axenix
innovation

Кунг-фу кастомизации

OpenAPI

OpenAPI, API First, Kafka и OpenAPI

16/10/2024

О себе

- На данный момент работаю над продуктом для применения OpenAPI для асинхронного сообщения.
- Стала экспертом OpenAPI в компании.
- Telegram: @AlexandraVolushkova
- Github:
  - https://github.com/SashaVolushkova
  - https://github.com/AxenAPI

Проблемы, с которыми мы столкнулись

Внедрение OpenAPI на проекты

Использование OpenAPI для асинхронного общения

Проблемы, с которыми мы столкнулись

Внедрение OpenAPI на проекты

- Работа с наследованием объектов.

Использование OpenAPI для асинхронного общения

Проблемы, с которыми мы столкнулись

Внедрение OpenAPI на проекты

- Работа с наследованием объектов.
- Работа с дополнительными аннотациями.

Использование OpenAPI для асинхронного общения

Проблемы, с которыми мы столкнулись

Внедрение OpenAPI на проекты

- Работа с наследованием объектов.
- Работа с дополнительными аннотациями.

Использование OpenAPI для асинхронного общения

- Работа со структурой сгенерированного кода.

Проблемы, с которыми мы столкнулись

Внедрение OpenAPI на проекты

- Работа с наследованием объектов.
- Работа с дополнительными аннотациями.

Использование OpenAPI для асинхронного общения

- Работа со структурой сгенерированного кода.
- Добавление схем в Swagger.

Проблемы, с которыми мы столкнулись

Внедрение OpenAPI на проекты

- Работа с наследованием объектов.
- Работа с дополнительными аннотациями.

Использование OpenAPI для асинхронного общения

- Работа со структурой сгенерированного кода.
- Добавление схем в Swagger.
- Добавление дополнительной информации для аналитиков.

Почему OpenAPI

Почему OpenAPI

Почему OpenAPI

Почему OpenAPI

~~Swagger 2.0 =~~
~~OpenAPI 2.0~~

Почему OpenAPI

~~Swagger 2.0 =~~
~~OpenAPI 2.0~~

OpenAPI 3.0 (2017), 3.1 (2019)

Почему OpenAPI

~~Swagger 2.0 = OpenAPI 2.0~~

OpenAPI 3.0 (2017), 3.1 (2019)

oneOf, anyOf, and not

Почему OpenAPI

~~Swagger 2.0 = OpenAPI 2.0~~

OpenAPI 3.0 (2017), 3.1 (2019)

oneOf, anyOf, and not

Обещают в 2024 OpenAPI 4.0

User *at 6:15 pm, Sep 23, 2024*

create openapi specification for application Library

User *at 6:15 pm, Sep 23, 2024*

create openapi specification for application Library

```yaml
openapi: 3.0.0
info:
  title: Your Application Library API
  version: 1.0.0
paths:
  /books:
    get:
      summary: Get a list of books
      responses:
        '200':
          description: Successful response
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref:
'#/components/schemas/Book'
  /books/{id}:
    get:
      summary: Get a specific book
      parameters:
        - in: path
          name: id
          required: true
          schema:
            type: string
      responses:
        '200':
          description: Successful response
          content:
            application/json:
              schema:
                $ref:
'#/components/schemas/Book'
components:
  schemas:
    Book:
      type: object
      properties:
        id:
          type: string
        title:
          type: string
        author:
          type: string
```

User *at 6:16 pm, Sep 23, 2024*

add more functionality

User *at 6:16 pm, Sep 23, 2024*

add more functionality

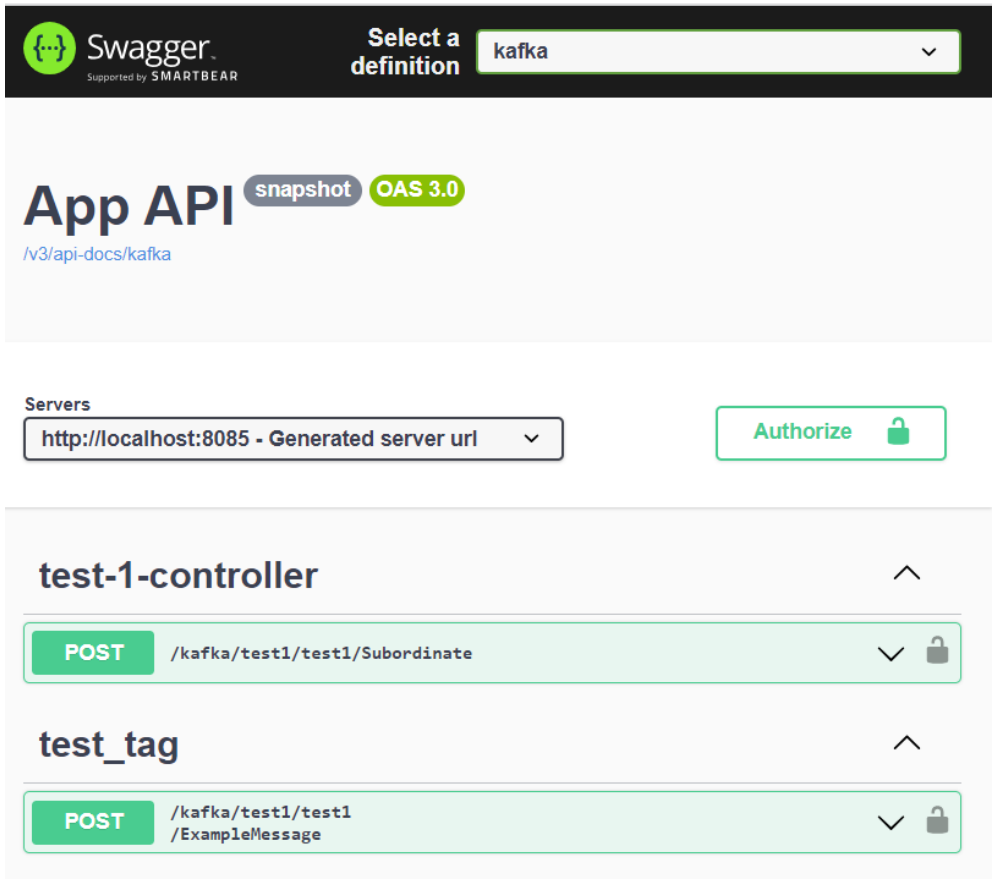
```
/books:
  post:
    summary: Create a new book
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref:
'#/components/schemas/Book'
    responses:
      '201':
        description: Book created
successfully
        content:
          application/json:
            schema:
              $ref:
'#/components/schemas/Book'
# Add an endpoint to update an
existing book:
/books/{id}:
  put:
    summary: Update a book
    parameters:
      - in: path
        name: id
        required: true
        schema:
          type: string
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref:
'#/components/schemas/Book'
    responses:
      '200':
        description: Book updated
successfully
        content:
          application/json:
            schema:
              $ref:
'#/components/schemas/Book'

# Add an endpoint to delete a book:
/books/{id}:
  delete:
    summary: Delete a book
    parameters:
      - in: path
        name: id
        required: true
        schema:
          type: string
    responses:
      '204':
        description: Book deleted
successfully
# Add an endpoint to search for books
by title or author:
/books/search:
  get:
    summary: Search for books
    parameters:
      - in: query
        name: q
        required: true
        schema:
          type: string
    responses:
      '200':
        description: Successful
response
        content:
          application/json:
            schema:
              type: array
              items:
                $ref:
'#/components/schemas/Book'
```

Почему OpenAPI

OpenAPI

Почему OpenAPI

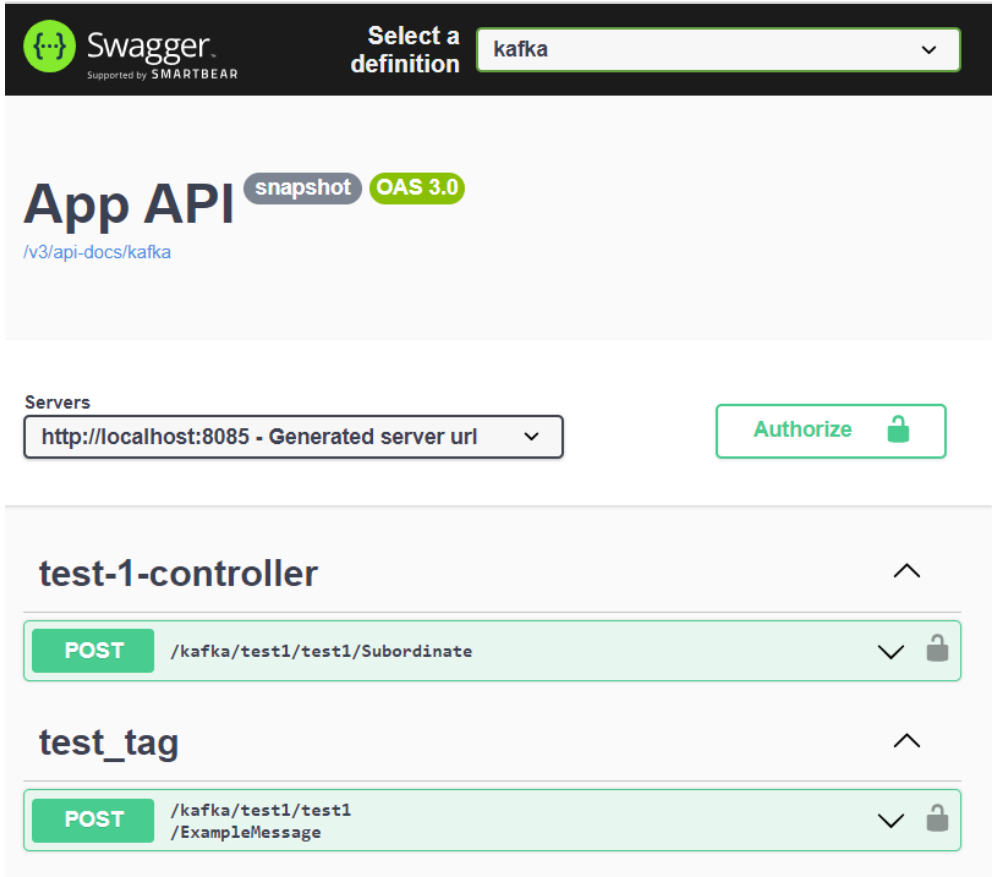
OpenAPI



SwaggerUI

Почему OpenAPI

OpenAPI

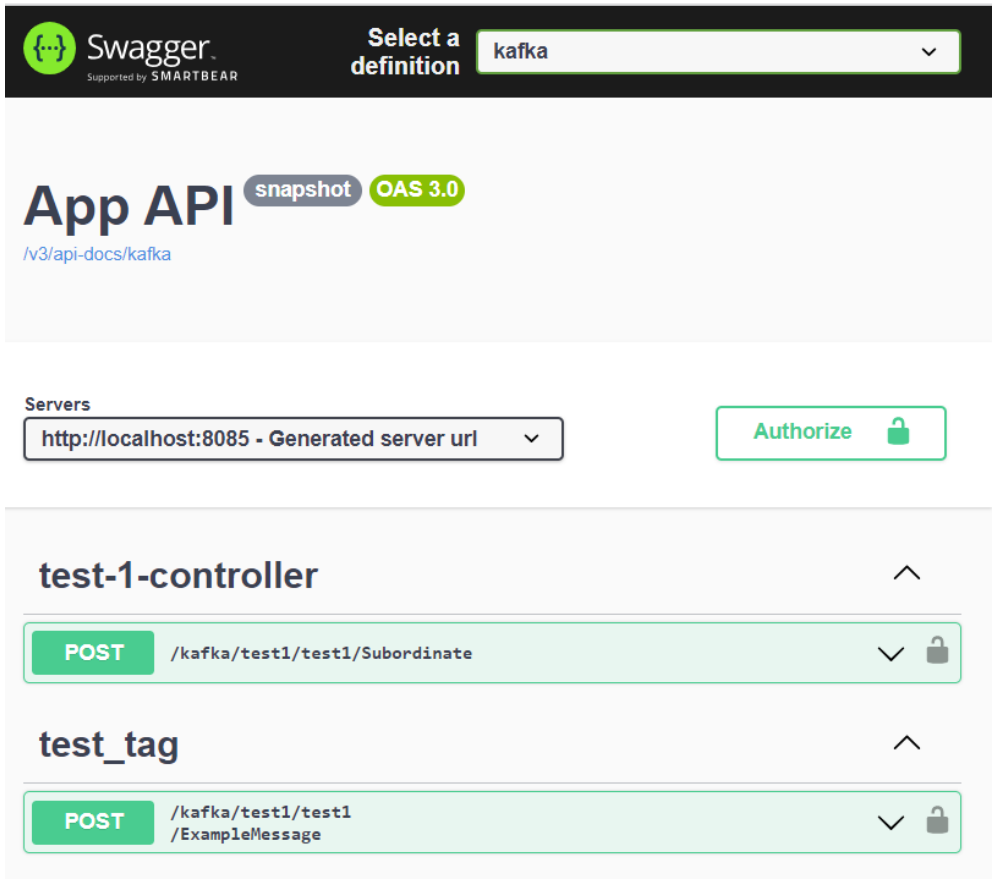


SwaggerUI

OpenAPI Generator

Почему OpenAPI

OpenAPI



SwaggerUI

OpenAPI Generator

1 OpenAPI Generator

2 Кастомные аннотации

3 Работа с Lombok

4 Свой генератор

5 Кастомизация SpringDocUI

6 Наследование в Swagger

7 Выводы

OpenAPI Generator. Mustache

- OpenAPI-parser
- Набор Mustache файлов.

OpenAPI Generator. Mustache

Mustache – простой шаблон с минимумом логики

- OpenAPI-parser
- Набор Mustache файлов.

OpenAPI Generator. Mustache

- OpenAPI-parser
- Набор Mustache файлов.

Mustache – простой шаблон с минимумом логики

1 OpenAPI Generator

2 Кастомные аннотации

3 Работа с Lombok

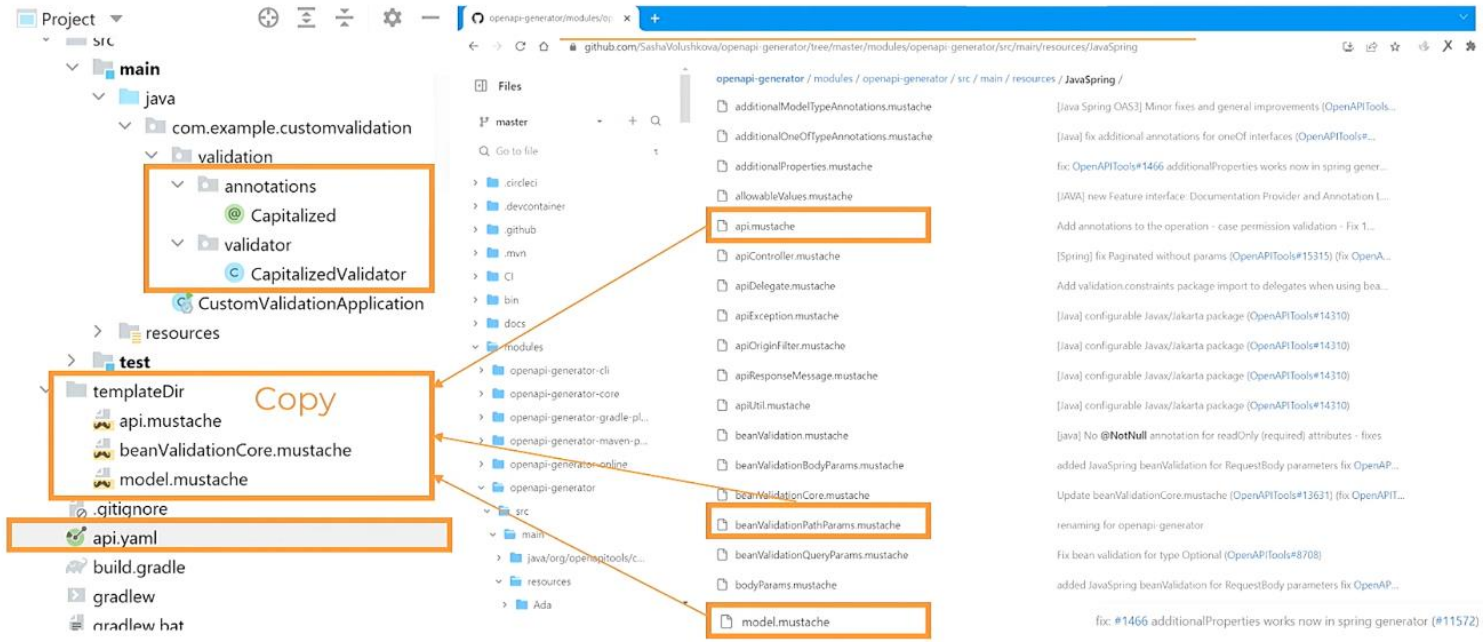
4 Свой генератор

5 Кастомизация SpringDocUI

6 Наследование в Swagger

7 Выводы

OpenAPI Generator. Кастомизация



Александра Волушкова

Axenix

25

OpenAPI Generator. Spring. Кастомные аннотации

Использовать x-дополнения к спецификации из документации для spring генератора: https://openapi-generator.tech/docs/generators/spring/

x-class-extra-annotation	List of custom annotations to be added to model	MODEL	null
x-field-extra-annotation	List of custom annotations to be added to property	FIELD	null
x-operation-extra-annotation	List of custom annotations to be added to operation	OPERATION	null

OpenAPI Generator. Spring. Кастомные аннотации

```
Error:
  x-class-extra-annotation: "@ClassAnnotation"
  properties:
    message:
      type: string
      x-field-extra-annotation: "@FieldAnnotation"
```

OpenAPI Generator. Spring. Кастомные аннотации

```
Error:
  x-class-extra-annotation: "@ClassAnnotation"
  properties:
    message:
      type: string
      x-field-extra-annotation: "@FieldAnnotation"
```

OpenAPI Generator. Spring. Кастомные аннотации

```
@ClassAnnotation
public class Error {

  private Integer code;

  @FieldAnnotation
  private String message;
```

OpenAPI Generator. Spring. Кастомные аннотации

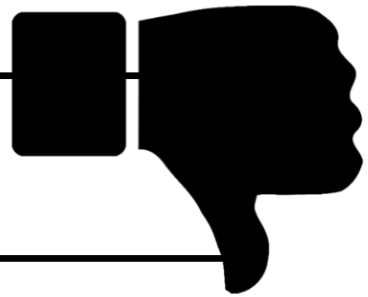
```
@ClassAnnotation
public class Error {

    private Integer code;

    @FieldAnnotation
    private String message;
```

Давайте исправим

```
Pet:
  properties:
    id:
      type: integer
      format: int64
      minimum: 5
      x-customValidation: "argument"
    name:
      type: string
      x-field-extra-annotation:
"@your.package.Capitalized(required = true)"
```

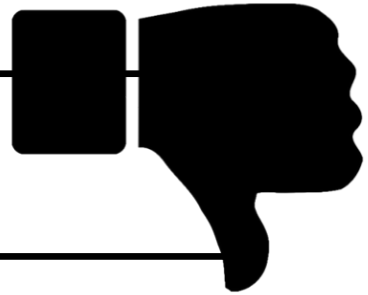
Давайте исправим

```
Pet:
  properties:
    id:
      type: integer
      format: int64
      minimum: 5
      x-customValidation: "argument"
    name:
      type: string
      x-field-extra-annotation:
"@your.package.Capitalized(required = true)"
```



Давайте исправим

```
Pet:
  properties:
    id:
      type: integer
      format: int64
      minimum: 5
      x-customValidation: "argument"
    name:
      type: string
      x-field-extra-annotation:
"@your.package.Capitalized(required = true)"
```



Итог

- Проблема с использованием других языков.

Итог

- Проблема с использованием других языков.
- **В спецификацию попадает код.**

Итог

- Проблема с использованием других языков.
- **В спецификацию попадает код.**

Рекомендую посмотреть документацию для каждого генератора и посмотреть какие
x-property они поддерживают

1 OpenAPI Generator

2 Кастомные аннотации

3 Работа с Lombok

4 Свой генератор

5 Кастомизация SpringDocUI

6 Наследование в Swagger

7 Выводы

Постановка задачи

1. Мы использовали Lombok. А именно **@Builder.**

Постановка задачи

1. Мы использовали Lombok. А именно **@Builder.**
2. Мы стали использовать сгенерированные DTO.

Постановка задачи

1. Мы использовали Lombok. А именно **@Builder.**
2. Мы стали использовать сгенерированные DTO.
3. И встретились с вот такой проблемой:

Проблема: Builder другого формата

```
@Override
public ResponseEntity<Void> createPets() {
    Pet.PetBuilder petBuilder = Pet.builder();
    Pet pet = petBuilder.id(1L).tag("tag1")
                        .name("cat").build();
    pets.add(pet);
    return ResponseEntity.status(201).build();
}
```

Проблема: Builder другого формата

```
@Override
public ResponseEntity<Void> createPets() {
    Pet.PetBuilder petBuilder = Pet.builder();
    Pet pet = petBuilder.id(1L).tag("tag1")
                        .name("cat").build();
    pets.add(pet);
    return ResponseEntity.status(201).build();
}
```

Сгенерированные Builder методы

```
@Generated(...)
public class Pet {
  // Fields: id, name, tag ...
  public Pet id(Long id) {
    this.id = id;
    return this;
  }

  public Pet name(String name) {
    this.name = name;
    return this;
  }

  public Pet tag(String tag) {
    this.tag = tag;
    return this;
  } //...
}
```

Сгенерированные Builder методы


```java
@Generated(...)
public class Pet {
  // Fields: id, name, tag ...
  public Pet id(Long id) {
    this.id = id;
    return this;
  }

  public Pet name(String name) {
    this.name = name;
    return this;
  }

  public Pet tag(String tag) {
    this.tag = tag;
    return this;
  } //...
}
```

Сгенерированные Builder методы

```
Pet pet = new Pet();
pet = pet.id(1L).name("name").tag("tag");
pets.add(pet);
```

Сгенерированные Builder методы

```
Pet pet = new Pet();
pet = pet.id(1L).name("name").tag("tag");
pets.add(pet);
```

Если использовать openapi-generator изначально, то можно использовать эти Builder методы

Хотим использовать @lombok.Builder

```yaml
Pet:

  required:
    - id
    - name
  properties:
    # Fields: id, name, tag...
```

Хотим использовать @lombok.Builder

```yaml
Pet:
  x-class-extra-annotation: "@lombok.Builder"
  required:
    - id
    - name
  properties:
    # Fields: id, name, tag...
```

С первого раза не получилось

```
@lombok.Builder
^
  constructor Pet.Pet() is not applicable
    (actual and formal argument lists differ in length)
  constructor Pet.Pet(Long,String) is not applicable
    (actual and formal argument lists differ in length)
```

С первого раза не получилось

@lombok.Builder

^

constructor Pet.Pet() is not applicable

(actual and formal argument lists differ in length)

constructor

Pet.Pet(Long,String) is not applicable

(actual and formal argument lists differ in length)

```
public Pet() {  super(); }

/**
 * Constructor with only required parameters
 */
public Pet(Long id, String name) {
  this.id = id;
  this.name = name;
}
```

Давайте починим

```
Pet:
  x-class-extra-annotation: "@lombok.Builder
@lombok.AllArgsConstructor @lombok.NoArgsConstructor"
  required:
    - id
    - name
  properties:
 # Fields: id, name, tag...
```

Давайте починим

```
Pet:
  x-class-extra-annotation: "@lombok.Builder
@lombok.AllArgsConstructor @lombok.NoArgsConstructor"
  required:
    - id
    - name
  properties:
 # Fields: id, name, tag...
```

Давайте починим

```
Pet:
  x-class-extra-annotation: "@lombok.Builder
@lombok.AllArgsConstructor @lombok.NoArgsConstructor"
  required:
    - id
    - name
  properties:
  # Fields: id, name, tag...
```

Настройка генерации

В файле **build.gradle** добавляем в этап **openApiGenerate** параметры:

```
openApiGenerate {
    generatorName = "spring"
    // ...
    configOptions = [
            // ...


    ]
}
```

Настройка генерации

В файле **build.gradle** добавляем в этап **openApiGenerate** параметры:

```
openApiGenerate {
    generatorName = "spring"
    // ...
    configOptions = [
            // ...
        generateConstructorWithAllArgs: "false",

    ]
}
```

Настройка генерации

В файле **build.gradle** добавляем в этап **openApiGenerate** параметры:

```
openApiGenerate {
    generatorName = "spring"
    // ...
    configOptions = [
            // ...
        generateConstructorWithAllArgs: "false",

    ]
}
```

Если в DTO есть обязательные поля, то этого недостаточно

Настройка генерации

В файле **build.gradle** добавляем в этап **openApiGenerate** параметры:

```
openApiGenerate {
    generatorName = "spring"
    // ...
    configOptions = [
            // ...
        generateConstructorWithAllArgs: "false",
        generatedConstructorWithRequiredArgs: "false"
    ]
}
```

Нужно убрать конструктор с обязательными полями

Что получилось

```
@Generated(...)
@lombok.Builder @lombok.AllArgsConstructor
@lombok.NoArgsConstructor
public class Pet {

  private Long id;
  private String name;
  private String tag;

//...
```

Что получилось

```
@Generated(...)
@lombok.Builder @lombok.AllArgsConstructor
@lombok.NoArgsConstructor
public class Pet {

   private Long id;
   private String name;
   private String tag;

//...
```

Итог

- Мы получили корректное использование Lombok.

Итог

- Мы получили корректное использование Lombok.
- @lombok.Getter, @lombok.Setter использовать не нужно – get и set методы уже есть в стандартной генерации.

Итог

- Мы получили корректное использование Lombok.
- @lombok.Getter, @lombok.Setter использовать не нужно – get и set методы уже есть в стандартной генерации.
- **В спецификацию попадает код.**

Итог

- Мы получили корректное использование Lombok.
- @lombok.Getter, @lombok.Setter использовать не нужно – get и set методы уже есть в стандартной генерации.
- **В спецификацию попадает код.**

Мы создали свой генератор – это решило проблему!

1 OpenAPI Generator

2 Кастомные аннотации

3 Работа с Lombok

4 Свой генератор

5 Кастомизация SpringDocUI

6 Наследование в Swagger

7 Выводы

Создание своего генератора

```
java -jar
```

Создание своего генератора

```
java -jar openapi-generator-cli.jar
```

Создание своего генератора

```
java -jar openapi-generator-cli.jar meta
```

Создание своего генератора

```
java -jar openapi-generator-cli.jar meta -o
out/generators/my-codegen -n my-codegen -p
com.my.company.codegen
```

Создание своего генератора

```
java -jar openapi-generator-cli.jar meta -o
out/generators/my-codegen -n my-codegen -p
com.my.company.codegen
```

Открываем сгенерированный проект

Создание своего генератора

- scr
  - main
    - java
      - com.my.company.codegen
        - ❑ **MyCodegenGenerator**
  - resources
    - META-INF.services
      - ❑ **org.openapitools.codegen.CodegenConfig**
    - my-codegen
      - ❑ api.mustache
      - ❑ model.mustache
      - ❑ myFile.mustache

Создание своего генератора

- scr
  - main
    - java
      - com.my.company.codegen
        - ❑ **MyCodegenGenerator**
  - resources
    - META-INF.services
      - ❑ **org.openapitools.codegen.CodegenConfig**
    - my-codegen
      - ❑ api.mustache
      - ❑ model.mustache
      - ❑ myFile.mustache

Класс-генератор

Создание своего генератора

- scr
  - main
    - java
      - com.my.company.codegen
        - ❑ **MyCodegenGenerator**
  - resources
    - META-INF.services
      - ❑ **org.openapitools.codegen.CodegenConfig**
    - my-codegen
      - ❑ api.mustache
      - ❑ model.mustache
      - ❑ myFile.mustache

Класс-генератор

Файл конфигурации

Создание своего генератора

- scr
  - main
    - java
      - com.my.company.codegen
        - ❑ **MyCodegenGenerator**
  - resources
    - META-INF.services
      - ❑ **org.openapitools.codegen.CodegenConfig**
    - my-codegen
      - ❑ api.mustache
      - ❑ model.mustache
      - ❑ myFile.mustache

Класс-генератор

Файл конфигурации

Шаблоны

Создание своего генератора

```
public class MyCodegenGenerator  extends DefaultCodegen
implements CodegenConfig {
{

  public CodegenType getTag() {
    return CodegenType.OTHER;
  }

  public String getName() {
    return "my-codegen";
  }
//...
}
```

Создание своего генератора

Файл org.openapitools.codegen.CodegenConfig содержит наименования всех классов-генераторов.

Пример файла(один класс-генератор):

```
com.my.company.codegen.MyCodegenGenerator
```

Создание своего генератора

Файл org.openapitools.codegen.CodegenConfig содержит наименования всех классов-генераторов.

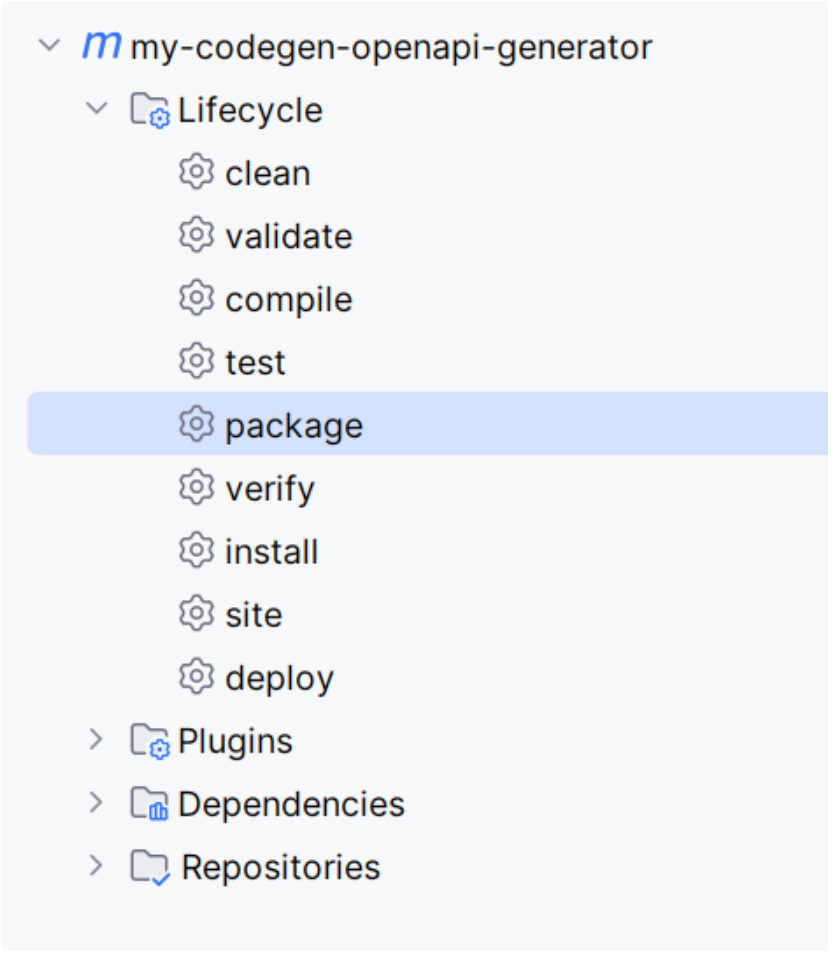
Пример файла(**много** классов-генераторов):

```
com.my.company.codegen.MyCodegenGenerator
com.my.company.codegen.MyPythonGenerator
com.my.company.codegen.MyKotlinGenerator
```

Создание своего генератора

Наш генератор готов.

Давайте соберем проект: `mvn package`



Использование своего генератора

```
java -cp
```

Использование своего генератора

```
java -cp “my-codegen.jar;
```

Использование своего генератора

```
java -cp “my-codegen.jar;openapi-generator-cli.jar"
org.openapitools.codegen.OpenAPIGenerator list
```

Использование своего генератора

```
java –cp “my-codegen.jar;openapi-generator-cli.jar"
org.openapitools.codegen.OpenAPIGenerator list
```

```
The following generators are available:
CLIENT generators:
    - ada
.....
OTHER generators:
    - my-codegen
```

Использование своего генератора

```
java –cp “my-codegen.jar;openapi-generator-cli.jar"
org.openapitools.codegen.OpenAPIGenerator list
```

```
The following generators are available:
CLIENT generators:
    - ada
.....
OTHER generators:
    - my-codegen
```

Использование своего генератора

```
java –cp “my-codegen.jar;openapi-generator-cli.jar"
org.openapitools.codegen.OpenAPIGenerator list
```

```
The following generators are available:
CLIENT generators:
    - ada
.....
OTHER generators:
    - my-codegen
```

Отредактируем класс-генератор

```
public class MyCodegenGenerator
extends DefaultCodegen implements CodegenConfig
```

Отредактируем класс-генератор

```
public class MyCodegenGenerator
extends DefaultCodegen implements CodegenConfig
```

Отредактируем класс-генератор

```
public class MyCodegenGenerator
extends DefaultCodegen implements CodegenConfig
extends SpringCodegen
```

Конструктор

```
public MyCodegenGenerator() {
  super();
  outputFolder = "generated-code/my-codegen";
  modelTemplateFiles.put("model.mustache",".sample");
  apiTemplateFiles.put("api.mustache",".sample");

  templateDir = "my-codegen";
  apiPackage = "org.openapitools.api";
  modelPackage = "org.openapitools.model";

  reservedWords = new HashSet<String> (...);
  additionalProperties.put("apiVersion", apiVersion);
  supportingFiles.add(...);
  languageSpecificPrimitives = new HashSet<String>(...);
}
```

Конструктор

```
public MyCodegenGenerator() {
  super();
  outputFolder = "generated-code/my-codegen";
  modelTemplateFiles.put("model.mustache",".sample");
  apiTemplateFiles.put("api.mustache",".sample");

  templateDir = "my-codegen";
  apiPackage = "org.openapitools.api";
  modelPackage = "org.openapitools.model";

  reservedWords = new HashSet<String> (...);
  additionalProperties.put("apiVersion", apiVersion);
  supportingFiles.add(...);
  languageSpecificPrimitives = new HashSet<String>(...);
}
```

Конструктор

```
public MyCodegenGenerator() {
  super();
  templateDir = "my-codegen";
  additionalProperties.put("apiVersion", apiVersion);
}
```

Конструктор

```
public MyCodegenGenerator() {
  super();
  outputFolder = "generated-code/my-codegen";
  modelTemplateFiles.put("model.mustache",".sample");
  apiTemplateFiles.put("api.mustache",".sample");

  templateDir = "my-codegen";
  apiPackage = "org.openapitools.api";
  modelPackage = "org.openapitools.model";

  reservedWords = new HashSet<String> (...);
  additionalProperties.put("apiVersion", apiVersion);
  supportingFiles.add(...);
  languageSpecificPrimitives = new HashSet<String>(...);
}
```

Конструктор. Другие настройки

`modelTemplateFiles`

`apiTemplateFiles`

`supportingFiles`

Конструктор. Другие настройки

`modelTemplateFiles` Шаблоны для моделей (schemas).

`apiTemplateFiles`

`supportingFiles`

Конструктор. Другие настройки

`modelTemplateFiles` Шаблоны для моделей (schemas)

`apiTemplateFiles` Шаблоны для методов (paths).

`supportingFiles`

Конструктор. Другие настройки

`modelTemplateFiles` Шаблоны для моделей (schemas).

`apiTemplateFiles` Шаблоны для методов (paths).

`supportingFiles` Шаблоны для всего остального (без спецификации).

Отредактируем конструктор

```
public class MyCodegenGenerator extends SpringCodegen {

    public CodegenType getTag() {
        return degenType.OTHER;
    }
    public String getName() {
      return "my-codegen";
    }
    public MyCodegenGenerator() {
      super();
      templateDir = "my-codegen";
    }
}
```

Добавим шаблоны

Добавим шаблоны

Возьмем шаблоны из Spring генератора openapi-generator.

Добавим шаблоны

Возьмем шаблоны из Spring генератора openapi-generator.

modules/openapi-generator/src/main/resources/JavaSpring:

Добавим шаблоны

Возьмем шаблоны из Spring генератора openapi-generator.

modules/openapi-generator/src/main/resources/JavaSpring:

api.mustache

beanValidationCore.mustache

model.mustache

Добавим шаблоны

Возьмем шаблоны из Spring генератора openapi-generator.

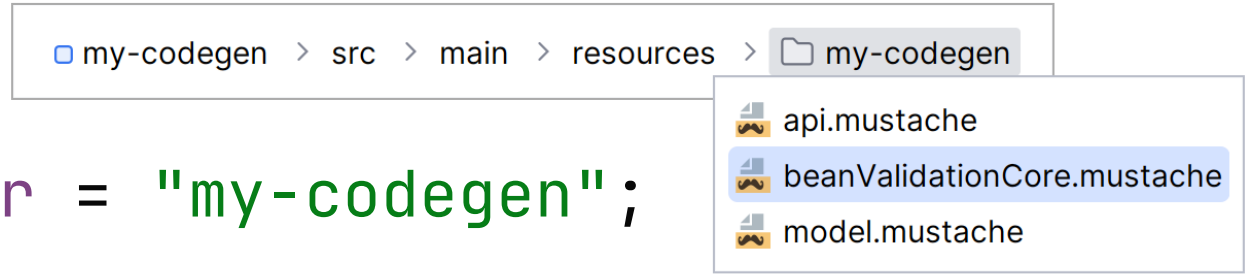
modules/openapi-generator/src/main/resources/JavaSpring:

api.mustache

beanValidationCore.mustache

model.mustache

Скопируем их в наш `templateDir = "my-codegen";`



Создание своего генератора

Добавить в beanValidationCore.mustache следующий текст:

Создание своего генератора

Добавить в beanValidationCore.mustache следующий текст:

```
{{#vendorExtensions.x-customValidation}}
    @pro.axenix.validation.CustomValidation(parameter =
"{{{vendorExtensions.x-customValidation}}}")
{{/vendorExtensions.x-customValidation}}
```

Добавление параметров в свой генератор (1)

```
public MyCodegenGenerator() {
  super();
  templateDir = "my-codegen";

  cliOptions.add(CliOption.newBoolean(

  "generateComments", “text if true", false));

}
```

Добавление параметров в свой генератор (1)

```
public MyCodegenGenerator() {
  super();
  templateDir = "my-codegen";
  cliOptions.add(CliOption.newBoolean(
  "generateComments", “text if true", false));
}
```

Добавление параметров в свой генератор (2)

```
//...
public MyCodegenGenerator() {
   // ...

}
```

Добавление параметров в свой генератор (2)

```
private Boolean defaultValue = true;
//...
public MyCodegenGenerator() {
  // ...

}
```

Добавление параметров в свой генератор (2)

```
private Boolean defaultValue = true;
//...
public MyCodegenGenerator() {
  // ...
  cliOptions.add(CliOption.newBoolean(...,..., defaultValue));
}
```

Добавление параметров в свой генератор (2)

```
private Boolean defaultValue = true;
//...
public MyCodegenGenerator() {
  // ...
  cliOptions.add(CliOption.newBoolean(...,..., defaultValue));
}

@Override
public void processOpts() {
  super.processOpts();
  if (this.additionalProperties.containsKey("defaultValue")) {
    defaultValue = this.convertPropertyToBoolean("defaultValue");
  }
}
```

Изменим шаблон api.mustache

```
{{#generateComments}}
    //generateComments is true. This will add a generated comment
{{/generateComments}}

//defaultValue is {{defaultValue}}
```

Использование собственного генератора с параметрами

```
java -cp "my-codegen.jar;openapi-generator-cli.jar"
org.openapitools.codegen.OpenAPIGenerator
```

Использование собственного генератора с параметрами

```
java -cp "my-codegen.jar;openapi-generator-cli.jar"
org.openapitools.codegen.OpenAPIGenerator
generate -g my-codegen
```

Использование собственного генератора с параметрами

```
java -cp "my-codegen.jar;openapi-generator-cli.jar"
org.openapitools.codegen.OpenAPIGenerator
generate -g my-codegen -i api.yaml -o ./out/myClient
```

Использование собственного генератора с параметрами

```
java -cp "my-codegen.jar;openapi-generator-cli.jar"
org.openapitools.codegen.OpenAPIGenerator
generate -g my-codegen -i api.yaml -o ./out/myClient
--additional-properties=generateComments=true
```

Использование собственного генератора с помощью gradle plugin

В файл build.gradle/build.gradle.kts добавляем зависимость следующим образом:

Использование собственного генератора с помощью gradle plugin

В файл build.gradle/build.gradle.kts добавляем зависимость следующим образом:

```
buildscript {
    repositories{
       mavenLocal()
    }
    dependencies {
       classpath "org.openapitools:my-codegen:1.0.0"
    }
}
```

Использование собственного генератора с помощью gradle plugin

В настройке этапа **`openApiGenerate`** в файле **`build.gradle`** необходимо указать наименование генератора и указать параметр:

Использование собственного генератора с помощью gradle plugin

В настройке этапа **openApiGenerate** в файле **build.gradle** необходимо указать наименование генератора и указать параметр:

```
openApiGenerate {
    generatorName = "my-codegen“
     …
    configOptions = [
            …
          generateComments:"true"
    ]
}
```

Использование собственного генератора с помощью gradle plugin

В настройке этапа **openApiGenerate** в файле **build.gradle** необходимо указать наименование генератора и указать параметр:

```
openApiGenerate {
    generatorName = "my-codegen“
     …
    configOptions = [
        …
        generateComments:"true"
    ]
}
```

Результат генерации (PetsApi, api.mustache)

```
@Generated(...)
@Validated
@Tag(name = "pets", description = "the pets API")
    //generateComments is true. This will add a generated comment

//defaultValue is true

public interface PetsApi {
```

Результат генерации (PetsApi, api.mustache)

```
@Generated(...)
@Validated
@Tag(name = "pets", description = "the pets API")
    //generateComments is true. This will add a generated comment

//defaultValue is true

public interface PetsApi {
```

Результат генерации (Pet, beanVaidationCore.mustache)

```
@NotNull
@pro.axenix.validation.CustomValidation(parameter = "some text")
@Min(5L) @Max(10L)
@Schema(name = "id", requiredMode = Schema.RequiredMode.REQUIRED)
@JsonProperty("id")
public Long getId() {
  return id;
}
```

Результат генерации (Pet, beanVaidationCore.mustache)

```
@NotNull
@pro.axenix.validation.CustomValidation(parameter = "some text")
@Min(5L) @Max(10L)
@Schema(name = "id", requiredMode = Schema.RequiredMode.REQUIRED)
@JsonProperty("id")
public Long getId() {
  return id;
}
```

Что у нас получилось

```
package {{apiPackage}};
```

Что у нас получилось

1. Наименование пакета.

Что у нас получилось

1. Наименование пакета.
2. Импорты.

```
package {{apiPackage}};

{{#imports}}import {{import}};
{{/imports}}

import java.util.Map;
```

Что у нас получилось

1. Наименование пакета.
2. Импорты.
3. Объявление интерфейса

```
package {{apiPackage}};

{{#imports}}import {{import}};
{{/imports}}

import java.util.Map;

public interface {{baseName}}Producer
```

Что у нас получилось

1. Наименование пакета.
2. Импорты.
3. Объявление интерфейса
4. Метод класса – отправка события в топик.

```
package {{apiPackage}};

{{#imports}}import {{import}};
{{/imports}}

import java.util.Map;

public interface {{baseName}}Producer
{

{{#operations}}
    {{#operation}}
        {{#vendorExtensions}}
    void
send{{modelName}}({{modelName}}
{{modelNameCamel}}, Map<String,
String> params);

        {{/vendorExtensions}}
    {{/operation}}
{{/operations}}
}
```

Итог

- Поменять можно абсолютно любой генератор.

Итог

- Поменять можно абсолютно любой генератор.
- Использовать свой генератор можно на всех проектах.

1 OpenAPI Generator

2 Кастомные аннотации

3 Работа с Lombok

4 Свой генератор

5 Кастомизация SpringDocUI

6 Наследование в Swagger

7 Выводы

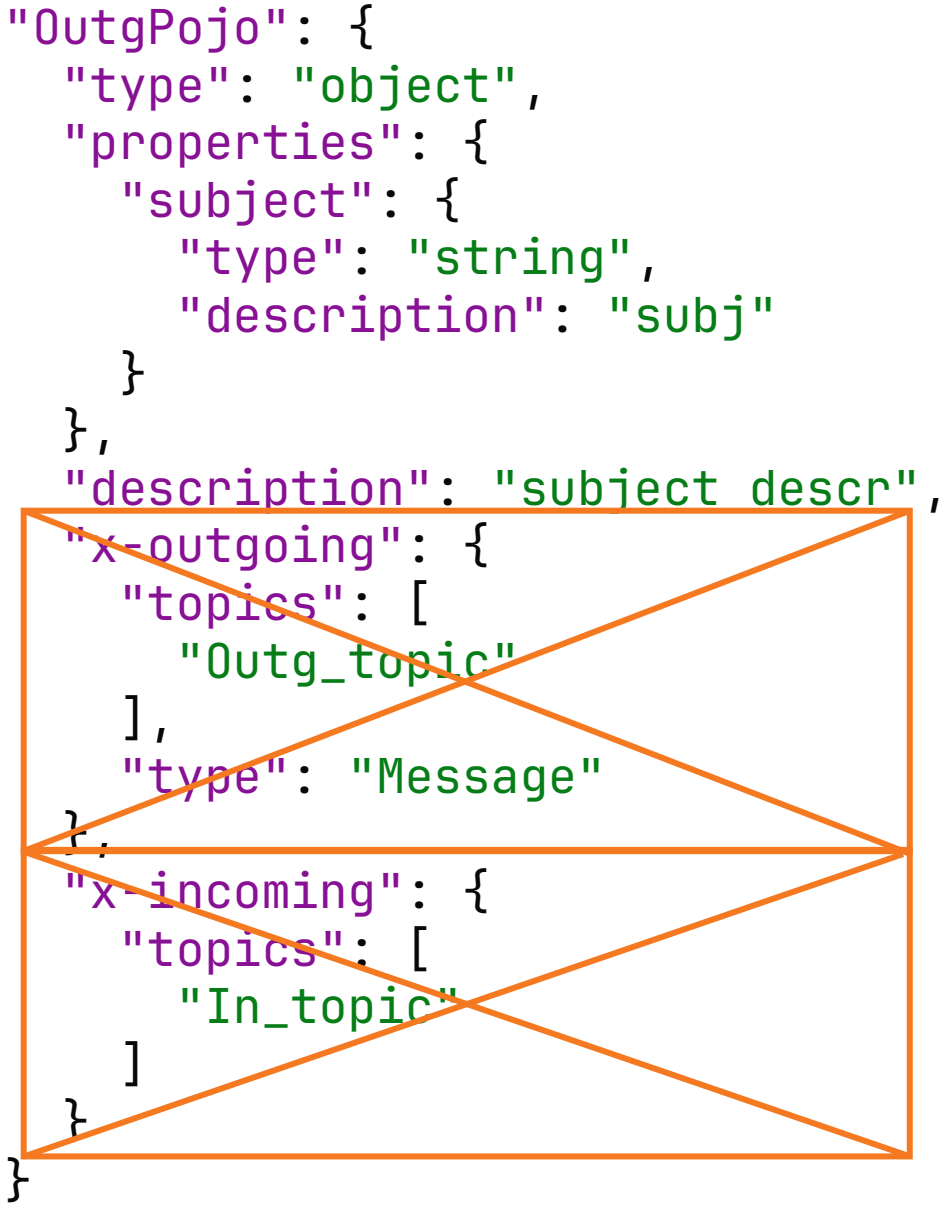
Наша спецификация

- Особенности:
- Блок x-outgoing
- Блок x-incoming

```
"OutgPojo": {
  "type": "object",
  "properties": {
    "subject": {
      "type": "string",
      "description": "subj"
    }
  },
  "description": "subject descr",
  "x-outgoing": {
    "topics": [
      "Outg_topic"
    ],
    "type": "Message"
  },
  "x-incoming": {
    "topics": [
      "In_topic"
    ]
  }
}
```

Наша спецификация, полученная с помощью Swagger

- Особенности:
- Блок x-outgoing
- Блок x-incoming



```
"OutgPojo": {
  "type": "object",
  "properties": {
    "subject": {
      "type": "string",
      "description": "subj"
    }
  },
  "description": "subject descr",
  "x-outgoing": {
    "topics": [
      "Outg_topic"
    ],
    "type": "Message"
  },
  "x-incoming": {
    "topics": [
      "In_topic"
    ]
  }
}
```

Когда нужно кастомизировать SpringDocUI

Если у вас гибридный вариант разработки:

Server – code first, client – API first

Кастомизация SpringDocUI

org.springdoc.core.customizers .**OpenApiCustomizer**	org.springdoc.core.customizers .**GlobalOpenApiCustomizer**
Implement and register a bean of type {@link OpenApiCustomizer} to customize Open api on default OpenAPI description but not on groups *@see GlobalOpenApiCustomizer customize **default** OpenAPI description and **groups***	*Implement and register a bean of type {@link GlobalOpenApiCustomizer} to customize Open api on default OpenAPI description and groups.* *@see OpenApiCustomizer customize **default** OpenAPI description **but not** **groups***

Кастомизация SpringDocUI

org.springdoc.core.customizers .**OpenApiCustomizer**	org.springdoc.core.customizers .**GlobalOpenApiCustomizer**
Implement and register a bean of type {@link OpenApiCustomizer} to customize Open api on default OpenAPI description ***but not on groups*** *@see GlobalOpenApiCustomizer customize* ***default*** *OpenAPI description and* ***groups***	*Implement and register a bean of type {@link GlobalOpenApiCustomizer} to customize Open api* ***on default OpenAPI description and groups.*** *@see OpenApiCustomizer customize* ***default*** *OpenAPI description* ***but not groups***

Кастомизация SpringDocUI

org.springdoc.core.customizers **.OpenApiCustomizer**	org.springdoc.core.customizers **.GlobalOpenApiCustomizer**
Implement and register a bean of type {@link OpenApiCustomizer} to customize Open api on default OpenAPI description ***but not on groups***	*Implement and register a bean of type {@link GlobalOpenApiCustomizer} to customize Open api* ***on default OpenAPI description and groups.***
@see GlobalOpenApiCustomizer customize ***default*** *OpenAPI description and* ***groups***	*@see OpenApiCustomizer customize* ***default*** *OpenAPI description* ***but not groups***

```java
@Configuration
@RequiredArgsConstructor
public class OpenAPIConfig {
    private final Group1OpenAPICustomizer group1OpenAPICustomizer;
    private final Group2OpenAPICustomizer group2OpenAPICustomizer;

    @Bean
    GroupedOpenApi group1() {
        return GroupedOpenApi.builder()
                .group("group1")
                .addOpenApiCustomizer(group1OpenAPICustomizer)
                .pathsToMatch("/**/group1/**").build();
    }

    @Bean
    GroupedOpenApi group2() {
        return GroupedOpenApi.builder()
                .group("group2")
                .addOpenApiCustomizer(group2OpenAPICustomizer)
                .pathsToMatch("/**/group2/**").build();
    }
}
```

```java
@Configuration
@RequiredArgsConstructor
public class OpenAPIConfig {
    private final Group1OpenAPICustomizer group1OpenAPICustomizer;
    private final Group2OpenAPICustomizer group2OpenAPICustomizer;

    @Bean
    GroupedOpenApi group1() {
        return GroupedOpenApi.builder()
                .group("group1")
                .addOpenApiCustomizer(group1OpenAPICustomizer)
                .pathsToMatch("/**/group1/**").build();
    }

    @Bean
    GroupedOpenApi group2() {
        return GroupedOpenApi.builder()
                .group("group2")
                .addOpenApiCustomizer(group2OpenAPICustomizer)
                .pathsToMatch("/**/group2/**").build();
    }
}
```

Задача

1. Добавить блок «incoming» в модель.
2. Добавить «исходящие» модели в спецификацию.
3. Добавить блок «outgoing» в модель.

Добавить «исходящие» модели в спецификацию

```
public class OpenApiCustomizerImpl implements OpenApiCustomizer, EnvironmentAware
{
    private Environment environment;

    @Override
    public void customise(OpenAPI openApi) {
```

Добавить «исходящие» модели в спецификацию

```
public class OpenApiCustomizerImpl implements OpenApiCustomizer, EnvironmentAware
{
    private Environment environment;

    @Override
    public void customise(OpenAPI openApi) {
```

Добавить «исходящие» модели в спецификацию

```
public class OpenApiCustomizerImpl implements OpenApiCustomizer, EnvironmentAware
{
    private Environment environment;

    @Override
    public void customise(OpenAPI openApi) {
        setIncomingXProp(openApi);
```

Добавить «исходящие» модели в спецификацию

```
public class OpenApiCustomizerImpl implements OpenApiCustomizer, EnvironmentAware
{
    private Environment environment;

    @Override
    public void customise(OpenAPI openApi) {
        setIncomingXProp(openApi);
```

Давайте посмотрим внутрь метода `setIncomingXProp(openApi)`

Добавить блок «incoming» в модель

```
Paths paths = openApi.getPaths();
```

Добавить блок «incoming» в модель

```
Paths paths = openApi.getPaths();
// обработали URL’ы получили наименования входящих моделей...
```

Добавить блок «incoming» в модель

```
Paths paths = openApi.getPaths();
// обработали URL’ы получили наименования входящих моделей...
for (Map.Entry<String, List<String>> e : incoming.entrySet(){
     var schema = openApi.getComponents()
                         .getSchemas().get(e.getKey());
```

Добавить блок «incoming» в модель

```
Paths paths = openApi.getPaths();
// обработали URL’ы получили наименования входящих моделей...
for (Map.Entry<String, List<String>> e : incoming.entrySet(){
     var schema = openApi.getComponents()
                         .getSchemas().get(e.getKey());
    if(schema != null) {
        Map<String, Object> extension = new HashMap<>();
        extension.put("topics", e.getValue());
        schema.addExtension("x-incoming", extension);
    }
}
```

Добавить блок «incoming» в модель

```
Paths paths = openApi.getPaths();
// обработали URL’ы получили наименования входящих моделей...
for (Map.Entry<String, List<String>> e : incoming.entrySet(){
     var schema = openApi.getComponents()
                         .getSchemas().get(e.getKey());
    if(schema != null) {
        Map<String, Object> extension = new HashMap<>();
        extension.put("topics", e.getValue());
        schema.addExtension("x-incoming", extension);
    }
}
```

Добавить «исходящие» модели в спецификацию

```
public class OpenApiCustomizerImpl implements OpenApiCustomizer, EnvironmentAware
{
    private Environment environment;
```

Добавить «исходящие» модели в спецификацию

```
public class OpenApiCustomizerImpl implements OpenApiCustomizer, EnvironmentAware
{
    private Environment environment;

    @Override
    public void customise(OpenAPI openApi) {
        setIncomingXProp(openApi);

        var oP = environment.getProperty(Info.PROP_OUTGOING_...);
       // ...
```

Добавить «исходящие» модели в спецификацию

```
public class OpenApiCustomizerImpl implements OpenApiCustomizer, EnvironmentAware
{
    private Environment environment;

    @Override
    public void customise(OpenAPI openApi) {
        setIncomingXProp(openApi);

        var oP = environment.getProperty(Info.PROP_OUTGOING_...);
       // ...
        ClassPathScanningCandidateComponentProvider scanner = new ...(false);
        scanner.addIncludeFilter(new AnnotationTypeFilter(Outgoing.class));
        var bd = scanner.findCandidateComponents(oP);
        // ...



    }
```

Добавить «исходящие» модели в спецификацию

```
public class OpenApiCustomizerImpl implements OpenApiCustomizer, EnvironmentAware
{
    private Environment environment;

    @Override
    public void customise(OpenAPI openApi) {
        setIncomingXProp(openApi);

        var oP = environment.getProperty(Info.PROP_OUTGOING_...);
       // ...
        ClassPathScanningCandidateComponentProvider scanner = new ...(false);
        scanner.addIncludeFilter(new AnnotationTypeFilter(Outgoing.class));
        var bd = scanner.findCandidateComponents(oP);
        // ...
        SpringDocAnnotationsUtils.extractSchema(openApi.getComponents(), cls,
                  null, null, openApi.getSpecVersion());
        //...
    }
```

Добавить «исходящие» модели в спецификацию

```
public class OpenApiCustomizerImpl implements OpenApiCustomizer, EnvironmentAware
{
    private Environment environment;

    @Override
    public void customise(OpenAPI openApi) {
        setIncomingXProp(openApi);

        var oP = environment.getProperty(Info.PROP_OUTGOING_...);
       // ...
        ClassPathScanningCandidateComponentProvider scanner = new ...(false);
        scanner.addIncludeFilter(new AnnotationTypeFilter(Outgoing.class));
        var bd = scanner.findCandidateComponents(oP);
        // ...
        SpringDocAnnotationsUtils.extractSchema(openApi.getComponents(), cls,
                null, null, openApi.getSpecVersion());
        //...
    }
```

OpenApiCustomizer vs ModelResolver

org.springdoc.core.customizers **.OpenApiCustomizer или** **GlobalOpenApiCustomizer**	io.swagger.v3.core.jackson**.ModelResolver**
Подходит для того, чтобы работать над общей структурой спецификации – добавить элементы, изменить опираясь на информацию уже добавленную в спецификацию.	**Подходит для кастомизации именно генерации модели. Есть прямой доступ ко всем аннотациям класса модели, к его полям и методам.**
Для использования достаточно объявить Bean	Для использования достаточно объявить Bean

Добавить блок «outgoing» в модель.

- Добавили свой класс наследник ModelResolver.
- Перегрузили метод resolve – возвращает схему для каждого класса модели.
- ModelResolver имплементирует ModelConverter.

Добавить блок «outgoing» в модель.

```
public Schema resolve(AnnotatedType annotatedType,
                      ModelConverterContext context,
                      Iterator<ModelConverter> next) {
    var model = super.resolve(annotatedType, context, next);
```

```
}
```

Добавить блок «outgoing» в модель.

```
public Schema resolve(AnnotatedType annotatedType,
                      ModelConverterContext context,
                      Iterator<ModelConverter> next) {
    var model = super.resolve(annotatedType, context, next);
    // получили класс модели  Class<Object> type из annotatedType
```

```
}
```

Добавить блок «outgoing» в модель.

```
public Schema resolve(AnnotatedType annotatedType,
                      ModelConverterContext context,
                      Iterator<ModelConverter> next) {
    var model = super.resolve(annotatedType, context, next);
    // получили класс модели  Class<Object> type из annotatedType
    // с помощью AnnotationUtils нашли аннотацию annoOutgoing
```

```
}
```

Добавить блок «outgoing» в модель.

```
public Schema resolve(AnnotatedType annotatedType,
                      ModelConverterContext context,
                      Iterator<ModelConverter> next) {
    var model = super.resolve(annotatedType, context, next);
    // получили класс модели  Class<Object> type из annotatedType
    // с помощью AnnotationUtils нашли аннотацию annoOutgoing
    // из context нашли схему schema
```

```
}
```

Добавить блок «outgoing» в модель.

```
public Schema resolve(AnnotatedType annotatedType,
                      ModelConverterContext context,
                      Iterator<ModelConverter> next) {
    var model = super.resolve(annotatedType, context, next);
    // получили класс модели  Class<Object> type из annotatedType
    // с помощью AnnotationUtils нашли аннотацию annoOutgoing
    // из context нашли схему schema
    var extensions = schema.getExtensions()
    if (extensions == null) {
        extensions = new HashMap<>();
        schema.setExtensions(extensions);
    }



}
```

Добавить блок «outgoing» в модель.

```
public Schema resolve(AnnotatedType annotatedType,
                      ModelConverterContext context,
                      Iterator<ModelConverter> next) {
    var model = super.resolve(annotatedType, context, next);
    // получили класс модели  Class<Object> type из annotatedType
    // с помощью AnnotationUtils нашли аннотацию annoOutgoing
    // из context нашли схему schema
    var extensions = schema.getExtensions()
    if (extensions == null) {
        extensions = new HashMap<>();
        schema.setExtensions(extensions);
    }
    outgoingMap.put(TOPICS_PROP, annoOutgoing.topics());
    outgoingMap.put(TAGS_PROP, annoOutgoing.tags());
    outgoingMap.put(TYPE_PROP, annoOutgoing.type().getText());
    return model;
}
```

Добавить блок «outgoing» в модель.

```
public Schema resolve(AnnotatedType annotatedType,
                      ModelConverterContext context,
                      Iterator<ModelConverter> next) {
    var model = super.resolve(annotatedType, context, next);
    // получили класс модели  Class<Object> type из annotatedType
    // с помощью AnnotationUtils нашли аннотацию annoOutgoing
    // из context нашли схему schema
    var extensions = schema.getExtensions()
    if (extensions == null) {
        extensions = new HashMap<>();
        schema.setExtensions(extensions);
    }
    outgoingMap.put(TOPICS_PROP, annoOutgoing.topics());
    outgoingMap.put(TAGS_PROP, annoOutgoing.tags());
    outgoingMap.put(TYPE_PROP, annoOutgoing.type().getText());
    return model;
}
```

Не меняли модель

Добавить блок «outgoing» в модель.

```
public Schema resolve(AnnotatedType annotatedType,
                      ModelConverterContext context,
                      Iterator<ModelConverter> next) {
    var model = super.resolve(annotatedType, context, next);
    // получили класс модели  Class<Object> type из annotatedType
    // с помощью AnnotationUtils нашли аннотацию annoOutgoing
    // из context нашли схему schema
    var extensions = schema.getExtensions()
    if (extensions == null) {
        extensions = new HashMap<>();
        schema.setExtensions(extensions);
    }
    outgoingMap.put(TOPICS_PROP, annoOutgoing.topics());
    outgoingMap.put(TAGS_PROP, annoOutgoing.tags());
    outgoingMap.put(TYPE_PROP, annoOutgoing.type().getText());
    return model;
}
```

Вся магия тут

Не меняли модель

Добавить блок «outgoing» в модель.

```
var schema =
context.getDefinedModels().entrySet().stream().findFirst()
        .map(Map.Entry::getValue).orElse(null);
```

1. Имена моделей чаще всего уникальны – поэтому одна схема в контексте.
2. После получения схемы из контекста вся работа идет с ней.

Другие ModelConverter в SpringDocUI

- PageableOpenAPIConverter
- PageOpenAPIConverter
- JavaTypeToIgnoreConverter
- AdditionalModelsConverter
- FileSupportConverter
- CollectionModelContentConverter
- JavadocPropertyCustomizer
- PolymorphicModelConverter
- PropertyCustomizingConverter
- ResponseSupportConverter
- SchemaPropertyDeprecatingConverter
- WebFluxSupportConverter
- SortOpenAPIConverter

Другие ModelConverter в SpringDocUI

PageableOpenAPIConverter

PolymorphicModelConverter

PageOpenAPIConverter

PropertyCustomizingConverter

JavaTypeToIgnoreConverter

ResponseSupportConverter

AdditionalModelsConverter

SchemaPropertyDeprecatingConverter

FileSupportConverter

WebFluxSupportConverter

CollectionModelContentConverter

SortOpenAPIConverter

JavadocPropertyCustomizer

Другие ModelConverter в SpringDocUI

PageableOpenAPIConverter

PolymorphicModelConverter

PageOpenAPIConverter

PropertyCustomizingConverter

JavaTypeToIgnoreConverter

ResponseSupportConverter

AdditionalModelsConverter

SchemaPropertyDeprecatingConverter

FileSupportConverter

WebFluxSupportConverter

CollectionModelContentConverter

SortOpenAPIConverter

JavadocPropertyCustomizer

Итог

Есть множество задач, которые требуют кастомизировать SpringDoc UI.

Давайте рассмотрим еще один пример.

1 OpenAPI Generator

2 Кастомные аннотации

3 Работа с Lombok

4 Свой генератор

5 Кастомизация SpringDocUI

6 Наследование в Swagger

7 Выводы

Схема классов

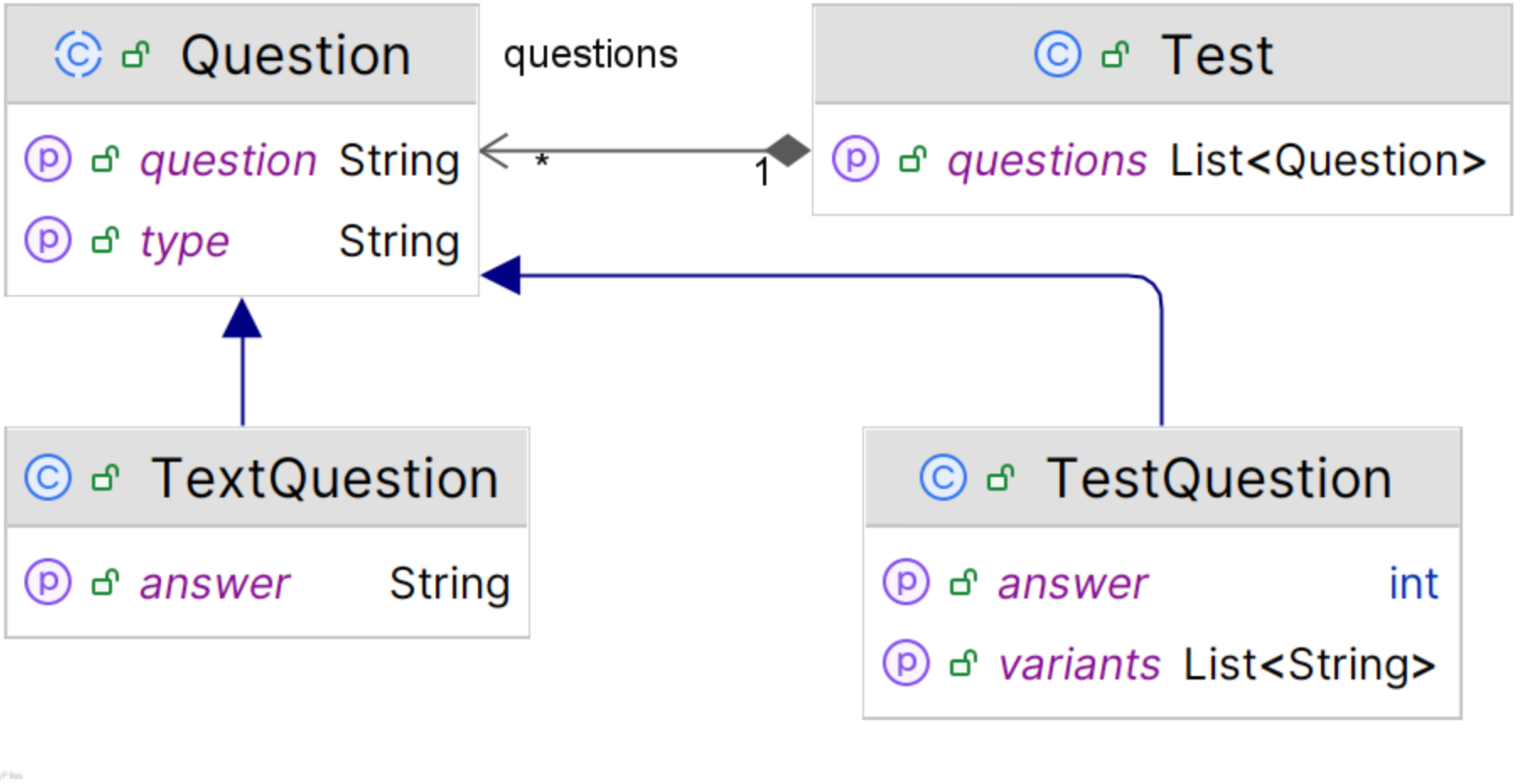


Схема классов

```
@JsonTypeInfo(
        use = JsonTypeInfo.Id.NAME,
        include = JsonTypeInfo.As.PROPERTY,
        property = "type")
@JsonSubTypes({
        @JsonSubTypes.Type(value = TestQuestion.class, name = "test"),
        @JsonSubTypes.Type(value = TextQuestion.class, name = "text")
})
public abstract class Question {
    // Fields: question, type ...
}
```

Схема классов

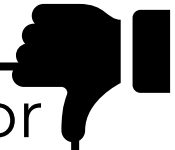
```
@JsonTypeInfo(
        use = JsonTypeInfo.Id.NAME,
        include = JsonTypeInfo.As.PROPERTY,
        property = "type")
@JsonSubTypes({
        @JsonSubTypes.Type(value = TestQuestion.class, name = "test"),
        @JsonSubTypes.Type(value = TextQuestion.class, name = "text")
})
public abstract class Question {
    // Fields: question, type ...
}
```

Сгенерируем спецификацию

```
Question:
  type: object
  properties:
    question:
      type: string
    type:
      type: string
  discriminator:
    propertyName: type
```

Сгенерируем спецификацию

Нет значений discriminator в спецификации



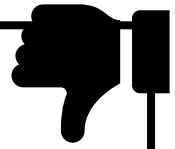
```yaml
Question:
  type: object
  properties:
    question:
      type: string
    type:
      type: string
  discriminator:
    propertyName: type
```

Добавим больше информации

```
@JsonTypeInfo(
        use = JsonTypeInfo.Id.NAME,
        include = JsonTypeInfo.As.PROPERTY,
        property = "type")
@JsonSubTypes({
        @JsonSubTypes.Type(value = TestQuestion.class, name = "test"),
        @JsonSubTypes.Type(value = TextQuestion.class, name = "text")
})
@Schema(
        description = "Question",
        discriminatorProperty = "type", discriminatorMapping = {
        @DiscriminatorMapping(value = "test", schema = TestQuestion.class),
        @DiscriminatorMapping(value = "text", schema = TextQuestion.class)
})
public abstract class Question {
    // Fields: question, type ...
}
```

Добавим больше информации

```
@JsonTypeInfo(
        use = JsonTypeInfo.Id.NAME,
        include = JsonTypeInfo.As.PROPERTY,
        property = "type")
@JsonSubTypes({
        @JsonSubTypes.Type(value = TestQuestion.class, name = "test"),
        @JsonSubTypes.Type(value = TextQuestion.class, name = "text")
})
@Schema(
        description = "Question",
        discriminatorProperty = "type", discriminatorMapping = {
        @DiscriminatorMapping(value = "test", schema = TestQuestion.class),
        @DiscriminatorMapping(value = "text", schema = TextQuestion.class)
})
public abstract class Question {
    // Fields: question, type ...
}
```



Хотим меньше аннотаций

Добавим реализацию GlobalOpenApiCustomizer

Хотим меньше аннотаций

Добавим реализацию GlobalOpenApiCustomizer

```
@Component
public class OpenApiCustomizerImpl implements
GlobalOpenApiCustomizer {
    @Override
    public void customise(OpenAPI openApi) {
//...
```

Создаем сканнер

Создаем сканнер

1. Создадим `ClassPathScanningCandidateComponentProvider`

Создаем сканнер

1. Создадим `ClassPathScanningCandidateComponentProvider`
2. Перегрузим метод

   `isCandidateComponent(AnnotatedBeanDefinition bn)`

Создаем сканнер

1. Создадим `ClassPathScanningCandidateComponentProvider`
2. Перегрузим метод

```
    isCandidateComponent(AnnotatedBeanDefinition bn)
Class...Provider p = new ...(false) {
    @Override
    protected boolean isCandidateComponent
                                (AnnotatedBeanDefinition bn) {
        return super.isCandidateComponent(bn) ||
                    bn.getMetadata().isAbstract();
    }
};
```

Создаем сканнер

1. Создадим `ClassPathScanningCandidateComponentProvider`
2. Перегрузим метод

```
    isCandidateComponent(AnnotatedBeanDefinition bn)
Class...Provider p = new ...(false) {
    @Override
    protected boolean isCandidateComponent
                                (AnnotatedBeanDefinition bn) {
        return super.isCandidateComponent(bn) ||
                bn.getMetadata().isAbstract();
    }
};
```

Находим классы

```
p.addIncludeFilter(new AnnotationTypeFilter (JsonSubTypes.class));
var bd = p.findCandidateComponents("your.package");
```

Для каждого найденного класса находим схему

```
openApi.getComponents().getSchemas().get(cls.getSimpleName());
```

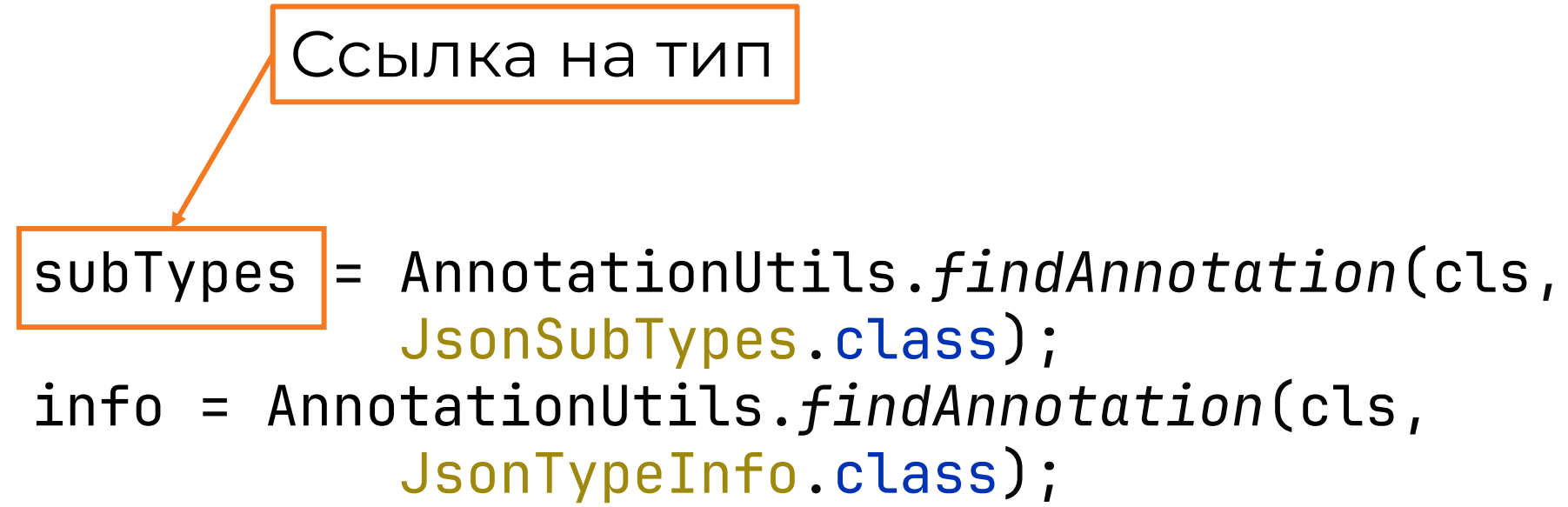
Для каждого найденного класса находим схему

```
openApi.getComponents().getSchemas().get(cls.getSimpleName());
```

Находим значения аннотаций

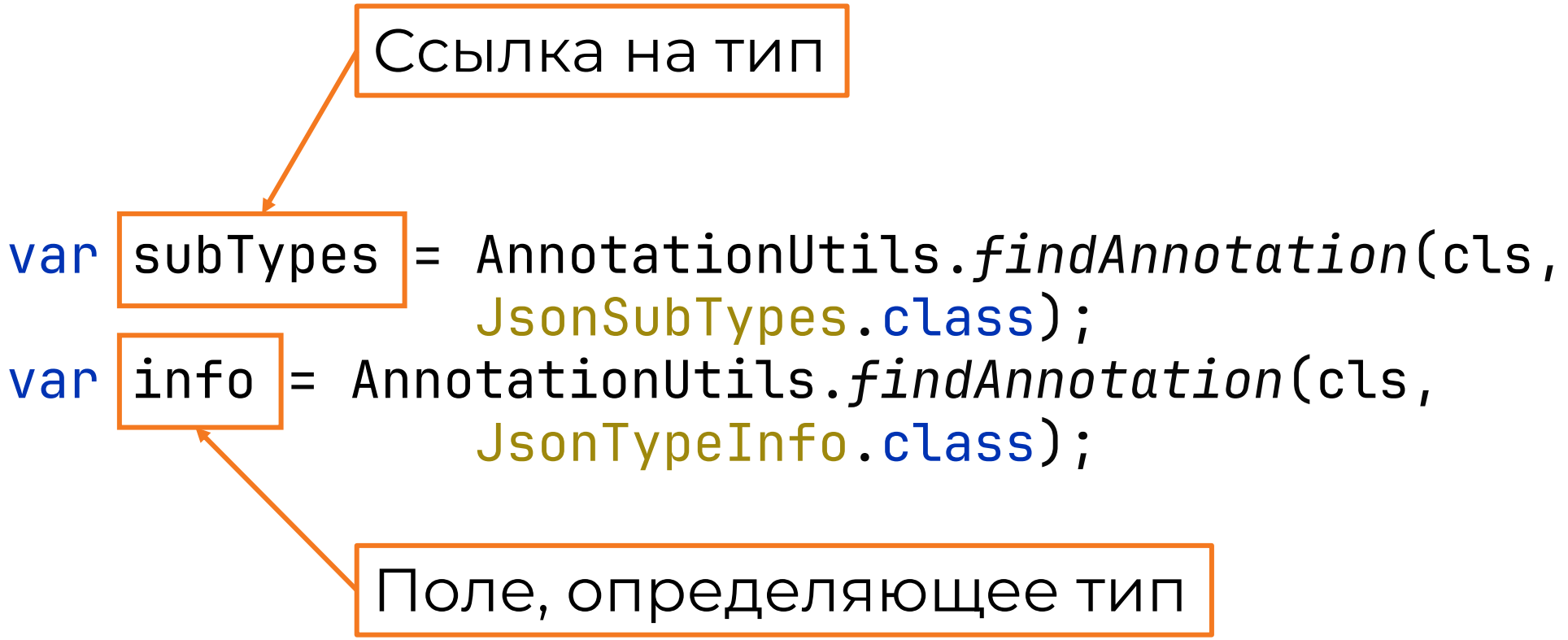
```
var subTypes = AnnotationUtils.findAnnotation(cls,
                    JsonSubTypes.class);
var info = AnnotationUtils.findAnnotation(cls,
                    JsonTypeInfo.class);
```

Находим значения аннотаций




```
var subTypes = AnnotationUtils.findAnnotation(cls,
                    JsonSubTypes.class);
var info = AnnotationUtils.findAnnotation(cls,
                    JsonTypeInfo.class);
```

Находим значения аннотаций



Добавляем discriminator

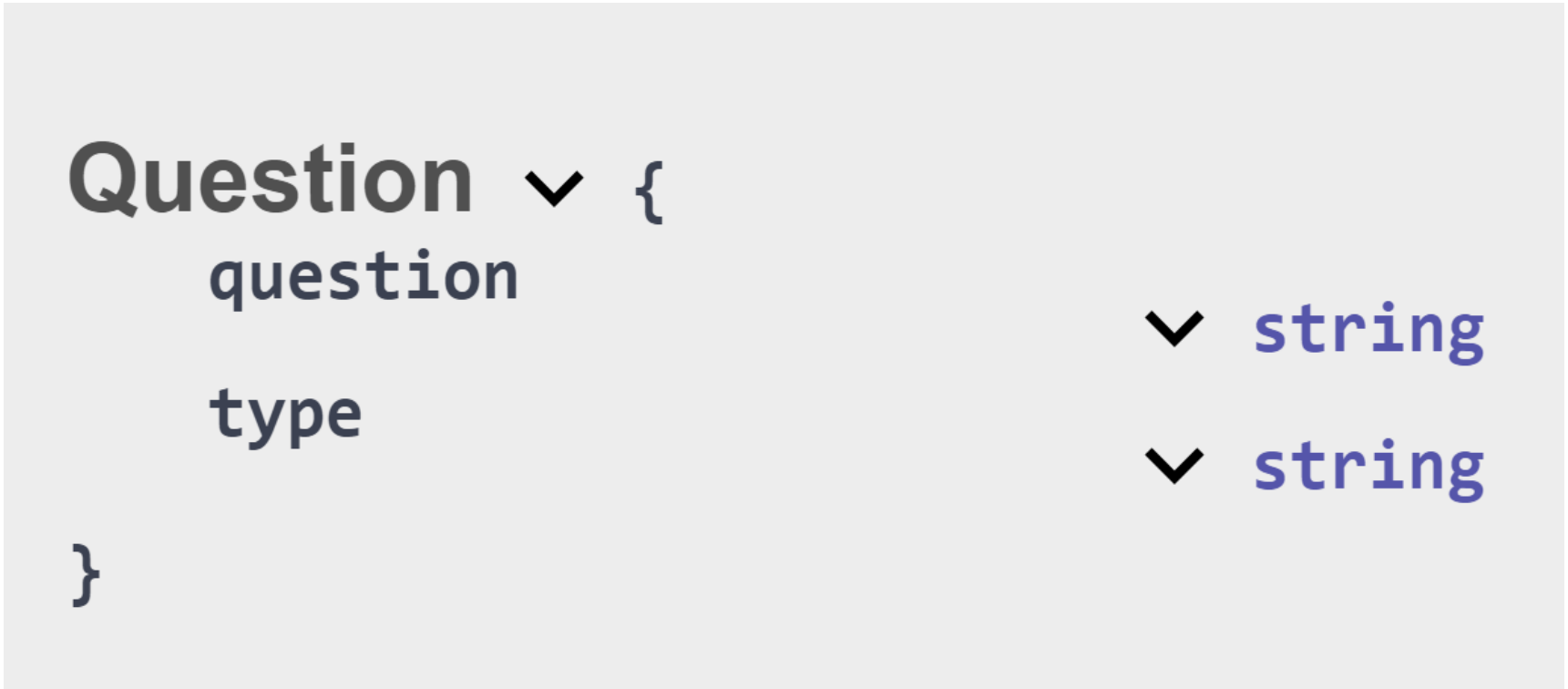
```
schema.getDiscriminator().mapping(name,
"#/components/schemas/" + child.getName());
```

Добавляем discriminator

```
schema.getDiscriminator().mapping(name,
"#/components/schemas/" + child.getName());
```

И немного внешнего вида

```
springdoc.api-docs.version=openapi_3_0
```



И немного внешнего вида

```
springdoc.api-docs.version=openapi_3_1
```

Question ^ Collapse all **object**

question **string**

type **string**

Discriminator ^ Collapse all type **object**

test=#/components/schemas/TestQuestion

text=#/components/schemas/TextQuestion

Тоже самое можно сделать с помощью ModelResolver

```
public Schema resolve(...Iterator<ModelConverter> chain) {
    super.resolve(annotatedType, context, chain);
    JavaType type = this._mapper.constructType(annotatedType.getType());
    if (type.isAbstract()) {
        var schema = // получаем схему из контекста ...
        JsonTypeInfo info = ...
        JsonSubTypes subTypes = ...
        if (schema != null && info != null && subTypes != null) {
            String property = info.property();
            for (JsonSubTypes.Type t : subTypes.value()) {
                String name = t.name();
                // ставим discriminator в schema из контекста ...
            }
        }
    }
    return (chain.hasNext()) ?
            chain.next().resolve(annotatedType, context, chain) : null;
}
```

Тоже самое можно сделать с помощью ModelResolver

```
public Schema resolve(...Iterator<ModelConverter> chain) {
    super.resolve(annotatedType, context, chain);
    JavaType type = this._mapper.constructType(annotatedType.getType());
    if (type.isAbstract()) {
        var schema = // получаем схему из контекста ...
        JsonTypeInfo info = ...
        JsonSubTypes subTypes = ...
        if (schema != null && info != null && subTypes != null) {
            String property = info.property();
            for (JsonSubTypes.Type t : subTypes.value()) {
                String name = t.name();
                // ставим discriminator в schema из контекста ...
            }
        }
    }
    return (chain.hasNext()) ?
            chain.next().resolve(annotatedType, context, chain) : null;
}
```

Итог

1. Можно внедрить обработку любой аннотации.
2. С помощью ModelResolver удобнее работать именно с генерацией schema по классу-модели.
3. С Помощью OpenApiCustomizer и GlobalOpenApiCustomizer можно работать со спецификацией целиком.

1 OpenAPI Generator

2 Кастомные аннотации

3 Работа с Lombok

4 Свой генератор

5 Кастомизация SpringDocUI

6 Наследование в Swagger

7 Выводы

Выводы

Выводы

- Можно добавить любые аннотации в спецификацию OpenAPI.

Выводы

- Можно добавить любые аннотации в спецификацию OpenAPI.
- Можно сделать свой генератор.

Выводы

- Можно добавить любые аннотации в спецификацию OpenAPI.
- Можно сделать свой генератор.
- DTO с наследованием корректно визуализируются.

Выводы

- Можно добавить любые аннотации в спецификацию OpenAPI.
- Можно сделать свой генератор.
- DTO с наследованием корректно визуализируются.
- Есть множество способов кастомизировать SpringDocUI.

axenix
innovation

Спасибо!

@AlexandraVolushkova

https://github.com/SashaVolushkova/Joker-2024-Links

О компании ООО «Аксеникс Инновации» – российская компания, входит в группу **«AXENIX»**, резидент Инновационного центра «Сколково», ведущая научно-технические исследования в области информационных технологий и предоставляющая на базе собственных разработок широкий спектр профессиональных услуг, направленных на цифровизацию бизнеса.

Подробнее на axenix-innovation.pro