Artículo
Alberto Fuentes · 25 mayo, 2021 Lectura de 12 min

Cómo desarrollar una API REST con un enfoque spec-first

En este artículo, me gustaría hablar sobre el enfoque *spec-first* para el desarrollo de una API REST.

Mientras que el desarrollo tradicional *code-first* de una API REST es así:

- Escribir el código
- Habilitarlo en REST
- Documentarlo (como una API REST)

Spec-first sigue los mismos pasos, pero a la inversa. Comenzamos con una especificación, — que también actúa como documentación — , generamos el código base de la aplicación REST a partir de ella, y finalmente escribimos la lógica de negocio concreta que nos haga falta.

Esto ofrece varias ventajas:

- Siempre se dispone de documentación relevante y útil para desarrolladores externos o de frontend que quieran utilizar tu API REST
- La especificación creada en OAS (Swagger) se puede importar a una variedad de herramientas que permiten la edición, generación de clientes, administración de la API, pruebas unitarias y automatización o simplificación de muchas otras tareas
- Arquitectura de la API mejorada. En el enfoque *code-first*, la API se desarrolla método a método, por lo que un desarrollador puede perder fácilmente la pista de la arquitectura general de la API. Sin embargo, con el enfoque *spec-first*, el desarrollador se ve obligado a interactuar con una API desde la posición de consumidor de la misma, lo que con frecuencia puede ayudarle a diseñar una arquitectura de API más limpia
- Desarrollo más rápido: como todo el código base se genera automáticamente, no tendrás que escribirlo, lo único que necesitas es desarrollar la lógica de negocio de tu aplicación en particular.
- Obtienes sugerencias de forma más rápida: los consumidores pueden obtener una visión de la API inmediatamente y pueden ofrecer sugerencias de forma más sencilla, simplemente modificando la especificación.

¡Vamos a desarrollar nuestra API con un enfoque *spec-first*!

Plan

1. Desarrollo de la especificación en swagger
    - Docker
    - Localmente
    - Online
2. Carga de la especificación en IRIS
    - API management REST API
    - ^REST
    - Clases
3. ¿Qué pasó con nuestra especificación?
4. Implementación
5. Desarrollos posteriores
6. Consideraciones

    - Parámetros especiales
    - CORS
7. Carga de la especificación en IAM

Desarrollo de la especificación

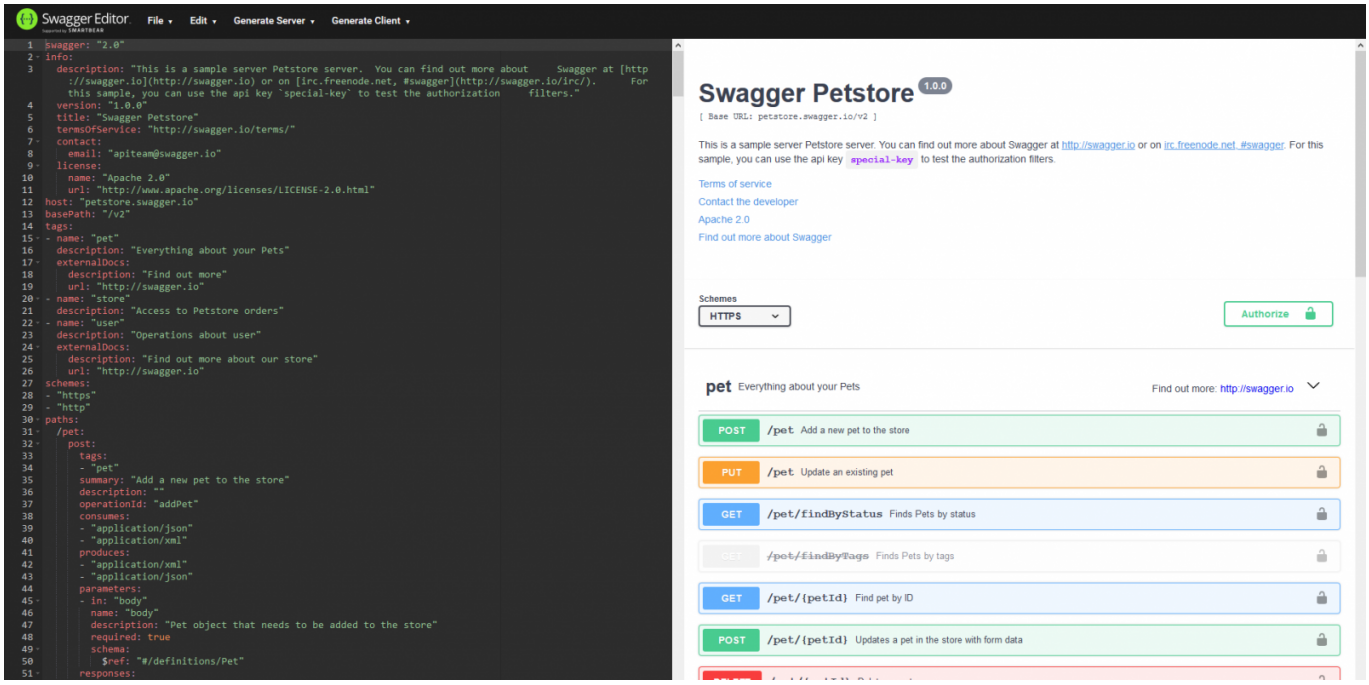
El primer paso es, naturalmente, escribir la especificación. InterSystems IRIS es compatible con la Open API Specification (OAS):

> **OpenAPI Specification** (anteriormente Swagger Specification) es un formato de descripción de las APIs para API REST. Un archivo OpenAPI te permite describir toda tu API, incluyendo:
>
> - Endpoints disponibles (/users) y operaciones en cada endpoint (GET /users, POST /users)
> - Los parámetros de operación entrada y salida, para cada operación
> - Métodos de autenticación
> - Información de contacto, licencia, términos de uso y otro tipo de información
>
> Las especificaciones de la API se pueden escribir en YAML o JSON. El formato es fácil de aprender y leer tanto para los humanos como para las máquinas. La especificación completa de OpenAPI se puede encontrar en GitHub: Especificación OpenAPI 3.0
>
> - de la documentación de Swagger.

Utilizaremos Swagger para escribir nuestra API. Hay varias formas de utilizar Swagger:

- Online
- Docker: docker run -d -p 8080:8080 swaggerapi/swagger-editor
- Instalación local

Después de instalar/ejecutar Swagger, deberías ver esta ventana en un navegador web:



A la izquierda se edita la especificación de la API; y a la derecha se ve inmediatamente la herramienta de

prueba/documentación de la API representada de una forma visual.

Vamos a cargar nuestra primera especificación de la API en él (en YAML). Se trata de una API sencilla con una solicitud GET que devuelve un número aleatorio en un rango especifico.

Especificación Math API

Esto es en lo que consiste.

Información básica sobre nuestra API y la versión de OAS utilizada.

```
swagger: "2.0"
info:
  description: "Math"
  version: "1.0.0"
  title: "Math REST API"
```

Host del servidor, protocolo (http, https) y nombres de las aplicaciones web:

```
host: "localhost:52773"
basePath: "/math"
schemes:
  - http
```

A continuación, especificamos una ruta (por lo que la URL completa sería http://localhost:52773/math/random/:min/:max) y el método de solicitud HTTP (get, post, put, delete):

```
paths:
  /random/{min}/{max}:
    get:
```

A continuación, especificamos la información sobre nuestra solicitud:

```
      x-ISC_CORS: true
      summary: "Get random integer"
      description: "Get random integer between min and max"
      operationId: "getRandom"
      produces:

      - "application/json"
      parameters:

      - name: "min"
        in: "path"
        description: "Minimal Integer"
        required: true
        type: "integer"
        format: "int32"
```

```
    - name: "max"
      in: "path"
      description: "Maximal Integer"
      required: true
      type: "integer"
      format: "int32"
    responses:
      200:
        description: "OK"
```

En esta parte definimos nuestra solicitud:

- Habilitar esta ruta para CORS (explicaremos esto más adelante)
- Proporcionar *summary* y *description*
- *operationId* permite una referencia dentro de las propias especificaciones, además es un nombre de método generado en nuestra clase de implementation
- *produces*: formato de respuesta (como text, xml, json)
- *parameters* especifica los parámetros de entrada (ya sea en la URL o en el cuerpo); en nuestro caso especificamos 2 parámetros, el rango para nuestro generador de números aleatorios
- *responses* lista de respuestas posibles del servidor

Como ves, este formato no es especialmente difícil, aunque hay muchas más funciones disponibles, aquí hay una especificación.

Por último, vamos a exportar nuestra definición como JSON. Ir a File → Convert y guardar como JSON. La especificación debería tener este aspecto:

Especificación Math API

Carga de la especificación en IRIS

Ahora que tenemos nuestra especificación, podemos generar el código base para esta API REST en InterSystems IRIS.

Para pasar a esta etapa necesitaremos tres cosas:

- Nombre de la aplicación REST: paquete para nuestro código generado (digamos math)
- La especificación OAS en formato JSON: la acabamos de crear en un paso anterior
- Nombre de la aplicación WEB: una ruta base para acceder a nuestra API REST (/math en nuestro caso)

Hay tres maneras de utilizar nuestra especificación para generar códigos, que son esencialmente lo mismo y solo ofrecen varias maneras de acceder a la misma función

1. Llamar a la rutina ^%REST (Do ^%REST en una sesión terminal interactiva), documentación.
2. Llamar a la clase %REST (Set sc = ##class(%REST.API).CreateApplication(applicationName, spec), forma no interactiva), documentación.
3. Utilizar API Management REST API, documentación.

Creo que la documentación describe adecuadamente los pasos necesarios, así que solo tienes que seleccionar uno. Añadiré dos notas:

- En los casos (1) y (2) puedes transmitir a un objeto dinámico, un nombre de archivo o una URL
- En los casos (2) y (3) **debes** realizar una llamada adicional para crear una aplicación WEB: set sc = #class(%SYS.REST). DeployApplication(restApp, webApp, authenticationType), así que en nuestro caso set sc = ##class(%SYS.REST). DeployApplication("math", "/math"), obtenemos los valores del argumento authenticationType desde el archivo *include* %sySecurity, las entradas correspondientes son $$$Authe*, por lo que para el acceso no autenticado transmitimos $$$AutheUnauthenticated. Si se omite, el parámetro se ajusta de forma predeterminada a la autenticación por contraseña.

¿Qué pasó con nuestra especificación?

Si has creado la aplicación con éxito, debería crearse un nuevo paquete math con tres clases:

- *Spec*: almacena la especificación tal y como está.
- *Disp*: se llama directamente cuando se invoca el servicio REST. Se ocupa de la gestión del manejo de REST y llama a los métodos de implementación.
- *Impl*: contiene la implementación interna real del servicio REST. Solo deberías editar esta clase.

Documentación con más información sobre las clases.

Implementación

Inicialmente nuestra *implementation class* math.impl contiene solo un método, que corresponde a nuestra operación /random/{min}/{max}:

```
/// Get random integer between min and max&lt;br/>
/// The method arguments hold values for:&lt;br/>
///     min, Minimal Integer&lt;br/>
///     max, Maximal Integer&lt;br/>
ClassMethod getRandom(min As %Integer, max As %Integer) As %DynamicObject
{
    //(Place business logic here)
    //Do ..%SetStatusCode(&lt;HTTP_status_code>)
    //Do ..%SetHeader(&lt;name>,&lt;value>)
    //Quit (Place response here) ; response may be a string, stream or dynamic object
}
```

Comenzaremos con la implementación sencilla:

```
ClassMethod getRandom(min As %Integer, max As %Integer) As %DynamicObject
{
    quit {"value":($random(max-min)+min)}
}
```

Y finalmente podemos llamar a nuestra API REST abriendo esta página en el navegador:
http://localhost:52773/math/random/1/100

El resultado debería ser:

```
{
    "value": 45
}
```

También en el editor Swagger al presionar el botón Try it out, y rellenando los parámetros de la solicitud, también se enviaría la misma solicitud:

¡Enhorabuena! ¡Nuestra primera API REST creada con un enfoque *spec-first* ya está funcionando!

Desarrollos posteriores

Por supuesto, nuestra API no es estática y tenemos que agregar nuevas rutas, y así sucesivamente. Con el desarrollo *spec-first*, empiezas por modificar la especificación, después actualizas la aplicación REST (las mismas llamadas que para crear la aplicación) y finalmente escribes el código. Ten en cuenta que las actualizaciones de la especificación son seguras: tu código no se verá afectado, incluso si la ruta se elimina de una especificación, el método no se eliminará en la *implementation class*.

Consideraciones

¡Más notas!

Parámetros especiales

InterSystems añadió parámetros especiales a la especificación de swagger. Son estos:

Nombre	Tipo de datos	Predeterminado	Lugar	Descripción
x-ISCDispatchParent	classname	%CSP.REST	información	Superclase para la clase *dispatch*.
x-ISCCORS	booleano	falso	operación	Marca para indicar que las solicitudes CORS para esta combinación de endpoint/método deben ser soportadas.
x-ISCRequiredResource	matriz		operación	Lista separada por comas de los recursos definidos y sus modos de acceso (resource:mode) que se requieren para acceder a este endpoint del servicio REST. Por ejemplo: ["%Development:USE"]
x-ISCServiceMethod	cadena		operación	Nombre del método de clase llamado en el back end para dar servicio a esta operación; de forma predeterminada es operationId, que normalmente es el más adecuado.

CORS

Hay tres maneras de activar el soporte de CORS.

1. En cada ruta, especificando x-ISCCORS como verdadero. Eso es lo que hemos hecho en nuestra API

REST Math.

2. En cada API, añadiendo

   Parameter HandleCorsRequest = 1;

y recompilando la clase. También sobreviviría a la actualización de las especificaciones.

3. (Recomendada) En cada API, mediante la implementación de la superclase *custom dispatcher* (debe extender %CSP.REST), y escribiendo la lógica del procesamiento de CORS allí. Para utilizar esta superclase, añade x-ISCDispatchParent a tu especificación.

Carga de la especificación en IAM

Por último, vamos a añadir nuestra especificación en IAM para que sea publicada por otros desarrolladores.

Si no has comenzado con IAM, consulta este artículo. También explica cómo publicar APIs REST por medio de IAM, por eso no lo describimos aquí. Es posible que quieras modificar los parámetros spec host y basepath para que apunten a IAM, en lugar de a la instancia de InterSystems IRIS.

Abre el portal del Administrador de IAM y ve a la pestaña Specs en el espacio de trabajo correspondiente.

Haz clic en el botón Add Spec e introduce el nombre de la nueva API (math en nuestro caso). Después de crear nuevas especificaciones en IAM, haz clic en Editar y pega el código de las especificaciones (JSON o YAML, no importa para IAM):

No olvides hacer clic en Update file.

Ahora nuestra API está publicada para los desarrolladores. Abre el *Developer Portal* y haz clic en Documentation en la esquina superior derecha. Además de las tres API predeterminadas, nuestra nueva API REST Math debería estar disponible:

Ábrela:

¡Ahora los desarrolladores pueden ver la documentación de nuestra nueva API y probarla en el mismo lugar!

Conclusiones

InterSystems IRIS simplifica el proceso de desarrollo de una API REST y el enfoque spec-first permite una gestión más rápida y sencilla del ciclo de vida de la API REST. Con este enfoque, puedes utilizar una variedad de herramientas para una variedad de tareas relacionadas, como la generación de clientes, las pruebas unitarias, la administración de la API y muchas otras más.

Enlaces

- Especificación OpenAPI 3.0
- Cómo crear servicios REST
- Presentación de IAM
- Documentación de IAM

#API #API REST #InterSystems API Manager (IAM) #Mejores prácticas #InterSystems IRIS

URL de fuente:https://es.community.intersystems.com/post/c%C3%B3mo-desarrollar-una-api-rest-con-un-enfoque-spec-first