COEN 168/268

Mobile Web Application Development

Ember Naming Conventions

Peter Bergström (pbergstrom@scu.edu)

Santa Clara University

Ember Uses Naming Conventions For Convenience

- Allows you to wire up your objects without a lot of boilerplate.
- You will want to use these conventional names for your routes, controllers and templates.
- Makes it easy to guess the names
- In the following examples 'App' is a name that we chose to namespace or represent our Ember application

The Application

When your application boots, Ember will look for these objects:

- `App.ApplicationRoute`
- `App.ApplicationController`
- the `application` template

The Application, and `App.ApplicationController`

- Ember.js will render the `application` template as the main template.
- If `App.ApplicationController` is provided, Ember.js will set an instance of `App.ApplicationController` as the controller for the template.
- This means that the template will get its properties from the controller.

The Application, and `App.ApplicationRoute`

- If your app provides an `App.ApplicationRoute`, Ember.js will invoke the route's hooks first, before rendering the `application` template.
- Hooks are implemented as methods and provide you access points within an Ember object's lifecycle to intercept and execute code to modify the default behavior at these points to meet your needs.

Here's a simple example that uses a route, controller, and template:

```javascript
App.ApplicationRoute = Ember.Route.extend({
  setupController: function(controller) {
    // `controller` is the instance of ApplicationController
    controller.set('title', "Hello world!");
  }
});

App.ApplicationController = Ember.Controller.extend({
  appName: 'My First Example'
});
```

```html
<!-- application template -->
<h1>{{appName}}</h1>

<h2>{{title}}</h2>
```

Specify Controllers As **classes**

- Ember.js is responsible for instantiating them and providing them to your templates.
- This makes it super-simple to test your controllers, and ensures that your entire application shares a single instance of each controller.

Simple Routes

Each of your routes will have a controller, and a template with the same name as the route.

Let's start with a simple router:

```
App.Router.map(function() {
  this.route('favorites');
});
```

If your user navigates to `/favorites`, Ember.js will look for these objects:

- `App.FavoritesRoute`
- `App.FavoritesController`
- the `favorites` template

Ember.js will render the `favorites` template into the `{{outlet}}` in the `application` template. It will set an instance of the `App.FavoritesController` as the controller for the template. If your app provides an `App.FavoritesRoute`, the framework will invoke it before rendering the template.

For a route like `App.FavoritesRoute`, you will probably implement the `model` hook to specify what model your controller will present to the template:

```
App.FavoritesRoute = Ember.Route.extend({
  model: function() {
    // the model is an Array of all of the posts
    return this.store.find('post');
  }
});
```

You Do Not Need to Provide a Controller

- On the previous slide, we didn't provide a `FavoritesController`.
- Because the model is an Array, Ember.js will automatically supply an instance of `Ember.ArrayController`, which will present the backing Array as its model.

You can treat the `ArrayController` as if it was the model itself.

This has two major benefits:

- You can replace the controller's model at any time without having to directly notify the view of the change.
- The controller can provide additional computed properties or view-specific state that do not belong in the model layer. This allows a clean separation of concerns between the view, the controller and the model.

The template can iterate over the elements of the controller:

```
<ul>
{{#each controller}}
  <li>{{title}}</li>
{{/each}}
</ul>
```

Dynamic Segments

If a route uses a dynamic segment (a URL that includes a parameter), the route's model will be based on the value of that segment provided by the user.

Consider this router definition:

```
App.Router.map(function() {
  this.resource('post', { path: '/posts/:post_id' });
});
```

In this case, the route's name is `post`, so Ember.js will look for these objects:

- `App.PostRoute`
- `App.PostController`
- the `post` template

Your route handler's `model` hook converts the dynamic `:post_id` parameter into a model. The `serialize` hook converts a model object back into the URL parameters for this route.

The PostRoute

```
App.PostRoute = Ember.Route.extend({
  model: function(params) {
    return this.store.find('post', params.post_id);
  },

  serialize: function(post) {
    return { post_id: post.get('id') };
  }
});
```

Because this pattern is so common, it is the default for route handlers.

- If your dynamic segment ends in `_id`, the default `model` hook will convert the first part into a model class on the application's namespace (`post` becomes `App.Post`).
- It will then call `find` on that class with the value of the dynamic segment.
- The default behaviour of the `serialize` hook is to replace the route's dynamic segment with the value of the model object's `id` property.

Route, Controller and Template Defaults

- If you don't specify a route handler for the `post` route (`App.PostRoute`), Ember.js will still render the `post` template with the app's instance of `App.PostController`.
- If you don't specify the controller (`App.PostController`), Ember will automatically make one for you based on the return value of the route's `model` hook. If the model is an Array, you get an `ArrayController`. Otherwise, you get an `ObjectController`.
- If you don't specify a `post` template, Ember.js won't render

Nesting

You can nest routes under a `resource`.

```
App.Router.map(function() {
  this.resource('posts', function() { // the `posts` route
    this.route('favorites');          // the `posts.favorites` route
    this.resource('post');            // the `post` route
  });
});
```

Nesting, Continued

- A **resource** is the beginning of a route, controller, or template name.
- Even though the `post` resource is nested, its route is named `App.PostRoute`, its controller is named `App.PostController` and its template is `post`.
- When you nest a **route** inside a resource, the route name is added to the resource name, after a `.`.
- The rule of thumb is to use resources for nouns, and routes for adjectives (`favorites`) or verbs (`edit`).

The Index Route

At every level of nesting (including the top level), Ember.js automatically provides a route for the `/` path named `index`.

For example, if you write a simple router like this:

```
App.Router.map(function() {
  this.route('favorites');
});
```

It is the equivalent of:

```
App.Router.map(function() {
  this.route('index', { path: '/' });
  this.route('favorites');
});
```

If the user visits `/`, Ember.js will look for these objects:

- `App.IndexRoute`
- `App.IndexController`
- the `index` template

The `index` template will be rendered into the `{{outlet}}` in the `application` template. If the user navigates to `/favorites`, Ember.js will replace the `index` template with the `favorites` template.

A nested router like this:

```
App.Router.map(function() {
  this.resource('posts', function() {
    this.route('favorites');
  });
});
```

Is the equivalent of:

```
App.Router.map(function() {
  this.route('index', { path: '/' });
  this.resource('posts', function() {
    this.route('index', { path: '/' });
    this.route('favorites');
  });
});
```

If the user navigates to `/posts`, the current route will be `posts.index`. Ember.js will look for objects named:

- `App.PostsIndexRoute`
- `App.PostsIndexController`
- The `posts/index` template

First, the `posts` template will be rendered into the `{{outlet}}` in the `application` template. Then, the `posts/index` template will be rendered into the `{{outlet}}` in the `posts` template. If the user then goes to `/posts/favorites`, Ember.js will replace the `{{outlet}}` in `posts` with `posts/favorites`.

The lecture contents is adapted from the Ember Guides available under the MIT license

http://emberjs.com/guides/concepts/naming-conventions/

COEN 168/268

Mobile Web Application Development

Ember Naming Conventions

Peter Bergström (pbergstrom@scu.edu)

Santa Clara University