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 BeanDefinitionLoader 。

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 BeanDefinitionLoader 。在 SpringApplication 中，我们可以看到 `#load(ApplicationContext context, Object[] sources)` 方法中，是如下一段代码：

```
protected void load(ApplicationContext context, Object[] sources) {
        if (logger.isDebugEnabled()) {
                logger.debug("Loading source " + StringUtils.arrayToCommaDelimitedString(sources));
        }

        BeanDefinitionLoader loader = createBeanDefinitionLoader(getBeanDefinitionRegistry(context), sources);

        if (this.beanNameGenerator != null) {
                loader.setBeanNameGenerator(this.beanNameGenerator);
        }
        if (this.resourceLoader != null) {
                loader.setResourceLoader(this.resourceLoader);
        }
        if (this.environment != null) {
                loader.setEnvironment(this.environment);
        }

        loader.load();
}

protected BeanDefinitionLoader createBeanDefinitionLoader(BeanDefinitionRegistry registry, Object[] sources) {
        return new BeanDefinitionLoader(registry, sources);
}
```

下面，我们来一起揭开它的面纱~

`org.springframework.boot.BeanDefinitionLoader` ，BeanDefinition 加载器（Loader），负责 Spring Boot 中，读取 BeanDefinition 。其类上的注释如下：

2.1 构造方法

```
private final Object[] sources;


private final AnnotatedBeanDefinitionReader annotatedReader;


private final XmlBeanDefinitionReader xmlReader;


private BeanDefinitionReader groovyReader;


private final ClassPathBeanDefinitionScanner scanner;


private ResourceLoader resourceLoader;
```

```
BeanDefinitionLoader(BeanDefinitionRegistry registry, Object... sources) {
    Assert.notNull(registry, "Registry must not be null");
    Assert.notEmpty(sources, "Sources must not be empty");
    this.sources = sources;

    this.annotatedReader = new AnnotatedBeanDefinitionReader(registry);

    this.xmlReader = new XmlBeanDefinitionReader(registry);


    if (isGroovyPresent()) {
        this.groovyReader = new GroovyBeanDefinitionReader(registry);
    }

    this.scanner = new ClassPathBeanDefinitionScanner(registry);
    this.scanner.addExcludeFilter(new ClassExcludeFilter(sources));
}
```

- <1> 处，设置 sources 属性。它来自方法参数 Object... sources ，来自 SpringApplication#getAllSources() 方法，代码如下：

```
private Set<Class<?>> primarySources;

private Set<String> sources = new LinkedHashSet<>();
```

```
public Set<Object> getAllSources() {
        Set<Object> allSources = new LinkedHashSet<>();
        if (!CollectionUtils.isEmpty(this.primarySources)) {
                allSources.addAll(this.primarySources);
        }
        if (!CollectionUtils.isEmpty(this.sources)) {
                allSources.addAll(this.sources);
        }
        return Collections.unmodifiableSet(allSources);
}
```

  - 默认情况下，返回的结果是 Spring#run(Class<?> primarySource, String... args) 方法的 Class<?> primarySource 的方法参数。例如说： MVCApplication 。

- <2.1> 处，创建 AnnotatedBeanDefinitionReader 对象，设置给 annotatedReader 属性。

- <2.2> 处，创建 XmlBeanDefinitionReader 对象，设置给 xmlReader 属性。

- <2.3> 处，创建 GroovyBeanDefinitionReader 对象，设置给 groovyReader 属性。其中， #isGroovyPresent() 方法，判断是否可以使用 Groovy 。代码如下：

```
private boolean isGroovyPresent() {
        return ClassUtils.isPresent("groovy.lang.MetaClass", null);
}
```

- <2.4> 处，创建 ClassPathBeanDefinitionScanner 对象，并设置给 scanner 属性。其中，ClassExcludeFilter 是 BeanDefinitionLoader 的内部静态类，继承 AbstractTypeHierarchyTraversingFilter 抽象类，用于排除对 sources 的扫描。代码如下：

```
private static class ClassExcludeFilter extends AbstractTypeHierarchyTraversingFilter {

        private final Set<String> classNames = new HashSet<>();

        ClassExcludeFilter(Object... sources) {
                super(false, false);
                for (Object source : sources) {
                        if (source instanceof Class<?>) {
                                this.classNames.add(((Class<?>) source).getName());
                        }
                }
        }

        @Override
        protected boolean matchClassName(String className) {
                return this.classNames.contains(className);
        }
```

```
    }
```

- 如果不排除，则会出现重复读取 BeanDefinition 的情况。

2.2 setBeanNameGenerator

#setBeanNameGenerator(BeanNameGenerator beanNameGenerator) 方法，代码如下：

```
public void setBeanNameGenerator(BeanNameGenerator beanNameGenerator) {
        this.annotatedReader.setBeanNameGenerator(beanNameGenerator);
        this.xmlReader.setBeanNameGenerator(beanNameGenerator);
        this.scanner.setBeanNameGenerator(beanNameGenerator);
}
```

2.3 setResourceLoader

#setResourceLoader(ResourceLoader resourceLoader) 方法，代码如下：

```
public void setResourceLoader(ResourceLoader resourceLoader) {
        this.resourceLoader = resourceLoader;
        this.xmlReader.setResourceLoader(resourceLoader);
        this.scanner.setResourceLoader(resourceLoader);
}
```

2.4 setEnvironment

#setEnvironment(ConfigurableEnvironment environment) 方法，代码如下：

```
public void setEnvironment(ConfigurableEnvironment environment) {
        this.annotatedReader.setEnvironment(environment);
        this.xmlReader.setEnvironment(environment);
        this.scanner.setEnvironment(environment);
}
```

2.5 load

#load() 方法，执行 BeanDefinition 加载。代码如下：

```
public int load() {
    int count = 0;

    for (Object source : this.sources) {
        count += load(source);
    }
    return count;
}

private int load(Object source) {
    Assert.notNull(source, "Source must not be null");

    if (source instanceof Class<?>) {
        return load((Class<?>) source);
    }

    if (source instanceof Resource) {
        return load((Resource) source);
    }

    if (source instanceof Package) {
        return load((Package) source);
    }
```

```
    if (source instanceof CharSequence) {
        return load((CharSequence) source);
    }

    throw new IllegalArgumentException("Invalid source type " + source.getClass());
}
```

- 针对不同 source 类型，执行不同的加载逻辑。
- <1> 处，如果是 Class 类型，则调用 #load(Class<?> source) 方法，使用 AnnotatedBeanDefinitionReader 执行加载。详细解析，见 「2.5.1 load(Class<?> source)」 。
- <2> 处，如果是 Resource 类型，则调用 #load(Resource source) 方法，使用 XmlBeanDefinitionReader 执行加载。详细解析，见 「2.5.2 load(Resource source)」 。
- <3> 处，如果是 Package 类型，则调用 #load(Package source) 方法，使用 ClassPathBeanDefinitionScanner 执行加载。详细解析，见 「2.5.3 load(Package source)」 。
- <4> 处，如果是 CharSequence 类型，则调用 #load(CharSequence source) 方法，各种尝试去加载。例如说 source 为 "classpath:/applicationContext.xml" 。详细解析，见 「2.5.4 load(CharSequence source)」 。
- <5> 处，无法处理的类型，抛出 IllegalArgumentException 异常。

2.5.1 load(Class<?> source)

#load(Class<?> source) 方法，使用 AnnotatedBeanDefinitionReader 执行加载。代码如下：

```
private int load(Class<?> source) {

    if (isGroovyPresent()
            && GroovyBeanDefinitionSource.class.isAssignableFrom(source)) {

        GroovyBeanDefinitionSource loader = BeanUtils.instantiateClass(source, GroovyBeanDefinitionSource.class);
        load(loader);
    }

    if (isComponent(source)) {
        this.annotatedReader.register(source);
        return 1;
    }
    return 0;
}
```

- <1> 处，调用 #isComponent(Class<?> type) 方法，判断是否为 Component 。代码如下：

```
private boolean isComponent(Class<?> type) {


    if (AnnotationUtils.findAnnotation(type, Component.class) != null) {
        return true;
    }


    if (type.getName().matches(".*\\$_.*closure.*") || type.isAnonymousClass()
            || type.getConstructors() == null || type.getConstructors().length == 0) {
        return false;
    }
    return true;
}
```

  - 因为 Configuration 类，上面有 @Configuration 注解，而 @Configuration 上，自带 @Component 注解，所以该方法返回 true 。

- <2> 处，调用 AnnotatedBeanDefinitionReader#register(Class<?>... annotatedClasses) 方法，执行注册。

2.5.2 load(Resource source)

#load(Resource source) 方法，使用 XmlBeanDefinitionReader 执行加载。代码如下：

```
private int load(Resource source) {
```

```
        if (source.getFilename().endsWith(".groovy")) {
            if (this.groovyReader == null) {
                throw new BeanDefinitionStoreException("Cannot load Groovy beans without Groovy on classpath");
            }
            return this.groovyReader.loadBeanDefinitions(source);
        }

        return this.xmlReader.loadBeanDefinitions(source);
    }
```

- 调用 XmlBeanDefinitionReader#loadBeanDefinitions(Resource resource) 方法，从 XML 中加载 BeanDefinition 。

2.5.3 load(Package source)

#load(Package source) 方法，使用 ClassPathBeanDefinitionScanner 执行加载。代码如下：

```
    private int load(Package source) {
            return this.scanner.scan(source.getName());
    }
```

2.5.4 load(CharSequence source)

#load(CharSequence source) 方法，各种尝试去加载。代码如下：

> 按照 source 是 Class > Resource > Package 的顺序，尝试加载。

```
    private int load(CharSequence source) {

        String resolvedSource = this.xmlReader.getEnvironment().resolvePlaceholders(source.toString());

        try {
            return load(ClassUtils.forName(resolvedSource, null));
        } catch (IllegalArgumentException | ClassNotFoundException ex) {

        }

        Resource[] resources = findResources(resolvedSource);
        int loadCount = 0;
        boolean atLeastOneResourceExists = false;
        for (Resource resource : resources) {
            if (isLoadCandidate(resource)) {
                atLeastOneResourceExists = true;
                loadCount += load(resource);
            }
        }
        if (atLeastOneResourceExists) {
            return loadCount;
        }

        Package packageResource = findPackage(resolvedSource);
        if (packageResource != null) {
            return load(packageResource);
        }

        throw new IllegalArgumentException("Invalid source '" + resolvedSource + "'");
    }
```

- <1> 处，解析 source 。因为，有可能里面带有占位符。
- <2> 处，将 source 转换成 Class ，然后执行 「2.5.1 load(Class<?> source)」 的流程。
- <3> 处，尝试按照 Resource 进行加载。
- <3.1> 处，调用 #findResources(String source) 方法，获得 source 对应的 Resource 数组。代码如下：

```
    private Resource[] findResources(String source) {
```

```
    ResourceLoader loader = (this.resourceLoader != null) ? this.resourceLoader : new PathMatchingResourcePatternResolver();
    try {

        if (loader instanceof ResourcePatternResolver) {
            return ((ResourcePatternResolver) loader).getResources(source);
        }

        return new Resource[] { loader.getResource(source) };
    } catch (IOException ex) {
        throw new IllegalStateException("Error reading source '" + source + "'");


    }
}
```

- <3.2> 处，遍历 resources 数组，调用 #isLoadCandidate(Resource resource) 方法，判断是否为符合条件的 Resource 。代码如下：

```
private boolean isLoadCandidate(Resource resource) {

    if (resource == null || !resource.exists()) {
        return false;
    }

    if (resource instanceof ClassPathResource) {




        String path = ((ClassPathResource) resource).getPath();
        if (path.indexOf('.') == -1) {
            try {
                return Package.getPackage(path) == null;
            } catch (Exception ex) {

            }
        }
    }

    return true;
}
```

- <3.3> 处，执行 「2.5.2 load(Resource source)」 的流程。
- <3.4> 处，有加载到，则认为成功，返回。
- <4> 处，尝试按照 Package 进行加载。
- <4.1> 处，调用 #findPackage(CharSequence source) 方法，获得 Package 对象。代码如下：

```
private Package findPackage(CharSequence source) {

    Package pkg = Package.getPackage(source.toString());
    if (pkg != null) {
        return pkg;
    }
    try {


        ResourcePatternResolver resolver = new PathMatchingResourcePatternResolver(getClass().getClassLoader());

        Resource[] resources = resolver.getResources(ClassUtils.convertClassNameToResourcePath(source.toString()) + "/*.class");

        for (Resource resource : resources) {

            String className = StringUtils.stripFilenameExtension(resource.getFilename());

            load(Class.forName(source.toString() + "." + className));
            break;
        }
    } catch (Exception ex) {

    }

    return Package.getPackage(source.toString());
}
```

  - 虽然逻辑比较复杂，我们只需要看看 <X> 处的前半部分的逻辑即可。

- <4.2> 处，执行 「2.5.3 load(Package source)」 的流程。
- <5> 处，无法处理，抛出 IllegalArgumentException 异常。

简单小文一篇。如果胖友不了解 Spring BeanDefinition ，可以补充看看 《【死磕 Spring】—— IoC 之加载 BeanDefinition》 文章。

如果想要测试 SpringFactoriesLoader 的各种情况，可以调试 BeanDefinitionLoaderTests 提供的单元测试。

参考和推荐如下文章：

- 一个努力的码农 《spring boot 源码解析 8-SpringApplication#run 第 8 步》
- oldflame-Jm 《Spring boot 源码分析 - BeanDefinitionLoader（7）》