Data binding and AJAX (Part 3 of the AngularJS - from beginner to expert in 7 steps series)

Published on July 31, 2013

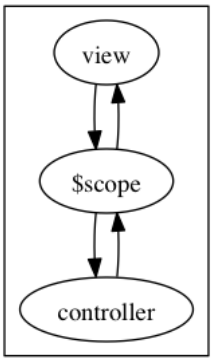
This is the third post of AngularJS - from beginner to expert in 7 steps (/posts/beginner2expert-how_to_start.html).

We started our first (/posts/beginner2expert-how_to_start.html) post by showing you how to start out building an AngularJS app. In the second (/posts/beginner2expert-scopes.html) post, we discussed how `scopes` and the `$scope` service work.

Throughout this tutorial series, we are building an NPR audio player that will show us the current stories on the show *Morning Edition* and play them in our browser. To see the fully finished demo, head over here (/code/beginner_series).

3. Data binding

We can make our app a bit more interesting by `binding` an input field to the person.name attribute. This step sets up a bi-directional binding from the input field to the page.



Bi-directional in this context means that if the view changes the value, the model *sees* the change, and if the model changes the value, then the view will see the change. AngularJS sets this up *automatically* for you. We'll discuss the `digest_loop` in depth in an upcoming article, if you're curious how it works.

To set up this binding, we'll use the `ng-model` function on the input, like so:

```
1  <div ng-controller="MyController">
2    <input type="text" ng-model="person.name" placeholder="Enter your name" />
3    <h5>Hello {{ person.name }}</h5>
4  </div>
```

Now that we have a binding set up (yes, it's *that* easy), we can see how the view changes the model:

Try it

Ari Lerner

Hello Ari Lerner

As you type in the input box, you'll see that the name underneath changes automatically. That change illustrates how our data binding works in one direction, from the view to the model.

We can also change the model on the (client-side) back end and see it reflected on the front end.

To illustrate our data binding from back end to front end, let's make a clock function in our `MyController` model that will update a value on the `$scope`. In this example, we'll create a clock that will tick every second (as clocks usually do) and change the data on the clock variable:

```
app.controller('MyController', function($scope) {
  $scope.person = { name: "Ari Lerner" };
  var updateClock = function() {
    $scope.clock = new Date();
  };
  var timer = setInterval(function() {
    $scope.$apply(updateClock);
  }, 1000);
  updateClock();
});
```

As you can see, as we change the clock variable on the model, the view automatically updates to reflect our changes.

We can show the `clock` variable that's attached on the `$scope` in the view simply by surrounding it in {{ }}:

```
<div ng-controller="MyController">
  <h5>{{ clock }}</h5>
</div>
```

See it it

"2014-06-19T17:40:03.172Z"

Interaction

Notice that we have bound data to the input field above. We don't have to limit this data binding to only data. We can also call functions on the `$scope` (as mentioned previously).

To bind buttons or links (or any DOM element, really), we'll use another built-in directive, `ng-click`. The `ng-click` directive binds the click event to the method (the `mousedown` browser event) to the DOM element (i.e., when the browser fires a click event on the DOM element, the method is called). Similar to our previous example, the binding looks like:

```
<div ng-controller="DemoController">
  <h4>The simplest adding machine ever</h4>
  <button ng-click="add(1)" class="button">Add</button>
  <button ng-click="subtract(1)" class="button">Subtract</button>
  <h4>Current count: {{ counter }}</h4>
</div>
```

Both the button and the link will be bound to an action on the containing `$scope`, so when they are pressed (clicked), Angular will call the method. Note that when we are telling Angular what method to call, we're putting it in a string *with* the parentheses.

Now, let's create an action on our `DemoController`.

```
app.controller('DemoController', function($scope) {
  $scope.counter = 0;
  $scope.add = function(amount) { $scope.counter += amount; };
  $scope.subtract = function(amount) { $scope.counter -= amount; };
});
```

See it

The simplest adding machine ever

Add Subtract

Current count: 1

Data binding and AJAX in our app

Interaction

As we saw in the last example of the previous section (/posts/beginner2expert-scopes.html), we bound a button to the view using the data binding we just learned about. In that example, we bound the button to the `play()` action and bound the `PlayerController`'s method `play` on the play button (and the same for the stop button, with the stop method).

AJAX

In the last tutorial, we referenced a locally hosted audio file; however, doing so gives us a static NPR file instead of a live NPR feed. In our NPR app, we'll use `$http` to populate the list of available news clips we can play.

Out of the box, AngularJS supports AJAX (or asynchronous JavaScript and XML). This support gives us the ability to make requests back and forth from a server or multiple servers, which is essential to a client-side app like ours that needs to get and set data.

AngularJS supports AJAX through a *service* (which we'll discuss in an upcoming section) called the `$http` service.

> All of the core AngularJS services are prefixed with `$`. We've seen this before with the `$scope` service.

The `$http` service is incredibly flexible and gives us many different ways to call AJAX services. To keep this tutorial simple, we're only going to focus on the simplest method possible. We'll dive deeper into the `$http` service in a more advanced section in the future.

Before we go too far into detail, let's make a request with the `$http` service:

```
1  $http({
2    method: 'JSONP',
3    url: 'https://api.github.com/events?callback=JSON_CALLBACK'
4  }).success(function(data, status, headers, config) {
5    // data contains the response
6    // status is the HTTP status
7    // headers is the header getter function
8    // config is the object that was used to create the HTTP request
9  }).error(function(data, status, headers, config) {
10 });
```

Try it

Make request

```
Request result:
[]
```

The `$http` service is a core AngularJS service that helps faciliate communication with remote HTTP servers via the XMLHttpRequest object or through JSONP.

> Note that AngularJS will take care of handling a JSONP request if you append the EXACT string: `callback=JSON_CALLBACK` as just above. AngularJS will replace `JSON_CALLBACK` with the proper callback it constructs for you.

The `$http` service is a function that takes a configuration object, which defines how the HTTP request is constructed. It will return a *promise* that has two methods *success* and *error*.

To get a list of the available audio clips, let's make a request to NPR's API. First, you'll need to register with NPR to get an API key. Head over to their site at http://www.npr.org/templates/reg/ (http://www.npr.org/templates/reg/) or click here to register from this page.

Keep note of your API key. We'll use that in a minute. Now, we're going to set up our PlayerController to call the `$http` service to fetch a list of the clips.

Just like we did above, let's call the `$http` service to make a request, this time to get all the clips. We want this service to run when the controller is instantiated, so we can simply put this method right in the controller function (which is run when the controller is created), like so:

```
var apiKey = 'YOUR_KEY',
    nprUrl = 'http://api.npr.org/query?id=61&fields=relatedLink,title,byline,text,audio,image,pullQuote,all&output=JSON';

app.controller('PlayerController', function($scope, $http) {
  // Hidden our previous section's content
  // construct our http request
  $http({
    method: 'JSONP',
    url: nprUrl + '&apiKey=' + apiKey + '&callback=JSON_CALLBACK'
  }).success(function(data, status) {
    // Now we have a list of the stories (data.list.story)
    // in the data object that the NPR API
    // returns in JSON that looks like:
    // data: { "list": {
    //   "title": ...
    //   "story": [
    //     { "id": ...
    //       "title": ...
  }).error(function(data, status) {
    // Some error occurred
  });
});
```

Now that we have the the list of clips in `data` in the success function, we can bind it to the `$scope` object simply by storing the list on `$scope` in the `success` callback:

```
// from above
}).success(function(data, status) {
  // Store the list of stories on the scope
  // from the NPR API response object (described above)
  $scope.programs = data.list.story;
}).error(function(data, status) {
  // Some error occurred
```

Now, just like above, we can reference this data in our view simply by referencing `programs` in our view. Note that one of the benefits of using AngularJS is that it will automatically populate promises into your view when the promise resolves.

```
<div ng-controller="PlayerController">
{{ programs }}
</div>
```

Try it

Enter your NPR API key

In the next section, we'll discuss how to display this data object in our view in a meaningful way using some built-in directives (and a little bit more).

The *official* repository for the beginner series is available as a git repo here: https://github.com/auser/ng-newsletter-beginner-series (https://github.com/auser/ng-newsletter-beginner-series).

To get this repo locally, ensure that you have `git` installed, clone the repo, and check out the `part3` branch:

```
git clone https://github.com/auser/ng-newsletter-beginner-series.git
git checkout part3
```

Previous in series (/posts/beginner2expert-scopes.html)
The AngularJS - from beginner to expert in 7 steps series
Next in series (/posts/beginner2expert-directives.html)

← Back to articles (/posts)

Share on Twitter (http://twitter.com/home/?status=Data binding and AJAX (Part 3 of the AngularJS - from beginner to expert in 7 steps series) - h

Follow us on Google+ (https://plus.google.com/+AriLerner?rel=author)