如何优雅的使用 mybatis

作者：yk
原文链接：https://ld246.com/article/1502953000595
来源网站：链滴

这两天启动了一个新项目因为项目组成员一直都使用的是mybatis，虽然个人比较喜欢jpa这种极简的式，但是为了项目保持统一性技术选型还是定了 mybatis。到网上找了一下关于spring boot和mybats组合的相关资料，各种各样的形式都有，看的人心累，结合了mybatis的官方demo和文档终于找到最简的两种模式，花了一天时间总结后分享出来。

orm框架的本质是简化编程中操作数据库的编码，发展到现在基本上就剩两家了，一个是宣称可以不写一句SQL的hibernate，一个是可以灵活调试动态sql的mybatis,两者各有特点，在企业级系统开发可以根据需求灵活使用。发现一个有趣的现象：传统企业大都喜欢使用hibernate,互联网行业通常使mybatis。

hibernate特点就是所有的sql都用Java代码来生成，不用跳出程序去写（看）sql，有着编程的完整性发展到最顶端就是spring data jpa这种模式了，基本上根据方法名就可以生成对应的sql了，有不太解的可以看我的上篇文章springboot(五)：spring data jpa的使用。

mybatis初期使用比较麻烦，需要各种配置文件、实体类、dao层映射关联、还有一大推其它配置。然mybatis也发现了这种弊端，初期开发了generator可以根据表结果自动生产实体类、配置文件和da层代码，可以减轻一部分开发量；后期也进行了大量的优化可以使用注解了，自动管理dao层和配置件等，发展到最顶端就是今天要讲的这种模式了，mybatis-spring-boot-starter就是springboot+mbatis可以完全注解不用配置文件，也可以简单配置轻松上手。

现在想想spring boot 就是牛逼呀，任何东西只要关联到spring boot都是化繁为简。

mybatis-spring-boot-starter

官方说明：MyBatis Spring-Boot-Starter will help you use MyBatis with Spring Boot

其实就是myBatis看spring boot这么火热也开发出一套解决方案来凑凑热闹,但这一凑确实解决了很问题，使用起来确实顺畅了许多。mybatis-spring-boot-starter主要有两种解决方案，一种是使用解解决一切问题，一种是简化后的老传统。

当然任何模式都需要首先引入mybatis-spring-boot-starter的pom文件,现在最新版本是1.1.1（刚好到双11了 :)）

```
org.mybatis.spring.boot
mybatis-spring-boot-starter
1.1.1
```

好了下来分别介绍两种开发模式

无配置文件注解版

就是一切使用注解搞定。

1 添加相关maven文件

```
org.springframework.boot
spring-boot-starter
```

```
org.springframework.boot
spring-boot-starter-test
```

```
    test


    org.springframework.boot
    spring-boot-starter-web


    org.mybatis.spring.boot
    mybatis-spring-boot-starter
    1.1.1


    mysql
    mysql-connector-java


    org.springframework.boot
    spring-boot-devtools
    true
```

完整的pom包这里就不贴了，大家直接看源码

2、application.properties 添加相关配置

```
mybatis.type-aliases-package=com.neo.entity

spring.datasource.driverClassName = com.mysql.jdbc.Driver
spring.datasource.url = jdbc:mysql://localhost:3306/test1?useUnicode=true&characterEncodig=utf-8
spring.datasource.username = root
spring.datasource.password = root
```

springboot会自动加载spring.datasource.*相关配置，数据源就会自动注入到sqlSessionFactory中sqlSessionFactory会自动注入到Mapper中，对了你一切都不用管了，直接拿起来使用就行了。

在启动类中添加对mapper包扫描@MapperScan

```
@SpringBootApplication
@MapperScan("com.neo.mapper")
public class Application {


public static void main(String[] args) {
    SpringApplication.run(Application.class, args);
}

}
```

或者直接在Mapper类上面添加注解@Mapper,建议使用上面那种，不然每个mapper加个注解也挺烦的

3、开发Mapper

第三步是最关键的一块，sql生产都在这里

public interface UserMapper {

```
@Select("SELECT * FROM users")
@Results({
    @Result(property = "userSex",  column = "user_sex", javaType = UserSexEnum.class),
    @Result(property = "nickName", column = "nick_name")
})
List<UserEntity> getAll();

@Select("SELECT * FROM users WHERE id = #{id}")
@Results({
    @Result(property = "userSex",  column = "user_sex", javaType = UserSexEnum.class),
    @Result(property = "nickName", column = "nick_name")
})
UserEntity getOne(Long id);

@Insert("INSERT INTO users(userName,passWord,user_sex) VALUES(#{userName}, #{passWor
}, #{userSex})")
void insert(UserEntity user);

@Update("UPDATE users SET userName=#{userName},nick_name=#{nickName} WHERE id =#
id}")
void update(UserEntity user);

@Delete("DELETE FROM users WHERE id =#{id}")
void delete(Long id);
```

}

为了更接近生产我特地将user_sex、nick_name两个属性在数据库加了下划线和实体类属性名不一致另外user_sex使用了枚举

- @Select 是查询类的注解，所有的查询均使用这个
- @Result 修饰返回的结果集，关联实体类属性和数据库字段一一对应，如果实体类属性和数据属性名保持一致，就不需要这个属性来修饰。
- @Insert 插入数据库使用，直接传入实体类会自动解析属性到对应的值
- @Update 负责修改，也可以直接传入对象
- @delete 负责删除

了解更多属性参考这里

注意，使用#符号和$符号的不同：

```
// This example creates a prepared statement, something like select * from teacher where na
e = ?;
@Select("Select * from teacher where name = #{name}")
Teacher selectTeachForGivenName(@Param("name") String name);

// This example creates n inlined statement, something like select * from teacher where name
```

= 'someName';

@Select("Select * from teacher where name = '${name}'")

Teacher selectTeachForGivenName(@Param("name") String name);

4、使用

上面三步就基本完成了相关dao层开发，使用的时候当作普通的类注入进入就可以了

@RunWith(SpringRunner.class)

@SpringBootTest

public class UserMapperTest {

```
@Autowired
private UserMapper UserMapper;

@Test
public void testInsert() throws Exception {
    UserMapper.insert(new UserEntity("aa", "a123456", UserSexEnum.MAN));
    UserMapper.insert(new UserEntity("bb", "b123456", UserSexEnum.WOMAN));
    UserMapper.insert(new UserEntity("cc", "b123456", UserSexEnum.WOMAN));

    Assert.assertEquals(3, UserMapper.getAll().size());
}

@Test
public void testQuery() throws Exception {
    List<UserEntity> users = UserMapper.getAll();
    System.out.println(users.toString());
}

@Test
public void testUpdate() throws Exception {
    UserEntity user = UserMapper.getOne(3l);
    System.out.println(user.toString());
    user.setNickName("neo");
    UserMapper.update(user);
    Assert.assertTrue(("neo".equals(UserMapper.getOne(3l).getNickName())));
}

}
```

源码中controler层有完整的增删改查，这里就不贴了

源码在这里spring-boot-mybatis-annotation

极简xml版本

极简xml版本保持映射文件的老传统，优化主要体现在不需要实现dao的是实现层，系统会自动根据法名在映射文件中找对应的sql.

1、配置

pom文件和上个版本一样，只是application.properties新增以下配置

mybatis.config-locations=classpath:mybatis/mybatis-config.xml
mybatis.mapper-locations=classpath:mybatis/mapper/*.xml

指定了mybatis基础配置文件和实体类映射文件的地址

mybatis-config.xml 配置

```
    alias="Integer" type="java.lang.Integer" />
    alias="Long" type="java.lang.Long" />
    alias="HashMap" type="java.util.HashMap" />
    alias="LinkedHashMap" type="java.util.LinkedHashMap" />
    alias="ArrayList" type="java.util.ArrayList" />
    alias="LinkedList" type="java.util.LinkedList" />
```

这里也可以添加一些mybatis基础的配置

2、添加User的映射文件

namespace="com.neo.mapper.UserMapper" >
id="BaseResultMap" type="com.neo.entity.UserEntity" >
column="id" property="id" jdbcType="BIGINT" />
column="userName" property="userName" jdbcType="VARCHAR" />
column="passWord" property="passWord" jdbcType="VARCHAR" />
column="user_sex" property="userSex" javaType="com.neo.enums.UserSexEnum"/>
column="nick_name" property="nickName" jdbcType="VARCHAR" />

```
id="Base_Column_List" >
   id, userName, passWord, user_sex, nick_name


id="getAll" resultMap="BaseResultMap"  >
  SELECT
   refid="Base_Column_List" />
  FROM users


id="getOne" parameterType="java.lang.Long" resultMap="BaseResultMap" >
   SELECT
   refid="Base_Column_List" />
  FROM users
  WHERE id = #{id}


id="insert" parameterType="com.neo.entity.UserEntity" >
  INSERT INTO
      users
      (userName,passWord,user_sex)
```

```
    VALUES
        (#{userName}, #{passWord}, #{userSex})


 id="update" parameterType="com.neo.entity.UserEntity" >
   UPDATE
        users
   SET
     test="userName != null">userName = #{userName},
     test="passWord != null">passWord = #{passWord},
    nick_name = #{nickName}
   WHERE
        id = #{id}


 id="delete" parameterType="java.lang.Long" >
   DELETE FROM
         users
   WHERE
         id =#{id}
```

其实就是把上个版本中mapper的sql搬到了这里的xml中了

3、编写Dao层的代码

```
public interface UserMapper {


List<UserEntity> getAll();

UserEntity getOne(Long id);

void insert(UserEntity user);

void update(UserEntity user);

void delete(Long id);

}
```

对比上一步这里全部只剩了接口方法

4、使用

使用和上个版本没有任何区别，大家就看代码吧

xml配置版本

如何选择

两种模式各有特点，注解版适合简单快速的模式，其实像现在流行的这种微服务模式，一个微服务就对应一个自己的数据库，多表连接查询的需求会大大的降低，会越来越适合这种模式。

老传统模式比适合大型项目，可以灵活的动态生成SQL，方便调整SQL，也有痛痛快快，洋洋洒洒的SQL的感觉。

完整代码地址

作者：纯洁的微笑

出处：http://www.ityouknow.com/