Artigo
Lorenzo Scalese · Fev. 1, 2021 8min de leitura

Open Exchange

Criando uma produção de interoperabilidade IRIS a partir do Swagger

Olá comunidade,

O OpenAPI-Client Gen acaba de ser lançado, este é um aplicativo para criar um cliente de produção de interoperabilidade IRIS a partir da especificação Swagger 2.0.

Em vez da ferramenta existente ^%REST que cria um aplicativo REST do lado do servidor, o OpenAPI-Client Gen cria um modelo de cliente de produção de interoperabilidade REST completo.

Instalação por ZPM:

```
zpm "install openapi-client-gen"
```

Como gerar produção a partir de um documento Swagger? É muito simples.

Abra um terminal e execute:

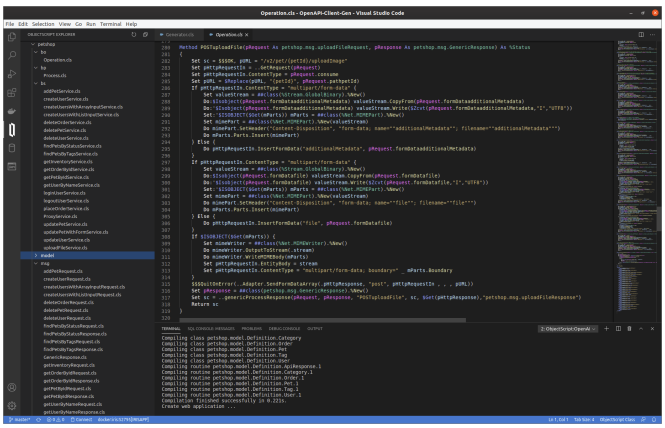
```
Set sc = ##class(dc.openapi.client.Spec).generateApp(<applicationName>, <Your Swagger
 2.0 document>>)
```

O primeiro argumento é o pacote de destino onde as classes de produção serão geradas. Ele deve ser um nome de pacote válido e não existente.
O segundo, é o documento Swagger. Estes valores são aceitos:
1) File path.
2) %DynamicObject.
3) URL.
A especificação deve estar no formato JSON.
Se sua especificação usa o formato YAML, ela pode ser facilmente convertida em JSON com ferramentas on-line como onlineyamltools.com
Exemplo:

```
Set sc = ##class(dc.openapi.client.Spec).generateApp("petshop", "https://petstore.swa
gger.io:443/v2/swagger.json")
Write "Status : ", $SYSTEM.Status.GetOneErrorText(sc)
```

Dê uma olhada no código gerado, podemos ver muitas classes, divididas em muitos subpacotes:
- Business Services: petshop.bs
- Business Operations: petshop.bo
- Business Processes: petshop.bp
- Aplicação REST Proxy: petshop.rest
- Ens.Request e Ens.Response: petshop.msg
- Objeto de entrada ou saída analisado: petshop.model.Definition

- Classe de configuração de produção: petshop.Production



Classe de operação de negócios

Para cada serviço definido no documento Swagger, existe um método relacionado denominado por <VERB><ServiceId>.

Aprofunde-se em um simples método gerado GETgetPetById

```
/// Retorna um único animal de estimação
Method GETgetPetById(pRequest As petshop.msg.getPetByIdRequest, pResponse As petshop.
msg.GenericResponse) As %Status
{
    Set sc = $$$OK, pURL = "/v2/pet/{petId}"
    Set pHttpRequestIn = ..GetRequest(pRequest)
    Set pHttpRequestIn.ContentType = pRequest.consume
    Set pURL = $Replace(pURL, "{petId}", pRequest.pathpetId)
    $$$QuitOnError(..Adapter.SendFormDataArray(.pHttpResponse, "get", pHttpRequestIn
, , , pURL))
    Set pResponse = ##class(petshop.msg.GenericResponse).%New()
    Set sc = ..genericProcessResponse(pRequest, pResponse, "GETgetPetById", sc, $Get(
pHttpResponse),"petshop.msg.getPetByIdResponse")
    Return sc
}
```

* Em primeiro lugar, o objeto %Net.HttpRequest é sempre criado pelo método GetRequest, fique à vontade para editar e adicionar alguns cabeçalhos, se necessário.
* Em segundo lugar, o objeto HttpRequest é preenchido usando pRequest petshop.msg.getPetByIdRequest' (Ens.Request subclass).
* Em terceiro lugar,EnsLib.HTTP.OutboundAdapteré usado para enviar solicitação http.
* E, finalmente, há um processamento de resposta genérico pelo métodogenericProcessResponse:`

```
Method genericProcessResponse(pRequest As Ens.Request, pResponse As petshop.msg.Gener
icResponse, caller As %String, status As %Status, pHttpResponse As %Net.HttpResponse,
 parsedResponseClassName As %String) As %Status
{
    Set sc = $$$OK
    Set pResponse.operation = caller
    Set pResponse.operationStatusText = $SYSTEM.Status.GetOneErrorText(status)
    If $Isobject(pHttpResponse) {
        Set pResponse.httpStatusCode = pHttpResponse.StatusCode
        Do pResponse.body.CopyFrom(pHttpResponse.Data)
```

```
        Set key = ""
        For  {
            Set key = $Order(pHttpResponse.Headers(key),1 , headerValue)
            Quit:key=""
            Do pResponse.headers.SetAt(headerValue, key)
        }
        Set sc = ##class(petshop.Utils).processParsedResponse(pHttpResponse, parsedRe
sponseClassName, caller, pRequest, pResponse)
    }
    Return sc
}
```

Então, podemos analisar um método um pouco mais complexo POSTuploadFile

```
Method POSTuploadFile(pRequest As petshop.msg.uploadFileRequest, pResponse As petshop
.msg.GenericResponse) As %Status
{
    Set sc = $$$OK, pURL = "/v2/pet/{petId}/uploadImage"
    Set pHttpRequestIn = ..GetRequest(pRequest)
    Set pHttpRequestIn.ContentType = pRequest.consume
    Set pURL = $Replace(pURL, "{petId}", pRequest.pathpetId)
    If pHttpRequestIn.ContentType = "multipart/form-data" {
        Set valueStream = ##class(%Stream.GlobalBinary).%New()
        Do:$Isobject(pRequest.formDataadditionalMetadata) valueStream.CopyFrom(pReque
st.formDataadditionalMetadata)
        Do:'$Isobject(pRequest.formDataadditionalMetadata) valueStream.Write($Zcvt(pR
equest.formDataadditionalMetadata,"I","UTF8"))
        Set:'$ISOBJECT($Get(mParts)) mParts = ##class(%Net.MIMEPart).%New()
        Set mimePart = ##class(%Net.MIMEPart).%New(valueStream)
        Do mimePart.SetHeader("Content-Disposition", "form-
data; name=""additionalMetadata""; filename=""additionalMetadata""")
        Do mParts.Parts.Insert(mimePart)
    } Else {
        Do pHttpRequestIn.InsertFormData("additionalMetadata", pRequest.formDataaddit
ionalMetadata)
    }
    If pHttpRequestIn.ContentType = "multipart/form-data" {
        Set valueStream = ##class(%Stream.GlobalBinary).%New()
        Do:$Isobject(pRequest.formDatafile) valueStream.CopyFrom(pRequest.formDatafil
e)
        Do:'$Isobject(pRequest.formDatafile) valueStream.Write($Zcvt(pRequest.formDat
afile,"I","UTF8"))
        Set:'$ISOBJECT($Get(mParts)) mParts = ##class(%Net.MIMEPart).%New()
        Set mimePart = ##class(%Net.MIMEPart).%New(valueStream)
        Do mimePart.SetHeader("Content-Disposition", "form-
data; name=""file""; filename=""file""")
        Do mParts.Parts.Insert(mimePart)
    } Else {
        Do pHttpRequestIn.InsertFormData("file", pRequest.formDatafile)
    }
    If $ISOBJECT($Get(mParts)) {
        Set mimeWriter = ##class(%Net.MIMEWriter).%New()
        Do mimeWriter.OutputToStream(.stream)
        Do mimeWriter.WriteMIMEBody(mParts)
        Set pHttpRequestIn.EntityBody = stream
        Set pHttpRequestIn.ContentType = "multipart/form-
data; boundary=" _ mParts.Boundary
```

```
    }
    $$$QuitOnError(..Adapter.SendFormDataArray(.pHttpResponse, "post", pHttpRequestIn
 , , , pURL))
    Set pResponse = ##class(petshop.msg.GenericResponse).%New()
    Set sc = ..genericProcessResponse(pRequest, pResponse, "POSTuploadFile", sc, $Get
(pHttpResponse),"petshop.msg.uploadFileResponse")
    Return sc
}
```

Como você pode ver, é exatamente a mesma lógica: GetRequest, preenchendo %Net.HttpRequest, enviar solicitação, processamento de resposta genérica.

Classe Proxy REST

Uma aplicação proxy REST também é gerada.
Esta classe REST usa um Projection para criar automaticamente a aplicação web relacionada (ex: "/petshoprest", consulte petshop.rest.REST e petshop.rest.Projection). Este proxy REST cria a mensagem Ens.Request e envia-a para o Business.Process.

```
Class petshop.rest.REST Extends %CSP.REST [ ProcedureBlock ]
{

Projection WebApp As petshop.rest.Projection;

...

ClassMethod POSTaddPet() As %Status
{
    Set ensRequest = ##class(petshop.msg.addPetRequest).%New()
    Set ensRequest.consume = %request.ContentType
    Set ensRequest.accept = $Get(%request.CgiEnvs("HTTP_ACCEPT"),"*/*")
    Set ensRequest.bodybody = ##class(petshop.model.Definition.Pet).%New()
    Do ensRequest.bodybody.%JSONImport(%request.Content)
    Return ##class(petshop.Utils).invokeHostAsync("petshop.bp.Process", ensRequest, "
petshop.bs.ProxyService")
}

ClassMethod GETgetPetById(petId As %String) As %Status
{
    Set ensRequest = ##class(petshop.msg.getPetByIdRequest).%New()
    Set ensRequest.consume = %request.ContentType
    Set ensRequest.accept = $Get(%request.CgiEnvs("HTTP_ACCEPT"),"*/*")
    Set ensRequest.pathpetId = petId
    Return ##class(petshop.Utils).invokeHostAsync("petshop.bp.Process", ensRequest, "
petshop.bs.ProxyService")
}
...
ClassMethod POSTuploadFile(petId As %String) As %Status
{
    Set ensRequest = ##class(petshop.msg.uploadFileRequest).%New()
    Set ensRequest.consume = %request.ContentType
    Set ensRequest.accept = $Get(%request.CgiEnvs("HTTP_ACCEPT"),"*/*")
    Set ensRequest.pathpetId = petId
    Set ensRequest.formDataadditionalMetadata = $Get(%request.Data("additionalMetadat
a",1))
```

```
    set mime = %request.GetMimeData("file")
    Do:$Isobject(mime) ensRequest.formDatafile.CopyFrom(mime)
    Return ##class(petshop.Utils).invokeHostAsync("petshop.bp.Process", ensRequest, "
petshop.bs.ProxyService")
}
...
}
```

Então, vamos testar a produção com este proxy REST.

Abra e inicie o petshop.Production

Crie um animal de estimação

Altere o número da porta, se necessário:

```
curl --location --request POST 'http://localhost:52795/petshoprest/pet' \
--header 'Content-Type: application/json' \
--data-raw '{
  "category": {
    "id": 0,
    "name": "string"
  },
  "id" : 456789,
  "name": "Kitty_Galore",
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}'
```

A produção é executada no modo assíncrono, portanto, a aplicação proxy restante não espera pela resposta. Este comportamento pode ser editado, mas normalmente, a produção de interoperabilidade usa o modo assíncrono. Veja o resultado no visualizador de mensagens e no traço visual

EDIÇÃO: desde a versão 1.1.0, a aplicação Proxy Rest funciona em modo de sincronização

Se tudo estiver bem, podemos observar um código http de status 200. Como você pode ver, recebemos uma resposta do corpo e ela não está analisada.

O que isso significa?
Isso ocorre quando resposta da aplicação/json ou a resposta 200 da especificação Swagger não é preenchida. Nesse caso, a resposta 200 não está preenchida.

Obtenha uma animal de estimação

Agora, tente obter o animal de estimação criado:

```
curl --location --request GET 'http://localhost:52795/petshoprest/pet/456789'
```

Verifique o traço visual:

Desta vez, esta é uma resposta da aplicação/json (a resposta 200 está completa nas especificações Swagger). Podemos ver um objeto de resposta analisado.

API REST - Gerar e baixar

Além disso, essa ferramenta pode ser hospedada em um servidor para permitir que os usuários gerem e baixem o código. Uma API REST e um formulário básico estão disponíveis:
* Aplicação web API REST: /swaggerclientgen/api
* Formulário básico: http://localhost:52795/csp/swaggerclientgen/dc.openapi.client.api.cspdem... **O gerador está disponível on-line no meu** servidor em nuvem aqui.

Nesse caso, o código é simplesmente gerado sem compilar, exportado e, em seguida, tudo é excluído.
Este recurso pode ser útil para centralização de ferramentas.
Veja o README.md para informações atualizadas. Obrigado pela leitura.

#Operação de negócios #Ferramentas #Interoperabilidade #REST API #InterSystems IRIS #Open Exchange
Confira o aplicativo relacionado no InterSystems Open Exchange

---

URL de origem:https://pt.community.intersystems.com/post/criando-uma-produ%C3%A7%C3%A3o-de-interoperabilidade-iris-partir-do-swagger