# 四个特性

## 原子性:

要么做，要么不做，支持回滚操作，

实现方式：1.在数据快照上进行，并不修改实际的数据，如果有错并不会提交

2.直接操作实际数据，先预演一遍所有要执行的操作，如果失败则这些操作不会被执行，可以基于日志操作

## 隔离性:

并发事务之间相互影响的程度，例如一个事务是否可以读到b事务未提交的数据

**问题**：

脏读：读未提交的数据，如果事务失败，就是脏读了

不可重复读：同一个事务中，对于同一份数据读取到的结果不一致；原因，并发修改记录，读的时候并不会把记录锁掉，解决方法：1.对修改的记录加锁2.mvcc

幻读：同一个事务中，同一个查询读取到的结果不一致（统计）；原因：并发事务增加记录，解决办法：事务串行化，才能避免幻读

第一类丢失：a事务撤销覆盖以及提交的B事务

第二类丢失：A事务覆盖B事务已经提交的数据

**级别**：

读未提交：可以读到另外一个事务未提交的结果

读提交（不可重复读）：只有事务提交后，其更新结果才能被其他事务看见，解决脏读问题

重复读：解决不可重复读的问题，锁或mvcc

解决方法：[数据库的悲观锁和乐观锁](#_数据库乐观锁与悲观锁)来避免问题

[Spring中事务](spring事务.docx)

问题：两个事务中读到的数据是不一致的，为什么？

两个事务中系统版本号是不一样的，操作的是不一致的行记录

如何一致性？

串行化：

## 一致性:

系统从一个一致性状态到另外一个一致性状态

强一致性：读操作可以立即读到提交的更新操作

弱一致性：读不到刚提交的更新数据

会话一致性：

## 持久性:：

事务提交后，对系统的影响是永久的

Start transaction 开启一项新的事务

Commit rollback 提交与回滚

Set autocommit 0 以后的操作需要明确的命令进行提交或回滚

# 事务传播行为

Service 层方法调用service层的其他方法，本身service方法就是一个事务，事务中的事务如何处理？

传统的做法：将session（关于数据库的内容）传递到另外一个方法中

## PROPAGATION\_REQUIRED

--支持当前事务，如果当前没有事务，就新建一个事务。这是最常见的选择。

A方法中调用b

如果a在事务中，b就不会再起新的事务

如果a 不在事务中，会创建一个事务，回到第一种

## PROPAGATION\_NESTED

Nested的事务和他的父事务是相依的，他的提交是要等和他的父事务一块提交的。也就是说，如果父事务最后回滚，他也要回滚的。Nested事务的好处是他有一个savepoint。

A 方法中调用b方法，

A在事务里，会在b调用之前设置一个savepoint,如果b失败回滚回滚，a 会回到saveponit这个位置，选择其他分支执行

A不在事务里，执行创建事务，回到第一种

# 三、Spring事务实现演变

## 编程式事务

## 声明式事务

### Xml声明式事务

* + 1. 使用原始的TransactionProxyFactoryBean
    2. 基于tx/aop命名空间的配置

### 注解声明式事务

* + 1. 使用@Transactional注解

# spring 事务管理难点剖析

## 3.1 事务嵌套调用的分析

调用本方法的的其他方法，天然融合在一个事务中，调用其他service的方法，采用传播机制

## 3.2 在多线程环境下事务方法的调用问题

一个事务方法中的多个线程的事务是相互独立的

## 3.3 Spring AOP事务增强有哪些限制

Aop是基于动态代理的，动态代理是基于jdk和cglib

Jdk的方法 除了publis方法的方法都不能代理，public static 也不能增强

Cglib private,stati,finan的方法都不能增强

解决方案：private方法最终会被public的方法调用，被事务管理器管理

Public static 和public final的方法是要小心的，如果被非事务方法调用，就会在一个非事务的环境中运行

# 五、分布式事务