## Hibernate中的事务与并发

### 事务相关的概念

#### 1. 什么是事务

事务就是逻辑上的一组操作，组成事务的各个执行单元，操作要么全都成功，要么全都失败.

**转账**的例子：赵信给狐狸转钱，扣钱，加钱。两个操作组成了一个事情！

#### 2. 事务的特性 ACID

原子性（Atomicity） -- 事务不可分割. 整体

在事务中的多条sql语句是一个整体，要么就全部执行成功，有一条执行失败，都不执行。

一致性 （Consistency）-- 事务执行的前后数据的完整性保持一致. 数据的完整等性。赵信给狐狸转账2000块。能量守恒

隔离性 （Isolation）-- 一个事务执行的过程中,不应该受到其他的事务的干扰. 关于事务的隔离性数据库提供了多种隔离级别

持久性 （Durability）-- 事务一旦提交,数据就永久保持到数据库中.

#### 3. 如果不考虑隔离性:引发一些读的问题

脏读 -- 一个事务读到了另一个事务未提交的数据.

**张三和李四做交易，张三给李四转账，但是没有提交事务，李四这个时候去查询数据，这个时候读到了张三还有提交的数据，张三回滚事务，导致李四在查询钱少了。**

不可重复读 -- 一个事务读到了另一个事务已经提交的update数据,导致多次查询结果不一致.

张三在取钱的时候，第一查询余额是1万，这个事务李四取走了1万块，导致张三在取钱的时候，取不出来。张三在一个事务中，两次查询的结果不一样，因为中间被另外一个事务做了update操作。

幻读/虚读 -- 一个事务读到了另一个事务已经提交的insert数据,导致多次查询结构不一致.

#### 4. 通过设置数据库的隔离级别来解决上述读的问题

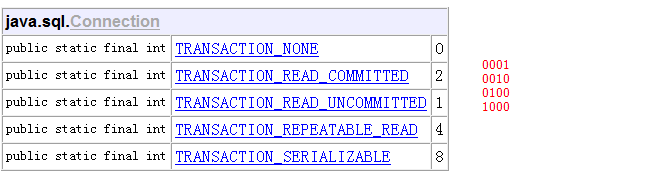
数据库为我们提供的四种隔离级别：

　　 Serializable (串行化)：可避免脏读、不可重复读、幻读的发生。 8

　　 Repeatable read (可重复读)：可避免脏读、不可重复读的发生。 4

　　 Read committed (读已提交)：可避免脏读的发生。 2

　　 Read uncommitted (读未提交)：最低级别，任何情况都无法保证。 1



#### 5. 如果想在Hibernate的框架中来设置隔离级别，需要在hibernate.cfg.xml的配置文件中通过标签来配置

通过：hibernate.connection.isolation = 4 来配置

取值

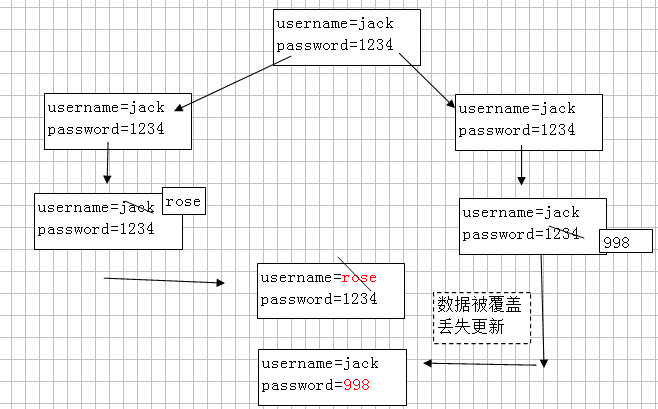
1—Read uncommitted isolation

2—Read committed isolation

4—Repeatable read isolation

8—Serializable isolation

### lost update 丢失更新



* 悲观锁：

顾名思义，就是很悲观，每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁，这样别人想拿这个数据就会block直到它拿到锁。传统的关系型数据库里边就用到了很多这种锁机制，比如 读锁，写锁等，都是在做操作之前先上锁。

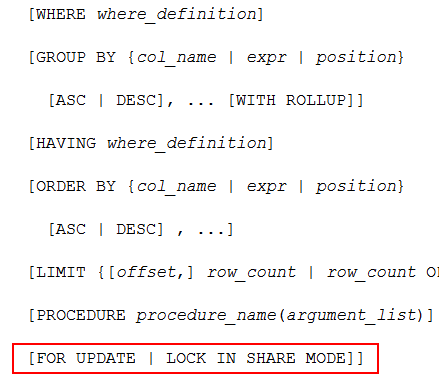
采用数据库锁机制。

读锁：共享锁。该锁可以共享

select .... from ... lock in share mode;

写锁：排他锁。（独占） 只能有一个线程拥有该锁。

select ... from .... for update

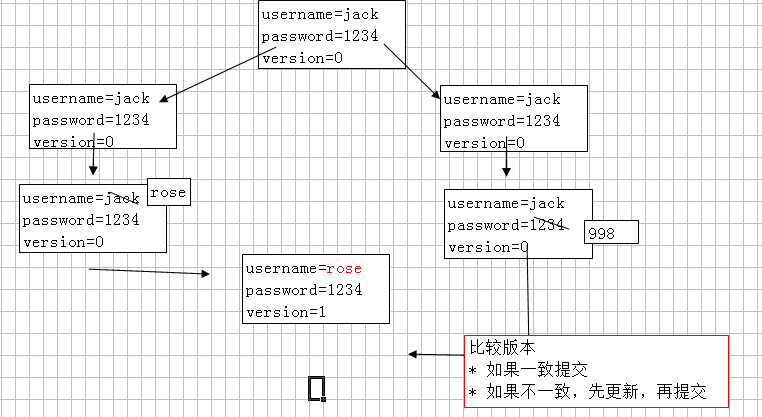


* 乐观锁：

顾名思义，就是很乐观，每次去拿数据的时候都认为别人不会修改，所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有去更新这个数据，可以使用版本号等机制。

在表中提供一个字段（版本字段），用于标识记录。如果版本不一致，不允许操作。

在数据库变里面整个version字段，0，



### 绑定本地的Session

1. 之前在讲JavaWEB的事务的时候，需要在业务层使用Connection来开启事务，

一种是通过参数的方式传递下去

另一种是把Connection绑定到ThreadLocal对象中

2. 现在的Hibernate框架中，使用session对象开启事务，所以需要来传递session对象，框架提供了ThreadLocal的方式

需要在hibernate.cfg.xml的配置文件中提供配置

<property name="hibernate.current\_session\_context\_class">thread</property>

重新HibernateUtil的工具类，使用SessionFactory的getCurrentSession()方法，获取当前的Session对象。并且该Session对象不用手动关闭，线程结束了，会自动关闭。

public static Session getCurrentSession(){

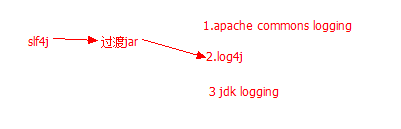
return factory.getCurrentSession();

}

**注意：想使用getCurrentSession()方法，必须要先配置才能使用。**

# 整合log4j(了解)

* slf4j 核心jar ： slf4j-api-1.6.1.jar 。slf4j是日志框架，将其他优秀的日志第三方进行整合。



* 整合导入jar包

log4j 核心包：log4j-1.2.17.jar

过渡jar（整合jar）：slf4j-log4j12-1.7.5.jar

* 导入配置文件

log4j.properties ，此配置文件通知log4j 如何输出日志

* 配置文件内容：

1.记录器

2.输出源

3.布局

* 记录器

例如： log4j.rootLogger=info, stdout,file

格式：log4j.rootLogger=日志级别, 输出源1，输出源2，。。。。

log4j 日志级别 ： fatal 致命错误 error 错误 warn 警告 info 信息 debug 调试信息 trace 堆栈信息 （由高到底顺序）

* 输出源：

例如：log4j.appender.file=org.apache.log4j.FileAppender

格式：log4j.appender.输出源的名称=输出源的实现类

名称：自定义

实现类：log4j提供

输出源属性例如：log4j.appender.file.File=d\\:mylog.log

输出源属性格式：log4j.appender.名称.属性=值

每一个输出源对应一个实现类，实现类都属性（setter），底层执行setter方法进行赋值

* 常见的输出源实现类

org.apache.log4j.FileAppender 输出文件中

file ,表示文件输出位置

org.apache.log4j.ConsoleAppender 输出到控制台

Target ，表示使用哪种输出方式，在控制台打印内容，取值：System.out / System.err

* 布局 -- 确定输出格式

例如：log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

格式：log4j.appender.数据源.layout=org.apache.log4j.PatternLayout

布局属性：log4j.appender. 数据源.layout.ConversionPattern=值

12:56:30,123 info

* 扩展：对指定的目录设置日志级别

例如：log4j.logger.org.hibernate.transaction=debug

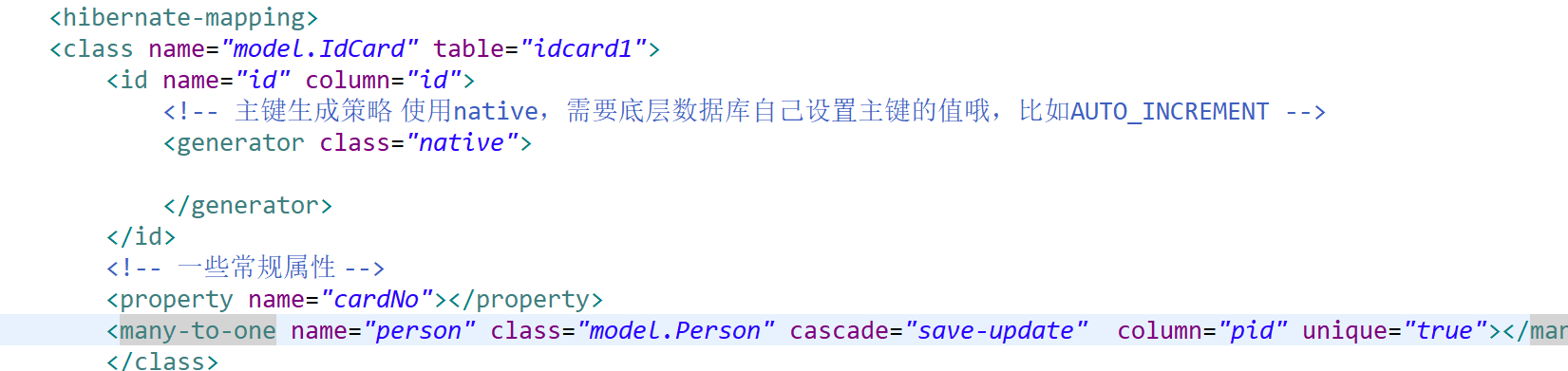
格式：log4j.logger.包结构=级别

# 一对一（了解）

* 情况1：主表的主键，与从表的外键（唯一），形成主外键关系



外键一方



* 情况2：主表的主键，与从表的主键，形成主外键关系 （从表的主键又是外键）

