## **Spring 事务中的隔离级别有哪几种?**

**TransactionDefinition 接口中定义了五个表示隔离级别的常量：**

**1.TransactionDefinition.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.TransactionDefinition.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.TransactionDefinition.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.TransactionDefinition.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.TransactionDefinition.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。

## **Spring 事务中哪几种事务传播行为?**

**支持当前事务的情况：**

**TransactionDefinition.PROPAGATION\_REQUIRED：** 如果当前存在事务，则加入该事务；如果当前没有事务，则创建一个新的事务。

**TransactionDefinition.PROPAGATION\_SUPPORTS：** 如果当前存在事务，则加入该事务；如果当前没有事务，则以非事务的方式继续运行。

**TransactionDefinition.PROPAGATION\_MANDATORY：** 如果当前存在事务，则加入该事务；如果当前没有事务，则抛出异常。（mandatory：强制性）

**不支持当前事务的情况：**

**TransactionDefinition.PROPAGATION\_REQUIRES\_NEW：** 创建一个新的事务，如果当前存在事务，则把当前事务挂起。

**TransactionDefinition.PROPAGATION\_NOT\_SUPPORTED：** 以非事务方式运行，如果当前存在事务，则把当前事务挂起。

**TransactionDefinition.PROPAGATION\_NEVER：** 以非事务方式运行，如果当前存在事务，则抛出异常。

**其他情况：**

**TransactionDefinition.PROPAGATION\_NESTED：** 如果当前存在事务，则创建一个事务作为当前事务的嵌套事务来运行；如果当前没有事务，则该取值等价于TransactionDefinition.PROPAGATION\_REQUIRED。

## **@Transactional(rollbackFor = Exception.class)注解了解吗？**

**我们知道：Exception分为运行时异常RuntimeException和非运行时异常。事务管理对于企业应用来说是至关重要的，即使出现异常情况，它也可以保证数据的一致性。**

**当@Transactional注解作用于类上时，该类的所有 public 方法将都具有该类型的事务属性，同时，我们也可以在方法级别使用该标注来覆盖类级别的定义。如果类或者方法加了这个注解，那么这个类里面的方法抛出异常，就会回滚，数据库里面的数据也会回滚。**

**在@Transactional注解中如果不配置rollbackFor属性,那么事物只会在遇到RuntimeException的时候才会回滚,加上rollbackFor=Exception.class,可以让事物在遇到非运行时异常时也回滚。**

## **列举一些重要的Spring模块？**

**Spring Core：** 基础,可以说 Spring 其他所有的功能都需要依赖于该类库。主要提供 IOC 依赖注入功能。

**Spring Aspects**  ：该模块为与AspectJ的集成提供支持。

**Spring AOP** ：提供了面向方面的编程实现。

**Spring JDBC** : Java数据库连接。

**Spring JMS** ：Java消息服务。

**Spring ORM** : 用于支持Hibernate等ORM工具。

**Spring Web** : 为创建Web应用程序提供支持。

**Spring Test** : 提供了对 JUnit 和 TestNG 测试的支持。

**Radis的内存淘汰机制：**

**Radis的LRU手写算法**

**解决哈希冲突的几个办法是什么：**

**MVCC多版本并发控制协议**

**Radis和数据库的缓存一致性问题**

**数据库的几大范式**

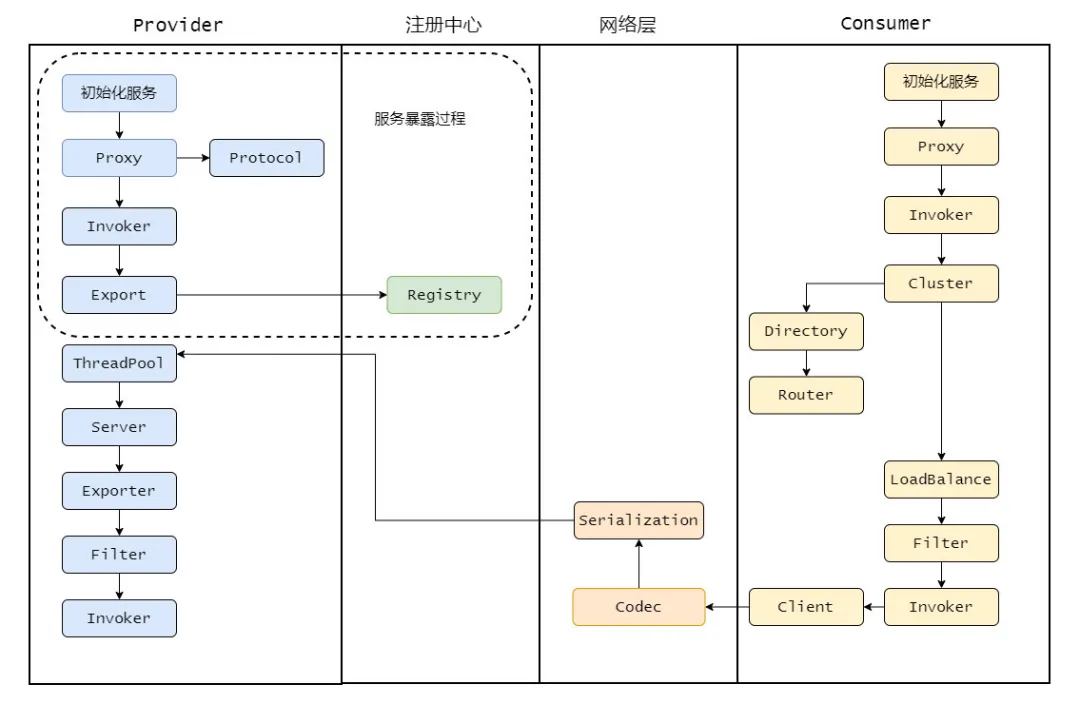
## **服务暴露过程**

首先 Provider 启动，通过 Proxy 组件根据具体的协议 Protocol 将需要暴露出去的接口封装成 Invoker，Invoker 是 Dubbo 一个很核心的组件，代表一个可执行体。

然后再通过 Exporter 包装一下，这是为了在注册中心暴露自己套的一层，然后将 Exporter 通过 Registry 注册到注册中心。这就是整体服务暴露过程。

## **消费过程**

接着我们来看消费者调用流程（把服务者暴露的过程也在图里展示出来了，这个图其实算一个挺完整的流程图了）。



首先消费者启动会向注册中心拉取服务提供者的元信息，然后调用流程也是从 Proxy 开始，毕竟都需要代理才能无感知。

Proxy 持有一个 Invoker 对象，调用 invoke 之后需要通过 Cluster 先从 Directory 获取所有可调用的远程服务的 Invoker 列表，如果配置了某些路由规则，比如某个接口只能调用某个节点的那就再过滤一遍 Invoker 列表。

剩下的 Invoker 再通过 LoadBalance 做负载均衡选取一个。然后再经过 Filter 做一些统计什么的，再通过 Client 做数据传输，比如用 Netty 来传输。

传输需要经过 Codec 接口做协议构造，再序列化。最终发往对应的服务提供者。

服务提供者接收到之后也会进行 Codec 协议处理，然后反序列化后将请求扔到线程池处理。某个线程会根据请求找到对应的 Exporter ，而找到 Exporter 其实就是找到了 Invoker，但是还会有一层层 Filter，经过一层层过滤链之后最终调用实现类然后原路返回结果。

完成整个调用过程！