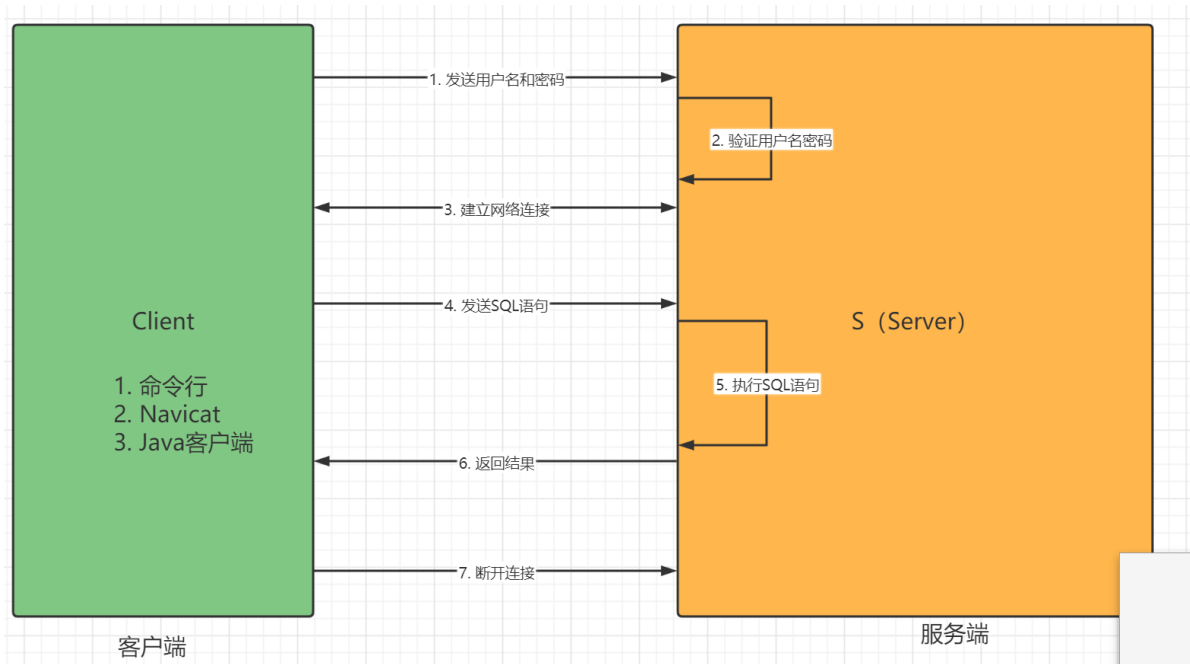


JDBC

1. 数据库的访问过程

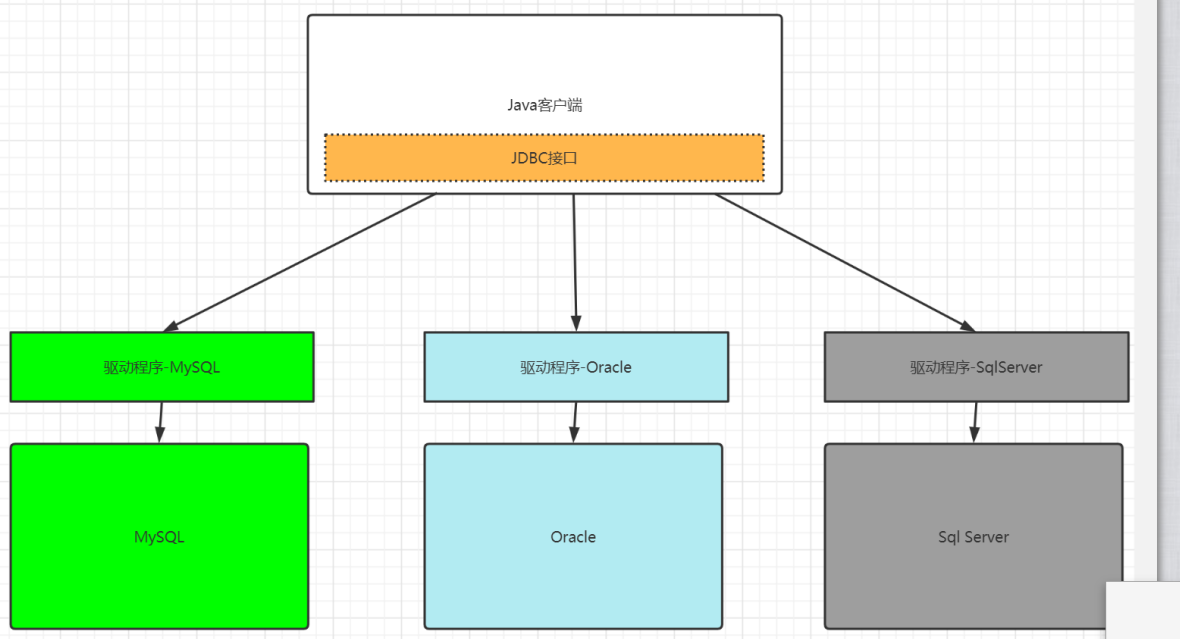


2. 简介

JDBC: Java Database Connection, Java数据库连接。其实就是Java提供的数据库的客户端。

JDBC在Java 中是如何通过什么来体现的呢？

JDBC在Java中其实是一套接口。是Java 为了更加方便的去操作各个数据库制定的一套统一的接口。



本质上来说，JDBC其实就是Java提供的一套操作数据库的接口；这些接口的实现由各个数据库厂商来实现。接口的实现类又被称为驱动程序包。

在Java中，各个数据库的驱动程序就是JDBC标准接口的实现类。这些驱动程序的表现形式是一个一个的jar包。

- jar包是什么？其实就是一种.jar文件的压缩包。 .jar 是一种压缩格式，.jar文件是可以直接被Java虚拟机识别并且运行的文件。jar文件中其实都是一些字节码文件，对于MySQL的驱动程序包来说，其实里面都是JDBC接口的实现类的字节码文件。

总结：

1. JDBC是一套Java制定的操作各个数据库的统一的接口
 - java.sql.*
 - javax.sql.*
2. 驱动程序是JDBC这套接口的实现，具体表现形式是jar包，jar包里面放的是JDBC接口实现类的字节码文件

3. JDBC操作流程

新建项目

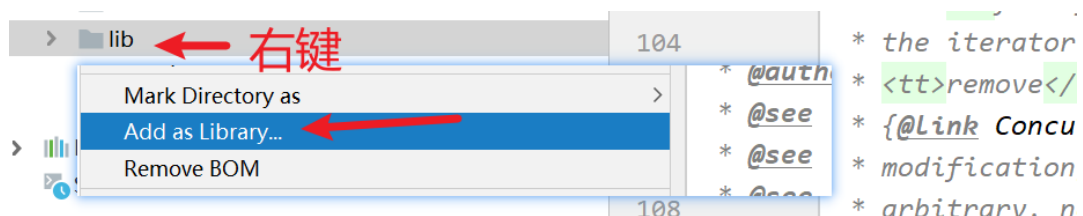
导包

- 下载驱动程序包

[MySQL驱动程序官方下载地址](#)

[maven仓库](#)

- 导包
 - 把jar包复制到项目中
 - 添加为library



写代码

```
public class FirstJDBC {

    // url = 协议 + 域名 + 路径 + 参数 （域名 = ip + 端口号）
    // characterEncoding=utf8 声明传递给MySQL的数据使用的字符集是utf8编码的

    static String url = "jdbc:mysql://localhost:3306/44th?
characterEncoding=utf8&useSSL=false";
    static String username = "root";
    static String password = "123456";

    public static void main(String[] args) throws SQLException {

        // 1. 注册驱动
        DriverManager.registerDriver(new Driver());
```

```

// 2. 发送用户名和密码，建立连接
// 这个地方获取到的连接实际上是接口来接收的，返回是接口的实现类（Jdbc4Connection）
Connection connection = DriverManager.getConnection(url, username,
password);

// 3. 获取Statement(这个Statement对象是用来发送语句并且获取返回的结果集的)
(StatementImpl)
Statement statement = connection.createStatement();

// 3. 发送SQL语句

// 执行 增删改类型的SQL语句
int affectedRows = statement.executeUpdate("insert into city values
(4001,'扬州',32),(4002,'无锡',32)");

// 执行查询SQL语句
// statement.executeQuery();

// 4. 获取返回的结果(展示出来)
System.out.println("affectedRows:" + affectedRows);

// 5. 关闭资源
statement.close();
connection.close();
}
}

```

```

public static void main(String[] args) throws SQLException {

// 1. 注册驱动
DriverManager.registerDriver(new Driver());

// 2. 建立连接
Connection connection = DriverManager.getConnection(FirstJDBC.url,
FirstJDBC.username, FirstJDBC.password);

// 3. 获取statement对象
Statement statement = connection.createStatement();

// 4. 发送SQL语句
ResultSet resultSet = statement.executeQuery("select * from city");

// 5. 解析结果集
while (resultSet.next()) {
// 获取对应的列的数据
int id = resultSet.getInt("id");
String name = resultSet.getString("name");
int pId = resultSet.getInt("p_id");

System.out.println("id:" + id + ",name:" + name + ",pId:" + pId);
}
}

```

```

    }

    // 6. 关闭资源
    resultSet.close();
    statement.close();
    connection.close();
}

```

4. API

DriverManager

```

// java.sql.DriverManager(JDBC提供的)

// 官方介绍: The basic service for managing a set of JDBC drivers
// 其实就是管理各个驱动程序的一个基础的服务

// 注册驱动的方法
DriverManager.registerDriver(new Driver());

// 获取连接
Connection connection = DriverManager.getConnection(String url,String
username,String password);

// 获取到的Connection对象是 com.mysql.jdbc.JDBC4Connection对象的实例

```

url详细格式:

jdbc:mysql:[]//localhost:3306/test?参数名1=参数值1&参数名2=参数值2

MySQL的url的格式: jdbc:mysql://localhost:3306/dbName?
param1=value1¶m2=value2...

Connection

```

// 指连接对象,是当前这个Java程序和MySQL数据库之间的网络连接对象

// java.sql.Connection (这是一个接口)
// A connection (session) with a specific database. SQL statements are executed
and results are returned within the context of a connection.

// 1. 创建Statement对象
Statement statement = connection.createStatement();

// 2. 创建PreparedStatement对象
PreparedStatement = connection.prepareStatement(String sql);

// 3. 关闭连接
connection.close();

```

```
// 4. 事务相关
setAutocommit(false);
commit();
rollback();
```

Statement

```
//The object used for executing a static SQL statement and returning the results
it produces.
// 这个对象是用来执行SQL语句并且返回这个SQL语句产生的结果

// 可以用来执行增删改的语句，也可以用来执行（DDL）建表或者是建库语句
int executeUpdate(String sql) throws SQLException;

// 执行查询语句
ResultSet executeQuery(String sql) throws SQLException;


// 执行任意类型SQL语句
boolean execute(String sql) throws SQLException;
```

ResultSet

```
// 结果集对象
// A table of data representing a database result set, which is usually generated
by executing a statement that queries the database.

// A <code>ResultSet</code> object maintains a cursor pointing to its current
row of data. // Initially the cursor is positioned before the first row.
```

id	name	p_id
1002	上海	32
1003	上海	34
1004	上海	34
1005	苏州	32
2001	扬州	32
2002	无锡	32
4001	扬州	32
4002	无锡	32



```
// 向下移动游标
// 当正常向下移动的时候，返回true；当当前行是最后一行，不能继续向下移动的时候，返回false
boolean next() throws SQLException;

// 向上移动
boolean previous() throws SQLException;

// 移动到第一行之前
void beforeFirst() throws SQLException;
```

```
// 移动到最后一行之后
void afterLast() throws SQLException;

// 移动到指定的行
boolean absolute( int row ) throws SQLException;
```

```
// 获取值相关的API
Int getInt(String columnName);

// columnIndex序号从 1开始...
Int getInt(Integer columnIndex);
```

SQL类型	Jdbc对应方法 (resultSet的get...方法)	返回类型 Java
TINYINT	getByte()	Byte
SMALLINT	getShort()	Short
Int	getInt()	Int
BIGINT	getLong()	Long
CHAR,VARCHAR, LONG VARCHAR	getString()	String
Text(clob) Blob	getClob getBlob()	Clob Blob
DATE	getDate()	java.sql.Date
TIME	getTime()	java.sql.Time
TIMESTAMP	getTimestamp()	java.sql.Timestamp

5. 数据库注入问题

登录的案例：

```
// 获取statement对象
Statement statement = connection.createStatement();

String sql = "select * from user where username = '"+username+"' and password = '"+password + "'";

System.out.println("sql:" + sql);

// 执行SQL语句
ResultSet resultSet = statement.executeQuery(sql);
```

```

boolean ret = login("张飞", "456");

// select * from user where username = 'aaa' and password = 'bbb';

// select * from user where username = 'aaa' and password = 'sdhdfjf' or
'1=1';

```

由于上面单引号中的内容都是用户自己输入的，后面会把这个内容直接拼接到SQL语句上，会产生安全性的问题。

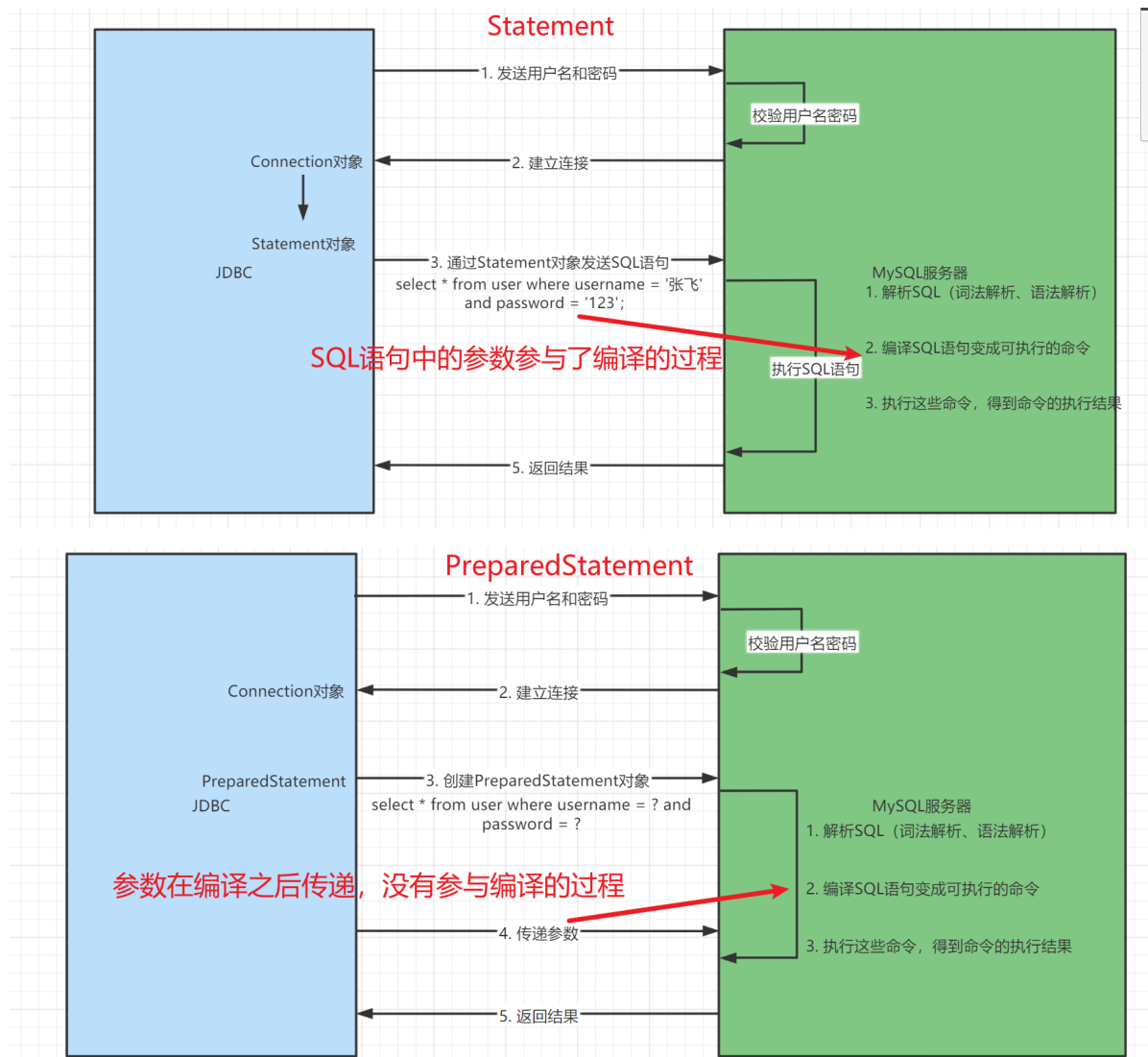
因为可能用户输入的字符串中有一些SQL中的关键字，可能会改变这个SQL语句的结构，导致绕过程序的检查，这个问题就是数据库的注入问题。

SQL注入问题产生的根本的原因是因为MySQL服务器把用户输入进来的参数当成了SQL语法中的关键字。

如何解决数据库的注入问题呢？

PreparedStatement可以解决这个问题。

6. PreparedStatement



```

// 安全的登录
public static boolean loginUsePreparedStatement(String username, String password)
throws SQLException {

```

```

// 注册驱动
DriverManager.registerDriver(new Driver());

// 获取连接
Connection connection = DriverManager.getConnection(FirstJDBC.url,
FirstJDBC.username, FirstJDBC.password);

// 创建PreparedStatement对象
PreparedStatement prepareStatement = connection.prepareStatement("select *
from user where username = ? and password = ?");

// 传参
prepareStatement.setString(1,username);
prepareStatement.setString(2,password);

// 真正的把SQL语句中缺失的参数传递给MySQL服务器
ResultSet resultSet = prepareStatement.executeQuery();

if (resultSet.next()) {
    return true;
}else {
    return false;
}
}

```

总结：PreparedStatement对比Statement有哪些优缺点？

- 优点：没有SQL注入的问题，更加安全
- 缺点：从单条SQL的执行上来说，Statement只用与数据库通信一次，而PreparedStatement要通信两次。

7. 优化JDBC流程

7.1 提取工具类

- 获取连接的方法
- 关闭资源

```

// 获取连接
public static Connection getConnection(){

    Connection connection = null;
    try {
        // 注册驱动
        DriverManager.registerDriver(new Driver());

        // 获取连接

        connection = DriverManager.getConnection(url, username, password);

    }catch (Exception ex) {
        ex.printStackTrace();
    }
}

```



```

    }

    return connection;
}

// 关闭资源
public static void close(ResultSet resultSet, Statement statement, Connection
connection){

    try {

        if (resultSet != null) resultSet.close();

        if (statement != null) statement.close();

        if (connection != null) connection.close();

    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

```

7.2 单例模式

上面的方式每一次调用方法都会创建一个新的Connection对象，而Connection对象是可以复用的，这里我们可以考虑使用单例模式，让每一次都获取到的是同一个Connection对象。

```

// 饿汉
static Connection connection = null;

public static String url = "jdbc:mysql://localhost:3306/44th?
characterEncoding=utf8&useSSL=false";
public static String username = "root";
public static String password = "123456";

static {

    try {
        // 注册驱动
        DriverManager.registerDriver(new Driver());

        // 获取连接

        connection = DriverManager.getConnection(url, username, password);

    } catch (Exception ex) {
        ex.printStackTrace();
    }

}

```

7.3 配置化

// 饿汉

```
static Connection connection = null;
```

```
public static String url = null;
```

```
public static String username = null;
```

```
public static String password = null;
```

```
static {
```

```
try {
```

```
// 获取配置文件中的值
```

```
Properties properties = new Properties();
```

```
FileInputStream fileInputStream = new FileInputStream(name: "jdbc.properties");
```

```
properties.load(fileInputStream);
```

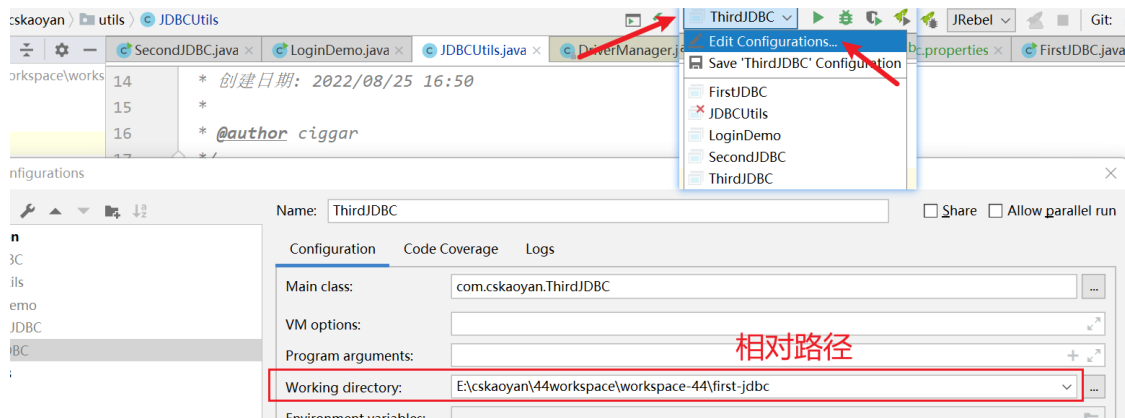
```
url = properties.getProperty("url");
```

```
username = properties.getProperty("username");
```

```
password = properties.getProperty("password");
```

从配置文件中取值

需要注意的是：在读取配置文件的输入流的时候，要搞清楚相对路径是相对的 working directory



7.4 反射注册驱动

1. 注册驱动

2. 获取连接

3. 获取Statement对象

4. 执行SQL语句

5. 解析结果集

6. 关闭资源

1. 提取公共方法

2. 可配置化

3. 单例模式

不安全，使用PreparedStatement替代

提取公共方法

```
url = properties.getProperty("url");  
username = properties.getProperty("username");  
password = properties.getProperty("password");  
driver = properties.getProperty("driver");
```

```
// 注册驱动
```

```
// 自动注册驱动 SPI
```

```
Class.forName(driver);
```

反射注册驱动

总结：通过反射来注册驱动的根本目的是为了解耦，让我们的程序拓展性更强，和导入的MySQL驱动程序jar包的耦合度更低。

8. 批处理

批处理就是指批量的处理SQL语句。

8.1 for循环

```
// for循环
public static void foreachInsert(int num) throws SQLException {

    for (int i = 0; i < num; i++) {

        Statement statement = connection.createStatement();

        String sql = "insert into stu values (null,'" + "foreach:" + i + "',null)";

        statement.executeUpdate(sql);

        statement.close();

    }
}
```

8.2 Statement

```
// statement进行批处理
public static void statementBatch(int num) throws SQLException {

    // 创建Statement对象
    Statement statement = connection.createStatement();

    for (int i = 0; i < num; i++) {

        String sql = "insert into stu values (null,'" + "statement:" + i
+ "',null)";
        statement.addBatch(sql);
    }

    // 批量的发送SQL语句
    statement.executeBatch();

    statement.close();

}
```

8.2 PreparedStatement

提升效率需要在url后面添加配置：rewriteBatchedStatements=true

```
// PreparedStatement批处理
public static void preparedStatementBatch(int num) throws SQLException {

    // 1. 创建PreparedStatement
    PreparedStatement preparedStatement = connection.prepareStatement("insert
into stu values (null,?,?)");

    // 2. 设值
    for (int i = 0; i < num; i++) {

        preparedStatement.setString(1,"preparedStatement:" + i);
        preparedStatement.setString(2,null);

        preparedStatement.addBatch();

    }

    // 3. 执行SQL语句
    preparedStatement.executeBatch();

    preparedStatement.close();
}
```

总结：For循环是最慢的，PreparedStatement的批处理是最快的。

假如需要执行n条SQL	通信次数	编译次数	执行次数
for循环	n	n	n
Statement	1	n	n
PreparedStatement	2	1	1

9. 事务(面试)

9.1 介绍

事务是指组成一个业务操作的各个单元，要么就都成功，要么就都不成功。

例如：在一个转账案例中，A给B转钱，转5000，这个业务操作分为以下的两步：

- 给A的账户扣钱
- 给B的账户加钱

以上的两步操作要保证，要么就都成功，要么就都不成功，不然就会出现问題。

9.2 使用

- SQL语句中的使用

```
# 开启事务
start transaction;

# 提交事务
commit;

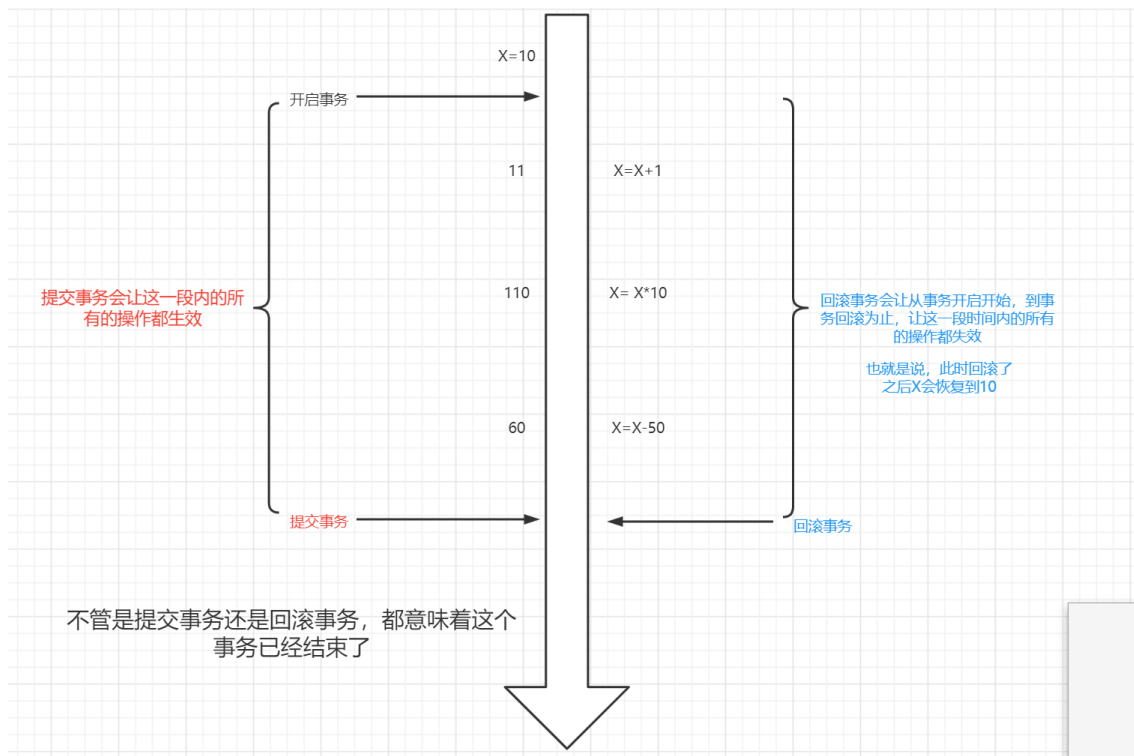
# 回滚事务
rollback;
```

- JDBC中的使用

```
// 开启事务 // 关闭事务的自动
connection.setAutoCommit(false);

// 提交事务
connection.commit();

// 回滚事务
connection.rollback();
```



转账案例:

```
// 转账的方法
/**
 * 创建日期: 2022/08/26 11:17
 * @param fromName 转账发起人
 * @param toName 转账收益人
```

```

    * @param money      转账金额
    * @return java.lang.Boolean
    * @author ciggar
    */
    public static Boolean transfer(String fromName,String toName,Integer money)
    throws SQLException {

        Connection connection = JDBCUtils.getConnection();

        // 开启事务
        connection.setAutoCommit(false);

        try {

            // 给指定的账户扣钱
            PreparedStatement decreaseStatement =
            connection.prepareStatement("update account set money = money - ? where name
            = ? and money >= ?");

            decreaseStatement.setInt(1, money);
            decreaseStatement.setString(2, fromName);
            decreaseStatement.setInt(3, money);

            int affectedRows = decreaseStatement.executeUpdate();

            if (affectedRows > 0) {
                System.out.println("扣钱成功! ");
            } else {
                throw new RuntimeException("扣钱失败! ");
            }
            decreaseStatement.close();

            int i = 1 / 0;

            // 给指定的账户加钱
            PreparedStatement addStatement = connection.prepareStatement("update
            account set money = money + ? where name = ?");

            addStatement.setInt(1, money);
            addStatement.setString(2, toName);

            int affectedRows2 = addStatement.executeUpdate();
            if (affectedRows2 > 0) {
                System.out.println("加钱成功! ");
            } else {
                throw new RuntimeException("加钱失败! ");
            }
        } catch (Exception ex) {
            ex.printStackTrace();
            // 回滚

```

```
        connection.rollback();
        return false;
    }
    // 提交事务
    connection.commit();
    return true;
}
```

9.3 特性

事务有四个特性：ACID

- 原子性 (Atomic)

事务就是一个原子操作，事务中的各个操作单元要么都成功，要么都不成功。

- 一致性 (Consistence)

事务在操作之前和操作之后要保持一致。

- 隔离性 (Isolation)

隔离性是指事务与事务之间互相隔离，互不影响。

- 持久性 (Durability)

事务一旦提交，对数据库的改变是永久性的。

//原子性是基础，隔离性是手段，一致性是限制条件，持久性是目的

事务的状态：

//活动的
//部分提交的
//提交的
//未提交的
//失败的
//中止的

9.4 隔离级别

针对隔离性，数据库制定的不同的隔离级别。

- read uncommitted

读未提交

- read committed

读已提交

- repeatable read

可重复读（这个是MySQL默认的隔离级别）

- serializable

串行化

四种不同的隔离级别下可能就有以下三种问题：

- 脏读

一个事务读取到了另外一个事务还没有提交的数据。

- 不可重复读（指修改）

在同一个事务中，读取同一份数据，前后不一致。

- 虚幻读（删除或者是新增）

在同一个事务中，可能有的时候能读取到一些数据，有的时候又读取不到这些数据，就好像这些数据虚无缥缈一样

查看

```
select @@transaction_isolation;
select @@tx_isolation;
```

修改(全局/当前连接)

```
set global/session transaction isolation level read uncommitted;
```

注意：修改了全局的隔离级别之后，需要重新打开连接

read uncommitted

1. 开启事务

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)
```

2. 查询

```
mysql> select * from account;
```

id	name	nickname	money
1001	风华哥	老板	9000
1002	长风	宝马哥	2000
1003	天明	U盘哥	4000
1004	景天	奶茶哥哥	3000

4 rows in set (0.00 sec)

3. 修改

```
mysql> update account set money = 10000 where id = 1001;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

4. 再次查询

```
mysql> rollback;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

查询到了另外一个事务还没有提交的数据，有脏读

1. 开启事务

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)
```

2. 查询

```
mysql> select * from account;
```

id	name	nickname	money
1001	风华哥	老板	9000
1002	长风	宝马哥	2000
1003	天明	U盘哥	4000
1004	景天	奶茶哥哥	3000

4 rows in set (0.00 sec)

3. 删除数据

```
mysql> delete from account where id = 1004;
Query OK, 1 row affected (0.00 sec)
```

4. 再次查询

```
mysql> rollback;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

景天这条数据时而有时无，有虚幻读的问题

在read uncommitted这种隔离级别下，有脏读、不可重复读、虚幻读的问题。

这个隔离级别是最不安全的隔离级别，一般不使用。

read committed

读已提交。

Session 1 (Left):

```
mysql> start transaction;
-> ;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from account;
+----+-----+-----+-----+
| id | name  | nickname | money |
+----+-----+-----+-----+
| 1001 | 风华哥 | 老板 | 9000 |
| 1002 | 长风 | 宝马哥 | 2000 |
| 1003 | 天明 | U盘哥 | 4000 |
| 1004 | 景天 | 奶茶哥哥 | 3000 |
+----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> update account set money = 10000 where id = 1002;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> select * from account;
+----+-----+-----+-----+
| id | name  | nickname | money |
+----+-----+-----+-----+
| 1001 | 风华哥 | 老板 | 9000 |
| 1002 | 长风 | 宝马哥 | 10000 |
| 1003 | 天明 | U盘哥 | 4000 |
| 1004 | 景天 | 奶茶哥哥 | 3000 |
+----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Session 2 (Right):

```
Database changed
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from account;
+----+-----+-----+-----+
| id | name  | nickname | money |
+----+-----+-----+-----+
| 1001 | 风华哥 | 老板 | 9000 |
| 1002 | 长风 | 宝马哥 | 2000 |
| 1003 | 天明 | U盘哥 | 4000 |
| 1004 | 景天 | 奶茶哥哥 | 3000 |
+----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> select * from account;
+----+-----+-----+-----+
| id | name  | nickname | money |
+----+-----+-----+-----+
| 1001 | 风华哥 | 老板 | 9000 |
| 1002 | 长风 | 宝马哥 | 2000 |
| 1003 | 天明 | U盘哥 | 4000 |
| 1004 | 景天 | 奶茶哥哥 | 3000 |
+----+-----+-----+-----+
4 rows in set (0.00 sec)

mysql> select * from account;
+----+-----+-----+-----+
| id | name  | nickname | money |
+----+-----+-----+-----+
| 1001 | 风华哥 | 老板 | 9000 |
| 1002 | 长风 | 宝马哥 | 10000 |
+----+-----+-----+-----+
2 rows in set (0.00 sec)
```

Annotations:

- 1. 开启事务
- 2. 查询
- 3. 修改数据
- 4. 再次查询
- 5. 提交
- 6. 再次查询
- 没有查询到另外一个事务未提交的数据
- 没有脏读的问题
- 读取到了另外一个事务提交的数据
- 有不可重复读的问题
- 1. 开启事务
- 2. 查询
- 3. 新增数据
- 4. 提交
- 5. 再次查询
- 存在虚幻读的问题

read committed这种隔离级别下没有 脏读的问题，但是有不可重复读和虚幻读的问题。

repeatable read

可重复读。

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from account;
ERROR 1046 (3D000): No database selected
mysql>
mysql> use 44th;
Database changed
mysql> select * from account;
```

id	name	nickname	money
1001	风华哥	老板	9000
1002	长风	宝马哥	10000
1003	天明	U盘哥	4000
1004	景天	奶茶哥哥	3000
2001	卢本伟	五五开	20000

```
5 rows in set (0.00 sec)

mysql> update account set money = 5000 where id = 1004;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from account;
```

id	name	nickname	money
1001	风华哥	老板	9000
1002	长风	宝马哥	10000
1003	天明	U盘哥	4000

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> use 44th;
Database changed
mysql>
mysql> select * from account;
```

id	name	nickname	money
1001	风华哥	老板	9000
1002	长风	宝马哥	10000
1003	天明	U盘哥	4000
1004	景天	奶茶哥哥	3000
2001	卢本伟	五五开	20000

```
5 rows in set (0.00 sec)

mysql> select * from account;
```

id	name	nickname	money
1001	风华哥	老板	9000
1002	长风	宝马哥	10000
1003	天明	U盘哥	4000
1004	景天	奶茶哥哥	3000
2001	卢本伟	五五开	20000

```
5 rows in set (0.00 sec)
```

1. 开启事务

2. 查询数据

3. 修改

4. 再次查询

5. 提交

6. 再次查

对于右边的事务来说，没有读取到左边的事务还没有提交的数据，没有脏读

也没有读取到左边的事务提交的数据，没有不可重复读的问题

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from account;
```

id	name	nickname	money
1001	风华哥	老板	9000
1002	长风	宝马哥	10000
1003	天明	U盘哥	4000
1004	景天	奶茶哥哥	5000
2001	卢本伟	五五开	20000

```
5 rows in set (0.00 sec)

mysql> insert into account values (3001,'张天爱','小爱同学',50000);
Query OK, 1 row affected (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

```
mysql> select * from account;
```

id	name	nickname	money
1001	风华哥	老板	9000
1002	长风	宝马哥	10000
1003	天明	U盘哥	4000
1004	景天	奶茶哥哥	5000
2001	卢本伟	五五开	20000

```
5 rows in set (0.00 sec)

mysql> update account set money = 1000 where id = 1004;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> select * from account;
```

id	name	nickname	money
1001	风华哥	老板	9000
1002	长风	宝马哥	10000
1003	天明	U盘哥	4000
1004	景天	奶茶哥哥	5000
2001	卢本伟	五五开	20000
3001	张天爱	小爱同学	1000

```
6 rows in set (0.00 sec)
```

1. 开启事务

2. 新增数据

3. 查询，没有查到

4. 修改数据

5. 再次查询，查到了对应的数据

认为有虚幻读的问题

总结：在可重复读这种隔离级别下，有虚幻读(MVCC)的问题，没有脏读和不可重复读的问题。

serializable

串行化。事务是串行化执行的，不存在并发执行事务的情况，所以没有脏读、不可重复读、虚幻读的问题。

但是有效率的问题。

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from account;
```

id	name	nickname	money
1001	风华哥	老板	9000
1002	长风	宝马哥	10000
1003	天明	U盘哥	4000
1004	景天	奶茶哥哥	5000
2001	卢本伟	五五开	20000
3001	张天爱	小爱同学	50000

```
6 rows in set (0.00 sec)

mysql> update account set money = 100 where id = 1001;
Query OK, 1 row affected (18.36 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql>
```

1. 开启事务

2. 修改

在等待另外一个事务执行结束

```
Database changed
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from account;
```

id	name	nickname	money
1001	风华哥	老板	9000
1002	长风	宝马哥	10000
1003	天明	U盘哥	4000
1004	景天	奶茶哥哥	5000
2001	卢本伟	五五开	20000
3001	张天爱	小爱同学	50000

```
6 rows in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

总结

	脏读	不可重复读	虚幻读
read uncommitted(读未提交)	√	√	√
read committed (读已提交)	X	√	√
repeatable read (可重复读)	X	X	√
serializable (串行化)	X	X	X