技能树之旅: 从模块分离到测试

Posted by: Phodal Huang (/blog/author/root/) March 3, 2015, 9:17 p.m.

在之前说到

> 奋斗了近半个月后，将*fork*的代码读懂、重构、升级版本、调整，添加新功能、添加测试、添加*CI*、添加分享之后，终于*almost finish*。

今天就来说说是怎样做的。

Github项目组成

以之前造的Lettuce (https://github.com/phodal/lettuce)为例，里面有:

- 代码质量(Code Climate)
- CI状态(Travis CI)
- 测试覆盖率(96%)
- 自动化测试(npm test)
- 文档

按照Web Developer路线图 (https://github.com/phodal/awesome-developer)来说，我们还需要有:

- 版本管理
- 自动部署

等等。

Skillock模块化

在SkillTree的源码里，大致分为三部分:

- namespace函数: 故名思意
- Calculator也就是TalentTree，主要负责解析、生成url，头像，依赖等等
- Skill 主要是tips部分。

而这一些都在一个js里，对于一个库来说，是一件好事，但是对于一个项目来说，并非如

此。

依赖的库有

- jQuery
- Knockout

好在Knockout可以用Require.js进行管理，于是，使用了Require.js进行管理:

```
<script type="text/javascript" data-main="app/scripts/main.js" src="app/lib/requ
```

main.js配置如下:

```
require.config({
  baseUrl: 'app',
  paths:{
    jquery: 'lib/jquery',
    json: 'lib/json',
    text: 'lib/text'
  }
});

require(['scripts/ko-bindings']);

require(['lib/knockout', 'scripts/TalentTree', 'json!data/web.json'], function(ko,
  'use strict';
  var vm = new TalentTree(TalentData);
  ko.applyBindings(vm);
});
```

text、json插件主要是用于处理web.json，即用json来处理技能，于是不同的类到了不同的js文件。

```
.
|____Book.js
|____Doc.js
|____ko-bindings.js
|____Link.js
|____main.js
|____Skill.js
|____TalentTree.js
|____Utils.js
```

加上了后来的推荐阅读书籍等等。而Book和Link都是继承自Doc。

```
define(['scripts/Doc'], function(Doc) {
  'use strict';
  function Book(_e) {
    Doc.apply(this, arguments);
```

```
  }
  Book.prototype = new Doc();

  return Book;
});
```

而这里便是后面对其进行重构的内容。Doc类则是Skillock中类的一个缩影

```
define([], function() {
  'use strict';
  var Doc = function (_e) {
    var e = _e || {};
    var self = this;

    self.label = e.label || (e.url || 'Learn more');
    self.url = e.url || 'javascript:void(0)';
  };

  return Doc;
});
```

或者说这是一个AMD的Class应该有的样子。考虑到this的隐性绑定，作者用了self=this来避免这个问题。最后Return了这个对象，我们在调用的就需要new一个。大部分在代码中返回的都是对象，除了在Utils类里面返回的是函数:

```
return {
    getSkillsByHash: getSkillsByHash,
    getSkillById: getSkillById,
    prettyJoin: prettyJoin
};
```

当然函数也是一个对象。

Skillock测试

自动化测试

一直习惯用Travis CI，于是也继续用Travis Ci，.travis.yml配置如下所示:

```
language: node_js
node_js:
  - "0.10"

notifications:
  email: false

branches:
```

```
branches:
  only:
    - gh-pages
```

使用gh-pages的原因是，我们一push代码的时候，就可以自动测试、部署等等，好处一堆堆的。

接着我们需要在package.json里面添加脚本

```
"scripts": {
    "test": "mocha"
  }
```

这样当我们push代码的时候便会自动跑所有的测试。因为mocha的主要配置是用mocha.opts，所以我们还需要配置一下mocha.opts

```
--reporter spec
--ui bdd
--growl
--colors
test/spec
```

最后的test/spec是指定测试的目录。

Jshint

> *JSLint*定义了一组编码约定，这比*ECMA*定义的语言更为严格。这些编码约定汲取了多年来的丰富编码经验，并以一条年代久远的编程原则 作为宗旨：能做并不意味着应该做。*JSLint*会对它认为有的编码实践加标志，另外还会指出哪些是明显的错误，从而促使你养成好的 *JavaScript*编码习惯。

当我们的js写得不合理的时候，这时测试就无法通过:

```
line 5   col 25   A constructor name should start with an uppercase letter.
line 21  col 62   Strings must use singlequote.
```

这是一种驱动写出更规范js的方法。

Mocha

> *Mocha* 是一个优秀的*JS*测试框架，支持*TDD/BDD*，结合
>
> *should.js/expect/chai/better-assert*，能轻松构建各种风格的测试用例。

最后的效果如下所示:

```
Book,Link
  Book Test
    ✓ should return book label & url
  Link Test
    ✓ should return link label & url
```

测试用例

简单地看一下Book的测试:

```
/* global describe, it */

var requirejs = require("requirejs");
var assert = require("assert");
var should = require("should");
requirejs.config({
  baseUrl: 'app/',
  nodeRequire: require
});

describe('Book,Link', function () {
  var Book, Link;
  before(function (done) {
    requirejs(['scripts/Book'、], function (Book_Class) {
      Book = Book_Class;
      done();
    });
  });

  describe('Book Test', function () {
    it('should return book label & url', function () {
      var book_name = 'Head First HTML与CSS';
      var url = 'http://www.phodal.com';
      var books = {
        label: book_name,
        url: url
      };

      var _book = new Book(books);
      _book.label.should.equal(book_name);
```

```
      _book.url.should.equal(url);
    });
  });
});
```

因为我们用require.js来管理浏览器端，在后台写测试来测试的时候，我们也需要用他来管理我们的依赖，这也就是为什么这个测试这么从的原因，多数情况下一个测试类似于这样子的。(用Jasmine似乎会是一个更好的主意，但是用习惯Jasmine了)

```
describe('Book Test', function () {
  it('should return book label & url', function () {
    var book_name = 'Head First HTML与CSS';
    var url = 'http://www.phodal.com';
    var books = {
      label: book_name,
      url: url
    };

    var _book = new Book(books);
    _book.label.should.equal(book_name);
    _book.url.should.equal(url);
  });
});
```

最后的断言，也算是测试的核心，保证测试是有用的。

结束语(小广告)

在最开始的时候想的是自己写技能树，直至在github上看到https://github.com/352Media/skilltree (https://github.com/352Media/skilltree)，就想着基于skilltree做一个，试着翻译了一下，加了点功能。最后没有避免被骂抄袭，最后想想还是自己写一个吧:https://github.com/phodal/sherlock (https://github.com/phodal/sherlock)。叫Sherlock的原因是，原来是打算叫shelflock，柯南道尔在书中写道:

> 人的大脑如同一间空空的阁楼，要有选择地把一些家具装进去。

基于D3.js，可以动态生成Url，而技能树则会基于:https://github.com/phodal/awesome-developer (https://github.com/phodal/awesome-developer)

希望自己看过的书、走过的路可以给大家提供帮助