記事
Mihoko Iijima · 2023年2月19日 14m read

OpenAPI-Suite（OpenAPI 3.0からObjectScriptコードを生成するためのツールセット）：InterSystemsデベロッパーツールコンテスト2023入賞作品のご紹介

開発者の皆さん、こんにちは！

InterSystems デベロッパーツールコンテスト2023 の**21**の応募作品の中から、**Experts Nomination**
第3位に輝いた @Lorenzo Scalese
さんの OpenAPI-Suite（OpenAPI3.0
からObjectScriptコードを生成するためのツールセット）についてご紹介します。

現時点でIRISは**OpenAPI2.0**
までの対応なのですが、このツールの凄いところは、**OpenAPI3.0に対応している**ところです！

提供している機能は、以下の通りです。

1. サーバー側クラスの作成（OpenAPI2.0と同様に、APIファーストで作成するRESTディスパッチクラスの作成に必要なクラス群を作成します。）
2. HTTPクライアントクラスの作成
3. クライアントプロダクション（ビジネス・サービス、ビジネス・オペレーション、ビジネス・プロセス、Ens.Request、Ens.Response）
4. このツールから生成する機能を指定できるWebインターフェース
5. OpenAPI 1.x、2.x からバージョン3.0への変換ツール

   各機能について詳しくは、@Lorenzo Scales さんが書かれた記事「OpenAPI Suite - Part 1」「OpenAPI Suite - Part 2」をご参照ください。

この記事では、1の機能を試してみました。その中でも、以下の対応がとても素晴らしい！と思いました。

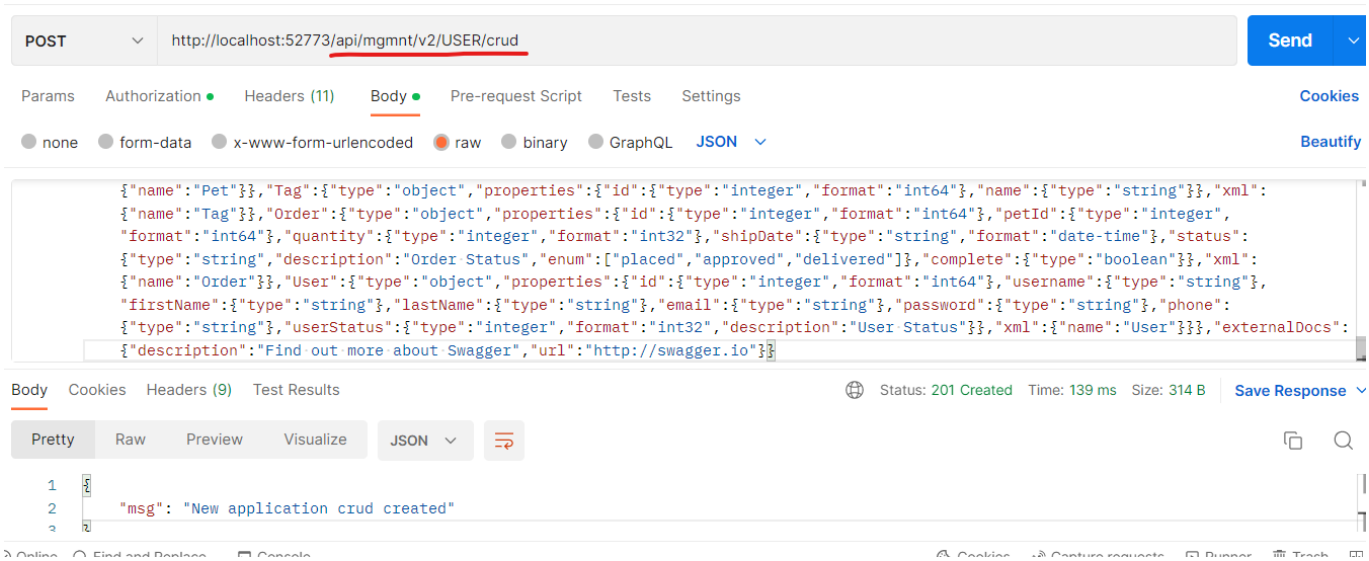
1. OpenAPI 3.0から加わったComponentsオブジェクトの内容に合わせ、クラス定義を自動生成している（指定するパッケージ以下にmodelパッケージができ、そこにインスタンス化できるクラスができています）
2. POSTやPUTの場合、HTTP要求で渡される情報を実装クラス（impl.cls）で処理しやすいように、Bodyで受け取ったJSON文字を1で生成されたクラスのインスタンスに設定するところまで処理してくれています。
   - 現時点では、一部手動で修正が必要ですが、そのまま作成されたインスタンスを利用してimpl.clsで%Save()を書いたら保存されました。）

IRISには、OpenAPI2.0の仕様でRESTディスパッチクラスを作成する方法が提供されていますが、使用するクラス群とベースURLの作成のみを行っています。

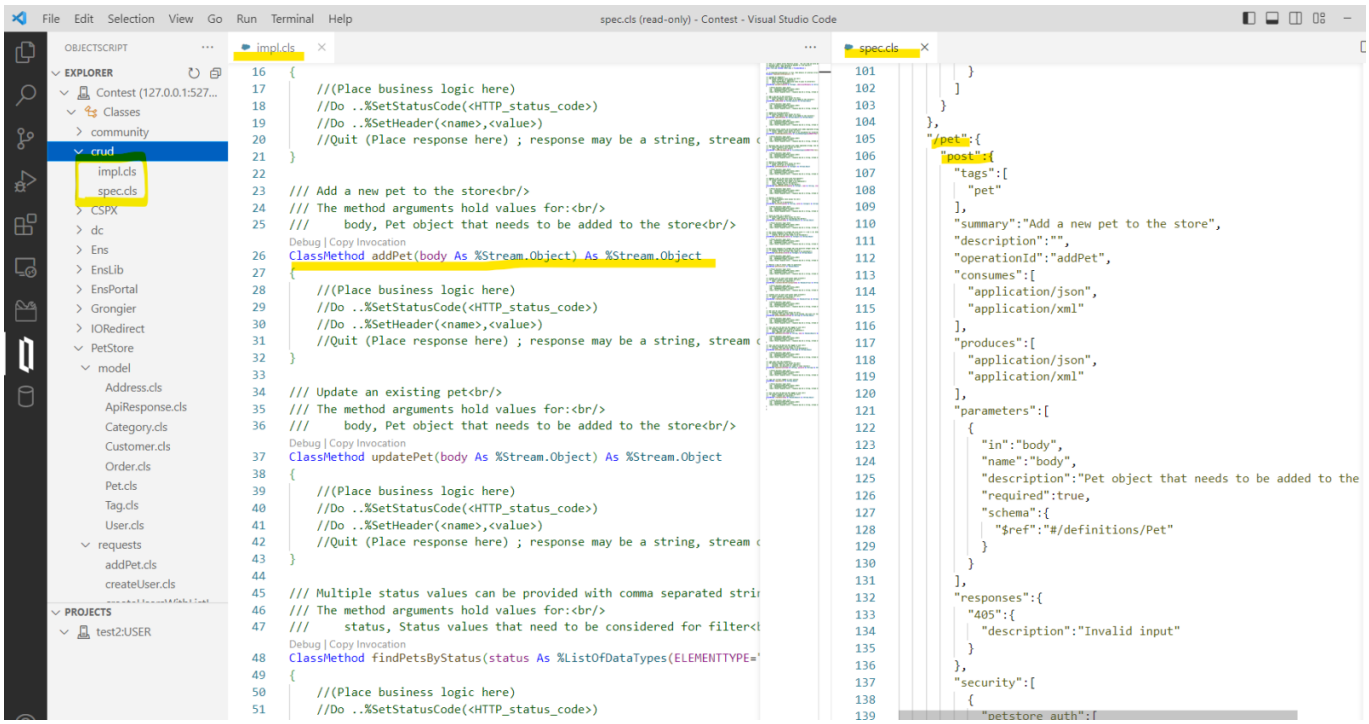
参考までに、IRISが対応しているOpenAPI2.0で作成した内容をご紹介します。

（OpenAPI2.0の仕様で作成されたJSONサンプルを使用しています。）

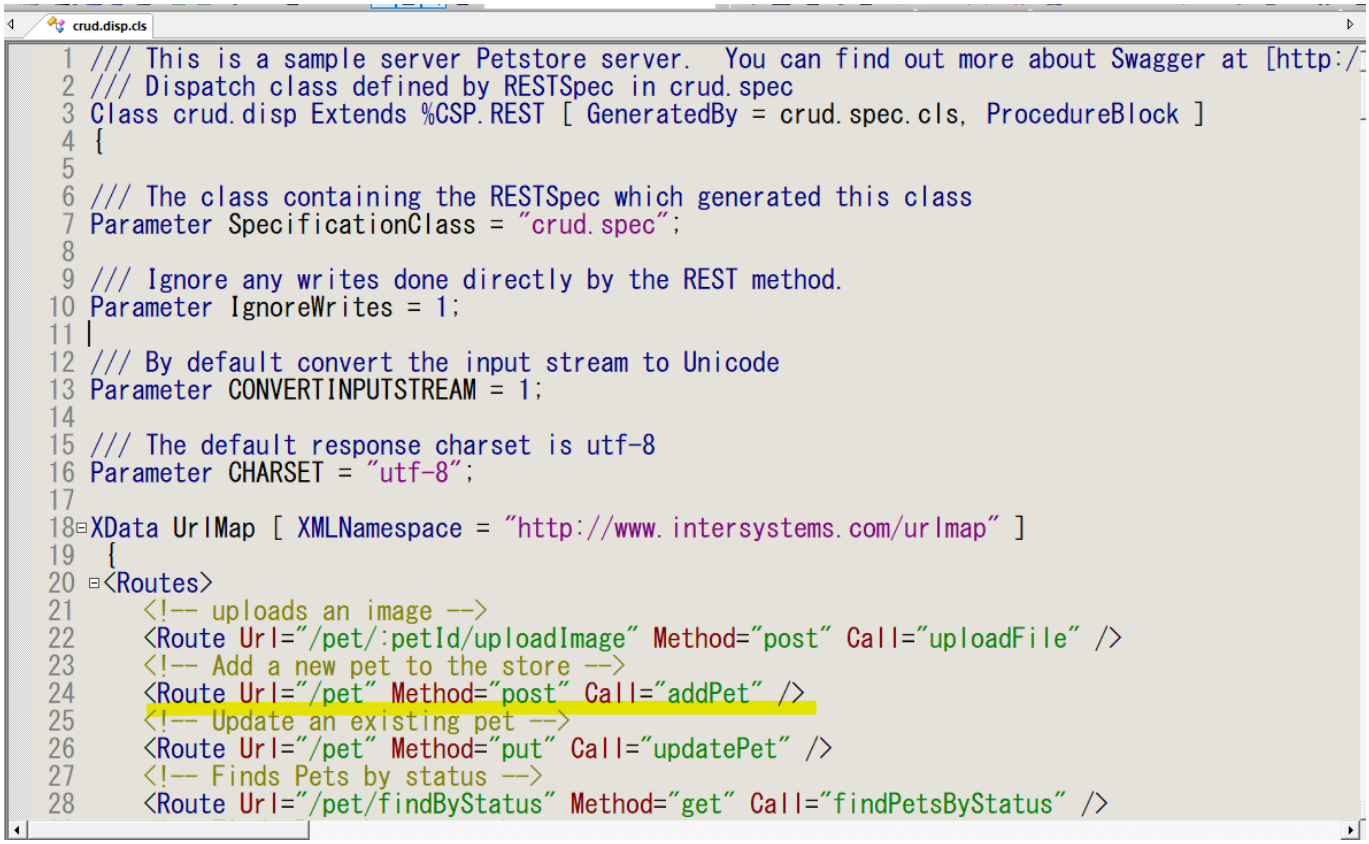
以下の例では、/curdがベースURLとなり、curdパッケージ以下にクラス群が作成されます。



生成されるクラスは以下の通りです（左：impl.cls、右：spec.cls）



ディスパッチクラス（disp.cls）は加工不要クラスのため、例ではスタジオでの表示でご紹介しています。



> 詳細は【はじめてのInterSystems IRIS】セルフラーニングビデオ：アクセス編：（REST）APIファーストで作成するRESTディスパッチクラス のビデオをご参照ください。

次は @Lorenzo Scalese さんの OpenAPI-Suite の結果をご紹介します。

使用するまでの手順はとても簡単です。

IPM（InterSystems Package
Mangaer：以前はZPMとも呼ばれていました）のクライアントツール
を管理ポータルかスタジオからインポートしたら、以下コマンドを実行するだけでツールの準備が整います。

> クライアントツール
> のインポートはどのネームスペースでも大丈夫です。管理ポータルからインポートされる場合は、以下メニューを利用します。
>
> 管理ポータル→システムエクスプローラ→クラス→インポート対象ネームスペースを選択→インポートボタンクリック

RESTディスパッチクラスを置きたいネームスペースに移動し、ZPMコマンドを利用してツールをインストールします。

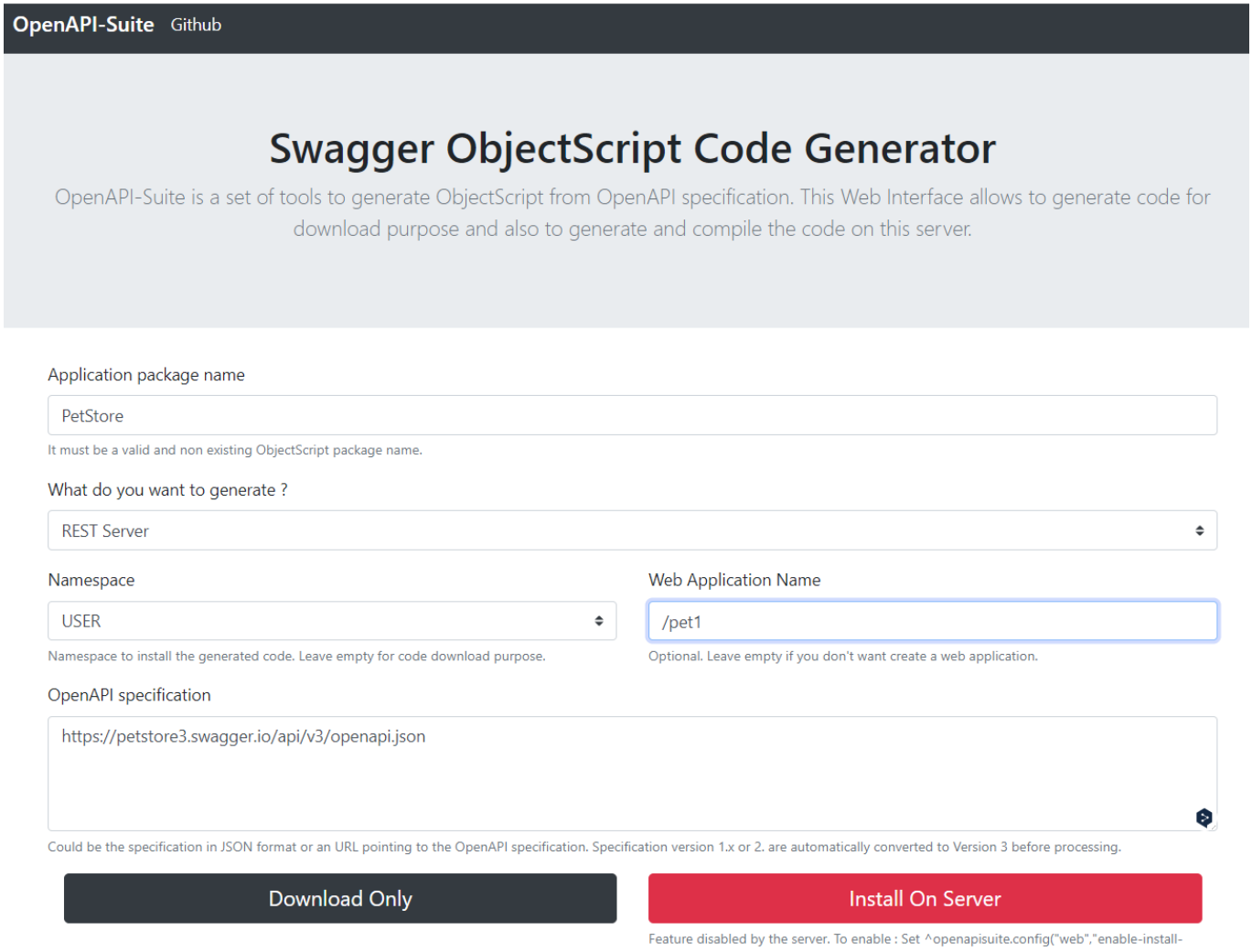
```
zpm "install openapi-suite"
```

インストール時の画面表示

インストール後、以下URLでツールの専用画面にアクセスできます。

http://localhost:ポート番号/openapisuite/ui/index.csp

例）http://localhost:52773/openapisuite/ui/index.csp



初期段階では、**画面右下の「Install On Server」ボタンは押せなくなっています。**
以下グローバル変数に値をセットすることで、ボタンが押せるようです。

（ZPMコマンドでツールをインストールしたネームスペースでセットしてください）

```
Set ^openapisuite.config("web","enable-install-onserver") = 1
```

準備が完了したら、以下設定します。

「Applcation package name」にはクラス定義のパッケージ名を（例では、PetStore）、「What do you want to generate?」では、「REST Server」を選択します。「NameSpace」にはRESTディスパッチクラスを登録したいネームスペースを選択し、「Web Application Name」には、定義したいベースURL（ウェブ・アプリケーションパス）を指定します（例では、/pet1）。

後は、OpenAPIの仕様に基づいて作成さ

れたJSONファイルを指定するだけですが、サンプルでは、https://petstore3.swagger.io/api/v3/openapi.json
がデフォルトで指定されていますのでこのまま使用します。

最後に、「Install On Server」ボタンをクリックします。

以下のようにログが表示されますので、正常に終了したことを確認してCloseボタンをクリックします。

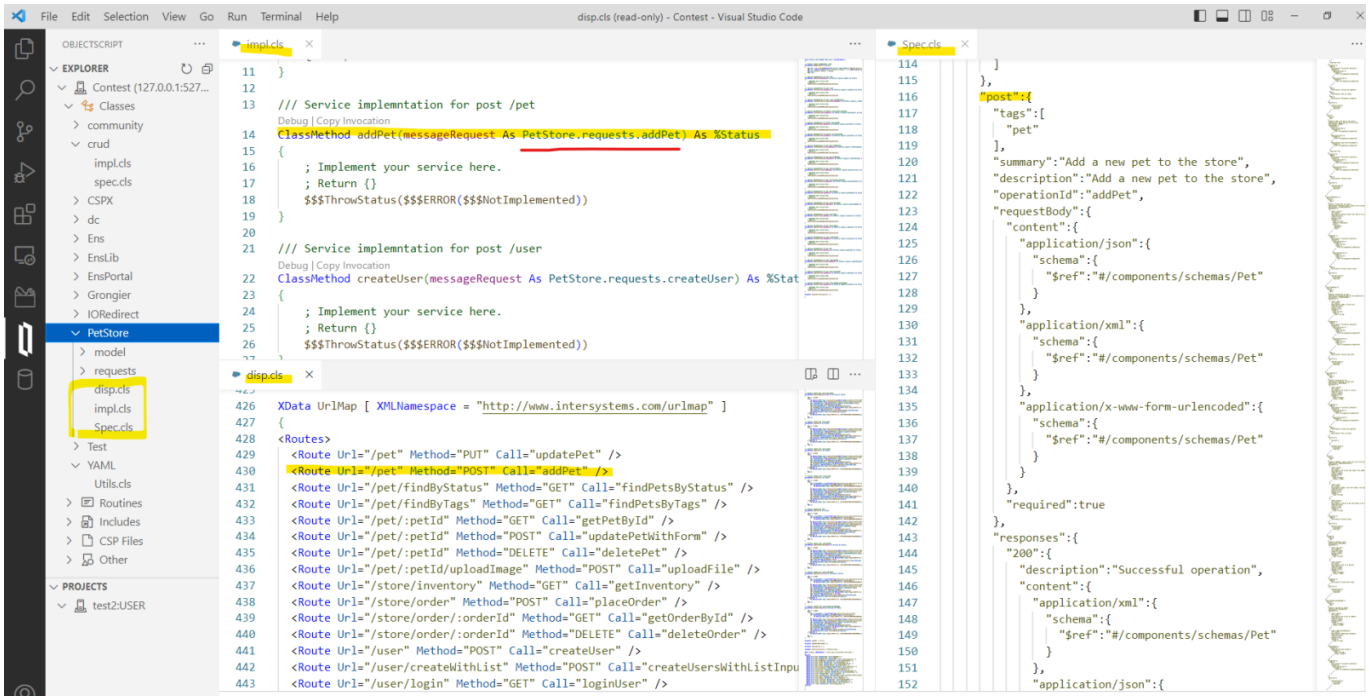
ツールを通して、ウェブアプリケーションパス：/pet1が作成されている予定です。確認してみましょう。

管理ポータル→システム管理→セキュリティ→アプリケーション→ウェブ・アプリケーション から
/pet1があるか確認します。

しっかり作成されていました！

では、入力されたパスに対応するメソッドのマッピングが記載されているdisp（ディスパッチクラス）や、コードを実装するimplクラスができているか確認してみます。

PetStoreパッケージ以下にしっかりクラス定義が作成されていました！



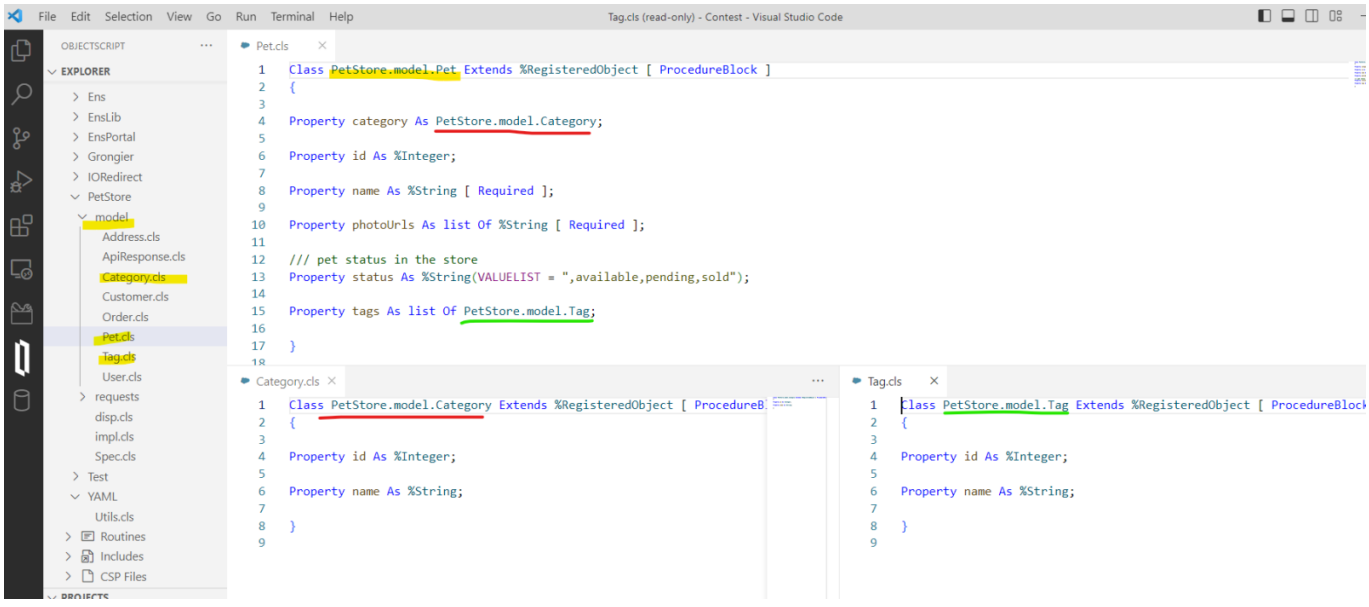
IRISが提供するOpenAPI2.0のクラス群と異なる点は、modelパッケージとrequestsパッケージが含まれている点です。

このツールの中では、OpenAPI3.0のComponentsオブジェクトで指定している情報を、インスタンス生成ができるクラス定義として作成しています。（初期段階では%RegisteredObjectクラスを継承しています）

ComponentsオブジェクトのPetの情報は、以下展開すると参照できます。

Components：PetのSpec

この情報から作成されたクラスは以下の通りです。



インスタンス化ができるクラスで用意されています。

スーパークラスを%RegisteredObjectから%Persistentに変更＋コンパイルを行えば、データベースに登録できる永続クラスとして利用できます。

また、POST要求では、以下の形式でJSONオブジェクトが送付されます。

```
{
  "id": 0,
  "category": {
    "id": 0,
    "name": "Dog"
  },
  "name": "Hachi",
  "photoUrls": [
    "https://x.gd/1pbYK"
  ],
  "tags": [
    {
      "id": 0,
      "name": "Middle"
    }
  ],
  "status": "available"
}
```

このJSONをそのままPet、Tag、Categoryクラスのインスタンスに当てはめることができればいいので、%JSON.Adaptorを追加で継承させるとよさそうです。

ということで、Pet、Tag、Categoryに手を加えるとしたら、スーパークラスの設定を %RegisterdObject から %Persistent と %JSON.Adaptor の多重継承に変えたらよさそうです。

Pet.cls

```
Class PetStore.model.Pet Extends (%Persistent, %JSON.Adaptor)
{

Property category As PetStore.model.Category;

Property id As %Integer;

Property name As %String [ Required ];

Property photoUrls As list Of %String [ Required ];

/// pet status in the store
Property status As %String(VALUELIST = ",available,pending,sol

Property tags As list Of PetStore.model.Tag;

Storage Default
{
<Data name="PetDefaultData">
<Value name="1">
<Value>%%CLASSNAME</Value>
</Value>
<Value name="2">
<Value>category</Value>
</Value>
<Value name="3">
<Value>id</Value>
</Value>
```

Category.cls

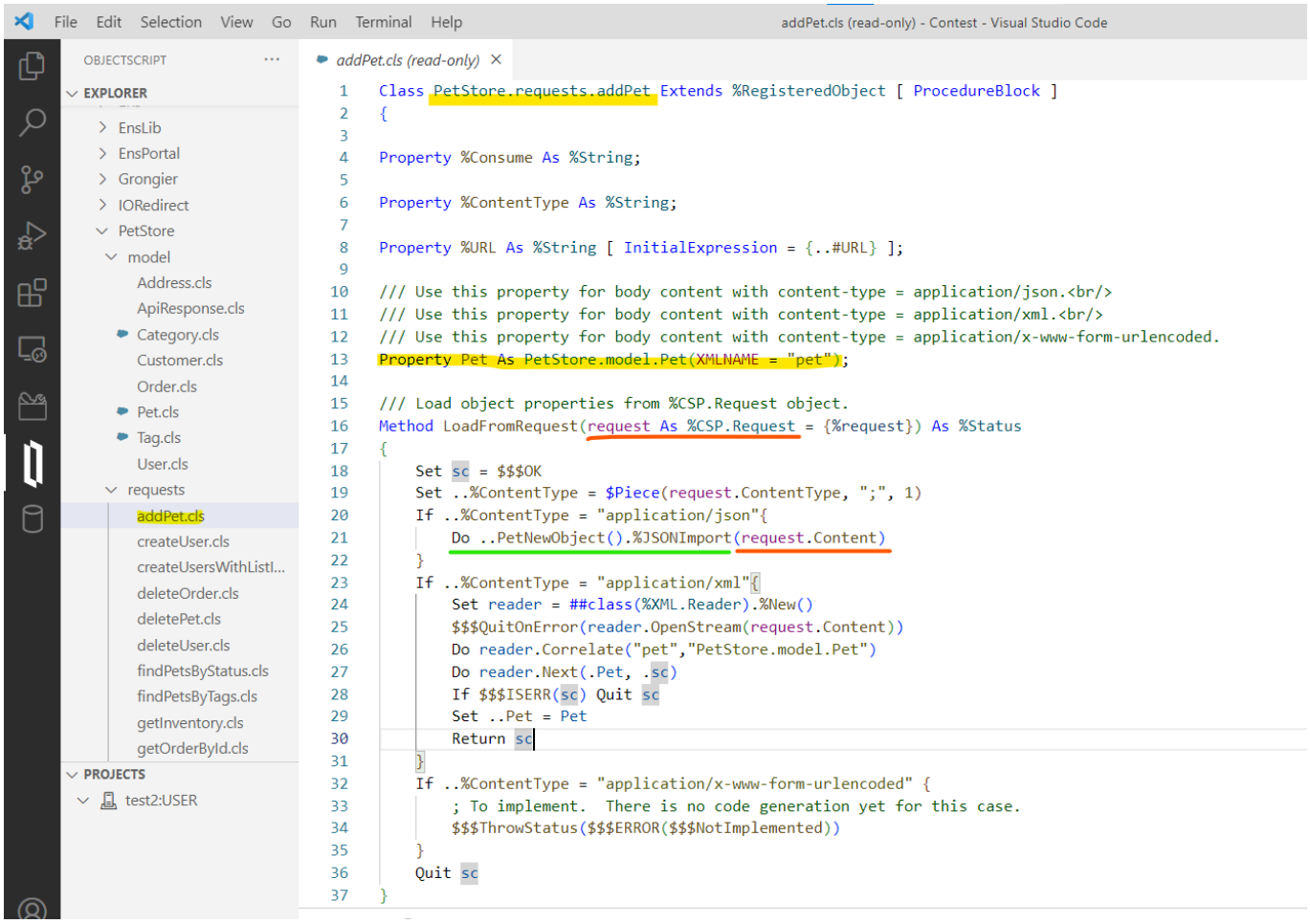
```
Class PetStore.model.Category Extends (%Persistent, %JSON.Adaptor) [
{

Property id As %Integer;

Property name As %String;

Storage Default
{
<Data name="CategoryDefaultData">
<Value name="1">
<Value>%%CLASSNAME</Value>
</Value>
<Value name="2">
<Value>id</Value>
</Value>
```

Tag.cls

```
Class PetStore.model.Tag Extends (%Persistent, %JSON.Adaptor) [ Proc
{

Property id As %Integer;

Property name As %String;

Storage Default
{
```

この後は、受け取ったBodyの情報をデータベースに更新する処理が必要です。

この処理の大枠を記載してくれているのが、requestパッケージ以下クラスです。以下例では、/petに対してPOST要求を行うときに動作するクラス（PetStore.requests.addPet.cls）の中身です。



LoadFromRequest()メソッドに注目すると、引数requestにHTTP要求を操作できる%requestが渡されるようになっています（IRISのRESTディスパッチクラスの内部処理では、HTTP要求は%CSP.Requestクラスのインスタンスに収納されます）。

```
request.Content
```

上記実行でBodyに指定されたJSONオブジェクトを取得できます。

```
Do ..PetNewObject().%JSONImport(request.Content)
```

と記述がありますが、このクラス内にPetNewObject()メソッドがないため、以下のように修正しています（Petのインスタンスを生成し、プロパティPetに設定した後、受信したJSONをインスタンスに割り当てるための処理に変えています）。

```
set ..Pet=##class(PetStore.model.Pet).%New()
do ..Pet.%JSONImport(request.Content)
```

これで、HTTP要求を受信したときの処理は終了です。残りは、impl.clsの実装です。

初期段階では、以下の状態で何も記載されていません。

```
ClassMethod addPet(messageRequest As PetStore.requests.addPet) As %Status
```

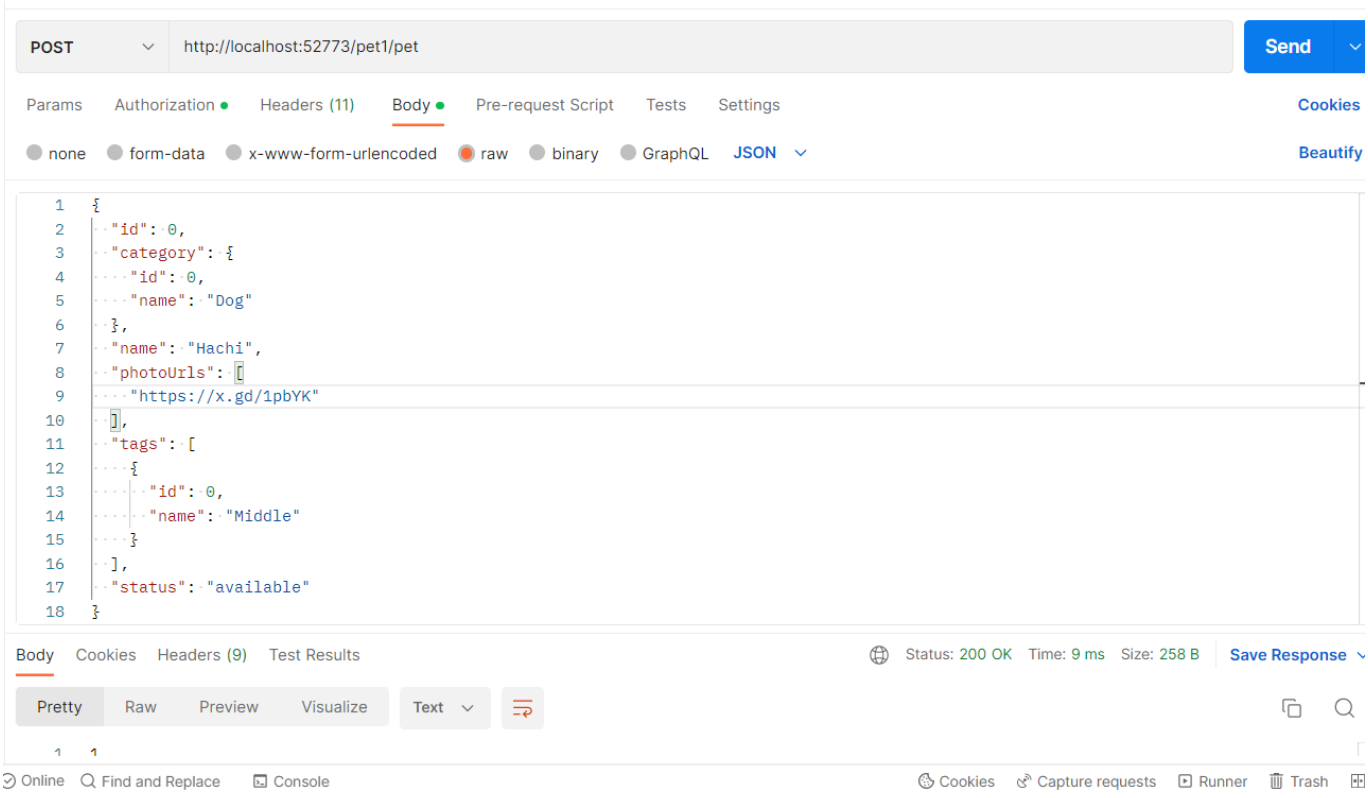
```
{
 ; Implement your service here.
 ; Return {}
 $$$ThrowStatus($$$ERROR($$$NotImplemented))
}
```

引数messageRequestには、1つ前の流れで確認した PetStore.requests.addPet クラスのインスタンスが指定されています（このインスタンスのPetプロパティにHTTP要求で受信したJSONオブジェクトから作成したインスタンスが設定されています。）

以下のように書き換え、コンパイルを行ったら修正は終了です！

```
ClassMethod addPet(messageRequest As PetStore.requests.addPet) As %Status
{
    ; Implement your service here.
    ; Return {}
    //$$$ThrowStatus($$$ERROR($$$NotImplemented))
    set status=$$$OK
    set status=messageRequest.Pet.%Save()
    return status
}
```

POST要求をテストしてみます。



PetもCategoryもTagもしっかりデータ登録できました！

```
select id,name,category->name,photourls,status from PetStore_model.Pet
```

Row count: **1** Performance: **0.002** seconds **325** global references **2308** commands executed **0** disk read later

id	name	name	photoUrls	status
0	Hachi	Dog	https://x.gd/1pbYK	available

1 row(s) affected

```
select * from PetStore_model.Tag
```

Row count: **1** Performance: **0.002** seconds **324** global referen

ID1	id	name
1	0	Middle

少し加工が必要ですが、OpenAPI3.0のComponentsオブジェクトからクラス定義が自動生成されるところが、とても便利だと思いました！

ぜひ皆さんもお試しください。

#REST API #開発環境 #InterSystems IRIS #InterSystems IRIS for Health

ソースURL:https://jp.community.intersystems.com/post/openapi-suite%EF%BC%88openapi-30%E3%81%8B%E3%82%89objectscript%E3%82%B3%E3%83%BC%E3%83%89%E3%82%92%E7%94%9F%E6%88%90%E3%81%99%E3%82%8B%E3%81%9F%E3%82%81%E3%81%AE%E3%83%84%E3%83%BC%E3%83%AB%E3%82%BB%E3%83%83%E3%83%88%EF%BC%89%EF%BC%9Aintersystems%E3%83%87%E3%83%99%E3%83%AD%E3%83%83%E3%83%91%E3%83%BC%E3%83%84%E3%83%BC%E3%83%AB%E3%82%B3%E3%83%B3%E3%83%86%E3%82%B9%E3%83%882023%E5%85%A5%E8%B3%9E%E4%BD%9C%E5%93%81%E3%81%A

E%E3%81%94%E7%B4%B9%E4%BB%8B