前端 MVC 入门与实践

hdm258i@gmail.com

2013-10

自我简介

```
var hudamin =
  {
     name: ' 胡大民 ',
     career: ' 程序猿 ',
     desc: ‘08 年北上，百度，资深工程师，垂直搜索部前端技术负责人， 10 年开始 HTML5 Mobile WebApp 技术研究， Rocket 框架作者， W3C WebApp Group 打酱油 '
  };
```

大纲

第一部分： MVC 简介。 why 前端 MVC， backbone 框架， SPA

第二部分： todos demo 讲解。单页面 SPA，前端 MVC 的基本用法

第三部分： mytoolkit demo 讲解。多页面 SPA，企业级 webapp 如何使用前端 MVC

课程按照理论到实践、入门到深入的顺序，逐步讲解前端 MVC 的理论和实践，以期通过该课程的学习，能够直接将 MVC 运用到前端项目中。

前置知识

掌握 BackboneJS 基本用法， http://backbonejs.org/

了解 Rocket 框架， https://github.com/MichaelHu/rocket

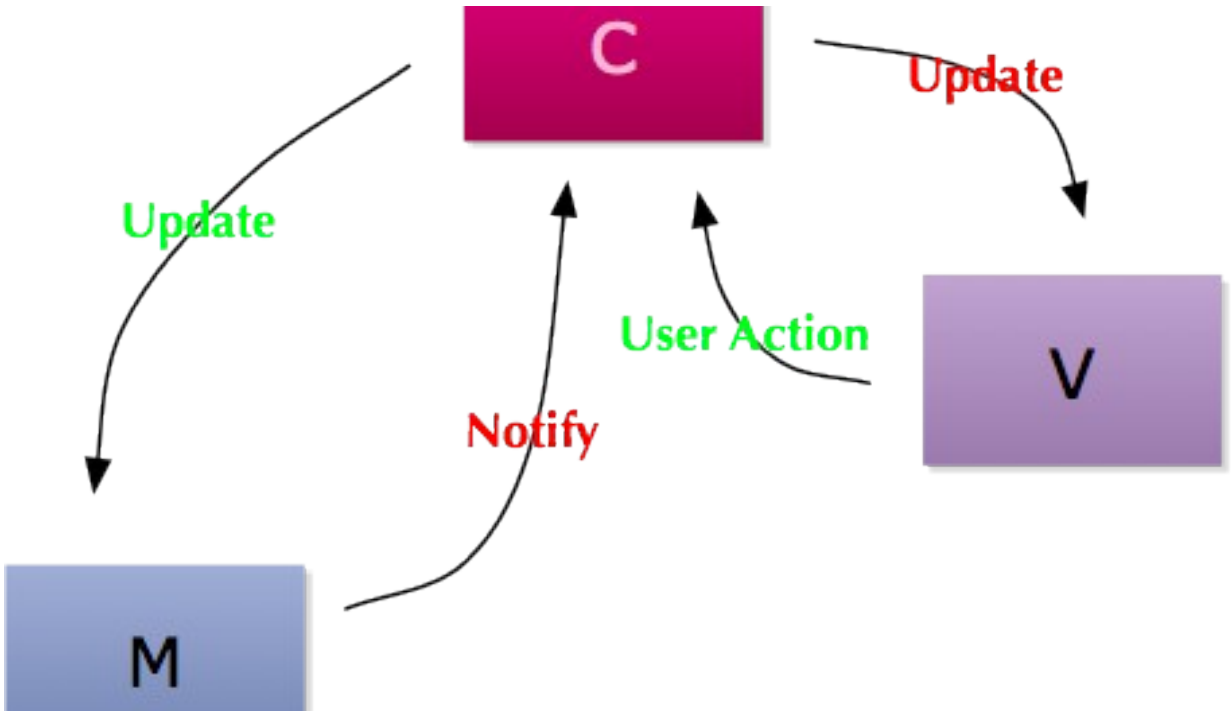
解 fis （ Fe Integrated Solution ）环境和基本命令，
http://fis.baidu.com

第一部分： **MVC** 简介

MVC

- MVC 是一种软件设计理念，帮助构建健壮的、伸缩性好的软件系统
- 将软件实现抽象成 Model， View 和 Controller 三个基本模块，每个模块功能明确，各司其职
- 隐含 Observer 的设计模式， M 和 V 充分解耦
- MVC 的好处：模块功能内聚，模块间松耦合，使开发和维护更加容易

MVC 关系图



前端 MVC

- 传统 MVC => 前端 MVC：优秀理念自然而然的扩散和渗透
- 未变的是 MVC 核心理念（模块抽象、模块解耦）
- 变的只是运用场景，催生新的前端开发模式

Why 前端 MVC

- 前端开发规模变大： RIA 的普及催肥了前端， js 代码轻松跨 w，需要多人参与开发
- 前端开发复杂度变大： SPA 普及， PC 端模拟桌面应用， Mobile 端模拟 Native App，可想而知
- 前端由鸡变成牛，得用牛刀了

前端 MVC 框架

- Backbone: http://backbonejs.org/
- Angular, Ember, CanJS
- 框架比较：
  http://sporto.github.io/blog/2013/04/12/comparison-angular-backbon
- 框架比较 (翻译版) □
  http://www.csdn.net/article/2013-04-25/2815032-A-Comparison-of-An
- 总的来说，咱不人云亦云，关键是看自己的真是需求是什么，
  最好是自己试一下再做决定。这个网站或许能帮上一点忙：
  http://todomvc.com
- 目前百度产品主要使用 backbone ：简单（易于扩展，满足快
  速迭代的需求）、性能（很关键）

Backbone 简介

- 支持 WebApp □ MVC 框架
- Model 键值绑定和自定义事件
- Collection 支持各类丰富的集合操作函数
- View （□ Sub Controller）支持声明式用户事件处理
- Router （□ App Controller）支持功能强大的配置式路由
- 所有 Backbone 对象均支持 Events 接口，原生支持 Observer 模式
- PS： Backbone 的控制器实际上就是 View

BACKBONE 模型事件

1. Model Events

事件名	： 方法
change, change:field	: set, unset, clear, fetch
change, request, sync	: save
destroy, request, sync	: destroy

2. Collection Events

事件名	： 方法
add	: add
remove	: remove
reset	: reset
add|remove|change	: set
reset, add|remove|change	: fetch
add, request, sync	: create

补充一个概念： SPA

1. Single Page Application ： 单网页应用

简单的说，页面交互采用无刷新方式

从浏览器角度说，网页的 document 只在刚打开时从服务器下载，后续与服务器的交互都是通过 AJAX 或者 JSONP 等方式，界面的更新由前端脚本通过操作 DOM 来实现

2. SPA 根据复杂程度，还可以分单页面 SPA 和多页面 SPA，通常使用 hash 来区分不同页面

第二部分： **todos** – 单页面 **SPA** 实践

2.1 需求描述

实现一个待办事项（todo）管理的 webapp，支持添加、删除、修改、存储功能。界面功能描述如下：

- 输入完毕，按回车按钮完成添加，默认状态为“未完成”
- 点击 todo item 的 checkbox，todo item 在“未完成”和“完成”两种状态间切换，选中状态下为“完成”状态，同时显示删除线
- 双击 todo item，可进行原地修改，按回车按钮完成修改
- 鼠标移过 todo item，显示删除按钮，若点击删除按钮，则删除一条 todo
- 显示 todo 的统计信息：有多少未办事项
- 支持选中全部功能，通过点击全选 checkbox，可批量切换 todo item 的状态

2.2 需求分析和设计

引入前端 MVC，使用 BackboneJS 作为前端 MVC 框架，设计过程遵守以下原则：

- 数据模型驱动，视图更新由模型事件来驱动
- 使用事件中心，模块间松耦合

2.2.1 抽象数据模型设计

- todo：一个待办事项，包括 title, done 字段
- todolist：待办事项组成的列表，可以进行增删改查

2.2.2 控制流程设计

控制流程负责响应用户交互事件，操作数据模型，响应数据模型事件

- 初始化（应用开启）：读取数据，填充 todolist，响应 reset 事件
- 新 todo：新增一个 todo 至 todolist，响应 add 事件
- 删除 todo：调用 todo 的 destroy 方法，响应 destroy 事件
- 更新 todo：调用 save 方法，响应 change 事件

2.3 具体实现

具体实现按以下步骤展开，代码文件 [todos.js]：

1. 数据模型定义

定义 Todo Model 以及 TodoList Collection

2. 控制器实现

监听系统交互事件，操作数据模型，更新视图展现

3. 视图实现

前端模板系统，数据和 HTML 的分离

4. 应用初始化

2.3.1 数据模型定义

- **Todo Model**

代码见 [todos.js] line 3 – line 19

- **TodoList Collection**

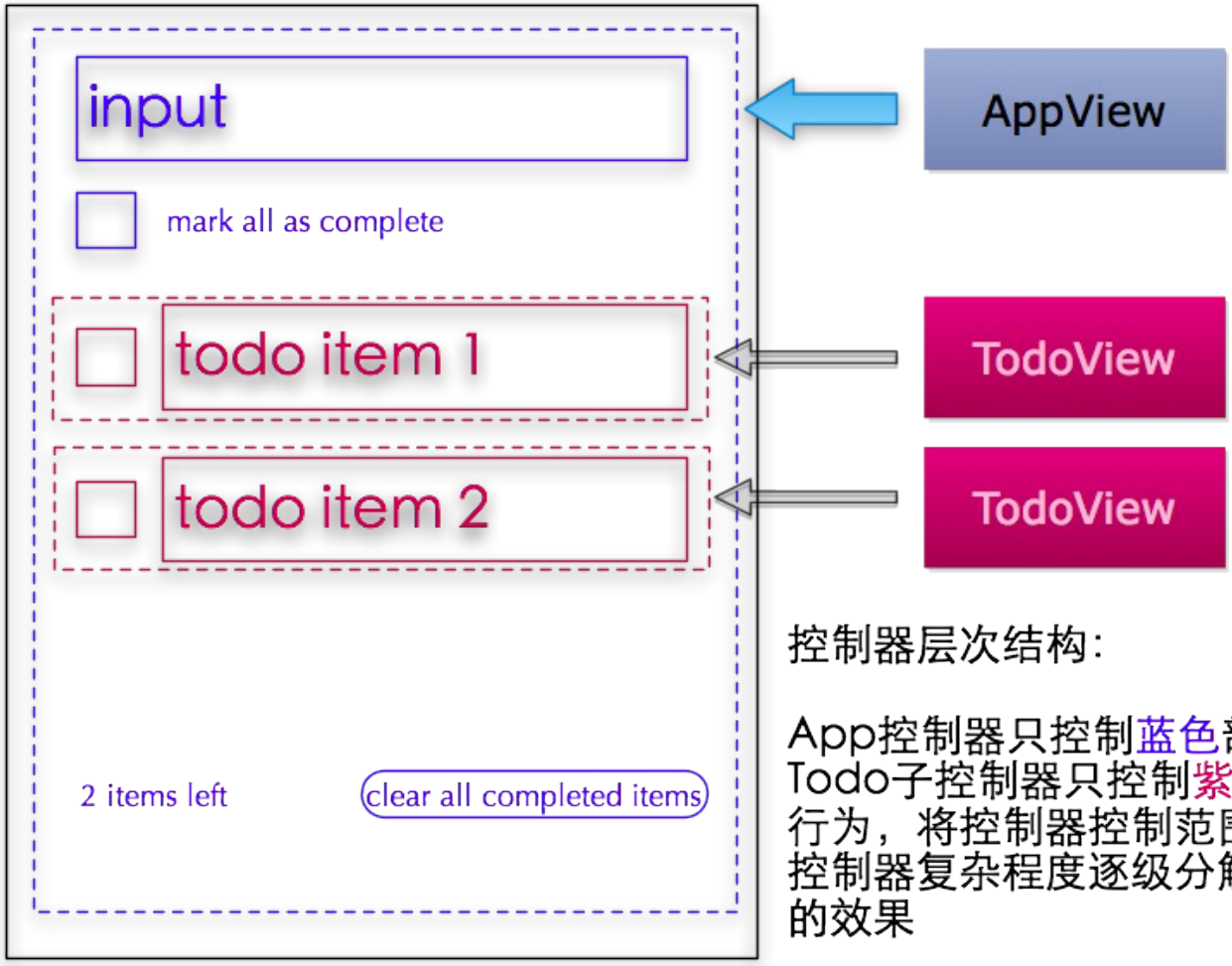
代码参考 [todos.js] line 21 – line 47

2.3.2 控制器实现

Backbone 的控制器实际上是 View 来承担，需实现两个控制器

- todo 控制器，处理 todo 的事件响应，协调数据模型和视图的同步，代码参考 [todos.js] line 49 – line 117
- app 控制器，处理整个应用的事件响应，协调数据模型和视图的同步，代码参考 [todos.js] line 119 – line 196

2.3.2 控制器实现 – 控制器层次结构



控制器层次结构：

App控制器只控制蓝色部分的用户行为，Todo子控制器只控制紫红色部分的用户行为，将控制器控制范围进行了分拆，使控制器复杂程度逐级分解，达到分而治之的效果

2.3.3 视图实现

使用 underscore 提供的前端模板，编写两套模板：

- todo item 模板，代码参考 [todos_entry_html]line 45 – line 52
- todo 统计信息模板，代码参考 [todos_entry_html]line 54 – line 59

2.3.4 应用初始化

创建 App Controller 的实例，完成应用初始化：

```
var App = new AppView;
```

2.4 项目代码结构

以上具体介绍了主 js 文件 todo.js 的实现，我们再来看看项目代码结构：

```
▼ todos
  // 依赖 js 库
  ▼ lib/
      backbone.js
      backbone.localStorage.js
      jquery.js
      json2.js
      underscore.js
  ▼ src/
    // 样式表和图片资源
    ▼ css/
        destroy.png
        todos.css
    ▼ js/
        todos.js
      index.html
```

2.4.1 入口页面

参考 [todos_entry_html]

关注几个主要部分：

1.Css 引用

2.Templates 定义

3. 依赖库

4. 初始化

2.5 小结

该部分介绍了 todos webapp 核心部分的设计和实现，大致了解了以下内容：

- MVC 作为一种软件设计的理念，引导开发者在开发过程中，自觉进行业务层、控制层、展现层的抽象和分离
- 使用 MVC 框架进行简单前端应用开发的基本步骤：模型抽象、控制器实现、视图实现
- BackboneJS 的基本使用方法
- 前端模板的基本使用方法

但是，这仅仅是一个简单 **demo**，是否仍感觉无从下手…

实际的 **webapp** 开发就是这样的么？不是的话，那应该是怎样的?

如何合理使用前端 **MVC**，确保代码的健壮性和伸缩特性?

如何合理使用前端 **MVC**，使代码开发简单和高效?

MVC 如何帮助复杂多页面 SPA 的开发?

第三部分： **mytoolkit** – 多页面 **SPA** 实践

前言

该部分延续 todos 的工作，将其扩展成多页面的 SPA，并通过企业级方案来开发，向大家介绍：

- 企业级 webapp 的开发方式
- 如何合理使用 MVC，使复杂 SPA 化繁为简
- 如何通过规范来提高代码的健壮性和伸缩性，提高开发效率

该案例代码可提前从此处获取：
https://github.com/MichaelHu/rocket_apps/tree/master/mytoolkit

案例地址：http://258i.com/template/mytoolkit/mytoolkit.html

3.1 需求描述

实现一个多页面 SPA mytoolkit，一期功能包含三个页面：

- index 页面：导航页面，默认页
- about 页面：版本更新历史
- todos 页面：待办事项管理工具

3.1.1 效果图


My Toolkit
Todos 1 item left
About
返回
Todos
What needs to be done?
Mark all as complete
hudamin@baidu.com
前端MVC简介和实践
10-09，清华大学讲课
1 item left
Clear 2 completed items
Double-click to edit a todo.
返回
v0.3 (2013-10-09)
Index page: Todos button show "item[s] completed"
v0.2 (2013-10-06)
add About page
some bugfix
v0.1 (2013-09-30)
add Todos page

3.1.2 界面功能描述（仿 iOS 设计）

- index 页面包含两个通栏按钮，点击后左滑进入 sayhello 或者 todos 页面， todos 按钮需显示待办事项状态
- about 页面顶部包含返回按钮，点击后右滑进入 index 页面
- todos 页面主体同第二部分，顶部需包含返回按钮，点击后右滑进入 index 页面

3.1.3 场景模拟

老板：这个项目很急，一天内完成！

码农 A：这个。。，一天完成不了，至少得两天

老板：给你加个人，我明天要看到成品！干活去吧

码农 A：。。。

老板这是要逆天么，**webapp** 开发，你要我 **1+1=2**？！

- **两人怎么分工？**
- **代码怎么合并？**
- **如何调试？**
- **...**

别抱怨了，想想如何保质保量完成任务吧 . . .

YY 一下，做完了给加薪就好了 ^_^

3.2 需求分析与设计

- 凭一己之力，从零开始码起，肯定不可行
- 那就站在他人肩膀上干起

3.2.1 前端框架选型

这是一个仿 iOS Native App 的 webapp 项目，是个多页面 SPA 。

要达到 1+1=2 ，必须对任务进行分拆，多人并行开发，才能完成。

前端框架选型要求：支持并行开发，支持前端 MVC

Rocket 框架就是为多页面 SPA 而生的

并行开发 <- 合作机制 <- 完善的规范

前端 MVC <- 基于 Backbone

化繁为简 <- 控制器树（ MVC 分解）

参考： Rocket框架概述

3.2.2 开发规范 – 目录规范

并行开发需规范的支持，让每个开发人员的工作有章可循，右方是目录格式

一个页面对应一个独立目录，页面所需各类资源均存放在同一目录下，最大限度隔离不同页面的开发，使多人合作成为可能

index, about, todos 三个页面，每个页面下对应 css， html， img，js， tpl 等资源

- ▼ mytoolkit/
  - ► css/
  - ► img/
  - ► js/
  - ▼ page/
    - ► about/
    - ▼ index/
      - ► css/
      - ► html/
      - ► img/
      - ► js/
      - ► tpl/
    - ► todos/
  - ► tpl/
  - mytoolkit.html

3.2.3 开发规范 – 其他

包括文件命名、类命名、 CSS class 、 CSS id 等命名规范。

参考 Rocket编码规范

思路：按规范进行细粒度分拆

1. 目录结构明确，可以先行由一个工程师将全部页面的目录都建好
2. 离散的页面代码文件，需有机制将其合并

3.2.3 代码合并和部署

所有离散的文件都合并成 All In One 文件，可采用 FIS 一百度前端集成解决方案作为前端开发工具，负责代码合并、部署等任务的自动化

- 入口页面： mytoolkit/mytoolkit.html
- CSS all-in-one: mytoolkit/css/mytoolkit-aio.css
- JS all-in-one: mytoolkit/js/mytoolkit-aio.js

入口页面： mytoolkit.html

代码参考： [mytoolkit.html]

使用 *<!—inline[…]-->* 语法将页面骨架和前端模板文件包含进来，只需少量几行代码，但却大大减少入口页面的 size

CSS all-in-one: mytoolkit-aio.css

代码参考： [mytoolkit-aio.css]

使用 @import url("…?__inline") 语法将 css 片段文件引入

jS all-in-one: mytoolkit-aio.js

代码参考： [mytoolkit-aio.js]

使用 __inline(”…“) 语法将 js 片段文件引入

3.3 具体实现

准备阶段（共同讨论，一人完成，耗时 0.5h ）：

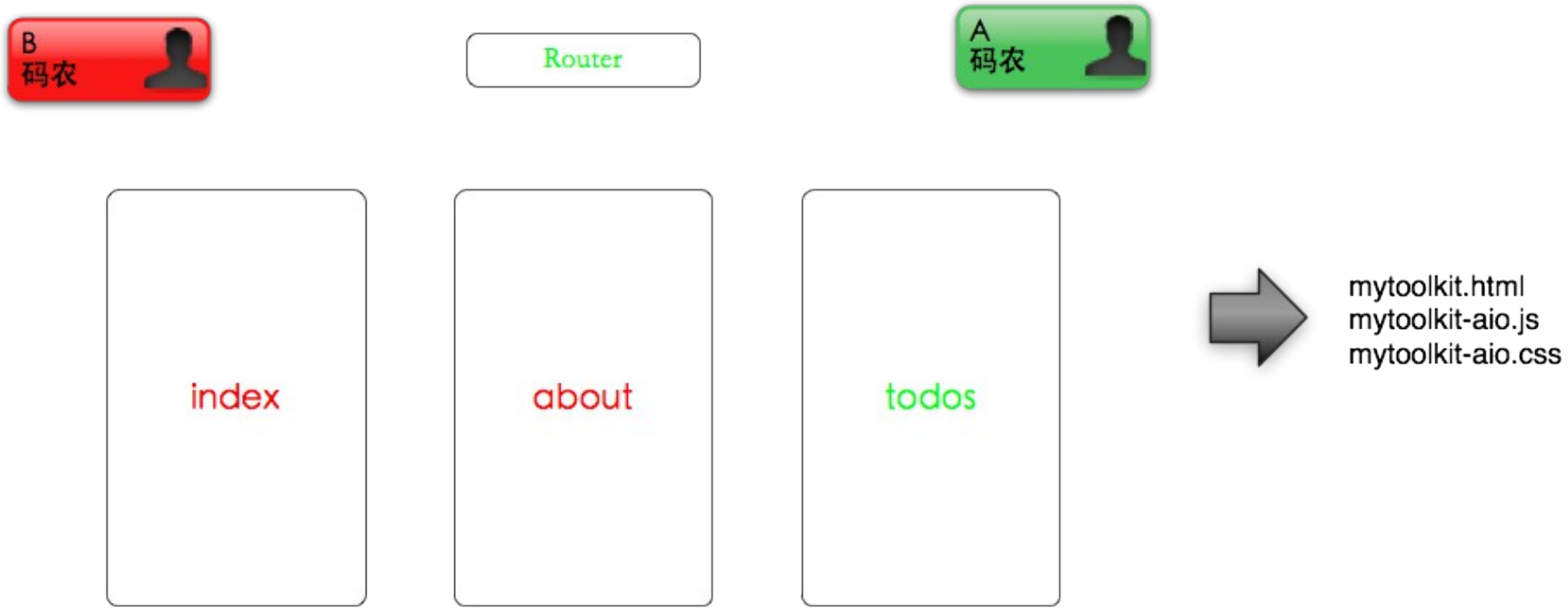
1. 路由配置
2. 建立目录结构和 All-In-One 文件

页面开发阶段（并行，独立开发、调试，耗时 4h ）：

1. 码农 A 在 page/todos 下开发、调试 todos 页面
2. 码农 B 在 page/index 与 page/about 下分别开发、调试 index 页面和 about 页面

PS：路由和目录结构是**薄实现的稳定的公共层**，一旦确定就很少需要改动，即使改动，只需做配置更改即可。所以并行过程中的开发和调试都可以独立进行

3.3.1 分工


B
码农
Router
A
码农
index
about
todos
mytoolkit.html
mytoolkit-aio.js
mytoolkit-aio.css

3.3.2 router 部分 – 公共层

多页面 SPA 中， router 作为路由分发中心，根据不同页面路由，选择不同的页面 handler 。

Rocket 框架对 Backbone 的 router 进行了封装扩展，提供的灵活的可配置项，天生支持多页面 SPA 。

代码参考： [rocket.router.mytoolkit.js]

3.3.2 todos 页面

- 目录结构
- 模型定义
- 控制器实现
- 视图实现
- 其他资源

3.3.3.1 todos 目录结构

- 按资源类型进行了细化拆分： css, html, js, img, tpl
- js 文件按 model、 view 分为两类，分别对应模型和控制器目录
- rocket.model.todo 对应第二部分 demo 的 Todo Model
- rocket.collction.todolist 对应第二部分 demo 的 Todolist Collection
- rocket.pageview.todos 为页面控制器，同时作为页面事件中心，对应第二部分 demo 的 AppView
- rocket.subview.todo 为 todo item 控制器，对应第二部分 demo 的 TodoView

3.3.3.2 todos 模型定义

- mytoolkit/
  - page/
    - todos/
      - js/
        - model/
          - rocket.collection.todolist.js
          - rocket.model.todo.js

代码参考： [rocket.model.todo.js]

[rocket.collection.todolist.js]

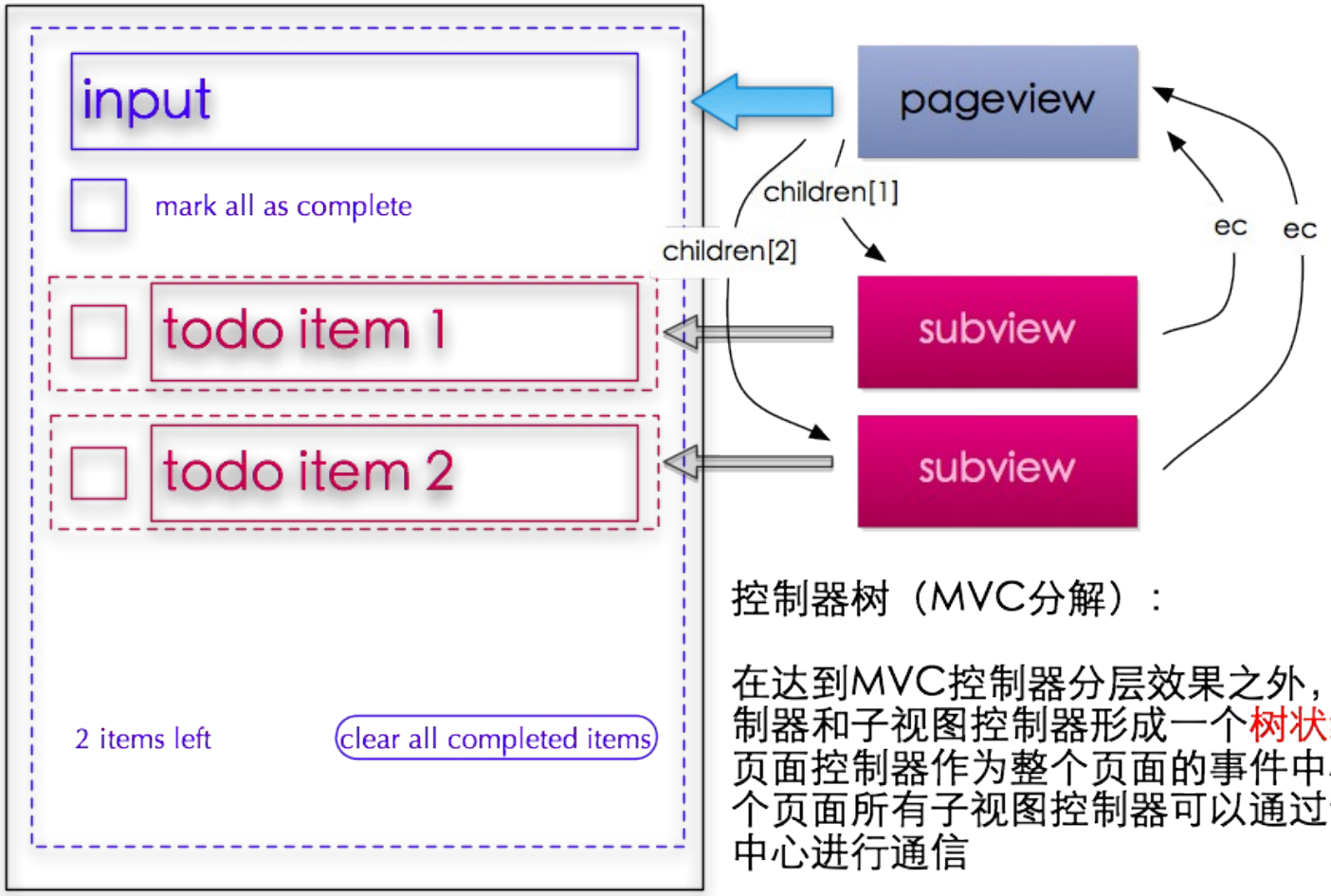
3.3.3.3 todos 控制器实现

- mytoolkit/
  - page/
    - todos/
      - js/
        - view/
          - rocket.pageview.todos.js
          - rocket.subview.todo.js

代码参考： [rocket.pageview.todos.js]

[rocket.subview.todo.js]

Todos 控制器树



控制器树（MVC分解）：

在达到MVC控制器分层效果之外，页面控制器和子视图控制器形成一个树状结构，页面控制器作为整个页面的事件中心，整个页面所有子视图控制器可以通过该事件中心进行通信

3.3.3.4 TODOS 视图实现

- mytoolkit/
  - page/
    - todos/
      - tpl/
        - todos.tpl.html

参考代码： [todos.tpl.html]

3.3.3.5 TODOS 其他部分

其他子目录包含页面骨架、样式表和图片资源，注意其命名规范，
css id 和 css class 命名规范，如下：

- mytoolkit/
  - page/
    - todos/
      - css/
        - todos.css　　代码参考：[todos.css]
      - html/
        - todos.html　　代码参考：[todos.html]
      - img/
        - destroy.png

3.3.3.6 todos 页面小结

通过这部分的介绍，我们大致了解了：

- todos 页面的工作主要按照 rocket 规范进行分拆、迁移
- router 联系各个页面的部分，主要工作为添加配置内容

至此，码农 A 的工作已经完成，运行 fis release 一下， url 后面添加 #todos 已经可以独立运行

3.3.4 index 页面和 about 页面

思路同 todos 页面，不再赘述。

参考代码： [index page], [about page]

3.3.5 完工

A 和 B 的工作的都完成以后， mytoolkit 一期整体工作随之完成，已经可以进行页面间切换。

二期准备添置一些诸如 Calendar 、常用网址等的小工具进去，还是早做准备吧，这老板，你懂的～

3.3.6 多页面 SPA 实践小结

通过 mytoolkit 的实践，我们了解了：

- **规范的力量，是合作开发的基础**
- **物理上的独立目录，强制隔离不同页面的开发，为并行开发、调试提供便利**
- **薄实现的配置型路由层，使公共部分简单明确，确保整体应用的健壮性**
- **扩展性得到极大的增强，增减页面变得非常容易**