



JDBC

Java Database Connectivity

Author: Gavin

Version: 9.0.2

一、引言

- 1.1 如何操作数据库
- 1.2 实际开发中，会采用客户端操作数据库吗？

二、JDBC (Java Database Connectivity)

- 2.1 什么是 JDBC？
- 2.2 JDBC 核心思想
 - 2.2.1 MySQL 数据库驱动
 - 2.2.2 JDBC API

2.3 环境搭建

三、JDBC 开发步骤【重点】

- 3.1 注册驱动
- 3.2 连接数据库
- 3.3 获取发送 SQL 的对象
- 3.4 执行SQL语句
- 3.5 处理结果
- 3.6 释放资源
- 3.7 综合案例

四、ResultSet (结果集)

- 4.1 接收结果集
- 4.2 遍历 ResultSet 中的数据
 - 4.2.1 遍历方法
- 4.3 综合案例
 - 4.3.1 根据列的名称获取
 - 4.3.2 根据列的编号获取数据

五、常见错误

六、综合案例【登录】

- 6.1 创建表

- 6.2 实现登录

七、SQL注入问题

- 7.1 什么是 SQL 注入
- 7.2 如何避免 SQL 注入

八、PreparedStatement 【重点】

- 8.1 PreparedStatement的应用
 - 8.1.1 参数标记
 - 8.1.2 动态参数绑定

九、封装工具类

- 9.1 重用性方案
 - 9.1.1 重用工具类实现
- 9.2 跨平台方案
 - 9.2.1 跨平台工具类实现

十、ORM

- 10.1 实体类(entity): 零散数据的载体
 - 10.1.1 ORM应用

十一、DAO 数据访问对象 (Data Access Object)

- 11.1 创建数据库
- 11.2 封装实体类
- 11.3 编写 DaoImpl 类

十二、Date工具类

- 12.1 java.util.Date
- 12.2 java.sql.Date
- 12.3 SimpleDateFormat
- 12.3.1 SimpleDateFormat应用

12.4 封装DateUtils工具类

十三、Service业务逻辑层

- 13.1 什么是业务?
- 13.2 Service开发流程
 - 13.2.1 编写 service 实现转账功能

十四、事务

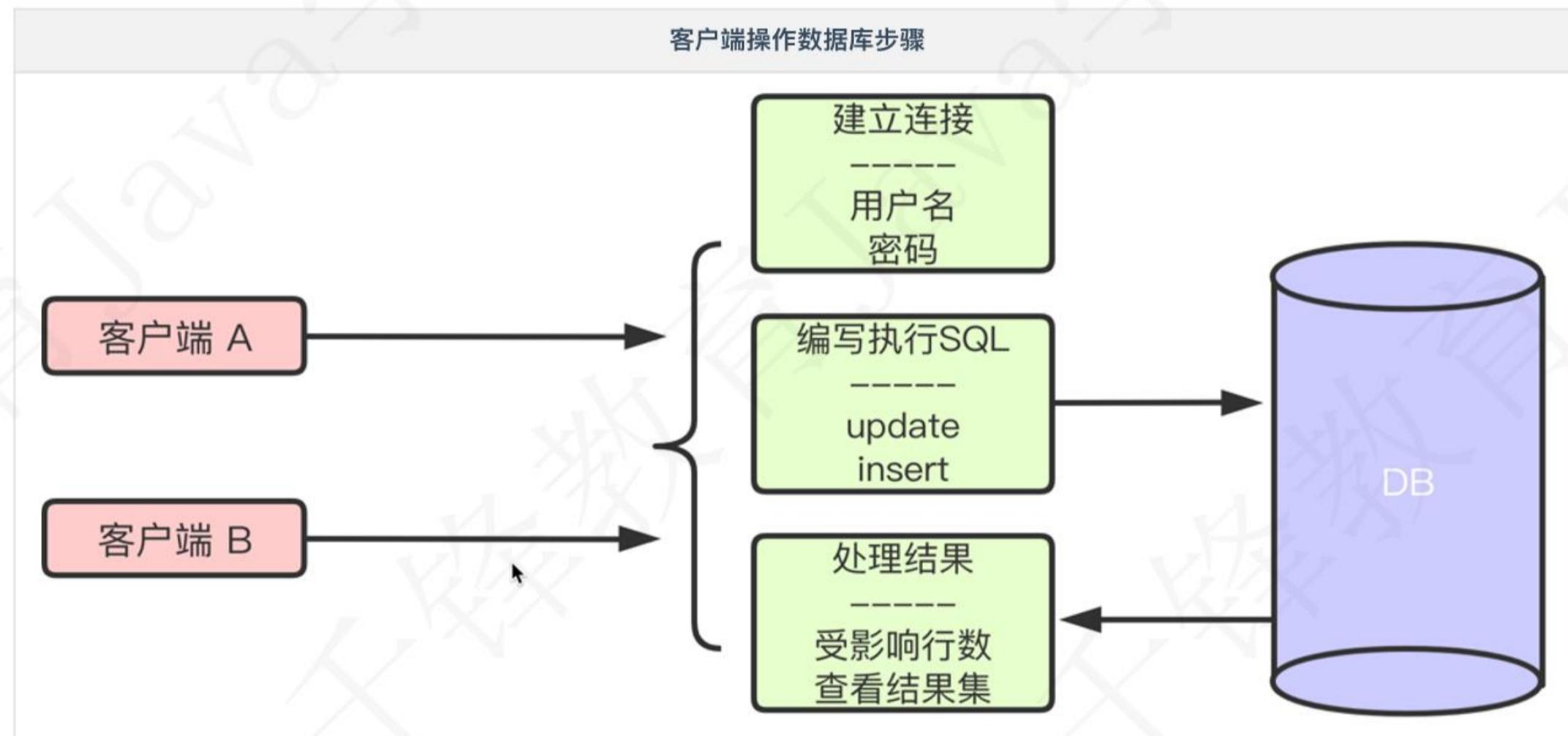
- 14.1 service 层控制事务
- 14.2 解决方案1：传递 Connection
 - 14.2.1 传递的问题

14.3 解决方案2: ThreadLocal
14.4 ThreadLocal应用
14.4.1 参数绑定
十五、 事务的封装
15.1 完善工具类
十六、 三层架构
16.1 什么是三层
16.2 三层架构项目搭建 (按开发步骤)
十七、 DaoUtils
17.1 commonsUpdate
17.2 commonsSelect
十八、 Druid连接池
18.1 Druid 连接池使用步骤
18.1.1 database.properties配置文件
18.1.2 连接池工具类
十九、 Apache的DbUtils使用
19.1 DbUtils简介
19.1.1 DbUtils主要包含
19.2 DbUtils的使用步骤
19.2.1 DbUtils工具类
19.2.2 UserDaoImpl 数据访问对象

一、引言

1.1 如何操作数据库

使用客户端工具访问数据库，需要手工建立连接，输入用户名和密码登录，编写 SQL 语句，点击执行，查看操作结果（结果集或受影响行数）。



1.2 实际开发中，会采用客户端操作数据库吗？

在实际开发中，当用户的数据发生改变时，不可能通过客户端操作执行 SQL 语句，因为操作量过大，无法保证效率和正确性。

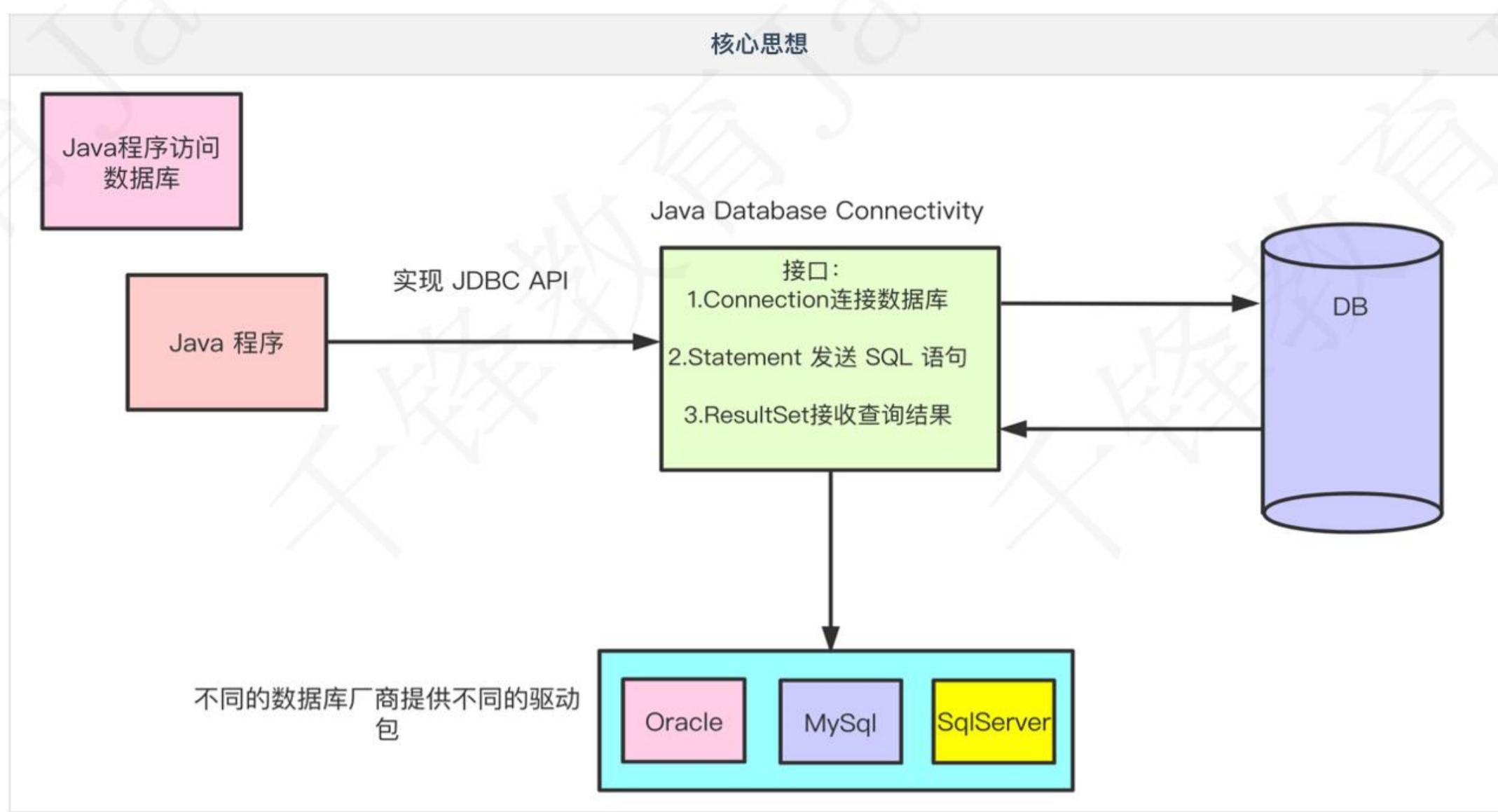
二、JDBC (Java Database Connectivity)

2.1 什么是 JDBC？

JDBC (Java Database Connectivity) Java 连接数据库的规范（标准），可以使用 Java 语言连接数据库完成 CRUD 操作。

2.2 JDBC 核心思想

Java 中定义了访问数据库的接口，可以为多种关系型数据库提供统一的访问方式。由数据库厂商提供驱动实现类（Driver 数据库驱动）。



2.2.1 MySQL 数据库驱动

- mysql-connector-java-5.1.X 适用于 5.X 版本
- mysql-connector-java-8.0.X 适用于 8.X 版本

2.2.2 JDBC API

JDBC 是由多个接口和类进行功能实现。

类型	权限定名	简介
class	java.sql.DriverManager	管理多个数据库驱动类，提供了获取数据库连接的方法
interface	java.sql.Connection	代表一个数据库连接（当connection不是null时，表示已连接数据库）
interface	java.sql.Statement	发送SQL语句到数据库工具
interface	java.sql.ResultSet	保存SQL查询语句的结果数据（结果集）
class	java.sql.SQLException	处理数据库应用程序时所发生的异常

2.3 环境搭建

- 在项目下新建 lib 文件夹，用于存放 jar 文件。
- 将 mysql 驱动mysql-connector-java-5.1.X复制到项目的 lib 文件夹中。
- 选中 lib 文件夹右键 Add as Library，点击 OK。

三、JDBC 开发步骤【重点】

3.1 注册驱动

使用 `Class.forName("com.mysql.jdbc.Driver");` 手动加载字节码文件到 JVM 中。

```
Class.forName("com.mysql.jdbc.Driver"); //加载驱动
```

3.2 连接数据库

- 通过 `DriverManager.getConnection(url,user,password)` 获取数据库连接对象
 - URL:`jdbc:mysql://localhost:3306/database`
 - username:`root`
 - password:`1234`

```
Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/database?
useUnicode=true&characterEncoding=utf8", "root", "1234");
```

- URL(Uniform Resource Locator) 统一资源定位符：由协议、IP、端口、SID（程序实例名称）组成

3.3 获取发送 SQL 的对象

通过 Connection 对象获得 Statement 对象，用于对数据库进行通用访问。

```
Statement statement = conn.createStatement();
```

3.4 执行SQL语句

执行 SQL 语句并接收执行结果。

```
String sql ="INSERT INTO t_jobs(JOB_ID,JOB_TITLE,MIN_SALARY,MAX_SALARY)  
VALUES('JAVA_Le','JAVA_Lecturer',4000,10000);"  
  
int result = statement.executeUpdate(sql); //执行SQL语句并接收结果
```

- 注意：在编写 DML 语句时，一定要注意字符串参数的符号是单引号 '值'
- DML 语句：增删改时，返回受影响行数（int 类型）。
- DQL 语句：查询时，返回结果数据(ResultSet 结果集)。

3.5 处理结果

接受处理操作结果。

```
if(result == 1){  
    System.out.println("Success");  
}
```

- 受影响行数：逻辑判断、方法返回。
- 查询结果集：迭代、依次获取。

3.6 释放资源

遵循 **先开后关** 原则，释放所使用到的资源对象。

```
statement.close();  
conn.close();
```

3.7 综合案例

整合以上核心六步，实现向数据库表中插入一条数据。

```
package com.qf.JDBC;  
  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.Statement;  
  
public class DeleteJdbc {  
    public static void main(String[] args) throws Exception{  
        //1.加载驱动  
        Class.forName("com.mysql.jdbc.Driver");  
        //2.获得连接对象  
        Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/companydb","root","1234");  
        //3.获得执行SQL的对象  
        Statement statement = connection.createStatement();  
        //4.执行SQL语句，并接收结果  
        int result = statement.executeUpdate("delete from t_jobs where job_id = 'H5_mgr'");  
        //5.处理结果  
        if(result==1){  
            System.out.println("删除成功! ");  
        }else{  
            System.out.println("删除失败! ");  
        }  
        //6.释放资源  
        statement.close();  
        connection.close();  
    }  
}
```

四、ResultSet (结果集)

在执行查询 SQL 后，存放查询到的结果集数据。

4.1 接收结果集

```
ResultSet rs = statement.executeQuery(sql);  
  
ResultSet rs= statement.executeQuery("SELECT * FROM t_employees");
```

4.2 遍历 ResultSet 中的数据

ResultSet 以表 (table) 结构进行临时结果的存储，需要通过 JDBC API 将其中数据进行依次获取。

- 数据行指针：初始位置在第一行数据前，每调用一次 boolean next() 方法 ResultSet 的指针向下移动一行，结果为 true，表示当前行有数据。
- rs.getXXX(整数); 代表根据列的编号顺序获得，从 1 开始。
- rs.getXXX("列名"); 代表根据列名获得。

```
boolean next() throws SQLException //判断 rs 结果集中下一行是否存在数据
```

4.2.1 遍历方法

```
int getInt(int columnIndex) throws SQLException //获得当前行第N列的int值  
int getInt(String columnLabel) throws SQLException //获得当前行columnLabel列的int值  
  
double getDouble(int columnIndex) throws SQLException //获得当前行第N列的double值  
double getDouble(String columnLabel) throws SQLException //获得当前行columnLabel列的double值  
  
String getString(int columnIndex) throws SQLException //获得当前行第N列的String值  
String getString(String columnLabel) throws SQLException //获得当前行columnLabel列的String值  
  
.....
```

- 注意：列的编号从 1 开始。

4.3 综合案例

对 t_jobs 表中的所有数据进行遍历。

4.3.1 根据列的名称获取

```
package com.qf.www.test;  
  
import java.sql.*;  
  
public class JobsQuery {  
    public static void main(String[] args) {  
        //1.加载驱动  
        Class.forName("com.mysql.jdbc.Driver");  
  
        //2.获取数据库连接对象  
        Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/product", "root",  
"1234");  
  
        //3.获取发送 sql 语句对象  
        Statement statement = connection.createStatement();  
  
        //4.执行 SQL 语句并接收结果集  
        ResultSet resultSet = statement.executeQuery("select * from t_jobs");  
        //5 处理结果集  
        while(resultSet.next()){  
            //5.1有数据，依据列名获取数据  
            String job_id = resultSet.getString("job_id");  
            String job_title = resultSet.getString("job_title");  
            int min_salary = resultSet.getInt("min_salary");  
            int max_salary = resultSet.getInt("max_salary");  
            System.out.println(job_id+"\t"+job_title+"\t"+min_salary+"\t"+max_salary);  
        }  
  
        //6.释放资源  
        rs.close();  
        statement.close();  
        connection.close();  
    }  
}
```

4.3.2 根据列的编号获取数据

```
//。。。与上无异
while(resultSet.next()){
    //5.2有数据，依据列名获取数据
    String job_id = resultSet.getString(1);
    String job_title = resultSet.getString(2);
    int min_salary = resultSet.getInt(3);
    int max_salary = resultSet.getInt(4);
    System.out.println(job_id+"\t"+job_title+"\t"+min_salary+"\t"+max_salary);
}
//释放资源
```

五、常见错误

- `java.lang.ClassNotFoundException`：找不到类（类名书写错误、没有导入jar包）
- `java.sql.SQLException`：与sql语句相关的错误（约束错误、表名列名书写错误）建议：在客户端工具中测试SQL语句之后再粘贴在代码中
- `com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: Unknown column` 原因：列值String类型没有加单引号
- `Duplicate entry '1' for key 'PRIMARY'` 原因：主键值已存在或混乱，更改主键值或清空表
- `com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: Unknown column 'password' in`
 - 原因：可能输入的值的类型不对，确定是否插入的元素时对应的值的类型正确

六、综合案例【登录】

6.1 创建表

- 创建一张用户表 User
 - id，主键、自动增长。
 - 用户名，字符串类型，唯一、非空
 - 密码，字符串类型，非空
 - 手机号码，字符串类型
- 插入 2 条测试语句

6.2 实现登录

- 通过控制台用户输入用户名和密码。
- 用户输入的用户名和密码作为条件，编写查询 SQL 语句。
- 如果该用户存在，提示登录成功，反之提示失败。

七、SQL注入问题

7.1 什么是 SQL 注入

用户输入的数据中有 SQL 关键字或语法并且参与了 SQL 语句的编译，导致 SQL 语句编译后的条件含义为 true，一直得到正确的结果。这种现象称为 SQL 注入。

7.2 如何避免 SQL 注入

由于编写的 SQL 语句是在用户输入数据，整合后再进行编译。所以为了避免 SQL 注入的问题，我们要使 SQL 语句在用户输入数据前就已进行编译成完整的 SQL 语句，再进行填充数据。

八、PreparedStatement【重点】

PreparedStatement 继承了 Statement 接口，执行 SQL 语句的方法无异。

8.1 PreparedStatement的应用

作用：

- 预编译SQL语句，效率高。
- 安全，避免SQL注入。

- 可以动态的填充数据，执行多个同构的SQL语句。

8.1.1 参数标记

```
//1.预编译 SQL 语句
PreparedStatement pstmt = conn.prepareStatement("select * from user where username=? and password=?");

• 注意：JDBC中的所有参数都由? 符号占位，这被称为参数标记。在执行SQL语句之前，必须为每个参数提供值。
```

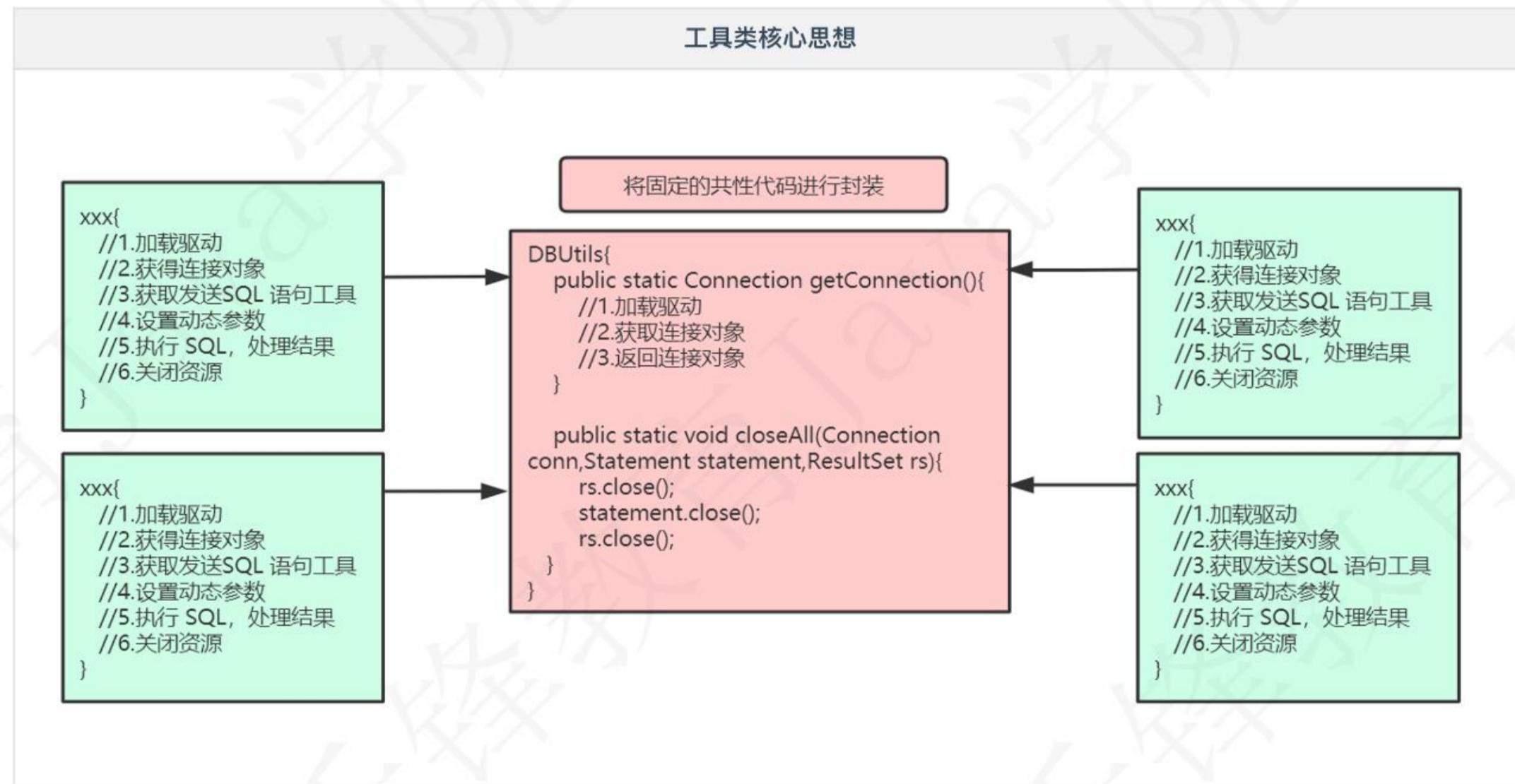
8.1.2 动态参数绑定

`pstmt.setXxx(下标,值)` 参数下标从 1 开始，为指定参数下标绑定值

```
//1.预编译 SQL 语句
PreparedStatement pstmt = conn.prepareStatement("select * from user where username=? and password=?");
//2.为参数下标赋值
pstmt.setString(1,username);
pstmt.setString(2,password);
```

九、封装工具类

- 在实际JDBC的使用中，存在着大量的重复代码：例如连接数据库、关闭数据库等这些操作！
- 我们需要把传统的JDBC代码进行重构，抽取出通用的JDBC工具类！以后连接任何数据库、释放资源都可以使用这个工具类。



9.1 重用性方案

- 封装获取连接、释放资源两个方法。
 - 提供`public static Connection getConnection(){}方法。`
 - 提供`public static void closeAll(Connection conn , Statement sm , ResultSet rs){}方法。`

9.1.1 重用工具类实现

```
package com.qf.JDBC;

import java.sql.*;

/**
 * 重用性方案
 * 获取连接
 * 释放资源
 */
public class DBUtils {

    static {//类加载，执行一次!
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    public static Connection getConnection() {
        //1.加载驱动
        //2.获得连接对象
        //3.返回连接对象
        return null;
    }

    public static void closeAll(Connection conn, Statement sm, ResultSet rs) {
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (sm != null) {
            try {
                sm.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (rs != null) {
            try {
                rs.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```

//1.获取连接
public static Connection getConnection() {
    Connection connection = null;
    try {
        connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/companydb", "root", "1234");
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return connection;
}

//2.释放资源
public static void closeAll(Connection connection, Statement statement, ResultSet resultSet) {
    try {
        if (resultSet != null) {
            resultSet.close();
        }
        if (statement != null) {
            statement.close();
        }
        if (connection != null) {
            connection.close();
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

```

9.2 跨平台方案

- 定义 `public static final Properties prop = new Properties(); //读取配置文件的Map`
- 定义static{
 - //首次使用工具类时，加载驱动
`InputStream is = JDBCUtil.class.getResourceAsStream("路径"); //通过复用本类自带流，读取jdbc.properties配置文件。`
`classPath = bin`
 - `prop.load(is); //通过prop对象将流中的配置信息分割成键值对`
 - `String driverName = prop.getProperty("driver"); //通过driverName的键获取对应的值 (com.mysql.jdbc.Driver)`
 - `Class.forName(driverName); //加载驱动`
}

9.2.1 跨平台工具类实现

在src目录下新建 db.properties 文件。

```

driver=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/mydb3
user=root
password=1234

```

工具类的封装。

```

package com.qf.jdbc2;

import java.io.IOException;
import java.io.InputStream;
import java.sql.*;
import java.util.Properties;

public class DBUtils {
    private static final Properties PROPERTIES = new Properties(); //存储配置文件的map

    static {

        InputStream is = DBUtils.class.getResourceAsStream("/db.properties");

        try {
            PROPERTIES.load(is); //通过流，将配置文件内容加载到properties集合
            Class.forName(PROPERTIES.getProperty("driver"));
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}

```

```

public static Connection getConnection() {
    Connection connection = null;
    try {
        connection = DriverManager.getConnection(PROPERTIES.getProperty("url"),
PROPERTIES.getProperty("username"), PROPERTIES.getProperty("password"));
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return connection;
}

public static void closeAll(Connection connection, Statement statement, ResultSet resultSet) {
    try {
        if (resultSet != null) {
            resultSet.close();
        }
        if (statement != null) {
            statement.close();
        }
        if (connection != null) {
            connection.close();
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
}

```

十、ORM

ORM (Object Relational Mapping)。

从数据库查询到的结果集(ResultSet)在进行遍历时，逐行遍历，取出的都是零散的数据。在实际应用开发中，我们需要将零散的数据进行封装整理。

10.1 实体类(entity): 零散数据的载体

- 一行数据中，多个零散的数据进行整理。
- 通过entity的规则对表中的数据进行对象的封装。
- 表名=类名；列名=属性名；提供各个属性的get、set方法。**
- 提供无参构造方法、(视情况添加有参构造)。

10.1.1 ORM应用

entity实体类

```

package com.qf.www.test;

public class T_Jobs {
    private String job_id;
    private String job_title;
    private int min_salary;
    private int max_salary;

    @Override
    public String toString() {
        return "T_Jobs{" +
            "job_id='" + job_id + '\'' +
            ", job_title='" + job_title + '\'' +
            ", min_salary=" + min_salary +
            ", max_salary=" + max_salary +
            '}';
    }

    public String getJob_id() {
        return job_id;
    }

    public void setJob_id(String job_id) {
        this.job_id = job_id;
    }

    public String getJob_title() {
        return job_title;
    }

    public void setJob_title(String job_title) {

```

```
        this.job_title = job_title;
    }

    public int getMin_salary() {
        return min_salary;
    }

    public void setMin_salary(int min_salary) {
        this.min_salary = min_salary;
    }

    public int getMax_salary() {
        return max_salary;
    }

    public void setMax_salary(int max_salary) {
        this.max_salary = max_salary;
    }

    public T_Jobs() {
    }

    public T_Jobs(String job_id, String job_title, int min_salary, int max_salary) {
        this.job_id = job_id;
        this.job_title = job_title;
        this.min_salary = min_salary;
        this.max_salary = max_salary;
    }
}
```

查询结果封装

```
package com.qf.www.test;

import java.sql.*;

public class JobsQuery {
    public static void main(String[] args) {

        Connection connection = null;
        Statement statement = null;
        ResultSet resultSet = null;

        String URL = "jdbc:mysql://localhost:3306/product";
        String user = "root";
        String password = "1234";

        try {
            //1.加载驱动
            Class.forName("com.mysql.jdbc.Driver");

            //2.获取数据库连接对象
            connection = DriverManager.getConnection(URL, user, password);

            //3.编写 SQL 语句
            String sql = "select * from t_jobs";

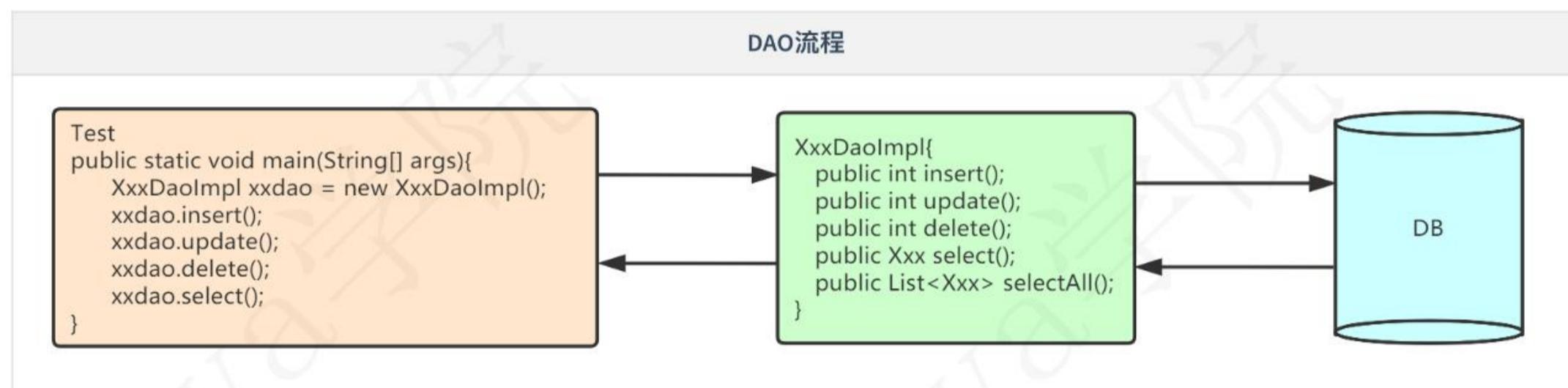
            //4.获取发送 sql 语句对象
            statement = connection.createStatement();

            //5.执行 SQL 语句并接收结果集
            resultSet = statement.executeQuery(sql);
            //5.1使用 while 循环判断下一行是否有数据
            while(resultSet.next()){
                //5.2有数据，依据列名获取数据
                String job_id = resultSet.getString(1);
                String job_title = resultSet.getString(2);
                int min_salary = resultSet.getInt(3);
                int max_salary = resultSet.getInt(4);
                //5.3 创建实体类对象
                T_Jobs t_jobs = new T_Jobs();
                //5.4 每列数据对应属性进行赋值
                t_jobs.setJob_id(job_id);
                t_jobs.setJob_title(job_title);
                t_jobs.setMin_salary(min_salary);
                t_jobs.setMax_salary(max_salary);
                System.out.println(t_jobs);
            }
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                //6.释放资源
            }
```

```
        if (resultSet != null) {
            resultSet.close();
        }
        if (statement != null) {
            statement.close();
        }
        if (connection != null) {
            connection.close();
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

十一、 DAO 数据访问对象 (Data Access Object)

- DAO 实现了业务逻辑与数据库访问相分离。
 - 对同一张表的所有操作封装在XxxDaoImpl对象中。
 - 根据增删改查的不同功能实现具体的方法 (insert、update、delete、select、selectAll) 。



11.1 创建数据库

- 创建一张表 Person，有以下列：
 - id: int, 主键, 自动增长
 - name: varchar(20) 非空
 - age: int 非空
 - bornDate: Date
 - email: 字符串
 - address: 字符串

11.2 封装实体类

创建entity实体类 Person， 编写属性私有化， 构造方法， get/set 方法。

11.3 编写 Daolmpl 类

编写 DaoImpl 类，提供增删改查方法，使用 JDBC 开发步骤，完成功能。

```
package com.qf.person;

import sun.awt.image.DataBufferNative;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

/**
 * 增、删、改、查单个、查所有
 * 只做数据库访问操作！不参与逻辑判断
 * 数据库一张表的访问的操作复用！
 */
public class PersonDaoImpl {
    //新增
    public int insert(Person person){
```

```
Connection connection = null;
PreparedStatement preparedStatement = null;
String sql = "insert into person(name,age,borndate,email,address) values(?,?,?,?,?)";
try {
    connection = DBUtils.getConnection();
    preparedStatement = connection.prepareStatement(sql);
    preparedStatement.setString(1,person.getName());
    preparedStatement.setInt(2,person.getAge());
    preparedStatement.setDate(3,null);
    preparedStatement.setString(4,person.getEmail());
    preparedStatement.setString(5,person.getAddress());
    int result = preparedStatement.executeUpdate();
    return result;
} catch (SQLException e) {
    e.printStackTrace();
}finally {
    DBUtils.closeAll(connection,preparedStatement,null);
}
return 0;
}

//修改
public int update(Person person){
Connection connection =null;
PreparedStatement preparedStatement = null;
String sql = "update person set name=?,age=?,borndate=?,email=?,address=? where id = ?";
try {
    connection = DBUtils.getConnection();
    preparedStatement = connection.prepareStatement(sql);
    preparedStatement.setString(1,person.getName());
    preparedStatement.setInt(2,person.getAge());
    preparedStatement.setDate(3,null);
    preparedStatement.setString(4,person.getEmail());
    preparedStatement.setString(5,person.getAddress());
    preparedStatement.setInt(6,person.getId());
    int result = preparedStatement.executeUpdate();
    return result;
} catch (SQLException e) {
    e.printStackTrace();
}finally {
    DBUtils.closeAll(connection,preparedStatement,null);
}
return 0;
}

//删除
public int delete(int id){
Connection connection = null;
PreparedStatement preparedStatement = null;
String sql = "delete from person where id= ?;";
connection = DBUtils.getConnection();
try {
    preparedStatement = connection.prepareStatement(sql);
    preparedStatement.setInt(1,id);
    int result = preparedStatement.executeUpdate();
    return result;
} catch (SQLException e) {
    e.printStackTrace();
}finally {
    DBUtils.closeAll(connection,preparedStatement,null);
}
return 0;
}

//查单个
public Person select(int id){
Connection connection = null;
PreparedStatement preparedStatement = null;
ResultSet resultSet = null;
String sql = "select * from person where id = ?;";
Person person = null;
try {
    connection = DBUtils.getConnection();
    preparedStatement = connection.prepareStatement(sql);
    preparedStatement.setInt(1,id);
    resultSet = preparedStatement.executeQuery();
    if(resultSet.next()){
        person = new Person();
        int pid= resultSet.getInt("id");
        String name = resultSet.getString("name");
        int age = resultSet.getInt("age");
        Date bornDate = resultSet.getDate("borndate");
        String email = resultSet.getString("email");
        String address = resultSet.getString("address");
        person.setId(pid);
        person.setName(name);
        person.setAge(age);
        person.setBornDate(bornDate);
        person.setEmail(email);
        person setAddress(address);
    }
    return person;
}
}
```

```

        } catch (SQLException e) {
            e.printStackTrace();
        }finally {
            DBUtils.closeAll(connection,preparedStatement,resultSet);
        }
        return null;
    }

    //查所有
    public List<Person> selectAll(){
        Connection connection = null;
        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;
        Person person = null;
        List<Person> personList = new ArrayList<>();

        try {
            connection = DBUtils.getConnection();
            preparedStatement = connection.prepareStatement("select * from person;");
            resultSet = preparedStatement.executeQuery();

            while(resultSet.next()){
                int pid= resultSet.getInt("id");
                String name = resultSet.getString("name");
                int age = resultSet.getInt("age");
                Date bornDate = resultSet.getDate("borndate");
                String email = resultSet.getString("email");
                String address = resultSet.getString("address");

                person = new Person(pid,name,age,bornDate,email,address);
                personList.add(person);
            }
            return personList;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
}

```

十二、Date工具类

现有问题：数据库存储的数据类型为java.sql.Date。而我们Java应用层存储日期类型为java.util.Date。当我们用Java应用程序插入带有日期的数据到数据库中时，需要进行转换。

12.1 java.util.Date

- Java语言常规应用层面的日期类型，可以通过字符串创建对应的时间对象。
- 无法直接通过JDBC插入到数据库。

12.2 java.sql.Date

- 不可以通过字符串创建对应的时间对象，只能通过毫秒值创建对象(1970年至今的毫秒值)。
- 可以直接通过JDBC插入到数据库。

12.3 SimpleDateFormat

格式化和解析日期的具体类。允许进行格式化（日期 -> 文本）、解析（文本 -> 日期）和规范化。

12.3.1 SimpleDateFormat应用

```

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");//指定日期格式

java.util.Date date = sdf.parse(String dateStr);//将字符串解析成日期类型 (java.util.Date)

String dates = sdf.format(date);//将日期格式化成字符串

```

12.4 封装DateUtils工具类

```

package com.qf.day43.t4;

import java.text.ParseException;
import java.text.SimpleDateFormat;

```

```

/**
 * 日期转换
 * 字符串转UtilDate
 * UtilDate转SqlDate
 * utilDate转成字符串
 */
public class DateUtils {
    private static final SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd");

    //字符串转Util
    public static java.util.Date strToUtilDate(String str) {
        try {
            return simpleDateFormat.parse(str);
        } catch (ParseException e) {
            e.printStackTrace();
        }
        return null;
    }
    //util转sql
    public static java.sql.Date utilToSql(java.util.Date date){
        return new java.sql.Date(date.getTime());
    }

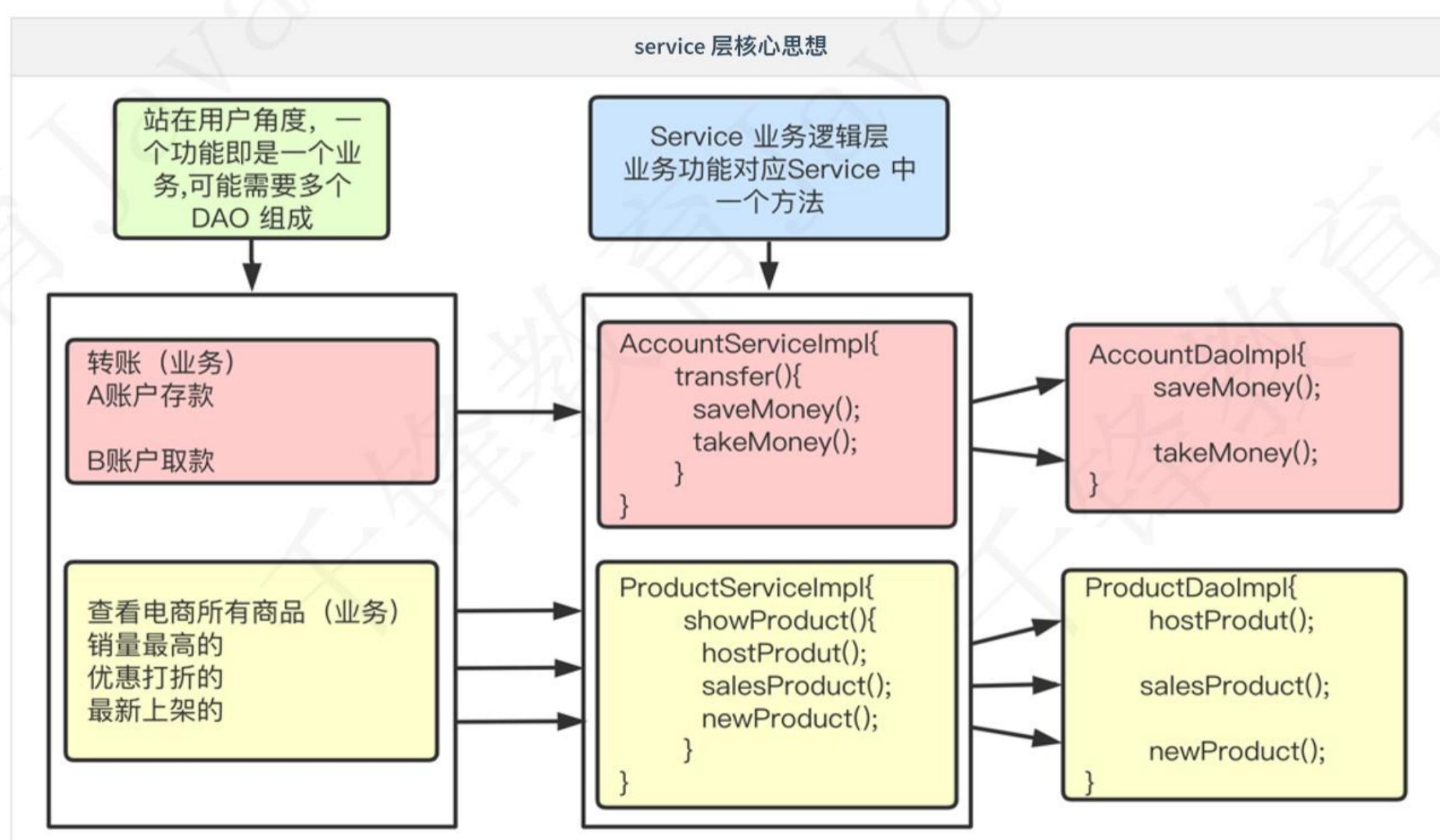
    //util转字符串
    public static String toStr(java.util.Date bornDate){
        return sdf.format(bornDate);
    }
}

```

十三、Service业务逻辑层

13.1 什么是业务？

代表用户完成的一个业务功能，可以由一个或多个DAO的调用组成。（软件所提供的一个功能都叫业务）。



13.2 Service开发流程



13.2.1 编写 service 实现转账功能

```

package com.qf.day44.accounts;

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

```

```

public class T_AccountServiceImpl {
    /**
     * 转账业务
     *
     * @param fromNo 转账卡号
     * @param pwd 转账卡号密码
     * @param toNo 收钱卡号
     * @param money 转账金额
     */
    public String transfer(String fromNo, String pwd, String toNo, double money) { //收参数
        String result = "转账失败!";
        //2.组织业务功能
        T_AccountDaoImpl accountDao = new T_AccountDaoImpl();

        try {
            //2.1验证fromNo是否存在
            T_Account fromAcc = accountDao.select(fromNo);

            if (fromAcc == null) {
                throw new RuntimeException("----卡号不存在----");
            }
            //2.2验证fromNo的密码是否正确
            if (!fromAcc.getPassword().equals(pwd)) {
                throw new RuntimeException("----密码错误----");
            }
            //2.3验证余额是否充足
            if (fromAcc.getBalance() < money) {
                throw new RuntimeException("----余额不足----");
            }
            //2.4验证toNo是否存在
            T_Account toAcc = accountDao.select(toNo);

            if (toAcc == null) {
                throw new RuntimeException("----对方卡号不存在----");
            }
            //2.5减少fromNo的余额
            //修改自己的金额, 将余额-转账金额替换原有的属性
            fromAcc.setBalance(fromAcc.getBalance() - money);

            accountDao.update(fromAcc);

            //2.6增加toNo的余额
            toAcc.setBalance(toAcc.getBalance() + money);
            accountDao.update(toAcc);
            result = "转账成功!";
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            DBUtils.closeAll(connection, null, null);
        }
        return result;
    }
}

```

十四、事务

- 在JDBC中，获得 Connection 对象开始事务--提交或回滚--关闭连接。其事务策略是
 - conn.setAutoCommit(false); //true 等价于 1, false 等价于 0
 - conn.commit(); //手动提交事务
 - conn.rollback(); //手动回滚事务

14.1 service 层控制事务

```

package com.qf.day44.accounts;

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

public class T_AccountServiceImpl {

    /**
     * 转账业务
     *
     * @param fromNo 转账卡号
     * @param pwd 转账卡号密码
     * @param toNo 收钱卡号
     * @param money 转账金额
     */

```

```

public String transfer(String fromNo, String pwd, String toNo, double money) { //收参数
    String result = "转账失败!";
    //2.组织业务功能
    T_AccountDaoImpl accountDao = new T_AccountDaoImpl();

    //拿一个连接
    Connection connection = null;

    try {
        //建立了数据库连接
        connection = DBUtils.getConnection();
        //开启事务! 并且关闭事务的自动提交
        connection.setAutoCommit(false);
        //2.1验证fromNo是否存在
        T_Account fromAcc = accountDao.select(fromNo);

        if (fromAcc == null) {
            throw new RuntimeException("----卡号不存在----");
        }
        //2.2验证fromNo的密码是否正确
        if (!fromAcc.getPassword().equals(pwd)) {
            throw new RuntimeException("----密码错误----");
        }
        //2.3验证余额是否充足
        if (fromAcc.getBalance() < money) {
            throw new RuntimeException("----余额不足----");
        }
        //2.4验证toNo是否存在
        T_Account toAcc = accountDao.select(toNo);

        if (toAcc == null) {
            throw new RuntimeException("----对方卡号不存在----");
        }

        //2.5减少fromNo的余额
        //修改自己的金额, 将余额-转账金额替换原有的属性
        fromAcc.setBalance(fromAcc.getBalance() - money);

        accountDao.update(fromAcc);

        int a = 10/0;//模拟程序转账出现异常!

        //2.6增加toNo的余额
        toAcc.setBalance(toAcc.getBalance() + money);
        accountDao.update(toAcc);
        result = "转账成功!";
        //执行到这里, 没有异常, 则提交事务!
        connection.commit();
    } catch (Exception e) {
        e.printStackTrace();
        try {
            //出现异常, 回滚!
            System.out.println("出现了异常! 回滚整个事务!");
            connection.rollback();
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
    } finally {
        DBUtils.closeAll(connection, null, null);
    }
    return result;
}
}

```

- 问题：当转账程序出现异常，事务控制成功了吗？

14.2 解决方案1：传递 Connection

为了解决线程中Connection对象不同步的问题，可以将Connection对象通过service传递给各个DAO方法吗？

14.2.1 传递的问题

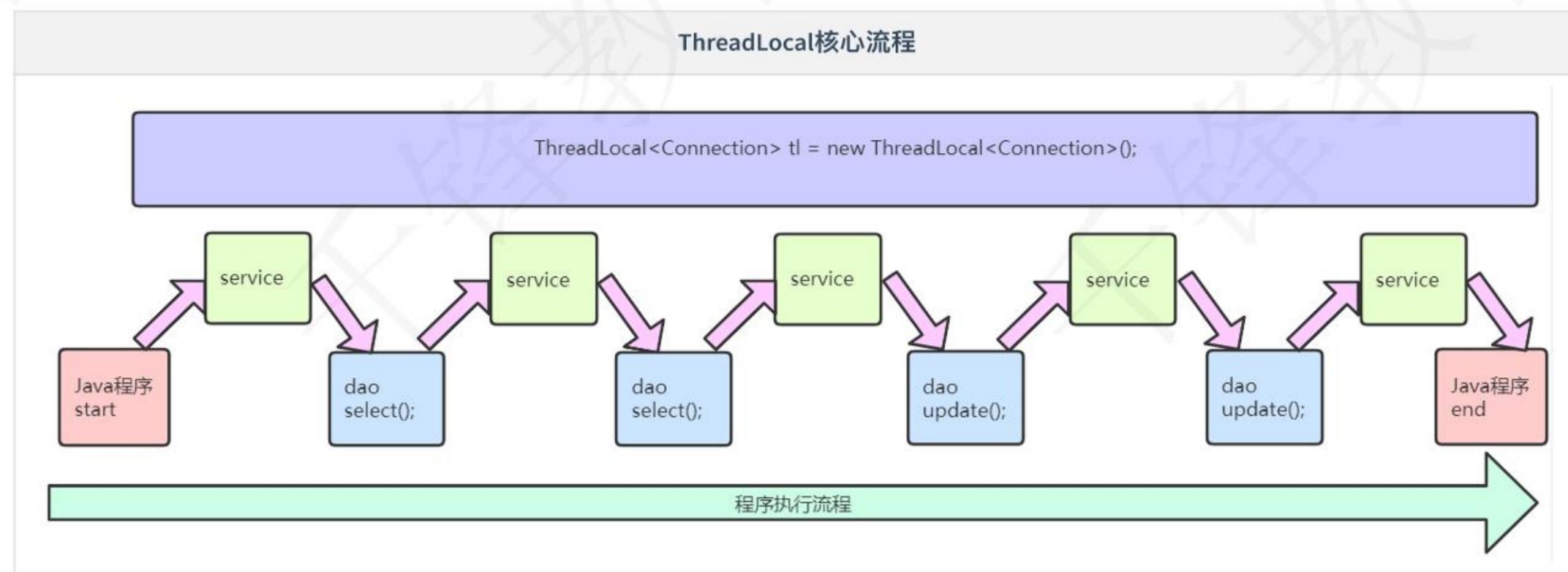
- 如果使用传递Connection，容易造成接口污染（BadSmell）。
- 定义接口是为了更容易更换实现，而将Connection定义在接口中，会造成污染当前接口。

14.3 解决方案2：ThreadLocal

- 可以将整个线程中（单线程）中，存储一个共享值。
- 线程拥有一个类似Map的属性，键值对结构<ThreadLocal对象，值>。

14.4 ThreadLocal应用

一个线程共享同一个 ThreadLocal，在整个流程中任一环节可以存值或取值。



14.4.1 参数绑定

在 DBUtils 中，将当前 Connection 对象添加到 ThreadLocal 中。

```
ThreadLocal<Connection> tl = new ThreadLocal<Connection>();

//getConnection 方法修改
public static Connection getConnection(){
    Connection conn = tl.get(); //获取线程中存储的 Connection 对象
    if(conn == null){
        conn = ds.getConnection(); //从连接池中获取一个连接
        tl.set(conn); //存储到线程对象中。
    }
    return conn;
}

//关闭所有连接 增加 tl.remove(); 移除
public static void closeAll(ResultSet rs, Statement st, Connection conn) {
    try {
        if (rs != null) {
            rs.close();
        }
        if (st != null) {
            st.close();
        }
        if (conn != null) {
            conn.close();
            tl.remove(); //将 conn 移除。
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

十五、事务的封装

将事务的开启、提交、回滚都封装在工具类中，业务层调用即可。

15.1 完善工具类

```
//开启事务
public static void begin(){
    Connection connection = getConnection();
    try {
        connection.setAutoCommit(false);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

//提交事务
public static void commit(){
    Connection connection = getConnection();
    try {
        connection.commit();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

```

        } catch (SQLException e) {
            e.printStackTrace();
        }finally {
            DBUtils.closeAll(connection,null,null);
        }
    }
    //回滚事务
    public static void rollback(){
        Connection connection = getConnection();
        try {
            connection.rollback();
        } catch (SQLException e) {
            e.printStackTrace();
        }finally {
            DBUtils.closeAll(connection,null,null);
        }
    }
}

```

十六、三层架构

16.1 什么是三层

- 表示层：
 - 命名：XXXView
 - 职责：收集用户的数据和需求、展示数据。
- 业务逻辑层：
 - 命名：XXXServiceImpl
 - 职责：数据加工处理、调用DAO完成业务实现、控制事务。
- 数据访问层：
 - 命名：XXXDaoImpl
 - 职责：向业务层提供数据，将业务层加工后的数据同步到数据库。



16.2 三层架构项目搭建（按开发步骤）

- utils 存放工具类 (DBUtils)
- entity 存放实体类 (Person)
- dao 存放 DAO 接口 (PersonDao)
 - impl 存放 DAO 接口实现类 (PersonDaoImpl)
- service 存放 service 接口 (PersonService)
 - impl 存放 service 接口实现类 (PersonServiceImpl)
- view 存放程序启动类 (main)

- 程序设计时，考虑易修改、易扩展，为Service层和DAO层设计接口，便于未来更换实现类

十七、DaoUtils

在DAO层中，对数据库表的增、删、改、查操作存在代码冗余，可对其进行抽取封装DaoUtils工具类实现复用。

17.1 commonsUpdate

```

/**
 * 公共处理增、删、改的方法
 * sql语句，参数列表
 *
 * @param sql 执行的sql语句
 */

```

```

 * @param args 参数列表。为占位符赋值
 * @return
 */
public int commonsUpdate(String sql, Object... args) {
    Connection connection = null;
    PreparedStatement preparedStatement = null;

    try {
        connection = DBUtils.getConnection();
        preparedStatement = connection.prepareStatement(sql);

        for (int i = 0; i < args.length; i++) {
            preparedStatement.setObject(i + 1, args[i]);
        }

        return preparedStatement.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        DBUtils.closeAll(null, preparedStatement, null);
    }
    return 0;
}

```

17.2 commonsSelect

```

/**
 * 公共查询方法（可查询单个对象，也可查询多个对象，可以查任何一张表）
 *
 * @param sql
 * @param args
 * @return
 */
//          select * from t_account
//          select * from t_student
//工具不知道查的是什么 调用者知道
//封装对象、对象赋值 调用者清楚
public List<T> commonsSelect(String sql, RowMapper<T> rowMapper, Object... args) {

    List<T> elements = new ArrayList<T>();

    Connection connection = null;
    PreparedStatement preparedStatement = null;
    ResultSet resultSet = null;

    try {
        connection = DBUtils.getConnection();
        preparedStatement = connection.prepareStatement(sql);

        if(args !=null){
            for (int i = 0; i < args.length; i++) {
                preparedStatement.setObject(i + 1, args[i]);
            }
        }
    }

    resultSet = preparedStatement.executeQuery();

    while (resultSet.next()) {
        //根据查询到的结果完成ORM，如何进行对象的创建及赋值？
        T t = rowMapper.getRow(resultSet); //回调---->调用者提供的一个封装方法ORM
        elements.add(t);
    }
} catch (SQLException e) {
    e.printStackTrace();
} finally {
    DBUtils.closeAll(null, preparedStatement, resultSet);
}
return elements;
}

```

十八、Druid连接池

在程序初始化时，预先创建指定数量的数据库连接对象存储在池中。当需要连接数据库时，从连接池中取出现有连接；使用完毕后，也不会进行关闭，而是放回池中，实现复用，节省资源。

18.1 Druid连接池使用步骤

- 创建 database.properties 配置文件。

- 引入druid-1.1.5.jar文件。

18.1.1 database.properties配置文件

```
#连接设置
driverClassName=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/school
username=root
password=root
#<!-- 初始化连接 -->
initialSize=10
#最大连接数量
maxActive=50
#<!-- 最小空闲连接 -->
minIdle=5
#<!-- 超时等待时间以毫秒为单位 60000毫秒/1000等于60秒 -->
maxWait=5000
```

18.1.2 连接池工具类

```
public class DbUtils {
    //声明连接池对象
    private static DruidDataSource ds;
    static{
        //实例化配置对象
        Properties properties=new Properties();
        try {
            //加载配置文件内容
            properties.load(DbUtils.class.getResourceAsStream("database.properties"));
            ds = (DruidDataSource)DruidDataSourceFactory.createDataSource(properties);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    //获取连接对象
    public static Connection getConnection() {
        try {
            return ds.getConnection();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }
    //释放资源。。
}
```

十九、Apache的DbUtils使用

Commons DbUtils 是Apache组织提供的一个对JDBC进行简单封装的开源工具类库，使用它能简化JDBC应用程序的开发！同时，不会影响程序的性能。

19.1 DbUtils简介

- DbUtils是Java编程中数据库操作实用小工具，小巧、简单、实用
 - 对于数据表的查询操作，可以把结果转换为List、Array、Set等集合。便于操作。
 - 对于数据表的DML操作，也变得很简单(只需要写SQL语句)。

19.1.1 DbUtils主要包含

- ResultSetHandler接口：转换类型接口
 - BeanHandler类：实现类，把一条记录转换成对象
 - BeanListHandler类：实现类，把多条记录转换成List集合。
 - ScalarHandler类：实现类，适合获取一行一列的数据。
- QueryRunner：执行sql语句的类
 - 增、删、改：update();
 - 查询：query();

19.2 DbUtils的使用步骤

- 导入jar包

- mysql连接驱动jar包
- druid-1.1.5.jar
- database.properties配置文件
- commons-dbutils-1.6.jar

19.2.1 DbUtils工具类

```

package com.project.utils;

import com.alibaba.druid.pool.DruidDataSource;
import com.alibaba.druid.pool.DruidDataSourceFactory;

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

/**
 * 连接池工具类
 */
public class DBUtils {
    private static DruidDataSource dataSource;

    static {
        Properties properties = new Properties();
        InputStream is = DBUtils.class.getResourceAsStream("/database.properties");
        try {
            properties.load(is);
            dataSource = (DruidDataSource) DruidDataSourceFactory.createDataSource(properties);
        } catch (IOException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // 返回一个数据源
    public static DataSource getDataSource(){
        return dataSource;
    }
}

```

19.2.2 UserDaoImpl 数据访问对象

```

package com.project.dao.impl;

import com.project.dao.UserDao;
import com.project.entity.User;
import com.project.utils.DBUtils;
import org.apache.commons.dbutils.QueryRunner;
import org.apache.commons.dbutils.handlers.BeanHandler;
import org.apache.commons.dbutils.handlers.BeanListHandler;

import java.sql.SQLException;
import java.util.List;

public class UserDaoImpl implements UserDao {
    //1.创建QueryRunner对象，并传递一个数据源对象
    private QueryRunner queryRunner = new QueryRunner(DBUtils.getDataSource());
    @Override
    public int insert(User user) {
        Object[] params=
        {user.getId(),user.getUsername(),user.getPassword(),user.getSex(),user.getEmail(),user.getAddress()};
        try {
            return queryRunner.update("insert into user (id,username,password,sex,email,address)
values(?, ?, ?, ?, ?, ?)",params);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return 0;
    }

    @Override
    public int update(User user) {
        Object[] params=
        {user.getUsername(),user.getPassword(),user.getSex(),user.getEmail(),user.getAddress(),user.getId()};
        try {
            return queryRunner.update("update user set username=?,password=?,sex=?,email=?,address=? where id =
?",params);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return 0;
    }
}

```

```
    @Override
    public int delete(int id) {
        try {
            return queryRunner.update("delete from user where id = ?", id);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return 0;
    }

    @Override
    public User select(int id) {
        try {
            //把查询到的记录封装成 指定对象
            return queryRunner.query("select * from user where id = ?", new BeanHandler<User>(User.class), id);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }

    /**
     * 查询所有
     * @return
     */
    @Override
    public List<User> selectAll() {
        try {
            return queryRunner.query("select * from user;", new BeanListHandler<User>(User.class));
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }
}
```