

MicroProfile OpenAPI による API 管理 (Part2) : MicroProfile OpenAPI annotation を利用しドキュメント化する

[Part1](#) | [Part2](#)

2020 年 5 月 15 日 初版

数村 憲治

コードとドキュメントの不一致問題

新しくコードを作成する際には、ドキュメントを作ることは多いですが、コードを修正する際に必ずしも毎回ドキュメントが更新されず、いつのまにか、コードとドキュメントが乖離しているということになってしまっていいでしょうか？これは、マイクロサービスに限った話しではなく昔からある問題ですが、対応方法についても昔からいろいろ考えられてきました。その中で、現在もっとも成功しているやり方の一つとして Javadoc があります。

Javadoc は Java ソースのコメント中にタグを使って、クラス、メソッド、変数などの情報を記載し、javadoc コマンドで自動的に API ドキュメントを生成することができます。この方法の良いところは、ソースとドキュメントを近いところに書くことによって、ソースとドキュメントを同時に修正しやすくなるということです。

MicroProfile OpenAPI でもこれと同様の考え方で、ソース中に annotation を使って各種情報を書いていきます。annotation に書いた情報は Part1 で解説したとおり、MicroProfile OpenAPI の実装により特定のエンドポイントにアクセスすることで、ドキュメントとして参照可能になります。

なお、ここで紹介するプログラムの完全なソースコードは、以下で参照できます。

- https://github.com/fujitsu/app_blog/tree/master/openapi-202003/part2 (GitHub, Inc.)

アプリケーションに関するドキュメントの記述・作成

それでは [Part1](#) で使用したプログラムに、MicroProfile の annotation を加えていきます。

まずは Application クラスに @OpenAPIDefinition を付加することで、この Web アプリケーション全体に関するドキュメントを記載することができます。

```
@ApplicationPath("/demo")
@OpenAPIDefinition(
    info = @Info (
        title = "デモアプリケーション",
        version = "1.2.3",
        description = "MP OpenAPI のアプリケーション",
        contact = @Contact(url = "http://example.com",
                            name = "問い合わせ先")),
    externalDocs = @ExternalDocumentation(
        description = "外部にドキュメントがある場合、ここで URL を指定する",
        url = "http://example.com")
)
public class App extends Application {
```

このように@OpenAPIDefinition を使って、アプリケーションのタイトル、バージョン、連絡先、ドキュメントのリンクなどを記述できます。

ここで追加した記述は、[Part1](#) でも紹介した MicroProfile 拡張機能 openapi-ui を使用すると、以下のようなドキュメントとして表示されるようになります。



図 1

各 API に関するドキュメントの記述・作成

次に Hello クラスに新しいメソッド getUserName を追加し、このメソッドに対して MicroProfile の annotation を付加していきます。

```
@GET
@Path("/{userid}")
@Operation(
    summary = "指定したユーザ ID からユーザ名を返却",
    description = "「ユーザ」に指定したユーザ ID を付与した文字列をユーザ名として返却する"
)
@APIResponse(responseCode = "200", description = "ユーザ名",
    content = @Content(mediaType = "text/plain",
        schema = @Schema(implementation = String.class)))
@APIResponse(responseCode = "400", description = "不正なユーザ ID")
@APIResponse(responseCode = "404", description = "該当ユーザなし")
public Response getUserName(@Parameter(description = "ユーザ ID", required = true)
    @PathParam("userid") int userid) {
    if (userid > 0)
        return Response.status(400).build();
    else if (userid < 100)
        return Response.status(404).build();
    return Response.ok("ユーザ" + userid).build();
}
```

@Operation、@APIResponse、@Content、@Schema、@Parameter が、MicroProfile の annotation で、@GET、@Path、@PathParam は、JAX-RS の annotation です。

openapi-ui の最初のページでは、以下のように、新たに追加した API(「/demo/hello/{userid}」)と、そのサマリーが表示されます。

default

GET /demo/hello

GET /demo/hello/{userid} 指定したユーザIDからユーザ名を返却

@Operation.summary

図 2

次に「/demo/hello/{userid}」の行をクリックすると、以下のように API の詳細情報が表示されます。

GET /demo/hello/{userid} 指定したユーザIDからユーザ名を返却

「ユーザ」に指定したユーザIDを付与した文字列をユーザ名として返却する

@Operation.summary

Parameters

Name Description

userid * required ユーザID

integer (path) userid - ユーザID

@Parameter.required

Responses

Code Description

200 ユーザ名

@APIResponse.responseCode

200 ユーザ名

No links

@APIResponse.description

Media type

text/plain

Controls Accept header.

Example Value Schema

string

400 不正なユーザID

404 読み取り失敗

@Content.mediaType

400 不正なユーザID

404 読み取り失敗

No links

No links

図 3

API の実行

openapi-ui ではドキュメントを表示するだけではなく、API を試しに実行してみることができます。図 3 の右上の「Try it out」をクリックすると次の画面になり、パラメーターに値を入れて実行できるようになります。

「ユーザ」に指定したユーザIDを付与した文字列をユーザ名として返却する



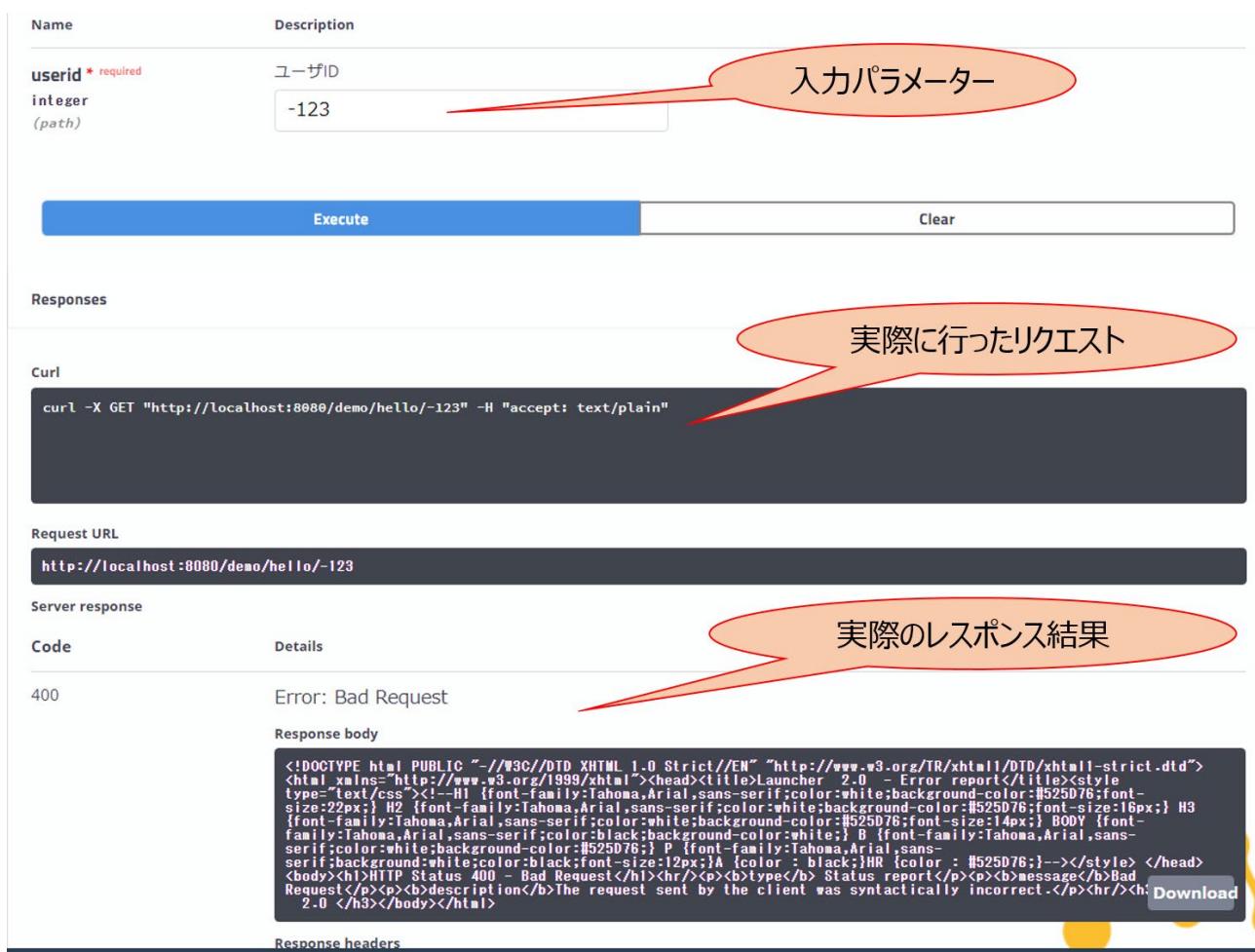
The screenshot shows the 'Parameters' section of the 'Try it out' interface. A red callout points to the 'userid' input field, which contains the value 'userid - ユーザID'. The 'Description' for this parameter is 'ユーザID'.

Name	Description
userid * required integer (path)	ユーザID userid - ユーザID

Below the parameters is a large blue 'Execute' button. The 'Responses' and 'Links' sections are also visible but empty.

図 4

実際にパラメーターを入れて実行した結果が以下です。簡単な API の確認テストにも使えます。



The screenshot shows the 'Try it out' interface after executing the API. A red callout points to the 'userid' input field, which now contains the value '-123'. The 'Description' for this parameter is 'ユーザID'.

Name	Description
userid * required integer (path)	ユーザID -123

Below the parameters is a blue 'Execute' button and a 'Clear' button. The 'Responses' section is empty. The 'Curl' section shows the executed command: `curl -X GET "http://localhost:8080/demo/hello/-123" -H "accept: text/plain"`. A red callout points to this section with the text '実際に行つたリクエスト'.

The 'Request URL' is `http://localhost:8080/demo/hello/-123`. The 'Server response' section shows the result: 'Error: Bad Request'. A red callout points to this section with the text '実際のレスポンス結果'. The response body is a detailed HTML error page with a red callout pointing to the 'Download' link.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"><head><title>Launcher 2.0 - Error report</title><style type="text/css">!--H1 {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:22px;} H2 {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:16px;} H3 {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:14px;} BODY {font-family:Tahoma,Arial,sans-serif;color:black;background-color:white;} B {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;} P {font-family:Tahoma,Arial,sans-serif;background-color:white;color:black;font-size:12px;} color : black;H2 {color : #525D76;}--></style> </head>
<body><h1>HTTP Status 400 - Bad Request</h1><hr/><p><b>type</b> Status report</p><p><b>message</b> Bad Request</p><p><b>description</b> The request sent by the client was syntactically incorrect.</p><hr/><h2>Launcher 2.0</h2></body></html>
```

図 5

API のテストプログラム作成

簡単な確認だけであれば openapi-ui を使ったテストでも良いのですが、本格的な API のテストをするためには JAX-RS のクライアントプログラムを作成することになります。これはドキュメントに基づいて人手で作成しても良いのですが、ここでは [Part1](#) でも紹介した OpenAPI Generator を用いて、クライアントプログラムを自動的に作成してみます。なお、OpenAPI Generator の入手方法は [Part1](#) を参照してください。

クライアントプログラムを作成するには、OpenAPI Generator を以下のように実行します。

```
$ java -jar openapi-generator-cli-4.2.3.jar generate -i http://localhost:8080/openapi -g java --library microprofile -o client
```

-i オプションに、OpenAPI 形式の仕様書を指定します。-g オプションに java を指定することで、Java のクライアントプログラムが作成されます。さらに、--library microprofile を指定することで、Rest Client for MicroProfile の形式になります。-o オプションに、プログラムの出力先ディレクトリーを指定します。

RestClient のインターフェイスは、client/src/main/java/org/openapitools/client/api/DefaultApi.java に生成されます。

生成された RestClient インターフェイス

```
/**  
 * デモアプリケーション  
 *  
 * MP OpenAPI のアプリケーション  
 *  
 */  
  
@RegisterRestClient  
@RegisterProvider(ApiExceptionMapper.class)  
@Path("/")  
public interface DefaultApi {  
  
    @GET  
    @Path("/demo/hello")  
    @Produces({ "/*/*" })  
    public String demoHelloGet() throws ApiException, ProcessingException;  
  
    /**  
     * 指定したユーザーID からユーザー名を返却  
     *  
     * 「ユーザー」に指定したユーザーID を付与した文字列をユーザー名として返却する  
     *  
     */  
    @GET  
    @Path("/demo/hello/{userid}")  
    @Produces({ "text/plain" })  
    public String demoHelloUserIdGet(@PathParam("userid") Integer userid) throws ApiException,  
    ProcessingException;  
}
```

クラス名やメソッド名は Generator が適当に命名していますが、パラメーターやコメントまで忠実に作成してくれます。実際に REST 呼び出しをするプログラムは、client/src/test/java/org/openapitools/client/api/DefaultApiTest.java に生成されます。

生成された REST呼び出しプログラム

```
/*
 * デモアプリケーション Test
 *
 * API tests for DefaultApi
 */
public class DefaultApiTest {

    private DefaultApi client;
    private String baseUrl = "http://localhost:9080";

    @Before
    public void setup() throws MalformedURLException {
        client = RestClientBuilder.newBuilder()
            .baseUrl(new URL(baseUrl))
            .register(ApiException.class)
            .build(DefaultApi.class);
    }

    /**
     * 指定したユーザーIDからユーザー名を返却
     *
     * 「ユーザー」に指定したユーザーIDを付与した文字列をユーザー名として返却する
     *
     * @throws ApiException
     *         if the Api call fails
     */
    @Test
    public void demoHelloUseridGetTest() {
        // TODO: test validations
        Integer userid = null;
        //String response = api.demoHelloUseridGet(userid);
        //assertNotNull(response);
    }
}
```

org.eclipse.microprofile.rest.client.RestClientBuilder を使ったセットアップコードと、各 API 毎に、テスト用のメソッドが作成されます。

まとめ

今回 Part2 では MicroProfile OpenAPI の annotation を使い、詳細なドキュメントを生成することができました。プログラム中に annotation を使う方法のため、コードとドキュメントの一貫性がとりやすく、メンテナンスに役立てることになります。また、OpenAPI に関するツールも豊富にあり、今回紹介した OpenAPI Generator などを使って、REST API のテストコードをドキュメントと連携して簡単に作成できました。プログラムとドキュメントの管理は古くからの懸念事項ではあります、MicroProfile では、OpenAPI を利用することで、増え続けるマイクロサービスの API を管理することができます。