Article
Iryna Mykhailova · Juin 27, 2022 7m de lecture

Open Exchange

Transmission de fichiers via REST pour les stocker en Property, partie 1

Une question a été posée dans la communauté des développeurs d'InterSystems concernant la possibilité de créer une interface TWAIN pour une application Caché. Il y a eu plusieurs suggestions intéressantes sur la façon d'obtenir des données d'un périphérique d'acquisition d'images sur un client Web vers un serveur, puis de stocker ces données dans une base de données

Toutefois, pour mettre en œuvre l'une de ces suggestions, vous devez être en mesure de transférer des données d'un client Web vers un serveur de base de données et de stocker les données reçues dans une propriété de classe (ou une cellule de tableau, comme c'était le cas dans la question). Cette technique peut être utile non seulement pour transférer des données d'images provenant d'un périphérique TWAIN, mais aussi pour d'autres tâches telles que l'organisation d'une archive de fichiers, d'un partage d'images, etc.

Ainsi, l'objectif principal de cet article est de montrer comment écrire un service RESTful pour obtenir des données du corps d'une commande HTTP POST, soit à l'état brut, soit enveloppées dans une structure JSON.

Principes de base de REST

Avant d'entrer dans les détails, permettez-moi de dire quelques mots sur REST en général et sur la façon dont les services RESTful sont créés dans IRIS.

Le transfert d'état représentationnel (REST) est un style architectural pour les systèmes hypermédia distribués. L'abstraction clé de l'information dans REST est une ressource qui a son propre identifiant et qui peut être représentée dans un format JSON, XML ou un autre format connu à la fois du serveur et du client.

En général, HTTP est utilisé pour transférer des données entre le client et le serveur. Chaque opération CRUD (création, lecture, mise à jour, suppression) a sa propre méthode HTTP (POST, GET, PUT, DELETE), tandis que chaque ressource ou collection de ressources a son propre URI.

Dans cet article, je n'utiliserai que la méthode POST pour insérer une nouvelle valeur dans la base de données, et j'ai donc besoin de connaître ses contraintes.

Selon la spécification IETF RFC7231 4.3.3 Post POST n'a pas de limite quant à la taille des données stockées dans son corps. Mais différents serveurs web et navigateurs imposent leurs propres limites, généralement de 1 Mo à 2 Go. Par exemple, Apache autorise un maximum de 2 Go. Dans tous les cas, si vous devez envoyer un fichier de 2 Go, vous devriez peut-être reconsidérer votre approche.

Exigences pour un service RESTful dans IRS

Dans IRIS, pour mettre en œuvre un service RESTful, vous devez :

Créer une classe broker qui étend la classe abstraite %CSP.REST. Celle-ci, à son tour, étend %CSP.Page, ce qui permet d'accéder à différentes méthodes, paramètres et objets utiles, en particulier %request).
Spécifier UrlMap pour définir les itinéraires.
En option, définir le paramètre UseSession pour préciser si chaque appel REST est exécuté dans sa propre session Web ou s'il partage une seule session avec d'autres appels REST.
Fournir des méthodes de classe pour effectuer les opérations définies dans les itinéraires
Définissez l'application web CSP et spécifiez sa sécurité sur la page de l'application web (Administration système > Sécurité > Applications > Applications web), où la classe Dispatch doit contenir le nom de la classe utilisateur et Name, la première partie de l'URL pour l'appel REST.
En général, il existe plusieurs façons d'envoyer de gros volumes de données (fichiers) et leurs métadonnées du client au serveur, par exemple :

Coder le fichier et les métadonnées en base64 et ajouter des frais de traitement au serveur et au client pour le codage/décodage.
Envoyer d'abord le fichier et renvoyez un ID au client, qui envoie ensuite les métadonnées contenant l'ID. Le serveur réassocie le fichier et les métadonnées.
Envoyer d'abord le fichier et renvoyez un ID au client, qui envoie ensuite le fichier contenant l'ID. Le serveur réassocie le fichier et les métadonnées.
Dans ce premier article, je vais essentiellement utiliser la deuxième approche, mais sans renvoyer l'ID au client et en ajoutant les métadonnées (le nom du fichier à stocker comme une autre propriété) car cette tâche n'a rien d'exceptionnel. Dans un deuxième article, j'utiliserai la première approche, mais j'empaqueterai mon fichier et les métadonnées (le nom du fichier) dans une structure JSON avant de l'envoyer au serveur.

Mise en œuvre du service RESTFul

Passons maintenant aux détails. Tout d'abord, définissons la classe, dont nous allons définir les propriétés :

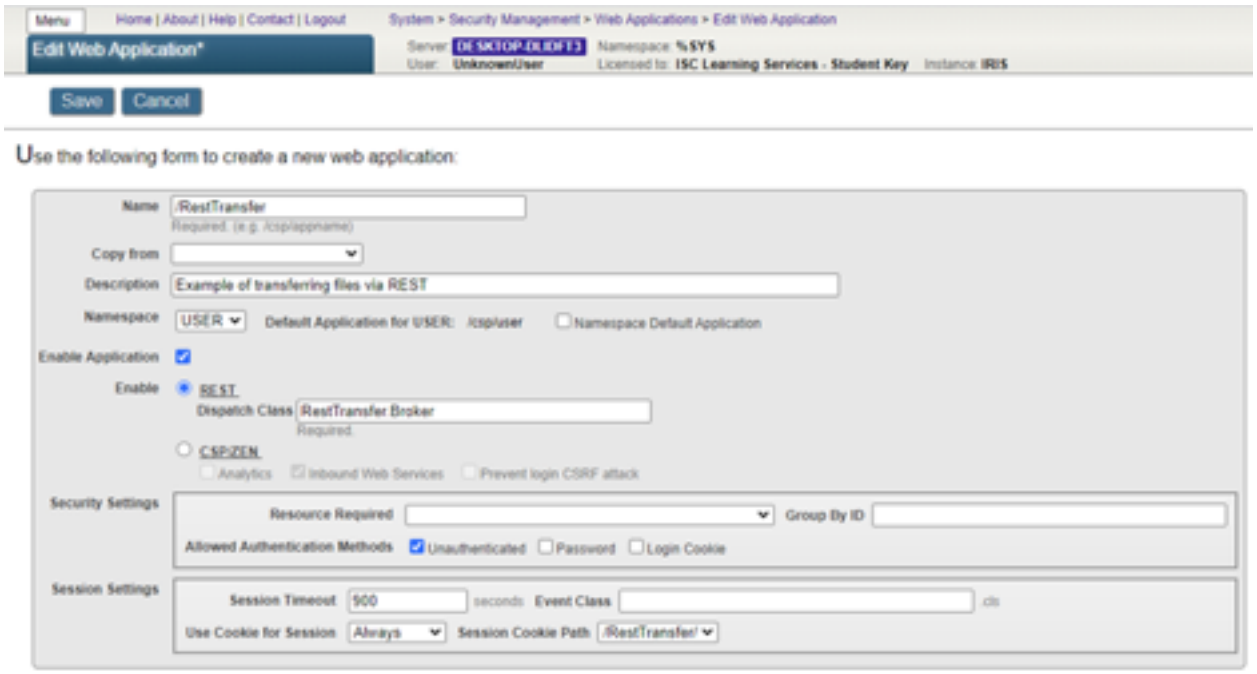
```
Class RestTransfer.FileDesc Extends %Persistent
{
  Property File As %Stream.GlobalBinary;
  Property Name As %String;
}
```

Bien sûr, vous aurez généralement plus de métadonnées pour accompagner le fichier, mais cela devrait suffire pour nos besoins.

Ensuite, nous devons créer un service broker de classe, que nous développerons plus tard avec des itinéraires et des méthodes :

```
Class RestTransfer.Broker Extends %CSP.REST
{
  XData UrlMap
  {
    <Routes>
    </Routes>
  }
}
```

Et enfin, pour la configuration préliminaire, nous devons spécifier cette application dans une liste d'applications web :



Maintenant que la configuration préliminaire est faite, nous pouvons écrire des méthodes pour enregistrer un fichier reçu d'un client REST (j'utiliserai le client Advanced REST Client) dans une base de données en tant que propriété File d'une instance de la classe RestTransfer.FileDesc.

Nous allons commencer par travailler avec des informations stockées sous forme de données brutes dans le corps de la méthode POST. Tout d'abord, ajoutons un nouvel itinéraire à UrlMap :

```
<Route Url="/file" Method="POST" Call="InsertFileContents"/>
```

Cet itinéraire spécifie que lorsque le service reçoit une commande POST avec l'URL /RestTransfer/file, il doit lancer la méthode de classe InsertFileContents. Pour faciliter les choses, à l'intérieur de cette méthode, je vais créer une nouvelle instance de la classe RestTransfer.FileDesc et définir sa propriété fichier File sur les données reçues. Cela renvoie le statut et un message formaté en JSON indiquant le succès ou l'erreur. Voici la méthode de classe :

```
ClassMethod InsertFileContents() As %Status
{
  Set result={}
  Set st=0
  set f = ##class(RestTransfer.FileDesc).%New()
  if (f = $$$NULLOREF) {
    do result.%Set("Message","Couldn't create an instance of the class")
  } else {
    set st = f.File.CopyFrom(%request.Content)
    If $$$ISOK(st) {
      set st = f.%Save()
      If $$$ISOK(st) {
        do result.%Set("Status","OK")
      } else {
        do result.%Set("Message",$system.Status.GetOneErrorText(st))
      }
    } else {
      do result.%Set("Message",$system.Status.GetOneErrorText(st))
```

```
    }
  }
  write result.%ToJSON()
  Quit st
}
```

Tout d'abord, une nouvelle instance de la classe RestTransfer.FileDesc est créée et on vérifie qu'elle a été créée avec succès et que nous avons un OREF. Si l'objet n'a pas pu être créé, nous formons une structure JSON :

```
{"Message", "Couldn't create an instance of class"}
```

Si l'objet a été créé, le contenu de la requête est copié dans la propriété fichier File. Si la copie n'a pas réussi, nous formons une structure JSON avec une description de l'erreur :

```
{"Message",$system.Status.GetOneErrorText(st)}
```

Si le contenu a été copié, nous sauvegardons l'objet, et si la sauvegarde est réussie, nous formons un JSON {"Status","OK"}. Dans le cas contraire, le JSON renvoie une description de l'erreur.

Enfin, nous écrivons le JSON dans une réponse et retournons l'état st.

Voici un exemple de transfert d'une image :

La manière dont il est sauvegardé dans la base de données :

Nous pouvons enregistrer ce flux dans un fichier et constater qu'il a été transféré sans être modifié :

```
set f = ##class(RestTransfer.FileDesc).%OpenId(4)
set s = ##class(%Stream.FileBinary).%New()
set s.Filename = "D:\Downloads\test1.jpg"
do s.CopyFromAndSave(f.File)
```

La même chose peut être faite avec un fichier texte :

Cela sera également visible dans les globales :

Et nous pouvons stocker des fichiers exécutables ou tout autre type de données :

Nous pouvons vérifier la dimension dans les globales, et c'est correct :

Maintenant que cette partie fonctionne comme il se doit, examinons la deuxième approche - obtenir le contenu du fichier et ses métadonnées (le nom du fichier) stockés au format JSON et transférés dans le corps d'une méthode POST.

Vous pouvez en savoir plus sur la création de services REST dans la documentation InterSystems.

Le code exemplaire pour les deux approches se trouve sur GitHub et InterSystems Open Exchange. Si vous avez des questions ou des suggestions concernant l'une ou l'autre, n'hésitez pas à les écrire dans la section des commentaires.

#REST API #InterSystems IRIS
Voir l'application sur InterSystems Open Exchange

URL de la source:https://fr.community.intersystems.com/post/transmission-de-fichiers-rest-pour-les-stocker-en-property-partie-1