Getting started with HTML mobile application development using jQuery mobile, RequireJS and BackboneJS

Target

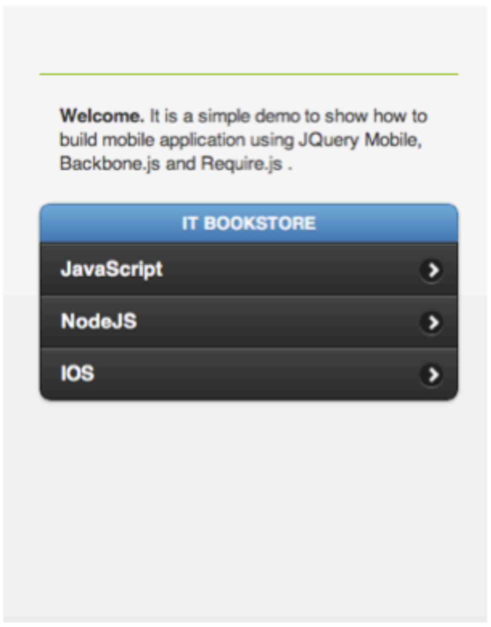
In this introductory tutorial, I will cover BackboneJS, jQuery Mobile and RequireJS, to help web developers to build a modular mobile application. With using PhoneGap , it will be very easy for you to package and deploy the application to multiple platforms.

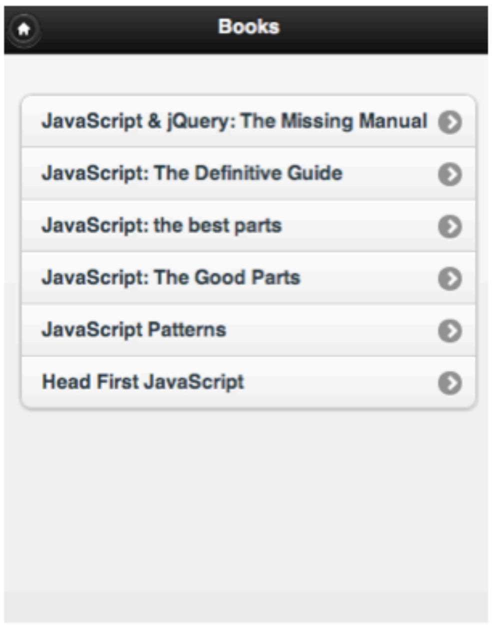
We will build a simple sample application step by step. With this progressive enhanced demo application, we will introduce:

1. how to modularize application using BackboneJS , jQuery Mobile and RequireJS;
2. how to use Backbone View, Collection/Model and Routers.
3. how to decouple mobile view and model/collection using Backbone events.

Sample Application

The demo application has 2 views: Home View and List View.


Welcome. It is a simple demo to show how to build mobile application using JQuery Mobile, Backbone.js and Require.js .


Books
JavaScript & jQuery: The Missing Manual
JavaScript: The Definitive Guide
JavaScript: the best parts
JavaScript: The Good Parts
JavaScript Patterns
Head First JavaScript

JavaScript library & CSS

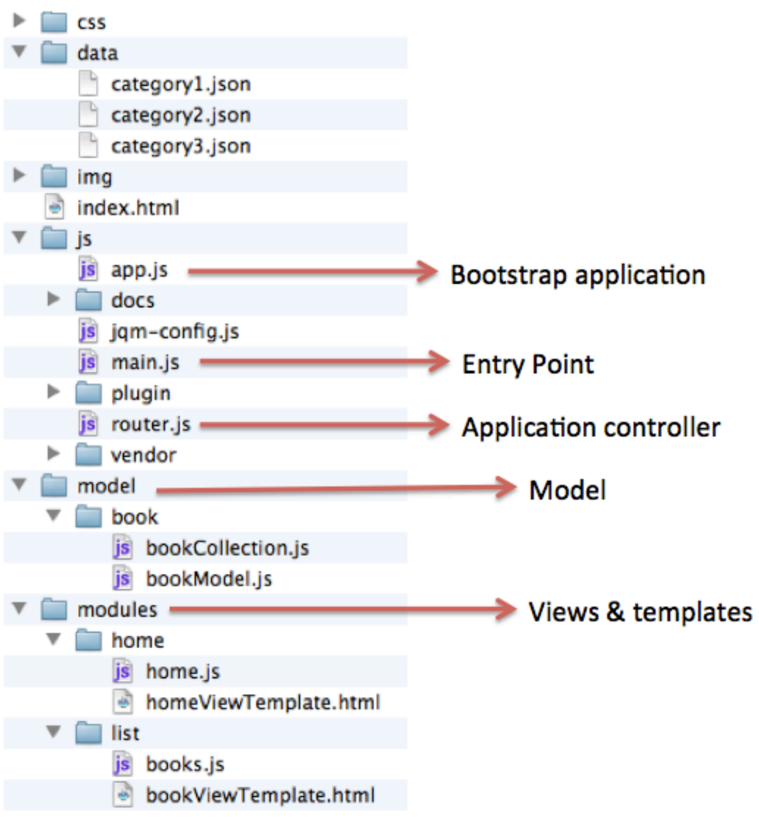
This tutorial will help you start combining jQuery mobile , Backboen.js and Require.js to build an mobile application with separation of concerns. The javascript libraries we will use are listed as below :

- jQuery : http://jQuery.com/
- Require and plugin :
  - require.js : http://requirejs.org/
  - require plugin text.js : http://requirejs.org/docs/download.html#text
- Backbone and Underscore :

UnderscoreJS removed AMD (require.js) support since version 1.3.0. And Backbone.js was affected as well. Fortunately, James Burke maintains an AMD compatible version of underscoreJS and backboneJS. That is why the download links of underscore and backbone below are not pointed to the official sites.

  - amd-underscore : https://github.com/amdjs/underscore
  - amd-backbone : https://github.com/amdjs/backbone
- jQuery Mobile : http://jQuerymobile.com/

Project Structure


```
css
data
    category1.json
    category2.json
    category3.json
img
index.html
js
    app.js
    docs
    jqm-config.js
    main.js
    plugin
    router.js
    vendor
model
    book
        bookCollection.js
        bookModel.js
modules
    home
        home.js
        homeViewTemplate.html
    list
        books.js
        bookViewTemplate.html
```

Idea

In a jQuery Mobile application, each 'view' on the mobile device is a 'page' and is declared with an element using data-role='page' attribute. So in this tutorial, when we say view, it means 'page' of jQuery Mobile. Multiple 'pages' could be organized inside the 'body' of html page and jQuery Mobile will manage them automatically, just like 'multi-page template structure' described in jQuery Mobile doc.

But we all know, loos coupled architecture is better. Using Require.js and Backbone.js, we can divide the 'pages' into separate files as View modules with templates. It allows modular development and testing. It is also easier to extract the model and logic of view into behavior objects.

Let's start building this modular application step by step.

index.html

index.html is just a shell page.

```
<!DOCTYPE html>
<html>
    <head>
        <title>DEMO APPLICATION</title>
        <meta name="viewport" content="width=device-width, initial-scale=1"/>
        <meta http-equiv="Access-Control-Allow-Origin" content="*"/>
        <link rel="stylesheet" href="css/jquery.mobile-1.1.0.css" />
        <link rel="stylesheet" href="css/style.css" />
        <!-- we will use cordova to package the mobile application-->
        <script src="js/vendor/phoneGap/cordova-1.6.0.js"></script>
        <!-- require.js:  data-main attribute tells require.js to load
             js/main.js after require.js loads. -->
        <script data-main="js/main" src="js/vendor/require/require.js"></script>
    </head>
<body>

</body>
</html>

```

Once require.js is loaded, it will take the value of data-main attribute and make a require call. In main.js, we usually configure path settings for require.js.

Entry point: js/main.js

In main.js, we configure require and jQuery mobile and then bootstrap application.

```
require.config({
    //path mappings for module names not found directly under baseUrl
    paths: {
        jquery:     'vendor/jqm/jquery_1.7_min',
        jqm:     'vendor/jqm/jquery.mobile-1.1.0.min',
        underscore: 'vendor/underscore/underscore_amd',
        backbone:   'vendor/backbone/backbone_amd',
        text:       'vendor/require/text',
        plugin:    'plugin',
        templates:  '../templates',
        modules:    '../modules',
        model:       '../model'
    }

});

//1. load app.js,
//2. configure jquery mobile to prevent default JQM ajax navigation
//3. bootstrapping application
define(['app','jqm-config'], function(app) {
    app.initialize();
});

```

Require.js will load and evaluate app.js and jqm.config.js first as the the dependencies of the module. App.js is mapped to app so that we can call app.initialize() to bootstrap Backbone application.

Disable jQuery Mobile AJAX navigation system: jqm-config.js

We all know that there are routing conflicts between jQuery mobile and Backbone.js. There are several ways to workaround it. I like the magic from Christophe Coenraets (Using Backbone.js with jQuery Mobile: http://coenraets.org/blog/2012/03/using-backbone-js-with-jQuery-mobile/) .

In jqm-config.js, we disable the default jQuery Ajax navigation system. Then, use Backbone router to control application and manually call changePage() function to switch between the views. We also remove the hide page from DOM so there is only one view in the DOM every time.

```
define(['jquery'], function($){
  'use strict';
    $(document).bind("mobileinit", function () {
        $.mobile.ajaxEnabled = false;
        $.mobile.linkBindingEnabled = false;
        $.mobile.hashListeningEnabled = false;
        $.mobile.pushStateEnabled = false;
        // Remove page from DOM when it's being replaced
        $('div[data-role="page"]').live('pagehide', function (event, ui) {
            $(event.currentTarget).remove();
        });
    });
});
```

Setup application router: router.js

We use Backbone router as the nav system of the application. Every time user clicks a link, jQuery mobile will change the hash segment which will trigger Backbone to navigate user to the right view thru Backbone router.

Here is router.js. The root url (‘’) is mapped to showHome() function, the same with “/#home” hash.

```
define(['jquery', 'underscore', 'backbone','jqm'],
    function($, _, Backbone) {

    'use strict';
    var Router = Backbone.Router.extend({
    //define routes and mapping route to the function
        routes: {
            '':     'showHome'           //home view
            'home': 'showHome',          //home view as well
            '*actions': 'defaultAction' //default action,mapping "/#anything"
        },

        defaultAction: function(actions){
            this.showHome();
        },

        showHome:function(actions){
            // will render home view and navigate to homeView
        }
    });

    return Router;
});
```

Bootstrapping application : app.js

app.js will create backbone router object and then expose the function to allow bootstrap backbone application.

```
define(['jquery','underscore', 'backbone', 'router'],
    function($, _, Backbone, Router) {
    'use strict';
    var init=function(){
        //create backbone router
        var router=new Router();
        Backbone.history.start();
    };

    return{
        initialize:init
    }
});
```

Home View

Now, we are ready to render the first view : home view.

1. Define a module for home view: home.js

To render homeView using template, we create a object by extending Backbone.View. Here are homeView codes: js/modules/home/home.js :

```
define(['jquery', 'underscore', 'backbone','text!modules/home/homeViewTemplate.html'],
function($, _, Backbone, homeViewTemplate){

  var HomeView = Backbone.View.extend({

    //initialize template
    template:_.template(homeViewTemplate),

    //render the content into div of view
    render: function(){
      //this.el is the root element of Backbone.View. By default, it is a div.
      //$el is cached jQuery object for the view's element.
      //append the compiled template into view div container
      this.$el.append(this.template());

      //return to enable chained calls
      return this;
    }
  });
  return HomeView;
});

```

text.js, a RequireJS plugin, can help us load text-based template file through 'text!' prefix so we can separate the template from script file. 'modules/home/homeViewTemplate.html' will be loaded automatically and then passed to the module function as the argument "homeViewTemplate".

Inside the module function, we use the template engine of Underscore.js to compile the template, and then append the result html segment into view's container: this.el, which is a div by default. So we have rendered the view but have not inserted it into DOM.

2. Define the template for home view: homeViewTemplate.html

The template for homeView is a static page: modules/home/homeViewTemplate.html

```
<div data-role="content" >
    <div class="content-primary">
        <p class="intro">
            <strong>Welcome.</strong> It is a simple demo to show how to build mobile application using JQuery
            Mobile, Backbone.js and Require.js .
        </p>

        <ul data-role="listview" data-inset="true" >
                <li data-role="list-divider" class="listTitle">IT BOOKSTORE</li>
                <li data-theme="a"><a href="#list/1">JavaScript</a></li>
                <li data-theme="a"><a href="#list/2">NodeJS</a></li>
                <li data-theme="a"><a href="#list/3">IOS</a></li>
        </ul>
    </div><!-- /content -->
</div>

```

As we can see, in our example application, the template of HomeView has no "Header" and "Footer". All contents are place into content div with 'data-role="content"' specified. jQuery Mobile uses HTML5 data- attributes to allow for markup-based initialization and configuration of widgets.

Inside content container, we add a listview with using 'data-role="listview"'. Each item has a hardcoded link which will change the hash segment of the url. For example, "#list/1" , 1 is categoryId, so we will use it to fetch book data from matching json file later. We will add the mapping in the routes of router.js later to allow Backbone to invoke mapping functions to response user's interaction.

3. ShowHome in router.js

We use showHome() to insert the view into DOM and present HomeView.The updated router.js is as below:

```
define(['jquery', 'underscore', 'backbone','modules/home/home','jqm'],
    function($, _, Backbone,HomeView) {

    'use strict';
    var Router = Backbone.Router.extend({
    //define routes and mapping route to the function
        routes: {
            '':    'showHome',           //home view
            'home': 'showHome',          //home view as well
            '*actions': 'defaultAction' //default action,mapping "/#anything"
        },

        defaultAction: function(actions){
            this.showHome();
        },

        showHome:function(actions){
            // will render home view and navigate to homeView
            var homeView=new HomeView();
            homeView.render();
            this.changePage(homeView);
        },

        //1. changePage will insert view into DOM and then call changePage to enhance and transition
        //2. for the first page, jQuery mobile will present and enhance automatically
        changePage:function (view) {
            //add the attribute 'data-role="page" ' for each view's div
            view.$el.attr('data-role', 'page');
            //append to dom
            $('body').append(view.$el);
        }
    });

    return Router;
```

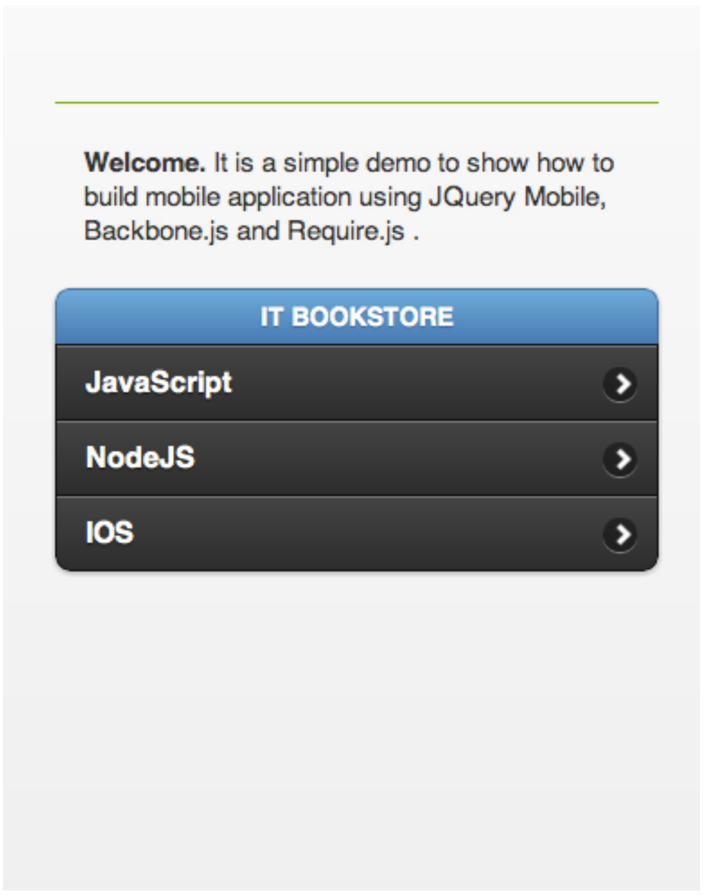
First, add one more dependency for router.js: 'modules/home/home', which is a module defined in home.js; and then pass it to router module function as argument "HomeView".

In showHome function, we create homeView object and render the view content, then pass homeView to changePage function. The changePage(view) is responsible for setting jQuery Mobile data-role attribute of view's root element (view.$el) and appending it into DOM.

Now, we have a html document containing one jQuery mobile 'page' div. This is the first 'page' of our demo application. jQuery mobile will find and enhance the pages in the DOM and transition to the first page automatically once the DOM is ready. So it is not necessary to call jQuery mobile $.mobile.changePage() manually for initial page.

4. Run

Open the browser and run the application by "http://localhost/" (assuming you deploy your application on the root of your web server) , you will see the first view of the application:


Welcome. It is a simple demo to show how to build mobile application using JQuery Mobile, Backbone.js and Require.js .

List View

When user clicks any item in the list, the application will navigate to the second view:book list view, to show the books of the selected category.

1. Prepare json data

To make this demo application as simple as possible, we emulate backend service using local json data. For the list view, we need 3 local json data files mapping to the category item: JavaScript, NodeJS and IOS. The name of json file should follow the format of 'category' + id + '.json', like 'data/category1.json', 'data/category2.json' and 'data/category3.json'.

For example , "category1.json" looks like as following :

```
[
    {
      "id": "1001",
      "name": "JavaScript & jQuery: The Missing Manual "
    },
    {
      "id": "1002",
      "name": "JavaScript: The Definitive Guide"
    },
    {
      "id": "1003",
      "name": "JavaScript: the best parts"
    },
    {
      "id": "1004",
      "name": "JavaScript: The Good Parts"
    },
    {
      "id": "1005",
      "name": "JavaScript Patterns"
    },
    {
      "id": "1006",
      "name": "Head First JavaScript"
    }
]
```

2. Define a module for model (bookModel.js) and collection (bookCollection.js)

Before we get into the list view, we need to define the model of book and the collection.

Book model is very simple. We just extend Backbone.Model and define the default value for the attributes.

```
define(function(){

  var Book=Backbone.Model.extend({
    //default attributes
    defaults:{
      id:"",
      name:'',
      category:''
    }
  });

  return Book;
});
```

Using Book model, we define the collection of book: bookCollection.js

```
define(['jquery', 'underscore', 'backbone','model/book/bookModel'],
       function ($, _, Backbone,Book){

        var Books=Backbone.Collection.extend({

          // Book is the model of the collection
          model:Book,

          //fetch data from books.json using Ajax
          //and then dispatch customized event "fetchCompleted:Books"
          fetch:function(categoryId){
            var self=this;
            var tmpItem;
            //fetch the data using ajax
            var jqxhr = $.getJSON("data/category" + categoryId+".json")
              .success(function(data, status, xhr) {
                $.each(data, function(i,item){
                  //create book for each item and then insert into the collection
                  tmpItem=new Book({id:item.id,category:categoryId,name:item.name});
                  self.add(tmpItem);
                });
                //dispatch customized event
                self.trigger("fetchCompleted:Books");
              })
              .error(function() { alert("error"); })
              .complete(function() {
                    console.log("fetch complete + " + this);
              });
          }
    });

    return Books;
});
```

Add dependency for BookCollection. BookModel is passed to the module function as the argument 'Book'.The item of collection is Book. So we set collection's attribute 'model' as Book.

For bookColllection, we will add a function "fetch" to read the json file and fill into the collection. Once we get the book list successfully, we will trigger a customized event : 'fetchCompleted:Books'. Later, we will bind the event listener on this event in the book list view.

3. Create dynamic template for list view: bookViewTemplate.html

Book list view is also rendered with the template. However, it is a dynamic template and is different with homeView's template.

We use <%...%> to add script and then Underscore template can execute arbitrary JavaScript code inside <% ... %>. In the following template, the variable 'data' will be passed from template function. Of course, you can use any variable name you prefer.

```
<div data-role="header" data-position="fixed">
    <h1>Books</h1>
    <a data-icon="home" data-iconpos="notext" data-direction="reverse">Home</a>
</div>

<div data-role="content">
        <ul data-role="listview" data-inset="true" >
          <!-- data is passed from template engine,
            and templat engine will execute the scripts inside <% %>
          -->
                <% for (var i = 0; i < data.length; i++) { %>
                    <% var item = data[i]; %>
                    <li>
                      <a href="#detail/<%=item.name%>/<%= item.id%>"><%= item.name %></a>
                    </li>
                <% } %>
        </ul>
</div>
```

4. Define a module for list view: book.js

We have already prepared model, collection and template. Now it's time to make a view: book.js

```
define(['jquery', 'underscore', 'backbone', 'text!modules/list/bookViewTemplate.html'],
        function ($, _, Backbone, bookViewTemplate) {
  'use strict';

  var BookListView = Backbone.View.extend({

    template: _.template(bookViewTemplate),

    update:function(categoryId){
      //set callback of the event "fetchCompleted:Books"
      this.collection.bind('fetchCompleted:Books',this.render,this);
      this.collection.fetch(categoryId);
    },

    render: function(){
      this.$el.empty();
      //compile template using the data fetched by collection
      this.$el.append(this.template({data:this.collection.toJSON()}));
      this.trigger("renderCompleted:Books",this);
      return this;
    }
  });

  return BookListView;
});
```

First, add text dependency "bookViewTemplate" .

BookListView has two functions: update(categoryId) and render() .

The update(categoryId) will call collection's fetch function to get book list of selected category by categoryId. Before that, we need to bind the render() function as eventListener for the event "fetchCompleted:Books". Once we get the data successfully, we will render the view with using template.

In the render function, we use Underscore template engine to compile the template and the data to produce the html segments and then insert into view's div container. We also trigger the event "renderCompleted:Books" to notify router that the view is ready and please change page.

In the real case, it is better to cache the template and the view to improve performance.

5. Add routes mapping in router.js

Now back to router.js. We will tell router.js how to "route" application once user clicks category item of HomeView.

The same with other modules, we need to add dependencies first.

```
define(['jquery', 'underscore', 'backbone','modules/home/home',
              'model/book/bookCollection',
              'modules/list/books',
              'jqm'],
    function($, _, Backbone,HomeView,BookCollection,BookListView) {

```

When user clicks any item in the list on the home view, it will change hash segment to "#list/id" . We add the mapping in routes to response user's action : 'list/:category': showBooks .

```
routes: {
    '':     'showHome',           //home view
    'home': 'showHome',          //home view as well
    'list/:category' : 'showBooks',
    '*actions': 'defaultAction' //default action,mapping "/#anything"
},
```

Now, once user clicks the item and changes hash segment of url, Backbone will call showBooks() and pass the category id.

The following codes are what we will add in router.js.

```
init:true,

showBooks:function(categoryId){
    //create a collection
    var bookList=new BookCollection();
    //create book list view and pass bookList as the collection of the view
    var bookListView=new BookListView({collection:bookList});
    //need to pass this as context
    bookListView.bind('renderCompleted:Books',this.changePage,this);
    //update view
    bookListView.update(categoryId);
},

//1. changePage will insert view into DOM and then call changePage to enhance and transition
//2. for the first page, jQuery mobile will present and enhance automatically
//3. for the other page, we will call $.mobile.changePage() to enhance page and make transition
//4. argument 'view' is passed from event trigger
changePage:function (view) {
    //add the attribute 'data-role="page" ' for each view's div
    view.$el.attr('data-role', 'page');
    //append to dom
    $('body').append(view.$el);

    if(!this.init){
        $.mobile.changePage($(view.el), {changeHash:false});
    }else{
        this.init = false;
    }
}
```

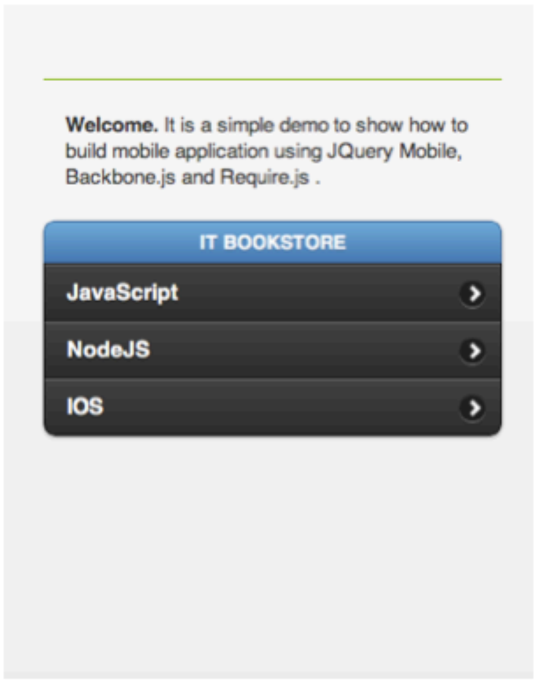
As the codes shows, we add an attribute "init", which is a flag attribute. Like we mentioned before, for the initial page of jQuery Mobile application, jQuery mobile will enhance and present it automatically. So we should not call $.mobile.changePage() manually.

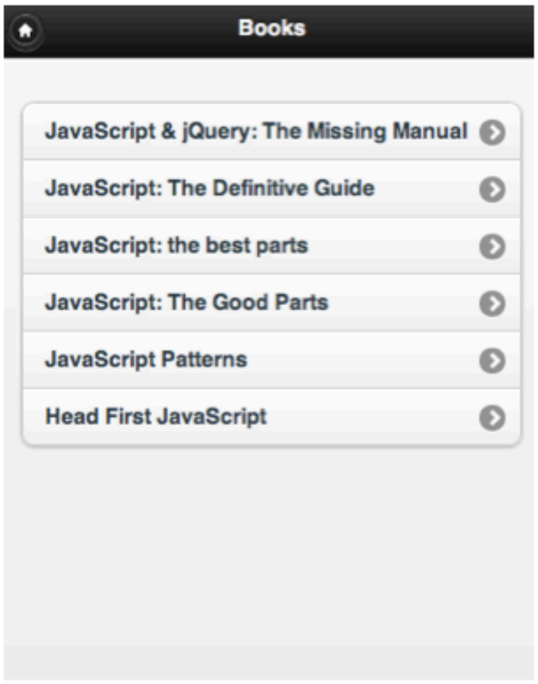
In the function showBooks() , we create BookCollection and bind it with BookListView. Before updating the view (which will call bookCollection's fetch function) , we bind the changePage() function with the event 'renderCompleted:Books'. So when the view is rendered , we will call changePage() to insert it into DOM and then enhance page and transit to the new page.

In the function changePage(), if it is not the initial page, we manually call jQuery mobile function $.mobile.changePage() to load new page and apply transition effect.

6. Run

For now, we can run the demo application and see what we've built:


Welcome. It is a simple demo to show how to build mobile application using JQuery Mobile, Backbone.js and Require.js .



Books
JavaScript & jQuery: The Missing Manual
JavaScript: The Definitive Guide
JavaScript: the best parts
JavaScript: The Good Parts
JavaScript Patterns
Head First JavaScript

Conclusion

It is just a very beginning application and far away from a real one. Actually, there are lots of things we can improve, like caching template and view, or even separating the control logic from router.

RequireJS+BackboneJS+jQuery Mobile is a powerful combination and an easy to use technology. Especially, with requireJS and BackboneJS, we can modularize the application development and make the application clean and clear.

In this introductory tutorial, we've only scratched the surface of jQuery mobile, BackboneJS and requireJS. If you want to learn more, you can go to the official sites of these libraries and frameworks.

Thank you for taking your time to read this tutorial!