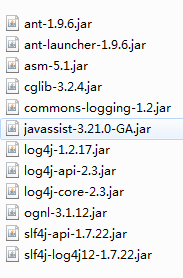
## 1.搭建Mybatis

1.导包

核心包



依赖包



数据库驱动包



4.在src下新建Mybatis核心配置文件mybatis-config.xml

从官网复制相关配置

<?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="${driver}"/>

<property name="url" value="${url}"/>

<property name="username" value="${username}"/>

<property name="password" value="${password}"/>

</dataSource>

</environment>

</environments>

<mappers>

<mapper resource="org/mybatis/example/BlogMapper.xml"/>

</mappers>

</configuration>

5.修改对应配置

****

## 2.获取SqlSession对象

1.在Util包下创建一个class( MyUtil.class )

2.在类中新建一个方法，返回SqlSessionFactory

**public** **static** SqlSessionFactory getSqlSessionFactory() **throws** IOException{

String resource = "mybatis-config.xml";

//读取配置文件

InputStream inputStream = Resources.*getResourceAsStream*(resource);

//通过SqlSessionFactoryBuilder创建SqlSessionFactory

// SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();

//

// SqlSessionFactory sqlSessionFactory = builder.build(inputStream);

SqlSessionFactory sqlSessionFactory =

**new** SqlSessionFactoryBuilder().build(inputStream);

**return** sqlSessionFactory;

}

对应的官网代码

String resource = "org/mybatis/example/mybatis-config.xml";

InputStream inputStream = Resources.getResourceAsStream(resource);

sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);

3.创建一个方法，返回SqlSession对象

**public** **static** SqlSession openSession() **throws** IOException{

//从 SqlSessionFactory 中获取 SqlSession

//SqlSession session = sqlSessionFactory.openSession();

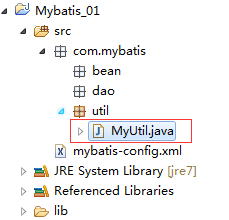
SqlSession sqlSession = *getSqlSessionFactory*().openSession();

**return** sqlSession;

}

官网对应代码

SqlSession session = sqlSessionFactory.openSession();



## 3.简单的查询

1.创建实体类

2.创建实体类对应的Mapper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文件的命名 实体类名字+Mapper.xml -->

<!-- namespace 访问该Mapper文件的路径 -->

<mapper namespace=*"com.mybatis.bean.EmpMappper"*>

<!--

id 唯一标识

id 访问该SQL语句的标识

resultType 返回数据的类型

parameterType 参数类型

获取参数语法 #{参数名}

-->

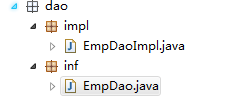
<select id=*"selectEmp"* resultType=*"com.mybatis.bean.EmpBean"* parameterType=*"int"*>

select \* from emp where id = #{id}

</select>

</mapper>

3.编写接口Interface





4.编写接口的实现类

@Override

**public** EmpBean getEmpById(**int** id) {

**try** {

//获取SqlSesssion

SqlSession sqlSession = MyUtil.*openSession*();

//两个参数

//1.你需要调用哪一个Mapper文件中的哪一条SQL语句

// Mapper文件的 namespace + "." + 文件中SQL语句的id属性

// namespace = com.mybatis.bean.EmpMappper id = selectEmp

//2.查询参数

EmpBean empBean =

sqlSession.selectOne("com.mybatis.bean.EmpMappper.selectEmp", id);

System.***out***.println(empBean.getId());

} **catch** (IOException e) {

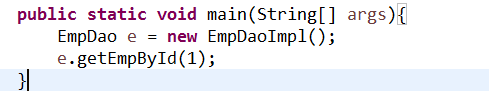
e.printStackTrace();

}

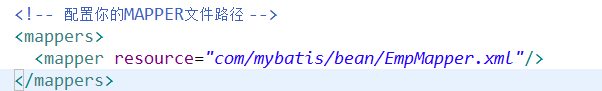
**return** **null**;

}

5.编写测试代码



6.修改mybatis-config.xml 配置文件中的mappers配置



路径一定要为 “/” 斜线

## 4.Insert

1.编写SQL语句

<!--

id 唯一标识

id 访问该SQL语句的标识

parameterType 参数类型 可传递一个对象 !!区分大小写

useGeneratedKeys="true" 主键自动递增

-->

<insert id=*"addEmp"* parameterType=*"com.mybatis.bean.EmpBean"* useGeneratedKeys=*"true"*>

insert into emp

(empname,dept\_id,role\_id,job,is\_admin,password,username,photo)

values

(

#{empName},#{deptId},#{roleId},#{job},

#{isAdmin},#{password},#{username},#{photo}

)

</insert>

2.编写接口方法

3.编写实现类

@Override

**public** **void** addEmp(EmpBean empBean) {

**try**{

//获取SqlSession

SqlSession sqlSession = MyUtil.*openSession*();

//执行SQL

sqlSession.insert("com.mybatis.bean.EmpMapper.addEmp", empBean);

//提交事务

sqlSession.commit();

//关闭sqlSession

sqlSession.close();

}**catch**(Exception e){

e.printStackTrace();

}

}

4.编写测试代码

//参数

EmpBean empBean = **new** EmpBean();

empBean.setDeptId(1);

empBean.setEmpName("mybatis");

empBean.setIsAdmin("0");

empBean.setJob("程序员");

empBean.setRoleId("1,2");

empBean.setUsername("mybatis");

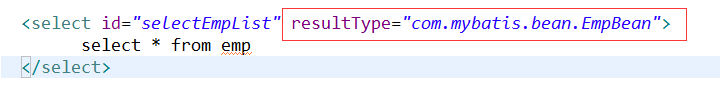
empBean.setPassword("123");

//新增

empDao.addEmp(empBean);

## 5.查询所有

1.编写SQL语句



注意：返回结果resultType不是List,而是List中的对象类型

2.编写实现类

@Override

**public** List<EmpBean> selectAll() {

List<EmpBean> list = **new** ArrayList<EmpBean>();

**try**{

//获取SqlSession

SqlSession sqlSession = MyUtil.*openSession*();

list = sqlSession.selectList("com.mybatis.bean.EmpMapper.selectEmpList");

**for** (EmpBean empBean : list) {

System.***out***.println(empBean.toString());

}

sqlSession.close();

}**catch**(Exception e){

e.printStackTrace();

}

**return** list;

}

3.编写测试代码

//查询所有

empDao.selectAll();

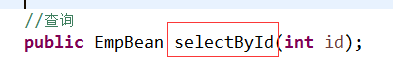
## 6.面向接口编程

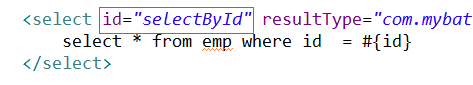
1.Mapper文件的namespace要和接口的完整路径对应 com.mybatis.dao.inf.EmpDao 包括名字



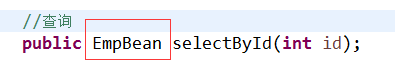


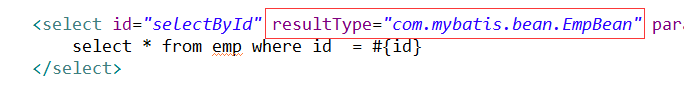
2.接口中的方法名字 要和 Mapper文件中SQL语句的ID对应





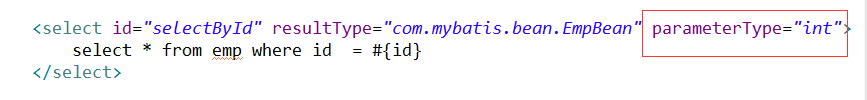
3.接口中方法的返回值 要和Mapper文件中的resultType对应





4.接口中方法的参数 要和Mapper文件中的parameterType对应





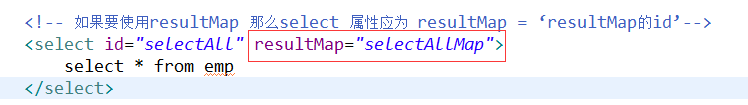
## 7.别名扫描与Mapper文件扫描





## 8.简单的ResultMap





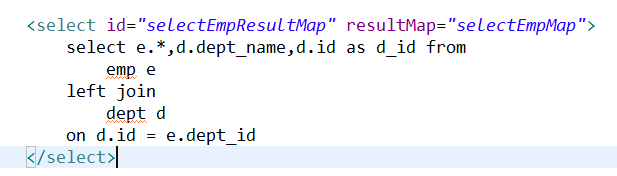
## 9.ResultMap 一对一(表连接)

在ResultMap中使用 association标签

EmpBean 实体类中存在一对一关系



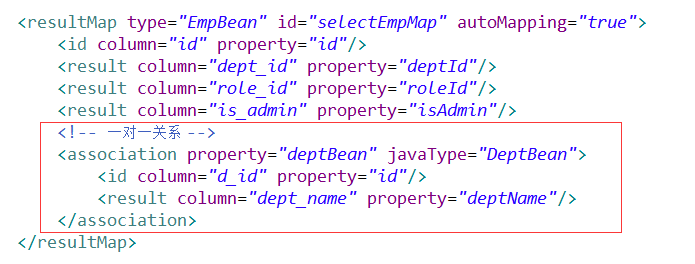
定义SQL语句



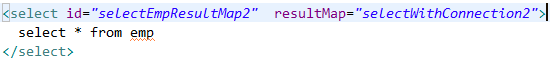
定义返回的ResultMap

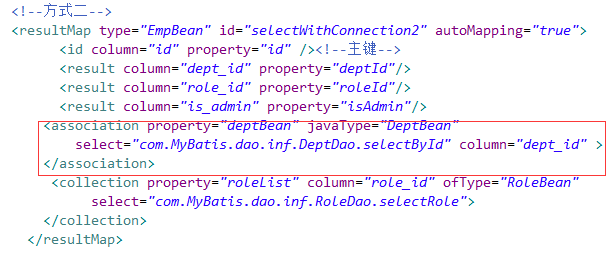
Column为数据库字段，property为bean模型属性

Bean模型属性与数据库字段不相同的，需要写一下result标签，其他的可以用自动映射，也可以全部写上

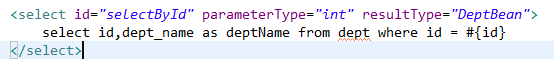


**一对一:子查询方式**





Select的子查询



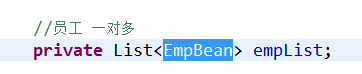
## 10.ResultMap 一对多

在resultMap 中使用 collection标签

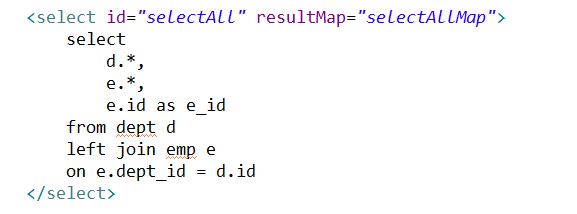
DeptBean 实体类中存在一对多关系

站在部门的角度，部门和员工是一对多关系

(一个部门对应多个员工)



编写SQL



编写ResultMap 映射结果集



## 11.ResultMap 多对一

以上都是表连接的方式，但有时候需要用到子查询

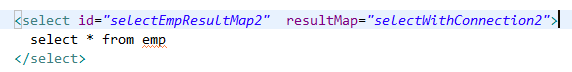
EmpBean存在多对一关系

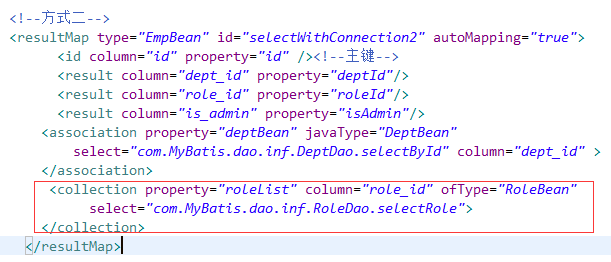
站在权限的角度,权限和员工是多对一关系

(多个权限对应一个员工,即一个员工有多个权限)

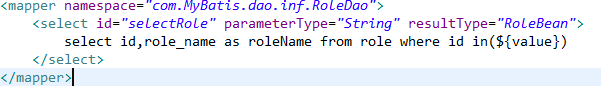


子查询语句:





用到的collection标签，select属性,需要单独写select语句



注意：是select \* from emp

作为

select id,role\_name as roleName from role where id in(${value})

的子查询