内嵌tomcat启动原理及零配置实现

2020年6月28日 21:23

外置的tomcat：
tomcat启动加载web.xml，通过web.xml配置初始化spring容器，并加载dispatcherservlet
内嵌tomcat：
springboot启动，初始化spring容器加载bean，然后启动内嵌tomcat，通过servlet3.1规范的 ServletContainerInitializer加载dispatcherservlet

tomcat7开始提供内嵌版本的tomcat

```
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-core</artifactId>
    <version>8.5.35</version>
</dependency>
```

使用方法：

```
Tomcat tomcat = new Tomcat();
tomcat.setPort(8088);
tomcat.addWebapp("/","C://tomcat");//初始化host  engine context
tomcat.start();//初始化server和service
tomcat.getServer().await();//阻塞
```

零配置： web.xml

```
<web-app>
  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/applicationContext.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>classpath:springmvc.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>app</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
</web-app>
```

1.ContextLoaderListener作用：监听Web项目的启动过程，两个核心方法contextInitialized和contextDestroyed，contextInitialized执行时，会初始化web应用的根上下文信息。

对于Spring承载的web应用而言，可以在指定在web应用程序启动时载入IOC容器（或者称为WebApplicationContext）。这个功能是由ContextLoaderListener这个监听器类来完成的。ContextLoaderListener继承自ContextLoader，实际上就是通过ContextLoader来完成IOC的初始化的。ContextLoaderListener可以看做是Spring的启动器，Spring从这里开始

可以通过下面代码取代**ContextLoaderListener**

```
AnnotationConfigWebApplicationContext ac = new AnnotationConfigWebApplicationContext();
ac.register(RegisterBeanConfig.class);//配置包扫描
ac.refresh();
```

完成IOC容器初始化，替代ContextLoaderListener

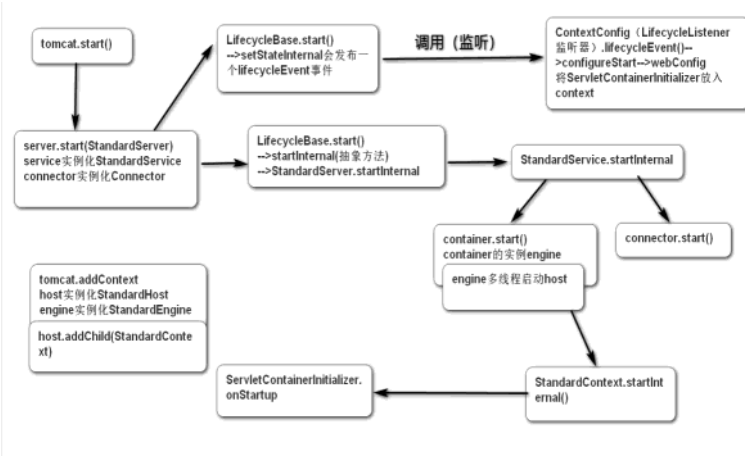
注：ac.refresh会与@EnableWedMvc冲突，调refresh时会扫描EnableWedMvc，会初始化 DelegatingWebMvcConfiguration，需要servlet，但此时**DispatcherServlet还没有，所以会报错，但是DispatcherServlet里面也会去调refresh，所以这里的ac.refresh可以省略**

2.DispatcherServlet也可以通过new出来

```
DispatcherServlet servlet = new DispatcherServlet(ac);
ServletRegistration.Dynamic registration = servletContext.addServlet("app",servlet);
registration.setLoadOnStartup(1);//0或者大于0，容器启动时加载servlet并init  负数或者没有这个属性则选择该servlet时才加载
registration.addMapping("/*");
```

怎么让这些替代xml配置的代码生效，让tomcat容器加载**DispatcherServlet?**
servlet3.0规范　要求servlet容器启动时检查（前提是web项目）　META-INF/services/javax.servlet.ServletContainerInitializer
调用里面的类的onstartup方法，并根据注解扫描
什么是web项目：tomcat.addWebapp()　tomcat就会以web项目的方式运行

tomcat是怎么调用ServletContainerInitializer的onStartup



spring-web项目的META-INF/services/javax.servlet.ServletContainerInitializer文件中记录了`org.springframework.web.SpringServletContainerInitializer`

spi简单说明
spi，即service privider interface，是jdk为厂商和插件提供的一种解耦机制。
spi的具体规范为：当服务的提供者，提供了服务接口的一种实现之后，在jar包的META-INF/services/目录里同时创建一个以服务接口命名的文件。该文件里就是实现该服务接口的具体实现类。而当外部程序装配这个模块的时候，就能通过该jar包META-INF/services/里的配置文件找到具体的实现类名，并通过反射机制实例化，完成模块的注入。 基于这样一个约定就能很好的找到服务接口的实现类，而不需要再代码里制定。jdk提供服务实现查找的一个工具类：java.util.ServiceLoader
spi在spring-web中的具体应用
从servlet3.0开始，web容器启动时为提供给第三方组件机会做一些初始化的工作，例如注册servlet或者filtes等，servlet规范中通过ServletContainerInitializer实现此功能。每个框架要使用ServletContainerInitializer就必须在对应的jar包的META-INF/services 目录创建一个名为javax.servlet.ServletContainerInitializer的文件，文件内容指定具体的ServletContainerInitializer实现类，那么，当web容器启动时就会运行这个初始化器做一些组件内的初始化工作。
一般伴随着ServletContainerInitializer一起使用的还有HandlesTypes注解，通过HandlesTypes可以将感兴趣的一些类注入到ServletContainerInitializerde的onStartup方法作为参数传入。
@HandlesTypes的实现原理：
作用是将注解指定的Class对象作为参数传递到onStartup（ServletContainerInitializer）方法中。
留给用户扩展的，他指定的Class对象并没有要继承ServletContainerInitializer，更没有写入META-INF/services/的文件（也不可能写入）中，那么Tomcat是怎么扫描到指定的类的呢。
答案是Byte Code Engineering Library (BCEL)，这是Apache Software Foundation 的Jakarta 项目的一部分，作用同ASM类似，是字节码操纵框架。
webConfig() 在调用processServletContainerInitializers()时记录下注解的类名，然后在Step 4和Step 5中都来到processAnnotationsStream这个方法，使用BCEL的ClassParser在字节码层面读取了/WEB-INF/classes和某些jar（应该可以在叫做fragments的概念中指定）中class文件的超类名和实现的接口名，判断是否与记录的注解类名相同，若相同再通过org.apache.catalina.util.Introspection类load为Class对象，最后保存起来，于交给org.apache.catalina.core.StandardContext，也就是tomcat实际调用ServletContainerInitializer.onStartup()的地方。

springboot内嵌tomcat启动源码：

```
springApplication.run
    -->refreshContext
      -->refresh
        -->ServletWebServerApplicationContext.refresh
          -->AbstractApplicationContext.refresh
            -->onRefresh
              -->ServletWebServerApplicationContext.onRefresh
                -->createWebServer
                  -->ServletWebServerFactory.getWebServer
                    -->TomcatServletWebServerFactory.getWebServer
                      -->new Tomcat
                        -->getTomcatWebServer
                          -->TomcatWebServer.initialize
                            -->tomcat.start()
                              -->startDaemonAwaitThread
                                -->tomcat.getServer().await()
```

注：springApplication.run-->`createApplicationContext`　初始化`context`为ServletWebServerApplicationContext　extends　AbstractApplicationContext

ServletWebServerApplicationContext.getWebServerFactory--spring-boot-starter-web的依赖是tomcat如果想改为netty，需要将这个依赖exclude，然后将jetty的包依赖进来

加载DispatcherServlet：

spring.factories中配置DispatcherServletAutoConfiguration，实例化DispatcherServlet

并实例化DispatcherServletRegistrationBean，同时注入DispatcherServlet实例并设置LoadOnStartup及UrlMappings

ServletWebServerApplicationContext.onRefresh-->createWebServer-->**getSelfInitializer().onStartup(servletContext)-->**

selfInitialize-->(ServletContextInitializer) beans.onStartup(servletContext)

createWebServer-->factory.getWebServer(getSelfInitializer())-->

TomcatServletWebServerFactory.getWebServer-->prepareContext-->configureContext(context,ServletContextInitializer)-->

context.addServletContainerInitializer(new TomcatStarter(initializers)), *NO_CLASSES*)

```
TomcatStarter implements ServletContainerInitializer -->onStartup
    --> (ServletContextInitializer)initializer.onStartup(servletContext)
     -->RegistrationBean.onStartup
      -->register
        -->DynamicRegistrationBean.register
          -->addRegistration
           -->ServletRegistrationBean.addRegistration
             -->servletContext.addServlet(name, this.servlet)
```

springboot源码的流程：IOC容器初始化-->扫描ServletContextInitializer的实现类放入initializers，创建TomcatStarter实例，initializers赋值给TomcatStarter，
将TomcatStarter放入StandardContext的initializers-->tomcat启动-->调用initializers的onStartUp dispatcherServlet加载到tomcat上下文

RegistrationBean（实现了ServletContextInitializer）的onstartup里面将dispatcherServlet加载到tomcat上下文