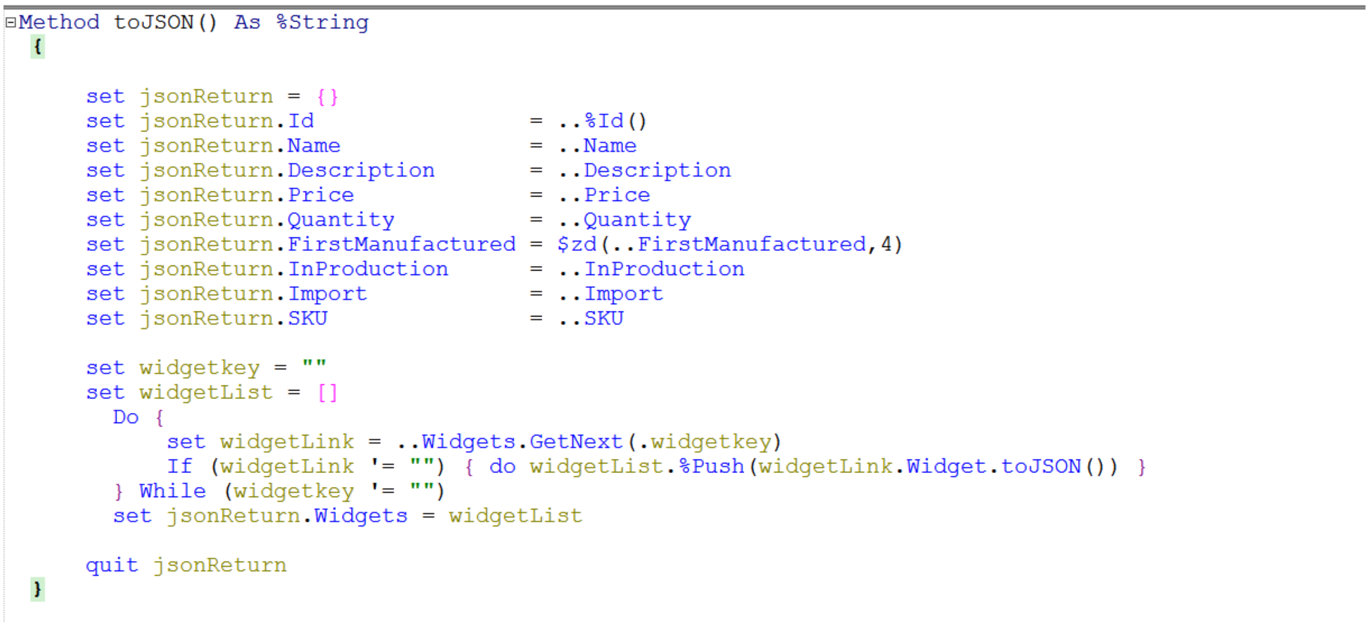
Article
Chris Stewart · Apr 24, 2017 2m read

Let's write an Angular 1.x app with a Caché REST backend - Part 8

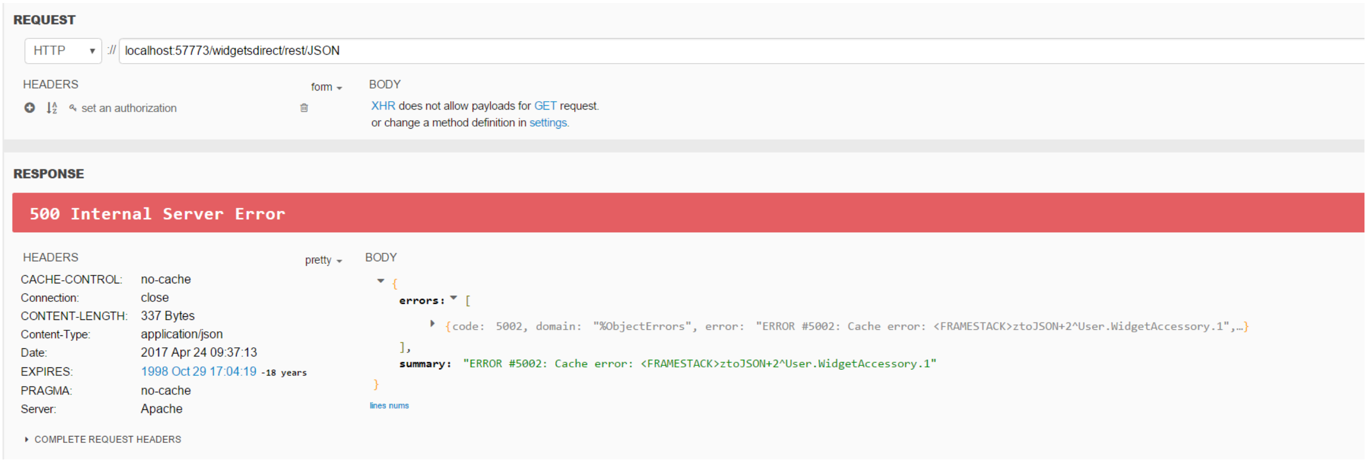
or *"Bonus Breakage"*

In our last lesson, we added a relationship between 2 persistent classes. We are clearly going to need to start creating REST Services to expose CRUD operations for each of these classes, but before we do that, we should really finish defining our linkages. We added code to our Widget toJSON to spool off related Accessory data, so we should really do the reciprocal and allow Accessories to return all Widgets that are compatible.

The code we can use is essentially the same as we used on the Widget side. We will iterate over the bridging class and return the toJSON of all referenced widgets. We are definitely sure we have a toJSON for the Widget class, so we shouldn't be expecting any errors. We add the code to WidgetAccessory and compile, before loading our service in the REST debugger to make sure everything is spooling OK.



```
Method toJSON() As %String
{

    set jsonReturn = {}
    set jsonReturn.Id                  = ..%Id()
    set jsonReturn.Name                = ..Name
    set jsonReturn.Description         = ..Description
    set jsonReturn.Price               = ..Price
    set jsonReturn.Quantity            = ..Quantity
    set jsonReturn.FirstManufactured   = $zd(..FirstManufactured,4)
    set jsonReturn.InProduction        = ..InProduction
    set jsonReturn.Import              = ..Import
    set jsonReturn.SKU                 = ..SKU

    set widgetkey = ""
    set widgetList = []
     Do {
         set widgetLink = ..Widgets.GetNext(.widgetkey)
         If (widgetLink '= "") { do widgetList.%Push(widgetLink.Widget.toJSON()) }
     } While (widgetkey '= "")
     set jsonReturn.Widgets = widgetList

    quit jsonReturn
}
```



REQUEST

HTTP :// localhost:57773/widgetsdirect/rest/JSON

HEADERS form BODY

set an authorization

XHR does not allow payloads for GET request.
or change a method definition in settings.

RESPONSE

500 Internal Server Error

HEADERS pretty BODY

```
CACHE-CONTROL:   no-cache
Connection:      close
CONTENT-LENGTH:  337 Bytes
Content-Type:    application/json
Date:            2017 Apr 24 09:37:13
EXPIRES:         1998 Oct 29 17:04:19 -18 years
PRAGMA:          no-cache
Server:          Apache
```

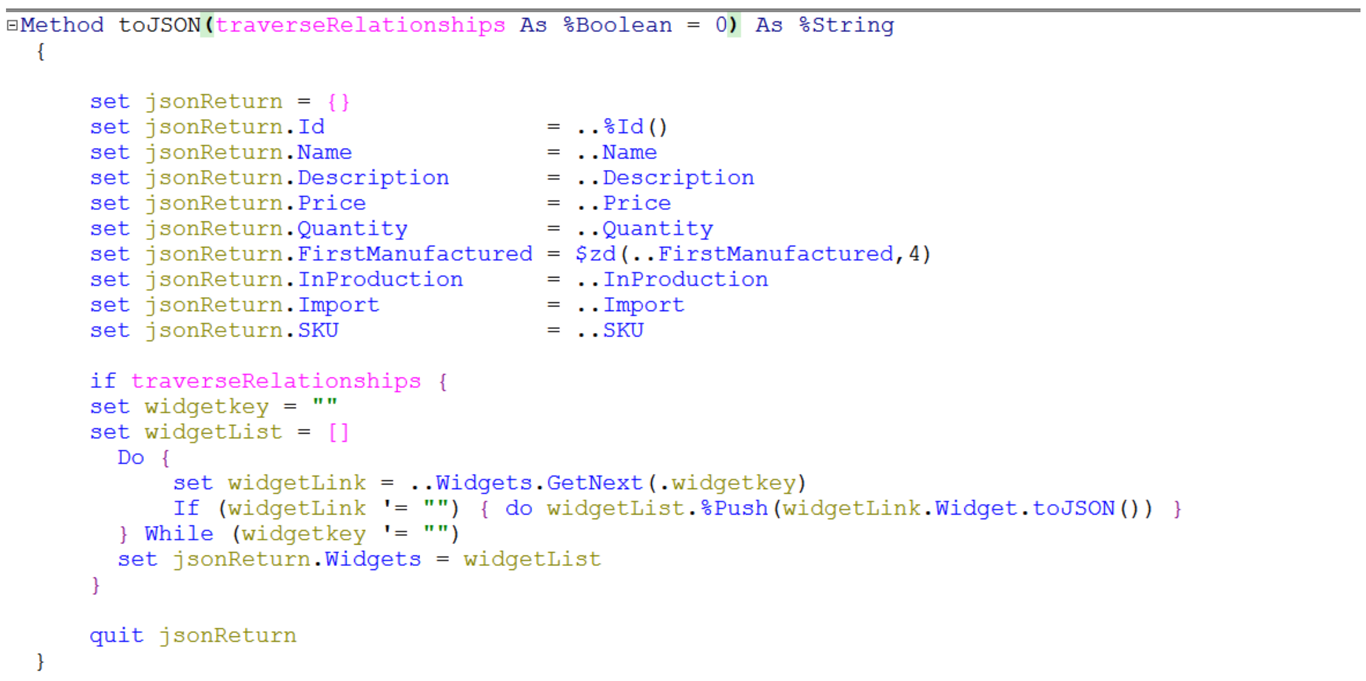
```
{
  errors: [
     {code: 5002, domain: "%ObjectErrors", error: "ERROR #5002: Cache error: <FRAMESTACK>ztoJSON+2^User.WidgetAccessory.1",…}
  ],
  summary: "ERROR #5002: Cache error: <FRAMESTACK>ztoJSON+2^User.WidgetAccessory.1"
}
```

lines nums

COMPLETE REQUEST HEADERS

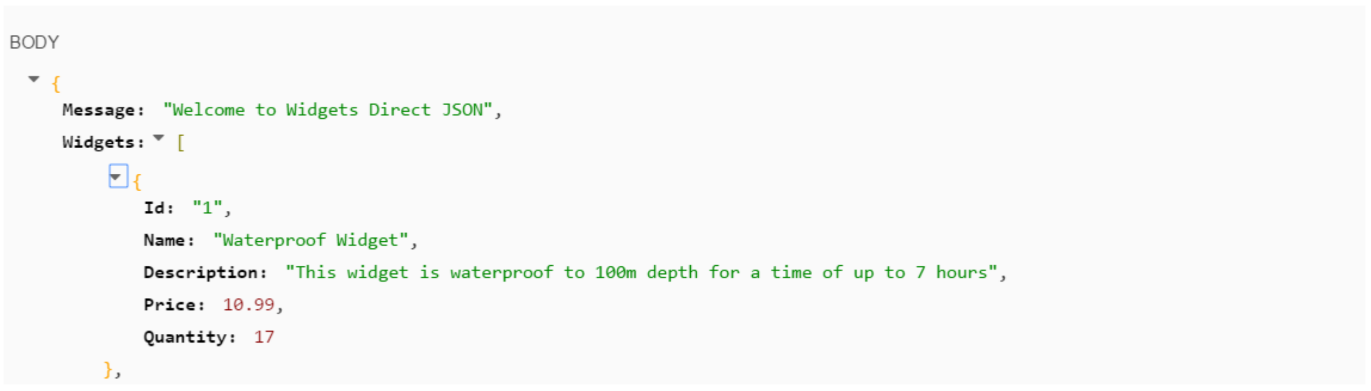
FRAMESTACK?! This seems bad. Why would adding code we have already used in the Widget class cause such a severe failure? The issue is that we have accidently created an infinte recursive loop. Our Widget references an

Accessory which references Widgets which reference Accessories.... and so on. We need to ensure that we limit our recursion when dealing with relationships. In most cases, we don't need deep recursion when spooling off referenced data, so the most simple solution is to parameterise toJSON with a "traverseRelationships" Boolean, and default this to False. This allows our REST Services to return basic information about a referenced class, and for our primary Services for an object to return all data when it needs to.



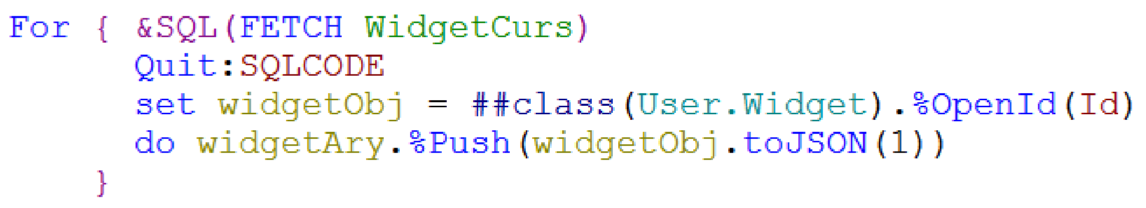
```
⊟Method toJSON(traverseRelationships As %Boolean = 0) As %String
  {

      set jsonReturn = {}
      set jsonReturn.Id                 = ..%Id()
      set jsonReturn.Name               = ..Name
      set jsonReturn.Description        = ..Description
      set jsonReturn.Price              = ..Price
      set jsonReturn.Quantity           = ..Quantity
      set jsonReturn.FirstManufactured = $zd(..FirstManufactured,4)
      set jsonReturn.InProduction       = ..InProduction
      set jsonReturn.Import             = ..Import
      set jsonReturn.SKU                = ..SKU

      if traverseRelationships {
      set widgetkey = ""
      set widgetList = []
        Do {
            set widgetLink = ..Widgets.GetNext(.widgetkey)
            If (widgetLink '= "") { do widgetList.%Push(widgetLink.Widget.toJSON()) }
        } While (widgetkey '= "")
        set jsonReturn.Widgets = widgetList
      }

      quit jsonReturn
  }
```

So, we add the Boolean parameter and then enclose the relationship code in an IF statement to prevent it from running. This should output the standard properties only when called from our service, since the default value is to prevent the output of relationships. We will implement this on both our Widget and Accessory class. After a compile, we load our Service again



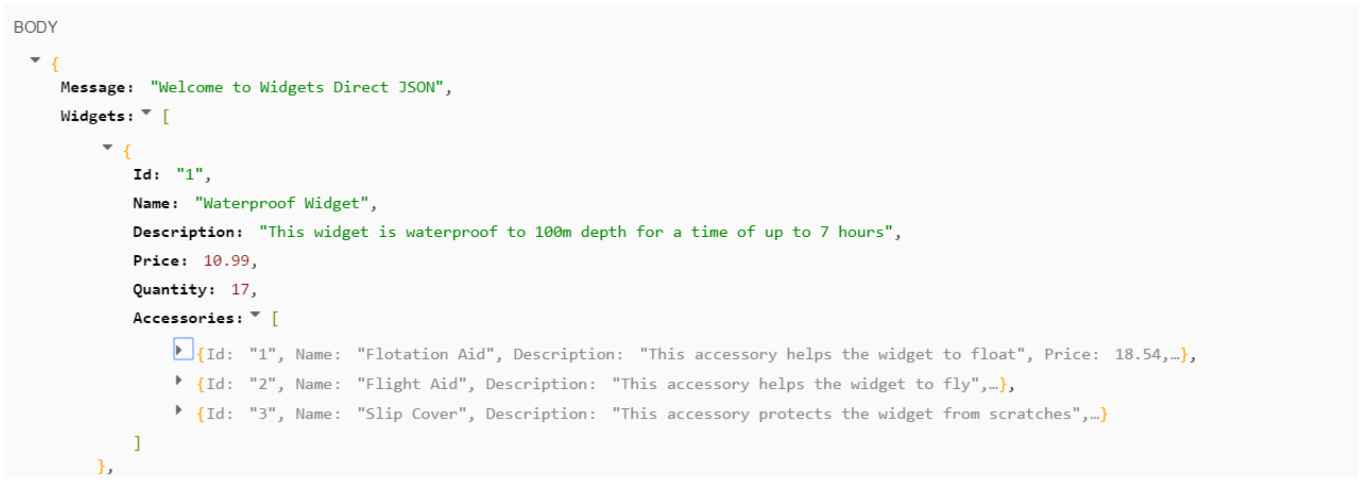
```
BODY

▾ {
    Message: "Welcome to Widgets Direct JSON",
    Widgets: ▾ [
        ▾{
            Id: "1",
            Name: "Waterproof Widget",
            Description: "This widget is waterproof to 100m depth for a time of up to 7 hours",
            Price: 10.99,
            Quantity: 17
        },
```

We've lost our accessory relationship. As the default behaviour is to inhibit relationships now, we have to amend our REST Service code to specifically include the references, by passing in a true value to the toJSON's first parameter.



```
For { &SQL(FETCH WidgetCurs)
      Quit:SQLCODE
      set widgetObj = ##class(User.Widget).%OpenId(Id)
      do widgetAry.%Push(widgetObj.toJSON(1))
    }
```

One more compile and refresh, and we have our references back without a FRAMESTACK error



```
BODY
▼ {
    Message: "Welcome to Widgets Direct JSON",
    Widgets: ▼ [
        ▼ {
            Id: "1",
            Name: "Waterproof Widget",
            Description: "This widget is waterproof to 100m depth for a time of up to 7 hours",
            Price: 10.99,
            Quantity: 17,
            Accessories: ▼ [
                ▶ {Id: "1", Name: "Flotation Aid", Description: "This accessory helps the widget to float", Price: 18.54,…},
                ▶ {Id: "2", Name: "Flight Aid", Description: "This accessory helps the widget to fly",…},
                ▶ {Id: "3", Name: "Slip Cover", Description: "This accessory protects the widget from scratches",…}
            ]
        },
```

In this lesson we:

1. Created an infinite recursion between 2 related classes
2. Implemented default behaviour to prevent infinite recursion
3. Amended our REST Service code to correct output relationships from a base class

Next lesson we will:

- Create direct REST services for Widget and Accessory
- Implement a fromJSON to read objects sent from a client

This article is part of a multi-part series on using Angular on top of Caché REST services. The listing of the full series can be found at the Start Here page

#Angular #CSP #HTML #JavaScript #REST API #Frontend #Caché

Source URL:https://community.intersystems.com/post/lets-write-angular-1x-app-cach%C3%A9-rest-backend-part-8