2021-02-10
Spring Boot

精尽 Spring Boot 源码分析 —— SpringFactoriesLoader

1. 概述

本文，我们来补充 《精尽 Spring Boot 源码分析 —— SpringApplication》 文章，并未详细解析的 SpringFactoriesLoader 。

> `spring.factories` 配置文件，我们可以认为是 Spring 自己的一套 SPI 机制的配置文件，在里面可以配置不同接口对应的实现类们。例如：
>
> ```
> # PropertySource Loaders
> org.springframework.boot.env.PropertySourceLoader=\
> org.springframework.boot.env.PropertiesPropertySourceLoader,\
> org.springframework.boot.env.YamlPropertySourceLoader
>
> # Run Listeners
> org.springframework.boot.SpringApplicationRunListener=\
> org.springframework.boot.context.event.EventPublishingRunListener
>
> // ... 省略其它
> ```
>
> - 并且，Spring 内置了多个 `spring.factories` 配置文件。另外，我们也可以添加 `spring.factories` 配置文件。
>
> SpringFactoriesLoader 类，用于加载 `spring.factories` 配置文件。
>
> 艿艿：如果对 Java SPI 不了解的胖友，推荐后面看看 《JAVA 拾遗 —— 关于 SPI 机制》 文章。

2. SpringFactoriesLoader

`org.springframework.core.io.support.SpringFactoriesLoader` ，加载 `spring.factories` 的工具类。其类上，注释如下：

```
// SpringFactoriesLoader.java

/**
 * General purpose factory loading mechanism for internal use within the framework.
 *
 * <p>{@code SpringFactoriesLoader} {@linkplain #loadFactories loads} and instantiates
 * factories of a given type from {@value #FACTORIES_RESOURCE_LOCATION} files which
 * may be present in multiple JAR files in the classpath. The {@code spring.factories}
 * file must be in {@link Properties} format, where the key is the fully qualified
 * name of the interface or abstract class, and the value is a comma-separated list of
 * implementation class names. For example:
 *
 * <pre class="code">example.MyService=example.MyServiceImpl1,example.MyServiceImpl2</pre>
 *
 * where {@code example.MyService} is the name of the interface, and {@code MyServiceImpl1}
 * and {@code MyServiceImpl2} are two implementations.
 */
```

从包名我们就可以看出，SpringFactoriesLoader 是 Spring Framework 就已经提供的工具类，而不是 Spring Boot 所特有的。

2.1 构造方法

```
// SpringFactoriesLoader.java

/**
 * The location to look for factories.
 * <p>Can be present in multiple JAR files.
 */
public static final String FACTORIES_RESOURCE_LOCATION = "META-INF/spring.factories";

private static final Log logger = LogFactory.getLog(SpringFactoriesLoader.class);

/**
 * 缓存
 *
 * KEY1 ：ClassLoader
 * KEY2 ：接口的类名
 * VALUE ：实现的类名的数组。注意哟，是个 MultiValueMap 类
 */
private static final Map<ClassLoader, MultiValueMap<String, String>> cache = new ConcurrentReferenceHashMap<>();

private SpringFactoriesLoader() {
}
```

FACTORIES_RESOURCE_LOCATION 静态属性，定义了读取的是 "META-INF/spring.factories" 配置文件。并且，每个 JAR 文件里，都可以有一个这个配置文件。
cache 静态属性，读取 "META-INF/spring.factories" 配置文件的缓存。

2.2 loadFactoryNames

#loadFactoryNames(Class<?> factoryClass, @Nullable ClassLoader classLoader) 静态方法，获得接口对应的实现类名们。代码如下：

```
// SpringFactoriesLoader.java

/**
 * Load the fully qualified class names of factory implementations of the
 * given type from {@value #FACTORIES_RESOURCE_LOCATION}, using the given
 * class loader.
 * @param factoryClass the interface or abstract class representing the factory
 * @param classLoader the ClassLoader to use for loading resources; can be
 * {@code null} to use the default
 * @throws IllegalArgumentException if an error occurs while loading factory names
 * @see #loadFactories
 */
public static List<String> loadFactoryNames(Class<?> factoryClass, @Nullable ClassLoader classLoader) {
    // 获得接口的类名
    String factoryClassName = factoryClass.getName();
    // 加载 FACTORIES_RESOURCE_LOCATION 配置文件，获得接口对应的实现类名们
    return loadSpringFactories(classLoader).getOrDefault(factoryClassName, Collections.emptyList());
}

private static Map<String, List<String>> loadSpringFactories(@Nullable ClassLoader classLoader) {
    // 如果缓存中已经存在，则直接返回
    MultiValueMap<String, String> result = cache.get(classLoader);
    if (result != null) {
        return result;
    }

    try {
        // 获得 FACTORIES_RESOURCE_LOCATION 对应的 URL 们
        Enumeration<URL> urls = (classLoader != null ? classLoader.getResources(FACTORIES_RESOURCE_LOCATION) : ClassL
        // 创建 LinkedMultiValueMap 对象
        result = new LinkedMultiValueMap<>();
        // 遍历 URL 数组
        while (urls.hasMoreElements()) {
            // 获得 URL
            URL url = urls.nextElement();
            // 创建 UrlResource 对象
            UrlResource resource = new UrlResource(url);
            // 加载 "META-INF/spring.factories" 配置文件，成为 Properties 对象
            Properties properties = PropertiesLoaderUtils.loadProperties(resource);
            // 遍历 Properties 对象
            for (Map.Entry<?, ?> entry : properties.entrySet()) {
                // 使用逗号分隔
                List<String> factoryClassNames = Arrays.asList(StringUtils.commaDelimitedListToStringArray((String) e
                // 添加到 result 中
                result.addAll((String) entry.getKey(), factoryClassNames);
            }
        }
        // 添加到 cache 中
        cache.put(classLoader, result);
        return result;
    } catch (IOException ex) {
        throw new IllegalArgumentException("Unable to load factories from location [" + FACTORIES_RESOURCE_LOCATION +
    }
}
```

加载 `FACTORIES_RESOURCE_LOCATION` 配置文件，获得接口对应的实现类名们。

2.3 loadFactories

#loadFactories(Class<T> factoryClass, @Nullable ClassLoader classLoader) 静态方法，获得接口对应的实现类名们，然后创建对应的对象们。代码如下：

```
// SpringFactoriesLoader.java

/**
 * Load and instantiate the factory implementations of the given type from
 * {@value #FACTORIES_RESOURCE_LOCATION}, using the given class loader.
 * <p>The returned factories are sorted through {@link AnnotationAwareOrderComparator}.
 * <p>If a custom instantiation strategy is required, use {@link #loadFactoryNames}
 * to obtain all registered factory names.
 * @param factoryClass the interface or abstract class representing the factory
 * @param classLoader the ClassLoader to use for loading (can be {@code null} to use the default)
 * @throws IllegalArgumentException if any factory implementation class cannot
 * be loaded or if an error occurs while instantiating any factory
 * @see #loadFactoryNames
 */
public static <T> List<T> loadFactories(Class<T> factoryClass, @Nullable ClassLoader classLoader) {
    Assert.notNull(factoryClass, "'factoryClass' must not be null");
    // 获得 ClassLoader
    ClassLoader classLoaderToUse = classLoader;
    if (classLoaderToUse == null) {
        classLoaderToUse = SpringFactoriesLoader.class.getClassLoader();
    }
    // 获得接口对应的实现类名们
    List<String> factoryNames = loadFactoryNames(factoryClass, classLoaderToUse);
    if (logger.isTraceEnabled()) {
        logger.trace("Loaded [" + factoryClass.getName() + "] names: " + factoryNames);
    }
    // 遍历 factoryNames 数组，创建实现类的对象
    List<T> result = new ArrayList<>(factoryNames.size());
    for (String factoryName : factoryNames) {
        result.add(instantiateFactory(factoryName, factoryClass, classLoaderToUse));
    }
    // 排序
    AnnotationAwareOrderComparator.sort(result);
    return result;
}

private static <T> T instantiateFactory(String instanceClassName, Class<T> factoryClass, ClassLoader classLoader) {
    try {
        // 获得 Class 类
        Class<?> instanceClass = ClassUtils.forName(instanceClassName, classLoader);
        // 判断是否实现了指定接口
        if (!factoryClass.isAssignableFrom(instanceClass)) {
            throw new IllegalArgumentException("Class [" + instanceClassName + "] is not assignable to [" + factoryCl
        }
        // 创建对象
        return (T) ReflectionUtils.accessibleConstructor(instanceClass).newInstance();
    } catch (Throwable ex) {
        throw new IllegalArgumentException("Unable to instantiate factory class: " + factoryClass.getName(), ex);
    }
}
```

666. 彩蛋

水文一篇，哇咔咔~

文章目录

回到首页