Artículo
Kurro Lopez · 10 feb, 2020 Lectura de 8 min

RESTForms - REST API para tus clases

En este artículo me gustaría presentar el proyecto RESTForms: back-end genérico REST API para aplicaciones web modernas.

La idea detrás del proyecto es simple: después de escribir varias API REST, me di cuenta de que, en general, la API REST consta de dos partes:

- Trabajar con clases persistentes.
- Lógica de negocio personalizada

Y, si bien tendrá que escribir su propia lógica de negocio personalizada, RESTForms proporciona todo lo relacionado con el trabajo con clases persistentes desde el primer momento.

Casos de uso

- Ya tiene un modelo de datos en Caché y desea exponer parte (o toda) de la información en forma de API REST
- Está desarrollando una nueva aplicación Caché y desea proporcionar una API REST

Lado cliente

Este proyecto se desarrolla como un back-end de aplicaciones web, por lo que JS lo consigue. No se requiere conversión de formato.

Nota: CRUD

Se pueden realizar 4 operaciones sobre un objeto o una colección:

- **C**reate
- **R**ead
- **U**pdate
- **D**elete

Caracteristicas

Qué puedes hacer con RESTForms:

- CRUD sobre clase expuesta: puede obtener metadatos de clase, crear, actualizar y eliminar propiedades de clase
- CRUD sobre objeto: puede obtener, crear, actualizar y eliminar objetos
- R(ead) sobre colecciones de objetos (a través de SQL): protegido de inyecciones de SQL
- Autodescubrimiento: primero obtienes una lista de clases disponibles, luego puedes obtener metadatos de clase y, en función de esos metadatos, puedes hacer CRUD sobre el objeto

Rutas

Aquí está la tabla de las rutas principales, que muestra lo que puede hacer a través de RESTForms.

URL	Descripción
info	Lista de todas las
info/all	Obtenga metadato
info/:class	Clase metadata
field/:class	Agregar propiedad
field/:class	Modificar propieda
field/:class/:propiedad	Borrarpropiedad d
object/:class/:id	Recuperar objeto
object/:class/:id/:propiedad	Recuperar una pro
object/:class	Crear objeto
object/:class/:id	Actualizar objeto d
object/:class	Actualizar objeto d
object/:class/:id	Eliminar objeto
objects/:class/:query	(SQL) Obtener obj
objects/:class/custom/:query	(SQL) Obtener obj

¿Cómo empiezo a usar RESTForms?

1. Importe el proyecto desde GitHub (método recomendado: agregue como submódulo a su propio repositorio, o simplemente descargue una versión)
2. Para cada clase, desea exponer a través de RESTForms
    - Heredar de la clase de adaptador
    - Especifique permisos (por ejemplo, puede exponer algunas clases como de solo lectura)
    - Especificar propiedad utilizada como valor de visualización para un objeto
    - Especifique los nombres para mostrar de las propiedades que desea mostrar

Setup

1. Descargue e importe desde la página de descarga la última versión de 20161.xml (for Caché 2016.1) o 201162.xml (for Caché 2016.2+) en cualquier napespace
2. Crear nueva aplicación web /forms con la clase Dispatch Form.REST.Main
3. Abrir http://localhost:57772/forms/test?Debug en el navegador para validar la instalación (debería salir {"Status": "OK"} y posiblemente solicitar contraseña).
4. Si desea datos de prueba, llame al:

```
do ##class(Form.Util.Init).populateTestForms()
```

Ejemplo

Primero, necesita saber qué clases están disponibles. Para obtener esa información, llame al:

```
http://localhost:57772/forms/form/info
```

Recibirá algo como esto como respuesta:

```
[
   { "name":"Company",     "class":"Form.Test.Company" },
   { "name":"Person",      "class":"Form.Test.Person"  },
   { "name":"Simple form", "class":"Form.Test.Simple"  }
]
```

Actualmente hay 3 clases de muestra (provistas con RESTForms), veamos los metadatos para Persona (clase Form.Test.Person). Para obtener esa información, llame al:

```
http://localhost:57772/forms/form/info/Form.Test.Person
```

En respuesta, recibirá metadatos de clase:

```
{
   "name":"Person",
   "class":"Form.Test.Person",
   "displayProperty":"name",
   "objpermissions":"CRUD",
   "fields":[
      { "name":"name",     "type":"%Library.String",    "collection":"", "displayName
":"Name",          "required":0, "category":"datatype" },
      { "name":"dob",      "type":"%Library.Date",      "collection":"", "displayName
":"Date of Birth", "required":0, "category":"datatype" },
      { "name":"ts",       "type":"%Library.TimeStamp", "collection":"", "displayName
":"Timestamp",     "required":0, "category":"datatype" },
      { "name":"num",      "type":"%Library.Numeric",   "collection":"", "displayName
":"Number",        "required":0, "category":"datatype" },
      { "name":"?ge",      "type":"%Library.Integer",   "collection":"", "displayName
":"Age",           "required":0, "category":"datatype" },
      { "name":"relative", "type":"Form.Test.Person",   "collection":"", "displayName
":"Relative",      "required":0, "category":"form"     },
      { "name":"Home",     "type":"Form.Test.Address",  "collection":"", "displayName
":"House",         "required":0, "category":"serial"   },
      { "name":"company",  "type":"Form.Test.Company",  "collection":"", "displayName
":"Company",       "required":0, "category":"form"     }
   ]
}
```

¿Qué significa todo eso?

Metadatos de clase:

- name - nombre para mostrar para la clase
- class - clase persistente subyacente
- displayProperty - propiedad del objeto a usar, cuando se muestra el objeto
- objpermissions - ¿Qué puede hacer un usuario con un objeto? En el caso actual, el usuario puede crear

nuevos objetos, modificar los existentes, eliminar objetos existentes y obtener el

Metadatos de propiedad:

- name - nombre de propiedad: igual que en la definición de clase
- type - clase de propiedad
- collection - es la colección de lista / matriz
- displayName - mostrar nombre de propiedad
- required - se requiere esta propiedad
- category - tipo de propiedad clase categoría. Sigue las categorías de clase Caché habituales, excepto que todas las clases habilitadas para RESTForms se muestran como "form"

En la definición de clase se ve así:

```
/// Test form: Person
Class Form.Test.Person Extends (%Persistent, Form.Adaptor, %Populate)
{

/// Form name, not a global key so it can be anything
/// Set to empty string (like here) to not have a class as a form
Parameter FORMNAME = "Person";

/// Default permissions
/// Objects of this form can be Created, Read, Updated and Deleted
/// Redefine this parameter to change permissions for everyone
/// Redefine checkPermission method (see Form.Security) for this class
/// to add custom security based on user/roles/etc.
Parameter OBJPERMISSIONS As %String = "CRUD";

/// Property used for basic information about the object
/// By default getObjectDisplayName method gets its value from it
Parameter DISPLAYPROPERTY As %String = "name";

/// Use value of this parameter in SQL, as ORDER BY clause value
Parameter FORMORDERBY As %String = "dob";

/// Person's name.
Property name As %String(COLLATION = "TRUNCATE(250)", DISPLAYNAME = "Name", MAXLEN =
2000);

/// Person's Date of Birth.
Property dob As %Date(DISPLAYNAME = "Date of Birth", POPSPEC = "Date()");

Property ts As %TimeStamp(DISPLAYNAME = "Timestamp") [ InitialExpression = {$ZDATETIM
E($ZTIMESTAMP, 3, 1, 3)} ];

Property num As %Numeric(DISPLAYNAME = "Number") [ InitialExpression = "2.15" ];

/// Person's age.<br>
/// This is a calculated field whose value is derived from <property>DOB</property>.
Property ?ge As %Integer(DISPLAYNAME = "Age") [ Calculated, SqlComputeCode = { set {*
}=##class(Form.Test.Person).currentAge({dob})}, SqlComputed, SqlComputeOnChange = dob
 ];

/// This class method calculates a current age given a date of birth <var>date</var>.
ClassMethod currentAge(date As %Date = "") As %Integer [ CodeMode = expression ]
{
```

```
$Select(date="":"",1:($ZD($H,8)-$ZD(date,8)\10000))
}

/// Person's spouse.
/// This is a reference to another persistent object.
Property relative As Form.Test.Person(DISPLAYNAME = "Relative");

/// Person's home address. This uses an embedded object.
Property Home As Form.Test.Address(DISPLAYNAME = "House");

/// The company this person works for.
Relationship company As Form.Test.Company(DISPLAYNAME = "Company") [ Cardinality = on
e, Inverse = employees ];
}
```

RESTForms que habilitan una clase

Entonces, para habilitar esta clase RESTForms habilitada, comencé con la clase persistente habitual y:

1. Extender desde Form.Adaptor
2. Añadir parámetro FORMNAME - con el valor - nombre de la clase
3. Añadir parámetro OBJPERMISSIONS - CRUD para todos los permisos
4. Añadir parámetro DISPLAYPROPERTY - nombre de propiedad utilizado para mostrar el nombre del objeto
5. Añadir parámetro FORMORDERBY - propiedad predeterminada para ordenar por consultas utilizando RESTForms
6. Para cada propiedad que quiero ver en los metadatos, Añadir el parámetro de propiedadDISPLAYNAME

Eso es todo. Después de la compilación, puede usar la clase con RESTForms.

A medida que generamos algunos datos de prueba (ver Instalación, paso 4), obtengamos Persona con id 1. Para obtener la llamada al objeto:

```
http://localhost:57772/forms/form/object/Form.Test.Person/1
```

Y aquí está la respuesta (los datos generados pueden diferir):

```
{
   "_class":"Form.Test.Person",
   "_id":1,
   "name":"Klingman,Rhonda H.",
   "dob":"1996-10-18",
   "ts":"2016-09-20T10:51:31.375Z",
   "num":2.15,
   "?ge":20,
   "relative":null,
   "Home":{
      "_class":"Form.Test.Address",
      "House":430,
      "Street":"5337 Second Place",
      "City":"Jackson"
   },
   "company":{
      "_class":"Form.Test.Company",
      "_id":60,
      "name":"XenaSys.com",
      "employees":[
```

```
         null
      ]
   }
}
```

Para modificar el objeto (específicamente, propiedad num), llame a:

```
PUT http://localhost:57772/forms/form/object/Form.Test.Person
```

Con este cuerpo:

```
{
   "_class":"Form.Test.Person",
   "_id":1,
   "num":3.15
}
```

Tenga en cuenta que para una mejor velocidad, solo class, id y las propiedades modificadas deben estar en el cuerpo de la solicitud.

Ahora, creemos un nuevo objeto. Llamada:

```
POST http://localhost:57772/forms/form/object/Form.Test.Person
```

Con este cuerpo:

```
{
   "_class":"Form.Test.Person",
    "name":"Test person",
    "dob":"2000-01-18",
    "ts":"2016-09-20T10:51:31.375Z",
    "num":2.15,
    "company":{ "_class":"Form.Test.Company", "_id":1 }
}
```

Si la creación del objeto fue exitosa, RESTForms devolvería una identificación:

```
{"Id": "101"}
```

De lo contrario, se devolverá un error en formato JSON. Tenga en cuenta que todas las propiedades de los objetos persistentes deben ser referenciadas solo por las propiedades class y id.

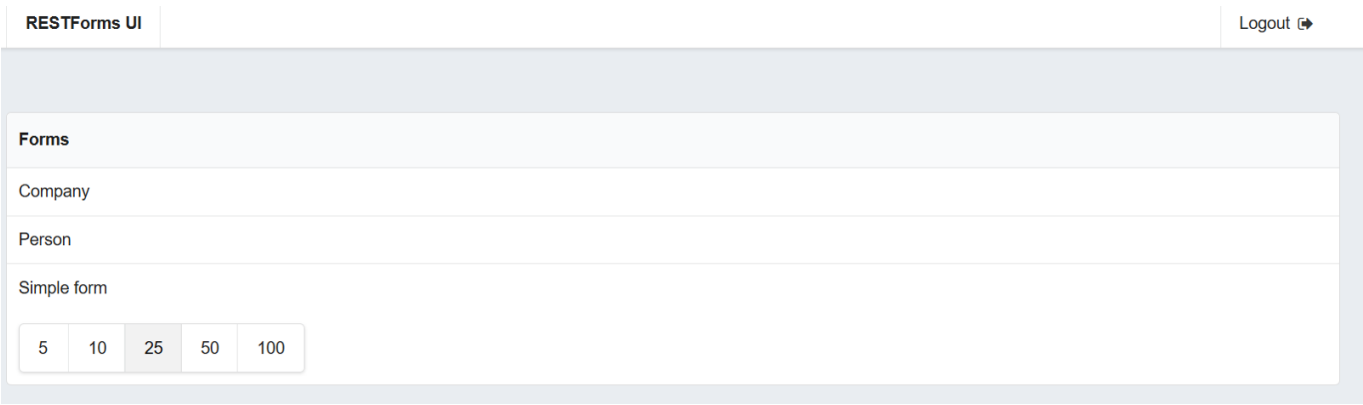
Y finalmente, eliminemos nuestro nuevo objeto. Llamada:

```
DELETE http://localhost:57772/forms/form/object/Form.Test.Person/101
```

Eso es el CRUD completo Form.Test.Person class.

Demo

Captura de pantalla de la lista de clases:



Conclusión

RESTForms realiza ~~toda~~ la mayor parte del trabajo, requerido de la API REST en lo que respecta a las clases persistentes.

Que sigue

En este artículo, acabo de empezar a hablar sobre las características de RESTForms. En el próximo artículo, me gustaría contarle algunas características avanzadas: consultas, que permiten al cliente obtener segmentos de datos de forma segura, sin riesgo de inyecciones SQL.

También hay un RESTFormsUI - editor para datos RESTForms.

Enlaces

- RESTForms GitHub repository
- RESTForms UI GitHub repository

#API REST #Frontend #Herramientas #Caché

URL de fuente:https://es.community.intersystems.com/post/restforms-rest-api-para-tus-clases