

OpenAPI Generator Gradle Plugin 시작하기

목표 [↗](#)

OpenAPI Generator Gradle Plugin 과 OpenAPI 3.0 형식의 명세서를 사용해서 Controller Interface 와 Model Object 를 추출하여 사용합니다!

준비 [↗](#)

- Spring boot 프로젝트를 생성합니다~
- 아래의 첨부파일과 같은 OpenAPI 3.0 스펙의 YAML 파일을 준비합니다~



build.gradle 수정 [↗](#)

build.gradle 전체 예시 [↗](#)

```
1  plugins {
2      id 'java'
3      id 'org.springframework.boot' version '3.1.2'
4      id 'io.spring.dependency-management' version '1.1.2'
5      // 추가
6      id "org.openapi.generator" version "6.6.0"
7  }
8
9  group = 'com.example'
10 version = '0.0.1-SNAPSHOT'
11
12 java {
13     sourceCompatibility = JavaVersion.VERSION_17
14 }
15
16 configurations {
17     compileOnly {
18         extendsFrom annotationProcessor
19     }
20 }
21
22 repositories {
23     mavenCentral()
24 }
25
26 dependencies {
27     // 추가
28     implementation 'org.springframework.boot:spring-boot-starter-web'
```

```

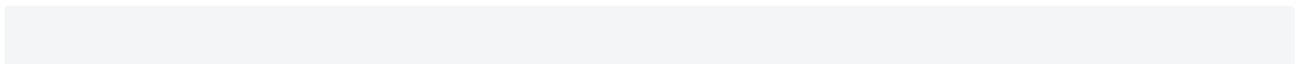
29     implementation 'org.springdoc:springdoc-openapi-starter-webmvc-ui:2.2.0'
30     implementation('org.openapitools:openapi-generator:6.6.0') {
31         exclude group: 'org.slf4j', module: 'slf4j-simple'
32     }
33     implementation 'org.openapitools:jackson-databind-nullable:0.2.6'
34     implementation 'org.springframework.boot:spring-boot-starter-validation'
35
36     compileOnly 'org.projectlombok:lombok'
37     developmentOnly 'org.springframework.boot:spring-boot-devtools'
38     annotationProcessor 'org.springframework.boot:spring-boot-configuration-processor'
39     annotationProcessor 'org.projectlombok:lombok'
40     testImplementation 'org.springframework.boot:spring-boot-starter-test'
41 }
42
43 tasks.named('test') {
44     useJUnitPlatform()
45 }
46
47 // 추가
48 openApiGenerate {
49     generatorName.set("spring")
50     verbose.set(true)
51     inputSpec.set("$rootDir/src/main/resources/static/simple-api.yaml")
52     outputDir.set("$buildDir/generate-resources")
53     apiPackage.set("com.example.openapigen.api")
54     invokerPackage.set("com.example.openapigen.invoker")
55     modelPackage.set("com.example.openapigen.model")
56     configOptions.set([
57         dateLibrary      : "java8",
58         library           : "spring-boot", //spring-boot, spring-mvc, spring-cloud
59         interfaceOnly    : "true",
60         unhandledException : "true",
61         useSpringBoot3    : "true",
62         useSpringController: "true",
63         skipDefaultInterface : "true"
64     ])
65 }
66
67 // 추가
68 compileJava.dependsOn tasks.named("openApiGenerate")
69
70 // 추가
71 sourceSets {
72     main {
73         java {
74             srcDirs = ['src/main/java', 'build/generate-resources/src/main/java']
75         }
76     }
77 }
78

```

단계별 설명 [↗](#)

▼ 단계별 설명

plugins 절에 OpenAPI Generator Plugin **추가** [↗](#)



```

1 plugins {
2     id 'java'
3     id 'org.springframework.boot' version '3.1.2'
4     id 'io.spring.dependency-management' version '1.1.2'
5     id "org.openapi.generator" version "6.6.0" //추가
6 }
7

```

dependencies 절에 의존성 추가 [🔗](#)

```

1 dependencies {
2     //추가
3     implementation 'org.springframework.boot:spring-boot-starter-web'
4     implementation 'org.springdoc:springdoc-openapi-starter-webmvc-ui:2.2.0'
5     implementation('org.openapitools:openapi-generator:6.6.0') {
6         exclude group: 'org.slf4j', module: 'slf4j-simple'
7     }
8     implementation 'org.openapitools:jackson-databind-nullable:0.2.6'
9     implementation 'org.springframework.boot:spring-boot-starter-validation'
10 }

```

openApiGenerate task 추가 [🔗](#)

```

1 openApiGenerate {
2     generatorName.set("spring")
3     verbose.set(true)
4     inputSpec.set("$rootDir/src/main/resources/static/simple-api.yaml")
5     outputDir.set("$buildDir/generate-resources")
6     apiPackage.set("com.example.openapigen.api")
7     invokerPackage.set("com.example.openapigen.invoker")
8     modelPackage.set("com.example.openapigen.model")
9     configOptions.set([
10         dateLibrary      : "java8",
11         library           : "spring-boot", //spring-boot, spring-mvc, spring-cloud
12         interfaceOnly     : "true",
13         unhandledException : "true",
14         useSpringBoot3    : "true",
15         useSpringController : "true",
16         skipDefaultInterface : "true"
17     ])
18 }

```

compileJava task에 openApiGenerate task 의존성 추가 [🔗](#)

```

1 compileJava.dependsOn tasks.named("openApiGenerate")

```

openApiGenerate task로 추출된 소스에 대한 소스 디렉토리 설정 [🔗](#)

```

1 sourceSets {
2     main {
3         java {
4             srcDirs = ['src/main/java', 'build/generate-resources/src/main/java']
5         }
6     }
7 }

```

```
6     }
7 }
```

이 설정을 통해서 추출된 소스를 참조할 수 있습니다~


openApiGenerate task 실행 [↗](#)

```
1 ./gradlew openApiGenerate
```

위의 명령어를 root 디렉토리 terminal에서 실행합니다~

▼ 다른 방법들

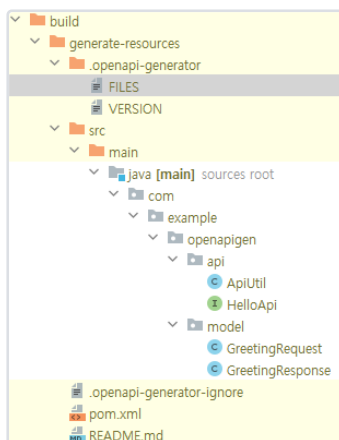
```
openApiGenerate {
    generatorName.set("spring")
    verbose.set(true)
    inputSpec.set("$rootDir/specs/simple-api.yaml")
    outputDir.set("$buildDir/generate-resources")
    apiPackage.set("com.example.openapigen.api")
    invokerPackage.set("com.example.openapigen.invoker")
    modelPackage.set("com.example.openapigen.model")
    configOptions.set([
        dateLibrary      : "java8",
        library           : "spring-boot",
        interfaceOnly     : "true",
        unhandledException : "true",
        useSpringBoot3    : "true",
        useSpringController : "true",
        skipDefaultInterface : "true"
    ])
}
```

위의 그림에서  버튼을 눌러서 openApiGenerate Task를 실행 혹은



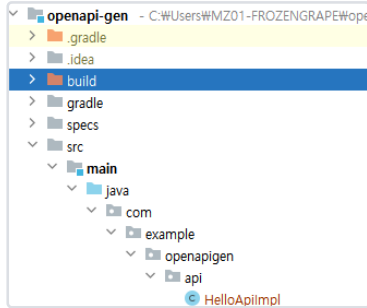
IntelliJ 우측 탭에서 Gradle -> {프로젝트 명} -> Tasks -> openapi tools -> openApiGenerate 실행합니다~

output 디렉토리(outputDir)로 추출된 소스 확인 [↗](#)



구현 및 테스트 ↗

추출된 Controller Interface 구현 ↗



```
@RestController
public class HelloApiImpl implements HelloApi {
    @PostMapping("/hello")
    @Override
    public ResponseEntity<GreetingResponse> helloPost(GreetingRequest greetingRequest) throws Exception {
        return ResponseEntity.ok(new GreetingResponse().message("Hello, " + greetingRequest.getName()));
    }
}
```

추출된 HelloApi Interface 를 구현합니다~

Springdoc Swagger UI 로 테스트

```
1 //application.properties
2 springdoc.swagger-ui.url=/simple-api.yaml
```

위처럼 작성하고 <http://localhost:8080/swagger-ui/index.html> 를 호출합니다~



TODO ↗

- 현행 코드로는 feign client 혹은 controller 둘 중 하나만 추출 가능합니다.. generator 를 두 개로 나눠 쓸 수 있는지 확인이 필요합니다~

- 현행 코드로는 `feign client` 혹은 `controller` 둘 중 하나만 UI 확인 및 테스트 가능합니다.. `Swagger UI` 를 두 가지로 나눠 쓸 수 있는지 확인이 필요합니다~
- `Controller` 접미사를 `Api` → `Controller || RestController` 로 변경 필요 등 naming 규칙에 대한 옵션 확인 필요
- `Frontend` 와 연계도 생각해 볼 수 있을 것으로 보입니다. `inputSpec.set()`을 원격으로 둘 수 있는 것으로 보이는데, `yaml` 파일을 S3와 같은 원격 저장소에 올려두고, `Frontend` 쪽에서는 `Client generator` 를 사용하여 `fetch`, `axios`, `typescript type` 를 추출하고, `Backend`에서는 `API` 와 `Model` 을 추출하는 방향으로도 사용 가능할 것으로 보입니다. 그렇게 한다면 커뮤니케이션에 대한 리소스를 상당 부분 줄일 수 있을 것으로 보입니다~(참고링크)

App is not responding. Wait or [cancel?](#)