1.PreparedStatement预编译的执行对象.是statement的子接口.

1.1:PreparedStatement配合占位传参方法一起使用,可以有效的防止Sql注入.

1.2:PreparedStatement比statement执行效率和灵活性更高.

方法名	说　明
ResultSet executeQuery()	执行预编译的SQL查询并获取到ResultSet对象
int executeUpdate()	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
boolean execute()	可以执行任意SQL语句，然后获得一个布尔值，表示是否返回ResultSet
void close()	关闭Statement对象
int[ ] executeBatch()	执行批处理,返回影响的行数
void addBatch()	将sql语句添加批处理中
void set数据类型(占位符，参数)	将参数传递到sql语句对应占位符的位置

```
eg:public static void main(String[] args) throws SQLException {
        Scanner input=new Scanner(System.in);
        /*从键盘上接收登录信息*/
        System.out.println("请输入登录名:");
        String loginName=input.next();
        System.out.println("请输入登录密码:");
        String loginPwd=input.next();

        //声明连接对象,执行对象,结果集对象
        Connection conn=null;
        PreparedStatement state=null;
        ResultSet rs=null;

        try {
            //1.加载驱动
            Class.forName("com.mysql.jdbc.Driver");
            //2.创建连接对象
```

```
conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/myschool",
"root","root");
                //3.准备Sql语句,?表示占位符,占位符从1开始计数的
                String sql="select uid,uname,age,address,upassword "
                        +" from t_user where uname=? and upassword=?";
                System.out.println("***"+sql);
                //4.创建预编译执行对象,将sql语句编译好
                state=conn.prepareStatement(sql);
                //向sql语句中占位符的位置上传参数,第一个参数是占位符的位置,
第二个参数是要传的参数
                state.setString(1, loginName);
                state.setString(2, loginPwd);

                //5.用执行对象调用相应的方法将Sql语句传到数据库中去执行,并
得到返回结果
                rs=state.executeQuery();
                //6.处理结果
                //读取结果集中第一行,如果第一行有数据说明登录成功,没有数据
说明登录失败
                if (rs.next()) {
                    System.out.println("登录成功");
                }else {
                    System.out.println("登录失败");
                }

            } catch (Exception e) {
                e.printStackTrace();
            }finally {
                //7.关闭对象,释放资源,先开后关
                if (rs!=null) {
                    rs.close();
                }
                if (state!=null) {
                    state.close();
```

```
                }
                if (conn!=null) {
                        conn.close();
                }
        }
    }
```

2.jdbc工具类:

2.1:为了调用方法:工具类中方法一般声明为静态方法,又因为静态方法中只能直接调用 静态变量,所以工具类中成员变量声明为静态变量.

2.2:将固定步骤封装到方法中,将变量数据作方法的参数或返回值.

一般情况下,将执行功能的所需要的数据封装成方法参数.

方法执行的结果作为方法的返回值.

eg:

```
/**
 * jdbc工具类
 * @author sx
 * @version 2020年2月14日
 */
public class DBUtils {
    /**
     * 声明连接对象,执行对象,结果集对象
     */
    public static Connection conn=null;
    public static PreparedStatement state=null;
    public static ResultSet rs=null;

    /**
     * 获得连接对象的方法
     * @throws ClassNotFoundException
     * @throws SQLException
     */
    public static Connection getConnection() throws
ClassNotFoundException, SQLException {
        //1.加载驱动
```

```
            Class.forName("com.mysql.jdbc.Driver");
            //2.创建数据连接对象

conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/myschool",
"root", "root");
            return conn;
        }

        /**
         * 用jdbc执行增加,修改,删除的Sql语句
         * @param sql 要执行的sql语句
         * @param obs Sql语句的动态参数
         * @return int
         */
        public static int update(String sql,Object...obs) {
            //声明一个变量存执行的结果
            int result=0;
            try {
                //调用获得连接对象的方法
                conn=getConnection();

                //4.创建执行对象
                state=conn.prepareStatement(sql);
                //给执行对象中sql语句传参
                for (int i = 0; i < obs.length; i++) {
                    state.setObject(i+1, obs[i]);
                }

                //5.让执行对象调用相应的方法将Sql语句传到数据库去执行,并接
收结果
                result=state.executeUpdate();

            } catch (Exception e) {
                e.printStackTrace();
```

```
        }
        return result;
    }

    /**
     *  用jdbc执行查询的Sql语句
     * @param sql 要执行sql语句
     * @param obs sql的动态参数
     * @return ResultSet
     */
    public static ResultSet query(String sql,Object...obs) {
        try {
            //调用获得连接对象的方法
            conn=getConnection();

            //4.创建执行对象
            state=conn.prepareStatement(sql);
            //给执行对象中的Sql语句传参,数组的索引从0开始,但是点位符索引从1开始的
            for (int i = 0; i < obs.length; i++) {
                state.setObject(i+1, obs[i]);
            }

            //5.用执行对象调用相应的方法将Sql语句传到数据库中去执行,并接收执行结果
             rs=state.executeQuery();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return rs;
    }

    /**
     * 关闭jdbc对象的方法
```

```
     */
    public static void closeObject() {
        try {
            if (rs!=null) {
                rs.close();
            }
            if (state!=null) {
                state.close();
            }
            if (conn!=null) {
                conn.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

3.单元测试:是指对软件中的最小可测试单元进行检查和验证.(编写一段测试某个方法是否能 按正常流程来执行.);

3.1:单元测试规范:选中项目名->右键new->Source Folder->新建一个test文件夹->选中 test文件夹新建一个与原类相同的包名,再创建一个测试类,测试类的类 名由原类名Test组成.

3.2:注意事项:单元测试的方法不能是静态方法.单元测试的方法一般无参无返回值的.

3.3:单元测试的使用步骤:

3.3.1:导测试包:选中项目名右键->Build path->Add Libraries->Junit->选择一 个版本的包->finish.

3.3.2:在测试类中测试方法前面@org.junit.Test

3.3.3:选中测试方法名->右键->run as->Junit Test

eg:import org.junit.Test;

```
/**
 * 测试类
```

```
 * @author sx
 * @version 2020年2月14日
 */
public class DBUtilsTest {
	/**
	 * 测试工具类中插入sql的方法
	 */
	@Test
	public void updateTest1() {
		//准备sql语句
		String sql="insert into t_user(uname,age,address,upassword)
values(?,?,?,?)";
		//调用工具类中相应的方法
		int result=DBUtils.update(sql,"杨旭辉",20,"千锋","123456");
		//处理结果
		if (result>0) {
			System.out.println("操作成功");
		} else {
			System.out.println("操作失败");
		}
		//调用工具类中关闭的方法
		DBUtils.closeObject();
	}

	/**
	 * 测试工具类中修改sql的方法
	 */
	@Test
	public void updateTest2() {
		//准备sql语句
		String sql="update t_user set age=? where uid=?";
		//调用工具类中相应的方法
		int result=DBUtils.update(sql,80,3);
		//处理结果
```

```
        if (result>0) {
            System.out.println("操作成功");
        } else {
            System.out.println("操作失败");
        }
        //调用工具类中关闭的方法
        DBUtils.closeObject();
    }

    /**
     * 测试工具类中删除sql的方法
     */
    @Test
    public void updateTest3() {
        //准备sql语句
        String sql="delete from t_user where uid=?";
        //调用工具类中相应的方法
        int result=DBUtils.update(sql,3);
        //处理结果
        if (result>0) {
            System.out.println("操作成功");
        } else {
            System.out.println("操作失败");
        }
        //调用工具类中关闭的方法
        DBUtils.closeObject();
    }

    /**
     * 测试工具类中查询sql的方法
     */
    @Test
    public void queryTest1() {
```

```
            try {
                //准备sql语句
                String sql="select uid,uname,age,address,upassword from
t_user";
                //调用工具类中相应的方法
                 DBUtils.rs=DBUtils.query(sql);
                //处理结果
                 System.out.println("编号\t用户名");
                while (DBUtils.rs.next()) {
                    //读取当前行的结果,获得指定列的值并存入到变量中
                    int uid=DBUtils.rs.getInt("uid");
                    String uname=DBUtils.rs.getString("uname");
                     System.out.println(uid+"\t"+uname);
                }
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }finally {
                //调用工具类中关闭的方法
                DBUtils.closeObject();
            }
        }

        /**
         * 测试工具类中查询sql的方法
         */
        @Test
        public void queryTest2() {

            try {
                //准备sql语句
                String sql="select count(uid) from t_user";
                //调用工具类中相应的方法
                 DBUtils.rs=DBUtils.query(sql);
```

```
                //处理结果
                while (DBUtils.rs.next()) {
                    //读取当前行的结果,获得指定列的值并存入到变量中
                    int count=DBUtils.rs.getInt(1);
                    System.out.println("表中记录数为:"+count);
                }
            } catch (SQLException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }finally {
                //调用工具类中关闭的方法
                DBUtils.closeObject();
            }
        }
}
```

4.批处理:一次性同时执行多条Sql语句.批处理是用于增加,修改,删除的Sql语句.

批处理的作用:减少应用程序与数据库的交互次数,提高效率.

5.批量执行多条不同Sql语句(用statement对象)

eg:

```
public static void main(String[] args) throws SQLException {
        //声明连接对象和执行对象
        Connection conn=null;
        Statement state=null;

        try {
            //1.加载驱动
            Class.forName("com.mysql.jdbc.Driver");
            //2.创建数据连接对象

conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/myschool",
"root", "root");
            //3.准备Sql语句
            String sql1="insert into t_user(uname,age,address,upassword)
```

```
values('吴慧贤',18,'千锋','123456')";
                String sql2="insert into t_user(uname,age,address,upassword)
values('高圆圆',22,'千锋','123456')";
                String sql3="insert into t_user(uname,age,address,upassword)
values('吴毅恒',22,'千锋','123456')";
                String sql4="insert into t_user(uname,age,address,upassword)
values('张杨',24,'千锋','123456')";
                String sql5="update t_user set address='中国' where uid=2";
                String sql6="delete from t_user where uid=1";
                //4.创建执行对象
                state=conn.createStatement();
                //5.将Sql语句添加到执行对象的批处理中
                state.addBatch(sql1);
                state.addBatch(sql2);
                state.addBatch(sql3);
                state.addBatch(sql4);
                state.addBatch(sql5);
                state.addBatch(sql6);
                //6.用执行调用相应的方法将批处理sql语句传到数据库中去执行,
并接收结果
                int[] results=state.executeBatch();
                //7.处理结果
                for (int i = 0; i < results.length; i++) {
                        System.out.println(results[i]);
                }

            } catch (Exception e) {
                e.printStackTrace();
            }finally {
                //8.关闭对象
                if (state!=null) {
                        state.close();
                }
                if (conn!=null) {
```

```
                    conn.close();
                }
            }
        }
```

6.批量执行多条相同Sql语句,sql参数数据不同(用PreparedStatement执行对象)

```
    eg:public static void main(String[] args) throws SQLException {
        //声明连接对象和执行对象
        Connection conn=null;
        PreparedStatement state=null;

        try {
            //1.加载驱动
            Class.forName("com.mysql.jdbc.Driver");
            //2.创建数据连接对象

conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/myschool",
"root", "root");
            //3.准备Sql语句
            String sql="insert into t_user(uname,age,address,upassword)
values(?,?,?,?)";
            //4.创建执行对象
            state=conn.prepareStatement(sql);

            //5.将Sql语句参数添加到执行对象的批处理中
            for (int i = 1; i <=100; i++) {
                state.setString(1, "张伟"+i);
                state.setInt(2, 20+i);
                state.setString(3, "千锋");
                state.setString(4, "123456"+i);
                //将当前这条Sql语句的参数传到批处理中
                state.addBatch();
            }
```

```
                //6.用执行调用相应的方法将批处理sql语句传到数据库中去执行,
并接收结果
                int[] results=state.executeBatch();
                //7.处理结果
                for (int i = 0; i < results.length; i++) {
                        System.out.print(results[i]);
                }

        } catch (Exception e) {
                e.printStackTrace();
        }finally {
                //8.关闭对象
                if (state!=null) {
                        state.close();
                }
                if (conn!=null) {
                        conn.close();
                }
        }
    }
```

7.批量执行多条相同的Sql语句,参数不同,分批次提交(用preparedstatement)

```
    eg:public static void main(String[] args) throws SQLException {
        //声明连接对象和执行对象
        Connection conn=null;
        PreparedStatement state=null;

        try {
                //1.加载驱动
                Class.forName("com.mysql.jdbc.Driver");
                //2.创建数据连接对象

conn=DriverManager.getConnection("jdbc:mysql://localhost:3306/myschool",
```

```
"root", "root");
                //3.准备Sql语句
                String sql="insert into t_user(uname,age,address,upassword)
values(?,?,?,?)";
                //4.创建执行对象
                state=conn.prepareStatement(sql);

                //5.将Sql语句参数添加到执行对象的批处理中
                for (int i = 1; i <=100; i++) {
                    state.setString(1, "李盛升"+i);
                    state.setInt(2, 20+i);
                    state.setString(3, "千锋");
                    state.setString(4, "123456"+i);
                    //将当前这条Sql语句的参数传到批处理中
                    state.addBatch();

                    if (i%10==0) {
                        //6.用执行调用相应的方法将批处理sql语句传到数据库
中去执行,并接收结果
                        int[] results=state.executeBatch();
                        //7.处理结果
                        for (int j = 0; j < results.length; j++) {
                            System.out.print(results[j]+"\t");
                        }
                        //清空批处理中sql语句及参数
                        state.clearBatch();
                        //执行完一批处理就换行
                        System.out.println();
                    }
                }

            } catch (Exception e) {
                e.printStackTrace();
            }finally {
```

```
                //8.关闭对象
                if (state!=null) {
                        state.close();
                }
                if (conn!=null) {
                        conn.close();
                }
        }
}
```