[CS304] Lab10 Software Documentation

Part 1 JavaDoc

JavaDoc tool is a document generator tool in Java programming language for generating standard documentation in HTML format. It generates API documentation. It parses the declarations ad documentation in a set of source file describing classes, methods, constructors, and fields.

Before using JavaDoc tool, you must include JavaDoc comments /*...../ providing information about classes, methods, and constructors, etc. For creating a good and understandable document API for any java file you must write better comments for every class, method, constructor.

Example:

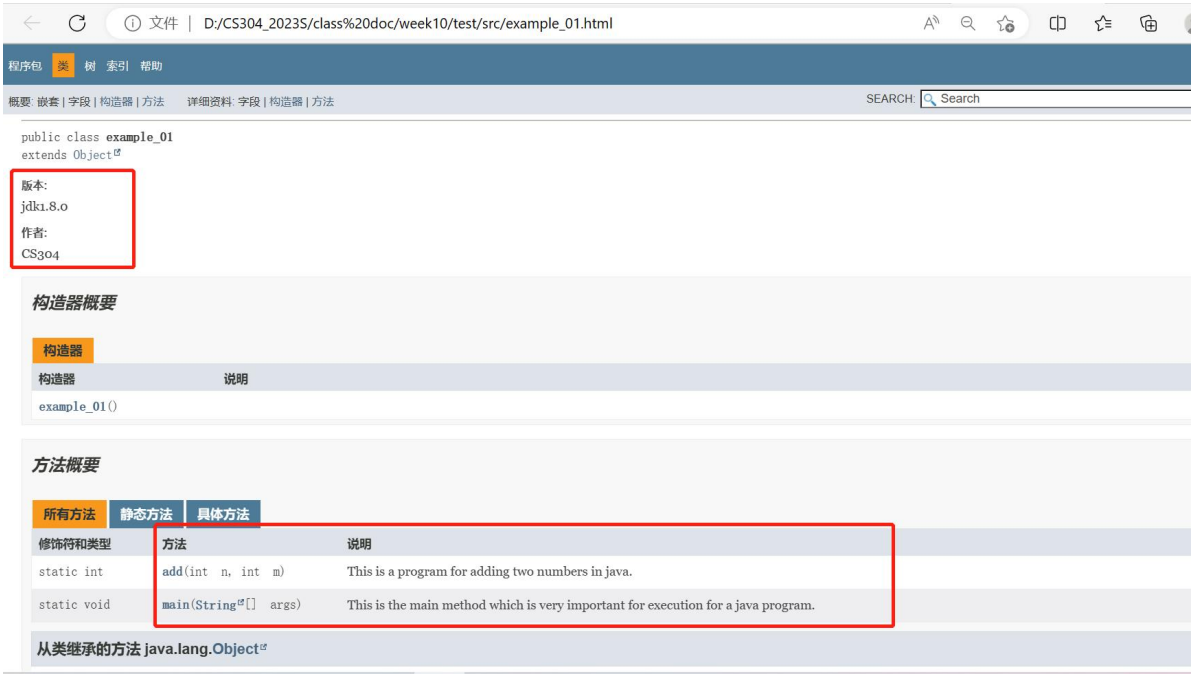
Syntax	Parameter	Description
@author	author_name	Describes an author
@param	description	provide information about method parameter or the input it takes
@version	version-name	provide version of the class, interface or enum.
@return	description	provide the return value

More Tag information links(Official links):
https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html

Part 1-1 Start a simple Java programe

Creat a simple Java program (int)a+b,please refer attachment(example_01.java),run command-line below the java path:

```
javadoc -d .\javadoc -author -version -encoding UTF-8 -charset UTF-8
example_01.java
```



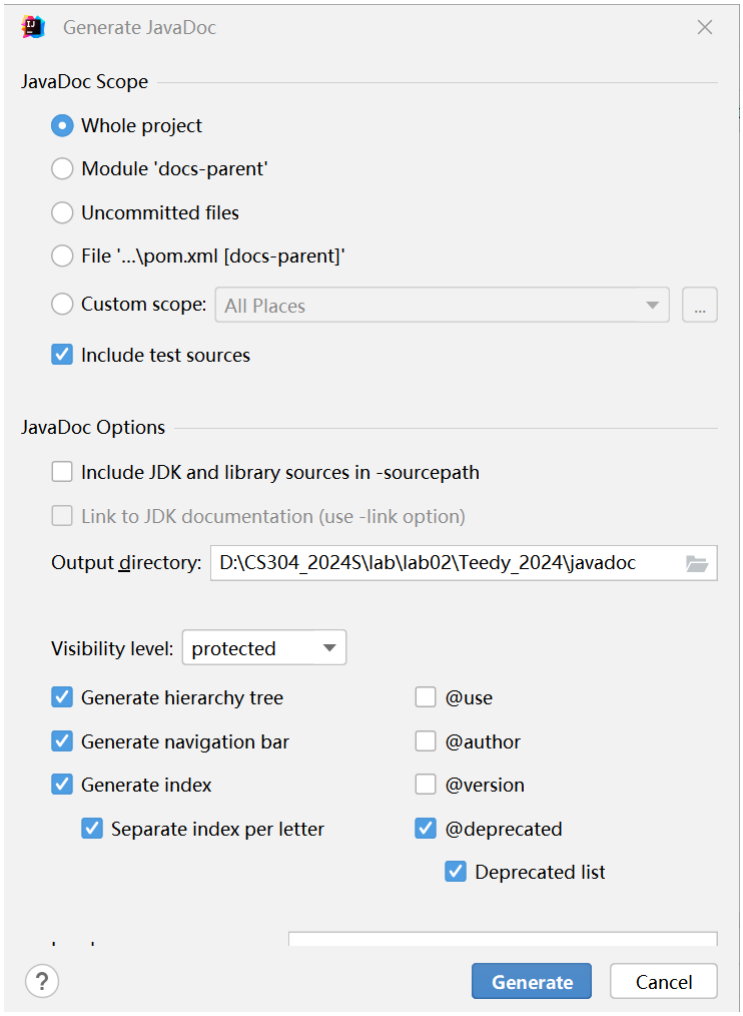
About command you can refer oracle document below:

https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html

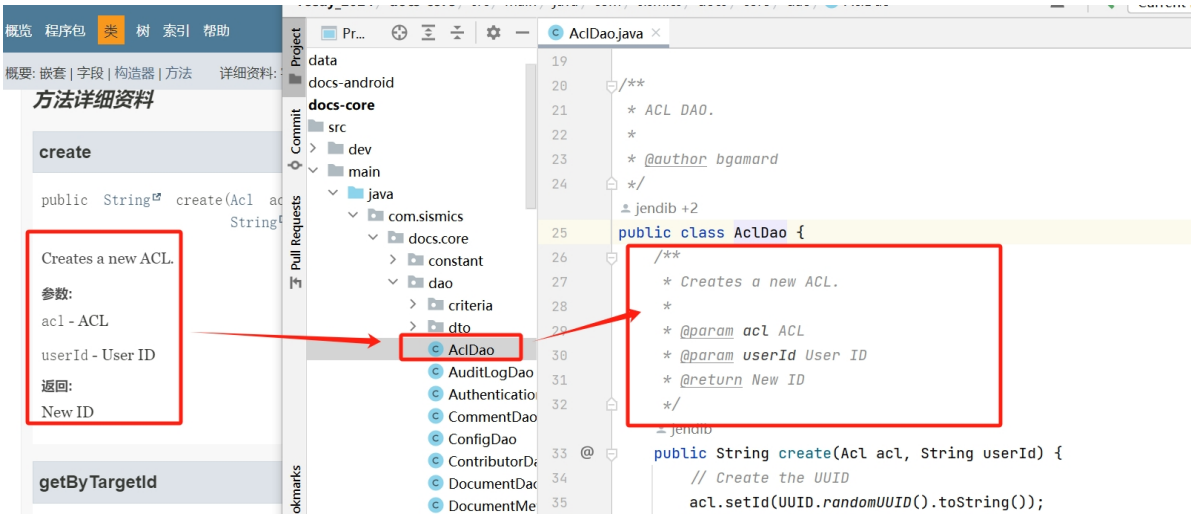
javadoc {packages|source-files} [options] [@argfiles]

Part 1-2 Create apidocs for Teedy Project

In IDEA, open tools->Generate JavaDoc



Open ..\ javadoc\index.html:



Teedy has 3 modules docs-core, docs-web-common, and docs-web, each can be generated report independently， if we want generate apidocs for each module also whole project by maven command.

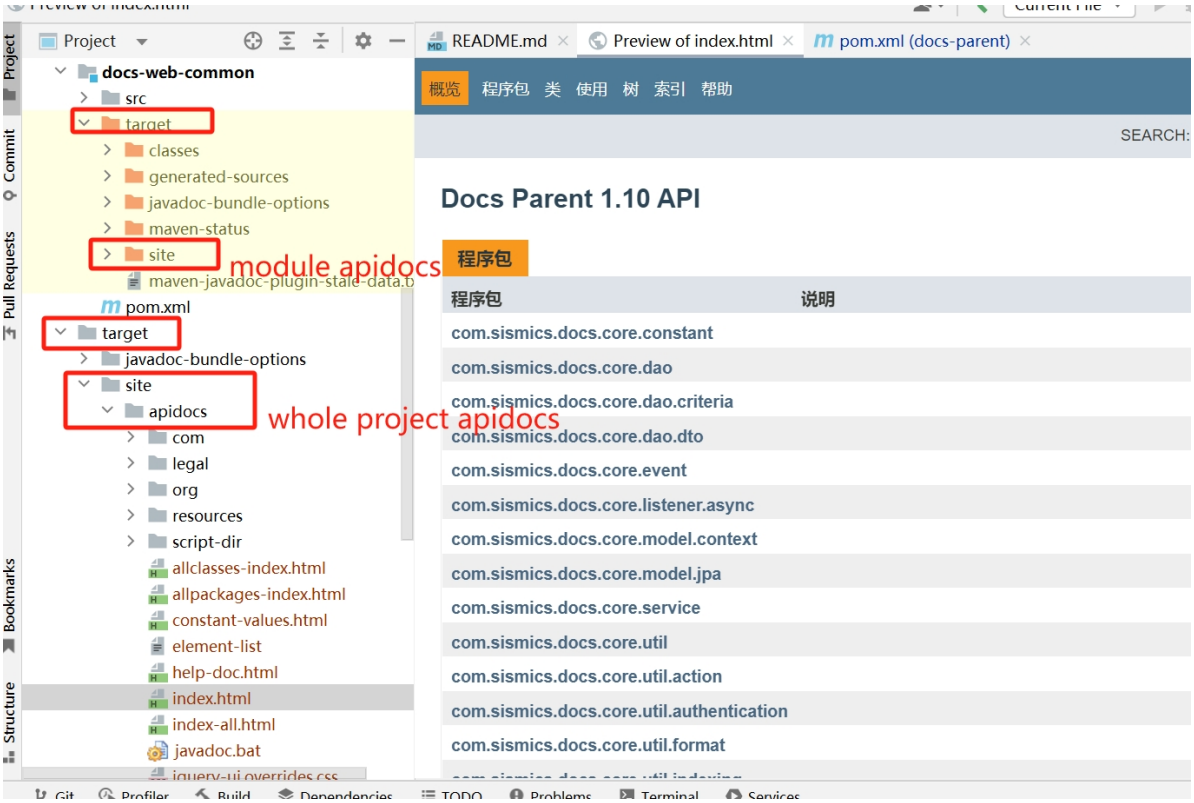
Generate Javadocs As Part Of Project Reports

To generate javadocs as part of the site generation, you should add the Javadoc Plugin in the section of your pom: add the following into the pom.xml of Teedy, Then, run `mvn site --fail-never`,which generates apidocs in html format in target/site/apidocs/index.html in root and module directory.

```
<project>
  ...

  <reporting>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-javadoc-plugin</artifactId>
        <version>3.6.3</version>
        <reportSets>
          <reportSet>
            <id>aggregate</id>
            <inherited>false</inherited>
            <reports>
              <report>aggregate</report>
            </reports>
          </reportSet>
          <reportSet>
            <id>default</id>
            <reports>
              <report>javadoc</report>
            </reports>
          </reportSet>
        </reportSets>
      </plugin>
    </plugins>
  </reporting>
```

```
    ...
</project>
```



Generate standalone javadocs for the project

You could also add the Javadoc Plugin in the section of your pom (if no configuration defined, the plugin uses default values).

Add the following into the pom.xml of Teedy, Then, run `mvn javadoc:javadoc --fail-never`,which generates apidocs in html format in target/site/apidocs/index.html in each module directory.

```
<project>
  ...

  <build>
    <plugins>

      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-javadoc-plugin</artifactId>
        <version>3.6.3</version>
        <executions>
          <execution>
            <id>aggregate</id>
            <goals>
              <goal>aggregate</goal>
            </goals>
            <phase>site</phase>
          </execution>
        </executions>
      </plugin>
```

```
    </plugins>
  </build>

  ...
</project>
```

reference:

https://maven.apache.org/plugins/maven-javadoc-plugin/index.html

Part 2 Swagger

reference:
https://swagger.io/

https://springdoc.org/index.html#Introduction

Swagger, is an API description format for RESTful APIs. It provides a standardized way to describe the structure of an API, making it easy for developers to understand and interact with the API. The OpenAPI Specification (OAS) defines a set of rules for describing the API's endpoints, methods, input parameters, and response objects.

Part 2-1 Configure Swagger in SpringBoot Project

Create a SpringBoot template project by IDEA,add Spring Web dependency.

Add package `controller` and `MyController.java`, we can simply run it.

```
@RestController
public class MyController {
    @RequestMapping(value = "/cs304")
    public String hello(){
        return "hello,cs304";
    }
}
```

run this project:

Configure Swagger and start:

mvn repo:

https://mvnrepository.com/artifact/org.springdoc/springdoc-openapi-starter-webmvc-ui

https://mvnrepository.com/artifact/org.springdoc/springdoc-openapi-starter-webmvc-api

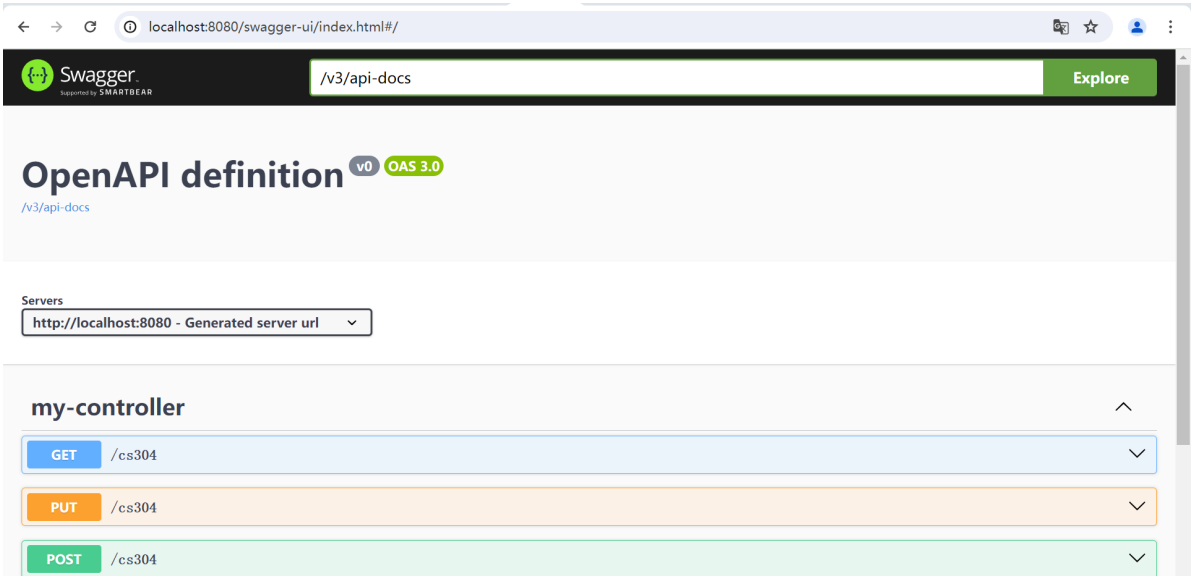
Add the following into the pom.xml:

```
        <dependency>
            <groupId>org.springdoc</groupId>
            <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
            <version>2.5.0</version>
        </dependency>

        <dependency>
            <groupId>org.springdoc</groupId>
            <artifactId>springdoc-openapi-starter-webmvc-api</artifactId>
            <version>2.5.0</version>
        </dependency>
```

Add package `config` and `SwaggerConfig.java`,

run this project , then Go to http://localhost:8080/swagger-ui/index.html#/



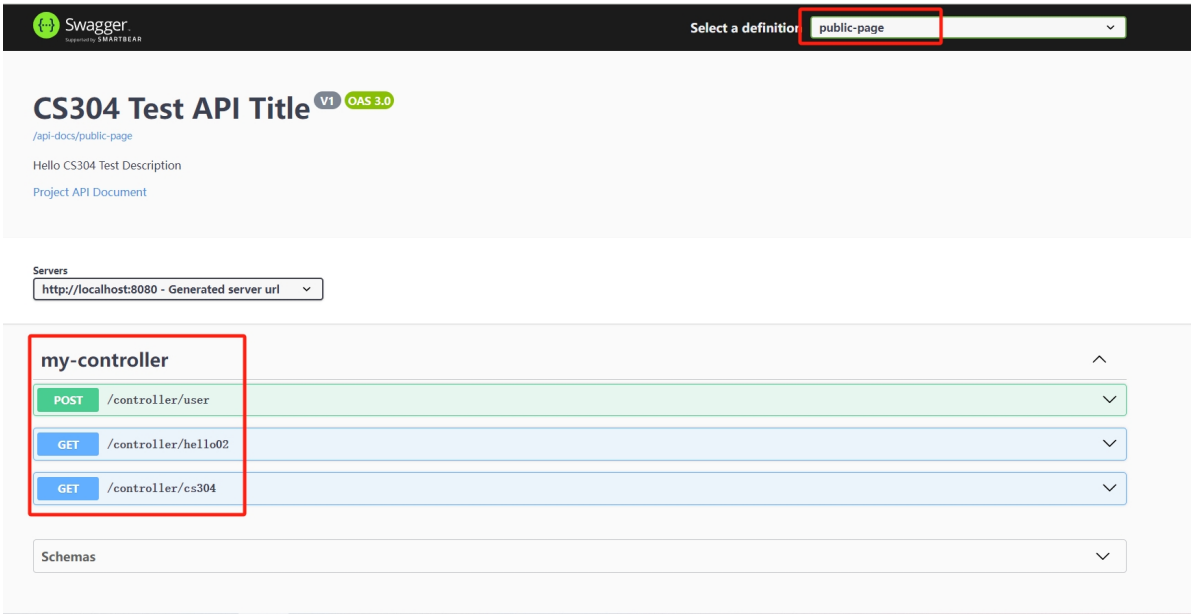
Configure basic information:

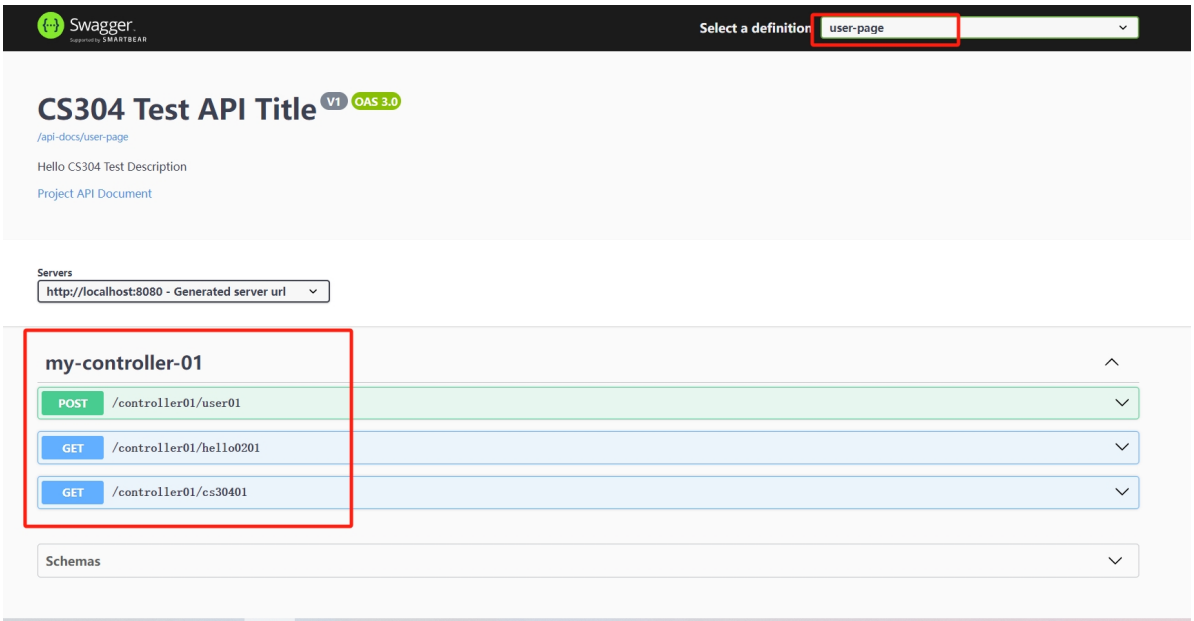
```
    @Bean
    public OpenAPI openAPI(){
        return new OpenAPI()
                .info(new Info()
                        .title("CS304 Test API Title")
                        .description("Hello CS304 Test Description")
                        .version("V1"))
                .externalDocs(new ExternalDocumentation()
                        .description("Project API Document")
                        .url("/"));
    }
```

You can define your own groups of API based on the combination of: API paths and packages to scan. Each group should have a unique groupName. Set group, then you can :

```
    //path
    @Bean
    public GroupedOpenApi publicApi() {
        return GroupedOpenApi.builder()
                .group("public-page")
                .pathsToMatch("/controller/**")
                .build();
    }

    //package
    @Bean
    public GroupedOpenApi userApi() {
        String packagesToscan[] = {"com.example.demo10.controller01"};
        return GroupedOpenApi.builder()
                .group("user-page")
                .packagesToScan(packagesToscan)
                .build();
    }
```



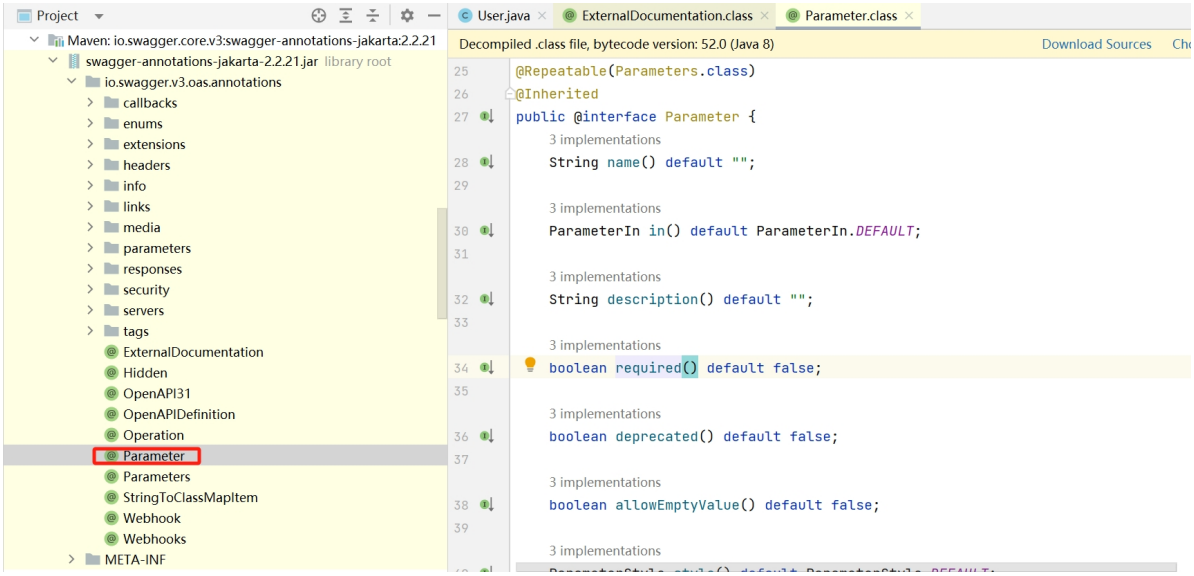


Part 2-2 swagger 3 annotations

Package for swagger 3 annotations is io.swagger.v3.oas.annotations. Here are some useful annotations:

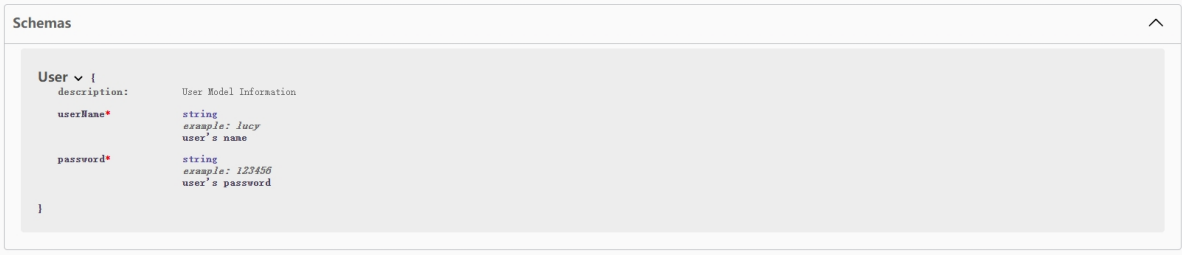
- `@Tag`
- `@Operation`
- `@Parameters` and `@Parameter`
- `@Schema`
- `@Hidden` or `@Parameter(hidden = true)` or `@Operation(hidden = true)`
- `@ApiResponses` and `@ApiResponse`

You can check detail in dependency package:



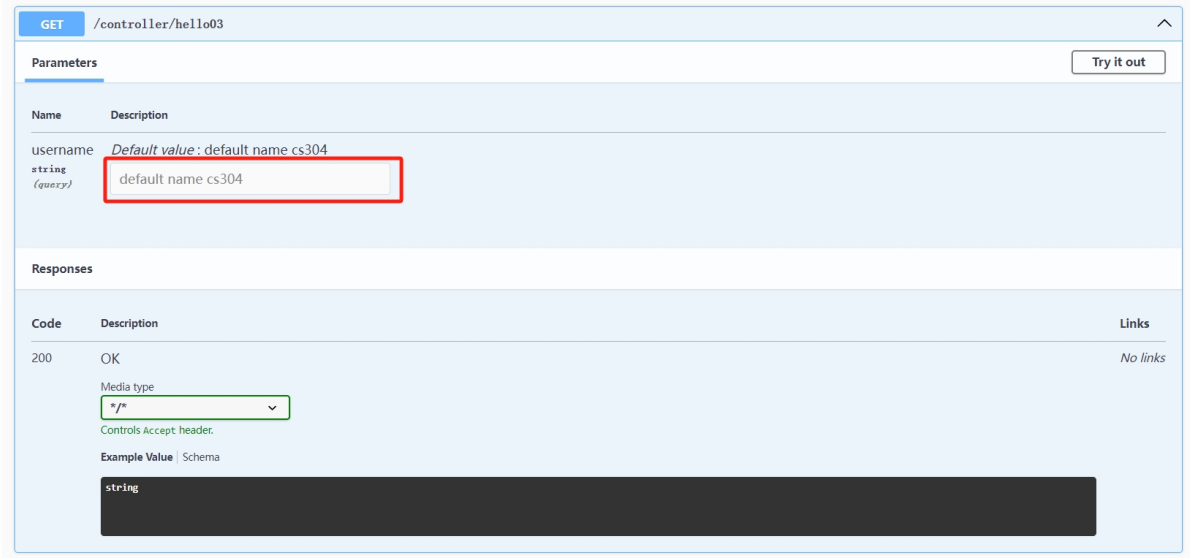
example 01:

```
@Schema(description = "User Model Information")
public class User {
    @Schema(description= "user's name ",required = true,example = "lucy")
    public String userName;
    @Schema(description = "user's password",required = true,example = "123456")
    public String password;
}
```



example 02:

```
    @GetMapping(value = "/hello03")
    public String hello03(@RequestParam(defaultValue = "default name cs304")
String username){
        return "hello02 "+username;
    }
```



Reference :

https://www.bezkoder.com/swagger-3-annotations/

https://docs.swagger.io/swagger-core/v2.0.0-RC3/apidocs/io/swagger/v3/oas/annotations/package-summary.html