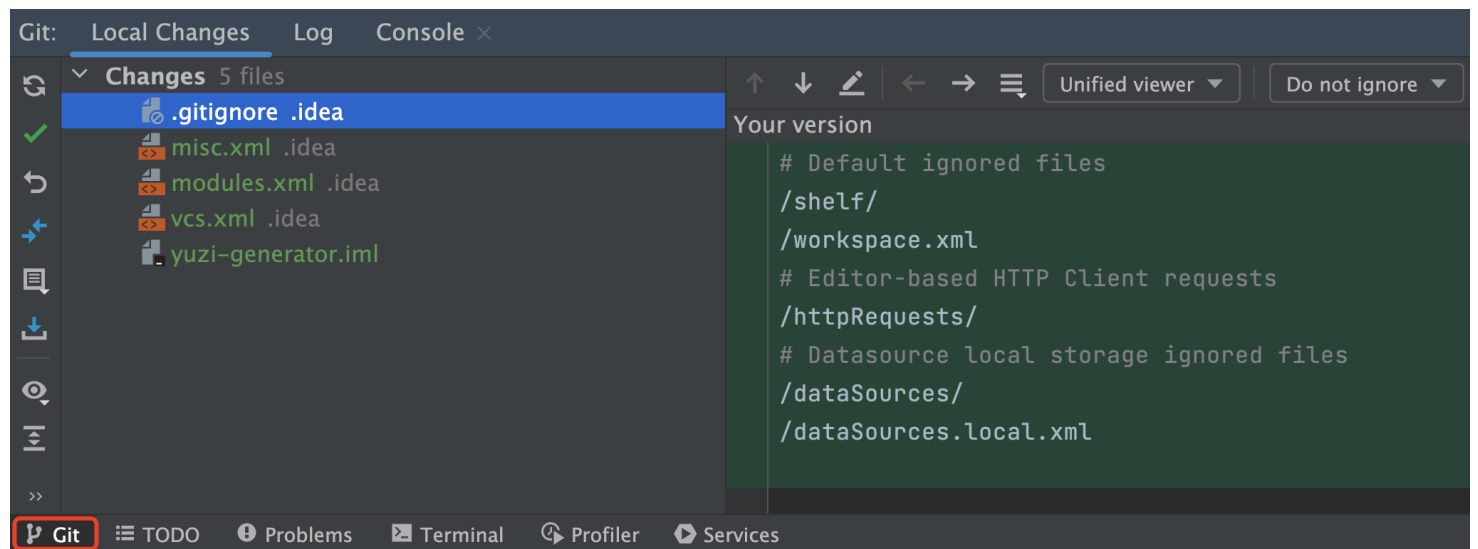


2、忽略无用提交

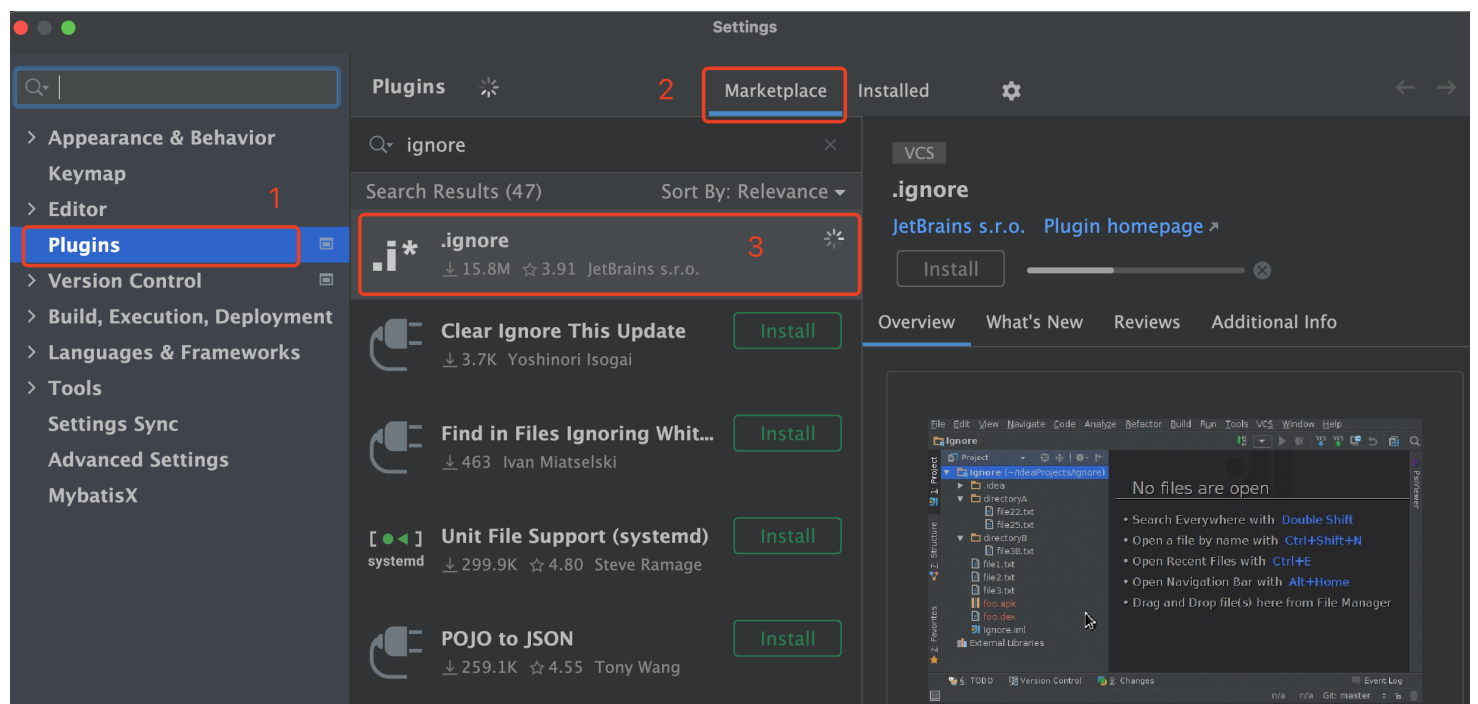
创建好新项目后，使用 IDEA 开发工具打开项目，进入底部的 **Git** 标签，会发现很多和项目无关的 IDEA 自动生成的工程文件被添加到了 Git 托管。



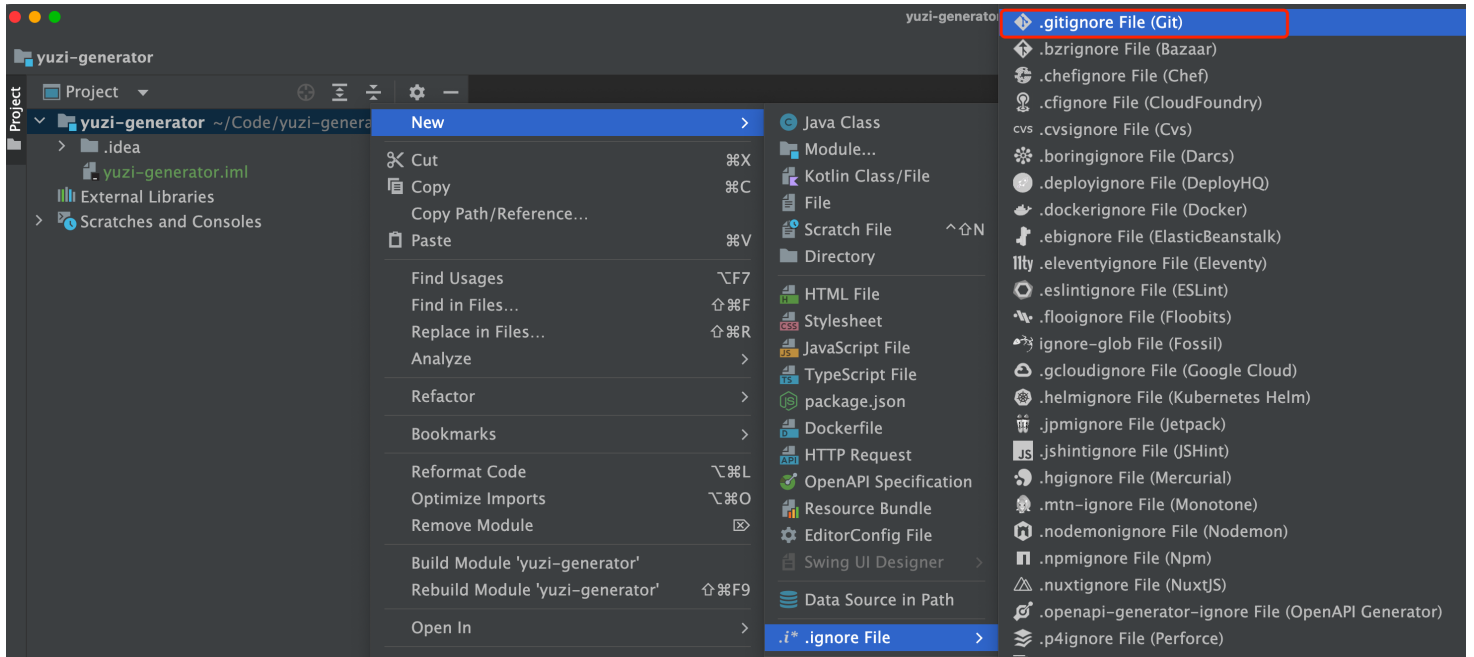
但我们是不希望提交这些文件的，没有意义，所以需要使用 **.gitignore** 文件来忽略这些文件，不让它们被 Git 托管。

如何编写 **.gitignore** 文件呢？

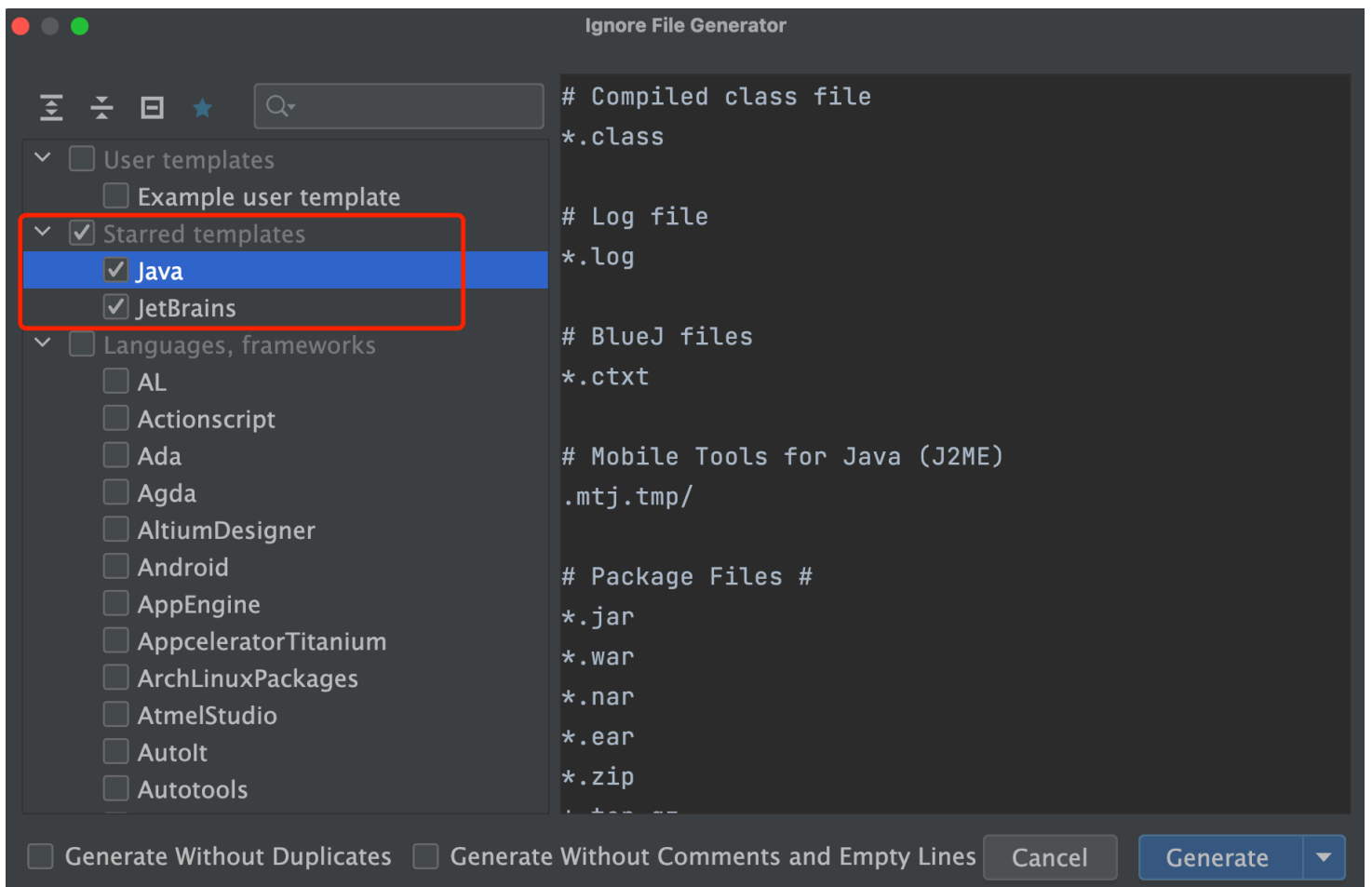
其实很简单，不用自己编写！我们在 IDEA 的 Settings => Plugins 中搜索 **.ignore** 插件并安装：



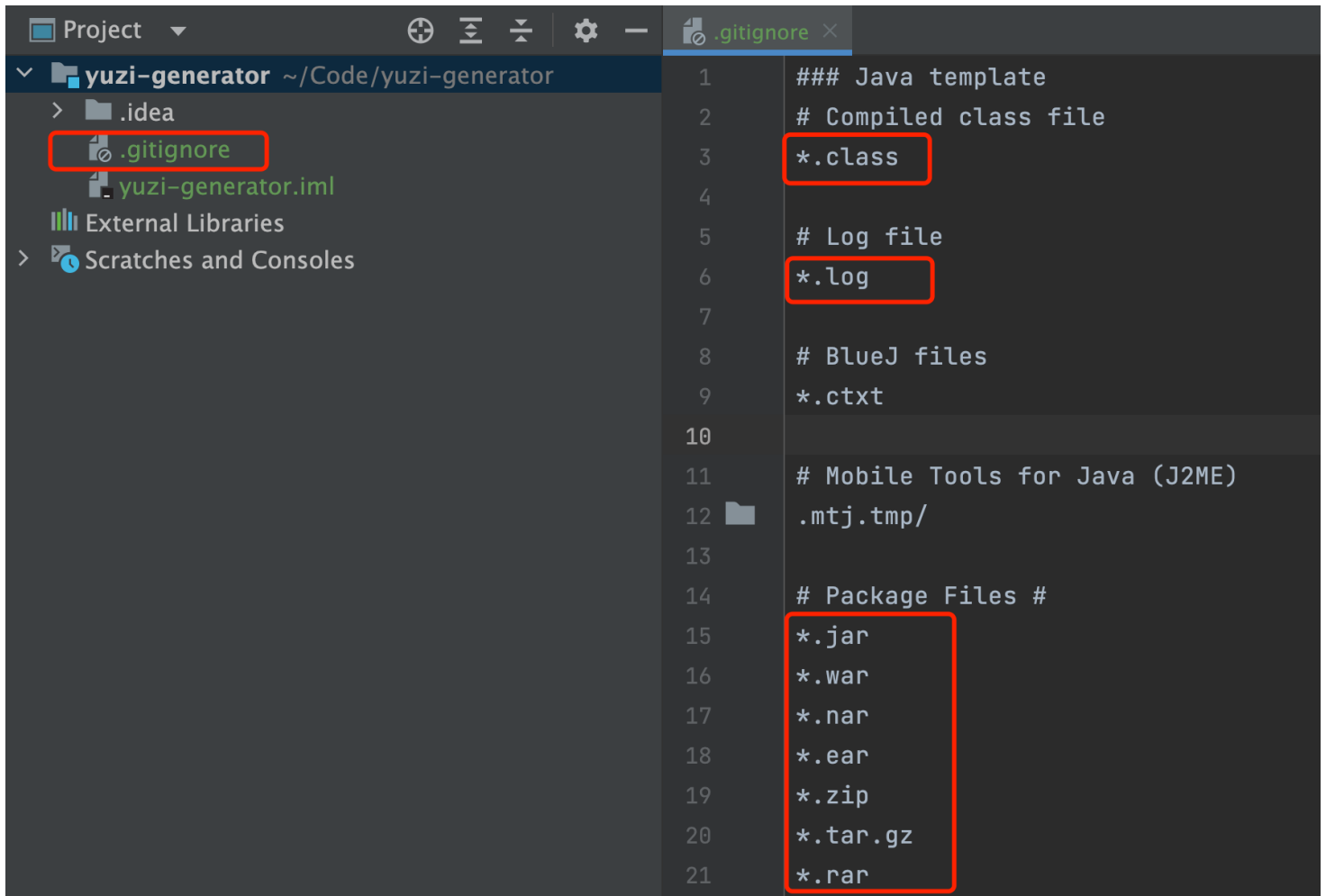
然后在项目根目录处选中右键，使用 **.ignore** 插件创建 **.gitignore** 文件：



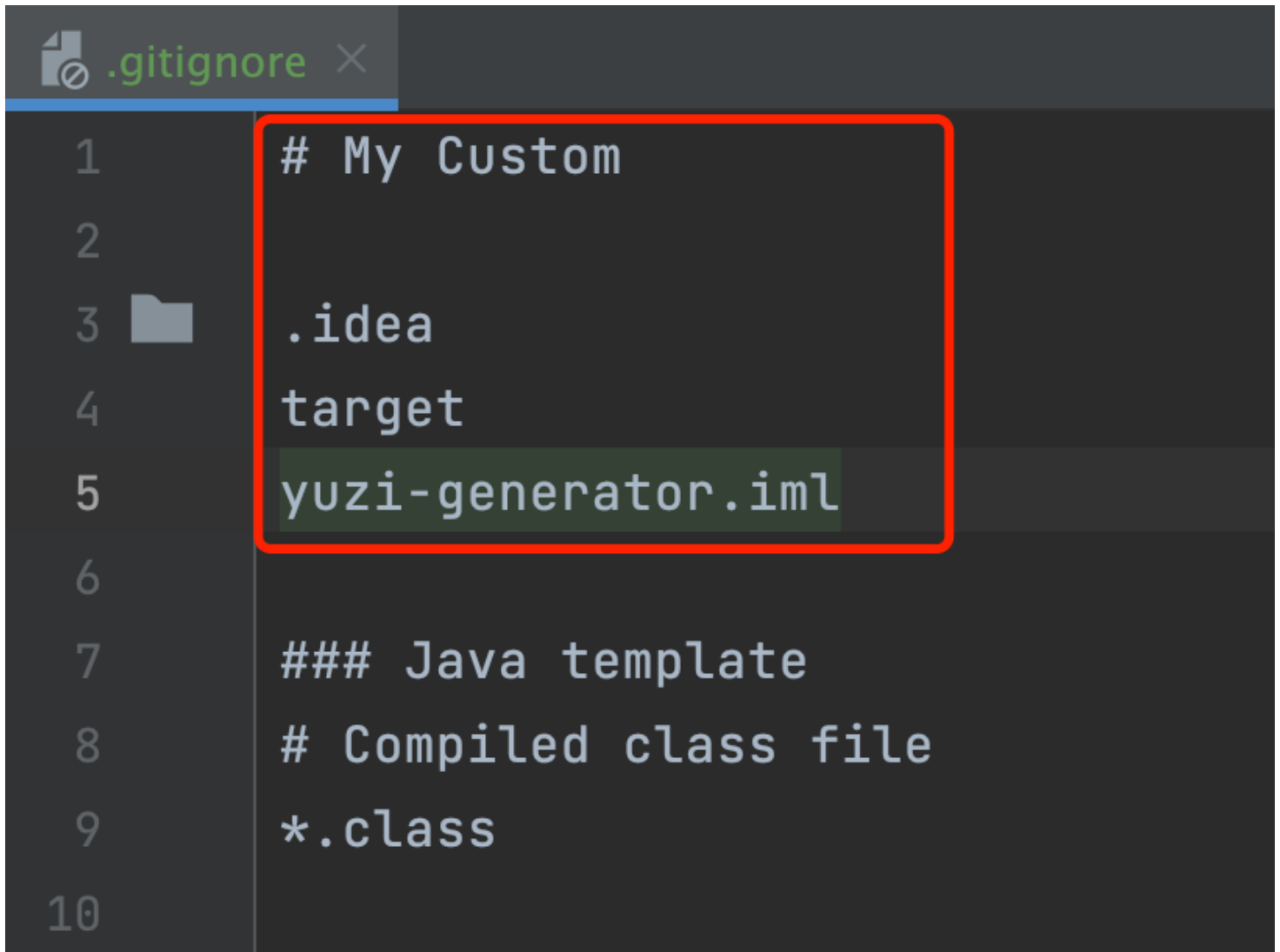
.ignore 插件提供了很多默认的 .gitignore 模板，根据自己的项目类型和使用的开发工具进行选择，此处我们选择 Java 和 JetBrains 模板：



然后可以在项目根目录看到生成的 .gitignore 文件，模板已经包含了常用的 Java 项目忽略清单，比如编译后的文件、日志文件、压缩包等：

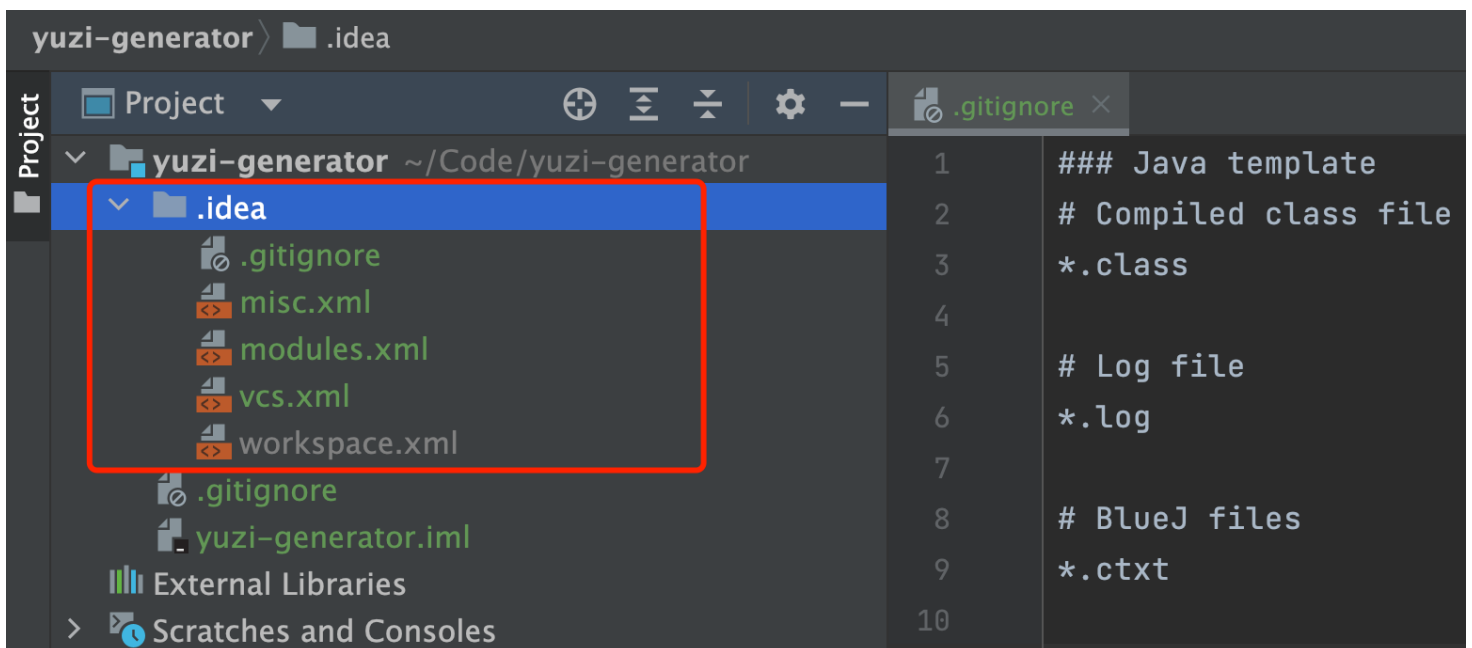


让我们再手动添加几个要忽略的目录和文件，比如打包生成的 target 目录：



```
1 # My Custom
2
3 .idea
4 target
5 yuzi-generator.iml
6
7 ### Java template
8 # Compiled class file
9 *.class
10
```

但是，我们会发现，即使有些文件已经添加到了 .gitignore 文件中，在 IDEA 中显示的还是绿色（已被 Git 托管）状态。如下图：

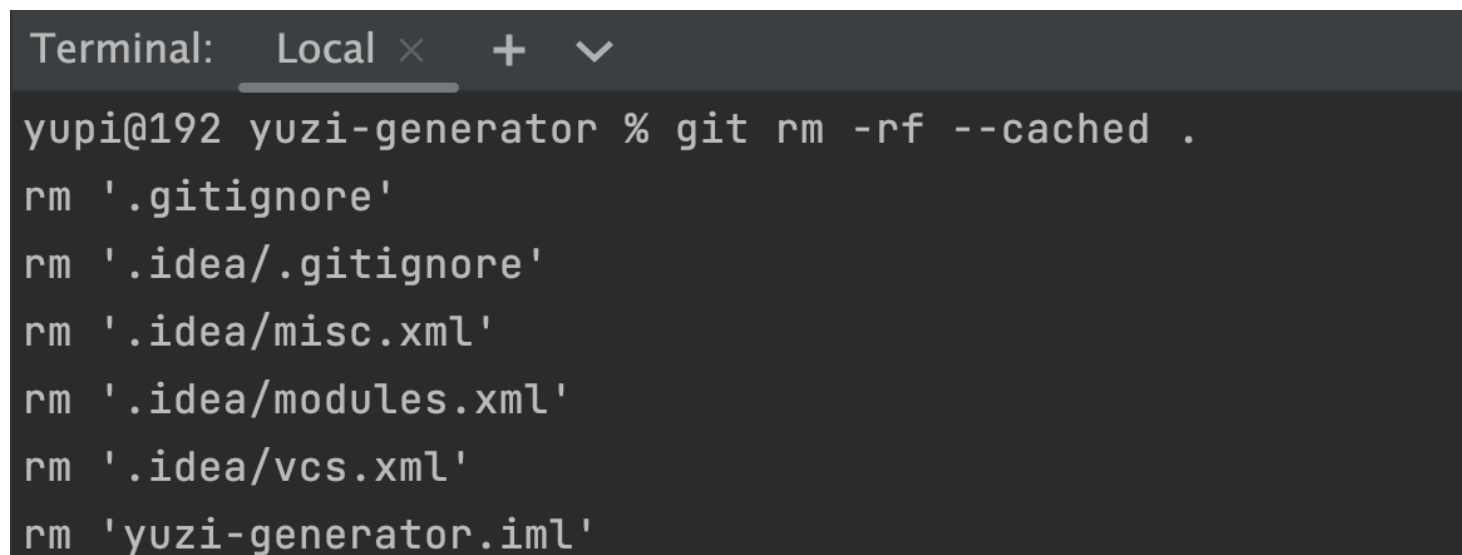


这是因为这些文件已经被 Git 跟踪。而 .gitignore 文件仅影响未跟踪的文件，如果文件已经被 Git 跟踪，那么 .gitignore 文件对它们没有影响。

所以我们需要打开终端，在项目根目录下执行如下命令，取消 Git 跟踪：

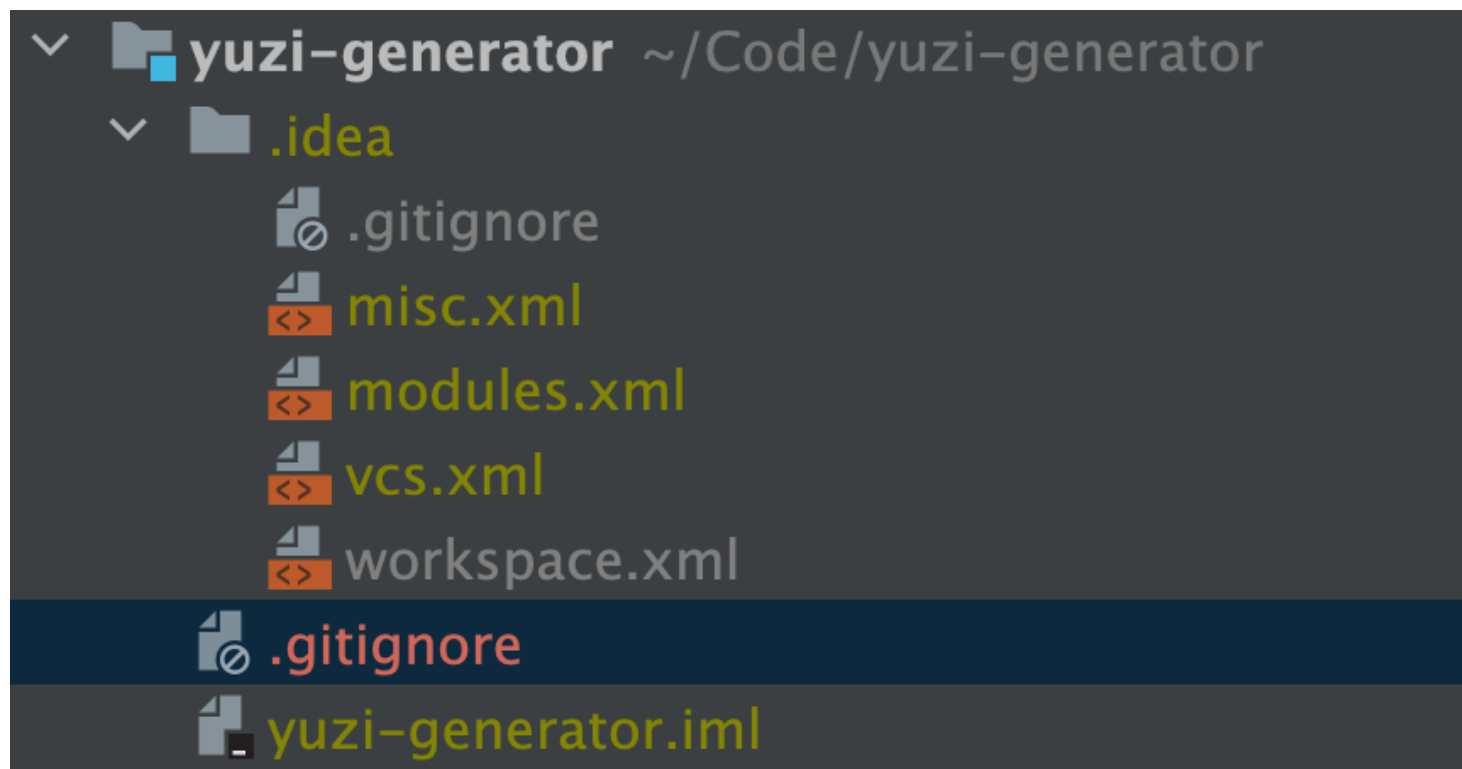
```
1 | shell复制代码git rm -rf --cached .
```

执行效果如图：

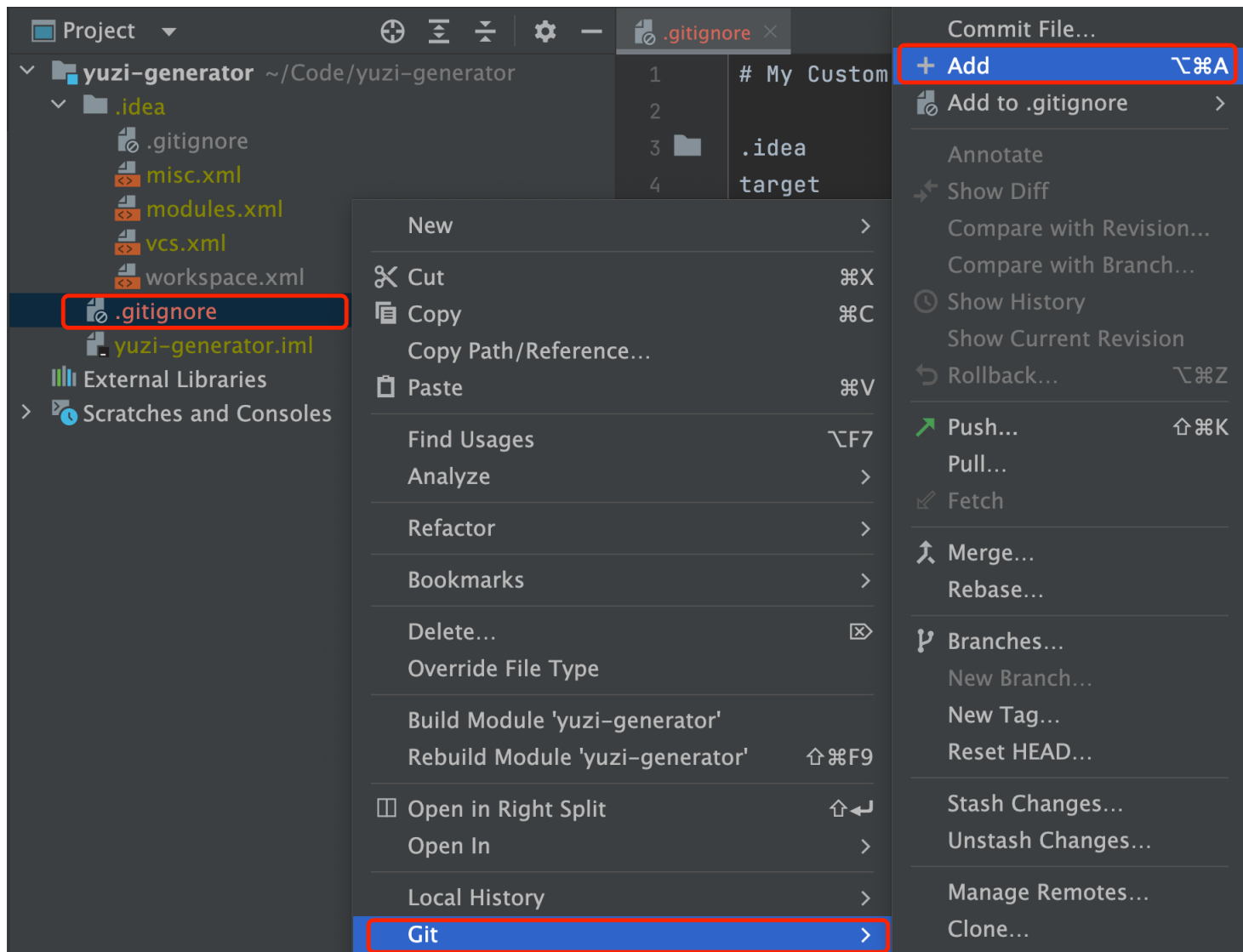


```
Terminal: Local x + v
yupi@192 yuzi-generator % git rm -rf --cached .
rm '.gitignore'
rm '.idea/.gitignore'
rm '.idea/misc.xml'
rm '.idea/modules.xml'
rm '.idea/vcs.xml'
rm 'yuzi-generator.iml'
```

可以看到文件变成了红色（未被 Git 托管）或黄色（被忽略）状态：



然后，让我们将 .gitignore 文件添加到 Git 暂存区，让它能够被 Git 管理。



项目根目录就初始化完成了，建议大家像鱼皮一样在项目根目录中新建一个 `README.md` 文件，用于介绍项目、记录自己的学习和开发过程等~



所以建议大家直接使用已有的 **模板引擎** 技术，轻松实现模板编写和动态内容生成。

五、FreeMarker 模板引擎入门

什么是模板引擎？为什么需要它？

模板引擎是一种用于生成动态内容的类库（或框架），通过将预定义的模板与特定数据合并，来生成最终的输出。

使用模板引擎有很多的优点，首先就是提供现成的模板文件语法和解析能力。开发者只要按照特定要求去编写模板文件，比如使用 `${参数}` 语法，模板引擎就能自动将参数注入到模板中，得到完整文件，不用再自己编写解析逻辑了。

其次，模板引擎可以将数据和模板分离，让不同的开发人员独立工作。比如后端专心开发业务逻辑提供数据，前端专心写模板等，让系统更易于维护。

此外，模板引擎可能还具有一些安全特性，比如防止跨站脚本攻击等。所以强烈大家掌握至少一种模板引擎的用法。

有很多现成的模板引擎技术，比如 Java 的 Thymeleaf、FreeMarker、Velocity，前端的 Mustache 等。

本项目中，我会以知名的、稳定的经典模板引擎 FreeMarker 为例，带大家掌握模板引擎的使用方法。

什么是 FreeMarker？

FreeMarker 是 Apache 的开源模板引擎，优点是入门简单、灵活易扩展。它不用和 Spring 开发框架、Servlet 环境、第三方依赖绑定，任何 Java 项目都可以使用。

我个人推荐的 FreeMarker 学习方式是直接阅读官方文档，虽然是英文的，但每一节基本都有代码示例，还是比较好理解的。

FreeMarker 官方文档：<https://freemarker.apache.org/docs/index.html>

DATA MODEL

```
(root)
|
+- mouse = "Yerri"
   |
   +- age = 12
      |
      +- color = "brown"
```

If you merge this template with the above data-model:

TEMPLATE

```
${mouse}      <!-- uses mouse as a string -->
${mouse.age}   <!-- uses mouse as a hash -->
${mouse.color} <!-- uses mouse as a hash -->
```

the output will be:

OUTPUT

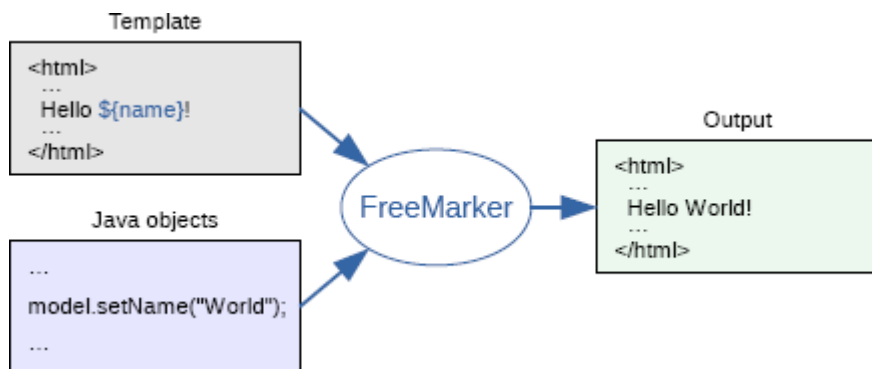
```
Yerri
12
brown
```

看不懂英文也没关系，鱼皮下面就带大家学习 FreeMarker，只讲常用的特性，主打一个快速入门！

模板引擎的作用

上面已经讲过了模板引擎的作用，这里就再用 FreeMarker 官网的一张图，强化下大家的理解。

如下图，FreeMarker 模板引擎的作用就是接受模板和 Java 对象，对它们进行处理，输出完整的内容。



下面我们先依次来学习 FreeMarker 的核心概念（模板和数据模型），然后通过一个 Demo 快速入门。

模板

FreeMarker 拥有自己的模板编写规则，一般用 FTL 表示 FreeMarker 模板语言。比如 `myweb.html.ftl` 就是一个 FreeMarker 的模板文件。

模板文件由 4 个核心部分组成：

- 1) 文本：固定的内容，会按原样输出。
- 2) 插值：用 `${...}` 语法来占位，尖括号中的内容在经过计算和替换后，才会输出。
- 3) FTL 指令：有点像 HTML 的标签语法，通过 `<#xxx ... >` 来实现各种特殊功能。比如 `<#list elements as element>` 实现循环输出。
- 4) 注释：和 HTML 注释类似，使用 `<#-- ... -->` 语法，注释中的内容不会输出。

让我们以《鱼皮官网》为例，举一个 FreeMarker 模板文件的例子：

学过前端开发框架的同学应该会觉得很眼熟~

```
1  html复制代码<!DOCTYPE html>
2  <html>
3    <head>
4      <title>鱼皮官网</title>
5    </head>
6    <body>
7      <h1>欢迎来到鱼皮官网</h1>
8      <ul>
9        <#-- 循环渲染导航条 -->
10       <#list menuItems as item>
11         <li><a href="${item.url}">${item.label}</a></li>
12       </#list>
13     </ul>
14     <#-- 底部版权信息（注释部分，不会被输出） -->
15     <footer>
16       ${currentYear} 鱼皮官网. All rights reserved.
17     </footer>
18   </body>
19 </html>
```

数据模型

我们把为模板准备的所有数据整体统称为 **数据模型**。

在 FreeMarker 中，数据模型一般是树形结构，可以是复杂的 Java 对象、也可以是 HashMap 等更通用的结构。

比如为上述《鱼皮官网》模板准备的数据模型，结构可能是这样的：

```
1 json复制代码{
2   "currentYear": 2023,
3   "menuItems": [
4     {
5       "url": "https://codefather.cn",
6       "label": "编程导航",
7     },
8     {
9       "url": "https://laoyujianli.com",
10      "label": "老鱼简历",
11    }
12  ]
13 }
14
```

Demo 实战

在了解模板和数据模型后，让我们通过 FreeMarker 对二者进行组合处理。

1、引入依赖

首先创建一个 Maven 项目（这里就用我们的 `yuzi-generator-basic` 项目），在 `pom.xml` 中引入 FreeMarker：

```
1 <!-- https://freemarker.apache.org/index.html -->
2 <dependency>
3   <groupId>org.freemarker</groupId>
4   <artifactId>freemarker</artifactId>
5   <version>2.3.32</version>
6 </dependency>
```

如果是 Spring Boot 项目的话，可以直接引入 starter 依赖：

```
1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-freemarker</artifactId>
4 </dependency>
```

2、创建配置对象

在 `test/java` 目录下新建一个单元测试类 `FreeMarkerTest`，在 `Test` 方法中创建一个 FreeMarker 的全局配置对象，可以统一指定模板文件所在的路径、模板文件的字符集等。

示例代码如下：

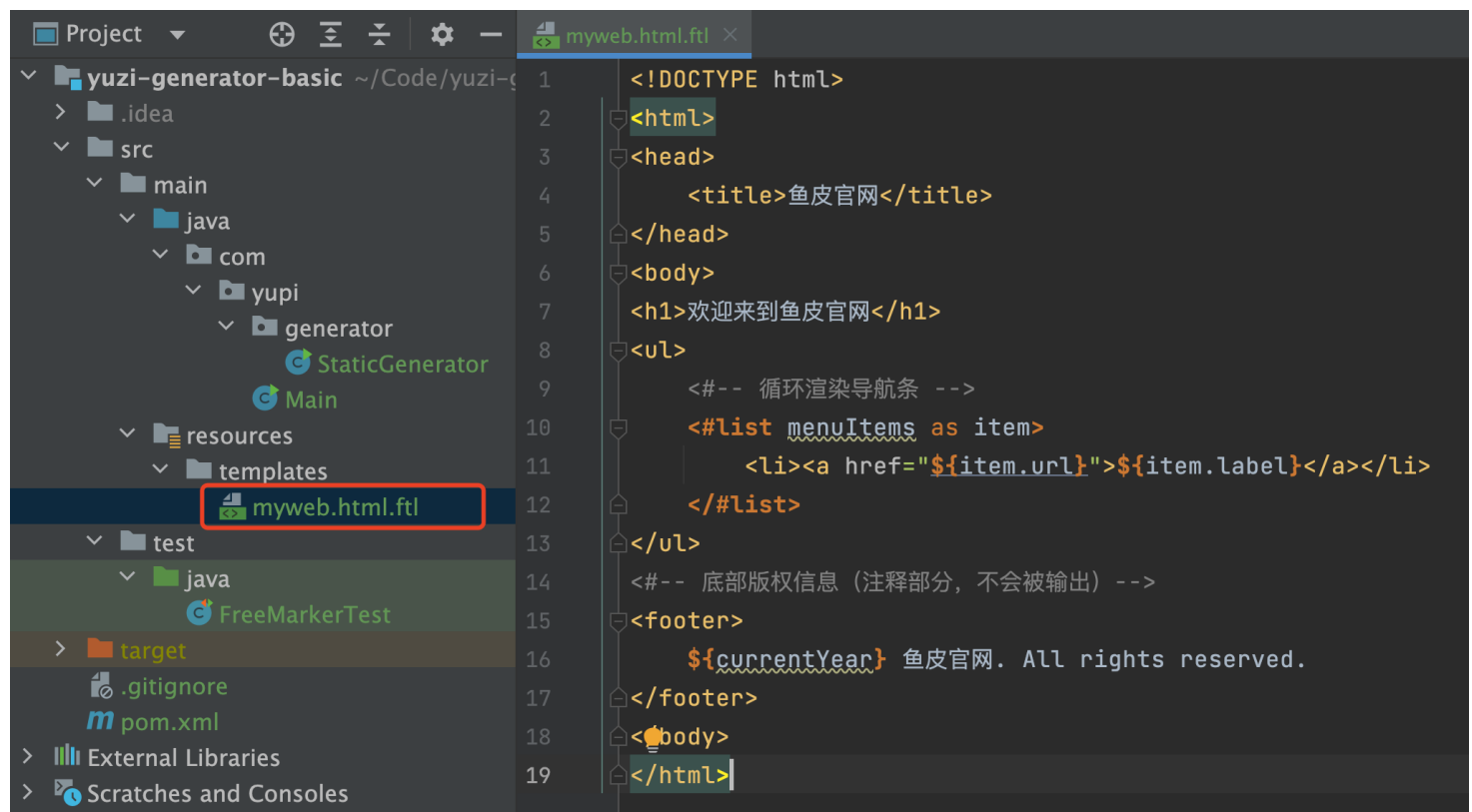
```

1 java复制代码// new 出 Configuration 对象, 参数为 FreeMarker 版本号
2 Configuration configuration = new Configuration(Configuration.VERSION_2_3_32);
3
4 // 指定模板文件所在的路径
5 configuration.setDirectoryForTemplateLoading(new File("src/main/resources/templates"));
6
7 // 设置模板文件使用的字符集
8 configuration.setDefaultEncoding("utf-8");

```

3、准备模版并加载

我们将上述《鱼皮官网》的模板代码保存为 `myweb.html.ftl` 文件，存放在上面指定的目录下。



准备好模板文件后，通过创建 Template 对象来加载该模板。示例代码如下：

```

1 java复制代码// 创建模板对象, 加载指定模板
2 Template template = configuration.getTemplate("myweb.html.ftl");

```

4、创建数据模型

如果想保证数据的质量和规范性，可以使用对象来保存“喂”给模板的数据；反之，如果想更灵活地构造数据模型，推荐使用 HashMap 结构。

比如我们想构造《鱼皮官网》的数据模型，需要制定当前年份和导航菜单项，示例代码如下：

```
1 java复制代码Map<String, Object> dataModel = new HashMap<>();
2 dataModel.put("currentYear", 2023);
3 List<Map<String, Object>> menuItems = new ArrayList<>();
4 Map<String, Object> menuItem1 = new HashMap<>();
5 menuItem1.put("url", "https://codefather.cn");
6 menuItem1.put("label", "编程导航");
7 Map<String, Object> menuItem2 = new HashMap<>();
8 menuItem2.put("url", "https://laoyujianli.com");
9 menuItem2.put("label", "老鱼简历");
10 menuItems.add(menuItem1);
11 menuItems.add(menuItem2);
12 dataModel.put("menuItems", menuItems);
```

5、指定生成的文件

可以直接使用 `FileWriter` 对象，指定生成的文件路径和名称：

```
1 java复制代码Writer out = new FileWriter("myweb.html");
```

6、生成文件

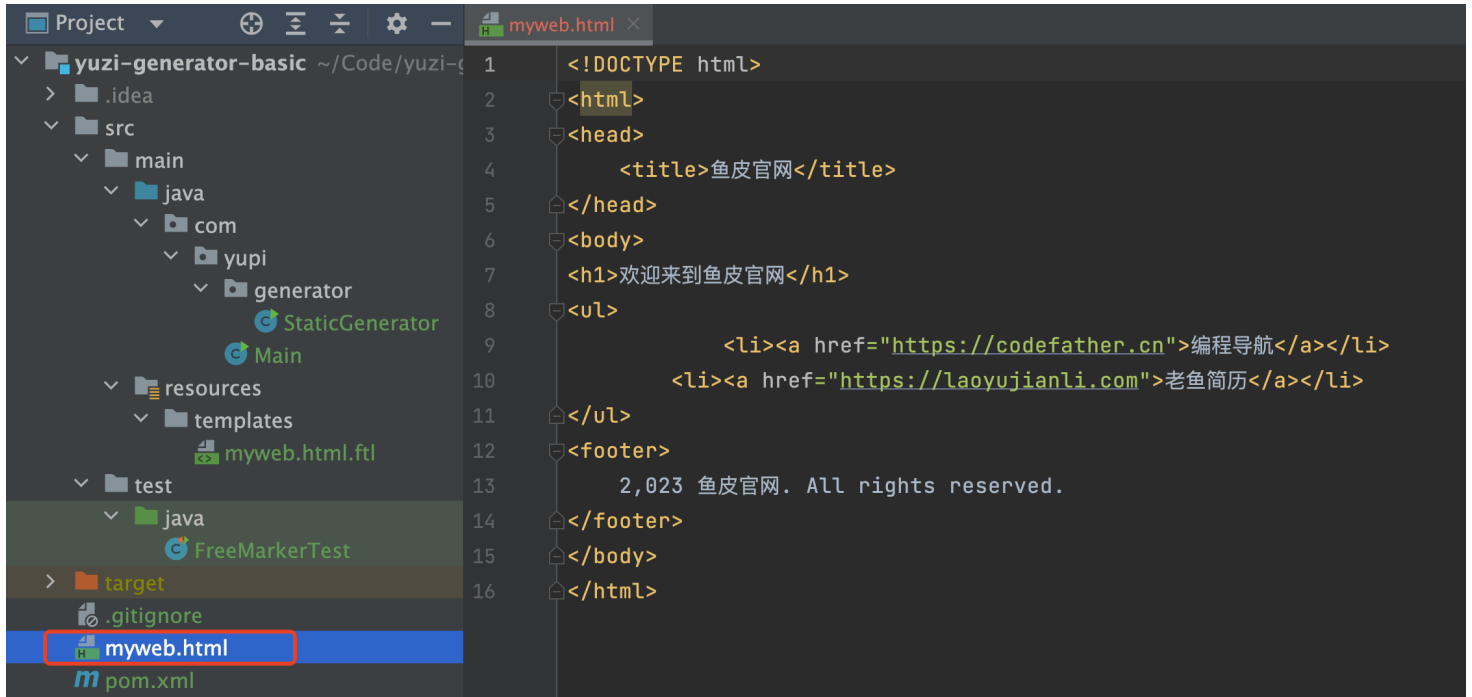
一切准备就绪，最后只需要调用 `template` 对象的 `process` 方法，就可以处理并生成文件了。

示例代码如下：

```
1 java复制代码template.process(dataModel, out);
2
3 // 生成文件后别忘了关闭哦
4 out.close();
5
```

7、完整代码

组合上面的所有代码并执行，发现在项目的根路径下生成了网页文件，至此 Demo 结束，很简单吧



FreeMarkerTest.java 文件的完整代码:

```
1  java复制代码import freemarker.template.Configuration;
2  import freemarker.template.Template;
3  import freemarker.template.TemplateException;
4  import org.junit.Test;
5
6  import java.io.File;
7  import java.io.FileWriter;
8  import java.io.IOException;
9  import java.io.Writer;
10 import java.util.ArrayList;
11 import java.util.HashMap;
12 import java.util.List;
13 import java.util.Map;
14
15 /**
16  * FreeMarker 学习测试
17  */
18 public class FreeMarkerTest {
19
20     @Test
21     public void test() throws IOException, TemplateException {
22         // new 出 Configuration 对象, 参数为 FreeMarker 版本号
23         Configuration configuration = new Configuration(Configuration.VERSION_2_3_32);
24
25         // 指定模板文件所在的路径
26         configuration.setDirectoryForTemplateLoading(new File("src/main/resources/templates"));
27
28         // 设置模板文件使用的字符集
29         configuration.setDefaultEncoding("utf-8");
30
31         // 创建模板对象, 加载指定模板
32         Template template = configuration.getTemplate("myweb.html.ftl");
```

```

33
34 // 创建数据模型
35 Map<String, Object> dataModel = new HashMap<>();
36 dataModel.put("currentYear", 2023);
37 List<Map<String, Object>> menuItems = new ArrayList<>();
38 Map<String, Object> menuItem1 = new HashMap<>();
39 menuItem1.put("url", "https://codefather.cn");
40 menuItem1.put("label", "编程导航");
41 Map<String, Object> menuItem2 = new HashMap<>();
42 menuItem2.put("url", "https://laoyujianli.com");
43 menuItem2.put("label", "老鱼简历");
44 menuItems.add(menuItem1);
45 menuItems.add(menuItem2);
46 dataModel.put("menuItems", menuItems);
47
48 // 生成
49 Writer out = new FileWriter("myweb.html");
50 template.process(dataModel, out);
51
52 // 生成文件后别忘了关闭哦
53 out.close();
54 }
55 }

```

常用语法

学会了 FreeMarker 的基本开发流程后，我们来学习一些 FreeMarker 中的实用特性。

注意，FreeMarker 的语法和特性非常多，本文仅带大家学习常用的、易用的语法。无需记忆，日后需要用到 FreeMarker 时，再去对照官方文档查漏补缺即可。

1、插值

在上面的 Demo 中，已经给大家演示了差值的基本语法（`${xxx}`）。但插值还有很多花样可以玩，比如支持传递表达式：

```
1 java复制代码表达式: ${100 + money}
```

不过个人不建议在模板文件中写表达式，为什么不在创建数据模型时就计算好要展示的值呢？

2、分支和判空

和程序开发一样，FreeMarker 模板也支持分支表达式（if ... else），示例代码如下：

```
1638649696178216961_0.6215895862791936
```

```

1 java复制代码<#if user == "鱼皮">
2   我是鱼皮
3 <#else>
4   我是猪皮
5 </#if>

```

分支语句的一个常用场景就是判空，比如要判断 user 参数是否存在，可以用下面的语法：

```
1 java复制代码<#if user??>
2     存在用户
3 <#else>
4     用户不存在
5 </#if>
```

3、默认值

FreeMarker 对变量的空值校验是很严格的，如果模板中某个对象为空，FreeMarker 将会报错而导致模板生成中断。

为了防止这个问题，建议给可能为空的参数都设置默认值。使用 `表达式!默认值` 的语法，示例代码如下：
1638649696178216961_0.6810562374889804

```
1 java复制代码${user!"用户为空"}
```

上述代码中，如果 user 对象为空，则会输出“用户为空”字符串。

4、循环

在上述 Demo 实战部分，已经给大家演示了循环的用法。即 `<#list items as item>` 表达式，可以遍历某个序列类型的参数并重复输出多条内容。

示例代码如下：

```
1 java复制代码<#list users as user>
2     ${user}
3 </#list>
```

其中，users 是整个列表，而 user 是遍历列表每个元素时临时存储的变量，跟 for 循环一样，会依次输出每个 user 的值。

5、宏定义

学过 C 语言和 C++ 的同学应该对“宏”这个词并不陌生。可以把“宏”理解为一个预定义的模板片段。支持给宏传入变量，来复用模板片段。

其实类似于前端开发中组件复用的思想。

在 FreeMarker 中，使用 `macro` 指令来定义宏。

让我们来定义一个宏，用于输出特定格式的用户昵称，比如：

```
1 java复制代码<#macro card userName>
2     -----
3     ${userName}
4     -----
5 </#macro>
```

其中，card 是宏的名称，userName 是宏接受的参数。

可以用 `@` 语法来使用宏，示例代码如下：

```
1 | java复制代码<@card userName="鱼皮"/>
2 | <@card userName="二黑"/>
```

实际生成的输出结果为：

```
1 | java复制代码-----
2 | 鱼皮
3 | -----
4 | -----
5 | 二黑
6 | -----
```

宏标签中支持嵌套内容，不过还是有些复杂的（再讲下去就成前端课了），大家需要用到时查看官方文档就好。

自定义指令：http://freemarker.foofun.cn/dgui_misc_userdefdir.html

6、内建函数

内建函数是 FreeMarker 为了提高开发者处理参数效率而提供的的语法糖，可以通过 `?` 来调用内建函数。

比如将字符串转为大写：

```
1 | java复制代码${userName?upper_case}
```

比如输出序列的长度：

```
1 | java复制代码${myList?size}
```

把内建函数想象成调用 Java 对象的方法，就很好理解了。

内建函数是 FreeMarker 非常强大的一个能力，比如想在循环语法中依次输出元素的下标，就可以使用循环表达式自带的 `index` 内建函数：

```
1 | java复制代码<#list users as user>
2 |     ${user?index}
3 | </#list>
```

内建函数种类丰富、数量极多，因此不建议大家记忆，需要用到时去查阅官方文档即可。

1638649696178216961_0.9047355195566265

内建函数大全参考：http://freemarker.foofun.cn/ref_builtins.html

7、其他

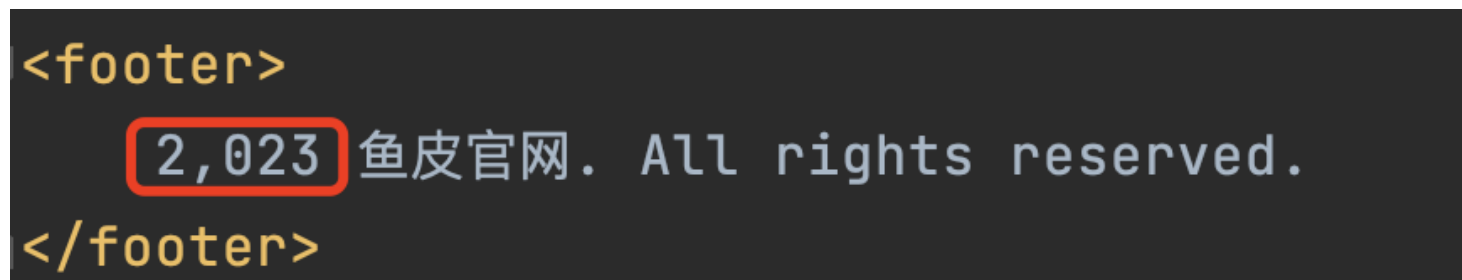
还有更多特性，比如命名空间，其实就相当于 Java 中的包，用于隔离代码、宏、变量等。

1638649696178216961_0.6023426584063643

不过没必要细讲，因为掌握上述常用语法后，基本就能够开发大多数模板文件了。更多内容自主查阅官方文档学习即可。

问题解决示例

给大家分享一个通过查阅官方文档解决具体问题的例子，比如之前生成的网站文件中，我们发现数字中间加了一个逗号分割符，如下图：1638649696178216961_0.56512427626304



这是因为 FreeMarker 使用 Java 平台的本地化敏感的数字格式信息，如果想把分割符取消掉，怎么办呢？

我们可以通过查阅官方文档看到以下信息：1638649696178216961_0.17661632955130657

地址：http://freemarker.foofun.cn/app_faq.html#faq_number_grouping

1638649696178216961_0.10777421576869828

3. 为什么 FreeMarker 打印奇怪的数字格式 (比如 1,000,000 或 1 000 000 而不是 1000000)?

FreeMarker 使用 Java 平台的本地化敏感的数字格式信息。默认的本地化数字格式可能是分组或其他不想要的格式。为了避免这种情况，你不得使用 [FreeMarker 设置](#) 中的 `number_format` 来重写 Java 平台建议的数字格式，比如：

```
cfg.setNumberFormat("0.#####"); // now it will print 1000000
// where cfg is a freemarker.template.Configuration object
```

请注意，人们通常在没有分组分隔符时阅读大数是有些困难的。所以通常建议保留分隔符，而在对“计算机”处理时的情况(分组分隔符会使其混乱)，就要使用 [c 内建函数](#) 了。比如：

TEMPLATE

```
<a href="/shop/productdetails?id=${product.id?c}">Details...</a>
```

按照文档的提示，修改 configuration 配置类的 `number_format` 设置，即可调整默认生成的数字格式啦。

1638649696178216961_0.7802647282930741

更多学习资源

官方文档：<https://freemarker.apache.org/docs/index.html>

中文教程：http://freemarker.foofun.cn/toc.html1638649696178216961_0.003273947627542295

FreeMarker 教程网：<http://www.freemarker.net/>