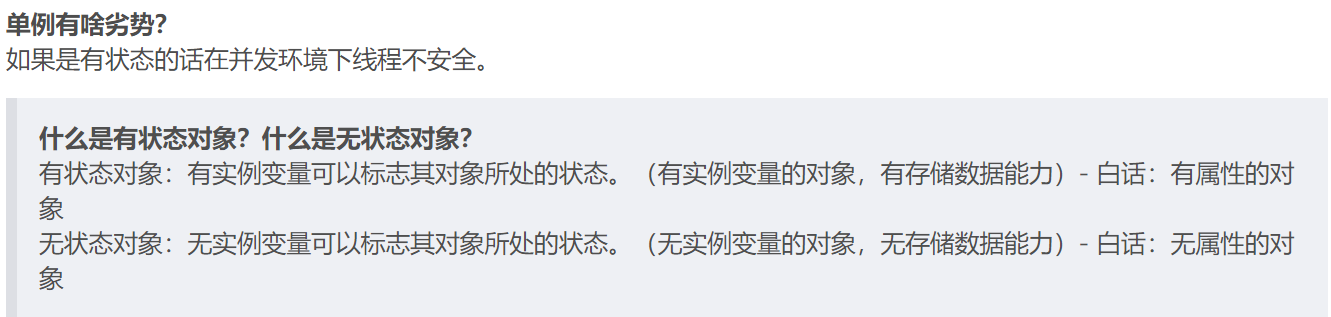


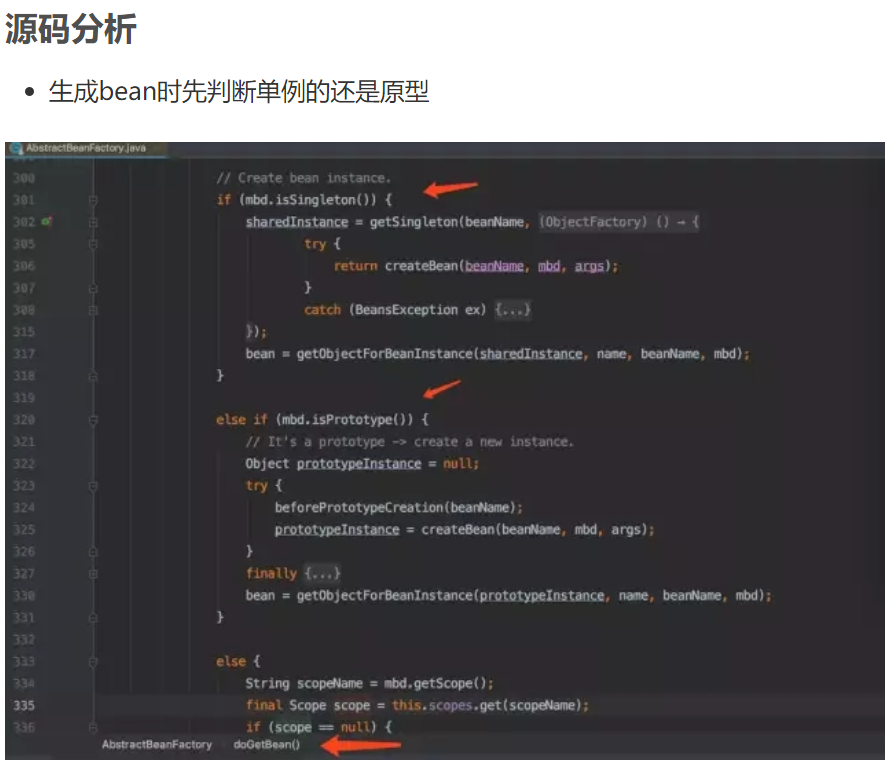
### Spring里面的bean都是单例吗？

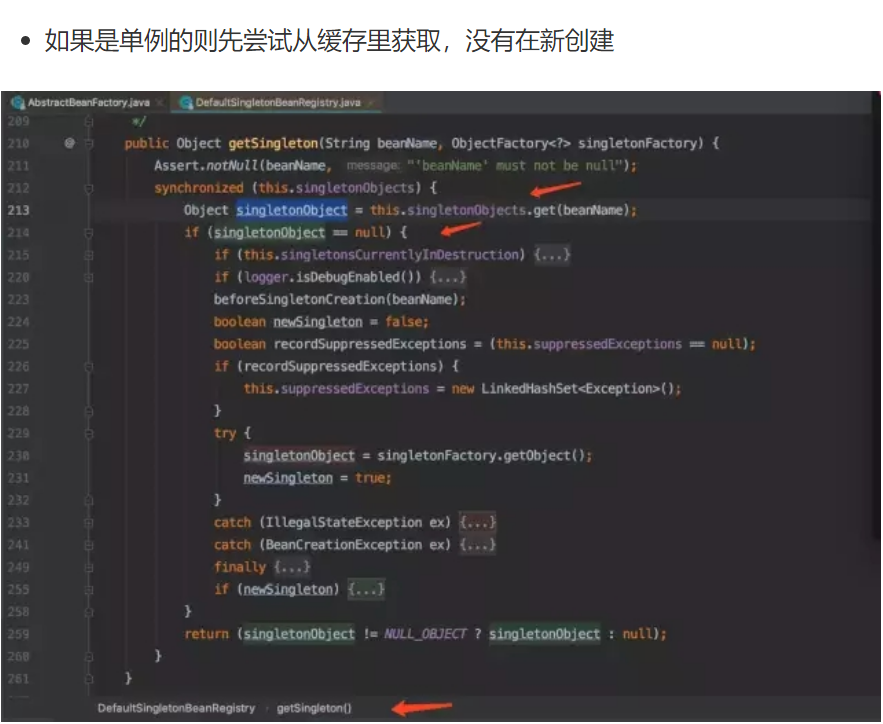
默认是单例。

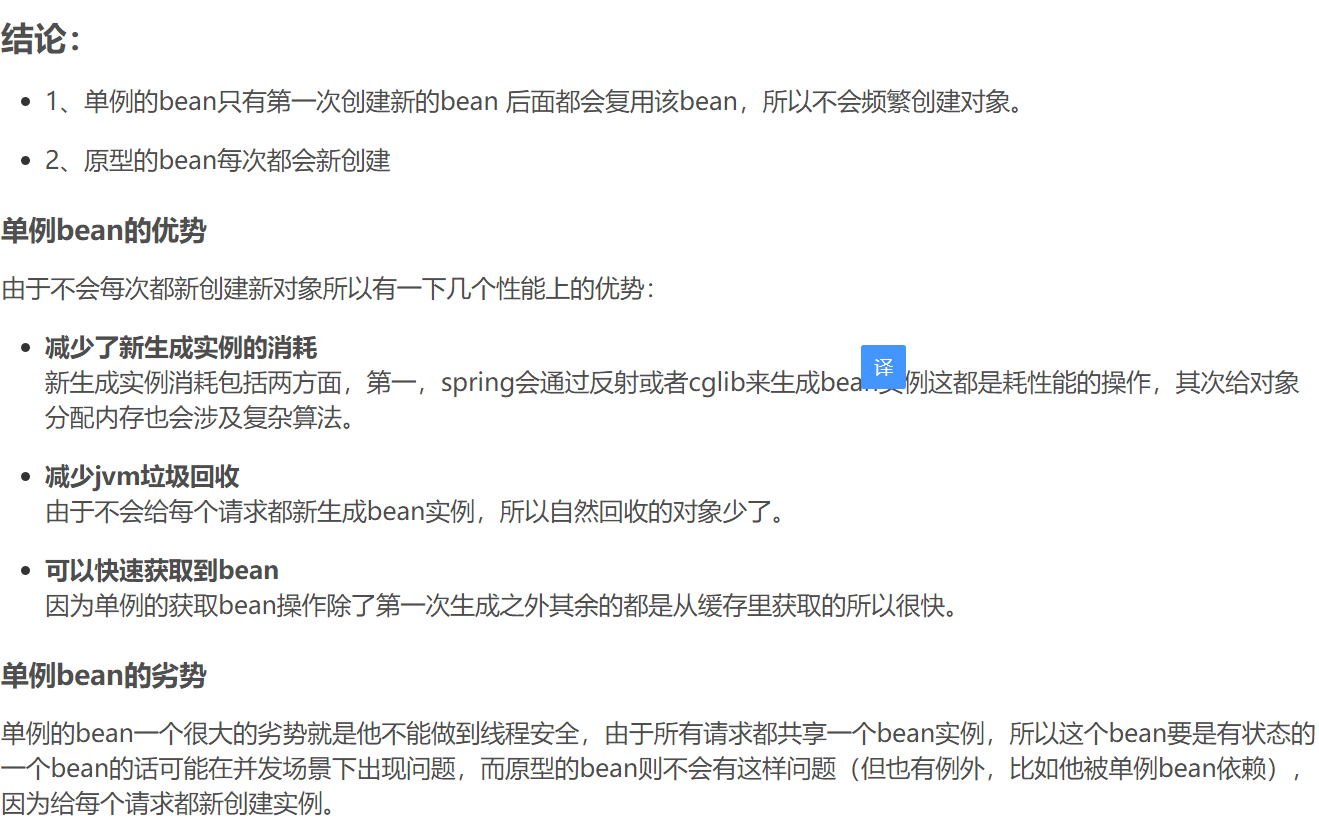
1. 为了提高性能 2.少创建实例\* 3.垃圾回收 4.缓存快速获取





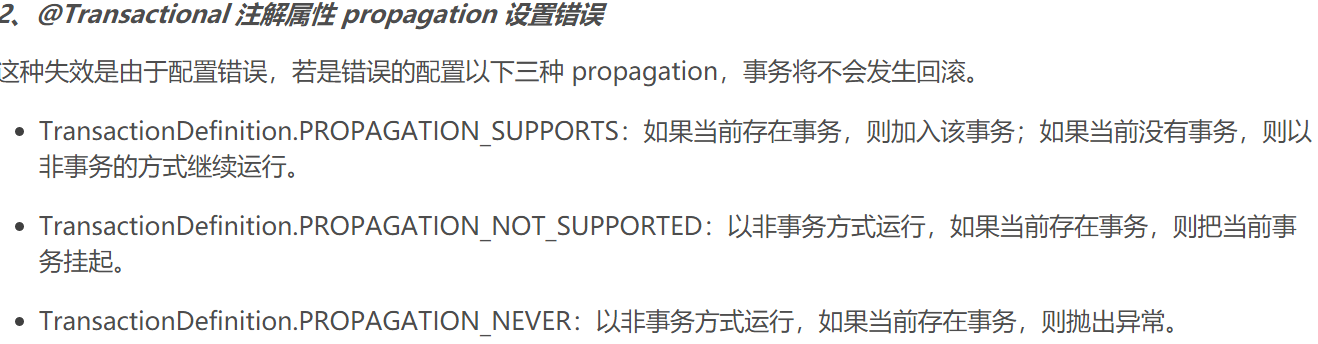


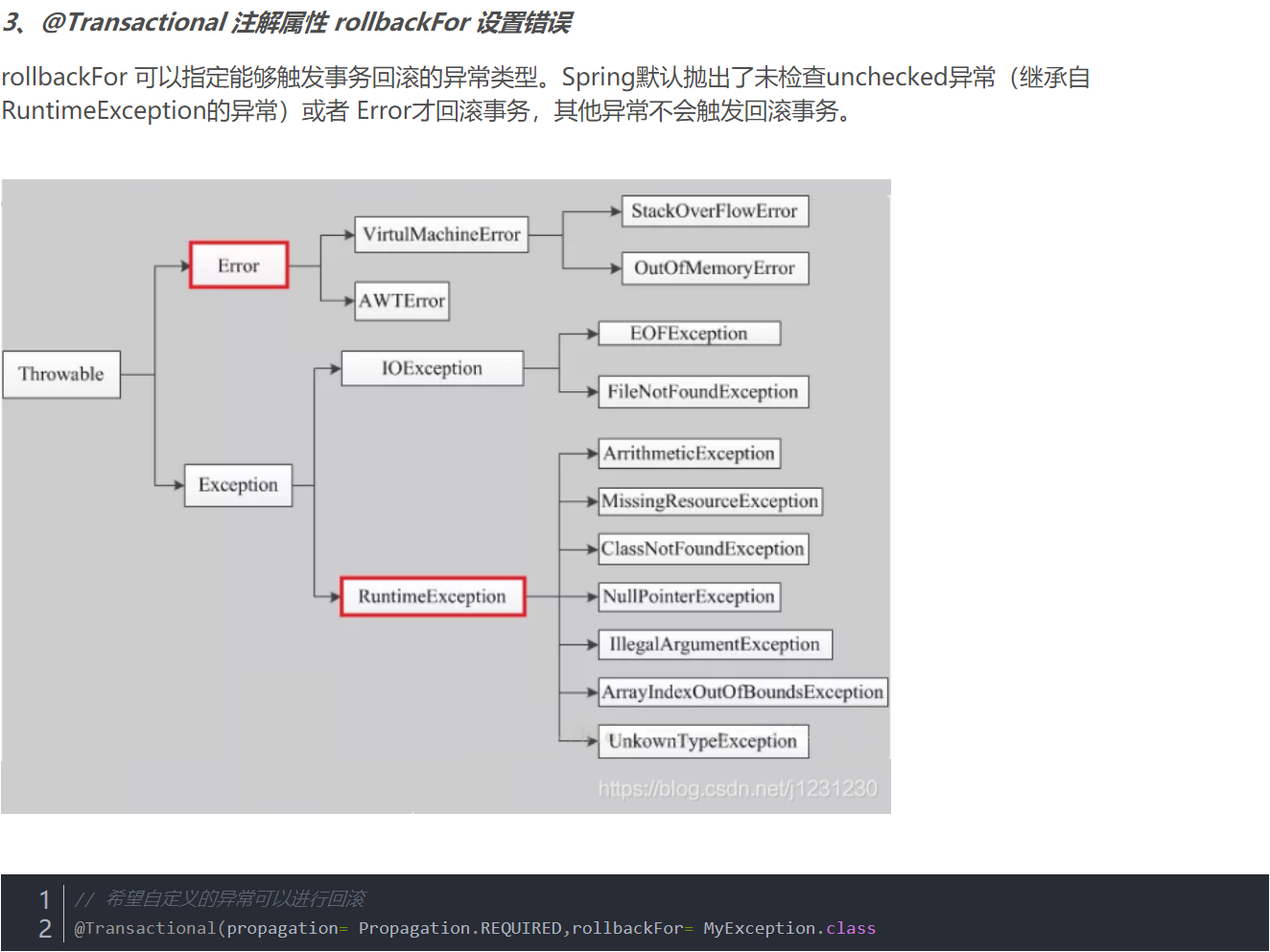




### 2.@Transactional注解的6种失效场景







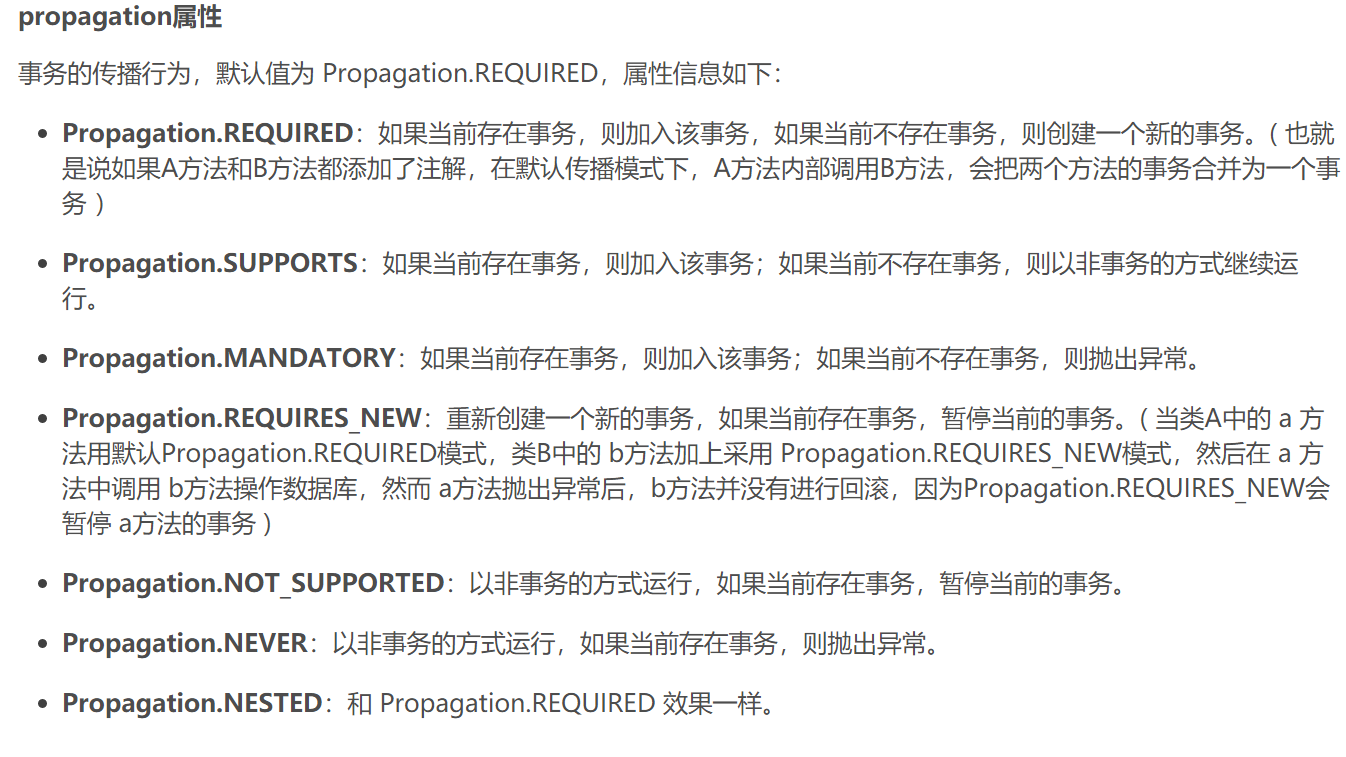
若在目标方法中抛出的异常是 rollbackFor 指定的异常的子类，事务同样会回滚







### 3.事务的传播属性



### 4什么是AOP?

AOP，一般称为面向切面，作为面向对象的一种补充，用于将那些与业务无关，但却对多个对象产生影响的公共行为和逻辑，抽取并封装为一个可重用的模块，这个模块被命名为“切面”（Aspect），减少系统中的重复代码，降低了模块间的耦合度，同时提高了系统的可维护性。

在面向对象编程中，关键的单元是对象，但AOP的关键单元是切面(关注点)，像日志或者事务这些分散的切面(横切关注点——可以影响到整个应用的关注点)应该被尽量地集中到一个地方以方便管理(事务管理、权限、日志、安全)。而AOP就是作用于这些横切关注点，使得代码在当下和将来都变得容易维护，

4.1有哪些通知类型？



4.2 在Spring AOP中关注点和横切关注点有什么不同？

关注点：我们的应用模块中实现以解决特定业务问题的方法。比如库存管理、航运管理、用户管理等。

横切关注点：贯穿整个应用程序的关注点。像事务管理、权限、日志、安全

4.3 核心

AOP实现的关键在于 代理模式，AOP代理主要分为静态代理和动态代理。静态代理的代表为AspectJ；动态代理则以Spring AOP为代表。

（1）AspectJ是静态代理的增强，所谓静态代理，就是AOP框架会在编译阶段生成AOP代理类，因此也称为编译时增强，他会在编译阶段将AspectJ(切面)织入到Java字节码中，运行的时候就是增强之后的AOP对象。

1. Spring AOP使用的动态代理，所谓的动态代理就是说AOP框架不会去修改字节码，而是每次运行时在内存中临时为方法生成一个AOP对象，这个AOP对象包含了目标对象的全部方法，并且在特定的切点做了增强处理，并回调原对象的方法。

Spring AOP中的动态代理主要有两种方式，JDK动态代理和CGLIB动态代理：

   ①JDK动态代理只提供接口的代理，不支持类的代理。核心InvocationHandler接口和Proxy类，InvocationHandler 通过invoke()方法反射来调用目标类中的代码，动态地将横切逻辑和业务编织在一起；接着，Proxy利用 InvocationHandler动态创建一个符合某一接口的的实例,  生成目标类的代理对象。

      ②如果代理类没有实现 InvocationHandler 接口，那么Spring AOP会选择使用CGLIB来动态代理目标类。CGLIB（Code Generation Library），是一个代码生成的类库，可以在运行时动态的生成指定类的一个子类对象，并覆盖其中特定方法并添加增强代码，从而实现AOP。CGLIB是通过继承的方式做的动态代理，因此如果某个类被标记为final，那么它是无法使用CGLIB做动态代理的。

（3）静态代理与动态代理区别在于生成AOP代理对象的时机不同，相对来说AspectJ的静态代理方式具有更好的性能，但是AspectJ需要特定的编译器进行处理，而Spring AOP则无需特定的编译器处理。

### DI

什么是DI机制？  
依赖注入（Dependecy Injection）和控制反转（Inversion of Control）是同一个概念，具体的讲：当某个角色  
需要另外一个角色协助的时候，在传统的程序设计过程中，通常由调用者来创建被调用者的实例。但在spring中  
创建被调用者的工作不再由调用者来完成，因此称为控制反转。创建被调用者的工作由spring来完成，然后注入调用者  
因此也称为依赖注入。  
spring以动态灵活的方式来管理对象 ， 注入的两种方式，设置注入和构造注入。  
设置注入的优点：直观，自然  
构造注入的优点：可以在构造器中决定依赖关系的顺序。

### Bean的作用域？

singleton：单例模式，Spring IoC容器中只会存在一个共享的Bean实例，无论有多少个Bean引用它，始终指向同一对象。Singleton作用域是Spring中的缺省作用域，也可以显示的将Bean定义为singleton模式，配置为：  
  
prototype:原型模式，每次通过Spring容器获取prototype定义的bean时，容器都将创建一个新的Bean实例，每个Bean实例都有自己的属性和状态，而singleton全局只有一个对象。根据经验，对有状态的bean使用prototype作用域，而对无状态的bean使用singleton作用域。

request：在一次Http请求中，容器会返回该Bean的同一实例。而对不同的Http请求则会产生新的Bean，而且该bean仅在当前Http Request内有效。  
,针对每一次Http请求，Spring容器根据该bean的定义创建一个全新的实例，且该实例仅在当前Http请求内有效，而其它请求无法看到当前请求中状态的变化，当当前Http请求结束，该bean实例也将会被销毁。

session：在一次Http Session中，容器会返回该Bean的同一实例。而对不同的Session请求则会创建新的实例，该bean实例仅在当前Session内有效。  
,同Http请求相同，每一次session请求创建新的实例，而不同的实例之间不共享属性，且实例仅在自己的session请求内有效，请求结束，则实例将被销毁。

global Session：在一个全局的Http Session中，容器会返回该Bean的同一个实例，仅在使用portlet context时有效。