mybatis学习日记

- 1.1 什么是 MyBatis ？
  MyBatis 是支持定制化 SQL、存储过程以及高级映射的优秀的持久层框架。MyBatis 避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。MyBatis 可以对配置和原生Map使用简单的 XML 或注解，将接口和 Java 的 POJOs(Plain Old Java Objects,普通的 Java对象)映射成数据库中的记录。

> - **SqlMapConfig.xml**：mybatis的核心配置文件，主要用于配置数据库连接等一些信息
> - **Mappers.xml**：这些配置文件有多个，每一个配置文件对应数据库当中的一张表，主要用于我们数据库中的字段与我们JavaBean实体类中的字段的相互映射，mappers.xml一定要被sqlMapConfig.xml加载
> - **SqlSessionFactory**：工厂类，主要用于产生我们的sqlSession
> - **SqlSession**：mybatis里面的一个核心的类，用于操作我们的数据库，通过executor来操作我们的数据库，executor是mybaits里面一个封装的对象，不需要我们去处理
> - **MappedStatement**：主要作用就是连接我们的输入映射与我们的输出映射，将我们的输入映射拿到之后，去执行我们数据库操作，然后将执行后的结果包装成我们的输出映射
> - mybatis的网址: http://www.mybatis.org/mybatis-3/zh/index.html
>
> ## 二 入门程序
>
> **在写程序之前,我们要先准备环境:**

主键查询

- 2.1 Maven : Web项目
- 2.2 mybatis包: http://mvnrepository.com/artifact/org.mybatis/mybatis 选择相应的版本
- 2.3 mysql包: http://mvnrepository.com/artifact/mysql/mysql-connector-java/5.1.38
- 2.4 在资源目录下配置
  - mapper.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">
  <!-- namespace:名称空间  -->
<mapper namespace="test">
 </mapper>
```

  - SqlMapConfig.xml

```
    <?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"/>
          <!-- 表示使用连接池连接数据库 -->
          <dataSource type="POOLED">
          <!-- 驱动的引用 -->
            <property name="driver" value="com.mysql.jdbc.Driver"/>
            <property name="url" value="jdbc:mysql://localhost:3306/mybatis?characterEncoding=utf-8"/>
            <!-- 数据库的用户名 -->
            <property name="username" value="root"/>
            <!-- 数据库密码 -->
            <property name="password" value="password"/>
          </dataSource>
        </environment>
      </environments>
      <mappers>
          <!-- 一组 mapper 映射器（这些 mapper 的 XML 文件包含了 SQL 代码和映射定义信息） -->
        <mapper resource="mapper.xml"/>
      </mappers>
    </configuration>>
```

- log4j.properties 在控制台打印我们的SQL语句

```
# Global logging configuration
log4j.rootLogger=DEBUG, stdout
# Console output...
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n
```

还需要4个jar包:

```
<dependency>
    <groupId>org.bgee.log4jdbc-log4j2</groupId>
    <artifactId>log4jdbc-log4j2-jdbc4.1</artifactId>
    <version>1.16</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.13</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.7.5</version>
</dependency>

<!-- https://mvnrepository.com/artifact/log4j/log4j -->
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.16</version>
</dependency>
```

- 2.5 创建javaBean类,**属性name要和数据库字段的name保持一致**
- 2.6 创建测试类 在这里我们使用单元测试

```
 @Test
public void getUserId() throws IOException{
    //获取工厂对象
    SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();
    //使用builder读取Mybatis的核心配置文件,然偶才可以创建SqlSesiionFactory
    //获取SqlSessionFactory
    SqlSessionFactory build = builder.build(Resources.getResourceAsStream("SqlMapConfig.xml"));
    //获取SqlSession
    SqlSession session = build.openSession();
    /**
     * 第一个参数:sql的Id
     * 第二个参数: 传入的参数值
     */
    Object session = session.selectOne("getUserId", 67);
    session.close();
    System.out.println(selectOne.toString());
}
```

- 2.7 在mapper.xml中写入映射语句

```
 <mapper namespace="test">
    <!--
        id: sqlId 和session.selectOne("getUserId", 68);的第一个参数保持一致
        resultType: 返回结果的类型,如果是对象的话我们就要写类的全限定名
        parameterType: 参数类型 , 可以自动拆装箱
        #{id} : 占位符 .
        select还有许多的属性,这里只写必须要的,如果想看其他的属性可以去官网查看
     -->
     <select id="getUserId" resultType="com.zhz.demo.User" parameterType="int">
         select * from user where id = #{id}
     </select>
</mapper>
```

到这里我们的入门程序就结束了,有没有感觉到特别简单那!~~~

三 模糊查询

- 3.1 测试类:

```
*@Test
```

```
public void getUserLike() throws IOException{

    SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();

    SqlSessionFactory build = builder.build(Resources.getResourceAsStream("SqlMapConfig.xml"));

    SqlSession openSession = build.openSession();

    List<Object> user = openSession.selectList("getUserLike","海" );
    openSession.close();
    for (Object object : user) {
        System.out.println(object.toString());
    }
}
```

- 3.2 mapper.xml:

```
 <select id="getUserLike" resultType="com.zhz.demo.User" parameterType="String">
  <!--
      #{}与${}的区别:

     #{}用作占位符，在接受简单数据类型的时候，大括号里面的参数可以随便写
     ${}表示连接符，连接我们传入的参数与左右两边的字符串，当我们传入简单数据类型的时候，大括号里面的值只能写value
   -->
      select * from user where username like  '%${value}%'
  </select>
```

模糊查询要特变注意'%${value}%'的写法,不要写错,还有一点,在写程序的过程中,能复制的就不要手敲,因为手敲容易敲错,导致程序出错

四 插入数据

- 4.1 测试类

```
    @Test
    public void insertUser() throws IOException{

        SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();

        SqlSessionFactory build = builder.build(Resources.getResourceAsStream("SqlMapConfig.xml"));

        SqlSession session = build.openSession();

        User user = new User();
        //id不用设置,他是自增长的
        user.setUserName("张大大");
        user.setPassword("password");

        session.insert("insertUser", user);
        //数据库操作,出了查询外,其他操作都需要提交
        session.commit();

        session.close();
    }
```

- 4.2 mapper.xml

```
<!-- 没有反回值,可以不写`-->
<insert id="insertUser" parameterType="com.zhz.demo.User">
     insert into user (username,password) values (#{username},#{password})
 </insert>
```

五 自增主键返回

SELECT LAST_INSRT_ID() 表示最后一次插入数据的主键

- 5.1 测试类

```
        .......
        session.commit();
        //获取主键
        System.out.println(user.getId());
        session.close();
        ....
```

其他代码我就省略了,在提交之后就可以获取主键的值了

- 5.2 mapper.xml

  - keyProperty（仅对 insert 和 update 有用）唯一标记一个属性，MyBatis 会通过 getGeneratedKeys 的返回值或者通过 insert 语句的 selectKey 子元素设置它的键值，默认：unset。如果希望得到多个生成的列，也可以是逗号分隔的属性名称列表。
    指定我们对象的哪个属性,这里是User对象的id属性
  - keyColumn（仅对 insert 和 update 有用）通过生成的键值设置表中的列名，这个设置仅在某些数据库（像 PostgreSQL）是必须的，当主键列不是表中的第一列的时候需要设置。如果希望得到多个生成的列，也可以是逗号分隔的属性名称列表。
    简单来说就是指定我们表单的那个字段 这里是Id
  - order 表示我们的这个sql语句与插入的sql语句执行的先后顺序 AFTER 后

```
<insert id="insertUser" parameterType="com.zhz.demo.User">
        <!-- order 表示我们的这个sql语句与插入的sql语句执行的先后顺序  AFTER 后-->
        <selectKey keyColumn="id" keyProperty="id" resultType="int" order="AFTER">
        SELECT  LAST_INSERT_ID();
        <!-- SELECT UUID() 在插入数据之前执行 -->
        </selectKey>
        insert into user (username,password) values (#{username},#{password})
 </insert>
```

六 更新操作

在这里我们对程序做一些优化

添加成员属性 : private SqlSession sqlSession;

添加方法,用于sqlSession的初始化:

```
    @Before
    public  void init() throws IOException{

        SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();

        SqlSessionFactory build = builder.build(Resources.getResourceAsStream("SqlMapConfig.xml"));

        sqlSession = build.openSession();

    }
```

这样就可以避免我们重复的去写获取SqlSession对象的代码.

> - @Before: 方法最先执行

- 6.1 测试类

```
@Test
public void updateUser() throws IOException{

    User user = new User();
    //设置要更新数据的主键
    user.setId(65);
    user.setUserName("曼殊沙华");
    user.setPassword("zhang");

    //SqlSession对象为类的成员属性
    sqlSession.update("updateUser",user);

    sqlSession.commit();
    sqlSession.close();
```

- 6.2 mapper.xml

```
<update id="updateUser" parameterType="com.zhz.demo.User">
 update user set username = #{username},password = #{password} where id = #{id}
 </update>
```

在这里删除就不演示了.删除和修改大致相同,这里写一下删除的映射语句吧
DELETE FROM USER WHERE ID = #{id}

七 基于接口与实现类的方式实现数据库的操作(很少使用)

- 7.1 创建一个interface UserDao
- 7.2 创建一个实现类 UserDaoImpl

- 7.3 创建一个演示类 UserDemo
- 7.4 UserDao:

```
User getUserById(int i);

List<User> getUserLike(String string);

void deleteUser(int i);
```

- 7.5 UserDaoImpl:

```
        private  SqlSession sqlSession;
        //有参构造
        public UserDaoImpl(SqlSession sqlSession) {
            super();
            this.sqlSession = sqlSession;
        }

        public User getUserById(int i) {
            User user = sqlSession.selectOne("getUserById", i);
            sqlSession.close();
            return user;
        }

        public List<User> getUserLike(String string) {
            List<User> list = sqlSession.selectList("getUserLike", string);
            sqlSession.close();
            return list;
        }

        public void deleteUser(int i) {
            sqlSession.delete("deleteUser",i);
            sqlSession.commit();
            sqlSession.close();
        }
```

- 7.6 UserDemo

```
*private  SqlSession sqlSession;

@Before
public  void setSession() throws IOException{

    SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();

    SqlSessionFactory build = builder.build(Resources.getResourceAsStream("SqlMapConfig.xml"));

    sqlSession = build.openSession();

}
/**
 * 通过Id查询
 */
@Test
public void getUserById(){
    UserDao dao = new UserDaoImpl(sqlSession);
    User user = dao.getUserById(65);

    System.out.println(user.toString());
}
/**
 * 模糊查询
 */

@Test
public void getUserLike(){
    UserDao dao = new UserDaoImpl(sqlSession);
    List<User> user  = dao.getUserLike("曼");
    for (User user2 : user) {
        System.out.println(user2.toString());
    }
}
/**
 * 删除
 */
@Test
public void deleteUser(){
    UserDao dao = new UserDaoImpl(sqlSession);
```

```
    dao.deleteUser(65);
}
```

- 7.7 映射语句就不写了

八 基于接口代理的方式实现数据库的操作(推荐使用)

基于这种操作有几点要求:

> - 1、 namespace要是接口的全限定名
> - 2、 接口名称与我们mapper文件名称保持一致
> - 3、 接口位置要与我们xml位置保持一致,在同一级目录下
> - 4、 接口中的方法名就是我们mapper文件中的sqlid

- 7.1 创建接口类 interface UserMapper
- 7.2 创建Mapper XML 文件 UserMapper.xml
- 7.3 创建测试类 UserMapperTest()
- 7.5 更改Mapper映射器

```
在 SqlMapConfig.xml中将<mapper resource="mapper.xml"/>
更改为:  <mapper class="com.zhz.demo3.UserMapper"/>
```

- 7.6 UserMapper.xml

```
<mapper namespace="com.zhz.demo3.UserMapper">

    <select id="getUserById" parameterType="int" resultType="com.zhz.demo.User">

    select * from user where  id = #{id}

    </select>

    <select id="getUserLike" parameterType="String" resultType="com.zhz.demo.User">

        select * from user where username like '%${value}%'

    </select>

    <insert id="insertUser" parameterType="com.zhz.demo.User">
        insert into user (username,password) values (#{username},#{password})
    </insert>
```

- 7.7 UserMapperTest

```
    *private  SqlSession sqlSession;

    @Before
    public  void init() throws IOException{

        SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();

        SqlSessionFactory build = builder.build(Resources.getResourceAsStream("SqlMapConfig.xml"));

        sqlSession = build.openSession();

    }
    /**
     * 查询
     */
    @Test
    public void test(){
        //使用JDK的动态代理来获取UserMapper接口的代理对象
        UserMapper mapper = sqlSession.getMapper(UserMapper.class);
        //User user = mapper.getUserById(66);
        List<User> list = mapper.getUserLike("曼");

        for(User user: list){
            System.out.println(user.toString());
        }

        sqlSession.close();

    }
```

```
    /*
     * 插入
     */
    @Test
    public void insertUser(){
        UserMapper mapper = sqlSession.getMapper(UserMapper.class);
        User user = new User();
        user.setUserName("小灰灰");
        user.setPassword("1234");

        mapper.insertUser(user);

        sqlSession.commit();
        sqlSession.close();

    }
```

九 引用外部数据库定义信息来实现数据库的连接

通过.properties文件来引入外部数据库

- 9.1 在资源目录下创建文件 jdbc.properties

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/product_manage?characterEncoding=utf-8
jdbc.username=root
jdbc.password=array
```

- 9.2 在SqlMapConfig.xml中设置

```
<properties resource="jdbc.properties">
    <property name="jdbc.driver" value=""/>
    <property name="jdbc.url" value=""/>
    <property name="jdbc.username" value="root"/>
    <property name="jdbc.password" value="fdsafd"/>
</properties>
```

如果 jdbc.properties和property中都有相同属性的值,则引用按照就近原则,先去property中找,找不到再去jdbc.properties中找

十 mybatis的别名配置

- 10.1类型别名是为 Java 类型设置一个短的名字。它只和 XML 配置有关，存在的意义仅在于用来减少类完全限定名的冗余。例如:

```
<typeAliases>
  <typeAlias alias="Author" type="domain.blog.Author"/>
  <typeAlias alias="Blog" type="domain.blog.Blog"/>
  <typeAlias alias="Section" type="domain.blog.Section"/>
  <typeAlias alias="Tag" type="domain.blog.Tag"/>
</typeAliases>
```

> - 当这样配置时，Blog可以用在任何使用domain.blog.Blog的地方。

- 10.2 使用包扫描的方式
  - 也可以指定一个包名，MyBatis 会在包名下面搜索需要的 Java Bean，比如:

```
<typeAliases>
  <package name="domain.blog"/>
</typeAliases>
```

  - 每一个在包 domain.blog 中的 Java Bean，在没有注解的情况下，会使用 Bean 的首字母小写的非限定类名来作为它的别名。 比如 domain.blog.Author 的别名为 author；若有注解，则别名为其注解值。

十一 使用包扫描的方式来加载我们所有的接口

使用Package的的方式来指定我们扫描那些包下面的所有接口,然后为这些接口产生代理

```
<mappers>
 <!--   <mapper class="com.zhz.demo3.UserMapper"/> -->
 <package name="com.zhz.demo3"/>
 </mappers>
```