Spring Configuration

Spring 使用AnnotationConfigApplicationContext，实现基于注解作为容器的配置信息来源来进行IoC的初始化，Spring提供4个注解，其作用与XML中bean定义效果一致，在于将组件交给Spring容器管理，组件的名称默认为类名，也可以自己修改：

* @Component，当对组件的层次难以定位的时候使用这个注解
* @Controller，表示控制层的组件
* @Service，表示业务逻辑层的组件
* @Repository，表示数据访问层的组件

通过AnnotationConfig，注解和配置信息都可以写到Java类中，使用方式是在指定的配置信息类上添加@Configuration注解，使用示例如下:

*@Configuration*

*public class BeanConfiguration {*

*@Bean*

*public Teacher teacher() {*

*Teacher teacher = new Teacher();*

*teacher.setName("Jerry");*

*return teacher;*

*}*

*......}*

添加该注解，用于明确指出该类是Bean配置的信息源。其使用方式如下：

*AnnotationConfigApplicationContext context =*

*new AnnotationConfigApplicationContext(BeanConfiguration.class);*

*Teacher teacher = (Teacher) context.getBean("teacher");*

其核心是AnnotationConfigApplicationContext。

# @Configuration注解

*@Target({ElementType.TYPE})*

*@Retention(RetentionPolicy.RUNTIME)*

*@Documented*

*@Component*

*public @interface Configuration {*

*@AliasFor(annotation = Component.class)*

*String value() default "";*

*}*

从定义来看，@Configuration本质上是@Component，因此@ComponentScan可以处理@Configuration注解的类。

对于@Configuration标记的类符合下面的要求：

* 配置类必须以类的形式提供（不能是工厂方法返回的实例），允许通过生成子类在运行时增强（cglib 动态代理）。
* 配置类不能是 final 类（没法动态代理）。
* 配置注解通常为了通过 @Bean 注解生成 Spring 容器管理的类，
* 配置类必须是非本地的（即不能在方法中声明，不能是 private）。
* 任何嵌套配置类都必须声明为static。
* @Bean 方法可能不会反过来创建进一步的配置类（也就是返回的 bean 如果带有 @Configuration，也不会被特殊处理，只会作为普通的 bean）。

# AnnotationConfigApplicationContext

在AnnotationConfigApplicationContext的类信息如下，构造函数：

*public AnnotationConfigApplicationContext(String... basePackages) {*

*this(); // 调用自身的构造函数*

*scan(basePackages); // 扫描指定package下的bean*

*refresh(); // 调用基类AbstractApplicationContext中的refresh()方法*

*}*

如果指定AnnotatedClass，则构造函数如下：

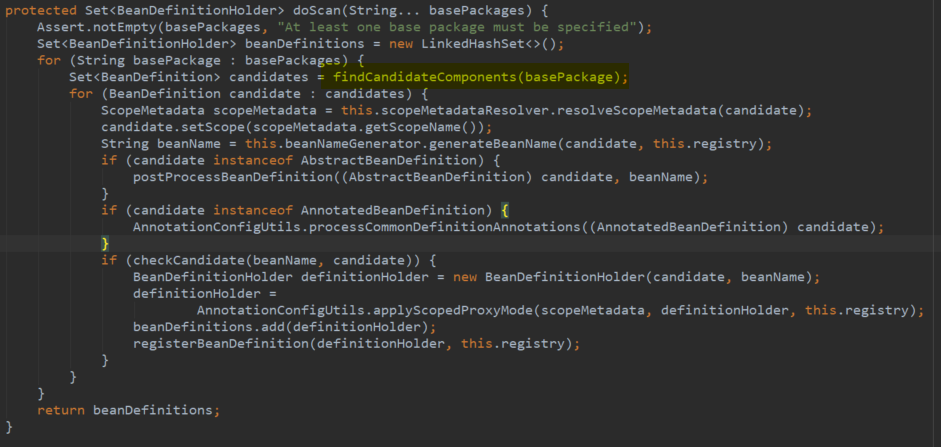
*public AnnotationConfigApplicationContext(Class<?>... annotatedClasses) {  
 this(); // 调用自身的构造函数   
 register(annotatedClasses); // 注册配置类   
 refresh(); // 调用基类AbstractApplicationContext中的refresh()方法  
}*

在AnnotationConfigApplicationContext中有两种重要的字段

* AnnotatedBeanDefinitionReader，加载Class类型的配置
* ClassPathBeanDefinitionScanner，扫描package下Bean注解的类，解析出BeanDefinition

## scan

AnnotationConfigApplicationContext构造函数传入basePackage后，通过scanner.scan从包中获取@Bean，然后注册到IoC中，其执行如下：



findCandidateComponents是基类ClassPathScanningCandidateComponentProvider中的方法，其核心如下：

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

*for (Resource resource : resources) {*

*if (resource.isReadable()) {*

*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);*

*}*

*}*

*......*

*return candidates;*

*}*

从类中读取Bean。

## **Register**

当传入Configure类，执行register方法，通过AnnotatedBeanDefinitionReader#register

*<T> void doRegisterBean(Class<T> annotatedClass, @Nullable Supplier<T> instanceSupplier, @Nullable String name, @Nullable Class<? extends Annotation>[] qualifiers, BeanDefinitionCustomizer... definitionCustomizers) {*

*AnnotatedGenericBeanDefinition abd =*

*new AnnotatedGenericBeanDefinition(annotatedClass);*

*......*

*abd.setInstanceSupplier(instanceSupplier);*

*ScopeMetadata scopeMetadata = this.scopeMetadataResolver.resolveScopeMetadata(abd);*

*abd.setScope(scopeMetadata.getScopeName());*

*String beanName = (name != null ?*

*name : this.beanNameGenerator.generateBeanName(abd, this.registry));*

*AnnotationConfigUtils.processCommonDefinitionAnnotations(abd);*

*......*

*for (BeanDefinitionCustomizer customizer : definitionCustomizers) {*

*customizer.customize(abd);*

*}*

*BeanDefinitionHolder definitionHolder = new BeanDefinitionHolder(abd, beanName);*

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

*BeanDefinitionReaderUtils.registerBeanDefinition(definitionHolder, this.registry);*

*}*

通过BeanNameGenerator获取类中的BeanDefinition。

https://blog.csdn.net/koflance/article/details/59304090

https://blog.csdn.net/lqzkcx3/article/details/78409305

https://blog.csdn.net/isea533/article/details/78072133?locationNum=7&fps=1