<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

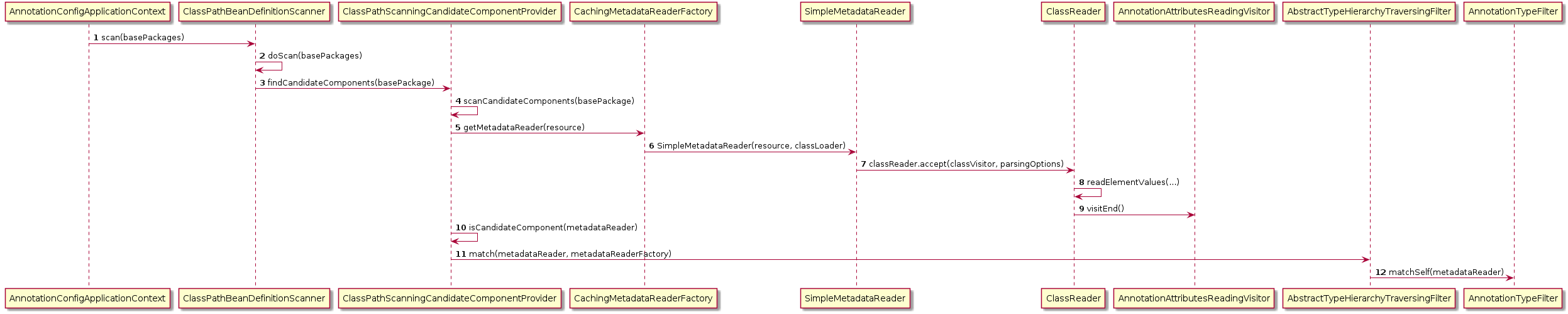
[IMG_256](https://www.cnblogs.com/wolf-bin/)

# [wolf\_lb](https://www.cnblogs.com/wolf-bin/)

* [博客园](https://www.cnblogs.com/)
* [首页](https://www.cnblogs.com/wolf-bin/)
* [新随笔](https://i.cnblogs.com/EditPosts.aspx?opt=1)
* [联系](https://msg.cnblogs.com/send/wolf_lb)
* [订阅](https://www.cnblogs.com/wolf-bin/rss/)
* [管理](https://i.cnblogs.com/)

# [Spring注解Component原理源码解析](https://www.cnblogs.com/wolf-bin/p/11667208.html)

在实际开发中，我们经常使用Spring的@Component、@Service、@Repository以及 @Controller等注解来实现bean托管给Spring容器管理。Spring是怎么样实现的呢？我们一起跟着源码看看整个过程吧！

照旧，先看调用时序图：

public AnnotationConfigApplicationContext(String... basePackages) {

this();

scan(basePackages);

refresh();

}

Spring启动时，会去扫描指定包下的文件。

public void scan(String... basePackages) {

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

this.scanner.scan(basePackages);

}

对应时序图方法1，ClassPathBeanDefinitionScanner#scan。交给ClassPathBeanDefinitionScanner处理。

ClassPathBeanDefinitionScanner 初始化时设置了注解过滤器

public ClassPathBeanDefinitionScanner(BeanDefinitionRegistry registry, boolean useDefaultFilters,Environment environment, @Nullable ResourceLoader resourceLoader) {

Assert.notNull(registry, "BeanDefinitionRegistry must not be null");

this.registry = registry;

if (useDefaultFilters) {

// 注册注解过滤器

registerDefaultFilters();

}

setEnvironment(environment);

setResourceLoader(resourceLoader);

}

protected void registerDefaultFilters() {

// 添加Component类型

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));

}

catch (ClassNotFoundException ex) {

}

try {

this.includeFilters.add(new AnnotationTypeFilter(

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

}

catch (ClassNotFoundException ex) {

}

}

在includeFilters添加了Component，ManagedBean两种注解类型。后面用来过滤加载到的class文件是否需要交给Spring容器管理。

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

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

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

for (String basePackage : basePackages) {

// 扫描包下有Spring Component注解，并且生成BeanDefinition

Set<BeanDefinition> candidates = findCandidateComponents(basePackage);

for (BeanDefinition candidate : candidates) {

// 设置scope，默认是singleton

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);

// 生成代理类信息

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

beanDefinitions.add(definitionHolder);

// 注册到Spring容器

registerBeanDefinition(definitionHolder, this.registry);

}

}

}

return beanDefinitions;

}

对应时序图方法2，ClassPathBeanDefinitionScanner#doScan。该方法对包下class文件解析，符合Spring容器管理的类生成BeanDefinition，并注册到容器中。

扫描包下的class文件，把有Component注解的封装BeanDefinition列表返回。

public Set<BeanDefinition> findCandidateComponents(String basePackage) {

if (this.componentsIndex != null && indexSupportsIncludeFilters()) {

return addCandidateComponentsFromIndex(this.componentsIndex, basePackage);

}

else {

return scanCandidateComponents(basePackage);

}

}

对应时序图方法3，ClassPathScanningCandidateComponentProvider#findCandidateComponents。

private Set<BeanDefinition> scanCandidateComponents(String basePackage) {

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

try {

// classpath\*:basePackage/\*\*/\*.class

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

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

// 获取 basePackage 包下的 .class 文件资源

Resource[] resources = getResourcePatternResolver().getResources(packageSearchPath);

for (Resource resource : resources) {

// 判断是否可读

if (resource.isReadable()) {

try {

// 获取.class文件类信息

MetadataReader metadataReader = getMetadataReaderFactory().getMetadataReader(resource);

if (isCandidateComponent(metadataReader)) {

ScannedGenericBeanDefinition sbd = new ScannedGenericBeanDefinition(metadataReader);

sbd.setResource(resource);

sbd.setSource(resource);

if (isCandidateComponent(sbd)) {

candidates.add(sbd);

}

}

} catch (Throwable ex) {

throw new BeanDefinitionStoreException("Failed to read candidate component class: " + resource, ex);

}

}

}

}

catch (IOException ex) {

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

}

return candidates;

}

对应时序图方法4，ClassPathScanningCandidateComponentProvider#scanCandidateComponents。

public MetadataReader getMetadataReader(Resource resource) throws IOException {

if (this.metadataReaderCache instanceof ConcurrentMap) {

// No synchronization necessary...

MetadataReader metadataReader = this.metadataReaderCache.get(resource);

if (metadataReader == null) {

// 获取.class类元信息

metadataReader = super.getMetadataReader(resource);

this.metadataReaderCache.put(resource, metadataReader);

}

return metadataReader;

}

else if (this.metadataReaderCache != null) {

synchronized (this.metadataReaderCache) {

MetadataReader metadataReader = this.metadataReaderCache.get(resource);

if (metadataReader == null) {

metadataReader = super.getMetadataReader(resource);

this.metadataReaderCache.put(resource, metadataReader);

}

return metadataReader;

}

}

else {

return super.getMetadataReader(resource);

}

}

对应时序图方法5，CachingMetadataReaderFactory#getMetadataReader。 super.getMetadataReader(resource) 调用的是 SimpleMetadataReaderFactory#getMetadataReader。

public MetadataReader getMetadataReader(Resource resource) throws IOException {

// 默认是SimpleMetadataReader实例

return new SimpleMetadataReader(resource, this.resourceLoader.getClassLoader());

}

SimpleMetadataReader(Resource resource, @Nullable ClassLoader classLoader) throws IOException {

// 加载.class文件

InputStream is = new BufferedInputStream(resource.getInputStream());

ClassReader classReader;

try {

classReader = new ClassReader(is);

}

catch (IllegalArgumentException ex) {

throw new NestedIOException("ASM ClassReader failed to parse class file - " +

"probably due to a new Java class file version that isn't supported yet: " + resource, ex);

}

finally {

is.close();

}

AnnotationMetadataReadingVisitor visitor = new AnnotationMetadataReadingVisitor(classLoader);

// 解析.class元信息

classReader.accept(visitor, ClassReader.SKIP\_DEBUG);

this.annotationMetadata = visitor;

this.classMetadata = visitor;

this.resource = resource;

}

对应时序图方法6，SimpleMetadataReader#SimpleMetadataReader。 组装SimpleMetadataReader。

public void accept(

final ClassVisitor classVisitor,

final Attribute[] attributePrototypes,

final int parsingOptions) {

Context context = new Context();

context.attributePrototypes = attributePrototypes;

context.parsingOptions = parsingOptions;

context.charBuffer = new char[maxStringLength];

... 省略代码

// Visit the RuntimeVisibleAnnotations attribute.

if (runtimeVisibleAnnotationsOffset != 0) {

int numAnnotations = readUnsignedShort(runtimeVisibleAnnotationsOffset);

int currentAnnotationOffset = runtimeVisibleAnnotationsOffset + 2;

while (numAnnotations-- > 0) {

// Parse the type\_index field.

String annotationDescriptor = readUTF8(currentAnnotationOffset, charBuffer);

currentAnnotationOffset += 2;

// 这里面封装Spring Component注解

currentAnnotationOffset =

readElementValues(classVisitor.visitAnnotation(annotationDescriptor,true),

currentAnnotationOffset,true,charBuffer);

}

}

... 省略代码

}

对应时序图方法7，ClassReader#accept。该方法把二进制的.class文件解析组装到AnnotationMetadataReadingVisitor

private int readElementValues(

final AnnotationVisitor annotationVisitor,

final int annotationOffset,

final boolean named,

final char[] charBuffer) {

... 省略代码

if (annotationVisitor != null) {

// 主要逻辑还在这里面

annotationVisitor.visitEnd();

}

return currentOffset;

}

对应时序图方法8，ClassReader#readElementValues。

public void visitEnd() {

super.visitEnd();

Class<? extends Annotation> annotationClass = this.attributes.annotationType();

if (annotationClass != null) {

... 省略代码

// 过滤java.lang.annotation包下的注解，及保留Spring注解

if (!AnnotationUtils.isInJavaLangAnnotationPackage(annotationClass.getName())) {

try {

// 获取该类上的所有注解

Annotation[] metaAnnotations = annotationClass.getAnnotations();

if (!ObjectUtils.isEmpty(metaAnnotations)) {

Set<Annotation> visited = new LinkedHashSet<>();

for (Annotation metaAnnotation : metaAnnotations) {

// 过滤java.lang.annotation包下的注解，及保留Spring注解

recursivelyCollectMetaAnnotations(visited, metaAnnotation);

}

// 封装需要的注解

if (!visited.isEmpty()) {

Set<String> metaAnnotationTypeNames = new LinkedHashSet<>(visited.size());

for (Annotation ann : visited) {

metaAnnotationTypeNames.add(ann.annotationType().getName());

}

this.metaAnnotationMap.put(annotationClass.getName(), metaAnnotationTypeNames);

}

}

}

catch (Throwable ex) {

}

}

}

}

对应时序图方法9，AnnotationAttributesReadingVisitor#visitEnd。过滤掉 java.lang.annotation 包下的注解，然后把剩下的注解放到metaAnnotationMap。

protected boolean isCandidateComponent(MetadataReader metadataReader) throws IOException {

for (TypeFilter tf : this.excludeFilters) {

if (tf.match(metadataReader, getMetadataReaderFactory())) {

return false;

}

}

for (TypeFilter tf : this.includeFilters) {

if (tf.match(metadataReader, getMetadataReaderFactory())) {

return isConditionMatch(metadataReader);

}

}

return false;

}

对应时序图方法10，ClassPathScanningCandidateComponentProvider#isCandidateComponent。使用前面提过的ClassPathBeanDefinitionScanner初始化时设置的注解类型过滤器，includeFilters 包含ManagedBean和Component类型。

public boolean match(MetadataReader metadataReader, MetadataReaderFactory metadataReaderFactory)

throws IOException {

if (matchSelf(metadataReader)) {

return true;

}

... 省略代码

return false;

}

对应时序图方法11，AbstractTypeHierarchyTraversingFilter#match。

protected boolean matchSelf(MetadataReader metadataReader) {

AnnotationMetadata metadata = metadataReader.getAnnotationMetadata();

return metadata.hasAnnotation(this.annotationType.getName()) ||

(this.considerMetaAnnotations && metadata.hasMetaAnnotation(this.annotationType.getName()));

}

对应时序图方法12，AnnotationTypeFilter#matchSelf。判断类的metadata中是否包含Component。

总结@Component到Spring bean容器管理过程。第一步，初始化时设置了Component类型过滤器；第二步，根据指定扫描包扫描.class文件，生成Resource对象；第三步、解析.class文件并注解归类，生成MetadataReader对象；第四步、使用第一步的注解过滤器过滤出有@Component类；第五步、生成BeanDefinition对象；第六步、把BeanDefinition注册到Spring容器。以上是@Component注解原理，@Service、@Controller和@Repository上都有@Component修饰，所以原理是一样的。

**标签:**[Spring](https://www.cnblogs.com/wolf-bin/tag/Spring/)**,**[@Component](https://www.cnblogs.com/wolf-bin/tag/@Component/)**,**[Java](https://www.cnblogs.com/wolf-bin/tag/Java/)

**[好文要顶](javascript:void(0);)** **[关注我](javascript:void(0);)** **[收藏该文](javascript:void(0);)** **[IMG_258](javascript:void(0);)** **[IMG_259](javascript:void(0);)**

[IMG_260](https://home.cnblogs.com/u/wolf-bin/)

[wolf\_lb](https://home.cnblogs.com/u/wolf-bin/)  
[关注 - 2](https://home.cnblogs.com/u/wolf-bin/followees/)  
[粉丝 - 3](https://home.cnblogs.com/u/wolf-bin/followers/)

[+加关注](javascript:void(0);)

0

0

[«](https://www.cnblogs.com/wolf-bin/p/11604227.html)上一篇： [ApplicationListener原理分析](https://www.cnblogs.com/wolf-bin/p/11604227.html" \o "发布于 2019-09-28 18:20)  
[»](https://www.cnblogs.com/wolf-bin/p/11772918.html)下一篇： [Arthas查看Spring配置](https://www.cnblogs.com/wolf-bin/p/11772918.html" \o "发布于 2019-10-31 18:25)

posted @ 2019-10-13 17:35  [wolf\_lb](https://www.cnblogs.com/wolf-bin/)  阅读(496)  评论(0)  [编辑](https://i.cnblogs.com/EditPosts.aspx?postid=11667208)  [收藏](javascript:void(0))

[刷新评论](javascript:void(0);)[刷新页面](https://www.cnblogs.com/wolf-bin/p/11667208.html)[返回顶部](https://www.cnblogs.com/wolf-bin/p/11667208.html" \l "top)

**注册用户登录后才能发表评论，请 [登录](javascript:void(0);) 或 [注册](javascript:void(0);)， [访问](https://www.cnblogs.com/) 网站首页。**

[【推荐】超50万行VC++源码: 大型组态工控、电力仿真CAD与GIS源码库](http://www.ucancode.com/index.htm" \t "/Users/liu123/Documents\\x/_blank)  
[【推荐】阿里云双11返场来袭，热门产品低至一折等你来抢！](http://click.aliyun.com/m/1000081987/" \t "/Users/liu123/Documents\\x/_blank)  
[【活动】开发者上云必备，腾讯云1核4G 2M云服务器11元/月起](https://cloud.tencent.com/act/developer?fromSource=gwzcw.3196335.3196335.3196335&utm_medium=cpc&utm_id=gwzcw.3196335.3196335.3196335" \t "/Users/liu123/Documents\\x/_blank)  
[【推荐】百度智能云岁末感恩季，明星产品低至1元新老用户畅享](https://cloud.baidu.com/campaign/Promotion-20191111/index.html?track=cp:dsp|pf:pc|pp:chui-bokeyuan-huodong-19shuangshiyiganenji-BCC-cpaxingshi-191210|pu:cpa-xingshi|ci:2019syj|kw:2172212" \t "/Users/liu123/Documents\\x/_blank)  
[【活动】京东云限时优惠1.5折购云主机，最高返价值1000元礼品！](https://www.jdcloud.com/cn/activity/newUser?utm_source=DMT_cnblogs&utm_medium=CH&utm_campaign=09vm&utm_term=Virtual-Machines" \t "/Users/liu123/Documents\\x/_blank)

**相关博文：**  
· [Spring注解@Repository、@Service、@Controller、@Component](https://www.cnblogs.com/EasonJim/p/7347534.html" \o "Spring注解@Repository、@Service、@Controller、@Component" \t "/Users/liu123/Documents\\x/_blank)  
· [spring常用注解使用解析](https://www.cnblogs.com/intsmaze/p/5224661.html" \o "spring常用注解使用解析" \t "/Users/liu123/Documents\\x/_blank)  
· [spring注解注入:<context:component-scan>详解](https://www.cnblogs.com/exe19/p/5391712.html" \o "spring注解注入:<context:component-scan>详解" \t "/Users/liu123/Documents\\x/_blank)  
· [Spring中@Component注解,@Controller注解详解](https://www.cnblogs.com/zivxiaowei/p/3489144.html" \o "Spring中@Component注解,@Controller注解详解" \t "/Users/liu123/Documents\\x/_blank)  
· [Spring注解实现原理](https://www.cnblogs.com/lsx1993/p/4793124.html" \o "Spring注解实现原理" \t "/Users/liu123/Documents\\x/_blank)  
» [更多推荐...](https://recomm.cnblogs.com/blogpost/11667208" \t "/Users/liu123/Documents\\x/_blank)

[阿里技术3年大合辑免费电子书一键下载](https://developer.aliyun.com/article/712655?utm_content=g_1000088930" \t "/Users/liu123/Documents\\x/_blank)

**最新 IT 新闻**:  
· [国内首例！72岁高位截瘫患者用意念喝可乐](https://news.cnblogs.com/n/653777/" \t "/Users/liu123/Documents\\x/_blank)  
· [别只知道马斯克，太空互联网还有这些重磅选手](https://news.cnblogs.com/n/653776/" \t "/Users/liu123/Documents\\x/_blank)  
· [拼多多年货节数据：百亿补贴下，苹果/华为/小米分列智能手机前三](https://news.cnblogs.com/n/653775/" \t "/Users/liu123/Documents\\x/_blank)  
· [中国个体从业人数超1.7亿，数字技术成背后“推手”](https://news.cnblogs.com/n/653774/" \t "/Users/liu123/Documents\\x/_blank)  
· [掌管百度战投三年后，马东敏把权力重新移交李彦宏](https://news.cnblogs.com/n/653773/" \t "/Users/liu123/Documents\\x/_blank)  
» [更多新闻...](https://news.cnblogs.com/" \o "IT 新闻" \t "/Users/liu123/Documents\\x/_blank)

### 公告

昵称： [wolf\_lb](https://home.cnblogs.com/u/wolf-bin/)  
园龄： [2年](https://home.cnblogs.com/u/wolf-bin/" \o "入园时间：2017-12-26)  
粉丝： [3](https://home.cnblogs.com/u/wolf-bin/followers/)  
关注： [2](https://home.cnblogs.com/u/wolf-bin/followees/)

[+加关注](javascript:void(0))

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | [<](javascript:void(0);) | 2020年1月 | [>](javascript:void(0);) | | | | | | | |
| **日** | **一** | **二** | **三** | **四** | **五** | **六** |
| 29 | 30 | 31 | 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| 26 | 27 | 28 | 29 | 30 | 31 | 1 |
| 2 | 3 | 4 | 5 | 6 | 7 | 8 |

### 搜索

### 我的标签

* [JVM](https://www.cnblogs.com/wolf-bin/tag/JVM/)(8)
* [Java](https://www.cnblogs.com/wolf-bin/tag/Java/)(6)
* [kafka](https://www.cnblogs.com/wolf-bin/tag/kafka/)(3)
* [Spring](https://www.cnblogs.com/wolf-bin/tag/Spring/)(3)
* [spring cloud](https://www.cnblogs.com/wolf-bin/tag/spring cloud/)(1)
* [并发](https://www.cnblogs.com/wolf-bin/tag/%E5%B9%B6%E5%8F%91/)(1)
* [观察者模式](https://www.cnblogs.com/wolf-bin/tag/%E8%A7%82%E5%AF%9F%E8%80%85%E6%A8%A1%E5%BC%8F/)(1)
* [@Component](https://www.cnblogs.com/wolf-bin/tag/@Component/)(1)
* [ApplicationListener源码解析](https://www.cnblogs.com/wolf-bin/tag/ApplicationListener%E6%BA%90%E7%A0%81%E8%A7%A3%E6%9E%90/)(1)
* [Arthas](https://www.cnblogs.com/wolf-bin/tag/Arthas/)(1)
* [更多](https://www.cnblogs.com/wolf-bin/tag/)

### 随笔档案

* [2019年10月(2)](https://www.cnblogs.com/wolf-bin/archive/2019/10.html" \t "/Users/liu123/Documents\\x/)
* [2019年9月(2)](https://www.cnblogs.com/wolf-bin/archive/2019/09.html" \t "/Users/liu123/Documents\\x/)
* [2019年3月(1)](https://www.cnblogs.com/wolf-bin/archive/2019/03.html" \t "/Users/liu123/Documents\\x/)
* [2018年8月(1)](https://www.cnblogs.com/wolf-bin/archive/2018/08.html" \t "/Users/liu123/Documents\\x/)
* [2018年7月(1)](https://www.cnblogs.com/wolf-bin/archive/2018/07.html" \t "/Users/liu123/Documents\\x/)
* [2018年5月(2)](https://www.cnblogs.com/wolf-bin/archive/2018/05.html" \t "/Users/liu123/Documents\\x/)
* [2018年4月(1)](https://www.cnblogs.com/wolf-bin/archive/2018/04.html" \t "/Users/liu123/Documents\\x/)
* [2018年3月(1)](https://www.cnblogs.com/wolf-bin/archive/2018/03.html" \t "/Users/liu123/Documents\\x/)
* [2018年1月(8)](https://www.cnblogs.com/wolf-bin/archive/2018/01.html" \t "/Users/liu123/Documents\\x/)
* [2017年12月(3)](https://www.cnblogs.com/wolf-bin/archive/2017/12.html" \t "/Users/liu123/Documents\\x/)

### 最新评论

* [1. Re:spring boot整合kafka](https://www.cnblogs.com/wolf-bin/p/8179001.html" \l "4300696)
* 弱弱问下,不写Config类也能运行,什么情况....
* --生活在阳光下

### 阅读排行榜

* [1. kafka topic消息分配partition规则（Java源码）(5452)](https://www.cnblogs.com/wolf-bin/p/9085370.html)
* [2. spring boot整合kafka(4913)](https://www.cnblogs.com/wolf-bin/p/8179001.html)
* [3. kafka对消费者分配分区规则（Java源码)(4224)](https://www.cnblogs.com/wolf-bin/p/9094309.html)
* [4. Protostuff序列化问题(3612)](https://www.cnblogs.com/wolf-bin/p/9269987.html)
* [5. Java验证手机号(1509)](https://www.cnblogs.com/wolf-bin/p/8177260.html)

### 评论排行榜

* [1. spring boot整合kafka(1)](https://www.cnblogs.com/wolf-bin/p/8179001.html)

### 推荐排行榜

* [1. Protostuff序列化问题(2)](https://www.cnblogs.com/wolf-bin/p/9269987.html)
* [2. kafka topic消息分配partition规则（Java源码）(1)](https://www.cnblogs.com/wolf-bin/p/9085370.html)
* [3. 类加载器(1)](https://www.cnblogs.com/wolf-bin/p/8280424.html)
* [4. Nacos配置服务原理(1)](https://www.cnblogs.com/wolf-bin/p/11532708.html)
* [5. 死亡的对象(1)](https://www.cnblogs.com/wolf-bin/p/8196759.html)

Copyright © 2020 wolf\_lb  
Powered by .NET Core 3.1.0 on Linux