

# OA13を使った Django REST frameworkの ドキュメント生成とカスタマイズ

Shirai Hono  
Django congress JP 2022

# 本発表のポイント

- Django REST framework (DRF) でドキュメント生成をする
- ドキュメント生成にはOpenAPI3 (OAI3) を利用する
- DRFのドキュメント生成の仕組みを (なんとなく) 理解する
- **内部実装**を確認しながらカスタマイズする

# TL; DR

ちゃちゃっとドキュメント化したい人は **drf-spectacular** を使いましょう

<https://github.com/tfranzel/drf-spectacular>

多分これが一番早いと思います

# サンプルコード

Github にあります

<https://github.com/shihono/drf-doc-demo>

このコードを例に進めます

<https://github.com/shihono/drf-doc-demo>

# 自己紹介: Shirai Hono (白井 穂乃)

## 経歴

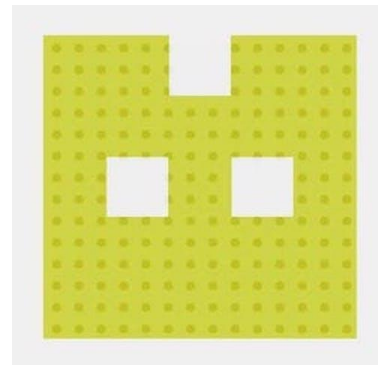
- 2017 - 2019: 修士 (情報・自然言語処理)
- 2019 - 2020: データサイエンティスト@データ分析会社
- 2020 - Now: エンジニア@日本経済新聞社

## 主な仕事: NLP関連のツール開発

- 記事校正
- 記事の読み原稿化
- Nikkei Waveアプリ



[@sh1\\_hono](https://twitter.com/sh1_hono)



Github [shihono](https://github.com/shihono)

<https://github.com/shihono/drif-doc-demo>

# 目次

- 前提知識: DRFとOAI3
- DRFの標準機能でドキュメント生成
- ドキュメントを独自カスタマイズ
- 3rd party library: drf-spectacular

# 前提知識: DRFとOAI3

# Django REST framework (DRF)

- <https://www.django-rest-framework.org/>
- RESTful な Web API を構築するためのフレームワーク
  - REpresentational State Transfer
- 通常のDjangoとの差分
  - APIView
  - Serializer



# DRF > Serializers

- Serializerクラス: djangoのmodel formのようにデータ構造を定義
- データ型はFieldクラスで指定

```
from rest_framework import serializers

class UserSerializer(serializers.Serializer):
    id = serializers.IntegerField()
    username = serializers.CharField()

s = UserSerializer(data={"id": 1, "username": "Shirai Hono"})

s.is_valid()
>> True

s.validated_data
>> OrderedDict([('id', 1), ('username', 'Shirai Hono')])
```

# OpenAPI (OAI)

- > The OpenAPI Specification (OAS) defines a standard, language-agnostic interface to HTTP APIs
  - <https://spec.openapis.org/oas/latest.html#introduction> より
  - 本発表ではOAIと表記します
  - Web API の仕様（リクエスト・レスポンス・パス等）を記述する形式
- swagger: OAIを読み込んでドキュメント生成ができるツール群
  - e.g. swagger-ui <https://github.com/swagger-api/swagger-ui>

Swagger Editor. Supported by SMARTBEAR File Edit Insert Generate Server Generate Client About Try our new Editor

Servers  
https://petstore3.swagger.io/api/v3 Authorize

**pet** Everything about your Pets Find out more

- PUT** /pet Update an existing pet
- POST** /pet Add a new pet to the store
- GET** /pet/findByStatus Finds Pets by status
- GET** /pet/findByTags Finds Pets by tags
- GET** /pet/{petId} Find pet by ID
- POST** /pet/{petId} Updates a pet in the store with form data
- DELETE** /pet/{petId} Deletes a pet
- POST** /pet/{petId}/uploadImage uploads an image

**store** Access to Petstore orders Find out more about our store

swaggerを使ったUI例 <https://editor.swagger.io/>

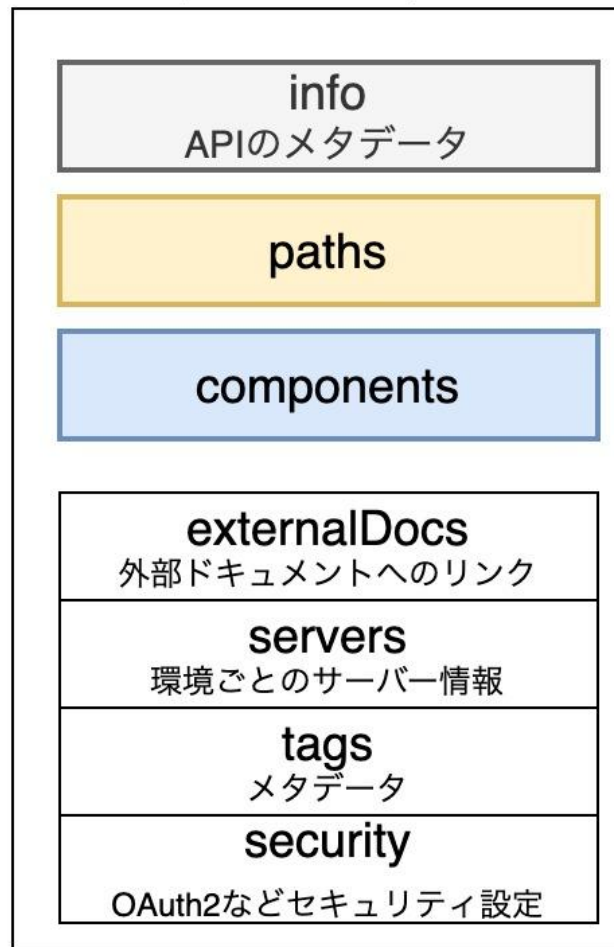
# OpenAPI 3 (OAI3)

## バージョン3の構成

### 今回主に扱う要素

- info
  - APIのメタデータ
  - e.g. title, summary, version
- paths
- components

## OpenAPI object



# OAI3 > paths

エンドポイント。methodごとに**operation object**として以下を定義する

- paramters
- requestBody
- response

```
40 paths:
41   /pet:
42     put:
43       tags:
44         - pet
45       summary: Update an existing pet
46       description: Update an existing pet by Id
47       operationId: updatePet
48       requestBody:
49         description: Update an existent pet in the store
50         content:
51           application/json:
52             schema:
53               $ref: '#/components/schemas/Pet'
54           application/xml:
55             schema:
56               $ref: '#/components/schemas/Pet'
57           application/x-www-form-urlencoded:
58             schema:
59               $ref: '#/components/schemas/Pet'
```

**pet** Everything about your Pets

[Find out more](#) ^

**PUT** /pet Update an existing pet

**POST** /pet Add a new pet to the store

**GET** /pet/findByStatus Finds Pets by status

**GET** /pet/findByTags Finds Pets by tags

**GET** /pet/{petId} Find pet by ID

# OAI3 > paths > parameters

- 主にGETのrequest情報

## required

- name: parameter名
- in: parameterの種類
  - query パラメーター `/users/?page=3`
  - path パラメーター `/users/123/`
- required: 必須parameterかどうか

## optional

- schema
  - パラメーターのデータ
  - 後述

```
parameters:  
- name: petId  
  in: path  
  description: ID of pet to update  
  required: true  
  schema:  
    type: integer  
  
- name: additionalMetadata  
  in: query  
  required: false  
  schema:  
    type: string
```

# OAI3 > paths > requestBody

- 主にPOSTのrequest情報

required

- content: media typeをkey, schemaをvalueとする
  - media type は json, xml, plain\_text など
  - schemaは parametersと同様

optional

- required
- description

```
requestBody:
  description: user object
  content:
    application/json:
      schema:
        type: object
        properties:
          id:
            type: integer
          username:
            type: string
    application/xml:
      schema:
        $ref: '#/components/schemas/User'
```

# OAI3 > paths > response

- 返却値

requestBody とほぼ同じ

status codeごとに contentを定義できる

```
responses:  
  '200':  
    description: success  
    content:  
      application/json:  
        schema:  
          $ref: '#/components/schemas/User'  
  '400':  
    description: Bad request  
  '404':  
    description: Not found
```



# OAI3 > components > schemas

- pathsで使うschemaを定義
- \$ref で参照
- 同じ schemaを使いまわせる

```
requestBody:  
  description: user object  
  content:  
    application/json:  
      schema:  
        $ref: '#/components/schemas/User'  
components:  
  schemas:  
    User:  
      type: object  
      properties:  
        id:  
          type: integer  
        username:  
          type: string
```

# OAI3 > components > schemas の データタイプ

type で指定できるデータ型

- string: str
- integer: int
- number: float
- boolean: bool
- object: dict
  - properties で key,value のschemaを指定
- array: list
  - items でリストの中身を指定

```
User:
  type: object
  properties:
    id:
      type: integer
    username:
      type: string
UserList:
  type: array
  items:
    $ref: '#/components/schemas/User'
```

# DRFのserializer と OAI3のschema、噛み合いそうな予感

```
class UserSerializer(serializers.Serializer):  
    id = serializers.IntegerField()  
    username = serializers.CharField()
```

```
User:  
  type: object  
  properties:  
    id:  
      type: integer  
    username:  
      type: string
```



# サンプルコードの設定

# django project settings

## エンドポイント

- GET /api/converter
- POST /api/converter
- GET /api/alphabets

[GitHub - shihono/alphabet2kana: Convert English alphabet to Katakana](#)

の実行結果を返すだけのAPI

# view.py: View クラスは RetrieveAPIView を継承

今回データベースは使わないため、read-only のクラスを利用

```
from rest_framework.generics import RetrieveAPIView

class ConverterView(RetrieveAPIView):
    serializer_class = ConverterRequestSerializer

    def get(self, request, *args, **kwargs):
        """アルファベットをカタカナに変換するGET method"""
        data = request.GET
        return self.convert(data)

# drf_doc_demo/api/views.py
```

# serializers.py: request, responseはserializerで設定

converter 用、alphabets用それぞれ用意

```
from rest_framework import serializers

class ConverterRequestSerializer(serializers.Serializer):
    text = serializers.CharField()
    delimiter = serializers.CharField(required=False, help_text="区切り文字")
    numeral = serializers.BooleanField(required=False, help_text="数字も変換するフラグ")

class ConvertResponseSerializer(serializers.Serializer):
    text = serializers.CharField()

# drf_doc_demo/api/serializers.py
```

# 実行例

<http://127.0.0.1:8000/api/converter/?text=ABC>

Django REST framework

## Converter

OPTIONS

GET

GET /api/converter/?text=ABC

HTTP 200 OK  
Allow: GET, POST, HEAD, OPTIONS  
Content-Type: application/json  
Vary: Accept

```
{  
  "text": "エービーシー"  
}
```



# DRF標準機能でドキュメント生成 swagger-ui で表示する

# SchemaView

- DRF には SchemaView が用意されている
  - [Schemas - Django REST framework](#)
- 動的にスキーマ (データ構造) を生成できる
  - 実行時のエンドポイント・Viewクラスに従った結果が出力できる
  - ファイルで出力する必要がない
  - ※ややこしいですが、OAI のschemaとは異なります
- get\_schema\_view : SchemaViewを設定できる便利関数

# get\_schema\_view で SchemaView を使う

- url.py の urlpatterns に追加
- 引数で OpenAPI 3 の info を設定

```
from rest_framework.schemas import get_schema_view

urlpatterns = [
    path("api/", include("api.urls")),
    path('openapi/', get_schema_view(
        title="drf-doc-demo Project",
        description="API for drf doc",
        version="1.0.0"
    ), name='openapi-schema'),
]

# drf_doc_demo/drf_doc_demo/urls.py
```

# get\_schema\_view でSchemaViewを使う

<http://127.0.0.1:8000/openapi> で表示。

info, pathsが含まれる OAI形式のyaml

Django REST framework

Schema

## Schema

OPTIONS

GET

GET /openapi/

HTTP 200 OK

Allow: GET, HEAD, OPTIONS

Content-Type: application/vnd.oai.openapi

Vary: Accept

openapi: 3.0.2

info:

title: drf-doc-demo Project

version: 1.0.0

description: API for drf doc

paths:

/api/converter/:

get:

operationId: retrieveConverterRequest

description: "\u30A2\u30EB\u30D5\u30A1\u30D9\u30C3\u30C8\u3092\u30AB\u30BF\u306B\u3099\u3099\u3068\u3059\u3088GET method"

parameters: []

# swagger-ui で SchemaView を表示する

- swagger-ui を利用しドキュメント化
  - [Documenting your API - Django REST framework](#)
- TemplateView を使って swagger-ui を読み込む
  - HTML ファイル swagger-ui.html を準備
  - 最新は [swagger-ui/installation.md at master](#) で確認すること

```
<div id="swagger-ui"></div>
<script src="https://unpkg.com/swagger-ui-dist@4.5.0/swagger-ui-bundle.js" crossorigin></script>
<script>
  window.onload = () => {
    window.ui = SwaggerUIBundle({
      url: "{% url schema_url %}",
      dom_id: '#swagger-ui',
    });
  };
</script>
# drf_doc_demo/templates/swagger-ui.html
```

# swagger-ui で SchemaView を表示する

- urlpatterns に追加
  - extra\_content で schema\_url に SchemaView のpath名を渡す

```
from django.views.generic import TemplateView

urlpatterns = [
    # 追加
    path('swagger-ui/', TemplateView.as_view(
        template_name='swagger-ui.html',
        extra_context={'schema_url': 'openapi-schema'}
    ), name='swagger-ui'),
]

# drf_doc_demo/drf_doc_demo/urls.py
```

# swagger-ui で SchemaView を表示する

- <http://127.0.0.1:8000/swagger-ui/>

## drf-doc-demo Project 1.0.0 OAS3

/openapi/

API for drf doc

### api



GET /api/converter/



POST /api/converter/



GET /api/alphabet/



# swagger-ui で SchemaView を表示する

- docstringのテキストも表示される
  - docstringをドキュメントがわりに使える

## drf-doc-demo Project 1.0.0 OAS3

/openapi/

API for drf doc

### api

GET /api/converter/

アルファベットをカタカナに変換するGET method

```
class ConverterView(RetrieveAPIView):
    def get(self, request, *args, **kwargs):
        """アルファベットをカタカナに変換するGET method"""
        data = request.GET
        return self.convert(data)

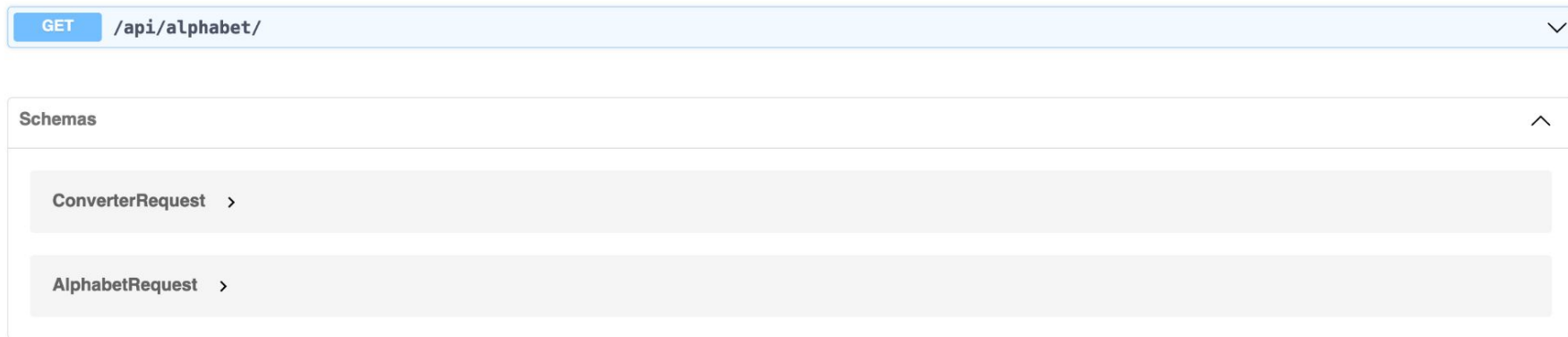
# drf_doc_demo/api/views.py
```



独自にカスタマイズする  
……ために  
SchemaViewを理解する

## 再掲: OAI3 の要素

- info → get\_schema\_view で設定可能
- paths operation object → 自動で生成？
- components schema object → 自動で生成？
  - request用のschemasが設定されている
  - どうやって設定された？



>> いろいろおかしい <<

### Responses

Code	Description	Links
200	<p>Media type</p> <div>application/json</div> <p>Controls Accept header.</p> <p>Example Value   Schema</p> <pre>{   "text": "string",   "delimiter": "string",   "numeral": true }</pre>	No links

POST /api/converter/

アルファベットをカタカナに変換するPOST method

#### Parameters

No parameters

Try it out

Request body

application/json

Example Value | Schema

```
{
  "text": "string",
  "delimiter": "string",
  "numeral": true
}
```

# SchemaViewの仕組み

SchemaViewのgetをみる

```
def get(self, request, *args, **kwargs):  
    schema = self.schema_generator.get_schema(request, self.public)  
    if schema is None:  
        raise exceptions.PermissionDenied()  
    return Response(schema)
```

# rest\_framework/schemas/views.py

**self.schema\_generator.get\_schema** でschemaを生成できるっぽい

- schema\_generator ??
- get\_schema ??

# SchemaGenerator

- SchemaViewのインスタンス要素
  - `get_schema_view` の引数で渡すことが可能
- 名前の通り、スキーマを生成するクラス
  - 記述形式に対応したgeneratorを使い分ける
  - e.g. OpenAPI, CoreAPI
- OpenAPI 用には `openapi.SchemaGenerator` がある
  - `get_schema_view` ではデフォルトで設定されている

# SchemaGenerator.get\_schema

- path, method, viewごとの要素を生成して返す
  - **paths**
  - info
  - operation
  - **components**

サンプルだと  
convert/, GET, ConverterView  
convert/, POST, ConverterView  
.....

```
def get_schema(self, request=None, public=False):  
    # 中略  
    _, view_endpoints = self._get_paths_and_endpoints(None if public else  
request)  
    for path, method, view in view_endpoints:  
        if not self.has_view_permissions(path, method, view):  
            continue  
        operation = view.schema.get_operation(path, method)  
        components = view.schema.get_components(path, method)  
  
# rest_framework/schemas/openapi.py
```

# SchemaGenerator.get\_schema での要素の設定方法

- SchemaGenerator.get\_schema で要素はどのように生成されるか？

```
operation = view.schema.get_operation(path, method)
components = view.schema.get_components(path, method)

# rest_framework/schemas/openapi.py
```

- view.schema ?

# view.schema (ViewInspector)

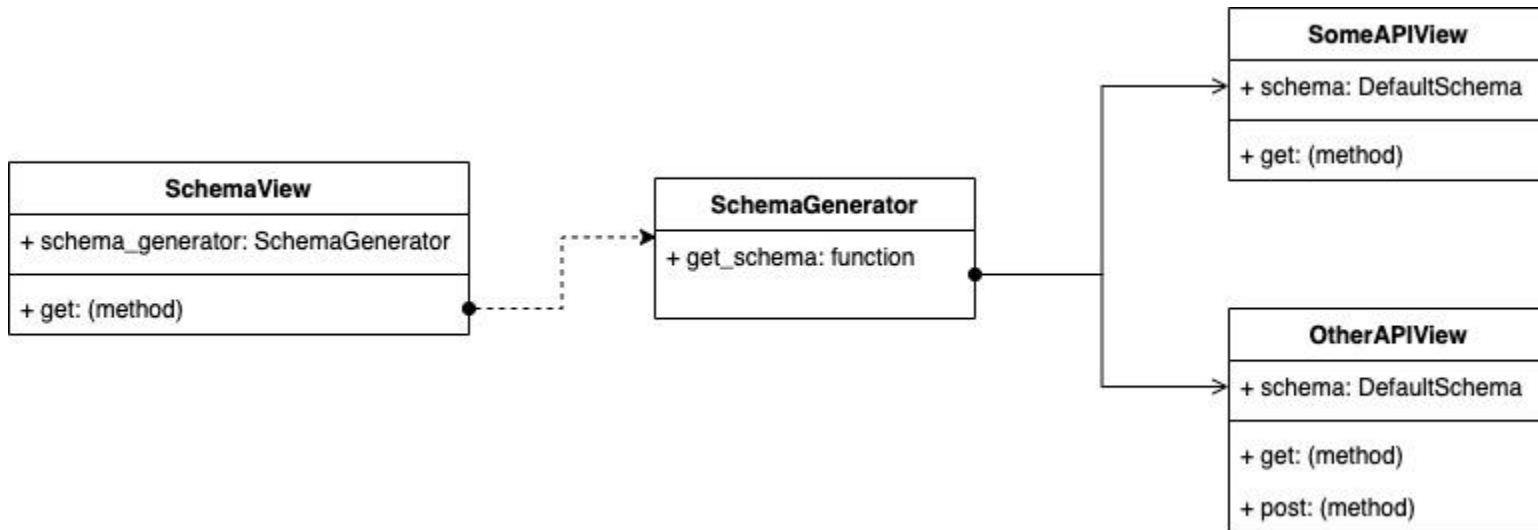
- ViewInspectorを継承する descriptor (記述子)クラス
  - クラスの構造を記述する
- Viewクラスのクラス変数として設定
- DRF の APIView には DefaultSchema が設定されている
  - OpenAPI 用に openapi.AutoSchema がある



# 内部の動き、ここまでのまとめ

SchemaViewは Viewクラスのdescriptorであるview.schemaで要素を生成する

→ **view.schema**をカスタマイズすれば良い



# 独自にカスタマイズする AutoSchemaを使う

# AutoSchema を設定する

- openAPI 用の schema クラス openapi.AutoSchema を使う
- view.py のViewクラスは schema=AutoSchema() に

```
from rest_framework.schemas.openapi import AutoSchema

class ConverterView(RetrieveAPIView):
    serializer_class = ConverterRequestSerializer
    # 追加
    schema = AutoSchema()

    def get(self, request, *args, **kwargs):
# drf_doc_demo/api/views.py
```

※ settings で設定することも可能: settings.DEFAULT\_SCHEMA\_CLASS

# AutoSchema の仕組み: 要素の取得方法

- AutoSchema.get\_operation で parameter, response, requestBodyなどを設定

```
class AutoSchema(ViewInspector):
    def get_operation(self, path, method):
        operation = {}
        # 中略
        request_body = self.get_request_body(path, method)
        if request_body:
            operation['requestBody'] = request_body
        operation['responses'] = self.get_responses(path, method)
        operation['tags'] = self.get_tags(path, method)

        return operation

# rest_framework/schemas/openapi.py
```

独自にカスタマイズする  
AutoSchemaを使う  
for response

# >> いろいろおかしい << の Responses を解決する

### Responses

Code	Description	Links
200	<p>Media type</p> <div>application/json</div> <p>Controls Accept header.</p> <p>Example Value   Schema</p> <pre>{   "text": "string",   "delimiter": "string",   "numeral": true }</pre>	No links

POST /api/converter/

アルファベットをカタカナに変換するPOST method

### Parameters

No parameters

Try it out

Request body

application/json

Example Value | Schema

```
{  
  "text": "string",  
  "delimiter": "string",  
  "numeral": true  
}
```

# AutoSchema: Responsesの取得方法

get\_responses

元をたどると self.get\_serializer ⇨ **view.get\_serializer** を実行している

```
class AutoSchema(ViewInspector):
    def get_responses(self, path, method):
        serializer = self.get_response_serializer(path, method)
        if not isinstance(serializer, serializers.Serializer):
            item_schema = {}
        else:
            item_schema = self.get_reference(serializer)
        # 以降省略

    def get_response_serializer(self, path, method):
        return self.get_serializer(path, method)
```

# rest\_framework/schemas/openapi.py

# view.get\_serializer

view.get\_serializer → GenericAPIView で定義されている

- viewのクラス変数 serializer\_class を使っている……
- あっ

```
class GenericAPIView(views.APIView):
    def get_serializer(self, *args, **kwargs):
        # 中略
        serializer_class = self.get_serializer_class()
        kwargs.setdefault('context', self.get_serializer_context())
        return serializer_class(*args, **kwargs)

    def get_serializer_class(self):
        # 中略
        return self.serializer_class

# rest_framework/generics.py
```



# view.get\_serializer

```
from rest_framework.generics import RetrieveAPIView

class ConverterView(RetrieveAPIView):
    serializer_class = ConverterRequestSerializer

    def get(self, request, *args, **kwargs):
        """アルファベットをカタカナに変換するGET method"""
        data = request.GET
        return self.convert(data)

# drf_doc_demo/api/views.py
```

>>> リクエストにつかうserializer渡してたわ <<<

~~データベースを使う方法だったらしいミス .....~~

# Responsesのため正しい設定をする

response用のserializerを指定

```
class ConverterView(RetrieveAPIView):  
-     serializer_class = ConverterRequestSerializer  
+     serializer_class = ConverterResponseSerializer
```

Responses

Code	Description	Links
200	<p>Media type <input type="text" value="application/json"/></p> <p>Controls Accept header.</p> <p>Example Value   Schema</p> <pre>{   "text": "string" }</pre>	<p>No links</p>



# POSTの表示

Parameters

No parameters

Request body


application/json

Example Value | Schema

```
{  
  "text": "string"  
}
```

Responses

Code	Description
201	<div>Media type</div> <div>application/json</div> <div>Controls Accept header.</div> <div>Example Value   Schema</div> <div><pre>{   "text": "string" }</pre></div>



# POSTの表示がおかしい

requestBody の取得方法は response と同じ、get\_serializer

→ 同じ結果が入ってしまう

```
class AutoSchema(ViewInspector):  
  
    def get_request_body(self, path, method):  
        serializer = self.get_request_serializer(path, method)  
  
    def get_request_serializer(self, path, method):  
        return self.get_serializer(path, method)  
  
# rest_framework/schemas/openapi.py
```

# AutoSchemaの限界

以下の情報を表示するため、さらにカスタマイズしていく

- requestBody
- parameters

# 独自にカスタマイズする AutoSchemaをカスタマイズ for requestBody

# CustomSchema

- AutoSchemaを継承したカスタムクラスを作成
  - drf\_doc\_demo/api/custom\_schema.py
- requestBodyをresponseと別の方法で設定
  - 取得方法を変更: get\_serializer -> get\_request\_serializer

```
from rest_framework.schemas.openapi import AutoSchema

class CustomSchema(AutoSchema):

    def get_request_serializer(self, path, method):
        view = self.view

        try:
            return view.get_request_serializer()
        except exception.APIException:
            return super().get_request_serializer(path, method)

# drf_doc_demo/api/custom_schema.py
```

# CustomSchema -> View

viewクラスにget\_request\_serializerを追加

```
class ConverterView(RetrieveAPIView):  
  
    def get_request_serializer(self):  
        return ConverterRequestSerializer()
```



POST /api/converter/

Parameters

No parameters

Request body

Example Value | Schema

```
{  
  "text": "string",  
  "delimiter": "string",  
  "numeral": true  
}
```

Try it out

on/json

56



独自にカスタマイズする  
FilterBackendをカスタマイズ  
for query parameters

## parameterの生成方法: 3通り

- `get_path_parameters`: endpointのpathから設定
  - for path parameter
- `get_pagination_parameters`: `view.pagination_class` を使う
- **`get_filter_parameters`**: `view.filter_backends`を使う
  - for query parameter
  - 今回はこれを無理やり使う

```
class AutoSchema(ViewInspector):
    def get_operation(self, path, method):
        # 中略
        parameters = []
        parameters += self.get_path_parameters(path, method)
        parameters += self.get_pagination_parameters(path, method)
        parameters += self.get_filter_parameters(path, method)
        operation['parameters'] = parameters

# rest_framework/schemas/openapi.py
```

# view.get\_filter\_parameters

view.filter\_backendsとは？

- rest\_framework/filters.py の BaseFilterBackend を継承したクラス
  - 本来queryset をフィルタリングするのに利用
- filter\_backend.get\_schema\_operation\_parameters でparametersを生成
  - BaseFilterBackendでは実装されていない

```
class AutoSchema(ViewInspector):
    def get_filter_parameters(self, path, method):
        parameters = []
        for filter_backend in self.view.filter_backends:
            parameters +=
filter_backend().get_schema_operation_parameters(self.view)
        return parameters
```

# rest\_framework/schemas/openapi.py

# CustomFilterBackend: FilterBackendをカスタマイズ

- get\_schema\_operation\_parametersでparametersのlistをつくるクラスを作成
  - serializerをparameterに変換する

```
from rest_framework.filters import BaseFilterBackend

class CustomFilterBackend(BaseFilterBackend):
    def get_schema_operation_parameters(self, view):
        parameter_schema = getattr(view, "parameter_serializer")
        parameter = []

# drf_doc_demo/api/custom_schema.py
```

# CustomFilterBackend: FilterBackendをカスタマイズ

- parametersはresponse, requestBody と異なる構造
  - serializer の Field をparameter objectに変換
  - required は Fieldの要素から取得
  - schema typeは ……

```
class ConverterRequestSerializer(serializers.Serializer):  
    text = serializers.CharField()  
    delimiter = serializers.CharField(required=False)
```

```
parameters:  
- name: text  
  required: true  
  in: query  
  schema:  
    type: string  
- name: numeral  
  required: false  
  in: query  
  schema:  
    type: boolean
```

# CustomFilterBackend: Fieldをschema typeに変換

- Fieldクラスごと条件分岐 (ちょっと無理やり)

```
class CustomFilterBackend(BaseFilterBackend):
    @staticmethod
    def get_oai_type(value: Field):
        value_field = type(value)
        type_ = "string"
        if value_field == IntegerField:
            type_ = "integer"
        elif value_field == BooleanField:
            type_ = "boolean"
        elif value_field == FloatField:
            type_ = "number"
        return type_
```

# CustomFilterBackend

Viewクラスに設定

- parameter\_serializer
- filter\_backends
  - リストなので注意

```
class ConverterView(RetrieveAPIView):  
    schema = CustomSchema()  
    parameter_serializer = ConverterRequestSerializer()  
    filter_backends = [CustomFilterBackend]  
    serializer_class = ConverterResponseSerializer
```

# parametersが表示されるように

api

GET /api/converter/

アルファベットをカタカナに変換するGET method

Parameters

Try it out

Name	Description
<b>text</b> ★ required string (query)	<input type="text" value="text"/>
<b>delimiter</b> string (query)	区切り文字 <input type="text" value="delimiter"/>
<b>numeral</b> boolean (query)	数字も変換するフラグ <input type="text" value="--"/>





# まとめ

- response -> View.serializer で指定
- requestBody -> AutoSchema をカスタマイズ
- parameter -> BaseFilterBackend をカスタマイズ



3rd party library、drf-spectacular を使う

## 3rd party library

<https://github.com/tfranzel/drf-spectacular>

> Sane and flexible OpenAPI 3.0 schema generation for Django REST framework.

- OAI3 に対応したツール
- デコレーター `@extend_schema` で仕様を設定できる

## 設定・url

settings

```
# drf-spectacular settings
REST_FRAMEWORK = {
    "DEFAULT_SCHEMA_CLASS": "drf_spectacular.openapi.AutoSchema",
}

SPECTACULAR_SETTINGS = {
    "TITLE": "drf-doc-demo Project",
    "DESCRIPTION": "API for drf doc",
    "VERSION": "1.0.0",
    "SERVE_INCLUDE_SCHEMA": False,
}

# drf_doc_demo/settings/__init__.py
```

# 設定・url

url.py

```
from drf_spectacular.views import SpectacularAPIView, SpectacularSwaggerView

urlpatterns = [
    path("api/schema/", SpectacularAPIView.as_view(), name="schema"),
    path(
        "api/schema/swagger-ui/",
        SpectacularSwaggerView.as_view(url_name="schema"),
        name="swagger-ui",
    ),
]
```

# @extend\_schema

viewクラスのデコレーター

- request
- response
- parameters

などを引数で渡す。引数の指定方法は色々

- DRF の serializer
- drf-spectacular の OpenApiParameter

## @extend\_schema: serializerを使った場合

methodごとdecoratorを追記し、parameters・responsesを渡す

```
class ConverterView(RetrieveAPIView):
    @extend_schema(
        parameters=[ConverterRequestSerializer],
        responses={200: ConverterResponseSerializer},
    )
    def get(self, request):
        data = request.GET

    @extend_schema(
        request=ConverterRequestSerializer,
        responses={200: ConverterResponseSerializer},
    )
    def post(self, request):
        """アルファベットをカタカナに変換するPOST method"""
        data = request.data
```

結果 <http://127.0.0.1:8000/api/schema/swagger-ui/>

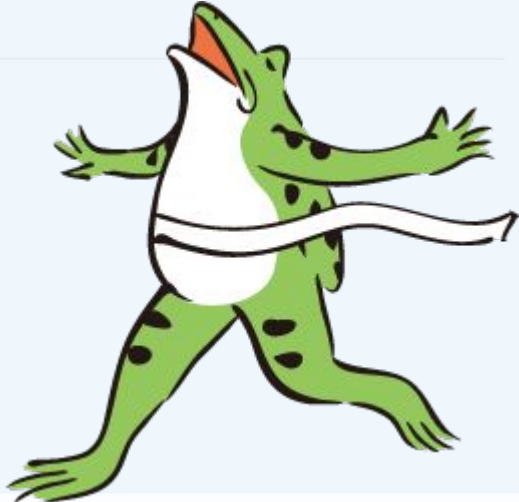
## converter

**GET** /api/converter/

Parameters

Try it out

Name	Description
delimiter string (query)	区切り文字
<input type="text" value="delimiter"/>	
numeral boolean (query)	数字も変換するフラグ
<input type="text" value=""/>	
text * required string (query)	
<input type="text" value="text"/>	



スライド40枚以上かけて説明した実装がわずかな修正で解決



## TL; DR

ちゃちゃっとドキュメント化したい人は **drf-spectacular** を使いましょう

<https://github.com/tfranzel/drf-spectacular>

+ 自力でカスタマイズしたい場合は内部実装をみましょう

ご清聴ありがとうございました

## 参考: drf-spectacularはsecurity schema も設定されている

> In drf-spectacular there is support for auto-generating the security definitions for a number of authentication classes built in to DRF as well as other popular third-party packages.

[https://drf-spectacular.readthedocs.io/en/latest/drf\\_yasg.html#authentication](https://drf-spectacular.readthedocs.io/en/latest/drf_yasg.html#authentication)

**drf-doc-demo Project** 1.0.0 OAS3

[/api/schema/](#)

API for drf doc



Authorize



## 参考資料

- <https://www.django-rest-framework.org/>
- <https://github.com/OAI/OpenAPI-Specification>
- <https://github.com/swagger-api/swagger-ui>
- <https://swagger.io/>
- <https://github.com/tfranzel/drf-spectacular>
- <https://eieito.hatenablog.com/entry/2021/08/24/090000>

イラスト: ダ鳥獣ギ画 (<https://chojugiga.com/>)

フォント: M PLUS 1p (<https://fonts.google.com/specimen/M+PLUS+1p>)