### 5.3.2 Business Microservice Generation Template

The code for the business microservice involves generating code and configuration files for the business microservice project. It mainly implements the generation of web service code under the Spring Boot framework and incorporates various technical frameworks of the Spring Cloud Alibaba microservice framework for the collaborative work of different microservices. The templates within the business microservice include: Spring Boot starter class template, microservice client call template, microservice client call fallback template, controller class template, data access class template, data entity class template, general subordinate class template, pom file templates, and application configuration file template. For clarity, based on the roles of instructions and variables in all templates within the business microservice, the code is classified into common class code, interface code, service pom file, and configuration file.

##### 5.3.2.1 Class Code

The class code includes dependencies, methods, properties, and related framework annotations within the class. Various parts of the class code will be used in templates to generate the basic components of the class. Figure 5-4 shows the class code template file, where the blue font represents the FTL template language and variable definitions. These variables are rendered into the attribute values of the middle layer data structure in the template engine. Specifically, (1) the first line sets the package name for the class; (2) lines 3-5 set the class dependencies, including the generated code and specific technology frameworks; (3) lines 9-25 generate properties, including basic Java properties and objects. For basic data properties, the code is assigned directly, while for object properties, the "@Autowired" annotation may be added automatically to fetch relevant Bean objects from the Spring container; (4) lines 27-38 generate the methods of the class. Since the current model cannot define the method logic, the code for the method body cannot be generated automatically and is instead filled with the comment "//TODO"; (5) to adapt to various technology frameworks, annotations will also be added in appropriate locations, such as lines 7 and 15.

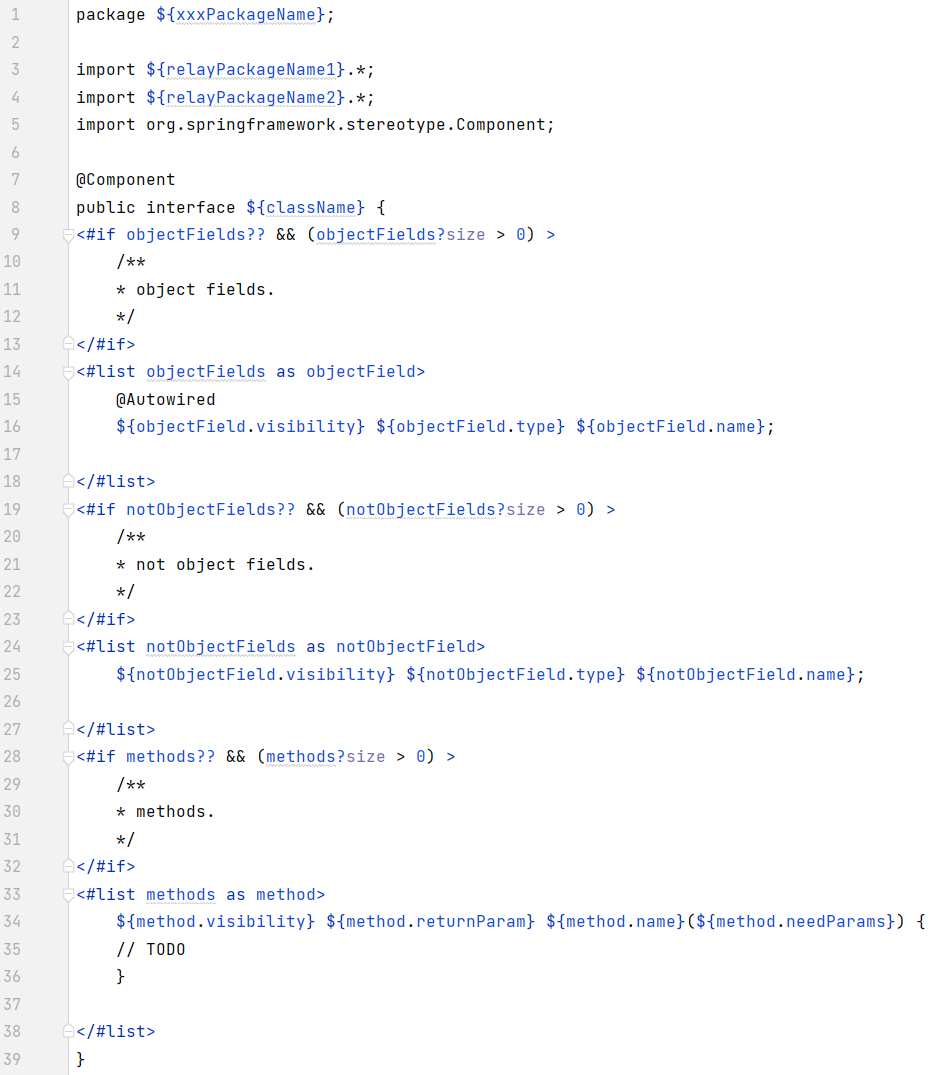


Fig 5-4 Class code template

##### 5.3.2.2 Client Code

The client code includes the dependencies of the client interface and automatically configures the service fallback handling. Figure 5-5 shows the interface code template file. The specific content is as follows: (1) Lines 1-2 specify the target service to be called and the class for fallback handling; (2) Lines 4-12 generate the client interface, which by default accesses the service via a GET request; (3) Lines 15-27 automatically generate the service fallback class, which is completed by the client's own service. The method body is filled with the comment "//TODO".

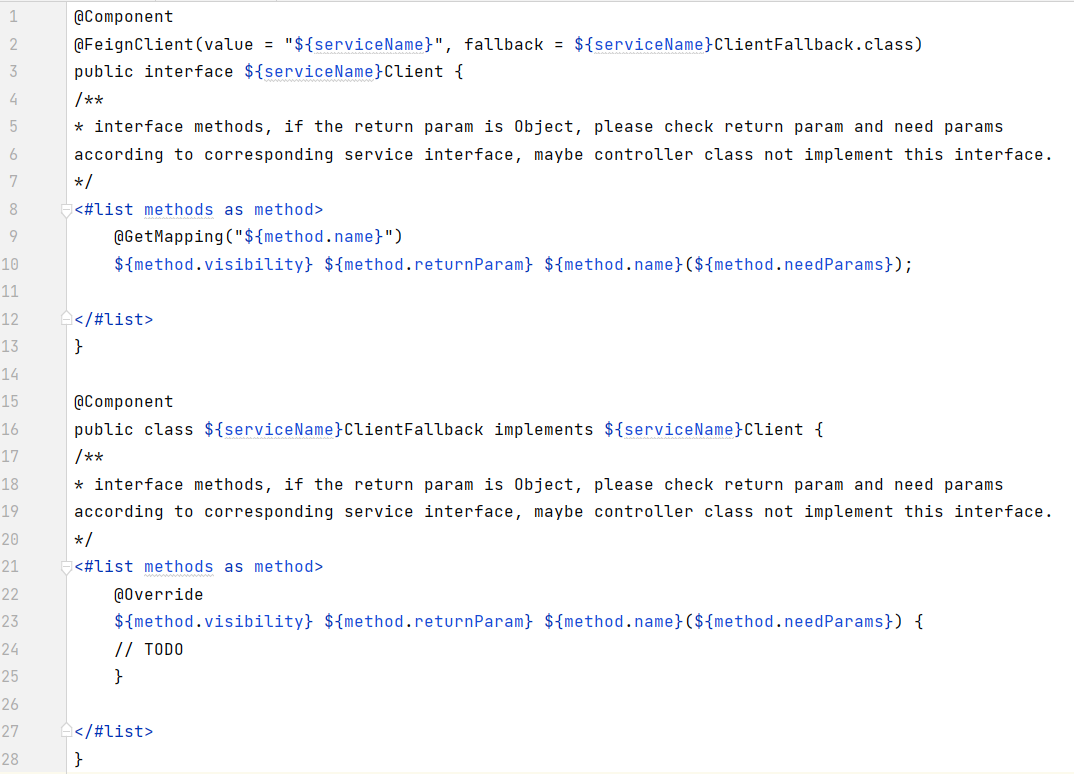


Fig 5-5 Client code template

##### 5.3.2.3 Interface Code

The interface code refers to the code of the controller class. In addition to using the class code template to generate the components of the class, it also includes code generation for the interface part, as shown in figure 5-6. (1) Lines 1-2 indicate that the class is an interface class, and the result returned by the class will be serialised; (2) Lines 4-15 define the interface, which includes methods from the controller class itself and methods identified from the subsidiary class that need to be promoted to the controller class. By default, all interfaces are accessed via the GET method, with the method name used as the access path. The function method body cannot be automatically generated and is filled with the comment "//TODO".

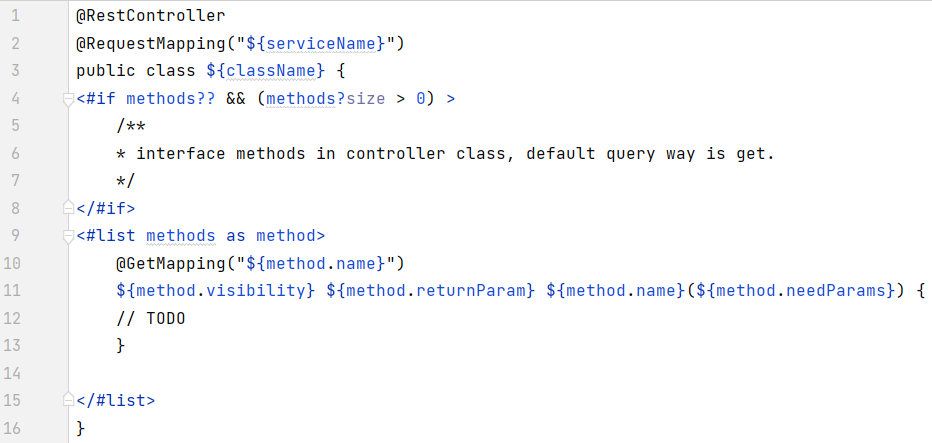


Fig 5-6 Interface code template

##### 5.3.2.4 Service Pom File

The pom file is an XML file that contains basic information about a project, describing how the project is built and declaring project dependencies, among other things. The pom file template for a service is used to generate technical framework dependencies and project information related to building a single microservice project. It mainly includes fixed dependency management and a project's building-related content.

##### 5.3.2.5 Configuration File

The configuration file contains the configuration for this microservice, as shown in Figure 5-7, which mainly includes: (1) Lines 1-2 specify the port on which the service is started; (2) Lines 3-18 contain Spring-related configurations for the service, including the service name, middleware dependencies, and database configuration; (3) Lines 19-22 contain the configuration for the commonly used database access framework, MyBatis; (4) Lines 23-29 contain the configuration for the client that calls other microservice interfaces.

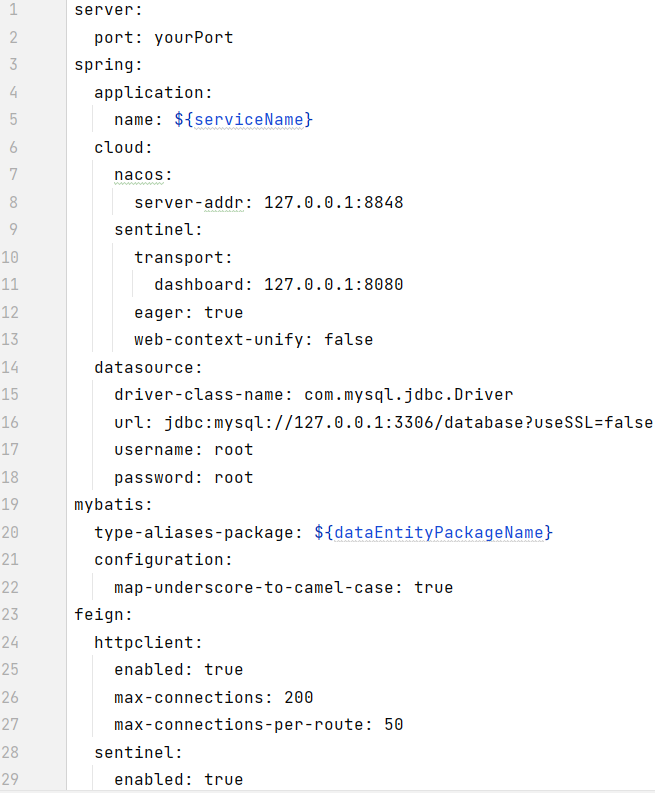


Fig 5-7 Microservice config template file

### 5.3.3 Infrastructure Service Generation Template

Infrastructure services involve generating code and configuration files for infrastructure-related components, which include the Gateway project and dependency management configuration for the entire microservice system. With the exception of the Gateway project, other infrastructure services are third-party services that can be run directly without generating code. Their related configuration templates have already been defined in the business microservice template. The template files for infrastructure services include the Gateway startup code template, the Gateway pom file template, the configuration template under the Gateway project, and the pom file template for the microservice system.

##### 5.3.3.1 Microservice System Pom File

The pom file for the microservice system contains basic information about the system, build methods, and dependency management (group, versions, etc.) for the entire microservice system. It is used to manage all microservice projects, mainly for dependency transfer and convenient packaging, and serves as the parent configuration for each microservice project's pom file. The microservice system pom file template is similar to the service pom file template, as it mostly consists of fixed dependency management and system build-related content.

##### 5.3.3.2 Gateway Project

The Gateway project template consists mainly of three parts: the dependency management pom file, the configuration management file, and the starter class code file. The dependency management pom file is similar to the pom files introduced earlier and contains fixed dependency management and build-related content. The configuration management file contains the access configuration of the gateway for all services and other middleware dependencies. Figure 5-8 shows the definition of the configuration template under the Gateway project: (1) Lines 1-8 define the gateway port, service registration centre name, and Nacos communication socket; (2) Lines 11-16 configure the routing of each business microservice so that the correct service routing address can be accessed after a successful match. The starter class code file includes the main startup class for starting this gateway. This part of the content is the fixed bootstrap code for starting the Spring Boot application, as shown in Figure 5-9.



Fig 5-8 Config template file of Gateway project

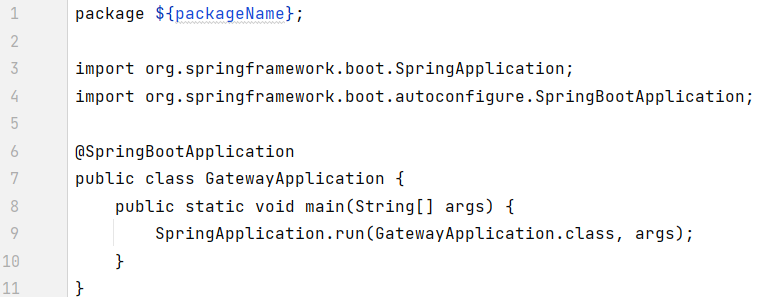


Fig 5-9 Bootstrap class template file of Gateway project

### 5.3.4 Test Code Generation Template

Microservices are designed to be relatively independent and encapsulate specific functions through service interfaces, which define necessary information such as service requirements, functions, parameters, and communication protocols. Testing of microservices is primarily based on black-box testing of service interfaces and is the main basis for ensuring that a system is running correctly. The microservice code generated by the method presented in this paper provides microservice function calls through web interfaces, and the testing code includes unit testing of all web interfaces of the microservice. The purpose of unit testing the interfaces is to ensure that the microservice's internals can run correctly and provide stable and correct results.

The testing code generation template includes a controller class interface testing template, so the code generated through this template already contains the process of interface testing. Users only need to design their own test cases and fill in the relevant parameters to complete the testing. The testing frameworks used in the template include Junit 5 and Mockito, which are used for unit testing and mocking objects to simulate their behaviour (such as simulating the calls of other dependent microservices during testing without the need for a real integration testing environment). Figure 5-10 shows the definition of the controller class interface testing template. The content before line 17 is the definition of the dependency framework used for testing. Lines 17 to 24 define the objects that the controller class may need to mock during testing, while lines 25 to 33 define the objects that do not need to be mocked. Lines 34 to 41 define the initialization logic before testing the controller class, and lines 43 to 70 define the test code for all interface methods in the controller class, including the initialization of interface input parameters, initialization of expected return results, calling the interface, and comparing actual running results with expected results.



Fig 5-10 Controller class interface test template