Accessing External Data

Ryan Hodson on Jan 12th 2013 with 0 Comments

Tutorial Details

-
- **Difficulty:** Intermediate
- **Completion Time:** 30 Minutes

View post on Tuts+ Beta**Tuts+ Beta** is an optimized, mobile-friendly and easy-to-read version of the Tuts+ network.

*This entry is part 8 of 9 in the **Knockout Succinctly** Session – Show All*

« PreviousNext »

For most web applications, collecting user input is relatively useless if you can't pass that data along to a server. In this lesson, we'll learn how to send and receive information from a server using AJAX requests. This puts the *model* back into the Model-View-ViewModel design pattern underpinning Knockout.js.

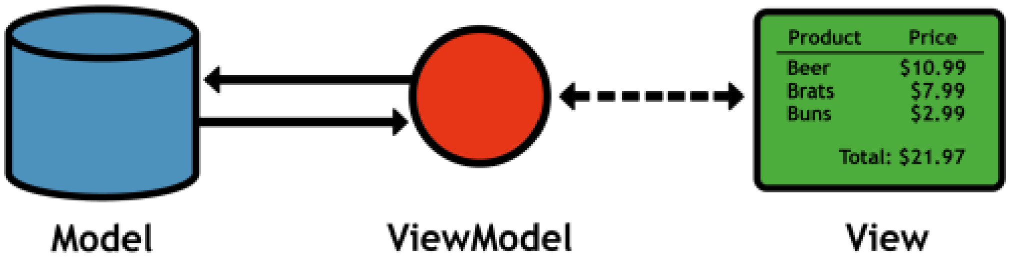


Figure 27: Adding the model back into our MVVM pattern

Remember that Knockout.js is designed to be compatible with any other client-side or server-side technology. This series uses jQuery's `$.getJSON()` and $.post() functions, but you're free to use any JavaScript framework that can send and receive JSON data. Similarly, the server-side scripting language is completely up to you. Instead of presenting back-end code samples, this lesson simply includes the JSON data expected by Knockout.js. Generating this output should be trivial to implement in any modern scripting language.

A New HTML Form

We're going to use a fresh HTML page to experiment with Knockout.js/AJAX integration. Since this page will have to interact with some server-side code, make sure it's accessible from the document root of your local server. We'll start out with something similar to the previous lesson:

```
<html lang='en'>
<head>
  <title>External Data</title>
  <meta charset='utf-8' />
  <link rel='stylesheet' href='style.css' />
</head>
<body>
  <h2>

  <form action="#" method="post">
    <p>First name: <input data-bind='value: firstName' /></p>
    <p>Last name: <input data-bind='value: lastName' /></p>
    <div>
      Your favorite food:
```

```
      <select data-bind='options: activities,
          value: favoriteHobby'></select>
    <em>Load Data</button></em></div></div>
  </form>

  <script src='knockout-2.1.0.js'></script>
  <script src='jquery-1.7.2.js'></script>
  <script>
    function PersonViewModel() {
      var self = this;
      self.firstName = ko.observable("");
      self.lastName = ko.observable("");
      self.activities = ko.observableArray([]);
      self.favoriteHobby = ko.observable("");
    }

    ko.applyBindings(new PersonViewModel());
  </script>
</body>
</html>
```

This is a basic form with a few `<input>` fields so we can see how to send and receive information from the server. Notice that we also include the jQuery library before our custom <script> element.

Loading Data

You probably noticed that unlike previous lessons, all of our observables are empty. Instead of hard-coding data into our ViewModel, we're going to load it from a server using jQuery's `$.getJSON()` method. First, let's make a button for loading data (typically, you would automatically load the data when your application starts up, but this way we can see how everything works step-by-step):

```
<p><button data-bind='click: loadUserData'>Load Data</button></p>
```

The handler for this button uses `$.getJSON()` to call a server-side script:

```
self.loadUserData = function() {
  $.getJSON("/get-user-data", function(data) {
    alert(data.firstName);
  });
}
```

The `/get-user-data` string should be the path to the script. Again, as long as it can encode and decode JSON, any server-side language can be used with Knockout.js. For our example, it should return a JSON-formatted string that looks something like the following:

```
{"firstName":"John",
 "lastName":"Smith",
 "activities":[
    "Golf",
    "Kayaking",
    "Web Development"],
  "favoriteHobby":"Golf"
}
```

The `$.getJson()` method automatically translates this string back into a JavaScript object and passes it to the handler method via the data parameter. It’s trivial to update our ViewModel with the new information:

```
self.loadUserData = function() {
  $.getJSON("/get-user-data", function(data) {
    self.firstName(data.firstName);
    self.lastName(data.lastName);
    self.activities(data.activities);
    self.favoriteHobby(data.favoriteHobby);
  });
}
```

After clicking the **Load Data** button, `$.getJSON()` loads data from the server and uses it to update all of our ViewModel’s observables. As always, Knockout.js automatically updates the form fields to match.

Saving Data

For normal web applications, saving data is a simple matter of converting objects to JSON and sending it to the server with something like jQuery’s `$.post()` method. Things are somewhat more complicated for Knockout.js applications. It’s not possible to use a standard JSON serializer to convert the object to a string because ViewModels use observables instead of normal JavaScript properties. Remember that observables are actually functions, so trying to serialize them and send the result to a server would have unexpected results.

Fortunately, Knockout.js provides a simple solution to this problem: the `ko.toJSON()` utility function. Passing an object to ko.toJSON() replaces all of the object's observable properties with their current value and returns the result as a JSON string.

Create another button called "Save Data" and point it to a `saveUserData()` method on the ViewModel. Then, you can see the JSON generated by ko.toJSON() with the following:

```
self.saveUserData = function() {
  alert(ko.toJSON(self));
}
```

Clicking this button should display the current data in your form fields transformed into a JSON string. Now that we've gotten rid of all our observables, we can send this to the server for processing:

banglalink
proudly towards
we are relentlessly working towards making a difference in your life everyday and this time with 3G.
3G

```
self.saveUserData = function() {
  var data_to_send = {userData: ko.toJSON(self)};
  $.post("/save-user-data", data_to_send, function(data) {
    alert("Your data has been posted to the server!");
  });
}
```

This sends the JSON string representing your ViewModel to a script called `/save-user-data` using the POST method. As a result, your script should find the string under a userData entry in its POST dictionary. You can then deserialize the JSON string into an object, save it into your database, or do whatever kind of server-side processing you need to do.

Mapping Data to ViewModels

The loading and saving mechanisms covered in the previous two sections provide everything you need to create rich user interfaces backed by an arbitrary server-side scripting language. However, manually mapping loaded data to observables can become quite tedious if you're working with more than just a few properties.

The `mapping` plug-in for Knockout.js solves this problem by letting you automatically map JSON objects loaded from the server to ViewModel observables. In essence, mapping is a generic version of our `saveUserData()` and loadUserData() methods.

The `mapping` plug-in is released as a separate project, so we'll need to download it and include it in our HTML page before using it:

```
<script src='knockout.mapping-latest.js'></script>
```

Next, we're going to completely replace our `PersonViewModel`. In its place, we'll use jQuery's `$.getJSON()` method to load some initial data from the server and let the mapping plug-in dynamically generate observables. Replace the entire custom <script> element with the following:

```
<script>
  $.getJSON("/get-user-data", function(data) {
    var viewModel = ko.mapping.fromJS(data);
    ko.applyBindings(viewModel);
  });
</script>
```

When our application loads, it immediately makes an AJAX request for the initial user data. Your server-side script for `/get-intial-data` should return the same thing as the sample JSON output from the Loading Data section of this lesson. Once the data is loaded, we create a ViewModel via `ko.mapping.fromJS()`. This takes the native JavaScript object generated by the script and turns each property into an observable. Aside from the saveUserData() and loadUserData() methods, this dynamically generated ViewModel has the exact same functionality as PersonViewModel.

At this point, we've only *initialized* our ViewModel with data from the server. The mapping plug-in also lets us *update* an existing ViewModel in the same fashion. Let's go ahead and add an explicit `loadUserData()` method back to the ViewModel:

```
viewModel.loadUserData = function() {
  $.getJSON("/get-user-data", function(data) {
    ko.mapping.fromJS(data, viewModel);
  });
}
```

In the old version of `loadUserData()`, we had to manually assign each data property to its respective observable. But now, the `mapping` plug-in does all of this for us. Note that passing the data object as the first argument to ko.mapping.fromJS() causes it to *update* the ViewModel instead of *initializing* it.

The mapping plug-in only relates to loading data, so `saveUserData()` remains unaffected except for the fact that it needs to be assigned to the viewModel object:

```
viewModel.saveUserData = function() {
  var data_to_send = {userData: ko.toJSON(viewModel)};
  $.post("/save-user-data", data_to_send, function(data) {
    alert("Your data has been posted to the server!");
  });
}
```

And now we should be back to where we started at the beginning of this section—both the **Load Data** and **Save Data** buttons should work, and Knockout.js should keep the view and ViewModel synchronized.

While not a necessary plug-in for all Knockout.js projects, the `mapping` plug-in does make it possible to scale up to complex objects without adding an extra line of code for every new property you add to your ViewModel.

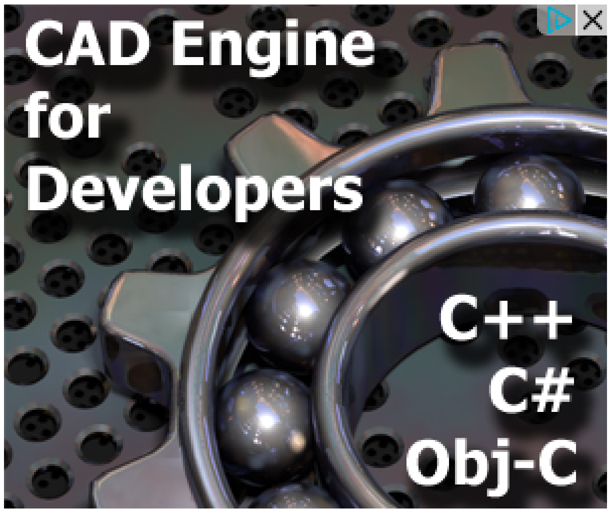
Summary

In this lesson, we learned how Knockout.js can communicate with a server-side script. Most of the AJAX-related functionality came from the jQuery web framework, although Knockout.js does provide a neat utility function for converting its observables into native JavaScript properties. We also discussed the mapping plug-in, which provided a generic way to convert a native JavaScript object to a ViewModel with observable properties.

Remember, Knockout.js is a pure client-side library. It's only for connecting JavaScript objects (ViewModels) with HTML elements. Once you have this relationship set up, you can use any other technology you like to communicate with the server. On the client-side, you could replace jQuery with Dojo, Prototype, MooTools, or any other framework that supports AJAX requests. On the server-side, you have the choice of ASP.NET, PHP, Django, Ruby on Rails, Perl, JavaServer Pages...you get the idea. This separation of concerns makes Knockout.js an incredibly flexible user interface development tool.

> This lesson represents a chapter from *Knockout Succinctly*, a free eBook from the team at Syncfusion.

Like Be the first of your friends to like this.



Tags: knockout

By Ryan Hodson

Ryan Hodson has worked in a diverse collection of programming fields, building everything from websites to e-publishing platforms, touch-screen thermostats, and natural language processing tools. These experiences have led to a love of exploring new software and a proficiency in several languages and frameworks.In 2012, Ryan founded an independent publishing firm called RyPress and published his first book, Ry’s Friendly Guide to Git. Since then, he has worked as a freelance technical writer for well–known software companies, including Syncfusion and Atlassian. Ryan continues to publish high–quality software tutorials via RyPress.com.

Note: Want to add some source code? Type <pre><code> before it and </code></pre> after it. Find out more