
回到首页

芋道源码 —— 知识星球

我是一段不羁的公告！
记得给芳芳这 3 个项目加油，添加一个 STAR 噢。
https://github.com/YunaiV/SpringBoot-Labs
https://github.com/YunaiV/onemall
https://github.com/YunaiV/ruoyi-vue-pro
2021-02-07
Spring Boot

精尽 Spring Boot 源码分析 —— BeanDefinitionLoader

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

```
// SpringApplication.java

protected void load(ApplicationContext context, Object[] sources) {
 if (logger.isDebugEnabled()) {
        logger.debug("Loading source " + StringUtils.arrayToCommaDelimitedString(sources));
    }
 // <1> 创建 BeanDefinitionLoader 对象
    BeanDefinitionLoader loader = createBeanDefinitionLoader(getBeanDefinitionRegistry(context), sources);
 // <2> 设置 loader 的属性
 if (this.beanNameGenerator != null) {
        loader.setBeanNameGenerator(this.beanNameGenerator);
    }
 if (this.resourceLoader != null) {
        loader.setResourceLoader(this.resourceLoader);
    }
 if (this.environment != null) {
        loader.setEnvironment(this.environment);
    }
 // <3> 执行 BeanDefinition 加载
    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~

2. BeanDefinitionLoader

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：

```
// BeanDefinitionLoader.java

/**
 * Loads bean definitions from underlying sources, including XML and JavaConfig. Acts as a
 * simple facade over {@link AnnotatedBeanDefinitionReader},
 * {@link XmlBeanDefinitionReader} and {@link ClassPathBeanDefinitionScanner}. See
 * {@link SpringApplication} for the types of sources that are supported.
 */
```

2.1 构造方法

```
// BeanDefinitionLoader.java

/**
 * 来源的数组
 */
private final Object[] sources;

/**
 * 注解的 BeanDefinition 读取器
 */
private final AnnotatedBeanDefinitionReader annotatedReader;
/**
 * XML 的 BeanDefinition 读取器
 */
private final XmlBeanDefinitionReader xmlReader;
/**
 * Groovy 的 BeanDefinition 读取器
 */
private BeanDefinitionReader groovyReader;
/**
 * Classpath 的 BeanDefinition 扫描器
 */
private final ClassPathBeanDefinitionScanner scanner;

/**
 * 资源加载器
 */
private ResourceLoader resourceLoader;

/**
 * Create a new {@link BeanDefinitionLoader} that will load beans into the specified
 * {@link BeanDefinitionRegistry}.
 * @param registry the bean definition registry that will contain the loaded beans
 * @param sources the bean sources
 */
BeanDefinitionLoader(BeanDefinitionRegistry registry, Object... sources) {
    Assert.notNull(registry, "Registry must not be null");
    Assert.notEmpty(sources, "Sources must not be empty");
    this.sources = sources; // <1>
```

```
    // 创建 AnnotatedBeanDefinitionReader 对象
    this.annotatedReader = new AnnotatedBeanDefinitionReader(registry);
    // 创建 XmlBeanDefinitionReader 对象
    this.xmlReader = new XmlBeanDefinitionReader(registry);
    // 创建 GroovyBeanDefinitionReader 对象
    if (isGroovyPresent()) {
        this.groovyReader = new GroovyBeanDefinitionReader(registry);
    }
    // 创建 ClassPathBeanDefinitionScanner 对象
    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：

```
// BeanDefinitionLoader.java

/**
 * 主要的 Java Config 类的数组
 */
private Set<Class<?>> primarySources;

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

/**
 * Return an immutable set of all the sources that will be added to an
 * ApplicationContext when {@link #run(String...)} is called. This method combines any
 * primary sources specified in the constructor with any additional ones that have
 * been {@link #setSources(Set) explicitly set}.
 * @return an immutable set of all sources
 */
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：

```
// BeanDefinitionLoader.java
```

```
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：

```
// BeanDefinitionLoader.java

/**
 * Simple {@link TypeFilter} used to ensure that specified {@link Class} sources are
 * not accidentally re-added during scanning.
 */
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：

```
// BeanDefinitionLoader.java

/**
 * Set the bean name generator to be used by the underlying readers and scanner.
 * @param beanNameGenerator the bean name generator
 */
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：

```
// BeanDefinitionLoader.java

/**
 * Set the resource loader to be used by the underlying readers and scanner.
 * @param resourceLoader the resource loader
 */
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：

```
// BeanDefinitionLoader.java

/**
 * Set the environment to be used by the underlying readers and scanner.
 * @param environment the environment
 */
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：

```
// BeanDefinitionLoader.java

/**
 * Load the sources into the reader.
 * @return the number of loaded beans
 */
public int load() {
    int count = 0;
    //  遍历 sources 数组，逐个加载
    for (Object source : this.sources) {
        count += load(source);
    }
    return count;
}

private int load(Object source) {
    Assert.notNull(source, "Source must not be null");
    // <1> 如果是 Class 类型，则使用 AnnotatedBeanDefinitionReader 执行加载
    if (source instanceof Class<?>) {
```

```
            return load((Class<?>) source);
        }
        // <2> 如果是 Resource 类型，则使用 XmlBeanDefinitionReader 执行加载
        if (source instanceof Resource) {
            return load((Resource) source);
        }
        // <3> 如果是 Package 类型，则使用 ClassPathBeanDefinitionScanner 执行加载
        if (source instanceof Package) {
            return load((Package) source);
        }
        // <4> 如果是 CharSequence 类型，则各种尝试去加载
        if (source instanceof CharSequence) {
            return load((CharSequence) source);
        }
        // <5> 无法处理的类型，抛出 IllegalArgumentException 异常
        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：

```
// BeanDefinitionLoader.java

private int load(Class<?> source) {
    // Groovy 相关，暂时忽略
    if (isGroovyPresent()
            && GroovyBeanDefinitionSource.class.isAssignableFrom(source)) {
        // Any GroovyLoaders added in beans{} DSL can contribute beans here
        GroovyBeanDefinitionSource loader = BeanUtils.instantiateClass(source, GroovyBeanDefinitionSource.class);
        load(loader);
    }
    // <1> 如果是 Component ，则执行注册
    if (isComponent(source)) {
        this.annotatedReader.register(source); // <2>
        return 1;
    }
    return 0;
}
```

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

```
// BeanDefinitionLoader.java

private boolean isComponent(Class<?> type) {
    // This has to be a bit of a guess. The only way to be sure that this type is
    // eligible is to make a bean definition out of it and try to instantiate it.
    // 如果有 @Component 注解，则返回 true
    if (AnnotationUtils.findAnnotation(type, Component.class) != null) {
        return true;
    }
    // Nested anonymous classes are not eligible for registration, nor are groovy
    // closures
    // 暂时忽略
    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：

```
// BeanDefinitionLoader.java

private int load(Resource source) {
    // Groovy 相关，暂时忽略
    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);
    }
    // 使用 XmlBeanDefinitionReader 加载 BeanDefinition
    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：

```
// BeanDefinitionLoader.java

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。

```
// BeanDefinitionLoader.java

private int load(CharSequence source) {
    // <1> 解析 source 。因为，有可能里面带有占位符。
    String resolvedSource = this.xmlReader.getEnvironment().resolvePlaceholders(source.toString());
    // <2> 尝试按照 Class 进行加载
    // Attempt as a Class
    try {
        return load(ClassUtils.forName(resolvedSource, null));
    } catch (IllegalArgumentException | ClassNotFoundException ex) {
        // swallow exception and continue
    }
    // <3> 尝试按照 Resource 进行加载
    // Attempt as resources
    Resource[] resources = findResources(resolvedSource); // <3.1>
    int loadCount = 0;
    boolean atLeastOneResourceExists = false;
    for (Resource resource : resources) {
        if (isLoadCandidate(resource)) { // <3.2>
            atLeastOneResourceExists = true;
            loadCount += load(resource); // <3.3>
        }
    }
    if (atLeastOneResourceExists) { // <3.4> 有加载到，则认为成功，返回。
        return loadCount;
    }
    // Attempt as package
    // <4> 尝试按照 Package 进行加载
    Package packageResource = findPackage(resolvedSource); // <4.1>
    if (packageResource != null) {
        return load(packageResource); // <4.2>
    }
    // <5> 无法处理，抛出 IllegalArgumentException 异常
    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：

```
// BeanDefinitionLoader.java

private Resource[] findResources(String source) {
    // 创建 ResourceLoader 对象
    ResourceLoader loader = (this.resourceLoader != null) ? this.resourceLoader : new PathMatchingResourcePatte
    try {
        // 获得 Resource 数组
```

```
            if (loader instanceof ResourcePatternResolver) {
                return ((ResourcePatternResolver) loader).getResources(source);
            }
            // 获得 Resource 对象
            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：

```
    // BeanDefinitionLoader.java

    private boolean isLoadCandidate(Resource resource) {
        // 不存在，则返回 false
        if (resource == null || !resource.exists()) {
            return false;
        }
        // 判断 resource 是 ClassPathResource 类，不是一个 package
        if (resource instanceof ClassPathResource) {
            // A simple package without a '.' may accidentally get loaded as an XML
            // document if we're not careful. The result of getInputStream() will be
            // a file list of the package content. We double check here that it's not
            // actually a package.
            String path = ((ClassPathResource) resource).getPath();
            if (path.indexOf('.') == -1) {
                try {
                    return Package.getPackage(path) == null;
                } catch (Exception ex) {
                    // Ignore
                }
            }
        }
        // 返回 true ，符合条件
        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：

```
    // BeanDefinitionLoader.java

    private Package findPackage(CharSequence source) {
        // <X> 获得 source 对应的 Package 。如果存在，则返回
        Package pkg = Package.getPackage(source.toString());
        if (pkg != null) {
            return pkg;
        }
        try {
```

```
                // Attempt to find a class in this package
                // 创建 ResourcePatternResolver 对象
                ResourcePatternResolver resolver = new PathMatchingResourcePatternResolver(getClass().getClassLoader());
                // 尝试加载 source 目录下的 class 们
                Resource[] resources = resolver.getResources(ClassUtils.convertClassNameToResourcePath(source.toString
                // 遍历 resources 数组
                for (Resource resource : resources) {
                    // 获得类名
                    String className = StringUtils.stripFilenameExtension(resource.getFilename());
                    // 按照 Class 进行加载 BeanDefinition
                    load(Class.forName(source.toString() + "." + className));
                    break;
                }
            } catch (Exception ex) {
                // swallow exception and continue
            }
            // 返回 Package
            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。

666. 彩蛋

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）》

文章目录

回到首页