jQuery 2013 Webinar 2 - Documentation

Table of Contents

1. Introduction
2. jQuery Plugins
3. Mobile apps
   3.1. Knockout based App with Bootstrap
   3.2. jQuery Mobile based App
   3.3. Kendo UI Mobile based App
4. More

1. Introduction

In the last jQuery webinar we had a closer look on jQuery core. It's time to review the jQuery plugins as well as the jQuery ecosystem. There are numerous frameworks that add missing features to jQuery (e.g. Knockout adds MVVM support) or that are build on top of it. (e.g. Bootstrap and Kendo UI). To make things more interesting, we will concentrate on mobile apps.

2. Plugins

Extending jQuery takes one of two forms:
1. Utility functions defined directly on $
2. Methods to operate on a jQuery wrapped set, called **wrapper methods**

Utility functions are easy to write. They work similar to global functions, but don't pollute the global namespace.

```
(function($) {
    $.say = function(what) {
        alert('I say ' + what);
    };
})(window.jQuery);
```

Methods that are supposed to operate on a jQuery wrapped set should be added to `$.fn`, as shown below:

```
(function($){
    $.fn.changeColor = function() {
        return this.css('color', 'green');
    };
})(window.jQuery);
```

As a plugins get more and more complex, it's a good idea to make it customizable by accepting options. The easiest way do this is with an object literal (called options here) that can, but mustn't be used.

```
(function($){
    $.fn.changeColor = function(options) {

        var settings = $.extend({
            color: "green"
        }, options );

        return this.css('color', settings.color);
    };
})(window.jQuery);
```

Good plugins are unit tested. Johannes will demonstrate in the Live Webinar, how to unit test plugins with the help of the Jasmine framework.

3. Mobile apps

JQuery is a fresh and modern framework. A good proof is the support of many mobile scenarios. Usually a mobile app belongs to one of these tree groups:

1. Native apps
2. Mobile optimized Websites
3. Hybrid apps

Native apps are built for a specific platform with the official platform SDK. Its important to know the specific development languages: Objective-C for iOS, Java for Android and C# for Windows Phone. The are published and sold via the official app store. Usually App stores are the most profitable distribution channel for publishers. On the one hand a closed app system guarantees revenue, but on the other hand it's hard to target different operation systems since specialized developers are required. Due to that fact, for most apps only the two big players (iOS and Android) are targeted.

Mobile optimized websites are just normal server-side apps, built with any server-side technology that renders HTML. The biggest challenges are the styling (it must feel "native" and must fit the the device form factor) and the fact, that a mobile device can be go offline or struggle with a limited connectivity. On the one hand its hard to monetize such an app, but on the other hand its easier to target various platform with the same team of developers, that can be specialized to any server-side programming language and framework.

Hybrid apps try to close the gap between both worlds. They are written with web technologies (HTML, CSS & JavaScript) and do not require specialized knowledge about the targeted platform and its tools. Hybrid apps are shipped with a native executable that provides a browser engine for the web stack as well as an abstraction layer that allows access of native libraries. This abstraction layer is presented as an JavaScript API. The browser engine (and the general performance JavaScript) as well as the amount of abstraction make hybrid apps slower in comparison to native apps. Having that in mind, hybrid platforms offer a suitable way to build apps with open standards that can be published to the app stores!

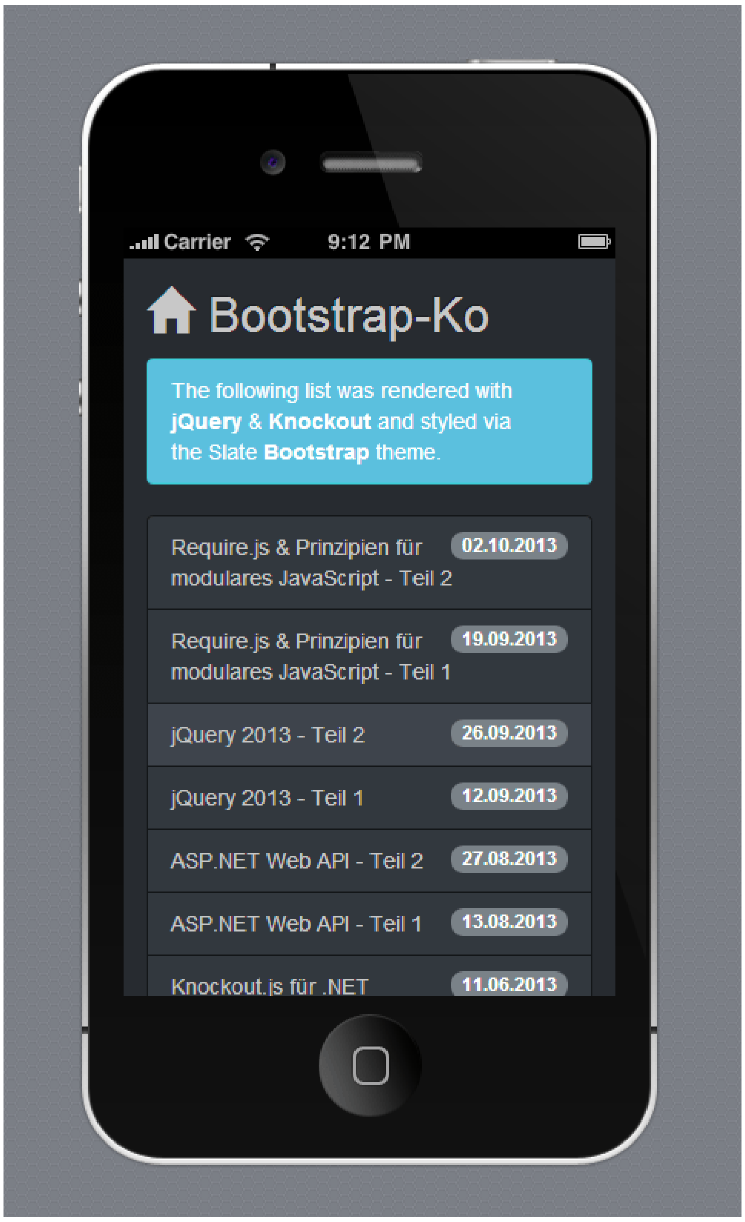
The most common framework for hybrid apps is Apache Cordova. The framework was originally called PhonGap until Adobe made it open source. Now PhoneGap is the name of the most common open source distribution of Cordova with an own app directory. However, as an windows user it is difficulty to create an iOS app. PhoneGap requires you to own a Mac, since tools required for building iOS applications run only on the OS X operating system. To avoid all platform specific struggles, Johannes will show all examples in Icenium Mist. This cloud-based IDE from Telerik a new rising star in the hybrid app world.

3.1 Knockout based App with Bootstrap

We have already revisited the MVVM framework Knockout in the first and second webinar. Lets continue with the already well-known "sticky notes" example. The demo **"CordovaBootstrapKnockoutDemo"** utilizes jQuery for data retrieval and iterates through the retrieved data with the help of Knockout. A plain white HTML5 application has not attractive appearance. The front-end framework Bootstrap is a great way to change this. It ships with a set of CSS classes (like `container`, `glyphicon`, `alert`, `list-group` or `badge`) that can be used to style our hybrid application. Themes, like the Bootswatch Slate theme, create a native app appearance. Here is an example where both the Bootstrap CSS classes as well as the Knockout **HTML 5 data-attributes** (`data-bind` with `click`, `foreach` or `text`) are used to show a list of notes:

```html
<div id="index_template" class="container">
    <i class="glyphicon glyphicon-home" data-bind="click: $root.loadData"></i>
    <h2>Bootstrap-Ko</h2>

    <p class="alert alert-info">
        The following list was rendered with <b>jQuery</b> &amp; <b>Knockout</b>
        and styled via the Slate <b>Bootstrap</b> theme.
    </p>

    <div class="list-group" data-bind="foreach: notes">
        <a class="list-group-item" data-bind="click: $root.showDetails">
            <span class="badge" data-bind="text: moment(Added()).format('DD.MM.YYYY')"></span>
            <!--ko text: Title--><!--/ko-->
        </a>
    </div>
</div>
```

Which renders to this screen in Apache Cordova / Icenium:



Bootstrap-Ko

The following list was rendered with **jQuery** & **Knockout** and styled via the Slate **Bootstrap** theme.

Require.js & Prinzipien für modulares JavaScript - Teil 2	02.10.2013
Require.js & Prinzipien für modulares JavaScript - Teil 1	19.09.2013
jQuery 2013 - Teil 2	26.09.2013
jQuery 2013 - Teil 1	12.09.2013
ASP.NET Web API - Teil 2	27.08.2013
ASP.NET Web API - Teil 1	13.08.2013
Knockout.js für .NET	11.06.2013

Bootstrap comes with a lot of interesting CSS tricks. For example a special font ("glyphicons") is used to create scalable icons. That font contains icons instead of letters. A good font looks sharp on all devices, so that higher pixel ratios of "retina displays" do not have any negative impact. This technique is called **icon fonts**, too.

3.2 jQuery Mobile based App

As we have seen, Bootstrap has "mobile first" responsive grid and a appealing flat graphical style. But it is still designed to work on mobile engines as well as in normal browsers. The demo **"CordovajQueryMobileDemo"** utilizes jQuery for data retrieval and renders its content with jQuery Mobile. JQuery mobile (often abbreviated as **jQM**) goes on step further and concentrates on mobile scenarios only. As the name suggests, it is a framework on top of jQuery. While Bootstrap does its (black) magic with CSS tricks, jQuery mobile is overcoming HTML5 limitations with JavaScript. For example a button is transformed from this:

```
<button type="submit">Go</button>
```

to this:

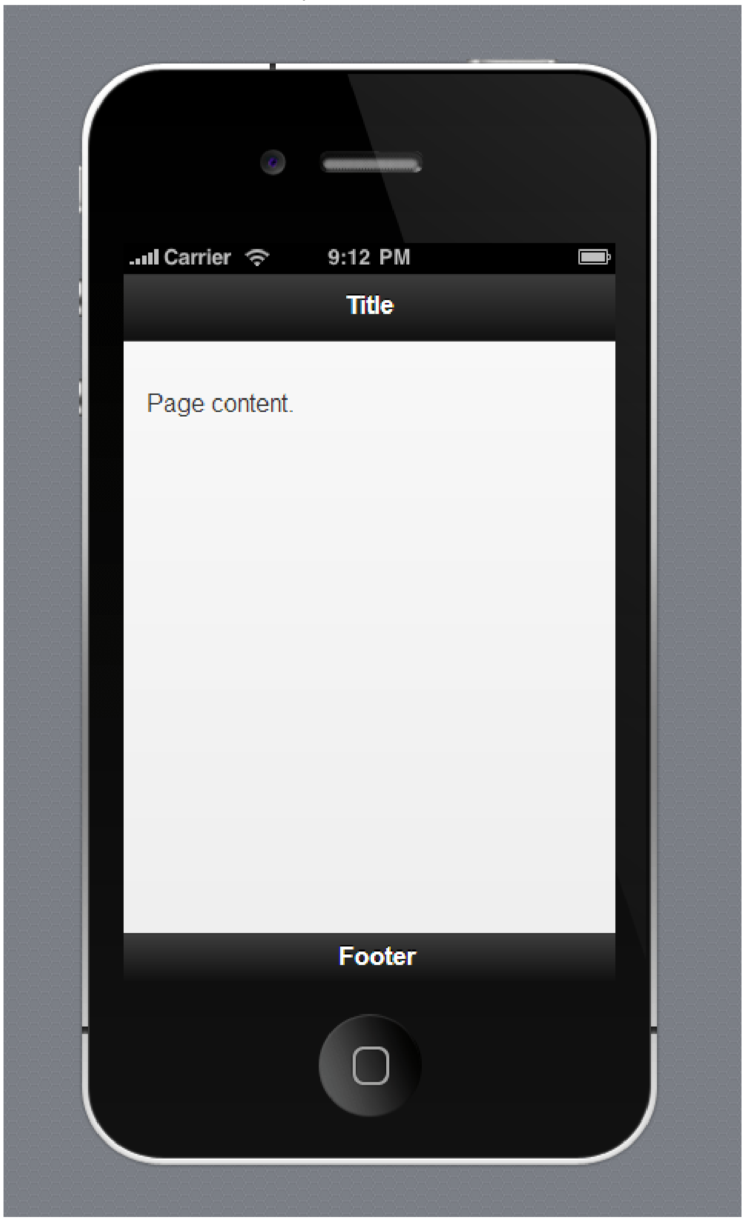
```
<div class="ui-btn ui-btn-inline ui-btn-corner-all ui-shadow ui-btn-up-c">
  <span class="ui-btn-inner ui-btn-corner-all">
    <span class="ui-btn-text">Go</span>
  </span>
  <input class="ui-btn-hidden" type="button" value="Go" />
</div>
```

On the one hand all these enhancement hide a lot of problems to the developer, on the other hand it is a bit trickier to enrich jQM apps with a MVVM framework, as explained here. Due to that reason, it is wise to concentrate on jQuery core and jQuery plugins only.

jQM is touch-optimized, themable and ships with a larger collection of widgets. In contrast to the "normal" jQuery way, jQuery mobile is not concentrating on particular DOM elements. It is seeing the page as a whole and enhances all found elements. jQuery Mobile automatically applies many markup enhancements as soon as it loads. These enhancements are applied based on jQuery Mobile's default settings, which are designed to work with common scenarios. Too hook into this process, read more in the documentation about the `mobileinit` event. This transformation can be steered by **HTML5 data attributes** (eg. `data-role`). The following snipped will render a page with three areas (a header, a content and a footer area).

```
<div data-role="page">
    <div data-role="header">
        <h1>Title</h1>
    </div>
        <div data-role="content">
            <p>Page content.</p>
        </div>

    <div data-role="footer">
        Page Footer
    </div>
</div>
```

This is the rendered result in Apache Cordova / Icenium:



We don't have to care about page transitions. By default, all local links get a click listener automatically. jQuery Mobile will automatically handle page requests in a single-page model, using Ajax when possible. As shown in the next example, if a link in a multi-page document points to an anchor (#page-details), the framework will look for a page wrapper with that id (id="page-details"). If it finds a page in the HTML document, it will bring the new page into view.

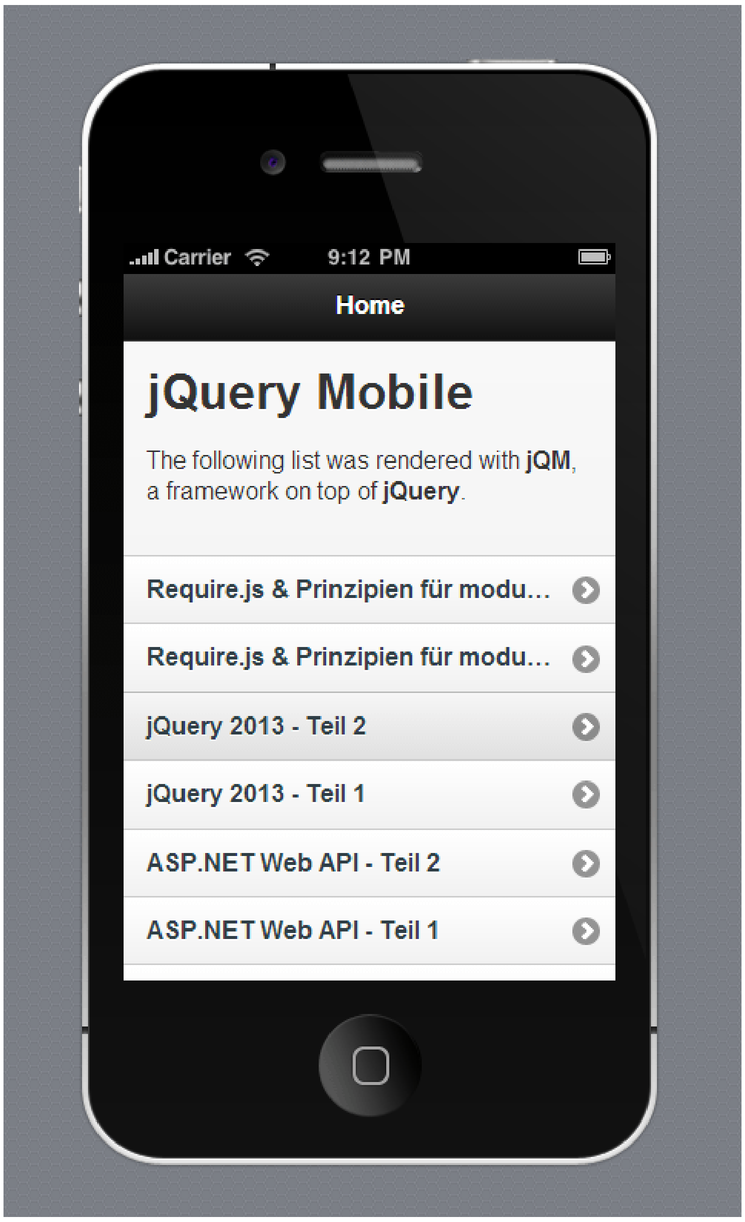
Loading of content is straightforward. We can use JQuery core as usual, so that this snipped is the only self-written JavaScript required to show the start screen:

```
<div id="page-home" data-role="page" >

    <div data-role="header" data-position="fixed">
        <h1>Home</h1>
    </div>

    <div data-role="content">
        <h1>jQuery Mobile</h1>
        <p>
            The following list was rendered with <b>jQM</b>, a framework on top of <b>jQuery</b>.
        </p>
    </div>
    <div class="nav">
        <ul id="home-listview" data-role="listview"></ul>   <!-- HERE  -->
    </div>
</div>
```

```
(function ($) {

    $("#page-home").on('pagebeforeshow', function () {

        $.ajax({
            url: 'http://johanneshoppe.github.io/DeveloperMediaSlides/examples/webinarp.json',
            dataType: 'jsonp',
            jsonpCallback: 'callback'
        }).done(function (result) {
            $('#home-listview')
                .empty()
                .append(createListItems(result))
                .listview('refresh');
        });
    });

    var createListItems = function(data) {

        var items = [];
        $.each(data, function(index, item) {

            var listItemWithLink = $('<li />').append(
                $('<a />')
                    .attr('href', '#page-details')
                    .data('transition', 'slide')
                    .text(item.Title));

            items.push(listItemWithLink);
        });
        return items;
    };

})(window.jQuery);
```

As you see, we are inserting some `<li><a href="" /></li>` elements directly into the document. It is important to know, that the Listview Widget requires a call at the **refresh** method to update the visual styling.

Here is the final result:



jQuery Mobile

The following list was rendered with **jQM**, a framework on top of **jQuery**.

Require.js & Prinzipien für modu...

Require.js & Prinzipien für modu...

jQuery 2013 - Teil 2

jQuery 2013 - Teil 1

ASP.NET Web API - Teil 2

ASP.NET Web API - Teil 1

3.3 Kendo UI Mobile based App

In my opinion there is a golden mean between a pure CSS-based framework (like Bootstrap) combined with a MVVM framework (like Knockout) and jQuery mobile. This framework is called Kendo UI mobile. Kendo UI Mobile was definitely inspired by both Knockout AND jQuery mobile. It uses well-known HTML5 data attributes, like `data-bind="text: Text"` for MVVM **data-binding** as in Knockout and `data-role="listview"` for the creation of a **widget** as in jQM. Please take a look at the **"CordovaKendoUiMobileDemo"** sample app.

But in contrast to jQM, with Kendo when you create a button:

```
<a data-role="button">Go</a>
```

the enhanced HTML will be only changed to this:

```
<a data-role="button" class="km-button">
    <span class="km-text">Go</span>
</a>
```

(...which is great!)

Unlike the jQuery mobile, the mobile version of Kendo UI is doing some hidden task under the hood. Initializing a `kendo.mobile.Application` does the following:

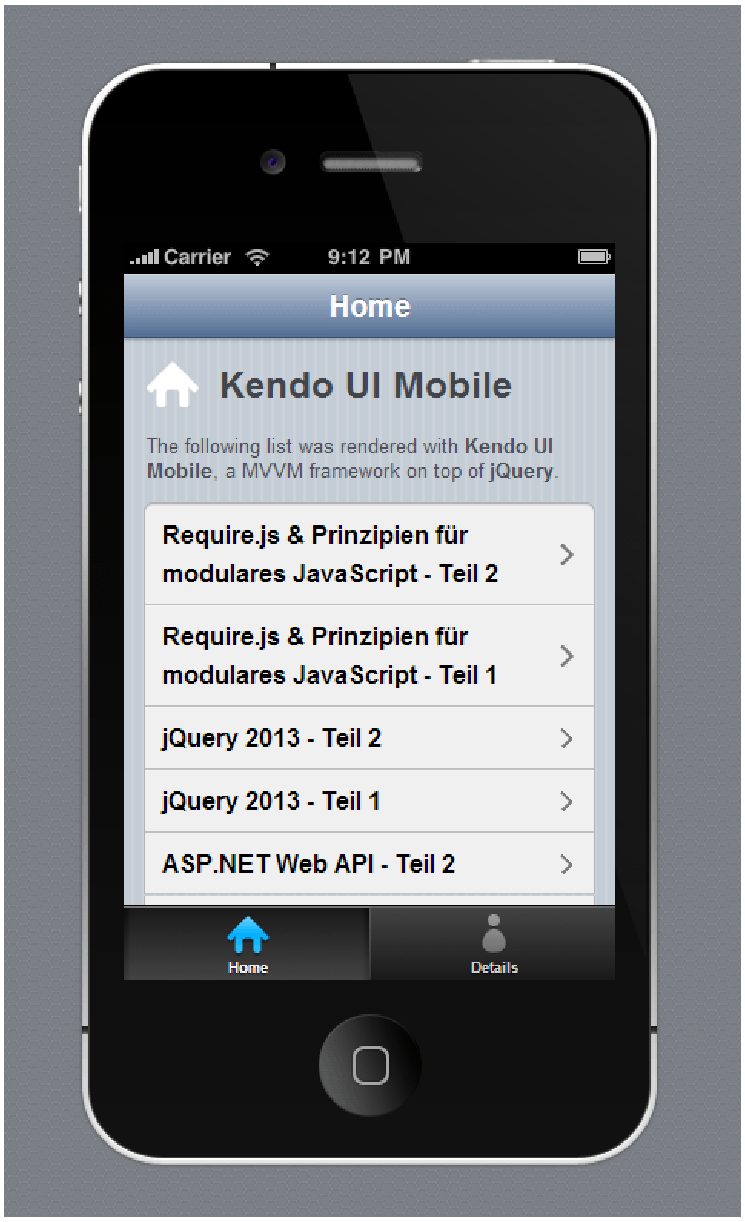
1. Attaches event handlers for various events
2. Adds various CSS classes to page elements
3. Appends meta tags for the viewport configuration
4. Listens for browser history changes

Similar to the Bootstrap glyphicons, Kendo UI Mobile includes integrated font icons, too. They can be applied by specifying the `data-icon` attribute on some widgets. All other elements can use the CSS classes directly. (eg. `class="km-icon km-featured"`)

The final source code looks like this and renders to the following screenshot.
Please notice that we can declare the required ViewModel directly in the source code. (`data-model="app.indexPageViewModel"`)

```html
<div id="indexPage" data-role="view" data-title="Home" data-model="app.indexPageViewModel">

    <div data-role="content" class="view-content">

        <i class="km-icon km-home"></i>
        <h1>Kendo UI Mobile</h1>
        <p>
            The following list was rendered with <b>Kendo UI Mobile</b>,
            a MVVM framework on top of <b>jQuery</b>.
        </p>

        <ul data-role="listview"
            data-template="notesTemplate"
            data-style="inset"
            data-bind="source: notes, events: { click: showDetails }">
        </ul>

    </div>
</div>

<script id="notesTemplate" type="text/x-kendo-template">
    <a>#=Title#</a>
</script>
```

```javascript
(function (global) {
    var app = global.app = global.app || {};

    document.addEventListener("deviceready", function() {

        app.indexPageViewModel = new IndexPageViewModel();
        app.indexPageViewModel.loadData();
        app.application = new kendo.mobile.Application($(document.body), { layout: "tabstrip-layout" });

    }, false);

})(window);


var IndexPageViewModel = function () {

    var viewModel = kendo.observable({
        notes: [],

        loadData: function() {

            $.ajax({
                url: 'http://johanneshoppe.github.io/DeveloperMediaSlides/examples/webinarp.json',
                dataType: 'jsonp',
                jsonpCallback: 'callback'
            }).done(function(result) {
                viewModel.set("notes", result);
            });
        },

        showDetails: function(e) {
            app.detailsPageViewModel.setData(e.dataItem);
            app.application.navigate('#detailsPage', 'slide:left');
        }
    });

    return viewModel;
}
```



Kendo UI Mobile

The following list was rendered with **Kendo UI Mobile**, a MVVM framework on top of **jQuery**.

- Require.js & Prinzipien für modulares JavaScript - Teil 2
- Require.js & Prinzipien für modulares JavaScript - Teil 1
- jQuery 2013 - Teil 2
- jQuery 2013 - Teil 1
- ASP.NET Web API - Teil 2

4. More

There are several more things to discover.
You should start by downloading the sources of the three demo apps.

» Download Demo-Code (.zip)

A great book is "jQuery in Action".
It goes into details where other books still cover boring topics!