Ember.js 的初学者指南

2012年08月23日 周裕波 JavaScript

现在，到处都可以看到复杂的 JavaScript 应用程序。 由于这些应用程序变得越来越复杂，一长串的 jQuery回调语句，或者通过应用程序在各个点执行不同的函数调用，这些都变得无法再让人接受。 这导致了JavaScript开发人员了解到传统的软件程序员已经知道了几十年的问题： 组织和效率非常重要，并且可以对应用程序的性能是否优异产生重大影响。

实现组织和效率的其中一个最常用的架构模式，被称为 Model View Controller （缩写为 MVC） 。这种模式鼓励开发人员将其应用程序的不同部分分割为更易于管理的块。 您不必使用一个函数直接调用数据库，您可以创建了一个 Model（模型）来为您管理数据库。 您不必使用一个布满输出和逻辑语句的 HTML文件，一个简单的模板或 View（视图）就可以您简化显示代码。最后，Controller（控制器）管理您的应用程序的流，帮助各种零散的部件更高效地互相沟通。 在您的应用程序中使用这个模式，可以更轻松地增加新的功能。

作为最近爆发的基于 Internet 的软件开发的一部分，出现了一堆令人眼花缭乱的 MVC 框架，比如 Ember.js、Backbone.js、Knockout.js、Spine.js、Batman.js 和 Angular.js。 一方面是初级和中级开发人员，另一方面是骨灰级程序员，以 JavaScript 编写并针对JavaScript开发而设计的这些库补充了这两者之间的空白。它们提供多种特性和功能，根据开发人员的需求满足技能水平各异的不同开发人员。

在本教程中，您将通过构建一个可用的 Twitter 时间轴查看器，更熟悉 Ember.js。

**Ember.js 简介**

Ember.js 是 JavaScript 框架包中最新的成员之一。 它演变出了最初于 2007 年创建的 SproutCore 项目，Apple 在包括 MobileMe 在内的各种 web 应用程序中大量使用了该项目。 在[emberjs.com](http://emberjs.com/)，Ember 被形容为 "一个 JavaScript 框架，用于创建可以消除样板并提供标准应用程序架构的大型 web 应用程序。" 它本身紧密集成了名为 Handlebars 的模板引擎，该引擎为 Ember 提供了其中一个最强大的功能： 双向数据绑定。 Ember 还提供了其他功能，比如状态管理（某个用户状态是已注销还是已登录）、自动更新模板（当底层数据发生变化时，您的 UI 也同样发生变化）以及计算属性 (firstName + lastName = fullName)。Ember经过一年可靠的开发后，已经成为一个强大的参与者。

Ember 只有一个依赖项—jQuery。 Ember 应用程序的样板 HTML 设置看起来应该与下面的代码类似。 请注意，jQuery 和 Ember 都从 CDN（内容交付网络）进行更新。如果用户在早些时候访问需要这些文件的其他网站时已经下载过这些文件，这会加快用户的页面加载速度。

<html>

<head>

<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js"></script>

<script src="http://cloud.github.com/downloads/emberjs/ember.js/ember-0.9.6.min.js"></script>

<script src="js/app.js"></script>

</head>

<body>

</body>

</html>

**定义 MVC**

在您继续本教程之前，更明确地定义 MVC可能会是一个好主意。这个概念在1979年已经出现，自那时以来，该模式已经出现一些不同的变体。 最常见的流程通常是这样的：

1. 用户执行一个操作，比如敲击键盘或单击鼠标按钮。
2. 控制器接收输入并触发一个消息给模型。
3. 模型根据消息修改其内容（删除一行或更新购物车数量）。
4. 视图监视模型中的变更，并相应地更新用户界面。

了解 MVC 模式的工作方式，可以使您的应用程序流动变得更简单。此外，由于代码被分割成不同的块，开发人员团队能够更轻松地协同工作，且不互相干扰。

**Ember 如何执行 MVC**

JavaScript 是一个灵活而强大的语言，但它也有不足之处。 它不提供那种使自己适合于 MVC 风格开发的开箱即用的功能。 因此 Ember 利用一批额外功能扩展了该基本语言。 在构建 Ember 应用程序时，您会使用四个主要部件：Application（应用程序）、Model（模型）、View（视图）和 Controller（控制器）。 以下各节将回顾每个部件。

**Application（应用程序）**

每个 Ember 应用程序都需要一个 Ember.Application实例。这是代码其余所有部分的基础，它提供有用的功能，以及名称空间（对应用程序的其余部件进行分组的一种方式）。 定义一个 Ember 应用程序很简单：

Songs = Ember.Application.create({

mixmaster: 'Andy'

});

该代码定义一个名为 Songs 的应用程序，将其名为 mixmaster 的属性设置为 Andy 。 您可以将应用程序的名称改为您喜欢的任何名称，但 Ember 要求变量的名称以一个大写字母开始，以便绑定的系统可以找到它。 在创建应用程序时还可以添加其他内置选项，并且您也可以添加任意的属性或方法，但初学者用户主要关心的可能是 ready() 方法。 该方法的工作方式与 jQuery 的 document.ready() 块完全一样，并且能够通过以下方式实现：

Songs = Ember.Application.create({

mixmaster: 'Andy',

totalReviews: 0,

ready: function(){

alert('Ember sings helloooooooooo!');

}

});

**Models（模型）**

如果没有数据，应用程序就没有意义。 Ember 使用 Models 帮助开发人员以结构化的方式管理数据。 除了保存数据之外，Ember Models 也对其内部的数据进行建模。 换句话说，如果您想储存有关 MP3 集合的信息，您的模型可能包含一个标题属性、一个艺术家属性和一个流派属性等。 该模型可能看起来如下所示：

Songs.Song = Ember.Object.extend({

title: null,

artist: null,

genre: null,

listens: 0

});

关于这几行代码，还有几件事情要注意。

* 您马上就可以看到您的应用程序在使用的名称空间。 Songs 是应用程序的名称，而 Song 是模型的名称。
* 当扩展对象时，您是在为此模型的未来实例创建蓝图。 因为这是 *主* 因为这是主对象，所有歌曲将以它为基础，所以它使用一个大写字母。 这些命名约定使得您在将来可以更轻松地分辨正在使用的对象类型。
* 在创建模型时，您可以为每个属性提供默认值。 title、 artist 和 genre 属性可在以后填写，所以被标记为 null （或无）。 listens 属性默认为 0 ，并且它的值将在您听音乐收藏时增加。

现在， Song 模型已到位，您可以添加第一首歌曲。 您使用了 extend 来初始化 Song 模型，但您将使用 create 来添加它的一个实例。 下面是它的样子：

mySong = Song.create({

title: 'Son of the Morning',

artist: 'Oh, Sleeper',

genre: 'Screamo'

});

请注意，变量没有以一个大写字母开始，那是因为它是 Song 模型的一个实例。 新的歌曲也不在 Songs 名称空间中。 您将几乎不再需要在您的应用程序中创建模型的实例。 您这样做当然没问题，但一般来说，您会将模型的每个实例放置在相近对象的较大集合中，比如 ArrayController（后面会详细介绍）。

**Views（视图）**

在一个 Ember 应用程序或任何 MVC 风格的应用程序中，View（视图）是用户可以看见并与之交互的组件。 通过将原始 HTML 直接添加到页面，可以定义一个内联模板。 该模板将被包含在script 标记中。 您可以将它添加到页面中您希望显示内容的任意位置。

<script type="text/x-handlebars">

Hello <b>{{Songs.mixmaster}}</b>

</script>

请注意， script 标记的类型是 text/x-handlebars 。 这使 Ember 在加载页面时，有一些东西可以抓取。 Ember 自动准备在这个脚本标记内所包含的任何 HTML，以便在您的应用程序中使用。 将这几行代码放在您的应用程序中，就会显示以下文本：

Hello <b>Andy</b>

在继续之前，先深入了解一下。 在您的浏览器中，右键单击粗体文本，并使用浏览器的开发工具检查它。 您可能注意到一些额外的元素。 为了知道在一个基本的属性发生变化时，要更新哪一部分的 HTML，Handlebars 将插入带有唯一 ID 的标记元素；例如：

<b>

<script id="metamorph-0-start" type="text/x-placeholder"></script>

Andy

<script id="metamorph-0-end" type="text/x-placeholder"></script>

</b>

您可以直接在 JavaScript 中定义一个视图，然后使用视图辅助程序将它显示到页面中。 Ember 有通用视图，可以在应用程序中创建简单的 div 标记，但它还配有预打包的一组视图，用于构建基本控件，比如文本输入框、复选框和选择列表。 从在 JavaScript 文件中定义简单的TextArea 视图开始。

Songs.ReviewTextArea = Ember.TextArea.extend({

placeholder: 'Enter your review'

});

然后，通过引用包含该视图的变量的、以单词 view 开头的*路径*，将它显示到页面。 运行下面的代码，在您的浏览器中显示 TextArea 字段，其占位符文本为 "Enter your review"。 您也可以在定义中指定 rows 和 cols 作为额外的属性。

<script type="text/x-handlebars">

{{view Songs.ReviewTextArea}}

</script>

**Handlebars**

现在，您可能想知道代码中的 {{ 和 }} 表示什么，我们正好可以谈谈 Handlebars，也称为*mustaches*。 稍微思考一下，您就会明白它们为什么被称为 Handlebars pard'ner。 Handlebars 是一个模板引擎，让开发人员可以混合原始 HTML 和 Handlebars *表达式* ，生成渲染的HTML。 表达式以 {{ 开始，并以 }} 结束。 如前所述，所有模板必须放在类型为 text/x-handlebars 的 script 标记内。

默认情况下，handlebars 内所包含的任何值据称都会绑定到它的值。 这意味着，如果因为应用程序内的一些其他操作使该值发生变化，显示给用户的值也将更新。 考虑以下代码：

<script type="text/x-handlebars">

My songs have {{Songs.totalReviews}} reviews.

</script>

第一次初始化您的应用程序时，用户将看到以下文本。

My songs have 0 reviews.

但是，凭借数据绑定，随着因更新 Songs.totalReviews 而添加的更多评论，该值将实时改变。

Handlebars 还通过使用 {{#if}} 和 {{else}} . 支持流控制。 这些元素使您可以根据应用程序中的值实现模板的 *条件化* 。 您可以修改前面的示例，在没有任何评论时向用户显示另一条消息：

<script type="text/x-handlebars">

{{#if Songs.totalReviews}}

Read all my reviews!

{{else}}

There are no reviews right now.

{{/if}}

</script>

如果在应用程序的生命周期中的任意时点， Songs.totalReviews 值发生了变化，该视图将更新并显示另一部分的消息。 值得注意的还有， # 和 / 符号只是为了告诉 Handlebars，这个特定的视图辅助程序有一个闭合标记。

**Controllers（控制器）**

此前，Model（模型）被定义为一种使开发人员能够管理数据的方法。 这没错，但这只是一种狭义的定义。 一个模型只包含与单一事物有关的数据；例如，一首歌曲（但不是多首歌曲）或一个人（但不是多个人）。 当您想管理多个相同类型的数据块时，您需要一个 Controller（控制器）。 有了 Ember，您可以使用 ArrayController 来管理多组歌曲、人员、部件或任何东西。 每个 ArrayController 都有内置的 content 属性，用于存储数据。 该数据可以是简单的字符串，也可以是复杂的值，比如数组或对象。 此外，ArrayController 中包含的函数可以用于与在 ArrayController 中所包含的数据进行交互。 您的 Song 集合的 ArrayController 看起来会是什么样呢？

Songs.songsController = Ember.ArrayController.create({

content: [],

init: function(){

// create an instance of the Song model

var song = Songs.Song.create({

title: 'Son of the Morning',

artist: 'Oh, Sleeper',

genre: 'Screamo'

});

this.pushObject(song);

}

});

init 函数不是必需的，但它很方便，因为 songsController 一旦就绪，就会触发 init 函数。 它可以用来将现有数据填充到控制器，在本例中，您将使用它来将一首歌曲添加到控制器中，以演示 Ember 的数据绑定。 添加之前的 ArrayController 定义和以下内联模板，并在您的浏览器中运行代码：

<script type="text/x-handlebars">

{{#each Songs.songsController}}

<h3>{{title}}</h3>

<p>{{artist}} - {{genre}}</p>

{{/each}}

</script>

Handlebars each 辅助程序收到一组数据的路径，然后对它进行循环。 与控制器中每一项分别匹配的各个 each 块中的一切都将显示在页面上。 请注意，您没有提供直接访问内容数组的路径，因为就 Ember 而言，控制器 *就是* 数组。所生成的 HTML 输出如下所示：

<h3>Son of the Morning</h3>

<p>Oh, Sleeper - Screamo</p>

**全面整合： EmberTweets**

此时，您应该已较好地了解了 Ember 是什么，以及它可以做什么。您也应该了解使 Ember 实现其能力的每个组件： Application（应用程序）、Model（模型）、View（视图）和 Controller（控制器）。 现在是时候应用这些知识了，以编写一个真实可用的应用程序。 您将跳过行业标准的 "todo 应用程序" ，并转移到对许多人来说更亲近和熟悉的： Twitter。 在本教程的剩余部分，您将构建一个 Twitter 时间轴查看器。 在编写任何代码之前， [看看最终结果](http://andymatthews.net/code/emberTweets)可能会有帮助。

**创建样板文件**

使用本文开头的样板 HTML 页面，您将首先构建基础 HTML。 复制下面的代码，并粘贴到一个名为 index.html 的新 HTML 文件中。您需要引用在本文的示例文件中的 CSS 文件。 示例文件还包含了这个项目的起点，也可以供您随时使用。

<!doctype html>

<html>

<head>

<title>Tweets</title>

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<link rel="stylesheet" href="styles.css">

<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js"></script>

<script src="http://cloud.github.com/downloads/emberjs/ember.js/ember-0.9.6.min.js"></script>

<script src="app.js"></script>

</head>

<body>

<script type="text/x-handlebars">

<div id="frm">

<b>Load Tweets for: </b>

</div>

<div id="content">

<div id="recent">

<h3>Recent Users</h3>

</div>

<div id="tweets">

<h3>Tweets</h3>

</div>

</div>

</script>

</body>

</html>

您可以看到这个应用程序有三个部分： 一个输入字段，允许用户输入一个 Twitter 用户名；时间轴查看器，显示所选 Twitter 用户的 tweet；以及最新的用户列表，它将存储以前的搜索。

搜索框将出现在页面顶部，最新的用户在左侧的栏中，而 tweet 本身将占据页面右侧的绝大部分版面。

下一步，创建名为 app.js 的另一个文件，并添加以下内容。 这些注释可以帮助您保持代码的条理性。 在您的浏览器中加载此页面，并确保没有任何错误。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Application

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Models

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Views

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Controllers

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**应用程序初始化**

您需要做的第一件事，是初始化您的应用程序。 直接在标记为 Application 的注释块下面，放入以下代码：

App = Em.Application.create();

请注意，此行使用的不是 Ember.Application，而是 Em.Application。在可能要使用 "Em" 任何地方，您都可以使用 "Ember"，Ember 团队通过加入这个方便的快捷方式，减少了要输入的字数。

接下来，您将添加 TextInput 视图和提交按钮。 直接在标记为 "Views" 的注释块下面，放入以下代码：

App.SearchTextField = Em.TextField.extend({

insertNewline: function(){

App.tweetsController.loadTweets();

}

});

此块从使用 App 名称空间开始，然后扩展 Ember 的其中一个预打包的视图 TextField。 除了在 Views 内允许任意属性和函数之外，Ember 也提供了内置的辅助函数。 那就是insertNewLine() 函数；每当光标在输入框中，并且用户在键盘上按 Enter/Return 键时，就会执行该函数。

**创建模板块**

现在已定义了 TextField View，您将要添加相应的视图辅助代码到 HTML 文件中。 切换到 index.html 并在显示为 "Load Tweets for" 的行后面直接添加以下代码。 请记住， {{ 和 }} 内的任何代码都是模板，并且将被 Ember 用于输出数据。 此外，任何以单词 view 开始的模板指的都是已经在您的 JavaScript 代码中定义过的视图。

{{view App.SearchTextField placeholder="Twitter username" valueBinding="App.tweetsController.username"}}

<button {{action "loadTweets" target="App.tweetsController"}}>Go!</button>

这部分的模板包含一个视图辅助程序，以及一个带有 {{action}} 辅助程序的按钮标记。 TextField View SearchTextField 以占位符文本开始，占位符文本是内置在 HTML5 文本输入字段中的一个属性。 如果该字段为空， placeholder 属性中的文本将被放进输入字段。 当有人开始输入时，该值将消失。 Ember 使开发人员可以在其内置视图内使用任何 HTML 5 标准属性。

第二个属性突出显示 Ember 数据绑定的能力。 Ember 使用一组约定来帮助它确定您想完成什么。 视图（一个模板或一个 JavaScript 文件中的视图）中以单词 "Binding" （注意大写字母）结尾的任何属性，自动为它前面的属性设置绑定。 在本例中，Ember 将App.tweetsController.username 的值绑定到输入字段的 value 属性。 每当变量的内容发生变化，输入字段中的值将自动更新，反之亦然。

{{action}} 使得更易于将功能添加到输入驱动的元素。 它有两个选项： 操作名称和目标。 两者综合起来，形成一个 "路径" ，指向 Ember 对象中所包含的函数。 在上面按钮的示例中， "路径" 将是 App.tweetsController.loadTweets() ，当用户在文本字段内按 Enter 键时，也会调用相同的函数。 在您的浏览器中加载 index.html，并单击提交按钮，或在输入字段内按 Enter 键。 如果您查看浏览器控制台，您就会看到一个错误。 这是因为还没有定义App.tweetsController 。

**准备 Tweet 存储对象**

现在是定义 App.tweetsController 的好时机。 在 app.js 中的 Controllers 注释块后面添加以下代码。您应该已经熟悉以下代码。 名称空间、ArrayController、内容数组 —— 一切都在。 但这次您将会添加一个任意属性 ( username ) 和一个函数 ( loadTweets )。 在添加 ArrayController 之后，重新加载您的浏览器。 在输入框中输入一个单词，然后单击按钮。 您会得到一个提示框，回显您所键入的单词。 您可以随时删除提示行。 您还将看到一个错误，表明尚未定义 addUser 方法。

App.tweetsController = Em.ArrayController.create({

content: [],

username: '',

loadTweets: function() {

var me = this;

var username = me.get("username");

alert(username);

if ( username ) {

var url = 'http://api.twitter.com/1/statuses/user\_timeline.json'

url += '?screen\_name=%@&callback=?'.fmt(me.get("username"));

// push username to recent user array

App.recentUsersController.addUser(username);

}

}

});

仔细看看 loadTweets 函数的定义；它有一些陌生的代码。 第一行为函数的其余部分设置一个*范围* 。 根据定义，范围或 *this* 对于所有 Ember 对象来说都是当前函数，在本例中是App.tweetsController 。 但是，在本教程中，稍后您可以将更多功能添加到 loadTweets函数。 现在设置当前范围，有助于 Ember 理解您正在使用的上下文。

如前所述，Ember 提供了许多辅助函数，使编写应用程序变得更容易，这些函数包括 get() 和set() 。 这两个函数内置在每个 Ember 对象中，并提供对任何属性或函数的快速访问。 下一行使用当前对象 App.tweetsController 的范围，然后调用 get() 函数，将您希望获取其值的属性的名称传递进去。 您可能会好奇，在开始时要使用的用户名的值从哪里来。 请记住，Ember 的数据绑定是双向的。 这意味着，一旦您在输入字段中键入了一个值，该输入字段视图的 valueBinding 属性就会用一个值更新 App.tweetsController 对象。

检索到用户名后，会运行一个测试，以确保它不是空的。 此时 if 块内只有两个语句，但这在以后会更改。 第一个语句为一个用户将 URL 设置到 Twitter 的 JSON 文件。 您可能没有马上注意到这里有什么特殊之处，但仔细再看看，您就会发现末尾的 %@ 和 .fmt() 。 .fmt() 函数执行一个方便的字符串替换，使用 %@ 作为标记。 由于应用程序的设计要求存储搜索列表，因此您必须以某种方式存储您的搜索字词。 最后一行执行该函数，将用户名的值推送到App.recentUsersController ArrayController。 因为该对象尚未存在，所以运行代码将产生一个错误。

**存储之前的搜索**

在下面这一节中，您将创建用于存储最近搜索的对象。 使用以下代码并将它添加到App.tweetsController 对象后面。

App.recentUsersController = Em.ArrayController.create({

content: [],

addUser: function(name) {

if ( this.contains(name) ) this.removeObject(name);

this.pushObject(name);

},

removeUser: function(view){

this.removeObject(view.context);

},

searchAgain: function(view){

App.tweetsController.set('username', view.context);

App.tweetsController.loadTweets();

},

reverse: function(){

return this.toArray().reverse();

}.property('@each')

});

您已经熟悉了如何创建 ArrayController 和添加一个空的内容数组，但该对象有一些新的元素，它们以 addUser 函数开始。 这将使用名称为 contains() 的内置 Ember 函数检查现有数组 (this )。 如果它找到一个结果，它会使用 ArrayController 的函数 removeObject() 删除该结果。 与该函数相反，有一个名为 pushObject() 的函数，它用于将单独的对象添加到内容数组。 这两个函数都有处理多个对象的复数版本： pushObjects() 和 removeObjects() 。 此代码在添加搜索词之前首先删除现有搜索词，那么相同的搜索词就不会被多次显示。 既然您已经知道如何从内容数组中删除一个对象， removeUser() 函数中唯一的新元素就是参数。 当使用 {{action}} 辅助程序调用一个函数时，Ember 隐式传递一个引用给当前视图。 在App.tweetsController 的示例中，视图有一个上下文，这基本上是当前被遍历的项。 该上下文用于从数组中删除选定的项。

searchAgain() 函数也将当前视图接收为一个参数。 当用户单击之前搜索过的字词，该函数会用选定的用户名填充 App.tweetsController.username ，然后触发 loadTweets() 函数，为前面的搜索提供一个单击视图。

默认情况下，Ember 按升序将内容显示到页面。 数组索引 1 是第一个，数组索引 2 是第二个，依次类推。 此应用程序的设计要求按降序显示最近的搜索。 这意味着该数组必须被逆转。 虽然这不是内置函数，但您可以看到要添加它是多么容易。 Reverse() 首先使用 EmbertoArray() 函数将 Ember 内容数组转换成一个普通的数组，逆转它，然后返回它。 使得该函数可以被用作一个数据源的，是在末尾添加的 property() 函数。 property() 函数使用一个指定函数所要求的以逗号分隔的属性列表。 在本例中， property() 函数隐式地使用内容数组本身，使用 @each 依赖键对该数组内每个元素进行寻址。 在下一节中，您将看到如何实施reverse() 函数。

**显示之前的搜索**

现在，您已存储了之前的搜索，是时候将它们显示在页面上。 复制以下模板，并将它添加到标签为 Recent Users 的 h3 标记后面。

<ol>

{{#each App.recentUsersController.reverse}}

<li>

<a href="#" title="view again" {{action "searchAgain" target="App.recentUsersController"}}>{{this}}</a> -

<a href="#" title="remove" {{action "removeUser" target="App.recentUsersController"}}>X</a>

</li>

{{/each}}

</ol>

现在，您应该已经熟悉所有这些代码。 each 块指向内容数组，它内部所包含的 HTML 将被应用到 App.recentUsersController 变量中的每一项。 不必显式指向内容数组，但在本例中，该代码指向 reverse 函数，它以反序提供数据。 {{action}} 辅助程序让用户单击每个锚标记，并触发指定的函数。 可能不太熟悉的唯一一个元素是 {{this}} 。 当遍历内容数组时，Ember 在 {{this}} 变量中保存对当前索引的引用。 因为每一项的值都只是一个字符串，您可以使用 {{this}} 直接输出当前项的值。 单击 Twitter 用户名，将再次加载该用户的 tweet，而单击这些 tweet 的名称则将从 recentUsersController 删除它们。

**加载 tweet**

保存搜索词没问题了，但实际执行搜索又如何呢？ 接下来，您将添加从 Twitter 检索 JSON 包的片段，并将它显示到页面。 使用以下 Ember Model 并直接将它添加到标签为 Model 的注释块后面。 请记住，Ember Model 是它们将包含的数据是一个蓝图。

App.Tweet = Em.Object.extend({

avatar: null,

screen\_name: null,

text: null,

date: null

});

在 app.js 中，找到显示为 App.recentUsersController.addUser(username); 的代码行；并在它后面直接添加以下代码：

$.getJSON(url,function(data){

me.set('content', []);

$(data).each(function(index,value){

var t = App.Tweet.create({

avatar: value.user.profile\_image\_url,

screen\_name: value.user.screen\_name,

text: value.text,

date: value.created\_at

});

me.pushObject(t);

})

});

如果您以前使用过 jQuery，您可能已经使用过 .get() 函数来检索数据。 .getJSON() 函数做同样的事情，只是它将 JSON 包视作一个结果。 此外，它使用返回的 JSON 字符串，并将它转换成可执行的 JavaScript 代码。 数据被检索到之后，内容数组被清空，删除所有现有的 tweet。 下一行提取数据包，并将它封装在一个 jQuery 对象中，因此 .each() 方法可以循环所生成的 Tweets。 在 each 块中，使用数据填充了 Tweet Model 的一个副本，然后将其推送到 ArrayController。

最后，您需要将以下显示代码添加到 index.html。直接复制并粘贴它到标签为 Tweets 的 h3标记后面。

<ul>

{{#each App.tweetsController}}

<li>

<img {{bindAttr src="avatar"}} />

<span>{{date}}</span>

<h3>{{screen\_name}}</h3>

<p>{{text}}</p>

</li>

{{/each}}

</ul>

Ember 使用纯 {{Handlebars}} 可以很容易地输出数据到页面，但有一个问题。 还记得 Ember 如何在脚本标记中包装输出的值吗？ 当您使用 HTML 属性时，这并不是一个选项。 所以 Ember 提供 {{bindAttr}} 辅助程序。 在这个辅助程序内放置的任何属性都将如常输出，但仍保留绑定。 继续进行操作，现在运行您的应用程序。 输入一个用户名，会看到 Tweets 出现。