链滴

Spring 去除 web.xml，使用 Java 配置 Servlet 的原理，SPI

作者：Clinan

原文链接：https://ld246.com/article/1564659201518

来源网站：链滴

Spring去除web.xml，使用Java配置Servlet的原理，SPI

1. spring使用java配置Servlet代码如下

```
import org.springframework.web.WebApplicationInitializer;
import org.springframework.web.context.support.AnnotationConfigWebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;

import javax.servlet.ServletContext;
import javax.servlet.ServletRegistration;

public class WebServletConfiguration implements WebApplicationInitializer {
    @Override
    public void onStartup(ServletContext ctx) {
        AnnotationConfigWebApplicationContext webCtx = new AnnotationConfigWebApplicat
onContext();
        // 注册context
        webCtx.register(ApplicationConfig.class);
        // 设置context
        webCtx.setServletContext(ctx);
        // 定义Servlet
        ServletRegistration.Dynamic servlet = ctx.addServlet("spring", new DispatcherServlet(we
Ctx));
        servlet.setLoadOnStartup(1);
        servlet.addMapping("/");
    }
}
```

这段代码配置了一个名为spring的Servlet, 所有的请求（/）都映射到这个Servlet中。

2. spring的源码

在上一点中，可以看到关键的代码是现实了WebApplicationInitializer这个接口，很容易的找到这个口唯一被引用的地方，即org.springframework.web.SpringServletContainerInitializer

```
// HandlesTypes注解再这里的作用是tomcat回去扫描所有的WebApplicationInitializer的实现类，括抽象类。
@HandlesTypes(WebApplicationInitializer.class)
public class SpringServletContainerInitializer implements ServletContainerInitializer {

    @Override
    // webAppInitializerClasses参数返回的就是所有的WebApplicationInitializer的实现类
    public void onStartup(@Nullable Set<Class<?>> webAppInitializerClasses, ServletContext
ervletContext)
            throws ServletException {

        List<WebApplicationInitializer> initializers = new LinkedList<>();

        if (webAppInitializerClasses != null) {
            for (Class<?> waiClass : webAppInitializerClasses) {
                // Be defensive: Some servlet containers provide us with invalid classes,
                // no matter what @HandlesTypes says...
                if (!waiClass.isInterface() && !Modifier.isAbstract(waiClass.getModifiers()) &&
                        WebApplicationInitializer.class.isAssignableFrom(waiClass)) {
                    try {
                        initializers.add((WebApplicationInitializer)
                                ReflectionUtils.accessibleConstructor(waiClass).newInstance());
                    }
                    catch (Throwable ex) {
                        throw new ServletException("Failed to instantiate WebApplicationInitializer cla
s", ex);
                    }
                }
            }
        }

        if (initializers.isEmpty()) {
            servletContext.log("No Spring WebApplicationInitializer types detected on classpath");
            return;
        }

        servletContext.log(initializers.size() + " Spring WebApplicationInitializers detected on clas
path");
        AnnotationAwareOrderComparator.sort(initializers);
        // 传入servletContext给WebApplicationInitializer的实现类，并调用onStartup方法，此方法定义Servlet
        for (WebApplicationInitializer initializer : initializers) {
            initializer.onStartup(servletContext);
        }
    }
}
```

3. tomcat源码

继续根据实现的接口ServletContainerInitializer追溯到tomcat源码中。

再org.apache.catalina.startup.ContextConfig#processServletContainerInitializers方法中对ServletContainerInitializer的实现类进行了扫描，也就是扫描到了SpringServletContainerInitializer

```
/**
* Scan JARs for ServletContainerInitializer implementations.
*/
protected void processServletContainerInitializers() {

    List<ServletContainerInitializer> detectedScis;
    try {
        WebappServiceLoader<ServletContainerInitializer> loader = new WebappServiceLoader
>(context);
        detectedScis = loader.load(ServletContainerInitializer.class);
    } catch (IOException e) {
        log.error(sm.getString(
                "contextConfig.servletContainerInitializerFail",
                context.getName()),
            e);
        ok = false;
        return;
    }
    // 省略一堆源码
}
```

到这里需要知道SPI(Service Provider Interface)，简单的说就是一个服务发现的解决方案，这个方案JDK的实现和tomcat的实现，上面源码中的WebappServiceLoader就是tomcat的实现。

SPI中不得不说的配置文件就是spring-web包下META-INFO/services目录下的javax.servlet.ServletContainerInitializer文件,内容如下

```
org.springframework.web.SpringServletContainerInitializer
```

原理是，文件名称是服务发现的接口全路径名称，内容是接口实现类的全路径名称。SPI会创建这个现类的实例，并通过接口调用org.springframework.web.SpringServletContainerInitializer#onStartup方法。

4.总结

1. 整个过程就不再追溯到tomcat的再往上的源码了，我们已经可以了解到tomcat和spring是怎么样现了不使用web.xml配置Servlet的了。

2. 如果有继续想要了解JDK的SPI实现，可以看下 java.util.ServiceLoader

3. 如果看完spring的这个实现还不是很懂，可以看 logback日志框架的ch.qos.logback.classic.servet.LogbackServletContainerInitializer实现。