前端工程化实践：大前端的转变之路

2017-08-18 00:00

# 前端工程化实践：大前端的转变之路

年初有幸加入到 TalkingData，到现在半年有余。是时候回头看看，这半年走过的路。

团队的充分信任和赋能，使我可以大胆地实践和尝试。

前端团队需要负责很多条产品线的前端开发工作，如果技术规范和技术栈不统一，就很难保证产品的快速迭代。在这半年里，通过推行组件化开发理念、规范 Git 工作流和 Git Commit message，并且将开发语言迁移到 ES6, 引入 Airbnb 规范，团队的工程化能力终于开始上路了。

而整个前端团队，还要经历从组件化到模块化、从引入开源到生态贡献和从纯前端到大前端团队的发展道路。

## 组件化

在工业制造领域，工程师们按设计规格，制造出机器的零部件，就可以快速组装好一部完整的机器。

同样，在软件开发领域，开发团队将设计稿拆分出一个个职责单一的组件，按组件的复用程度去依次开发。像这样从全局出发拆分组件的方式，无疑能带来很大的好处。

如今的前端开发，已不是简单的页面制作加上jQuery，就能满足要求的。前端工程师们要面对的不再是简简单单的网站，而是更为复杂的 web 应用。

### 技术架构统一到 Vue 技术栈

可以说，Vue 是为组件化而生的 View 层框架。它为应用的组件化，提供了很好的支持，还有完善的脚手架工具可用。 团队统一技术栈有很多好处，可以降低产品间人员调动地成本、复用组件和模块代码。

到目前为止，团队中的大部分产品，都完成了向 Vue 技术栈的重构迁移。

### 组件化开发过程

产品更新迭代速度，很大程度上决定了产品成败。组件化开发理念，可以快速拼装出产品，而且每个组件都可以独立的升级或替换。

在产品正式开发之前，我们仔细研究设计图，统计出整个产品所包含的组件。哪些组件可复用度更高， 就优先开发这些组件。一旦这些工作完成，就能快速拼装出产品功能页面。

不难想到，前端开发的效率取决于设计稿的完整度。设计师提供的设计稿越完整，前端工程师的开发效率会越高。

在智能数据市场产品（ SDMK ）的开发初期，就选择了组件化地开发方式，这很大程度上缩短了产品迭代发布周期。

## 源代码管理要规范化

产品团队中，几位工程师分工协作，开发不同的功能组件。各功能组件相互依赖关联，只有规范化的源代码管理流程，才能避免了因多人协作而导致的风险，保证产品的快速持续的功能集成。

### 规范化 Git 工作流

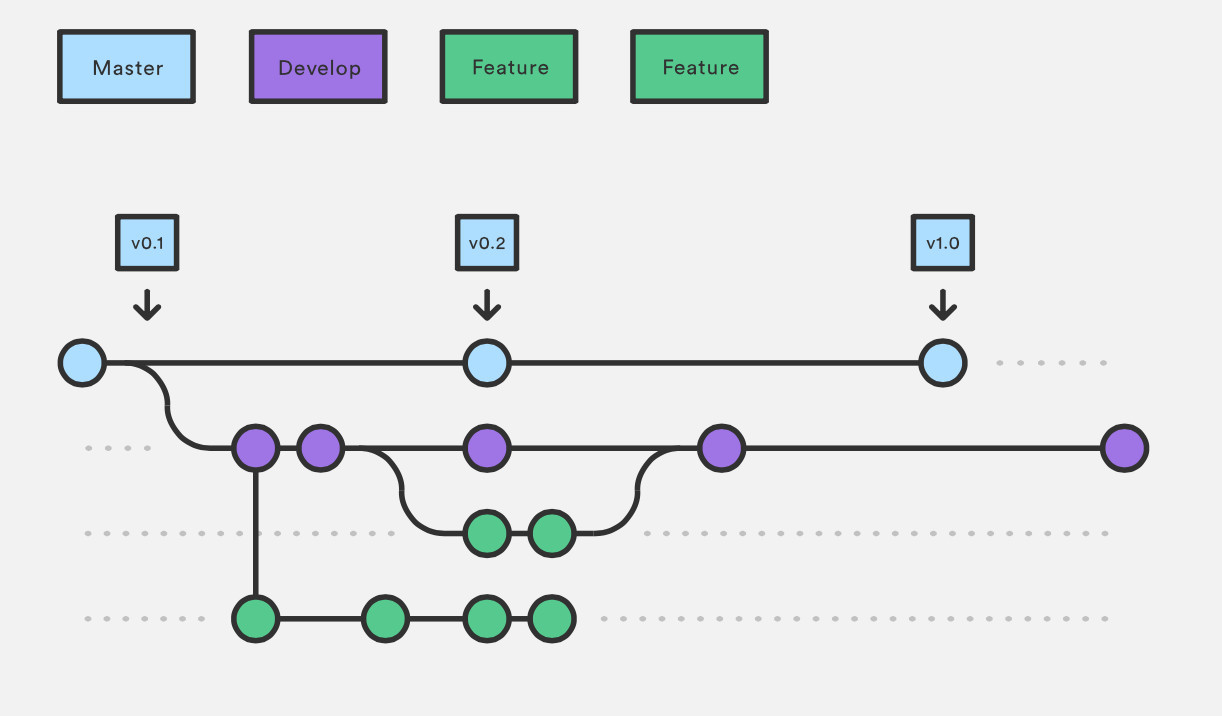
高效的开发团队，需要规范化的代码管理流程做保证，团队直接引入了成熟的 Git 分支开发流程，并采用了每日集成的开发策略。

在产品的攻坚和优化阶段，我们采用了不同的分支开发策略。

在攻坚阶段，每位工程师会在自己的开发分支，完成分配到的功能组件。负责人会在每天固定的时间，将大家的代码 commit 到 daily 分支上，同时还会进行 Code Review，做规范性检查和业务逻辑优化等等。

在产品上线后，进入产品的优化阶段。这个时候大家会将分支转移到 Feature 分支，同样会进行每天的 Daily 过程。

如此，大家每天的工作目标明确、成果可见，每天都很有成就感。

 (图片来源：Git版本控制与工作流，by Sam\_Lau)

### 规范化 Git Commit message

一次代码提交，应该尽量做到原子性，比如：完成了一次 Bugfix、完善某功能文档、对某个方法做了重构等。而每次代码提交，都应该提供格式化的 Commit message。

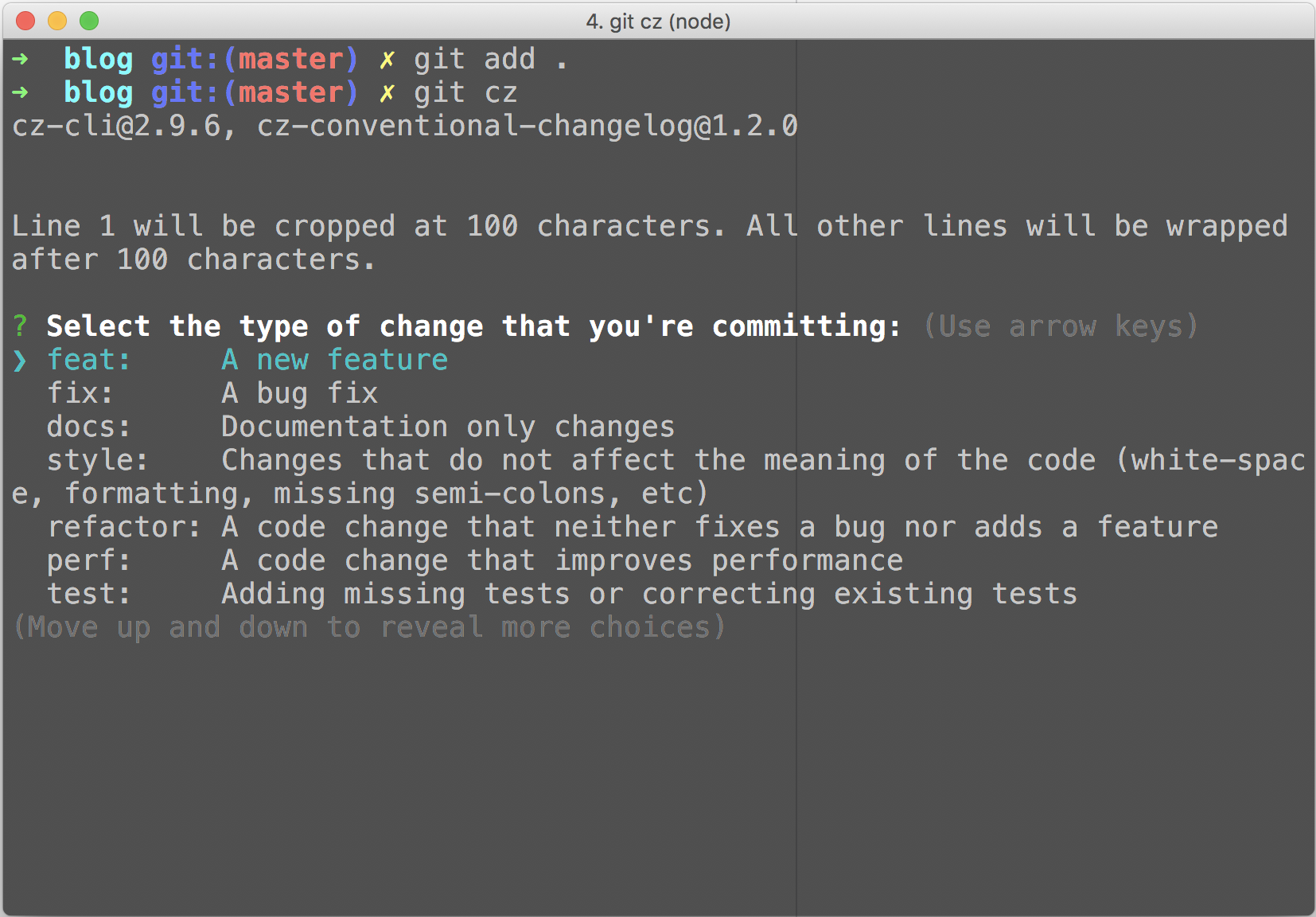
格式化的 Commit message，有很多个好处：

1. 提供更多的历史信息，方便代码 CodeReview。
2. 可以过滤不同类型的 Commit，方便快速查找信息。
3. 还可以由 Commit message 生成产品的 Changelog。

在每天的 Daily 过程中，快速浏览 Gitlab 的提交记录中的 Commit message，可以高效的完成 Code Review。

团队直接引入了广泛使用的 Angular 提交规范，比较合理和系统化，并且有现成配套的工具。

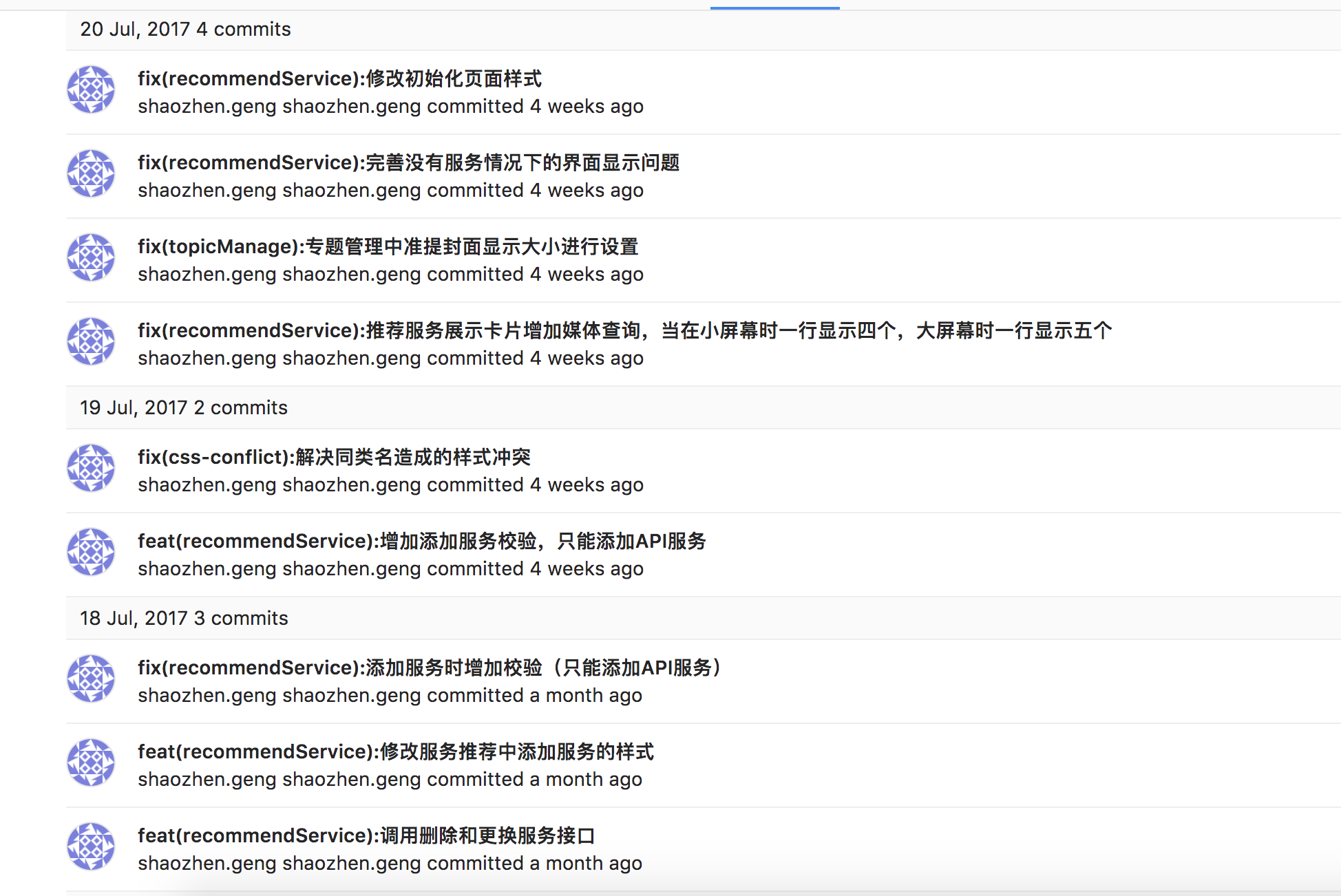
commitizen (安装 npm install -g commitizen):



commit-message

Angular规范：

feat：新功能（feature）  
fix：修补bug  
docs：文档（documentation）  
style： 格式（不影响代码运行的变动）  
refactor：重构（即不是新增功能，也不是修改bug的代码变动）  
test：增加测试  
chore：构建过程或辅助工具的变动

团队规范示例： 

在执行 Jenkins 构建后，还可以清楚地了解，这次构建所修改的内容： 

## 代码风格的规范化

阅读规范化产品的代码，最直观感受是：所有代码，要看起来像是是出自一人之手，无论功能实现，还是代码注释。

在进行规范化之前，团队需要将主要开发语言迁移到 ES6。把匕首换成宝剑，提高了攻城成功的把握。

### 迁移到 ES6 和 SASS

过于灵活的ES5，并不适合开发企业级 Web 应用。

因为产品存在历史 ES5 代码，大家需要循序渐进的完成 ES6 的迁移，比如： 1. 新功能代码，禁止使用 var 关键字声明变量，而是改为使用 let 或 const； 2. 对于函数的声明，鼓励去掉 function 关键字，更多的使用箭头函数和使用函数的默认值； 3. 用模版字符串替换掉之前的字符串拼接；

像这样，逐渐深入更多的ES6特性。从实际效果看，各产品线代码迁移过程，十分地平滑自然。

下面一段代码示例，实现数据的复制，可以发现ES6代码会更简洁高效：

// bad  
const len = items.length;  
const itemsCopy = [];  
let i;  
  
for (i = 0; i < len; i += 1) {  
 itemsCopy[i] = items[i];  
}  
  
// good  
const itemsCopy = [...items];

在 CSS 方面，各产品统一为 SASS 预处理器，为 CSS 引入了许多编程语言的特性和模块化的能力。

### 引入 Airbnb 的 Javascirpt 和 CSS 规范

感谢 Airbnb 开源了其内部的语言规范，免去了大家设计和维护规范文档的麻烦。

团队采用了，目前广泛使用和成熟的 Airbnb 语言规范，有现成的规范文档和工具可用。

只有工具，才能约束大家的编码习惯。团队引入了 eslint 检查工具，再配合 Airbnb的规范，保证了大家代码的规范性。

在代码编辑和“编译”阶段，eslint 插件会自动检查产品代码是否符合规范。对于不符合规范的代码，会抛出错误提示，阻止代码的执行。

除了代码规范的约束，eslint 还为解释型的 Javascript 带来了部分编译特性，比如：

1. 项目中声明了没有用的变量、函数和语句，eslint 会抛出错误提示；
2. 代码中使用了具有一定风险的 for in 循环，也会给出修改建议；

大家在学习 Airbnb 规范的过程中，对照 ES5 和 ES6 语法的不同，快速完成了语言的切换。

对于 CSS 预处理器 SASS，我们尝试了 stylelint 工具，配合 Airbnb 开源的 CSS 规范，来规范化大家的 SASS 代码。但遗憾的是，社区开源的 CSS 规范化相关工具并不完善。编辑器 lint 插件，只找到了 atom 的版本。

## 文档的规范化

工程师们，大都不喜欢写文档。

而实际情况是，文档的更新，一般都是滞后的。比如接口新增了参数，文档也需要做对应的修改，而这带来了一定的维护成本。

### 源代码即文档

组件和模块的设计拆分，应尽量符合简单的单一职责原则。这样源代码就能够做到自解释，文件名表明了文件的功能、方法名称解释了该方法的意图等等。

这里，好的文件名称、方法名称就起到了文档的作用。

对于接口文档，我们设计了声明式的 model 层 apischema （https://www.npmjs.com/package/apischema）。一个 model 的定义，就是一系列接口的文档。

下面的代码示例中，可以直观的看出接口协议、地址和参数规则：

// define api  
export const addInternalGroup = schema.post('/path/to/action', {  
 name: { type: String, required: true },  
 info: { type: String, required: true },  
});  
  
// use api  
addInternalGroup({ name: 'xxx', info: 'xxx' }).then((json) => {  
 log(json);   
});

除了代码，团队中还会有很多东西要沉淀记录，必要的文档还是要有的。

### 用 Markdown 写周报

现如今，除了编程语言之外，Markdown 也早已成为了合格工程师的必备技能。大家的周报，会统一格式，用 Markdown 记录到 gitlab 之中，内容包括：工作概要、详情记录和计划等。

### 大前端周刊

合格的工程师，每周会阅读一定量的技术文章。团队鼓励大家把每周读到的优秀内容，分享给团队小伙伴，简单记录到 Github 的 issues 之中。

每周五会有一位轮岗小编，负责将 issues 中的内容，用 Markdown 编辑整理到 Github pages 中。

这就成了我们的大前端周刊（https://talkingdata.github.io/fsd/）。



fsd

## 未来展望

### 从组件化到模块化

Vue的组件化能力，缩短了开发周期。但纵览多条产品线的代码，依然存在很多职责相同的代码。比如好几个产品，各自实现了一套time-format模块。

我们需要将公共代码，提取到模块，不同产品分别引用这个模块就好了。感谢社区为我们提供了完善的模块化能力。

产品引入的iView，为大家竖立了榜样。团队成员，都要适度参与到开源社区，每各人都是前端生态的一部分。

提倡渐进式地开发方式。大家在完成功能开发后，能够多一些思考，考虑哪些代码具有通用性，将这些代码提炼出去，最后 npm publish.

希望每位工程师，不止于业务，向前一步。

### 从纯前端到大前端

工程师只会浏览器端的开发技术，就很难设计出功能完整的作品，创造力也因此受到了限制。团队鼓励大家，在掌握了前端的技能后，能够突破边界，再多涉及一些 Node 开发技术、多了解一些 Http 协议的常识。

工程师站在大前端的高度，就能在不同的层面考虑问题，给出最适当的解决方案。比如为了提高产品发布的可靠性，要引入灰度发布流程，团队实现了完整的灰度分流和对应的配置平台。

浏览器端和 Node 端，要关注的方面不同，团队不能因此失去专注。半年来，希望团队能够从单纯的前端团队，成长为大前端团队。按照二八原则，希望能有 20% 的团队成员能够专注在 Node 方面。对于其他队员，也希望能够在仿照固有的模式，开发一些 Node 端的代码。

从这一想法出发，团队启动了 Flclover（四叶草）项目，一个基于 Koa 的企业级 Node 解决方案。

目前 Flclover 稳定在 0.7.x 版本，已经在内部的两个项目，完成了验证。

### 从引入开源到生态共建

Hats off to Github.

团队中引入了大量的开源模块和工具，像 NodeJS、NPM、Webpack、Vue 和 iView 等，这些开源工具让大家可以更专注在业务上面，不需要花时间去研发这些基础的东西。

在我刚入行的几年，除了业务研发，还需要花费大量的时间和精力，开发工具类的东西。像公共组件、模块加载器和打包工具等，如今所有这些，都可以在社区找到完整的解决方案。

团队鼓励大家都参与到开源生态的共建，形成从引入到贡献良性的开源闭环。

## 最后

半年来，团队完成了规范的推广和技术架构的统一，这些为团队规模的迅速扩大，打好了基础。接下来，团队还要普及渐进式的开发方式，完成从纯前端到大前端团队的过度。

加油 2017！