JDBC 实战教程——尚硅谷学习笔记 2023 年

第 1 章 前言

1.1 课程需要哪些前置技术

技术	版本	备注
IDEA	2022.3.3	最新版本
JDK	17	
MySQL-JDBC 驱动	8.0.31	8.0.25+
Druid	1.2.17	
MySQL	8.0.31	

- 前置技术
    - 需要软件
        - MySQL 软件安装(8+ 版本)
        - MySQL 可视化工具安装
        - IDEA 工具安装(推荐 2022 版本)
    - SQL 语句
        - 掌握数据库连接命令
        - 掌握基本的 DDL，DQL，DML 等命令
        - 掌握数据库事务概念
    - Java 基础语法
        - 多态机制
        - 基本容器使用(集合和数组等)
        - 泛型
        - 反射等技术

1.2 课程学习路线设计

- 课程学习路径
    - 阶段一： JDBC 版本和概念理解
        - 标题 1：全新 JDBC 课程前言
        - 标题 2：全新 JDBC 技术概述
    - 阶段二：JDBC 技术核心使用学习
        - 标题 3：全新 JDBC 技术核心 API
        - 标题 4：全新 JDBC 扩展拔高
    - 阶段三：JDBC 连接性能优化连接池使用
        - 标题 5：国货之光 Druid 连接池技术使用
    - 阶段四：JDBC 使用极致优化工具类高阶封装
        - 标题 6: 全新 JDBC 使用优化以及工具类封装
    - 阶段五：实践出真知，CustomerManageSystem 项目实战
        - 标题 7: 基于 CustomerManageSystem 项目 JDBC 实战练习

第 2 章 全新 JDBC 技术概述

2.1 JDBC 技术概念和理解

- JDBC 技术理解
    - JDBC（Java Database Connectivity）是 Java 语言中访问关系型数据库的标准接口，它定义了一组 API，使得 Java 程序可以通过统一的方式连接、访问、操作不同的关系型数据库。JDBC API 提供了一套标准的接口，使得开发者可以使用 Java 语言来访问关系型数据库，而不必关心不同数据库之间的差异。
    - JDBC 有两个核心组件：JDBC 驱动程序和 JDBC API。JDBC 驱动程序通过提供特定数据库的实现，将 JDBC API 转换成数据库可以理解的命令。JDBC API 包括了一系列 Java 类和接口，使得开发者可以使用 Java 语言来执行 SQL 查询、更新和管理数据库连接等操作。
    - 使用 JDBC 连接数据库的步骤一般包括以下几个步骤：
      1 . 加载 JDBC 驱动程序：使用 Class.forName()方法加载特定数据库的

JDBC 驱动程序。
2 . 创建数据库连接：使用 DriverManager.getConnection()方法创建一个数据库连接对象，该方法需要指定数据库的连接字符串、用户名和密码等信息。
3 . 创建 Statement 对象：使用数据库连接对象创建一个 Statement 对象，用于执行 SQL 查询和更新操作。
4 . 执行 SQL 语句：使用 Statement 对象执行 SQL 语句，包括查询、更新和删除等操作。
5 . 处理结果集：如果执行查询操作，则需要使用 ResultSet 对象处理查询结果。
6 . 关闭数据库连接：使用数据库连接对象的 close()方法关闭数据库连接。

- JDBC 是 Java 平台上与数据库交互的标准方式，它的应用广泛，可以用于开发各种类型的 Java 应用程序，包括 Web 应用程序、桌面应用程序和移动应用程序等。

- JDBC 概念总结
    1. JDBC 是(Java Database Connectivity)单词的缩写,翻译为 java 连接数据库。
    2. JDBC 是 java 程序连接数据库的技术统称。
    3. JDBC 由 java 语言的规范(接口)和各个数据库厂商的实现驱动(jar)组成。
    4. JDBC 是一种典型的面向接口编程。
    5. JDBC 优势
        - 只需要学习 JDBC 规范接口的方法，即可操作所有的数据库软件。
        - 项目中期切换数据库软件，只需要更换对应的数据库驱动 jar 包，不需要更改代码。

2.2 JDBC 核心 API 和使用路线

- JDBC 技术组成
    1. JDK 下 JDBC 规范接口, 存储在 java.sql 和 javax.sql 包中的 API

        > 为了项目代码的可移植性，可维护性，SUN 公司从最初就制定了 Java 程序连接各种数据库的统一接口规范。这样的话，不管是连接哪一种 DBMS 软件，Java 代码可以保持一致性。

2. 各个数据库厂商提供的驱动 jar 包

   > 因为各个数据库厂商的 DBMS 软件各有不同，那么内部如何通过 sql 实现增、删、改、查等管理数据，只有这个数据库厂商自己更清楚，因此把接口规范的实现交给各个数据库厂商自己实现。

   jar 包是什么?

   > Java 程序打成的一种压缩包格式，你可以将这些 jar 包引入你的项目中，然后你可以使用这个 Java 程序中类和方法以及属性。

- 涉及具体核心类和接口
  - DriverManager
    - 将第三方数据库厂商的实现驱动 jar 注册到程序中。
    - 可以根据数据库连接信息获取 connection。
  - Connection
    - 和数据库建立的连接,在连接对象上,可以多次执行数据库 CURD 动作。
    - 可以获取 statement 和 preparedstatement，callablestatement 对象。
  - Statement 、 PreparedStatement 、 CallableStatement
    - 具体发送 SQL 语句到数据库管理软件的对象。
    - 不同发送方式稍有不同， preparedstatement 使用为重点!
  - Result
    - 面向对象思维的产物(抽象成数据库的查询结果表)。
    - 存储 DQL 查询数据库结果的对象。
    - 需要我们进行解析，获取具体的数据库数据。
- JDBC API 使用路线
  - 静态 SQL 路线(没有动态值语句)
    - DriverManager
    - Connection
    - Statement
    - Result
  - 预编译 SQL 路线(有动态值语句)
    - DriverManager
    - Connection
    - PreparedStatement

        - Result
    - 执行标准存储过 SQL 路线
        - DriverManager
        - Collection
        - CallableStatement
        - Result

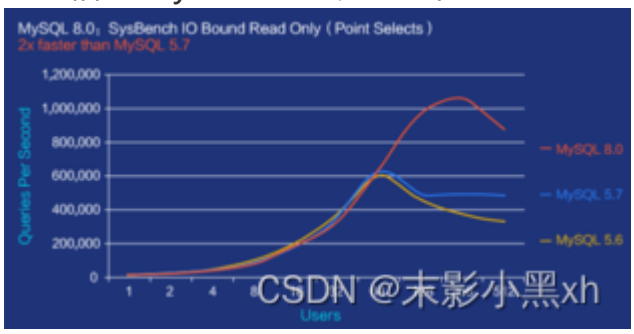
2.3 为什么选择全新 8+ 版本 MySQL-JDBC 驱动?

MySQL Connector/J 8.0 开发人员指南 https://dev.mysql.com/doc/connector-j/8.0/en/

- 支持 8.0+ 版本 MySQL 数据管理软件
    - MySQL 软件知名版本迭代时间

版本号	迭代时间	大小
MySQL-8.0.25	4 月 30, 2021	435.7M
MySQL-5.7.25	1 月 10, 2019	387.7M
MySQL-5.5.30	9 月 19, 2012	201.5M

    - MySQL 8.x 版本数据库性能提升介绍
      性能提升级。官方表示 MySQL 8.0 的速度要比 MySQL 5.7 快 2 倍。MySQL 8.0 在读写工作负载、IO 密集型工作负载、以及高竞争工作负载时相比 MySQL5.7 有更好的性能。



- 支持 Java JDBC 规范 4.2+ 版本新特性
    - Java JDBC 规范驱动版本和更新时间

Year	JDBC Version	JSR Specification	JDK Implementation
2017	JDBC 4.3	JSR 221	Java SE 9
2014	JDBC 4.2	JSR 221	Java SE 8
2011	JDBC 4.1	JSR 221	Java SE 7
2006	JDBC 4.0	JSR 221	Java SE 6
2001	JDBC 3.0	JSR 54	JDK 1.4
1999	JDBC 2.1		JDK 1.2
1997	JDBC 1.2		JDK 1.1

- JDBC 规范版本更新内容
  - JDBC 4.3 中引入的主要新功能包括：
    添加了对分片的支持。
    添加了 java.sql 连接生成器接口。
    添加了 java.sql.ShardigKey 接口。
    添加了 java.sql 分片密钥生成器接口。
    添加了 java.sql.XA 连接生成器接口。
    添加了 javax.sql 池连接生成器接口。
  - JDBC 4.2 中引入的主要新功能包括：
    添加了对引用光标的支持。
    添加了 java.sql 驱动程序操作接口。
    添加了 java.sql.SQLType 接口。
    添加 java.sql.JDBCType 枚举。
    一些 JDBC 接口更改。
  - JDBC 4.1 中引入的主要新功能包括：
    添加了对“使用资源试用”语句的支持。
    增强的日期值和时间戳值。
    从 Java 对象到 JDBC 类型的其他映射。
    一些 JDBC 接口更改。
  - JDBC 4.0 中引入的主要新功能包括：
    自动加载 java.sql 驱动程序。

数据类型支持。
国家字符集转换支持。
    - 由于 JDBC 4.3 API 是向后兼容的，因此将 Java SE 9 或更高版本与 JDBC 4.2、4.1、4.0 或 3.0 驱动程序一起使用没有问题，只要不使用 JDBC 4.3 API 中引入的新方法或类。
- 支持 JDK1.8 版本语法变更新特性
  Connector/J 8.0 是专门为在 Java 8 平台上运行而创建的。
  众所周知，Java8 与早期的 Java 版本高度兼容，但还是存在少量不兼容性，所以驱动技术版本尽量选择支持 jdk 8.0+。
- 支持全新的驱动 API，增加自动时区选择和默认 utf-8 编码格式等配置。

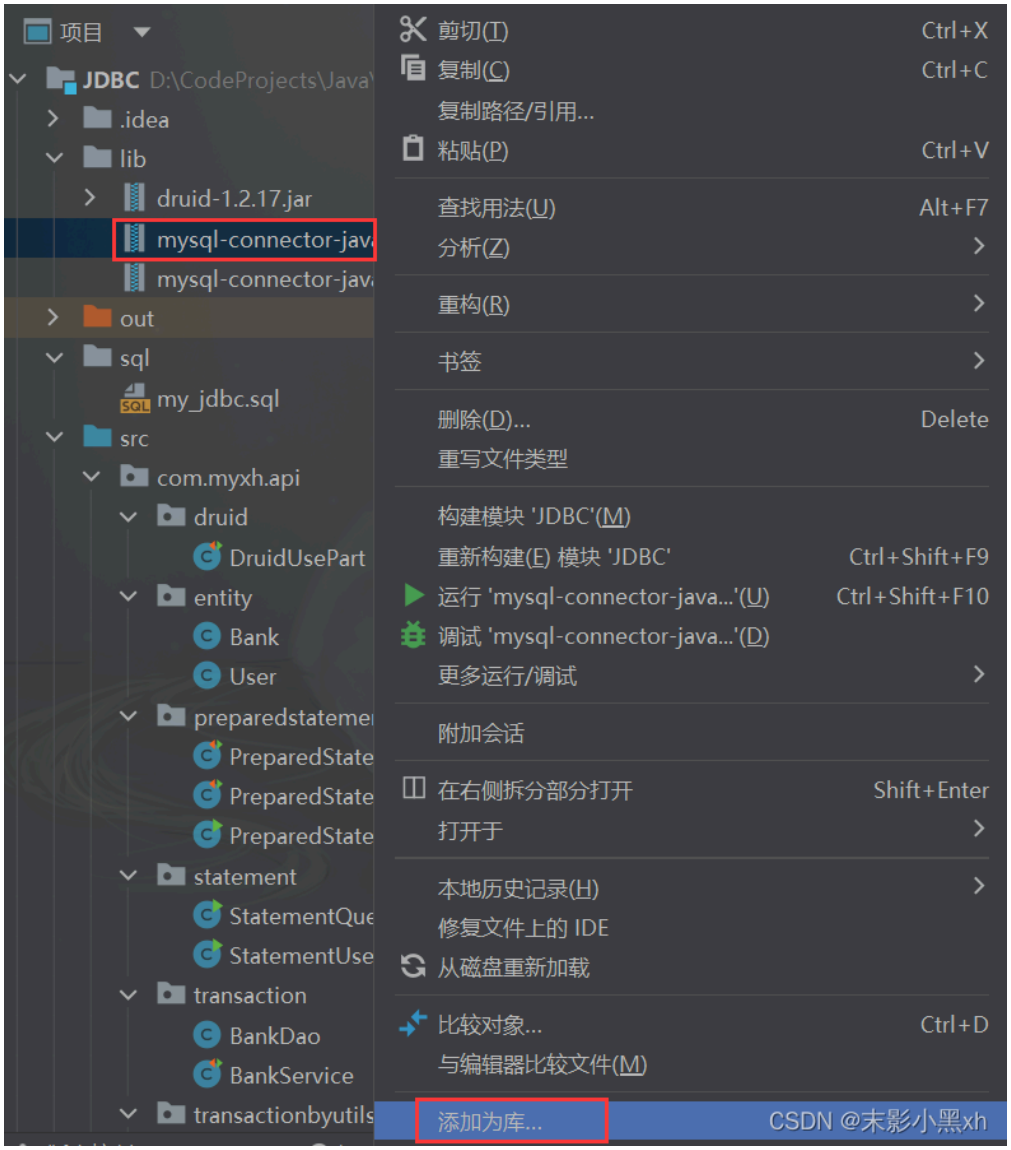
第 3 章 全新 JDBC 核心 API

3.1 引入 MySQL-JDBC 驱动 jar

1. 驱动 jar 版本选择
   我们选择版本 8.0.27 版本。

MySQL 版本	推荐驱动版本	备注
MySQL 5.5.x	5.0.x	com.mysql.jdbc.Driver
MySQL 5.7.x	5.1.x	com.mysql.jdbc.Driver
MySQL 8.x	8.0.x	建议：8.0.25+ 省略时区设置 com.mysql.cj.jdbc.Driver

2. Java 工程导入依赖
    1. 项目创建 lib 文件夹。
    2. 导入驱动依赖 jar 包。
    3. jar 包右键添加为项目依赖。



3.2 JDBC 基本使用步骤分析（6 步）

1. 注册驱动。
2. 获取连接。
3. 创建发送 SQL 语句对象。
4. 发送 SQL 语句，并获取返回结果。
5. 结果集解析。
6. 资源关闭。

3.3 基于 statement 演示查询

- 准备数据库数据

```sql
CREATE
    DATABASE IF NOT EXISTS my_jdbc;

USE
    my_jdbc;

CREATE TABLE IF NOT EXISTS t_user
(
    id       INT PRIMARY KEY AUTO_INCREMENT COMMENT '用户主键',
    account  VARCHAR(20) NOT NULL UNIQUE COMMENT '账号',
    password VARCHAR(64) NOT NULL COMMENT '密码',
    nickname VARCHAR(20) NOT NULL COMMENT '昵称'
);

INSERT INTO t_user(account, password, nickname)
VALUES ('root', '123456', '经理'),
    ('admin', '000000', '管理员');

-- 查询需求: 查询全部用户数据

SELECT id, account, password, nickname
FROM t_user;

SELECT *
FROM t_user
WHERE account = 'root'
AND password = '123456';
```

- 查询目标

  查询全部用户信息，进行控制台输出。

id	account	password	nickname
1	root	123456	经理
2	admin	000000	管理员

- 基于 statement 实现查询(演示步骤)

```java
package com.myxh.api.statement;

import com.mysql.cj.jdbc.Driver;

import java.sql.*;

/**
 * @author MYXH
 * @date 2023/5/9
 * @Description 利用 JDBC 技术，完成用户数据查询工作
 *              使用 Statement 查询 t_user 表的全部数据
 * TODO: 步骤总结（6步）
 *     1. 注册驱动
 *     2. 获取连接
 *     3. 创建 Statement
 *     4. 发送 SQL 语句，并获取结果
 *     5. 结果集解析
 *     6. 关闭资源
 */
public class StatementQueryPart
{
    /**
     * TODO:
     *     DriverManger
     *     Connection
     *     Statement
     *     ResultSet
     */
    public static void main(String[] args) throws SQLException
    {
        // 1. 注册驱动
        /*
        TODO:
            注册驱动
            依赖：驱动版本 8+ com.mysql.cj.jdbc.Driver
                驱动版本 5+ com.mysql.jdbc.Driver
        */
        DriverManager.registerDriver(new Driver());

        // 2. 获取链接
        /*
        TODO:
            Java 程序要和数据库创建连接
            Java 程序连接数据库，需要调用某个方法，方法也需要填入连接数据库的基本信息：
```

```
                数据库 ip 地址: 127.0.0.1
                数据库端口号: 3306
                账号: MYXH
                密码: 520.ILY!
                连接数据库的名称: my_jdbc
        */

        /*
        参数1: url
            jdbc:数据库厂商名://ip地址:port/数据库名
            jdbc:mysql://127.0.0.1:3306/my_jdbc
        参数2: username 数据库软件的账号 MYXH
        参数3: password 数据库软件的密码 520.ILY!
        */

        // java.sql 接口 = 实现类
        Connection connection = DriverManager.getConnection("jdbc:mysql://127.0.0.1:330

        // 3. 创建 Statement
        Statement statement = connection.createStatement();

        // 4. 发送 SQL 语句, 并且获取返回结果
        String sql = "select * from t_user;";
        ResultSet resultSet = statement.executeQuery(sql);

        // 5. 进行结果集解析
        // 判断有没有下一行数据, 并获取
        while (resultSet.next())
        {
            int id = resultSet.getInt("id");
            String account = resultSet.getString("account");
            String password = resultSet.getString("password");
            String nickname = resultSet.getString("nickname");

            System.out.println(id + "--" + account + "--" + password + "--" + nickname
        }

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

3.4 基于 statement 方式问题

- 本案例目标
    - 明确 JDBC 流程和详细讲解使用(注册驱动，获取连接，发送语句，结果解析)。
    - 发现问题，引出 preparedstatement。
- 准备数据库数据

  > 上个案例相同的数据库。

```sql
CREATE
    DATABASE IF NOT EXISTS my_jdbc;

USE
    my_jdbc;

CREATE TABLE IF NOT EXISTS t_user
(
    id       INT PRIMARY KEY AUTO_INCREMENT COMMENT '用户主键',
    account  VARCHAR(20) NOT NULL UNIQUE COMMENT '账号',
    password VARCHAR(64) NOT NULL COMMENT '密码',
    nickname VARCHAR(20) NOT NULL COMMENT '昵称'
);

INSERT INTO t_user(account, password, nickname)
VALUES ('root', '123456', '经理'),
    ('admin', '000000', '管理员');
```

- 演示目标

  模拟登录，控制台输入账号和密码，判断是否登陆成功成功。

- 基于 statement 实现模拟登录

```java
package com.myxh.api.statement;

import java.sql.*;
import java.util.Scanner;

/**
 * @author MYXH
 * @date 2023/5/10
 * @Description 输入账号密码，模拟用户登录
 * TODO:
 *      1. 明确 JDBC 的使用流程和详细讲解内部设计 API 方法
 *      2. 发现问题，引出 PreparedStatement
 * TODO:
 *      输入账号和密码
 *      进行数据库信息查询(t_user)
 *      反馈登录成功还是登录失败
 * TODO:
 *      1. 键盘输入事件，收集账号和密码信息
 *      2. 注册驱动
 *      3. 获取连接
 *      4. 创建 Statement
 *      5. 发送查询 SQL 语句，并获取返回结果
 *      6. 结果判断，显示登录成功还是失败
 *      7. 关闭资源
 */
public class StatementUserLoginPart
{
    public static void main(String[] args) throws SQLException, ClassNotFoundException
    {
        // 1. 获取用户输入信息
        Scanner scanner = new Scanner(System.in);
        System.out.println("请输入账号: ");
        String account1 = scanner.nextLine();
        System.out.println("请输入密码: ");
        String password1 = scanner.nextLine();

        // 2. 注册驱动
        /*
        方案 1:
            DriverManager.registerDriver(new com.mysql.cj.jdbc.Driver())
            注意: 8+ com.mysql.cj.jdbc.Driver
                5+ com.mysql.jdbc.Driver
            问题: 注册两次驱动
                1. DriverManager.registerDriver方法本身会注册一次
```

```
            2. Driver.static{ DriverManager.registerDriver() }静态代码块，也会注册一
        解决：只想注册一次驱动
            只触发静态代码块即可，Driver
        触发静态代码块：
            类加载机制：类加载的时刻,会触发静态代码块
                    加载[class 文件 -> JVM 虚拟机的 class 对象]
                    连接[验证(检查文件类型) -> 准备(静态变量默认值) -> 解析(触发静态代码
                    初始化(静态属性赋真实值)
            触发类加载：
                1. new关键字
                2. 调用静态方法
                3. 调用静态属性
                4. 接口 jdk1.8 default 默认实现
                5. 反射
                6. 子类触发父类
                7. 程序的入口 main
    */

    // 方案 1:
    // DriverManager.registerDriver(new Driver());

    // 方案 2: 代码扩展性差，升级 MySQL 新版本的驱动或换成 Oracle
    // new Driver();

    // 方案 3: 反射，字符串的 Driver 全限定符可以引导外部的配置文件 -> xx.properties ->
    Class.forName("com.mysql.cj.jdbc.Driver");

    // 3. 获取数据库连接
    /*
    getConnection(1,2,3)方法，是一个重载方法
    允许开发者，用不同的形式传入数据库连接的核心参数

    核心属性：
        1. 数据库软件所在的主机的 ip 地址：localhost 或 127.0.0.1
        2. 数据库软件所在的主机的端口号：3306
        3. 连接的具体库：my_jdbc
        4. 连接的账号：MYXH
        5. 连接的密码：520.ILY!
        6. 可选的信息

    3 个参数：
        String url    数据库软件所在的信息，连接的具体库，以及其他可选信息
                    语法：jdbc:数据库管理软件名称[mysql 或 oracle]://ip地址 或 主机名:
                            &key=value 可选信息
```

```
                具体: jdbc:mysql://127.0.0.1:3306/my_jdbc
                      jdbc:mysql://localhost:3306/my_jdbc
                本机的省略写法: 如果你的数据库软件安装到本机, 可以进行一些省略
                    jdbc:mysql://127.0.0.1:3306/my_jdbc -> jdbc:mysql:///my_jdb
                    省略了[本机地址]和[3306 默认端口号]
                    强调: 必须是本机, 并且端口号使用 3306 才可省略, 用///

    String user         数据库的账号 MYXH
    String password     数据库的密码 520.ILY!

2 个参数:
    String url      : 此 url 和 3 个参数的 url 的作用一样, 数据库 ip,端口号,具体的数
    Properties info: 存储账号和密码
                    Properties 类似于 Map, 只不过 key = value 都是字符串形式

1 个参数:
    String url      : 数据库 ip, 端口号, 具体的数据库, 可选信息(账号密码)
                    jdbc:数据库软件名://ip:port/数据库?key=value&key=value&key=v

                    jdbc:mysql://localhost:3306/my_jdbc?user=MYXH&password=520
                    携带固定的参数名 user password 传递账号和密码信息

url 的路径属性可选信息:
    url?user=账号&password=密码

    8.0.31 版本驱动, 下面都是一些可选属性:
        8.0.25 以后, 自动识别时区, serverTimezone=Asia/Shanghai 不用添加, 8.0.25
        8 版本以后,默认使用的就是 utf-8 格式, useUnicode=true&characterEncoding=u

    serverTimezone=Asia/Shanghai&useUnicode=true&characterEncoding=utf8&usesSL=
*/

Connection connection = DriverManager.getConnection("jdbc:mysql:///my_jdbc", "M

// Properties info = new Properties();
// info.put("user", "MYXH");
// info.put("password", "520.ILY!");
// Connection connection = DriverManager.getConnection("jdbc:mysql://127.0.0.1

// Connection connection = DriverManager.getConnection("jdbc:mysql:///my_jdbc?u

// 4. 创建发送 SQL 语句的 statement 对象
// statement 可以发送 SQL 语句到数据库, 并且获取返回结果
Statement statement = connection.createStatement();
```

```
// 5. 发送 SQL 语句(编写 SQL 语句，发送 SQL 语句)
/*
请输入账号:
hacker
请输入密码:
' or '1' = '1
登录成功!
*/
String sql = "SELECT * FROM t_user WHERE account = '"+account1+"' AND password

/*
SQL 分类: DDL(容器创建，修改,剔除) DML(插入，修改,剔除) DQL(查询) DCL(权限控制) TPL

参数: sql 非 DQL
返回: int
    情况1: DML 返回影响的行数，例如: 删除了三条数据 return 3; 插入了两条 return 2;
    情况2: 非 DML return 0;
int row = executeUpdate(sql)

参数: sql DQL
返回: resultSet 结果封装对象
ResultSet resultSet = executeQuery(sql);
*/

// int i = statement.executeUpdate(sql);
ResultSet resultSet = statement.executeQuery(sql);

// 6. 查询结果集解析 ResultSet
/*
Java 是一种面向对象的思维，将查询结果封装成了 resultSet 对象，我们应该理解，内部一定

resultSet -> 逐行获取数据，行 -> 行的列的数据

A ResultSet object maintains a cursor pointing to its current row of data.
Initially the cursor is positioned before the first row. The next method moves
and because it returns false when there are no more rows in the ResultSet obje
it can be used in a while loop to iterate through the result set.
ResultSet 对象维护一个指向其当前数据行的光标。最初，光标位于第一行之前。next方法将光
它会返回false，所以可以在while循环中使用它来迭代结果集。

想要进行数据解析，我们需要进行两件事情: 1. 移动游标指定获取数据行 2. 获取指定数据行的

1. 游标移动问题
```

```
            resultSet 内部包含一个游标，指定当前行数据
            默认游标指定的是第一行数据之前
            我们可以调用next方法向后移动一行游标
            如果我们有很多行数据，我们可以使用 while(next){获取每一行的数据}

            boolean = next()    true: 有更多行数据,并且向下移动一行
                                false: 没有更多行数据,不移动

            TODO: 移动光标的方法有很多，只需要记 next 即可，配合 while 循环获取全部数据

        2. 获取列的数据问题(获取光标指定的行的列的数据)

            resultSet.get 类型(String columnLabel 或 int columnIndex);
                columnLabel: 列名，如果有别名写别名    select * 或 (id, account, passwor
                                                      select id as aid, account as ac fr
                columnIndex: 列的下角标获取从左向右从 1 开始
        */

        // while (resultSet.next())
        // {
        //     int id = resultSet.getInt(1);
        //     String account2 = resultSet.getString("account");
        //     String password2 = resultSet.getString(3);
        //     String nickname = resultSet.getString("nickname");
        //     System.out.println(id + "--" + account2 + "--" + password2 + "--" + nic
        // }

        // 移动一次光标，只要有数据，就代表登录成功
        if (resultSet.next())
        {
            System.out.println("登录成功! ");
        }
        else
        {
            System.out.println("登录失败! ");
        }

        // 7. 关闭数据
        resultSet.close();
        statement.close();
        connection.close();
    }
}
```

- 存在问题
    1. SQL 语句需要字符串拼接，比较麻烦。
    2. 只能拼接字符串类型，其他的数据库类型无法处理。
    3. 可能发生注入攻击。

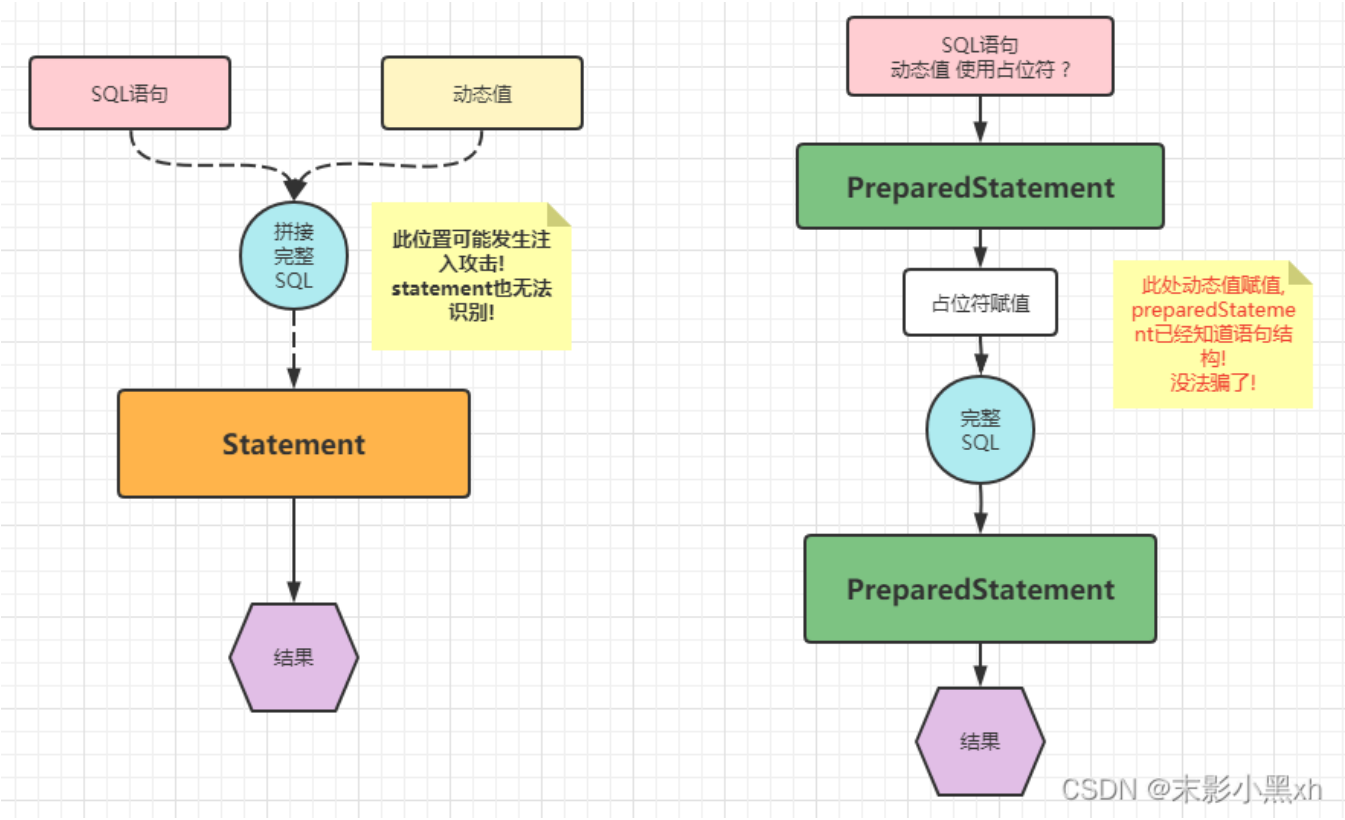
        > 动态值充当了 SQL 语句结构，影响了原有的查询结果。

3.5 基于 preparedStatement 方式优化

> 利用 preparedStatement 解决上述案例注入攻击和 SQL 语句拼接问题。

```java
package com.myxh.api.preparedstatement;

import java.sql.*;
import java.util.Scanner;

/**
 * @author MYXH
 * @date 2023/5/10
 * @Description 使用预编译 Statement 解决注入攻击问题，完成用户登录
 *
 * TODO: 防止注入攻击，演示 PreparedStatement 的使用流程
 */
public class PreparedStatementUserLoginPart
{
    public static void main(String[] args) throws ClassNotFoundException, SQLException
    {
        // 1. 获取用户输入信息
        Scanner scanner = new Scanner(System.in);
        System.out.println("请输入账号: ");
        String account = scanner.nextLine();
        System.out.println("请输入密码: ");
        String password = scanner.nextLine();

        // PreparedStatement 的数据库流程
        // 2. 注册驱动
        Class.forName("com.mysql.cj.jdbc.Driver");

        // 3. 获取连接
        Connection connection = DriverManager.getConnection("jdbc:mysql:///my_jdbc?user=MYXH&passw

        /*
        Statement
            1. 创建 Statement
            2. 拼接 SQL 语句
            3. 发送 SQL 语句，并且获取返回结果

        PreparedStatement
            1. 编写 SQL 语句结果，不包含动态值部分的语句，动态值部分使用占位符 ? 替代，注意: ? 只能替代动
            2. 创建 PreparedStatement，并且传入动态值
            3. 动态值占位符赋值 ? 单独赋值即可
            4. 发送 SQL 语句即可,并获取返回结果
         */

        // 4. 编写 SQL 语句结果
```

```
        String sql = "select * from t_user where account = ? and password = ? ;";

        // 5. 创建预编译 Statement 并且设置 SQL 语句结果
        PreparedStatement preparedStatement = connection.prepareStatement(sql);

        // 6. 单独的占位符进行赋值
        /*
        参数 1: index 占位符的位置从左向右数从 1 开始，账号 ? 1
        参数 2: object 占位符的值可以设置任何类型的数据，避免了我们拼接且类型更加丰富
         */
        preparedStatement.setObject(1, account);
        preparedStatement.setObject(2, password);

        // 7. 发送 SQL 语句，并获取返回结果
        /*
        statement.executeUpdate 或 executeQuery (String sql);
        preparedStatement.executeUpdate 或 executeQuery();    TODO: 因为它已经知道语句，知道语句动态值
         */
        ResultSet resultSet = preparedStatement.executeQuery();

        // 8. 结果集解析
        if (resultSet.next())
        {
            System.out.println("登录成功! ");
        }
        else
        {
            System.out.println("登录失败! ");
        }

        // 9. 关闭数据
        resultSet.close();
        preparedStatement.close();
        connection.close();
    }
}
```



3.6 基于 preparedStatement 演示 CURD

- 数据库数据插入

```java
package com.myxh.api.preparedstatement;

import org.junit.Test;

import java.sql.*;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;


/**
 * @author MYXH
 * @date 2023/5/11
 * @Description 使用 PreparedStatement 进行 t_user 表 的 CURD 操作
 */
public class PreparedStatementCurdPart
{
    // 测试方法需要导入 junit 的测试包
    @Test
    public void testInsert() throws ClassNotFoundException, SQLException
    {
        /*
        t_user 插入一条用户数据
            account: test
            password: test
            nickname: 测试
        */

        // 1. 注册驱动
        Class.forName("com.mysql.cj.jdbc.Driver");

        // 2. 获取连接
        Connection connection = DriverManager.getConnection("jdbc:mysql:///my_jdbc", "
        // 3. 编写 SQL 语句结果，动态值的部分使用 ? 代替
        String sql = "insert into t_user(account, password, nickname) values(?, ?, ?);"
        // 4. 创建 PreparedStatement，并且传入 SQL 语句结果
        PreparedStatement preparedStatement = connection.prepareStatement(sql);

        // 5. 占位符赋值
        preparedStatement.setObject(1, "test");
        preparedStatement.setObject(2, "test");
```

```
        preparedStatement.setObject(3, "测试");

        // 6. 发送 SQL 语句
        // DML 类型
        int rows = preparedStatement.executeUpdate();

        // 7. 输出结果
        if (rows > 0)
        {
            System.out.println("数据插入成功! ");
        }
        else
        {
            System.out.println("数据插入失败! ");
        }

        // 8. 关闭资源
        preparedStatement.close();
        connection.close();
    }
```

- 数据库数据修改

```java
@Test
public void testUpdate() throws ClassNotFoundException, SQLException
{
    /*
    修改 id = 3 的用户 nickname = 测试员
    */

    // 1. 注册驱动
    Class.forName("com.mysql.cj.jdbc.Driver");

    // 2. 获取连接
    Connection connection = DriverManager.getConnection("jdbc:mysql:///my_jdbc", "
    // 3. 编写 SQL 语句结果，动态值的部分使用 ? 代替
    String sql = "update t_user set nickname = ? where id = ?;";

    // 4. 创建 PreparedStatement，并且传入 SQL 语句结果
    PreparedStatement preparedStatement = connection.prepareStatement(sql);

    // 5. 占位符赋值
    preparedStatement.setObject(1, "测试员");
    preparedStatement.setObject(2, 3);

    // 6. 发送 SQL 语句
    int rows = preparedStatement.executeUpdate();

    // 7. 输出结果
    if (rows > 0)
    {
        System.out.println("修改成功! ");
    }
    else
    {
        System.out.println("修改失败! ");
    }

    // 8. 关闭资源
    preparedStatement.close();
    connection.close();
}
```

- 数据库数据删除

```java
@Test
public void testDelete() throws ClassNotFoundException, SQLException
{
    /*
    删除 id = 3 的用户数据
    */

    // 1. 注册驱动
    Class.forName("com.mysql.cj.jdbc.Driver");

    // 2. 获取连接
    Connection connection = DriverManager.getConnection("jdbc:mysql:///my_jdbc", "
    // 3. 编写 SQL 语句结果，动态值的部分使用 ? 代替
    String sql = "delete from t_user where id = ?;";

    // 4. 创建 PreparedStatement，并且传入 SQL 语句结果
    PreparedStatement preparedStatement = connection.prepareStatement(sql);

    // 5. 占位符赋值
    preparedStatement.setObject(1, 3);

    // 6. 发送 SQL 语句
    int rows = preparedStatement.executeUpdate();

    // 7. 输出结果
    if (rows > 0)
    {
        System.out.println("数据删除成功！");
    }
    else
    {
        System.out.println("数据删除失败！");
    }

    // 8. 关闭资源
    preparedStatement.close();
    connection.close();
}
```

- 数据库数据查询

```java
@Test
public void testSelect() throws ClassNotFoundException, SQLException
{
    /*
    目标：查询所有用户数据，并且封装到一个 List<Map> list 集合中

    解释：
        行    id account password nickname
        行    id account password nickname
        行    id account password nickname

    数据库 -> resultSet -> java -> 一行 -> Map(key=列名，value=列的内容) -> List<Map

    实现思路：
        遍历行数据，一行对应一个 Map，获取一行的列名和对应的列的属性，装配即可
        将 Map 装到一个集合就可以了

    难点：
        如何获取列的名称？
    */

    // 1. 注册驱动
    Class.forName("com.mysql.cj.jdbc.Driver");

    // 2. 获取连接
    Connection connection = DriverManager.getConnection("jdbc:mysql:///my_jdbc", "

    // 3. 编写 SQL 语句结果，动态值的部分使用 ? 代替
    String sql = "select id, account as ac, password, nickname from t_user;";

    // 4. 创建 PreparedStatement，并且传入 SQL 语句结果
    PreparedStatement preparedStatement = connection.prepareStatement(sql);

    // 5. 占位符赋值
    // 省略占位符赋值

    // 6. 发送 SQL 语句
    ResultSet resultSet = preparedStatement.executeQuery();

    // 7. 结果集解析
    /*
    TODO：回顾
        resultSet：有行和有列，获取数据的时候，一行一行数据
                内部有一个游标，默认指向数据的第一行之前
```

```
            我们可以利用 next()方法移动游标，指向数据行
            获取行中的列的数据
        */
        List<Map<String, Object>> list = new ArrayList<>();

        // 获取列的信息对象
        //TODO: metaData 装的当前结果集列的信息对象(可以获取列的名称根据下角标,可以获取列的数
        ResultSetMetaData metaData = resultSet.getMetaData();

        // 可以水平遍历列
        int columnCount = metaData.getColumnCount();

        while (resultSet.next())
        {
            Map<String, Object> map = new HashMap<>();
            // 一行数据对应一个 Map

            // 手动取值
            // map.put("id", resultSet.getInt("id"));
            // map.put("account", resultSet.getString("account"));
            // map.put("password", resultSet.getString("password"));
            // map.put("nickname", resultSet.getString("nickname"));

            // 自动遍历列：注意，要从 1 开始，并且小于等于总列数
            for (int i = 1; i <= columnCount; i++)
            {
                // 获取指定列下角标的值，ResultSet
                Object value = resultSet.getObject(i);

                // 获取指定列下角标的列的名称，ResultSetMetaData
                // select * [列名] 或 xxx as name
                // getColumnLabel：会获取别名，如果没有写别名才是列的名称，不要使用getColu
                String columnLabel = metaData.getColumnLabel(i);

                map.put(columnLabel, value);
            }

            // 一行数据的所有列全部存到了 Map 中
            //将 Map 存储到集合中即可
            list.add(map);
        }

        System.out.println("list = " + list);
```

```
        // 8. 关闭资源
        resultSet.close();
        preparedStatement.close();
        connection.close();
    }
}
```

3.7 preparedStatement 使用方式总结

- 使用步骤总结

1. 注册驱动。
2. 获取连接。
3. 编写 SQL 语句。
4. 创建 preparedstatement 并且传入 SQL 语句结构。
5. 占位符赋值。
6. 发送 SQL 语句,并且获取结果。
7. 结果集解析。
8. 关闭资源。

- 使用 API 总结

1 . 注册驱动

```
// 方案1: 调用静态方法, 但是会注册两次
DriverManager.registerDriver(new com.mysql.cj.jdbc.Driver());
// 方案2: 反射触发
Class.forName("com.mysql.cj.jdbc.Driver");
```

2 . 获取连接

```
// 3个参数: (String url,String user,String password)
// 2个参数: (String url,Properties info(user password))
// 1个参数: (String url?user=账号&password=密码 )
Connection connection = DriverManager.getConnection("jdbc:mysql:///my_jdbc", "MYXH", "
```

3 . 创建 Statement

```
// 静态
Statement statement = connection.createStatement("insert into t_user(account, password
// 预编译
PreparedStatement preparedstatement = connection.preparedStatement("insert into t_user
```

4 . 占位符赋值

```
preparedStatement.setObject(1, "test");
preparedStatement.setObject(2, "test");
preparedStatement.setObject(3, "测试");
```

5 . 发送 SQL 语句获取结果

```
int rows = executeUpdate();     //非 DQL
ResultSet resultSet = executeQuery();     // DQL
```

6 . 查询结果集解析

```
// 移动光标指向行数据: next();
// if(next()) {} 或 while(next()) {}
// 获取列的数据即可, get类型(int 列的下角标 从1开始 或 int 列的label (别名或者列名))
// 获取列的信息: getMetadata();
// ResultsetMetaData 对象包含的就是列的信息
// getColumnCount(); 或 getCloumnLebal(index)
```

7 . 关闭资源

```
resultSet.close();
preparedStatement.close();
connection.close();
```

第 4 章 全新 JDBC 扩展提升

4.1 自增长主键回显实现

- 功能需求
    1. java 程序获取插入数据时 mysql 维护自增长维护的主键 id 值，这就是主键回显。

2. 作用：在多表关联插入数据时，一般主表的主键都是自动生成的，所以在插入数据之前无法知道这条数据的主键，但是从表需要在插入数据之前就绑定主表的主键，这是可以使用主键回显技术。

- 功能实现

> 继续沿用之前的表数据

```
CREATE
    DATABASE IF NOT EXISTS my_jdbc;

USE
    my_jdbc;

CREATE TABLE IF NOT EXISTS t_user
(
    id       INT PRIMARY KEY AUTO_INCREMENT COMMENT '用户主键',
    account  VARCHAR(20) NOT NULL UNIQUE COMMENT '账号',
    password VARCHAR(64) NOT NULL COMMENT '密码',
    nickname VARCHAR(20) NOT NULL COMMENT '昵称'
);

INSERT INTO t_user(account, password, nickname)
VALUES ('root', '123456', '经理'),
    ('admin', '000000', '管理员');
```

- PreparedStatementOtherPart 类

```java
package com.myxh.api.preparedstatement;

import org.junit.Test;

import java.sql.*;

/**
 * @author MYXH
 * @date 2023/5/12
 * Description: 练习 PreparedStatement 的特殊使用情况
 */
public class PreparedStatementOtherPart
{
    /**
     * TODO:
     *     t_user 插入一条数据，并且获取数据库自增长的主键
     * TODO: 使用总结
     *     1. 创建 PrepareStatement 的时候,告知携带回数据库自增长的主键, prepareStatement(
     *     2. 获取 PrepareStatement 装主键值的结果集对象，一行一列，获取对应的数据即可, Res
     */
    @Test
    public void testReturnPrimaryKey() throws ClassNotFoundException, SQLException
    {
        // 1. 注册驱动
        Class.forName("com.mysql.cj.jdbc.Driver");

        // 2. 获取连接
        Connection connection = DriverManager.getConnection("jdbc:mysql:///my_jdbc?user

        // 3. 编写 SQL 语句
        String sql = "insert into t_user(account, password, nickname) values (?, ?, ?)

        // 4. 创建 PreparedStatement
        PreparedStatement preparedStatement = connection.prepareStatement(sql, Prepared

        // 5. 占位符赋值
        preparedStatement.setObject(1, "test");
        preparedStatement.setObject(2, "test");
        preparedStatement.setObject(3, "测试员");

        // 6. 发送 SQL 语句,并且获取结果
        int rows = preparedStatement.executeUpdate();

        // 7. 结果解析
```

```
        if (rows > 0)
        {
            System.out.println("数据插入成功! ");

            // 可以获取回显的主键
            // 获取 PrepareStatement 装主键的结果集对象，一行一列，id = 值
            ResultSet generatedKeys = preparedStatement.getGeneratedKeys();
            generatedKeys.next();   // 移动下光标
            int id = generatedKeys.getInt(1);

            System.out.println("id = " + id);
        }
        else
        {
            System.out.println("数据插入失败! ");
        }

        // 8. 关闭资源
        preparedStatement.close();
        connection.close();
    }
```

4.2 批量数据插入性能提升

- 功能需求
    1. 批量数据插入优化。
    2. 提升大量数据插入效率。
- 功能实现

```java
/**
 * 使用普通的方式插入 10000 条数据
 */
@Test
public void testInsert() throws ClassNotFoundException, SQLException
{
    // 1. 注册驱动
    Class.forName("com.mysql.cj.jdbc.Driver");

    // 2. 获取连接
    Connection connection = DriverManager.getConnection("jdbc:mysql:///my_jdbc?user

    // 3. 编写 SQL 语句
    String sql = "insert into t_user(account, password, nickname) values (?, ?, ?)

    // 4. 创建 PreparedStatement
    PreparedStatement preparedStatement = connection.prepareStatement(sql);

    // 5. 占位符赋值
    long startTime = System.currentTimeMillis();

    for (int i = 1; i <= 10000; i++)
    {
        preparedStatement.setObject(1, "test" + i);
        preparedStatement.setObject(2, "test" + i);
        preparedStatement.setObject(3, "测试员" + i);

        // 6. 发送 SQL 语句,并且获取结果
        preparedStatement.executeUpdate();
    }

    long endTime = System.currentTimeMillis();

    // 7. 结果解析: 执行 10000 次数据插入消耗的时间: 15921ms
    System.out.println("执行 10000 次数据插入消耗的时间: " + (endTime - startTime) +

    // 8. 关闭资源
    preparedStatement.close();
    connection.close();
}

/**
 * 使用批量插入的方式插入 10000 条数据
 * TODO: 总结批量插入
```

```
     *      1. 路径后面添加 ?rewriteBatchedStatements=true, 允许批量插入
     *      2. insert into values 必须写, 语句不能添加;结束
     *      3. 不是执行语句每条, 是批量添加 addBatch();
     *      4. 遍历添加完毕以后, 统一批量执行 executeBatch()
     */
    @Test
    public void testBatchInsert() throws ClassNotFoundException, SQLException
    {
        // 1. 注册驱动
        Class.forName("com.mysql.cj.jdbc.Driver");

        // 2. 获取连接
        Connection connection = DriverManager.getConnection("jdbc:mysql:///my_jdbc?rewr

        // 3. 编写 SQL 语句
        String sql = "insert into t_user(account, password, nickname) values (?, ?, ?)"

        // 4. 创建 PreparedStatement
        PreparedStatement preparedStatement = connection.prepareStatement(sql);

        // 5. 占位符赋值
        long startTime = System.currentTimeMillis();

        for (int i = 1; i <= 10000; i++)
        {
            preparedStatement.setObject(1, "BatchTest" + i);
            preparedStatement.setObject(2, "BatchTest" + i);
            preparedStatement.setObject(3, "批量测试员" + i);

            preparedStatement.addBatch();   // 不执行, 追加到values后面
        }

        // 6. 发送 SQL 语句,并且获取结果
        preparedStatement.executeBatch();   // 执行批量操作

        long endTime = System.currentTimeMillis();

        // 7. 结果解析: 执行 10000 次数据插入消耗的时间: 797ms
        System.out.println("执行 10000 次数据插入消耗的时间: " + (endTime - startTime) +

        // 8. 关闭资源
        preparedStatement.close();
        connection.close();
    }
```

```
}
```

4.3 JDBC 中数据库事务实现

- 章节目标
  使用 JDBC 代码，添加数据库事务动作。开启事务，事务提交或事务回滚。
- 事务概念回顾
    - 事务概念
        - 数据库事务就是一种 SQL 语句执行的缓存机制，不会单条执行完毕就更新数据库数据，最终根据缓存内的多条语句执行结果统一判定。
        - 一个事务内所有语句都成功及事务成功，我们可以触发 commit 提交事务来结束事务，更新数据。
        - 一个事务内任意一条语句失败，及事务失败，我们可以触发 rollback 回滚结束事务，数据回到事务之前状态。
    - 优势
        - 允许我们在失败情况下，数据回归到业务之前的状态。
    - 场景
        - 一个业务涉及多条修改数据库语句。
        - 例如：经典的转账案例，转账业务(存钱和取钱)，批量删除(涉及多个删除)，批量添加(涉及多个插入)。
    - 事务特性
        - 原子性（Atomicity）原子性是指事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生。
        - 一致性（Consistency）事务必须使数据库从一个一致性状态变换到另外一个一致性状态。
        - 隔离性（Isolation）事务的隔离性是指一个事务的执行不能被其他事务干扰，即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。
        - 持久性（Durability）持久性是指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来的其他操作和数据库故障不应该对其有任何影响。
    - 事务类型

        - 自动提交：每条语句自动存储一个事务中，执行成功自动提交，执行失败自动回滚。 (MySQL)
        - 手动提交：手动开启事务，添加语句，手动提交或者手动回滚即可。
    - SQl 开启事务方式
        - 针对自动提交： 关闭自动提交即可，多条语句添加以后，最终手动提交或者回滚。 (推荐)

```
-- 关闭当前连接自动事务提交方式
SET autocommit = off;

-- 只有当前连接有效
-- 编写 SQL 语句即可

-- 手动提交或者回滚，结束当前的事务
COMMIT;
ROLLBACK;
```

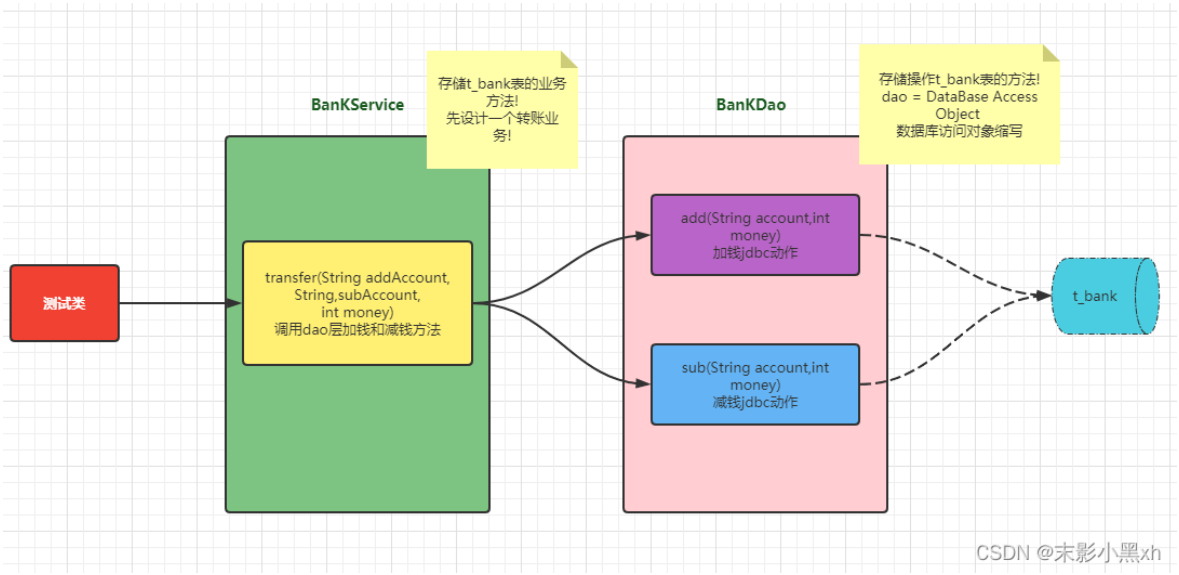
        - 手动开启事务：开启事务代码，添加 SQL 语句，事务提交或者事务回滚.(不推荐)
    - 使用 JDBC 技术

```
try
{
    connection.setAutoCommit(false);    //关闭自动提交

    //注意，只要当前 connection 对象，进行数据库操作，都不会自动提交事务

    //数据库动作
    //statement，单一的数据库动作 CURD

    connection.commit();

}
catch(Execption e)
{
    connection.rollback();
}
```

- 数据库表数据

```sql
-- 继续在 my_jdbc 的库中创建银行表

CREATE TABLE t_bank
(
    id      INT PRIMARY KEY AUTO_INCREMENT COMMENT '账号主键',
    account VARCHAR(20) NOT NULL UNIQUE COMMENT '账号',
    money   DECIMAL(10, 2) UNSIGNED COMMENT '金额不能为负值! '
);

INSERT INTO t_bank(account, money)
VALUES ('经理', 10000),
    ('管理员', 9000);
```

- 代码结构设计



- JDBC 事务实现
    - 测试方法

```java
@Test
public void testStart() throws SQLException, ClassNotFoundException
{
    // 经理给管理员转 500 元
    transFer("管理员", "经理", BigDecimal.valueOf(500));
}
```

    - BankService 类

```java
package com.myxh.api.transaction;

import org.junit.Test;

import java.math.BigDecimal;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

/**
 * @author MYXH
 * @date 2023/5/12
 * Description: bank 表业务类，添加转账业务
 *              银行卡业务方法，调用 BankDAO 的方法
 */
public class BankService
{
    @Test
    public void testStart() throws SQLException, ClassNotFoundException
    {
        // 经理给管理员转 500 元
        transFer("管理员", "经理", BigDecimal.valueOf(500));
    }

    /**
     * TODO:
     *     事务添加是在业务方法中
     *     利用 try catch 代码块，开启事务，提交事务和事务回滚
     *     将 Connection 传入 BankDao 层即可，BankDao 只负责使用，不负责 close
     */
    public void transFer(String addAccount, String subAccount, BigDecimal m
    {
        BankDao bankDao = new BankDao();

        // 一个事务的最基本要求，必须是同一个连接对象 connection
        // 一个转账方法属于一个事务〔存钱、取钱〕
        // 1. 注册驱动
        Class.forName("com.mysql.cj.jdbc.Driver");

        // 2. 获取连接
        Connection connection = DriverManager.getConnection("jdbc:mysql:///

        try
        {
```

```
            // 开启事务
            // 关闭事务提交
            connection.setAutoCommit(false);

            // 执行数据库动作
            bankDao.add(addAccount, money, connection);
            System.out.println("----------");
            bankDao.sub(subAccount, money, connection);

            // 事务提交
            connection.commit();
        }
        catch (Exception e)
        {
            // 事务回滚
            connection.rollback();

            // 抛出异常信息
            throw e;
        }
        finally
        {
            connection.close();
        }
    }
}
```

- BankDao 类

```java
package com.myxh.api.transaction;

import java.math.BigDecimal;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

/**
 * @author MYXH
 * @date 2023/5/12
 * Description: bank 表的数据库操作方法存储类
 */
public class BankDao
{
    /**
     * 存钱的数据库操作方法(jdbc)
     * @param account 存钱的账号
     * @param money 存钱的金额
     */
    public void add(String account, BigDecimal money, Connection connection
    {
        // 3. 编写 SQL 语句结构
        String sql = "update t_bank set money = money + ? where account = ?

        // 4. 创建 PreparedStatement
        PreparedStatement preparedStatement = connection.prepareStatement(s

        // 5. 占位符赋值
        preparedStatement.setObject(1, money);
        preparedStatement.setObject(2, account);

        // 6. 发送 SQL 语句
        preparedStatement.executeUpdate();

        // 7. 关闭资源
        preparedStatement.close();

        System.out.println("存钱成功! ");
    }

    /**
     * 取钱的数据库操作方法(jdbc)
     * @param account 取钱的账号
     * @param money 取钱的金额
```

```
     */
    public void sub(String account, BigDecimal money, Connection connection
    {
        // 3. 编写 SQL 语句结构
        String sql = "update t_bank set money = money - ? where account = ?

        // 4. 创建 PreparedStatement
        PreparedStatement preparedStatement = connection.prepareStatement(s

        // 5. 占位符赋值
        preparedStatement.setObject(1, money);
        preparedStatement.setObject(2, account);

        // 6. 发送 SQL 语句
        preparedStatement.executeUpdate();

        // 7. 关闭资源
        preparedStatement.close();

        System.out.println("取钱成功! ");
    }
}
```

第 5 章 国货之光 Druid 连接池技术使用

5.1 连接性能消耗问题分析

connection 可以复用，不要浪费连接。

5.2 数据库连接池作用

- 不使用连接池的缺点
    - 不使用数据库连接池，每次都通过 DriverManager 获取新连接，用完直接抛弃断开，连接的利用率太低，太浪费。
    - 对于数据库服务器来说，压力太大了。我们数据库服务器和 Java 程序对连接数也无法控制，很容易导致数据库服务器崩溃。
- 管理连接

- 我们可以建立一个连接池，这个池中可以容纳一定数量的连接对象，一开始我们可以先替用户先创建好一些连接对象，等用户要拿连接对象时，就直接从池中拿，不用新建了，这样也可以节省时间。然后用户用完后，放回去，别人可以接着用。
- 可以提高连接的使用率。当池中的现有的连接都用完了，那么连接池可以向服务器申请新的连接放到池中。
- 直到池中的连接达到“最大连接数”，就不能在申请新的连接了，如果没有拿到连接的用户只能等待。

5.3 市面常见连接池产品和对比

- JDBC 的数据库连接池使用 javax.sql.DataSource 接口进行规范，所有的第三方连接池都实现此接口，自行添加具体实现。也就是说，所有连接池获取连接的和回收连接方法都一样，不同的只有性能和扩展功能。
    - DBCP 是 Apache 提供的数据库连接池，速度相对 c3p0 较快，但因自身存在 BUG。
    - C3P0 是一个开源组织提供的一个数据库连接池，速度相对较慢，稳定性还可以。
    - Proxool 是 sourceforge 下的一个开源项目数据库连接池，有监控连接池状态的功能，稳定性较 c3p0 差一点。
    - Druid 是阿里提供的数据库连接池，据说是集 DBCP 、C3P0 、Proxool 优点于一身
      的数据库连接池。

mock性能数据 (单位:ms)

	5	20	50	100
tomcat-jdbc	442	447	1,013	1,264
c3p0	4,480	5,527	7,449	10,725
dbcp	676	689	867	1,292
hikari	38	33	38	30
druid	291	293	562	985

功能	dbcp	druid	c3p0	tomcat-jdbc	HikariCP
是否支持PSCache	是	是	是	否	否
监控	jmx	jmx/log/http	jmx,log	jmx	jmx
扩展性	弱	好	弱	弱	弱
sql拦截及解析	无	支持	无	无	无
代码	简单	中等	复杂	简单	简单
更新时间	2015.8.6	2015.10.10	2015.12.09		2015.12.3
特点	依赖于common-pool	阿里开源，功能全面	历史久远，代码逻辑复杂，且不易维护		优化力度大，功 boneCP
连接池管理	LinkedBlockingDeque	数组		FairBlockingQueue	threadlocal+Co

5.4 国货之光 Druid 连接池使用

> 记得导入 Druid 工具类 jar

- 硬编码方式(不推荐)

```java
package com.myxh.api.druid;

import com.alibaba.druid.pool.DruidDataSource;
import com.alibaba.druid.pool.DruidDataSourceFactory;
import org.junit.Test;

import javax.sql.DataSource;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Properties;

/**
 * @author MYXH
 * @date 2023/5/14
 * Description: druid 连接池使用类
 */
public class DruidUsePart
{
    /**
     * 直接使用代码设置连接池连接参数方式
     * 1. 创建一个 druid 连接池对象
     * 2. 设置连接池参数(必须或非必须)
     * 3. 获取连接(通用方法, 所有连接池都一样)
     * 4. 回收连接(通用方法,所有连接池都一)
     */
    @Test
    public void testHardCoding() throws SQLException
    {
        // 连接池对象
        DruidDataSource druidDataSource = new DruidDataSource();

        // 设置参数
        // 必须参数: 连接数据库驱动类的全限定符, 注册驱动(url 或 user 或 password)
        druidDataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");    // 帮助我们进
        druidDataSource.setUrl("jdbc:mysql://localhost:3306/my_jdbc");
        druidDataSource.setUsername("MYXH");
        druidDataSource.setPassword("520.ILY!");

        // 非必须参数: 初始化连接数量, 最大的连接数量...
        druidDataSource.setInitialSize(5);    // 初始化的连接数量
        druidDataSource.setMaxActive(10);    // 最大的连接数量

        // 获取连接
```

```
        Connection connection = druidDataSource.getConnection();

        // 数据库 CURD

        // 回收连接
        connection.close();     // 连接池提供的连接，close()就是回收连接
    }
```

- 软编码方式
    - 外部配置

      存放在 src/druid.properties

      ```
      # key = value => java Properties 读取 (key 和 value)
      # druid 配置 的key 固定命名
      # druid 连接池需要的配置参数，key固定命名
      driverClassName=com.mysql.cj.jdbc.Driver
      url=jdbc:mysql:///my_jdbc
      username=MYXH
      password=520.ILY!
      initialSize=5
      maxActive=10
      ```

    - Druid 声明代码

```
    /**
    * 通过读取外部配置文件的方法，实例化 druid 连接池对象
    */
    @Test
    public void testSoftCoding() throws Exception
    {
        // 1. 读取外部配置的文件 Properties
        Properties properties = new Properties();

        // src 下的文件，可以使用类加载器提供的方法
        InputStream inputStream = DruidUsePart.class.getClassLoader().getRe

        properties.load(inputStream);

        // 2. 使用连接池的工具类的工场模式，创建连接池
        DataSource druidDataSource = DruidDataSourceFactory.createDataSourc

        Connection connection = druidDataSource.getConnection();

        // 数据库 CURD

        connection.close();
    }
}
```

- Druid 配置

配置	缺省	说明
name		配置这个属性的意义在于，如果存在多个数据源 如果没有配置，将会生成一个名字，格式是：" System.identityHashCode(this)
jdbcUrl		连接数据库的 url，不同数据库不一样。例如： jdbc:mysql://10.20.153.104:3306/druid2 ora jdbc:oracle:thin:@10.20.149.85:1521:ocnaut
username		连接数据库的用户名
password		连接数据库的密码。如果你不希望密码直接写在 ConfigFilter。详细看这里：https://github.co wiki/%E4%BD%BF%E7%94%A8ConfigFilter

配置	缺省	说明
driverClassName		根据 url 自动识别 这一项可配可不配，如果不配 dbType，然后选择相应的 driverClassName(
initialSize	0	初始化时建立物理连接的个数。初始化发生在显 getConnection 时
maxActive	8	最大连接池数量
maxIdle	8	已经不再使用，配置了也没效果
minIdle		最小连接池数量
maxWait		获取连接时最大等待时间，单位毫秒。配置了 之后，缺省启用公平锁，并发效率会有所下降， 属性为 true 使用非公平锁。
poolPreparedStatements	false	是否缓存 preparedStatement，也就是 PSCa 对支持游标的数据库性能提升巨大，比如说 or
maxOpenPreparedStatements	-1	要启用 PSCache，必须配置大于 0，当大于 0 自动触发修改为 true。在 Druid 中，不会存在 占用内存过多的问题，可以把这个数值配置大一
validationQuery		用来检测连接是否有效的 sql，要求是一个查询 null，testOnBorrow、testOnReturn、testW
testOnBorrow	true	申请连接时执行 validationQuery 检测连接是否
testOnReturn	false	归还连接时执行 validationQuery 检测连接是否
testWhileIdle	false	建议配置为 true，不影响性能，并且保证安全性。申请连接 timeBetweenEvictionRunsMillis，执行 valid
timeBetweenEvictionRunsMillis		有两个含义： 1)Destroy 线程会检测连接的间隔 的判断依据，详细看 testWhileIdle 属性的说明
numTestsPerEvictionRun		不再使用，一个 DruidDataSource 只支持一个
minEvictableIdleTimeMillis		

配置	缺省	说明
connectionInitSqls		物理连接初始化的时候执行的 sql
exceptionSorter		根据 dbType 自动识别 当数据库抛出一些不可
filters		属性类型是字符串，通过别名的方式配置扩展 filter:stat 日志用的 filter:log4j 防御 sql 注入的
proxyFilters		类型是 List，如果同时配置了 filters 和 proxyF

第 6 章 全新 JDBC 使用优化以及工具类封装

6.1 JDBC 工具类封装 Version1.0

> 我们封装一个工具类，内部包含连接池对象，同时对外提供连接的方法和回收连接的方法。

- 外部配置文件
  位置: src/druid.properties

```
# key = value => java Properties 读取 (key 和 value)
# druid 配置 的key 固定命名
# druid 连接池需要的配置参数，key固定命名
driverClassName=com.mysql.cj.jdbc.Driver
url=jdbc:mysql:///my_jdbc
username=MYXH
password=520.ILY!
initialSize=5
maxActive=10
```

- Version1.0 版本工具类

```java
package com.myxh.api.utils;

import com.alibaba.druid.pool.DruidDataSourceFactory;

import javax.sql.DataSource;
import java.io.IOException;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Properties;

/**
 * @author MYXH
 * @date 2023/5/15
 * Description:
 *     V1.0 版本工具类:
 *         内部包含一个连接池对象，并且对外提供获取连接和回收连接的方法
 *     建议:
 *         工具类的方法，推荐写成静态，外部调用会更加方便
 *     实现:
 *         属性: 连接池对象，实例化一次
 *             单例模式
 *             static
 *             {
 *                 全局调用一次
 *             }
 *         方法:
 *             对外提供连接的方法
 *             回收外部传入连接方法
 */
public class JdbcUtilsVersion1
{
    private static final DataSource dataSource;    // 连接池对象

    static
    {
        // 初始化连接池对象
        Properties properties = new Properties();
        InputStream inputStream = JdbcUtilsVersion1.class.getClassLoader().getResourceA
        try
        {
            properties.load(inputStream);
        }
        catch (IOException e)
```

```
        {
            throw new RuntimeException(e);
        }

        try
        {
            dataSource = DruidDataSourceFactory.createDataSource(properties);
        }
        catch (Exception e)
        {
            throw new RuntimeException(e);
        }
    }

    /**
     * 对外提供连接的方法
     */
    public static Connection getConnection() throws SQLException
    {
        return dataSource.getConnection();
    }

    public static void freeConnection(Connection connection) throws SQLException
    {
        connection.close();     // 连接池的连接，调用 close()就是回收
    }
}
```

6.2 JDBC 工具类封装 Version2.0

> 优化工具类 Version1.0 版本，考虑事务的情况下，如何一个线程的不同方法获取同一个连接。

- ThreadLocal 的介绍
    - JDK 1.2 的版本中就提供 java.lang.ThreadLocal，为解决多线程程序的并发问题提供了一种新的思路。
    - 使用这个工具类可以很简洁地编写出优美的多线程程序。通常用来在在多线程中管理共享数据库连接、Session 等。
    - ThreadLocal 用于保存某个线程共享变量，原因是在 Java 中，每一个线程对象中都有一个。

    - ThreadLocalMap<ThreadLocal, Object>，其 key 就是一个 ThreadLocal，而 Object 即为该线程的共享变量。而这个 map 是通过 ThreadLocal 的 set 和 get 方法操作的。对于同一个 static ThreadLocal，不同线程只能从中 get，set，remove 自己的变量，而不会影响其他线程的变量。
    - ThreadLocal 对象.get: 获取 ThreadLocal 中当前线程共享变量的值。
    - ThreadLocal 对象.set: 设置 ThreadLocal 中当前线程共享变量的值。
    - ThreadLocal 对象.remove: 移除 ThreadLocal 中当前线程共享变量的值。
- Version2.0 版本工具类

```java
package com.myxh.api.utils;

import com.alibaba.druid.pool.DruidDataSourceFactory;

import javax.sql.DataSource;
import java.io.IOException;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Properties;

/**
 * @author MYXH
 * @date 2023/5/15
 * Description:
 *     V2.0 版本工具类:
 *         内部包含一个连接池对象，并且对外提供获取连接和回收连接的方法
 *     建议:
 *         工具类的方法，推荐写成静态，外部调用会更加方便
 *     实现:
 *         属性: 连接池对象，实例化一次
 *             单例模式
 *             static
 *             {
 *                 全局调用一次
 *             }
 *         方法:
 *             对外提供连接的方法
 *             回收外部传入连接方法
 * TODO:
 *     利用线程本地变量，存储连接信息，确保一个线程的多个方法可以获取阿一个 Connection
 *     优势: 事务操作的时候 Service和 DAO 属于同一个线程，不用再传递参数
 *     都可以调用 getConnection()自动获取相同的连接池
 */
public class JdbcUtilsVersion2
{
    private static final DataSource dataSource;    // 连接池对象

    private static final ThreadLocal<Connection> threadLocal
            = new ThreadLocal<>();

    static
    {
        // 初始化连接池对象
```

```
        Properties properties = new Properties();
        InputStream inputStream = JdbcUtilsVersion2.class.getClassLoader().getResourceA
        try
        {
            properties.load(inputStream);
        }
        catch (IOException e)
        {
            throw new RuntimeException(e);
        }

        try
        {
            dataSource = DruidDataSourceFactory.createDataSource(properties);
        }
        catch (Exception e)
        {
            throw new RuntimeException(e);
        }
    }

    /**
     * 对外提供连接的方法
     */
    public static Connection getConnection() throws SQLException
    {
        // 线程本地变量中是否存在
        Connection connection = threadLocal.get();

        // 第一次没有
        if (connection == null)
        {
            // 线程本地变量没有，连接池获取
            connection = dataSource.getConnection();
            threadLocal.set(connection);
        }

        return connection;
    }

    public static void freeConnection() throws SQLException
    {
        Connection connection = threadLocal.get();
```

```
        if (connection != null)
        {
            threadLocal.remove();     // 清空线程本地变量数据
            connection.setAutoCommit(true);     // 事务状态回归 false
            connection.close();     // 回收到连接池
        }
    }
}
```

- 基于此工具类的 CURD

```
package com.myxh.api.utils;

import org.junit.Test;

import java.sql.Connection;
import java.sql.SQLException;

/**
* @author MYXH
* @date 2023/5/15
* Description: 基于工具类的 CURD
*/
public class JdbcCrudPart
{
    @Test
    public void testInsert() throws SQLException
    {
        Connection connection = JdbcUtilsVersion2.getConnection();

        // 数据库 CURD 动作

        JdbcUtilsVersion2.freeConnection();
    }
}
```

- 修改转账业务，使用此工具类
    - 测试方法

```
@Test
public void testStart() throws SQLException
{
    // 经理给管理员转 500 元
    transFer("管理员", "经理", BigDecimal.valueOf(500));
}
```

- BankService 类

```java
package com.myxh.api.transactionbyutils;

import com.myxh.api.utils.JdbcUtilsVersion2;
import org.junit.Test;

import java.math.BigDecimal;
import java.sql.Connection;
import java.sql.SQLException;

/**
 * @author MYXH
 * @date 2023/5/15
 * Description: bank 表业务类，添加转账业务
 *              银行卡业务方法，调用 BankDAO 的方法
 */
public class BankService
{
    @Test
    public void testStart() throws SQLException
    {
        // 经理给管理员转 500 元
        transFer("管理员", "经理", BigDecimal.valueOf(500));
    }

    /**
     * TODO:
     *     事务添加是在业务方法中
     *     利用 try catch 代码块，开启事务，提交事务和事务回滚
     *     将 Connection 传入 BankDao 层即可，BankDao 只负责使用，不负责 close
     */
    public void transFer(String addAccount, String subAccount, BigDecimal m
    {
        BankDao bankDao = new BankDao();

        Connection connection = JdbcUtilsVersion2.getConnection();

        try
        {
            // 开启事务
            // 关闭事务提交
            connection.setAutoCommit(false);

            // 执行数据库动作
            bankDao.add(addAccount, money);
```

```
            System.out.println("----------");
            bankDao.sub(subAccount, money);

            // 事务提交
            connection.commit();
        }
        catch (Exception e)
        {
            // 事务回滚
            connection.rollback();

            // 抛出异常信息
            throw e;
        }
        finally
        {
            JdbcUtilsVersion2.freeConnection();
        }
    }
}
```

- BankDao 类

```java
package com.myxh.api.transactionbyutils;

import com.myxh.api.utils.JdbcUtilsVersion2;

import java.math.BigDecimal;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

/**
 * @author MYXH
 * @date 2023/5/15
 * Description: bank 表的数据库操作方法存储类
 */
public class BankDao
{
    /**
     * 存钱的数据库操作方法(jdbc)
     * @param account 存钱的账号
     * @param money 存钱的金额
     */
    public void add(String account, BigDecimal money) throws SQLException
    {
        Connection connection = JdbcUtilsVersion2.getConnection();

        // 3. 编写 SQL 语句结构
        String sql = "update t_bank set money = money + ? where account = ?

        // 4. 创建 PreparedStatement
        PreparedStatement preparedStatement = connection.prepareStatement(s

        // 5. 占位符赋值
        preparedStatement.setObject(1, money);
        preparedStatement.setObject(2, account);

        // 6. 发送 SQL 语句
        preparedStatement.executeUpdate();

        // 7. 关闭资源
        preparedStatement.close();

        System.out.println("存钱成功! ");
    }
```

```
    /**
     * 取钱的数据库操作方法(jdbc)
     * @param account 取钱的账号
     * @param money 取钱的金额
     */
    public void sub(String account, BigDecimal money) throws SQLException
    {
        Connection connection = JdbcUtilsVersion2.getConnection();

        // 3. 编写 SQL 语句结构
        String sql = "update t_bank set money = money - ? where account = ?

        // 4. 创建 PreparedStatement
        PreparedStatement preparedStatement = connection.prepareStatement(s

        // 5. 占位符赋值
        preparedStatement.setObject(1, money);
        preparedStatement.setObject(2, account);

        // 6. 发送 SQL 语句
        preparedStatement.executeUpdate();

        // 7. 关闭资源
        preparedStatement.close();

        System.out.println("取钱成功! ");
    }
}
```

6.3 高级应用层封装 BaseDao

> 基本上每一个数据表都应该有一个对应的 DAO 接口及其实现类，发现对所有表的操作（增、删、改、查）代码重复度很高，所以可以抽取公共代码，给这些 DAO 的实现类可以抽取一个公共的父类，我们称为 BaseDao。

- BaseDao 类

```java
package com.myxh.api.utils;

import java.lang.reflect.Field;
import java.lang.reflect.InvocationTargetException;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

/**
 * @author MYXH
 * @date 2023/5/15
 * Description: 封装 DAO 数据库重复代码
 * TODO:
 *     封装两个方法，一个简化非 DQL
 *                 一个简化 DQL
 */
public abstract class BaseDao
{
    /**
     * 封装简化非 DQL 语句
     *
     * @param sql    带占位符的 SQL 语句
     * @param params 占位符的值    注意：传入占位符的值，必等于 SQL 语句 ? 位置
     * @return 执行影响的行数
     */
    public int executeUpdate(String sql, Object... params) throws SQLException
    {
        // 获取连接
        Connection connection = JdbcUtilsVersion2.getConnection();

        // 创建 PreparedStatement，并且传入 SQL 语句结果
        PreparedStatement preparedStatement = connection.prepareStatement(sql);

        // 占位符赋值
        // 可变参数可以当做数组使用
        for (int i = 1; i <= params.length; i++)
        {
            preparedStatement.setObject(i, params[i - 1]);
        }

        // 发送 SQL 语句
        // DML 类型
        int rows = preparedStatement.executeUpdate();
```

```
        preparedStatement.close();

        // 是否回收连接，需要考虑是不是事务
        // 开启事务后，不要管连接，业务层处理，connection.setAutoCommit(false);
        if (connection.getAutoCommit())
        {
            // 没有开启事务，正常回收连接
            JdbcUtilsVersion2.freeConnection();
        }

        return rows;
    }

    /*
    非 DQL 语句封装方法 -> 返回值，固定为 int0

    DQL 语句封装方法 -> 返回值是什么类型?    List<T>

                        并不是 List<Map> Map 中的 key 和 value自定义，不用先设定好
                                        Map 没有数据校验机制
                                        Map 不支持反射操作

                        数据库数据 -> java的实体类

                        table
                            t_user
                                id
                                account
                                password
                                nickname

                        Java
                            User
                                id
                                account
                                password
                                nickname

                        表中一行数据 -> Java类的一个对象 -> 多行 -> List<Java实体类>list;

    <T>声明一个泛型，不确定类型
        1. 确定泛型 User.class T = User
        2. 要使用反射技术属性赋值
```

```
    public <T> List<T> executeQuery(class<T> clazz,String sql,Object...params);
    */

    /**
     * 将查询结果封装到一个实体类集合
     *
     * @param clazz  要接受返回值的实体类集合的模板对象
     * @param sql    查询语句，要求列名或者别名等于实体类的属性名 u_id as uId -> uId
     * @param params 占位符的值,要和 ? 位置对象传递
     * @param <T>    声明的结果的类型
     * @return 查询的实体类集合
     */
    public <T> List<T> executeQuery(Class<T> clazz, String sql, Object... params) throw
    {
        // 获取连接
        Connection connection = JdbcUtilsVersion2.getConnection();

        // 创建 PreparedStatement，并且传入 SQL 语句结果
        PreparedStatement preparedStatement = connection.prepareStatement(sql);

        //  占位符赋值
        if (params != null && params.length != 0)
        {
            for (int i = 1; i <= params.length; i++)
            {
                preparedStatement.setObject(i, params[i - 1]);
            }
        }

        // 发送 SQL 语句
        ResultSet resultSet = preparedStatement.executeQuery();

        // 结果集解析
        List<T> list = new ArrayList<>();

        // 获取列的信息对象
        ResultSetMetaData metaData = resultSet.getMetaData();

        // 列名数组，用于存储查询结果集中所有列名
        String[] columnNames = new String[metaData.getColumnCount()];

        // 获取列名或别名，并将其转换成对应的实体类属性名
        for (int i = 1; i <= columnNames.length; i++)
        {
```

```
            String columnName = metaData.getColumnLabel(i);
            Field field;

            try
            {
                field = clazz.getDeclaredField(columnName);
            }
            catch (NoSuchFieldException e)
            {
                // 如果找不到同名属性，则尝试将列名转换成驼峰命名法格式的属性名
                String propertyName = StringUtils.toCamelCase(columnName);
                field = clazz.getDeclaredField(propertyName);
            }

            columnNames[i - 1] = field.getName();
        }

        while (resultSet.next())
        {
            // 一行数据对应一个 T 类型的对象
            T t = clazz.getDeclaredConstructor().newInstance();     // 调用类的无参构造函

            // 自动遍历列：注意，要从 1 开始，并且小于等于总列数
            for (int i = 1; i <= columnNames.length; i++)
            {
                // 对象的属性值
                Object value = resultSet.getObject(i);

                // 获取属性名称
                String propertyName = metaData.getColumnLabel(i);

                // 反射给对象的属性值赋值
                Field field;

                try
                {
                    field = clazz.getDeclaredField(propertyName);
                }
                catch (NoSuchFieldException e)
                {
                    // 如果找不到同名属性，则尝试将属性名转换成下划线分隔符格式的列名
                    String columnName = StringUtils.toUnderscoreCase(propertyName);
                    field = clazz.getDeclaredField(columnName);
                }
```

```
                field.setAccessible(true);    // 属性可以设置，取消 private 的修饰限制

                /*
                参数1：要赋值的对象，如果属性是静态，第一个参数可以为null
                参数2：具体的属性值
                */
                field.set(t, value);
            }

            // 一行数据的所有列全部存到了 T 中
            //将 T 存储到集合中即可
            list.add(t);
        }

        // 关闭资源
        resultSet.close();
        preparedStatement.close();

        if (connection.getAutoCommit())
        {
            // 没有事务，可以关闭
            JdbcUtilsVersion2.freeConnection();
        }

        return list;
    }

    public static class StringUtils
    {
        /**
         * 将字符串转换成驼峰命名法格式的字符串（首字母小写）
         *
         * @param str 原始字符串
         * @return 转换后的字符串
         */
        public static String toCamelCase(String str)
        {
            StringBuilder sb = new StringBuilder();
            boolean isUpperCase = false;

            for (int i = 0; i < str.length(); i++)
            {
                char c = str.charAt(i);
```

```
            if (c == '_')
            {
                isUpperCase = true;
            }
            else if (isUpperCase)
            {
                sb.append(Character.toUpperCase(c));
                isUpperCase = false;
            }
            else if (i == 0)
            {
                sb.append(Character.toLowerCase(c));
            }
            else
            {
                sb.append(c);
            }
        }

        return sb.toString();
    }

    /**
     * 将字符串转换成下划线分隔符格式的字符串
     *
     * @param str 原始字符串
     * @return 转换后的字符串
     */
    public static String toUnderscoreCase(String str)
    {
        StringBuilder sb = new StringBuilder();

        for (int i = 0; i < str.length(); i++)
        {
            char c = str.charAt(i);

            if (Character.isUpperCase(c))
            {
                sb.append('_').append(Character.toLowerCase(c));
            }
            else
            {
                sb.append(c);
```

```
                }
            }

            return sb.toString();
        }
    }
}
```

- User 类

```java
package com.myxh.api.entity;

/**
 * @author MYXH
 * @date 2023/5/15
 */
public class User
{
    private int id;
    private String account;
    private String password;
    private String nickname;

    public User()
    {

    }

    public User(int id, String account, String password, String nickname)
    {
        this.id = id;
        this.account = account;
        this.password = password;
        this.nickname = nickname;
    }

    public int getId()
    {
        return id;
    }

    public void setId(int id)
    {
        this.id = id;
    }

    public String getAccount()
    {
        return account;
    }

    public void setAccount(String account)
    {
        this.account = account;
```

```
    }

    public String getPassword()
    {
        return password;
    }

    public void setPassword(String password)
    {
        this.password = password;
    }

    public String getNickname()
    {
        return nickname;
    }

    public void setNickname(String nickname)
    {
        this.nickname = nickname;
    }

    @Override
    public String toString()
    {
        return "User{" +
                "id=" + id +
                ", account='" + account + '\'' +
                ", password='" + password + '\'' +
                ", nickname='" + nickname + '\'' +
                '}';
    }
}
```

- PreparedStatementCurdPart 类

```java
package com.myxh.api.utils;

import com.myxh.api.entity.User;
import org.junit.Test;

import java.lang.reflect.InvocationTargetException;
import java.sql.SQLException;
import java.util.List;


/**
 * @author MYXH
 * @date 2023/5/15
 * @Description 使用 PreparedStatement 进行 t_user 表 的 CURD 操作
 */
public class PreparedStatementCurdPart extends BaseDao
{
    // 测试方法需要导入 junit 的测试包
    @Test
    public void testInsert() throws SQLException
    {
        /*
        t_user 插入一条用户数据
            account: DaoTest1
            password: test1
            nickname: Dao测试员1
        */

        // 编写 SQL 语句结果，动态值的部分使用 ? 代替
        String sql = "insert into t_user(account, password, nickname) values(?, ?, ?);"

        int rows = executeUpdate(sql, "DaoTest1", "test1", "Dao测试员1");
        System.out.println("rows = " + rows);
    }

    @Test
    public void testUpdate() throws SQLException
    {
        /*
        修改 id = 3 的用户 nickname = 新的测试员
        */

        // 编写 SQL 语句结果，动态值的部分使用 ? 代替
        String sql = "update t_user set nickname = ? where id = ?;";
```

```
        int rows = executeUpdate(sql, "新的测试员", 3);
        System.out.println("rows = " + rows);
    }

    @Test
    public void testDelete() throws SQLException
    {
        /*
        删除 id = 20004 的用户数据
        */

        // 编写 SQL 语句结果，动态值的部分使用 ? 代替
        String sql = "delete from t_user where id = ?;";

        int rows = executeUpdate(sql, 20004);
        System.out.println("rows = " + rows);
    }

    @Test
    public void testSelect() throws SQLException, NoSuchFieldException, InvocationTarge
    {
        /*
        目标：查询所有用户数据
        */

        // 编写 SQL 语句结果，动态值的部分使用 ? 代替
        String sql = "select id, account, password, nickname from t_user;";

        List<User> list = executeQuery(User.class, sql);
        System.out.println("list = " + list);
    }
}
```

- Bank 类

```java
package com.myxh.api.entity;

import java.math.BigDecimal;

/**
 * @author MYXH
 * @date 2023/5/15
 */
public class Bank
{
    private int id;
    private String account;
    private BigDecimal money;

    public Bank()
    {

    }

    public Bank(String account, BigDecimal money)
    {
        this.account = account;
        this.money = money;
    }

    public Bank(int id, String account, BigDecimal money)
    {
        this.id = id;
        this.account = account;
        this.money = money;
    }

    public int getId()
    {
        return id;
    }

    public void setId(int id)
    {
        this.id = id;
    }

    public String getAccount()
    {
```

```
        return account;
    }

    public void setAccount(String account)
    {
        this.account = account;
    }

    public BigDecimal getMoney()
    {
        return money;
    }

    public void setMoney(BigDecimal money)
    {
        this.money = money;
    }

    @Override
    public String toString()
    {
        return "Bank{" +
                "id=" + id +
                ", account='" + account + '\'' +
                ", money=" + money +
                '}';
    }
}
```

第 7 章 基于 CustomerManageSystem 项目 JDBC 实战练习

7.1 CustomerManageSystem 项目介绍和导入

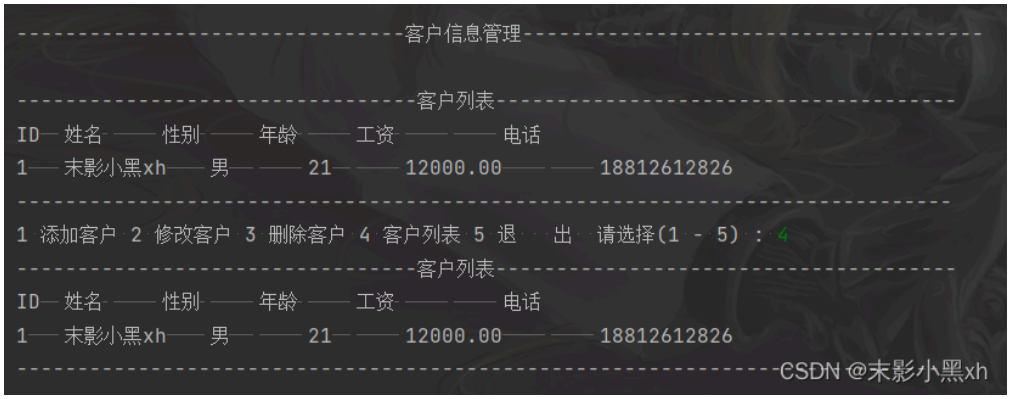
- 项目介绍

  > 利用 JavaSE 技术，进行控制台输出的客户管理系统。主要功能让包含客户展示，客户删除，客户添加，客户修改，退出系统。

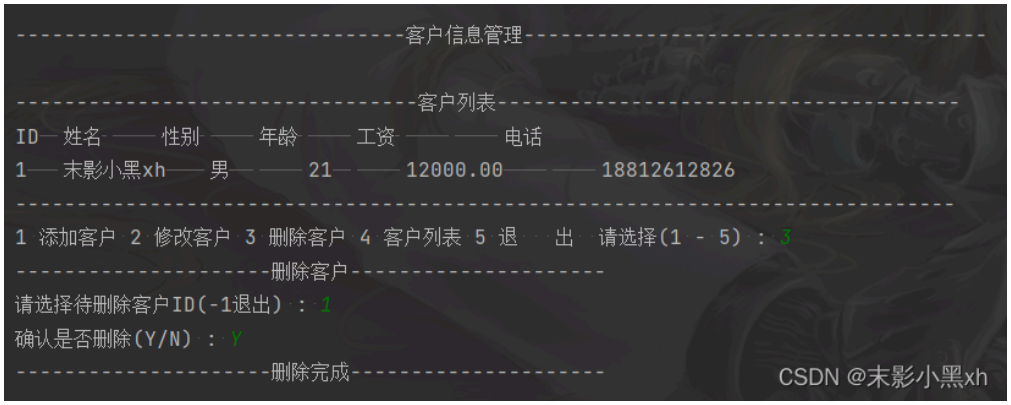
- 添加客户

- 修改客户

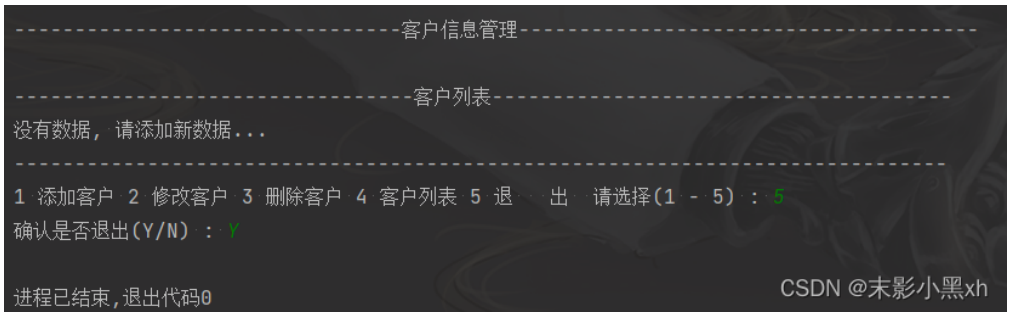
- 展示客户列表



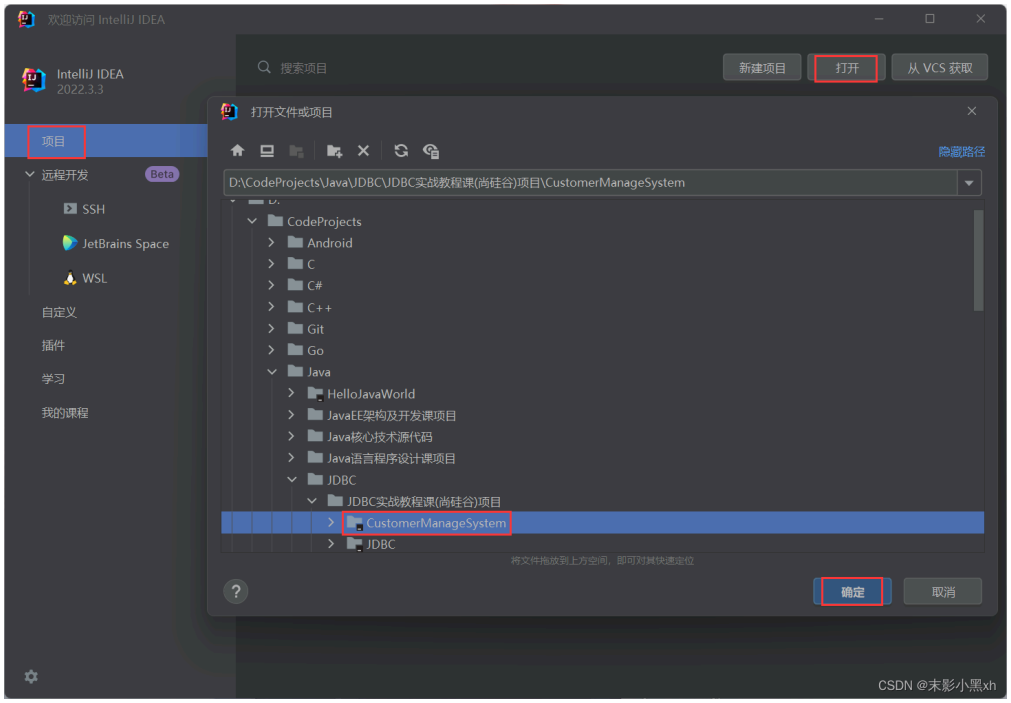
- 删除客户



- 退出系统



- 项目导入
  - 打开项目



  - 配置 JDK

7.2 基于 CustomerManageSystem 项目添加数据库相关配置

- 准备数据库脚本

```sql
CREATE
    DATABASE IF NOT EXISTS my_jdbc_customer_manage_system;

USE
    my_jdbc_customer_manage_system;

-- 员工表

CREATE TABLE t_customer
(
    id     INT PRIMARY KEY AUTO_INCREMENT COMMENT '客户主键',
    `name`   VARCHAR(20) COMMENT '客户名称',
    gender VARCHAR(4) COMMENT '客户性别',
    age    INT COMMENT '客户年龄',
    salary DECIMAL(8, 2) COMMENT '客户工资',
    phone  VARCHAR(11) COMMENT '客户电话'
)
```

- 添加配置文件

> 位置: src 下，druid.properties

```properties
# key = value -> Java Properties 读取 (key 或 value)
# druid 配置的 key 固定命名
# druid 连接池需要的配置参数, key 固定命名
driverClassName=com.mysql.cj.jdbc.Driver
url=jdbc:mysql:///my_jdbc_customer_manage_system
username=MYXH
password=520.ILY!
initialSize=5
maxActive=10
```

- 导入 JdbcUtils 类

```java
package com.myxh.cms.utils;

import com.alibaba.druid.pool.DruidDataSourceFactory;

import javax.sql.DataSource;
import java.io.IOException;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Properties;

/**
 * @author MYXH
 * @date 2023/5/16
 * Description:
 * 工具类
 *     内部包含一个连接池对象，并且对外提供获取连接和回收连接的方法
 * 建议:
 *   工具类的方法，推荐写成静态，外部调用会更加方便
 * 实现:
 *   属性: 连接池对象，实例化一次
 *        单例模式
 *        static
 *        {
 *            全局调用一次
 *        }
 *   方法:
 *       对外提供连接的方法
 *       回收外部传入连接方法
 * TODO:
 *    利用线程本地变量，存储连接信息，确保一个线程的多个方法可以获取同一个 Connection
 *    优势: 事务操作的时候 Service 和 DAO 属于同一个线程，不用再传递参数
 *    都可以调用 getConnection() 自动获取的是相同的连接池
 */
public class JdbcUtils
{
    private static final DataSource dataSource;    // 连接池对象

    private static final ThreadLocal<Connection> threadLocal = new ThreadLocal<>();

    static
    {
        // 初始化连接池对象
        Properties properties = new Properties();
```

```
        InputStream inputStream = JdbcUtils.class.getClassLoader().getResourceAsStream
        try
        {
            properties.load(inputStream);
        }
        catch (IOException e)
        {
            throw new RuntimeException(e);
        }

        try
        {
            dataSource = DruidDataSourceFactory.createDataSource(properties);
        }
        catch (Exception e)
        {
            throw new RuntimeException(e);
        }
    }

    /**
     * 对外提供连接的方法
     */
    public static Connection getConnection() throws SQLException
    {
        // 线程本地变量中是否存在
        Connection connection = threadLocal.get();

        // 第一次没有线程本地变量
        if (connection == null)
        {
            // 线程本地变量没有,连接池获取
            connection = dataSource.getConnection();
            threadLocal.set(connection);
        }

        return  connection;
    }

    public static void freeConnection() throws SQLException
    {
        Connection connection = threadLocal.get();
```

```
        if (connection != null)
        {
            threadLocal.remove();     // 清空线程本地变量数据
            connection.setAutoCommit(true);     // 事务状态回归 false
            connection.close();     // 回收到连接池即可
        }
    }
}
```

- 导入 BaseDao 工具类

```java
package com.myxh.cms.utils;

import java.lang.reflect.Field;
import java.lang.reflect.InvocationTargetException;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

/**
 * @author MYXH
 * @date 2023/5/16
 * @Description: 封装 DAO 数据库重复代码
 * TODO:
 *    封装两个方法：一个简化非 DQL
 *                一个简化 DQL
 */
public abstract class BaseDao
{
    /**
     * 封装简化非 DQL 语句
     * @param sql    带占位符的 SQL 语句
     * @param params 占位符的值，注意：传入占位符的值，必须等于 SQL 语句 ? 位置
     * @return 执行影响的行数
     */
    public int executeUpdate(String sql,Object... params) throws SQLException
    {
        // 获取连接
        Connection connection = JdbcUtils.getConnection();

        PreparedStatement preparedStatement = connection.prepareStatement(sql);

        // 占位符赋值
        // 可变参数可以当做数组使用
        for (int i = 1; i <= params.length; i++)
        {
            preparedStatement.setObject(i, params[i - 1]);
        }

        // 发送 SQL 语句
        // DML 类型
        int rows = preparedStatement.executeUpdate();

        preparedStatement.close();
```

```
        // 是否回收连接，需要考虑是不是事务
        if (connection.getAutoCommit())
        {
            // 没有开启事务
            // 没有开启事务，正常回收连接
            JdbcUtils.freeConnection();
        }

        // connection.setAutoCommit(false); //开启事务后，不要管连接，业务层处理

        return rows;
    }


    /*
    非 DQL 语句封装方法 -> 返回值 固定为int

    DQL 语句封装方法 -> 返回值 是什么类型? List<T>
                        并不是list<Map> Map key 和 value自定义，不用先设定好
                                    Map 没有数据校验机制
                                    Map 不支持反射操作

                        数据库数据 -> Java的实体类

                        table
                            t_user
                                id
                                account
                                password
                                nickname

                        java
                            User
                                id
                                account
                                password
                                nickname

        表中 -> 一行数据 -> Java类的一个对象 -> 多行数据 -> List<Java实体类> list;

        DQL -> List<Map> -> 一行数据 -> Map -> List<Map>

        <T> 声明一个泛型，不确定类型
            1. 确定泛型 User.class T = User
```

```
        2. 要使用反射技术属性赋值
    public <T> List<T> executeQuery(Class<T> clazz,String sql,Object... params);
*/

/**
 * 将查询结果封装到一个实体类集合
 * @param clazz  要接值的实体类集合的模板对象
 * @param sql    查询语句，要求列名或者别名等于实体类的属性名，u_id as uId -> uId
 * @param params 占位符的值，要和 ? 位置对象传递
 * @return       查询的实体类集合
 * @param <T>    声明的结果的类型
 */
public <T> List<T> executeQuery(Class<T> clazz, String sql,Object... params) throws
{
    // 获取连接
    Connection connection = JdbcUtils.getConnection();

    PreparedStatement preparedStatement = connection.prepareStatement(sql);

    // 占位符赋值
    if (params != null && params.length != 0)
    {
        for (int i = 1; i <= params.length; i++)
        {
            preparedStatement.setObject(i,params[i - 1]);
        }
    }

    // 发送 SQL 语句
    ResultSet resultSet = preparedStatement.executeQuery();

    // 结果集解析
    List<T> list = new ArrayList<>();

    // 获取列的信息对象
    // TODO: metaData 装的当前结果集列的信息对象，可以获取列的名称根据下角标，可以获取列
    ResultSetMetaData metaData = resultSet.getMetaData();

    //可以水平遍历列
    int columnCount = metaData.getColumnCount();

    while (resultSet.next())
    {
        T t = clazz.getDeclaredConstructor().newInstance();     // 调用类的无参构造函
```

```
            // 自动遍历列，注意：要从 1 开始，并且小于等于总列数
            for (int i = 1; i <= columnCount; i++)
            {
                // 对象的属性值
                Object value = resultSet.getObject(i);

                // 获取指定列下角标的列的名称，ResultSetMetaData()
                String propertyName = metaData.getColumnLabel(i);

                //反射，给对象的属性值赋值
                Field field = clazz.getDeclaredField(propertyName);
                field.setAccessible(true);    // 属性可以设置，取消 private 的修饰限制

                /*
                参数1：要要赋值的对象，如果属性是静态，第一个参数，可以为null
                参数2：具体的属性值
                */
                field.set(t,value);
            }

            // 一行数据的所有列全部存到了 map 中
            // 将 map 存储到集合中即可
            list.add(t);
        }

        // 关闭资源
        resultSet.close();
        preparedStatement.close();

        if (connection.getAutoCommit())
        {
            // 没有事务，可以关闭
            JdbcUtils.freeConnection();
        }

        return list;
    }
}
```

7.3 基于 CustomerManageSystem 项目实战

- CustomerService 类

```java
package com.myxh.cms.service;

import com.myxh.cms.dao.CustomerDao;
import com.myxh.cms.javabean.Customer;

import java.lang.reflect.InvocationTargetException;
import java.sql.SQLException;
import java.util.List;

/**
 * @author MYXH
 * @date 2023/5/16
 * 这是一个具有管理功能的功能类，内部数据不允许外部随意修改，具有更好的封装性
 */
public class CustomerService
{
    private final CustomerDao customerDao = new CustomerDao();

    /**
     * 用途：  返回所有客户对象
     * 返回：  集合
     */
    public List<Customer> getList()
    {
        try
        {
            return customerDao.findAll();
        }
        catch (SQLException | NoSuchFieldException | InstantiationException | IllegalA
            InvocationTargetException | NoSuchMethodException e)
        {
            throw new RuntimeException(e);
        }
    }

    /**
     * 用途：添加新客户
     * 参数：customer 指定要添加的客户对象
     */
    public void addCustomer(Customer customer)
    {
        try
        {
            customerDao.addCustomer(customer);
```

```
        }
        catch (SQLException e)
        {
            throw new RuntimeException(e);
        }
    }

    /**
     * 修改指定 id 号的客户对象的信息
     * @param id 客户 id
     * @param customer 对象
     * @return 修改成功返回 true, false 表明指定 id 的客户未找到
     */
    public boolean modifyCustomer(int id, Customer customer)
    {
        int rows = 0;

        try
        {
            rows = customerDao.updateById(customer);
        }
        catch (SQLException e)
        {
            throw new RuntimeException(e);
        }

        // 返回修改成功
        return rows != 0;
    }

    /**
     * 用途: 返回指定 id 的客户对象记录
     * 参数:  id 就是要获取的客户的 id 号
     * 返回: 封装了客户信息的 Customer 对象
     */
    public Customer getCustomer(int id)
    {
        try
        {
            return customerDao.findById(id);
        }
        catch (SQLException | NoSuchFieldException | InstantiationException | IllegalA
            NoSuchMethodException | InvocationTargetException e)
        {
```

```
            throw new RuntimeException(e);
        }
    }

    /**
     * 用途：删除指定 id 号的的客户对象记录
     * 参数：  id 要删除的客户的 id 号
     * 返回：删除成功返回 true，false表 示没有找到
     */
    public boolean removeCustomer(int id)
    {
        int rows = 0;

        try
        {
            rows = customerDao.removeById(id);
        }
        catch (SQLException e)
        {
            throw new RuntimeException(e);
        }

        return rows != 0;
    }
}
```

- CustomerDao 类

```java
package com.myxh.cms.dao;

import com.myxh.cms.javabean.Customer;
import com.myxh.cms.utils.BaseDao;

import java.lang.reflect.InvocationTargetException;
import java.sql.SQLException;
import java.util.List;

/**
 * @author MYXH
 * @date 2023/5/16
 * @Description: CustomerDao对应的数据库方法
 */
public class CustomerDao extends BaseDao
{
    /**
     * 查询数据库客户集合
     */
    public List<Customer> findAll() throws SQLException, NoSuchFieldException, Instanti
    {
        List<Customer> customerList = executeQuery(Customer.class, "select * from t_cus

        return customerList;
    }

    /**
     * 添加客户的方法
     */
    public void addCustomer(Customer customer) throws SQLException
    {

        String sql = "insert into t_customer(name, gender, age, salary, phone) values(

        executeUpdate(sql, customer.getName(), customer.getGender(),
                customer.getAge(), customer.getSalary(), customer.getPhone());
    }

    /**
     * 修改对象信息
     * @return 影响行数
     */
    public  int updateById(Customer customer) throws SQLException
    {
```

```
        String sql = "update t_customer set name = ?, gender = ?, age= ?, salary = ?, p

        int rows = executeUpdate(sql, customer.getName(), customer.getGender(), custome
                customer.getSalary(), customer.getPhone(), customer.getId());

        return rows;
    }

    /**
     * 根据id查询客户信息
     */
    public Customer findById(int id) throws SQLException, NoSuchFieldException, Instant
    {
        String sql = "select * from t_customer where id = ?;";

        List<Customer> customerList = executeQuery(Customer.class, sql, id);

        if (customerList != null && customerList.size() > 0)
        {
            return customerList.get(0);
        }

        return null;
    }

    public int removeById(int id) throws SQLException
    {
        String sql = "delete from t_customer where id = ? ;";

        int rows = executeUpdate(sql, id);

        return rows;
    }
}
```

- Customer 类

```java
package com.myxh.cms.javabean;

import java.math.BigDecimal;

/**
 * @author MYXH
 * @date 2023/5/16
 */
public class Customer
{
    private int id;
    private String name;    // 姓名
    private String gender;    // 性别
    private int age;    // 年龄
    private BigDecimal salary;    // 工资
    private String phone; // 电话

    public Customer()
    {

    }

    public int getId()
    {
        return id;
    }

    public void setId(int id)
    {
        this.id = id;
    }

    public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public int getAge()
    {
```

```
        return age;
    }

    public void setAge(int age)
    {
        this.age = age;
    }

    public BigDecimal getSalary()
    {
        return salary;
    }

    public void setSalary(BigDecimal salary)
    {
        this.salary = salary;
    }

    public String getPhone()
    {
        return phone;
    }

    public void setPhone(String phone)
    {
        this.phone = phone;
    }

    public String getGender()
    {
        return gender;
    }

    public void setGender(String gender)
    {
        this.gender = gender;
    }

    @Override
    public String toString()
    {
        return id + "\t" + name + (name.length() < 3 ? "\t\t" : "\t") + gender + "\t\t
                salary + "\t\t" + phone;
    }
```

```
}
```

- CustomerManage 类

```
package com.myxh.cms.main;

import com.myxh.cms.view.CustomerView;

/**
 * @author MYXH
 * @date 2023/5/16
 */
public class CustomerManage
{
    public static void main(String[] args)
    {
        CustomerView view = new CustomerView();
        view.enterMainMenu();
    }
}
```

- KeyboardUtility 类

```java
package com.myxh.cms.view;

import java.math.BigDecimal;
import java.util.Scanner;

/**
 * @author MYXH
 * @date 2023/5/16
 */
public class KeyboardUtility
{
    private static final Scanner scanner = new Scanner(System.in);

    public static int readInt()
    {
        int n;

        while (true)
        {
            String str = readKeyBoard(8, false);

            try
            {
                n = Integer.parseInt(str);
                break;
            }
            catch (NumberFormatException e)
            {
                System.out.print("数字输入错误，请重新输入：");
            }
        }

        return n;
    }

    public static int readInt(int defaultValue)
    {
        int n;

        while (true)
        {
            String str = readKeyBoard(8, true);

            if (str.equals(""))
```

```
            {
                return defaultValue;
            }

            try
            {
                n = Integer.parseInt(str);
                break;
            }
            catch (NumberFormatException e)
            {
                System.out.print("数字输入错误，请重新输入：");
            }
        }

        return n;
    }

    public static BigDecimal readBigDecimal()
    {
        BigDecimal bigDecimal;

        while (true)
        {
            String str = readKeyBoard(8, false);

            try
            {
                bigDecimal = new BigDecimal(str);
                break;
            }
            catch (NumberFormatException e)
            {
                System.out.print("数字输入错误，请重新输入：");
            }
        }

        return bigDecimal;
    }

    public static BigDecimal readBigDecimal(BigDecimal defaultValue)
    {
        BigDecimal bigDecimal;
```

```
        while (true)
        {
            String str = readKeyBoard(8, true);

            if (str.equals(""))
            {
                return defaultValue;
            }

            try
            {
                bigDecimal = new BigDecimal(str);
                break;
            }
            catch (NumberFormatException e)
            {
                System.out.print("数字输入错误，请重新输入：");
            }
        }

        return bigDecimal;
    }

    public static char readChar()
    {
        String str = readKeyBoard(1, false);

        return str.charAt(0);
    }

    public static char readChar(char defaultValue)
    {
        String str = readKeyBoard(1, true);

        return (str.length() == 0) ? defaultValue : str.charAt(0);
    }

    public static String readString(int limit)
    {
        return readKeyBoard(limit, false);
    }

    public static String readString(int limit, String defaultValue)
    {
```

```
        String str = readKeyBoard(limit, true);

        return str.equals("")? defaultValue : str;
    }

    public static void readReturn()
    {
        System.out.print("按回车键继续...");
        readKeyBoard(100, true);
    }

    public static char readMenuSelection()
    {
        char c;

        while (true)
        {
            String str = readKeyBoard(1, false);

            c = str.charAt(0);

            if (c != '1' && c != '2' && c != '3' && c != '4' && c != '5')
            {
                System.out.print("选择错误，请重新输入：");
            }
            else
            {
                break;
            }
        }

        return c;
    }

    public static char readConfirmSelection()
    {
        char c;
        boolean inputIsValid;

        do {
            String str = readKeyBoard(1, false).toUpperCase();
            c = str.charAt(0);
            inputIsValid = (c == 'Y' || c == 'N');
```

```
            if (!inputIsValid)
            {
                System.out.print("选择错误，请重新输入：");
            }
        }
        while (!inputIsValid);

        return c;
    }

    private static String readKeyBoard(int limit, boolean blankReturn)
    {
        String line = "";

        while (scanner.hasNextLine())
        {
            line = scanner.nextLine();

            if (line.length() == 0)
            {
                if (blankReturn)
                {
                    return line;
                }

                else
                {
                    continue;
                }
            }

            if (line.length() > limit)
            {
                System.out.print("输入长度（不大于" + limit + "）错误，请重新输入：");

                continue;
            }

            break;
        }

        return line;
    }
}
```

- CustomerView 类

```java
package com.myxh.cms.view;

import com.myxh.cms.javabean.Customer;
import com.myxh.cms.service.CustomerService;

import java.math.BigDecimal;
import java.util.List;

/**
 * @author MYXH
 * @date 2023/5/16
 * 这是主控模块，负责菜单显示和用户交互，也称为UI，内部要频繁到管理器对象，所以使用对象关联
 */
public class CustomerView
{
    /**
     * 关联到的管理器对象
     */
    private final CustomerService customerService = new CustomerService();

    /**
     * 进入主菜单，是项目的真正入口，不可以轻易结束
     */
    public void enterMainMenu()
    {
        // 1. 声明布尔变量
        boolean loopFlag = true;

        // 2. 循环
        do
        {
            System.out.println("\n---------------------------------客户信息管理----------

            listAllCustomers();

            System.out.print("1 添加客户 2 修改客户 3 删除客户 4 客户列表 5 退    出   请选择

            // 读取用户选择
            char choice = KeyboardUtility.readMenuSelection();

            switch (choice)
            {
                case '1' -> addNewCustomer();
                case '2' -> modifyCustomer();
```

```
                case '3' -> deleteCustomer();
                case '4' -> listAllCustomers();
                case '5' ->
                {
                    System.out.print("确认是否退出(Y/N) : ");

                    // 获取用户输入的确认
                    char confirm = KeyboardUtility.readConfirmSelection();

                    if (confirm == 'Y')
                    {
                        loopFlag = false;
                    }
                }
            }
        } while (loopFlag);
    }

    /**
     * 添加新员工
     */
    private void addNewCustomer()
    {
        Customer customer = new Customer();

        System.out.println("---------------------添加客户---------------------");

        System.out.print("姓名 : ");
        String name = KeyboardUtility.readString(10);
        customer.setName(name);
        System.out.print("性别 : ");
        String gender = KeyboardUtility.readString(1);
        customer.setGender(gender);
        System.out.print("年龄 : ");
        int age = KeyboardUtility.readInt();
        customer.setAge(age);
        System.out.print("工资 : ");
        BigDecimal salary = KeyboardUtility.readBigDecimal();
        customer.setSalary(salary);
        System.out.print("电话 : ");
        String phone = KeyboardUtility.readString(15);
        customer.setPhone(phone);

        // 通过调用管理器对象完成员工添加
```

```
        customerService.addCustomer(customer);

        System.out.println("---------------------添加完成---------------------");
    }

    /**
     * 修改员工
     */
    private void modifyCustomer ()
    {
        System.out.println("---------------------修改客户---------------------");

        System.out.print("请选择待修改客户ID(-1退出) : ");

        // 获取用户输入的 id

        int id = KeyboardUtility.readInt();
        if (id == -1)
        {
            return;
        }

        // 根据编号定位要修改的目标对象
        Customer target = customerService.getCustomer(id);

        if (target == null)
        {
            System.out.println("--------------指定ID[" + id + "]的客户不存在----------

            return;
        }

        System.out.println("<直接回车表示不修改>");

        System.out.print("姓名(" + target.getName() + ") : ");
        String name = KeyboardUtility.readString(10, target.getName());
        target.setName(name);
        System.out.print("年龄(" + target.getAge() + ") : ");
        int age = KeyboardUtility.readInt(target.getAge());
        target.setAge(age);
        System.out.print("工资(" + target.getSalary() + ") : ");
        BigDecimal salary = KeyboardUtility.readBigDecimal(target.getSalary());
        target.setSalary(salary);
        System.out.print("电话(" + target.getPhone() + ") : ");
```

```
        String phone = KeyboardUtility.readString(15, target.getPhone());
        target.setPhone(phone);

        customerService.modifyCustomer(id, target);

        System.out.println("----------------------修改完成----------------------");
    }

    /**
     * 删除员工
     */
    private void deleteCustomer ()
    {
        System.out.println("----------------------删除客户----------------------");

        System.out.print("请选择待删除客户ID(-1退出) : ");

        // 获取用户输入的 id
        int id = KeyboardUtility.readInt();

        if (id == -1)
        {
            return;
        }

        System.out.print("确认是否删除(Y/N) : ");

        // 获取用户输入的确认
        char confirm = KeyboardUtility.readConfirmSelection();

        if (confirm == 'Y')
        {
            boolean flag = customerService.removeCustomer(id);

            if (flag)
            {
                System.out.println("----------------------删除完成----------------------"
            }
            else
            {
                System.out.println("--------------指定ID[" + id + "]的客户不存在--------
            }
        }
    }
```

```
    /**
     * 员工列表
     */
    private void listAllCustomers()
    {
        System.out.println("---------------------------------客户列表---------------------

        // 真的获取所有员工
        List<Customer> list = customerService.getList();

        if (list == null || list.size() == 0)
        {
            System.out.println("没有数据，请添加新数据...");
        }
        else
        {
            System.out.println("ID\t姓名\t\t性别\t\t年龄\t\t工资\t\t\t电话");

            for (Customer customer : list)
            {
                System.out.println(customer);
            }
        }

        System.out.println("-----------------------------------------------------------
    }
}
```