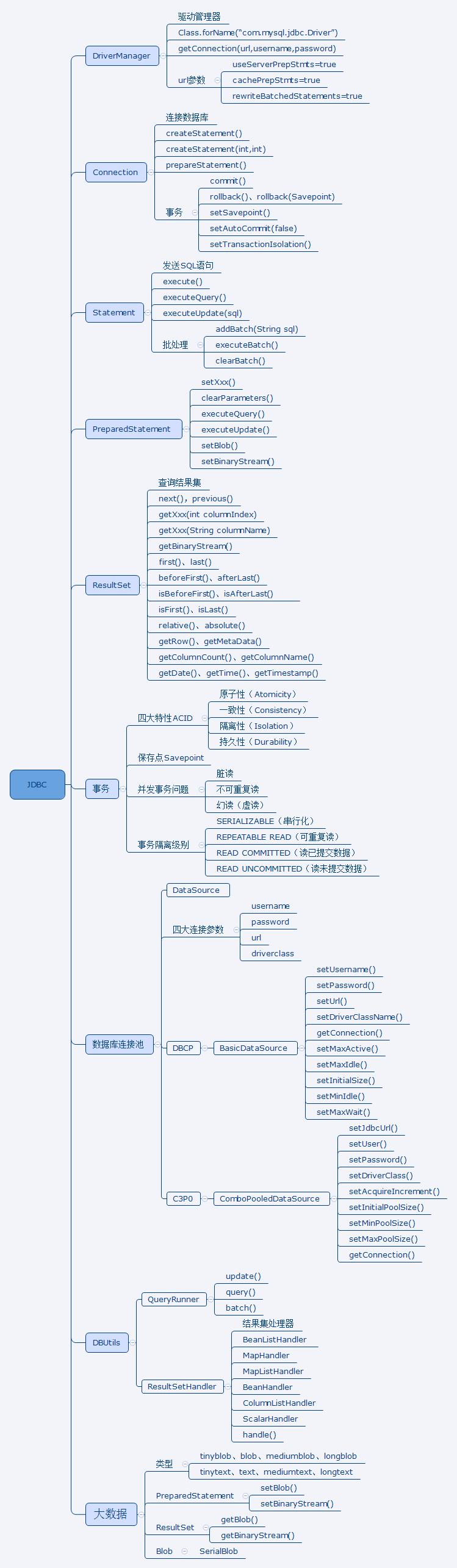
**JavaWeb：JDBC之事务**

**https://blog.csdn.net/axi295309066/article/details/52981430**



**1. 事务**

* 事务的四大特性：ACID
* mysql中操作事务
* jdbc中操作事务

**1.1 事务概述**

为了方便演示事务，我们需要创建一个account表：

CREATE TABLE account(

id INT PRIMARY KEY AUTO\_INCREMENT,

NAME VARCHAR(30),

balance NUMERIC(10.2)

);

INSERT INTO account(NAME,balance) VALUES('zs', 100000);

INSERT INTO account(NAME,balance) VALUES('ls', 100000);

INSERT INTO account(NAME,balance) VALUES('ww', 100000);

SELECT \* FROM account;

**1.2 什么是事务**

银行转账！张三转10000块到李四的账户，这其实需要两条SQL语句：

* 给张三的账户减去10000元
* 给李四的账户加上10000元

如果在第一条SQL语句执行成功后，在执行第二条SQL语句之前，程序被中断了（可能是抛出了某个异常，也可能是其他什么原因），那么李四的账户没有加上10000元，而张三却减去了10000元。这肯定是不行的！

你现在可能已经知道什么是事务了吧！事务中的多个操作，要么完全成功，要么完全失败！不可能存在成功一半的情况！也就是说给张三的账户减去10000元如果成功了，那么给李四的账户加上10000元的操作也必须是成功的；否则给张三减去10000元，以及给李四加上10000元都是失败的！

**1.3 事务的四大特性（ACID）**

事务的四大特性是：

* **原子性（Atomicity）**：事务中所有操作是不可再分割的原子单位。事务中所有操作要么全部执行成功，要么全部执行失败
* **一致性（Consistency）**：事务执行后，数据库状态与其它业务规则保持一致。如转账业务，无论事务执行成功与否，参与转账的两个账号余额之和应该是不变的
* **隔离性（Isolation）**：隔离性是指在并发操作中，不同事务之间应该隔离开来，使每个并发中的事务不会相互干扰
* **持久性（Durability）**：一旦事务提交成功，事务中所有的数据操作都必须被持久化到数据库中，即使提交事务后，数据库马上崩溃，在数据库重启时，也必须能保证通过某种机制恢复数据

**1.4 MySQL中的事务**

在默认情况下，MySQL每执行一条SQL语句，都是一个单独的事务。如果需要在一个事务中包含多条SQL语句，那么需要开启事务和结束事务。

* 开启事务：start transaction
* 结束事务：commit或rollback

在执行SQL语句之前，先执行strat transaction，这就开启了一个事务（事务的起点），然后可以去执行多条SQL语句，最后要结束事务，commit表示提交，即事务中的多条SQL语句所做出的影响会持久化到数据库中。或者rollback，表示回滚，即回滚到事务的起点，之前做的所有操作都被撤消了！

下面演示zs给li转账10000元的示例：

START TRANSACTION;

UPDATE account SET balance=balance-10000 WHERE id=1;

UPDATE account SET balance=balance+10000 WHERE id=2;

ROLLBACK;

START TRANSACTION;

UPDATE account SET balance=balance-10000 WHERE id=1;

UPDATE account SET balance=balance+10000 WHERE id=2;

COMMIT;

START TRANSACTION;

UPDATE account SET balance=balance-10000 WHERE id=1;

UPDATE account SET balance=balance+10000 WHERE id=2;

quit;

**2. JDBC事务**

在jdbc中处理事务，都是通过Connection完成的！同一事务中所有的操作，都在使用同一个Connection对象！

**2.1、JDBC中的事务**

Connection的三个方法与事务相关：

* setAutoCommit(boolean)：设置是否为自动提交事务，如果true（默认值就是true）表示自动提交，也就是每条执行的SQL语句都是一个单独的事务，如果设置false，那么就相当于开启了事务了
* commit()：提交结束事务
* rollback()：回滚结束事务

jdbc处理事务的代码格式：

try {

con.setAutoCommit(false);//开启事务…

….

…

con.commit();//try的最后提交事务

} catch() {

con.rollback();//回滚事务

}

public void transfer(boolean b) {

Connection con = null;

PreparedStatement pstmt = null;

try {

con = JdbcUtils.getConnection();

//手动提交

con.setAutoCommit(false);

String sql = "update account set balance=balance+? where id=?";

pstmt = con.prepareStatement(sql);

//操作

pstmt.setDouble(1, -10000);

pstmt.setInt(2, 1);

pstmt.executeUpdate();

// 在两个操作中抛出异常

if(b) {

throw new Exception();

}

pstmt.setDouble(1, 10000);

pstmt.setInt(2, 2);

pstmt.executeUpdate();

//提交事务

con.commit();

} catch(Exception e) {

//回滚事务

if(con != null) {

try {

con.rollback();

} catch(SQLException ex) {}

}

throw new RuntimeException(e);

} finally {

//关闭

JdbcUtils.close(con, pstmt);

}

}

**3. 保存点**

保存点是JDBC3.0的东西！当要求数据库服务器支持保存点方式的回滚。   
校验数据库服务器是否支持保存点！

boolean b = con.getMetaData().supportsSavepoints();

保存点的作用是允许事务回滚到指定的保存点位置。在事务中设置好保存点，然后回滚时可以选择回滚到指定的保存点，而不是回滚整个事务！注意，回滚到指定保存点并没有结束事务！！！只有回滚了整个事务才算是结束事务了！

Connection类的设置保存点，以及回滚到指定保存点方法：

* 设置保存点：Savepoint setSavepoint()
* 回滚到指定保存点：void rollback(Savepoint)

/\*

\* 李四对张三说，如果你给我转1W，我就给你转100W。

\* ==========================================

\*

\* 张三给李四转1W（张三减去1W，李四加上1W）

\* 设置保存点！

\* 李四给张三转100W（李四减去100W，张三加上100W）

\* 查看李四余额为负数，那么回滚到保存点。

\* 提交事务

\*/

@Test

public void fun() {

Connection con = null;

PreparedStatement pstmt = null;

try {

con = JdbcUtils.getConnection();

//手动提交

con.setAutoCommit(false);

String sql = "update account set balance=balance+? where name=?";

pstmt = con.prepareStatement(sql);

//操作1（张三减去1W）

pstmt.setDouble(1, -10000);

pstmt.setString(2, "zs");

pstmt.executeUpdate();

//操作2（李四加上1W）

pstmt.setDouble(1, 10000);

pstmt.setString(2, "ls");

pstmt.executeUpdate();

// 设置保存点

Savepoint sp = con.setSavepoint();

//操作3（李四减去100W）

pstmt.setDouble(1, -1000000);

pstmt.setString(2, "ls");

pstmt.executeUpdate();

//操作4（张三加上100W）

pstmt.setDouble(1, 1000000);

pstmt.setString(2, "zs");

pstmt.executeUpdate();

//操作5（查看李四余额）

sql = "select balance from account where name=?";

pstmt = con.prepareStatement(sql);

pstmt.setString(1, "ls");

ResultSet rs = pstmt.executeQuery();

rs.next();

double balance = rs.getDouble(1);

　　　　　　//如果李四余额为负数，那么回滚到指定保存点

if(balance < 0) {

con.rollback(sp);

System.out.println("张三，你上当了！");

}

//提交事务

con.commit();

} catch(Exception e) {

//回滚事务

if(con != null) {

try {

con.rollback();

} catch(SQLException ex) {}

}

throw new RuntimeException(e);

} finally {

//关闭

JdbcUtils.close(con, pstmt);

}

}

**4. 事务隔离级别**

**4.1 事务的并发读问题**

* 脏读：读取到另一个事务未提交数据
* 不可重复读：两次读取不一致
* 幻读（虚读）：读到另一事务已提交数据

**4.2 并发事务问题**

因为并发事务导致的问题大致有5类，其中两类是更新问题，三类是读问题

* 脏读（dirty read）：读到另一个事务的未提交更新数据，即读取到了脏数据
* 不可重复读（unrepeatable read）：对同一记录的两次读取不一致，因为另一事务对该记录做了修改
* 幻读（虚读）（phantom read）：对同一张表的两次查询不一致，因为另一事务插入了一条记录

**4.2.1 脏读**

事务1：张三给李四转账100元   
事务2：李四查看自己的账户

* t1：事务1：开始事务
* t2：事务1：张三给李四转账100元
* t3：事务2：开始事务
* t4：事务2：李四查看自己的账户，看到账户多出100元（脏读）
* t5：事务2：提交事务
* t6：事务1：回滚事务，回到转账之前的状态

**4.2.2不可重复读**

事务1：酒店查看两次1048号房间状态   
事务2：预订1048号房间

* t1：事务1：开始事务
* t2：事务1：查看1048号房间状态为空闲
* t3：事务2：开始事务
* t4：事务2：预定1048号房间
* t5：事务2：提交事务
* t6：事务1：再次查看1048号房间状态为使用
* t7：事务1：提交事务   
  对同一记录的两次查询结果不一致！

**4.2.3幻读**

事务1：对酒店房间预订记录两次统计   
事务2：添加一条预订房间记录

* t1：事务1：开始事务
* t2：事务1：统计预订记录100条
* t3：事务2：开始事务
* t4：事务2：添加一条预订房间记录
* t5：事务2：提交事务
* t6：事务1：再次统计预订记录为101记录
* t7：事务1：提交

对同一表的两次查询不一致！

不可重复读和幻读的区别：

* 不可重复读是读取到了另一事务的更新；
* 幻读是读取到了另一事务的插入（MySQL中无法测试到幻读）；

**4.3 四大隔离级别**

4个等级的事务隔离级别，在相同数据环境下，使用相同的输入，执行相同的工作，根据不同的隔离级别，可以导致不同的结果。不同事务隔离级别能够解决的数据并发问题的能力是不同的

**4.3.1 SERIALIZABLE（串行化）**

* 不会出现任何并发问题，因为它是对同一数据的访问是串行的，非并发访问的
* 性能最差

**4.3.2 REPEATABLE READ（可重复读）（MySQL）**

* 防止脏读和不可重复读，不能处理幻读问题
* 性能比SERIALIZABLE好

**4.3.3 READ COMMITTED（读已提交数据）（Oracle）**

* 防止脏读，没有处理不可重复读，也没有处理幻读；
* 性能比REPEATABLE READ好

**4.3.4 READ UNCOMMITTED（读未提交数据）**

* 可能出现任何事务并发问题
* 性能最好

MySQL的默认隔离级别为REPEATABLE READ，这是一个很不错的选择吧！

**4.3.5 MySQL隔离级别**

MySQL的默认隔离级别为Repeatable read，可以通过下面语句查看：

select @@tx\_isolation

也可以通过下面语句来设置当前连接的隔离级别：

set transaction isolationlevel [4先1]

**4.3.6 JDBC设置隔离级别**

con. setTransactionIsolation(int level);参数可选值如下：

* Connection.TRANSACTION\_READ\_UNCOMMITTED
* Connection.TRANSACTION\_READ\_COMMITTED
* Connection.TRANSACTION\_REPEATABLE\_READ
* Connection.TRANSACTION\_SERIALIZABLE

**5. 事务总结**

* 事务的特性：ACID
* 事务开始边界与结束边界：开始边界（con.setAutoCommit(false)），结束边界（con.commit()或con.rollback()）
* 事务的隔离级别： READ\_UNCOMMITTED、READ\_COMMITTED、REPEATABLE\_READ、SERIALIZABLE。多个事务并发执行时才需要考虑并发事务

版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/axi295309066/article/details/52981430