SpringBoot集成FreeMarker实现自动生成增删改查代码

简单例子： https://zhuanlan.zhihu.com/p/507267815
https://blog.csdn.net/weixin_44001965/article/details/105748661
https://blog.51cto.com/u_15459458/4831631

1.测试

pom.xml导入依赖

```
<!-- FreeMarker自动生成代码依赖 -->
<dependency>
    <groupId>org.freemarker</groupId>
    <artifactId>freemarker</artifactId>
</dependency>
```

避免报错，在pom.xml下的`<build>`补一下

```
<resources>
    <resource>
    <directory>${basedir}/src/main/java</directory>
        <includes>
        <include>**/*.*</include>
        </includes>
        <excludes>
        <exclude>**/*.java</exclude>
        </excludes>
        <filtering>false</filtering>
    </resource>
dd</resources>
```

test.ftl 放在templates下

```
<#-- assign指令 在ftl模板中定义数据存入到root节点下 --><#assign name="傻子">
<#--然后就可以取出name的值-->
${name}

你好，${username}

<#--- if指令 --><#if password=1234>
    简单密码
```

```
<#elseif password=123456>
    一般密码
<#else>
    复杂密码
</#if>

<#-- list指令 迭代循环 --><#list list as abc>
    ${abc}
</#list>
```

测试代码

```
package com.example.springb_protect.test;

import freemarker.cache.FileTemplateLoader;
import freemarker.template.Configuration;
import freemarker.template.Template;
import freemarker.template.TemplateException;
import org.junit.Test;

import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class autoNewCode {

    @Test
    public void test() throws IOException, TemplateException {
        //1，创建FreeMarker的配置类
        Configuration cfg = new Configuration();
        //2，指定模板加载器，将模板加入缓存中
        //文件路径加载器，获取到templates文件的路径
        String templates =
this.getClass().getClassLoader().getResource("templates").getPath();
        System.out.println(templates);
        FileTemplateLoader fileTemplateLoader = new FileTemplateLoader(new
File(templates));
        cfg.setTemplateLoader(fileTemplateLoader);
        //3，获取模板
        Template template = cfg.getTemplate("test.ftl");
        //4，构造数据模型
        Map<String, Object> map = new HashMap<String, Object>();
```

```
        map.put("username", "测试人员");
        map.put("password", 1234);

        List<String> list = new ArrayList<>();
        list.add("第一个");
        list.add("第二个");
        map.put("list", list);

        //5，文件输出
        /**
         * 处理模型
         *      参数一 数据模型
         *      参数二 writer对象（FileWriter（文件输出），printWriter（控制台输
出））
         */
        //template.process(map,new FileWriter(new File("D:\\a.txt")));
        template.process(map, new PrintWriter(System.out));
    }
}
```

2.测试生成pojo类

创建4个实体类

Columu.java

```
package com.example.springb_protect.autoNewCode.pojo;

import lombok.Data;

@Data       //使用这个注解可以省去代码中大量的get()、 set()、 toString()等方法;
public class Column {
    //列名称
    private String columnName;
    //处理后的列名称
    private String columnName2;
    //列类型
    private String columnType;
    //列在数据库中的类型
    private String columnDbType;
    //本工程暂不处理备注和主键
    //列备注id
    private String columnComment;
    //是否是主键
    private String columnKey;
}
```

DataBase.java

```
package com.example.springb_protect.autoNewCode.pojo;

import lombok.Data;

@Data       //使用这个注解可以省去代码中大量的get()、 set()、 toString()等方法;
public class DataBase {
    private static String mysqlUrl = "jdbc:mysql://[ip]:[port]/[db]?
useUnicode=true&characterEncoding=utf-8&serverTimezone=UTC";
    private static String oracleUrl = "jdbc:oracle:thin:@[ip]:[port]:[db]";
    private String dbType;//数据库类型
    private String userName;
    private String passWord;
    private String driver;
    private String url;

    public DataBase() {
    }

    public DataBase(String dbType) {
        this(dbType, "127.0.0.1", "3306", "");
    }

    public DataBase(String dbType, String db) {
        this(dbType, "127.0.0.1", "3306", db);
    }

    /**
     * @param dbType 数据库类型 mysql/oracle
     * @param ip     ip
     * @param port   3306
     * @param db     数据库名称 test
     */    public DataBase(String dbType, String ip, String port, String db)
{
        this.dbType = dbType;
        if ("MYSQL".equals(dbType.toUpperCase())) {
            this.driver = "com.mysql.cj.jdbc.Driver";
            this.url = mysqlUrl.replace("[ip]", ip).replace("[port]",
port).replace("[db]", db);
        } else {
            this.driver = "oracle.jdbc.driver.OracleDriver";
            this.url = oracleUrl.replace("[ip]", ip).replace("[port]",
port).replace("[db]", db);
        }
    }
}
```

Settings.java

```
package com.example.springb_protect.autoNewCode.pojo;

import lombok.Data;

@Data       //使用这个注解可以省去代码中大量的get()、 set()、 toString()等方法;
public class Settings {
    private String project = "example";
    private String pPackage = "com.example.demo";
    private String projectComment;
    private String author;
    private String path1 = "com";
    private String path2 = "example";
    private String path3 = "demo";
    private String pathAll;
}
```

Table.java

```
package com.example.springb_protect.autoNewCode.pojo;

import lombok.Data;

import java.util.List;

@Data       //使用这个注解可以省去代码中大量的get()、 set()、 toString()等方法;
public class Table {
    //表名称
    private String name;
    //处理后的表名称
    private String name2;
    //介绍
    private String comment;
    //主键列
    private String key;
    private List<Column> columnList;
}
```

和src同层下的 properties/typeConverter.properties

```
# sql类型和java类型的替换规则
BIT=boolean    LONGVARCHAR=String
CHAR=String
VARCHAR=String
DATE=java.sql.Date TIME=java.sql.Time TIMESTAMP=java.sql.Timestamp
BIGINT=Long
LONGTEXT=String
TEXT=String
INT=Integer

# table的前缀或者后缀
tableRemovePrefixes="tb_,co_"
```

工具类

DataBaseUtils.java

```
package com.example.springb_protect.autoNewCode.utils;

import com.example.springb_protect.autoNewCode.pojo.Column;
import com.example.springb_protect.autoNewCode.pojo.DataBase;
import com.example.springb_protect.autoNewCode.pojo.Table;
import org.junit.Test;
import javax.servlet.http.HttpSession;
import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.List;

/**
 * 方法介绍:
 * 1，获取数据库连接
 * 2，获取数据库列表
 * 3，获取数据库中的所有表和字段并构造实体类
 * 4，根据表名的截取操作生成类名
 */
public class DataBaseUtils {
    //获取数据库连接
    public static Connection getConnection(DataBase db) throws Exception {
        //获取连接
        Class.forName(db.getDriver());//注册驱动
        Connection connection = DriverManager.getConnection(db.getUrl(),
db.getUserName(), db.getPassWord());
```

```
        return connection;
    }

    //获取数据库列表
    public static List<String> getShemas(DataBase db) throws Exception {
        Connection connection = getConnection(db);
        //获取元数据
        DatabaseMetaData metaData = connection.getMetaData();
        //获取所有数据库列表
        ResultSet resultSet = metaData.getCatalogs();
        List<String> list = new ArrayList<>();
        while (resultSet.next()) {
            list.add(resultSet.getString(1));
        }
        resultSet.close();
        connection.close();
        return list;
    }

    //获取数据库中的所有表和字段并构造实体类（相当于一键生成数据库中所有表的增删改查代
码）
    public static List<Table> getDbInfo(DataBase db, String dbss) throws
Exception {
        //获取连接
        Connection connection = getConnection(db);
        //获取元数据
        DatabaseMetaData metaData = connection.getMetaData();
        List<Table> list = new ArrayList<>();
        //获取当前数据库的所有表
        //String dbss = (String) session.getAttribute("db");
        ResultSet tables = metaData.getTables(dbss, null, null, new String[]
{"TABLE"});
        while (tables.next()) {
            //表名
            String table_name = tables.getString("TABLE_NAME");
            //构造生成对应实体类的类名
            String className = removePrefix(table_name);
            //主键
            ResultSet primaryKeys = metaData.getPrimaryKeys(null, null,
table_name);
            //对主键遍历的原因（或许一张表有多个主键）
            String keys = "";
            while (primaryKeys.next()) {
                String keyName = primaryKeys.getString("COLUMN_NAME");
                keys += keyName + ",";
            }
            Table tab = new Table();
            tab.setName(table_name);
```

```
            tab.setName2(className);
            tab.setKey(keys);
            //处理表中的所有字段
            ResultSet columns = metaData.getColumns(dbss, null, table_name,
null);
            List<Column> cols = new ArrayList<>();
            while (columns.next()) {
                Column column = new Column();
                //列名称
                String column_name = columns.getString("COLUMN_NAME");
                //java实体的属性名
                String attName = column_name;
                //java类型和数据库类型
                String type_name = columns.getString("TYPE_NAME");
                String javaType = PropertiesUtils.customMap.get(type_name);
                column.setColumnName(column_name);
                column.setColumnName2(attName);
                column.setColumnDbType(type_name);
                column.setColumnType(javaType);
                cols.add(column);
            }
            tab.setColumnList(cols);
            list.add(tab);
            //关闭连接，释放资源
            columns.close();
            primaryKeys.close();
        }
        tables.close();
        connection.close();
        return list;
    }

    //根据表名的截取操作生成类名
    public static String removePrefix(String tableName) {
        //从自定义的配置文件中拿到前缀的配置
        String prefixes =
PropertiesUtils.customMap.get("tableRemovePrefixes");
        //这里就不字符串处理了，直接把表名当类名用了
        String replace = tableName;
        return replace;
    }
}
```

pojo模板

```
package ${pPackage}.model;

import com.fasterxml.jackson.annotation.JsonInclude;
import lombok.Data;

@Data
@JsonInclude(JsonInclude.Include.NON_NULL)
public class ${table.name?cap_first} {
<#list table.columnList as t>
    private ${t.columnType} ${t.columnName};
</#list>
}
```

单元测试

```
@Test
public void test3() throws Exception {
    String username = "数据库用户名";
    String password = "数据库密码";
    String ip = "服务器IP地址";
    String db = "数据库库名";
    DataBase dataBase = new DataBase("MYSQL",ip,"3306",db);
    dataBase.setUserName(username);
    dataBase.setPassWord(password);
    List<Table> tables = DataBaseUtils.getDbInfo(dataBase, "数据库库名");
    /*for (Table table : tables) {
        //对每个table进行代码生成
        System.out.println(table.toString());
    }*/    Table t = tables.get(0);

    /* 转成Map */
    Map<String, Object> map = new HashMap<>();
    //自定义配置
    map.putAll(PropertiesUtils.customMap);
    //元数据
    map.put("table", t);
    //settings
    String packagename = "com.ftx.demo";
    String projectEngName = "qaq";
    map.put("project", projectEngName);
    map.put("pPackage", packagename);
    map.put("path1", "com");
    map.put("path2", "ftx");
    map.put("path3", "demo");
    //类名
```

```
        map.put("className", t.getName2());
        System.out.println(map);

        //1，创建FreeMarker的配置类
        Configuration cfg = new Configuration();
        //2，指定模板加载器，将模板加入缓存中
        //文件路径加载器，获取到templates文件的路径
        String templates =
    this.getClass().getClassLoader().getResource("templates").getPath();
        FileTemplateLoader fileTemplateLoader = new FileTemplateLoader(new
    File(templates));
        cfg.setTemplateLoader(fileTemplateLoader);
        //3，获取模板
        Template template = cfg.getTemplate("pojo.ftl");
        //4，构造数据模型
        //5，文件输出
        /**
         * 处理模型
         *       参数一 数据模型
         *       参数二 writer对象（FileWriter（文件输出），printWriter（控制台输出））
         */
        //template.process(map,new FileWriter(new File("D:\\a.txt")));
        template.process(map, new PrintWriter(System.out));

    }
```

3.其他模板

dao.ftl

```
package ${pPackage}.dao;


import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;
import org.apache.ibatis.annotations.Delete;
import org.apache.ibatis.annotations.Update;
import java.util.List;

<#--
    这次主键默认id是int
-->
@Mapper
public interface ${table.name?cap_first}Mapper {
    /**
```

```
     * 查询所有数据
     */
    @Select("select * from ${table.name}")
    List<${table.name?cap_first}> list${table.name?cap_first}();

    /**
     * 根据id获取单条数据
     */
    @Select("select * from ${table.name} where ${table.key}=${r"#
{"}${table.key?lower_case}}")
    ${table.name?cap_first} get${table.name?cap_first}By${table.key?
cap_first}(int ${table.key?lower_case});

    /**
     * 分页查询数据
     */
    @Select("select * from ${table.name} limit <#noparse>#{first},#{second}
</#noparse>;")
    List<${table.name?cap_first}> list${table.name?cap_first}ByPage(int
first, int second);

    /**
     * 插入数据
     */
    @Insert("insert into ${table.name}(<#list table.columnList as t><#if
t.columnName!=table.key>${t.columnName}<#if t_has_next >,</#if></#if>
</#list>) values(<#list table.columnList as t><#if t.columnName!=table.key>
<#noparse>#</#noparse>{${t.columnName}}<#if t_has_next >,</#if></#if>
</#list>)")
    int insert${table.name?cap_first}(${table.name?cap_first} ${table.name?
lower_case});

    /**
     * 根据id修改数据
     * 注意这里要根据实际修改一下
     */
    @Update("update ${table.name} set ${table.columnList[1].columnName} =
${table.columnList[1].columnName} where ${table.key}=${r"#{"}${table.key?
lower_case}}")
    int update${table.name?cap_first}ById(int ${table.key?lower_case});

    /**
     * 根据id删除数据
     */
    @Delete("delete from ${table.name} where ${table.key}=${r"#
{"}${table.key?lower_case}}")
    int delete${table.name?cap_first}ById(int ${table.key?lower_case});
```

```
}
```

（2022.9.10）

最终完整代码

项目结构

```
├─autoNewCode
│  ├─pojo
│  │      Column.java
│  │      DataBase.java
│  │      Settings.java
│  │      Table.java
│  │
│  ├─test
│  │      autoNewCode.java
│  │      dbTset.java
│  │
│  └─utils
│          ConvertUtils.java
│          DataBaseUtils.java
│          FileUtils.java
│          Generator.java
│          GeneratorFacade.java
│          PropertiesUtils.java
```

代码

pojo层
Column.java

```
package com.example.springb_protect.autoNewCode.pojo;

import lombok.Data;

@Data       //使用这个注解可以省去代码中大量的get()、 set()、 toString()等方法;
public class Column {
    //列名称
```

```
    private String columnName;
    //处理后的列名称
    private String columnName2;
    //列类型
    private String columnType;
    //列在数据库中的类型
    private String columnDbType;
    //本工程暂不处理备注和主键
    //列备注id
    private String columnComment;
    //是否是主键
    private String columnKey;
}
```

DataBase.java

```
package com.example.springb_protect.autoNewCode.pojo;

import lombok.Data;

@Data       //使用这个注解可以省去代码中大量的get()、 set()、 toString()等方法;
public class DataBase {
    private static String mysqlUrl = "jdbc:mysql://[ip]:[port]/[db]?
useUnicode=true&characterEncoding=utf-8&serverTimezone=UTC";
    private static String oracleUrl = "jdbc:oracle:thin:@[ip]:[port]:[db]";
    private String dbType;//数据库类型
    private String userName;
    private String passWord;
    private String driver;
    private String url;

    public DataBase() {
    }

    public DataBase(String dbType) {
        this(dbType, "127.0.0.1", "3306", "");
    }

    public DataBase(String dbType, String db) {
        this(dbType, "127.0.0.1", "3306", db);
    }

    /**
     * @param dbType 数据库类型 mysql/oracle
     * @param ip     ip
     * @param port   3306
```

```
     * @param db      数据库名称 test
     */    public DataBase(String dbType, String ip, String port, String db)
{
        this.dbType = dbType;
        if ("MYSQL".equals(dbType.toUpperCase())) {
            this.driver = "com.mysql.cj.jdbc.Driver";
            this.url = mysqlUrl.replace("[ip]", ip).replace("[port]",
port).replace("[db]", db);
        } else {
            this.driver = "oracle.jdbc.driver.OracleDriver";
            this.url = oracleUrl.replace("[ip]", ip).replace("[port]",
port).replace("[db]", db);
        }
    }
}
```

Settings.java

```
package com.example.springb_protect.autoNewCode.pojo;

import lombok.Data;

@Data
public class Settings {
    private String project = "example";
    private String pPackage = "com.example.demo";
    private String projectComment;
    private String author = "Orall";
    private String path1 = "com";
    private String path2 = "example";
    private String path3 = "demo";
    private String pathAll;
    //controller层的返回值，请求成功或者失败对应的函数名（函数默认一个参数）
    private String returnValue = "Map<String, Object>";
    private String successFunction = "StatusCode.success";
    private String failFunction = "StatusCode.error";
    //主键类型
    private String keyType = "Integer";
    //文件名称
    private String daoName = "dao";
    private String controllerName = "controller";
    private String pojoName = "pojo";
    private String serviceName = "service";
    private String serviceImplName = "serviceImpl";
```

```
}
```

Table.java

```
package com.example.springb_protect.autoNewCode.pojo;

import lombok.Data;

import java.util.List;

@Data       //使用这个注解可以省去代码中大量的get()、 set()、 toString()等方法;
public class Table {
    //表名称
    private String name;
    //处理后的表名称
    private String name2;
    //介绍
    private String comment;
    //主键列
    private String key;
    private List<Column> columnList;
}
```

utils层
ConvertUtils.java

```
package com.example.springb_protect.autoNewCode.utils;

import java.lang.reflect.Field;
import java.lang.reflect.Method;
import java.util.*;

public class ConvertUtils {
    /**
     * 将一个类查询方式加入map（属性值为int型时，0时不加入，
     * 属性值为String型或Long时为null和“”不加入）
     *
     */    public static Map<String, Object> setConditionMap(Object obj){
        Map<String, Object> map = new HashMap<String, Object>();
        if(obj==null){
            return null;
        }
        Field[] fields = obj.getClass().getDeclaredFields();
```

```
        for(Field field : fields){
            String fieldName =  field.getName();
            if(getValueByFieldName(fieldName,obj)!=null){
                map.put(fieldName,  getValueByFieldName(fieldName,obj));
            }

        }
        return map;

    }
    /**
     * 根据属性名获取该类此属性的值
     * @param fieldName
     * @param object
     * @return
     */
    private static Object getValueByFieldName(String fieldName,Object object)
{
        String firstLetter=fieldName.substring(0,1).toUpperCase();
        String getter = "get"+firstLetter+fieldName.substring(1);
        try {
            Method method = object.getClass().getMethod(getter, new Class[]
{});
            Object value = method.invoke(object, new Object[] {});
            return value;
        } catch (Exception e) {
            return null;
        }
    }
}
```

DataBaseUtils.java

```
package com.example.springb_protect.autoNewCode.utils;

import com.example.springb_protect.autoNewCode.pojo.Column;
import com.example.springb_protect.autoNewCode.pojo.DataBase;
import com.example.springb_protect.autoNewCode.pojo.Table;
import org.junit.Test;
import javax.servlet.http.HttpSession;
import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.List;
```

```
/**
 * 方法介绍：
 * 1，获取数据库连接
 * 2，获取数据库列表
 * 3，获取数据库中的所有表和字段并构造实体类
 * 4，根据表名的截取操作生成类名
 */
public class DataBaseUtils {
    /**
     * 获取数据库连接
     * @param db
     * @return
     * @throws Exception
     */    public static Connection getConnection(DataBase db) throws
Exception {
        //获取连接
        //注册驱动
        Class.forName(db.getDriver());
        Connection connection = DriverManager.getConnection(db.getUrl(),
db.getUserName(), db.getPassWord());
        return connection;
    }

    /**
     * 获取数据库列表
     * @param db
     * @return
     * @throws Exception
     */    public static List<String> getShemas(DataBase db) throws Exception
{
        Connection connection = getConnection(db);
        //获取元数据
        DatabaseMetaData metaData = connection.getMetaData();
        //获取所有数据库列表
        ResultSet resultSet = metaData.getCatalogs();
        List<String> list = new ArrayList<>();
        while (resultSet.next()) {
            list.add(resultSet.getString(1));
        }
        resultSet.close();
        connection.close();
        return list;
    }

    /**
     * 获取数据库中的所有表和字段并构造实体类（相当于一键生成数据库中所有表的增删改查代
码）
```

```
     * @param db
     * @param dbss
     * @return
     * @throws Exception
     */    public static List<Table> getDbInfo(DataBase db, String dbss)
throws Exception {
        //获取连接
        Connection connection = getConnection(db);
        //获取元数据
        DatabaseMetaData metaData = connection.getMetaData();
        List<Table> list = new ArrayList<>();
        //获取当前数据库的所有表
        //String dbss = (String) session.getAttribute("db");
        ResultSet tables = metaData.getTables(dbss, null, null, new String[]
{"TABLE"});
        while (tables.next()) {
            //表名
            String table_name = tables.getString("TABLE_NAME");
            //构造生成对应实体类的类名
            String className = removePrefix(table_name);
            //主键
            ResultSet primaryKeys = metaData.getPrimaryKeys(null, null,
table_name);
            //对主键遍历的原因（或许一张表有多个主键）
            String keys = "";
            if (primaryKeys.next()) {
                String keyName = primaryKeys.getString("COLUMN_NAME");
                keys += keyName;
            }
            Table tab = new Table();
            tab.setName(table_name);
            tab.setName2(className);
            tab.setKey(keys);
            //处理表中的所有字段
            ResultSet columns = metaData.getColumns(dbss, null, table_name,
null);
            List<Column> cols = new ArrayList<>();
            while (columns.next()) {
                Column column = new Column();
                //列名称
                String column_name = columns.getString("COLUMN_NAME");
                //java实体的属性名
                String attName = column_name;
                //java类型和数据库类型
                String type_name = columns.getString("TYPE_NAME");
                String javaType = PropertiesUtils.customMap.get(type_name);
                column.setColumnName(column_name);
                column.setColumnName2(attName);
```

```
                column.setColumnDbType(type_name);
                column.setColumnType(javaType);
                cols.add(column);
            }
            tab.setColumnList(cols);
            list.add(tab);
            //关闭连接，释放资源
            columns.close();
            primaryKeys.close();
        }
        tables.close();
        connection.close();
        return list;
    }

    /**
     * 根据表名的截取操作生成类名
     * @param tableName
     * @return
     */
    public static String removePrefix(String tableName) {
        //从自定义的配置文件中拿到前缀的配置
        String prefixes =
PropertiesUtils.customMap.get("tableRemovePrefixes");
        //这里就不字符串处理了，直接把表名当类名用了
        String replace = tableName;
        return replace;
    }
}
```

FileUtils.java

```
package com.example.springb_protect.autoNewCode.utils;

import freemarker.template.utility.StringUtil;
import org.springframework.util.StreamUtils;
import org.springframework.util.StringUtils;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;

/**
 * 文件处理工具类
```

```
 * 1.查询整个目录的文件夹
 * 2.递归获取某个目录下的所有文件夹
 */

public class FileUtils {
    /**
     * 得到相对路径
     * @param baseDir
     * @param file
     * @return
     */
    public static String getRelativePath(File baseDir, File file) {
        if (baseDir.equals(file)) {
            return "";
        }
        if (baseDir.getParentFile() == null) {
            return
file.getAbsolutePath().substring(baseDir.getAbsolutePath().length());
        } else {
            return
file.getAbsolutePath().substring(baseDir.getAbsolutePath().length() + 1);
        }
    }

    /**
     * 查询整个目录下的所有文件
     * @param dir
     * @return
     * @throws IOException
     */    public static List<File> searchAllFile(File dir) throws
IOException {
        ArrayList arrayList = new ArrayList();
        searchFiles(dir, arrayList);
        return arrayList;
    }

    /**
     * 递归获取某个目录下的所有文件
     * @param dir
     * @param collector
     */    public static void searchFiles(File dir, List<File> collector) {
        if (dir.isDirectory()) {
            File[] files = dir.listFiles();
            for (int i = 0; i < files.length; i++) {
                searchFiles(files[i], collector);
            }
        } else {
            collector.add(dir);
```

```
        }
    }

    /**
     * 递归获取某个目录下的所有文件并重命名
     * @param dir
     */    public static void searchFilesAndRename(File dir) {
        if (dir.isDirectory()) {
            File[] files = dir.listFiles();
            for (int i = 0; i < files.length; i++) {
                searchFilesAndRename(files[i]);
            }
        } else {
            String name = dir.getName();
            dir.renameTo(new
File(name.substring(0,1).toUpperCase()+name.substring(1)));
        }
    }

    /**
     * 创建文件
     * @param dir
     * @param file
     * @return
     */
    public static File mkdir(String dir, String file) {
        if (dir == null) {
            throw new IllegalArgumentException("文件夹不许为空");
        }
        File result = new File(dir, file);
        if (result.getParentFile() != null) {
            result.getParentFile().mkdirs();
        }
        return result;
    }
}
```

Generator.java

```
package com.example.springb_protect.autoNewCode.utils;

import freemarker.cache.FileTemplateLoader;
import freemarker.template.Configuration;
import freemarker.template.Template;

import java.io.File;
```

```
import java.io.FileWriter;
import java.io.StringReader;
import java.io.StringWriter;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

//代码生成器的核心处理类，使用FreeMarker完成文件生成，数据模型+模板
//数据：数据模型 模板的位置  生成文件的路径
public class Generator {
    private String templatePath;//模板路径
    private String outPath;      //代码生成路径
    private Configuration cfg;

    public Generator(String templatePath, String outPath) throws Exception {
        this.outPath = outPath;
        //实例化Configuration对象
        cfg = new Configuration();
        //指定模板加载器
        //在代码中动态加载jar、资源文件的时候，首先应该是使用
Thread.currentThread().getContextClassLoader()。
        // 如果你使用Test.class.getClassLoader()，可能会导致和当前线程所运行的类加载
器不一致（因为Java天生的多线程）
        String templates =
Thread.currentThread().getContextClassLoader().getResource("").getPath()+"\\
模板\\";
        //
        this.templatePath = templates;
        FileTemplateLoader fileTemplateLoader = new FileTemplateLoader(new
File(templates));
        cfg.setTemplateLoader(fileTemplateLoader);
    }

    /**
     * 代码生成
     * 1，扫描模板路径下的所有模板
     * 2，对每个模板进行文件生成（数据模板）
     * 3，参数：数据模板
     */
    public void scanAndGenerator(Map<String, Object> dataModel) throws
Exception {
        //根据模板路径找到此路径下的所有模板文件
        List<File> fileList = FileUtils.searchAllFile(new
File(templatePath));
        System.out.println(fileList);
        //对每个模板进行文件生成
        for (File file : fileList) {
            //参数1：数据模型  参数2：文件模板
```

```
            System.out.println(file);
            excuteGenerator(dataModel, file);
        }
    }

    //对模板进行文件生成
    //参数1：数据模型  参数2：文件模板
    private void excuteGenerator(Map<String, Object> dataModel, File file)
throws Exception {
        //1，文件路径处理
        /**
         * file:D:/模板存在的文件
夹/${path1}/${path2}/${path3}/${classname}.java  绝对路径
         * replace的目的：得到文件名，文件名之前的路径都不要了，只留包名之后的内容
         */
        //  得到模板文件的这样的路径
${path1}/${path2}/${path3}/${className}.java        String one =
file.getAbsolutePath();
        int i = one.indexOf("$");
        String templateFileName = one.substring(i);
        //把${path1}/${path2}/${path3}/${className}.java 替换成
com/ftx/demoUser.java (数据模型中的内容)
        String outFileName = processString(templateFileName, dataModel);
        //System.out.println(outFileName);
        int index = outFileName.lastIndexOf('\\');
        //将java文件首字母大写
        outFileName = outFileName.substring(0,index+1) +
outFileName.substring(index+1,index+2).toUpperCase() +
outFileName.substring(index+2);
        //System.out.println(outFileName);

        //2，读取文件模板
        //上面把模板整个文件夹都加载到了模板加载器，所以这里拿模板只需要传入该文件夹下的
文件名即可
        Template template = cfg.getTemplate(templateFileName);//相对路径
${path1}/${path2}/${path3}/${classname}.java
template.setOutputEncoding("utf-8");//指定生成文件的字符集编码
        //3，创建文件
        File file1 = FileUtils.mkdir(this.outPath, outFileName);
        //4，模板处理（文件生成）
        FileWriter fileWriter = new FileWriter(file1);
        System.err.println(dataModel);
        template.process(dataModel, fileWriter);
        fileWriter.close();
    }

    /**
     * 把${path1}/${path2}/${path3}/${className}.java 替换成
```

```
com/ttx/demoUser.java (数据模型中的内容)
     * FreeMarker的字符串模板 替换
     */
    public String processString(String templateString, Map dataModel) throws
Exception {
        StringWriter stringWriter = new StringWriter();
        Template template = new Template("ts", new
StringReader(templateString), cfg);
        template.process(dataModel, stringWriter);
        return stringWriter.toString();
    }

    /**
     * 测试代码生成主类scanAndGenerator是否管用
     */
    public static void main(String[] args) throws Exception {
        String templatePath = "D:\\工作\\学习资料\\FreeMarker\\模板";
        String outPath = "D:\\工作\\学习资料\\FreeMarker\\生成路径";
        Generator generator = new Generator(templatePath, outPath);
        Map<String, Object> dataModel = new HashMap<>();
        dataModel.put("username", "张三");
        generator.scanAndGenerator(dataModel);
    }
}
```

GeneratorFacade.java

```
package com.example.springb_protect.autoNewCode.utils;


import com.example.springb_protect.autoNewCode.pojo.DataBase;
import com.example.springb_protect.autoNewCode.pojo.Settings;
import com.example.springb_protect.autoNewCode.pojo.Table;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * 准备数据模型，调用核心处理类Generator类完成代码生成工作
 */
public class GeneratorFacade {
    //模板位置
    private String templatePath;
    //代码生成路径
    private String outPath;
```

```
    //工程配置对象
    private Settings settings;
    //数据库对象
    private DataBase db;
    private Generator generator;

    public GeneratorFacade(String templatePath, String outPath, Settings
settings, DataBase db) throws Exception {
        this.templatePath = templatePath;
        this.outPath = outPath;
        this.settings = settings;
        this.db = db;
        generator = new Generator(templatePath, outPath);
    }

    /**
     * 准备数据模型
     * 调用核心处理类完成代码生成工作
     */
    public boolean generatorByDataBase(String dbss) throws Exception {
        List<Table> tables = DataBaseUtils.getDbInfo(db, dbss);
        for (Table table : tables) {
            //对每个table进行代码生成
            Map<String, Object> dataModel = getDataModel(table);
            /**
             * 得到的数据模型如下
             * {NUMBER=Long, CHAR=String, project=test, BIGINT=Long,
TEXT=String, className=r, VARCHAR2=String,
             * INT=Integer, NVARCHAR2=String, DATE=java.util.Date,
DATETIME=java.util.Date, path1=com, path2=ftx,             * path3=demo,
pPackage=com.ftx.demo, VARCHAR=String, testKey=testValue, DOUBLE=Double,
tableRemovePrefixes="tb_,co_",             * table=Table{name='user',
name2='r', comment='null', key='userid,userno,id,Host,User,id,',
* columnList=[Column{columnName='id', columnName2='id', columnType='Integer',
columnDbType='INT', columnComment='null',             * columnKey='null'},
Column{columnName='account', columnName2='account', columnType='String',
columnDbType='VARCHAR',             * columnComment='null',
columnKey='null'}, Column{columnName='password', columnName2='password',
columnType='String',             * columnDbType='VARCHAR',
columnComment='null', columnKey='null'}, Column{columnName='islogin',
columnName2='islogin',             * columnType='Integer',
columnDbType='INT', columnComment='null', columnKey='null'}]}}             */
//调用代码生成方法，把数据模型传过去，进行生成
            generator.scanAndGenerator(dataModel);
        }
        return true;
    }
```

```
    /**
     * 根据table对象获取数据模型
     *
     * @param table
     * @return
     */
    private Map<String, Object> getDataModel(Table table) {
        //Map<String, Object> map = new HashMap<>();
        Map<String, Object> map = ConvertUtils.setConditionMap(settings);
        //自定义配置
        map.putAll(PropertiesUtils.customMap);
        //元数据
        map.put("table", table);
        //settings
        map.put("project", this.settings.getProject());
        map.put("pPackage", this.settings.getPPackage());
        map.put("path1", this.settings.getPath1());
        map.put("path2", this.settings.getPath2());
        map.put("path3", this.settings.getPath3());
        //类名
        map.put("className", table.getName2());
        return map;
    }
}
```

PropertiesUtils.java

```
package com.example.springb_protect.autoNewCode.utils;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Properties;

/**
 * 此工具类说明：静态代码块预加载，将自定义的配置文件properties的内容全部加载到
 * customMap中，然后在其他类中调用此类获取customMa中的键值对（键值对就是字都
 * 应以配置文件中所配置的内容）
 */
public class PropertiesUtils {
    public static Map<String, String> customMap = new HashMap<>();

    //静态块，预加载，将自定义的配置文件properties的内容全部加载到customMap中
```

```
    static {
        File dir = new File("properties");
        try {
            List<File> files = FileUtils.searchAllFile(new
File(dir.getAbsolutePath()));
            for (File file : files) {
                if (file.getName().endsWith("properties")) {
                    Properties prop = new Properties();
                    prop.load(new FileInputStream(file));
                    customMap.putAll((Map) prop);
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    //测试预加载是否成功（看是否打印出了properties配置文件的key和value）
    public static void main(String[] args) {
        for (String key : customMap.keySet()) {
            System.out.println(key + "---" + customMap.get(key));
        }
    }
}
```

和src同层下的 properties/typeConverter.properties

```
# sql类型和java类型的替换规则
BIT=boolean    LONGVARCHAR=String
CHAR=String
VARCHAR=String
DATE=java.sql.Date TIME=java.sql.Time TIMESTAMP=java.sql.Timestamp
BIGINT=Long
LONGTEXT=String
TEXT=String
INT=Integer

# table的前缀或者后缀
tableRemovePrefixes="tb_,co_"
```

模板

controller.ftl

```
package ${pPackage}.${controllerName};

import ${pPackage}.${pojoName}.${table.name?cap_first};
import ${pPackage}.${serviceName}.${table.name?cap_first}Service;
import io.swagger.annotations.ApiOperation;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.bind.annotation.*;
import javax.validation.Valid;
import java.util.Map;
import java.util.List;

/**
* @author ${author}
* @version 1.0
* @description ${table.name?cap_first}的Controller层
* @data ${.now?date}
*/
@RestController
@RequestMapping(value = "/api/${table.name?lower_case}")
public class ${table.name?cap_first}Controller {
    @Autowired
    ${table.name?cap_first}Service ${table.name?lower_case}Service;

    /**
    * @param ${table.name?lower_case} 添加的实体类
    * @return ${returnValue} 自定义响应体
    * @description 添加数据
    * @author ${author}
    * @data ${.now?date}
    */
    @PostMapping("/insert")
    @ApiOperation(value = "添加数据")
    public ${returnValue} insert${table.name?cap_first}(@Valid ${table.name?
cap_first} ${table.name?lower_case}) {
        try {
            if( ${table.name?lower_case}Service.insert${table.name?cap_first}
(${table.name?lower_case}) >= 1 ){
```

```
                return ${successFunction}("添加成功");
            }else{
                return ${successFunction}("添加失败");
            }
        } catch (Exception e) {
            e.printStackTrace();
            return ${failFunction}("服务器内部错误：" + e.toString());
        }
    }


    /**
     * @param ${table.key?lower_case} 主键id
     * @return ${returnValue} 自定义响应体
     * @description 根据id获取单条数据
     * @author ${author}
     * @data ${.now?date}
     */
    @GetMapping("/get")
    @ApiOperation(value = "根据id获取单条数据")
    public ${returnValue} get${table.name?cap_first}By${table.key?cap_first}
(@RequestParam("${table.key?lower_case}") ${keyType} ${table.key?lower_case})
{
        try {
            return ${successFunction}(${table.name?
lower_case}Service.get${table.name?cap_first}By${table.key?cap_first}
(${table.key?lower_case}));
        } catch (Exception e) {
            e.printStackTrace();
            return ${failFunction}(3001, "服务器内部错误：" + e.toString());
        }
    }

    /**
     * @param page 查询的页数
     * @return ${returnValue} 自定义响应体
     * @description 分页查询数据（备注：limit默认为10）
     * @author ${author}
     * @data ${.now?date}
     */
    @GetMapping("/list/page")
    @ApiOperation(value = "分页查询数据")
    public ${returnValue} list${table.name?
cap_first}ByPage(@RequestParam("page") int page) {
        try {
            //limit默认为10
            return ${successFunction}(${table.name?
lower_case}Service.list${table.name?cap_first}ByPage(page,10));
```

```
        } catch (Exception e) {
            e.printStackTrace();
            return ${failFunction}("服务器内部错误: " + e.toString());
        }
    }

    /**
     * @param ${table.name?lower_case} 需要修改的实体类
     * @return ${returnValue} 自定义响应体
     * @description 根据id修改数据
     * @author ${author}
     * @data ${.now?date}
     */
    @PutMapping("/update")
    @ApiOperation(value = "根据id修改数据")
    public ${returnValue} update${table.name?cap_first}By${table.key?
cap_first}(@Valid ${table.name?cap_first} ${table.name?lower_case}) {
        try {
            if( ${table.name?lower_case}Service.update${table.name?
cap_first}By${table.key?cap_first}(${table.name?lower_case}) >= 1 ){
                return ${successFunction}("修改成功");
            }else{
                return ${successFunction}("修改失败");
            }
        } catch (Exception e) {
            e.printStackTrace();
            return ${failFunction}("服务器内部错误: " + e.toString());
        }
    }

    /**
     * @return ${returnValue} 自定义响应体
     * @description 查询所有数据（备注: 不常用）
     * @author ${author}
     * @data ${.now?date}
     */
    @GetMapping("/list")
    @ApiOperation(value = "查询所有数据")
    public ${returnValue} list${table.name?cap_first}() {
        try {
            return ${successFunction}(${table.name?
lower_case}Service.list${table.name?cap_first}());
        } catch (Exception e) {
            e.printStackTrace();
            return ${failFunction}("服务器内部错误: " + e.toString());
        }
    }
```

```
    /**
     * @param ${table.key?lower_case} 主键id
     * @return ${returnValue} 自定义响应体
     * @description 根据id删除数据
     * @author ${author}
     * @data ${.now?date}
     */
    @DeleteMapping("/delete")
    @ApiOperation(value = "删除数据")
    public ${returnValue} delete${table.name?cap_first}By${table.key?
cap_first}(@RequestParam("${table.key?lower_case}") ${keyType} ${table.key?
lower_case}) {
        try {
            if( ${table.name?lower_case}Service.delete${table.name?
cap_first}By${table.key?cap_first}(${table.key?lower_case}) >= 1 ){
                return ${successFunction}("删除成功");
            }else{
                return ${successFunction}("删除失败");
            }
        } catch (Exception e) {
            e.printStackTrace();
            return ${failFunction}("服务器内部错误：" + e.toString());
        }
    }

}
```

dao.ftl

```
package ${pPackage}.${daoName};

import ${pPackage}.${pojoName}.${table.name?cap_first};
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;
import org.apache.ibatis.annotations.Delete;
import org.apache.ibatis.annotations.Update;
import java.util.List;

/**
 * @author ${author}
 * @version 1.0
 * @description ${table.name?cap_first}的Mapper类
 * @date ${.now?date}
 */
```

```
@Mapper
public interface ${table.name?cap_first}Mapper {
    /**
     * @return 以列表形式返回${table.name?cap_first}实体类
     * @description 查询所有数据
     * @author ${author}
     * @data ${.now?date}
     */
    @Select("select * from ${table.name}")
    List<${table.name?cap_first}> list${table.name?cap_first}();

    /**
     * @param ${table.key?lower_case} 主键id
     * @return 返回${table.name?cap_first}实体类
     * @description 根据id获取单条数据（备注：这里的*换成对应想要获取的数据）
     * @author ${author}
     * @data ${.now?date}
     */
    @Select("select * from ${table.name} where ${table.key}=${r"#
{"}${table.key?lower_case}}")
    ${table.name?cap_first} get${table.name?cap_first}By${table.key?
cap_first}(${keyType} ${table.key?lower_case});

    /**
     * @param first 查询结果的索引值（默认从0开始）
     * @param second 查询结果返回的数量
     * @return 以列表形式返回${table.name?cap_first}实体类
     * @description 分页查询数据（备注：这里的*换成对应想要获取的数据）
     * @author ${author}
     * @data ${.now?date}
     */
    @Select("select * from ${table.name} limit <#noparse>#{first},#{second}
</#noparse>;")
    List<${table.name?cap_first}> list${table.name?cap_first}ByPage(int
first, int second);

    /**
     * @param ${table.name?lower_case} 插入的实体类
     * @return 新增数据的ID
     * @description 插入数据
     * @author ${author}
     * @data ${.now?date}
     */
    @Insert("insert into ${table.name}(<#list table.columnList as t><#if
t.columnName!=table.key>${t.columnName}<#if t_has_next >,</#if></#if>
</#list>) values(<#list table.columnList as t><#if t.columnName!=table.key>
<#noparse>#</#noparse>{${t.columnName}}<#if t_has_next >,</#if></#if>
</#list>)")
```

```
    int insert${table.name?cap_first}(${table.name?cap_first} ${table.name?
lower_case});

    /**
     * @param ${table.name?lower_case} 要修改的实体类
     * @return 修改数据的条数
     * @description 根据id修改数据（备注：这里要修改的内容要根据实际改一下）
     * @author ${author}
     * @data ${.now?date}
     */
    @Update("update ${table.name} set ${table.columnList[1].columnName} =
${table.columnList[1].columnName} where ${table.key}=${r"#{"}${table.key?
lower_case}}")
    int update${table.name?cap_first}By${table.key?cap_first}(${table.name?
cap_first} ${table.name?lower_case});

    /**
     * @param ${table.key?lower_case} 主键id
     * @return 删除数据的条数
     * @description 根据id删除数据
     * @author ${author}
     * @since ${.now?date}
     */
    @Delete("delete from ${table.name} where ${table.key}=${r"#
{"}${table.key?lower_case}}")
    int delete${table.name?cap_first}By${table.key?cap_first}(${keyType}
${table.key?lower_case});

}
```

pojo.ftl

```
package ${pPackage}.${pojoName};

import lombok.Data;
import lombok.AllArgsConstructor;
import lombok.NoArgsConstructor;
import com.fasterxml.jackson.annotation.JsonInclude;

/**
 * @author ${author}
 * @version 1.0
 * @description ${table.name?cap_first}的实体类
 * @date ${.now?date}
 */
@Data
```

```
@NoArgsConstructor
@JsonInclude(JsonInclude.Include.NON_NULL)
public class ${table.name?cap_first} {
<#list table.columnList as t>
    private ${t.columnType} ${t.columnName};
</#list>
}
```

service.ftl

```
package ${pPackage}.${serviceName};

import ${pPackage}.${pojoName}.${table.name?cap_first};
import java.util.List;

/**
 * @author ${author}
 * @version 1.0
 * @description: 用于${table.name?cap_first}的Service提供接口
 * @date ${.now?date}
 */
public interface ${table.name?cap_first}Service {

    /**
     * @description 查询所有数据
     * @author ${author}
     * @date ${.now?date}
     */
    List<${table.name?cap_first}> list${table.name?cap_first}();

    /**
     * @description 根据id获取单条数据
     * @author ${author}
     * @date ${.now?date}
     */
    ${table.name?cap_first} get${table.name?cap_first}By${table.key?
cap_first}(${keyType} ${table.key?lower_case});

    /**
     * @description 分页查询数据
     * @author ${author}
     * @date ${.now?date}
     */
    List<${table.name?cap_first}> list${table.name?cap_first}ByPage(int page,
int limit);
```

```
    /**
     * @description 插入数据
     * @author ${author}
     * @date ${.now?date}
     */
    int insert${table.name?cap_first}(${table.name?cap_first} ${table.name?
lower_case});

    /**
     * @description 根据id修改数据
     * @author ${author}
     * @date ${.now?date}
     */
    int update${table.name?cap_first}By${table.key?cap_first}(${table.name?
cap_first} ${table.name?lower_case});

    /**
     * @description 根据id删除数据
     * @author ${author}
     * @date ${.now?date}
     */
    int delete${table.name?cap_first}By${table.key?cap_first}(${keyType}
${table.key?lower_case});
}
```

serviceImpl.ftl

```
package ${pPackage}.${serviceName}.${serviceImplName};

import java.util.List;
import ${pPackage}.${pojoName}.${table.name?cap_first};
import ${pPackage}.${serviceName}.${table.name?cap_first}Service;
import ${pPackage}.${daoName}.${table.name?cap_first}Mapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

/**
* @author ${author}
* @version 1.0
* @description: 用于实现${table.name?cap_first}Service接口中的函数
* @date ${.now?date}
*/
@Service
public class ${table.name?cap_first}ServiceImpl implements ${table.name?
cap_first}Service {
    @Autowired
```

```
    ${table.name?cap_first}Mapper ${table.name?lower_case}Mapper;

    /**
     * @return 以列表形式返回实体类对象
     * @description 查询所有数据
     * @author ${author}
     * @date ${.now?date}
     */
    @Override
    public List<${table.name?cap_first}> list${table.name?cap_first}(){
        return ${table.name?lower_case}Mapper.list${table.name?cap_first}();
    }

    /**
     * @param ${table.key?lower_case} 主键id
     * @return 实体类对象
     * @description 根据id获取单条数据
     * @author ${author}
     * @date ${.now?date}
     */
    @Override
    public ${table.name?cap_first} get${table.name?cap_first}By${table.key?
cap_first}(${keyType} ${table.key?lower_case}){
        return ${table.name?lower_case}Mapper.get${table.name?
cap_first}By${table.key?cap_first}(${table.key?lower_case});
    }

    /**
     * @param page 页数
     * @param limit 每页限制数据量
     * @return 以列表形式返回实体类对象
     * @description 分页查询数据
     * @author ${author}
     * @date ${.now?date}
     */
    @Override
    public List<${table.name?cap_first}> list${table.name?
cap_first}ByPage(int page, int limit){
        int first = (page - 1) * limit;
        int second = limit;
        return ${table.name?lower_case}Mapper.list${table.name?
cap_first}ByPage(first,second);
    }

    /**
     * @param ${table.name?lower_case} 要添加的实体类
     * @return 大于等于1则插入成功
     * @description 插入数据
```

```
     * @author ${author}
     * @date ${.now?date}
     */
    @Override
    public int insert${table.name?cap_first}(${table.name?cap_first}
${table.name?lower_case}){
        return ${table.name?lower_case}Mapper.insert${table.name?cap_first}
(${table.name?lower_case});
    }

    /**
     * @param ${table.name?lower_case} 要修改的实体类
     * @return 大于等于1则修改成功
     * @description 根据id修改数据
     * @author ${author}
     * @date ${.now?date}
     */
    @Override
    public int update${table.name?cap_first}By${table.key?cap_first}
(${table.name?cap_first} ${table.name?lower_case}){
        return ${table.name?lower_case}Mapper.update${table.name?
cap_first}By${table.key?cap_first}(${table.name?lower_case});
    }

    /**
     * @param ${table.key?lower_case} 主键id
     * @return 大于等于1则删除成功
     * @description 根据id删除数据
     * @author ${author}
     * @date ${.now?date}
     */
    @Override
    public int delete${table.name?cap_first}By${table.key?cap_first}
(${keyType} ${table.key?lower_case}){
        return ${table.name?lower_case}Mapper.delete${table.name?
cap_first}By${table.key?cap_first}(${table.key?lower_case});
    }
}
```

test关键代码

```
package com.example.springb_protect.autoNewCode.test;

import com.example.springb_protect.autoNewCode.pojo.DataBase;
import com.example.springb_protect.autoNewCode.pojo.Settings;
import com.example.springb_protect.autoNewCode.pojo.Table;
```

```java
import com.example.springb_protect.autoNewCode.utils.*;
import freemarker.cache.FileTemplateLoader;
import freemarker.template.Configuration;
import freemarker.template.Template;
import freemarker.template.TemplateException;
import org.junit.Test;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class autoNewCode {
    @Test
    public void test4() throws Exception {
        String packagename = "com.ftx.demo";
        String projectEngName = "qaq";
        String ip = "服务器IP地址";
        String port = "3306";
        //数据库
        String db = "数据库库名";
        String username = "数据库用户名";
        String password = "数据库密码";
        String dbKind = "MYSQL";
        String fileUrl = "D:\\code\\springb_protect\\src\\main\\resources";
        Settings settings = new Settings();
        //包名(com.ftx.demo)
        settings.setPPackage(packagename);
        //split(".")无法分割字符串，必须加上\\
        String[] split = packagename.split("\\.");
        //com
        settings.setPath1(split[0]);
        //ftx
        settings.setPath2(split[1]);
        //demo
        settings.setPath3(split[2]);
        //项目名（没啥用）
        settings.setProject(projectEngName);
        //默认只支持mysql数据库吧，oracle暂时先不处理，先写死为mysql
        DataBase dbs = new DataBase("MYSQL", ip, port,db);
        dbs.setUserName(username);
        dbs.setPassWord(password);

        GeneratorFacade generatorFacade = new GeneratorFacade(dbKind,
```

```
fileUrl, settings, dbs);
        System.out.println(generatorFacade.toString());
        boolean b = generatorFacade.generatorByDataBase(db);
        if( b ){
            System.out.println("代码已生成");
        }else{
            System.out.println("代码生成失败");
        }

    }

}
```

生成

Setting中有部分默认的东西，有需要可以先改一下
项目结构

```
└模板
|   |           └${path1}
|   |               └${path2}
|   |                   └${path3}
|   |                       ├${controllerName}
|   |                       |       ${table.name}Controller.java
|   |                       |
|   |                       ├${daoName}
|   |                       |       ${table.name}Mapper.java
|   |                       |
|   |                       ├${pojoName}
|   |                       |       ${table.name}.java
|   |                       |
|   |                       └${serviceName}
|   |                           |   ${table.name}Service.java
|   |                           |
|   |                           └${serviceImplName}
|   |                                   ${table.name}ServiceImpl.java
```

将该模板文件放到test-class中然后运行test

后续

需要自己导入controller响应体的类以及导入
需要的依赖

```
<!-- ApiOperation的依赖 -->
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
</dependency>

<!-- Valid的依赖 -->
<dependency>
   <groupId>javax.validation</groupId>
   <artifactId>validation-api</artifactId>
</dependency>
```