عنوان: **شروع به کار با Ember.js**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۹/۱۴ ۸:۵
آدرس: www.dotnettips.info
گروه‌ها: JavaScript, jQuery, SPA, EmberJS

Ember.js کتابخانه‌ای است جهت ساده سازی تولید برنامه‌های تک صفحه‌ای وب. برنامه‌هایی که شبیه به برنامه‌های دسکتاپ در مرورگر کاربر عمل می‌کنند. دو برنامه نویس اصلی آن Yehuda Katz که عضو اصلی تیم‌های jQuery و Ruby on Rails است و Tom Dale که ابتدا SproutCore را به وجود آورد و بعدها به Ember.js تغییر نام یافت، هستند.

منابع اصلی Ember.js

پیش از شروع به بحث نیاز است با تعدادی از سایت‌های اصلی مرتبط با Ember.js آشنا شد:
سایت اصلی: http://emberjs.com
مخزن کدهای آن: https://github.com/emberjs
انجمن اختصاصی پرسش و پاسخ: http://discuss.emberjs.com
موتور قالب‌های آن: http://handlebarsjs.com
لیست منابع مطالعاتی مرتبط مانند ویدیوهای آموزشی و لیست مقالات موجود: http://emberwatch.com
و بسته‌ی نیوگت آن: https://www.nuget.org/packages/EmberJS

مفاهیم پایه‌ای Ember.js

شیء Application

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

یک برنامه‌ی Ember.js با تعریف وهله‌ای از شیء Application آن آغاز می‌شود. با اینکار به صورت خودکار رویدادگردان‌هایی به صفحه اضافه می‌شوند. کامپونت‌های پیش فرض آن ایجاد شده و همچنین قالب اصلی برنامه رندر می‌شود.

مسیر یابی

با مرور قسمت‌های مختلف برنامه توسط کاربر، نیاز است حالات برنامه را مدیریت کرد؛ اینجا است که کار قسمت مسیریابی شروع می‌شود. مسیریابی، منابع مورد نیاز جهت آدرس‌های مشخصی را تامین می‌کند.

```
App.Router.map(function() {
    this.resource('accounts'); // takes us to /accounts
    this.resource('gallery'); // takes us to /gallery
});
```

در اینجا نحوه‌ی تعریف آغازین مسیریابی Ember.js را مشاهده می‌کنید که توسط متد resource آن مسیرهای قابل ارائه توسط برنامه مشخص می‌شوند.
به این ترتیب مسیرهای accounts/ و gallery/ قابل پردازش خواهند شد.

این مسیرها، تو در تو نیز می‌توانند باشند. برای مثال:

```
App.Router.map(function() {
    this.resource('news', function() {
        this.resource('images', function () { // takes us to /news/images
            this.route('add');// takes us to /news/images/add
        });
    });
});
```

به این ترتیب نحوه‌ی تعریف مسیریابی آدرس news/images/add را مشاهده می‌کنید. همچنین در این مثال از دو متد resource و route استفاده شده‌است. از متد resource برای حالت تعریف اسامی استفاده کنید و از متد route برای تعریف افعال و تغییر دهنده‌ها. برای نمونه در اینجا فعل افزودن تصاویر با متد route مشخص شده‌است.

مدل‌ها

مدل‌ها همان اشیایی هستند که برنامه مورد استفاده قرار می‌دهد و می‌توانند یک آرایه‌ی ساده و یا اشیاء JSON دریافتی از وب سرور باشند.

حداقل به دو روش می‌توان مدل‌ها را تعریف کرد:

الف) با استفاده از افزونه‌ی Ember Data

ب) با کمک شیء Ember.Object

```
App.SiteLink = Ember.Object.extend({});
App.SiteLink.reopenClass({
    findAll: function() {
        var links = [];
        //… $.getJSON …

        return links;
    }
});
```

ابتدا یک زیرکلاس از Ember.Object به کمک متد extend ایجاد خواهد شد. سپس از متد توکار reopenClass برای توسعه‌ی API کمک خواهیم گرفت.

در ادامه متد دلخواهی را ایجاد کرده و برای مثال آرایه‌ای از اشیاء دلخواه جاوا اسکریپتی را بازگشت خواهیم داد.

پس از تعریف مدل، نیاز است آن‌را به سیستم مسیریابی معرفی کرد:

```
App.GalleryRoute = Ember.Route.extend({
    model: function() {
        return App.SiteLink.findAll();
    }
});
```

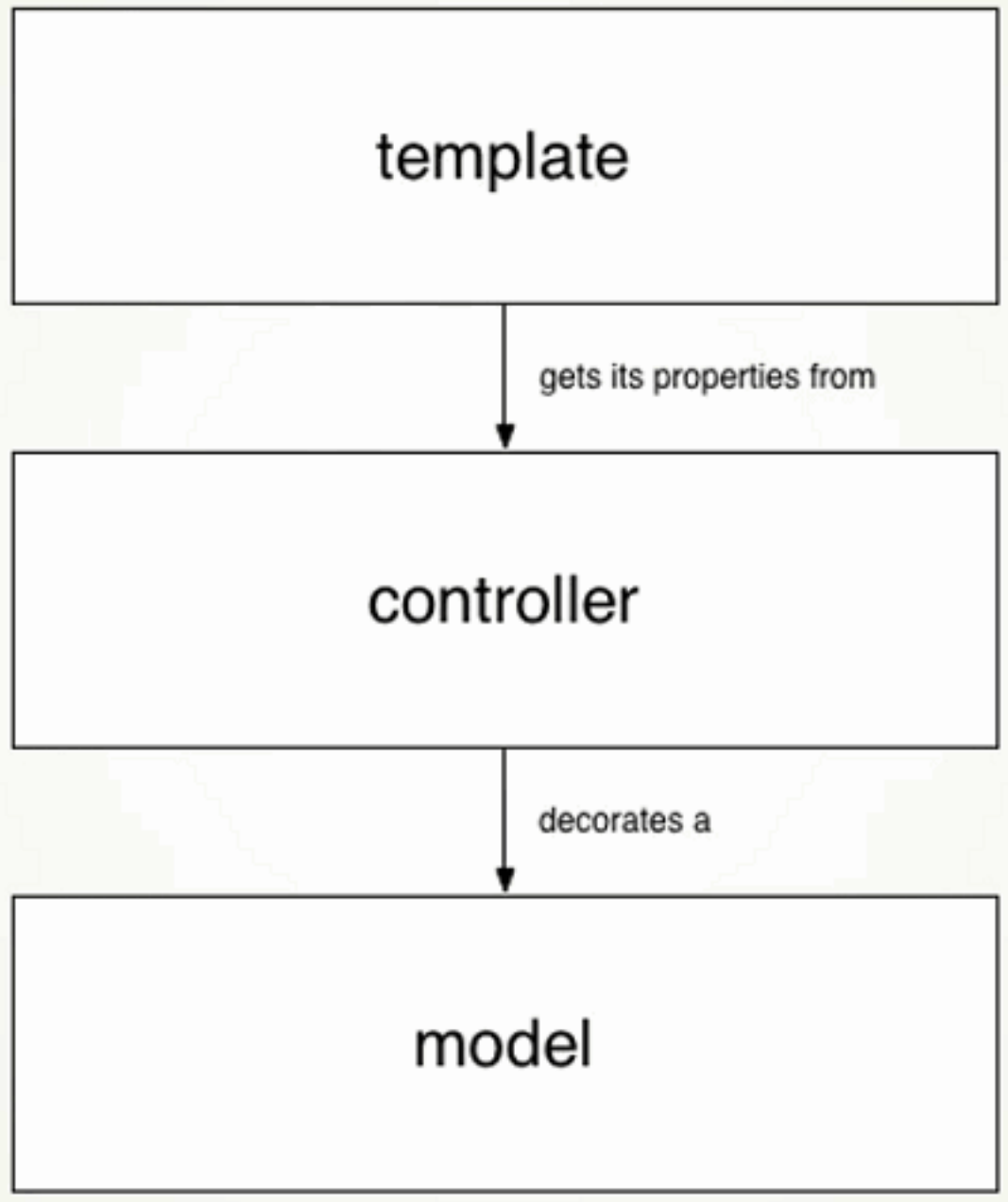
به این ترتیب زمانیکه کاربر به آدرس gallery/ مراجعه می‌کند، دسترسی به model وجود خواهد داشت. در اینجا model یک واژه‌ی کلیدی است.

کنترلرها

کنترلرها جهت ارائه‌ی اطلاعات مدل‌ها به View و قالب برنامه تعریف می‌شوند. در اینجا همیشه باید بخاطر داشت که model تامین کننده‌ی اطلاعات است. کنترلر جهت در معرض دید قرار دادن این اطلاعات، به View برنامه کاربرد دارد و مدل‌ها هیچ اطلاعی از وجود کنترلرها ندارند.

کنترلرها علاوه بر اطلاعات model، می‌توانند حاوی یک سری خواص و اشیاء صرفا نمایشی که قرار نیست در بانک اطلاعاتی ذخیره شوند نیز باشند.

در Ember.js قالب‌ها (templates) اطلاعات خود را از کنترلر دریافت می‌کنند. کنترلرها اطلاعات مدل را به همراه سایر خواص نمایشی مورد نیاز در اختیار View و قالب‌های برنامه قرار می‌دهند.



برای تعریف یک کنترلر می‌توان درون شیء مسیریابی، با تعریف متد setupController شروع کرد:

```
App.GalleryRoute = Ember.Route.extend({
    setupController: function(controller) {
        controller.set('content', ['red', 'yellow', 'blue']);
    }
});
```

در این مثال یک خاصیت دلخواه به نام content تعریف و سپس آرایه‌ای به آن انتساب داده شده‌است.

روش دوم تعریف کنترلرها با ایجاد یک زیر کلاس از شیء Ember.Controller انجام می‌شود:

```
App.GalleryController = Ember.Controller.extend({
    search: '',
    content: ['red', 'yellow', 'blue'],
    query: function() {
        var data = this.get('search');
        this.transitionToRoute('search', { query: data });
```

```
    }
});
```

قالب‌ها یا templates

قالب‌ها قسمت‌های اصلی رابط کاربری را تشکیل خواهند داد. در اینجا از کتابخانه‌ای به نام handlebars برای تهیه قالب‌های سمت کاربر کمک گرفته می‌شود.

```
<script type="text/x-handlebars" data-template-name="sayhello">
    Hello,
    <strong>{{firstName}} {{lastName}}</strong>!
</script>
```

این قالب‌ها توسط تگ اسکریپت تعریف شده و نوع آن‌ها text/x-handlebars مشخص می‌شود. به این ترتیب Ember.js، این قسمت از صفحه را یافته و عبارات داخل {{}} را با مقادیر دریافتی از کنترلر جایگزین می‌کند.

```
<script type="text/x-handlebars" data-template-name="sayhello">
    Hello,
    <strong>{{firstName}} {{lastName}}</strong>!

    {{#if person}}
    Welcome back,
    <strong>{{person.firstName}} {{person.lastName}}</strong>!
    {{/if}}

    <ul>
        {{#each friend in friends}}
        <li>
            {{friend.name}}
        </li>
        {{/each}}
    </ul>

    <img {{bindAttr src="link.url" }} />
    {{#linkTo ''about}}About{{/linkTo}}
</script>
```

در این مثال نحوه‌ی تعریف عبارات شرطی و یا یک حلقه را نیز مشاهده می‌کنید. همچنین امکان اتصال به ویژگی‌هایی مانند src یک تصویر و یا ایجاد لینک‌ها را نیز دارا است.
بهترین مرجع آشنایی با ریز جزئیات کتابخانه‌ی handlebars، مراجعه به سایت اصلی آن است.

قواعد پیش فرض نامگذاری در Ember.js

اگر به مثال‌های فوق دقت کرده باشید، خواصی مانند GalleryController و یا GalleryRoute به شیء App اضافه شده‌اند. این نوع نامگذاری‌ها در ember.js بر اساس روش convention over configuration کار می‌کنند. برای نمونه اگر مسیریابی خاصی را به نحو ذیل تعریف کردید:

```
this.resource('employees');
```

شیء مسیریابی آن App.EmployeesRoute
کنترلر آن App.EmployeesController
مدل آن App.Employee
View آن App.EmployeesView
و قالب آن employees
بهتر است تعریف شوند. به عبارتی اگر اینگونه تعریف شوند، به صورت خودکار توسط Ember.js یافت شده و هر کدام با مسئولیت‌های خاص مرتبط با آن‌ها پردازش می‌شوند و همچنین ارتباطات بین آن‌ها به صورت خودکار برقرار خواهد شد. به این ترتیب برنامه نظم بهتری خواهد یافت. با یک نگاه می‌توان قسمت‌های مختلف را تشخیص داد و همچنین کدنویسی پردازش و

اتصال قسمت‌های مختلف برنامه نیز به شدت کاهش می‌یابد.

تهیه‌ی اولین برنامه‌ی Ember.js

تا اینجا نگاهی مقدماتی داشتیم به اجزای تشکیل دهنده‌ی هسته‌ی Ember.js. در ادامه مثال ساده‌ای را جهت نمایش ساختار ابتدایی یک برنامه‌ی Ember.js، بررسی خواهیم کرد.

بسته‌ی Ember.js را همانطور که در قسمت منابع اصلی آن در ابتدای بحث عنوان شد، می‌توانید از سایت و یا مخزن کد آن دریافت کنید و یا اگر از VS.NET استفاده می‌کنید، تنها کافی است دستور ذیل را صادر نمائید:

```
PM> Install-Package EmberJS
```

پس از اضافه شدن فایل‌های js آن به پوشه‌ی Scripts برنامه، در همان پوشه، فایل جدید Scripts\app.js را نیز اضافه کنید. از آن برای افزودن تعاریف کدهای Ember.js استفاده خواهیم کرد.

در این حالت ترتیب تعریف اسکریپت‌های مورد نیاز صفحه به صورت ذیل خواهند بود:

```
<script src="Scripts/jquery-2.1.1.js" type="text/javascript"></script>
<script src="Scripts/handlebars.js" type="text/javascript"></script>
<script src="Scripts/ember.js" type="text/javascript"></script>
<script src="Scripts/app.js" type="text/javascript"></script>
```

کدهای ابتدایی فایل app.js جهت وهله سازی شیء Application و سپس تعریف مسیریابی صفحه‌ی index بر اساس روش convention over configuration به همراه تعریف یک کنترلر و افزودن متغیری به نام content به آن که با یک آرایه مقدار دهی شده‌است:

```
App = Ember.Application.create();
App.IndexRoute = Ember.Route.extend({
    setupController:function(controller) {
        controller.set('content', ['red', 'yellow', 'blue']);
    }
});
```

باید دقت داشت که تعریف مقدماتی Ember.Application.create به همراه یک سری تنظیمات پیش فرض نیز هست. برای مثال مسیریابی index به صورت خودکار به نحو ذیل توسط آن تعریف خواهد شد و نیازی به تعریف مجدد آن نیست:

```
App.Router.map(function() {
    this.resource('application');
    this.resource('index');
});
```

سپس برای اتصال این کنترلر به یک template خواهیم داشت:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <script src="Scripts/jquery-2.1.1.js" type="text/javascript"></script>
    <script src="Scripts/handlebars.js" type="text/javascript"></script>
    <script src="Scripts/ember.js" type="text/javascript"></script>
    <script src="Scripts/app.js" type="text/javascript"></script>
</head>
<body>
    <script type="text/x-handlebars" data-template-name="index">
        Hello,
        <strong>Welcome to Ember.js</strong>!
        <ul>
            {{#each item in content}}
            <li>
                {{item}}
            </li>
            {{/each}}
        </ul>
    </script>
```

```
</body>
</html>
```

توسط اسکریپتی از نوع text/x-handlebars، اطلاعات آرایه content دریافت و در طی یک حلقه در صفحه نمایش داده خواهد شد.
مقدار data-template-name در اینجا مهم است. اگر آن‌را به هر نام دیگری بجز index تنظیم کنید، منبع دریافت اطلاعات آن مشخص نخواهد بود. نام index در اینجا به معنای اتصال این قالب به اطلاعات ارائه شده توسط کنترلر index است.

تا همینجا اگر برنامه را اجرا کنید، به خوبی کار خواهد کرد. نکته‌ی دیگری که در مورد قالب‌های Ember.js قابل توجه هستند، قالب پیش فرض application است. با تعریف Ember.Application.create یک چنین قالبی نیز به ابتدای هر صفحه به صورت خودکار اضافه خواهد شد:

```
<body>
    <script type="text/x-handlebars" data-template-name="application">
        <h1>Header</h1>
        {{outlet}}
    </script>
```

outlet واژه‌ای است کلیدی که سبب رندر سایر قالب‌های تعریف شده در صفحه می‌گردد. مقدار data-template-name آن نیز به application تنظیم شده‌است (اگر این مقدار ذکر نگردد نیز به صورت خودکار از application استفاده می‌شود). برای مثال اگر بخواهید به تمام قالب‌های رندر شده در صفحات مختلف، مقدار ثابتی را اضافه کنید (مانند هدر یا منو)، می‌توان قالب application را به صورت دستی به نحو فوق اضافه کرد و آن‌را سفارشی سازی نمود.

کدهای کامل این قسمت را از اینجا می‌توانید دریافت کنید:

EmberJS01.zip

نظرات خوانندگان

نویسنده: نصیری
تاریخ: ۱۳۹۳/۰۹/۱۸ ۱۷:۴۹

با سلام؛ با توجه به قدرت خیلی زیاد AngularJs و نیز اینکه پشتیبانی اون داره از طرف تیم قدرتمندی در google اداره میشه و دلیل بعدی اینکه تعداد خیلی زیادی از برنامه نویسهای دنیا به سمت این فریمورک رفتن بهتر نیست یادگرفتن و دنبال AngularJs رفتن را به EmberJs ترجیح داد
با تشکر

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۹/۱۸ ۱۸:۱۸

- صرف نوشتن مطلبی در مورد موضوعی خاص، به معنای «بهترین بودن آن» و «برترین حالت ممکن» نیست. امروز راجع به ember.js مطلب نوشتم. شاید هفته‌های بعد در مورد meteor.js مطلب نوشتم.
- مقایسه‌ای در اینجا « AngularJS vs Ember »
- مقایسه‌ای کامل‌تر در اینجا « AngularJs vs EmberJs »
- « Rails JS frameworks: Ember.js vs. AngularJS »
- « AngularJS vs. Backbone.js vs. Ember.js »
- یک نمونه‌ی دیگر « The Top 10 Javascript MVC Frameworks Reviewed »
- این نظرسنجی را هم دنبال کنید: « آیا به یادگیری یا ادامه‌ی استفاده از AngularJS خواهید پرداخت؟ »