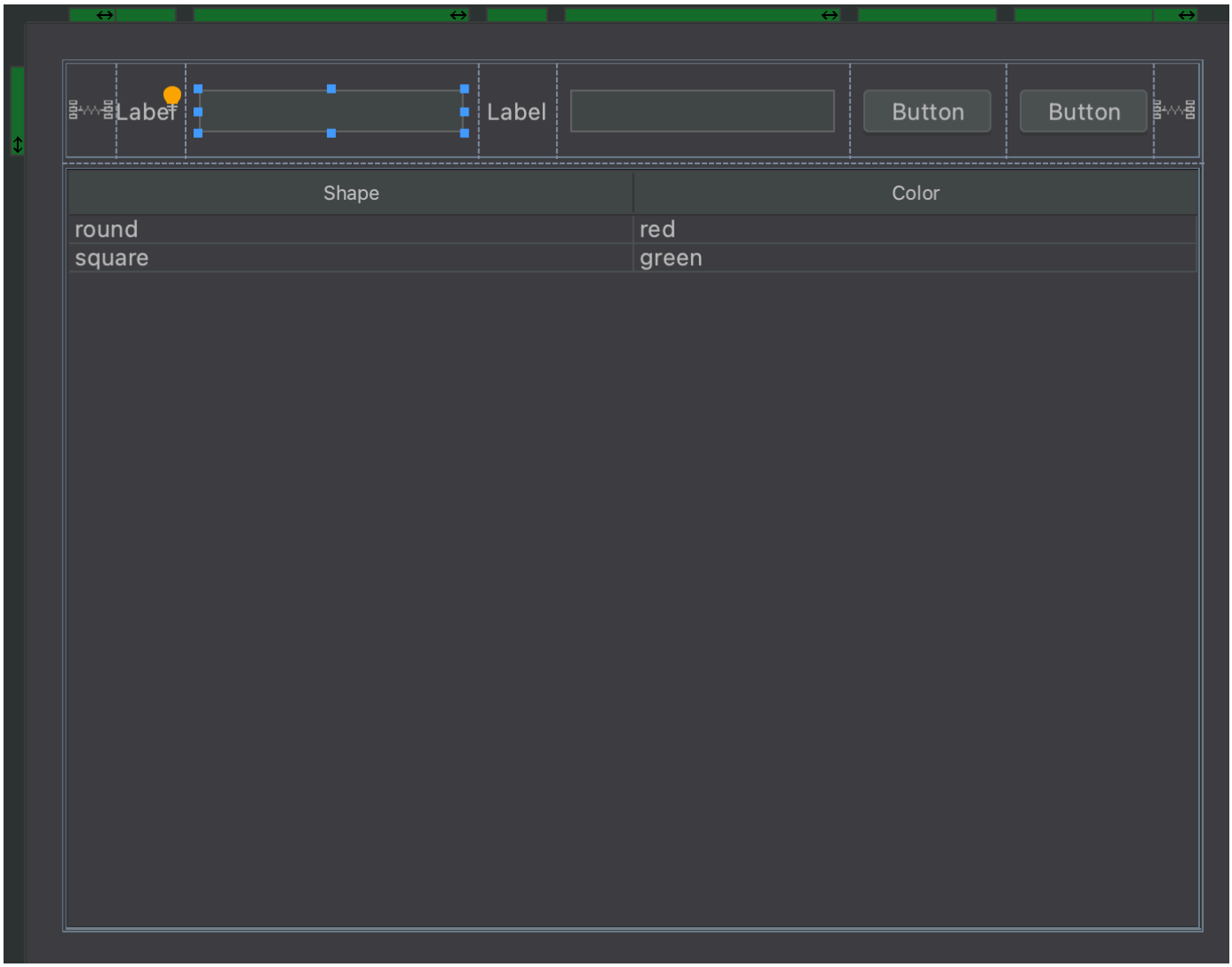
编码中遇到的问题及解决方法

1.UI界面的实现

UI界面实现过程

- 原参考插件的UI界面是通过一款界面设计插件实现的，需要付费。
- 其他方法实现UI界面，初步找到的有两种，一是利用Java swing进行编码实现；二是通过GUI Form，通过拖拽的方式进行实现。
- 一开始的界面开发选择了第二种，通过GUI Form进行实现，并且选取了GirdLayoutManager实现界面的布局。



遇到问题

在实现页面跳转时，通过GUI Form拖拽实现的界面一定需要通过调用mian函数显示，否则跳转后的界面为空，没有组件。

解决方法

- 以编码形式来实现UI界面。借鉴源码，使用MigLayout布局，复用原UI，并在此基础上进行修改。
- 创建handler包，来处理页面跳转，初始化界面时传入对应所需参数。
- 解决了跳转界面为空的问题，解决了跳转界面传参的问题。

```
{
    panel_ds.setLayout(new MigLayout(
            layoutConstraints: "hidemode 3",
            // columns
            colConstraints: "[fill]" +
                    "[571,fill]" +
                    "[fill]" +
                    "[fill]" +
                    "[30,fill]",
            // rows
            rowConstraints: "[]"));
    panel_ds.add(label_datasource,  constraints: "cell 0 0");
    panel_ds.add(comboBox_datasource,  constraints: "cell 1 0,dock center");

    //---- button_connect ----
    button_connect.setActionCommand("connect_datasource");
    button_connect.addActionListener(e -> button_connectActionPerformed(e));
    panel_ds.add(button_connect,  constraints: "cell 2 0");

    //---- button_add_ds ----
    button_add_ds.setActionCommand("add_datasource");
    button_add_ds.addActionListener(e -> button_add_dsActionPerformed(e));
    panel_ds.add(button_add_ds,  constraints: "cell 3 0");
    panel_ds.add(button_popup_menu,  constraints: "cell 4 0");
}
```

2.数据源保存、全局配置保存

- 添加的数据源以及更改之后的全局配置应该缓存起来，再次打开插件时可以直接读取缓存起来的内容。

实现方法：

- 通过MapDB缓存数据源记录以及全局的配置，MapDB相当于可持久化存储的Map。(MapDB是一个开源的，内嵌的Java数据引擎和集合框架。提供了**Map**,**Set**,**List**,**Queue**,**BitMap**，支持范围查询，数据过期，压缩，堆外存储的特性。)

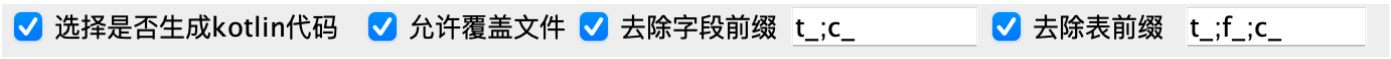
```
INFO  [AWT-EventQueue-0] - MapDB FilePath : /Users/zolitary/mybatisplusZoli.db
INFO  [AWT-EventQueue-0] - MapDB 初始化成功!
```

3.支持生成kotlin代码和Java代码

- 一开始的编码只实现了生成kotlin代码，freemarker模板文件和生成的代码文件都是kotlin格式。但同时也应该支持生成Java代码，并且要提供选择。

实现方法：

- 首先，在全局配置的UI界面上添加一个JCheckBox控件，如果勾选则生成kotlin代码，否则生成Java代码。
- 在全局配置类中新增一个boolean值 kotlin，表示是否是生成kotlin代码。其他有关的配置类则根据kotlin的值来对应生成。
- 模版文件中添加对应的Java模板。



4.去除字段前缀和去除表前缀

- 根据输入的前缀，既能实现去除表前缀的功能，也能实现去除字段前缀的功能。

实现方法：

- 通过定义工具类HumpUtils，实现去除前缀的功能，以及实现下划线转驼峰。

```
public static String lineToHump(String str, String removePrefixs, String addSuffix) {
    //前缀以分号分隔
    if (!StringUtils.isEmpty(removePrefixs)){
        String[] strings = removePrefixs.split( regex: ";");
        for (String prefix : strings) {
            if (!StringUtils.isEmpty(prefix) && str.startsWith(prefix.trim())){
                str = str.substring(prefix.trim().length());
            }
        }
    }
```

表前缀去除

```
entityName = StringUtils.capitalize(globalConfig.isTablePrefixEnabled() ?
        HumpUtils.lineToHump(tableName, globalConfig.getTablePrefix()) :
        HumpUtils.lineToHump(tableName));
```

字段前缀去除

```
// fieldName
String fieldName = globalConfig.isFieldPrefixEnabled() ?
        HumpUtils.lineToHump(columnInfo.getColumnName(), globalConfig.getFieldPrefix()) :
        HumpUtils.lineToHump(columnInfo.getColumnName());
```

5.插件的创建与开发

- mybatisplus代码自动生成模块开发完成后，需要将该模块的功能集成到插件上。
- 开始时，idea版本为(IntelliJ IDEA 2022.1.3 (Ultimate Edition))，创建插件工程后，插件的依赖和配置是通过gradle.kts文件来实现，没有找到对应实现模块依赖的方法。
- 下载了之前的idea，版本为(IntelliJ IDEA 2020.2.4 (Community Edition))，按照步骤创建工程后，直接通过模块依赖使用所需类，编译调试正常。

6.依赖与打包方式

遇到问题

- 插件打包后，idea安装本地打包好的插件，运行时报错，找不到mybatisplus代码自动生成模块中的类。

解决方法：

- 更改依赖方式，先通过maven将mybatisplus代码自动生成模块及其依赖的包打包成一个jar包，然后再在插件中引入该jar包。
- 插件编译调试正常，正常打包后，idea安装本地打包好的插件，可以正常运行和使用。换了另一个版本的idea也可以成功运行和使用。

依旧存在的问题

- 如果要更改或优化插件，那么对mybatisplus代码自动生成模块进行修改之后，又需要重新打包成jar包，然后插件再重新添加依赖。过程较为繁琐，不利于代码的修改和优化。(未解决)
- 表详情的字段名。(已解决)
- 默认值添加。(已解决)
- 生成java or kotlin 单选框 (已解决)
- 配置文件，unicode不可维护，需要转成中文，部分直接set到代码中 (已解决)