MontageJS模板声明语法

这个文档主要讲解MontageJS序列化，反序列化格式，以及在模板中使用的声明语法。

MontageJS通过声明的方式定义对象、组件、属性、组件的数据绑定和对象与DOM元素之间的关联。MontageJS使用JavaScript的对象定义（JSON）作为序列化的格式。在运行时，MontageJS解析这些JSON内容，反序列化它们为JavaScript，然后在浏览器中执行。

JSON 概述

JSON是一个用来序列化结构数据的文本格式，可以包括六种数据类型：

- 四种基本类型：字符串，数字，布尔，和null
- 两种结构类型：对象和数组

在JSON格式中，一个对象定义为一个无序的零个或者多个键/值对。键是一个字符串，值是JSON基础类型字符串，数字，布尔，和null其中一个。数组类型的值是用中括号包括零个或者多个基础类型（或者结构类型）数据。数组中的每个元素用逗号分隔。

比如，下面的JSON定义了一个名字是 `anObject` 的对象，对象包括三个属性：

- 一个字符串类型 `id`
- 一个数组类型 `colors`
- 一个布尔值类型 `readyState`

```
"anObject": {
    "id": "123asd",
    "colors": [ "red", "green", "blue"],
    "readyState": false
}
```

除了这些标准的数据类型，MontageJS还为能够序列化复杂对象提供了几种特殊类型。这些类型包括引用同一个模板中的其它对象，DOM元素关联，函数，和正则表达式。

声明例子

以下是一个简单（完整）的MontageJS应用，全部定义在一个HTML文档中。这个例子向你展示在MontageJS中如何使用声明以及为什么它很方便使用。

```
<html>
   <script src="../../montage.js"></script>
   <script type="text/montage-serialization">
   {
       "firstName": {
           "prototype": "digit/ui/text-field.reel",
           "properties": {
               "element": {"#": "fName"}
           }
        }
   }
   </script>
   <body>
    <input data-montage-id="fName"></input>
   <body>
</html>
```

需要注意以下几点：

- HTML body中包含一个 `<input>` 标签，用custom data-attribute data-montage-id 标记为 `fName`
- 在头部定义了类型为 `text/montage-serialization` 的 `<script>` 区块。文档中所有的MontageJS序列化对象包含在里面。
- 在HTML模板中定义了一个名字为 `firstName` 的TextField组件对象。在表单中组件的原型id("digit/ui/text-field.reel")，是为了在运行时MontageJS能够根据id创建相应的组件

组件。

- `properties` 在MontageJS组件中是一个非常重要的属性，它负责把HTML元素和对应的组件对象关联在一起。在这个例子中TextField组件的 `element` 属性设置为 `<input>` 标签的ID" `fName` "。这是MontageJS序列化提供一种特殊JSON格式，让HTML元素和对象关联在一起。键值是一个井号（"#"），值是HTML元素的ID。
- MontageJS从一个.reel结尾的目录中加载组件。模块系统会重定向 `require("x.reel")` 为 `require("x.reel/x")` 。

声明中的Owner

在MontageJS模板声明中可以定义一个可选的对象"owner"。这个特殊的 `owner` 对象功能是文档的控制器。例如，下面的代码就创建一个新的模块(main.js)，模块名是 `Main` 。

```
// Module: main.js
// Exported object name: Main
var Component = require("montage/ui/component").Component;
//
exports.Main = Component.specialize({
// Prototype methods and properties
})
<script type="text/montage-serialization">
{
   "owner": {
       "prototype": "main",
       "properties": {
           "element": {"#": "main"}
       }
    }
}
</script>
```

声明格式

下面部分开始讲解object-dependent声明。

序列化自定义对象

为了序列化自定义JavaScript对象和MontageJS组件，需要定义JSON对象并且设置 `prototype` 属性。这个属性定义用什么组建来实例化对象。

例如，下面的代码片段定义了一个Button组件：

```
<script type="text/montage-serialization">
{
    "loginButton": {
        "prototype": "digit/ui/button.reel"
    }
}
</script>
```

在运行时，MontageJS把这些声明解析为以下的JavaScript然后运行：

```
var Button = require("digit/ui/button.reel").Button;
```

注意，在声明中的对象标签（例如上面例子中的" `loginButton` "）只能够通过owner组件的 `templateObjects` 属性来获取。

在声明中你可以设置对象的初始属性值，在声明中添加一个 `properties` 属性。例如，按钮组件可以设置显示在按钮上面的文本内容 `value` 。下面的代码设置按钮组件的 `value` 属性值为"Click me"：

```
"loginButton": {
    "prototype": "digit/ui/button.reel",
    "properties": {
       "value": "Click me"
    }
}
```

关联DOM元素

你可以在MontageJS声明中通过一种特殊的JSON对象表示方法将声明中的对象和DOM元素关联在一起。关联DOM元素的方法是把DOM元素的ID设置为组件的 `element` 属性。

用以下的语法可以通过元素的ID关联， `elementID` 是当前文档中DOM元素的ID值：

`{"#": "elementID"}`

例如，下面模板定义中 `Button` 组件的 `element` 属性值设置为 `<div>` 的ID `loginButton` ：

```
// index.html
<html>
 <script src="../../montage.js"></script>
 <script type="text/montage-serialization">
 {
      "loginBtn": {
        "prototype": "montage/ui/button.reel",
        "properties": {
            "element": {"#": "loginButton"}
        }
      }
 }
 </script>
 <body>
      <div data-montage-id="loginButton" class="Text">Click to ente
r</div>
 <body>
</html>
```

引用其它对象

很多时候你需要在模板定义中的一个对象中引用当前模板中的另外一个对象。例如，在模板定义中定义了一个MontageJS按钮然后你想在控制器（owner）对象中引用它。

用以下的JSON语法可以实现通过ID引用对象。在这个例子中，`objectLabel`是对象的标签。

`{"@": "objectLabel"}`

首先在owner中定义一个用来引用对象的变量。owner对象是一个在JavaScript文件main.js中自定义的对象Main。Main组件定义了一个变量`loginButton`，然后把模板定义中的按钮对象赋值给它。 定义的变量可以在Main组件的其它任何地方使用，比如在组件加入文档之后会自动调用的`enterDocument()`方法中。 在这个例子中，我们在`enterDocument()`回调方法中为按钮对象添加一个事件监听器。在事件处理函数中向JavaScript控制台输出一个消息。

```
// Module: main.js
// Name: Main
var Component = require("montage/ui/component").Component;
exports.Main = Component.specialize({
    loginButton: {
        value:null
    },
    handleAction: {
        value: function(event) {
            console.log("Button event handled...");
            // Do login stuff...
        }
    },
    enterDocument: {
        value: function(firstTime) {
            if (firstTime) {
                this.loginButton.addEventListener("action", this)
            }
        }
    }
});
```

接下来，在HTML文档中定义按钮和Main两个组件。在第10行，按钮对象"`loginBtn`"

被赋值给Main组件的"`loginButton`"属性。

```
<html>
 <script src="../../montage.js"></script>
 <script type="text/montage-serialization">
 {
    "owner": {
        "prototype": "ui/main.reel"
        "properties": {
            "element": {"#": "main"},
            "loginButton": {"@": "loginBtn"}
        }
    },
    "loginBtn": {
        "prototype": "digit/ui/button.reel",
        "properties": {
            "element": {"#": "buttonDiv"}
        }
    }
 }
 </script>
<body>
    <div id="main">
        <div data-montage-id="buttonDiv" class="Text">Click to ente
r</div>
    </div>
<body>
</html>
```

模板声明中数据绑定

你可以在模板声明中定义事件监听器和数据绑定。为了更容易理解声明中的绑定语法，先讲解下JavaScript数据绑定函数`Object.defineBinding()`。这个函数接收三个参数:

- 绑定目标对象

- 绑定目标对象
- 绑定目标对象的属性名
- 绑定方向和源路径

在模板定义中是通过JSON对象实现绑定，所以你只需要定义JSON对象。

```
"bindings": {
       "propertyName": {"<-": "@targetObject.key.path.of.property"}
    }

Above can also be defined in JavaScript:

    this.defineBinding("propertyName", {
    "<-": "targetObject.key.path.of.property"
});

// or

this.defineBindings({
    "propertyName": {
        "<-": "targetObject.key.path.of.property"
    }
});
```

以下的例子就是在声明中定义数据绑定。第一个滑块的值被绑定到第二个滑块的值。默认的数据绑定方式是单方向，也就是源属性的值发生变化之后，这个变化传递到绑定的目标属性。 箭头 "<-"或者"<->"设置绑定是单向还是双向。

```
<html>
<head>
    <title>Slider binding text</title>
    <script src="../../montage.js"></script>
    <script type="text/montage-serialization">
    {
        "slider1": {
            "prototype": "digit/ui/slider.reel",
            "properties": {
```

```
                "element": {"#": "slider1"}
            },
            "bindings": {
                "value": {"<-": "@slider2.value"}
            }
        },
        "slider2": {
            "prototype": "digit/ui/slider.reel",
            "properties": {
                "element": {"#": "slider2"}
            }
        }
    }
    </script>
</head>
<body>
    <div data-montage-id="slider1"></div>
    <div data-montage-id="slider2"></div>
</body>
</html>
```

模板声明中定义事件监听器

你可以在模板声明中通过 `listeners` 属性设置事件监听器。

用这种方式可以减少很多的事件绑定代码。下面的例子定义两个对象：一个控制器对象（Controller）和一个按钮。控制器对象就是事件监听器的响应对象，当按钮被点击或者触摸之后会发送出"action"事件。

以下代码在组件控制器中定义一个叫做 `handleAction()` 方法，当用户点击按钮之后这个方法会被触发：

```
// Module: controller.js
// Name: Controller
var Montage = require("montage/core/core").Montage;
```

```
exports.Controller = Montage.specialize({
    handleAction: {
        value: function(event) {
            console.log("Button event handled...");
            // Do login stuff...
        }
    },
});
```

下面是一个组件的模板声明文档。在模板中的"loginBtn"对象包含一个数组类型的属性"`listeners`"。 这个数组可以包含一个或者多个监听器。

```
<html>
 <script src="../../montage.js"></script>
 <script type="text/montage-serialization">
 {
    "controller": {
        "prototype": "controller",
        "properties": {
            "element": {"#": "main"},
            "loginButton": {"@": "loginBtn"}
        }
    },
    "loginBtn": {
        "prototype": "digit/ui/button.reel",
        "properties": {
            "element": {"#": "buttonDiv"}
        },
        "listeners": [
            {
                "type": "action",
                "listener": {"@": "controller"},
                "capture": false
            }
        ]
    }
 }
```

```
 </script>
<body>
    <div id="main">
        <div data-montage-id="buttonDiv" class="Text">Click to ente
r</div>
    </div>
<body>
</html>
```

序列化格式

Serialization must be written in valid JSON format, as MontageJS uses the browser's native JSON parsing APIs to parse the serialization block. If there are formatting errors, MontageJS throws an error and does not attempt to deserialize the JSON object. Two common formatting concerns are:

序列化格式必须是一个合法的JSON格式，因为MontageJS使用浏览器本身的JSON解析APIs解析序列化内容。如果有格式错误，MontageJS抛出一个错误然后停止解析序列化里面的JSON对象。两个需要注意的格式：

- 结尾的逗号。在属性或者数组最后一个项目如果有逗号结束将会触发运行时错误。在下面的例子中 `readyState` 属性末尾有一个逗号，这就会触发JSON解析错误：

  ```
  "anObject": {
      "id": "123asd",
      "colors": [ "red", "green", "blue"],
      "readystate": false,
  }
  ```

- 括号匹配。每一个开始括号与一个结束括号要一一对应。

运行MontageJS应用的时，所有关于格式的错误信息会打印在控制台里面。
在http://www.json.org/可查看更多关于JSON格式信息。