# 4-2 J2EE企业级项目开发

## SSM-MyBatis框架

日照职业技术学院电子信息工程学院 • 盛雯雯

# 一、Mybatis介绍

# 关于MyBatis

- MyBatis 本是apache的一个开源项目iBatis, 2010年这个项目由apache software foundation 迁移到了google code，并且改名为MyBatis 。2013年11月迁移到Github。
- MyBatis是一个优秀的**持久层框架**，它对jdbc的操作数据库的过程进行封装，使开发者只需要关注 SQL 本身，而不需要花费精力去处理例如注册驱动、创建connection、创建statement、手动设置参数、结果集检索等jdbc繁杂的过程代码。
- Mybatis通过xml或注解的方式将要执行的各种statement（statement、preparedStatemnt、CallableStatement）配置起来，并通过java对象和statement中的sql进行映射生成最终执行的sql语句，最后由mybatis框架执行sql并将结果映射成java对象并返回。

# 使用jdbc编程问题总结

- 加载数据库驱动
- 创建并获取数据库链接
- 创建jdbc statement对象
- 设置sql语句
- 设置sql语句中的参数(使用preparedStatement)
- 通过statement执行sql并获取结果
- 对sql执行结果进行解析处理
- 释放资源(resultSet、preparedstatement、connection)

# jdbc**问题总结如下：**

- 数据库连接创建、释放频繁造成系统资源浪费，从而影响系统性能。如果使用数据库连接池可解决此问题。
- Sql语句在代码中硬编码，造成代码不易维护，实际应用中sql变化的可能较大，sql变动需要改变java代码。
- 使用preparedStatement向占有位符号传参数存在硬编码，因为sql语句的where条件不一定，可能多也可能少，修改sql还要修改代码，系统不易维护。
- 对结果集解析存在硬编码（查询列名），sql变化导致解析代码变化，系统不易维护，如果能将数据库记录封装成pojo对象解析比较方便。

# Mybatis**介绍**

MyBatis

- MyBatis是一个支持***普通SQL查询***，***存储过程***和***高级映射***的优秀**持久层框架**。MyBatis消除了几乎所有的JDBC代码和参数的手工设置以及对结果集的检索封装。
- MyBatis可以使用简单的**XML或注解**用于配置和原始**映射**，将接口和Java的POJO（Plain Old Java Objects，普通的Java对象）**映射**成数据库中的记录

# 4-2 J2EE企业级项目开发

## SSM-MyBatis框架

日照职业技术学院电子信息工程学院 • 盛雯雯

## 二、Mybatis搭建

# 搭建环境

- 1.创建Maven项目
- 2.pom配置
- 3.创建数据表
- 4.创建POJO类
- 5.创建MyBatis的配置文件—配置数据库（sqlMapConfig.xml）
- **SqlMapConfig.xml是mybatis核心配置文件，配置文件内容为数据源、事务管理。**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <!-- 引入数据源配置文件  -->
    <properties resource="db.properties" />
    <!-- 定义数据库环境，且默认使用development环境 -->
    <environments default="development">
        <!-- 定义id为development的数据库环境 -->
        <environment id="development">
            <!-- 采用jdbc事务管理 -->
            <transactionManager type="JDBC"/>
            <!-- 配置数据库连接信息 -->
            <dataSource type="POOLED">
                <property name="driver" value="${jdbc.driver}"/>
                <property name="url" value="${jdbc.url}"/>
                <property name="username" value="${jdbc.username}"/>
                <property name="password" value="${jdbc.password}"/>
            </dataSource>
        </environment>
    </environments>
</configuration>
```

## db.propertie

```
jdbc.driver=com.mysql.cj.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/ssm?characterEncoding=utf-8&userSSL=false&serverTimezone=GMT%2B8
jdbc.username=root
jdbc.password=123123
```

**为什么要这样配？**

**将参数定义放在单独的文件里，不需要修改其他配置文件。**

# 4-2 J2EE企业级项目开发

## SSM-MyBatis框架

日照职业技术学院电子信息工程学院 ● 盛雯雯

## 三、使用MyBatis查询表中的数据

# （一）配置一个映射

## 1.定义文件头：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
    PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-config.dtd" >
```

# （一）配置一个映射

## 2.定义命名空间

```
<mapper namespace=" “>
</mapper>
```

为这个mapper指定一个唯一的namespace

namespace= “**student**”

## 注册mapper

将mapper配置到核心配置文件

# （一）配置一个映射

## 3.在mapper中写一个select语句

在select标签中编写查询的SQL语句， 设置select标签的id属性为getStudent。

id属性值必须是唯一的，不能够重复 。

使用parameterType属性指明查询时使用的参数类型，

resultType属性指明查询返回的结果集类型

esultType=“**com.myBatisTest.pojo.Student**” 就表示将查询结果封装成一个Student类的对象返回

**Student**类就是**Student**表所对应的实体类

参数用#{}

# （二）在配置文件中注册一个mapper

# 编写一个测试类

## 1.加载MyBatis配置

```
//使用类加载器加载mybatis的配置文件（它也加载关联的映射文件）
InputStream inputStream=
Resources.getResourceAsStream("myBatisConfig.xml");
```

# 编写一个测试类

## 2.构建Session--Connection

```
//构建sqlSession的工厂
SqlSessionFactory ssf=new SqlSessionFactoryBuilder().build(inputStream);
//创建能执行映射文件中sql的sqlSession
SqlSession session=ssf.openSession();
```

# 4-2 J2EE企业级项目开发

## SSM-MyBatis框架

日照职业技术学院电子信息工程学院 ● 盛雯雯

## 四、各种SQL操作

# 查询

## 查询所有用户

**（1）没有参数**

**（2）结果是集合类型**

# 查询

## 根据用户名模糊查询用户

（1）模糊查询

（2）有参数传入

（3）结果为集合类型

**方案1：**SELECT * FROM `user` WHERE username LIKE #{username}

sqlSession.selectList("queryUserByUsername1", "%王%");

**方案2：**SELECT * FROM `user` WHERE username LIKE '%${value}% ‘

sqlSession.selectList("queryUserByUsername2", "王");

# 查询

## #{}和${}

#{}表示一个占位符号，通过#{}可以实现preparedStatement向占位符中设置值，自动进行java类型和jdbc类型转换。#{}可以有效防止sql注入。 #{}可以接收简单类型值或pojo属性值。 如果parameterType传输单个简单类型值，#{}括号中可以是value或其它名称。

${}表示拼接sql串，通过${}可以将parameterType 传入的内容拼接在sql中且不进行jdbc类型转换， ${}可以接收简单类型值或pojo属性值，如果parameterType传输单个简单类型值，${}括号中只能是value。

# 查询

## parameterType和resultType

parameterType：指定输入参数类型，mybatis通过ognl从输入对象中获取参数值拼接在sql中。

resultType：指定输出结果类型，mybatis将sql查询结果的一行记录数据映射为resultType指定类型的对象。如果有多条数据，则分别进行映射，并把对象放到容器List中

# 查询

## selectOne和selectList

selectOne查询一条记录，如果使用selectOne查询多条记录则抛出异常：

org.apache.ibatis.exceptions.TooManyResultsException: Expected one result (or null) to be returned by selectOne(), but found: 3

at org.apache.ibatis.session.defaults.DefaultSqlSession.selectOne(DefaultSqlSession.java:70)

selectList可以查询一条或多条记录

# 添加

- 使用的sql：
  - insert into Student (name,pwd,age,sex,birth) values(#{name},#{pwd},#{age},#{sex},#{birth})
  - session.insert("**student.insertStudent**",stu);
  - 注意，更新数据库要session.commit();

# 小结

- MyBatis传入参数类型可以是Java的基本数据类型，但是这种情况只适用于只有一个参数的情况，通过#{参数名}就可以获得传入值。
- 如果要多个参数传入，则要复杂数据类型来支持。
  - Java实体类：通过#{属性名}获得传入的参数值
  - Map：通过#{Map的key}获得传入的值

# 通过Map传值

- 问题：根据name和sex查询学生列表。

- 不管什么类型的参数，有多少个参数，我们都可以将之封装成Map数据结构进行入参，通过Map的key获取传入的值。

# 自定义查询结构映射-1

## resultMap

MyBatis中使用resultType做自动映射的时候，要注意字段名与POJO的属性名保持一致。

问题：查询所有学生的id、姓名、年龄、性别、专业名称。（注意：专业名称而不是专业ID）

1.在Student类中添加stuName属性

2.在Mapper中定义<resultMap>

# 自定义查询结构映射-1

- resultMap元素用来描述如何将结果集映射到Java对象，此处使用resultMap对列表展示所需要的必要字段进行自由映射，特别是当数据库的字段名和POJO中的属性名不一致的情况下。
  - id：对应于select元素的resultMap属性的引用
  - type：resultMap映射的结果类型
  - result子节点：标识一些简单的属性，其中column标识从数据库查询的字段，property标识查询出的字段给POJO的那个属性。

# resultMap和resultType的区别

- resultType表示返回类型，包括基础数据类型和复杂数据类型
- resultMap：对外部resultMap的引用。应用场景：数据库的字段与对象属性不一致或者做复杂的关联查询。
- 两者只能使用其一。

# 添加

## mysql自增主键返回

```
SELECT LAST_INSERT_ID()
```

```
<selectKey resultType="int" keyColumn="id" keyProperty="id" order="AFTER">
SELECT LAST_INSERT_ID()
</selectKey>
```

```
<!-- selectKey 标签实现主键返回 -->
<!-- keyColumn:主键对应的表中的哪一列 -->
<!-- keyProperty：主键对应的pojo中的哪一个属性 -->
<!-- order：设置在执行insert语句前执行查询id的sql，孩纸在执行insert语句之后执行查询id的sql -->
<!-- resultType：设置返回的id的类型 -->
```

# 修改

## 根据Id修改

自动匹配属性。。

```
<update id="resetName" parameterType="com.myBatisTest.pojo.Student" >
 update Student set  name=#{name}  where  id=#{id}
</update>
```

# 删除

## 根据Id删除

```
<delete id="deleteById" parameterType="com.myBatisTest.pojo.Student">
 delete from student where id=#{id}
</delete>
```

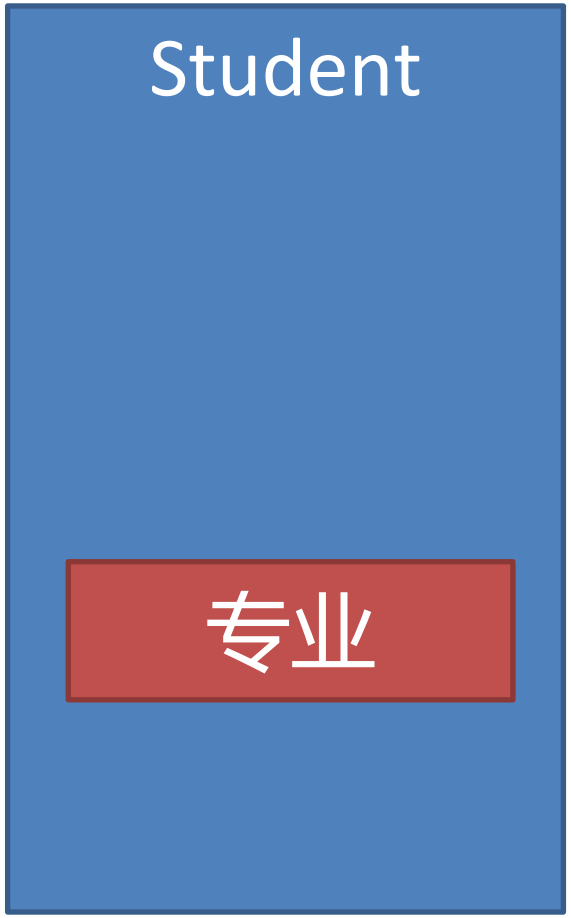
# 4-2 J2EE企业级项目开发

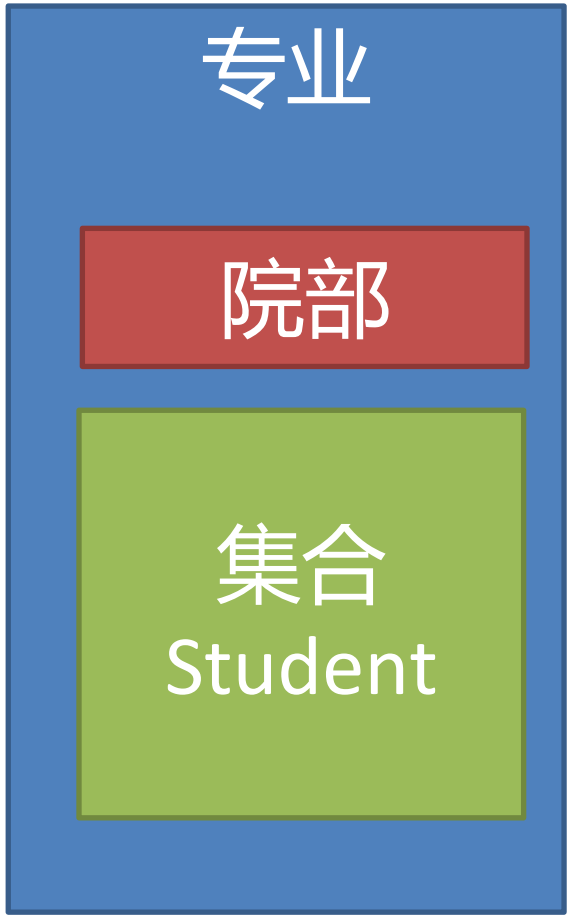
## SSM-MyBatis框架

日照职业技术学院电子信息工程学院 ● 盛雯雯

## 五、实现高级结果映射

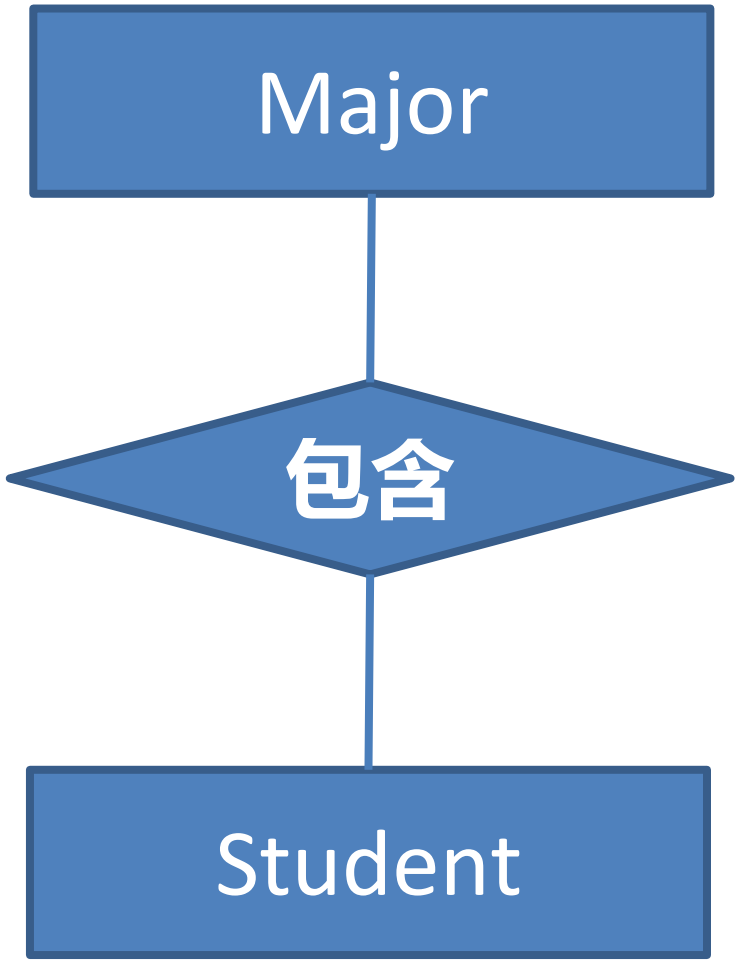
# 对象图导航---OGNL




stu.getMajor().getName
stu.major.dept.name

major.getStudents().get(i).getName()
major.students[i].name




1.建立POJO类
2.对象导航的思想，去描述POJO之间的关系
（1）1：n，则在1的一方加上n方的集合
（2）n:1，则在n的一方加上1的对象属性
（3）getter和setter方法
从OOP的层面描述ER图
3.majorMapper.xml
（1）头，nameSpace，sql语句
（2） majorMapper.xml注册到核心配置文件
4.测试

# 1.resultMap

**1.属性**

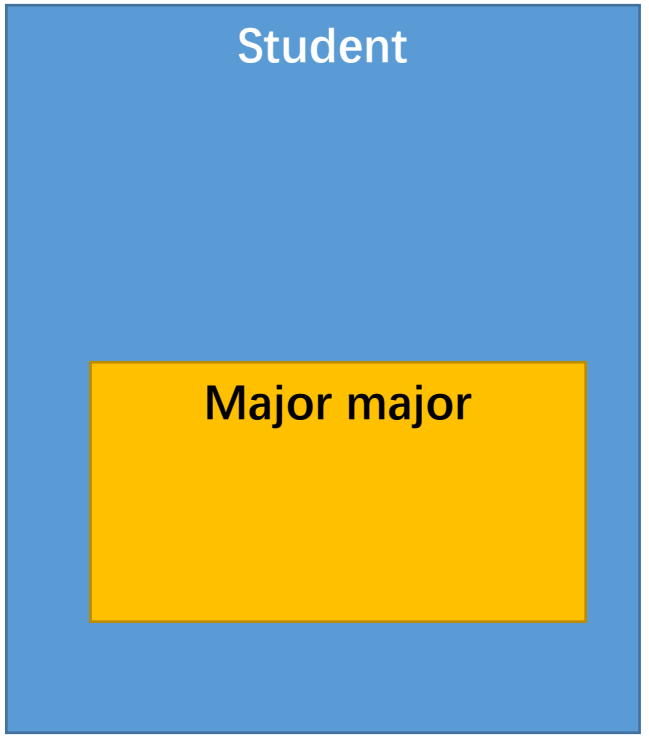
- id属性：对应于select元素的resultMap属性的引用
- type属性：resultMap映射的结果类型

**2.子节点**

- result：标识一些简单的属性，其中column标识从数据库查询的字段，property标识查询出的字段给POJO的那个属性。

# 2.association-1

- **一对一关系的处理，n:1**
- 映射到pojo某个“复杂类型”属性。比如另一个pojo。
- 案例：获取所有用户信息，包括该用户所在的专业
  - 1.在Student类中加入Major属性（getter和setter）
  - 2.配置resultMap



# 2.association-2

- **属性**
  - **javaType：完整的Java类名或者别名**
  - **property：对应JavaBean的属性名**
- **子元素：**
  - **result：**
  - **id：resultMap的唯一标识，可以是主键或联合主键。如果没有配置id，不影响结果，但是会影响性能。**

# association**的**resultMap**属性**

- association可以引用外部的resultMap
- 案例：查询学生的专业
- 1.定义resultMap---major
- 2.应用major
- 推荐

# 3.Collection处理一对多关联关系

- collection也是映射到JavaBean的某个“复杂数据类型”属性。这个属性还是一个集合列表。即：JavaBean找那个内嵌一个复杂数据类型（集合）属性。
- 案例：查询出某个班级所有的学生。
  - 1.在Major类中定义集合（getter和setter）
  - 2.定义resultMap

# 3.Collection处理一对多关联关系

## collection属性

ofType：完整的java类名或别名

property：映射数据库列的实体对象的属性。即那个类的集合

resultMap：提取相应代码到resultMap中，给collection增加resultMap属性

与association元素的resultMap一个用法

# 问题

## 只能查询出一条记录

如果两表联查，主表和明细表的主键都是id的话，明细表的多条只能查询出来第一条。

**解决办法：**

级联查询的时候，主表和从表有一样的字段名的时候，在mysql上命令查询是没问题的。

1.在mybatis中主从表需要为相同字段名设置别名。

2.在设计表的时候，避免同名字段

# 4-2 J2EE企业级项目开发

## SSM-MyBatis框架

日照职业技术学院电子信息工程学院 ● 盛雯雯

六、小结

# Mybatis**解决**jdbc**编程的问题**

- 数据库连接创建、释放频繁造成系统资源浪费从而影响系统性能，如果使用数据库连接池可解决此问题。
- 解决：在SqlMapConfig.xml中配置数据连接池，使用连接池管理数据库链接。
- Sql语句写在代码中造成代码不易维护，实际应用sql变化的可能较大，sql变动需要改变java代码。
- 解决：将Sql语句配置在XXXXmapper.xml文件中与java代码分离。
- 向sql语句传参数麻烦，因为sql语句的where条件不一定，可能多也可能少，占位符需要和参数一一对应。
- 解决：Mybatis自动将java对象映射至sql语句，通过statement中的parameterType定义输入参数的类型。
- 对结果集解析麻烦，sql变化导致解析代码变化，且解析前需要遍历，如果能将数据库记录封装成pojo对象解析比较方便。
- 解决：Mybatis自动将sql执行结果映射至java对象，通过statement中的resultType定义输出结果的类型。

# 4-2 J2EE企业级项目开发

## SSM-MyBatis框架

日照职业技术学院电子信息工程学院 ● 盛雯雯

## 六、原始Dao开发方式

# 步骤

- 原始Dao开发方法需要程序员编写Dao接口和Dao实现类。
- 1.建表
- 2.新建POJO类
- 3.**映射文件**
- 4. **Dao接口**
- 5. **Dao实现类**
- 6.测试。。

# 原始DAO

## 原始Dao开发中存在的问题

Dao方法体存在重复代码：通过SqlSessionFactory创建SqlSession，调用SqlSession的数据库操作方法

调用sqlSession的数据库操作方法需要指定statement的id，这里存在硬编码，不得于开发维护

# 4-2 J2EE企业级项目开发

## SSM-MyBatis框架

日照职业技术学院电子信息工程学院 • 盛雯雯

## 七、 Mapper动态代理方式

# 开发规范

- Mapper接口开发方法只需要程序员编写Mapper接口（相当于Dao接口），由Mybatis框架根据接口定义创建接口的动态代理对象，代理对象的方法体同上边Dao接口实现类方法。

**如此让人无语。。。。**

# Mapper**接口开发需要遵循以下规范：**

- Mapper.xml文件中的namespace与mapper接口的类路径相同。
- Mapper接口方法名和Mapper.xml中定义的每个statement的id相同
- Mapper接口方法的输入参数类型和mapper.xml中定义的每个sql 的parameterType的类型相同
- Mapper接口方法的输出参数类型和mapper.xml中定义的每个sql的resultType的类型相同

# 4-2 J2EE企业级项目开发

## SSM-MyBatis框架

日照职业技术学院电子信息工程学院 • 盛雯雯

八、动态Mapper

# 什么是动态SQL?

- 通过mybatis提供的各种标签方法实现动态拼接sql。
- 动态SQL是MyBatis的一个强大特性。在使用JDBC操作数据时，如果查询条件特别多，将条件串联成SQL字符串是一件痛苦的事情。通常解决方案是写多个if-else语句对字符串进行拼接。这样并不能确保忘了空格、在条件前后添加“，”、“and”等错误。
- MyBatis有强大的动态SQL来改善这种情况。
  - if：条件选择
  - choose（when，otherwise）：相当于switch
  - where：SQL语句中的where
  - set：动态更新语句
  - trim foreach：迭代一个集合

# If标签

- 案例：在高级查询中，可以根据用户的输入按照名字、专业、性别查询。
- 用途举例：论文检索

# Where**标签**

- where 1=1，多余的and
- 举例：按照学生的姓名和性别查询

# if+trim

- trim元素会自动识别标签内是否有返回值，如果有返回值，则在自己包含的内容前加上某些前缀，也可以在其后加上某些后缀。
- prefix：前缀
- suffix：后缀
- prefixOverride：对trim包含内容的首部进行指定内容。
- suffixOverride：对应trim包含内容的尾部进行制定内容的忽略。

# if+set**用法**

- 在update语句中，更新时如果某个字段没有传值，则不需要修改。
- <set>
- <if></if>
- </set>

# if+trim**用法**

- <trim prefix= “set” suffixOverrides= “,” suffix= “where id=#{id}” >
  - <if>
  - <if>
  - ……
- </trim>

# 使用foreach完成复杂查询

- 使用foreach迭代数组类型的入参，实现根据用户角色列表获得该角色列表下的用户信息。
- 使用foreach迭代List类型的入参，实现根据用户角色列表获得该角色列表下的用户信息。
- 使用foreach迭代Map类型的入参，实现根据用户角色列表获得该角色列表下的用户信息。

# foreach的入参

- foreach主要用在构建in条件中，它可以在SQL语句中迭代一个集合。
- item：表示集合中每一个元素进行迭代的别名。
- open：表示以什么开始--（
- separate：迭代之间以什么为分隔符--，
- close：表示以什么结束--）
- collection，必须指定，集合类型
  - （1）List—list
  - （2）数组---array
  - （3）如果传入的是多个参数，则要将这些数据封装在一个Map中。

# foreach

## 查询学生姓名在某列表的集合-List

List类型入参

```
<select id="getStusInNames1" parameterType="List" resultType="Student">
  select *from Student where name in
  <foreach collection="list" item="names" open="(" separator="," close=")">
    #{names}
  </foreach>
</select>
```

# foreach

## 查询学生姓名在某列表的集合-数组

数组类型入参

```
<select id="getStusInNames2" resultType="Student" parameterType="String">
 select *from Student where name in
  <foreach collection="array" item="names" open="(" separator="," close=")">
     #{names}
  </foreach>
</select>
```

# foreach

## Map类型入参

collection：key的值

原理：无论传入那种数据类型，MyBatis都将参数放在一个Map中。

myBatis接收的参数类型：基本数据类型、对象、List、数组、Map

基本数据类型：变量名作为key，变量值作为value，此时Map只有一个元素

对象：属性名作为key，属性值作为value

List：默认“list”作为key，该List即为value

数组：默认array作为key，该数组即为value

Map：键值不变

# foreach - Map**类型入参**

- <**select id="getStusInNames3" resultType="Student" parameterType="Map**">
  select *from Student where name in
  <**foreach collection="names" item="a" open="(" separator="," close=")**">
  #{**a**}
  </**foreach**>
  and id in
  <**foreach collection="ids"  item="ids" open="(" separator="," close=")**">
  #{**ids**}
  </**foreach**>
  </**select**>
- 也可以不用Map入参，用@Param入参

# choose（when、otherwise）

- when：当test中的条件满足的时候，输出when元素的内容。类似于java的switch，同样按照条件的顺序来处理，当when中一旦有条件满足了，则跳出choose。也就是说，所有的when和otherwise条件中，只有一个条件会输出。
- otherwise：当所有的when都不满足的时候，自动输出otherwise中的内容，相当于default

# 例

- <select id="getStus2" resultType="Student" parameterType="Map">
  select *from Student where 1=1
  <choose>
  <when test="sex='男'">
  and mId=3
  </when>
  <when test="sex='女'">
  and mId=2
  </when>
  <when test="age!=0">
  and name like "%stu%"
  </when>
  </choose>
  </select>

根据提示的信息，我们很容易知道是数据格式化的时候出了问题，不过为什么哪？我们定义的没有错呀！传递的也没有错呀！想不通，百度一下吧！下面是百度的一个结果，指明了引起错误原因的所在！是OGNL的语法问题，这里'Y'将被认为是char类型的数据，但是'YY'或者 “Y” 将被认为是String类型的数据，解决方案如下所示：

1：将代码改为test="param eq 'Y'.toString()"

2：将代码改为test="name == "Y""

3：将代码改为test='index == "Y"'

当对象的映射文件及对应的属性如下编写时，<result column="IF_SHIELD" property="ifShield" jdbcType="VARCHAR" /> private String ifShield;

<if test="ifShield==1">是没问题的（并且数字是全部没问题的，无论是小数还是整数是正数还是负数），不过这样编写<result column="IF_SHIELD" property="ifShield" jdbcType="CHAR" /> private String ifShield;

<if test= “ifShield==1” >就是有问题的。

# 分页

## limit（起始位置，页面容量）

```
public List<Student> getStus3(@Param("from") int from, @Param("pageSize")
int pageSize);
```

这样就不用用Map了

分页：

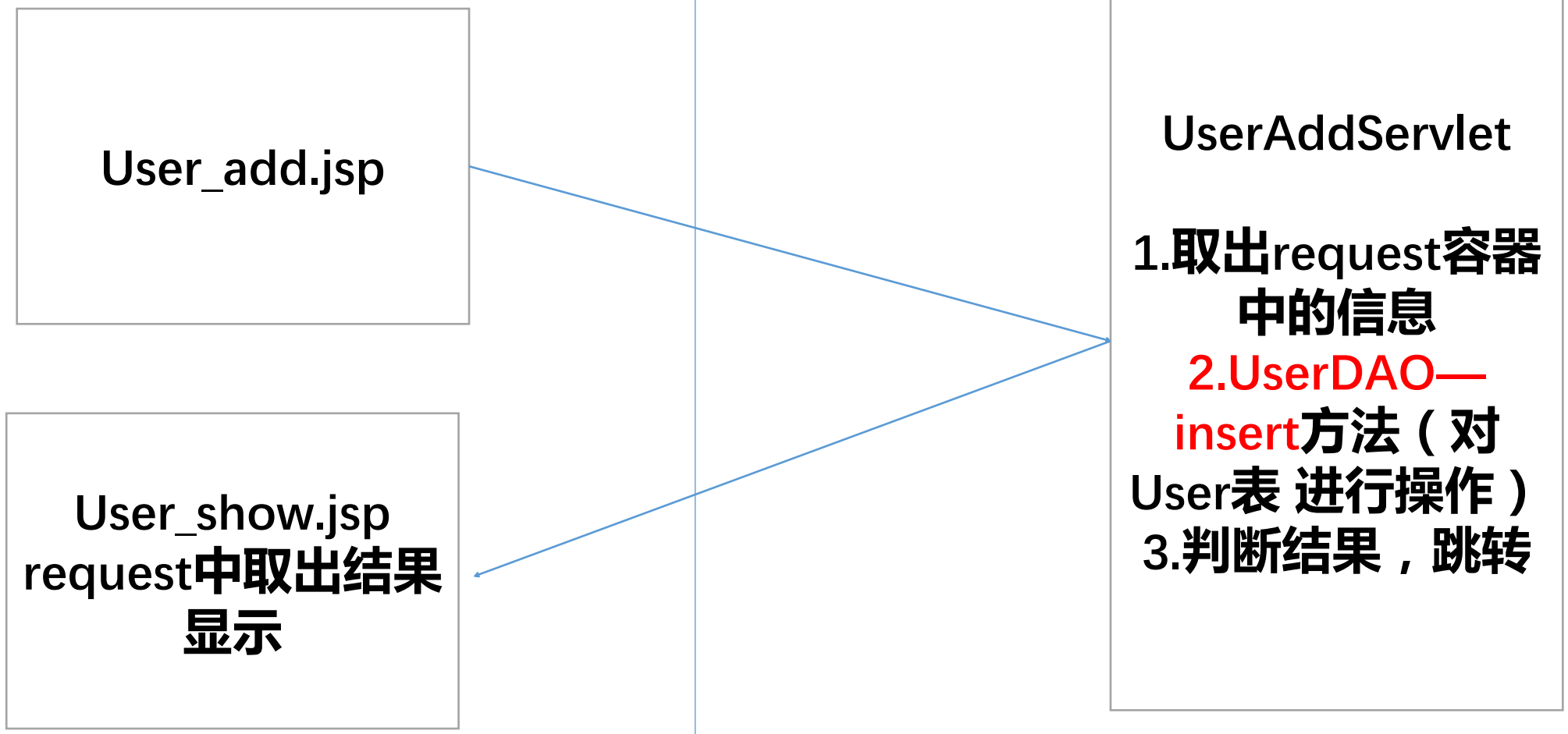
```
limit #{from},#{pageSize}
```

# 4-2 J2EE企业级项目开发

## SSM-MyBatis框架

日照职业技术学院电子信息工程学院 • 盛雯雯

## 九、Mapper动态代理方式


User_add.jsp
User_show.jsp
request中取出结果
显示
UserAddServlet
1.取出request容器
中的信息
2.UserDAO—
insert方法（对
User表 进行操作）
3.判断结果，跳转

# UserDAO

- 继承DBManager
- 插入
- 删除
- 检索
- 修改
- 。。。

# 对数据库的操作--JDBC

- 1.建桥：建立连接
- 2.派车：Statement小车
- 3.下订单：**SQL----mpper.xml**
- 4.拉去执行
- 5.将结果载回来分析（int---ResultSet）

# 关于Mapper动态代理

- Mapper接口开发方法只需要程序员编写Mapper接口（相当于Dao接口），由Mybatis框架根据接口定义创建接口的动态代理对象，代理对象的方法体同上边Dao接口实现类方法。
- Mapper接口开发需要遵循以下规范：
  - Mapper.xml文件中的namespace与mapper接口的类路径相同。
  - Mapper接口方法名和Mapper.xml中定义的每个statement的id相同
  - Mapper接口方法的输入参数类型和mapper.xml中定义的每个sql 的parameterType的类型相同
  - Mapper接口方法的输出参数类型和mapper.xml中定义的每个sql的resultType的类型相同

# 步骤

- 1.定义接口IStudentMapper
- 2.StudentMapper.xml文件的namespace属性，指向所对应的的接口
- 3.在接口中定义方法，注意方法定义的规范。
- 4.测试类
  - （1）session工厂
  - （2）session
  - （3）生成IStudentMapper接口的代理对象
  - （4）调用接口的方法获得结果

# 小结1

## selectOne和selectList

动态代理对象调用sqlSession.selectOne()和sqlSession.selectList()是根据mapper接口方法的返回值决定。

如果返回list则调用selectList方法。

如果返回单个对象则调用selectOne方法。

# 小结2

## namespace

mybatis官方推荐使用mapper代理方法开发mapper接口，程序员不用编写mapper接口实现类，使用mapper代理方法时，输入参数可以使用pojo包装对象或map对象，保证dao的通用性。

# 配置别名

## typeAliases

```
<typeAliases>
    <typeAlias type="com.myBatisTest.pojo.Student" alias="Student “></typeAlias>
</typeAliases>
```

type：所对应的的类

alias：别名，大小写不敏感

# 配置别名

## 批量配置别名