RequireJS 的简单教程

韩博孙

2020 年 2 月 11 日 麻省理工学院

本教程将介绍使用 RequireJS 使用 JavaScript 构建模块化 Web 应用程序所需的基础知识。

本教程简要介绍了如何使用 RequireJS 通过组件和依赖项注入来实现 Web 应用程序。示例应用程序很简单，有一个表单、七个输入字段和两个用于提交和重置表单的按钮。提交表单时，字段验证完成，将显示错误状态显示或成功状态显示。状态显示被提取为可重用的组件。本教程展示了如何将所有这些拼凑在一起。

下载源 - 426.4 KB

介绍

最近，我开始了另一个项目。它是一个基于 Web 的应用程序，需要一个 JavaScript 框架来支持一些简单的前端用户交互。我不需要 AngularJS，因为我需要的功能很简单，而且不需要像 AngularJS 这样强大的 JavaScript 库。我想再次使用 JQuery，但知道单独使用它最终会变得一团糟，我决定最好将几个小而好的 JavaScript 库一起使用，并使 JavaScript 最终实现模块化。RequireJS 可用于管理这些不同的库组件，它迫使我考虑模块化我的代码并使组件可在应用程序的不同区域中重用。在本教程中，我将讨论如何使用 RequireJS 的基础知识。我将讨论两个基本概念：

1. 依赖注入
2. 在应用程序的不同部分重用组件

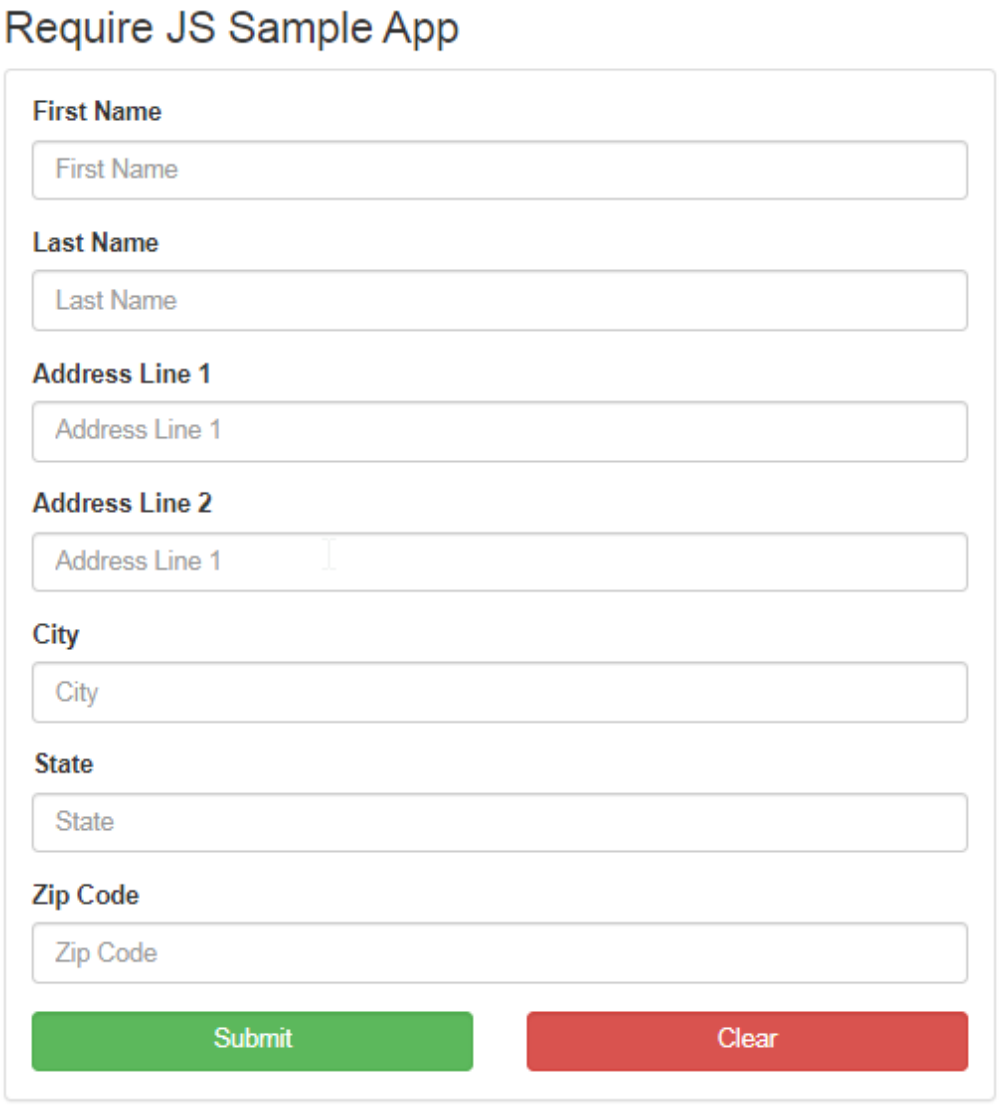
本教程中的示例应用程序非常简单。Web 应用程序是使用 Spring Boot 创建的。它没有控制器。它提供的唯一功能是提供静态内容：网页和 JavaScript 文件。该应用程序仅包含一个网页。它显示一个表单，允许用户输入一个人的联系信息，然后单击“**提交**”将信息发送到后端。该页面将显示一条状态消息，表明请求已成功发送。该应用程序还执行输入验证。如果字段输入无效，它将显示一条状态消息，指示该字段的验证失败。

状态消息显示可以是一个可重复使用的组件。在一页或许多其他页面中可以有许多位置，可以在其中显示验证状态消息。人们可以在涉及这些地方的所有 JavaScript 文件中一遍又一遍地重复相同的代码。或者我们可以编写一个可重用的函数或对象来替换这些重复的代码。在本教程中，我将展示如何创建这样的组件。最后，应用程序将演示依赖项配置和注入，这使应用程序看起来像一个 AngularJS 应用程序。

我知道这可能没有多大意义，那么看看源代码怎么样？

页面标记

让我从 HTML 页面标记开始。此示例应用程序只有一页，如下所示：



让我重申一下我之前介绍过的有关此应用程序的内容。该页面接受一个人的全名和邮寄地址。输入字段没有基于 HTML 5 的验证（输入验证是通过 JavaScript 代码完成的）。屏幕下方有两个按钮，允许用户提交（单击按钮"**提交**"）或重置（单击按钮"**清除**"）表单。

应用程序页面标记的 HTML 文件存储在名为" *index.html* "的文件中。页面标记如下所示：

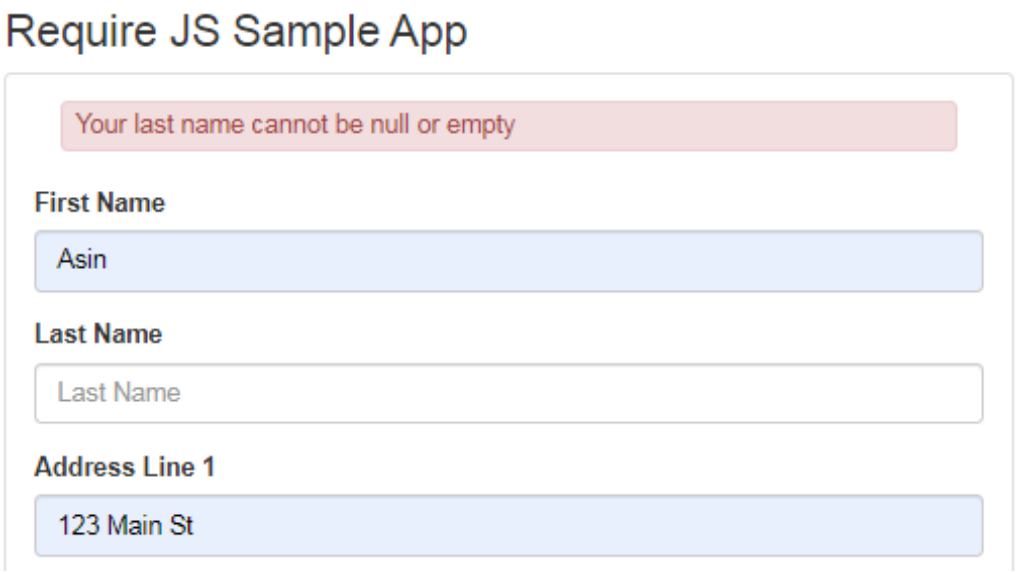
HTML 缩小▲ 复制代码

```
<!DOCTYPE html>
<html>
   <head>
      <title>Test</title>
      <link href="/assets/bootstrap/css/bootstrap.min.css" rel="stylesheet">
      <link href="/assets/css/index.css" rel="stylesheet">

      <script src="/assets/jquery/js/jquery.min.js"></script>
      <script src="/assets/bootstrap/js/bootstrap.min.js"></script>
      <script src="/assets/requirejs/require.js"></script>
   </head>
   <body>
      <div class="container">
         <div class="row">
            <div class="col-xs-12 col-sm-offset-1 colsm-10 col-md-offset-2
                        col-md-8 col-lg-offset-3 col-lg-6">
               <h3>Require JS Sample App</h3>
               <div class="panel panel-default">
                  <div class="panel-body">
                     <form id="testAppForm">
                        <div id="statusDisp"></div>
                        <div class="form-group">
                           <label for="firstName">First Name</label>
                           <input class="form-control" type="text"
```

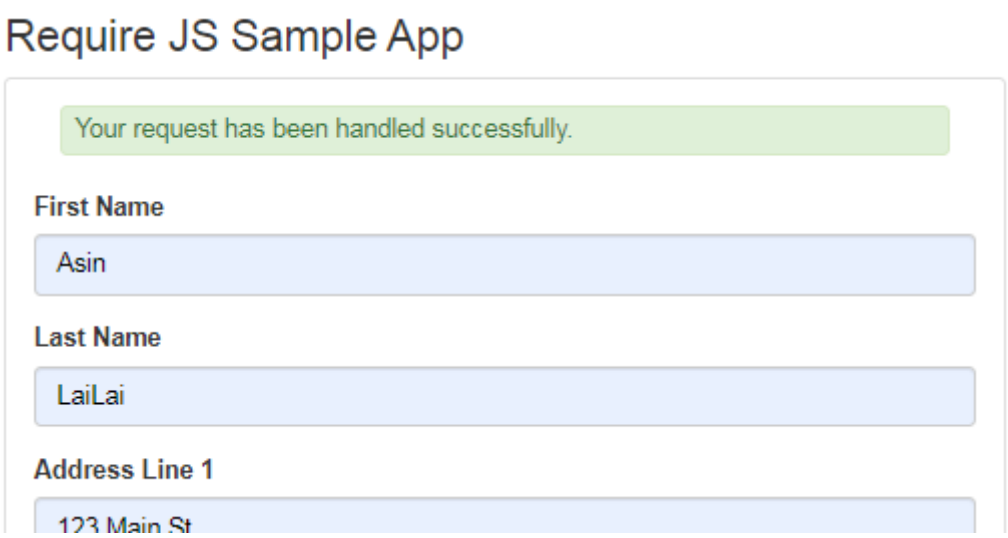
```
                        id="firstName" name="firstName" placeholder="First Name">
                    </div>
                    <div class="form-group">
                        <label for="lastName">Last Name</label>
                        <input class="form-control" type="text"
                        id="lastName" name="lastName" placeholder="Last Name">
                    </div>
                    <div class="form-group">
                        <label for="addressLine1">Address Line 1</label>
                        <input class="form-control" type="text"
                        id="addressLine1" name="addressLine1" placeholder="Address Line 1">
                    </div>
                    <div class="form-group">
                        <label for="addressLine2">Address Line 2</label>
                        <input class="form-control" type="text"
                        id="addressLine2" name="addressLine2" placeholder="Address Line 1">
                    </div>
                    <div class="form-group">
                        <label for="city">City</label>
                        <input class="form-control" type="text"
                        id="city" name="city" placeholder="City">
                    </div>
                    <div class="form-group">
                        <label for="state">State</label>
                        <input class="form-control" type="text"
                        id="state" name="state" placeholder="State">
                    </div>
                    <div class="form-group">
                        <label for="zipCode">Zip Code</label>
                        <input class="form-control" type="text"
                        id="zipCode" name="zipCode" placeholder="Zip Code">
                    </div>
                    <div class="row">
                        <div class="col-xs-12 col-sm-6">
                            <button class="btn btn-success form-control"
                            type="submit" id="submitBtn">Submit</button>
                        </div>
                        <div class="col-xs-12 col-sm-6">
                            <button class="btn btn-danger form-control"
                            type="reset" id="clearBtn">Clear</button>
                        </div>
                    </div>
                </form>
            </div>
        </div>
    </div>
</div>

<script type="text/javascript">
...
</script>
</body>
</html>
```

这是一个非常简单的标记，具有七个输入字段的表单。状态字段应该是下拉菜单，状态应该从后端服务器加载。不在这里。相反，我匆忙完成了这项工作，只是将其创建为文本框。而且我现在还清空了底部的脚本。

自从 JQuery 出现后，我们将事件处理方法移到脚本文件或部分中。如前所述，页面上的按钮具有单击事件。在**提交**按钮，点击后会触发输入验证，并会显示在这样的巅峰状态消息：



当所有输入数据检查完毕后，单击"**提交**"按钮将触发以下状态消息显示：



为了让所有这些工作，我需要添加一些 JavaScript 代码。我做的第一件事是创建引导程序代码，即启动配置。还记得上面空白的 JavaScript 部分吗？那是引导 JavaScript 代码。

启动配置的"引导程序"

我要指出的一件事是，在这种情况下，当我提到"bootstrap"时，我指的不是 Bootstrap UI 框架，我也用于这个项目。我所说的"引导程序"是我将设置的配置，以便 RequireJS 可以加载不同的组件形成一个工作的 Web 应用程序。它不同于 Bootstrap 框架。

您可以在最底部的文件" *index.html* "中找到我的引导程序代码。这是它的样子：

JavaScript 复制代码

```
require.config({
    paths: {
      jquery: "./assets/jquery/js/jquery.min",
      underscore: "./assets/underscore/underscore-min-1.9.2",
      statusDisplay: "./assets/app/components/statusDisplay/statusDisplay",
      stapes: "./assets/stapes/stapes-min-1.0.0",
      app: "./assets/app/app"
   }
});

require(["app"], function(app) {
   app.init();
});
```

这段代码有两部分。第一部分是设置所有组件的配置。RequireJS 需要知道的是两件事：

- 组件的名称。这个名字可以被认为是一个哈希映射的键，与键相关联的值将是组件。
- JavaScript 文件所在的位置，以便 RequireJS 可以加载组件的源代码。

在这部分中，我又添加了两个库。一个叫Stapes，是BackboneJS框架的升级版。但是，我认为它是 JQuery 之上的包装器。另一个是UnderscoreJS。我使用它的模板功能来使用模板字符串呈现实际的 HTML。这就是我创建可重用和交互式 HTML 组件的方式。我将针对这两个库创建另一个教程。

第二部分是实际应用的启动。因为我使用 Stapes JS 库进行 HTML 元素操作。我所要做的就是调用初始化代码，UI 会自己处理交互。

这仅仅是开始。这个应用程序是关于可重用的组件。接下来，我将向您展示如何创建这种可重用组件。

可重用组件

如前所述，状态显示可用于应用程序的许多地方。最好将其提取为可重复使用的组件。在这些位置，或者可以设置占位符，以便后端 JS 代码可以插入可重用的组件以形成最终的输出显示。

这是这个可重用组件的完整源代码：

JavaScript 缩小▲ 复制代码

```
define(["jquery", "underscore"], function ($, _) {
   var statusHtml = '<div class="col-xs-12">\n'+
   '   <div class="<%= alertStyleVal%>"><%= msgToShow%></div>\n'+
   '</div>';

   var statusTemplate = _.template(statusHtml);
   const errorStyle = "alert alert-danger small-alert";
   const successStyle = "alert alert-success small-alert";

   function renderStatus(outerDiv, msg, style) {
      if (outerDiv) {
         if (msg != null && msg.length > 0) {
            var divToAdd = statusTemplate({
               alertStyleVal: style,
               msgToShow: msg
            });
            $(outerDiv).html(divToAdd);
         }
      }
   }

   return {
      clearRender: function(outerDiv) {
         if (outerDiv) {
            $(outerDiv).html("");
         }
      },

      renderSuccess: function (outerDiv, msg) {
         renderStatus(outerDiv, msg, successStyle);
      },

      renderError: function (outerDiv, msg) {
         renderStatus(outerDiv, msg, errorStyle);
      }
   };
});
```

让我一步一步地经历所有这些疯狂，一次一个。第一个是这个外壳：

JavaScript 复制代码

```
define(["jquery", "underscore"], function ($, _) {
...
});
```

这是一个函数调用。该函数被调用`define(...)`。它是 RequireJS 的一个函数，用于定义一个组件。在这种情况下，该函数接受两个参数，第一个是一个数组，它接受一些字符串值，即依赖组件的名称。这就是依赖注入发生的地方。对于这个可重用的组件，我需要 JQuery 和 UnderscoreJS。第二个参数是函数定义，参数列表与依赖项的名称顺序相同。参数“ `$`”是对JQuery的引用，“ “是对UnderscoreJS的`_`引用。有了这两个，我可以将它们用作这两个框架的使用方式。

接下来，我声明一些变量并实例化它们：

JavaScript 复制代码

```
...
   var statusHtml = '<div class="col-xs-12">\n'+
      '   <div class="<%= alertStyleVal%>"><%= msgToShow%></div>\n'+
      '</div>';

   var statusTemplate = _.template(statusHtml);
   const errorStyle = "alert alert-danger small-alert";
   const successStyle = "alert alert-success small-alert";
...
```

前几行定义了显示 HTML 源代码模板。这里：

JavaScript 复制代码

```
var statusHtml = '<div class="col-xs-12">\n'+
   '   <div class="<%= alertStyleVal%>"><%= msgToShow%></div>\n'+
   '</div>';
```

然后我需要一个模板函数，后面可以用来注入一些值，并编译成HTML字符串放在最终的HTML页面中。这是我声明的方式：

JavaScript 复制代码

```
var statusTemplate = _.template(statusHtml);
```

我所做的是叫一叫方法`template()`在`UnderscoreJS`库中。它接受模板源值，并返回一个函数引用。接下来我们将看到这一点。以下代码定义了一个函数，该函数将模板源呈现为 HTML 代码，然后附加到现有的 HTML 元素。这里是：：

JavaScript 复制代码

```
function renderStatus(outerDiv, msg, style) {
   if (outerDiv) {
      if (msg != null && msg.length > 0) {
         var divToAdd = statusTemplate({
            alertStyleVal: style,
            msgToShow: msg
         });
         $(outerDiv).html(divToAdd);
      }
   }
}
```

该函数执行一些基本的数据验证，确保要显示的消息不为 null 或为空。它还检查以确保状态显示将附加到的元素也不为空。一旦检查成功，它就会使用模板函数引用来呈现 HTML 显示字符串。然后外部元素用于附加新创建的 HTML 显示值。

最后一部分是返回代表我定义的组件的对象。这里是：

JavaScript 复制代码

```
...
return {
   clearRender: function(outerDiv) {
      if (outerDiv) {
         $(outerDiv).html("");
      }
   },

   renderSuccess: function (outerDiv, msg) {
```

```
      renderStatus(outerDiv, msg, successStyle);
   },

   renderError: function (outerDiv, msg) {
      renderStatus(outerDiv, msg, errorStyle);
   }
};
...
```

返回的这个对象有三个方法：

- 第一个被称为`clearRender()`。它获取输入 HTML 元素的引用，然后清除其内部 HTML 值。
- 第二个向目标 HTML 元素添加成功状态消息。成功状态消息具有绿色背景。
- 第三个向目标 HTML 元素添加错误状态消息。错误状态消息具有红色背景。

现在我们有了一个可重用的组件，让我们看看它是如何被注入到页面组件中并被利用的。

控制页面元素的代码

下面是示例应用程序的复杂部分。有一个包含七个输入字段和两个按钮的页面。按钮必须具有关联的事件处理方法。正如我在开头所说的，我不想使用 JQuery。但它是如此受欢迎，以至于不可能放弃它。所以我将它用作可注入组件，主要用作 DOM 查询，以及附加新元素，如上一节所示。在本节中，我使用了`Stapes`框架来对用户交互进行编程。

`Stapes`框架基于 BackboneJS。在我看来这很酷。不幸的是，它不再维护，可能没有人使用这个框架。总之，让我印象深刻。我可能会在今年的某个时候写一篇关于它的教程。在本教程中，我使用它的目的只是为按钮添加事件处理：

- 当用户点击“**提交**”时，首先进行输入数据验证，如果输入数据有任何问题，将显示红色错误状态消息。如果所有输入数据都正确，则会显示成功状态消息。
- 另一个按钮清除所有输入字段。

包含所有这些的文件称为“ *app.js* ”。以下是其内容的完整列表：

JavaScript 缩小▲ 复制代码

```
define(["jquery", "stapes", "statusDisplay"], function ($, Stapes, statusDisplay) {

   var testAppForm = Stapes.subclass({
      constructor : function() {
         var self = this;
         self.$el = $("#testAppForm");

         var statusDisp = self.$el.find("#statusDisp");
         var firstNameInput = self.$el.find("#firstName");
         var lastNameInput = self.$el.find("#lastName");
         var addrLine1Input = self.$el.find("#addressLine1");
         var addrLine2Input = self.$el.find("#addressLine2");
         var cityInput = self.$el.find("#city");
         var stateInput = self.$el.find("#state");
         var zipCodeInput = self.$el.find("#zipCode");

         self.$el.on("submit", function(e) {
            e.preventDefault();
            if (validateForm()) {
               statusDisplay.renderSuccess(statusDisp,
                           "Your request has been handled successfully.");
            }
         });

         self.$el.on("reset", function(e) {
            e.preventDefault();
            clearCommentForm();
         });

         function clearCommentForm() {
            statusDisplay.clearRender(statusDisp);
            firstNameInput.val("");
```

```
            lastNameInput.val("");
            addrLine1Input.val("");
            addrLine2Input.val("");
            cityInput.val("");
            stateInput.val("");
            zipCodeInput.val("");
        }

        function validateForm() {
            statusDisplay.clearRender(statusDisp);
            var firstNameVal = firstNameInput.val();
            if (firstNameVal == null || firstNameVal.length <= 0) {
                statusDisplay.renderError(statusDisp,
                    "Your first name cannot be null or empty");
                return false;
            }

            if (firstNameVal != null && firstNameVal.length > 128) {
                statusDisplay.renderError(statusDisp,
                    "Your first name is too long, 128 characters or fewer.");
                return false;
            }

            var lastNameVal = lastNameInput.val();
            if (lastNameVal == null || lastNameVal.length <= 0) {
                statusDisplay.renderError(statusDisp,
                    "Your last name cannot be null or empty");
                return false;
            }

            if (lastNameVal != null && lastNameVal.length > 128) {
                statusDisplay.renderError(statusDisp,
                    "Your last name is too long, 128 characters or fewer.");
                return false;
            }

            var addressLine1Val = addrLine1Input.val();
            if (addressLine1Val == null || addressLine1Val.length <= 0) {
                statusDisplay.renderError(statusDisp,
                    "Your address line #1 cannot be null or empty.");
                return false;
            }

            if (addressLine1Val != null && addressLine1Val.length > 128) {
                statusDisplay.renderError(statusDisp,
                    "Your address line #1 cannot have more than 128 characters");
                return false;
            }

            var addressLine2Val = addrLine2Input.val();
            if (addressLine2Val != null && addressLine2Val.length > 128) {
                statusDisplay.renderError(statusDisp,
                    "Your address line #2 cannot have more than 128 characters");
                return false;
            }

            var cityVal = cityInput.val();
            if (cityVal == null || cityVal.length <= 0) {
                statusDisplay.renderError(statusDisp,
                    "Your city is null or empty.");
                return false;
            }

            if (cityVal != null && cityVal.length > 48) {
                statusDisplay.renderError(statusDisp,
                    "Your city cannot have more than 48 characters");
                return false;
            }
```

```
            var stateVal = stateInput.val();
            if (stateVal == null || stateVal.length <= 0) {
               statusDisplay.renderError(statusDisp,
                  "Your state is null or empty.");
               return false;
            }

            if (stateVal != null && stateVal.length > 2) {
               statusDisplay.renderError(statusDisp,
                  "Your state cannot have more than 2 characters");
               return false;
            }

            var zipCodeVal = zipCodeInput.val();
            if (zipCodeVal == null || zipCodeVal.length <= 0) {
               statusDisplay.renderError(statusDisp,
                  "Your state is null or empty.");
               return false;
            }

            if (zipCodeVal != null && zipCodeVal.length > 12) {
               statusDisplay.renderError(statusDisp,
                  "Your zip code cannot have more than 12 characters");
               return false;
            }

            return true;
         }
      }
   });

   return {
      init: function() {
         new testAppForm();
      }
   };
});
```

你应该知道这是做什么的：

JavaScript 复制代码

```
define(["jquery", "stapes", "statusDisplay"], function ($, Stapes, statusDisplay) {
...
});
```

它将 JQuery、Stapes和 my 组件statusDisplay注入到这个组件中。RequireJS 使用定义将其注册为另一个组件。

接下来，我声明一个Stapes类类型。这是我如何做到的：

JavaScript 复制代码

```
var testAppForm = Stapes.subclass({
      constructor : function() {
         ...
      }
   });
```

请注意，我说的是我正在创建的类型，而不是对象。对于这种新类型，它只包含一个方法，即构造函数。在这个构造函数中，我做的第一件事是获取输入字段和按钮的句柄，如下所示：

JavaScript 复制代码

```
...
   var self = this;
   self.$el = $("#testAppForm");

   var statusDisp = self.$el.find("#statusDisp");
```

```
   var firstNameInput = self.$el.find("#firstName");
   var lastNameInput = self.$el.find("#lastName");
   var addrLine1Input = self.$el.find("#addressLine1");
   var addrLine2Input = self.$el.find("#addressLine2");
   var cityInput = self.$el.find("#city");
   var stateInput = self.$el.find("#state");
   var zipCodeInput = self.$el.find("#zipCode");
...
```

如此处所示，我必须使用 JQuery 来获取这些输入字段句柄。并将这些保存为我正在创建的这种类型的一部分。至于按钮，我附上了点击事件的处理方法，就像这样：

JavaScript 复制代码

```
      ...
         self.$el.on("submit", function(e) {
            e.preventDefault();
            if (validateForm()) {
               statusDisplay.renderSuccess(statusDisp,
                             "Your request has been handled successfully.");
            }
         });

         self.$el.on("reset", function(e) {
            e.preventDefault();
            clearCommentForm();
         });
      ...
```

如图所示，所有这些都是 JQuery 代码，如何查询 HTML 元素，以及如何将事件处理附加到按钮。对于这两个按钮，在事件处理方法中，首先是调用事件对象的“ e”preventDefault()方法。这将阻止按钮执行实际的提交或重置功能。然后该方法将执行定制的功能。

在上面的代码片段中，您可以看到正在使用的状态显示可重用组件。在我们的页面中，有这样的 ：

HTML 复制代码

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

在我上面的代码中，它引用了这个div：

JavaScript 复制代码

```
...
var statusDisp = self.$el.find("#statusDisp");
...
```

最后，代码可以将状态显示添加到页面：

JavaScript 复制代码

```
...
statusDisplay.renderSuccess(statusDisp, "Your request has been handled successfully.");
...
```

对于**提交**按钮，有输入字段的验证。任何验证失败都会触发错误状态消息的显示。如果所有输入字段验证成功，则会显示成功消息，模拟假设的数据传输到后端服务器。

数据输入验证又长又无聊。我只会展示这个方法的一部分，validateForm()：

JavaScript 复制代码

```
function validateForm() {
   statusDisplay.clearRender(statusDisp);
   var firstNameVal = firstNameInput.val();
   if (firstNameVal == null || firstNameVal.length <= 0) {
      statusDisplay.renderError(statusDisp,
```

```
            "Your first name cannot be null or empty");
         return false;
      }

      if (firstNameVal != null && firstNameVal.length > 128) {
         statusDisplay.renderError(statusDisp,
            "Your first name is too long, 128 characters or fewer.");
         return false;
      }

      ...

      return true;
}
```

最后是清除输入字段的功能，它所做的就是将所有字段的值设置为空字符串：

JavaScript 复制代码

```
function clearCommentForm() {
   statusDisplay.clearRender(statusDisp);
   firstNameInput.val("");
   lastNameInput.val("");
   addrLine1Input.val("");
   addrLine2Input.val("");
   cityInput.val("");
   stateInput.val("");
   zipCodeInput.val("");
}
```

这就是我使用Stapes. 接下来，我将返回该组件，以便它可以在页面 HTML 中使用。这里是：

JavaScript 复制代码

```
define(["jquery", "stapes", "statusDisplay"], function ($, Stapes, statusDisplay) {
...
   return {
      init: function() {
         new testAppForm();
      }
   };
});
```

这是应用程序组件。它只有一种方法叫做init(). 它所做的只是从Stapes我创建的类型中实例化一个对象。它怎么可能工作？嗯，它是有效的，因为类型构造函数将从页面中找到所有 HTML 元素句柄，并附加所有事件处理程序。所以所有对元素和事件处理方法的引用都会被保存，直到页面卸载。

如何启动应用程序

现在回到页面源代码。JavaScript 源代码部分有这个：

JavaScript 复制代码

```
...
require(["app"], function(app) {
   app.init();
});
...
```

对 RequireJS 函数的require()调用基本上是调用作为第二个参数提供的函数。第一个是依赖项数组。提供的函数调用对象 app 的方法init()，这在术语中创建了我使用创建的类型的实例Stapes。实例化将在页面上设置事件处理。如图所示，这个应用程序的整个设置非常简单。

现在已经讨论了有关此应用程序的所有内容，让我们看看如何测试此应用程序。

如何构建和测试

在构建此应用程序之前，请将示例项目中的所有文件从*.sj重命名为*.js。

获得示例项目并在本地将其解压缩到您的计算机后，您需要有 Java 1.8 和 Maven 3.0 或更高版本才能编译。编译示例对象的方式是运行以下命令：

复制代码

```
mvn clean install
```

构建成功完成后，您可以运行以下命令来启动 Web 应用程序：

复制代码

```
java -jar target/testapp-0.0.1-SNAPSHOT.jar
```

命令成功启动后，您可以使用浏览器访问以下 URL 来测试应用程序：

复制代码

```
http://localhost:8080/
```

如果启动命令成功运行，页面应该看起来像顶部的屏幕截图 #1。您可以在页面上输入一些信息，然后单击**提交**按钮以查看将显示什么状态消息。

概括

在本教程中，我简要介绍了如何使用 RequireJS 通过组件和依赖项注入来实现 Web 应用程序。使用 RequireJS，人们可以轻松地将应用程序分解为不同的组件，然后将它们拼接成一个工作应用程序。模块化应用程序始终是一个好主意。它将应用程序分解为可以组合在一起的不同齿轮，其中一些可以在同一应用程序的不同位置重复使用。模块化可以使应用程序干净、整洁，有时还易于测试。

我已经使用 JQuery 很长时间了，从来不满足于它是许多复杂应用程序中使用的唯一框架，并且使用了可怕的妥协并安装了丑陋的代码段。这一次，我决定做一些不同的事情，所以我选择了 RequireJS、Stapes 和 UnderscoreJS。框架是否过时或没有人使用它真的无关紧要。只要它可以在应用程序中发挥作用，就可以而且应该使用它。我从未使用过这三个库中的任何一个。所以它把这些用于这个简单的示例应用程序中。而且我觉得这些库非常有效。

我拼凑的示例应用程序是我刚刚拼凑起来的一个概念应用程序。我将把我在这里所做的一切转移到我构建并且仍在构建的应用程序中。在这个示例应用程序中，应用程序很简单，它有一个表单、七个输入字段和两个用于提交和重置表单的按钮。提交表单时，字段验证完成，将显示错误状态显示或成功状态显示。状态显示被提取为可重用的组件。本教程展示了如何将所有这些拼凑在一起。不多。对我来说，这些库在如何正确使用它们来创建出色的应用程序方面显示出巨大的潜力。

历史

- 2020 年 2 月9日 - 初稿

执照

分享

关于作者

韩博孙
组长 The Judge Group
美国
手表该会员
没有提供传记

评论和讨论

添加评论或问题 电子邮件提醒 Search Comments

第一 页上一页 下一页

我前一段时间做了其中一个，可能会感兴趣
Sacha Barber 20-Feb-20 16:49

我的5票
Jay Bardeleben 15-Feb-20 1:29

我的5票
arroway 14-Feb-20 17:05

刷新 1

一般 新闻 建议 问题 错误 答案 笑话 赞美 咆哮 管理员

使用Ctrl+Left/Right 切换消息，Ctrl+Up/Down 切换主题，Ctrl+Shift+Left/Right 切换页面。