<https://www.jianshu.com/p/6b8b4bf98d02>

[首页](https://www.jianshu.com/)[下载APP](https://www.jianshu.com/apps?utm_medium=desktop&utm_source=navbar-apps)

# **Spring对@ComponentScan注解的解析处理**

[数齐](https://www.jianshu.com/u/8232dfd27d77)关注赞赏支持

# **Spring对@ComponentScan注解的解析处理**

[](https://www.jianshu.com/u/8232dfd27d77)

[数齐](https://www.jianshu.com/u/8232dfd27d77" \t "/Users/liu123/Documents\\x/_blank)关注

2018.05.27 10:23:03字数 755阅读 5,454

上篇文章[Spring对@Configuration的解析处理](https://www.jianshu.com/p/27bf08ef9744" \t "/Users/liu123/Documents\\x/_blank)在解析@ComponentScan注解的时候跳过了，只是说了一下他大体的过程，今天我们看下他的在扫描BeanDefinition的处理流程。

代码开始于org.springframework.context.annotation.ComponentScanAnnotationParser#parse中的下面一段代码

Set<BeanDefinitionHolder> scannedBeanDefinitions =

this.componentScanParser.parse(componentScan, sourceClass.getMetadata().getClassName());

首先这个componentScanParser创建于ConfigurationClassParser的构造方法中

this.componentScanParser = new ComponentScanAnnotationParser(

resourceLoader, environment, componentScanBeanNameGenerator, registry);

下面我们看看具体的解析过程org.springframework.context.annotation.ComponentScanAnnotationParser#parse

public Set<BeanDefinitionHolder> parse(AnnotationAttributes componentScan, final String declaringClass) {

Assert.state(this.environment != null, "Environment must not be null");

Assert.state(this.resourceLoader != null, "ResourceLoader must not be null");

ClassPathBeanDefinitionScanner scanner =

new ClassPathBeanDefinitionScanner(this.registry, componentScan.getBoolean("useDefaultFilters"));

scanner.setEnvironment(this.environment);

scanner.setResourceLoader(this.resourceLoader);

Class<? extends BeanNameGenerator> generatorClass = componentScan.getClass("nameGenerator");

boolean useInheritedGenerator = BeanNameGenerator.class == generatorClass;

scanner.setBeanNameGenerator(useInheritedGenerator ? this.beanNameGenerator :

BeanUtils.instantiateClass(generatorClass));

ScopedProxyMode scopedProxyMode = componentScan.getEnum("scopedProxy");

if (scopedProxyMode != ScopedProxyMode.DEFAULT) {

scanner.setScopedProxyMode(scopedProxyMode);

}

else {

Class<? extends ScopeMetadataResolver> resolverClass = componentScan.getClass("scopeResolver");

scanner.setScopeMetadataResolver(BeanUtils.instantiateClass(resolverClass));

}

scanner.setResourcePattern(componentScan.getString("resourcePattern"));

for (AnnotationAttributes filter : componentScan.getAnnotationArray("includeFilters")) {

for (TypeFilter typeFilter : typeFiltersFor(filter)) {

scanner.addIncludeFilter(typeFilter);

}

}

for (AnnotationAttributes filter : componentScan.getAnnotationArray("excludeFilters")) {

for (TypeFilter typeFilter : typeFiltersFor(filter)) {

scanner.addExcludeFilter(typeFilter);

}

}

boolean lazyInit = componentScan.getBoolean("lazyInit");

if (lazyInit) {

scanner.getBeanDefinitionDefaults().setLazyInit(true);

}

Set<String> basePackages = new LinkedHashSet<String>();

String[] basePackagesArray = componentScan.getAliasedStringArray("basePackages", ComponentScan.class, declaringClass);

for (String pkg : basePackagesArray) {

String[] tokenized = StringUtils.tokenizeToStringArray(this.environment.resolvePlaceholders(pkg),

ConfigurableApplicationContext.CONFIG\_LOCATION\_DELIMITERS);

basePackages.addAll(Arrays.asList(tokenized));

}

for (Class<?> clazz : componentScan.getClassArray("basePackageClasses")) {

basePackages.add(ClassUtils.getPackageName(clazz));

}

if (basePackages.isEmpty()) {

basePackages.add(ClassUtils.getPackageName(declaringClass));

}

scanner.addExcludeFilter(new AbstractTypeHierarchyTraversingFilter(false, false) {

@Override

protected boolean matchClassName(String className) {

return declaringClass.equals(className);

}

});

return scanner.doScan(StringUtils.toStringArray(basePackages));

}

这段代码说白了就是委托给ClassPathBeanDefinitionScanner来做事情，随后调用org.springframework.context.annotation.ClassPathBeanDefinitionScanner#doScan来进行真实的扫描逻辑。

protected Set<BeanDefinitionHolder> doScan(String... basePackages) {

Assert.notEmpty(basePackages, "At least one base package must be specified");

Set<BeanDefinitionHolder> beanDefinitions = new LinkedHashSet<BeanDefinitionHolder>();

for (String basePackage : basePackages) {

Set<BeanDefinition> candidates = findCandidateComponents(basePackage); //关注点1

for (BeanDefinition candidate : candidates) {

ScopeMetadata scopeMetadata = this.scopeMetadataResolver.resolveScopeMetadata(candidate);

candidate.setScope(scopeMetadata.getScopeName());

String beanName = this.beanNameGenerator.generateBeanName(candidate, this.registry);

if (candidate instanceof AbstractBeanDefinition) {

postProcessBeanDefinition((AbstractBeanDefinition) candidate, beanName);

}

if (candidate instanceof AnnotatedBeanDefinition) {

AnnotationConfigUtils.processCommonDefinitionAnnotations((AnnotatedBeanDefinition) candidate);

}

if (checkCandidate(beanName, candidate)) {

BeanDefinitionHolder definitionHolder = new BeanDefinitionHolder(candidate, beanName);

definitionHolder = AnnotationConfigUtils.applyScopedProxyMode(scopeMetadata, definitionHolder, this.registry);

beanDefinitions.add(definitionHolder);

registerBeanDefinition(definitionHolder, this.registry);

}

}

}

return beanDefinitions;

}

我们先看下关注点1，根据包名来获取BeanDefinitionHolder-findCandidateComponents(basePackage)

/\*\*

\* Scan the class path for candidate components.

\* @param basePackage the package to check for annotated classes

\* @return a corresponding Set of autodetected bean definitions

\*/

public Set<BeanDefinition> findCandidateComponents(String basePackage) {

Set<BeanDefinition> candidates = new LinkedHashSet<BeanDefinition>();

try {

String packageSearchPath = ResourcePatternResolver.CLASSPATH\_ALL\_URL\_PREFIX +

resolveBasePackage(basePackage) + "/" + this.resourcePattern;

Resource[] resources = this.resourcePatternResolver.getResources(packageSearchPath);

boolean traceEnabled = logger.isTraceEnabled();

boolean debugEnabled = logger.isDebugEnabled();

for (Resource resource : resources) {

if (traceEnabled) {

logger.trace("Scanning " + resource);

}

if (resource.isReadable()) {

try {

MetadataReader metadataReader = this.metadataReaderFactory.getMetadataReader(resource);

if (isCandidateComponent(metadataReader)) {

ScannedGenericBeanDefinition sbd = new ScannedGenericBeanDefinition(metadataReader);

sbd.setResource(resource);

sbd.setSource(resource);

if (isCandidateComponent(sbd)) {

if (debugEnabled) {

logger.debug("Identified candidate component class: " + resource);

}

candidates.add(sbd);

}

else {

if (debugEnabled) {

logger.debug("Ignored because not a concrete top-level class: " + resource);

}

}

}

else {

if (traceEnabled) {

logger.trace("Ignored because not matching any filter: " + resource);

}

}

}

catch (Throwable ex) {

throw new BeanDefinitionStoreException(

"Failed to read candidate component class: " + resource, ex);

}

}

else {

if (traceEnabled) {

logger.trace("Ignored because not readable: " + resource);

}

}

}

}

catch (IOException ex) {

throw new BeanDefinitionStoreException("I/O failure during classpath scanning", ex);

}

return candidates;

}

首先根据包名拼接要扫描资源的路径，随后交给resourcePatternResolver来加载资源，遍历这些资源，根据资源找到对应的MetadataReader的实例，通过isCandidateComponent(MetadataReader metadataReader)来根据excludeFilters和includeFilters判断是否可以进行下一步的操作，如果这个资源被排除的filter匹配上，就返回false，代表不是我们所需要的。如果被包含的filter匹配上，并且他还要通过条件判断isConditionMatch的话，返回true，代表是我们需要的资源，可以进行下一步的操作。这里我们插一句Spring有默认的includ类型的filter实现，如果上层传入的话，就是用上层传入的，否则就使用默认的，默认的是扫描@Component注解，详情请看org.springframework.context.annotation.ClassPathScanningCandidateComponentProvider#registerDefaultFilters

protected void registerDefaultFilters() {

this.includeFilters.add(new AnnotationTypeFilter(Component.class));

ClassLoader cl = ClassPathScanningCandidateComponentProvider.class.getClassLoader();

try {

this.includeFilters.add(new AnnotationTypeFilter(

((Class<? extends Annotation>) ClassUtils.forName("javax.annotation.ManagedBean", cl)), false));

logger.debug("JSR-250 'javax.annotation.ManagedBean' found and supported for component scanning");

}

catch (ClassNotFoundException ex) {

// JSR-250 1.1 API (as included in Java EE 6) not available - simply skip.

}

try {

this.includeFilters.add(new AnnotationTypeFilter(

((Class<? extends Annotation>) ClassUtils.forName("javax.inject.Named", cl)), false));

logger.debug("JSR-330 'javax.inject.Named' annotation found and supported for component scanning");

}

catch (ClassNotFoundException ex) {

// JSR-330 API not available - simply skip.

}

}

通过上面的org.springframework.context.annotation.ClassPathScanningCandidateComponentProvider#isCandidateComponent(org.springframework.core.type.classreading.MetadataReader)的初步判断后，可以继续下去的话，就会创建一个ScannedGenericBeanDefinition类型的实例，随后我们进入了第二个判断这个ScannedGenericBeanDefinition是否使我们需要的的方法org.springframework.context.annotation.ClassPathScanningCandidateComponentProvider#isCandidateComponent(org.springframework.beans.factory.annotation.AnnotatedBeanDefinition)

/\*\*

\* Determine whether the given bean definition qualifies as candidate.

\* <p>The default implementation checks whether the class is concrete

\* (i.e. not abstract and not an interface). Can be overridden in subclasses.

\* @param beanDefinition the bean definition to check

\* @return whether the bean definition qualifies as a candidate component

\*/

protected boolean isCandidateComponent(AnnotatedBeanDefinition beanDefinition) {

return (beanDefinition.getMetadata().isConcrete() && beanDefinition.getMetadata().isIndependent());

}

这个就是判断我们改将什么类型的类扫进去作为BeanDefinition，就是根据isConcrete和isIndependent两个方法判断，我们看一下这写方法的注释，来自原ClassMetadata类

/\*\*

\* Return whether the underlying class represents a concrete class,

\* i.e. neither an interface nor an abstract class.

\*/

boolean isConcrete();

/\*\*

\* Determine whether the underlying class is independent,

\* i.e. whether it is a top-level class or a nested class

\* (static inner class) that can be constructed independent

\* from an enclosing class.

\*/

boolean isIndependent();

符合两个条件，第一个是具体的，就是他不是一个接口，也不是一个抽象类。第二个是Independent独立的，就是顶级类或者是静态内部类，说白了就是可以单独进行实例化的，满足这两个条件就会被扫描成具体的Bean进行下一步的操作。我们要扫描的也就是这些家伙，findCandidateComponents我们就说的差不多了。我们说说下面的操作，目前为止我们已经得到了这写BeanDefination，但是他们不完整，还需要填充，所以下面有针对AbstractBeanDefinition和AnnotatedBeanDefinition两种类型的BeanDefinition的数据填充。填充完后，就进行注册到Spring容器中。

OK,上面就是受@ComponentScan注解作用的扫描BeanDefination的全部过程了。

4人点赞

[spring](https://www.jianshu.com/nb/8797901" \t "/Users/liu123/Documents\\x/_blank)

"小礼物走一走，来简书关注我"

赞赏支持还没有人赞赏，支持一下

[](https://www.jianshu.com/u/8232dfd27d77)

[数齐](https://www.jianshu.com/u/8232dfd27d77" \o "数齐" \t "/Users/liu123/Documents\\x/_blank)90后代码狗，专注于钻研框架源码，喜欢一切新奇的事物，现在任职于众安科技，希望找到更多志同道合...

总资产311 (约28.75元)共写了6.0W字获得391个赞共846个粉丝

关注

### 被以下专题收入，发现更多相似内容

[IMG_259程序员](https://www.jianshu.com/c/NEt52a" \t "/Users/liu123/Documents\\x/_blank)[IMG_260微服务架构和实践](https://www.jianshu.com/c/3f476518d832" \t "/Users/liu123/Documents\\x/_blank)[IMG_261IT技术篇](https://www.jianshu.com/c/f380da1de1c9" \t "/Users/liu123/Documents\\x/_blank)[IMG_262spring精...](https://www.jianshu.com/c/3b091ee73d70" \t "/Users/liu123/Documents\\x/_blank)[IMG_263我爱编程](https://www.jianshu.com/c/7847442e0728" \t "/Users/liu123/Documents\\x/_blank)[IMG_264Java 杂谈](https://www.jianshu.com/c/0b39448c4e08" \t "/Users/liu123/Documents\\x/_blank)

[](https://www.jianshu.com/u/8232dfd27d77)

[数齐](https://www.jianshu.com/u/8232dfd27d77" \t "/Users/liu123/Documents\\x/_blank)

关注

总资产311 (约28.75元)

[Arthas实践](https://www.jianshu.com/p/9b9cfa0da429" \t "/Users/liu123/Documents\\x/_blank)

阅读 148

[插入式注解处理器](https://www.jianshu.com/p/554c5491bea6" \t "/Users/liu123/Documents\\x/_blank)

阅读 1,847

### 推荐阅读

[字节跳动抖音社招后台开发工程师面经](https://www.jianshu.com/p/b9dea1ca51df" \t "/Users/liu123/Documents\\x/_blank)

阅读 13,349

[蚂蚁二面，面试官问我零拷贝的实现原理，当场懵了…](https://www.jianshu.com/p/2581342317ce" \t "/Users/liu123/Documents\\x/_blank)

阅读 76,776

[Spring Cloud gateway 七 Sentinel 注解方式使用](https://www.jianshu.com/p/654732a0e819" \t "/Users/liu123/Documents\\x/_blank)

阅读 2,075

[最新的高级java面试](https://www.jianshu.com/p/e56ef17e72a3" \t "/Users/liu123/Documents\\x/_blank)

阅读 19,212

[idea2020注册码，亲测可用！](https://www.jianshu.com/p/09c9afae9a05" \t "/Users/liu123/Documents\\x/_blank)

阅读 79,605

广告

评论0

赞4