第12章 代码生成器原理分析

- 理解代码生成器的需求和实现思路
- 掌握freemaker的使用
- 理解数据库中的元数据
- 完成环境搭建工作

1 浅谈代码生成器

1.1 概述

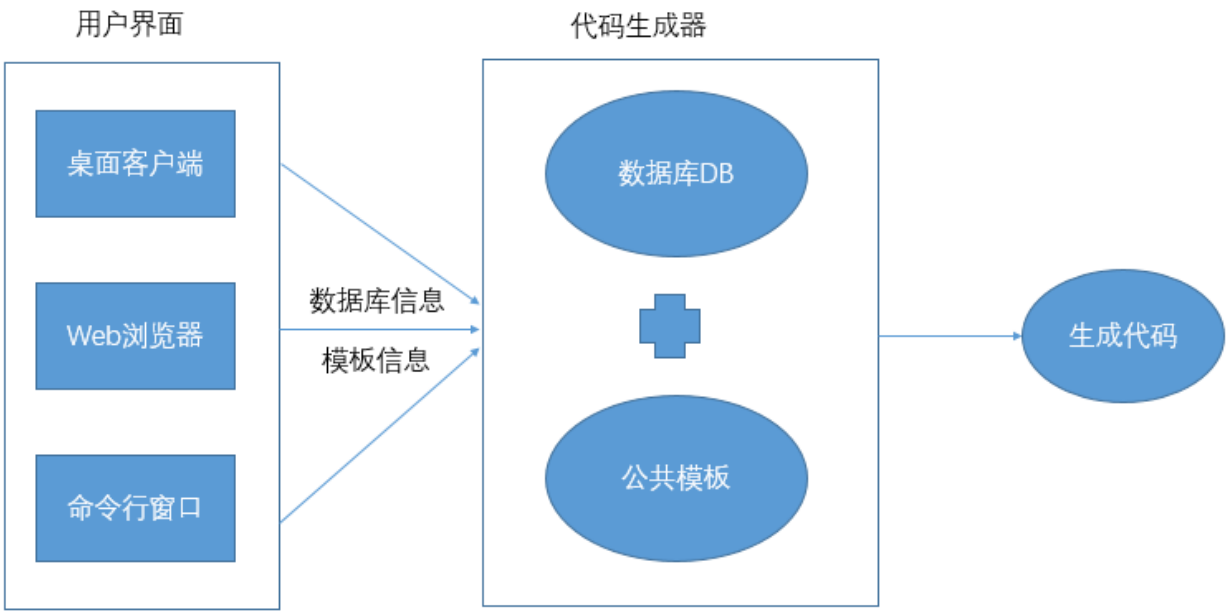
在项目开发过程中，关注点更多是在业务功能的开发及保证业务流程的正确性上，对于重复性的代码编写占据了程序员大量的时间和精力，而这些代码往往都是具有规律的。就如controller、service、serviceImpl、dao、daoImpl、model、jsp的结构，用户、角色、权限等等模块都有类似的结构。针对这部分代码，就可以使用代码生成器，让计算机自动帮我们生成代码，将我们的双手解脱出来，减小了手工的重复劳动。

传统的方式程序员进行模块开发步骤如下：

- 创建数据库表
- 根据表字段设计实体类
- 编写增删改查dao
- 根据业务写service层
- web层代码和前台页面

通常只需要知道了一个表的结构，增删改查的前后台页面的代码格式就是固定的，剩下的就是复杂的业务。而代码生成工具的目标就是自动生成那部分固定格式的增删改查的代码

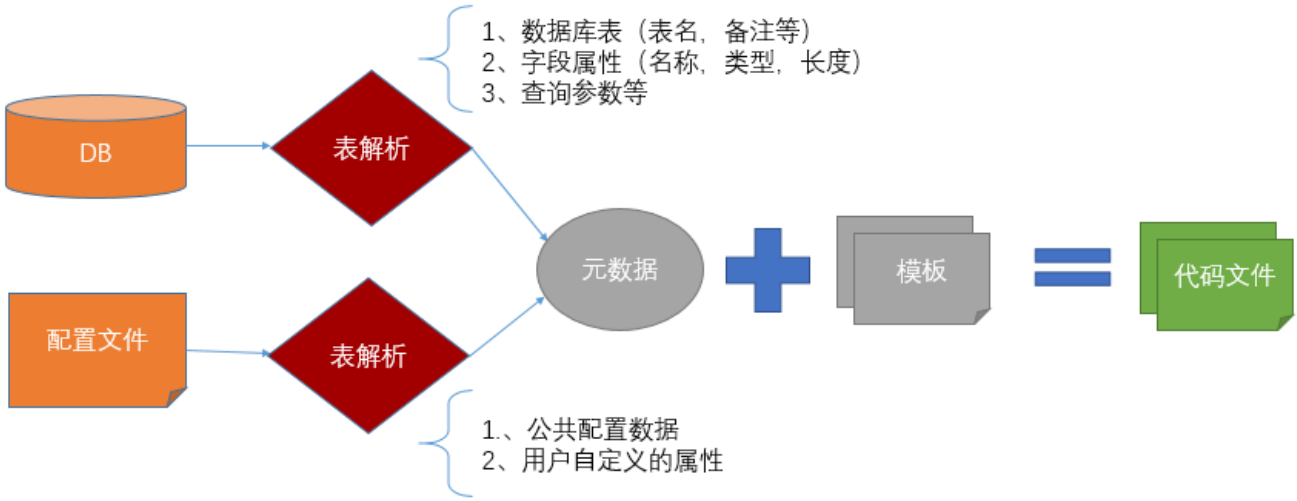
1.2 需求分析



再我们的代码生成器中就是根据公共的模板和数据库表中的信息自动的生成代码。

- 对于不借助代码生成工具的开发，程序员通常都是以一份已经写好的代码为基础进行代码Copy和修改，根据不同业务数据库表完善需求，可以将这份代码称之为公共的代码模板。
- 生成的代码和数据库表中信息息息相关，所以除了模板之外还需要数据库表信息作为数据填充模板内容

1.3 实现思路



代码生成器的实现有很多种，我们以从mysql数据库表结构生成对应代码为例来说明如何实现一个代码生成器。有以下几个重点：

1. 数据库和表解析，用于生成model及其他代码

   通过数据库解析获取数据库中表的名称、表字段等属性：可以根据表名称确定实体类名称，根据字段确定实体类中属性（如：tb_user表对应的实体类就是User）

2. 模板开发生成代码文件

   模板中定义公共的基础代码和需要替换的占位符内容（如:${tableName}最终会根据数据库表替换为User），根据解析好的数据库信息进行数据替换并生成代码文件

3. 基础框架的模板代码抽取

通过思路分析不难发现，对于代码生成工具而言只需要搞定数据库解析和模板开发。那么代码自动生成也并没有那么神秘和复杂。那接下来的课程和各位着重从这两个方面开始讲解直至完成属于自己的代码生成器。

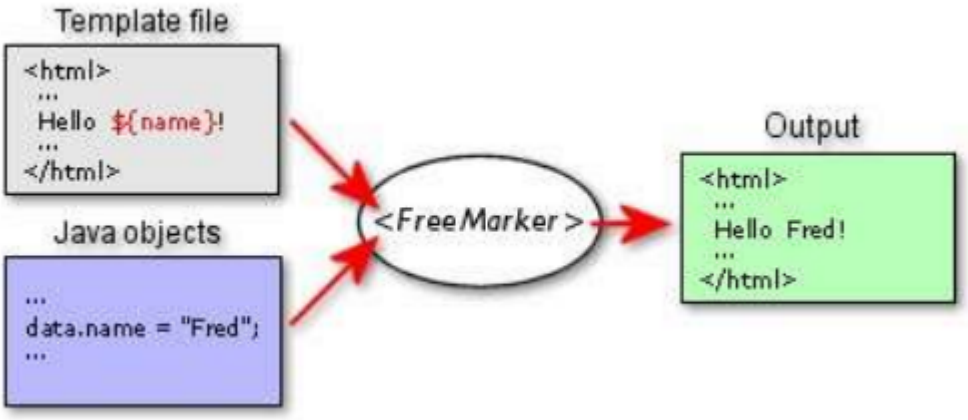
2 深入FreeMarker

2.1 什么是FreeMarker

FreeMarker 是一款模板引擎：一种基于模板的、用来生成输出文本(任何来自于 HTML格式的文本用来自动生成源代码)的通用工具。它是为 Java 程序员提供的一个开发包或者说是类库。它不是面向最终用户，而是为程序员提供的可以嵌入他们开发产品的一款应用程序。

FreeMarker 的设计实际上是被用来生成 HTML 网页，尤其是通过基于实现了 MVC(ModelView Controller，模型-视图-控制器)模式的 Servlet 应用程序。使用 MVC 模式的动态网页的构思使得你可以将前端设计者(编写 HTML)从程序员中分离出来。所有人各司其职，发挥其擅长的一面。网页设计师可以改写页面的显示效果而不受程序员编译代码的影响，因为应用程序的逻辑(Java 程序)和页面设计(FreeMarker 模板)已经分开了。页面模板代码不会受到复杂的程序代码影响。这种分离的思想即便对一个程序员和页面设计师是同一个人的项目来说都是非常有用的，因为分离使得代码保持简洁而且便于维护。

尽管 FreeMarker 也有编程能力，但它也不是像 PHP 那样的一种全面的编程语言。反而，Java 程序准备的数据来显示(比如 SQL 查询)，FreeMarker 仅仅使用模板生成文本页面来呈现已经准备好的数据



FreeMarker 不是 Web 应用框架。它是 Web 应用框架中的一个适用的组件，但是FreeMarker 引擎本身并不知道 HTTP 协议或 Servlet。它仅仅来生成文本。即便这样，它也非常适用于非 Web 应用环境的开发

2.2 Freemarker的应用场景

（1）动态页面

基于模板配置和表达式生成页面文件，可以像jsp一样被客户端访问

（2）页面静态化

对于系统中频繁使用数据库进行查询但是内容更新很小的应用，都可以用FreeMarker将网页静态化，这样就避免了大量的数据库访问请求，从而提高网站的性能

（3）代码生成器

可以自动根据后台配置生成页面或者代码

freemarker的特征与亮点

- 强大的模板语言：有条件的块，迭代，赋值，字符串和算术运算和格式化，宏和函数，编码等更多的功能；
- 多用途且轻量：零依赖，输出任何格式，可以从任何地方加载模板（可插拔），配置选项丰富；
- 智能的国际化和本地化：对区域设置和日期/时间格式敏感。
- XML处理功能：将dom-s放入到XML数据模型并遍历它们，甚至处理他们的声明
- 通用的数据模型：通过可插拔适配器将java对象暴露于模板作为变量树。

2.3 Freemarker的基本使用

2.3.1 构造环境

创建maven工程codeutil，并引入响应坐标

```
    <dependencies>
        <!--freemarker核心包 -->
        <dependency>
            <groupId>org.freemarker</groupId>
            <artifactId>freemarker</artifactId>
            <version>2.3.20</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

2.3.2 入门案例

（1）创建模板template.ftl

```
欢迎您：${username}
```

（2）使用freemarker完成操作

```
public class FreemarkTest01 {
    @Test
    public void testProcessTemplate() throws Exception {

        //1.创建freeMarker配置实例
        Configuration cfg = new Configuration();
        //2.设置模板加载器：开始加载模板，并且把模板加载在缓存中
        cfg.setTemplateLoader(new FileTemplateLoader(new File("templates")));
        //3.创建数据模型
        Map<String,Object> dataModel = new HashMap<>();
        dataModel.put("username","张三");
        //4.获取模板
        Template template = cfg.getTemplate("temp01.ftl");
        //
        /**
         * 5.处理模板内容(i.输出到文件)
         *      process:
         *          参数一：数据模型（map集合）
         *          参数二：Writer对象（文件，控制台）
         */
        //i.输出到文件
        //template.process(dataModel, new FileWriter(new
File("C:\\Users\\ThinkPad\\Desktop\\ihrm\\day12\\测试\\aa.text")));
        //i.打印到控制台
        template.process(dataModel, new PrintWriter(System.out));//在控制台输出内容
    }
```

```
}
```

2.3.3 字符串模板

```
public class FreemarkTest02 {

    private Configuration conf;

    @Before
    public void init() {
        conf = new Configuration();
    }

    @Test
    public void testProcessTemplateString() throws Exception {
        String templateString = "欢迎您：${username}";
        Map<String,Object> dataMap = new HashMap();
        dataMap.put("username","张三");
        StringWriter out = new StringWriter();
        /**
         * 自定义模板
         * 1.模板名称
         * 2.模板的正文内容
         * 3.configuration对象
         */
        Template template = new Template("templateString...",new
StringReader(templateString),conf);
        //处理模板内容
        template.process(dataMap, out);
        System.out.println(out.toString());
    }
}
```

2.4 Freemarker模板

2.4.1 概述

FreeMarker模板文件主要有5个部分组成：

1. 数据模型：模板能用的所有数据
2. 文本，直接输出的部分
3. 注释，即<#--...-->格式不会输出
4. 插值（Interpolation）：即${..}或者#{..}格式的部分,将使用数据模型中的部分替代输出
5. FTL指令：FreeMarker指令，和HTML标记类似，名字前加#予以区分，不会输出。

2.4.2 数据模型

FreeMarker（还有模板开发者）并不关心数据是如何计算的，FreeMarker 只是知道真实的数据是什么。模板能用的所有数据被包装成 data-model 数据模型

```
(root)
  |
  +- user = "Big Joe"
  |
  +- latestProduct
      |
      +- url = "products/greenmouse.html"
      |
      +- name = "green mouse"
```

2.4.3 模板的常用标签

在FreeMarker模板中可以包括下面几个特定部分：

1. **${...}：称为interpolations，FreeMarker会在输出时用实际值进行替代。**
   - ${name}可以取得root中key为name的value。
   - ${person.name}可以取得成员变量为person的name属性
2. **<#...>：FTL标记（FreeMarker模板语言标记）：类似于HTML标记，为了与HTML标记区分**
3. <@>：宏，自定义标签
4. 注释：包含在<#--和-->（而不是）之间

2.4.4 模板的常用指令

- **if指令**

  分支控制语句

```
<#if condition>
....
<#elseif condition2>
...
<#elseif condition3>
...
<#else>
...
</#if>
```

- **list、break指令**

  list指令时一个典型的迭代输出指令，用于迭代输出数据模型中的集合

```
<#list sequence as item>
  ...
</#list>
```

除此之外，迭代集合对象时，还包括两个特殊的循环变量：

a、item_index：当前变量的索引值。

b、item_has_next:是否存在下一个对象

也可以使用<#break>指令跳出迭代

```
<#list ["星期一","星期二","星期三","星期四","星期五"] as x>
    ${x_index +1}.${x} <#if x_has_next>,</#if>
    <#if x = "星期四"><#break></#if>
</#list>
```

- **include 指令**

include指令的作用类似于JSP的包含指令，用于包含指定页，include指令的语法格式如下

```
<#include filename [options]></#include>
```

在上面的语法格式中，两个参数的解释如下

a、filename：该参数指定被包含的模板文件

b、options：该参数可以省略，指定包含时的选项，包含encoding和parse两个选项，encoding指定包含页面时所使用的解码集，而parse指定被包含是否作为FTL文件来解析。如果省略了parse选项值，则该选项值默认是true

- **assign指令**

它用于为该模板页面创建或替换一个顶层变量

```
<#assign name = zhangsan />
```

- 内置函数

FreeMarker还提供了一些内建函数来转换输出，可以在任何变量后紧跟?,?后紧跟内建函数，就可通过内建函数来转换输出变量。下面是常用的内建的字符串函数：

```
?html:html字符转义

  ?cap_first: 字符串的第一个字母变为大写形式

  ?lower_case :字符串的小写形式

  ?upper_case :字符串的大写形式

  ?trim:去掉字符串首尾的空格

  ?substring:截字符串

  ?lenth: 取长度

  ?size: 序列中元素的个数

  ?int : 数字的整数部分（比如- 1.9?int 就是- 1）
```

```
?replace:字符串替换
```

3 数据库之元数据

3.1 数据库中的元数据

（1）什么是数据元数据？

元数据（MetaData），是指定义数据结构的数据。那么数据库元数据就是指定义数据库各类对象结构的数据。 例如数据库中的数据库名，表明， 列名、用户名、版本名以及从SQL语句得到的结果中的大部分字符串是元数据

（2）数据库元数据的作用

- 在应用设计时能够充分地利用数据库元数据
- 深入理解了数据库组织结构，再去理解数据访问相关框架的实现原理会更加容易。

（3）如何获取元数据

在我们前面使用JDBC来处理数据库的接口主要有三个，即Connection，PreparedStatement和ResultSet这三个，而对于这三个接口，还可以获取不同类型的元数据，通过这些元数据类获得一些数据库的信息。下面将对这三种类型的元数据对象进行各自的介绍并通过使用MYSQL数据库进行案例说明（部分代码再不同数据库中略有不同，学员如有其他需求请查阅API）

3.2 数据库元数据

3.2.1 概述

数据库元数据（DatabaseMetaData）：是由Connection对象通过getMetaData方法获取而来，主要封装了是对数据库本身的一些整体综合信息，例如数据库的产品名称，数据库的版本号，数据库的URL，是否支持事务等等。

以下有一些关于DatabaseMetaData的常用方法：

- getDatabaseProductName：获取数据库的产品名称
- getDatabaseProductName：获取数据库的版本号
- getUserName：获取数据库的用户名
- getURL：获取数据库连接的URL
- getDriverName：获取数据库的驱动名称
- driverVersion：获取数据库的驱动版本号
- isReadOnly：查看数据库是否只允许读操作
- supportsTransactions：查看数据库是否支持事务

3.2.2 入门案例

（1）构建环境

```
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.6</version>
        </dependency>
```

（2）获取数据库综合信息

```
public class DataBaseMetaDataTest {

    private Connection conn;

    @Before
    public void init() throws Exception {
        Class.forName("com.mysql.jdbc.Driver");
        Properties props =new Properties();
        //设置连接属性,使得可获取到表的REMARK(备注)
        props.put("remarksReporting","true");
        props.put("user", "root");
        props.put("password", "111111");
        conn = java.sql.DriverManager.
                getConnection("jdbc:mysql://127.0.0.1:3306/?
useUnicode=true&amp;characterEncoding=UTF8", props);
    }

    @Test
    public void testDatabaseMetaData() throws SQLException {
        //获取数据库元数据
        DatabaseMetaData dbMetaData =  conn.getMetaData();
        //获取数据库产品名称
        String productName = dbMetaData.getDatabaseProductName();
        System.out.println(productName);
        //获取数据库版本号
        String productVersion = dbMetaData.getDatabaseProductVersion();
        System.out.println(productVersion);
        //获取数据库用户名
        String userName = dbMetaData.getUserName();
        System.out.println(userName);
        //获取数据库连接URL
        String userUrl = dbMetaData.getURL();
        System.out.println(userUrl);
        //获取数据库驱动
        String driverName = dbMetaData.getDriverName();
        System.out.println(driverName);
        //获取数据库驱动版本号
        String driverVersion = dbMetaData.getDriverVersion();
        System.out.println(driverVersion);
        //查看数据库是否允许读操作
        boolean isReadOnly = dbMetaData.isReadOnly();
        System.out.println(isReadOnly);
        //查看数据库是否支持事务操作
        boolean supportsTransactions = dbMetaData.supportsTransactions();
        System.out.println(supportsTransactions);
    }
}
```

（3）获取数据库列表

```
    @Test
    public void testFindAllCatalogs() throws Exception {
        //获取元数据
        DatabaseMetaData metaData = conn.getMetaData();
        //获取数据库列表
        ResultSet rs = metaData.getCatalogs();
        //遍历获取所有数据库表
        while(rs.next()){
            //打印数据库名称
            System.out.println(rs.getString(1));
        }
        //释放资源
        rs.close();
        conn.close();
    }
```

（4）获取某数据库中的所有表信息

```
    @Test
    public void testFindAllTable() throws Exception{
        //获取元数据
        DatabaseMetaData metaData = conn.getMetaData();
        //获取所有的数据库表信息
        ResultSet tablers = metaData.getTables("ihrm", "", "bs_user", new String[]
{"TABLE"});
        //拼装table
        while(tablers.next()) {
            //所属数据库
            System.out.println(tablers.getString(1));
            //所属schema
            System.out.println(tablers.getString(2));
            //表名
            System.out.println(tablers.getString(3));
            //数据库表类型
            System.out.println(tablers.getString(4));
            //数据库表备注
            System.out.println(tablers.getString(5));
        }
    }
```

3.3 参数元数据

参数元数据（ParameterMetaData）：是由PreparedStatement对象通过getParameterMetaData方法获取而来，主要是针对PreparedStatement对象和其预编译的SQL命令语句提供一些信息，ParameterMetaData能提供占位符参数的个数，获取指定位置占位符的SQL类型等等

以下有一些关于ParameterMetaData的常用方法：

- getParameterCount：获取预编译SQL语句中占位符参数的个数

```
@Test
public void test() throws Exception {
    String sql = "select * from bs_user where id=?";
    PreparedStatement pstmt = conn.prepareStatement(sql);
    pstmt.setString(1, "1063705482939731968");
    //获取ParameterMetaData对象
    ParameterMetaData paramMetaData = pstmt.getParameterMetaData();
    //获取参数个数
    int paramCount = paramMetaData.getParameterCount();
    System.out.println(paramCount);
}
```

3.4 结果集元数据

结果集元数据（ResultSetMetaData）：是由ResultSet对象通过getMetaData方法获取而来，主要是针对由数据库执行的SQL脚本命令获取的结果集对象ResultSet中提供的一些信息，比如结果集中的列数、指定列的名称、指定列的SQL类型等等，可以说这个是对于框架来说非常重要的一个对象。

以下有一些关于ResultSetMetaData的常用方法：

- getColumnCount：获取结果集中列项目的个数
- getColumnType：获取指定列的SQL类型对应于Java中Types类的字段
- getColumnTypeName：获取指定列的SQL类型
- getClassName：获取指定列SQL类型对应于Java中的类型(包名加类名)

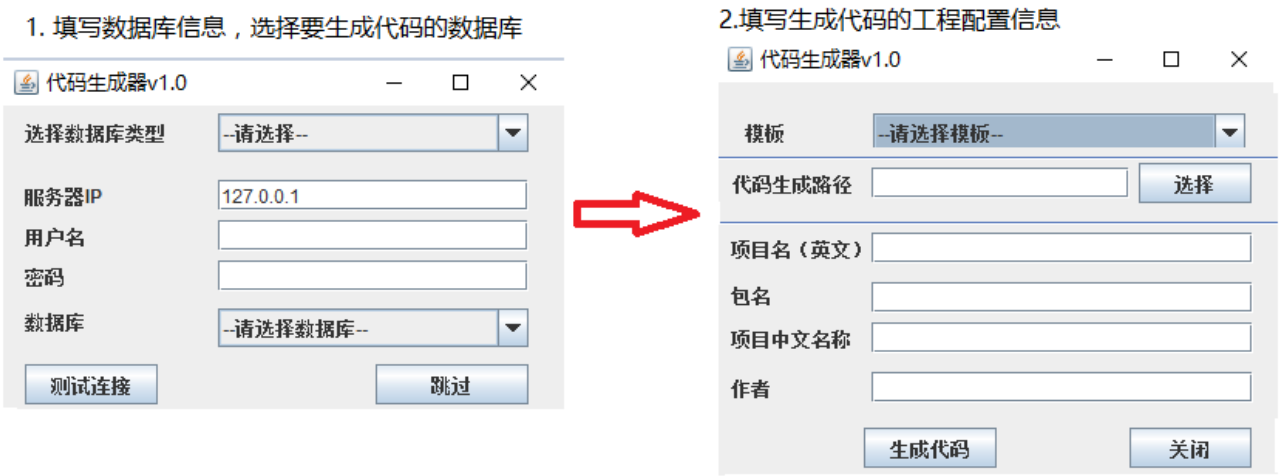
```
@Test
public void test() throws Exception {
    String sql = "select * from bs_user where id=?";
    PreparedStatement pstmt = conn.prepareStatement(sql);
    pstmt.setString(1, "1063705482939731968");
    //执行sql语句
    ResultSet rs = pstmt.executeQuery() ;
    //获取ResultSetMetaData对象
    ResultSetMetaData metaData = rs.getMetaData();
    //获取查询字段数量
    int columnCount = metaData.getColumnCount() ;

    for (int i=1;i<=columnCount;i++) {
        //获取表名称
        String columnName = metaData.getColumnName(i);
        //获取java类型
        String columnClassName = metaData.getColumnClassName(i);
        //获取sql类型
        String columnTypeName = metaData.getColumnTypeName(i);
        System.out.println(columnName);
        System.out.println(columnClassName);
        System.out.println(columnTypeName);
    }
    System.out.println(columnCount);
}
```

4 代码生成器搭建环境

4.1 思路分析

工具的执行逻辑如下图所示：



如上分析，得知完成代码生成器需要以下几个操作：

1. 用户填写的数据库信息，工程搭建信息需要构造到实体类对象中方便操作
2. 数据库表信息，数据库字段信息需要构造到实体类中
3. 构造Freemarker数据模型，将数据库表对象和基本配置存入到Map集合中
4. 借助Freemarker完成代码生成
5. 自定义公共代码模板

4.2 搭建环境

4.2.1 配置坐标

```
    <dependencies>
        <dependency>
            <groupId>org.freemarker</groupId>
            <artifactId>freemarker</artifactId>
            <version>2.3.20</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.6</version>
```

```
        </dependency>
    </dependencies>
```

4.2.2 配置实体类

（1）UI页面获取的数据库配置，封装到数据库实体类中

```
//数据库实体类
public class DataBase {

    private static String mysqlUrl = "jdbc:mysql://[ip]:[port]/[db]?
useUnicode=true&amp;characterEncoding=UTF8";
    private static String oracleUrl = "jdbc:oracle:thin:@[ip]:[port]:[db]";

    private String dbType;//数据库类型
    private String driver;
    private String userName;
    private String passWord;
    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);
    }

    public DataBase(String dbType,String ip,String port,String db) {
        this.dbType = dbType;
        if("MYSQL".endsWith(dbType.toUpperCase())) {
            this.driver="com.mysql.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);
        }
    }

    public String getDbType() {
        return dbType;
    }

    public void setDbType(String dbType) {
        this.dbType = dbType;
    }

    public String getDriver() {
        return driver;
```

```
    }

    public void setDriver(String driver) {
        this.driver = driver;
    }

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

    public String getPassWord() {
        return passWord;
    }

    public void setPassWord(String passWord) {
        this.passWord = passWord;
    }

    public String getUrl() {
        return url;
    }

    public void setUrl(String url) {
        this.url = url;
    }
}
```

（2）UI页面获取的自动生成工程配置，封装到设置实体类中

```
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;

    public Settings(String project, String pPackage, String projectComment, String
author) {
        if(StringHelper.isNotBlank(project)) {
            this.project = project;
        }
        if(StringHelper.isNotBlank(pPackage)) {
            this.pPackage = pPackage;
        }
        this.projectComment = projectComment;
```

```
        this.author = author;
        String[] paths = pPackage.split("\\.");
        path1 = paths[0];
        path2 = paths.length>1?paths[1]:path2;
        path3 = paths.length>2?paths[2]:path3;
        pathAll = pPackage.replaceAll(".","/");
    }

    public Map<String, Object> getSettingMap(){
        Map<String, Object> map = new HashMap<>();
        Field[] declaredFields = Settings.class.getDeclaredFields();
        for (Field field : declaredFields) {
            field.setAccessible(true);
            try{
                map.put(field.getName(), field.get(this));
            }catch (Exception e){}
        }
        return map;
    }

    public String getProject() {
        return project;
    }

    public void setProject(String project) {
        this.project = project;
    }

    public String getpPackage() {
        return pPackage;
    }

    public void setpPackage(String pPackage) {
        this.pPackage = pPackage;
    }

    public String getProjectComment() {
        return projectComment;
    }

    public void setProjectComment(String projectComment) {
        this.projectComment = projectComment;
    }

    public String getAuthor() {
        return author;
    }

    public void setAuthor(String author) {
        this.author = author;
    }

    public String getPath1() {
```

```
        return path1;
    }

    public void setPath1(String path1) {
        this.path1 = path1;
    }

    public String getPath2() {
        return path2;
    }

    public void setPath2(String path2) {
        this.path2 = path2;
    }

    public String getPath3() {
        return path3;
    }

    public void setPath3(String path3) {
        this.path3 = path3;
    }

    public String getPathAll() {
        return pathAll;
    }

    public void setPathAll(String pathAll) {
        this.pathAll = pathAll;
    }
}
```

（3）将查询数据表的元数据封装到Table实体类

```
public class Table {

    private String name;//表名称
    private String name2;//处理后的表名称
    private String comment;//介绍
    private String key;// 主键列
    private List<Column> columns;//列集合

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getName2() {
        return name2;
    }
```

```
    public void setName2(String name2) {
        this.name2 = name2;
    }

    public String getComment() {
        return comment;
    }

    public void setComment(String comment) {
        this.comment = comment;
    }

    public String getKey() {
        return key;
    }

    public void setKey(String key) {
        this.key = key;
    }

    public List<Column> getColumns() {
        return columns;
    }

    public void setColumns(List<Column> columns) {
        this.columns = columns;
    }
}
```

（4）将查询数据字段的元数据封装到Column实体类

```
/**
 * 列对象
 */
public class Column {

    private String columnName;//列名称
    private String columnName2;//列名称(处理后的列名称)
    private String columnType;//列类型
    private String columnDbType;//列数据库类型
    private String columnComment;//列备注D
    private String columnKey;//是否是主键

    public String getColumnName() {
        return columnName;
    }

    public void setColumnName(String columnName) {
        this.columnName = columnName;
    }

    public String getColumnName2() {
```

```
        return columnName2;
    }

    public void setColumnName2(String columnName2) {
        this.columnName2 = columnName2;
    }

    public String getColumnType() {
        return columnType;
    }

    public void setColumnType(String columnType) {
        this.columnType = columnType;
    }

    public String getColumnDbType() {
        return columnDbType;
    }

    public void setColumnDbType(String columnDbType) {
        this.columnDbType = columnDbType;
    }

    public String getColumnComment() {
        return columnComment;
    }

    public void setColumnComment(String columnComment) {
        this.columnComment = columnComment;
    }

    public String getColumnKey() {
        return columnKey;
    }

    public void setColumnKey(String columnKey) {
        this.columnKey = columnKey;
    }
}
```

4.2.3 导入工具类

导入资料中提供的工具类

FileUtils：文件处理工具类：

- getRelativePath：获取文件的相对路径

- searchAllFile ：查询文件夹下所有文件
- mkdir ：创建文件目录

PropertiesMaps：加载所有properties并存入Map集合中：

- 加载代码生成器的基本配置文件
- 加载用户的自定义配置文件
- 所有配置文件需要放置到"/properties"文件夹下

StringHelper：字符串处理工具类

4.2.4 *配置UI界面

为了方便演示，使用swing程序构造了一套UI页面，对于swing不需要大家掌握，直接使用即可