Anatomy of a JavaScript MV* Framework

Craig McKeachie(http://www.sitepoint.com/author/cmckeachie/)

September 25, 2014

The key to quickly learning JavaScript MV* Frameworks is to break them down into a series of features. The main features of an MV* application are routing, data binding, templates/views, models, and data access. In this post I'll describe these features and show code examples from AngularJS, Backbone, and Ember for each feature. You will begin to concretely understand what these frameworks are trying to help you accomplish and realize they are more alike than they are different. In fact, it becomes apparent that most of the frameworks borrow heavily from the successes of the others.

Don't be too concerned about understanding every line of code. For now, try to appreciate how similar they are and the problems they can solve for your project.

Routing

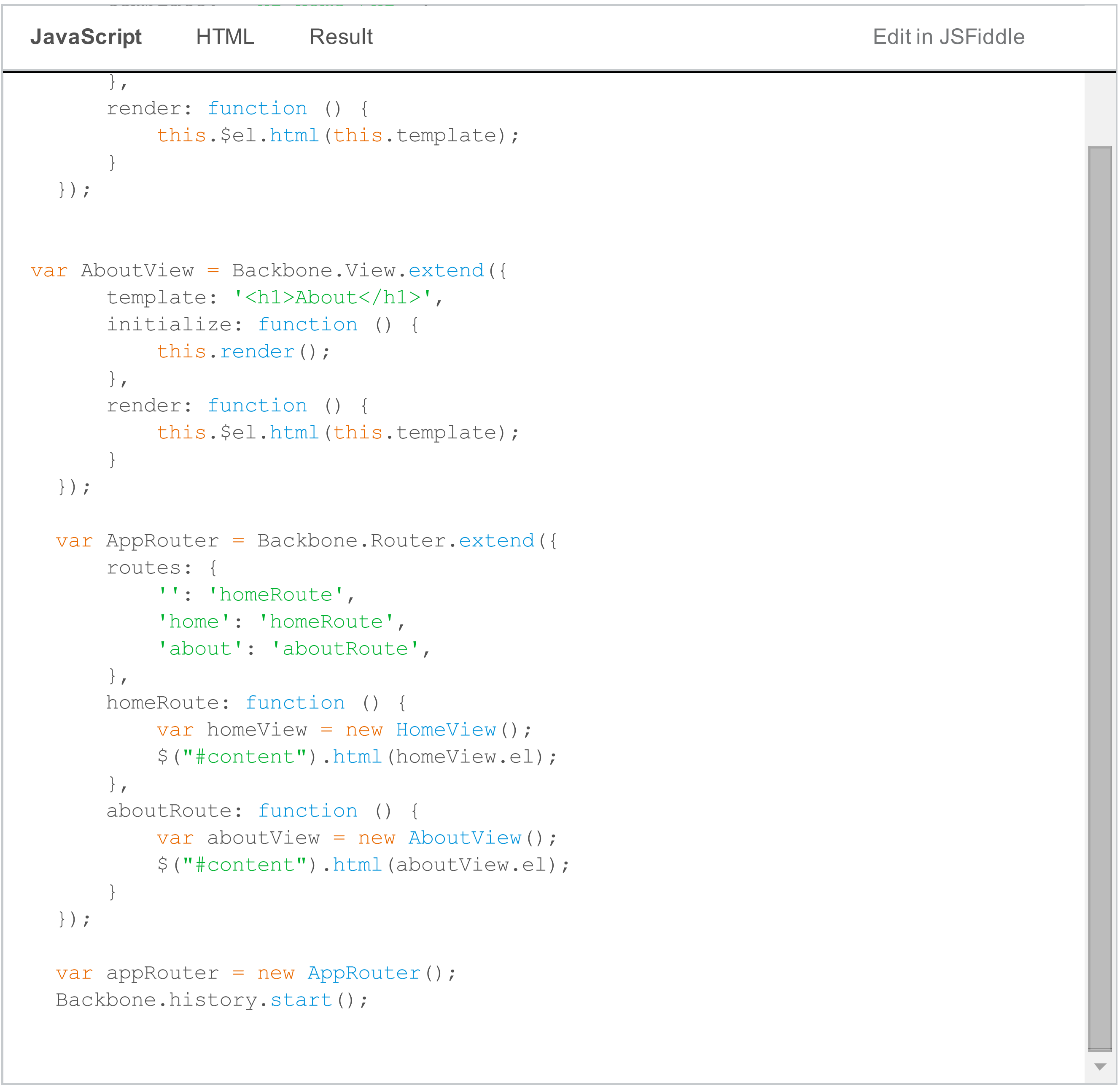
Routing, at a minimum maps your URLs to a function, but sometimes goes as far as implementing a full “state machine” design pattern for managing state transitions within a view. If you’ve ever used the router in a server-side MVC framework such as Rails, CodeIgniter, CakePHP, ASP.NET MVC, etc., then you can just think of the JavaScript MV* routers as the same thing but running on the client in JavaScript.

You may be wondering how this works and will this work on older browsers? Everything after the hash tag in a URL in considered the route, however, if HTML push-state support is configured (with one line of code in most frameworks) then URLs without hashes that match the routes will be intercepted on the client and run JavaScript as well.

Enough details let’s see some code.

Backbone Example

Here is a simple example of routing in Backbone.js:



```
      },
      render: function () {
          this.$el.html(this.template);
      }
  });


var AboutView = Backbone.View.extend({
      template: '<h1>About</h1>',
      initialize: function () {
          this.render();
      },
      render: function () {
          this.$el.html(this.template);
      }
  });

  var AppRouter = Backbone.Router.extend({
      routes: {
          '': 'homeRoute',
          'home': 'homeRoute',
          'about': 'aboutRoute',
      },
      homeRoute: function () {
          var homeView = new HomeView();
          $("#content").html(homeView.el);
      },
      aboutRoute: function () {
          var aboutView = new AboutView();
          $("#content").html(aboutView.el);
      }
  });

  var appRouter = new AppRouter();
  Backbone.history.start();
```

Notice the `AppRouter` object. Routes are mapped to functions. The functions simply create a view object which manages a DOM fragment and adds it to the page when the URL changes. The `Backbone.history.start()` tells Backbone to start listening for URL changes.

AngularJS Example

Here is a simple example of routing in AngularJS:

JavaScript HTML Result Edit in JSFiddle

```
routingExample.controller('HomeController', function ($scope) {});
routingExample.controller('AboutController', function ($scope) {});

routingExample.config(function ($routeProvider) {
    $routeProvider.
    when('/home', {
        templateUrl: 'embedded.home.html',
        controller: 'HomeController'
    }).
    when('/about', {
        templateUrl: 'embedded.about.html',
        controller: 'AboutController'
    }).
    otherwise({
        redirectTo: '/home'
    });
});
```

The AngularJS example is very similar to the Backbone example except routes map to `templateUrl` s and controller functions.

Ember Example

Below is a simple example of routing in Ember:

JavaScript **HTML** Result Edit in JSFiddle

```
<script type="text/x-handlebars" data-template-name="application">
    <h1> Ember Routing</h1>
    <div id="navigation">
       <a href="#">Home</a>
       <a href="#about">About</a>
    </div>
    <br />
    {{outlet}}
</script>
<script type="text/x-handlebars" data-template-name="home">
    <h2> Home Page </h2>
</script>
<script type="text/x-handlebars" data-template-name="about">
    <h2> About Page </h2>
</script>
```

Again, very similar to the others except with Ember.js the first parameter to the router’s “resource” object is a `routeName` and the second is the URL. The order of these parameters confused me at first until someone pointed out that the path parameter is optional and frequently can be set by convention as it is with the about page in the example. Also, Ember templates are required to make this simple routing example work but I'll go over them in a later section. For now it's enough to know the templates get placed in the `{{outlet}}`.

Data Binding

Data binding allows changes in the model data to be updated in the view and/or changes in the view to be automatically updated in the model without additional code. One-way data binding generally indicates changes to the model are propagated to the view. Two-way data binding adds the ability for view changes to immediately be shown on the model. Data binding eliminates a lot of boilerplate code developers write and frees the developer to focus on the unique problems in the application.

AngularJS Example

Below is a simple example of two-way data binding in AngularJS. Typing in the input field will show the entered text after the welcome message.

```
Name: 

Welcome to AngularJS
```

Backbone Example

Backbone doesn't have automatic data binding, but it is possible to do manually. In practice, I've found one-way data binding which updates the view when changes are made to the model to be extremely useful. Data binding from the view to the model real-world use cases are less common.

Below is a simple example where code has been implemented to bind both ways.

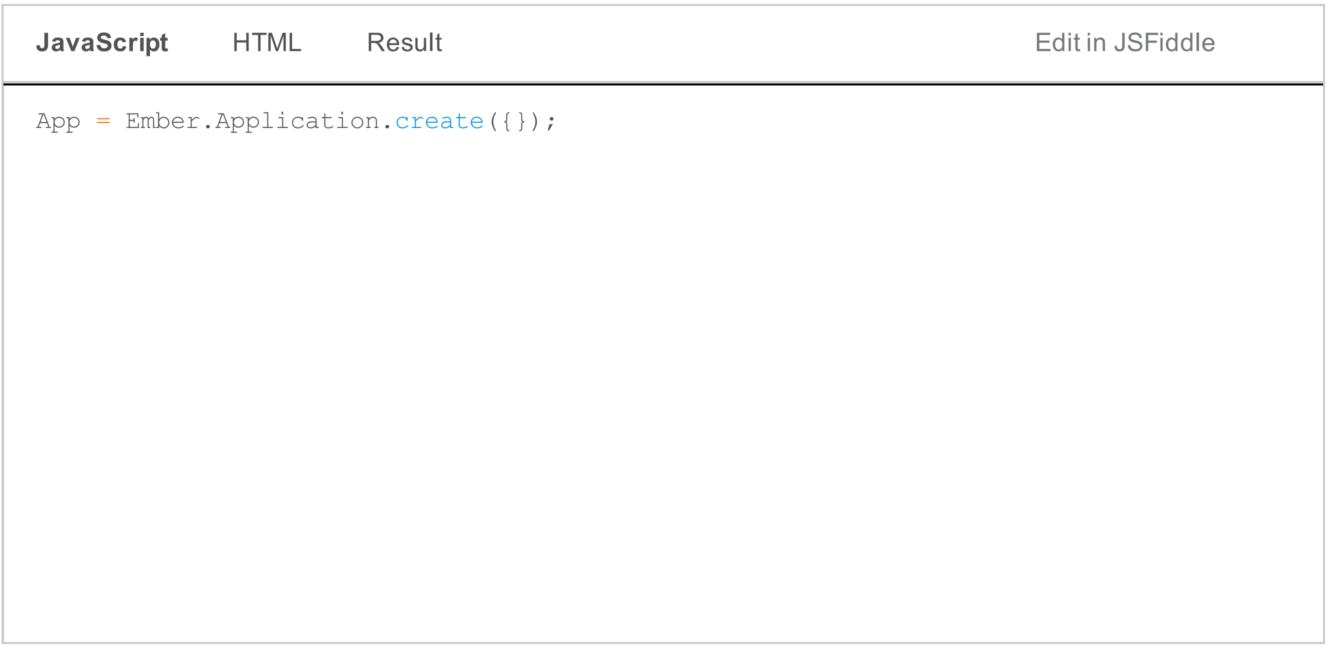
JavaScript HTML Result Edit in JSFiddle

```
var MessageView = Backbone.View.extend({
  template: _.template($('#message-template').html()),
  events: {
    'keyup #name': 'updateModel'
  },
  updateModel: function(event) {
    this.model.set({name:$("#name").val()});
  },
  initialize: function() {
    this.listenTo(this.model, "change", this.render);
    this.render();
  },
  render: function() {
      this.$('#message').html(this.template(this.model.toJSON()));
    return this;
  }
});

var person = new Backbone.Model({name:''});
messageView = new MessageView({el:   $('#message-container') ,model: person});
```

In summary, you listen for a change event on the model and call the view's render property to get the model to update the view. Similarly, you listen for `keyup` on an input and change the model by getting the value out of the input with jQuery and setting it on the model to have the view update the model. This example should give you a feel for how much code it takes to get data binding working. It is also worth noting that there are numerous plug-ins that add support for data binding to Backbone.

Ember Example

Data binding in Ember looks like this:



```
App = Ember.Application.create({});
```

Ember uses the familiar Handlebars for templating, but the framework also includes “input helpers” to bind common form input fields. The curly braces `{{` replace the angle brackets `<` on the input in this example and the `name` property has no quotes so the helper knows to bind it.

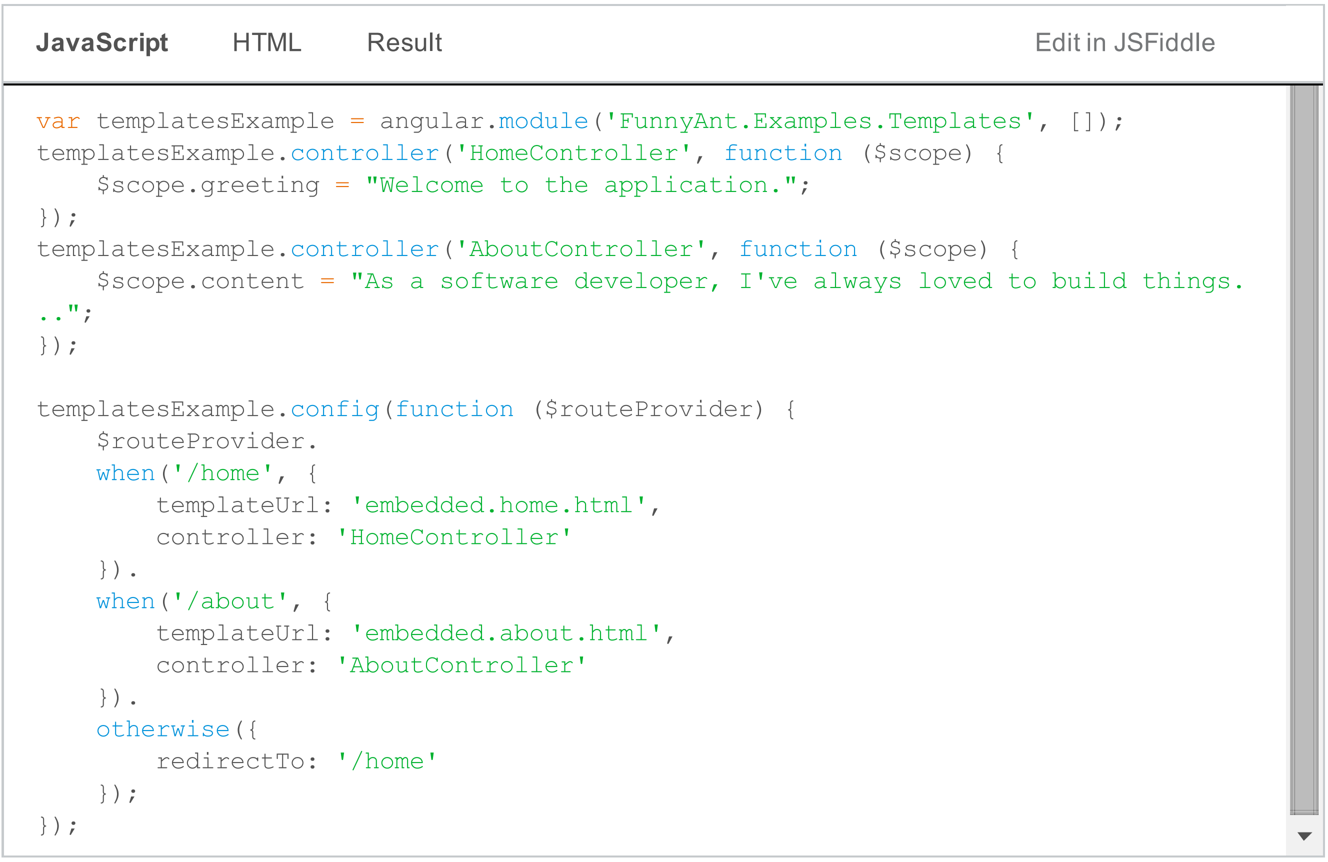
Templates/Views

Templates can be entire pages of HTML, but more commonly are smaller fragments of HTML with data binding placeholder expressions included for dynamic data. They can be logic-less with the philosophy that there should be little to no logic in your views, while others allow you to embed JavaScript directly in the template. Templates can be DOM-based and use the DOM to dynamically insert dynamic data or string-based, treating the HTML as strings and replacing the dynamic parts.

Lets look at some examples.

AngularJS Example

Here is a simple templates example in AngularJS.



```
var templatesExample = angular.module('FunnyAnt.Examples.Templates', []);
templatesExample.controller('HomeController', function ($scope) {
    $scope.greeting = "Welcome to the application.";
});
templatesExample.controller('AboutController', function ($scope) {
    $scope.content = "As a software developer, I've always loved to build things.
..";
});

templatesExample.config(function ($routeProvider) {
    $routeProvider.
    when('/home', {
        templateUrl: 'embedded.home.html',
        controller: 'HomeController'
    }).
    when('/about', {
        templateUrl: 'embedded.about.html',
        controller: 'AboutController'
    }).
    otherwise({
        redirectTo: '/home'
    });
});
```

You'll notice this is very similar to the earlier routing example with some data binding added to show how templates can help in your application. Templates are all included in `script` tags in the main HTML file to make the example easy to follow and work in jsfiddle.net, but templates can be external to the view in AngularJS by giving a valid file path to the `templateUrl` property when configuring the `$routeProvider`.

The preferred way of handling templates in larger scale applications where performance is a concern is to concatenate and register your AngularJS templates in the Angular `$templateCache` at compile time with a build task such as this one (https://www.npmjs.org/package/grunt-angular-templates).

Ember Example

Below is an example of templates in Ember.

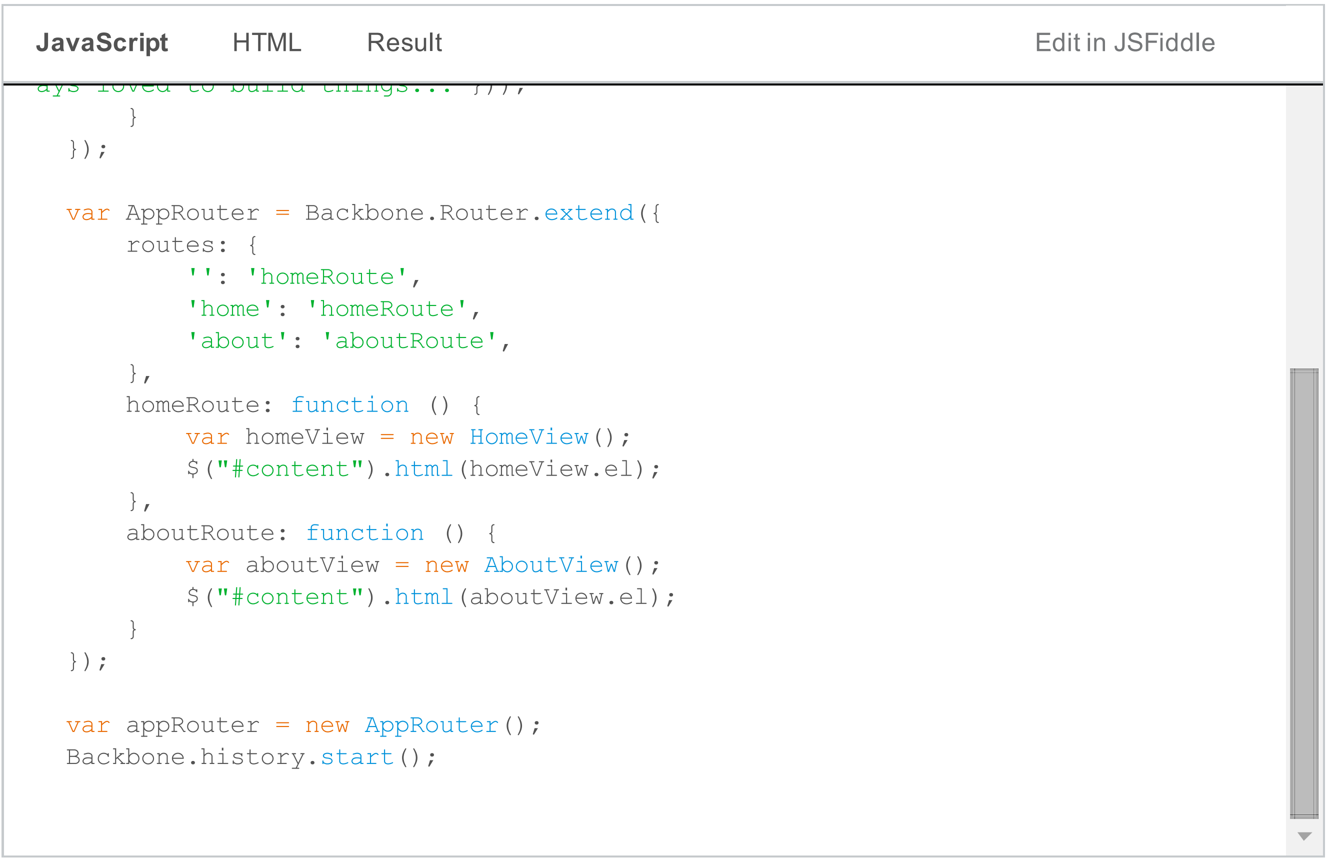
JavaScript HTML Result Edit in JSFiddle

```
App = Ember.Application.create({});

App.Router.map(function() {
    //first paramater refers to the template
    this.resource('home', { path: '/' });
    this.resource('about', {path: '/about'});
});

App.HomeRoute = Ember.Route.extend({
    model:function(){
        return{
            greeting: 'Welcome to the application.'
        }
    }
});

App.AboutRoute = Ember.Route.extend({
  model: function(){
      return{
          pagecontent: 'As a software developer, I have always loved to
           build things...'
      };
  }
});
```

An Ember route is an object that tells the template which model it should display. I think of it as the most basic controller for your template and resource (URL) whose main job is to load the model. If you need to get fancy and store application state then you need a controller.

Backbone Example

Now, lets look at a simple example of templates in Backbone.


JavaScript HTML Result Edit in JSFiddle

```
ays loved to build things.' }));
        }
    });

    var AppRouter = Backbone.Router.extend({
        routes: {
            '': 'homeRoute',
            'home': 'homeRoute',
            'about': 'aboutRoute',
        },
        homeRoute: function () {
            var homeView = new HomeView();
            $("#content").html(homeView.el);
        },
        aboutRoute: function () {
            var aboutView = new AboutView();
            $("#content").html(aboutView.el);
        }
    });

    var appRouter = new AppRouter();
    Backbone.history.start();
```

This is a modification of the routing example, but instead of the HTML being hard-coded in the the view object's template property, the markup is now in the HTML page inside a `script` tag with an `id` attribute (browsers ignore script tags with types they do not recognize such as text/template so the template won't be shown or executed). To get the template (HTML fragment) we use a jQuery selector to find the element by the `script` tag's `id`, grab the `innerHTML`, and then assign the HTML to the template property of the view object (it's just a string).

Models

Models are the client-side version of what is commonly referred to as business objects, domain objects, or entities. In general, the idea behind models in client-side MV* frameworks is to establish a central point for the data in the application as well as any behavior that should be encapsulated with that data. This model can be contrasted with server-side MVC plus jQuery architectures where the model data is commonly stored in the DOM. By having a model the goal is remove that data and state from the DOM and put it in a common place where it can be reused.

Backbone Example

Models hold data and keep it out of the DOM, and emit events such as `change` which allows numerous views to react accordingly and update the user interface everywhere it is needed. This gives you one source of truth, which is not the user interface.

JavaScript HTML Result Edit in JSFiddle

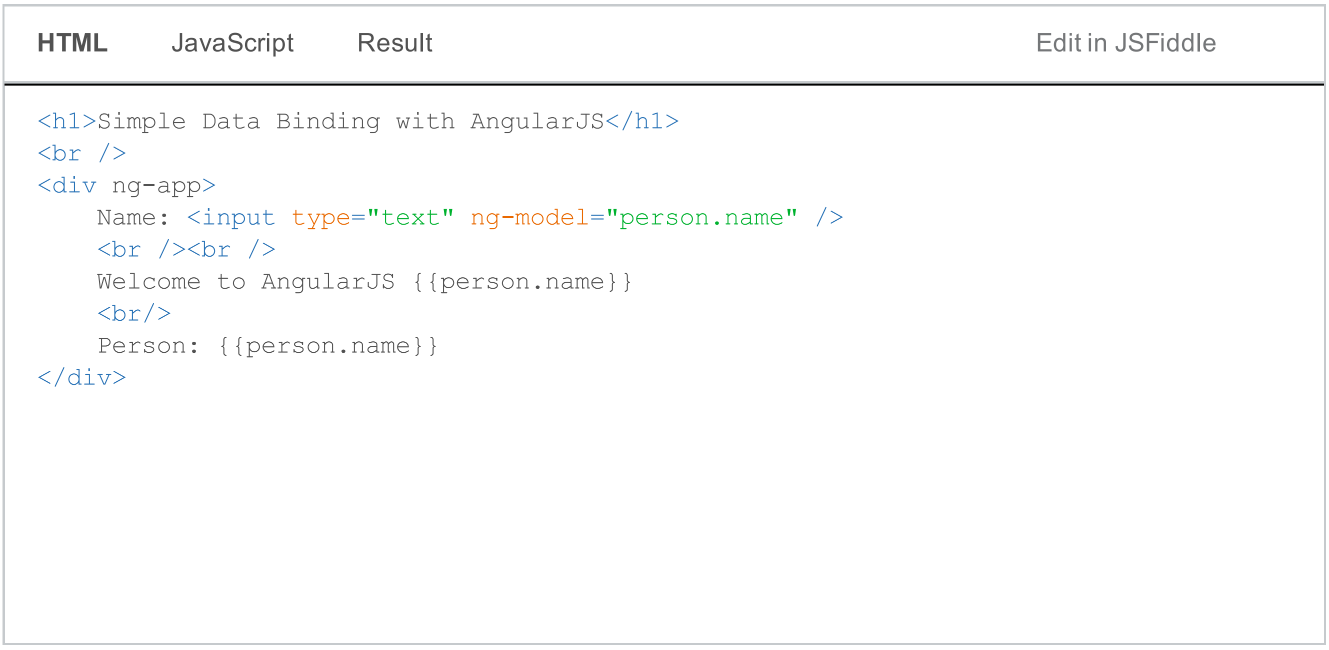
```
    },
    updateModel: function (event) {
        this.model.set({
            name: $("#name").val()
        });
        $("#name").html('');
    },
    initialize: function () {
        _.bindAll(this, 'render');
        this.listenTo(this.model, "change", this.render);
    },
    render: function () {
        this.$('#message').html(this.template(this.model.toJSON()));
        return this;
    }
});

var NameView = Backbone.View.extend({
    template: _.template($('#name-template').html()),
    initialize: function () {
        _.bindAll(this, 'render');
        this.listenTo(this.model, "change", this.render);
    },
    render: function () {
        this.$el.html(this.template(this.model.toJSON()));
        return this;
    }
});

var Person = Backbone.Model.extend({
    defaults: {
        name: ''
    }
});

var person = new Person();

var messageView = new MessageView({
    el: $('#message-container'),
    model: person
});

var nameView = new NameView({
    el: $('#name-container'),
    model: person
});
```

I've modified the data binding example from earlier by adding a new template and view that looks at the same Person model object. Previously, I declared the Person model on the fly to keep things simple, but now I've added the call to `Backbone.Model.extend()` to demonstrate how you create a prototype for a model that can be used over and over similar to classes in classical languages. Notice how both views are listening to the same person model object (the change event) and updating themselves. By having this single source of data the numerous calls to specific DOM elements can be encapsulated in their own tidy views and one model can serve them all.

AngularJS Example

The idea of one model that is the truth about the state in your application exists in AngularJS but Angular allows you to use plain old JavaScript objects as your model and then adds watchers “under the hood” to any property that is data bound in the view with the directive `ng-model`. These watchers then automatically alert other parts of the application that are bound to that same model and these DOM elements know how to update themselves.

Here is the updated AngularJS data binding example showing two parts of the view being updated.



```html
<h1>Simple Data Binding with AngularJS</h1>
<br />
<div ng-app>
    Name: <input type="text" ng-model="person.name" />
    <br /><br />
    Welcome to AngularJS {{person.name}}
    <br/>
    Person: {{person.name}}
</div>
```

Data Access

Data access is about how you get and save data for your application. In general, the frameworks assume you are making a call to an API that is returning you JSON.

AngularJS Example

AngularJS handles data in two different ways. First, by providing support for manual Ajax calls in a very similar way to jQuery's `$.ajax` functionality via `$http`. In addition, if your backend is a strictly RESTful service, AngularJS provides a `$resource` class that makes calls to the RESTful service extremely terse.

`$http` Example

```
app.factory('myService', function($http) {
  return {
    getFooOldSchool: function(callback) {
      $http.get('foo.json').success(callback);
    }
  };
});

app.controller('MainCtrl', function($scope, myService) {
  myService.getFooOldSchool(function(data) {
    $scope.foo = data;
  });
});
```

`$resource` Example

```
//create a todo
var todo1 = new Todo();
todo1.foo = 'bar';
todo1.something = 123;
todo1.$save();

//get and update a todo
var todo2 = Todo.get({id: 123});
todo2.foo += '!';
todo2.$save();

//delete a todo
Todo.$delete({id: 123});
```

Backbone Example

Backbone assumes you are interacting with a RESTful API but allows you to override one method, `Backbone.sync()`, if not. You tell your model where the resource is on the server (the URL) and then you can just call `save()`.

```
var UserModel = Backbone.Model.extend({
  urlRoot: '/user',
  defaults: {
    name: '',
    email: ''
  }
});
var user = new Usermodel();
// Notice that we haven't set an `id`
var userDetails = {
  name: 'Craig',
  email: 'craigmc@funnyant.com'
};
// Because we have not set an `id` the server will call
// POST /user with a payload of {name:'Craig', email: 'craigmc
// The server should save the data and return a response conta
user.save(userDetails, {
  success: function (user) {
    alert(user.toJSON());
  }
});
```

Ember Example

Ember has Ember Data which is not technically part of the core framework but is shooting to provide a more robust data persistence/data storage story. It provides many of the facilities you'd find in server-side ORMs like ActiveRecord, but is designed specifically for the unique environment of JavaScript in the browser. At the time of writing, the Ember Core Team is close to releasing v1.0, but has not and many Ember projects simply use the `$.ajax` methods in jQuery just as AngularJS uses `$http` in the above examples.

Conclusion

This post broke down JavaScript MV* frameworks by features to provide insight into what functionality is provided by these frameworks and to bring readers to the realization that they are actually very similar. Once you understand the framework features and how they fit together it becomes much easier to quickly learn multiple frameworks and find the right one for your project.

Craig McKeachie (http://www.sitepoint.com/author/cmckeachie/)

Craig McKeachie recently released the JavaScript Framework Guide: AngularJS, Backbone, and Ember (http://www.funnyant.com/javascript-framework-guide/). He blogs about web development and JavaScript at funnyant.com (http://funnyant.com/) and hosts a podcast frontendcast.com (http://frontendcast.com/). Craig has been a web developer for over 15 years and doesn't plan on stopping anytime soon.

22 Comments SitePoint dongguangming

Recommended 2 Share Sort by Best

Join the discussion…

Daniel • 10 months ago

Backbone seems way too verbose...
Why should I write
initialize: function () {
this.render();
},
or
render: function () {
this.$el.html(this.template);
}

for every Backbone view?

1 • Reply • Share ›

rajushank84 Daniel • 9 months ago

Its for control. If you dont want to do it for every view, you can define a "base" view and extend() other views from it. Two-way "automatic" data binding looks good for a small app, but the control backbone gives you is immensely helpful

when you run into "what-if" situations.
1 ^ | ˅ • Reply • Share ›

Tarabass • 10 months ago
This is why I love ExtJs and Sencha Touch!
1 ^ | ˅ • Reply • Share ›

Bnny • 10 months ago
There are hundreds better JS markup/libraries/fw than just backbone and angular and other 'famous'. You better wake up.
1 ^ | ˅ • Reply • Share ›

Stardrive Engineering • 10 months ago
It would be helpful to see React.js and Flux in this comparison as well.
1 ^ | ˅ • Reply • Share ›

Konstantin Tarkus ➦ Stardrive Engineering • 10 months ago
Yeah, would be nice. Especially in a light that many folks switch from MVC frameworks to React.js lately.
^ | ˅ • Reply • Share ›

cmckeachie ➦ Konstantin Tarkus • 10 months ago
Here is a recent article I wrote trying to figure out where React.js fits in with these frameworks. It is so different it is difficult to directly compare as I've done in this article.

http://www.funnyant.com/reactj...
^ | ˅ • Reply • Share ›

Hudson Pereira • 2 months ago
Great article. It gives every reader the big picture before studying any javascript framework.

Well done!
^ | ˅ • Reply • Share ›

Gabor Babicz • 10 months ago
In the "Data Binding" section the Ember example has a whole lot of unnecessary JavaScript. In reality, you don't need to anything besides creating your application: http://jsfiddle.net/gXcfL/6/

Could you please update the JSFiddle?
^ | ˅ • Reply • Share ›

cmckeachie ➦ Gabor Babicz • 10 months ago
I updated the fiddle but the article is specifically pointed to a version /4 so I'm working through the sitepoint editors to update it to point to the base version on JSFiddle. Thanks for pointing this out.

^ | ˅ • Reply • Share ›

Gabor Babicz ➦ cmckeachie • 10 months ago

Wow, awesome! It's so rare to see authors bother to make minor updates like this. Thank you! :)

^ | ˅ • Reply • Share ›

Ibrahim • 10 months ago

I'd give Backbone a whirl even though use Angular more; to cut back on some of them "magic".

^ | ˅ • Reply • Share ›

Philip J Cox • 10 months ago

Great write up. Has helped me to decide which FW to learn first. Angular is the winner in my book, nice clean succinct code.

^ | ˅ • Reply • Share ›

TrubiT • 10 months ago

Good overview, thats what I was looking for. First I did not understand why in "Models" examples you used such complicated Backbone code for simple thing. Then I realised you were trying to show how to connect two views with one model. Thanks!

^ | ˅ • Reply • Share ›

cmckeachie ➦ TrubiT • 10 months ago

Thanks for the feedback.

^ | ˅ • Reply • Share ›

Bassil Redman • 10 months ago

some of the ember examples are missing?

^ | ˅ • Reply • Share ›

cmckeachie ➦ Bassil Redman • 10 months ago

Probably should have included an Ember Model.

^ | ˅ • Reply • Share ›

Bassil Redman • 10 months ago

loved the examples

^ | ˅ • Reply • Share ›

Vivek Kumar Bansal • 10 months ago

Great Article!! Definitely going into my Bookmarks

^ | ˅ • Reply • Share ›

cmckeachie ➦ Vivek Kumar Bansal • 10 months ago

Thanks Vivek.

^ | ˅ • Reply • Share ›

cmckeachie • 10 months ago

Thanks Lucien!

^ | ⌄ • Reply • Share ›

Lucien Dubois • 10 months ago

Next Article

[Understanding Two-way Data Binding in AngularJS ➔ (http://www.sitepoint.com/two-way-data-binding-angularjs/)]

About

Our Story (/about-us)
Advertise (/advertising)
Press Room (/press)
Reference (http://reference.sitepoint.com/css)
Terms of Use (/legals)
Privacy Policy (/legals/#privacy)
FAQ (https://sitepoint.zendesk.com/hc/en-us)
Contact Us (mailto:feedback@sitepoint.com)
Contribute (/write-for-us)

Visit

SitePoint Home (/)
Forums (http://community.sitepoint.com)
Newsletters (/newsletter)
Premium (/premium)
References (/sass-reference)
Store (/store)
Versioning (/versioning)

Connect

(/feed/) (/newsletter/) (https://www.facebook.com/sitepoint) (http://twitter.com/sitepointdotcom) (https://plus.google.com/+sitepoint)