Tomcat是怎样处理Spring Boot应用的?

2018-02-27 侯树成 Java后端技术

来源 | 公众号 | Tomcat那些事儿

作者 | 侯树成

近一两年， SpringBoot 由于其减少了大量原本繁琐的 Spring 配置，以及基于 Boot 的 SpringCloud 的推广，越来越多的应用开始使用 SpringBoot进行开发。

而 SpringBoot 以标准Java 应用的形式，来启动了一个 Web 服务，而将容器的存在，隐藏在一个配置文件中，使用起来很方便。而 Tomcat 就是 Spring Boot 内置的容器之一。

这次我们来看在 SpringBoot 中， Tomcat 中怎样被集成进来提供服务的。

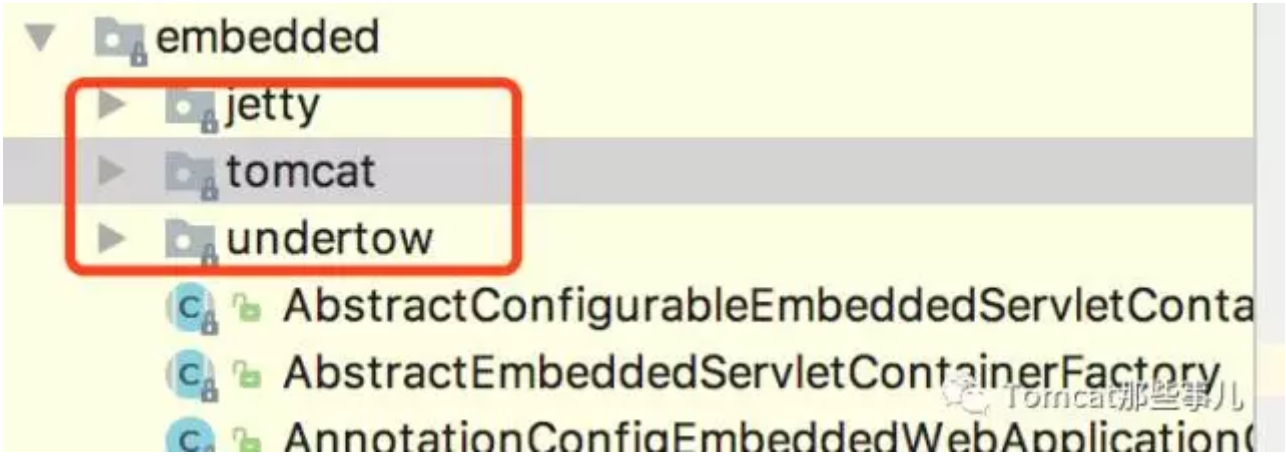
前面的文章写过关于 Tomcat 的 Digester 组件解析 配置文件 server.xml， 根据配置信息生成 Tomcat 实例。(Tomcat配置文件解析与Digester)

在 Spring Boot 中，实现也基本类似。 区别在于配置信息大部分是默认的，另外一些用户特定设置的，通过在 application.properties 之类的 Boot 配置文件里，读出来解析并设置到 Tomcat 的各个组件上。

另外一个区别是， Spring Boot 使用的是 Embedded Tomcat 。这个我们在前面的文章里也曾简单介绍过(Embedded Tomcat，朋友，要不要试试)

当然，上面这两点，是整个 Boot 项目中使用到 Tomcat 的基本原理，但具体对于 Embedded Tomcat 的使用，Boot 里和 Maven 插件的使用还是有一些区别的。

这是 Boot 使用的三个embedded 容器，默认启动的是 Tomcat。



要分析这个问题，该从哪看起呢？

Boot 在启动的时候，很清楚的告诉我们这样一条信息

> s.b.c.e.t.**TomcatEmbeddedServletContainer** : Tomcat initialized with port

我们看到的这一条是logback输出的信息。前面是缩略形式写的包名，最主要的是这个Container，跳转到类里看一眼。

可以匹配到这一行 log 的, 是 container 的init 方法

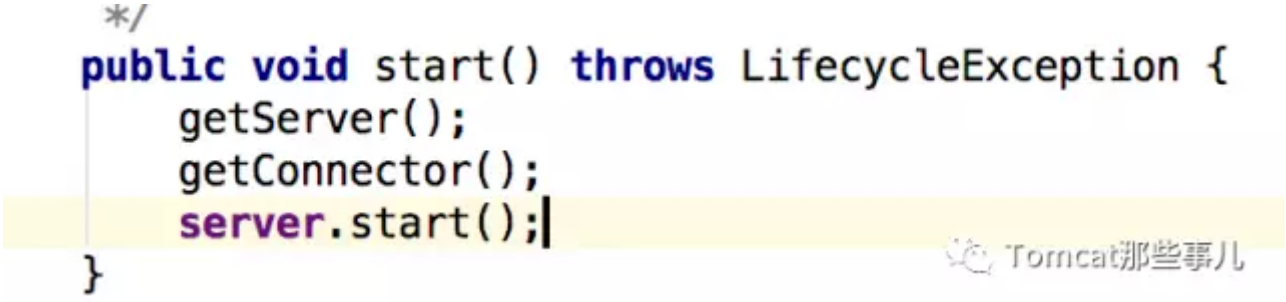
```
private void initialize() throws EmbeddedServletContainerException {
    TomcatEmbeddedServletContainer.logger
            .info( o: "Tomcat initialized with port(s): " + getPortsDescription( localPort: false));
    synchronized (this.monitor) {
        try {
            addInstanceIdToEngineName();
            try {
                // Remove service connectors to that protocol binding doesn't happen
                // yet
                removeServiceConnectors();

                // Start the server to trigger initialization listeners
                this.tomcat.start();

                // We can re-throw failure exception directly in the main thread
                rethrowDeferredStartupExceptions();

                Context context = findContext();
                try {
                    ContextBindings.bindClassLoader(context, getNamingTok(context),
                            getClass().getClassLoader());
                }
```


前面一些细节类的内容先不过多关注，进入眼里的， 一定是这个

> this.**tomcat**.start();

这里这个 tomcat ，就是 Embedded Tomcat类的实例。

这里 start 的操作，是将 容器启动起来



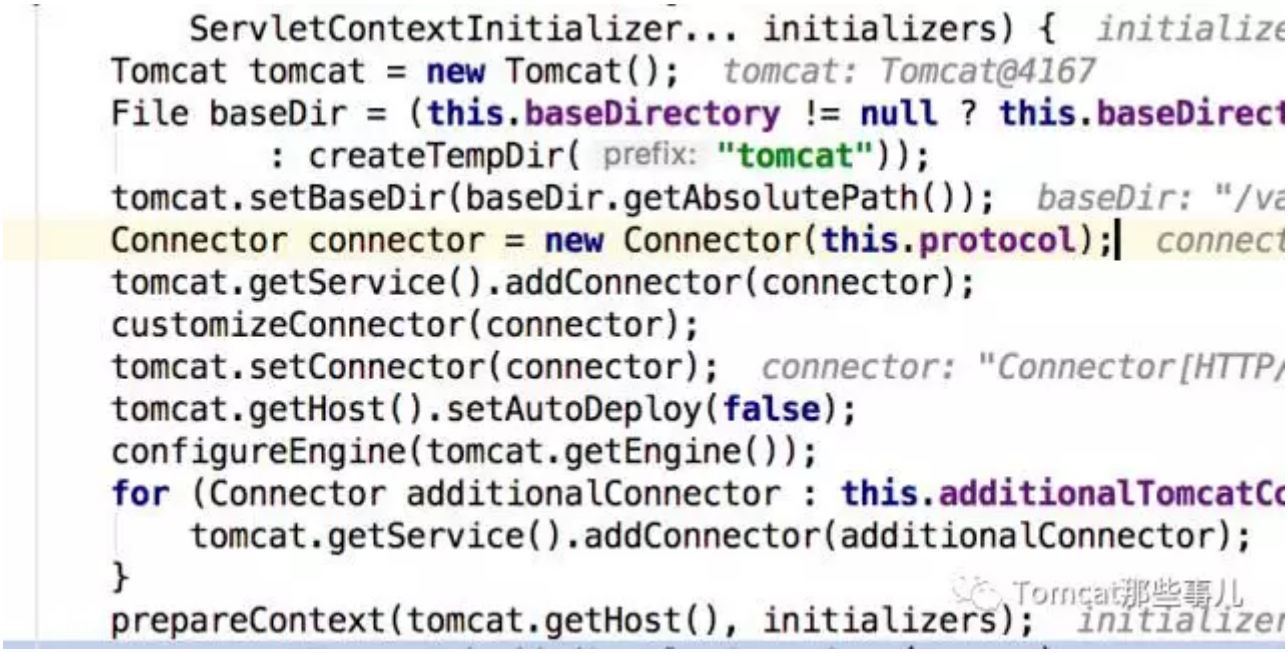
方法里的 getServer, getConnector 这些， 熟悉 Tomcat 的朋友都了解，Tomcat 内部有以下几个主要的组件:

- Egine
- Host
- Context
- Wrapper
- **Connector**

前四个是容器从上到下的组件，是一个包含的关系。而光有这些还不足以让我们访问到部署的应用，此时容器连接外界的组件 **Connector** 就显的必不可少了。

而且，真正到了**start** 这一步的时候，容器的组件配置都已经完成了，只是要启动以提供服务。 配置的这些读取，都是在 initial 阶段之前，已经完成。

下图是初始化阶段读取配置时的一些代码， 没有特别的地方，设置 BaseDir， 解析配置设置各个组件。



此外，在Spring Boot 应用启动时,会有这样几条日志输出。

```
o.s.web.context.ContextLoader                : Root WebApplicationContext: initialization completed in 4436
o.s.b.w.servlet.ServletRegistrationBean      : Mapping servlet: 'dispatcherServlet' to [/]
o.s.b.w.servlet.FilterRegistrationBean       : Mapping filter: 'characterEncodingFilter' to: [/*]
o.s.b.w.servlet.FilterRegistrationBean       : Mapping filter: 'hiddenHttpMethodFilter' to: [/*]
o.s.b.w.servlet.FilterRegistrationBean       : Mapping filter: 'httpPutFormContentFilter' to: [/*]
o.s.b.w.servlet.FilterRegistrationBean       : Mapping filter: 'requestContextFilter' to: [/*]
s.w.s.m.m.a.RequestMappingHandlerAdapter     : Looking for @ControllerAdvice: org.springframework.boot.cont
```

我们知道，Spring MVC 是通过 DispatcherServlet 来分发处理请求，在 Spring Boot 出现之前，都是需要在web.xml里配置，来实现请求的拦截。

而在Servlet 3.0 之后，规则中新增了Dynamic Servlet、Dynamic Filter这些概念, 可以在运行时动态注册组件到 Context 中。

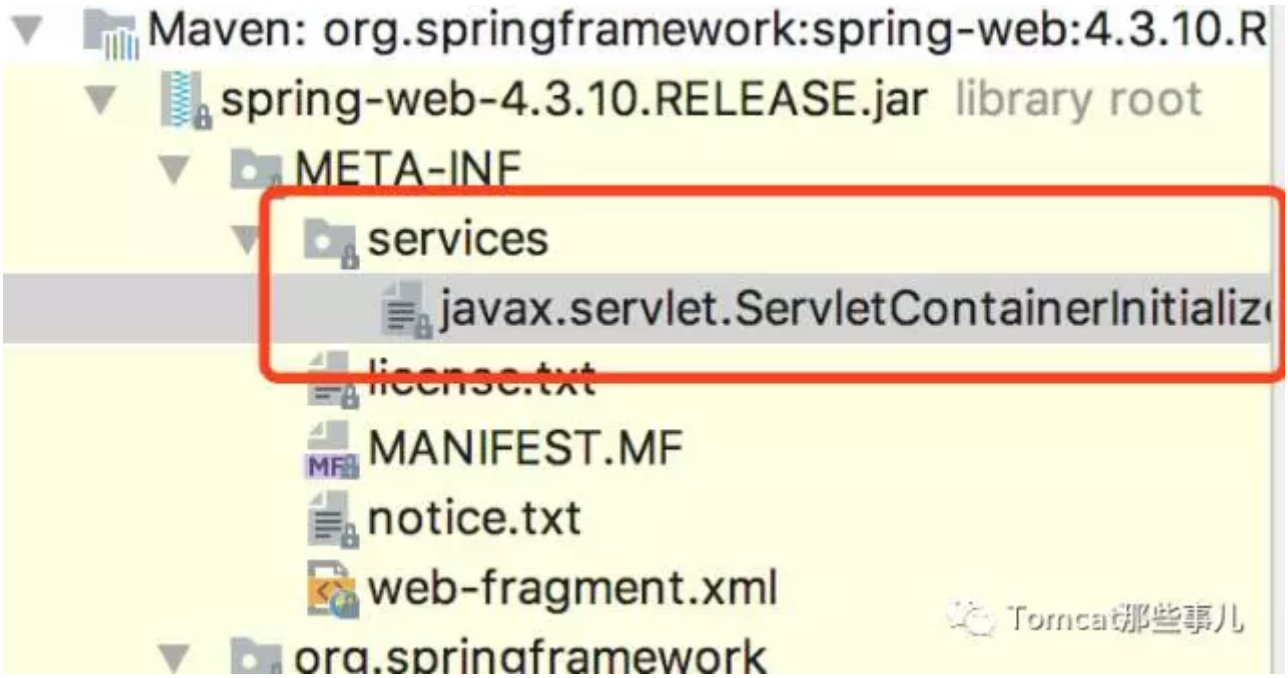
```
public void onStartup(ServletContext servletContext) throws ServletException {
    Assert.notNull(this.servlet,  message: "Servlet must not be null");
    String name = getServletName();  name: "dispatcherServlet"
    if (!isEnabled()) {
        logger.info( o: "Servlet " + name + " was not registered (disabled)");
        return;
    }
    logger.info( o: "Mapping servlet: " + name + " to " + this.urlMappings);
    Dynamic added = servletContext.addServlet(name, this.servlet);  added: App
    if (added == null) {
        logger.info( o: "Servlet " + name + " was not registered "  name: "disp
                + "(possibly already registered?)");
        return;
    }
    configure(added);  added: ApplicationServletRegistration@5044
}
```

所以我们观察到的 Context 仅仅是一个空的应用，然后再通过动态添加Servlet、 Filter 等内容进去。

除了以 Jar 的形式直接执行 Main 方法外， Spring Boot 还支持将 Boot 应用打包成 War 文件，部署到标准和容器中，不使用 Embedded 容器。

相比执行 Main 方法来启动 Spring Boot 应用，以 Web 应用提供时， Boot 的能力是如何提供的呢？

来看下面这张图，Jar文件的META-INF中services中包含一个SCI的声明。



这就是Spring Boot 在标准Web容器中能生效的秘密。

SCI是做什么的呢？

容器启动时会依次处理每个 **ServletContainerInitializer** 的HandlesTypes注解，然后分别调用所有ServletContainerInitializer对象的onStartup方法，并将处理HandlesTypes注解得到的类数组，传递给ServletContainerInitializer的onStartup方法。

在configure阶段，我们将 Boot 打包成 war 时提供的Initalizer，并将其 run 起来。

此时处理 dispatcherServlet 这些，和 以Main方法启动执行没什么区别。

所以，当我们看到 Boot 应用能够以如此少的配置便利的作为 Web 应用执行时，要清楚的认识到，背后的 Embedded 容器 还是做了不少工作，同时也是和各种新的 J2EE规范有关。 而最重要的是，无论怎么变化，本质上还是那样，做为一个标准的 Context 在使用，区别只在于是通过解析静态文件进行配置，还是通过动态添加进行配置。

送书规则

1、本次活动还是在评论区随机抽取一名幸运吃瓜群众！

2、截止日期：**2018年02月27日23时00分**，幸运的小伙伴**名单评论区留言会置顶**，获奖的小伙伴请在一个工作日内在公众号界面联系我，发送**手机号、姓名、收货地址**，逾期则认为放弃处理；

3、注意啦：经常关注Java后端技术公众号留言点赞的小伙伴会自带红蓝buff加成哦！

两本书中一本从留言区抽取，一本从“抽奖助手”中抽！

点击图片查看更多推荐内容

↓↓↓

年后返程的程序员:带不走的，是家里的爱！

Web 和 Chrome 开发者之间的那些事！

一步步带你了解前后端分离利器之JWT

阅读原文