SpringBoot 整合 Druid+ 全局事务管理 + Mybatis-Plus+ 代码生成器

作者：hjljy

原文链接：https://ld246.com/article/1594022886181

来源网站：链滴

SpringBoot整合Druid+全局事务管理+Mybatis-Plus代码生成器

在springboot开发当中，Druid，全局事务管理，代码生成器都是非常实用的，特此记录下整合的过程

整合Druid连接池

springboot默认的连接池是：HikariCP，但是Druid的功能相对来说比较全面。

数据库连接池了解和常用连接池对比Druid连接池官网

第一步：引入相关JAR

```
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid-spring-boot-starter</artifactId>
    <version>1.1.22</version>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
```

第二步：配置相关参数

```
spring:
  datasource:
    type: com.alibaba.druid.pool.DruidDataSource
    druid:
      name: 数据源名称
```

```
    driver-class-name: com.mysql.jdbc.Driver
    username: root
    password: 123456
    url: jdbc:mysql://127.0.0.1:3306/springboot?characterEncoding=utf8&useSSL=false
    # 连接池的配置信息
    # 初始化大小，最小，最大
    initial-size: 5
    min-idle: 5
    maxActive: 20
    # 配置获取连接等待超时的时间
    maxWait: 60000
    # 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒
    timeBetweenEvictionRunsMillis: 60000
    # 配置一个连接在池中最小生存的时间，单位是毫秒
    minEvictableIdleTimeMillis: 300000
    validationQuery: SELECT 1
    testWhileIdle: true
    testOnBorrow: false
    testOnReturn: false
    # 打开PSCache，并且指定每个连接上PSCache的大小
    poolPreparedStatements: true
    maxPoolPreparedStatementPerConnectionSize: 20
    # 配置监控统计拦截的filters，去掉后监控界面sql无法统计，'wall'用于防火墙
    filters: stat,wall,slf4j,config
    # 通过connectProperties属性来打开mergeSql功能；慢SQL记录
    connectionProperties: druid.stat.mergeSql\=true;druid.stat.slowSqlMillis\=5000
    web-stat-filter:
      enabled: true
      url-pattern: "/"
      exclusions: "*.js,*.gif,*.jpg,*.bmp,*.png,*.css,*.ico,/druid/*"
    stat-view-servlet:
      enabled: true
      url-pattern: "/druid/*"
      login-username: admin  # 登录账号   不设置就不需要登录就可以访问druid可视化界面
      login-password: 123456 # 登录密码
      reset-enable: false
      allow: ""  # 白名单 表示所有
      deny: 192.168.1.12 # 黑名单
```

第三步：在浏览器当中输入：http://127.0.0.1/druid/index.html 即可进入可视化界面

全局事务管理器

springboot当中添加事务直接使用注解@Transactional 即可，但是每个方法都要添加比较麻烦，可直接通过切面的方式添加一个全局的事务管理器。注意事项是，要注意方法名开头的问题

```
@Configuration
@EnableTransactionManagement
public class TransactionConfiguration {

    /**
     * 配置全局事务的切点为service层的所有方法  AOP切面表达式 可参考（https://blog.csdn.net/yc921244819/article/details/106599489）
     * TODO 设置service层所在位置
```

```
     */
    private static final String AOP_POINTCUT_EXPRESSION = "execution (* cn.hjljy.fastboot..*.service..*.*(..))";

    /**
     * 注入事务管理器
     */
    @Autowired
    private TransactionManager transactionManager;


    /**
     * 配置事务拦截器
     */
    @Bean
    public TransactionInterceptor txAdvice() {

        RuleBasedTransactionAttribute txAttrRequired = new RuleBasedTransactionAttribute();
        txAttrRequired.setName("REQUIRED事务");
        //设置事务传播机制，默认是PROPAGATION_REQUIRED：如果当前存在事务，则加入该事务如果当前没有事务，则创建一个新的事务
        txAttrRequired.setPropagationBehavior(TransactionDefinition.PROPAGATION_REQUIRED)

        //设置异常回滚为Exception 默认是RuntimeException
        List<RollbackRuleAttribute> rollbackRuleAttributes = new ArrayList<>();
        rollbackRuleAttributes.add(new RollbackRuleAttribute(Exception.class));
        txAttrRequired.setRollbackRules(rollbackRuleAttributes);

        RuleBasedTransactionAttribute txAttrRequiredReadOnly = new RuleBasedTransactionAttribute();
        txAttrRequiredReadOnly.setName("SUPPORTS事务");
        //设置事务传播机制，PROPAGATION_SUPPORTS：如果当前存在事务，则加入该事务；如果前没有事务，则以非事务的方式继续运行
        txAttrRequiredReadOnly.setPropagationBehavior(TransactionDefinition.PROPAGATION_UPPORTS);
        //设置异常回滚为Exception 默认是RuntimeException
        txAttrRequiredReadOnly.setRollbackRules(rollbackRuleAttributes);
        txAttrRequiredReadOnly.setReadOnly(true);

        /*事务管理规则，声明具备事务管理的方法名*/
        NameMatchTransactionAttributeSource source = new NameMatchTransactionAttributeSource();
        //方法名规则限制，必须以下列开头才会加入事务管理当中
        source.addTransactionalMethod("add*", txAttrRequired);
        source.addTransactionalMethod("save*", txAttrRequired);
        source.addTransactionalMethod("create*", txAttrRequired);
        source.addTransactionalMethod("insert*", txAttrRequired);
        source.addTransactionalMethod("submit*", txAttrRequired);
        source.addTransactionalMethod("del*", txAttrRequired);
        source.addTransactionalMethod("remove*", txAttrRequired);
        source.addTransactionalMethod("update*", txAttrRequired);
        source.addTransactionalMethod("exec*", txAttrRequired);
        source.addTransactionalMethod("set*", txAttrRequired);
```

```
        //对于查询方法，根据实际情况添加事务管理 可能存在查询多个数据时，已查询出来的数据刚被改变的情况
        source.addTransactionalMethod("get*", txAttrRequiredReadOnly);
        source.addTransactionalMethod("select*", txAttrRequiredReadOnly);
        source.addTransactionalMethod("query*", txAttrRequiredReadOnly);
        source.addTransactionalMethod("find*", txAttrRequiredReadOnly);
        source.addTransactionalMethod("list*", txAttrRequiredReadOnly);
        source.addTransactionalMethod("count*", txAttrRequiredReadOnly);
        source.addTransactionalMethod("is*", txAttrRequiredReadOnly);
        return new TransactionInterceptor(transactionManager, source);
    }

    /**
     * 设置切面
     */
    @Bean
    public Advisor txAdviceAdvisor() {
        AspectJExpressionPointcut pointcut = new AspectJExpressionPointcut();
        pointcut.setExpression(AOP_POINTCUT_EXPRESSION);
        return new DefaultPointcutAdvisor(pointcut, txAdvice());
    }
}
```

整合Mybatis-Plus

第一步：引入JAR包

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

第二步：添加配置信息

```
mybatis-plus:
  mapper-locations: classpath:mapper/*.xml  #xml所在位置  不设置默认是在mapper类同级
  configuration:
    mapUnderscoreToCamelCase: true  # 开启驼峰匹配  默认为true
    log-impl: org.apache.ibatis.logging.stdout.StdOutImpl  # 打印sql语句和入参数据
  global-config:
    db-config:
      logic-delete-value: 1 #逻辑删除  配合@TableLogic注解
      logic-not-delete-value: 0 #逻辑不删除
      update-strategy: not_null # 更新时字段如果为null，就不进行更新该字段。
      insert-strategy: not_null # 插入时如果字段为null,就不插入数据，建议数据库表字段设置默认值
```

第三步：添加分页和mapper扫描

```
@Configuration
@MapperScan("cn.hjljy.fastboot.mapper")
public class MybatisPlusConfiguration {
    /**
     * mybatis-plus分页插件
```

```
     */
    @Bean
    public PaginationInterceptor paginationInterceptor() {
        PaginationInterceptor page = new PaginationInterceptor();
        //设置分页数据库类型
        page.setDbType(DbType.MYSQL);
        page.setDialect(new MySqlDialect());
        //优化count sql
        page.setCountSqlParser(new JsqlParserCountOptimize(true));
        //设置每页最大值
        page.setLimit(999L);
        return page;
    }
}
```

第四步：创建一个Mapper类继承BaseMapper，就可以简单使用了。

可以参考官方文档入门：https://mp.baomidou.com/guide/quick-start.html

整合代码生成器

AutoGenerator 是 MyBatis-Plus 的代码生成器，通过 AutoGenerator 可以快速生成 Entity、Mapper、Mapper XML、Service、Controller 等各个模块的代码，极大的提升了开发效率

考虑到dto和po在大部分情况下字段都是一样的，官方未提供DTO,所以可以拷贝一份entity.java.vm改为dto.java.vm放在resources目录下面。然后根据自定义提示进行修改。

具体结果如下：

```
package $!{cfg.dtoPackage};

#foreach($pkg in ${table.importPackages})
import ${pkg};
#end
#if(${swagger2})
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
#end
#if(${entityLombokModel})
import lombok.Data;
import lombok.EqualsAndHashCode;
#if(${chainModel})
import lombok.experimental.Accessors;
#end
#end

/**
 * <p>
 * $!{table.comment}
 * </p>
 *
 * @author ${author}
 * @since ${date}
 */
#if(${entityLombokModel})
```

```
@Data
 #if(${superEntityClass})
@EqualsAndHashCode(callSuper = true)
 #else
@EqualsAndHashCode(callSuper = false)
 #end
 #if(${chainModel})
@Accessors(chain = true)
 #end
#end
#if(${table.convert})
@TableName("${table.name}")
#end
#if(${swagger2})
@ApiModel(value="${entity}Dto对象", description="$!{table.comment}")
#end
#if(${superEntityClass})
public class ${entity}Dto extends ${superEntityClass}#if(${activeRecord})<${entity}>#end {
#elseif(${activeRecord})
public class ${entity}Dto extends Model<${entity}> {
#else
public class ${entity}Dto implements Serializable {
#end

#if(${entitySerialVersionUID})
    private static final long serialVersionUID=1L;
#end
## ----------  BEGIN 字段循环遍历  ----------
#foreach($field in ${table.fields})

#if(${field.keyFlag})
#set($keyPropertyName=${field.propertyName})
#end
#if("$!field.comment" != "")
 #if(${swagger2})
   @ApiModelProperty(value = "${field.comment}")
 #else
   /**
    * ${field.comment}
    */
 #end
#end
#if(${field.keyFlag})
## 主键
 #if(${field.keyIdentityFlag})
   @TableId(value = "${field.annotationColumnName}", type = IdType.AUTO)
 #elseif(!$null.isNull(${idType}) && "$!idType" != "")
   @TableId(value = "${field.annotationColumnName}", type = IdType.${idType})
 #elseif(${field.convert})
   @TableId("${field.annotationColumnName}")
 #end
## 普通字段
#elseif(${field.fill})
## -----   存在字段填充设置   -----
```

```
  #if(${field.convert})
    @TableField(value = "${field.annotationColumnName}", fill = FieldFill.${field.fill})
  #else
    @TableField(fill = FieldFill.${field.fill})
  #end
#elseif(${field.convert})
    @TableField("${field.annotationColumnName}")
#end
## 乐观锁注解
#if(${versionFieldName}==${field.name})
    @Version
#end
## 逻辑删除注解
#if(${logicDeleteFieldName}==${field.name})
    @TableLogic
#end
    private ${field.propertyType} ${field.propertyName};
#end
## ----------  END 字段循环遍历  ----------

#if(!${entityLombokModel})
#foreach($field in ${table.fields})
  #if(${field.propertyType.equals("boolean")})
    #set($getprefix="is")
  #else
    #set($getprefix="get")
  #end

    public ${field.propertyType} ${getprefix}${field.capitalName}() {
        return ${field.propertyName};
    }

  #if(${chainModel})
    public ${entity} set${field.capitalName}(${field.propertyType} ${field.propertyName}) {
  #else
    public void set${field.capitalName}(${field.propertyType} ${field.propertyName}) {
  #end
        this.${field.propertyName} = ${field.propertyName};
  #if(${chainModel})
        return this;
  #end
    }
#end
## --foreach end---
#end
## --end of #if(!${entityLombokModel})--

#if(${entityColumnConstant})
  #foreach($field in ${table.fields})
    public static final String ${field.name.toUpperCase()} = "${field.name}";

  #end
#end
#if(${activeRecord})
```

```
    @Override
    protected Serializable pkVal() {
  #if(${keyPropertyName})
        return this.${keyPropertyName};
  #else
        return null;
  #end
    }

#end
#if(!${entityLombokModel})
    @Override
    public String toString() {
        return "${entity}{" +
  #foreach($field in ${table.fields})
    #if($!{foreach.index}==0)
        "${field.propertyName}=" + ${field.propertyName} +
    #else
        ", ${field.propertyName}=" + ${field.propertyName} +
    #end
  #end
        "}";
    }
#end
}
```

具体代码生成器的执行代码如下：

```
public class CodeGenerator {

    public static void main(String[] args) {
        // 代码生成器
        AutoGenerator mpg = new AutoGenerator();

        // 全局配置
        GlobalConfig gc = new GlobalConfig();
        String projectPath = System.getProperty("user.dir");
        gc.setOutputDir(projectPath + "/src/main/java");
        gc.setAuthor("海加尔金鹰（www.hjljy.cn）");
        gc.setOpen(false);
        //设置实体类后缀
        gc.setEntityName("%sPo");
        //实体属性 Swagger2 注解
        gc.setSwagger2(true);
        gc.setBaseColumnList(true);
        gc.setBaseResultMap(true);
        mpg.setGlobalConfig(gc);

        // 数据源配置
        DataSourceConfig dsc = new DataSourceConfig();
        dsc.setUrl("jdbc:mysql://localhost:3306/springboot?serverTimezone=GMT&useUnicode=
rue&characterEncoding=UTF-8&zeroDateTimeBehavior=convertToNull&allowMultiQueries=t
ue");
        dsc.setDriverName("com.mysql.jdbc.Driver");
```

```
        dsc.setUsername("root");
        dsc.setPassword("123456");
        mpg.setDataSource(dsc);

        // 包配置
        PackageConfig pc = new PackageConfig();
        pc.setModuleName(null);
        String scanner = scanner("请输入整体业务包名");
        String modelName = StringUtils.isBlank(scanner) ? "" : "."+scanner;
        //moduleName是整体分模块

        pc.setParent("cn.hjljy.fastboot");
        pc.setMapper("mapper"+modelName);
        pc.setService("service"+modelName);
        pc.setServiceImpl("service"+modelName+".impl");
        pc.setEntity("pojo"+modelName+".po");
        pc.setController("controller"+modelName);
        mpg.setPackageInfo(pc);


        String dtoPath = pc.getParent() + ".pojo.dto";
        // 配置模板
        TemplateConfig templateConfig = new TemplateConfig();
        // 不输出默认的XML 默认生成的xml在mapper层里面
        templateConfig.setXml(null);
        mpg.setTemplate(templateConfig);

        //配置自定义输出的文件   xml和dto
        //模板引擎是 velocity
        String xmlTemplatePath = "/templates/mapper.xml.vm";
        // 自定义输出配置
        List<FileOutConfig> focList = new ArrayList<>();
        // 自定义配置会被优先输出
        focList.add(new FileOutConfig(xmlTemplatePath) {
            @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
！！
                return projectPath + "/src/main/resources/mapper/" + scanner
                        + "/" + tableInfo.getEntityName() + "Mapper" + StringPool.DOT_XML;
            }
        });

        String dtoTemplatePath = "/dto.java.vm";
        // 自定义配置会被优先输出
        focList.add(new FileOutConfig(dtoTemplatePath) {
            @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
！！
                return projectPath + "/src/main/java/cn/hjljy/fastboot/pojo/"+scanner+"/dto/" +
                        tableInfo.getEntityName() + "Dto" + StringPool.DOT_JAVA;
            }
        });
```

```
        // 自定义配置
        InjectionConfig cfg = new InjectionConfig() {

            @Override
            public void initMap() {
                Map<String, Object> map = new HashMap<>();
                map.put("dtoPackage", dtoPath);
                this.setMap(map);
            }
        };
        cfg.setFileOutConfigList(focList);
        mpg.setCfg(cfg);
        // 策略配置
        StrategyConfig strategy = new StrategyConfig();
        strategy.setNaming(NamingStrategy.underline_to_camel);
        strategy.setColumnNaming(NamingStrategy.underline_to_camel);
        strategy.setEntityLombokModel(true);
        strategy.setRestControllerStyle(true);
        strategy.setInclude(scanner("表名，多个英文逗号分割").split(","));
        strategy.setControllerMappingHyphenStyle(true);
        //设置逻辑删除字段
        strategy.setLogicDeleteFieldName("status");
        mpg.setStrategy(strategy);
        mpg.setTemplateEngine(new VelocityTemplateEngine());
        mpg.execute();
    }

    /**
     * <p>
     * 读取控制台内容
     * </p>
     */
    public static String scanner(String tip) {
        Scanner scanner = new Scanner(System.in);
        StringBuilder help = new StringBuilder();
        help.append("请输入" + tip + "：");
        System.out.println(help.toString());
        if (scanner.hasNext()) {
            String ipt = scanner.next();
            if (StringUtils.isNotEmpty(ipt)) {
                return ipt;
            }
        }
        throw new MybatisPlusException("请输入正确的" + tip + "！");
    }
}
```

总结

算是框架里面非常基础的一些东西。不过能够提高不少的开发效率！！！