# 1. SSM框架整合

所谓SSM框架整合，就是利用Spring将SpringMVC和Mybatis整合到一起，这是现在比较流行的方式。

## 1.1 创建Maven工程

首先新建Maven工程，使用如下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/xsd/maven-4.0.0.xsd"**>  <**modelVersion**>4.0.0</**modelVersion**>  <**groupId**>com.zhang</**groupId**>  <**artifactId**>ssm\_demo</**artifactId**>  <**version**>1.0-SNAPSHOT</**version**>  <**packaging**>war</**packaging**>  <**properties**>  <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  *<!-- 定义使用的Spring版本号，因为下面会多次使用到 -->* <**spring-version**>5.0.2.RELEASE</**spring-version**>  </**properties**>  <**dependencies**>  *<!-- SpringMVC的依赖。会自动依赖Spring核心包 -->* <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-webmvc</**artifactId**>  <**version**>${spring-version}</**version**>  <**scope**>compile</**scope**>  </**dependency**>  *<!-- 需要加上Spring的JDBC和事务支持包 -->* <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-jdbc</**artifactId**>  <**version**>${spring-version}</**version**>  <**scope**>compile</**scope**>  </**dependency**>  <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-tx</**artifactId**>  <**version**>${spring-version}</**version**>  <**scope**>compile</**scope**>  </**dependency**>  *<!-- MySQL的JDBC驱动 -->* <**dependency**>  <**groupId**>mysql</**groupId**>  <**artifactId**>mysql-connector-java</**artifactId**>  <**version**>5.1.42</**version**>  <**scope**>runtime</**scope**>  </**dependency**>  *<!-- druid数据库连接池 -->* <**dependency**>  <**groupId**>com.alibaba</**groupId**>  <**artifactId**>druid</**artifactId**>  <**version**>1.1.6</**version**>  </**dependency**>  *<!-- Mybatis -->* <**dependency**>  <**groupId**>org.mybatis</**groupId**>  <**artifactId**>mybatis</**artifactId**>  <**version**>3.4.4</**version**>  <**scope**>compile</**scope**>  </**dependency**>  *<!-- Mybatis与Spring的整合包 -->* <**dependency**>  <**groupId**>org.mybatis</**groupId**>  <**artifactId**>mybatis-spring</**artifactId**>  <**version**>1.3.1</**version**>  <**scope**>compile</**scope**>  </**dependency**>  *<!-- fastjson -->* <**dependency**>  <**groupId**>com.alibaba</**groupId**>  <**artifactId**>fastjson</**artifactId**>  <**version**>1.2.33</**version**>  <**scope**>compile</**scope**>  </**dependency**>  *<!-- servlet-api -->* <**dependency**>  <**groupId**>org.apache.tomcat</**groupId**>  <**artifactId**>tomcat-servlet-api</**artifactId**>  <**version**>8.5.20</**version**>  </**dependency**>  *<!-- jstl -->* <**dependency**>  <**groupId**>javax.servlet</**groupId**>  <**artifactId**>jstl</**artifactId**>  <**version**>1.2</**version**>  </**dependency**>  *<!-- log4j2日志组件 -->* <**dependency**>  <**groupId**>org.apache.logging.log4j</**groupId**>  <**artifactId**>log4j-core</**artifactId**>  <**version**>2.10.0</**version**>  <**scope**>runtime</**scope**>  </**dependency**>  *<!-- junit -->* <**dependency**>  <**groupId**>junit</**groupId**>  <**artifactId**>junit</**artifactId**>  <**version**>4.12</**version**>  <**scope**>test</**scope**>  </**dependency**>  *<!-- Spring的test包，为了测试用 -->* <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-test</**artifactId**>  <**version**>${spring-version}</**version**>  <**scope**>test</**scope**>  </**dependency**>  </**dependencies**>  <**build**>  <**resources**>  <**resource**>  <**directory**>src/main/java</**directory**>  <**includes**>  *<!-- 包括src目录下的xml配置文件 -->* <**include**>\*\*/\*.xml</**include**>  </**includes**>  </**resource**>  *<!-- 既然配置了resources，则还要说明包含的文件夹 -->* <**resource**>  <**directory**>src/main/resources</**directory**>  </**resource**>  <**resource**>  <**directory**>src/main/webapp</**directory**>  </**resource**>  </**resources**>  <**pluginManagement**>  <**plugins**>  <**plugin**>  <**groupId**>org.apache.maven.plugins</**groupId**>  <**artifactId**>maven-compiler-plugin</**artifactId**>  <**version**>3.6.1</**version**>  <**configuration**>  <**source**>1.8</**source**>  <**target**>1.8</**target**>  <**encoding**>UTF-8</**encoding**>  </**configuration**>  </**plugin**>  </**plugins**>  </**pluginManagement**>  </**build**> </**project**> |

注意以上引入的相关包和一些设置。（上述log4j的日志组件是runtime，不能是provided）

接下来就是利用Spring来集成Mybatis和SpringMVC了，重点在于集成Mybatis，SpringMVC的集成和之前类似。目的就是用Spring管理对象的创建，以供不同层使用。

（一）在src/main/java下新建如下包：

1. com.emp.domain：实体类包；

2. com.emp.mapper：mapper接口和映射文件所在包；

3. com.emp.service：业务层包；

4. com.emp.service.impl：业务层实现包；

5. com.emp.controller：Controller所在包。

（二）在src/main/resources下新建：

1. spring/applicationContext.xml文件：用于进行Spring的配置；

2. db.properties文件：基本的数据库配置文件；

3. mybatis/config.xml文件：该文件就是之前的Mybatis主配置文件SqlMapConfig.xml，采用Spring配置后，该文件有所精简。

下面进行具体的配置。

## 1.2 Spring和Mybatis的配置（Service层及其下层）

（1）首先，db.properties文件内容如下：

|  |
| --- |
| **driverClassName**=**com.mysql.jdbc.Driver username**=**root password**=**123qwe!@# url**=**jdbc:mysql://114.55.86.230:3306/mytest connectionProperties**=**useUnicode=false;characterEncoding=UTF8;useSSL=false** |

（2）下面配置applicationContext.xml，其中主要配置：

1. 数据源dataSource

2. Mybatis的sqlSessionFactory，即Mybatis的sqlSessionFactory是由Spring管理的。该配置需要利用Mybatis-Spring的包，因此我们之前在pom.xml声明了该依赖包，该包是Mybatis官方提供的。

3. Service和Controller的bean配置。使用注解方式，使得Service能成功使用Mapper对象；Controller能成功使用Service对象。

4. 事务配置。采用注解方式，Mybatis中事务就是使用JDBC提供的事务，在Service层方法中配置事务。

5. SpringMVC配置。

新建spring文件夹，其中的applicationContext.xml具体配置如下：

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?> <!-- 注意beans节点中使用的名称空间要正确 -->* <**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" xmlns:tx="http://www.springframework.org/schema/tx"  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd"**>  *<!-- 一、配置数据源 -->  <!-- 1. 加载db.properties文件 -->* <**context:property-placeholder location="classpath:db.properties"** />  *<!-- 2. 使用druid配置数据源 -->* <**bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="init" destroy-method="close"**>  <**property name="url" value="${url}"** />  <**property name="connectionProperties" value="${connectionProperties}"** />  <**property name="username" value="${username}"** />  <**property name="password" value="${password}"** />  <**property name="initialSize" value="1"** />  <**property name="maxActive" value="10"** />  <**property name="minIdle" value="1"** />  <**property name="maxWait" value="60000"** />  <**property name="timeBetweenEvictionRunsMillis" value="60000"** />  <**property name="minEvictableIdleTimeMillis" value="300000"** />  <**property name="testWhileIdle" value="true"** />  <**property name="testOnBorrow" value="false"** />  <**property name="testOnReturn" value="false"** />  <**property name="poolPreparedStatements" value="true"** />  <**property name="maxOpenPreparedStatements" value="20"** />  <**property name="asyncInit" value="true"** />  </**bean**>  *<!-- 二、Mybatis的整合配置 -->  <!-- 1. 配置会话工厂。 -->* <**bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean"**>  *<!-- 引用数据源 -->* <**property name="dataSource" ref="dataSource"** />  *<!-- Mybatis的配置文件位置 -->* <**property name="configLocation" value="classpath:mybatis/config.xml"** />  </**bean**>  *<!-- 2. 配置mapper。这里使用MapperScannerConfigurer扫描mapper包 -->* <**bean class="org.mybatis.spring.mapper.MapperScannerConfigurer"**>  *<!-- 指定mapper所在包 -->* <**property name="basePackage" value="com.emp.mapper"** />  *<!-- 指明sqlSessionFactory配置的bean的名称，注意这里不是引用sqlSessionFactory -->* <**property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"** />  </**bean**>  *<!-- 三、配置Service -->  <!-- 只要开启注解扫描即可。注解扫描这里实际上就是为Service服务的，因为需要把他们加入到容器管理中 -->* <**context:component-scan base-package="com.emp.service"** />  *<!-- 四、事务配置 -->* <**bean id="tx" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"**>  *<!-- 引用dataSource -->* <**property name="dataSource" ref="dataSource"** />  </**bean**>  *<!-- 使用事务注解 -->* <**tx:annotation-driven transaction-manager="tx" proxy-target-class="true"** /> </**beans**> |

配置的说明：

（1）关于数据源，这里提醒一下：在开发时应连接数量应设置地较小，够用即可。避免多台开发电脑连接同一个局域网的数据库时连接数太多。

（2）上述注解扫描只扫描了sevice层，这是因mapper接口根本无需扫描，也不用加上@Repository注解。因为mapper接口已经通过mapper扫描器MapperScannerConfigurer注入Spring中了，注入的mapper代理类的bean名称就是类名（首字母小写，例如employeeMapper）。

这样，mybatis目录下的config.xml文件。由于数据库环境、Mapper类等已经在applicationContext.xml中配置好了，所以该Mybatis主配置文件只需进行“特有”的配置，例如，下面配置了“别名”：

|  |
| --- |
| *<?***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**>  <**typeAliases**>  <**package name="com.emp.entity"** />  </**typeAliases**> </**configuration**> |

这样，Spring和Mybatis的配置就完成了，接下来进行Service层及其下层的编写和测试工作。

（1）在com.emp.domain.Employee类中写：

|  |
| --- |
| **package** com.emp.domain;  **import** java.io.Serializable;  **public class** Employee **implements** Serializable {  **private** Integer **id**;  **private** String **name**;   *// setter和getter* } |

（2）在com.emp.mapper.EmployeeMapper类中写：

|  |
| --- |
| **package** com.emp.mapper;  **import** com.emp.entity.Employee;  **public interface** EmployeeMapper {  Employee selectEmp(Integer id); } |

（3）在com/emp/mapper/EmployeeMapper.xml文件中写：

|  |
| --- |
| *<?***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 namespace="com.emp.mapper.EmployeeMapper"**>  <**select id="selectEmp" parameterType="int" resultType="employee"**>  select *\** from employee where id=#{id}  </**select**> </**mapper**> |

（4）在com.emp.service.IEmployeeService接口中写：

|  |
| --- |
| **package** com.emp.service; **import** com.emp.entity.Employee; **public interface** IEmployeeService {  Employee getEmployee(Integer id); } |

（5）在com.emp.service.impl.EmployeeServiceImpl实现类中写：

|  |
| --- |
| **package** com.emp.service.impl;  **import** com.emp.entity.Employee; **import** com.emp.mapper.EmployeeMapper; **import** com.emp.service.IEmployeeService; **import** org.springframework.stereotype.Service; **import** org.springframework.transaction.annotation.Transactional; **import** javax.annotation.Resource; *// 使用Service注解* @Service **public class** EmployeeServiceImpl **implements** IEmployeeService {  *// 使用注解使用mapper的bean* @Resource  **private** EmployeeMapper **employeeMapper**;   @Override  @Transactional(readOnly = **true**) *// 控制事务*  **public** Employee getEmployee(Integer id) {  **return employeeMapper**.selectEmp(id);  } } |

上述写好之后，最后我们在src/test/java中进行单元测试Service。新建com.emp.test包，ServiceTest类的测试代码如下：

|  |
| --- |
| **package** com.emp.test;  **import** com.emp.domain.Employee; **import** com.emp.mapper.EmployeeMapper; **import** com.emp.service.IEmployeeService; **import** org.junit.Test; **import** org.junit.runner.RunWith; **import** org.springframework.test.context.ContextConfiguration; **import** org.springframework.test.context.junit4.SpringJUnit4ClassRunner; **import** org.springframework.transaction.annotation.Transactional; **import** javax.annotation.Resource;  @RunWith(SpringJUnit4ClassRunner.**class**) @ContextConfiguration(locations={**"classpath:spring/applicationContext.xml"**}) **public class** ServiceTest {  *// 注入需要的属性* @Resource(name = **"employeeService"**)  **private** IEmployeeService **employeeService**;  @Resource(name = **"employeeMapper"**)  **private** EmployeeMapper **employeeMapper**;   @Test  @Transactional  **public void** testService() {  Employee employee = **employeeService**.getEmployee(1);  System.***out***.println(employee.getName());  }   @Test  @Transactional  **public void** testDao() {  Employee employee = **employeeMapper**.selectEmp(1);  System.***out***.println(employee.getName());  }  } |

上述代码即可完成对DAO和Service框架的初步测试。但运行上述代码却死循环出现下述错误：

|  |
| --- |
| 14:17:14.903 ERROR com.alibaba.druid.support.logging.Log4j2Impl 53 error - create connection SQLException, url: jdbc:mysql://114.55.86.230:3306/mytest, errorCode 1045, state 28000 java.sql.SQLException: Access denied for user 'zmm'@'58.214.233.214' (using password: YES) |

这实际不是MySQL的原因，可能是Mybatis的问题。因为经过这样的处理就没问题了：只要把db.properties中的所有键，加上一个“前缀”，再在applicationContext.xml中引用，问题就解决了。例如加上“jdbc\_”或者“jdbc.”前缀等。因此我们把db.properties文件修改为：

|  |
| --- |
| **jdbc\_driverClassName**=**com.mysql.jdbc.Driver jdbc\_username**=**root jdbc\_password**=**123qwe!@# jdbc\_url**=**jdbc:mysql://114.55.86.230:3306/mytest jdbc\_connectionProperties**=**useUnicode=false;characterEncoding=UTF8;useSSL=false** |

相应的读者自行修改applicationContext.xml文件。因此说这个问题可能是Mybatis的，和MySQL数据库没有关系，这是需要注意的大坑！

## 1.3 集成SpringMVC

先在src/main/resources/spring目录下配置springmvc.xml文件：

|  |
| --- |
| *<?***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:mvc="http://www.springframework.org/schema/mvc"  xmlns:context="http://www.springframework.org/schema/context"  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"**>  *<!-- SpringMVC的配置 -->  <!-- 1. 扫描controller类的注解 -->* <**context:component-scan base-package="com.emp.controller"** />  *<!-- 2. SpringMVC驱动配置 -->* <**mvc:annotation-driven**>  <**mvc:message-converters register-defaults="true"**>  *<!--配置fastjson支持-->* <**bean class="com.alibaba.fastjson.support.spring.FastJsonHttpMessageConverter"**>  <**property name="supportedMediaTypes"**>  <**list**>  <**value**>text/html;charset=utf-8</**value**>  <**value**>application/json</**value**>  </**list**>  </**property**>  </**bean**>  </**mvc:message-converters**>  </**mvc:annotation-driven**> </**beans**> |

说明：

（1）上述springmvc中又增加了对controller的扫描，使得能扫描到Controller类。

（2）实际上完全能把springmvc.xml中的配置写到applicationContext.xml中，但这样分开写也比较清晰。也不用在applicationContext.xml文件中直接扫描“com.emp”下的所有类，这样分成两个包扫描也挺好的，减少了扫描耗时。

最后，只需要在src/main/webapp/WEB-INF/web.xml中配置加载Spring和SpringMVC即可：

|  |
| --- |
| **<!DOCTYPE web-app PUBLIC  "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"  "http://java.sun.com/dtd/web-app\_2\_3.dtd" *>*** <**web-app**>  *<!-- 设置Spring配置文件 -->* <**context-param**>  <**param-name**>contextConfigLocation</**param-name**>  <**param-value**>classpath:spring/applicationContext.xml</**param-value**>  </**context-param**>  *<!-- 设置Spring监听器 -->* <**listener**>  <**listener-class**>org.springframework.web.context.ContextLoaderListener</**listener-class**>  </**listener**>  *<!-- SpringMVC的DispatcherServlet配置 -->* <**servlet**>  <**servlet-name**>springmvc</**servlet-name**>  <**servlet-class**>org.springframework.web.servlet.DispatcherServlet</**servlet-class**>  <**init-param**>  <**param-name**>contextConfigLocation</**param-name**>  *<!-- SpringMVC配置文件 -->* <**param-value**>classpath:spring/springmvc.xml</**param-value**>  </**init-param**>  <**load-on-startup**>1</**load-on-startup**>  </**servlet**>  *<!-- DispatcherServlet映射 -->* <**servlet-mapping**>  <**servlet-name**>springmvc</**servlet-name**>  <**url-pattern**>\*.action</**url-pattern**>  </**servlet-mapping**> </**web-app**> |

现在，编写Controller和JSP测试。

在com.emp.controller.ControllerTest类中写：

|  |
| --- |
| **package** com.emp.controller;  **import** com.emp.domain.Employee; **import** com.emp.service.IEmployeeService; **import** org.springframework.stereotype.Controller; **import** org.springframework.web.bind.annotation.RequestMapping; **import** org.springframework.web.bind.annotation.ResponseBody; **import** javax.annotation.Resource; **import** javax.servlet.http.HttpServletRequest;  @Controller @RequestMapping(**"test"**) **public class** ControllerTest {  *// 使用employeeServiceImpl* @Resource(name = **"employeeService"**)  **private** IEmployeeService **employeeService**;   @RequestMapping(**"getEmp"**)  **public** String getEmp(HttpServletRequest request, Integer id) {  Employee employee = **employeeService**.getEmployee(id);  request.setAttribute(**"emp"**, employee);  **return "/test.jsp"**;  } } |

test.jsp代码：

|  |
| --- |
| <%@ **page contentType**="**text/html;charset=UTF-8**" **language**="**java**" **isELIgnored**="**false**" %> <**html**> <**head**>  <**title**>测试</**title**> </**head**> <**body**> 姓名：**${**emp.name**}** </**body**> </**html**> |

这样就编写完毕了。启动Tomcat服务器，请求“localhost:8080/项目名/test/getEmp.action?id=1”就能获得“1”号员工的信息。其他的，测试事务也无问题。

整合完毕！此SSM的示例项目也被我放到了Github上，地址是：

https://github.com/zhang13690/ssm\_demo

可供参考使用。

# 2. Mybatis逆向工程（Mybatis Generator）

Mybatis也有逆向工程，就是由数据库表生成POJO类、mapper类和映射文件的过程。Mybatis官方就提供了“Mybatis Generator”工具来生成Mybatis基本文件和代码。Mybatis Generator的介绍和下载可进入Mybatis的GitHub账号寻找。

我们使用下载包中提供的“mybatis-generator-core-1.3.5.jar”工具来自动生成文件。首先我们需要编写一个配置文件，如generatorConfig.xml，该配置文件用来指示Mybatis Generator如何根据数据库表生成对应的实体类、映射文件和mapper接口文件。

这里，配置的内容为：

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* **<!DOCTYPE generatorConfiguration  PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"  "http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd"*>*** <**generatorConfiguration**>  *<!-- 配置所引用jar包路径，这里要引用MySQL的JDBC驱动 -->* <**classPathEntry location="D:/DevTool/repository/mysql/mysql-connector-java/5.1.42/mysql-connector-java-5.1.42.jar"** />  *<!-- 配置context -->* <**context id="testTables" targetRuntime="MyBatis3"**>  <**commentGenerator**>  *<!-- 是否去除自动生成的注释 -->* <**property name="suppressAllComments" value="false"** />  </**commentGenerator**>  *<!-- 数据库连接信息 -->* <**jdbcConnection driverClass="com.mysql.jdbc.Driver" connectionURL="jdbc:mysql://localhost:3306/mytest" userId="root" password="123456"** />  *<!-- Java类型转换 -->* <**javaTypeResolver**>  *<!-- 配置forceBigDecimals表示是否将JDBC的DECIMAL和NUMERIC类型解析为Java的BigDecimal类型。这里设置为false，便是不转换成BigDecimal类型，而是转换成Integer类型 -->* <**property name="forceBigDecimals" value="false"** />  </**javaTypeResolver**>  *<!-- 在javaModelGenerator中配置POJO类的生成 -->  <!-- targetPackage表示目标包，targetProject表示目标路径 -->* <**javaModelGenerator targetPackage="com.emp.entity" targetProject="./src"**>  *<!-- enableSubPackages表示是否让schema作为包的后缀 -->  <!-- schema在下面会配置。一般不用 -->* <**property name="enableSubPackages" value="false"** />  *<!-- 从数据库返回的值将被清理前后的空格 -->* <**property name="trimStrings" value="true"** />  </**javaModelGenerator**>  *<!-- 在sqlMapGenerator中配置映射文件的生成 -->* <**sqlMapGenerator targetPackage="com.emp.mapper" targetProject="./src"**>  <**property name="enableSubPackages" value="false"** />  </**sqlMapGenerator**>  <**javaClientGenerator type="XMLMAPPER" targetPackage="com.emp.mapper" targetProject="./src"**>  <**property name="enableSubPackages" value="false"** />  </**javaClientGenerator**>  *<!-- 指定数据库表。这里用employee表作为示例 -->* <**table schema="" tableName="employee"** />  </**context**> </**generatorConfiguration**> |

这样，我们再用工具和文件生成文件。首先需要在该目录下新建src目录（避免下一步报错），然后在mybatis-generator-core-1.3.5.jar所在目录执行如下命令（即使用该工具）：

|  |
| --- |
| java -jar mybatis-generator-core-1.3.5.jar -configFile generatorConfig.xml |

这样，就在src目录下生成了包文件夹和相关的文件。以后如果想重新生成，需要清空src文件，因为该工具不会清空之前的文件。

我们把生成的文件拷贝到项目中即可使用（自行做调整）。

我们发现生成的mapper中提供了基本的CRUD方法，直接使用即可。除此之外，还自动生成了一个EmployeeExample.java文件，这样的文件是方便开发者进行有条件的选择（相当于where子句），使用方法如下：

|  |
| --- |
| **public** List<Employee> findById(String id) {  EmployeeExample employeeExample = **new** EmployeeExample();  employeeExample.clear();  *// 获得Criteria内部类。条件就封装在其中* EmployeeExample.Criteria criteria = employeeExample.createCriteria();  *// 可使用多个条件* criteria.andDeptidEqualTo(**"DE02"**);  criteria.andNameEqualTo(**"李四"**);  *// 最后查询* **return employeeMapper**.selectByExample(employeeExample); } |

当然，也可在update等操作中使用criteria：

|  |
| --- |
| **public void** updateEmployee() {  *// 修改的员工信息* Employee newEmp = **new** Employee();  newEmp.setName(**"李四的姓名"**);  *// 匹配的条件* EmployeeExample employeeExample = **new** EmployeeExample();  employeeExample.createCriteria().andIdEqualTo(**"EM02"**);  **employeeMapper**.updateByExampleSelective(newEmp, employeeExample); } |

说明：

（1）除了利用“andXxxEqualTo()”方法设置某个字段“相等”的条件，还能使用其他方法如“andXxxBetween()”设置符合的“区间”。其中“Xxx”就是字段名，Mybatis Generator为各个字段都生成了这些方法，简化了开发者工作，大家可自行了解其他方法。

（2）上述更新时使用了“updateByExampleSelective()”方法，这是带有Selective的方法，还有不带Selective的方法。还有其他的很多方法也有带和不带Selective两个“版本”，他们的区别是：不带Selective的方法，会更新传入的POJO对象的所有属性，如果某属性未设置（为null或者空、0等）那么也会更新，因此我们一般会先把原有对象查询出来再更新。但是如果是使用带有Selective的方法，该方法就不会更新未设置的属性（就像例子中只会更新“Name”属性），比较好用。还有updateByPrimary和updateByPrimarySelective方法，类似的。

可以看出，自动生成的代码方便了开发者。但对于一些较复杂的查询等操作，还需要自己写SQL映射。

我还在project-demos文件夹中提供了“ssmdemo”项目，该项目就是一个SSM的整合，利用Mybatis Generator生成了employee表相关文件进行示例操作。用户可直接使用“mvn clean package”命令得到war包放在Tomcat目录下运行（需要注意数据库的设置），访问网址“ http://localhost:8080/mbdemo/test/getEmp.action?id=员工编号 ”即可查看示例结果。

当然，Mybatis Generator网站上还提供了多种方式，例如集成工具到IDE中，但是我们为了不依赖任何东西，就用了上面jar包的方式。