用互联网思维扩展电商后台的CRUD功能

--图灵：楼兰

一、自定义实现MyBatis-Plus逆向工程

多数据源的问题解决了，接下来开始进行实际开发时，你会发现，最麻烦的一件事情就是要创建与数据库表对应的POJO了。这些没什么难度，但是繁琐的内容会占据大量的开发时间。比如一个PmsProducr对象，有三四十个属性。这就需要开发一个庞大的POJO对象。相反，上层的CRUD操作则相当简单。只需要继承MyBatis-plus框架提供的BaseMapper接口即可。

```
@DS("goods")
public interface PmsProductMapper extends BaseMapper<PmsProduct> {
}
```

标准的CRUD操作完全都不需要进行声明，直接就可以拿来用。只需要补充一些复杂的SQL操作即可。接下来当然是希望能够用程序快速自动的生成这些POJO类了，这样可以节省大量的开发时间。

关于如何生成POJO类，你当然可以使用MyBatis的逆向工程或者MyBatis-plus的逆向工程，这些网上有大量的资料，我们这里就不多做介绍。但是，你会不会有一种感觉，这些通用的逆向工程虽然优秀，但是却都太过复杂。他们为了工具的通用性，做了很多对我们没有用的封装。你有没有想过自己做一个简单使用的逆向工程出来呢？做一些这样的思考会让你对枯燥的CRUD工作产生一些不一样的想法。

其实你可以思考一下，需要根据数据库的表创建出对应的POJO类，需要哪些信息？ 其实要的信息并不多。表名、列名、列类型、主键信息。有这些就差不多了。而这些信息，其实都可以从最简单的JDBC操作中获取到。

```
public static void main(String[] args) throws Exception {
        //mysql
        Class.forName("com.mysql.cj.jdbc.Driver");
        Properties props = new Properties();
        props.put("useInformationSchema", "true"); //mysql获取表注释需要加上这个属性
        props.put("user", "root");
        props.put("password", "root");
        Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/genserver?
serverTimezone=GMT%2B8&characterEncoding=utf-8&autoReconnect=true",props);
        System.out.println("========映射表信息==============");
        DatabaseMetaData meta = con.getMetaData();
        ResultSet tables = meta.getTables("genserver", "%", "black_info", new String[]
{"TABLE"});
        while(tables.next()) {
```

```
            ResultSetMetaData metaData = tables.getMetaData();
            System.out.println(metaData.getColumnCount());
            for(int i = 1 ; i <= metaData.getColumnCount(); i ++) {
                System.out.println(metaData.getColumnName(i)+" ==>
"+tables.getString(metaData.getColumnName(i)));
            }

            System.out.println(tables.getString("TABLE_NAME")+" --->>>
"+tables.getString("REMARKS"));
        }
        System.out.println("========映射列信息==============");
        ResultSet columns = meta.getColumns("genserver", "%", "black_info", "%");
        while(columns.next()) {
            String columnName = columns.getString("COLUMN_NAME");
            String columnType = columns.getString("TYPE_NAME");
            int datasize = columns.getInt("COLUMN_SIZE");
            int digits = columns.getInt("DECIMAL_DIGITS");
            int nullable = columns.getInt("NULLABLE");
            String remarks = columns.getString("REMARKS");
            System.out.println(columnName+" "+columnType+" "+datasize+" "+digits+" "+
nullable+" "+remarks);
        }
        System.out.println("========映射主键信息==============");
        ResultSet primaryKeys = meta.getPrimaryKeys("genserver", "%", "black_info");
        while(primaryKeys.next()) {
            ResultSetMetaData metaData = primaryKeys.getMetaData();
            System.out.println(metaData.getColumnCount());
            for(int i = 1 ; i <= metaData.getColumnCount(); i ++) {
                System.out.println(metaData.getColumnName(i)+" ==>
"+primaryKeys.getString(metaData.getColumnName(i)));
            }
        }
    }
```

接下来如何将这些信息拼凑成一个POJO呢？你可以使用一个StringBuffer，一点点拼接出POJO的完整代码，再一次输出到文件当中，这没有问题。但是这样显然会比较麻烦，而且容易出错。

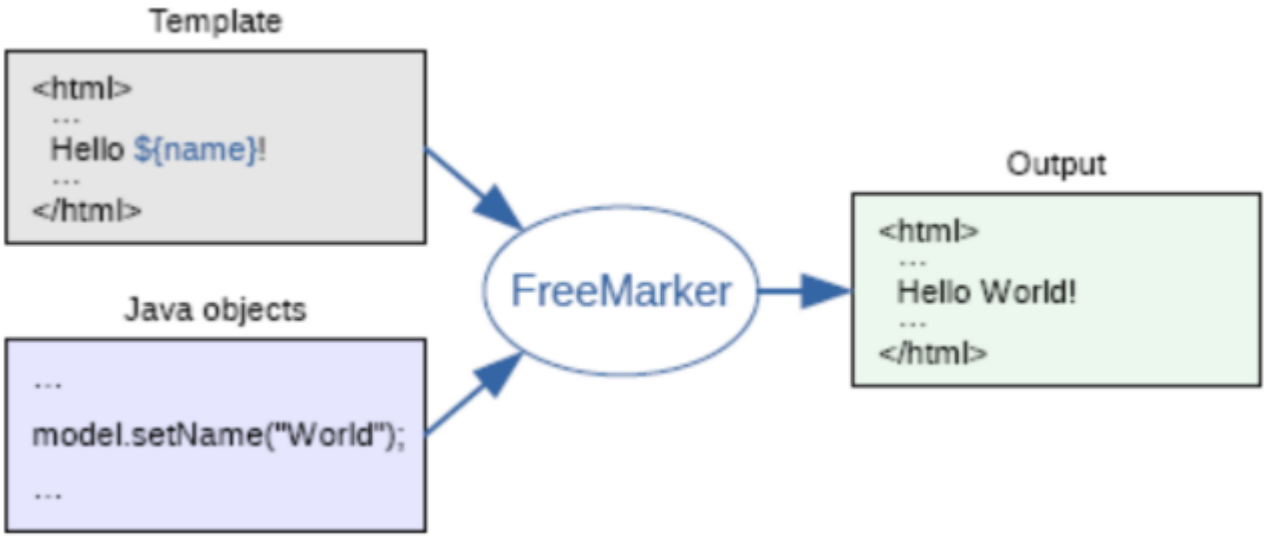
> MyBatis的逆向工程使用的就是这种方式。

二、使用Freemarker模板引擎实现一键开发模式

对于这种问题，其实可以用模版引擎来做。将代码中静态的部分写到模版当中，然后将动态部分交由模版生成。最为常用的模版引擎就是freemarker了。大部分场景下，freemarker通常是用来生成静态HTML页面的。比如在我们的电商场景中，就实现了对产品单品页的静态化功能。

> 使用静态化功能，需要你创建 %{user_home}\template\ftl\目录下放置report.ftl模版文件，同时需要提前创建 %{user_home}\template\report目录

freemarker是一个基于模版和数据输出文本的通用工具。只需要准备好动态的业务数据，以及基于FTL语言编写的模版文件，就可以快生成静态的文本。



如果你对freemarker不是很了解，可以从这个示例中快速理解freemarker模版引擎。这个引擎上手非常简单，对于有开发经验的你，肯定没什么问题。按照以下几个步骤就可以快速上手freemarker了。

1、引入maven依赖

```
<dependency>
    <groupId>org.freemarker</groupId>
    <artifactId>freemarker</artifactId>
    <version>2.3.23</version>
</dependency>
```

2、构建后台数据

```
public class FreemarkerTest {
    public static void main(String[] args) throws Exception {
        // 第一步：创建一个Configuration对象，直接new一个对象。构造方法的参数就是freemarker对于的版本号。
        Configuration configuration = new Configuration(Configuration.getVersion());
        // 第二步：设置模板文件所在的路径。
        configuration.setDirectoryForTemplateLoading(new File("D:\\ftl"));

        // 第三步：设置模板文件使用的字符集。一般就是utf-8. 注意版本。新版本不需要
```

```
//        configuration.setDefaultEncoding("UTF-8");
        // 第四步：加载一个模板，创建一个模板对象。
        Template template = configuration.getTemplate("test.ftl");
        // 第五步：创建一个模板使用的数据集，可以是pojo也可以是map。一般是Map。
        Map dataModel = new HashMap<>();
        //向数据集中添加数据
        dataModel.put("hello", "图灵学院电商VIP");
        // 第六步：创建一个Writer对象，一般创建一FileWriter对象，指定生成的文件名。
        Writer out = new FileWriter(new File("D:\\ftl\\out\\test.html"));
        // 第七步：调用模板对象的process方法输出文件。
        template.process(dataModel, out);
        // 第八步：关闭流。
        out.close();
    }
}
```

3、编写ftl模版文件

最简单模版文件就长这样

```
<h1>
${hello}
</h1>
```

执行完成后，就会将模版中的${hello}部分替换成 图灵学院电商VIP

一个ftl模版文件，是由少数几个动态标签加上其他静态的内容组成。动态标签包含以下几种：

- 普通参数

  例如${hello}

- list标签

```
<#list studentList as student>
    ${student.id}/${studnet.name}
</#list>
```

- if条件标签

```
<#if student_index % 2 == 0>
<#else>
</#if>
```

在if标签中，还可以进行简单的null值判断

```
<#if a??>
a不为空时。。
<#else>
a为空时###
</#if>
```

- 日期标签

```
当前日期: ${date?date}
当前时间:${date?time}
当前日期和时间:${date?datetime}
自定义日期格式:${date?string("yyyyMM/dd HH:mm: ss")}
```

- 包含标签

```
<#include "hello.ftl"/>
```

接下来如果你发挥一些想象，freemarker既然可以生成html文件，那是不是可以用来生成java源文件呢？显然是可以的。

> 示例参见EntityGeneratorTest.java

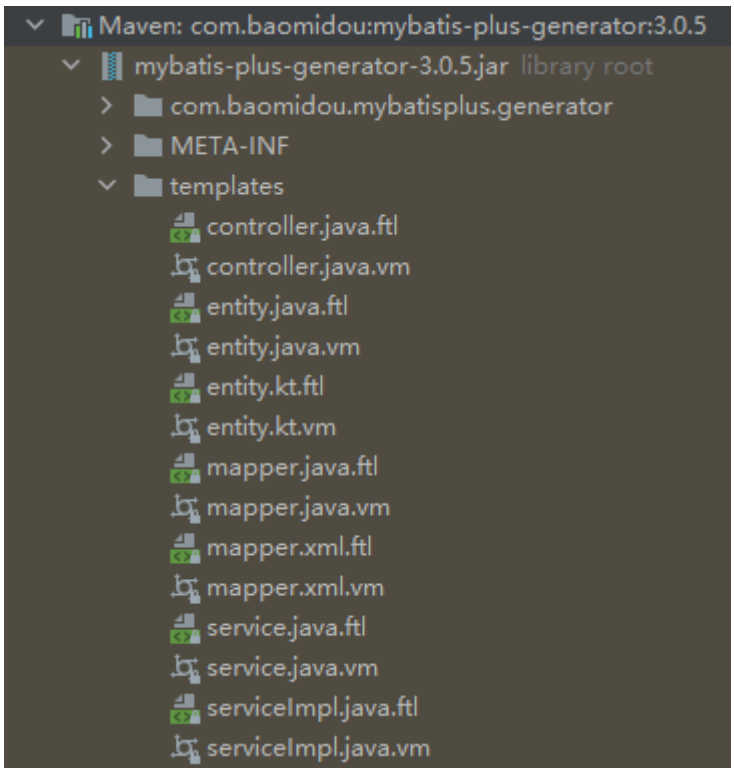
能够自己生成POJO了，那是不是可以把Service、Mapper、Controller等等这些重复性的代码一起生成呢？实际上，如果你有这种规范化的思想，你甚至可以将前台页面都一并生成了。减少大部分的复制粘贴的重复工作。

> 示例查看GenUI

最后，有了这个示例后，再来理解MyBatis-plus的逆向工程就非常容易了。 引入对应的依赖

```
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-generator</artifactId>
            <version>3.0.5</version>
        </dependency>
```

之后进入引入的jar包中，就能看到，MyBatis-plus的逆向工程也是使用freemarker和velocity提供的模版完成的逆向工程。



> vm是velocity框架的模版文件。velocity是和freemarker功能类似的一个模版引擎。

> 后续在设计秒杀场景时，也会使用freemarker自动生成前端商品单品页，实现动态页面静态化。

然后，发挥一下你自己的想象力，你还可以给这样简单的CRUD项目还能添加哪些与众不同的，实用的设计？比如，MyBatis-plus使用模板引擎生成了后端代码， 那么，对于一些长得差不多的数据管理页面，我们能不能也使用模板引擎，把前台页面到后端管理的全栈功能都一起开发出来呢?

> 参见GenUI示例工程。

三、结合CBoard报表工具实现拖拽式报表开发

当我们将前后端整合到一起之后，就可以继续发挥想象力，给普通的CRUD工作带来一些不一样的乐趣。

CBoard是一款开源的拖拽式报表开发工具，前端使用的是和我们项目一样的VUE技术。那么，可不可以做这样的设想，把CBoard中最后展现报表的前端页面挪用到我们的前端项目中，然后将后端请求通过Dubbo开放出来，这样我们就可以用很小的代码集成一套拖拽式的报表开发工具了。

> 参见GenUI示例工程。

有道云笔记链接：https://note.youdao.com/s/UQLSR7Ni