无

> 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 。

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。

> 芳芳：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。

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

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

```
public static final String FACTORIES_RESOURCE_LOCATION = "META-INF/spring.factories";

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

```
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：

```
public static List<String> loadFactoryNames(Class<?> factoryClass, @Nullable ClassLoader classLoader) {
```

```
    String factoryClassName = factoryClass.getName();

    return loadSpringFactories(classLoader).getOrDefault(factoryClassName, Collections.emptyList());
}

private static Map<String, List<String>> loadSpringFactories(@Nullable ClassLoader classLoader) {

    MultiValueMap<String, String> result = cache.get(classLoader);
    if (result != null) {
        return result;
    }


    try {

        Enumeration<URL> urls = (classLoader != null ? classLoader.getResources(FACTORIES_RESOURCE_LOCATION) : ClassLoader.getSystemResources(FACTORIES_RESOURCE_LOCATION));

        result = new LinkedMultiValueMap<>();

        while (urls.hasMoreElements()) {

            URL url = urls.nextElement();

            UrlResource resource = new UrlResource(url);

            Properties properties = PropertiesLoaderUtils.loadProperties(resource);

            for (Map.Entry<?, ?> entry : properties.entrySet()) {

                List<String> factoryClassNames = Arrays.asList(StringUtils.commaDelimitedListToStringArray((String) entry.getValue()));

                result.addAll((String) entry.getKey(), factoryClassNames);
            }
        }

        cache.put(classLoader, result);
        return result;
    } catch (IOException ex) {
        throw new IllegalArgumentException("Unable to load factories from location [" + FACTORIES_RESOURCE_LOCATION + "]", ex);
    }
}
```

- 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：

```
public static <T> List<T> loadFactories(Class<T> factoryClass, @Nullable ClassLoader classLoader) {
    Assert.notNull(factoryClass, "'factoryClass' must not be null");

    ClassLoader classLoaderToUse = classLoader;
    if (classLoaderToUse == null) {
        classLoaderToUse = SpringFactoriesLoader.class.getClassLoader();
    }

    List<String> factoryNames = loadFactoryNames(factoryClass, classLoaderToUse);
    if (logger.isTraceEnabled()) {
        logger.trace("Loaded [" + factoryClass.getName() + "] names: " + factoryNames);
    }

    List<T> result = new ArrayList<>(factoryNames.size());
    for (String factoryName : factoryNames) {
        result.add(instantiateFactory(factoryName, factoryClass, classLoaderToUse));
    }

    AnnotationAwareOrderComparator.sort(result);
    return result;
}

private static <T> T instantiateFactory(String instanceClassName, Class<T> factoryClass, ClassLoader classLoader) {
    try {

        Class<?> instanceClass = ClassUtils.forName(instanceClassName, classLoader);
```

```
            if (!factoryClass.isAssignableFrom(instanceClass)) {
                throw new IllegalArgumentException("Class [" + instanceClassName + "] is not assignable to [" + factoryClass.getName() + "]");
            }

            return (T) ReflectionUtils.accessibleConstructor(instanceClass).newInstance();
        } catch (Throwable ex) {
            throw new IllegalArgumentException("Unable to instantiate factory class: " + factoryClass.getName(), ex);
        }
    }
```

水文一篇，哇咔咔~