Article
Chris Stewart · Apr 19, 2017 3m read

Let's write an Angular 1.x app with a Caché REST backend - Part 3

We finished our last lesson with our Widgets Direct page receiving a Welcome message as a property of a JSON object, which was unpacked and displayed on the page. However, we are on Lesson 3, and we still haven't had any talk of displaying widgets yet.

Let's change that. Widgets Direct are a leader in widget and widget accessories, so we clearly have numerous items to display. To transfer this, we will need to declare and return our first array. Let's go back to our REST.Dispatch class, and add this data to our Service.

Adding an array is simple, but we need to set up some temporary storage before we can assign the array to our JSON return object. First, we need to declare a JSON array (using the [] shortcut). Then, in a loop, we will create a JSON object, assign an ID and Name property to it, and then "push" this onto the Array, using our %Push JSON method. Once we have looped over all of our widgets and placed them on the array, we can set the array as a property of our return object. We don't need to worry about the JSON representation of the array, as our final %TOJSON() call will take care of that for us.

```
ClassMethod HelloWorld(Name As %String) As %Status
 {
      Set %response.ContentType="application/json"

      SET retObj = {}
      SET retMessage = "Welcome to Widgets Direct "_Name
      SET retObj.Message = retMessage
      //
      SET widgetAry = []
      FOR i=1:1:10 {
          SET widgetObj = {}
          SET widgetObj.Id = i
          SET widgetObj.Name = "Widget "_i
          DO widgetAry.%Push(widgetObj)
      }
      SET retObj.Widgets = widgetAry
      //
      WRITE retObj.%ToJSON()
      QUIT $$$OK
 }

}
```

Let's compile our class, and run a fresh GET to check our array is output correctly

As we expected, we now have an array property bound to Widgets on our response. We should get this data bound to our page! We need to attach the property to the $scope, so that Angular can access the data, and add the bound property to our page.

```
$http.get('/widgetsdirect/rest/Chris').then(
  function(response) { //success
    $scope.message = response.data.Message;
    $scope.widgets = response.data.Widgets;
  }
```

```
<body ng-controller="PageController">

        <!-- Put your page code here -->
        {{message}}

        <hr>
        {{widgets}}
        <hr>
```

Let's reload our page, and see our Widgets in all their glory

Welcome to Widgets Direct Chris

[{"Id":1,"Name":"Widget 1"},{"Id":2,"Name":"Widget 2"},{"Id":3,"Name":"Widget 3"},{"Id":4,"Name":"Widget 4"},{"Id":5,"Name":"Widget 5"},
{"Id":6,"Name":"Widget 6"},{"Id":7,"Name":"Widget 7"},{"Id":8,"Name":"Widget 8"},{"Id":9,"Name":"Widget 9"},{"Id":10,"Name":"Widget 10"}]

Well, this clearly isn't right. We can't work directly with array data on the page. Instead, we need a way to iterate over it and display the data in a consistent form. Luckily Angular provides the very helpful ng-repeat, which allows us to concisely declare an iterator object based on an array collection, providing direct access to the object properties without us having to worry about counting logic, or having to write a loop construct. All we need to do is declare the ng-repeat condition on the tags we wish to use as the parent point of our repeating data, then refer to our current array element using the alias we give it in the ng-repeat. So let's have each widget be its own row in a table

```
<hr>
<table>
    <tr ng-repeat="widget in widgets">
        <td>{{widget.Id}}</td>
        <td>{{widget.Name}}</td>
    </tr>
</table>
<hr>
```

If we reload our page now, we get a nicely formatted tabular view of our data, output in the order that the service provides.

Welcome to Widgets Direct Chris

1 Widget 1
2 Widget 2
3 Widget 3
4 Widget 4
5 Widget 5
6 Widget 6
7 Widget 7
8 Widget 8
9 Widget 9
10 Widget 10

So, after a very small amount of coding, we now have a JSON array in our service, and are able to output it in a structured manner. However, the page is looking a little bit basic, so we probably want to start working on styling our output.

Recap

In this lesson we:

1. added a JSON array to our Service
2. bound a second property to our $scope
3. used an ng-repeat to output our array data in a templated fashion

In our next lesson we will:

- Discuss display libraries
- Add some responsive UI components to our page

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 #JSON #REST API #Frontend #Caché
