



KendoUI开发教程

极客学院出版

前言

jQuery UI 是一套 JavaScript 函数库，提供抽象化、可自订主题的 GUI 控制项与动画效果。基于 jQuery JavaScript 函数库，可用来建构互动式的 Web 应用。

在开发 Web 应用时，可以直接使用 jQueryUI，也可以使用其它一些基于 jQuery 的其它 UI 框架，其中 Kendo (日文剑道) UI 就是其中的佼佼者。两种框架各有所长，下表列出两种 UI 框架的简单比较：

分类	jQuery UI	Kendo UI
Widgets (Total)	Yes (8)	Yes (18 Web; 8 Mobile)
Themes (Total)	Yes (25)	Yes (5)
Interactions (Total)	Yes (5)	Yes (6)
Templates	No (see Template)	Yes
DataSource	No (see Dataview)	Yes
Model-View-ViewModel (MVVM)	No	Yes
Data Visualization	No	Yes
Globalization	No (see Globalize)	Yes
Validation	No (see jQuery Validation Plugin)	Yes
Browser Support	IE 6+, Firefox 3+, Safari 3.1+, Opera 9.6+, Chrome	IE 7+, Firefox 3+, Safari 4+, Opera 10+, Chrome
Mobile Device Compatibility	No (see jQuery Mobile)	Yes

可以看到 Kendo UI 和 jQuery UI 相比添加了模板 (Template)，数据源绑定 (DataSource)，MVVM 和移动设备的支持，功能上相对要强大很多，但 Kendo UI 本身不是免费，它是由 [Telerik](#) 公司开发并支持的。

Kendo UI 是一个基于 HTML5 和 jQuery 的 UI 框架用来开发时尚Web应用。这个UI框架包括的很多 UI 控制项，数据显示组件，和自适应的手机框架，并支持数据绑定，使用模板，拖放功能。

Kendo UI 包含下面下面几个开发包：

- Kendo UI Web - 用于桌面浏览器的 HTML5UI 组件.
- Kendo UI DataViz - 用于显示数据的 HTML5UI 组件.
- Kendo UI Mobile - 用于开发基于移动设备的 HTML5 UI 框架.
- Kendo UI Complete -包含了上面三个开发包.

- Kendo UI Complete for ASP.NET MVC –包括了 Web, DataViz 和 Mobile 并提供这些 UI 组件的 ASP.NET MVC 服务器端封装.
- Kendo UI Complete for JSP – 包括了 Web, DataViz 和 Mobile 并提供这些 UI 组件的 JSP 服务器端封装.
- Kendo UI Complete for PHP – 包括了 Web, DataViz 和 Mobile 并提供这些 UI 组件的 PHP 服务器端封装



桌面Web应用



移动Web应用



数据显示控件



服务器端封装

imobilebbs.com

将在后面的文章逐步介绍 Kendo UI 的使用。

更新日期

2015-07-23

更新内容

Kendo UI 开发教程

目录

前言	1
第 1 章 准备	7
Kendo UI Web	9
Kendo UI DataViz	11
Kendo UI Mobile	12
第 2 章 初始化 Data 属性	14
配置	16
事件	17
数据源	18
模板	19
第 3 章 UI Widgets 概述	20
第 3 章 使用 jQuery 初始化 Kendo UI 组件	22
第 3 章 配置 Kendo UI 组件	24
第 3 章 获取 Kendo UI 组件的引用对象	27
第 3 章 使用 Kendo UI 组件的方法	29
第 3 章 监听 Kendo UI 事件	31
第 3 章 事件处理函数	33
第 4 章 使用 Kendo UI 库实现对象的继承	35
使用 kendo.Class.extend 创建对象	37
创建构造函数	38
创建一个派生对象	39
第 5 章 Kendo DataSource 概述	41
准备开始	43

	绑定数据源到 UI 组件	44
第 6 章	Kendo UI 模板概述.....	47
	显示原始数据	49
	显示 HTML 数据.....	50
	嵌入式模板 vs 外部模板	52
第 7 章	Kendo UI 特效概述.....	54
	准备开始	43
	指定特效显示的方向指定特效显示的方向.....	57
	Kendo UI 支持的特效种类	59
第 8 章	Kendo UI Validator 概述	60
	HTML 5 表单校验.....	62
	– HTML 5 数据类型（如 EMail，URL，数值等）为了使用这些规则，可以通过为 HTML 输入添加对应的属性的方法来设置。比如：	63
	Kendo UI Validator 的基本配置.....	64
	预设支持的校验规则	65
	自定义规则	66
	自定义输入提示的位置.....	67
第 9 章	Kendo MVVM (一) 概述	68
	准备开始	43
	数据绑定	72
第 10 章	Kendo MVVM (二) ObservableObject 对象.....	73
	概述	74
	读取 ObservableObject	76
	设置 ObservableObject 属性.....	77
第 11 章	Kendo MVVM 数据绑定(一) attr	79
第 12 章	Kendo MVVM 数据绑定(二) Checked	82
	绑定一个数组到一组多选框	84

第 13 章	Kendo MVVM 数据绑定(三) Click	85
	使用 Click 绑定	87
	中止事件向上传递	88
	停止事件缺省处理	89
第 14 章	Kendo MVVM 数据绑定(四) Disabled/Enabled	90
第 15 章	Kendo MVVM 数据绑定(五) Events	92
	中止事件向上传递	88
	停止事件缺省处理	89
第 16 章	Kendo MVVM 数据绑定(六) Html	96
第 17 章	Kendo MVVM 数据绑定(七) Invisible/Visible	98
第 18 章	Kendo MVVM 数据绑定(八) Style	100
第 19 章	Kendo MVVM 数据绑定(九) Text	102
第 20 章	Kendo MVVM 数据绑定(十) Source	104
	Source 绑定到数组	106
	Source绑定到非数组	108
	Source 绑定 Select 元素	110
第 21 章	Kendo MVVM 数据绑定(十一) Value	112
	Input 和 textarea Value 绑定	114
	Select 元素绑定 value	115
第 22 章	单页面应用(一)概述	116
第 23 章	单页面应用(二) Router 类	118
	Router 根路径回调函数	120
	参数	121
	页面切换	124
第 24 章	单页面应用(三) View	126
	使用 HTML 字符串创建 View	128

	使用 Script 模板创建 View	129
	显示 View 内容	130
	集成 MVVM	131
第 25 章	单页面应用(四) Layout	132
第 26 章	移动应用开发简介	135
	第一步： 创建 HTML 页面	137
	第二步：添加 Kendo UI Mobile 的引用	138
	第三步：定义应用布局文件	139
	第四步：构造 View	140
	第五步：初始化移动应用	141



T



准备



首先你需要从 Telerik 网站[下载试用版开发包](#)，注意需要注册后才能下载，或者从本站[下载](#)（18M）

下载后直接解压后包含下面几个文件和目录：

- ./examples - 示例.
- /js - minified 化后的 JavaScript 库.
- /vsdoc JavaScript Intellisense 支持文件
- /styles - minified 后的 CSS 及主题资源.
- changelog.html - Kendo UI 发布文件.

如果你下载伺服器端支持（比如 ASP.NET，PHP 等）还可能包含 - .wrappers 目录，支持伺服器端 UI 组件的封装。

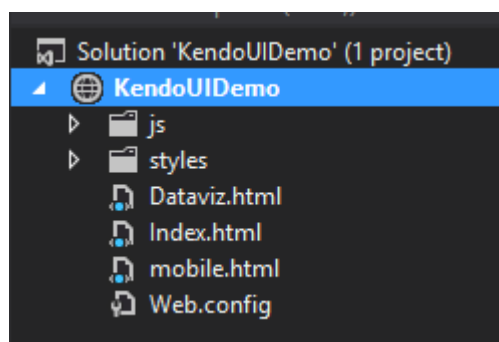
本教程侧重于直接使用 JavaScript，伺服器端支持只是使用 PHP,Net 等伺服器端 API 生产这些客户端代码（主要是 UI 组件的配置），Kendo UI 应用不需要这些伺服器端封装就可以运行。

这裡我们使用 Visual Studio 2012 IDE 作为开发环境，你可以使用你自己熟悉的开发工具，或者直接使用 Note Pad 都可以开发基于 Kendo UI 的 Web 应用。

下面就可以使用 Keudo UI 来开发 Web 应用了。首先使用 Visual Studio 创建一个空的 Web Site，然后及 Kendo UI 的 js 和 styles 目录拷贝到这个新创建的 Website 应用中。你如果直接使用 Notepad 作为开发工具，可以创建一个开发目录，然后把 js,和 Styles 目录拷贝过来也是一样的。使用 Visual Studio 的一个好处是支持 Intellisense，帮助编写 JavaScript 代码，这是就需要把 vsdoc 的内容拷贝到 js 目录下。

然后添加三个 HTML 文档用来测试。

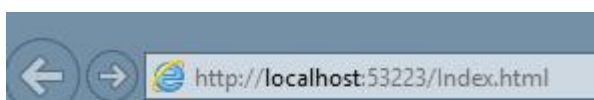
- index.html Web 应用测试页
- dataviz.html 数据显示测试页
- mobile.html 移动应用测试页



Kendo UI Web

编写基于桌面系统的 Web 应用，使用 KendoUI WEB Javascript 库，在 Html 的 Head 部分添加对应的 CSS 和 JavaScript，这裡我们使用一个 DatePicker 控制项做为示例，完整代码如下：

```
<!doctype html>
<html>
  <head>
    <title>Kendo UI Web</title>
    <link href="styles/kendo.common.min.css" rel="stylesheet" />
    <link href="styles/kendo.default.min.css" rel="stylesheet" />
    <script src="js/jquery.min.js"></script>
    <script src="js/kendo.web.min.js"></script>
  </head>
  <body>
    <input id="datepicker" />
    <script>
      $(function () {
        $("#datepicker").kendoDatePicker();
      });
    </script>
  </body>
</html>
```

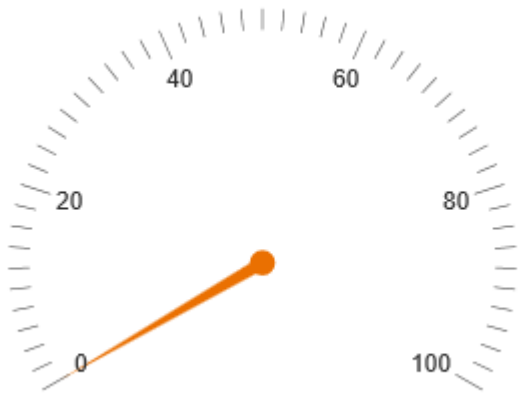


如果能够正确显示 datePicker 控制项，就表示 Kendo UI Web 开发环境已经正确设置好了。

Kendo UI DataViz

如果需要开发数据显示控制项的 Web 页面（比如 DataGrid，表格等），在页面添加 DataViz 库和 CSS 的应用，如下例显示一个仪表盘：

```
<!doctype html>
<html>
  <head>
    <title>Kendo UI DataViz</title>
    <link href="styles/kendo.dataviz.min.css" rel="stylesheet" />
    <script src="js/jquery.min.js"></script>
    <script src="js/kendo.dataviz.min.js"></script>
  </head>
  <body>
    <div id="gauge"></div>
    <script>
      $(function () {
        $("#gauge").kendoRadialGauge();
      });
    </script>
  </body>
</html>
```



Kendo UI Mobile

最后，修改 mobile.html 测试一下移动 Web 应用

```
<!doctype html>
<html>
  <head>
    <title>Kendo UI Mobile</title>
    <link href="styles/kendo.mobile.all.min.css" rel="stylesheet" />
    <script src="js/jquery.min.js"></script>
    <script src="js/kendo.mobile.min.js"></script>
  </head>
  <body>
    <div data-role="view" data-title="View" id="index">
      <header data-role="header">
        <div data-role="navbar">
          <span data-role="view-title"></span>
        </div>
      </header>
      <ul data-role="listview">
        <li>Item 1</li>
        <li>Item 2</li>
      </ul>
      <footer data-role="footer">
        <div data-role="tabstrip">
          <a data-icon="home" href="#index">Home</a>
          <a data-icon="settings" href="#settings">Settings</a>
        </div>
      </footer>
    </div>
    <script>
      var app = new kendo.mobile.Application();
    </script>
  </body>
</html>
```



这样就设置好了 Kendo UI 的开发环境。



T



2

初始化 Data 属性



前面在介绍准备 Kendo UI 开发环境时我们使用 jQuery 的方法将一个 HTML 元素转换成一个 Kendo UI 控制项：\$("#datepicker").kendoDatePicker(); 除了使用 jQuery 插件的方法来初始化方法外，每个 Kendo 控制项还可以通过 data 属性来初始化，此时你需要设置 data 的 role 属性，然后调用 kendo.init 方法。使用初始化 Data 属性的方法在页面上包含有大量 Kendo UI 控制项时非常便利。首先，组件的配置包含在目标元素中，第二无需首先查找每个元素然后调用 jQuery 方法，你只需调用一次 kendo.init() 方法。Kendo UI Mobile 应用通常使用 Data 属性来初始化。

如下例使用 data 属性来初始化一个 UI 组件

```
<div id="container">
  <input data-role="numerictextbox" />
</div>
<script>
kendo.init($("#container"));
</script>
```



其中方法 kendo.init(\$("#container")) 会查找所有包含有 data-role 属性的元素，然后初始化这些 Kendo UI 组件。每个 kendo UI 组件的 role 的值为该 UI 组件名称的小写字母，比如 "autocomplete" 或 "dropdownlist"。

预设情况下，kendo.init 只会初始化 Kendo UI Web 组件和 Kendo UI DataViz 组件（按这个顺序）。而 Kendo UI Mobile 应用首先初始化 Kendo UI Mobile 组件，然后是 Kendo UI Web 组件，最后是 Kendo UI DataViz 组件。这意味著 data-role="listview" 在 Mobile 应用中会预设初始化为 Kendo UI Mobile Listview。然而可以通过指明组件全称的方法在修改这个预设初始化顺序。比如：在 Mobile 应用中指明使用 Kendo UI Web 的 listview

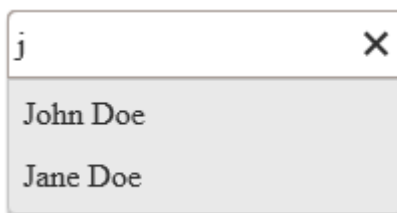
```
<div data-role="view">
  <!-- specify the Kendo UI Web ListView widget -->
  <div data-role="kendo.ui.ListView"></div>
</div>
<script>
var app = new kendo.mobile.Application();
</script>
```


配置

每个组件可以通过 Data 属性来进行配置，对于配置的属性，只需在属性名前加上 data-前缀就可以做为目标元素的属性进行配置。比如 data-delay=" 100" 。对于一些使用 Camel-cased 的属性则是通过添加「 - 」，比如 AutoComplete 的 ignoreCase 的属性可以通过 data-ignore-case 来设置。而对于一些已经是使用 data 前缀的属性则无需再添加 data-前缀。比如 dataTextField 属性可以通过 data-text-field 属性来配置, dataSource 属性可以通过 data-source 属性来配置。对于一些复杂的配置可以通过 JavaScript 对象字面量来配置，比如：data-source="{data: [{name: 'John Doe' }, {name: 'Jane Doe' }]}" 。

如下例：

```
<div id="container">
  <input data-role="autocomplete"
    data-ignore-case="true"
    data-text-field="name"
    data-source="{data: [{name: 'John Doe'}, {name: 'Jane Doe'}]}"
  />
</div>
<script>
  kendo.init($("#container"));
</script>
```



事件

你也可以通过 data 属性添加对 Kendo UI 组件的事件处理，属性的值被当成一个 JavaScript 函数，其作用域为全局。

如下例：

```
<div id="container">
  <input data-role="numerictextbox" data-change="numerictextbox_change" />
</div>
<script>
function numerictextbox_change(e) {
  // Handle the "change" event
}
kendo.init($("#container"));
</script>
```

事件处理函数也可以为一个成员函数，比如 foo.bar 可以看出为 foo 对象的 bar 方法。

例如：

```
<div id="container">
  <input data-role="numerictextbox" data-change="handler.numerictextbox_change" />
</div>
<script>
var handler = {
  numerictextbox_change: function (e) {
    // Handle the "change" event
  }
};
kendo.init($("#container"));
</script>
```

数据源

支持数据绑定的 UI 组件的数据源也可以通过 data-source 属性来指定。这个属性可以 为一个 JavaScript 对象，一个数组或是一个全局变数。

例如：

使用 JavaScript 数组作为数据源。

```
<div id="container">
  <input data-role="autocomplete" data-source="{data:['One', 'Two']}" />
</div>
<script>
kendo.init($("#container"));
</script>
```

使用 JavaScript 数组作为数据源。

```
<div id="container">
  <input data-role="autocomplete" data-source="['One', 'Two']" />
</div>
<script>
kendo.init($("#container"));
</script>
```

使用一个可以全局访问的变数作为数据源。

```
<div id="container">
  <input data-role="autocomplete" data-source="dataSource" />
</div>
<script>
var dataSource = new kendo.data.DataSource( {
  data: [ "One", "Two" ]
});
kendo.init($("#container"));
</script>
```

模板

模板也可以通过 Data 属性来设置，属性的值代表用来定义模板的 Script 元素的 Id。比如：

```
<div id="container">
  <input data-role="autocomplete"
    data-source="[{firstName:'John', lastName: 'Doe'}, {firstName:'Jane', lastName: 'Doe'}]"
    data-text-field="firstName"
    data-template="template" />
</div>
<script id="template" type="text/x-kendo-template">
  <span>#: firstName # #: lastName #</span>
</script>
<script>
kendo.init($("#container"));
</script>
```



3

UI Widgets 概述



Kendo UI 是基于 jQuery 库开发的，Kendo UI widgets 是以 jQuery 插件形式提供的。这些插件的名称基本上都是以 kendo 作为前缀。比如 Kendo 的 autocomplete UI 组件名称为 kendoAutoComplete ,Kendo UI 手机 UI 组件是以 “kendoMobile” 为前缀。比如： ” kendoMobileListView” 。



第 3 章 使用 jQuery 初始化 Kendo UI 组件



Kendo UI 组件使用页面上 HTML 元素来创建，使用 CSS 选择器 然后调用 jquery 插件（kendo UI 组件）将这些 HTML 元素转换为 Kendo UI 组件（基本方法和 jQuery UI 类似）。

例如：初始化一个自动提示输入框组件（autocomplete）

```
<input id="autocomplete" />
<script>
    $("#autocomplete").kendoAutoComplete(["Apples", "Oranges", "Grapes"]);
</script>
```

其中 `$("#autocomplete").kendoAutoComplete(["Apples", "Oranges", "Grapes"]);` 完成两项任务：

查找 Id 为 autocomplete 的 HTML 元素，#autocomplete 为 CSS 选择器 使用 kendoAutoComplete jQuery 插件初始化 Kendo UI 组件，并使用数组 ["Apples", "Oranges", "Grapes"] 作为配置参数传给 kendoAutoComplete 组件 注意：如果 jQuery 找不到由 css 选择器指定的 HTML 元素，Kendo UI 组件不会被创建，你可以使用任意合法的 CSS 选择器来初始化 Kendo UI 组件，对于每个符合选择器条件的 HTML 元素都会初始化一个 Kendo UI 组件。



第3章 配置 Kendo UI 组件




如前面例子，可以通过传递配置对象的方法来配置 Kendo UI 组件，配置对象为一组 Key/Value 对，这些 Key/Value 值用来配置 UI 组件。

如下例，配置一个 Grid 组件。

```
<div id="grid"></div>
<script>
$("#grid").kendoGrid({
  height: 200,
  columns:[
    {
      field: "FirstName",
      title: "First Name"
    },
    {
      field: "LastName",
      title: "Last Name"
    }
  ],
  dataSource: {
    data: [
      {
        FirstName: "John",
        LastName: "Doe"
      },
      {
        FirstName: "Jane",
        LastName: "Doe"
      }
    ]
  }
});
</script>
```

Kendo UI Demos


Overview Web Demos Mobile Demos DataViz Demos Theme Builders



Web demos

23 slick UI widgets plus demos of MVVM, Validation, Globalization, Templates, etc.


Launch Demos



Mobile demos

See what you can do on phones & tablets. Make sure to check the mobile simulator.

Launch Demos



DataViz demos

You won't believe this is just HTML5. Rich, interactive, touch-enabled charts.

Launch Demos

imobilebbs.com

上面例子为 Grid 组件配置了 height, columns 和 dataSource. API 文档 包含了每个 Kendo UI 组件支持的配置项。

第 3 章 获取 Kendo UI 组件的引用对象

Kendo UI 通过 jQuery 插件的方式来初始化，但是调用这些方法时不会返回这些实例对象的引用，而是使用传统的 jQuery 方法来获取所创建的 Kendo UI 对象的引用，为了获得所创建的 Kendo UI 组件对象的引用，使用 jQuery data 方法，例如获取前面例子中创建 kendoAutoComplete 的对象，可以使用下面代码：

```
<input id="autocomplete" />
<script>
$("#autocomplete").kendoAutoComplete(["Apples", "Oranges", "Grapes"]);
var autocomplete = $("#autocomplete").data("kendoAutoComplete");
</script>
```

方法 `$("#autocomplete").data("kendoAutoComplete")` 返回所创建的 Kendo AutoComplete 对象。data 的参数为 Kendo UI 组件的名称，比如 `"kendoAutoComplete"`，`"kendoGrid"` 等。



第 3 章 使用 Kendo UI 组件的方法



在获取 Kendo UI 组件对象的引用之后，就可以调用该 UI 组件的方法，例如：

```
<input id="autocomplete" />
<script>
$("#autocomplete").kendoAutoComplete(["Apples", "Oranges", "Grapes"]);
var autocomplete = $("#autocomplete").data("kendoAutoComplete");
autocomplete.value("Cherries");
var value = autocomplete.value();
alert(value); // Displays "Cherries"
</script>
```

上面的例子中获取 autocomplete 对象之后，调用了它的 value()方法来写入和读取该输入框的内容。



第3章 监听 Kendo UI 事件



Kendo UI 组件支持多种事件，比如按键，鼠标，内容变化等事件，有两种方法可以为 Kendo UI 组件定义事件处理方法：

```
<input id="autocomplete" />
<script>
function autocomplete_change() {
    // Handle the "change" event
}
$("#autocomplete").kendoAutoComplete({
    change: autocomplete_change
});
</script>
```

下面例子，使用 bind 方法。

```
<input id="autocomplete" />
<script>
function autocomplete_change() {
    // Handle the "change" event
}
$("#autocomplete").kendoAutoComplete();
var autocomplete = $("#autocomplete").data("kendoAutoComplete");
autocomplete.bind("change", autocomplete_change);
</script>
```

两种方法都把一个函数绑定到 autocomplete 的 "change" 事件。此时如果 autocomplete 内容发生变化，则触发 change 事件，相应的事件处理方法会被调用。



第 3 章 事件处理函数



事件处理函数在事件发生时被调用，该事件处理函数的传入参数包含了事件相关的 JavaScript 对象，你可以通过 sender 参数获得触发该事件的 UI 组件，比如：

```
<input id="autocomplete" />
<script>
function autocomplete_change(e) {
    var autocomplete = e.sender;
    var value = autocomplete.value();
    alert(value); // Displays the value of the AutoComplete widget
}
$("#autocomplete").kendoAutoComplete({
    change: autocomplete_change
});
</script>
```

此外，也可以使用 this 关键字来获取触发事件的 UI 对象引用，比如：

```
<input id="autocomplete" />
<script>
function autocomplete_change(e) {
    var autocomplete = this;
    var value = autocomplete.value();
    alert(value); // Displays the value of the AutoComplete widget
}
$("#autocomplete").kendoAutoComplete({
    change: autocomplete_change
});
</script>
```



4



使用 Kendo UI 库实现对象的继承



JavaScript 也是一种面向对象的开发语言，但和 C++，Java,C# 所不同的是，它的对象不是基于类（Class），而是基于对象原型（ProtoType），因此对于来自 C++,Java 等背景的程序员，初次接触到 JavaScript 的面向对象的开发时，开始会有些不适应。而 JavaScript 语言本身也非常灵活，实现面向对象的方法也很多，不同的框架使用的方法也不同。

对于 JavaScript 的面向对象的方法和 C++，Java 面向对象的不同点，举个简单的类比，使用 C++,Java 来建房，是先有蓝图（Class），然后根据这个蓝图（Class）来建房（对象）。而 JavaScript 是直接建个房（Object），如果要建个新房，就参考这个建好的房作为原型（prototype），然后复制一个对象。

Kendo UI 不仅仅提供了一些好看的UI组件，而且也提供一个 JavaScript 构建对象，实现继承的方法，其形式接近于 C++，Java 的类继承方法。

使用 kendo.Class.extend 创建对象

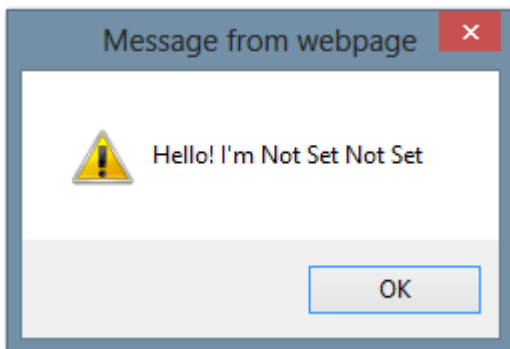
首先可以创建一个新对象（注意 JavaScript 中没有类的概念），可以通过 `kendo.Class.extend` 来定义。

```
var person = kendo.Class.extend({});
```

上面代码创建一个 `Person` 对象，但没有定义 `Person` 对象任何属性和方法。下面可以为 `Person` 定义一些属性和方法（函数），可以通过对象字面量的方法来定义，Javascript 对象的属性或方法都是以 `Key: value` 的形式来定义。也使用 `this` 来引用对象的方法或属性。

```
var Person = kendo.Class.extend({
  firstName: 'Not Set',
  lastName: 'Not Set',
  isAPrettyCoolPerson: false,
  sayHello: function() {
    alert("Hello! I'm " + this.firstName + " " + this.lastName);
  }
});

var person = new Person();
person.sayHello();
```



创建构造函数

也可以为对象添加一个构造函数，Kendo UI 使用 `init` 来定义构造函数，这样在创建新对象时，可以通过构造函数来创建新的对象。下面代码重新定义 `Person` 对象，并为其添加一个属性 `isAPrettyCoolPerson`，

```
var Person = kendo.Class.extend({
  firstName: 'Not Set',
  lastName: 'Not Set',
  isAPrettyCoolPerson: false,
  init: function (firstName, lastName) {
    if (firstName) this.firstName = firstName;
    if (lastName) this.lastName = lastName;
  },

  sayHello: function () {
    alert("Hello! I'm " + this.firstName + " " + this.lastName);
  }
});

var person = new Person("John", "Bristowe");
person.isAPrettyCoolPerson = true;
person.sayHello();
```

我们使用这个对象，创建一个名为 `John`，`Bristowe` 的 `Person`，并把它的 `isAPrettyCoolPerson` 属性设为 `true`。



创建一个派生对象

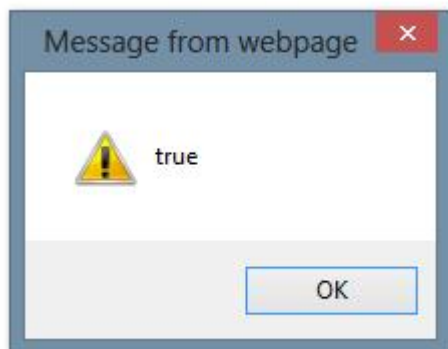
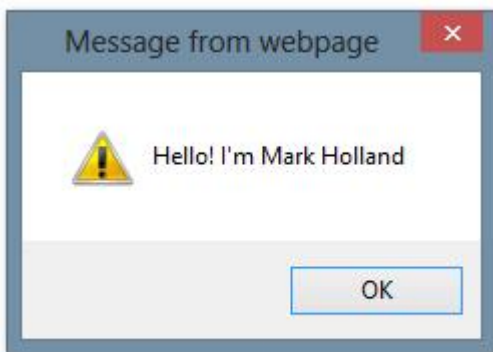
现在我们可以创建 Person 对象的一个派生对象 Parent，Parent 对象继承 Person 对象，然后我们创建一个 Dad 对象。

```
var person = new Person("John", "Bristowe");
person.isAPrettyCoolPerson = true;

var Parent = Person.extend({
    firstName: 'Mark',
    lastName: 'Holland'
});

var myDad = new Parent();
myDad.isAPrettyCoolPerson = true;

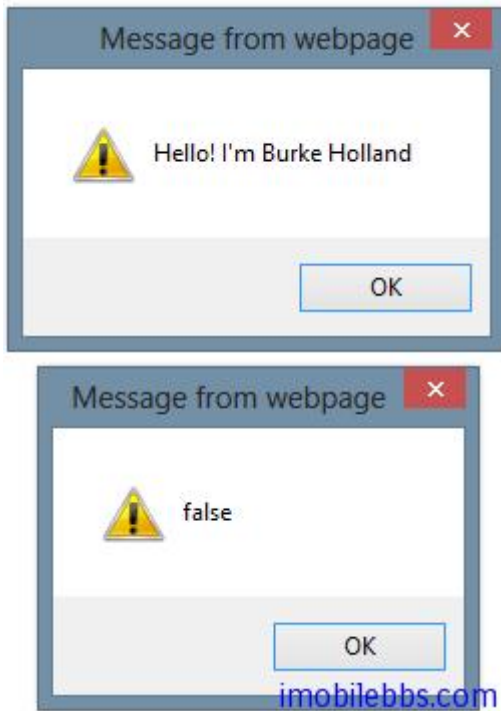
myDad.sayHello();
alert(myDad.isAPrettyCoolPerson);
```



imobilebbs.com

我们再创建一个 Child 对象，继承自 Parent，要注意的是 isCoolPerson 属性。想想它的值是真还是假呢？


```
var Child = Parent.extend({});  
  
var me = new Child();  
me.firstName = "Burke";  
me.sayHello();  
alert(me.isAPrettyCoolPerson);
```



可以看到 me 的 isAPrettyCoolPerson 的值为 false, 没有因为 myDad 的 isAPrettyCoolPerson 为 True 而变为 true, 这些因为 Child 继承自 Parent, Parent 缺省的 isAPrettyCoolPerson 为 false, myDad 修改的只是某个特定的实例的值, 没有修改作为原型的对象 (Parent) 的属性。



5

Kendo DataSource 概述



Kendo 的数据源支持本地数据源（JavaScript 对象数组），或者远程数据源（XML, JSON, JSONP），支持 CRUD 操作（创建，读取，更新和删除操作），并支持排序，分页，过滤，分组和集合等。

准备开始

下面创建一个本地数据源。

```
var movies = [ {
    title: "Star Wars: A New Hope",
    year: 1977
}, {
    title: "Star Wars: The Empire Strikes Back",
    year: 1980
}, {
    title: "Star Wars: Return of the Jedi",
    year: 1983
}
];
var localDataSource = new kendo.data.DataSource({data: movies});
```

创建一个远程数据源（ Twitter ）

```
var dataSource = new kendo.data.DataSource({
    transport: {
        read: {
            // the remote service url
            url: "http://search.twitter.com/search.json",

            // JSONP is required for cross-domain AJAX
            dataType: "jsonp",

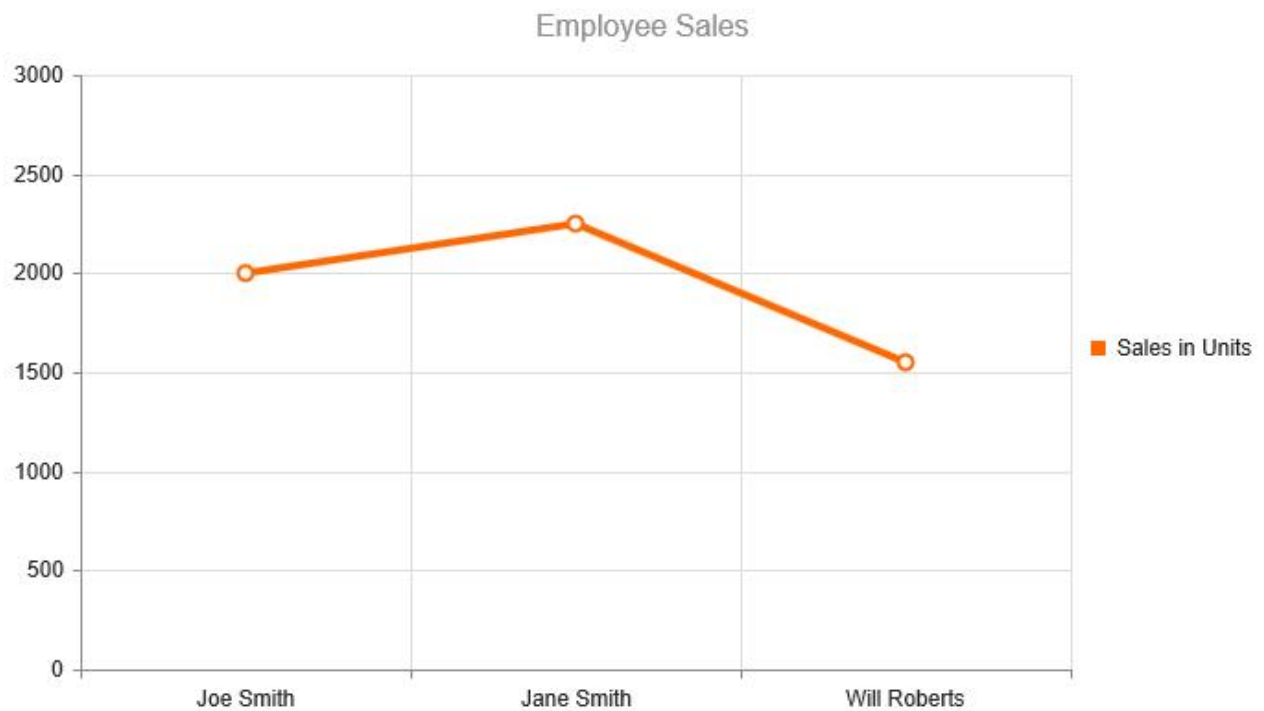
            // additional parameters sent to the remote service
            data: {
                q: "html5"
            }
        }
    },
    // describe the result format
    schema: {
        // the data which the data source will be bound to is in the "results" field
        data: "results"
    }
});
```

绑定数据源到 UI 组件

Kendo UI 组件很多都支持数据绑定，UI 组件绑定的数据源可以在配置 UI 组件时设置，或是多个 UI 组件共享同一个数据源。

创建UI组件时设置 DataSource 属性

```
$("#chart").kendoChart({
  title: {
    text: "Employee Sales"
  },
  dataSource: new kendo.data.DataSource({
    data: [
      {
        employee: "Joe Smith",
        sales: 2000
      },
      {
        employee: "Jane Smith",
        sales: 2250
      },
      {
        employee: "Will Roberts",
        sales: 1550
      }
    ]
  }),
  series: [{
    type: "line",
    field: "sales",
    name: "Sales in Units"
  }],
  categoryAxis: {
    field: "employee"
  }
});
```

imobilebbs.com

使用共享的远程数据源

```
var sharableDataSource = new kendo.data.DataSource({
  transport: {
    read: {
      url: "data-service.json",
      dataType: "json"
    }
  }
});
```

// Bind two UI widgets to same DataSource

```
$("#chart").kendoChart({
  title: {
    text: "Employee Sales"
  },
  dataSource: sharableDataSource,
  series: [{
    field: "sales",
    name: "Sales in Units"
  }],
  categoryAxis: {
    field: "employee"
  }
});
```

```
    }  
  });  
  
$("#grid").kendoGrid({  
  dataSource: sharableDataSource,  
  columns: [  
    {  
      field: "employee",  
      title: "Employee"  
    },  
    {  
      field: "sales",  
      title: "Sales",  
      template: '#= kendo.toString(sales, "N0") #'  
    }  
  ]  
});
```

这个例子使用了模板 `template` ,模板的用法参见后面的文章。



6

Kendo UI 模板概述



Kendo UI 框架提供了一个易用，高性能的 JavaScript 模板引擎。通过模板可以创建一个 HTML 片段然后可以和 JavaScript 数据合并成最终的 HTML 元素。

Kendo 模板侧重于 UI 显示，支持关键的模板功能，着重于性能而不是语法上的方便。

模板语法 Kendo 模板使用了一种称为“#”的语法形式，使用这种语法，#用来表明模板中的某个部分可以使用 JavaScript 数据来替代。

用三种方式使用 # 语法：

1. 显示字面量 `#=#`
2. 显示HTML元素 `#: #`
3. 执行任意的Javascript代码 `#if() {# ...#}#`

注意：如何你的模板中包含有“#”字符，不是用来绑定的部分，你必须使用转义字符，否则会引起模板编译错误。你可以通过“\#”转义需要显示“#”的地方。

显示原始数据

显示数据的本来的形式是使用模板的一个最基本的用法，使用 Kendo UI 模板，可以使用如下类似的代码：

```
var template = kendo.template("<div id='box'>#= firstName #</div>")
```

上面代码创建了“编译”过的嵌入式模板，使用这个模板可以用来显示数据，比如下面的代码

```
<div id="example"></div>
<script>
  var template = kendo.template("<div id='box'>#= firstName #</div>");
  var data = { firstName: "Todd" }; //A value in JavaScript/JSON
  var result = template(data); //Pass the data to the compiled template
  $("#example").html(result); //display the result
</script>
```

通过模板与数据的合并，最终显示“Todd”。

显示 HTML 数据

如果你需要显示经过 HTML 编码过的数据，使用 Kendo UI 模板可以自动处理这些编码过的 HTML 元素，但需要使用不同的语法 `#: ...#`，例如：

```
var template = kendo.template("<div id='box'>#: firstName #</div>");
```

完整的示例如下：

```
<div id="example"></div>
<script>
  var template = kendo.template("<div id='box'>#: firstName #</div>");
  var data = { firstName: "<b>Todd</b>" }; //Data with HTML tags
  var result = template(data); //Pass the data to the compiled template
  $("#example").html(result); //display the resulting encoded HTML Output (<b>Todd</b>)
</script>
```

这个例子的显示结果为

```
<b>Todd </b>
```

而不是 Todd，如果需要显示 Todd，则需要使用 `#= #` 语法，显示 HTML 编码的一个主要作用是当你无需再模板中显示 HTML 标记，而是把整个标记和其内容作为字符串显示出来。

使用外部模板和表达式 在模板中也可以使用表达式，Kendo UI 支持在模板中执行 JavaScript 代码，在模板中使用 JavaScript 代码的方法是在 JavaScript 语句的前后加上 `#`，比如下面模板显示一组列表：

```
<script id="javascriptTemplate" type="text/x-kendo-template">
<ul>
  # for (var i = 0; i < data.length; i++) { #
    <li>#= data[i] #</li>
  } #
</ul>
</script>
```

然后为了使用这个模板，可以通过模板的 id，通过 `kendo.template` 创建这个模板，然后和数据合并，比如：

```

<div id="example"></div>

<script id="javascriptTemplate" type="text/x-kendo-template">
  <ul>
    # for (var i = 0; i < data.length; i++) { #
      <li>#= data[i] #</li>
    # } #
  </ul>
</script>

<script type="text/javascript">
  //Get the external template definition using a jQuery selector
  var template = kendo.template($("#javascriptTemplate").html());

  //Create some dummy data
  var data = ["Todd", "Steve", "Burke"];

  var result = template(data); //Execute the template
  $("#example").html(result); //Append the result
</script>

```

- **Todd**
- **Steve**
- **Burke**

可以看到模板执行了 JavaScript 的 for 循环，并且我们使用了外部模板，外部模板的定义使用 type="text/x-kendo-template" 来定义，并通过其id来访问这个外表模板。在模板中也可以定义变量，使用这个自定义变量的方法和使用字面量的方法类似。比如定义一个变量 myCustomVariable

```

<script id="javascriptTemplate" type="text/x-kendo-template">
  # var myCustomVariable = "foo"; #
  <p>
    #= myCustomVariable #
  </p>
</script>

```

嵌入式模板 vs 外部模板

Kendo UI 模板可以使用嵌入式模板和外部模板：

- inline: 使用 JavaScript 字符串定义
- external: 使用 HTML Script 块定义

嵌入式模板使用比较简单的情况，对于比较复杂的模板一般使用外部模板。外部模板的定义的基本格式如下：

```
<script type="text/x-kendo-template" id="myTemplate">
  <!--Template content here-->
</script>
```

外部模板必须定义一个 id,这样你才可以通过 id 来访问这个模板：

```
//extract the template content from script tag
var templateContent = $("#myTemplate").html();
//compile a template
var template = kendo.template(templateContent);
```

使用外部模板，你可以使用任意合法的 HTML 元素和 JavaScript，只需语法正确，比如：

```
<ul id="users"></ul>

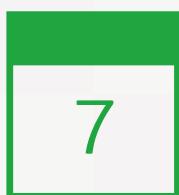
<script type="text/x-kendo-template" id="myTemplate">
  #if(isAdmin){#
    <li>#: name # is Admin</li>
  #}else{#
    <li>#: name # is User</li>
  #}#
</script>

<script type="text/javascript">
  var templateContent = $("#myTemplate").html();
  var template = kendo.template(templateContent);

  //Create some dummy data
  var data = [
    { name: "John", isAdmin: false },
    { name: "Alex", isAdmin: true }
```

```
];  
  
var result = kendo.render(template, data); //render the template  
  
$("#users").html(result); //append the result to the page  
</script>
```

- **John is User**
- **Alex is Admin**



Kendo UI 特效概述



Kendo UI Fx 提供了一个丰富，可扩展，性能经过优化的工具集合用来完成 HTML 元素的过渡显示。每种特效近可能的使用 CSS Transition，对于一些老版本浏览器使用修改属性的方法作为补充。所有动画可以反向显示从而可以方便的实现元素的显示和隐藏。本篇介绍了 Kendo UI 特效的概要，完整的文档可以参考 [API 文档](#)

准备开始

所有 Kendo UI 特效都是通过 kendo.fx JQuery 选择器封装来创建，每个封装支持显示多种特效。例如：

```
<div id="foo">
  I will be animated
</div>

<script>
  var effectWrapper = kendo.fx($("#foo"));
  var fadeOutEffect = effectWrapper.fadeOut();
  fadeOutEffect.play();
</script>
```

和 jQuery 方法一样，kendo UI fx 也支持方法链，比如上面代码可以简化为：

```
<div id="foo">
  I will be animated
</div>

<script>
  kendo.fx($("#foo")).fadeOut().play();
</script>
```

指定特效显示的方向指定特效显示的方向

大部分特效可以指定多个方向。可以通过特效构造方法的第一个参数来指定方向，或者通过调用构造方法的快捷方法来指明方向。比如下面三种方法的效果是一样的。

大部分特效可以指定多个方向。可以通过特效构造方法的第一个参数来指定方向，或者通过调用构造方法的快捷方法来指明方向。比如下面三种方法的效果是一样的。

```
<div id="foo">
  I will be animated
</div>

<script>
  var fadeOut1 = kendo.fx($("#foo")).fadeOut();
  var fadeOut2 = kendo.fx($("#foo")).fade("out");
  var fadeOut3 = kendo.fx($("#foo")).fade().direction("out");

  //Call .play() to run any of the above animations
</script>
```

组合特效 可以将多个特效组合中一起。比如：

```
<div id="foo">
  I will be faded out and zoomed out.
</div>

<script>
  var effectWrapper = kendo.fx($("#foo"));
  var fadeOutEffect = effectWrapper.fadeOut();
  fadeOutEffect.add(effectWrapper.zoomOut());
  fadeOutEffect.play();
  // Calling reverse will zoom in and fade in.
</script>
```

组合特效也可以同时应用到多个元素，这时需要通过 \$when 方法。比如下面代码：

```
<div id="foo">
  I will fade out.
</div>
<div id="baz">
  I will also fade out.
```

```
</div>

<script>
  //Use jQuery Deferred to chain multiple effects
  var eleFoo = $("#foo"),
      eleBaz = $("#baz");

  $.when(kendo.fx(eleFoo).fadeOut().play(),
        kendo.fx(eleBaz).fadeOut().play()).then(function(){
    //This will be called when both animations are done
    alert("Both elements faded!");
  });
</script>
```

Kendo UI 支持的特效种类

Kendo UI 支持下面几种特效，具体请参见其文档

- [Expand](#)
- [Fade](#)
- [Flip](#)
- [PageTurn](#)
- [SlideIn](#)
- [Tile](#)
- [Transfer](#)
- [Zoom](#)



8

Kendo UI Validator 概述



Kendo UI Validator 支持了客戶端校驗的便捷方法，它基於 HTML 5 的表單校驗功能，支持很多內置的校驗規則，同時也提供了自定義規則的便捷方法。

HTML 5 表單校驗

HTML5 的一項重要功能是[HTML 5 表單校驗屬性](#)，通過設置限制屬性為 HTML 輸入設置輸入類型，值域等，然後由瀏覽器來檢查輸入是否合法。支持的幾種規則有：

- 必填域
- 正規表達式規則
- 最大，最小值域

– HTML 5 數據類型（如 EMail，URL，數值等）為了使用這些規則，可以通過為 HTML 輸入添加對應的屬性的方法來設置。比如：

```
<input type="email" required>
```

如果瀏覽器支持 HTML5，則它會自動根據這些規則來檢查輸入的值是否符合規則，如果輸入數據無效，瀏覽器會顯示錯誤信息給用戶，也不會提交表單。HTML5 也支持了一些新添的 JavaScript 方法來實現手工校驗，比如 `checkValidity()` 方法。

HTML 5 表單校驗存在的問題 HTML 5 表單校驗非常有用，但它也存在一些問題，比如：

一些舊版本瀏覽器不支持 HTML5. 某些支持 HTML5 的瀏覽器對 HTML 5 表單支持不完整。由瀏覽器生成的錯誤信息很難為它們重新定義顯示風格。Kendo UI Validator 就是為了解決上面的這些問題而實現的。

Kendo UI Validator 的基本配置

Kendo UI Validator 支持標準的 HTML5 表單校驗屬性，從而允許你正常使用 HTML 5 表單校驗屬性，從而可以在所有瀏覽器（IE7+）上使用這些屬性，比如：

```
<div id="myform">
  <input type="text" name="firstName" required />
  <input type="text" name="lastName" required />
  <button id="save" type="button">Save</button>
</div>
```

然後，在頁面上添加 Kendo UI Validator，添加在 Script 部分，比如：

```
// Initialize the Kendo UI Validator on your "form" container
// (NOTE: Does NOT have to be a HTML form tag)
var validator = $("#myform").kendoValidator().data("kendoValidator");

// Validate the input when the Save button is clicked
$("#save").on("click", function() {
  if (validator.validate()) {
    // If the form is valid, the Validator will return true
    save();
  }
});
```

使用這樣的簡單配置，這些 HTML5 表單校驗在舊版本瀏覽器中也可以工作，並且 Web 應用可以完全控制錯誤信息的顯示和其顯示風格，當點擊「Save」按鈕後，如果輸入不滿足輸入規則，Kendo UI Validator 顯示合適的錯誤信息，每個 HTML 元素也可以通過 validatorMessage 定義一個自定義的錯誤信息，比如：

```
<input type="tel" pattern="\d{10}" validationMessage="Plase enter a ten digit phone number" />
```

預設支持的校驗規則

輸入項必填規則

```
<input type="text" name="firstName" required />
```

輸入必須符合指定的正規表達式

```
<input type="text" name="twitter" pattern="https?://(?:www\.)?twitter\.com/.+i" />
```

最大，最小值限制

```
<input type="number" name="age" min="1" max="42" />
```

輸入步驟和最大，最小值限制一同使用

```
<input type="number" name="age" min="1" max="100" step="2" />
```

輸入為有效的 URL

```
<input type="url" name="url" />
```

輸入為有效的 EMail

```
<input type="email" name="email" />
```

除此之外，Kendo UI Validator 也支持自定義的規則。

自定義規則

使用自定義規則時的注意事項：

- 表單的每個元素都會執行自定義規則，因此如果表單中有多個輸入項，注意檢查輸入項的類型，然後再執行合適的校驗規則，比如：

```
custom: function (input) {  
    if (input.is("[name=firstName]")) {  
        return input.val() === "Test"  
    } else {  
        return true;  
    }  
}
```

- 如果自定義規則返回 true,那麼表示校驗成功。
- 如果有多個自定義規則，那麼會按屬性執行這些自定義規則，在發生錯誤時停止後續校驗規則的執行，而是顯示錯誤信息。只有所有規則都通過才表示表單校驗成功。
- 自定義錯誤信息必須和自定義規則匹配，如果沒有定義自定義錯誤信息，則顯示一個簡單的出錯圖標。

自定義輸入提示的位置

預設情況下 Kendo UI 將輸入提示顯示在輸入框附近，然而，如果輸入通過 Kendo UI 插件轉換為 ComboBox，AutoComplete 或其它 Kendo UI 組件後，預設的輸入提示可能會影響到某些重要信息的顯示，這時，你可以指定在什麼地方顯示輸入提示，這時，可以通過添加一個 span 元素，定義 data-for 屬性到需要校驗的輸入框的 name，並添加 .k-invalid-msg 類。

比如：

```
custom: function (input) {  
    if (input.is("[name=firstName]")) {  
        return input.val() === "Test"  
    } else {  
        return true;  
    }  
}
```





9

Kendo MVVM (一) 概述



Model View ViewModel (MVVM) 是开发人员经常使用的一种设计模式，以实现数据模型（Model）和视图（View）的分离。MVVM 中的 ViewModel 部分负责把模型中的数据对象以某种方便的形式和 View 结合起来（通常是通过数据绑定的方式）。

Kendo MVVM 实现了 MVVM 设计模式，并且支持和 Kendo 框架的其它部分（如 UI 组件和数据源）的无缝连接。

准备开始

使用 MVVM 模式首先创建 ViewModel 对象，ViewModel 对象代表了可以使用 View 显示的数据对象，Kendo 框架中使用 `kendo.observable` 函数通过传入 JavaScript 对象的方法来定义一个 ViewModel 对象。比如：

```
var viewModel = kendo.observable({
  name: "John Doe",
  displayGreeting: function() {
    var name = this.get("name");
    alert("Hello, " + name + "!!!");
  }
});
```

然后使用 HTML 创建一个 View，这个 View 包含一个按钮和一个文本框。

```
<div id="view">
  <input data-bind="value: name" />
  <button data-bind="click: displayGreeting">Display Greeting</button>
</div>
```

其中文本框(input) 通过 `data-bind` 属性指明绑定到 ViewModel 对象的 `name` 域。此时 `name` 域值发生变化将会反映到 UI 界面的 Input 输入框内容的变化。反之亦然，当 UI 输入框内容发生变化时，ViewModel 的 `name` 域也发生变化。按钮的 `click` 事件绑定到 ViewModel 的 `displayGreeting` 方法。

最后，通过 `bind` 方法将 View 和 ViewModel 绑定起来。

`kendo.bind($("#view"), viewModel);` 完整的代码如下：

```
<!doctype html>
<html>
<head>
  <title>Kendo UI Web</title>
  <link href="styles/kendo.common.min.css" rel="stylesheet" />
  <link href="styles/kendo.default.min.css" rel="stylesheet" />
  <script src="js/jquery.min.js"></script>
  <script src="js/kendo.web.min.js"></script>

</head>
<body>
<div id="view">
```

```
<input data-bind="value: name" />
<button data-bind="click: displayGreeting">Display Greeting</button>
</div>

<script>
  var viewModel = kendo.observable({
    name: "John Doe",
    displayGreeting: function () {
      var name = this.get("name");
      alert("Hello, " + name + "!!!");
    }
  });

  kendo.bind($("#view"), viewModel);
</script>
</body>
</html>
```


数据绑定

数据绑定将 DOM 元素（或者 UI 组件）的属性绑定到 ViewModel 的某个属性或是方法。绑定通过设置 data-bind 属性，采用“绑定名称：ViewModel 的属性或方法”的格式，也就是 value : name 的形式来指明。上面的例子使用了两种不同类型的绑定，value 和 click。Kendo MVVM 也支持其它类型的绑定，如 source, html, attr, visible, enable 等。data-bind 也可以支持通过逗号分隔的属性列表。Kendo MVVM 数据绑定也支持嵌套的 ViewModel 属性。

比如下例 把 div 绑定到 person.name:

```
<div data-bind="text: person.name">
</div>
<script>
var viewModel = kendo.observable({
  person: {
    name: "John Doe"
  }
});
kendo.bind($("#div"), viewModel);
</script>
```

....

要注意的是 data-bindings 的值不是 Javascript 代码，不可以使用在 data-bindings 中使用 javascript 方法，比如

要实现上面使用小写的功能，可以使用下面的实现：

...



10

Kendo MVVM (二) ObservableObject 对象



概述

Kendo MVVM 框架关键的一个部分为 ViewModel，它主要是通过 `kendo.data.ObservableObject` 来提供支持的。它可以监控改变（UI 变化或是值的变化）并通知关心该变化的组件。本篇以下 ViewModel 和 `ObservableObject` 代表同一对象。

为了创建一个 `ObservableObject` 对象，可以通过创建一个新 `kendo.data.ObservableObject` 实例或是使用 `kendo.observable` 方法，这两种方法效果相同。

例如：

```
var viewModel1 = new kendo.data.ObservableObject( {
  field1: "value1",
  field2: "value2"
});

var viewModel2 = kendo.observable( {
  field1: "value1",
  field2: "value2"
});
```

`kendo.bind` 方法内部实现时自动将给定的 ViewModel 对象转换为一个 `ObservableObject` 对象，除非传入的参数类型已经是一个 `ObservableObject` 对象。

注：如果某个 ViewModel 对象在初始后以后还会使用到（在调用 `kendo.bind` 之前或之后），则必须使用 `kendo.observable` 方法或是 `new kendo.data.ObservableObject` 来创建一个 ViewModel 对象。比如：

```
var viewModel = kendo.observable({
  name: "John Doe"
});

viewModel.set("name", "Jane Doe"); // use the View-Model object after initialization
```

如果 ViewModel 对象在初始化后不再访问这个对象，那么你可以使用普通的 JavaScript 对象，此时 `kendo.bind` 方法不会把原始的 ViewMode 对象转化为 `kendo.data.ObservableObject`。例如，下面的代码出错：

```
var viewModel = {
  name: "John Doe"
};
```

```
kendo.bind(document.body, viewModel);

/*
The following statement will fail because the View-Model
is not an instance of kendo.data.ObservableObject.
*/
viewModel.set("name", "Jane Doe");
```

因此强烈建议总是使用 `kendo.observable` 来初始化一个 `ViewModel` 对象。

读取 ObservableObject

使用 get 方法来读取 ObservableObject 对象的属性。例如：

```
var viewModel = kendo.observable({
  name: "John Doe"
});

var name = viewModel.get("name");
alert(name); // shows "John Doe"
```

get 也支持读取嵌套的属性，例如：

```
var viewModel = kendo.observable({
  person: {
    name: "John Doe"
  }
});

var personName = viewModel.get("person.name");
alert(personName); // shows "John Doe"
```

设置 ObservableObject 属性

使用 set 方法来设置 ObservableObject 属性，例如：

```
var viewModel = kendo.observable({
  name: "John Doe"
});

viewModel.set("name", "Jane Doe"); //set the "name" field to "Jane Doe"

var name = viewModel.get("name");
alert(name); // shows "Jane Doe"
```

同样，set 也支持设置嵌套的属性，例如：

```
var viewModel = kendo.observable({
  person: {
    name: "John Doe"
  }
});

viewModel.set("person.name", "Jane Doe");

var personName = viewModel.get("person.name");
alert(personName); // shows "Jane Doe"
```

创建关联属性（或者成为计算后属性）在应用中常常需要把某个 ViewModel 的属性重新格式成适合 View 显示的形式，在这种情况下可以通过创建一个新的关联属性来实现，比如：

```
<span data-bind="text: fullName"></span>
<script>
var viewModel = kendo.observable({
  firstName: "John",
  lastName: "Doe",
  fullName: function() {
    return this.get("firstName") + " " + this.get("lastName");
  }
});
```

```
kendo.bind($("#span"), viewModel);  
</script>
```

在这个例子中 fullName 为一关联属性，它依赖于 firstName 和 lastName，使用 set 修改 firstName 或是 LastName 后，FullName 的值也随之变化。

要注意的是 fullName 的实现，对 firstName 和 lastName 的访问，是通过 get 方法来实现的，如果使用下面的方法：

```
var viewModel = kendo.observable({  
  firstName: "John",  
  lastName: "Doe",  
  fullName: function() {  
    return this.firstName + " " + this.lastName;  
  }  
});
```

上面代码直接使用 this.firstName 来访问 ObservableObject 的属性，在这种情况下，fullName 不会跟踪 firstName 和 lastName 的变化，此时改变 firstName 和 lastName，fullName 的值不变，因此建议总是使用 get 来访问某个属性。



11



Kendo MVVM 数据绑定(一) attr



Kendo UI MVVM 数据绑定支持的绑定属性有 attr, checked, click, custom , disabled,enabled, events, html, invisible, , style, text ,value, visible , 这些属性可以绑定到 DOM 元素或是 Kendo UI 组件的属性。本篇介绍 attr 绑定。attr 支持把 ViewModel 的属性或方法绑定到 DOM 元素的某个属性, 比如设置 hyperlink 的 href 和 title 属性, image 元素的 src 或 alt 属性。其基本用法为 attr: { attribute1: field1, attribute2: field2 } 其中 attribute1 和 attribute2 为 DOM 元素的属性名称, 而 field1,field2 为 ViewModel 对象的值域(属性)的名称。比如:

```
<img id="logo" data-bind="attr: { src: imageSource, alt: imageAlt }" />
<script>
var viewModel = kendo.observable({
  imageSource: "http://www.kendoui.com/image/kendo-logo.png",
  imageAlt: "Kendo Logo"
});

kendo.bind($("#logo"), viewModel);
</script>
```

在本例中, image 元素的 src 和 alt 属性被绑定到 viewModel 对象的 imageSource 和 imageAlt 属性。当调用 kendo.bind 方法, image 元素和下面 HTML 元素等效:

```

```

此时, 如果修改 viewModel 的 imageSource 和 imageAlt 属性的值, 页面上显示的图片也随之发生变化。



注意: 如果需要绑定到 DOM 元素的 value 或 checked 属性, 你需要使用 Kendo MVVM 的 value 和 checked 绑定方法。

attr 绑定也支持设置 HTML5 数据属性绑定, 例如:

```
<div data-bind="attr: { data-foo: fooValue, data-bar: barValue }"></div>

<script>
var viewModel = kendo.observable({
  fooValue: "foo",
  barValue: "bar"
});
```

```
});  
  
kendo.bind($("#div"), viewModel);  
</script>
```



12

Kendo MVVM 数据绑定(二) Checked



Checked 绑定用在 checkbox ()或 radio button ()上。注意：checked 绑定只适用于支持 checked 的 DOM 元素，其它 DOM 元素的值可以使用 value 绑定。

多选钮(Checkedbox) checked 绑定 使用 Kendo checked 绑定到 checkbox 时，当 ViewModel 对应的值为 true, Checkbox 显示选中状态，而当用户点击 checkbox 选择状态时，对应的 ViewModel 的值也随之变化。

例如：

```
<input type="checkbox" data-bind="checked: isChecked" />
<script>
var viewModel = kendo.observable({
  isChecked: false
});

kendo.bind($("#input"), viewModel);
</script>
```

本例，因为 viewModel 的 isChecked 初始值为 false,因此 Checkbox 显示未选状态，如果此时用户点击选择该选项，那么 viewModel 的 isChecked 的值为 true 。

绑定一个数组到一组多选框

checked 绑定支持把 ViewModel 对象的一个数组属性绑定到一组多选框，选择一组多选框的某个 Checkbox，它的值被添加到 ViewModel 的数组中，反之，该值从数组中移除。

```
<input type="checkbox" value="Red" data-bind="checked: colors" />Red
<input type="checkbox" value="Green" data-bind="checked: colors" />Green
<input type="checkbox" value="Blue" data-bind="checked: colors" />Blue
<script>
    var viewModel = kendo.observable({
        colors: ["Red"]
    });

    kendo.bind($("#input"), viewModel);
</script>
```

☒ Red ☐ Green ☐ Blue

在这个例子中，第一个 checkbox 显示选择状态，是因为它的 value 值 "Red" 包含在 ViewModel 数组 colors 中，如果此时选择 Green，那么 colors 数组变为 Red 和 Green。如果去除选择 Red，则 Colors 数组只包含 Green。

单选按钮(Radio Button) checked 绑定 Kendo MVVM 只有在 ViewModel 的属性和对应的 radio button 的 value 值一致时才会显示该 Radio Button 为选中状态。

```
<input type="radio" value="Red" name="color" data-bind="checked: selectedColor" />Red
<input type="radio" value="Green" name="color" data-bind="checked: selectedColor" />Green
<input type="radio" value="Blue" name="color" data-bind="checked: selectedColor" />Blue
<script>
    var viewModel = kendo.observable({
        selectedColor: "Green"
    });

    kendo.bind($("#input"), viewModel);
</script>
```

☐ Red ☒ Green ☐ Blue



13



Kendo MVVM 数据绑定(三) Click



Click 绑定可以把由 ViewModel 定义的方法不绑定到目标 DOM 的 click 事件。当点击目标 DOM 元素时触发 ViewModel 的对应方法。例如：

使用 Click 绑定

```
<div id="view">
  <span data-bind="click: showDescription">Show description</span>
  <span data-bind="visible: isDescriptionShown, text: description"></span>
</div>
<script>
  var viewModel = kendo.observable({
    description: "Description",
    isDescriptionShown: false,
    showDescription: function (e) {
      // show the span by setting isDescriptionShown to true
      this.set("isDescriptionShown", true);
    }
  });

  kendo.bind($("#view"), viewModel);
</script>
```

Show description Description

实际上，click 绑定是 events 绑定的一个特例，下面的两种实现是等效的：

```
<span data-bind="click: clickHandler"></span>

<span data-bind="events: { click: clickHandler }"></span>
```

注：Kendo MVVM 的 DOM 事件参数 e 封装在 jQuery 的 [Event](#) 对象中。

中止事件向上传递

如果需要终止事件向上传递(event bubbling)，可以调用 stopPropagation 方法。

```
<span data-bind="click: click">Click</span>
<script>
var viewModel = kendo.observable({
  click: function(e) {
    e.stopPropagation();
  }
});

kendo.bind($("#span"), viewModel);
</script>
```

停止事件缺省处理

如果要取消某些 DOM 元素单击后的缺省处理函数，比如转到其它页面或是提交表单，为了取消这些缺省实际处理，可以调用 `preventDefault()` 方法。例如：

```
<a href="http://example.com" data-bind="click: click">Click</span>
<script>
var viewModel = kendo.observable({
  click: function(e) {
    // stop the browser from navigating to http://example.com
    e.preventDefault();
  }
});

kendo.bind($("#a"), viewModel);
</script>
```



14

Kendo MVVM 数据绑定(四) Disabled/Enabled



Disabled 和 Enabled 绑定可以根据 ViewModel 的某个属性值的 true,false 来设置 DOM 元素的 enabled 和 disabled 属性。Disabled/enabled 属性只适用于 input,select 和 textarea 元素,当这些输入元素 disabled 后,用户无法修改其值。

```
<div id="view">
<input type="text" data-bind="value: name, disabled: isNameDisabled" />
<button data-bind="click: disableInput">Disable</button>
</div>
<script>
var viewModel = kendo.observable({
  isNameDisabled: false,
  name: "John Doe",
  disableInput: function () {
    this.set("isNameDisabled", true);
  }
});

kendo.bind($("#view"), viewModel);
</script>
```



注: 对于一些非 boolean 值如: 0,null, undefined 会被看作 false ,其它值均当成 true .



15

Kendo MVVM 数据绑定(五) Events



本篇和 [Kendo UI 开发教程\(14\): Kendo MVVM 数据绑定\(三\) Click](#) 类似，为事件绑定的一般形式。Events 绑定支持将 ViewModel 的方法绑定到 DOM 元素的事件处理（如鼠标事件）。例如：

```
<div id="view">
  <span data-bind="events: { mouseover: showDescription, mouseout: hideDescription }">Show description</span>
  <span data-bind="visible: isDescriptionShown, text: description"></span>
</div>
<script>
  var viewModel = kendo.observable({
    description: "Description",
    isDescriptionShown: false,
    showDescription: function (e) {
      // show the span by setting isDescriptionShown to true
      this.set("isDescriptionShown", true);
    },
    hideDescription: function (e) {
      // hide the span by setting isDescriptionShown to false
      this.set("isDescriptionShown", false);
    }
  });

  kendo.bind($("#view"), viewModel);
</script>
```

Show description Description

实际上，click 绑定是 events 绑定的一个特例，下面的两种实现是等效的：

```
<span data-bind="click: clickHandler"></span>

<span data-bind="events: { click: clickHandler }"></span>
```

注：Kendo MVVM 的 DOM 事件参数 e 封装在 jQuery 的 [Event](#) 对象中。

中止事件向上传递

如果需要终止事件向上传递(event bubbling)，可以调用 stopPropagation 方法。

```
<span data-bind="click: click">Click</span>
<script>
var viewModel = kendo.observable({
  click: function(e) {
    e.stopPropagation();
  }
});

kendo.bind($("#span"), viewModel);
</script>
```

停止事件缺省处理

如果要取消某些 DOM 元素单击后的缺省处理函数，比如转到其它页面或是提交表单，为了取消这些缺省实际处理，可以调用 `preventDefault()` 方法。例如：

```
<a href="http://example.com" data-bind="click: click">Click</span>
<script>
var viewModel = kendo.observable({
  click: function(e) {
    // stop the browser from navigating to http://example.com
    e.preventDefault();
  }
});

kendo.bind($("#a"), viewModel);
</script>
```




16



Kendo MVVM 数据绑定(六) Html



Html 绑定可以使用 ViewModel 的属性来设置 DOM 元素的 innerHTML 属性。如果 ViewModel 的属性的数据类型不是字符串, 比如 (Text, Number 或者 Date), 那么会调用 toString() 方法, 将这些值转为字符串。注意: 如果需要设置 input, textarea 或是 select 的值, 需要使用 value 绑定。例如:

```
<span data-bind="html: name"></span>
<script>
var viewModel = kendo.observable({
  name: "John Doe"
});

kendo.bind($("#span"), viewModel);
</script>
```

这个结果显示如下 html 元素:

```
<span>John Doe</span>
```

如果 ViewModel 的值包含 HTML 标记, 这些标记和作为最后结果显示出来, 比如:

```
<span data-bind="html: name"></span>
<script>
var viewModel = kendo.observable({
  name: "<strong>John Doe</strong>"
});

kendo.bind($("#span"), viewModel);
</script>
```

显示如下:

John Doe

如果只想显示 ViewModel 的值, 可以使用 text 绑定。



17

Kendo MVVM 数据绑定(七) Invisible/Visible



Invisible/Visible 绑定可以根据 ViewModel 的某个属性来显示/隐藏 DOM 元素。例如：

```
<div id="view">
  <div data-bind="invisible: isInvisible">some content

  </div>
  <button data-bind="click: show">Show</button>
</div>
<script>
var viewModel = kendo.observable({
  isInvisible: true,
  show: function () {
    this.set("isInvisible", false);
  }
});

kendo.bind($("#view"), viewModel);
</script>
```

Show



some content

Show



18



Kendo MVVM 数据绑定(八) Style



Style 绑定可以通过 ViewModel 绑定到 DOM 元素 CSS 风格属性，例如：

```
<span data-bind="style: {color: priceColor, fontWeight: priceFontWeight},
    text: price"></span>

<script>
var viewModel = kendo.observable({
    price: 42,
    priceColor: function() {
        var price = this.get("price");

        if (price <= 42) {
            return "#00ff00";
        } else {
            return "#ff0000";
        }
    },
    priceFontWeight: function() {
        var price = this.get("price");

        if (price <= 42) {
            return "bold";
        } else {
            return ""; //will reset the font weight to its default value
        }
    }
});

kendo.bind($("#span"), viewModel);
</script>
```

这个例子显示：

```
<span style="color: #00ff00; font-weight: bold">42</span>
```

42

要注意的是 CSS 属性的名称，如果 CSS 名称中含有连字符（-），比如 font-weight, font-size, background-color 等，在使用时需要省略到连字符，使用 camel 风格的命名，如 fontWeight, fontSize, backgroundColor 等。



19



Kendo MVVM 数据绑定(九) Text



Text 绑定可以使用 ViewModel 来设置 DOM 元素的文本属性, 如果需要设置 input, textarea, 或 select 的显示, 需要使用 value 属性。

```
<span data-bind="text: name"></span>
<script>
var viewModel = kendo.observable({
  name: "John Doe"
});

kendo.bind($("#span"), viewModel);
</script>
```

本例输出:

John Doe

如果 ViewModel 的值包含 html 标记, 这些标记会直接显示出 html 标记, 比如:

```
<span data-bind="text: name"></span>
<script>
var viewModel = kendo.observable({
  name: "<strong>John Doe</strong>"
});

kendo.bind($("#span"), viewModel);
</script>
```

John Doe

如果需要以粗体显示, 则可以使用前面介绍的 [html](#) 绑定。



20

Kendo MVVM 数据绑定(十) Source



Source 绑定可以把 ViewModel 的值和由 Kendo 模板定义的目标元素绑定，如果 ViewModel 的值发生变化，被绑定的目标元素也随之发生变化。模板由属性 data-template 指定，它的值为某个 script 定义的模板的 id。如果没有指明模板，则根据元素的标记使用缺省的模版。

Source 绑定到数组

当 ViewModel 的值为一个数组时，那么通过 Source 绑定到模板时，会把数组中每个元素逐个应用到模板，最后的输出为应用这些模板的结果的综合。添加删除数组中的内容，显示的内容也随之变化。

注：绑定到 ViewModel 数组时，Source 指明的为单个跟元素名称，例如：

```
<ul data-template="ul-template" data-bind="source: products">
</ul>
<script id="ul-template" type="text/x-kendo-template">
  <li>
    id: <span data-bind="text: id"></span>
    name: <span data-bind="text: name"></span>
  </li>
</script>
<script>
var viewModel = kendo.observable({
  products: [
    { id: 1, name: "Coffee" },
    { id: 2, name: "Tea" },
    { id: 3, name: "Juice" }
  ]
});

kendo.bind($("#ul"), viewModel);
</script>
```

这个例子会输出三个 li 元素 - 每个对应到 products 数组中一个元素，下面为输出的 HTML 内容：

```
<ul>
  <li>
    id: <span>1</span>
    name: <span>Coffee</span>
  </li>
  <li>
    id: <span>2</span>
    name: <span>Tea</span>
  </li>
  <li>
    id: <span>3</span>
    name: <span>Juice</span>
  </li>
</ul>
```

```
</li>
</ul>
```

- **id: 1 name: Coffee**
- **id: 2 name: Tea**
- **id: 3 name: Juice**

如果 ViewModel 绑定的数组的内容为简单类型（如数字，字符串，日期），这时在模板中需要使用 ” this ” 关键字来引用当前数组项：

```
<ul data-template="ul-template" data-bind="source: products">
</ul>
<script id="ul-template" type="text/x-kendo-template">
  <li data-bind="text: this"></li>
</script>
<script>
var viewModel = kendo.observable({
  products: [ "Coffee", "Tea", "Juice" ]
});

kendo.bind($("#ul"), viewModel);
</script>
```

输出内容如下：

```
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Juice</li>
</ul>
```

Source 绑定到非数组

source 绑定也支持绑定到非数组值，此时在模板中引用到 ViewModel 的某个属性，最终的结果为模板使用 ViewModel 后的结果。

```
<div data-template="div-template" data-bind="source: person">
  <script id="div-template" type="text/x-kendo-template">
    Name: <span data-bind="text: name"></span>
  </script>
</div>
<script>
var viewModel = kendo.observable({
  person: {
    name: "John Doe"
  }
});

kendo.bind($("#div"), viewModel);
</script>
```

输出：

```
<div>
  Name: <span>John Doe</span>
</div>
```

你也可以直接绑定到 ViewModel 对象本身，此时可以使用把 source 的值设置为 “this”，例如：

```
<div data-template="div-template" data-bind="source: this">
  <script id="div-template" type="text/x-kendo-template">
    Name: <span data-bind="text: name"></span>
  </script>
</div>
<script>
var viewModel = kendo.observable({
  name: "John Doe"
});

kendo.bind($("#div"), viewModel);
</script>
```

结果如下：

```
<div>  
Name: <span>John Doe</span>  
</div>
```

Source 绑定 Select 元素

当数组绑定到 select 元素时，就创建多个 option 元素。

```
<select data-bind="source: colors"></select>
<script>
var viewModel = kendo.observable({
  colors: [ "Red", "Green", "Blue" ]
});

kendo.bind($("#select"), viewModel);
</script>
```

输出的 HTML 元素如下：

```
<select>
  <option>Red</option>
  <option>Green</option>
  <option>Blue</option>
</select>
```

select 元素也可以绑定到 JavaScript 对象数组（非简单类型），此时可以同时指定 data-text-field, data-value-field 用来指定 option 元素的 value 和 text 属性，例如：

```
<select data-text-field="name" data-value-field="id"
  data-bind="source: products"></select>
<script>
var viewModel = kendo.observable({
  products: [
    { id: 1, name: "Coffee" },
    { id: 2, name: "Tea" },
    { id: 3, name: "Juice" }
  ]
});

kendo.bind($("#select"), viewModel);
</script>
```

输出如下：

```
<select>  
  <option value="1">Coffee</option>  
  <option value="2">Tea</option>  
  <option value="3">Juice</option>  
</select>
```




21

Kendo MVVM 数据绑定(十一) Value



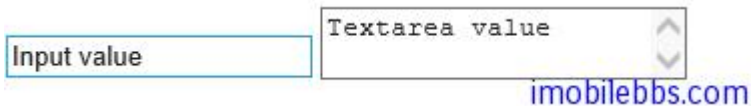
Value 绑定可以把 ViewModel 的某个属性绑定到 DOM 元素或某个 UI 组件的 Value 属性。当用户修改 DOM 元素或 UI 组件的值时，绑定的 ViewModel 的值也随之发生改名。同样，如果 ViewModel 的值发生变化，对应的 UI 也会发生变化。注：Value 绑定只可以用在支持 Value 属性的 DOM 元素或 UI 组件。支持 Value 绑定的元素有 input,textarea 和 select, 支持value绑定的 UI 组件有 AutoComplete, DropDownList, ComboBox, DatePicker, TimePicker, NumericTextBox 和 Slider. 如果你需要设置 DOM 元素或 UI 组件的文本或是 HTML 内容，请使用 text 和 html 绑定。对于Checkboxes () 或 radio button()请使用 checked 绑定。

Input 和 textarea Value 绑定

```
<div id="view">
  <input data-bind="value: inputValue" />
  <textarea data-bind="value: textareaValue"></textarea>
</div>
<script>
var viewModel = kendo.observable({
  inputValue: "Input value",
  textareaValue: "Textarea value"
});

kendo.bind($("#view"), viewModel);
</script>
```

上面代码当调用 bind 方法后, input 元素显示 inputValue 的值, 而 textarea 显示 textareaValue 的值。如果用户修改 input 或 textarea 的值, 对应的 inputValue 和 textareaValue 也随之变化。



缺省情况下, Value 绑定依赖于 DOM 的 change 事件, 也就是当 DOM 元素失去焦点时触发该事件, UI 的变化实现对 ViewModel 的更新。然而可以通过修改 data-value-update 属性来使用不同的 DOM 事件, 比如 keyup 或 keypress 事件 (不可使用 keydown 事件, 只是因为 keydown 事件 DOM 元素的 value 尚未发生变化)。

```
<div id="view">
  <input data-value-update="keyup" data-bind="value: inputValue" />
</div>
<script>
var viewModel = kendo.observable({
  inputValue: "Input value"
});

kendo.bind($("#view"), viewModel);
</script>
```

Select 元素绑定 value

当 Select 元素使用了预定义的选项时，Kendo MVVM 将根据 ViewModel 的值把和 ViewModel 值相同的 option 选项设定为选中状态。

```
<select data-bind="value: selectedColor">
  <option value="red">Red</option>
  <option value="green">Green</option>
  <option value="blue">Blue</option>
</select>
<script>
var viewModel = kendo.observable({
  selectedColor: "green"
});

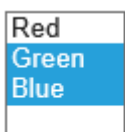
kendo.bind($("#select"), viewModel);
</script>
```

在本例中，第二个选项 (Green)被选中，这是因为它的 value 和 selectedColor 相同。UI修改选项也会更新 selectedColor 的值。如果 option 元素没有定义 value，那么使用 option 的 text 属性。

如果 select 支持多项选择，此时对应的 ViewModel 的属性也应该为一个数组。例如：

```
<select data-bind="value: selectedColors" multiple="multiple">
  <option>Red</option>
  <option>Green</option>
  <option>Blue</option>
</select>
<script>
var viewModel = kendo.observable({
  selectedColors: ["Blue","Green"]
});

kendo.bind($("#select"), viewModel);
</script>
```





22

单页面应用(一)概述



Kendo 单页面应用(Single-Page Application,缩写为 SPA)定义了一组类用于简化 Web 应用(Rich Client)开发,最常见的单页面应用为 Gmail 应用,使用单页面可以给用户有使用桌面应用的用户体验。Kendo 的 Route 类负责跟踪应用的当前状态和支持在应用的不同状态之间切换。Route 通过 Url 的片段功能(#url)和浏览器的浏览历史功能融合在一起。从而可以支持把应用的某个状态作为书签添加到浏览器中。Route 也支持通过代码在应用的不同状态之间切换。View 和 Layout 类用于 UI 的显示。UI 事件和数据绑定可以通过 MVVM 或 data 初始化属性来完成。下面为一个最简单的 SPA 应用框架。

```
<div id="app"></div>

<script id="index" type="text/x-kendo-template">
  Hello <span data-bind="text: foo"></span>
</script>

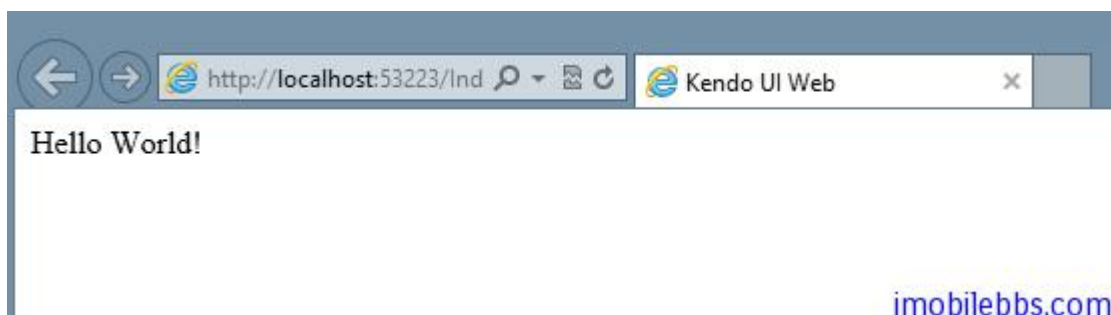
<script>
  var index = new kendo.View(
    "index", // the id of the script element that contains the view markup
    { model: kendo.observable({ foo: "World!" }) }
  );

  var router = new kendo.Router();

  router.route("/", function() {
    index.render("#app");
  });

  $(function() {
    router.start();
  });
</script>
```

运行这个应用,显示“Hello, World”。





23

单页面应用(二) Router 类



Route 类负责跟踪应用的当前状态和支持在应用的不同状态之间切换。Route 通过 Url 的片段功能 (#url)和浏览器的浏览历史功能融合在一起。从而可以支持把应用的某个状态作为书签添加到浏览器中。Route 也支持通过代码在应用的不同状态之间切换。

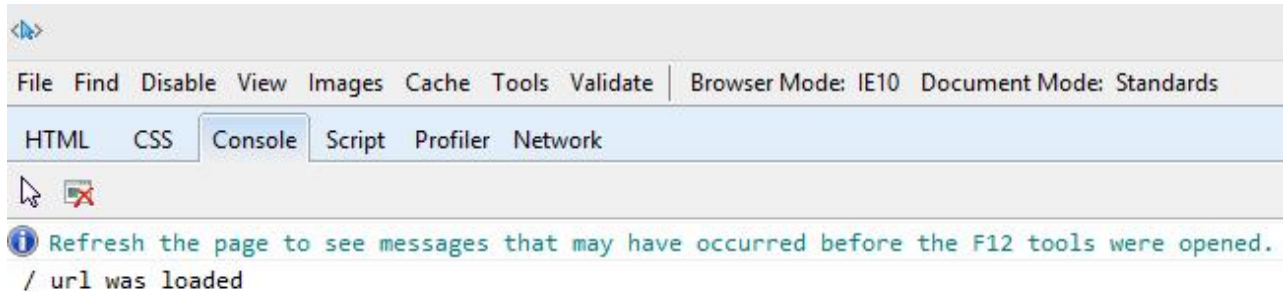
Router 根路径回调函数

```
<script>
  var router = new kendo.Router();

  router.route("/", function() {
    console.log("/ url was loaded");
  });

  $(function() {
    router.start();
  });
</script>
```

缺省情况下，如果 URL fragment 为空，将使用缺省的 “/” 的根路径，此时对于的回调函数被调用，不管初始 URL 是什么，这个初始化的回调函数总会调用。如果使用 IE，按 F12 可以打开 Developer Window，选择 Console 可以看到 console.log 的打印信息。



参数

Router 支持 bound parameters, optional segments, 和 route globbing, 类似于绑定参数, 可选参数, 匹配符匹配参数等。例如: **绑定参数**

```
<script>
  var router = new kendo.Router();

  router.route("/items/:category/:id", function(category, id) {
    console.log(category, "item with", id, " was requested");
  });

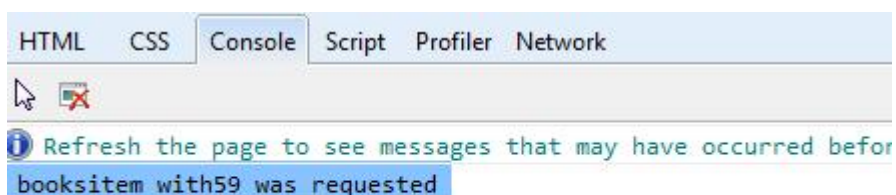
  $(function() {
    router.start();

    // ...

    router.navigate("/items/books/59");
  });
</script>
```

当运行这个页面时, 注意地址栏中的地址为:

<http://localhost:53223/Index.html#/items/books/59 -> #/items/books/59>



imobilebbs.com

可选参数

如果 URL 的部分参数为可选的, 此时 Route 的规则为使用 "()" ,将可选参数放在括号内。

```
<script>
  var router = new kendo.Router();

  router.route("/items(/:category)/(:id)", function(category, id) {
```

```

    console.log(category, "item with", id, " was requested");
  });

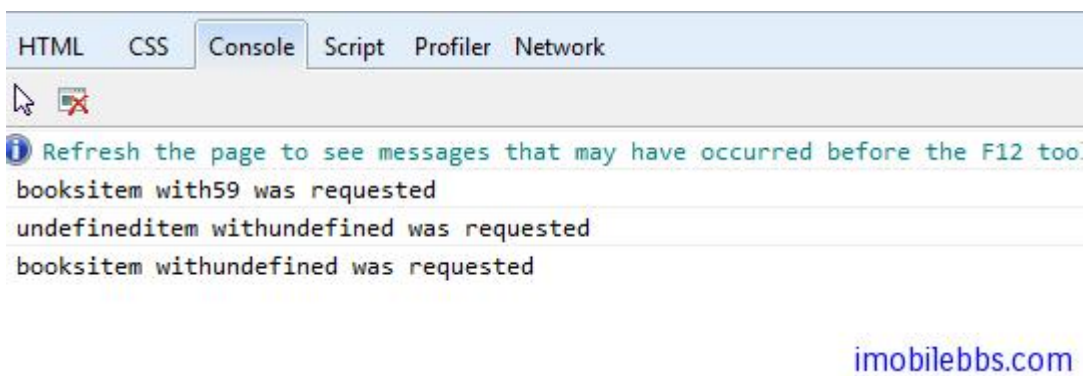
$(function() {
  router.start();

  // ...
  router.navigate("/items/books/59");

  // ...
  router.navigate("/items");

  // ...
  router.navigate("/items/books");
});
</script>

```



imobilebbs.com

使用 × 通配符匹配参数

```

<script>
  var router = new kendo.Router();

  router.route("/items/*suffix", function(suffix) {
    console.log(suffix);
  });

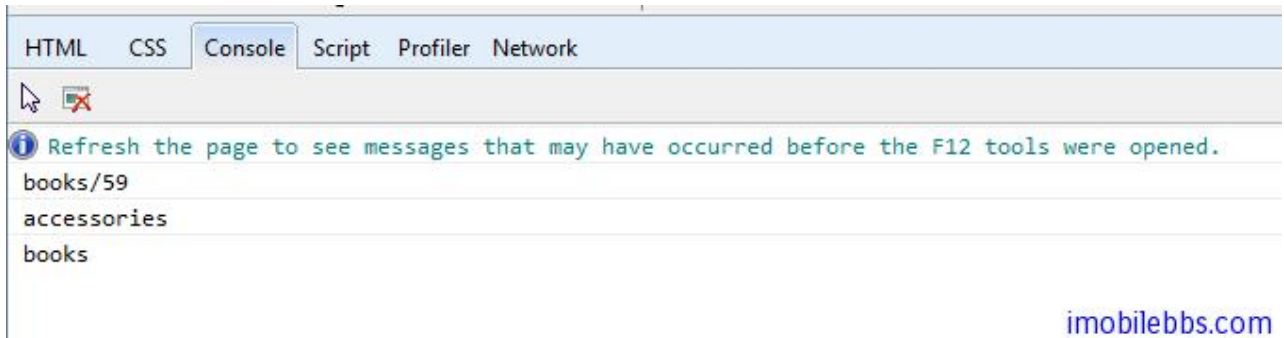
  $(function() {
    router.start();

    // ...
    router.navigate("/items/books/59");

    // ...
    router.navigate("/items/accessories");
  });

```

```
// ...  
router.navigate("/items/books");  
});  
</script>
```



页面切换

navigation 方法可以用来切换应用，对应的路径的回调方法被调用, navigation 方法修改 URL 的 fragment 部分(#后面部分)。比如：

```
<a href="#/foo">Foo</a>

<script>
  var router = new kendo.Router();

  router.route("/foo", function() {
    console.log("welcome to foo");
  });

  $(function() {
    router.start();
    router.navigate("/foo");
  });
</script>
```

这个例子，将在地址栏显示 `http://xxx/index.html#/foo`。如果对应的路径不存在，Router 类触发 `routeMissing` 事件，并把 URL 作为参数传入。

```
<script>
var router = new kendo.Router({ routeMissing: function(e) { console.log(e.url) } });

$(function() {
  router.start();
  router.navigate("/foo");
});
</script>
```

你可以通过 `change` 事件来截获这种页面之间的切换，然后调用 `preventDefault` 阻止页面切换。

```
<script>
var router = new kendo.Router({
  change: function(e) {
    console.log(url);
    e.preventDefault();
  }
});
```

```
$(function() {  
  router.start();  
  router.navigate("/foo");  
});  
</script>
```



24

单页面应用(三) View



iew 为屏幕上某个可视部分，可以处理用户事件。View 可以通过 HTML 创建或是通过 script 元素。缺省情况下 View 将其所包含的内容封装在一个 Div 元素中。Kendo 创建 View 有两种方式：

使用 HTML 字符串创建 View

```
<script>  
    var index = new kendo.View('<span>Hello World!</span>');  
</script>
```

或是使用

使用 Script 模板创建 View

```
<script id="index" type="text/x-kendo-template">
  <span>Hello World!</span>
</script>

<script>
  var index = new kendo.View('index');
</script>
```

显示 View 内容

使用上述两种方法创建 View，可以使用 view 的 render 方法来显示，render 参数支持 jQuery 选择器，表示将 View 的内容显示到指定的 DOM 元素中或添加到指定的 DOM 元素。例如：显示 View

```
<div id="app"></div>

<script>
  var index = new kendo.View('<span>Hello World!</span>');

  index.render("#app");
</script>
```



本例将 View 的内容显示到 div 元素中，如果需要向某个 DOM 元素中添加 View 的内容，可以使用 append 方法。例如：

```
<div id="app"></div>

<script>
  var index = new kendo.View('<span>Hello World!</span>');

  $("#app").append(index.render());
</script>
```

集成 MVVM

在创建 View 时，可以传入一个 model 对象，此时 model 可以和创建的 view 绑定。例如：

```
<div id="app"></div>
<script id="index" type="text/x-kendo-template">
<div>Hello <span data-bind="text:foo"></span>!</div>
</script>

<script>
  var model = kendo.observable({ foo: "World" });
  var index = new kendo.View('index', { model: model });
  index.render("#app");
</script>
```



T

25



单页面应用(四) Layout



ayout 继承自 View，可以用来包含其它的 View 或是 Layout。下面例子使用 Layout 来显示一个 View

```
<div id="app"></div>

<script>
  var view = new kendo.View("<span>Foo</span>");

  var layout = new kendo.Layout("<header>Header</header><section id='content'></section><footer></footer>");

  layout.render($("#app"));

  layout.showIn("#content", view);
</script>
```

这个例子创建一个 Layout 对象，这个 Layout 含有一个 Header，一个 Content 和一个 footer，其中 Content 以 setion 元素定义，作为一个 Placeholder，实际应用时可以使用某个 View 来替换。

Header Foo

Layout 本身也是一个 View，因此在 showIn 方法中也可以传入一个 Layout 对象，从而实现 Layout 的嵌套支持。

Layout 定义多个 View 统一的布局，定义了 View 的 Placeholder，因此在应用中可以实现 View 的切换。例如：

```
<div id="app"></div>

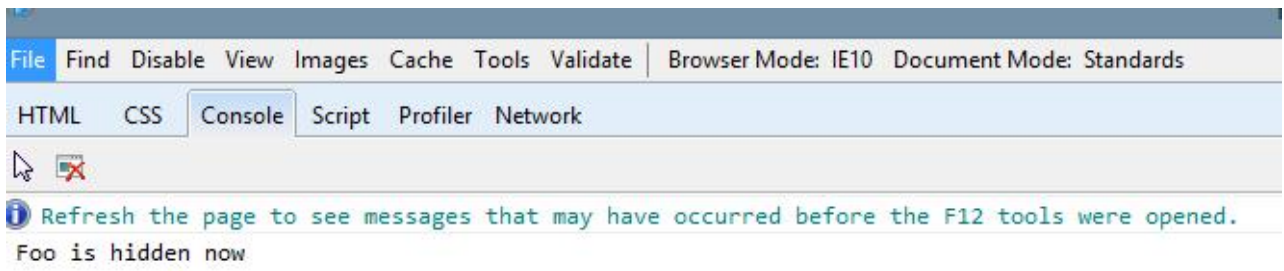
<script>
  var foo = new kendo.View("<span>Foo</span>", { hide: function() { console.log("Foo is hidden now"); }});
  var bar = new kendo.View("<span>Bar</span>");

  var layout = new kendo.Layout("<header>Header</header><section id='content'></section><footer></footer>");

  layout.render($("#app"));

  layout.showIn("#content", foo);
  layout.showIn("#content", bar);
</script>
```

这段代码首先显示”foo”，然后很快切换到显示“bar”。这可以通过检查 log 来确认：



imobilebbs.com



26

移动应用开发简介



Kendo UI 支持开发 Web 应用，前面介绍的 SPA，也支持开发移动应用，至于使用 HTML5 + JavaScript + CSS 开发移动是不是一个好的选择不在本文的讨论之中。Kendo UI Mobile 提供了一种快速开发跨手机平台的方法（Kendo UI 可以使得这种 Web 应用在界面上看起来和本地应用非常类似）。如果你的移动应用需要数据的支持，了解一些 [JSON](#) 方面的知识也是必须的。借助于 PhoneGap 等工具可以 HTML5 打包成移动平台的本地应用，并支持使用 JavaScript 访问一些平台相关的功能，如 GPS，Camera 等功能，有兴趣的可以参考相关文档。

下面三点为构成 Kendo 移动应用的几个组成部分：

1. [Application](#): Kendo 移动应用的主应用类，用来管理应用部分部分之间切换，应用页面历史，加载 View 以及其它一些重要的移动应用相关的任务。
2. [Layout](#): 定义移动应用 UI 的布局，类似于 Web 应用的 MasterPage，主要可以用来定义不同 View 之间一些公用的部分，比如菜单。
3. [Views](#): 移动应用的每个页面，每个应用包含一个或多个页面。

Layouts 和 View 使用 HTML 来定义，而 Application 为 JavaScript。下面的步骤给出了编写 Kendo UI 移动应用的基本步骤。

第一步： 创建 HTML 页面

Kendo UI 移动应用可以使用简单的 HTML 页面来创建，这里我们创建一个简单的 index.html 如下：

```
<!DOCTYPE html>
<html>
<head>
  <title>My App</title>
  <!--TODO: Add CSS links-->
</head>
<body>

  <!--TODO: Add JavaScript referneces-->
</body>
</html>
```

第二步：添加 Kendo UI Mobile 的引用

添加 Kendo UI Mobile CSS 和 Javascript 的引用。

```
<!DOCTYPE html>
<html>
<head>
  <title>My App</title>

  <link href="css/kendo.mobile.all.min.css" rel="stylesheet" />
</head>
<body>

  <script src="js/jquery.min.js"></script>
  <script src="js/kendo.all.min.js"></script>
</body>
</html>
```

第三步：定义应用布局文件

Layout 为应用 UI 的模板，应用所有的 View 的内容都使用模板来显示，一个 Layout 可以包含任意的内容，通常它包含有标题头和任务栏。比如下面的 Layout:

```
<!DOCTYPE html>
<html>
<head>
  <title>My App</title>

  <link href="css/kendo.mobile.all.min.css" rel="stylesheet" />
</head>
<body>
  <section data-role="layout" data-id="default">
    <header data-role="header">
      <div data-role="navbar">My App</div>
    </header>
    <!--View content will render here-->
    <footer data-role="footer">
      <div data-role="tabstrip">
        <a href="#home">Home</a>
      </div>
    </footer>
  </section>

  <script src="js/jquery.min.js"></script>
  <script src="js/kendo.all.min.js"></script>
</body>
</html>
```

代码中使用 data-role 属性，这个属性用来建立 HTML 和 Kendo UI mobile 库之间的联系。因此

```
<section data-role="layout" data-id="default">
```

在应用初始化时，这部分定义将转换为 Layout 定义。data-id 为该 Layout 的 id，后面定义的 view 可以通过这个 id 来引用某个 layout。最后，为完整起见，这段代码还使用了 NavBar 和 TabStrip 两个用在移动应用中的 UI 组件。

第四步：构造 View

创建好 Layout 之后，应用至少要创建一个 View 用来显示，大部分应用包含有多个 View，这里我们创建一个简单的 View 如下：

```
<!DOCTYPE html>
<html>
<head>
  <title>My App</title>

  <link href="css/kendo.mobile.all.min.css" rel="stylesheet" />
</head>
<body>
  <div id="home" data-role="view" data-layout="default">
    Hello Mobile World!
  </div>

  <section data-role="layout" data-id="default">
    <header data-role="header">
      <div data-role="navbar">My App</div>
    </header>
    <!--View content will render here-->
    <footer data-role="footer">
      <div data-role="tabstrip">
        <a href="#home">Home</a>
      </div>
    </footer>
  </section>

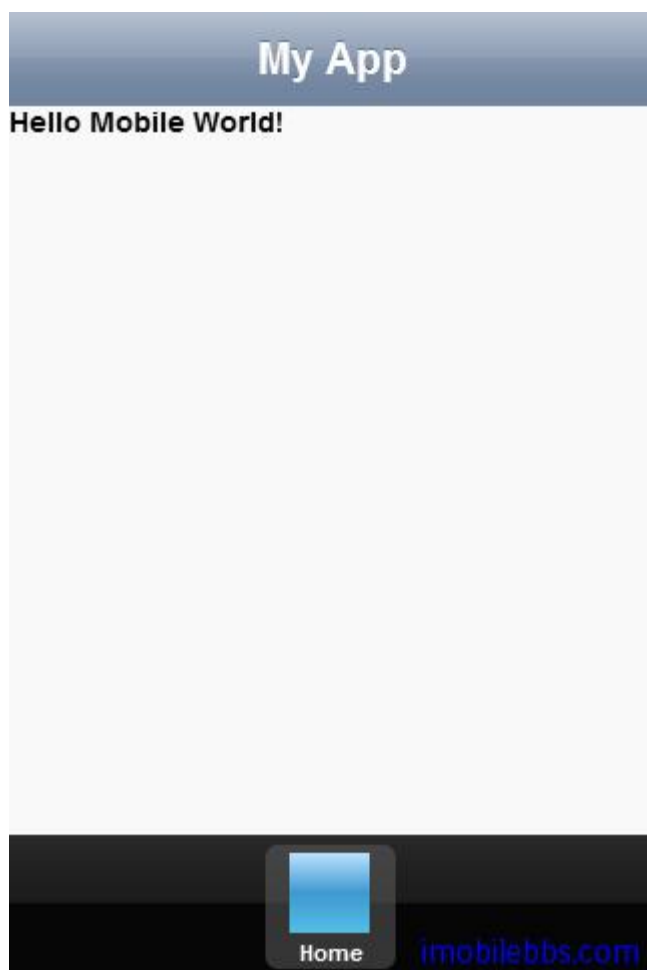
  <script src="js/jquery.min.js"></script>
  <script src="js/kendo.all.min.js"></script>
</body>
</html>
```

View 定义使用 data-role 属性 “view”，data-layout 定义使用那个 layout。

第五步：初始化移动应用

前面定义了一些 HTML 元素，还没有使用任何 JavaScript，使用下面一行代码，可以使得前面定义的 HTML 变得和本地应用类似：

```
<script>
  var app = new kendo.mobile.Application();
</script>
```



样一个简单的移动应用就出现了，Kendo UI 缺省情况下使用 iOS 界面（如上图），在手机上运行会根据手机平台的不同选择合适的界面风格，你也可以通过指定平台类型，比如：

```
<script>
  var app = new kendo.mobile.Application(document.body,
  {
    platform:'android'
```

```
});  
</script>
```

来测试你的应用在不同平台上显示，也可以根据平台的不同，对应用做些调整，比如：

```
<div data-role="layout" data-id="foo" data-platform="ios">  
  <div data-role="header">iOS App</div>  
</div>  
  
<div data-role="layout" data-id="foo" data-platform="android">  
  <div data-role="header">Android App</div>  
</div>
```

注意的是 data-platform 属性目前只支持在 layout 中使用。

极客学院

jikexueyuan.com

中国最大的IT职业在线教育平台



更多信息请访问 

<http://wiki.jikexueyuan.com/project/kendo-ui-development-tutorial/>