Artigo
Eduard Lebedyuk · Nov. 9, 2020 12min de leitura

Desenvolvendo API REST com uma abordagem de especificação primeiro (spec-first)

Neste artigo eu gostaria de falar sobre a abordagem de especificação primeiro (spec-first) para o desenvolvimento de APIs REST.

Embora o desenvolvimento de API REST com código primeiro (code-first) tradicional seja assim:

- Escrever o código
- Habilitando-o com REST
- Documentando-o (como uma API REST)

A especificação primeiro (spec-first) segue os mesmos passo, mas ao contrário. Começamos com uma especificação, também usando-a como documentação, geramos uma aplicação REST padrão a partir dela e, finalmente, escrevemos alguma lógica de negócios.

Isso é vantajoso porque:

- Você sempre tem uma documentação relevante e útil para desenvolvedores externos ou front-end que desejam usar sua API REST
- A especificação criada em OAS (Swagger) pode ser importada em uma variedade de ferramentas permitindo edição, geração de cliente, gerenciamento de API, teste de unidade e automação ou simplificação de muitas outras tarefas
- Arquitetura de API aprimorada. Na abordagem de código primeiro (code-first), a API é desenvolvida método a método então um desenvolvedor pode facilmente perder o controle da arquitetura geral da API, no entanto, com a especificação primeiro (spec-first), o desenvolvedor é forçado a interagir com uma API a partir da posição de um consumidor de API, o que geralmente ajuda no design de uma arquitetura melhor da API.
- Desenvolvimento mais rápido - como todo código padrão é gerado automaticamente, você não terá que escrevê-lo, tudo o que resta é desenvolver a lógica de negócios.
- Loops de feedback mais rápidos - os consumidores podem obter uma visão da API imediatamente e podem oferecer sugestões com mais facilidade, simplesmente modificando as especificações
  Vamos desenvolver nossa API em uma abordagem de especificação primeiro!

Plano

1. Desenvolver especificação no swagger
   - Docker
   - Localmente
   - On-line
2. Carregar especificações no IRIS
   - API REST de Gerenciamento de API
   - ^REST
   - Classes
3. O que aconteceu com a nossa especificação?
4. Implementação
5. Desenvolvimento adicional
6. Considerações

    - Parâmetros especiais
    - CORS
7. Carregar especificações no IAM

Desenvolver especificação

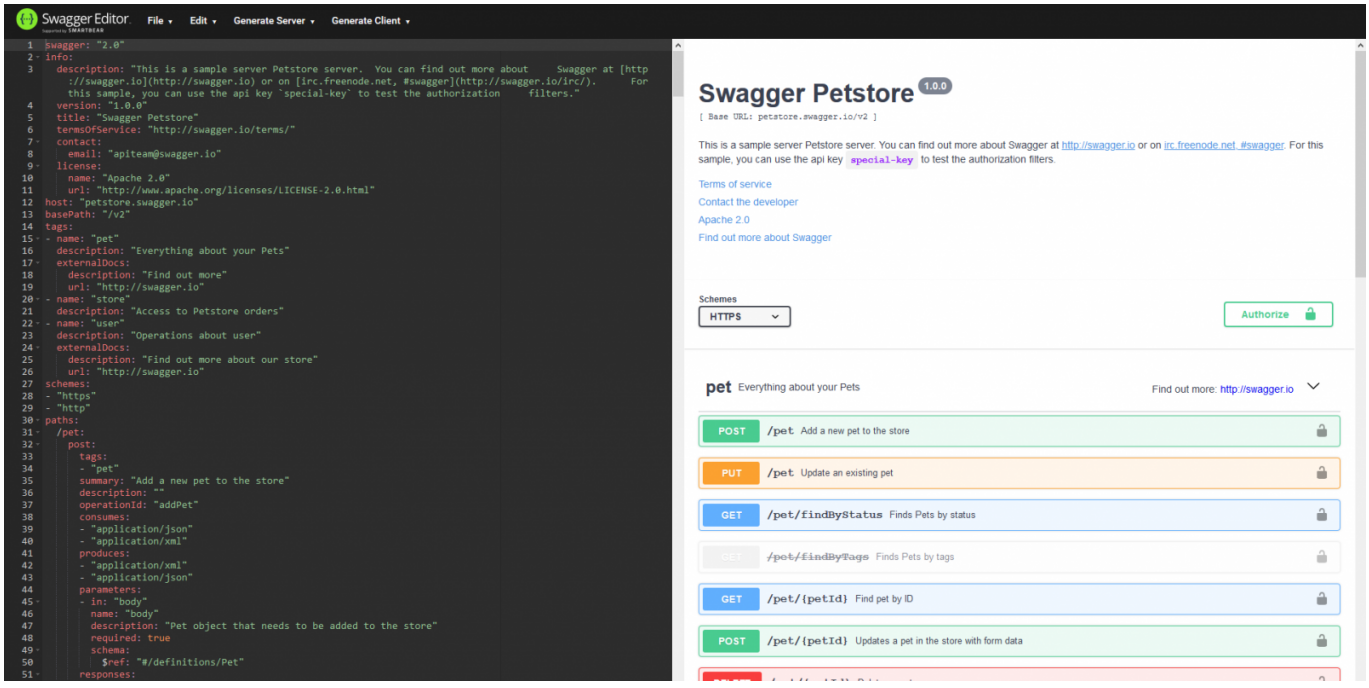
O primeiro passo é, sem surpresa, escrever a especificação. O InterSystems IRIS oferece suporte à Especificação Open API (OAS):

> A **Especificação OpenAPI** (anteriormente Especificação Swagger) é um formato de descrição de API para APIs REST. Um arquivo OpenAPI permite que você descreva toda a sua API, incluindo:
>
> - Endpoints disponíveis (/users) e operações em cada endpoint (GET /users, POST /users)
> - Parâmetros de entrada e saída para cada operação
> - Métodos de autenticação
> - Informações de contato, licença, termos de uso e outras informações.
>
> As especificações das APIs podem ser escritas em YAML ou JSON. O formato é fácil de aprender e legível tanto para humanos como para máquinas. A Especificação OpenAPI completa pode ser encontrada no GitHub: Especificação OpenAPI 3.0
>
> - da documentação Swagger.

Usaremos Swagger para escrever nossa API. Existem várias maneiras de usar o Swagger:

- On-line
- Docker: docker run -d -p 8080:8080 swaggerapi/swagger-editor
- Instalação local

Após instalar/executar o Swagger, você deverá ver esta janela em um navegador web:



No lado esquerdo, você edita a especificação da API e, à direita, vê imediatamente a documentação/ferramenta de teste da API renderizada.

Vamos carregar nossa primeira especificação de API nele (em YAML). É uma API simples com uma solicitação GET - retornando um número aleatório em um intervalo especificado.

Especificação da API matemática

Aqui está seu conteúdo:

Informações básicas sobre nossa API e versão OAS usada.

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

Host do servidor, protocolo (http, https) e nomes de aplicações web:

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

Em seguida, especificamos um caminho (para que a URL completa seja http://localhost:52773/math/random/:min/:max) e o método de solicitação HTTP (get, post, put, delete):

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

Depois disso, especificamos informações sobre nossa solicitação:

```
      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"
```

Nesta parte, definimos nossa solicitação:

- Habilita suporte a CORS (falarei mais sobre isso posteriormente)
- Fornece um *resumo* e *descrição*
- O *operationId* permite referência dentro das especificações, também é um nome de método gerado em nossa classe de implementação
- *produz* - formato de resposta (como texto, xml, json)
- *parâmetros* especificam parâmetros de entrada (sejam eles em URL ou corpo), no nosso caso especificamos 2 parâmetros - intervalo para nosso gerador de número aleatório
- *respostas* lista respostas possíveis do servidor

Como você pode ver, este formato não é particularmente desafiador, embora haja muitos outros recursos disponíveis, aqui está uma especificação.

Finalmente, vamos exportar nossa definição como um JSON. Vá em File → Convert and save as JSON. A especificação deve ser semelhante a esta:

Especificação da API matemática

Carregar especificação no IRIS

Agora que temos nossas especificações, podemos gerar um código padrão para esta API REST no InterSystems IRIS.

Para passar para este estágio, precisaremos de três coisas:

- Nome da aplicação REST: pacote para nosso código gerado (utilizaremos math)
- Especificação OAS em formato JSON: acabamos de criá-la em uma etapa anterior
- Nome da aplicação WEB: um caminho base para acessar nossa API REST (/math em nosso caso)

Existem três maneiras de usar nossa especificação para geração de código, elas são essencialmente as mesmas e apenas oferecem várias maneiras de acessar a mesma funcionalidade

1. Chamar a rotina ^%REST (Do ^%REST em uma sessão de terminal interativa), documentação.
2. Chamar a classe %REST (Set sc = ##class(%REST.API).CreateApplication(applicationName, spec), não interativa), documentação.
3. Usar a API REST de gerenciamento de API, documentação.

Acho que a documentação descreve adequadamente as etapas necessárias, então apenas escolha uma. Vou adicionar duas notas:

- No caso (1) e (2) você pode passar a um objeto dinâmico, um nome de arquivo ou uma URL
- Nos casos (2) e (3) você **deve** fazer uma chamada adicional para criar uma aplicação WEB:: set sc = ##class(%SYS.REST).DeployApplication(restApp, webApp, authenticationType), então em nosso caso, set sc = ##class(%SYS.REST).DeployApplication("math", "/math"), obter valores para o argumento authenticationType do arquivo de inclusão %sySecurity, entradas relevantes são $$$Authe*, então para um acesso não autenticado use $$$AutheUnauthenticated. Se omitido, o parâmetro padroniza para autenticação de senha.

O que aconteceu com a nossa especificação?

Se você criou a aplicação com sucesso, um novo pacote math deve ter sido criado com três classes:

- *Spec* - armazena a especificação no estado em que se encontra.
- *Disp* - chamado diretamente quando o serviço REST é chamado. Ele empacota o tratamento REST e chama os métodos de implementação.
- *Impl* - contém a implementação interna atual do serviço REST. Você deve editar apenas esta classe.

Documentação com mais informações sobre as classes.

Implementação

Inicialmente, nossa classe de implementação math.impl contém apenas um método, correspondendo à nossa operação /random/{min}/{max}:

```
///  Obtenha um número inteiro aleatório entre min e max<br/>
/// Os argumentos do método contêm valores para:<br/>
///     min, número inteiro mínimo<br/>
///     max, número inteiro máximo<br/>
ClassMethod getRandom(min As %Integer, max As %Integer) As %DynamicObject
{
    //(Place business logic here)
    //Do ..%SetStatusCode(<HTTP_status_code>)
    //Do ..%SetHeader(<name>,<value>)
    //Quit (Coloque a resposta aqui) ; a resposta pode ser uma string, uma stream ou
um objeto dinâmico
}
```

Vamos começar com a implementação trivial:

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

E, finalmente, podemos chamar nossa API REST abrindo esta página no navegador:
http://localhost:52773/math/random/1/100

A saída deve ser:

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

Também no editor Swagger, pressionando o botão Try it out e preenchendo os parâmetros da solicitação, também será enviada a mesma solicitação:

Parabéns! Nossa primeira API REST criada com uma abordagem de especificação primeiro (spec-first) está agora disponível!

Desenvolvimento adicional

Claro, nossa API não é estática e precisamos adicionar novos caminhos e assim por diante. Com o desenvolvimento de especificação primeiro, você começa modificando a especificação, em seguida atualiza a aplicação REST (utilizando as mesmas chamadas que foram utilizadas para criar a aplicação) e finalmente escreve o código. Observe que as atualizações de especificações são seguras: seu código não é afetado, mesmo se o caminho for removido de uma especificação, o método não seria excluído da classe de implementação.

Considerações

Mais notas!

Parâmetros especiais

A InterSystems adicionou parâmetros especiais à especificação swagger, aqui estão:

Nome	Tipo de Dado	Padrão	Local	Descrição
x-ISCDispatchParent	classname	%CSP.REST	info	Super classe para a classe de despacho.
x-ISCCORS	boolean	false	operation	Flag para indicar que requisições CORS para esta combinação endpoint/método deve ser suportada.
x-ISCRequiredResource	array		operation	Lista separada por vírgulas dos recursos definidos e seus modos de acesso (recurso:modo) que são requeridos para acesso a este endponit do serviço REST. Exemplo: ["%Development:USE"]
x-ISCServiceMethod	string		operation	Nome do método de

				classe invocado internamente para atender a esta operação; o valor padrão é o operationId, o que é normalmente adequado.

CORS

Existem três maneiras de habilitar o suporte ao CORS.

1. Em uma rota na rota base, especificando x-ISCCORS como verdadeiro (true). Isso é o que fizemos em nossa API REST de matemática.

2. Por API, adicionando

   Parameter HandleCorsRequest = 1;

e recompilando a classe. Ele também sobreviveria à atualização das especificações.

3. (Recomendado) Por API, implementando a superclasse de dispatcher customizada (deve estender %CSP.REST) e escrevendo a lógica de processamento CORS lá. Para usar esta superclasse, adicione x-ISCDispatchParent à sua especificação.

Carregar especificações no IAM

Finalmente, vamos adicionar nossa especificação no IAM para que seja publicada para outros desenvolvedores.

Se você ainda não começou a utilizar o IAM, consulte este artigo. Ele também fala a respeito da disponibilização da API REST através do IAM, por isso não o descreverei aqui. Você pode querer modificar a especificação do host e os parâmetros do basepath para que eles apontem para o IAM, em vez de apontar para a instância InterSystems IRIS.

Abra o portal do administrador do IAM e acesse a aba Specs no espaço de trabalho relevante.

Clique no botão Add Spec e insira o nome da nova API (math em nosso caso). Depois de criar uma nova especificação no IAM, clique em Edit e cole o código da especificação (JSON ou YAML - não importa para o IAM):

Não se esqueça de clicar em Update File.

Agora nossa API está publicada para desenvolvedores. Abra o Portal do Desenvolvedor e clique em Documentação no canto superior direito. Além das três APIs padrão, nossa nova API REST Math deve estar disponível:

Abra-a:

Agora os desenvolvedores podem ver a documentação de nossa nova API e testá-la no mesmo lugar!

Conclusão

O InterSystems IRIS simplifica o processo de desenvolvimento de APIs REST e, a abordagem de especificação primeiro (spec-first) permite um gerenciamento mais rápido e fácil do ciclo de vida das APIs REST. Com essa abordagem, você pode usar uma variedade de ferramentas para uma variedade de tarefas relacionadas, como geração de cliente, teste de unidade, gerenciamento de API e muitos outros.

Links

- Especificação OpenAPI 3.0
- Criação de serviços REST
- Começando com IAM
- Documentação IAM

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

URL de origem:https://pt.community.intersystems.com/post/desenvolvendo-api-rest-com-uma-abordagem-de-especifica%C3%A7%C3%A3o-primeiro-spec-first