Rocket依赖之Backbone和Underscore的体积优化

目录

[背景 1](#_Toc356224242)

[优化目标 1](#_Toc356224243)

[优化点 2](#_Toc356224244)

[普通优化 2](#_Toc356224245)

[功能优化 2](#_Toc356224246)

[代码拆分合并工具 2](#_Toc356224247)

[工具说明 2](#_Toc356224248)

[工具设计 2](#_Toc356224249)

[源码拆分 3](#_Toc356224250)

[源码合并 3](#_Toc356224251)

[工具特点 3](#_Toc356224252)

[优点 3](#_Toc356224253)

[缺点 3](#_Toc356224254)

[优化效果 3](#_Toc356224255)

[环境说明 3](#_Toc356224256)

[效果数据 3](#_Toc356224257)

[总结 4](#_Toc356224258)

[关于Underscore 4](#_Toc356224259)

[关于Backbone 4](#_Toc356224260)

[关于Rocket 4](#_Toc356224261)

# 背景

为什么要做体积优化？

1. 无线端对文件体积的要求比较高
2. 根据业务特征的不同，需要对backbone部分功能做拆分、重构或删除。

在无线端，IE的支持已经没有存在的必要。Underscore有近70%的API并未被Backbone使用。同时，在GMU库引入的情况下，Underscore部分API和Zepto扩展(GMU)功能是重复的。Backbone的emulateHTTP和emulateJSON基本不会被使用，Collection、Model要么被移除，要么仅当作ajax快捷方式而存在。总体上有60%左右的代码是冗余的。

# 优化目标

优化的目标是想做一个Backbone精简版，精简版是指：

1. GMU的使用作为前置条件。
2. 仅支持无线端环境
3. 保留的功能在所有业务线中必被使用，且这些功能可以完整的创建普通webapp。

# 优化点

## 普通优化

1. Underscore仅保留被Backbone依赖的API。
2. Underscore和Zepto (或GMU) 重复实现的功能做转换。
3. 移除IE的支持。
4. 移除emulateHTTP/emulateJSON的支持。
5. 移除HTTP Method PUT/CREATE/DELETE/PATCH。

## 功能优化

1. 精简Backbone的Event、Model、Collection、View、Router、History、Sync及其子功能
   * 删除Model、Collection、Sync
   * Event作为View的依赖被完整保留，但是View移除了setElement方法
   * History仅保留了start、navigate方法

# 代码拆分合并工具

## 工具说明

我对Backbone和Underscore做了源文件拆分，为了根据优化点的配置来合并这些拆分的文件，我写了自动合并的工具，它能根据优化点的配置来生成需要的代码。

之所以要设计一套规则和工具来做拆分，主要是基于以下原因：

1. 自动化管理细粒度依赖要比手工方便
2. 业务特点的不同可能会要求根据配置的指定生成需要的代码
3. 使得拆分过程可以很好的控制和还原

## 工具设计

工具设计本身比较简单，主要是将源码看作模板，通过加入模板语法使得代码可以合并及合并判断。主要有以下几个功能：

1. 配置

配置文件设置了一些拆分后代码的合并规则，包含：

* 引入的其它常用类库的配置及其命名空间设置

常用的有Zepto、GMU。

* Browser/node.js环境的配置

本次优化取消node.js的支持

* 浏览器支持的配置

主要是移除对IE的支持

* Backbone细粒度的配置

比如对Model的支持或对Model的API的细粒度支持等。

1. 扩展模板语法功能

* 浏览器等条件判断的支持If
* 代码块依赖的支持Require
* 代码引入的支持Include/IncludeIf
* API合并入源码的支持Join

## 源码拆分

* Underscore按照API拆分。

比如\_.has.js

* Backbone按照模块-模块API拆分。

比如Backbone.Mode.js,Backbone.Model.clone.js…

## 源码合并

源码合并时，需要根据配置来做操作。以下几点需要特殊处理：

* 对Underscore依赖的支持需要先解析Backbone后实现Underscore API的提取及合并
* JSON格式的API合并时需要处理分隔符

## 工具特点

### 优点

* 比手工合并方便
* 能够方便的根据不同的要求生成不同需求的源码

这种需求需要尽量控制，否则后续生成各种版本维护起来就比较麻烦

### 缺点

官方版本变更时需要手工处理拆分子文件的对比。

# 优化效果

## 环境说明

代码压缩工具使用uglifyjs，开启-m参数

服务端开启gzip

## 效果数据

|  |  |
| --- | --- |
| **优化点** | **体积**(k) |
| Backbone+Underscore | 10.8 |
| 删除未被Backbone使用的Underscore API | 8.0 |
| 适配Underscore API至GMU | 7.8 |
| 移除IE支持 | 7.4 |
| 移除emulateHTTP/emulateJSON及HTTP PUT/CREATE/DELETE/PATCH支持 | 7.1 |
| 移除Collection | 5.6 |
| 精简Backbone非常用API | 4.3 |
| 移除model | 3.4 |

# 总结

## 关于Underscore

优化后的Underscore比较小，可以考虑将这部分API移出来作为基础库的扩展。

优化后Underscore保留的API(不包含和GMU相同功能的部分)：

1. \_. isFunction
2. \_. isArguments
3. \_.functions
4. \_. bindAll
5. \_.once
6. \_.pick
7. \_. defaults
8. \_. result
9. \_. uniqueId

## 关于Backbone

优化后的Backbone仅保留了View(及Event)、History、Router。还需要实现一个轻量级的Model来解决Rocket的模型共享。而View部分也需要做一些修改或重构，使得和GMU的结合更为紧密一些，以及在事件绑定解除上和Rocket的两套机制抹平。

## 关于Rocket

Rocket作为MVC框架Backbone的最终呈现，在体积优化中，有两点还需要继续关注：

其一是Rocket的脚手架，实际的项目产出代码相似部分是很多的，可以进一步提炼，让业务部分的代码更为精简。

另一个是关于通用subview的提炼。

框架对业务部分的代码精简，要比优化框架本身的体积作用更大也更有效。