课程主题

手写mybatis框架V1和V2版本

课程目标

1. 了解mybatis录播课程中有哪些知识点（课下看预习录播课程）
2. 通过手写mybatis框架的V1版本（JDBC+properties配置文件），去了解参数处理的思路和结果映射的思路
3. 不断去改造V1版本，将V1版本的代码，逐渐改造成通用代码（多个业务复用）
4. 通过手写mybatis框架的V2版本（JDBC+XML配置文件+其他，以面向过程思维去编码）

课程内容

```
OOAD（面向对象分析及设计）
程序设计能力----手写公共组件、程序设计（类或者接口的设计、设计模式）
源码阅读能力----
```

有问题的JDBC

```
    @Test
     public void test() {
         Connection connection = null;
         PreparedStatement preparedStatement = null;
         ResultSet rs = null;

         try {
             // 加载数据库驱动(硬编码、连接的频繁创建)
             Class.forName("com.mysql.jdbc.Driver");

             // 通过驱动管理类获取数据库链接connection = DriverManager
             connection =
DriverManager.getConnection("jdbc:mysql://localhost:3306/mybatis?
characterEncoding=utf-8","root", "root");

             // 定义sql语句 ?表示占位符
             String sql = "select * from user where username = ?";

             // 获取预处理 statement
             preparedStatement = connection.prepareStatement(sql);

             // 设置参数，第一个参数为 sql 语句中参数的序号（从 1 开始），第二个参数为
设置的
             preparedStatement.setString(1, "王五");

             // 向数据库发出 sql 执行查询，查询出结果集
```

```
            rs = preparedStatement.executeQuery();

            // 遍历查询结果集
            while (rs.next()) {
                System.out.println(rs.getString("id") + " " +
rs.getString("username"));
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            // 释放资源
            if (rs != null) {
                try {
                    rs.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (preparedStatement != null) {
                try {
                    preparedStatement.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (connection != null) {
                try {
                    connection.close();
                } catch (SQLException e) {
                    // TODO Auto-generated catch block e.printStackTrace();
                }
            }
        }
    }
```

升级改造

XML配置文件（参考Mybatis的XML）

- 全局配置文件（配置的是和业务无关的全局信息，比如数据源信息）

```
<configuration>
    <properties  resource="phase01/db.properties"></properties>
    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC" />
            <dataSource type="POOLED">
                <property name="driver" value="${db.driver}" />
                <property name="url" value="${db.url}" />
                <property name="username" value="${db.username}" />
                <property name="password" value="${db.password}" />
            </dataSource>
        </environment>
    </environments>
    <mappers>
        <mapper resource="phase01/UserMapper.xml" />
    </mappers>
```

```
</configuration>
```

- 映射文件（和业务有关的配置信息，比如SQL语句、参数信息、映射结果信息。

```
<mapper namespace="test">

    <select id="findUserById" parameterType="int"
        resultType="com.kkb.mybatis.phase01.po.User"
statementType="STATEMENT">
        SELECT * FROM user WHERE id = #{id} AND usernaem = #{username}

    </select>
    <select id="findUserByName" parameterType="string"
        resultType="com.kkb.mybatis.phase01.po.User">
        SELECT * FROM user WHERE username = ${value}
    </select>
</mapper>
```

手写框架思路分析

整体思路

首先写V1.0版本：直接改造JDBC代码去完成。

其次写V2.0版本：是以面向过程的思维方式去实现的，（一个类去完成）。

其次写V3.0版本：是以面向对象的思维方式去实现的，（就是模仿Mybatis的类去实现手写）。

V1.0版本思路

- 需求

> 根据用户信息，查询用户列表

- 将JDBC中的硬编码，写入properties文件中

```
db.driver=com.mysql.jdbc.Driver
db.url=jdbc:mysql://111.231.106.221:3306/kkb
db.username=kkb
db.password=kkb111111
db.sql.queryUserById=select * from user where username = ?
db.sql.queryUserById.parametertype=com.kkb.mybatis.po.User
db.sql.queryUserById.params=username
db.sql.queryUserById.resultclassname=com.kkb.mybatis.po.User
db.sql.queryOrderById=select * from user where username = ?
```

- 加载properties配置文件

```
    private Properties properties = new Properties();

    private final static String DRIVER = "db.driver";
```

```
    /**
     * 加载properties配置文件
     */
    public void loadProperties() {
        try {
            InputStream inputStream =
this.getClass().getClassLoader().getResourceAsStream("jdbc.properties");
            properties.load(inputStream);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

- 执行JDBC查询

```
/**
     * 执行JDBC代码，完成查询用户操作
     *
     * @return
     */
    public List<Object> selectList(String statementId, Object paramObject)
{
        Connection connection = null;
        PreparedStatement preparedStatement = null;
        ResultSet rs = null;
        // 要返回的结果集合
        List<Object> results = new ArrayList<Object>();
        try {
            // 加载数据库驱动
            Class.forName(properties.getProperty(DRIVER));

            // 通过驱动管理类获取数据库链接connection = DriverManager
            connection =
DriverManager.getConnection(properties.getProperty("db.url"),
                    properties.getProperty("db.username"),
properties.getProperty("db.password"));

            // 定义sql语句 ?表示占位符
            String sql = properties.getProperty("db.sql." + statementId);

            // 获取预处理 statement
            preparedStatement = connection.prepareStatement(sql);

            // 设置参数，第一个参数为 sql 语句中参数的序号（从 1 开始），第二个参数为
设置的

            // 先获取入参类型
            String parametertype = properties.getProperty("db.sql." +
statementId + ".parametertype");
            Class<?> parameterTypeClass = Class.forName(parametertype);

            // 先判断入参类型(8种基本类型、String类型、各种集合类型、自定义的Java类
型)
            if (SimpleTypeRegistry.isSimpleType(parameterTypeClass)) {
```

```
                preparedStatement.setObject(1, paramObject);
            } else {
                // 自定义的Java类型
                // 此处需要遍历SQL语句中的参数列表
                String params = properties.getProperty("db.sql." +
statementId + ".params");
                String[] paramArray = params.split(",");
                for (int i = 0; i < paramArray.length; i++) {
                    // 列名
                    String param = paramArray[i];

                    // 根据列名获取入参对象的属性值，前提：列名和属性名称要一致
                    Field field =
parameterTypeClass.getDeclaredField(param);
                    field.setAccessible(true);
                    // 获取属性值
                    Object value = field.get(paramObject);
                    preparedStatement.setObject(i + 1, value);
                }
            }

            // 向数据库发出 sql 执行查询，查询出结果集
            rs = preparedStatement.executeQuery();

            // 获取要映射的结果类型
            String resultclassname = properties.getProperty("db.sql." +
statementId + ".resultclassname");
            Class<?> resultTypeClass = Class.forName(resultclassname);

            Object result = null;
            while (rs.next()) {
                // 每一行对应一个对象
                result = resultTypeClass.newInstance();

                ResultSetMetaData metaData = rs.getMetaData();
                int columnCount = metaData.getColumnCount();

                for (int i = 1; i <= columnCount; i++) {
                    // 获取结果集中列的名称
                    String columnName = metaData.getColumnName(i);

                    Field field =
resultTypeClass.getDeclaredField(columnName);
                    field.setAccessible(true);
                    field.set(result, rs.getObject(columnName));

                }

                results.add(result);
            }

        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            // 释放资源
            if (rs != null) {
                try {
                    rs.close();
```

```
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (preparedStatement != null) {
                try {
                    preparedStatement.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (connection != null) {
                try {
                    connection.close();
                } catch (SQLException e) {
                    // TODO Auto-generated catch block
e.printStackTrace();
                }
            }
        }
        return results;
    }
```

- 测试

```
    @Test
    public void test() {
        loadProperties();

        User user = new User();
        user.setUsername("王五");
        List<Object> list = selectList("queryUserById", user);
        System.out.println(list);
    }
```