1. spring事务失效的场景？

事务不生效：

1. 方法访问权限，spring要求被代理的方法必须是public的
2. 方法是final（static）修饰的，事务是aop实现的，jdk代理或者cgilib代理无法重写final方法
3. 方法内部调用，service的方法中调用另个事务的方法，没有经过aop代理
4. 未被spring管理
5. 多线程调用，不同的线程获取的数据连接不一样

事务不回滚：

1. 错误使用事务的传播机制，如使用了NEVER传播机制
2. 自己吞了异常，在代码中使用了try……catch，spring事务要能正常回滚，必须抛出它能处理的异常
3. 手动抛了别的异常，spring事务默认情况下只会回滚runtimeexception和error异常，不会回滚exception普通异常
4. 自定义了回滚异常，报错的异常与自定义的异常不同
5. spring事务传播机制
6. required：如果当前上下文存在事务，那么加入该事务，若没有，则新建个事务，这是默认的传播机制
7. supports：如果当前上下文存在事务，则支持事务加入事务，若没有，则使用非事务的方式执行
8. mandatory：如果当前上下文中存在事务，则抛除异常
9. requires\_new：每次都会新建一个事务，并且同时将上下文中的事务挂起，执行当前新建事务完成后，上下文事务在恢复执行
10. not\_supported：如果当前上下文中存在事务，则挂起当前事务，然后新的方法在没有事务的环境中执行
11. never：若当前上下文中存在事务，则抛出异常，否则在无事务环境上执行代码
12. nested：如果当前上下文中存在事务，则嵌套事务执行；若没有，则新建事务
13. spring框架中用到了哪些设计模式？

工厂设计模式：Spring使用工厂模式通过BeanFactory和ApplicationContext创建bean对象

代理设计模式：Spring AOP功能的实现

单例设计模式：Spring中的bean默认都是单例的

模板方法模式：Spring中的jdbcTemplate、hibernateTemplate、redisTemplate等

观察者模式：Spring事件驱动模型就是观察者模式的应用

适配器模式：Spring AOP的增强或通知使用到了设配器模式，Spring MVC中也是用到了适配器模式适配Controller

4、Spring MVC的工作原理？

（1）用户发送请求至前端控制器dispatcherservlet

（2）dispatchersevlet收到请求调用处理器映射器handerMapping

（3）handerMapping根据请求url找到具体的处理器，生成handerExecutionChain（包括处理器对象和处理器拦截器）一并返回给dispatcherServlet

（4）dispatcherSevlet根据处理器hander获取处理器适配器handerAdapter执行handerAdapter处理一系列操作，如：参数封装、格式转换、数据验证等

（5）执行处理器hander（Controller，也叫页面控制器）

（6）hander执行完成后返回modelAndView

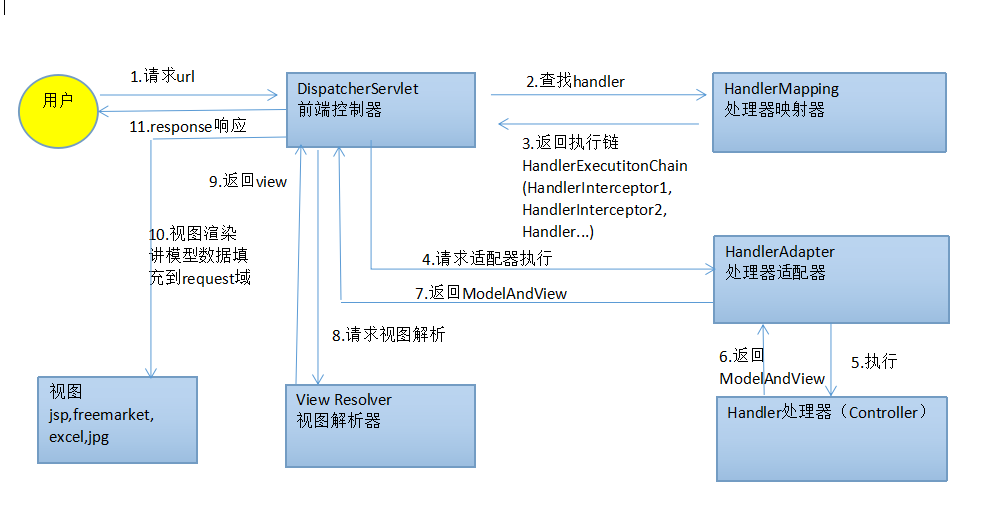
（7）handerAdapter将modelAndView返回到dispatcherServlet

（8）dispatcherServlet将modelAndView传给ViewReslover视图解析器

（9）viewReslover解析后返回具体view

（10）dispatcherServlet对view进行渲染（即将模型数据model填充到视图中）

（11）dispatcherServlet响应用户（把view返回给浏览器）



1. Spring中bena的生命周期

（1）Bean容器/BeanFactory 通过对象的构造器或工厂方法先实例化 Bean；

（2）再根据 Resource 中的信息再通过设定好的方法（典型的有setter，统称为BeanWrapper）对 Bean 设置属性值，得到 BeanDefintion 对象，然后 put 到 beanDefinitionMap 中，调用 getBean 的时候，从 beanDefinitionMap 里，拿出 Class 对象进行注入，同时，如果有依赖关系，将递归调用 getBean 方法，即依赖注入的过程。

（3）检查 xxxAware 相关接口，比如 BeanNameAware，BeanClassLoaderAware，ApplicationContextAware（ BeanFactoryAware）等等，如果有就调用相应的 setxxx 方法把所需要的xxx传入到 Bean 中。

**补充**：关于 Aware ，Aware 就是感知的意思， Aware 的目的是为了让Bean获得Spring容器的服务。 实现了这类接口的 bean 会存在“意识感”，从而让容器调用 setxxx 方法把所需要的 xxx 传到 Bean 中。

（4）此时检查是否存在有于 Bean 关联的任何 BeanPostProcessors， 执行 postProcessBeforeInitialization() 方法（前置处理器）。

（5）如果 Bean 实现了InitializingBean接口（正在初始化的 Bean），执行 afterPropertiesSet() 方法。

（6）检查是否配置了自定义的 init-method 方法，如果有就调用。

（7）此时检查是否存在有于 Bean 关联的任何 BeanPostProcessors， 执行 postProcessAfterInitialization() 方法（后置处理器）。返回 wrapperBean（包装后的 Bean）。

（8）这时就可以开始使用 Bean 了，当容器关闭时，会检查 Bean 是否实现了 DisposableBean 接口，如果有就调用 destory() 方法。

（9）如果 Bean 配置文件中的定义包含 destroy-method 属性，执行指定的方法。

1. BeanFactory和ApplicationContext有什么区别

两者都可以通过xml装载bean，前者是延迟加载，用到的时候再去实例化bean；后者是在启动的时候就把bean全都实例化好了等待使用。

1. ApplicationContext的通常实现是什么
2. FileSystemXmlAppiicationContext：此容器从一个XML文件中加载beans的定义，XML bean配置文件的全路径名必须提供给它的构造函数
3. ClassPathXmlApplicationContext：此容器也从一个XML文件中加载beans的定义，这里你需要正确设置classpath，因为这个容器将在classpath里找bean的配置
4. WebXmlApplicationContext：此容器加载一个XML文件，该文件中定义了一个web应用里的所有bean
5. Spring事务的底层原理

Spring事务就是对数据库事务的支持，Spring只提供统一事务管理接口；一般我们会在方法上加上@Transaction注解来开启事务，加上这个注解后，执行时会把事务的自动提交设置为false，然后再去执行对应的逻辑代码。

1. Spring事务的隔离级别

（1）ISOLATION\_DEFAULT：使用后端数据库默认的隔离级别，Mysql默认采用的 REPEATABLE\_READ隔离级别；Oracle默认采用的READ\_COMMITTED隔离级别。

（2）ISOLATION\_READ\_UNCOMMITTED：最低的隔离级别，允许读取尚未提交的数据变更， 可能会导致脏读、幻读或不可重复读。

（3）ISOLATION\_READ\_COMMITTED：允许读取并发事务已经提交的数据，可以阻止脏读， 但是幻读或不可重复读仍有可能发生

（4）ISOLATION\_REPEATABLE\_READ：对同一字段的多次读取结果都是一致的，除非数据是 被本身事务自己所修改，可以阻止脏读和不可重复读，但幻读仍有可能发生。

（5）ISOLATION\_SERIALIZABLE：最高的隔离级别，完全服从ACID的隔离级别。所有的事务依 次逐个执行，这样事务之间就完全不可能产生干扰，也就是说，该级别可以防止脏读、不可重 复读以及幻读。但是这将严重影响程序的性能。通常情况下也不会用到该级别。

1. Spring中bean的作用域有哪些

singleton：唯一bean实例，Spring中的bean默认都是单例的。

prototype：每次请求都会创建一个新的bean实例。

request：每一次HTTP请求都会产生一个新的bean，该bean仅在当前HTTP request内有 效。

session：每一次HTTP请求都会产生一个新的bean，该bean仅在当前HTTP session内有效。

global-session：全局session作用域，仅仅在基于Portlet的Web应用中才有意义，

10、Spring AOP 与 AspectJ AOP有什么区别

Spring AOP是属于运行时增强，而AspectJ是编译时增强。Spring AOP基于代理 （Proxying），而AspectJ基于字节码操作（Bytecode Manipulation）。

Spring AOP已经集成了AspectJ，AspectJ应该算得上是Java生态系统中最完整的AOP框架 了。AspectJ相比于Spring AOP功能更加强大，但是Spring AOP相对来说更简单。

如果我们的切面比较少，那么两者性能差异不大。但是，当切面太多的话，最好选择 AspectJ，它比SpringAOP快很多。

11、Spring是如何解决循环依赖的？

整个IOC容器解决循环依赖，用到的几个重要成员： singletonObjects：一级缓存，存放完全初始化好的Bean的集合，从这个集合中取出来的 Bean可以立马返回 earlySingletonObjects：二级缓存，存放创建好但没有初始化属性的Bean的集合，它用来解 决循环依赖 singletonFactories：三级缓存，存放单实例Bean工厂的集合 singletonsCurrentlyInCreation：存放正在被创建的Bean的集合

IOC容器解决循环依赖的思路：

初始化Bean之前，将这个BeanName放入三级缓存

创建Bean将准备创建的Bean放入 singletonsCurrentlyInCreation （正在创建的 Bean）

createNewInstance 方法执行完后执行 addSingletonFactory，将这个实例化但没有 属性赋值的Bean放入二级缓存，并从三级缓存中移除

属性赋值&自动注入时，引发关联创建

关联创建时，检查“正在被创建的Bean”中是否有即将注入的Bean。如果有，检查二级 缓存中是否有当前创建好但没有赋值初始化的Bean。如果没有，检查三级缓存中是否有 正在创建中的Bean。至此一般会有，将这个Bean放入二级缓存，并从三级缓存中移除

之后Bean被成功注入，最后执行 addSingleton，将这个完全创建好的Bean放入一级缓 存，从二级缓存和三级缓存移除，并记录已经创建了的单实例Bean

12、springboot2.x的新特性

配置变更，JDK版本升级，第三方类库升级，响应式Spring编程支持，HTTP/2支持，配置属性绑定

13、Springboot的核心注解是哪个？

@SpringbootAppilcation，包含以下三个注解：

@SpringbootConfiguration：组合了@Configuration，实现配置文件功能

@EnableAutoConfig：打开自动配置功能

@ComponentScan：Spring组件扫描

14、Spring Security和shiro对比

前者是一个重量级的安全管理框架，概念复杂，配置繁琐，初步使用起来比较困难；而后者是个轻量级的安全管理框架，概念简单，配置简单，初步上手的话比较容易

15、zookeeper和eureka对提供服务注册和发现功能的区别

Zookeeper保证的是cp（一致性、分区容错性）；eureka保证的是ap（高可用，分区容错性）

16、如何动态实现zuul网关路由转发

通过path配置拦截请求，再通过serviceId到配置中心获取转发的服务列表，zuul内部通过Ribbon实现本地负载均衡和转发

17、Eureka的自我保护机制

当一个服务未按时进行心跳续约时（15分钟内超过85%的客户端都没有正常的心跳），eureka就会把当前实例的注册信息保护起来，不予剔除。