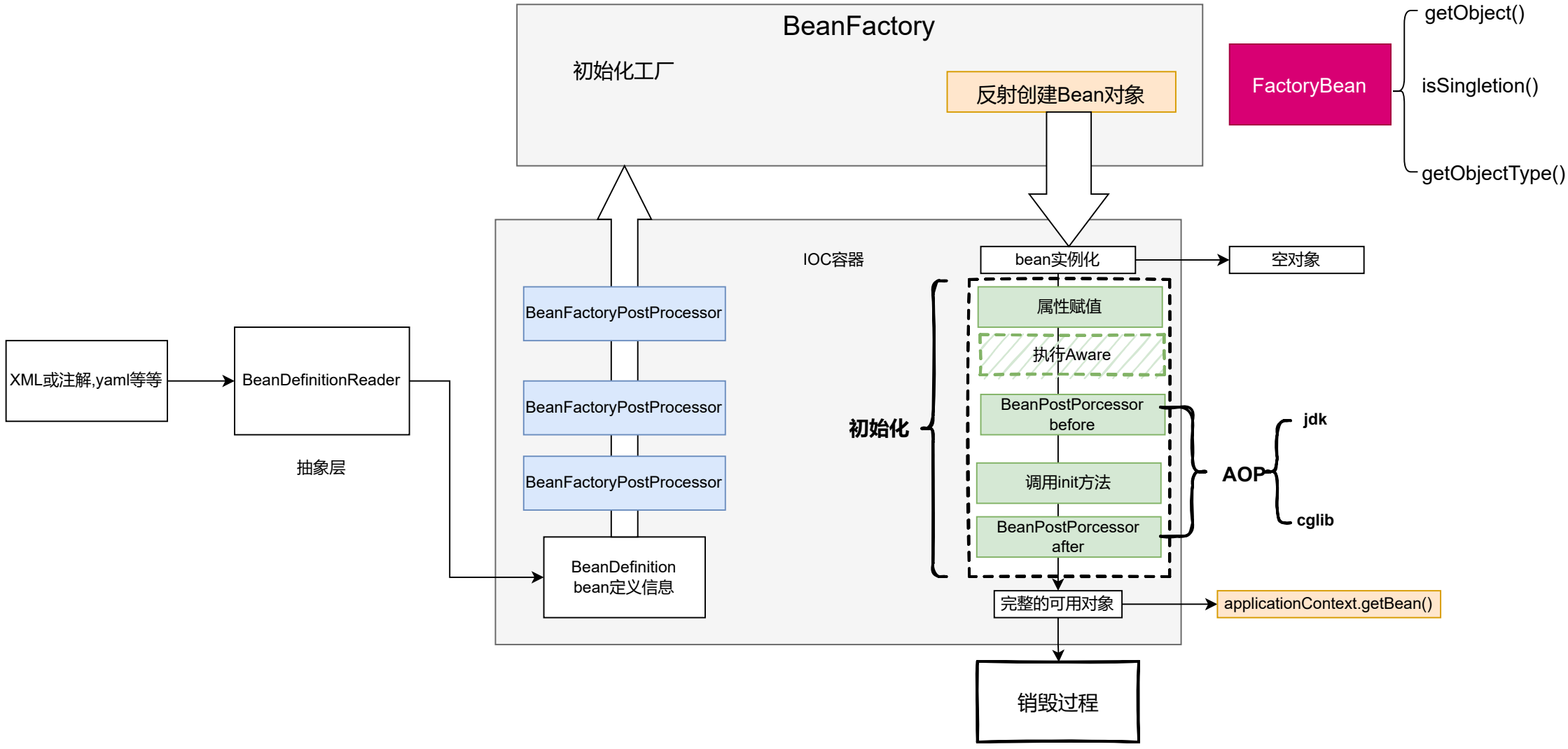


spring IOC的底层原理?

概念: 控制翻转:原来对象是由使用者进行控制的,有了Spring之后,吧整个对象的创建交给spring帮我们管理;DI依赖注入就是吧对应的属性注入到具体的对象中,可以使用@Autowaired注解等方式;容器: 使用map结构存储对象,在spring中存在三级缓存,singletonObject存放完整的bean对象,bean的整个生命周期从创建到使用到销毁都交给容器来管理

1. 创建bean工厂通过顶级的接口beanFactory的实现类例如DefaultListableFactory并设置一些参数
2. 加载解析bean的定义信息就是beanDefinition
3. BeanFactoryPostProcessor扩展点的处理,例如PlaceHolderConfigurSupport
4. BeanFactoryPostProcessor的注册功能,方便后续对bean对象具体的扩展功能
5. 通过反射将BeanDefinition对象实例化成具体的bean对象
6. bean对象的初始化过程,完成属性的填充,调用Aware子类的方法,完成BeanPostPocessor的前置方法,init方法和后置方法
7. 生成完整的bean对象,通过getBean()方法直接获取最后就是销毁,这是我对IOC整体的理解包涵一些详细过程

bean的生命周期

首先启动ApplicationContext
1. 创建bean工厂通过顶级的接口beanFactory的实现类例如DefaultListableFactory并设置一些参数
2. 加载解析bean的定义信息就是beanDefinition
3. BeanFactoryPostProcessor扩展点的处理,例如PlaceHolderConfigurSupport
4. BeanFactoryPostProcessor的注册功能,方便后续对bean对象具体的扩展功能
5. 通过反射将BeanDefinition对象实例化成具体的bean对象
6. 调用creatBean方法完成对象的初始化过程,完成属性的填充,调用Aware子类的方法,完成BeanPostPocessor的前置方法,init方法和后置方法
7. 生成完整的bean对象,通过getBean()方法直接获取最后就是销毁

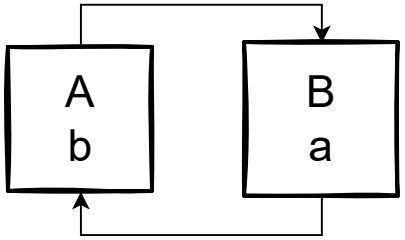
Spring AOP的底层原理?

aop是ioc的一个扩展功能,在ioc的整个流程中新增的扩展点BeanPostProcessor
aop主要用到了动态代理,对于接口的代理是jdk动态代理,对于类的代理是用到了cglib
1. 代理对象创建过程 advice 切面 切点
2. 通过jdk或者cglib生成代理对象
3. 在执行方法调用的时候,找到DynamicAdvisoredInterceptor类中的intercept()方法,从此方法开始执行
4. 根据定义好的通知生成拦截链,进行增强

BeanFactory和FactoryBean的区别?

两者都是用来创建Bean对象的
1. BeanFactory是spring的容器中的Bean工厂，Bean的创建，获取都是它来完成的，也就是说BeanFactory就是spring的容器，里面放入了所有的Bean，包括单例Bean，原型的Bean定义，也就是BeanDefinition,用它创建对象必须遵循严格的生命周期很复杂.
2. 如果想要简单的自定义某个对象交给spring管理就要实现FactoryBean接口，FactoryBean是BeanFactory的一种Bean，是一种特殊的Bean，FactoryBean中是只能放入一种类型的Bean，而BeanFactory中可以放入不同类型的Bean
3.FactoryBean方法有三个 getObject()：该方法就是你通过factoryBean放入容器的对象，也就是当你getBean的时候返回的就是spring调用getObject返回的Bean对象；getObjectType():FactoryBean代表返回对象的类型isSingleton：这是factoryBean默认的方法，你不需要实现，当然你也可以实现，默认是单例的bean
4..Mybatis中的Mapper是一个接口，而接口在spring容器中是不能实例化的，所以Mybatis就是利用了FactoryBean来重新构建了对象

循环依赖问题?



singltonOjects:一级缓存
earlySingltonObjects：二级缓存
singltonFactories：三级缓存

```
private final Map<String, Object> singletonObjects = new ConcurrentHashMap<>(256); //一级缓存
private final Map<String, Object> earlySingletonObjects = new HashMap<>(16); // 二级缓存
private final Map<String, ObjectFactory<?>> singletonFactories = new HashMap<>(16); // 三级缓存
```

singletonObjects：用于存放完全初始化好的 bean，从该缓存中取出的 bean 可以直接使用
earlySingletonObjects：提前曝光的单例对象的cache，存放原始的 bean 对象（尚未填充属性），用于解决循环依赖
singletonFactories：单例对象工厂的cache，存放 bean 工厂对象，用于解决循环依赖

循环依赖就是A对象属性依赖B对象,B对象依赖A对象产生了循环依赖问题
构造器注入无法解决循环依赖问题, setter注入利用三级缓存提前暴露对象解决,核心在于bean的实例化和初始化分开操作,将完成实例化但是没有完成初始化的半成品对象和完整状态的对象分开放入不同容器map中,这就有了 一级缓存和二级缓存,一级缓存放的是完整对象,二级缓存放的是半成品对象
三级缓存存放的是ObjectFactory是一个函数式接口,存放bean工厂对象,保证整个容器运行过程中同名的bean对象只有一个;所有的对象创建的时候都要优先放入到三级缓存中,在后续使用过程中出现循环依赖,调用Lambad,返回代理对象,然后将代理对象放入二级缓存移除三级缓存

spring中的设计模式?

单例模式:工厂中的bean默认都是单例的
原型模式
工厂模式:BeanFactory
策略模式:XmlBeanDefinitionReader, PropertiesBeanDefinitionReader
观察者模式:Listener event
适配器模式:Adapter
责任链模式:AOP生成拦截器链
代理模式:动态代理
装饰者模式:BeanWrapper

Spring事务管理是如何实现的?

spring的事务是由AOP实现的,首先生成具体的代理对象然后执行逻辑操作,事务是通过TransactionInterceptor来实现的

1. 解析事务的相关属性,来判断是否开启新事务
2. 获取连接,关闭自动提交开启事务
3.执行具体的sql操作
4. 如果执行过程中失败了就执行doRollBack()回滚事务
5.如果没有任何意外发生就doCommit()提交事务
6. 事务执行完毕之后清除相关的事务信息

Spring事务传播特性?

spring中事务有7种传播行为

- **REQUIRED**：默认的事务传播行为；需要事务：存在事务则使用已存在事务，否则创建新的事务；
- **SUPPORTS**：支持已存在事务：存在事务则使用已存在事务，否则以非事务方式运行；
- **MANDATORY**：强制使用事务：存在事务则使用已存在事务，否则抛出异常；
- **REQUIRES_NEW**：需要新事务：存在事务则挂起已存在事务，否则创建新事务；
- **NOT_SUPPORTED**：不支持事务：存在事务则挂起已存在事务，否则以非事务方式运行；
- **NEVER**：从不使用事务：存在事务则抛出异常，否则以非事务方式运行；
- **NESTED**：嵌套事务：存在事务则使创建保存点使用嵌套的事务，否则创建新的事务。

Spring中事务隔离级别?

read-uncommited	读未提交
read-commited	读提交 （Oracle数据库默认的隔离级别）
repeatable-read	可重复读（MySQL数据库默认的隔离级别）
serialized-read	序列化读