MontageJS组件

MontageJS应用可以看作是一个模型控制层，它同时需要处理数据和将模型里面的数据读取显示到视图上面。组件处理的是MontageJS应用的视图部分。按照约定，这些组件放在MontageJS应用的ui文件夹里面（这样有助于你使用任何的MontageJS包和容易定位它里面提供的用户界面组件）。

MontageJS组件是封装的；通过结构（HTML），样式（CSS），功能（JavaScript)来构建一个强大的用户界面组件，所有这些文件都存放在同一个目录里面，目录的名字以 `.reel` 结尾。比如，一个 `FooBar` 组件存在放MontageJS应用的 `ui` 目录里面 `montageapp/ui/foo-bar.reel` 。 这个 `.reel` 目录包含 `foo-bar.css` , `foo-bar.html` , 和 `foo-bar.js` 。因为组件都是自包含的，这样就非常容易使用，重命名，或者移除一个组件，移除的时候不需要在很多目录里面去查找组件的各个部分然后逐一删除。

MontageJS组件是模块化的；不管组件在哪里被使用，同样的HTML, CSS, 和 JavaScrip使得组件的结构，样式和功能都是一样的。

> **备注:** MontageJS已经内置了几类用户界面组件（也可以算主题）： Digit, Matte, 和 Native。Digit组件是针对手持设备优化，触摸友好的。Matte组件是针对桌面优化的。Native组件是浏览器原生的。通过把HTML元素封装成组件之后就可以使用MontageJS的功能，例如数据绑定和MontageJS事件处理机制。样式就由浏览和开发者来自定义。

组件详解

让我们详细地看下包含在FooBar组件里面的 HTML, CSS, 和 JS文件的各自功能。

HTML

HTML 模板是一个完整和合法的HTML 文档。在头部包含组件的CSS文件和脚本。在这个

HTML模板是一个完整符合法的HTML文档。在其部包含组件的CSS文件和脚本，在这个模板中也包含所有的MontageJS序列化对象。

```
<link rel="stylesheet" type="text/css" href="foo-bar.css">
<script type="text/montage-serialization">
{
    "owner": {
        "properties": {
            "element": {"#": "foo-bar"}
        }
    }
}
</script>
```

在MontageJS Serialization Format可以获取更多关于序列化格式的详细信息。请注意以下规则:

- script的type要设置为 `text/montage-serialization` 。
- 序列化是JSON格式，包含一些特有的语义。
- "owner"是序列化中一个特殊的标签，代指当前的组件。（把它想象成JavaScript里面的 `this` ）
- { `"#": "foo-bar"` }指向HTML文件body的根元素，根元素使用custom data-attribute标记， `data-montage-id` 值为 `foo-bar` :

```
<body>
      <div data-montage-id="foo-bar" class="FooBar">
       </div>
</body>
```

> 备注:在对象序列化JSON树中，“#”标识符指向的是DOM元素（我们也可以使用“@”标识符来指向模板定义的对象或者序列化中的其它组件）。

当你在一个MontageJS应用中使用FooBar组件的时候，模板中只有foo-bar元素会被绘制在界面上。

> **注意**: MontageJS组件是可重用的，也就是说你可以在同一个文档中多次插入，这也是为什么我们需要使用 custom data-attribute，data-montage-id是用来标识元素的，不是HTML文档的唯一标识属性id。

CSS

样式表默认刚开始的时候差不多只是一个空白文件，你可以在里面添加你的自定义样式。这里我们只创建了一个和根元素同名的CSS类。

```
.FooBar {

}
```

同时也要注意CSS类名是首字母大写的.reel目录名。这是我们内部CSS名称规范的一部分；这样的名称规范让组件的CSS都在同一作用域下面，就不会"漏掉"或跟其它组件冲突。

JS

一般来讲一个组件是继承自Montage的一些基础类，比如通过 `.specialize()` 方法继承 `Component` 。 它的第一个参数提供了一种定义属性（包括方法和数据属性）的方式，这些属性存在于新创建的组件类里面。 第二个参数提供了定义类自身属性的方式，通过类名来使用（如， `Component.method()` ）。

> **不赞成警告**: `Montage.create(Component, {})` 已经不赞成使用了，用 `Component.specialize({})` 替换。

```
exports.FooBar = Component.specialize(/** @lends FooBar# */ {
    constructor: {
        value: function FooBar() {
            this.super();
        }
    }
```

```
});
```

`this.super()`是一个用来调用父类同名方法的特殊方法。在这个例子中它就会调用Component类定义的构造方法。用来扩展一个方法很有用，跟Java处理方式类似。

这个文件会产生一个名字跟`.reel`目录同名但是首字母大写的类，当使用`FooBar`组件的时候会在文档中根据它包含的模板文件绘制，样式就是它包含的CSS，有哪些功能就是由它的JavaScript决定。