

#

# 项目总介:

---

## 项目概述:

---

对于一个多模块的项目 最麻烦的就是对项目创建了, 太过于耗费时间 以及文件夹的创建、 依赖的导入、 配置文件的书写 对于新手来说无非就是需要去到老项目或者是以前的笔记中copy 过来 十分麻烦。对于 maven 的多模块创建 如果对于maven不甚了解 有可能还会再创建项目的时候就会报错 真是令人十分头大。然而对于市面上的代码生成器 不是很单一 就是只有表的增删改查, 故而 我直接构写了这个多模块项目构建生成器。

## 使用方式:

---

1. 首先将com文件解压后放入到你的maven 仓库中
2. 创建一个maven 项目
3. 之后引入这个jar包的依赖

```
<dependency>
  <groupId>com.yszhdhy</groupId>
  <artifactId>generator</artifactId>
  <version>1.0-SNAPSHOT</version>
</dependency>
```

4. 随便创建 一个文件 内容为:

```
import com.yszhdhy.generator.model.project.Project;
import org.dom4j.DocumentException;

import java.io.FileNotFoundException;

public class ceshi {

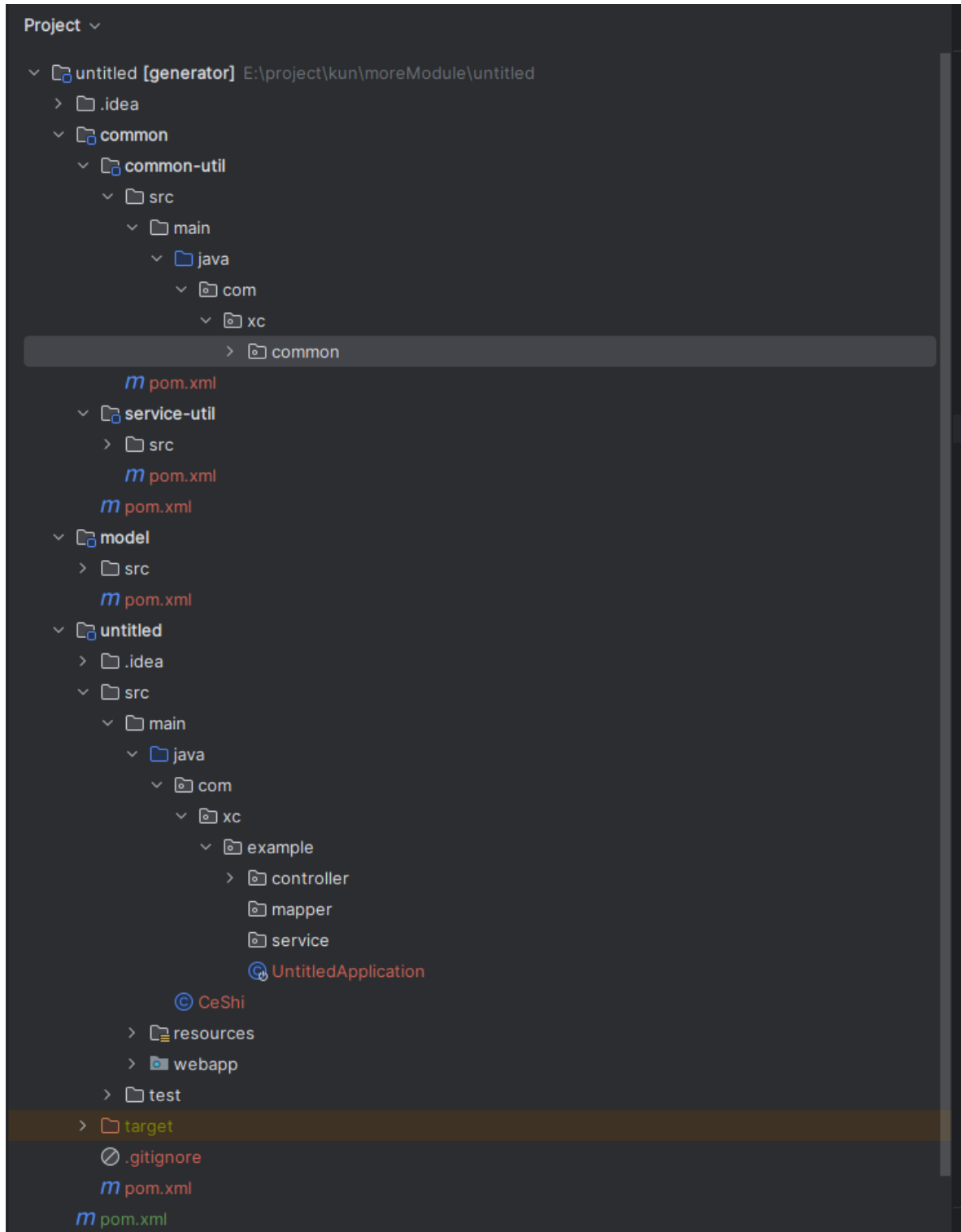
    public static void main(String[] args) throws DocumentException,
FileNotFoundException {

        Project project = new Project();
        project.generate();

    }

}
```

5. 运行即可得到：这样的项目结构：



6. yaml文件

```
spring:
  jackson:
    time-zone: Asia/Shanghai
    date-format: yyyy-MM-dd
  application:
    name: untitled
  datasource:
    password: '123456'
```

```

driver-class-name: com.mysql.cj.jdbc.Driver
url: jdbc:mysql://localhost:3306/dbName?
useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai
username: root
mvc:
  format:
    date: yyyy-MM-dd
    date-time: yyyy-MM-dd HH:mm:ss
server:
  port: 8080
mybatis-plus:
  configuration:
    log-impl: org.apache.ibatis.logging.stdout.StdOutImpl
    map-underscore-to-camel-case: true
  global-config:
    db-config:
      logic-not-delete-value: 0
      logic-delete-value: 1
      logic-delete-field: isDelete
  type-aliases-package: org.example.entity

```

## 7. 父pom文件

```

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.yszhdhy.generator</groupId>
  <artifactId>generator</artifactId>
  <packaging>pom</packaging>
  <version>1.0</version>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.6.RELEASE</version>
  </parent>
  <properties>
    <maven.compiler.source>8</maven.compiler.source>
    <maven.compiler.target>8</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <java.version>1.8</java.version>
  </properties>
  <modules>
    <module>untitled</module>
    <module>model</module>
    <module>common</module>
  </modules>

```

```
<dependencies>
  <dependency>
    <groupId>org.dom4j</groupId>
    <artifactId>dom4j</artifactId>
    <version>2.1.3</version>
  </dependency>
</dependencies>
</project>
```

## 项目架构：

---

首先导入依赖：

```
<dependency>
  <groupId>org.dom4j</groupId>
  <artifactId>dom4j</artifactId>
  <version>2.1.3</version>
</dependency>
<dependency>
  <groupId>jaxen</groupId>
  <artifactId>jaxen</artifactId>
  <version>1.2.0</version>
</dependency>
<dependency>
  <groupId>org.yaml</groupId>
  <artifactId>snakeyaml</artifactId>
  <version>1.25</version>
</dependency>
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>1.18.24</version>
</dependency>
```

包结构分为：

1. 构建项目框架的 [FileBuilderOfProject](#)
2. 构建项目中每个pom文件 或者 java文件 或是yaml文件的 [structure](#)
3. 存放实体的model
4. 工具类 utils
5. 枚举常量 [constant](#)

- main
  - java
    - com
      - yszhdhy
        - generator
          - constant
            - code
            - common
            - dependency
            - yaml
          - construct
            - build
              - FileBuilder
                - Builder
                - Director
              - FileBuilderOfProject
                - common
                - model
                - service
                  - BuildTheFile
              - structure
                - common
                  - commonUtil
                  - serviceUtil
                  - BuilderOfCommon
                - Model
                  - Builder
                  - FileResolverOfModel
                  - ModelPomResolver
                  - ItemMovement
            - model
              - dto
              - project
              - vo
                - BaseStarter
                - BaseStarterAdapter
              - service
            - utils

## 项目技术：

---

1. dom4j
2. io流
3. SnakeYAML

此版本可以构建java后端的一个maven多模块的项目结构。里面的配置类采用的构建者模式进行写入 达到可插拔的效果 后续功能等待更新eg：整合mybatis 以及前段Vue3

## 此项目中java 知识点：

---

### 重写 hashCode () 的意义：

```
/**
 * 在上述示例中，Person 类重写了 hashCode() 方法和 equals() 方法。
 *
 * 在 hashCode() 方法中，使用常量 17 初始化结果变量 result，然后使用质数 31 乘以 result 并与属性值进行混合运算。这样做的目的是生成一个具有较好分布性的哈希码。
 * 在 equals() 方法中，首先判断两个对象的引用是否相同，如果是则直接返回 true。然后判断传入的对象是否为 null 或者类不同，如果是则返回 false。最后比较两个对象的属性值是否相等。
 * 在 main 方法中，创建了两个相等的 Person 对象，并使用 equals() 方法进行比较，以及使用 hashCode() 方法获取哈希码。根据重写的逻辑，两个对象被判断为相等，且它们的哈希码相同。
 *
 * 这个示例展示了重写 hashCode() 和 equals() 方法的基本原则和使用方式。实际应用中，根据对象的属性和需求，你可能需要调整哈希码的计算逻辑和相等性判断的条件。
 */
public class Person {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

```
/**
 * 是的，重写 hashCode() 方法的一个主要目的是确保生成的哈希码具有较好的分布性。一个好的哈希码应该尽可能地将不同对象映射到不同的哈希码值上，以减少在哈希表等数据结构中的碰撞（冲突）概率。
 *
 * 在哈希表数据结构中，对象的哈希码被用作索引来存储和查找对象。如果哈希码的分布不均匀，即不同对象的哈希码经常发生碰撞，那么哈希表的性能将受到影响，查找操作的效率会下降。
 *
 * 通过重写 hashCode() 方法，你可以根据对象的属性值来计算哈希码。在计算哈希码时，通常使用一些算法，如乘法和位运算，将对象的属性值组合起来生成一个整数值。这样做的目的是尽量使不同的对象具有不同的哈希码，减少哈希碰撞的概率。
 *
 * 同时，为了进一步提高哈希码的分布性，经常选择一个质数作为乘法因子，并使用不同的属性进行混合运算。这种方式可以避免相同属性值在不同位置产生相同的哈希码。
```

```

*
* 总结来说，重写 hashCode() 方法的目的是生成具有较好分布性的哈希码，以提高哈希表等数据结构的
性能，减少碰撞的概率，从而提高查找操作的效率。
* @return
*/
@Override
public int hashCode() {
    int result = 17;
    result = 31 * result + name.hashCode();
    result = 31 * result + age;
    return result;
}

@Override
public boolean equals(Object obj) {
    if (this == obj) {
        return true;
    }
    if (obj == null || getClass() != obj.getClass()) {
        return false;
    }
    Person otherPerson = (Person) obj;
    return name.equals(otherPerson.name) && age == otherPerson.age;
}

public static void main(String[] args) {
    Person person1 = new Person("John", 25);
    Person person2 = new Person("John", 25);

    System.out.println(person1 == person2); //false 这个是判断了内存地址是否相同
    System.out.println(person1.equals(person2)); // true 这个只是判断属性值是否相
等 素以还是不错的
    System.out.println(person1.hashCode() == person2.hashCode()); // true
}
}

```

## 枚举的使用：

```
enum MyEnum {
    BASE_PACKAGE(Arrays.asList("vo", "to", "dto"));

    private List<String> packages;

    MyEnum(List<String> packages) {
        this.packages = packages;
    }

    public List<String> getPackages() {
        return packages;
    }
}
```

这样，在调用 `MyEnum.BASE_PACKAGE.getPackages()` 时，实际上是获取了 `BASE_PACKAGE` 枚举值中的字符串列表，并将其赋值给了 `packageList` 变量。因此，输出结果为 `[vo, to, dto]`。

请注意，以上代码片段是一种推测，并假设了 `MyEnum` 枚举类的结构。如果你能提供更多关于 `MyEnum` 枚举类的信息，我可以给出更准确的解释。

## 枚举使用2：

```
package cloud.makeronbean.generate.constant.code;

import cloud.makeronbean.generate.resolver.PomResolver;

@SuppressWarnings("all")
public enum Knife4jCodeConst implements CodeTemplate {
    KNIFE4J_CONFIG("config/knife4j", "Knife4jConfiguration.java", "%s\n\n" +
        "import org.springframework.beans.factory.annotation.Value;\n" +
        "import org.springframework.context.annotation.Bean;\n" +
        "import org.springframework.context.annotation.Configuration;\n" +
        "import springfox.documentation.builders.ApiInfoBuilder;\n" +
        "import springfox.documentation.builders.PathSelectors;\n" +
        "import springfox.documentation.builders.RequestHandlerSelectors;\n" +
        "import springfox.documentation.spi.DocumentationType;\n" +
        "import springfox.documentation.spring.web.plugins.Docket;\n" +
        "import\nspringfox.documentation.swagger2.annotations.EnableSwagger2WebMvc;\n" +
        "\n" +
        "\n" +
        "@Configuration\n" +
        "@EnableSwagger2WebMvc\n" +
        "public class Knife4jConfiguration {\n" +
```



```

        "\n" +
        @Bean(value = \"defaultApi2\")\n" +
        public Docket defaultApi2() {\n" +
        return new Docket(DocumentationType.SWAGGER_2)\n" +
        .apiInfo(new ApiInfoBuilder()\n" +
        .title(\"swagger-bootstrap-ui-demo RESTful
APIS\")\n" +
        .description(\"# swagger-bootstrap-ui-demo
RESTful APIS\")\n" +
        .build())\n" +
        //分组名称\n" +
        .select()\n" +
        //这里指定Controller扫描包路径\n" +
        .apis(RequestHandlerSelectors.basePackage(\"\" +
PomResolver.getInstance().getGroupId() + \".controller\")\n" +
        .paths(PathSelectors.any())\n" +
        .build();\n" +
        }\n" +
        "}),

;
private final String path;
private final String fileName;
private final String codeTemplate;

knife4jCodeConst(String path, String fileName, String codeTemplate) {
    this.path = path;
    this.fileName = fileName;
    this.codeTemplate = codeTemplate;
}

@Override
public String getPath() {
    return path;
}

@Override
public String getFileName() {
    return fileName;
}

@Override
public String getCodeTemplate() {
    return codeTemplate;
}
}

```

```
List<CodeTemplate> codeTemplateList = new ArrayList<>
(Arrays.asList(Knife4jCodeConst.values()));
为什么可以这样写
```

这段代码定义了一个枚举类 `Knife4jCodeConst`，它实现了一个接口 `CodeTemplate`。枚举类中定义了一个构造方法和几个成员变量，其中包含一个 `codeTemplate` 字符串，表示代码模板。

通过调用 `Knife4jCodeConst.values()` 方法，可以获取 `Knife4jCodeConst` 枚举类的所有枚举值，返回一个数组。然后通过 `Arrays.asList()` 方法将这个数组转换为一个列表对象。这样得到的列表就包含了 `Knife4jCodeConst` 枚举类的所有枚举值。

最后，通过 `new ArrayList<>(...)` 的方式，将列表对象转换为 `ArrayList` 集合对象。这样就得到了一个包含 `Knife4jCodeConst` 枚举类的所有枚举值的 `ArrayList` 集合对象。

这样写的目的是将枚举值存储在一个集合中，方便进行遍历和操作。

## 文件的复制移动删除：

```
// 当前项目根目录路径
String projectRootPath = System.getProperty("user.dir");

try {
    // 获取当前项目文件夹的名称
    String projectName = new File(projectRootPath).getName();

    // 创建临时文件夹
    File tempFolder = new File(projectRootPath, projectName);
    if (tempFolder.mkdir()) {
        System.out.println("成功创建临时文件夹：" +
tempFolder.getAbsolutePath());

        // 将当前项目的内容复制到临时文件夹中
        copyFolder(new File(projectRootPath), tempFolder);

        // 删除原项目文件夹
        deleteFolder(new File(projectRootPath), new File(projectRootPath,
projectName));

        System.out.println("项目移动成功！");
    } else {
        System.out.println("无法创建临时文件夹：" +
tempFolder.getAbsolutePath());
    }
}
```

```

    } catch (IOException e) {
        System.out.println("项目移动失败: " + e.getMessage());
    }
}

private static void copyFolder(File source, File destination) throws IOException
{
    // 如果目标文件夹不存在, 则创建它
    if (!destination.exists()) {
        destination.mkdirs();
    }

    // 遍历源文件夹中的所有文件和子文件夹
    File[] files = source.listFiles();
    if (files != null) {
        for (File file : files) {
            if (file.equals(destination)) {
                return;
            } else {
                if (file.isDirectory()) {
                    // 递归复制子文件夹
                    copyFolder(file, new File(destination, file.getName()));
                } else {
                    // 复制文件
                    Files.copy(file.toPath(), new File(destination,
file.getName()).toPath(), StandardCopyOption.REPLACE_EXISTING);
                }
            }
        }
    }
}

private static void deleteFolder(File folder, File projectName) {
    // 遍历文件夹中的所有文件和子文件夹
    File[] files = folder.listFiles();
    if (files != null) {
        for (File file : files) {
            if (file.equals(projectName)) {
                System.out.println("不删除" + file);
            } else {
                if (file.isDirectory()) {
                    // 递归删除子文件夹
                    deleteFolder(file, null);
                } else {
                    // 删除文件
                    file.delete();
                }
            }
        }
    }
}

// 删除空文件夹

```

```
        folder.delete();
    }
```

# DOM4j:

导入依赖:

```
<dependency>
  <groupId>org.dom4j</groupId>
  <artifactId>dom4j</artifactId>
  <version>2.1.3</version>
</dependency>
```

pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.example</groupId>
  <artifactId>untitled</artifactId>
  <packaging>war</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>untitled Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <properties>
    <maven.compiler.source>8</maven.compiler.source>
    <maven.compiler.target>8</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <java.version>1.8</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>

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

```
        <version>2.3.31</version>
    </dependency>

    <dependency>
        <groupId>cloud.makaronbean.generate</groupId>
        <artifactId>generate4j</artifactId>
        <version>1.0</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>
    <dependency>
        <groupId>com.baomidou</groupId>
        <artifactId>mybatis-plus-boot-starter</artifactId>
        <version>3.4.1</version>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.30</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <optional>true</optional>
        <scope>runtime</scope>
    </dependency>

    <dependency>
        <groupId>org.dom4j</groupId>
        <artifactId>dom4j</artifactId>
        <version>2.1.3</version>
    </dependency>

</dependencies>
<build>
    <finalName>untitled</finalName>
</build>
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.6.RELEASE</version>
</parent>
</project>
```

## 完整进行解析：

```
import org.dom4j.Document;
import org.dom4j.DocumentException;
import org.dom4j.Element;
import org.dom4j.io.SAXReader;

import java.io.File;
import java.util.List;

public class Demo {
    public static void main(String[] args) {
        //创建解析器对象
        SAXReader saxReader = new SAXReader();

        Element dep = null;

        try {
            // 读取文件
            Document document = saxReader.read(new File("pom.xml"));

            // 获取文件的中的根标签 名称
            Element rootElement = document.getRootElement();
            System.out.println("文件的跟标签是: " + rootElement.getName());
            System.out.println("文件的跟标签是: " + rootElement);

            // 获取属性值
            System.out.println(rootElement.attributeValue("xmlns"));

            // 获取根标签中的子标签集合
            List<Element> elements = rootElement.elements();
            // 遍历
            for (Element element : elements) {
                System.out.println(element.getName());

                if(element.getName().equals("dependencies")){
                    //之后再获取旗下的 子标签
                    List<Element> dependencies = element.elements();
                    dep = element;
                    for (Element de : dependencies) {
                        //获取再获取旗下的子标签
                        List<Element> elements2 = de.elements();
                        for (Element element1 : elements2) {
                            // 获取文本
                            System.out.println(element1.getText());
                        }
                    }
                }
            }
        } else {
            //之后再获取旗下的 子标签
        }
    }
}
```

```

        List<Element> elements1 = element.elements();
        for (Element element1 : elements1) {
//            获取文本
            System.out.println(element1.getText());
        }
    }

//            获取第一个 dependency
Element dependency = dep.element("dependency");
//            并且获取其中artifactId 标签的内容
System.out.printf( dependency.elementText("artifactId")); // junit

    } catch (DocumentException e) {
        throw new RuntimeException(e);
    }
}
}
}

```

## 常用方法:

1. `string getName();`返回标签的名称
2. `List<Element> elements();`获取标签的子标签
3. `string attributevalue(string name);`获取指定属性名称的属性值
4. `string getText();`获取标签的文本
5. `string elementText(String name);`获取指定名称的子标签的文本，返回子标签文本的值

## DOM4j 配合XPATH:

### 1. 引入依赖:

```
<dependency>
  <groupId>jaxen</groupId>
  <artifactId>jaxen</artifactId>
  <version>1.2.0</version>
</dependency>
```

Document/Element关于XPath的api

1. `Node selectSingleNode(String xpathExpression);`根据XPath表达式获取单个标签(元素/节点)
2. `List<Node> selectNodes(String xpathExpression);`根据XPath表达式获取多个标签(元素/节点)

### XPath的语法

·绝对路径方式方式, 以/开头的路径表示绝对路径, 绝对路径是从根元素开始写。例如/元素/子元素/子子元素...

使用正常方式:

```
//      获取第一个dependency 中的 artifactId 值
Element element = (Element) document.selectSingleNode("/project");
System.out.println(element); //org.dom4j.tree.DefaultElement@b7f23d9
[Element: <project uri: http://maven.apache.org/POM/4.0.0 attributes:
[org.dom4j.tree.DefaultAttribute@61d47554 [Attribute: name xsi:schemaLocation value
"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd"]]/>]

Element element2 = (Element)
document.selectSingleNode("/project/modelVersion");
System.out.println(element2); // null
```

我们发现获取不到其中的内容: 因为 project 是有一个默认的命名空间:

只有这样获取



```
// 设置默认命名空间
Namespace namespace = Namespace.get("http://maven.apache.org/POM/4.0.0");

// 创建XPath对象
XPath xpath =
DocumentHelper.createXPath("/ns:project/ns:dependencies/ns:dependency/ns:artifactId"
);

xpath.setNamespaceURIs(Collections.singletonMap("ns", namespace.getURI()));

// 获取第一个 dependency 中的 artifactId 值
Element element = (Element) xpath.selectSingleNode(document);
System.out.println(element.getText());
```

我们使用 `Namespace.get` 方法来创建命名空间对象，并使用 `DocumentHelper.createXPath` 创建XPath对象。然后，我们通过 `xpath.setNamespaceURIs` 方法将命名空间与XPath对象关联起来。最后，使用修正后的XPath对象来选择 `modelVersion` 元素。

## 向pom中添加一个元素：

```
//      获取根节点
Element project= (Element) document.selectSingleNode("/project");

//      向节点中添加一个元素
Element element1 = project.addElement("Ceshi",
"http://maven.apache.org/POM/4.0.0").addText("CeshiTest");
```

## 之后进行写入pom文件中：

```

    /**
     * 以下操作就是向 pom文件中写入
     */
    // 创建XMLWriter对象
    XMLWriter writer = new XMLWriter(new FileWriter(new File("pom.xml")));

    // 将Document对象写入文件
        writer.write(document);

    // 关闭XMLWriter
        writer.close();

```

## 完整 流程:

```

import org.dom4j.Document;
import org.dom4j.DocumentHelper;
import org.dom4j.Element;
import org.dom4j.Namespace;
import org.dom4j.QName;
import org.dom4j.io.OutputFormat;
import org.dom4j.io.XMLWriter;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class XMLGenerator {
    public static void XMLGenerator() {
        // 创建Document对象
        Document document = DocumentHelper.createDocument();

        // 创建根元素project, 并设置命名空间
        Element project = document.addElement("project");
        Namespace namespace = new Namespace("p",
"http://maven.apache.org/POM/4.0.0");
        Namespace xsiNamespace = new Namespace("xsi",
"http://www.w3.org/2001/XMLSchema-instance");
        project.add(namespace);
        project.add(xsiNamespace);

        // 设置xsi:schemaLocation属性
        QName schemaLocationQName = new QName("schemaLocation", xsiNamespace);
        project.addAttribute(schemaLocationQName, "http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd");

        // 添加子元素modelVersion
        Element modelVersion = project.addElement("modelVersion");

```

```
modelVersion.setText("4.0.0");

// 添加其他子元素和属性
project.addElement("groupId").setText("cloud.makaronbean.generate");
project.addElement("artifactId").setText("generate4j");
project.addElement("packaging").setText("pom");
project.addElement("version").setText("1.0");

// 创建parent元素并添加子元素
Element parent = project.addElement("parent");
parent.addElement("groupId").setText("org.springframework.boot");
parent.addElement("artifactId").setText("spring-boot-starter-parent");
parent.addElement("version").setText("2.3.6.RELEASE");

// 创建properties元素并添加子元素
Element properties = project.addElement("properties");
properties.addElement("maven.compiler.source").setText("8");
properties.addElement("maven.compiler.target").setText("8");
properties.addElement("project.build.sourceEncoding").setText("UTF-8");
properties.addElement("java.version").setText("1.8");

// 在<dependencies xmlns="">和</properties>之间添加间距
properties.addText("\n    ");

// 创建modules元素并添加子元素
Element modules = project.addElement("modules");
modules.addElement("module").setText(new
File(System.getProperty("user.dir")).getName());

// 在<dependencies xmlns="">和</properties>之间添加间距
modules.addText("\n    ");

// 创建dependencies元素并添加子元素
Element dependencies = project.addElement("dependencies");
Element dependency = dependencies.addElement("dependency");
dependency.addElement("groupId").setText("org.dom4j");
dependency.addElement("artifactId").setText("dom4j");
dependency.addElement("version").setText("2.1.3");

// 添加其他依赖项，省略代码

// 输出XML文件
try {
    // 创建XMLWriter对象，并设置输出格式
    OutputFormat format = OutputFormat.createPrettyPrint();
    format.setIndent("    "); // 设置缩进为4个空格
    format.setNewlines(true); // 设置换行符
    XMLWriter writer = new XMLWriter(new FileWriter("pom.xml"), format);

    // 输出到文件
    writer.write(document);
    writer.close();
}
```

```

        System.out.println("XML文件生成成功! ");
    } catch (IOException e) {
        System.out.println("生成XML文件时发生错误: " + e.getMessage());
    }
}
}

```

效果:

```

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>cloud.makeronbean.generate</groupId>
    <artifactId>generate4j</artifactId>
    <packaging>pom</packaging>
    <version>1.0</version>
    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <java.version>1.8</java.version>
    </properties>
    <modules>
        <module>untitled1</module>
    </modules>
    <dependencies>
        <dependency>
            <groupId>org.dom4j</groupId>
            <artifactId>dom4j</artifactId>
            <version>2.1.3</version>
        </dependency>
    </dependencies>
</project>

```

## 学习地址：

(4条消息) [【Dom4j】Dom4j完整教程详解 C\\_jx的博客-CSDN博客](#)

# SnakeYAML

在Java中，SnakeYAML是一个流行的YAML解析和生成库。它提供了一种简单而强大的方式来处理YAML数据。下面是一些使用SnakeYAML的常见用法示例：

1. 在Java中，SnakeYAML是一个流行的YAML解析和生成库。它提供了一种简单而强大的方式来处理YAML数据。下面是一些使用SnakeYAML的常见用法示例：

添加SnakeYAML依赖项到Maven项目中：

```
<dependency>
  <groupId>org.yaml</groupId>
  <artifactId>snakeyaml</artifactId>
  <version>1.29</version>
</dependency>
```

## 基本使用：

1. 将Java对象转换为YAML格式：

```
// 创建Java对象
Map<String, Object> data = new HashMap<>();
data.put("name", "John");
data.put("age", 30);
data.put("city", "New York");

// 创建Yaml对象并转换为YAML格式
Yaml yaml = new Yaml();
String yamlData = yaml.dump(data);
System.out.println(yamlData);
```

2. 从YAML格式加载数据并转换为Java对象：

```
// 从文件中加载YAML数据
try (FileInputStream inputStream = new FileInputStream("data.yaml")) {
    Yaml yaml = new Yaml();
    Map<String, Object> data = yaml.load(inputStream);
    System.out.println(data);
} catch (IOException e) {
    e.printStackTrace();
}
```

### 3. 将Java对象写入YAML文件:

```
// 创建Java对象
Map<String, Object> data = new HashMap<>();
data.put("name", "John");
data.put("age", 30);
data.put("city", "New York");

// 将Java对象写入YAML文件
try (FileWriter writer = new FileWriter("data.yaml")) {
    Yaml yaml = new Yaml();
    yaml.dump(data, writer);
} catch (IOException e) {
    e.printStackTrace();
}
```

## DumperOptions

DumperOptions是SnakeYAML库中的一个类，用于设置和控制YAML输出的选项。通过使用DumperOptions，你可以自定义生成的YAML文档的格式和样式。

以下是DumperOptions的一些常见用法：

#### 1. 设置缩进和缩进宽度：

```
DumperOptions options = new DumperOptions();
options.setIndent(4); // 设置缩进为4个空格
options.setDefaultFlowStyle(DumperOptions.FlowStyle.BLOCK); // 设置默认块样式
```

#### 2. 控制是否输出标量值引号：

```
options.setPrettyFlow(true); // 美化输出，标量值有引号
options.setDefaultScalarStyle(DumperOptions.ScalarStyle.DOUBLE_QUOTED); // 默认标量值使用双引号
```

### 3. 设置是否输出映射键的引号：

```
options.setExplicitStart(true); // 显式输出文档开始标记
options.setExplicitEnd(true); // 显式输出文档结束标记
```

### 4. 控制是否输出空行：

```
options.setCanonical(false); // 禁用规范输出
options.setPrettyFlow(true); // 输出美化的文档
options.setIndicatorIndent(2); // 设置标记的缩进宽度
```

### 5. 设置序列化顺序：

```
options.setSortKeys(true); // 按键排序输出
options.setFlowStyle(DumperOptions.FlowStyle.FLOW); // 使用流样式输出
```

## 解释模块意思

```
final DumperOptions options = new DumperOptions();

options.setDefaultFlowStyle(DumperOptions.FlowStyle.BLOCK);
options.setDefaultScalarStyle(DumperOptions.ScalarStyle.PLAIN);
```

在SnakeYAML中，`DumperOptions` 类用于设置和控制YAML输出的选项。`setDefaultFlowStyle()` 和 `setDefaultScalarStyle()` 是 `DumperOptions` 的两个方法，用于设置默认的流样式和标量值样式。

1. `setDefaultFlowStyle(DumperOptions.FlowStyle flowStyle)` 方法用于设置默认的流样式，即控制复杂数据结构（如映射和序列）在YAML输出中的显示方式。`FlowStyle` 是一个枚举类，它有三个选项：

- `FLOW`：使用流样式输出，将映射和序列包裹在大括号和方括号中。
- `BLOCK`：使用块样式输出，将映射和序列按照缩进和换行进行格式化，提高可读性。
- `AUTO`：自动选择流样式或块样式，默认选项。

示例：`options.setDefaultFlowStyle(DumperOptions.FlowStyle.BLOCK);`

2. `setDefaultScalarStyle(DumperOptions.ScalarStyle scalarStyle)` 方法用于设置默认的标量值样式，即控制字符串的引号和转义方式。`ScalarStyle` 是一个枚举类，它有四个选项：

- `PLAIN`：普通样式，不添加引号，不进行特殊字符的转义。
- `DOUBLE_QUOTED`：双引号样式，将字符串用双引号包裹，特殊字符进行转义。

- `SINGLE_QUOTED`：单引号样式，将字符串用单引号包裹，不进行转义。
- `LITERAL`：字面样式，使用竖线 `|` 表示换行，保留换行和缩进。

示例：`options.setDefaultScalarStyle(DumperOptions.ScalarStyle.PLAIN);`

通过设置 `DumperOptions` 中的这些选项，你可以控制SnakeYAML在生成YAML文档时的样式和格式。

## 以下是一个yaml 文件生成的流程：

```
// 导入所需的类
import org.yaml.snakeyaml.DumperOptions;
import org.yaml.snakeyaml.Yaml;

import java.io.Filewriter;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

public class YamlDemo2 {

    public static void main(String[] args) {

        // 创建Java对象并设置其属性：
        Map<String, Object> yamlData = new HashMap<>();
        yamlData.put("spring", new HashMap<String, Object>());
        yamlData.put("server", new HashMap<String, Object>());
        yamlData.put("mybatis-plus", new HashMap<String, Object>());

        Map<String, Object> spring = (Map<String, Object>) yamlData.get("spring");
        spring.put("application", new HashMap<String, Object>());
        spring.put("jackson", new HashMap<String, Object>());
        spring.put("mvc", new HashMap<String, Object>());
        spring.put("datasource", new HashMap<String, Object>());

        Map<String, Object> application = (Map<String, Object>)
spring.get("application");
        application.put("name", "untitled");

        Map<String, Object> jackson = (Map<String, Object>) spring.get("jackson");
        jackson.put("date-format", "yyyy-MM-dd");
        jackson.put("time-zone", "Asia/Shanghai");

        Map<String, Object> mvc = (Map<String, Object>) spring.get("mvc");
```



```

mvc.put("format", new HashMap<String, Object>());

Map<String, Object> format = (Map<String, Object>) mvc.get("format");
format.put("date", "yyyy-MM-dd");
format.put("date-time", "yyyy-MM-dd HH:mm:ss");

Map<String, Object> datasource = (Map<String, Object>)
spring.get("datasource");
datasource.put("password", "123456");
datasource.put("driver-class-name", "com.mysql.cj.jdbc.Driver");
datasource.put("url", "jdbc:mysql://localhost:3306/untitled?
useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai");
datasource.put("username", "root");

Map<String, Object> server = (Map<String, Object>) yamlData.get("server");
server.put("port", 8080);

Map<String, Object> mybatisPlus = (Map<String, Object>)
yamlData.get("mybatis-plus");
mybatisPlus.put("configuration", new HashMap<String, Object>());
mybatisPlus.put("global-config", new HashMap<String, Object>());

Map<String, Object> configuration = (Map<String, Object>)
mybatisPlus.get("configuration");
configuration.put("log-impl",
"org.apache.ibatis.logging.stdout.StdOutImpl");
configuration.put("map-underscore-to-camel-case", true);

Map<String, Object> globalConfig = (Map<String, Object>)
mybatisPlus.get("global-config");
globalConfig.put("db-config", new HashMap<String, Object>());

Map<String, Object> dbConfig = (Map<String, Object>) globalConfig.get("db-
config");
dbConfig.put("logic-not-delete-value", 0);
dbConfig.put("logic-delete-value", 1);
dbConfig.put("logic-delete-field", "isDelete");

globalConfig.put("type-aliases-package", "org.example.entity");

//      对文件的格式进行梳理
/**
 * DumperOptions是SnakeYAML库中的一个类，用于设置和控制YAML输出的选项。通过使用
DumperOptions，你可以自定义生成的YAML文档的格式和样式。
 */
final DumperOptions options = new DumperOptions();
options.setDefaultFlowStyle(DumperOptions.FlowStyle.BLOCK);
options.setDefaultScalarStyle(DumperOptions.ScalarStyle.PLAIN);

//      将Java对象转换为YAML格式并写入文件：
try (FileWriter writer = new FileWriter("data.yaml")) {

```

```

        Yaml yaml = new Yaml(options);
        yaml.dump(yamlData, writer);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}
}

```

效果:

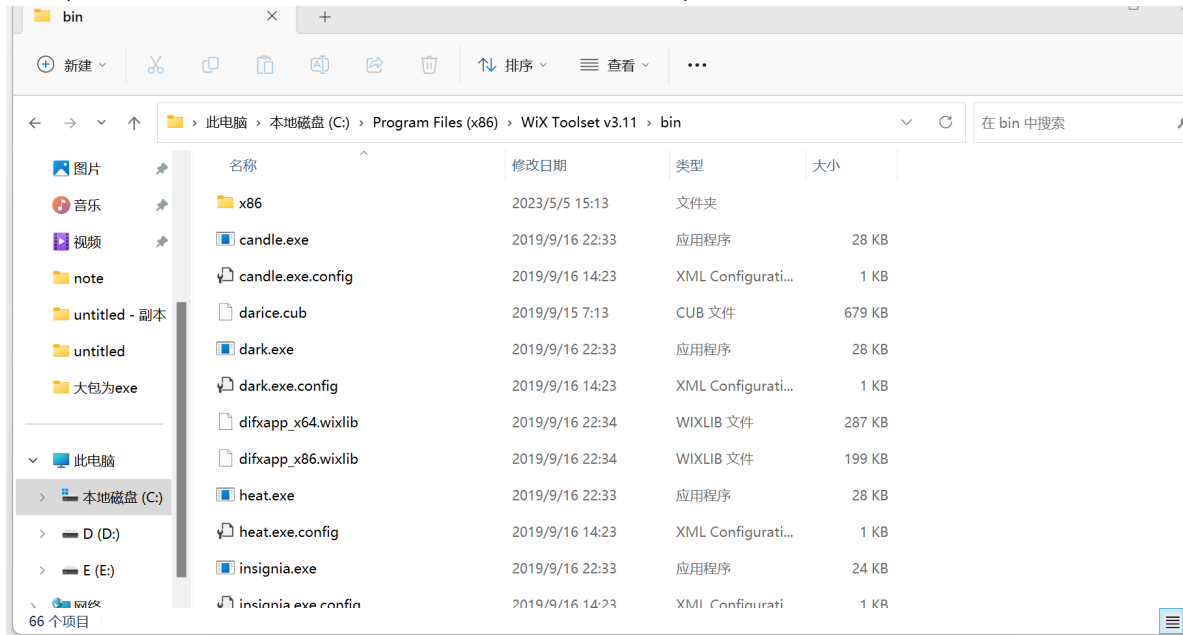
```

spring:
  jackson:
    date-format: yyyy-MM-dd
    time-zone: Asia/Shanghai
  application:
    name: untitled
  datasource:
    password: '123456'
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/untitled?useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai
    username: root
  mvc:
    format:
      date: yyyy-MM-dd
      date-time: yyyy-MM-dd HH:mm:ss
  server:
    port: 8080
  mybatis-plus:
    configuration:
      log-impl: org.apache.ibatis.logging.stdout.StdoutImpl
      map-underscore-to-camel-case: true
  global-config:
    db-config:
      logic-not-delete-value: 0
      logic-delete-value: 1
      logic-delete-field: isDelete
    type-aliases-package: org.example.entity

```

## 将springboot项目打包为 exe文件:

1. 首先我们的项目要使用jdk17或以上来进行打包
2. 之后下载wix 3版本或以上 地址 [Release WiX Toolset v3.11.2 · wixtoolset/wix3 \(github.com\)](https://github.com/wixtoolset/wix3)
3. 配置 path 安装时没有选择 一般在这里 把这个路径配置到path中



#### 4. 使用命令测试

```
candle.exe  
light.exe
```

#### 5. 执行编译：

```
jpackage --type exe --input <path_to_jar> --main-jar <jar_file> --name  
<output_name> --win-dir-chooser
```

1.
  - `<path_to_jar>`：指定你的Spring Boot应用程序的JAR文件路径。
  - `<jar_file>`：指定你的Spring Boot应用程序的主JAR文件名称。
  - `<output_name>`：指定输出的EXE文件的名称。

`--win-dir-chooser` 选项允许用户选择安装目录。

你可以根据需要添加其他选项，如指定输出目录、设置图标、添加应用程序描述等。

2. 执行 jpackage 命令：运行上述 jpackage 命令，它将使用 jpackage 工具将你的Spring Boot应用程序打包成一个可执行的EXE文件。

请注意，jpackage 命令的具体参数和选项可能因所使用的Java版本而有所不同。你可以使用 `jpackage --help` 命令查看可用的选项和语法。

eg：

```
jpackage --type exe --input E:\project\大包为exe\demo1\target --main-jar demo1-0.0.1-SNAPSHOT.jar --name MyAppInstaller --win-dir-chooser
```

在安装 wix 时会报错: WiX Toolset requires the .NET Framework 3.5.1 Windows feature to be enabled.

**解决:**

1. 打开"控制面板": 点击Windows开始菜单, 然后选择"控制面板"。
2. 打开"程序"或"程序和功能": 根据你的Windows版本, 选择"程序"或"程序和功能"。
3. 选择"打开或关闭Windows功能": 在"程序"或"程序和功能"窗口的左侧面板中, 你将找到"打开或关闭Windows功能"的选项。点击它。
4. 启用.NET Framework 3.5.1: 在"Windows功能"窗口中, 找到".NET Framework 3.5 (includes .NET 2.0 and 3.0)"的复选框。选中该复选框并点击"确定"按钮。
5. 安装.NET Framework 3.5.1: 启用.NET Framework 3.5.1后, 系统会自动下载和安装所需的文件。这可能需要一些时间, 因此请耐心等待。
6. 完成安装: 一旦安装完成, 你将收到一条成功的消息。

## 后续:

项目持续更新中: 此项目为第一个版本 只构建了 基本的多模块项目框架

当然对于后续 也就是慢慢 一些前端页面 使之达到一键生成一个maven的多模块的项目 包含前端后端, 无需动手创建或者拉取前端项目。

我的想法为随着版本的迭代, 可以添加更多的功能 比如 此次版本升级 改动为 添加一个人脸识别登录的模块:

## Mybatis:

### jdbc 测试数据库

使用 jdbc 测试数据库是否连接成功:

1. 导入依赖:

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.31</version>
</dependency>
```

2. 测试代码:

```

Connection conn = null;
try {
    // 加载 JDBC 驱动程序
    Class.forName("com.mysql.cj.jdbc.Driver");
    // 设置数据库连接
    String url = "jdbc:mysql://localhost:3306/guigu-oa";
    String user = "root";
    String password = "123456";
    conn = DriverManager.getConnection(url, user, password);
    System.out.println("数据库连接成功建立! ");
} catch (ClassNotFoundException e) {
    System.out.println("JDBC 驱动程序未找到");
    e.printStackTrace();
} catch (SQLException e) {
    System.out.println("数据库连接失败");
    e.printStackTrace();
}

```

## freeMarker的使用:

### 1. 导入 依赖:

```

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-freemarker</artifactId>
</dependency>

```

### 2. 创建模版文件 后缀名为 .ftl

```

package ${packageName};

public class ${className} {
    public static void main(String[] args) {
        // TODO: Add your code here
    }
}

```

### 3. 进行加载模板:

```

package com.ds.dserp;

import freemarker.template.Configuration;

```

```

import freemarker.template.Template;
import freemarker.template.TemplateException;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.Writer;
import java.util.HashMap;
import java.util.Map;

@SpringBootTest
class DsErpApplicationTests {

    @Test
    void contextLoads() throws IOException {
        Configuration cfg = new Configuration(Configuration.VERSION_2_3_31);
        cfg.setDirectoryForTemplateLoading(new
File("D:\\GonZuoShi\\Project\\ERP\\ds-erp\\src\\main\\resources"));

        Template template = cfg.getTemplate("FreeMarker.ftl");

        Map<String, Object> data = new HashMap<>();
        data.put("packageName", "com.example");
        data.put("className", "MyClass");

        try (Writer out = new FileWriter("D:\\GonZuoShi\\Project\\ERP\\ds-
erp\\src\\test\\java\\com\\ds\\dserp\\MyClass.java")) {
            template.process(data, out);
        } catch (TemplateException e) {
            throw new RuntimeException(e);
        }

    }

}

```

第二种加载模版文件的方式

使用这个好处就是不用填写绝对路径

```
cfg.setTemplateLoader(new ClassTemplateLoader(DsErpApplicationTests.class,  
"/service-util"));
```