



简要来说，是在SpringBoot启动的时候，也即SpringApplication的run方法中，其创建的IOC容器AnnotationConfigEmbeddedWebApplicationContext继承EmbeddedWebApplicationContext，后者重写了OnRefresh()方法，在该方法中创建了内天的Web容器。

@Override

protected void onRefresh() {

super.onRefresh();

try {

createEmbeddedServletContainer();

}

createEmbeddedServletContainer()方法的流程很简单：

\* 1）从IOC容器获取EmbeddedServletContainerFactory类型的Bean，默认情况下就是TomcatEmbeddedServletContainerFactory

\* 2）调用容器工厂的getEmbeddedServletContainer()方法

TomcatEmbeddedServletContainerFactory的getEmbeddedServletContainer()方法如下：

@Override

public EmbeddedServletContainer getEmbeddedServletContainer(

ServletContextInitializer... initializers) {

Tomcat tomcat = new Tomcat();

终于真相大白了！这就是内嵌Web容器创建的真正地方。

### 总结

\* 1）纯Spring环境下，可以配置父子级联的IOC容器。DispatcherServlet创建的sub context管理Controller Bean，而ContextLoaderListener创建的root context管理业务类的Bean，包括service和dao层的Bean。

\* 2）SpringBoot中创建IOC容器的方式与纯Spring中创建方式有很大不同，SpringBoot默认只会在启动时，也即SpringApplication.run()方法中创建IOC容器，其类型为AnnotationConfigEmbeddedWebApplicationContext，并且该容器会作为DispatcherServlet的webApplicationContext。

\* 3）SpringBoot中创建内嵌Tomcat容器也是在启动的时候，EmbeddedServletContainerAutoConfiguration是控制Web容器自动装配的核心类，其会根据classpath的jar里面含有什么类进而创建对应的Web容器工厂，默认加载到IOC容器的Web容器工厂是TomcatEmbeddedServletContainerFactory，其在EmbeddedWebApplicationContext的重写方法onRefresh()中会创建Tomcat容器。

下面是SpringBoot启动时，创建IOC容器和内嵌Web容器的流程图。

