Artículo
Ricardo Paiva · 4 mar, 2021 Lectura de 8 min

Open Exchange

Cómo crear una producción de interoperabilidad en IRIS a partir de Swagger

¡Hola Comunidad!

Se acaba de lanzar OpenAPI-Client Gen, una aplicación para crear una producción cliente de interoperabilidad de IRIS a partir de la especificación Swagger 2.0.

En vez de la herramienta existente ^%REST que crea una aplicación REST del lado del servidor, OpenAPI-Client Gen crea una plantilla completa de producción cliente de interoperabilidad REST.

La instalación se realiza por ZPM:

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

¿Cómo generar la producción a partir de un documento Swagger?

Hacerlo es muy sencillo.

Abre un terminal y ejecuta:

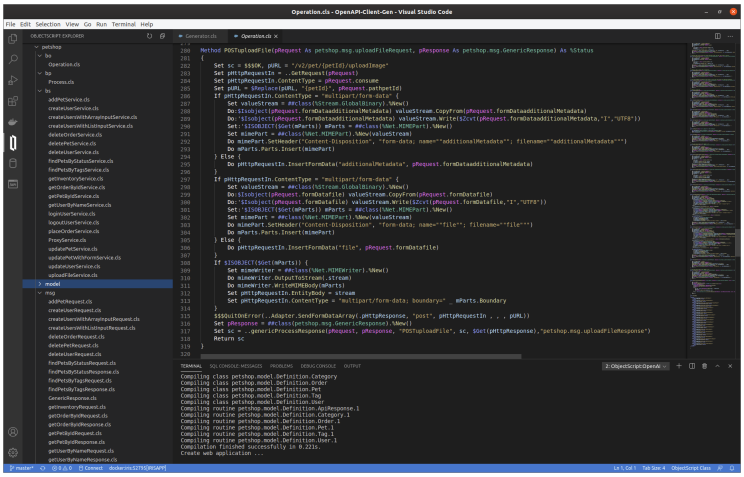
```
Set sc = ##class(dc.openapi.client.Spec).generateApp(<applicationName>, <Your Swagger
 2.0 document>>)
```

El primer argumento es el paquete objetivo donde se generarán las clases de la producción. El nombre del paquete debe ser un nombre válido e inexistente.
El segundo es el documento Swagger. Estos son los valores que se aceptan:
1) Ruta del archivo.
2) %DynamicObject.
3) URL.
La especificación debe estar en formato JSON.
Si tu especificación utiliza el formato YAML puede convertirse fácilmente a JSON con algunas herramientas online, como onlineyamltools.com
Ejemplo:

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

Echa un vistazo al código generado, podemos ver muchas clases divididas en muchos subpaquetes:
- Business Service: petshop.bs
- Business Operation: petshop.bo

- Business Process: petshop.bp
- REST Proxy application: petshop.rest
- Ens.Request y Ens.Response: petshop.msg
- Parsed input o Output object: petshop.model.Definition
- Production configuration class: petshop.Production



Business Operation class

Para cada servicio definido en el documento Swagger, hay un método relacionado llamado por <VERB><ServiceId>.

Análisis detallado de un método GETgetPetById simple generado

```
/// Returns a single pet
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
}
```

* En primer lugar, el objeto %Net.HttpRequest se crea por el método GetRequest. Si es necesario, no dudes en editarlo para agregar algunos encabezados.
* En segundo lugar, el objeto HttpRequest se completa utilizando pRequest petshop.msg.getPetByIdRequest' (subclase Ens.Request).
* En tercer lugar,EnsLib.HTTP.OutboundAdapterse utiliza para enviar una solicitud http.
* Y finalmente hay un proceso donde se origina una respuesta genérica por el 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
}
```

Entonces, podemos analizar un método un poco más complejo 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 puedes ver, es exactamente la misma lógica: GetRequest, completar %Net.HttpRequest, enviar solicitudes, procesar respuestas genéricas.

Proxy REST class

También se genera una Proxy REST application.
Esta clase REST utiliza un Projection para crear automáticamente la aplicación web relacionada (por ejemplo: "/petshoprest", consulta petshop.rest.REST y petshop.rest.Projection). Este proxy REST crea el mensaje Ens.Request y lo envía hacia 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")
}
...
}
```

Así que vamos a probar la producción con este REST proxy.

Abrir e iniciar petshop.Production

Cómo crear una tienda de mascotas

Modifícalo con tu número de puerto, si es necesario:

```
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"
}'
```

La producción se ejecuta en modo asíncrono, así que la rest proxy application no espera la respuesta. Este comportamiento podría editarse, pero generalmente, la producción de interoperabilidad utiliza el modo asíncrono. Puedes ver el resultado en Message Viewer y en Visual Trace.

Corrección: desde la versión 1.1.0, REST proxy application funciona en el modo de sincronización

Si todo funciona bien, podremos observar un código de estado http 200. Como puedes ver, recibimos una respuesta del cuerpo y esta no se analizó.

¿Qué quiere decir eso?

Esto ocurre cuando no es respuesta de la aplicación/json o la respuesta 200 de la especificación Swagger no está completa.
En este caso, la respuesta 200 no está completa.

Consigue una mascota

Ahora, intenta obtener la mascota que se creó:

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

Comprueba Visual Trace:

Esta vez, es una respuesta de la aplicación/json (la respuesta 200 se completó en la especificación de Swagger). Podemos ver el análisis de un objeto de respuesta.

API REST - Generar y descargar

Además, esta herramienta puede alojarse en un servidor para permitir que los usuarios generen y descarguen el código. La API REST y un formulario básico están disponibles:
* Aplicación web de API REST: /swaggerclientgen/api
* Formulario básico: http://localhost:52795/csp/swaggerclientgen/dc.openapi.client.api.cspdem... **El generador está disponible online en el** servidor que tengo en la nube aquí.

En este caso, el código simplemente se genera sin compilar o exportar, y después se elimina por completo.
Esta función puede ser útil para la centralización de herramientas.
Consulta el archivo README.md para obtener información actualizada. Gracias por leer el artículo.

#API REST #Herramientas #Interoperabilidad #Operación empresarial #InterSystems IRIS #Open Exchange
Ir a la aplicación en InterSystems Open Exchange

URL de fuente:https://es.community.intersystems.com/post/c%C3%B3mo-crear-una-producci%C3%B3n-de-interoperabilidad-en-iris-partir-de-swagger