第五章 企业项目开发--mybatis注解与xml并用

本章的代码建立在第四章《Java框架整合--切分配置文件》的项目代码之上

在实际开发中，我们在使用mybatis的时候，会注解与xml形式一起使用。

1、二者的使用场景

xml使用场景（3个）：

- 条件不定的查询（eg.下边代码中的getAdminByConditions方法）
- 增加对象返回自增主键（eg.下边代码的insertAdminWithBackId方法）
- 在一个Mapper接口中，出现多个select查询（>=3个），且每个查询都需要写相同的返回@Results内容（这一部分内容通常很多），这样的话，为了使Mapper接口比较整洁，重复代码比较少，我们会将这些select方法的具体实现写在xml文件中，因为在xml文件的顶部我们就会配置与注解@Results异曲同工的东西。（当然，这一点如果嫌配置xml麻烦，这一点可忽略）

注意：前两条是硬性的，是注解所解决不了的，而第三条只是建议。

除了以上这三条之外，其他的都使用去注解就好。

2、代码实现

基本代码不变，这只列出修改过得代码：

2.1、ssmm0-userManagement:

AdminController

```
package com.xxx.web.admin;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.servlet.ModelAndView;

import com.xxx.model.userManagement.Admin;
import com.xxx.service.userManagement.AdminService;

/**
 * adminController
 */
@Controller
@RequestMapping("/admin")
public class AdminController {

    @Autowired
    private AdminService adminService;

    /**
     * 管理员注册
     */
    @ResponseBody
    @RequestMapping("/register")
    public boolean register(@RequestParam("username") String username,
                            @RequestParam("password") String password){
        Admin admin = new Admin();
        admin.setUsername(username);
        admin.setPassword(password);

        boolean isRegisterSuccess = adminService.register(admin);

        return isRegisterSuccess;
    }

```

```java
    /**
     * 管理员登录
     */
    @RequestMapping("/login")
    public ModelAndView login(@RequestParam("username") String username,
                              @RequestParam("password") String password){
        Admin admin = adminService.login(username, password);

        ModelAndView modelAndView = new ModelAndView();
        if(admin == null){
            modelAndView.addObject("message", "用户不存在或者密码错误！请重新输入");
            modelAndView.setViewName("error");
        }else{
            modelAndView.addObject("admin", admin);
            modelAndView.setViewName("userinfo");
        }

        return modelAndView;
    }

    /****************************mybatis xml方式解决的问题******************************/
    /**
     * 根据username或password查找List<Admin>
     */
    @ResponseBody
    @RequestMapping("/findAdmin")
    public List<Admin> findAdmin(@RequestParam(value="username",required=false) String username,
                                 @RequestParam(value="password",required=false) String password,
                                 @RequestParam("start") int start,
                                 @RequestParam("limit") int limit){
        List<Admin> adminList = adminService.findAdmin(username, password, start, limit);
        return adminList;
    }

    /**
     * 插入一个用户并返回主键
     * 注意：get请求也会自动装配（即将前台传入的username和password传入admin）
     */
    @ResponseBody
    @RequestMapping("/insert")
    public Admin insertAdminWithBackId(Admin admin){
        return adminService.insertAdminWithBackId(admin);
    }
}
```

说明：在这里增加了两个方法，具体看代码与注释

注：

- **springMVC通过get方式传递的属性值username、password也能自动装配到对象admin中**

2.2、ssmm0-data：

AdminService

```java
package com.xxx.service.userManagement;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.xxx.dao.userManagement.AdminDao;
```

```java
import com.xxx.model.userManagement.Admin;

/**
 * 管理员service
 */
@Service
public class AdminService {
    @Autowired
    private AdminDao adminDao;

    public boolean register(Admin admin){
        return adminDao.register(admin);
    }

    public Admin login(String username, String password) {
        return adminDao.login(username, password);
    }

    /***********以下方法是为了测试mybatis中使用xml**********/
    public List<Admin> findAdmin(String username, String password, int start, int limit){
        return adminDao.findAdmin(username, password, start, limit);
    }

    public Admin insertAdminWithBackId(Admin admin){
        int record = adminDao.insertAdminWithBackId(admin);
        if(record==1){
            return admin;//这时的admin已经被赋予主键了
        }
        return null;
    }
}
```

AdminDao

```java
package com.xxx.dao.userManagement;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

import com.xxx.mapper.userManagement.AdminMapper;
import com.xxx.model.userManagement.Admin;

/**
 * 管理员DAO
 */
@Repository
public class AdminDao {
    @Autowired
    private AdminMapper adminMapper;

    public boolean register(Admin admin){
        return adminMapper.insertAdmin(admin)==1?true:false;
    }

    public Admin login(String username ,String password){
        return adminMapper.selectAdmin(username, password);
    }

    public List<Admin> findAdmin(String username, String password, int start, int limit){
```

```
        return adminMapper.getAdminByConditions(username, password, start, limit);
    }

    public int insertAdminWithBackId(Admin admin){
        return adminMapper.insertAdminWithBackId(admin);
    }
}
```

AdminMapper

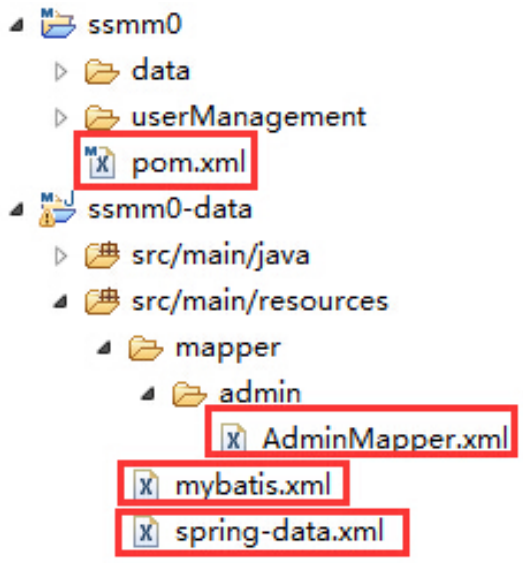
```
package com.xxx.mapper.userManagement;

import java.util.List;

import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Param;
import org.apache.ibatis.annotations.Result;
import org.apache.ibatis.annotations.Results;
import org.apache.ibatis.annotations.Select;

import com.xxx.model.userManagement.Admin;

/**
 * 管理员Mapper
 */
public interface AdminMapper {

    /**************注解**************/
    @Insert("INSERT INTO userinfo(username, password) VALUES(#{username},#{password})")
    public int insertAdmin(Admin admin);

    @Select("SELECT * FROM userinfo WHERE username = #{username} AND password = #{password}")
    @Results(value = {
            @Result(id = true, column = "id", property = "id"),
            @Result(column = "username", property = "username"),
            @Result(column = "password", property = "password") })
    public Admin selectAdmin(@Param("username") String username,
                             @Param("password") String password);

    /***************xml**************/
    /**
     * 条件不定式查询
     * 我们这里使用@Param指定参数，这样的话，在AdminMapper.xml中就不用再使用parameterType属性了；否则得写parameterType属性
     */
    public List<Admin> getAdminByConditions(@Param("username")String username,
                                            @Param("password")String password,
                                            @Param("start")int start,
                                            @Param("limit")int limit);

    /**
     * 返回主键
     */
    public int insertAdminWithBackId(Admin admin);
}
```

注意：在用xml传参的时候，

- 如果你直接传参，eg.insertAdminWithBackId(Admin admin)，则在xml中的insertAdminWithBackId方法处要添加parameterType；
- 如果你用了注解传参的话，eg.getAdminByConditions(@Param("username")String username)，则在xml中的getAdminByConditions方法处

不用添加parameterType，当然，注解传参的时候，不能传引用类型，一般只传基本类型，eg.insertAdminWithBackId(@Param("admin")Admin admin)就是不行的

接口定义好之后，需要添加两个配置文件+修改两个配置文件。目录结构如下：



AdminMapper.xml（该xml的名字最好与对应接口的接口名完全相同）

```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">

<!-- 指定工作空间，要与接口名相同，源代码没有去看，猜测应该是通过"这里的namespace.下边方法的id"来定位方法的 -->
<mapper namespace="com.xxx.mapper.userManagement.AdminMapper">
    <!-- 指定字段映射 -->
    <resultMap type="Admin" id="adminResultMap">
        <id property="id"  column="id" jdbcType="INTEGER" />
        <result property="username" column="username" jdbcType="VARCHAR" />
        <result property="password" column="password" jdbcType="VARCHAR" />
    </resultMap>

    <select id="getAdminByConditions" resultMap="adminResultMap"><!-- 返回结果为上边指定的adminResultMap -->
        <![CDATA[ SELECT * FROM userinfo WHERE 1=1 ]]>
        <if test="username != null"><![CDATA[ AND username = #{username} ]]></if>
        <if test="password != null"><![CDATA[ AND password = #{password} ]]></if>
        <![CDATA[ ORDER BY id ASC LIMIT #{start}, #{limit} ]]>
    </select>

    <!-- 若不需要自动返回主键，将useGeneratedKeys="true" keyProperty="id"去掉即可 -->
    <insert id="insertAdminWithBackId" parameterType="Admin" useGeneratedKeys="true" keyProperty="id">
       <![CDATA[
       INSERT INTO userinfo
       (
           username,
           password
       )
       VALUES
       (
           #{username, jdbcType=VARCHAR},
           #{password, jdbcType=VARCHAR}
       )
       ]]>
   </insert>

</mapper>
```

注意：

- **该xml的名字最好与对应接口的接口名完全相同**（eg.AdminMapper.xml对于应接口AdminMapper）
- parameterType有无参照上边对AdminMapper处所讲的注意点
- 返回自增主键有两种方法，我这里列出了最常用的也是最简单的一种

mybatis.xml

```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>
    <properties>
        <property name="dialect" value="mysql" />
    </properties>

    <typeAliases>
        <!-- 这样会将com.xxx.model包及其子包下的所有类起别名为相应的简单类名 -->
        <package name="com.xxx.model"/>
        <!-- 如果这样去起别名的话，每一个模型类都要写一个，就比较麻烦 -->
        <!-- <typeAlias alias="Admin" type="com.xxx.model.userManagement.Admin"/> -->
    </typeAliases>
</configuration>
```

注意：这个文件一般用于指定属性和别名。

- 通常，属性只指定数据库方言即可；
- 有两种别名方式指定，请参照上述代码给出的注释进行选择，一般而言，都会选择package方式的

spring-data.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
                        http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
                        http://www.springframework.org/schema/context
                        http://www.springframework.org/schema/context/spring-context-3.2.xsd">

    <!-- 注解扫描 -->
    <context:component-scan base-package="com.xxx" />

    <!-- 引入数据源，这里变量的读取都是从ssmm0的pom.xml中读取的 -->
    <bean id="xxxDataSource" class="org.apache.tomcat.jdbc.pool.DataSource" destroy-method="close">
        <property name="driverClassName" value="${jdbc.driverClassName}" />
        <property name="url" value="${jdbc.url}" />
        <property name="username" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
    </bean>

    <!-- 引入mybatis -->
    <bean id="xxxSqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="xxxDataSource" />
        <!-- 以下两个属性是专门为xml方式配置的，若只使用注解方式，这两个属性行可以不配置 -->
        <property name="configLocation" value="classpath:mybatis.xml"/>
        <property name="mapperLocations">
            <list>
                <value>classpath*:mapper/admin/*Mapper.xml</value>
            </list>
        </property>
    </bean>
    <bean id="xxxMapperScannerConfigurer" class="org.mybatis.spring.mapper.MapperScannerConfigurer">
```

```
        <!--
            这里就是包名为什么就做com.xxx.mapper.user而非com.xxx.user.mapper,
            这样的话，比如说有两个项目com.xxx.mapper.user和com.xxx.mapper.hotel，value只需写作com.xxx.mapper即可
            否则，value就要写作com.xxx.user.mapper,com.xxx.hotel.mapper
         -->
        <property name="basePackage" value="com.xxx.mapper" />
        <property name="sqlSessionFactoryBeanName" value="xxxSqlSessionFactory" />
    </bean>

</beans>
```

说明：只增加了两个属性配置，看代码与注释。

注：关于classpath与classpath*的具体区别自己去查，简单来说就是两句话：**classpath只加载第一个找到文件；classpth*加载找到的多个文件**。

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>com.xxx</groupId>
    <artifactId>ssmm0</artifactId>
    <version>1.0-SNAPSHOT</version>

    <name>ssmm0</name>
    <packaging>pom</packaging><!-- 父模块 -->

    <!-- 管理子模块 -->
    <modules>
        <module>userManagement</module><!-- 具体业务1-人员管理系统 -->
        <module>data</module><!-- 封装数据操作 -->
    </modules>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    </properties>

    <!-- dependencyManagement不会引入实际的依赖，只是作为一个依赖池，供其和其子类使用 -->
    <dependencyManagement>
        <dependencies>
            <!-- json -->
            <dependency>
                <groupId>com.alibaba</groupId>
                <artifactId>fastjson</artifactId>
                <version>1.1.39</version>
            </dependency>
            <!-- servlet -->
            <dependency>
                <groupId>javax.servlet</groupId>
                <artifactId>javax.servlet-api</artifactId>
                <version>3.0.1</version>
                <scope>provided</scope>
            </dependency>
            <!-- spring -->
            <dependency>
                <groupId>org.springframework</groupId>
```

```
44                <artifactId>spring-core</artifactId>
45                <version>3.2.6.RELEASE</version>
46            </dependency>
47            <dependency>
48                <groupId>org.springframework</groupId>
49                <artifactId>spring-beans</artifactId>
50                <version>3.2.6.RELEASE</version>
51            </dependency>
52            <dependency>
53                <groupId>org.springframework</groupId>
54                <artifactId>spring-context</artifactId>
55                <version>3.2.6.RELEASE</version>
56            </dependency>
57            <dependency>
58                <groupId>org.springframework</groupId>
59                <artifactId>spring-web</artifactId>
60                <version>3.2.6.RELEASE</version>
61            </dependency>
62            <dependency>
63                <groupId>org.springframework</groupId>
64                <artifactId>spring-webmvc</artifactId>
65                <version>3.2.6.RELEASE</version>
66            </dependency>
67            <!-- 这个是使用velocity的必备包 -->
68            <dependency>
69                <groupId>org.springframework</groupId>
70                <artifactId>spring-context-support</artifactId>
71                <version>3.2.6.RELEASE</version>
72            </dependency>
73            <!-- mysql -->
74            <dependency>
75                <groupId>mysql</groupId>
76                <artifactId>mysql-connector-java</artifactId>
77                <version>5.1.27</version>
78                <scope>runtime</scope>
79            </dependency>
80            <!-- 数据源 -->
81            <dependency>
82                <groupId>org.apache.tomcat</groupId>
83                <artifactId>tomcat-jdbc</artifactId>
84                <version>7.0.47</version>
85            </dependency>
86            <!-- mybatis -->
87            <dependency>
88                <groupId>org.mybatis</groupId>
89                <artifactId>mybatis</artifactId>
90                <version>3.1.1</version>
91            </dependency>
92            <dependency>
93                <groupId>org.mybatis</groupId>
94                <artifactId>mybatis-spring</artifactId>
95                <version>1.1.1</version>
96            </dependency>
97            <!-- velocity -->
98            <dependency>
99                <groupId>org.apache.velocity</groupId>
100               <artifactId>velocity</artifactId>
101               <version>1.5</version>
102           </dependency>
103           <dependency>
104               <groupId>velocity-tools</groupId>
105               <artifactId>velocity-tools-generic</artifactId>
106               <version>1.2</version>
107           </dependency>
108           <!-- 用于加解密 -->
```

```
            <dependency>
                <groupId>commons-codec</groupId>
                <artifactId>commons-codec</artifactId>
                <version>1.7</version>
            </dependency>
            <dependency>
                <groupId>org.bouncycastle</groupId>
                <artifactId>bcprov-jdk15on</artifactId>
                <version>1.47</version>
            </dependency>
            <!-- 集合工具类 -->
            <dependency>
                <groupId>org.apache.commons</groupId>
                <artifactId>commons-collections4</artifactId>
                <version>4.0</version>
            </dependency>
            <!-- http -->
            <dependency>
                <groupId>org.apache.httpcomponents</groupId>
                <artifactId>httpclient</artifactId>
                <version>4.2.6</version>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <!-- 引入实际依赖 -->
    <dependencies>
        <!-- json -->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
        </dependency>
        <!-- spring -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
        </dependency>
        <!-- 集合工具类 -->
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-collections4</artifactId>
        </dependency>
    </dependencies>

    <build>
        <resources>
            <!-- 这里配置了这一块儿true，才可以让指定文件（这里是src/main/resources/spring-data.xml）读到pom.xml中的配置信息
                 ， 值得注意的是，如果src/main/resources下还有其他文件，而你不想让其读pom.xml， 你还必须得把src/main/resources下的其余文件再配置一遍，配置为false（不可读pom.xml），
                 如下边的注释那样，否则，会报这些文件（在这里，就是*.properties）找不到的错误
            -->
            <resource>
                <directory>src/main/resources</directory>
                <filtering>true</filtering>
                <includes>
                    <include>*.xml</include>
```

```
                </includes>
            </resource>
            <!--
            <resource>
                <directory>src/main/resources</directory>
                <filtering>false</filtering>
                <includes>
                    <include>*.properties</include>
                </includes>
            </resource>
            -->
            <resource>
                <directory>src/main/resources</directory>
                <filtering>false</filtering>
                <includes>
                <!-- 这里如果不加这一条，那么在spring-data.xml中配置的xml将找不到classpath:mapper/admin/AdminMapper.xml -->
                    <include>mapper/**/*.xml</include>
                </includes>
            </resource>
        </resources>
    </build>

    <!--
        profiles可以定义多个profile，然后每个profile对应不同的激活条件和配置信息，从而达到不同环境使用不同配置信息的效果
        注意两点：
        1）<activeByDefault>true</activeByDefault>这种情况表示服务器启动的时候就采用这一套env（在这里，就是prod）
        2）当我们启动服务器后，想采用开发模式，需切换maven的env为dev，如果env的配置本身就是dev，需要将env换成rc或prod，点击apply，然后再将env切换成dev，点击apply才行
    -->
    <profiles>
        <!-- 开发env -->
        <profile>
            <id>dev</id>
            <activation>
                <activeByDefault>false</activeByDefault>
                <property>
                    <name>env</name>
                    <value>dev</value>
                </property>
            </activation>
            <properties>
                <env>dev</env>

                <jdbc.driverClassName>com.mysql.jdbc.Driver</jdbc.driverClassName>
                <!--
                     对于jdbc.url中内容的配置，如果需要配置 &amp;时，有两种方法：
                    1）如下边这样，使用<![CDATA[XXX]]>包起来
                    2）使用jdbc.properties文件来读取此pom.xml，然后spring.xml再读取jdbc.properties文件 显然，前者更方便，而且还省了一个jdbc.properties的文件，但是，有的时候，还是会用后者的；
                    在使用后者的时候，注意三点：
                    1）需要修改上边的build中的内容
                    2）需要在spring.xml中配置<context:property-placeholder location="classpath:jdbc.properties"/>
                    3）将jdbc.properties放在ssmm0-data项目中，之后需要将ssmm0-data项目的env配置为dev
                -->
                <jdbc.url><![CDATA[jdbc:mysql://127.0.0.1:3306/blog?zeroDateTimeBehavior=convertToNull&amp;useUnicode=true&amp;characterEncoding=utf-8]]></jdbc.url>
                <jdbc.username>root</jdbc.username>
                <jdbc.password>123456</jdbc.password>
            </properties>
        </profile>
        <!-- 预上线env -->
```

```xml
        <profile>
            <id>rc</id>
            <activation>
                <activeByDefault>false</activeByDefault>
                <property>
                    <name>env</name>
                    <value>rc</value>
                </property>
            </activation>
            <properties>
                <env>rc</env>

                <jdbc.driverClassName>com.mysql.jdbc.Driver</jdbc.driverClassName>
                <!-- 假设的一个地址 -->
                <jdbc.url><![CDATA[jdbc:mysql://10.10.10.100:3306/blog?zeroDateTimeBehavior=convertToNull&amp;useUnicode=true&amp;characterEncoding=utf-8]]></jdbc.url>
                <jdbc.username>root2</jdbc.username>
                <jdbc.password>1234562</jdbc.password>
            </properties>
        </profile>
        <!-- 线上env -->
        <profile>
            <id>prod</id>
            <activation>
                <activeByDefault>true</activeByDefault>
                <property>
                    <name>env</name>
                    <value>prod</value>
                </property>
            </activation>
            <properties>
                <env>prod</env>

                <jdbc.driverClassName>com.mysql.jdbc.Driver</jdbc.driverClassName>
                <!-- 假设的一个地址 -->
                <jdbc.url><![CDATA[jdbc:mysql://99.99.99.999:3307/blog?zeroDateTimeBehavior=convertToNull&amp;useUnicode=true&amp;characterEncoding=utf-8]]></jdbc.url>
                <jdbc.username>sadhijhqwui</jdbc.username>
                <jdbc.password>zxczkchwihcznk=</jdbc.password>
            </properties>
        </profile>
    </profiles>
</project>
```

说明：只在resource部分增加了一行关于"接口.xml"的过滤配置（作用看注释）

测试：测试的具体操作见前一章。