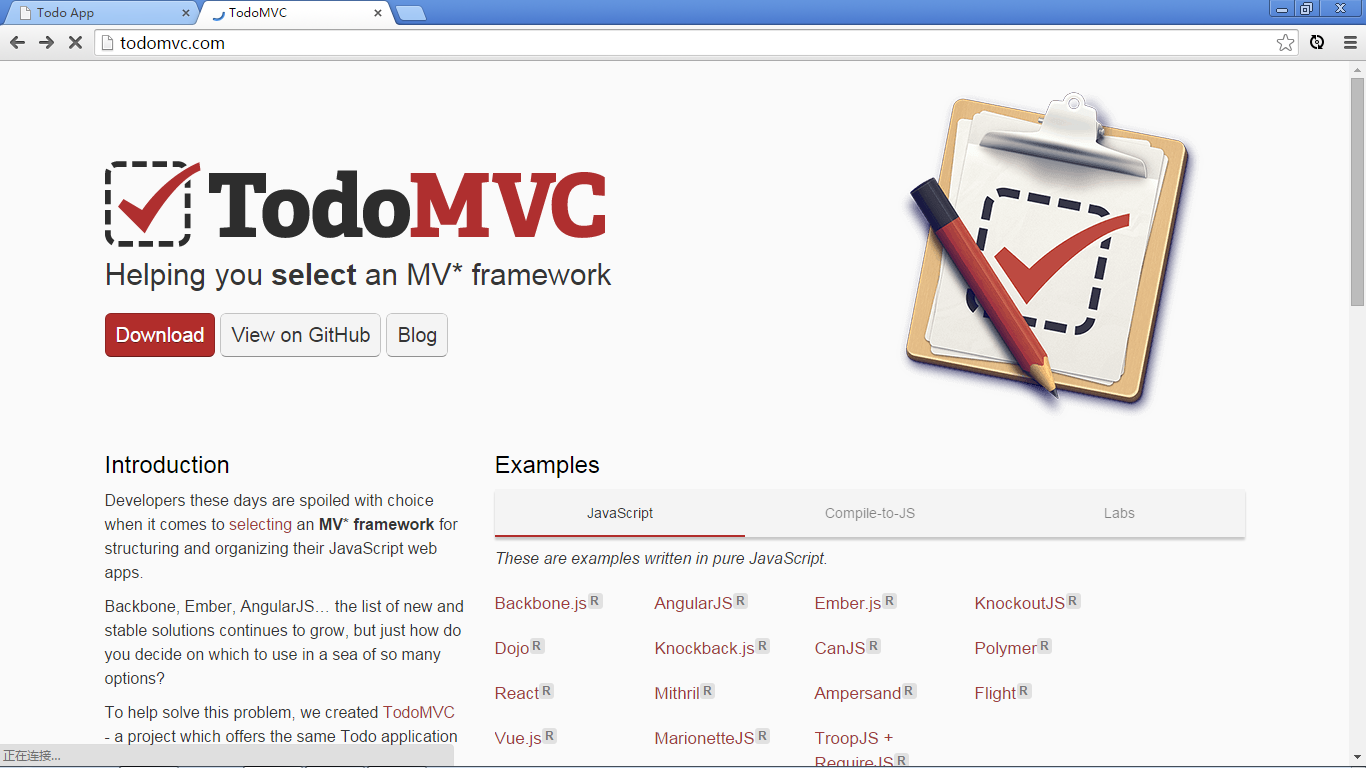
JavaScript组件化开发实战——TodoMVC

1. 简介

TodoMVC（http://todomvc.com/）这个开源项目是帮助小伙伴们选择合适的MV\*框架。TodoMVC旨在用各种框架实现TodoList的增、删、改、查功能，麻雀虽小，五脏俱全，是供小伙伴学习、练习、再学习的好例子。





（看看人家老外的例子，再看看咱们过来的例子。。。哎）

1. 我的版本

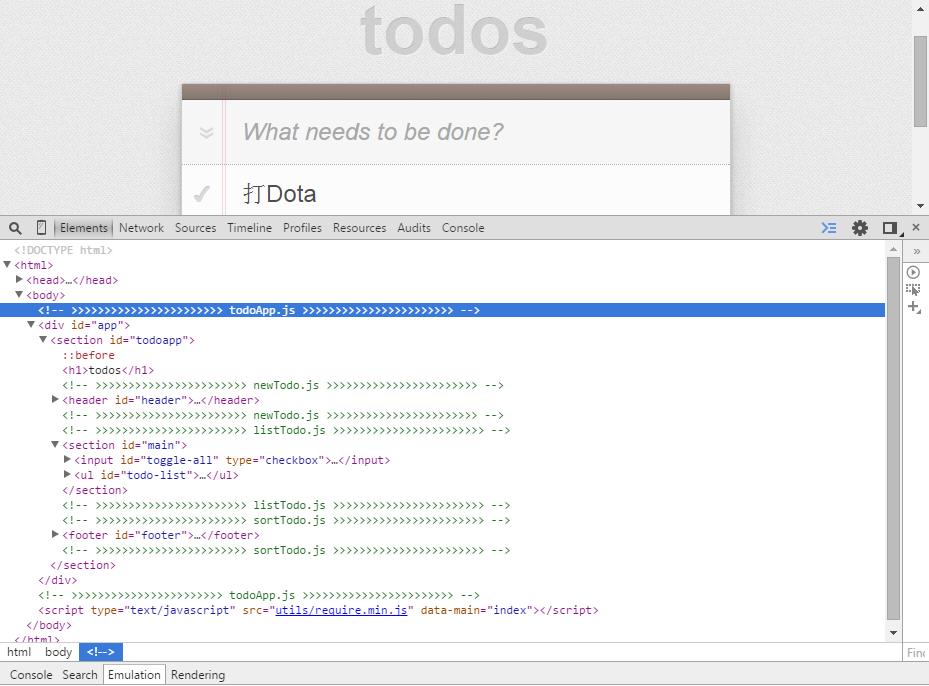
虽然官网上有各种实现版本，但仿佛缺少那么一个版本，就是，就是。。。我实现的版本。为此，本猿抛砖引玉，用RequireJs+jQuery实现一个组件化的TodoMVC。

之所以没有用到高级的前端MV\*框架，如AngularJs、ReactJs+Flux等，是为了帮助大家学习前端组件化开发思想。同时，也希望大家将这种较为单纯的实现方法与其它前端MV\*框架进行对比，理解其它框架的实现初衷和原理。

本猿实现的这个版本有以下两大特点：

1. 组件之间的关系可视化：

在页面上右键“审查元素”就能通过注释，清楚地看见组件之间的关系（如下图）。这些注释是通过覆写jquery的html、append等函数实现的，并不需要程序猿自己去打印注释。这样一来，调试或修改代码时就能够快速定位代码的出处，即使不是本人写的代码，调试起来也很轻松。当需要正式发布代码时，可以通过设置一个参数来关闭这个注释功能。



1. 组件之间不直接调用对方的函数，而是通过消息来完成交互。



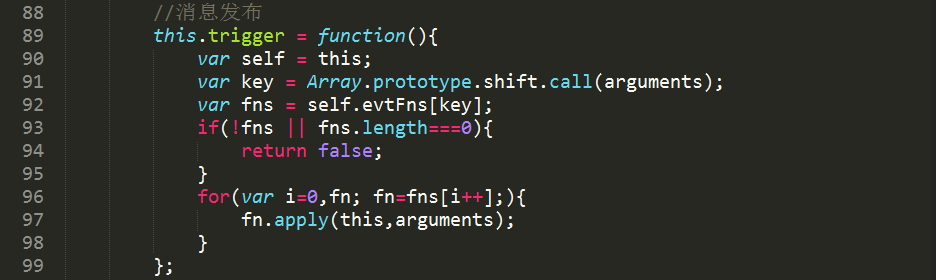
如上图所示，本人设计了两种消息，一种是请求消息（request），一种是触发消息（trigger）。

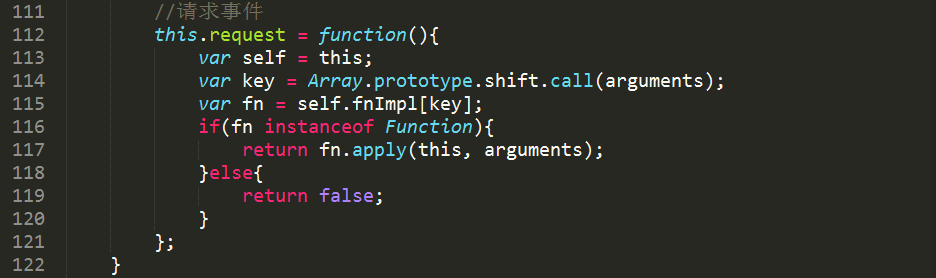
请求消息（request）：意思是当这个组件要做某个操作，比如添加Todo，它便发起“addTodo”这个消息，并附带上newTodo这个参数。至于谁来实现这个请求，它不关心。

触发消息（trigger）：意思是当这个组件完成了某个操作，比如成功添加Todo，它便触发“NewTodo\_addTodo”这个消息，谁关心这个消息，谁就把自己的函数绑定到这个消息上。

请求消息与触发消息的最大区别是：请求消息只会有一个实现，一对一的关系，而且往往会返回处理后的结果；而触发消息是一对多的关系，就像广播消息一样，谁关心谁监听，一旦有这个消息触发，便执行相关操作。

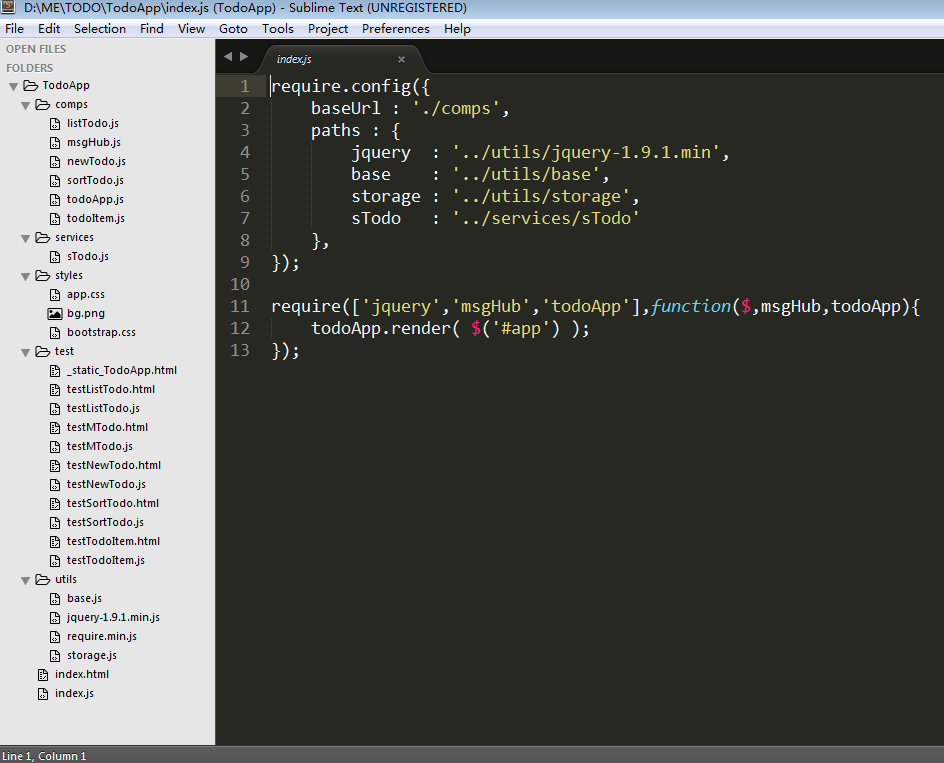
具体实现代码请查看TodoApp\utils\base.js（代码很挫，只供学习参考）





1. 实现

Talk is cheap, show me the code!——忍不住引用一下Linus Torvalds的名言。但在看代码前，先看看项目完成后的代码目录结构，方便理解。



公用类库（包括自己写的base.js、storage.js）

（不解释）

样式

Services

组件（包括消息中间件msgHub.js）

单元测试（单元开发）

* 1. 拆分组件



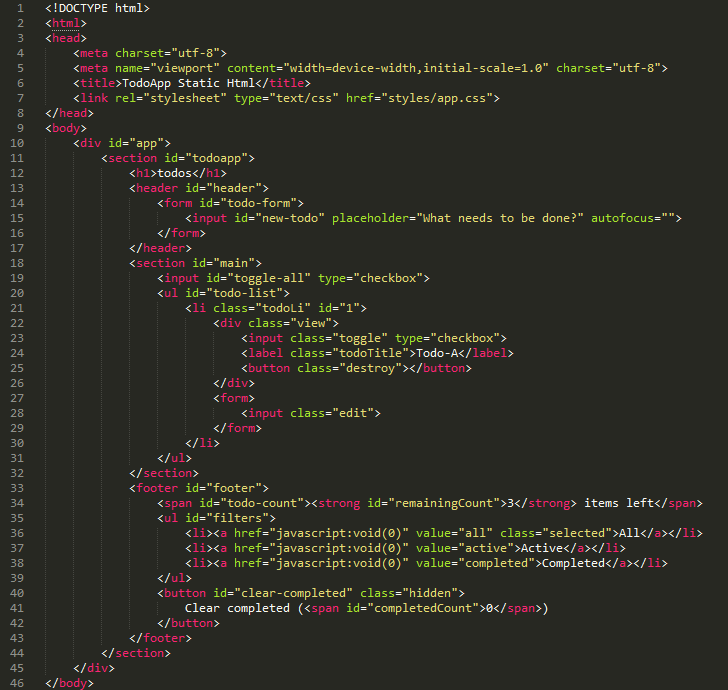
TodoItem组件

NewTodo组件

SortTodo组件

TodoList组件

假设下面是设计工程师给你的静态页面（\TodoApp\ \_static\_TodoApp.html），我们首先要做的事情是用组件化的思维，将其分为若干个组件来动态渲染。



何为组件？组件就是你给我一个锚点——一个section，一个div，或者一个span，我就在这个锚点指定的空间内渲染组件内容。

如何拆分？请遵守两个原则：

1、一切皆组件：我们将TodoApp这个应用看作一个大组件，它嵌套了三个子组件：NewTodo组件、ListTodo组件、SortTodo组件。而ListTodo组件又嵌套了子组件TodoItem。

2、单一职责原则：组件的功能尽可能地单一明了，不要将太多的功能设计到同一个组件中。功能多了，就想办法设计子组件，将功能分拆出去。

根据上面提到两个设计原则，我们将这个静态页面代码分拆成5个组件，它们的关系和功能如下：

NewTodo

TodoList

TodoItem

SortTodo

**TodoApp**

1. 根据过滤条件遍历Todos
2. 批量变更Todo的状态
3. 显示Todo总数和完成数量
4. 删除已完成的Todo
5. 提交过滤条件

1、添加新Todo

1. 删除Todo
2. 变更完成状态
3. 修改Title

TodoApp也是一个组件，负责嵌套它的子组件。

* 1. 构建组件

根据上述的设计，我们将设计好的组件构建出来。从MVC的角度分析，其关系如下图所示。

HTML

NewTodo

TodoList

TodoItem

SortTodo

**TodoApp**

View

Controller

Storage

图-4

但这样的模型存在一些明显的弊端：

1. 组件之间的交互是直接通过调用对方的函数来实现，耦合性高，不易维护与扩张。
2. 组件直接访问与操作数据储存(Storage)，这个Storage可能是服务端，也可能是本地存储，不管是什么，直接访问与操作会导致代码维护与扩展的极为不灵活。比如说项目的前后台并行开发。前端数据的存储操作通过一个假服务端或浏览器的本地存储来完成。如果你将这部分操作直接写在组件的代码中，以后改你代码的同事一定会骂你九条街。

HTML

NewTodo

TodoList

TodoItem

SortTodo

**TodoApp**

View

Controller

Storage

* 1. 增加Service层

为了降低组件与Storage层的耦合性，我们可以增加一个Service层，专门负责模型的增删改查等操作。你也可以将这个Service层理解成Model层，因为它封装了模型的所有操作。

HTML

NewTodo

TodoList

TodoItem

SortTodo

**TodoApp**

View

Controller

Storage

sTodo

Service

(Model)

Service1

Service2

**…**

但目前组件之间的耦合性依然很高。比如现在增加了一个新模块ProjectApp，要求当TodoApp新增Todo时，ProjectApp也要做出联动响应。那么你不得不找到TodoApp中新增Todo的代码进行修改。大部分人遇到这种情况都会来一句——找你妹的代码！

* 1. 增加消息中间件

之前已经提到过，本实例中的组件是通过消息来完成交互的。但问题就来了，消息都发到哪了？谁来处理这些消息了？如果直接在各个组件中监听、处理消息，将会导致一个消息被发送后，不知道谁会处理了这个消息。代码维护起来十分困难。

为此，我们可以增加了一个特殊的组件——消息中间件msgHub。

HTML

NewTodo

TodoList

TodoItem

SortTodo

**TodoApp**

View

Controller

Storage

sTodo

Service

(Model)

**msgHub**

Service1

Service2

**…**

请求消息的实现与触发消息的绑定都是在msgHub中完成。如此一来，以后变更或扩展项目，只需要在msgHub中修改消息与函数之间的绑定关系即可（TodoApp\comps\ msgHub.js）。



待。