# MyBatis

使用说明参考：<https://mybatis.org/mybatis-3/zh/index.html>

## 简单使用Mybatis

### 配置数据源源文件mybatis.xml

该配置文件主要是配置数据源信息，以及配置实体类和数据库操作的配置文件信息（mappers标签）。

|  |
| --- |
| *<?***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**>  <**environments default="development"**>  <**environment id="development"**>  <**transactionManager type="JDBC"**/>  <**dataSource type="POOLED"**>  <**property name="driver" value="com.mysql.cj.jdbc.Driver"**/>  <**property name="url" value="jdbc:mysql://localhost:3306/test"** />  <**property name="username" value="root"**/>  <**property name="password" value="123456"**/>  </**dataSource**>  </**environment**>  </**environments**>  <**mappers**>  <**mapper resource="user.mapper.xml"**/>  </**mappers**> </**configuration**> |

### 获得SqlSessionFactory和SqlSession

通过代码读取配置文件，获取session数据库会话对象。以备于执行sql

|  |
| --- |
| String resource = **"mybatis.cfg.xml"**;  InputStream inputStream = Resources.*getResourceAsStream*(resource);  SqlSessionFactory sqlSessionFactory = **new** SqlSessionFactoryBuilder().build(inputStream);  **return** sqlSessionFactory; } **public static** SqlSession getSqlSession() **throws** IOException {  SqlSessionFactory sqlSessionFactory = *getSqlSessionFactory*();  **return** sqlSessionFactory.openSession(); } |

### 实体类映射文件

封装对该类对象的数据库操作语句

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"** *?>* **<!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd"*>*** <**mapper namespace="com.pcq.entity.UserMapper"**>  <**select id="selectUser" resultType="com.pcq.entity.User"**>  select *\** from user where id = #{id}  </**select**> </**mapper**> |

### 测试代码

通过会话，传入实体类配置文件的映射id和参数，执行sql。

注意，执行完需要关闭数据库会话连接，如果有删除插入或者更新的操作，则还需要提交事务。

|  |
| --- |
| @Test **public void** testSelectUser() {  **try** {  SqlSession sqlSession = MyBatisUtil.*getSqlSession*();  User user = sqlSession.selectOne(**"com.pcq.entity.UserMapper.selectUser"**, 1);  sqlSession.close();  System.***out***.println(user);  }  **catch** (IOException e) {  e.printStackTrace();  } } |

### 实体类属性名和数据库列名不一致

在mapper文件中写的sql，查询出来的数据库列名字段（例如password），Mybatis默认以set列名的方式来设值的（setPassword），此时实体类的字段名如果不一致（pwd），则无法正确注入值。为解决这种问题，mybatis提供了两种解决方式：

1. 直接在sql中写别名，例如select password pwd from user；
2. 通过配置resultMap

|  |
| --- |
| <**resultMap id="userMap" type="User"**>  <**result column="password" property="pwd"**/> </**resultMap**> <**select id="findAll" resultMap="userMap"**>  select *\** from user </**select**> |

## 基本的crud操作

### 1.实体类操作配置文件

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"** *?>* **<!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd"*>*** <**mapper namespace="com.pcq.entity.UserMapper"**>  <**select id="findAll" resultType="com.pcq.entity.User"**>  select *\** from user  </**select**>  <**select id="selectUser" resultType="com.pcq.entity.User"**>  select *\** from user where id = #{id}  </**select**>  <**insert id="addUser" parameterType="com.pcq.entity.User" useGeneratedKeys="true"**>  insert user(username,password) values(#{username},#{password})  </**insert**>   <**update id="updUser"**>  update user set username=#{username},password=#{password} where id = #{id};  </**update**>    <**delete id="deleteUserById"**>  delete from user where id = #{id}  </**delete**>   <**insert id="addUsers" useGeneratedKeys="true" keyProperty="id"**>  insert into user (username,password) values  <**foreach collection="list" item="user" separator=","**>  (#{user.username}, #{user.password})  </**foreach**>  </**insert**> </**mapper**> |

### 2.Dao文件代码

|  |
| --- |
| **public class** UserDao {   */\*\*  \* 根据id返回用户  \** ***@param id*** *\** ***@return*** *\** ***@throws*** *IOException  \*/* **public** User findUserById(Integer id) **throws** IOException {  SqlSession session = MyBatisUtil.*getSqlSession*();  User user = session.selectOne(**"com.pcq.entity.UserMapper.selectUser"**, id);  session.close();  **return** user;  }   **public** List<User> findAll() **throws** IOException {  SqlSession session = MyBatisUtil.*getSqlSession*();  List<User> users = session.selectList(**"com.pcq.entity.UserMapper.findAll"**);  session.close();  **return** users;  }   **public int** addUser(User user) **throws** IOException {  SqlSession session = MyBatisUtil.*getSqlSession*();  **int** result = session.insert(**"com.pcq.entity.UserMapper.addUser"**, user);  session.commit();*//执行后需要提交事务* session.close();  **return** result;  }   **public int** addUsers(List<User> users) **throws** IOException {  SqlSession session = MyBatisUtil.*getSqlSession*();  **int** result = session.insert(**"com.pcq.entity.UserMapper.addUsers"**, users);  session.commit();  session.close();  **return** result;  }   **public int** updUser(User user) **throws** IOException {  SqlSession session = MyBatisUtil.*getSqlSession*();  **int** result = session.update(**"com.pcq.entity.UserMapper.updUser"**, user);  session.commit();  session.close();  **return** result;  }   **public int** deleteUserById(Integer id) **throws** IOException {  SqlSession session = MyBatisUtil.*getSqlSession*();  **int** result = session.delete(**"com.pcq.entity.UserMapper.deleteUserById"**, id);  session.commit();  session.close();  **return** result;  } } |

### 3.分页实现

#### 1）通过mysql的limit语句来实现

分页实现需要知道两个参数，当前页（currentPage），以及每一页展示多少项（pageSize）数据。这样可算出查询的起始行为 (currentPage - 1) \* pageSize,查询pageSize个数据。

|  |
| --- |
| <**select id="findAll" parameterType="Map" resultType="User"**>  select *\** from user limit #{startIndex}, #{pageSize} </**select**> |
| **public** List<User> findAll(**int** currentPage, **int** pageSize) **throws** IOException {  SqlSession session = MyBatisUtil.*getSqlSession*();  Map<String, Integer> map = **new** HashMap<String, Integer>();  map.put(**"startIndex"**, (currentPage - 1) \* pageSize);  map.put(**"pageSize"**, pageSize);  List<User> users = session.selectList(**"com.pcq.entity.UserMapper.findAll"**, map);  session.close();  **return** users; } |

#### 通过RowBounds来实现

RowBounds是mybatis分装的一种方式，其原理和第一种实现一样。

此时配置文件无需做任何改变：

|  |
| --- |
| <**select id="findAll" resultType="User"**>  select *\** from user </**select**> |
| **public** List<User> findAll(**int** currentPage, **int** pageSize) **throws** IOException {  SqlSession session = MyBatisUtil.*getSqlSession*();  //同样需要传入起始行，和页面展示数据大小参数  RowBounds rowBounds = **new** RowBounds((currentPage - 1) \* pageSize, pageSize);  List<User> users = session.selectList(**"com.pcq.entity.UserMapper.findAll"**, **null**, rowBounds);  session.close();  **return** users; } |

## 三、配置文件优化

### 1.db.properties

可以通过db.properties文件来记录数据库连接信息

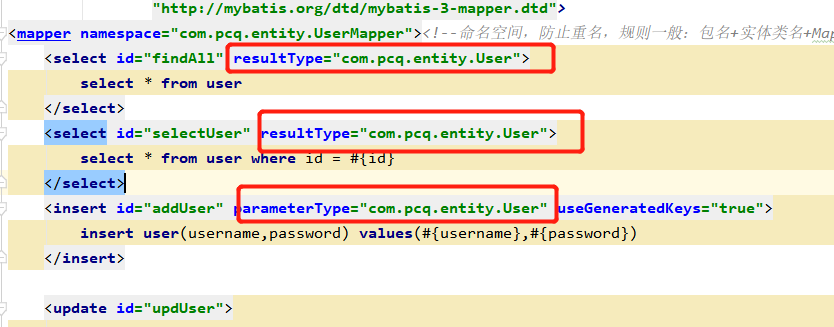
|  |
| --- |
| **driver**=**com.mysql.cj.jdbc.Driver url**=**jdbc:mysql://localhost:3306/test username**=**root password**=**123456** |

然后在Mybatis主配置文件中引用即可

|  |
| --- |
| <**properties resource="db.properties"**/> *<!--可以配置多个环境，也就是多个数据源，一个sqlSessionFactory对应一个环境-->* <**environments default="development"**>  <**environment id="development"**>  <**transactionManager type="JDBC"**/>*<!--事务管理类型-->* <**dataSource type="POOLED"**>*<!--数据源类型 连接池方式，如果是UNPPPLED则每次执行都会申请新的链接，使用完毕再关闭连接。-->* <**property name="driver" value="${driver}"**/>  <**property name="url" value="${url}"** />  <**property name="username" value="${username}"**/>  <**property name="password" value="${password}"**/>  </**dataSource**>  </**environment**> </**environments**> |

### 2.配置别名

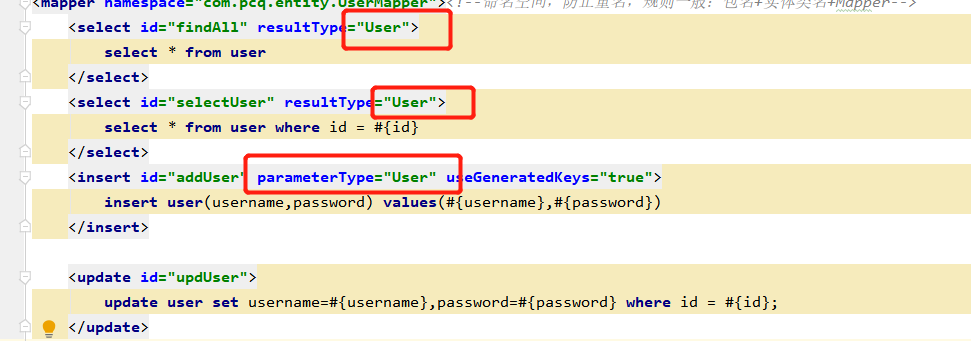
另外可以通过别名的方式，简化mapper配置文件的繁琐：



主配置文件中配置别名

|  |
| --- |
| <**typeAliases**>  <**package name="com.pcq.entity"**/>*<!--默认该包下所有实体类在mapper配置文件中的别名就是类名-->* </**typeAliases**> |

而后mapper配置文件中可简写：



## 四、通过注解来使用MyBatis

使用注解是一种面向接口编程的方式，好处是能够解耦，扩展性好。此时不再需要配置文件，只需要接口。

接口：

|  |
| --- |
| **public interface** UserDao {   */\*\*  \* 根据id返回用户  \** ***@param id*** *\** ***@return*** *\** ***@throws*** *IOException  \*/* @Select(**"select \* from user where id = #{id}"**)  **public** User findUserById(Integer id);   @Select(**"select \* from user"**)  **public** List<User> findAll(**int** currentPage, **int** pageSize);   @Insert(**"insert user(username,password) values(#{username},#{password})"**)  **public int** addUser(User user);    @Insert(**"<script>"** +  **" insert into user (username,password) values\n"** +  **" <foreach collection=\"list\" item=\"user\" separator=\",\">\n"** +  **" (#{user.username}, #{user.password})\n"** +  **" </foreach>"** + **"</script>"** )  **public int** addUsers(@Param(**"list"**)List<User> users);   @Update(**"update user set username=#{username},password=#{password} where id = #{id}"**)  **public int** updUser(User user);   @Delete(**"delete from user where id = #{id}"**)  **public int** deleteUserById(Integer id); } |

代码使用需要通过SqlSession帮我们获得一个代理对象：

|  |
| --- |
| SqlSession **session**; UserDao **userDao**; @Before **public void** init() {  **try** {  **session** = MyBatisUtil.*getSqlSession*();  } **catch** (IOException e) {  e.printStackTrace();  }  **userDao** = **session**.getMapper(UserDao.**class**); } |

主配置文件则需要修改引入接口：

|  |
| --- |
| <**mappers**>  <**mapper class="com.pcq.dao.UserDao"**/> </**mappers**> |

## 联表处理

### 多对一的处理方式

现有学生和老师两个实体类，其中每个学生对应一个老师，即多个学生对应一个老师。

#### 按照结果嵌套处理

studentMapper配置文件：

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"** *?>* **<!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd"*>*** <**mapper namespace="com.pcq.entity.StudentMapper"**>  <**select id="findStudents" resultMap="StudentsTeacher"**>  SELECT s.id sid, s.name sname, s.tid stid, t.id tid,t.name tname FROM student s, teacher t WHERE s.tId = t.id  </**select**>  <**resultMap id="StudentsTeacher" type="Student"**>  <**id column="sid" property="id"**/>  <**result column="sname" property="name"**/>  *<!--关联对象property在Student中的属性名称-->* <**association property="teacher" javaType="Teacher"**>  <**id column="tid" property="id"**/>  <**result column="tname" property="name"**/>  </**association**>  </**resultMap**> </**mapper**> |

代码：

|  |
| --- |
| **public** List<Student> findStudents() {  SqlSession session = **null**;  List<Student> students = **new** ArrayList<Student>();  **try** {  session = MyBatisUtil.*getSqlSession*();  students = session.selectList(**"com.pcq.entity.StudentMapper.findStudents"**);  **return** students;  } **catch** (IOException e) {  e.printStackTrace();  } **finally** {  **if**(**null** != session) {  session.commit();  session.close();  }  }  **return** students; } |

#### 按照查询嵌套处理

studentMapper配置文件：

|  |
| --- |
| *<!--按照查询嵌套-->* <**select id="findStudents" resultMap="StudentsTeacher"**>  select *\** from student </**select**> <**resultMap id="StudentsTeacher" type="Student"**>  <**association property="teacher" column="tid" javaType="Teacher" select="com.pcq.entity.TeacherMapper.findTeacherById"**>  </**association**> </**resultMap**> |

teacherMapper配置文件：

|  |
| --- |
| <**mapper namespace="com.pcq.entity.TeacherMapper"**>  <**select id="findTeacherById" resultType="Teacher"**>  select *\** from teacher where id = #{id}  </**select**> </**mapper**> |

**注：第一种执行一次查询，第二种方式执行了两次查询**

### 一对多的处理方式

#### 按照结果

teacherMapper.xml

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"** *?>* **<!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd"*>*** <**mapper namespace="com.pcq.entity.TeacherMapper"**>  <**select id="findTeacherById" resultMap="TeacherStudents"**>  SELECT s.id sid, s.name sname, s.tid stid, t.id tid,t.name tname FROM student s, teacher t WHERE s.tId = t.id and t.id = #{id}  </**select**>  <**resultMap id="TeacherStudents" type="Teacher"**>  <**id column="tid" property="id"**/>  <**result column="tname" property="name"**/>  <**collection property="students" ofType="Student"**>  <**id column="sid" property="id"**/>  <**result column="sname" property="name"**/>  </**collection**>  </**resultMap**> </**mapper**> |

#### 按照查询

TeacherMapper.xml

|  |
| --- |
| <**select id="findTeacherById" resultMap="TeacherStudents"**>  select *\** from teacher where id = #{id} </**select**> <**resultMap id="TeacherStudents" type="Teacher"**>  <**id column="id" property="id"**/>  <**collection property="students" javaType="ArrayList" ofType="Student" column="id" select="com.pcq.entity.StudentMapper.findStudentsByTid"**>  </**collection**> </**resultMap**> |

studentMapper.xml

|  |
| --- |
| <**select id="findStudentsByTid" resultType="Student"**>  select *\** from student where tid = #{tid} </**select**> |

## 动态sql

动态sql是基于OGNL来通过标签动态传入sql，达到灵活性高的目的。

配置文件：

|  |
| --- |
| <**select id="findStudentByCondition" parameterType="map" resultType="Student"**>  select *\** from student  <**where**>  <**if test="name != null"**>  name like concat('%',#{name}, '%')  </**if**>  </**where**> </**select**> |

dao代码：

|  |
| --- |
| **public** Student findStudentByCondition(Map<String, Object> map) {  SqlSession session = **null**;  Student student = **null**;  **try** {  session = MyBatisUtil.*getSqlSession*();  student = session.selectOne(**"com.pcq.entity.StudentMapper.findStudentByCondition"**, map);  } **catch** (IOException e) {  e.printStackTrace();  } **finally** {  **if**(**null** != session) {  session.close();  }  }  **return** student; } |

测试代码：

|  |
| --- |
| @Test **public void** testFindStudentByCondition() {  Map<String, Object> map = **new** HashMap<String,Object>();  map.put(**"name"**, **"甫"**);  Student student = **studentDao**.findStudentByCondition(map);  System.***out***.println(student); } |

**更多标签用法参考官方使用文档。**