1. mybatis是一个orm（object relation mappin）框架,持久层框架，相较于jdbcTemplate他有更强大的的功能，是一款数据库框架
2. Maybtis他将数据库里面的每条记录映射为一个对象，将数据库语句分离出来，由mybatis来操作
3. hiberanate是全自动的数据库，他的映射会映射到所有的字段，相当于select\*，使用hql查询语句，比较不灵活，对于数据库语句的优化，用hql比较困难，mybatis任然将数据库的语句交由程序员编写，虽然是半自动化，却大大增强了灵活性。

4、orm：对象关系映射，将数据库中每一条记录映射为一个对象

# Mybatis框架提示模板

|  |
| --- |
|  |

# Mybatis环境搭建（\*）

1. 导入jar包



2书写主配置文件mybatis.Xml和properities

<?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>

<!-- 配置数据源，environments为大环境，里面包含多个数据源mysql oracle -->

<environments default=*"mysql"*>

<!--配置mysql environment为一个环境，里面包含一个数据库 -->

<environment id=*"mysql"*>

<transactionManager type=*"JDBC"*></transactionManager>

<dataSource type=*"POOLed"*>

<property name=*"driver"* value=*"com.mysql.jdbc.Driver"*/>

<property name=*"url"* value=*"jdbc:mysql:///dt48\_mybatis"*/>

<property name=*"username"* value=*"root"*/>

<property name=*"password"* value=*"123456"*/>

</dataSource>

</environment>

<!-- <environment id="oracle"></environment> -->

</environments>

<!-- 主配置文件关联局部配置文件 -->

<!-- mappers表示关联多个局部配置文件 -->

<mappers>

<!--mapper表示关联一个配置文件，这里要用/隔开，并且从src下开始 -->

<mapper resource=*"mybatis1/dao/userdaoimpl.xml"*/>

</mappers>

</configuration>

3书写局部配置文件

<?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">

<!-- 局部配置文件，一个mapper对应一个dao层的实现类 namespace为指定这个配置文件那个dao层实现类映射-->

<mapper namespace="mybatis1.dao.userdaoimpl">

<!-- 数据库的语句，在mybatis里面有增删改查四个标签与之对应，id对应方法名，resultType对应着数据库中一条记录的类型 -->

<select id="getAllUser" resultType="mybatis1.entity.User">

select \* from users;

</select>

</mapper>

第四步：启动Mybatis框架，测试

@Test

public void getAllUser() throws IOException {

// 1、启动mybatis框架

// SqlSession---->SqlSessionFactory---->SqlSessionFactoryBuilder

SqlSessionFactoryBuilder sfb = new SqlSessionFactoryBuilder();

// 将mybatis.xml文件转化成流

InputStream ins = Resources.getResourceAsStream("mybatis.xml");

SqlSessionFactory ssf = sfb.build(ins);

SqlSession session = ssf.openSession();

// 2、调用局部配置文件中的sql语句

List<User> userList = session.selectList("cn.java.dao.impl.UserDaoImpl.getAllUser");

for (User user : userList) {

System.out.println(user);

}

}

第五：原理

Mybatis首先读取主配置文件，然后得到数据库连接，再从主配置文件的mappers里面得到所有的子配置文件，一并读入容器编译，然后调用方法的之后只要指出是哪个配置文件的什么方法，mybatis就会自动执行其中的数据库语言得到结果返回出来，

如cn.java.dao.impl.UserDaoImpl.getAllUser

cn.java.dao.impl.UserDaoImpl：为指定是哪个配置文件

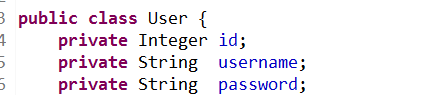
getAllUser：指定执行什么数据库查询语句（或者说指定方法）

# Mybatis返回值的类型的设计，与#和$的区别



我们仔细观察数据库，他们的值都是键值对的关系，（id=6,username=”王二麻子”，password=“123”），所以对于数据库中每一条数据的存储方式有以下两种方案

1. **应用实体类来存储**



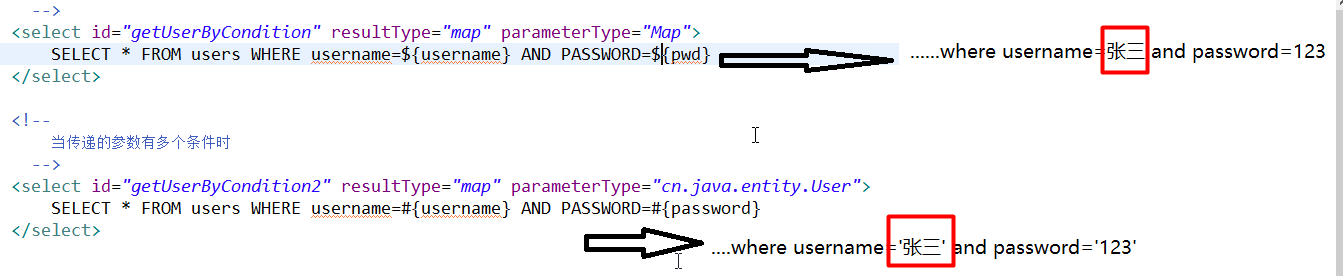
将实体类里面的字段和数据库字段对应，然后进行映射，通过set方法注入装满对象，一条记录对应一个对象，那么多条记录为list<object>

**2、采用map来存储返回值**

每条记录都采用一个map<String,Object> 来存储，将键值对的键放入map的key，值放入value，那么任何一条数据库记录都可以放入一个map里面，那么从数据库中拿出多个记录的时候，则返回list<map<String,Object>

**3、#和$的区别**

#获取参数值时，默认会把参数当做字符串处理，在参数之间会添加单引号；



#获取参数时，可以防止sql注入;$不能够防止sql注入。因此能够用#获取参数时，尽量使用#

# Mybatis查询

**不带条件的查询，返回多条记录**

@Test

**public** **void** getAllUser() **throws** IOException {

// 1、启动mybatis框架

// SqlSession---->SqlSessionFactory---->SqlSessionFactoryBuilder

SqlSessionFactoryBuilder sfb = **new** SqlSessionFactoryBuilder();

// 将mybatis.xml文件转化成流

InputStream ins = Resources.*getResourceAsStream*("mybatis.xml");

SqlSessionFactory ssf = sfb.build(ins);

SqlSession session = ssf.openSession();

List<User> userlist =session.selectList("mybatis1.dao.userdaoimpl.getAllUser");

**for** (User user : userlist) {

System.***out***.println(user);

}

}

//配置文件

<select id="getAllUser" resultType="mybatis1.entity.User">

select \* from users;

</select>

**不带条件的查询返回一条记录**

//查询单挑记录，不带参数

@Test

public void getUserById() throws IOException {

// 1、启动mybatis框架

// SqlSession---->SqlSessionFactory---->SqlSessionFactoryBuilder

SqlSessionFactoryBuilder sfb = new SqlSessionFactoryBuilder();

// 将mybatis.xml文件转化成流

InputStream ins = Resources.getResourceAsStream("mybatis.xml");

SqlSessionFactory ssf = sfb.build(ins);

SqlSession session = ssf.openSession();

User user = session.selectOne("mybatis1.dao.userdaoimpl.getUserById");

System.out.println(user);

}

<select id="getUserById" resultType="mybatis1.entity.User">

select \* from users where id=6;

</select>

**带条件的查询**

//多条件查询,用map来存储参数，也可以用对象来存储

@Test

**public** **void** getUserByconfing() **throws** IOException {

// 1、启动mybatis框架

// SqlSession---->SqlSessionFactory---->SqlSessionFactoryBuilder

SqlSessionFactoryBuilder sfb = **new** SqlSessionFactoryBuilder();

// 将mybatis.xml文件转化成流

InputStream ins = Resources.*getResourceAsStream*("mybatis.xml");

SqlSessionFactory ssf = sfb.build(ins);

SqlSession session = ssf.openSession();

Map<String, Object> config = **new** HashMap<String,Object>();

config.put("username", "王二麻子");

config.put("password", "123");

User user= session.selectOne("mybatis1.dao.userdaoimpl.getUserByconfing",config);

System.***out***.println(user);

}

<!-- 带多个参数的查询，用map的key来得到参数，记得要写参数类型parameterType -->

<select id=*"getUserByconfing"* resultType=*"mybatis1.entity.User"* parameterType=*"map"* >

select \* from users where username=#{username} and password = #{password};

</select>

**来进行模糊查询**

//模糊查询,用map来存储参数

@Test

**public** **void** getUserByMH() **throws** IOException {

// 1、启动mybatis框架

// SqlSession---->SqlSessionFactory---->SqlSessionFactoryBuilder

SqlSessionFactoryBuilder sfb = **new** SqlSessionFactoryBuilder();

// 将mybatis.xml文件转化成流

InputStream ins = Resources.*getResourceAsStream*("mybatis.xml");

SqlSessionFactory ssf = sfb.build(ins);

SqlSession session = ssf.openSession();

Map<String, Object> config = **new** HashMap<String,Object>();

config.put("password", "12");

List<User> user = session.selectList("mybatis1.dao.userdaoimpl.getUserByMH",config);

**for** (User user2 : user) {

System.***out***.println(user);

}

}

<!-- 模糊查询，用map的key来得到参数，使用concat来拼接条件 -->

<select id=*"getUserByMH"* resultType=*"mybatis1.entity.User"* parameterType=*"map"* >

select \* from users where password like concat('%',#{password},'%');

</select>

**动态查询**

&lt;代表小于（low than）

&gt；代表大于（great than）

//动态查询

@Test

**public** **void** getComputerDynamic() **throws** IOException {

Init();

Map<String, Object> param = **new** HashMap<String,Object>();

param.put("computer\_name", "联想(Lenovo)拯救者R720 15.6英寸游戏");

param.put("brand", "联想");

param.put("run\_mem", "8");

param.put("price", "6500");

List<Map<String, Object>> userlist = session.selectList("mybatis1.dao.userdaoimpl.getComputerDynamic",param);

**for** (Map<String, Object> map : userlist) {

System.***out***.println(map);

}

}

<!-- 动态查询 &lt;表示小于，&gt;表示大于-->

<select id=*"getComputerDynamic"* resultType=*"map"* parameterType=*"map"*>

select \* from computers

<where>

<if test=*"computer\_name"*>

computer\_name = #{computer\_name}

</if>

<if test=*"brand"*>

and brand = #{brand}

</if>

<if test=*"run\_mem"*>

and run\_mem = #{run\_mem}

</if>

<if test=*"price"*>

and price &lt;= #{price}

</if>

</where>

</select>

# Mybatis插入，修改，删除（没有resoultType，必须提交事务）

在mybatis的增删改中在子配置文件里面insert delete update 均没有返回值标签resoultType,默认返回值为int，不需要指定，返回值表示影响的行数。在进行mybatis增删改的时候必须提交事务

**//添加一条数据**

@Test

**public** **void** adduser() **throws** IOException {

Init();

Map<String, Object> config = **new** HashMap<String,Object>();

config.put("username", "李子鸣");

config.put("password", "123456");

**int** result = session.insert("mybatis1.dao.userdaoimpl.adduser", config);

session.commit();

System.***out***.println(result);

}

<!-- 添加一条数据 -->

<insert id=*"adduser"* parameterType=*"map"*>

insert into users set username = #{username} , password = #{password};

</insert>

//修改一条数据

@Test

**public** **void** updateuser() **throws** IOException {

Init();

Map<String, Object> config = **new** HashMap<String,Object>();

config.put("username", "李子鸣");

config.put("password", "123");

**int** result = session.update("mybatis1.dao.userdaoimpl.updateuser", config);

session.commit();

System.***out***.println(result);

}

<!-- 修改一条数据-->

<update id=*"updateuser"* parameterType=*"map"*>

update users set password = #{password} where username = #{username};

</update>

//动态修改

@Test

**public** **void** updateComputerDynamic() **throws** IOException {

Init();

Map<String, Object> param = **new** HashMap<String,Object>();

param.put("computer\_name", "拯救者R720");

param.put("brand", "联想");

param.put("run\_mem", "8");

param.put("price", "6500");

param.put("id", 2);

**int** resoult = session.update("mybatis1.dao.userdaoimpl.updateComputerDynamic",param);

session.commit();

System.***out***.println(resoult);

}

<update id=*"updateComputerDynamic"* parameterType=*"map"*>

update computers

<set>//可去除多余的“,”

<if test=*"computer\_name"*>

computer\_name = #{computer\_name},

</if>

<if test=*"brand"*>

brand = #{brand},

</if>

<if test=*"run\_mem"*>

run\_mem = #{run\_mem},

</if>

<if test=*"price"*>

price = #{price}

</if>

</set>

where id = #{id}

</update>

//删除一条数据

@Test

**public** **void** deleteuser() **throws** IOException {

Init();

Map<String, Object> config = **new** HashMap<String,Object>();

config.put("username", "李子鸣");

**int** result = session.delete("mybatis1.dao.userdaoimpl.deleteuser", config);

session.commit();

System.***out***.println(result);

}

<!-- 删除一条数据 -->

<delete id=*"deleteuser"* parameterType=*"map"*>

delete from users where username = #{username}

</delete>

//mybatis的批量增删改

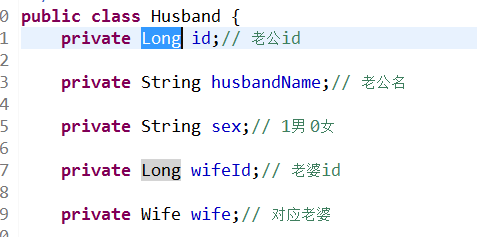
# 六、sql标签与include标签

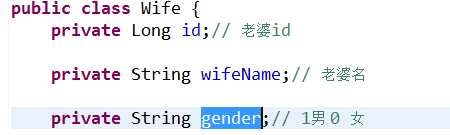
|  |
| --- |
|  |

# 一对一的关联关系

一个对象里面有另一个对象的的引用，如何实现对联级属性的赋值

例子，husband了里面有wife对象的引用





1. 使用resultMap（resoult mapper）来实现联级属性赋值

<!--resultMap 结果映射，type表示要映射的类，id是别名，方便以后引用 -->

<resultMap type=*"mybatis1.entity.Husband"* id=*"hus"*>

<!--result对应类里面一个字段，javaType表示该字段的类型， column：对应数据库中的类-->

<result property=*"id"* javaType=*"Long"* column=*"id"*/>

<result property=*"husbandName"* javaType=*"String"* column=*"husbandName"*/>

<result property=*"sex"* javaType=*"String"* column=*"sex"*/>

<result property=*"wifeId"* javaType=*"Long"* column=*"wifeId"*/>

<!--association实现对联级属性的赋值 -->

<!-- property：数据库中引用的字段的名字，javaType：他的类型，对应全类名 -->

<association property=*"wife"* javaType=*"mybatis1.entity.Wife"*>

<result property=*"id"* javaType=*"Long"* column=*"id"*/>

<result property=*"wifeName"* javaType=*"String"* column=*"wifeName"*/>

<result property=*"gender"* javaType=*"String"* column=*"gender"*/>

</association>

</resultMap>

1. 使用这个resoultmap来接收结果并对实体类赋值

<select id=*"one2one"* resultMap=*"hus"*>

select h.\*,w.id as wid,w.wifeName,w.gender from husband as h left join wife as w on h.wifeId = w.id

</select>

3、前台显示

//一对一的关联关系

@Test

**public** **void** one2one() **throws** IOException {

Init();

List<Husband> husbands = session.selectList("mybatis1.dao.userdaoimpl.one2one");

**for** (Husband husband : husbands) {

System.***out***.println(husband);

}

}

4、一对一的关联关系如果不对实体类进行映射，也可以直接用map接收返回的数据，因为不论怎样，数据库查询返回的都是键值对，均可以用map来接收，因此这个查询和普通查询无异。

如

//一对一的关联关系

@Test

**public** **void** one2onebyMap() **throws** IOException {

Init();

List<Map<String, Object>> husbandlist = session.selectList("mybatis1.dao.userdaoimpl.one2onebyMap");

**for** (Map<String, Object> map : husbandlist) {

System.***out***.println(map);

}

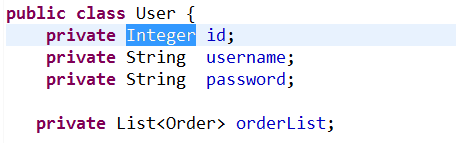
}

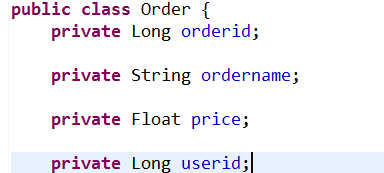
<select id=*"one2onebyMap"* resultType=*"map"*>

select h.\*,w.id as wid,w.wifeName,w.gender from husband as h left join wife as w on h.wifeId = w.id

</select>

# 一对多的关联关系





数据库查询结果，要注入到user



案例：一个user有多张订单，对user实体类进行赋值

1、书写resoultmap来接收赋值（association是为引用对象赋值，collection标签是为集合属性赋值 ）

<resultMap type=*"mybatis1.entity.User"* id=*"user"*>

<result property=*"id"* javaType=*"Integer"* column=*"id"*/>

<result property=*"username"* javaType=*"String"* column=*"username"*/>

<result property=*"password"* javaType=*"String"* column=*"password"*/>

<!-- collection对集合属性进行赋值，ofType集合里面一条记录的类型 -->

<collection property=*"orderList"* ofType=*"mybatis1.entity.Order"*>

<id column=*"orderId"* property=*"orderid"* jdbcType=*"BIGINT"* />

<result column=*"orderName"* property=*"ordername"* jdbcType=*"VARCHAR"* />

<result column=*"price"* property=*"price"* jdbcType=*"REAL"* />

<result column=*"userId"* property=*"userid"* jdbcType=*"BIGINT"* />

</collection>

</resultMap>

2、书写查询语句

<select id=*"one2many"* resultMap=*"user"*>

select \* from users as u join orders as o on u.id=o.userId

</select>

3、书写方法

//一对多的关联关系

@Test

**public** **void** one2many() **throws** IOException {

Init();

User user = session.selectOne("mybatis1.dao.userdaoimpl.one2many");

System.***out***.println(user);

}

# 八、P6spy工具使用

|  |
| --- |
| **第一步：导入p6spy的jar包**    **第二步：导入p6spy的配置文件**    **第三步：改写mybatis.xml** |

# 九、静态工厂模式与单例模式(\*)

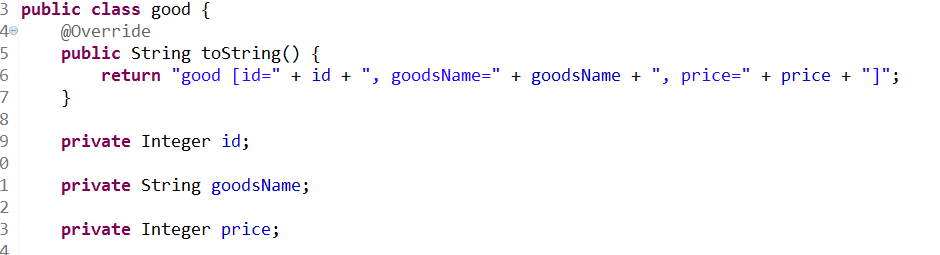
|  |
| --- |
| **工厂模式：**   1. 降低模块之间耦合 2. 利于拓展 |
| **单例模式:** |

# 十、面试题:

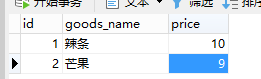
|  |
| --- |
| 1. **Mybatis中#与$取值的区别** 2. **Mybatis与Hibernate的区别：** |
| **Mybatis--->Spring(IOC、DI、AOP)[spring boot]----->springMVC** |

1. 当数据库字段两个单词之间用\_隔开，而实体类使用驼峰命名的时候，使用实体类接收数据会导致注入失败，需要使用resoultMap标签进行映射 ，或者在数据库语句起别名。

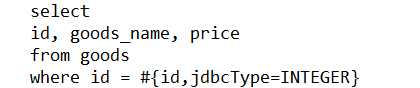
如

商品实体类 ，使用驼峰命名 

数据库，使用\_隔开



数据库查询语句



解决方案1--》使用resoultmap（建议使用）

<resultMap id=*"BaseResultMap"* type=*"mybatis1.entity.good"* >

<id column=*"id"* property=*"id"* jdbcType=*"INTEGER"* />

<result column=*"goods\_name"* property=*"goodsName"* jdbcType=*"VARCHAR"* />

<result column=*"price"* property=*"price"* jdbcType=*"INTEGER"* />

</resultMap>

<select id=*"selectByPrimaryKey"* resultMap=*"BaseResultMap"* parameterType=*"map"* >

select

id, goods\_name, price

from goods

where id = #{id,jdbcType=INTEGER}

</select>

解决方案2、改写sql语句,起别名

select

id, goods\_name as goodsName, price

from goods

where id = #{id,jdbcType=INTEGER}