JDBC day2

1.ORM思想

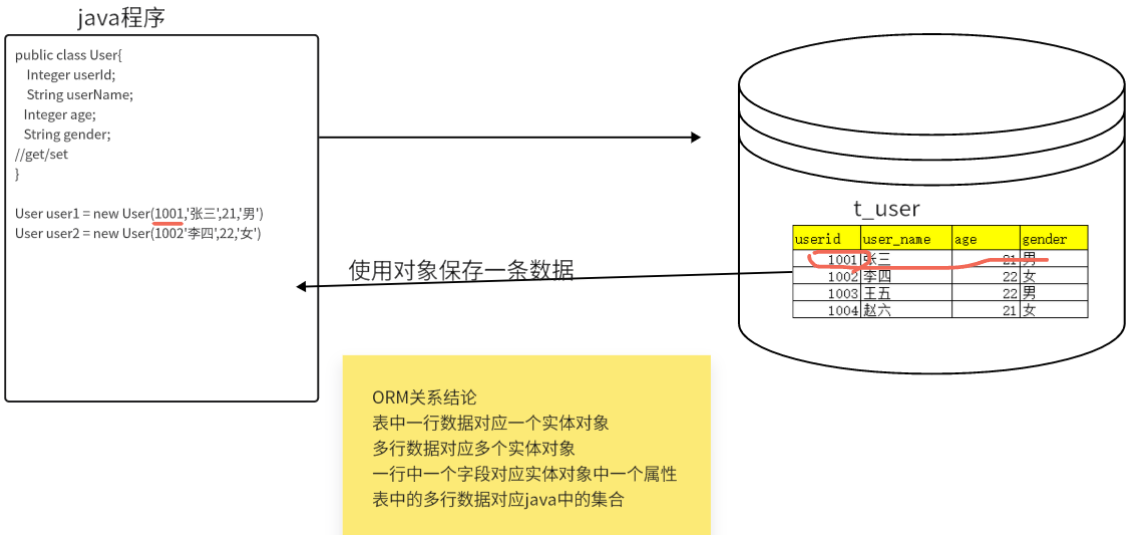
ORM(Object Relational Mapping) 对象关系映射，在对象模型和关系性数据库之间做了一个映射关系，通过改映射关系解决面向对象和关系型数据库之间的匹配，简单来说，通过ORM可以将Java程序中的对象转换为数据库中的一条数据，反之，也可以将数据库中的一条数据转化为对应的java对象。

1.1 ORM解决的问题

一个典型的JAVAEE项目一定要用到提供功能的java代码以及保存数据的数据库，数据在程序运行期间会在java程序和数据库之间进行流转。

解决的问题

1. Java程序中的对象转换为数据库中的一条数据
2. 数据库中的一条数据转换为对应的Java对象



数据在java程序和数据库中的存储和操作的方式存在差异

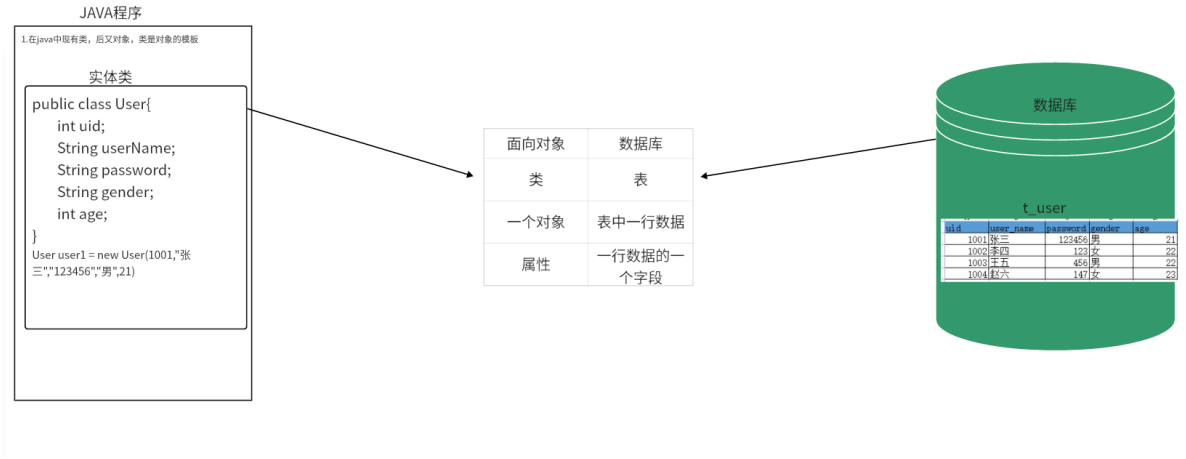
模型	数据格式	操作方法
面向对象的语言（JAVA）	对象，类，属性，	通过方法进行操作
关系型数据库（MYSQL）	行，表，字段	通过SQL语句进行操作

1.2 实现ORM需要解决的问题?

ORM本质要解决数据在Java程序和数据库之间的存储和操作的差异问题，要在java和数据库之间搭建一个桥梁，使Java程序员可以以面向对象的方式操作数据库中的数据，那就需要解决两个问题

1.数据的存储格式在java和数据库之间的转换，

2.数据的操作方式在java和数据库之间的转换。



2.实体类

实体类：用来和数据库中的表对应，解决的是数据格式在Java和数据库之间的转换。

面向对象	数据库
类	表
一个对象	表中一行数据
属性	一行数据的一个字段

实体类的要求

- 一张表对应于一个实体类
- 实体类必须实现Serializable接口
- 表中的一个字段对应实体类中的一个属性
- 属性需要进行封装，尽量不要使用基本数据类型
- 需要提供无参构造方法，
- 实体类类名和表名进行关联， User ----> t_user
- 实体类的属性名和表中字段名进行关联， userName---->user_name, age -------> age
- 开发规范：所有的实体类必须存放在entity包中（bean包，pojo）

示例：

```
DROP TABLE IF EXISTS `t_person`;
CREATE TABLE `t_person`  (
  `p_id` int(10) NOT NULL AUTO_INCREMENT,
  `p_name` varchar(20) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
  `age` int(4) NULL DEFAULT 1,
  `gender` varchar(10) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT
'男',
  `mobile` varchar(20) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT
NULL,
  `address` varchar(50) CHARACTER SET utf8 COLLATE utf8_general_ci NULL DEFAULT
NULL,
  PRIMARY KEY (`p_id`) USING BTREE,
  UNIQUE INDEX `mobile`(`mobile`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 3 CHARACTER SET = utf8 COLLATE =
utf8_general_ci ROW_FORMAT = Compact;
```

实体类代码

```
package com.yunhe.entity;
public class Person {
    private Integer pid;
    private String pName;
    private Integer age;
    private String mobile;
    private String address;
//    get/set

    public Integer getPid() {
        return pid;
    }

    public void setPid(Integer pid) {
        this.pid = pid;
    }

    public String getpName() {
        return pName;
    }

    public void setpName(String pName) {
        this.pName = pName;
    }

    public Integer getAge() {
        return age;
    }

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

    public String getMobile() {
        return mobile;
    }

    public void setMobile(String mobile) {
```

```
        this.mobile = mobile;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }
//    有参无参构造方法

    public Person() {
    }

    public Person(Integer pid, String pName, Integer age, String mobile, String
address) {
        this.pid = pid;
        this.pName = pName;
        this.age = age;
        this.mobile = mobile;
        this.address = address;
    }

    @Override
    public String toString() {
        return "Person{" +
                "pid=" + pid +
                ", pName='" + pName + '\'' +
                ", age=" + age +
                ", mobile='" + mobile + '\'' +
                ", address='" + address + '\'' +
                '}';
    }
}
```

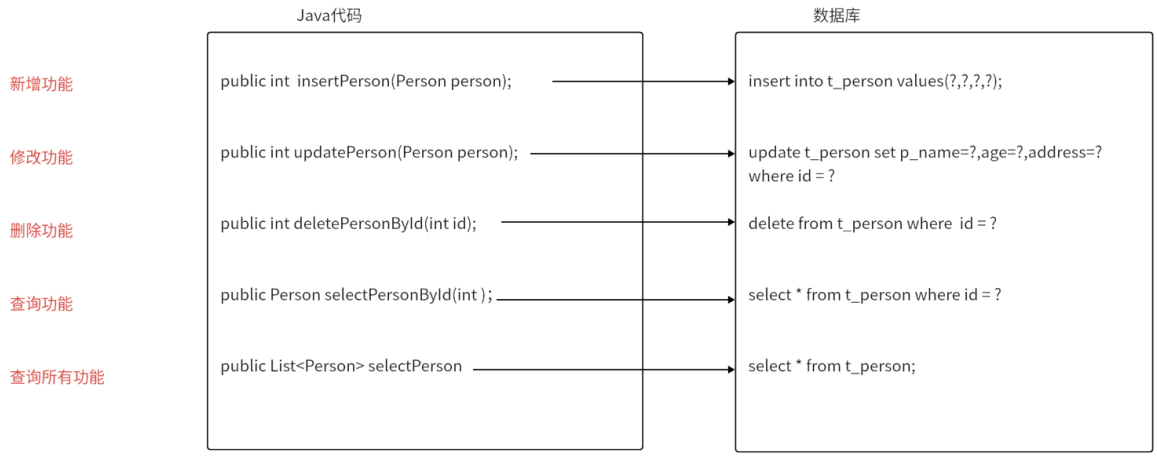
3.DAO模式

3.1 Dao模式分析

DAO(Database access Object) 数据访问对象，DAO模式是用来解决ORM中数据的操作方式在java和数据库直接的转换

操作	Java	数据库
新增	insertXXX(XXX x)	insert into
修改	updateXXX(XXX x)	update 表名 set
删除	deleteXXXById(int id)	delete from 表名 where id = ?
查询单个	selectXXXById(int id)	select * from 表名 where i d= ?
查询多个	List selectXXXAll()	select * from 表名;

对应关系



Dao中声明对一张表的增删改查等操作方法，封装JDBC6步操作，方法的实现是通过JDBC6步代码进行的，从而可以以面向对象的方式调用dao中的方法，操作数据库中对应对数据的操作

模板

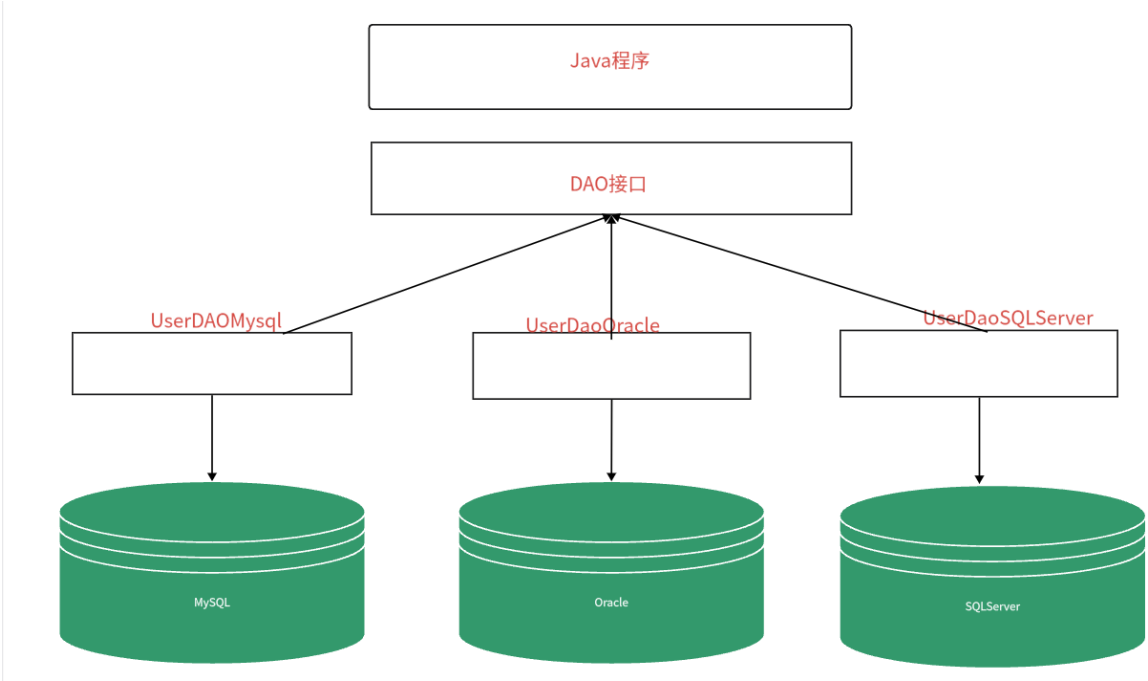
```
class XXXDAO{
    public int insertXXX(XXX x){
        //通过JDBC6步操作完成执行sql  insert into 表名。。。
    }
    public int updateXXX(XXX x){
        //通过JDBC6步操作完成执行SQL  update 表名 set...
    }
    public int deleteXXXById(int id){
        //通过JDBC6步操作完成执行SQL   delete from 表名 。。。。
    }
    public XXX selectXXXById(int id){
        //通过JDBC6步操作完成执行SQL   select * from 表名....
        //通过ORM关系对象模型，将数据封装为java对象
    }
    public List<XXX> selectAllXXX(){
        // 通过JDBC6步操作完成执行SQL  select * from 表名。。。
        // 通过ORM将查询的数据封装为List对象
    }
}

//使用DAO操作数据库
public class XXXDAoTest{
    public static void main(String[] args){
        XXXDao dao = new XXXDao();
        XXX x = new XXX();
        //添加动作
        dao.insertXXX(x);
        //查询
        XXX x=(XXX)dao.selectXXXById(1101);

    }
}
```

3.2 DAO模式的要求

一个表对应一个dao,但是dao一般不会只定义一个类，而是定义成对应的接口，以及一个接口对应多个实现类



Dao层分为接口成和实现层好处：接口定义标准，可以使用不同的实现类实现对不同数据库的操作，提高了程序的可扩展性。

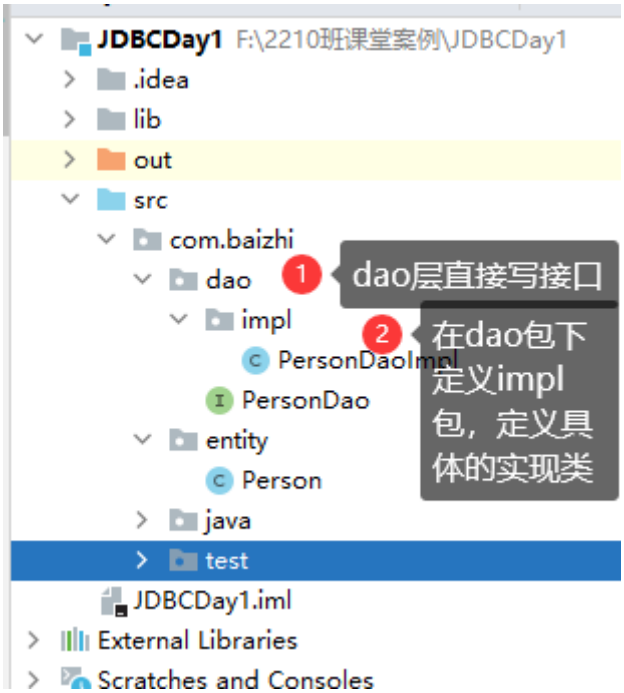
dao接口的要求：

- 一个表一个接口
- 所有的接口都在dao包中
- Dao接口中定义对该表的增删改查操作方法
- 接口名和表名关联 ： t_person ------> PersonDao

dao实现类的要求：

- Dao实现类实现DAO接口，提供接口中每个方法的实现
- 方法实现： JDBC6步
- Dao实现类类名 = 接口名+impl: PersonDao --------> PersonDaoImpl
- 所有的实现类必须放在dao包下的impl子包中， dao.impl

项目结果如下



3.3 DAO模式案例

案例代码：

DAO接口

```
package com.baizhi.dao;

import com.baizhi.entity.Person;
import java.sql.SQLException;
import java.util.List;

//一个接口对应一张表
public interface PersonDao {
//    接口中定义对表中数据的操作方法
//    1.添加
    public int insertPerson(Person person) throws ClassNotFoundException,
SQLException;
//    2.修改
    public int updatePerson(Person person);
//    3.删除
    public int deletePersonById(int id);
//    4.根据id进行查询
    public Person selectPersonById(int id) throws ClassNotFoundException,
SQLException;
//    5.查询所有操作
    public List<Person> selectAllPerson() throws SQLException,
ClassNotFoundException;
}
```

DAO实现类

```
package com.baizhi.dao.impl;

import com.baizhi.dao.PersonDao;
import com.baizhi.entity.Person;
```

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class PersonDaoImpl implements PersonDao {
    @Override
    public int insertPerson(Person person) throws ClassNotFoundException,
SQLException {
//        1.加载驱动
        Class.forName("com.mysql.jdbc.Driver");
//        2.创建链接
        String url = "jdbc:mysql://localhost:3306/java2210?
useUnicode=true&characterEncoding=utf-
8&useSSL=false&serverTimeZone=Asia/shanghai";
        String username="root";
        String password="root";
        Connection conn = DriverManager.getConnection(url,username,password);
//        3.编写sql
        String sql = "insert into t_person values(null,?,?,?,?,?)";
        PreparedStatement pstm = conn.prepareStatement(sql);
//        设置参数
        pstm.setString(1,person.getpName());
        pstm.setInt(2,person.getAge());
        pstm.setString(3,person.getGender());
        pstm.setString(4,person.getMobile());
        pstm.setString(5,person.getAddress());
//        4.发送sql
        int i = pstm.executeUpdate();
//       6.关闭链接
        pstm.close();
        conn.close();
        return i;
    }

    @Override
    public int updatePerson(Person person) {
        return 0;
    }

    @Override
    public int deletePersonById(int id) {
        return 0;
    }

    @Override
    public Person selectPersonById(int id) throws ClassNotFoundException,
SQLException {
//       1.加载驱动
        Class.forName("com.mysql.jdbc.Driver");
//        2.获取链接
        String url = "jdbc:mysql://localhost:3306/java2210?
useUnicode=true&characterEncoding=utf-
8&useSSL=false&serverTimeZone=Asia/shanghai";
        String username="root";
        String password="root";
        Connection conn = DriverManager.getConnection(url,username,password);
//      3.编写sql，并获取执行sql语句对象、
```

```
        String sql = "select * from t_person where p_id =?";
        PreparedStatement pstm = conn.prepareStatement(sql);
//        绑定参数
        pstm.setInt(1,id);
//        4.发送sql
        ResultSet rs = pstm.executeQuery();
//        5.处理结果集
        Person person = null;
        while (rs.next()){
//            如果结果集中有数据，则创建一个对象，通过将结果集中的数据封装到该对象中
            person = new Person();
//            1.获取数据
           int pid =  rs.getInt("p_id");
           String pname = rs.getString("p_name");
           int age = rs.getInt("age");
           String gender = rs.getString("gender");
           String mobile = rs.getString("mobile");
           String address = rs.getString("address");
//            2.封装数据
            person.setPid(pid);
            person.setpName(pname);
            person.setAge(age);
            person.setGender(gender);
            person.setMobile(mobile);
            person.setAddress(address);
        }
//        6.关闭链接
        rs.close();
        pstm.close();
        conn.close();
        return person;
    }

    @Override
    public List<Person> selectAllPerson() throws SQLException,
ClassNotFoundException {
//      声明一个list集合，用来返回
        List<Person> list = new ArrayList<>();
        //                1.加载驱动
        Class.forName("com.mysql.jdbc.Driver");
//        2.获取链接
        String url = "jdbc:mysql://localhost:3306/java2210?
useUnicode=true&characterEncoding=utf-
8&useSSL=false&serverTimeZone=Asia/shanghai";
        String username="root";
        String password="root";
        Connection conn = DriverManager.getConnection(url,username,password);
//      3.编写sql，并获取执行sql语句对象、
        String sql = "select * from t_person ";
        PreparedStatement pstm = conn.prepareStatement(sql);

//        4.发送sql
        ResultSet rs = pstm.executeQuery();
//        5.处理结果集
        Person person = null;
        while (rs.next()){
//            如果结果集中有数据，则创建一个对象，通过将结果集中的数据封装到该对象中
            person = new Person();
```

```
//            1.获取数据
            int pid =  rs.getInt("p_id");
            String pname = rs.getString("p_name");
            int age = rs.getInt("age");
            String gender = rs.getString("gender");
            String mobile = rs.getString("mobile");
            String address = rs.getString("address");
//            2.封装数据
            person.setPid(pid);
            person.setpName(pname);
            person.setAge(age);
            person.setGender(gender);
            person.setMobile(mobile);
            person.setAddress(address);
//            3.将封装好的数据存放到list集合中
            list.add(person);
        }
//        6.关闭链接
        rs.close();
        pstm.close();
        conn.close();

        return list;
    }
}
```

4.JDBC工具类第三版

配置文件

```
driverClass=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/java2210?useUnicode=true&characterEncoding=utf-8&useSSL=false&serverTimeZone=Asia/shanghai
user=root
password=root
```

代码案例

```
package com.baizhi.utils;

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Properties;

public class JdbcUtils2 {
//    声明属性对象
    static Properties properties = new Properties();
    public static String driverClass = null;
    public static String url = null;
    public static String user = null;
```

```
    public static String password = null;
//    使用静态代码块读取配置文件的内容
    static{
//            通过IO流读取配置文件
//        通过属性对象加载配置文件
        try {
//            InputStream input = new FileInputStream("jdbc.properties");
            InputStream input =
JdbcUtils2.class.getResourceAsStream("jdbc.properties");
            properties.load(input);
            driverClass = properties.getProperty("driverClass");
            url = properties.getProperty("url");
            user = properties.getProperty("user");
            password = properties.getProperty("password");
        } catch (IOException e) {
            e.printStackTrace();
        }
   }
   public static Connection getConnection(){
        Connection conn = null;
        try{
//            加载驱动
            Class.forName(driverClass);
//            获取链接
            conn = DriverManager.getConnection(url,user,password);
        }catch (Exception e){
            e.printStackTrace();
        }
        return conn;
   }
    public static void close(Connection conn, PreparedStatement pstm, ResultSet
rs){
        try{
//            在关闭之前判断，该对象是否为空
            if(rs!=null){
                rs.close();
            }
//            不要将三个关闭动作写作同一个trycatch中，否则等前面的出现异常，后面的资源就无法
关闭

        }catch (Exception e){
            e.printStackTrace();
        }
        try{
            if(pstm!=null){
                pstm.close();
            }
        }catch (Exception e){
            e.printStackTrace();
        }
        try{
            if(conn!=null){
                conn.close();
            }
        }catch (Exception e){
            e.printStackTrace();
        }
    }
```

```
    /**
     * close方法的重载，为了方便处理不同的关闭情况，两个参数的close方法主要用来关闭增删改操作
     * @param conn
     * @param pstm
     */
    public static void close(Connection conn, PreparedStatement pstm){

        try{
            if(pstm!=null){
                pstm.close();
            }
        }catch (Exception e){
            e.printStackTrace();
        }
        try{
            if(conn!=null){
                conn.close();
            }
        }catch (Exception e){
            e.printStackTrace();
        }

    }
}
```

工具类使用

```
 @Override
    public List<Student> selectAllStudent() {
        Connection conn = null;
        PreparedStatement pstm = null;
        ResultSet rs = null;
        List<Student> list = new ArrayList<>();
        try{
            //        1.获取链接
            conn = JdbcUtils2.getConnection();
//            2.编写sql
            String sql = "select * from t_student";
            pstm = conn.prepareStatement(sql);
//            3.发送sql
            rs = pstm.executeQuery();
//            处理结果集
            Student stu = null;
            while (rs.next()){
//                如果下一行有数据，创建对象，
                stu = new Student();
//                将数据封装到对象中
                stu.setStuId(rs.getInt("stu_id"));
                stu.setStuName(rs.getString("stu_name"));
                stu.setAge(rs.getInt("age"));
                stu.setGender(rs.getString("gender"));
                stu.setPhoneNumber(rs.getString("phone_number"));
                stu.setAddress(rs.getString("address"));
                stu.setBirthday(rs.getString("birthday"));
                stu.setClassId(rs.getInt("class_id"));
```

```
//                将对象添加到list集合中
                list.add(stu);
            }
        }catch (Exception e){
            e.printStackTrace();
        }finally {
            JdbcUtils2.close(conn,pstm,rs);
        }
        return list;
    }
```

工具类注意事项，大家只需要最后掌握第三版即可，前两版理解抽取思路即可