記事
Shintaro Kaminaka · 2020年7月30日 11m read

RESTForms - 永続クラスにREST APIをアドオンする

この記事では、RESTFormsプロジェクト（モダンなWebアプリケーション用の汎用REST APIバックエンド）を紹介します。

プロジェクトの背後にあるアイデアは単純です。私はいくつかのREST APIを書いた後、REST APIが一般的に次の2つの部分で構成されていることに気付きました。

- 永続クラスの操作
- カスタムビジネスロジック

また、独自のカスタムビジネスロジックを書く必要はありますが、RESTFormsには永続クラスの操作に関連するすべての機能を提供しています。

使用例

- Cachéにすでにデータモデルがあり、REST API形式で情報の一部（またはすべて）を公開したい
- 新しいCachéアプリケーションを開発しており、REST APIを提供したい

クライアントサイド

このプロジェクトはWebアプリケーションのバックエンドとして開発されているため、JSだけで事足ります。形式の変換は必要ありません。

補足：CRUD

オブジェクトまたはコレクションに対し、次の4つの操作を実行できます。

- Create(作成)
- Read(読み込み)
- Update(更新)
- Delete(削除)

機能

RESTFormsを使用して以下を実行できます。

- 公開されたクラスに対するCRUD - クラスのメタデータを取得し、クラスのプロパティを作成 / 更新 / 削除できます。
- オブジェクトに対するCRUD - オブジェクトを取得 / 作成 / 更新 / 削除できます。
- オブジェクトコレクションに対するRead（SQL経由） - SQLインジェクションから保護します。
- 自己検出 – 最初に使用可能なクラスのリストを取得し、その後でクラスのメタデータを取得し、そのメタデータを基にしてオブジェクトに対するCRUDを実行できます。

パス

以下の表には、主なパスとRESTFormsを使用して実行できる操作を掲載しています。

URL	説明
info	利用可能なすべてのクラスを一覧表示します
info/all	すべてのクラスのメタデータを取得します
info/:class	クラスのメタデータ
field/:class	プロパティをクラスに追加します
field/:class	クラスのプロパティを変更します
field/:class/:property	クラスのプロパティを削除します
object/:class/:id	オブジェクトを取得します
object/:class/:id/:property	オブジェクトの1つのプロパティを取得します
object/:class	オブジェクトを作成します
object/:class/:id	動的オブジェクトからオブジェクトを更新します
object/:class	オブジェクトからオブジェクトを更新します
object/:class/:id	オブジェクトを削除します
objects/:class/:query	（SQL）クエリでクラスのオブジェクトを取得します
objects/:class/custom/:query	（SQL）カスタムクエリでクラスのオブジェクトを取得します

RESTFormsを使い始めるには？

1. GitHubからプロジェクトをインポートします（お勧めの方法は独自リポジトリにサブモジュールとして追加する方法ですが、単にリリースをダウンロードしても良いです）。
2. RESTFormsを介して公開したい各クラスについて以下を実施します。

- アダプタクラスから継承する
- 権限を指定します（一部のクラスを読み取り専用として公開する場合などに実施）。
- オブジェクトの表示値として使用されるプロパティを指定します。
- 表示したいプロパティの表示名を指定します。

セットアップ

1. [リリースページ]（https://github.com/intersystems-ru/RESTForms/releases/tag/v1.0 ）で最新リリースである20161.xml（ Caché 2016.1用）または201162.xml（Caché 2016.2以降用）をダウンロードして任意のネームスペースにインポートします。
2. 新しいWebアプリケーション /forms をDispatchクラス Form.REST.Main を使用して作成します。
3. http://localhost:57772/forms/test?Debug をブラウザで開き、インストールを検証します（{"Status": "OK"} が出力され、場合によってはパスワードの入力が求められます）。
4. テストデータが必要な場合は、次を呼び出します：

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

例

最初に、利用可能なクラスを知る必要があります。 この情報を取得するには、次を呼び出します。

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

次のような応答が返されます。

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

```
]
```

現在3つのサンプルクラス（RESTFormで提供）があります。Person（Form.Test.Personクラス）のメタデータを見てみましょう。 この情報を取得するには、次を呼び出します。

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

次のように、クラスのメタデータが応答として返されます。

```
{
   "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"     }
   ]
}
```

これらの情報は次のような意味を持ちます。

クラスのメタデータ：

- name - クラスの表示名。
- class - 基本となる永続クラス。
- displayProperty - オブジェクトを表示するときに使用するオブジェクトのプロパティ。
- objpermissions - ユーザーがオブジェクトを使用して実行できる操作。 この例では、ユーザーは新しいオブジェクトを作成し、既存のオブジェクトを変更し、既存のオブジェクトを削除し、次を取得できます。

プロパティのメタデータ：

- name - プロパティ名 - クラスの定義と同じです。

- type - プロパティのクラス。
- コレクション - リスト/配列のコレクションです。
- displayName - 表示プロパティ名。
- required - このプロパティが必須であるかどうか。
- category - プロパティのタイプクラスのカテゴリ。 RESTForms対応のすべてのクラスが「form」として表示されることを除き、通常のCachéクラスのカテゴリに従います。

クラス定義では次のようになります。

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

/// ??????????????????????????????
/// ????????????????????????????????????????????
Parameter FORMNAME = "Person";

/// ????????
/// ????????????????????????????????????
/// ????????????????????????????????????
/// ??????checkPermission???????????Form.Security??????????
/// ????????????????????????????????
Parameter OBJPERMISSIONS As %String = "CRUD";

/// ?????????????????????????
/// ????????getObjectDisplayName???????????????????
Parameter DISPLAYPROPERTY As %String = "name";

/// ???????????SQL?ORDER BY????????????
Parameter FORMORDERBY As %String = "dob";

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

/// Person??????
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????<br>
/// ???? <property>DOB</property> ?????????????????????????
Property ?ge As %Integer(DISPLAYNAME = "Age") [ Calculated, SqlComputeCode = { set {*
}=##class(Form.Test.Person).currentAge({dob})}, SqlComputed, SqlComputeOnChange = dob
 ];

/// ????????????? <var>date</var> ?????????????????????
ClassMethod currentAge(date As %Date = "") As %Integer [ CodeMode = expression ]
{
$Select(date="":"",1:($ZD($H,8)-$ZD(date,8)\10000))
}

/// Person?????
/// ????????????????????
Property relative As Form.Test.Person(DISPLAYNAME = "Relative");

/// Person?????? ????????????????????
Property Home As Form.Test.Address(DISPLAYNAME = "House");

/// ??Person?????????
Relationship company As Form.Test.Company(DISPLAYNAME = "Company") [ Cardinality = on
e, Inverse = employees ];
```

```
}
```

クラスでRESTFormsを有効にする

そして、このクラスでRESTFormsを有効にするため、通常の永続クラスから始めて次のことを行いました。

1. Form.Adaptor から拡張しました。
2. 値を含むパラメーター FORMNAME（クラス名）を追加しました。
3. OBJPERMISSIONS パラメーター（すべての権限のCRUD）を追加しました。
4. DISPLAYPROPERTY
   パラメーター（オブジェクト名の表示に使用されるプロパティ名）を追加しました。
5. FORMORDERBY
   パラメーター（RESTFormsを使用するクエリでソートするデフォルトのプロパティ）を追加しました。
6. メタデータで確認したいプロパティごとに DISPLAYNAME プロパティのパラメーターを追加しました。

以上です。 コンパイル後、RESTFormsを含むクラスを使用できるようになります。

いくつかのテストデータを生成しましたので（インストールのステップ4を参照）、IDが1のPersonを取得してみましょう。 オブジェクトを取得するには、次を呼び出します。

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

その応答は以下のとおりです（生成されるデータは異なる場合があります）。

```
{
   "_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
      ]
   }
}
```

オブジェクト（具体的にはnumプロパティ）を変更するには、次を呼び出します。

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

このボディを使用します。

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

速度を上げるには、class、id、および変更対象のプロパティのみをリクエストのボディに含める必要があります。

では、新しいオブジェクトを作成しましょう。 以下を呼び出します。

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

このボディを使用します。

```
{
   "_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 }
}
```

オブジェクトの作成が成功した場合、RESTFormsは以下のようにIDを返します。

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

成功しなかった場合、エラーがJSON形式で返されます。 すべての永続オブジェクトのプロパティは、 class および id プロパティによってのみ参照する必要があります。

そして最後に、新しいオブジェクトを削除しましょう。 以下を呼び出します。

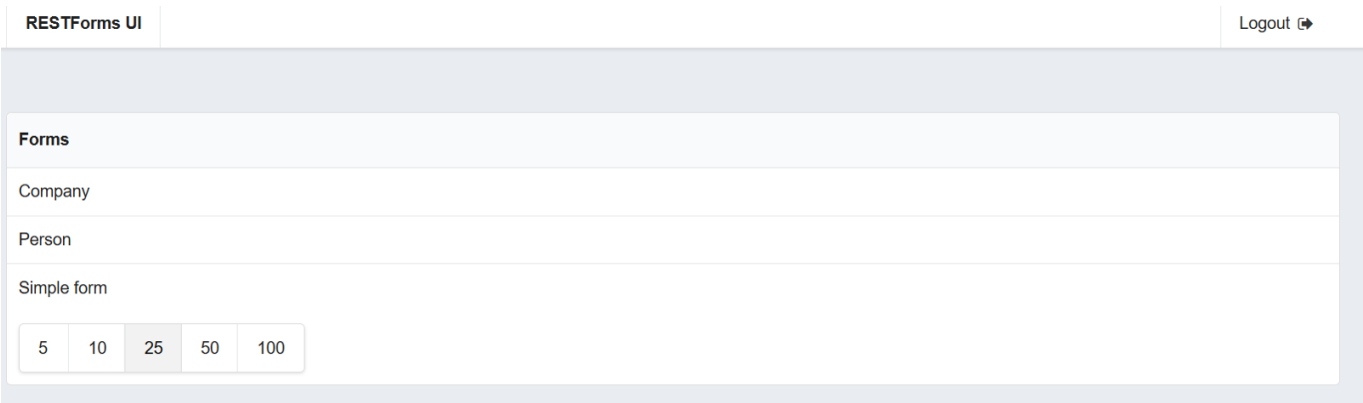
```
DELETE http://localhost:57772/forms/form/object/Form.Test.Person/101
```

これがForm.Test.Personクラスに対する完全なCRUDです。

デモ

[現在デモ環境はお試しいただくことができません。]
こちらでRESTFormsをオンラインで試すことができます（ユーザー名：Demo、パスワード：Demo）。

また、RESTFormsUIアプリ
ケーション（RESTFormsデータエディタ）もあります。こちら
をご確認ください（ユーザー名：Demo、パスワード：Demo）。
クラスリストのスクリーンショットを以下に掲載しています。



まとめ

RESTFormsは永続クラスに関する、REST APIから要求されるほとんどの機能を提供します。

次の内容

この記事では、RESTFormsの機能について説明しました。 次回の記事では、いくつかの高度な機能（クライアントからSQLインジェクションのリスクを冒さずにデータの一部を安全に取得できるクエリなど）についてお話ししたいと思います。 この記事のパート2でクエリに関する情報をお読みください。

RESTFormsUI（RESTFormsデータエディタ）もあります。

リンク

- RESTForms GitHubリポジトリ
- RESTForms UI GitHubリポジトリ
- パート2：クエリ

#REST API #ツール #フロントエンド #Caché

ソースURL:
https://jp.community.intersystems.com/post/restforms-%E6%B0%B8%E7%B6%9A%E3%82%AF%E3%83%A9%E3%82%B9%E3%81%ABrest-api%E3%82%92%E3%82%A2%E3%83%89%E3%82%AA%E3%83%B3%E3%81%99%E3%82%8B