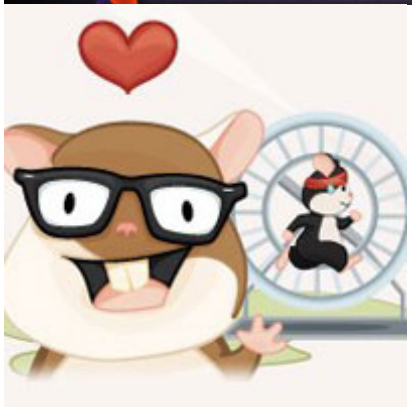

Getting Into Ember: Part 4

Rey Bango on Apr 30th 2013 with 7 Comments

Tutorial Details

-
- **Difficulty:** Intermediate
- **Completion Time:** 1 Hour

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

In my previous tutorial, I touched on how to use `Ember.Object` to define your models and work with datasets. In this section, we'll look more closely at how Ember uses the Handlebars templating framework to define your app's user interface.

Client-side Templates

Most server-side developers are used to using templates to define markup that will be dynamically filled on the fly. If you've ever used ASP.NET, ColdFusion, PHP or Rails then it's pretty much assured you know what I'm talking about.

JavaScript Client–side templating has really taken off of late especially because of the focus on building more desktop–like experiences. This means that more of the processing is done on the client–side with data being mainly pulled via server–side API requests.

I remember writing about client–side templates some time ago when the jQuery Template plugin was first released. Nearly three years later, it's still the most read post on my blog, showing how interest in client–side templating has risen. Since then, a number of other frameworks have been released, offering rich features and supportive communities. Handlebars is one of the more popular options and the framework chosen by the Ember project to power it's templating needs. This makes sense as Handlerbars was created by Ember.js co–founder and core team member, Yehuda Katz. Note, though, that I'm not planning on doing comparisons between templating frameworks and I will strictly focus on Handelbars since this is what Ember.js uses by default.

In the previous articles, I showed some very basic templates in the code:

```
<script type="text/x-handlebars">
    <h2><strong>{{firstName}} {{lastName}}</strong></h2>
</script>
```

Two things that stand out are the type declaration for the script tag and the curly braces which act as delimiters for the expressions that Handlebars will act upon. This is very typical syntax that I'll discuss in more detail soon and you'll use consistently as you build Ember templates.

The Syntax

Despite the fact that Handlebars uses a special syntax, at the end of the day, you're really working primarily with standard HTML markup. Handlebars serves to inject content into this markup to render data to the user. It does this by parsing the delimited expressions and replacing them with the data you've asked Handlebars to work with. In the case of Ember, Handlebars provides the hooks and Ember uses

them. That data typically comes from your controller (remember that controllers serve as the interface to your models).

The first thing any template needs is a script tag definition. Most of you have probably defined script tags to load your JavaScript library. In fact, you've already done this to load Handlebars into your Ember project:

```
<script src="js/libs/jquery-1.9.1.js"></script>
<script src="js/libs/handlebars-1.0.0-rc.3.js"></script>
<script src="js/libs/ember-1.0.0-rc.1.js"></script>
<script src="js/app.js"></script>
```

There's a slight difference with using it to define a template. First, we're specifying a `type` attribute of "text/x-handlebars". This `type` is ignored by the browser but leaves the text available for inspection and allows Ember to identify templates within the app. In addition, Ember uses a data attribute called "data-template-name" which Ember can use to associate specific parts of your application with a template. For example, the following declaration defines a template with a name "employee":

```
<script type="text/x-handlebars" data-template-name="employee">
...
</script>
```

When your application starts, Ember scans the DOM for `type="text/x-handlebars`, compiles the templates it finds, and stores them in a property of the Ember object, called `Ember.TEMPLATES` which it uses to figure out what to render for a given route. This is why following Ember's naming conventions is so important. In the example above, this template will be automatically associated to the employee route and controller you created in your application. Again, I can't stress enough how these naming conventions will make your development much easier.

Ember is reliant on URLs to determine the resources that need to be used and the templates that need to be rendered. Let's imagine that you had a profile page with the URL "/profile". You would have a resource, called `profile` that would load specific resources for that URL (like a route object) and you would also have a template by the same name. We reviewed defining resources and route objects in part 2 of my Ember series so if you're not sure about what I'm discussing, be sure to hop back

there to refresh yourself on this.

When you visit that URL, Ember knows it needs to load these resources and parse the template you've defined. It does this via its naming conventions, knowing that because you went to "/profile" it needs to load the resources defined in the `profile`, and render the template, named `data-template-name="profile"`.

- **Route**: ProfileRoute
- **Controller**: ProfileController
- **Template**: profile (note that it's lowercase)

Going over the naming conventions again, you'll see that the route, controller and template are all tied together using the same URL name with the exception that the template is spelled in lowercase. This is what allows Ember to manage everything behind the scenes without you having to do a lot of wiring up.

Also important to note is that, if you declare a template without a `data-template-name` attribute, Ember will assume that it is the Application-scoped template – the one typically used as a site-wide template for creating user interface elements, such as headers, footers and navigation. And if you don't explicitly define a template for an application or even a resource (e.g: for a URL), Ember does that automatically for you to ensure stability and consistency in your app.

Expressions

The next step is to include your markup and the delimited expressions you'll be using to represent your data. Expressions are delimited, via double curly braces which allow them to be easily identified and parsed with data being passed from your controller. Here's an example:

```
<script type="text/x-handlebars">
    <h2><strong>{{firstName}} {{lastName}}</strong></h2>
</script>
```

In this case, the `{{firstName}}` and `{{lastName}}` expressions will be parsed by Ember

and replaced by actual data. In addition, Ember sets up observers so that as your data changes, your template is automatically updated and the updates reflected to the user of your app.

So far, I've shown you a very simple example, but the takeaway is that:

- Ember uses a special type attribute to define templates.
- Templates use standard markup along with delimited expressions, which are parsed on the client–side.
- These templates have the full feature set capabilities of Handlebars.
- Ember sets up observers to dynamically update your user interface data, as it changes.

This offers a lot of flexibility in how you structure your user interface. Let's continue looking at the features that are available.

Advanced Expressions

Remember that Ember leverages Handlebars, so you have access to its full breadth of expressions here. Conditional expressions are a must, in order to render almost anything useful; Handlebars offers quite a number of options.

Let's say that I had a JSON dataset that looked like this:

```
"items": [{
    "title": "Tearable Cloth Simulation in JavaScript",
    "url": "http://codepen.io/stuffit/pen/KrAwx",
    "id": 5592679,
    "commentCount": 20,
    "points": 127,
    "postedAgo": "1 hour ago",
    "postedBy": "NathanKP"
}, {
    "title": "Netflix now bigger than HBO",
    "url": "http://qz.com/77067/netflix-now-bigger-than-hbo/",
    "id": 5592403,
    "commentCount": 68,
    "points": 96,
    "postedAgo": "2 hours ago",
    "postedBy": "edouard1234567"
}
```

If I wanted to ensure that the `title` data is available, I could add a conditional "if" statement by using the `#if` expression:


```
{{#if item.title}}
    <li>{{item.title}} - {{item.postedAgo}} by {{item.postedBy}}</l
{{/if}}
```

This checks to see if `item.title` is not undefined, and continues processing the subsequent expressions for the `title`, `postedAgo` and `postedBy` data expressions.

Since this dataset contains more than one "record", it's safe to assume that we'd probably want to loop over each element of `item`. That's where the `{{#each}}` expression comes into play. It allows you to enumerate over a list of objects. So, again, keeping in mind that templates are a combination of markup and Handlebars expressions, we can use the `#each` expression to loop through every item available within our Ember model object. Remember that the Ember model is derived from the controller, which is associated to the template, via Ember's naming conventions.

```
<ul>
    {{#each item in model}}
    {{#if item.title}}
        <li>{{item.title}} - {{item.postedAgo}} by {{item.postedBy}}
    {{/if}}
    {{/each}}
</ul>
```

This would render out something similar to:

```
<ul>
<li>Tearable Cloth Simulation in JavaScript - 1 hour ago by NathanK
<li>Netflix now bigger than HBO - 2 hours ago by edouard1234567</li
<li>Fast Database Emerges from MIT Class, GPUs and Student&#39;s In
<li> Connecting an iPad retina LCD to a PC - 6 hours ago by noonesp
</ul>
```

The distinct advantage is Ember's implicit specification of observer,s which will update your data upon an update.

If your conditional expression needs to be more complex, you'll want to create a computed property. This allows you to create a property based off of a method that can apply complex code conditions to your data. Let's say I wanted to solely wanted to display data that had the title "Tearable Cloth Simulation in JavaScript". There's a couple of things I need to setup:

- I need a computed property to scan each item and tell me if the title matches
- I need to create a controller that can be used by each item being enumerated over in the template
- I need to update the template so that it uses this controller for each item

  The first thing I need to do is create the new controller that will wrap each item being looped over and create the computed property within it:

```
App.TitleController = Ember.ObjectController.extend({
    titleMatch: function() {
      return this.get(&#39;title&#39;) === &quot;Tearable Cloth Sim
    }.property()
});
```

Looking at the code, we're subclassing `Ember.ObjectController` to create the controller. This is the controller that will wrap each item being looped over in our template. Next, we're creating a method, called `titleMatch` which uses the `get()` method to pull back the current title, compare it to the text I've defined, and return a boolean. Lastly, the Ember *property()* method is called to define the *titleMatch* method as a computed property.

Once we have this in place, we update the template's `{{#each}}` expression to represent each item with the new controller we created. This is done by using the

itemController directive. A key thing to understand is that `itemController` is a key phrase in Ember meant to associate a controller to items of a template. Don't confuse it for an actual controller name (as I did initially). The controller name is assigned to `itemController`, like this:

```
<ul>
   {{#each item in model itemController="title"}}
      {{#if titleMatch}}
        <li>{{foo.title}} - {{foo.postedAgo}} by {{foo.postedBy}}</
      {{/if}}
    {{/each}}
</ul>
```

Again, naming conventions dictate that, when assigning names in templates, we use lowercase. In this case, we're assigning `TitleController` to `itemController`.

Now, as each item is looped over, the computed property, `titleMatch`, is used to evaluate the title and display data if it matches.

Binding Data to Elements

Creating dynamic templates isn't just about spitting out text. There are times when the look and feel of the UI needs to be affected by data being processed. Displaying an image or building a link are great examples of this.

Binding data to an element requires using special Ember helpers that assist in defining an attribute's context, as well as ensuring that the attributes are updated properly when data changes. For element attributes, the `{{bindAttr}}` helper is used to fill in the values for an attribute. If we needed to dynamically specify the URL of an image, we'd use the following syntax:

```
<img {{bindAttr src="logoUrl"}} alt="Logo">
```

The same can be done for attributes that don't receive a value, such as `disabled`:

```
<input type="checkbox" {{bindAttr disabled="isAdministrator"}}>
```

In this case, `isAdminstrator` could be a computed property based off a method in the

controller, or just a normal object property giving you a lot of flexibility in defining the conditions for disabling the checkbox. This flexibility carries over to defining class names as well. If I wanted to use a conditional statement to define if a class should be applied to my element, I could use the following code:

```
<div {{bindAttr class="isUrgent"}}>
  Warning!
</div>
```

Depending on the boolean state, my markup would either be:

```
<div {{bindAttr class="is-urgent"}}>
  Warning!
</div>
```

for a `true` condition, or:

```
<div>
  Warning!
</div>
```

for a `false` condition. Note that, when I specified `isUrgent` for the class, Ember dasherized the name and rendered the class as `is-urgent`. If you'd prefer to specify your own class based on the results, you can use a conditional expression similar to a ternary statement:

```
<div {{bindAttr class="isUrgent:urgent:normal"}}>
```

This will return `urgent` or `normal` for the class, based on the conditional value of `isUrgent`.

Get to Know Templates

Templates will be the foundation of your user interface, so it's going to be important that you spend time reading the docs at both the Ember and Handlebars site to get a good feel for their overall power. Even if you don't use Ember, Handlebars is a great framework for you to use day–to–day, and worth the investment in learning how to use it.

Gabriel Manricks wrote a great tutorial on Handlebars here on Nettuts+ that you can

use to get up to speed on the framework.

Tags: ember

By Rey Bango

Rey Bango is developer evangelist at Microsoft focused on meeting the needs of the web development community. He's an advocate for cross-browser, standards-based development using JavaScript & HTML5 and a former member of the jQuery Project Team.

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