本文介绍springMVC配置mybatisPlus的maven项目的例子。

前提：springMVC项目已经齐备，例子项目请转移到地址：

<https://github.com/kongchengsu/git/tree/master/CodeStandard2.0-mybatis_Plus/CodeStandard>

## 一、pom配置依赖

首先是mybatisPlus主体依赖，注意，如果项目中存在mybatis的依赖，请删除掉，因为mybatisPlus中已经存在了对mybatis的依赖，如果重复依赖，可能会出现版本错误。

<dependency>  
 <groupId>com.baomidou</groupId>  
 <artifactId>mybatis-plus</artifactId>  
 <version>3.3.2</version>  
</dependency>

然后是mybatisPlus的代码生成依赖

<dependency>  
 <groupId>com.baomidou</groupId>  
 <artifactId>mybatis-plus-generator</artifactId>  
 <version>3.3.2</version>  
</dependency>  
<dependency>  
 <groupId>org.freemarker</groupId>  
 <artifactId>freemarker</artifactId>  
 <version>2.3.30</version>  
</dependency>

其中的freemarker依赖用于代码生成模板的解析方法。mybatisPlus的模板解析方式有三种： Velocity（默认）、Freemarker、Beetl。本文使用freemarker进行解析。

## 二、spring配置文件applicationContext.xml的配置

将SqlSessionFactory的配置改为使用mybatisplus的。

<!-- 配置会话工厂SqlSessionFactory -->  
<bean id="sqlSessionFactory" class="com.baomidou.mybatisplus.extension.spring.MybatisSqlSessionFactoryBean">  
 <!-- 数据源 -->  
 <property name="dataSource" ref="dataSource"/>  
 <property name="mapperLocations" value="classpath:sqlmap/\*Mapper.xml"/>  
 <property name="typeAliasesPackage" value="com.jtexplorer.entity"/>  
 <!-- 分页插件-->  
 <property name="plugins">  
 <array>  
 <bean class="com.baomidou.mybatisplus.extension.plugins.PaginationInterceptor"/>  
 </array>  
 </property>  
</bean>

## 三、代码生成

代码生成功能将给出样例类，里边有详细的注释。

package com.jtexplorer.create.code;  
  
import com.baomidou.mybatisplus.core.toolkit.StringPool;  
import com.baomidou.mybatisplus.generator.AutoGenerator;  
import com.baomidou.mybatisplus.generator.InjectionConfig;  
import com.baomidou.mybatisplus.generator.config.\*;  
import com.baomidou.mybatisplus.generator.config.po.TableField;  
import com.baomidou.mybatisplus.generator.config.po.TableInfo;  
import com.baomidou.mybatisplus.generator.config.rules.IColumnType;  
import com.baomidou.mybatisplus.generator.config.rules.NamingStrategy;  
import com.baomidou.mybatisplus.generator.engine.FreemarkerTemplateEngine;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Set;  
  
public class CreateCode {  
 public static void main(String[] args) {  
 // 代码生成类  
 AutoGenerator mpg = new AutoGenerator();  
 // 配置模板，用于定义模板类型和设置模板地址（模板文件）  
 TemplateConfig templateConfig = new TemplateConfig();  
 // 配置模板地址注意不要带上.ftl/.vm, 会根据使用的模板引擎自动识别，注意的是这些配置如果不做配置，则会自动寻找以下地址（这些地址的模板是mybatis-plus-generator包中自带的）  
 // 注意/templates/指的位置是项目中的resources文件夹下的templates文件夹  
 // 注意，如果想要使用InjectionConfig来自定义生成文件的名称，则必须将对应的set方法设成null，如mapper.xml想要自定义，则必须增加语句templateConfig.setXml(null);否则会出错  
// templateConfig.setMapper("/templates/mapper.java");  
// templateConfig.setEntity(/templates/entity.java);  
// templateConfig.setService("/templates/service.java");  
// templateConfig.setServiceImpl("/templates/serviceImpl.java");  
// templateConfig.setController("/templates/controller.java");  
// templateConfig.setXml("/templates/mapper.xml");  
 mpg.setTemplate(templateConfig);  
 // 设置模板类型是freemarker  
 mpg.setTemplateEngine(new FreemarkerTemplateEngine());  
 // 获取项目地址：到项目最上层地址，比如该项目会到/CodeStandard  
 String projectPath = System.*getProperty*("user.dir");  
  
  
 // 设置公用配置  
 GlobalConfig globalConfig = new GlobalConfig();  
 // 设置文件输出地址  
 globalConfig.setOutputDir(projectPath + "/Main/src/main/java");  
 // 设置作者名称，用于类文件中的类注释的 @author 标签  
 globalConfig.setAuthor("苏空城");  
 // 未明  
 globalConfig.setOpen(false);  
 // 设置xml中是否生成<resultMap id="BaseResultMap" type="com.jtexplorer.entity.Exam">标签  
 globalConfig.setBaseResultMap(true);  
 // 设置xml中是否生成<sql id="Base\_Column\_List">标签  
 globalConfig.setBaseColumnList(true);  
  
 mpg.setGlobalConfig(globalConfig);  
  
 // 设置数据库连接信息  
 DataSourceConfig dataSourceConfig = new DataSourceConfig();  
 dataSourceConfig.setUrl("jdbc:mysql://192.168.123.168:3306/arts\_see?useSSL=false&serverTimezone=Hongkong&useUnicode=true&characterEncoding=utf-8&allowPublicKeyRetrieval=true");  
 dataSourceConfig.setDriverName("com.mysql.cj.jdbc.Driver");  
 dataSourceConfig.setUsername("root");  
 dataSourceConfig.setPassword("sandong2012");  
 mpg.setDataSource(dataSourceConfig);  
  
 // 设置各class文件输出包  
 PackageConfig pc = new PackageConfig();  
 // 设置父包，下面设置的其他包将自动加上该项设置  
 pc.setParent("com.jtexplorer");  
 // controller类的包，会自动加上com.jtexplorer  
 pc.setController("controller.admin");  
 // entity类的包，会自动加上com.jtexplorer  
 pc.setEntity("entity");  
 // mapper类的包，会自动加上com.jtexplorer  
 pc.setMapper("mapper");  
 // service类的包，会自动加上com.jtexplorer  
 pc.setService("service");  
 // service.impl类的包，会自动加上com.jtexplorer  
 pc.setServiceImpl("service.impl");  
 // 注意，因为xml要生成的地址不在java（globalConfig.setOutputDir的设置）文件中，因此在这里无法设置，需要使用InjectionConfig单独设置  
 mpg.setPackageInfo(pc);  
  
 // 自定义配置，用于设置需要单独设置输出路径和名称有特殊前后缀的类  
 InjectionConfig cfg = new InjectionConfig() {  
 @Override  
 public void initMap() {  
 }  
 };  
 // 自定义配置列表，在初始化FileOutConfig的方法中可以进行TableInfo（模板中${table.xx}中的table）信息的获取和修改  
 List<FileOutConfig> focList = new ArrayList<>();  
 // 首先是controller，这里的参数值是自己做的模板文件  
 focList.add(new FileOutConfig("/templates/controllerMy.java.ftl") {  
 @Override  
 public String outputFile(TableInfo tableInfo) {  
 // 获取列信息  
 List<TableField> tableFields = tableInfo.getFields();  
 // 由于mybatis中的datetime类的列会自动设置成LocalDateTime类型，但是这个类型在查询时会出现java.lang.AbstractMethodError: Method org/apache/commons/dbcp/DelegatingResultSet.getObject(Ljava/lang/String;Ljava/lang/Class;)Ljava/lang/Object; is abstract  
 // 因此，这里将LocalDateTime类型改为Date类型  
 tableFields.forEach(v->{  
 if("LocalDateTime".equals(v.getPropertyType())){  
 Set<String> packages = tableInfo.getImportPackages();  
 packages.remove("java.time.LocalDateTime");  
 IColumnType iColumnType = new IColumnType() {  
 @Override  
 public String getType() {  
 return "Date";  
 }  
  
 @Override  
 public String getPkg() {  
 return "java.util.Date";  
 }  
 };  
 v.setColumnType(iColumnType);  
 }  
 });  
 // 重新将列数据导入该类用于重新设置需要导入的包  
 tableInfo.setFields(tableFields);  
 // 设置controller类的类名的前后缀  
 tableInfo.setControllerName("Admin" + tableInfo.getEntityName() + "Controller");  
 // 自定义输出文件名 ， 如果你 Entity 设置了前后缀、此处注意 xml 的名称会跟着发生变化！！  
 return projectPath + "/Main/src/main/java/com/jtexplorer/controller/admin/" + tableInfo.getControllerName() + StringPool.*DOT\_JAVA*;  
 }  
 });  
 // 设置了controller的自定义配置之后，就必须将配置类中的该项设置为null，即使前文没有设置该项，也不可遗漏，否则，将还会生成一个默认名称的类  
 templateConfig.setController(null);  
 // 然后是mapper.xml，这里的配置值是默认的模板文件，该文件在自己的项目文件夹下没有，但是运行之后，在target文件夹下是存在的（是引的包中自带的）  
 focList.add(new FileOutConfig("/templates/mapper.xml.ftl") {  
 @Override  
 public String outputFile(TableInfo tableInfo) {  
 // 自定义输出文件名 ， 如果你 Entity 设置了前后缀、此处注意 xml 的名称会跟着发生变化！！  
 return projectPath + "/Main/src/main/resources/sqlmap/" + tableInfo.getEntityName() + "Mapper" + StringPool.*DOT\_XML*;  
 }  
 });  
 templateConfig.setXml(null);  
  
 // 将列表放进InjectionConfig类对象中  
 cfg.setFileOutConfigList(focList);  
 mpg.setCfg(cfg);  
  
 // 策略配置  
 StrategyConfig strategy = new StrategyConfig();  
 // 未明  
 strategy.setNaming(NamingStrategy.*underline\_to\_camel*);  
 // 未明  
 strategy.setColumnNaming(NamingStrategy.*underline\_to\_camel*);  
 // 未明  
 strategy.setEntityLombokModel(true);  
 // 未明  
 strategy.setRestControllerStyle(true);  
 // 表名，多个表的话可以使用英文逗号隔开，如：strategy.setInclude("exam","user");  
 strategy.setInclude("exam");  
 // 未明  
 strategy.setControllerMappingHyphenStyle(true);  
 mpg.setStrategy(strategy);  
 // 运行  
 mpg.execute();  
 }  
}

这里的要点是模板的编辑，这里以controller的模板为例介绍模板中的对应关系：

首先是模板

package **${**package.Controller**}**;  
  
import com.baomidou.mybatisplus.extension.plugins.pagination.Page;  
import **${**package.Entity**}**.**${**entity**}**;  
import **${**package.Service**}**.**${**table.serviceName**}**;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.web.bind.annotation.\*;  
  
import javax.annotation.Resource;  
import java.util.List;  
  
/\*\*  
\* <p>  
 \* **${**table.comment!**}** 接口类  
 \* </p>  
\*  
\* @author **${**author**}**\* @since **${**date**}**\*/  
@Slf4j  
@RestController  
@SuppressWarnings("SpringJavaAutowiringInspection")  
@RequestMapping(value = "/admin/**${**table.entityPath**}**")  
public class **${**table.controllerName**}** {  
 @Resource  
 private **${**table.serviceName**}** service;  
  
 /\*\*  
 \* 查询全部  
 \*  
 \* @return List  
 \*/  
 @PostMapping(value = "/selectAll")  
 public List<**${**entity**}**> selectAll() {  
 return service.list();  
 }  
  
 /\*\*  
 \* 分页查询全部  
 \*  
 \* @return List  
 \*/  
 @PostMapping(value = "/selectPage")  
 public List<**${**entity**}**> selectPage(@RequestParam(required = false,defaultValue = "1") int page,  
 @RequestParam(required = false,defaultValue = "10") int limit) {  
 Page<**${**entity**}**> pages = new Page<>(page,limit);  
 service.page(pages).getRecords();  
 System.out.println("总页数"+pages.getPages());  
 System.out.println("当前页记录数量"+pages.getSize());  
 System.out.println("总记录数量"+pages.getTotal());  
 return pages.getRecords();  
 }  
  
}

然后是生成效果：

package com.jtexplorer.controller.admin;  
  
import com.baomidou.mybatisplus.extension.plugins.pagination.Page;  
import com.jtexplorer.entity.Exam;  
import com.jtexplorer.service.IExamService;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.web.bind.annotation.\*;  
  
import javax.annotation.Resource;  
import java.util.List;  
  
*/\*\*  
\* <p>  
 \* 接口类  
 \* </p>  
\*  
\** ***@author*** *苏空城  
\** ***@since*** *2020-07-16  
\*/*@Slf4j  
@RestController  
@SuppressWarnings("SpringJavaAutowiringInspection")  
@RequestMapping(value = "/admin/exam")  
public class AdminExamController {  
 @Resource  
 private IExamService service;  
  
 */\*\*  
 \* 查询全部  
 \*  
 \** ***@return*** *List  
 \*/* @PostMapping(value = "/selectAll")  
 public List<Exam> selectAll() {  
 return service.list();  
 }  
  
 */\*\*  
 \* 分页查询全部  
 \*  
 \** ***@return*** *List  
 \*/* @PostMapping(value = "/selectPage")  
 public List<Exam> selectPage(@RequestParam(required = false,defaultValue = "1") int page,  
 @RequestParam(required = false,defaultValue = "10") int limit) {  
 Page<Exam> pages = new Page<>(page,limit);  
 service.page(pages).getRecords();  
 System.*out*.println("总页数"+pages.getPages());  
 System.*out*.println("当前页记录数量"+pages.getSize());  
 System.*out*.println("总记录数量"+pages.getTotal());  
 return pages.getRecords();   
 }  
  
}

上下比较可以发现。相关的对应关系，如果这里面的对应关系不够使用的，可以通过代码生成类中的TableInfo类获取和修改模板中的${table.xxx}。

## 四、分页

分页功能上文中其实已经展示出来了，首先是applicationContext.xml中配置分页插件，具体使用参考controller中的selectPage的接口逻辑。

另外，想要自定义分页查询的话，在mapper中Page类使用类似于RowBounds类，但是必须在参数第一个。

最后是注意，IPage是一个接口类，Page类则是IPage的实现类。

## 五、条件构造器相关

这里比较麻烦，详情参考官网介绍，这里做出使用的相关例子

*/\*\*  
 \* 使用条件查询  
 \*  
 \** ***@param*** *session session  
 \** ***@return*** *JsonResult  
 \*/* @PostMapping(value = "/selectAllWithQueryWrapper")  
 public List<Exam> selectAllWithQueryWrapper(@RequestParam(required = false) Long id,  
 @RequestParam(required = false) String name,  
 @RequestParam(required = false) String nameLike,  
 HttpSession session) {  
 QueryWrapper<Exam> queryWrapper = new QueryWrapper<>();  
// Map<String, Object> allEq = new HashMap<>();  
  
// allEq.put("exam\_id", id);  
// if (StringUtil.isNotEmpty(name))  
// allEq.put("exam\_name", name);  
// // 注意，该语句必须在allEq的赋值语句之后。  
// queryWrapper.allEq(allEq, true);  
 // 这里的第一个参数的意思是将allEq（第二个参数）遍历，将不符合StringUtil.isNotEmpty(v)条件的条件剔除  
// queryWrapper.allEq((k, v) ->  
// StringUtil.isNotEmpty(v)  
// , allEq, true);  
  
 // 相等  
 queryWrapper.eq("exam\_id", id);  
 // 不等于  
 queryWrapper.ne("exam\_id", id);  
 // 大于  
 queryWrapper.gt("exam\_id", id);  
 // 大于等于  
 queryWrapper.ge("exam\_id", id);  
 // 小于  
 queryWrapper.lt("exam\_id", id);  
 // 小于等于  
 queryWrapper.le("exam\_id", id);  
 // between  
 queryWrapper.between("exam\_id", id, id + 1);  
 // not between  
 queryWrapper.notBetween("exam\_id", id, id + 1);  
 // like(会自动在前后增加%)  
 queryWrapper.like("exam\_name", name);  
 // not like(会自动在前后增加%)  
 queryWrapper.notLike("exam\_name", name);  
  
 // likeLeft(会自动在前增加%)  
 queryWrapper.likeLeft("exam\_name", name);  
 // likeRight(会自动在后增加%)  
 queryWrapper.likeRight("exam\_name", name);  
  
 // is null  
 queryWrapper.isNull("exam\_name");  
 // is not null(会自动在后增加%)  
 queryWrapper.isNotNull("exam\_name");  
 // in  
 queryWrapper.in("exam\_name", id, id, id);  
 // not in  
 queryWrapper.notIn("exam\_name", id, id, id);  
  
 // in sql  
 queryWrapper.inSql("exam\_name", "select 1 from dual");  
  
 // not in sql  
 queryWrapper.notInSql("exam\_name", "select 1 from dual");  
  
 // 分组  
 queryWrapper.groupBy("exam\_name", "exam\_id");  
  
 // 排序  
 queryWrapper.orderBy(true, true, "exam\_name", "exam\_id");  
 // 排序  
 queryWrapper.orderByAsc("exam\_name", "exam\_id");  
 // 排序  
 queryWrapper.orderByDesc("exam\_name", "exam\_id");  
  
 // having  
 queryWrapper.having("sum(exam\_id) > {0} and sum(exam\_id) < {1}", id, id);  
 queryWrapper.having("sum(exam\_id) > 10");  
  
 // or  
 queryWrapper.eq("exam\_id", 100).or().eq("exam\_id", 200);  
 // or嵌套  
 queryWrapper.or(i -> i.eq("exam\_id", 100).eq("exam\_id", 100));  
  
 // and嵌套  
 queryWrapper.and(i -> i.eq("exam\_id", 100).eq("exam\_id", 100));  
  
 // 正常嵌套 不带 AND 或者 OR-----nested  
 queryWrapper.nested(i -> i.eq("exam\_id", 100).eq("exam\_id", 100));  
  
 // 拼接 sql  
 queryWrapper.apply("date\_format(now(),'%Y-%m-%d') = '2008-08-08'");  
 queryWrapper.apply("date\_format(now(),'%Y-%m-%d') = {0} and date\_format(now(),'%Y-%m-%d') = {1} ", "2008-08-10", "2008-08-09");  
  
 // last 无视优化规则直接拼接到 sql 的最后 只能调用一次,多次调用以最后一次为准 有sql注入的风险,请谨慎使用  
 queryWrapper.last("limit 10");  
  
 // exists  
 queryWrapper.exists("select 1 from dual");  
  
 // not exists  
 queryWrapper.notExists("select 1 from dual");  
  
 List<Exam> resultExam = service.list(queryWrapper);  
  
 queryWrapper = new QueryWrapper<>();  
 // 设置查出列  
 // 第二类方法为:过滤查询字段(主键除外),入参不包含 class 的调用前需要wrapper内的entity属性有值! 这两类方法重复调用以最后一次为准  
 queryWrapper.select("exam\_id", "exam\_name");  
 // 莫名出现空指针异常  
// queryWrapper.select(i->i.getProperty().startsWith("exam"));  
 resultExam = service.list(queryWrapper);  
  
 // 用于更新的，更新条件的拼接与查询的类相同  
 UpdateWrapper<Exam> updateWrapper = new UpdateWrapper<>();  
 updateWrapper.set("exam\_remark", "aaa");  
 updateWrapper.setSql("exam\_remark = 'bbb'");  
  
 boolean a = service.update(updateWrapper);  
  
 IPage<Exam> iPage = new Page<>(10, 1);  
 iPage.setTotal(service.count());  
 service.page(iPage, queryWrapper);  
 resultExam = iPage.getRecords();  
 System.*out*.println(a);  
 return resultExam;  
 }

值得注意的是官网明确警告，在controller中不可使用条件构建类QueryWrapper作为传值参数，因为该类体量很大，且使用该类传值其实就类似于使用map进行传值。有sql注入的风险。