检查 getBean、doGetBean（单例or原型）

创建对象（一级二级三级缓存）

1. preInstantiateSingletons
2. 创建所有对象

* 遍历beanDefinitions
* 合并父容类属性值，将BeanDefinition转化为RootBeanDefinition

见文档2

* beanDefinition如果是非抽象、非懒加载、单例，两种bean，通过getBean()创建对象；否则不创建

1. FactoryBean
2. 普通bean
3. 处理所有对象

如果bean实现SmartInitializingSingleton接口，调用afterSingletonsInstantiated方法

1. getBean创建对象入口

调用doGetBean

1. doGetBean

getSingleton(String beanName)从一二三缓存中查找对象

（1）如果对象不为空且参数为空

通过getObjectForBeanInstance返回对象（可能返回对象本身，也可能返回FactoryBean创建的对象）

见文档3

（2）如果对象为空，且存在父类容器不为空，且当前容器BeanDefinition不存在当前bean

调用父类容器创建（Spring没有父容器，SpringCloud有父容器）

1. 其他情况
2. 先创建依赖对象
3. 如果是单例
4. 先调用getSingleton(String beanName, singletonFactory) 创建对象，然后加入一级缓存，并从二三级缓存删除

* getSingleton(String beanName, ObjectFactory<?> singletonFactory)

从一级缓存中查找，找到返回；

找不到校验（当前对象是否正在被销毁）通过，createBean创建对象；

校验不过，抛出异常

1. 最后通过getObjectForBeanInstance返回对象
2. 如果是原型，调用createBean创建对象，通过getObjectForBeanInstance返回对象
3. createBean
4. 获取Bean的Class对象，用于后续反射使用
5. prepareMethodOverrides验证及准备覆盖的方法（lookup-method、replace-method）

见文档4

1. 实例化前，RootBeanDefinition如果允许通过代理创建对象，则通过resolveBeforeInstantiation代理创建对象（BPP创建对象）

如果有自定义实现InstantiationAwareBeanPostProcessor，则通过它利用动态代理或者注解new创建对象

见文档5

1. 否则，通过doCreateBean创建对象
2. doCreateBean

BeanWrapper（实现类为BeanWrapperImpl），包装类：包含bean对象、class等等。

1. 创建对象

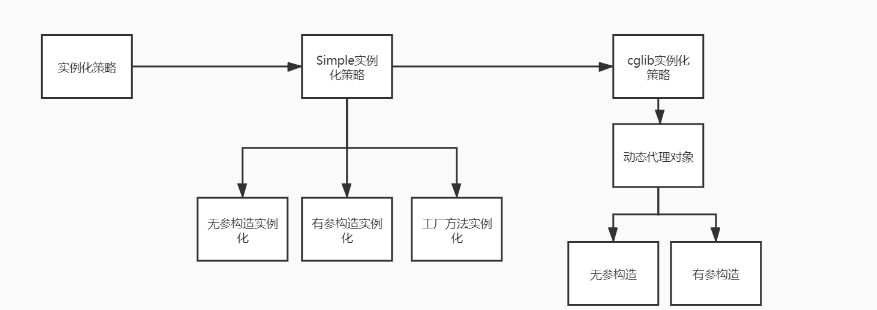
从缓存中获取包装类（包装类就包含bean对象、属性编辑器、属性转化器）

1. 包装类不为空，从缓存中获取
2. 包装类为空，通过createBeanInstance创建bean对象，并设置编辑器、转化器

创建bean对象，有5种方式（BPP方式在createBean中、FactoryBean）：

见文档5

* Supplier
* FactoryMethod
* 反射（根据参数选择构造器，通过反射创建）



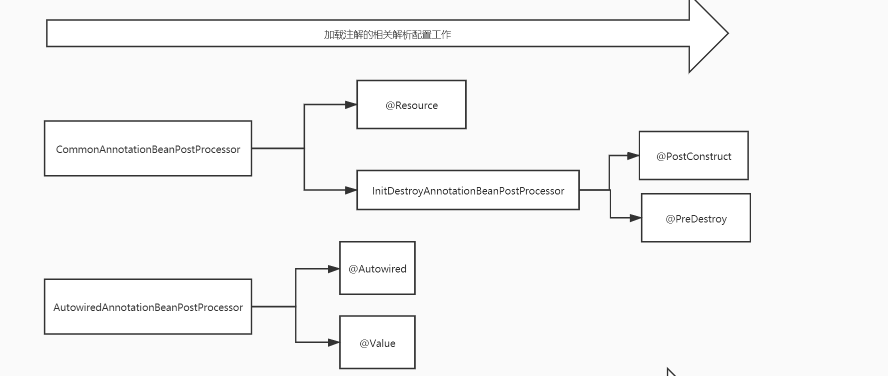
1. applyMergedBeanDefinitionPostProcessors，通过调用BPP处理注解，修改BeanDefinition

（获取方法名 or 属性名，加入到BeanDefinition集合中，或许调用）

主要有下面两种BPP：

CommonAwareBeanProcessor：处理@PostConstruct、@PreDestroy、@Resource注解

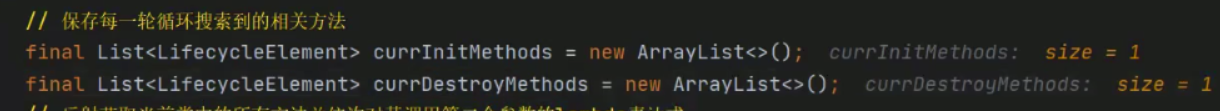
AutowiredAnnotationBeanPostProcessor：处理@Autowire、@Value注解

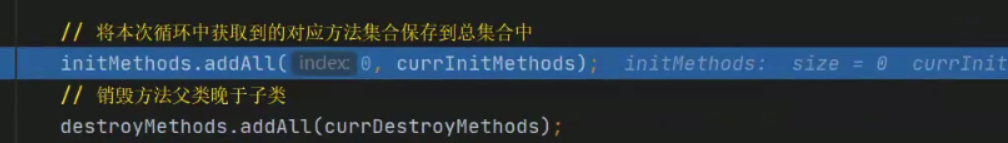


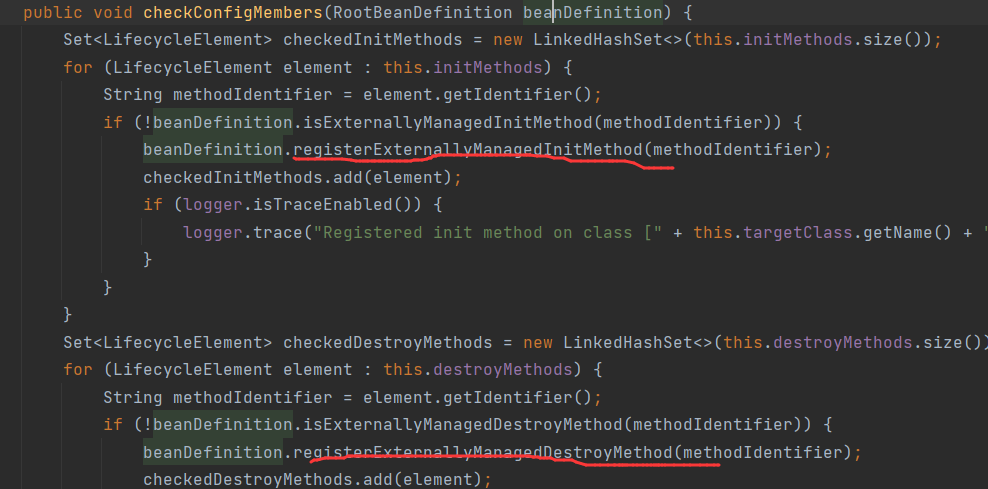
处理@Autowire、@Value、@Resource、@PostConstruct、@PreDestory标签

举例：

@PostConstruct：获取该注解的方法（包含父类），最后加入BeanDefinition里面，以便后续调用







1. 解决循环依赖，当前bean为单例，且允许循环引用

将bean、lamda表达式加入三级缓存，并移出二级缓存

1. populateBean填充属性

会涉及循环依赖

1. initializeBean初始化
2. 返回创建的完整对象
3. populateBean填充属性
4. 获取所有属性PropertyValues，即标签<property></property>
5. 获取属性值，即PropertyValue（属性名，属性值）->（name,value）

会调用getBean创建依赖对象

* autowireByName通过名字获取属性对象
* autowireByType通过类型获取属性对象

1. 解析属性值，通过applyPropertyValues函数，即PropertyValue（convertedValue）

（String->实际对象）把xml配置文件的value（可能是string、可能是beanName）转换为实际的对象（可能是），因此需要不同的解析器进行解析。例如：属性编辑器

* 验证工作
* 遍历PropertyValue，通过resolveValueIfNecessary根据不同类型解析属性值

1. 设置属性值，通过setPropertyValues函数
2. initializeBean初始化
3. 调用Aware接口，设置以下三个Aware属性：

BeanNameAware、BeanClassLoaderAware、BeanFactoryAware

1. 执行applyBeanPostProcessorsBeforeInitialization

* 设置其他Aware接口属性

ApplicationContextAware、EnvironmentAware、ResourceLoaderAware、

EmbeddedValueResolverAware、ApplicationEventPublisherAware、MessageSourceAware

* 解析其他注解（@post-construct、@Destroy）

CommonAnnotationBeanPostProcessor

（3）执行init-method方法

* 实现InitializingBean接口，直接调用afterPropertiesSet方法
* RootBeanDefinition且包含init-method方法，执行init-method方法

1. 执行applyBeanPostProcessorsAfterInitialization方法

通过cglib或者jdk生成具体对象的代理对象。

切面编程（AOP）：为对象增加属性或者方法