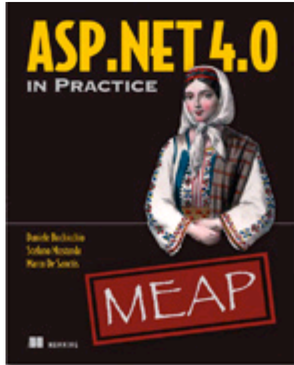
MANNING PUBLICATIONS

Invoking REST web services with jQuery

An article based on



ASP.NET 4.0 in Practice

EARLY ACCESS EDITION

Daniele Bochicchio, Stefano Mostarda, and Marco De Sanctis

MEAP Began: February 2010
Softbound print: March 2011 (est.) | 425 pages
ISBN: 9781935182467

This article is taken from the book ASP.NET 4.0 in Practice*. The authors discuss enabling AJAX in ASP.NET applications using jQuery.*

Tweet this button! (instructions here)

Get **35% off** any version of *ASP.NET 4.0 in Practice* with the checkout code **fcc35**. Offer is only valid through www.manning.com.

jQuery lets you invoke the server in different ways. It has a low-level method named `ajax` that you can use to specify all call parameters and then a set of specific high-level methods that are built upon it. For instance, you have a method to perform POST, another one for GET, and other ones for retrieving JSON data or a JavasScript file. In the end, you have a very broad choice but `ajax` method still remains the best way to go.

Problem

Suppose that you have a page that shows a customer's details. The user might want to know the total amount of the orders placed by the customer. Since the query might be heavy, it is performed only when the user explicitly requests it through the click of a button. We need to intercept the click, call the server to get the total amount of the orders for that customer, and then show it on the page.

Solution

Creating this solution is pretty simple. At first, you have to create a web service on the server that exposes such function via a REST call. To do that, we have to add an item of type *AJAX-enabled WCF Service* to the project and name it `RestService`.

Visual Studio automatically creates the plumbing to expose the web service via REST. More precisely, it inserts all of the needed WCF configurations in the web.config file, as shown in the listing 1.

Listing 1 The web.config code needed to configure the REST service

```
<system.serviceModel>
  <behaviors>
    <endpointBehaviors>                                                    #1
      <behavior name="RestServiceAspNetAjaxBehavior">                      #1
        <enableWebScript />                                                #1
      </behavior>                                                          #1
    </endpointBehaviors>                                                   #1
  </behaviors>
  <serviceHostingEnvironment aspNetCompatibilityEnabled="true"             #2
    multipleSiteBindingsEnabled="true" />                                  #2
  <services>
    <service name="RestService">                                           #3
      <endpoint address=""                                                 #3
        behaviorConfiguration="RestServiceAspNetAjaxBehavior"              #3
        binding="webHttpBinding" contract="RestService" />                 #3
    </service>                                                             #3
  </services>
</system.serviceModel>
```

#1 Exposes service to Javascript
#2 Makes service compatible with ASP.NET
#3 Exposes the service

Once the web.config is ready, we need to create a method that exposes the total orders amount for the client. Such method must be put in the `RestService` class that is in the `RestService.cs|vb` file inside the `App_Code` directory. The code for the entire class is shown in listing 2.

Listing 2 Service class that exposes the total orders amount

C#

```
[ServiceContract]                                                          #1
[AspNetCompatibilityRequirements(RequirementsMode =                        #2
AspNetCompatibilityRequirementsMode.Allowed)]                              #2
public class RestService
{
  [OperationContract]                                                      #3
  public decimal GetOrdersAmount(string CustomerId)
  {
    using (var ctx = new NorthwindEntities())
    {
      return ctx.Orders.Where(o => o.CustomerID == CustomerId).
      Sum(o => o.Order_Details.Sum(d => d.UnitPrice * d.Quantity));
    }
  }
}
```

VB

```
<ServiceContract> _                                                        #1
<AspNetCompatibilityRequirements(_                                         #2
  RequirementsMode := AspNetCompatibilityRequirementsMode.Allowed)> _      #2
Public Class RestService
  <OperationContract> _                                                    #3
  Public Function GetOrdersAmount(CustomerId As String) As Decimal
    Using ctx = New NorthwindEntities()
      Return ctx.Orders.Where(Function(o) _
        o.CustomerID = CustomerId).Sum(Function(o) _
          o.Order_Details.Sum(Function(d) d.UnitPrice * d.Quantity))
    End Using
  End Function
End Class
```

#1 Exposes the class as service
#2 Makes service ASP.NET compatible
#3 Exposes method to consumers

The web service class is pretty simple. You just mark it with the `ServiceContract` **#1** (`System.ServiceModel` namespace) and `AspNetCompatibilityRequirements` **#2** (`System.ServiceModel.Activation` namespace) attributes**.** The methods to be exposed must be marked with the `OperationContract` attribute **#3**

(System.ServiceModel namespace). The method itself is very simple because it just calculates the total amount for the input customer.

Now that the web service is created, we need to write the JavaScript to invoke it. The method to use is ajax. It is a low-level method that lets us specify all parameters of the call. Let's take a look at the code in listing 3 before discussing it.

Listing 3 Invoking the server using jQuery API for Ajax

```
$.ajax({
  url: "RestService.svc/GetOrdersAmount",                               #1
  data: '{ "CustomerId": "ALFKI" }',                                    #2
  type: "POST",                                                         #3
  contentType: "application/json",                                      #4
  dataType: "json",                                                     #5
  success: function (result) {                                          #6
    //code
  }
});
```

#1 The url of the service
#2 The parameters of the method
#3 Invocation method
#4 ContentType format
#5 Server serialization format
#6 Callback to invoked after completion

There's a lot to talk about in the above listing. First of all, the ajax method accepts just one parameter that is a class containing all the real parameters. The first parameter of the class is url, which specifies the web service url **#1**. The url is made of the web service name (RestService.svc) plus the / char and the method name (GetOrdersAmount).

The data parameter contains the method parameters. This class must be a *stringified* Json class **#2**. It is important because otherwise the server is not able to process information.

The type parameter specifies how the request is submitted to the server **#3**. WCF REST services allow only POST calls (unless manually configured to accept GET) therefore you should force a POST.

The contentType **#4** and dataType **#5** parameters inform the server how data are serialized when they are sent from client to server and from server to client, respectively. In this case, we both send and receive data using Json format.

Finally, the success parameter specifies the callback to invoke once data are returned **#6**. Notice that the result is a class that contains several properties; the server result is exposed via the d property. Other than success, you can use error to specify the callback to execute when the server call generates an error.

Now that we have data from the server, we have to update the interface to show the orders amount. This is all but difficult thanks to the manipulation methods of jQuery. In the end, we just need to write the following statement in the success handler.

```
$("#amount").html(result.d);
```

We just retrieve the span tag that shows the amount (which has the id amount) and set its content to the value returned by the server.

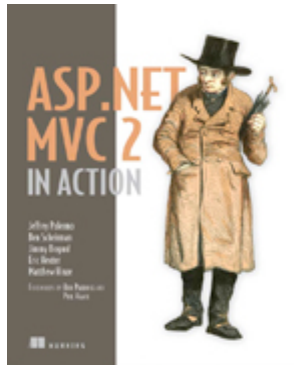
Discussion

As you see, manipulating the interface using server data is not that difficult. In this case, the interaction has been simple because only one field had to be updated. When more complex widget must be updated (say a grid for instance), the things gets complicated. However, it's just a matter of retrieving the objects and setting their values—nothing more. Surely, more code is necessary, but the technique doesn't change.

Summary

In scenarios where performance is critical, it's best to have the server retrieve only data and use JavaScript code to update the interface. jQuery makes this pattern very simple to follow. What's more, jQueryUI further simplifies the building of user-friendly interface, making the development of better applications easier than ever.

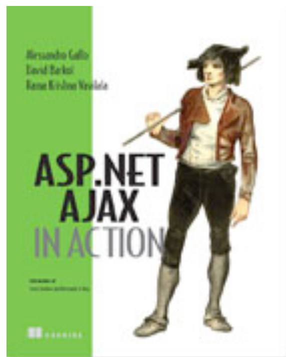
Here are some other Manning titles you might be interested in:



ASP.NET MVC 2 in Action

IN PRINT

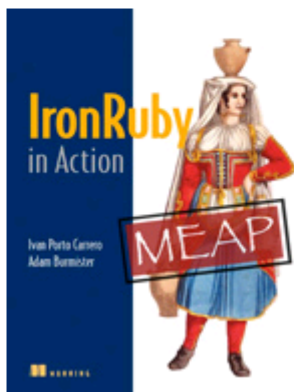
Jeffrey Palermo, Ben Scheirman, Jimmy Bogard, Eric Hexter, and Matthew Hinze
June 2010 | 432 pages
ISBN: 9781935182795



ASP.NET AJAX in Action

IN PRINT

Alessandro Gallo, David Barkol, and Rama Krishna Vavilala
August 2007 | 550 pages
ISBN: 1933988142



IronRuby in Action

EARLY ACCESS EDITION

Ivan Porto Carrero and Adam Burmister
MEAP Began: May 2008
Softbound print: February 2011 (est.) | 350 pages
ISBN: 1933988614