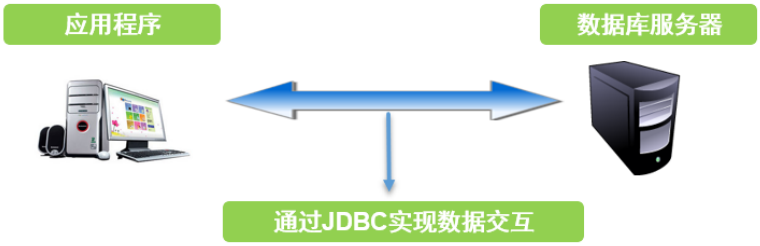
JDBC介绍

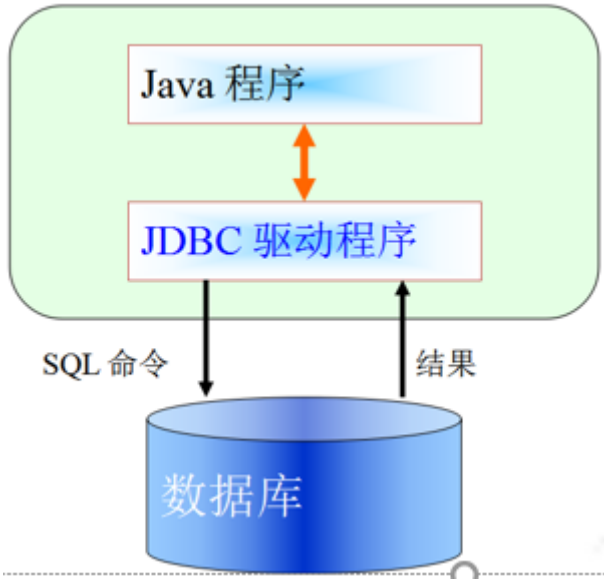
提问

- 在之前的程序中，数据都是预先写好的，如何实现数据的动态变化?

- JDBC
  - Java数据库连接技术(Java DataBase Connectivity)



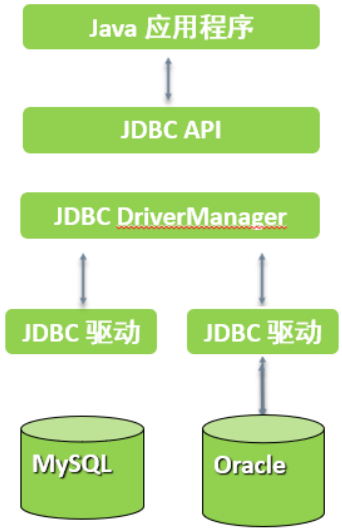
- 数据库访问技术简介:
  - JDBC（Java DataBase Connectivity）是由Sun Microsystem公司提供的API（Application Programming Interface应用程序编程接口）；它为Java应用程序提供了一系列的类，使其能够快速高效地访问数据库；这些功能是由一系列的类和对象来完成的，我们只需使用相关的对象，即可完成对数据库的操作



JDBC的工作原理

JDBC的工作原理

- JDBC的内容
  - JDBC API
    - 定义了一系列的接口和类，集成在java.sql和javax.sql包中
  - DriverManager
    - 管理各种不同的JDBC驱动
  - JDBC 驱动
    - 负责连接不同类型的数据库



JDBC访问数据库步骤

JDBC访问数据库的步骤

- JDBC访问数据库步骤

步骤1：加载驱动

- 使用Class类的forName方法，将驱动程序类加载到JVM（Java虚拟机）中；

方 法 原 型	说　　明
static Class forName(String className) throws ClassNotFoundException	将由className指定完整名称的类加载到JVM中，如果加载失败，将抛出异常，必须捕捉

语法　Class.forName("JDBC驱动类名称");

示例：Class.forName("com.mysql.jdbc.Driver");

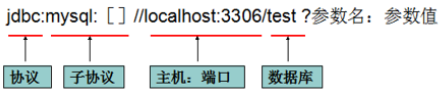
步骤2：获取数据库连接

- 成功加载驱动后，必须使用DriverManager类的静态方法getConnection来获得连接对象；

方 法 原 型	说　　明
static Connection getConnection (String url, String user, String password) throws SQLException	参数url是连接字符串，参数user是数据库用户名，参数password是登录口令，成功连接到数据库返回Connection对象，连接失败则抛出SQLException异常，必须捕捉

语法
- 示例：Co

Connection conn=DriverManager.getConnection(数据库URL,数据库用户名,密码)
注意：在加载驱动及获取连接过程中，可能会出现异常，因此需要注意进行异常处理



步骤3：创建Statement执行SQL语句

- 通过Connection对象创建

方 法 原 型	说　　明
Statement createStatement() throws SQLException	成功创建返回Statement对象，否则抛出SQLException异常，必须捕捉

语法　Statement stmt=conn.createStatement();

- 用于执行SQL语句

方法	说明
ResultSet executeQuery(String sql)	执行SQL查询并获取到ResultSet对象
int executeUpdate(String sql)	可以执行插入、删除、更新等操作，返回值是执行该操作所影响的行数
void close()	关闭Statement对象

步骤4：处理ResultSet结果集

- 用于存储查询结果
  - 只在执行select语句时返回

方法	说明
boolean next()	将光标从当前位置向下移动一行
void close()	关闭ResultSet 对象
String getString(String colLabel)	根据列名称对应的值
String getString(int colIndex))	根据列的位置获取对应的值
int getInt(int colIndex)	根据列的位置获取对应值
int getInt(String colLabel)	根据列的名称获取对应的值

步骤5：释放资源 close();

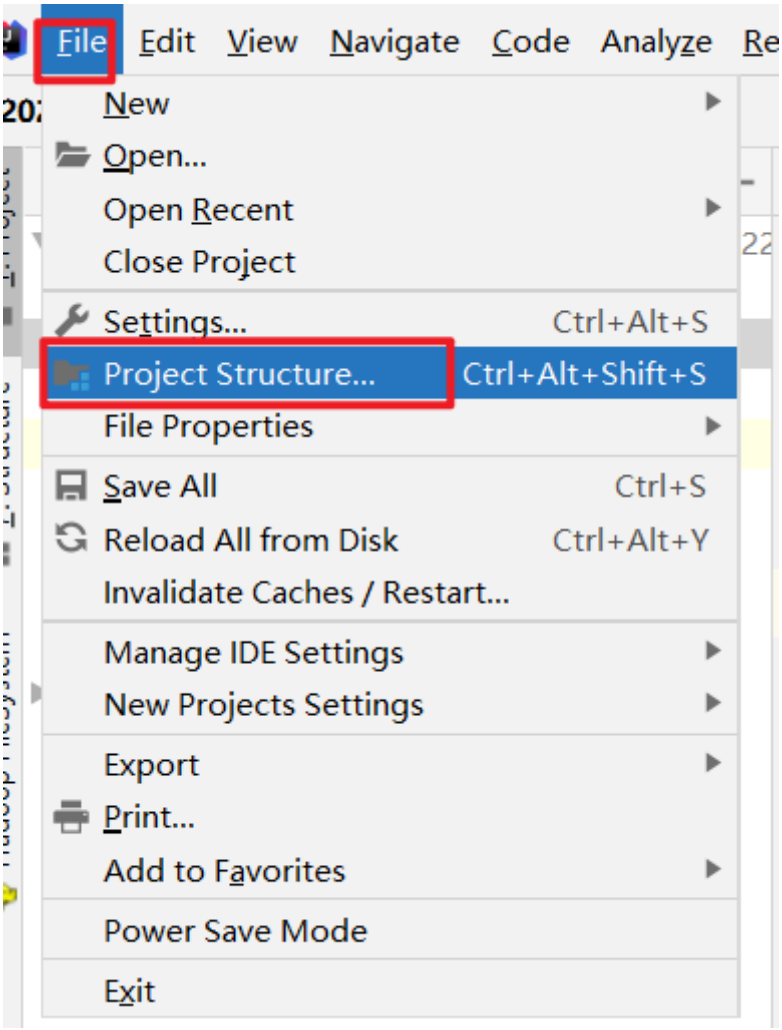
案例

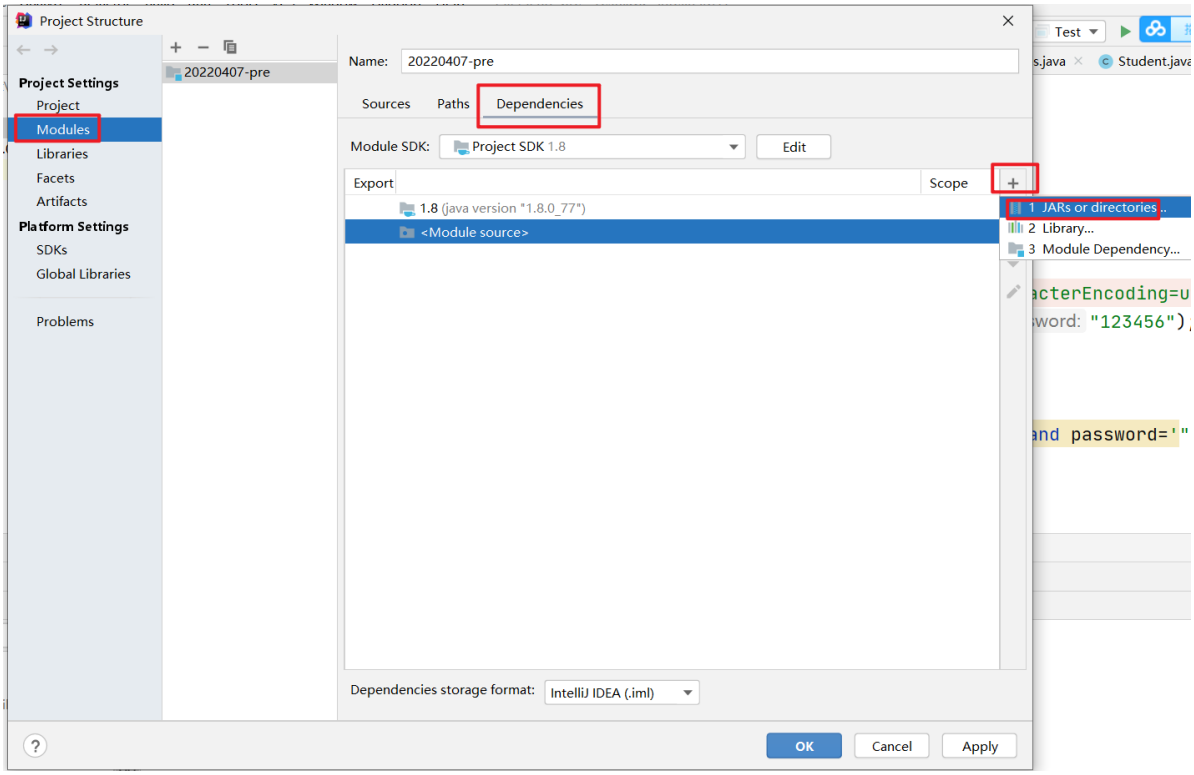
User定义

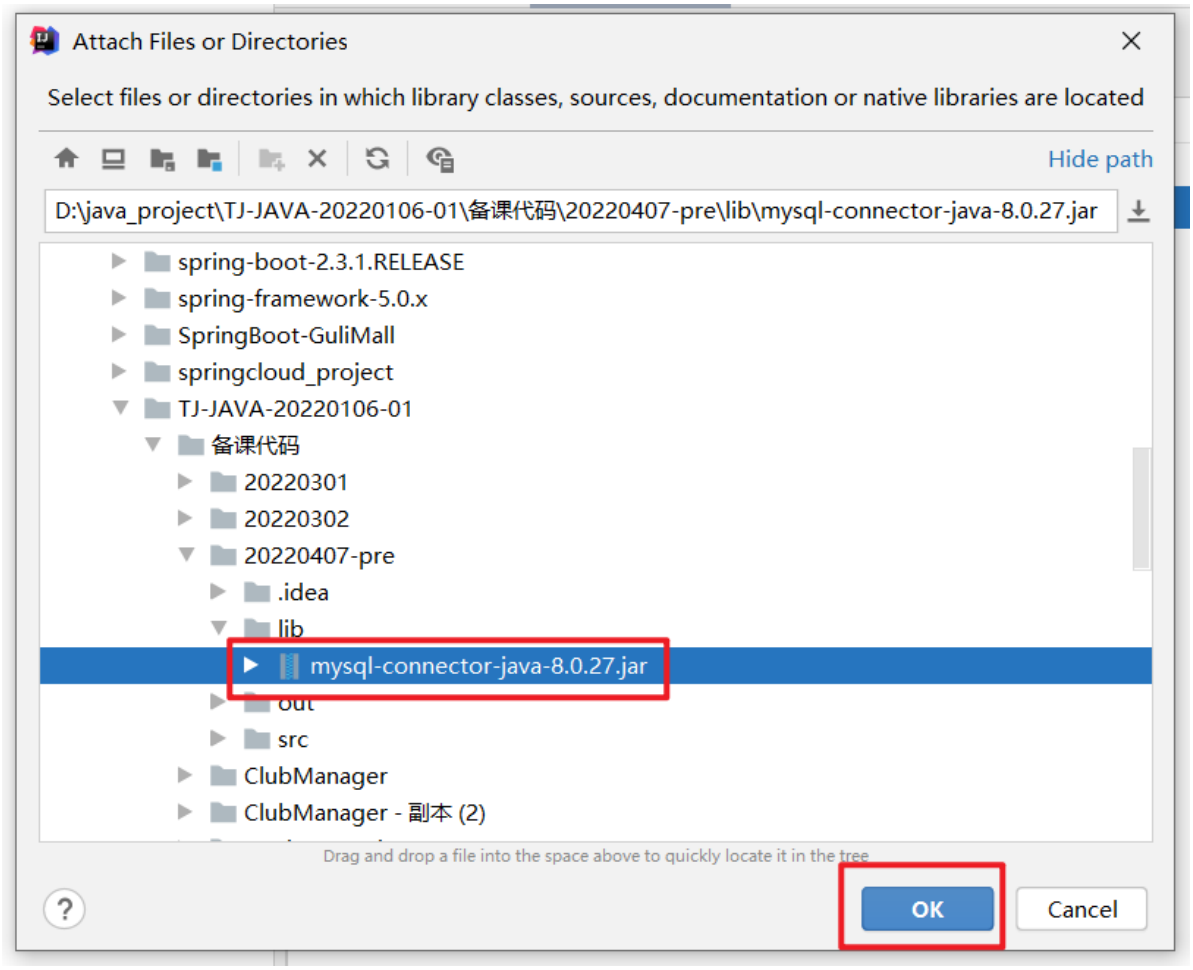
```
DROP TABLE IF EXISTS `user`;
CREATE TABLE `user` (
  `id` bigint NOT NULL AUTO_INCREMENT,
  `username` varchar(255) NOT NULL,
  `password` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of user
-- ----------------------------
INSERT INTO `user` VALUES ('1', 'jim', '11111');
INSERT INTO `user` VALUES ('2', 'tom', '22222');
INSERT INTO `user` VALUES ('3', 'lucy', '3333');
INSERT INTO `user` VALUES ('4', 'laowang', '4444');
INSERT INTO `user` VALUES ('5', 'laoli', '5555');
```

使用Statement查询User

引入MySql驱动jar


File
Edit
View
Navigate
Code
Analyze
New
Open...
Open Recent
Close Project
Settings... Ctrl+Alt+S
Project Structure... Ctrl+Alt+Shift+S
File Properties
Save All Ctrl+S
Reload All from Disk Ctrl+Alt+Y
Invalidate Caches / Restart...
Manage IDE Settings
New Projects Settings
Export
Print...
Add to Favorites
Power Save Mode
Exit


Project Structure
Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Libraries
Problems
20220407-pre
Name: 20220407-pre
Sources
Paths
Dependencies
Module SDK: Project SDK 1.8
Edit
Export
Scope
1.8 (java version "1.8.0_77")
<Module source>
Dependencies storage format: IntelliJ IDEA (.iml)
OK
Cancel
Apply
1 JARs or directories...
2 Library...
3 Module Dependency...
Test
Student.java




```java
/**
 * user表的实体类
 */
public class User {
    /**
     * 主键
     */
    private Long id;
    /**
     * 用户名
     */
    private String username;
    /**
     * 密码
     */
    private String password;

    public User() {
    }

    public User(String username, String password) {
        this.username = username;
        this.password = password;
    }

    public int getId() {
        return id;
    }
```

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

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

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

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

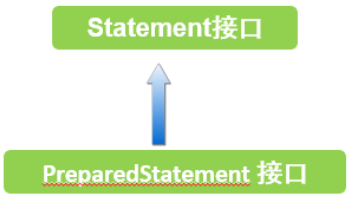
```
 /*
    * statement 只能拼接sql语句，可能会出现sql注入的风险（例如sql注入 用户名输入    ' or
1=1 #  密码随便输入，就会查出数据），不推荐用，
    * */
    public User query(String username, String password) {

        try {
            //1、加载驱动
            //加载mysql8.0+的驱动
            Class.forName("com.mysql.cj.jdbc.Driver");
            //Class.forName("com.mysql.jdbc.Driver");
            //2、获取连接
            String url = "jdbc:mysql://localhost:3306/test?
useUnicode=true&characterEncoding=utf8&useSSL=false&serverTimezone=Asia/Shanghai"
;
            Connection conn = DriverManager.getConnection(url, "root",
"123456");
            //3、获取Statement对象
            Statement statement = conn.createStatement();
            //4、执行sql返回ResultSet
            String sql = "select * from user where username='" + username + "'
and password='" + password + "'";
            ResultSet rs = statement.executeQuery(sql);
```

```
            User user=null;
            if (rs.next()) {
                int id = (Integer) rs.getObject(1);
                String username1 = (String) rs.getObject(2);
                String password1 = (String) rs.getObject(3);

                user = new User();
                user.setId(id);
                user.setUsername(username1);
                user.setPassword(password1);
            }

            //5、关闭资源
            rs.close();
            statement.close();
            conn.close();
            return user;
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }
```

JDBC的PreparedStatement

PreparedStatement概述

- PreparedStatement概述
  - 继承自Statement接口
  - 能够对SQL语句进行预编译
- PreparedStatement的优势
  - 提高SQL语句执行效率
  - 提高安全性



- SQL语句使用"?"作为数据占位符
- 在创建时对SQL语句进行预编译
- 使用setXxx()方法设置数据

使用PreparedStatement查询

```
public User query(String username, String password) {
    try {
        //1、加载驱动
```

```
        //加载mysql8.0+的驱动
        Class.forName("com.mysql.cj.jdbc.Driver");
        //Class.forName("com.mysql.jdbc.Driver");
        //2、获取连接
        String url = "jdbc:mysql://localhost:3306/test?
useUnicode=true&characterEncoding=utf8&useSSL=false&serverTimezone=Asia/Shanghai"
;
        Connection conn = DriverManager.getConnection(url, "root", "123456");
        //3、获取PreparedStatement
        String sql = "select * from user where username= ? and password= ?";
        PreparedStatement ps = conn.prepareStatement(sql);
        ps.setObject(1,username);
        ps.setObject(2,password);
        //4、执行sql返回ResultSet
        ResultSet rs = ps.executeQuery();
        User user =null;
        if (rs.next()) {
            int id = (Integer) rs.getObject(1);
            String username1 = (String) rs.getObject(2);
            String password1 = (String) rs.getObject(3);

            user = new User();
            user.setId(id);
            user.setUsername(username1);
            user.setPassword(password1);
        }

        //5、关闭资源
        rs.close();
        ps.close();
        conn.close();
        return user;
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return null;
}
```

新增

```
/**
 * 新增
 *
 * @throws ClassNotFoundException
 * @throws SQLException
 */
private static void insert() throws ClassNotFoundException, SQLException {
    //1、加载jdbc驱动
    Class.forName("com.mysql.cj.jdbc.Driver");
    //2、获取连接
    //在mysql8.0的数据库 serverTimezone=Asia/shanghai是必须的参数
```

```
    String url = "jdbc:mysql://localhost:3306/db1?
useUnicode=true&characterEncoding=utf8&useSSL=false&serverTimezone=Asia/Shanghai"
;
    Connection conn = DriverManager.getConnection(url, "root", "123456");
    //3、获取PreparedStatement对象
    //注意点：sql语句中使用？作为参数的占位符
    String sql = "insert into `user`(username,password) values(?,?)";
    PreparedStatement ps =
conn.prepareStatement(sql,Statement.RETURN_GENERATED_KEYS);
    //把问号替换成参数
    ps.setObject(1, "jim");
    ps.setObject(2, 30);
    //4、执行sql，新增 要使用executeUpdate()
    ps.executeUpdate();
    // 执行此 Statement 对象而创建的所有自动生成的键
    ResultSet rs = ps.getGeneratedKeys();
    if (rs.next()) {
        // 指定返回生成的主键
        Object id = rs.getObject(1);
        System.out.println("新增的主键为："+id);
    }
    //5、关闭资源
    ps.close();
    conn.close();
}
```

修改和删除

```
/**
 * 更新
 *
 * @throws ClassNotFoundException
 * @throws SQLException
 */
private static void update() throws ClassNotFoundException, SQLException {
    //1、加载jdbc驱动
    Class.forName("com.mysql.cj.jdbc.Driver");
    //2、获取连接
    //在mysql8.0的数据库 serverTimezone=Asia/shanghai是必须的参数
    String url = "jdbc:mysql://localhost:3306/db1?
useUnicode=true&characterEncoding=utf8&useSSL=false&serverTimezone=Asia/Shanghai"
;
    Connection conn = DriverManager.getConnection(url, "root", "123456");
    //3、获取PreparedStatement对象
    //注意点：sql语句中使用？作为参数的占位符
    String sql = "update user set password=? where id = ?";
    PreparedStatement ps = conn.prepareStatement(sql);
    //把问号替换成参数
    ps.setObject(1, "888");
    ps.setObject(2, 1);
    //4、执行sql，更新或者删除 要使用executeUpdate()，处理返回值，返回值代表的含义：影响的行
数
    int result = ps.executeUpdate();
    if (result > 0) {
```

```
            System.out.println("更新密码成功");
        } else {
            System.out.println("更新密码失败");
        }
        //5、关闭资源
        ps.close();
        conn.close();
    }
```

封装DbUtils工具类

```
package com.tjetc.jdbc;

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

public class DbUtils {
    private static String driverName;
    private static String url;
    private static String username;
    private static String password;

    static {
        InputStream inputStream = null;
        try {
            //创建db.properties的流
            inputStream =
DbUtils.class.getClassLoader().getResourceAsStream("db.properties");
            //创建Properties对象
            Properties p = new Properties();
            //把数据流读入Properties对象中
            p.load(inputStream);
            //从Properties对象中获取配置数据
            url = p.getProperty("url");
            username = p.getProperty("username");
            password = p.getProperty("password");
            driverName = p.getProperty("driverName");
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (inputStream != null) {
                    inputStream.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    /**
     * insert语句使用，返回新增数据的自增主键。
```

```
     *
     * @param sql
     * @return
     */
    public static Object insert(Connection connection, String sql, Object[]
params) throws SQLException {
        PreparedStatement ps = null;
        Object id = null;
        try {
            //创建PreparedStatement
            ps = connection.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS);
            //设置参数
            setPreparedStatementParam(ps, params);
            //执行sql
            ps.executeUpdate();
            // 执行此 Statement 对象而创建的所有自动生成的键
            ResultSet rs = ps.getGeneratedKeys();
            if (rs.next()) {
                // 指定返回生成的主键
                id = rs.getObject(1);
            }
        } finally {
            close(ps);
        }
        return id;
    }

    /**
     * insert语句使用，返回新增数据的自增主键。
     *
     * @param sql
     * @return
     */
    public static Object insert(String sql, Object[] params) throws
SQLException, ClassNotFoundException {
        Connection conn = null;
        Object id;
        try {
            //创建连接
            conn = getConnection();
            id = insert(conn, sql, params);
        } finally {
            close(conn);
        }
        return id;
    }

    /**
     * 更新、删除
     *
     * @param sql
     * @param params
     * @return
     * @throws SQLException
```

```
     */
    public static boolean update(Connection connection, String sql, Object[]
params) throws SQLException {
        PreparedStatement ps = null;
        try {
            //步骤2：设置SQL语句以及对应的参数
            ps = connection.prepareStatement(sql);
            setPreparedStatementParam(ps, params);
            //步骤3：执行update
            int result = ps.executeUpdate();
            //返回执行的结果
            return result > 0 ? true : false;
        } finally {
            //步骤4：关闭资源
            close(ps);
        }
    }

    /**
     * 更新、删除
     *
     * @param sql
     * @param params
     * @return
     * @throws SQLException
     */
    public static boolean update(String sql, Object[] params) throws
SQLException, ClassNotFoundException {
        Connection connection = null;
        try {
            //步骤1：获取链接
            connection = getConnection();
            return update(connection, sql, params);
        } finally {
            //步骤2：关闭连接资源
            close(connection);
        }
    }

    /**
     * 查询一个
     *
     * @param sql
     * @param params
     * @return
     * @throws SQLException
     */
    public static Map<String, Object> selectOne(Connection connection, String
sql, Object[] params) throws SQLException {
        PreparedStatement ps = null;
        ResultSet rs = null;
        try {
            //步骤2：设置SQL语句以及对应的参数
            ps = connection.prepareStatement(sql);
            setPreparedStatementParam(ps, params);
```

```
            //步骤3：执行查询，把查询结果的列作为key，列对应的值作为value，保存到Map中
            rs = ps.executeQuery();

            if (rs.next()) {
                return getResultMap(rs);
            }
        } finally {
            //步骤4：关闭资源
            close(rs, ps);
        }
        return null;
    }

    /**
     * 获取ResultMap
     *
     * @param rs
     * @return
     * @throws SQLException
     */
    private static Map<String, Object> getResultMap(ResultSet rs) throws
SQLException {
        //获取到result的元数据，包含了列的信息
        ResultSetMetaData metaData = rs.getMetaData();

        //获取到当前表的所有的列的列数
        int columnCount = metaData.getColumnCount();

        //存储数据库列与值的map
        Map<String, Object> map = new HashMap<>();

        //根据列的数量，获取到每一个列的列名以及对应的值
        for (int i = 0; i < columnCount; i++) {
            //能够获取到每一个列的名称，参数是每个列的序号值
            String columnLabel = metaData.getColumnLabel(i + 1);
            Object columnValue = rs.getObject(columnLabel);
            map.put(columnLabel, columnValue);

        }
        return map;
    }

    /**
     * 查询一个
     *
     * @param sql
     * @param params
     * @return
     * @throws SQLException
     */
    public static Map<String, Object> selectOne(String sql, Object[] params)
throws SQLException, ClassNotFoundException {
        Connection connection = null;
        try {
```

```
            //步骤1：获取链接
            connection = getConnection();
            return selectOne(connection, sql, params);
        } finally {
            //步骤4：关闭资源
            close(connection);
        }
    }

    /**
     * 查询集合
     *
     * @param sql
     * @param params
     * @return
     */
    public static List<Map<String, Object>> selectList(Connection connection,
String sql, Object[] params) throws SQLException {
        PreparedStatement ps = null;
        ResultSet rs = null;
        List<Map<String, Object>> list;
        try {
            //步骤2：设置SQL语句以及对应的参数
            ps = connection.prepareStatement(sql);
            setPreparedStatementParam(ps, params);

            //步骤3：执行查询，把查询结果的列作为key，列对应的值作为value，保存到Map中
            rs = ps.executeQuery();
            list = new ArrayList<>();

            while (rs.next()) {
                list.add(getResultMap(rs));
            }
        } finally {
            //步骤4：关闭资源
            close(rs, ps);
        }
        return list;
    }

    /**
     * 查询集合
     *
     * @param sql
     * @param params
     * @return
     */
    public static List<Map<String, Object>> selectList(String sql, Object[]
params) throws SQLException, ClassNotFoundException {
        Connection connection = null;
        try {
            connection = getConnection();
            return selectList(connection, sql, params);
        } finally {
            close(connection);
```

```
        }
    }

    /**
     * 设置参数
     *
     * @param ps
     * @param params
     * @throws SQLException
     */
    private static void setPreparedStatementParam(PreparedStatement ps, Object[]
params) throws SQLException {
        if (params != null && params.length > 0) {
            for (int i = 0; i < params.length; i++) {
                ps.setObject(i + 1, params[i]);
            }
        }
    }

    /**
     * 获取连接
     *
     * @return
     * @throws SQLException
     */
    public static Connection getConnection() throws ClassNotFoundException,
SQLException {
        //加载数据库驱动
        Class.forName(driverName);
        return DriverManager.getConnection(url, username, password);
    }

    //开启事务
    public static void beginTransaction(Connection conn) throws SQLException {
        conn.setAutoCommit(false);
    }

    //提交事务
    public static void commit(Connection conn) throws SQLException {
        conn.commit();
    }

    //回滚事务
    public static void rollback(Connection conn) throws SQLException {
        conn.rollback();
    }

    public static void close(Connection connection) {
        if (connection != null) {
            try {
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
```

```
    }


    /**
     * 释放PreparedStatement,Connection
     */
    private static void close(PreparedStatement ps, Connection connection) {
        close(ps);
        close(connection);
    }

    /**
     * 释放ResultSet,PreparedStatement
     */
    private static void close(ResultSet rs, PreparedStatement ps) {
        close(rs);
        close(ps);
    }

    /**
     * 释放ResultSet,PreparedStatement,Connection
     */
    private static void close(ResultSet rs, PreparedStatement ps, Connection
conn) {
        close(rs);
        close(ps);
        close(conn);
    }


    private static void close(Statement statement) {
        if (statement != null) {
            try {
                statement.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    private static void close(PreparedStatement statement) {
        if (statement != null) {
            try {
                statement.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    private static void close(ResultSet rs) {
        if (rs != null) {
            try {
                rs.close();
            } catch (SQLException e) {
```

```
                e.printStackTrace();
            }
        }
    }
}
```