近几年前端技术盘点以及 2016 年技术发展方向

作者: 小胡子哥

Web 发展了几十个春秋，风起云涌，千变万化。我很庆幸自己没有完整地经历过这些年头，而是站在前人的肩膀上行走。Web 技术发展的速度让人感觉那几乎不是继承式的迭代，而是一次又一次的变革，一次又一次的创造。这几年的前端，更为之甚！ 我要说话

我从 12 年底开始接触前端，12 年之前的前端发展情况只能从上一辈的笔触中领会。本文会盘点从 09 年开始到 15 年间前端技术的革新，同时也会从多个角度，解读近几年前端技术发展的潜在因素，其中穿插了若干对前端演进的拙见，难免会有错误和疏漏，忘读者可以补充和斧正。

我要说话

那些年，一度追捧，一度放弃

下面，花一些篇幅简单回顾下 09 年到 15 年前端的发展历程。 我要说话

09 年，基础类库完善，寻求突破

09 年之前，JavaScript 还处于对自身语言的完善过程中，而到了 09 年，JavaScript 类库已经颇为成熟，jQuery/Prototype/Script.aculo.us/Dojo 等都已经发布了好几个 stable 版本，各大类库也是相互吸收优点，不断完善并提高自身性能，然而功能上已经没有太多增加的势头。部分框架开始了思想上的转变，更加注重前端开发的组织和结构，条理性强了很多，如 YUI，Dojo 等。 我要说话

从 ECMAScript 规范的争执，开启了浏览器引擎大战，各大厂商也趁机瓜分 IE6 份额，Chrome 和 Firefox 在这场战役中取得小胜，V8 也敲响了前端的大门。为了迎合市场的激烈竞争，IE 开始了升级之旅，09 年初发布 IE8，全面兼容 CSS2.1。 我要说话

而此时，Node.js 和 3G Mobile 这两只巨兽开始浮出水面，Web 标准也开始向 HTML5、ECMAScript5.0 靠拢。 我要说话

10 年，Web2.0 深入人心，开始性能挑战

毫无疑问，这一年，各大巨头都看清了 HTML5 是 web 发展的未来，在保留原来前端技术的状态下，都簇拥着拉扯 HTML5 的裙摆。富客户端应用也在这一年蓬勃生长，ExtJS/Dojo 摇身变为企业级框架，各类组件化概念和产品如约而至。 我要说话

延续着 09 年的变化，10 年的前端显得颇为沉寂，然而在标准的运用和推动上，各大厂商也是十分卖力。IE 9 出来了预览第三版，iPhone 的 Safari 已经能够支持众多 HTML5 内容：Canvas/Video/Audio/Geolocation/Storage/Application Cache/Web SQL Database 等。 我要说话

W3C 宣布成立 Web 性能工作组，Google 和 Mozilla 纷纷推出应用商店，浏览器调试工具也丰富了起来，人们开始更多地关注开发体验和性能问题。 我要说话

11 年，HTML5 抗大旗，Flash 堪忧

2011 年 HTML5 的技术发展和推广都向前迈进了一大步，语义明确的标签体系、简洁明了的富媒体支持、本地数据的储存技术、canvas 等等各类技术被广泛应用。这一年，很多 web 开发者也面临一项技术的抉择，HTML5 or Flash？从 Flash Player 11.1 开始，Adobe 不再继续开发面向移动设备浏览器的 Flash 插件，积极投身于 HTML5，这意味着 Flash 技术的凋零。 我要说话

这一年，HTML5 游戏火爆到了一个高潮，他的低门槛和高收益让很多开发者眼红，正因如此，移动端开发工具和调试工具也日益成熟。jQuery 已经成为大小公司日常开发的标配，成千上万的 JQ 插件让网页开发变得尤为轻松，而随之而来的也是页面的臃肿和性能调优的深入探索。 我要说话

Node.js 已经悄然崛起，在 github 上的访问量已经超过了 Rails，国内的云应用开始尝试使用 Node.js，Node.js 相关工具也纷纷出来。 我要说话

12 年，响应式开发，工程化推进

随着硬件技术的发展，各手机厂商又开始骚动起来，为了占有更多的市场，不断提高产品的性价比，体验也得到了不断的优化。借着先前两年 HTML5 刮起的东风，移动端上的 web 开发也颤抖了起来。移动端的开发挑战不亚于 PC 上对多个浏览器的支持，这一年，萌生了众多移动端框架，如 Sencha Touch/Zepto.js/JQ Mobile 等，相对 PC 端框架，它们更加轻便。 我要说话

而移动端的崛起，带来了许多终端开发难题：多终端适配，多分辨率适配，远程调试等等，而随着这些难题一个个被解决，移动端生长的势头变得更加强盛。此时 Twitter 也推出了 Bootstrap， 这个前端开发工具包不仅方便了前端，也方便了后端同学，它的出现让快速建站更加简单。 我要说话

编程思想的切换，迎来了 CoffeeScript 和 TypeScript，这两个预处理语言的出现又为 JavaScript 引来了不少其他方向转型过来的开发者。JavaScript 的兄弟 Node.js，也在命令行领域开拓了一片不小的疆域，甚至有动摇 Perl 和 Ruby 地位的趋势。 我要说话

在前端工程化上，几个派系相互争斗，产出了 AMD、CMD、KMD 等规范，也衍生了 SeaJS、RequireJS 等模块化工具。前端在这一年很有跳跃感。 我要说话

13 年，爆发式增长，百花齐放

规范和标准上有不少产出。Web Components 的出现给前端开发开辟了新思路；WebDriver 规范的出来推动了自动化测试的进程，ECMAScript 6 的规范草案落地，Webapp 工作小组在这一年也是相当活跃。 我要说话

Chrome 浏览器在这一年也有了很大的突破，开始支持 SPDY，使用 Blink 取代 webkit 作为 Chromium 的新渲染引擎，Chrome DevTools 的调试体验大幅度提升。这一年中，Chrome 连同其他浏览器厂商快速推动了各项草案规范的实现。 我要说话

语言能力上依旧在增强，并且从 JS 开始扩散到 CSS，出现了 LESS、SASS 和 Stylus 等预处理语言，Web 开发变得更加紧凑。 我要说话

而在无线端，应用不再局限于 Webapp，由于流畅度、性能等方面不能满足用户体验的需求，各大公司开始转向 Native 方向的研究，进而出现了 Hybrid 和 PhoneGap 的繁荣，它们为 JS 调用了提供更多的设备 API。 我要说话

Node.js 大放异彩，很多公司在生产环境中使用 Node.js，同时也出现了诸如 Express、Meteor 等小巧的快速搭建 Node.js Server 的应用框架。 我要说话

各浏览器的调试也是种类繁多、功能丰富，PhantomJS 在自动化测试上开始取代 Selenium，出现了众多的远程调试方案和工具。 我要说话

前端工程化开始普及，各公司开始推出自己的前端集成开发解决方案。 我要说话

14 年，移动端的崛起，HTML5 和 ES6 落地

HTML5 正式定稿，这意味着，web page 正式演变为 web application。ES6 华丽丽走进前端，走的很稳重，它的 Module/Class 等特性已经完全让这们语言具备了开发大型应用的能力。 我要说话

大而厚的基础库难以满足灵活场景，Mobile 要求极致体验，MV* 库铺卷而来，如 avalon/angular/knockout 等。 我要说话

Web Components 跨终端组件快速发展，移动端开发迎来一次升华。Node.js 前后端分离的流行，中间层的出现改变了前后端的合作模式。 我要说话

2014 是颠覆式的一年，前端发展在这一年开始形成了一个短暂的稳定格局。 我要说话

15 年，观念的转变，步入前端工业化生产

今年格外引人注目的框架是，类 React。Facebook 在 React.js Conf 2015 大会上推出了基于 JavaScript 的开源框架 React Native，它结合了 Web 应用和 Native 应用的优势，可以使用 JavaScript 来开发 iOS 和 Android 原生应用。在 JavaScript 中用 React 抽象操作系统原生的 UI 组件，代替 DOM 元素来渲染等。敲一次代码，能够运行在多个平台上，其优势可见一斑。除了 React ，还有手机淘宝推出的 Weex 框架，它吸收了 vue.js 的编程精华，编程风格更加简约。 我要说话

在众多构建工具中，如今潇洒存活的并不多。体验完 grunt 和 browserify 后，gulp 顺势而至，尔后又出现了 webpack、jspm 等。而包管理工具，经历了 components、bower、spm 后，npm 开始主导整个市场。 我要说话

Node.js 的应用已经铺天盖地，各大公司前端都把 Node.js 作为分离前后端的主要手段，并且在测试、监控等方面沉淀了大量内容。不过，这个市场是很苛刻的，Node.js 的性能难以达到 C/C++ 的水平，那么接下来要做的就是要提升性能，至少得接近 C/C++。 我要说话

Web 规范和标准

最开始，我们看到的 JavaScript 还只是一个简单的脚本语言，配合着 AJAX，在网页上翻腾了好几个年头。随着互联网趋势越来越明显，互联网业务量和业务复杂度不断增加，很多网页变得相当复杂，如让我们震惊了好一会儿的 Gmail，交互复杂，体验优良。为了更好的多人协作，代码中的 Utils 库越来越大，在这些库中，基础部分更多的是对 JavaScript 语言本身的拓展，比如给 String 加一个 repeat 函数，再加一个 trim 函数，再加一个 endWith 函数等等。 我要说话

复杂的业务中会经常看到一层又一层的回调处理，回调的嵌套让代码的可读性变的很差，而且很难将多个异步并行处理。为了改变这种编程范式，我们做了很多的思考，使用事件监听，使用各种手段拉直回调，平坦地调用。 我要说话

慢慢的，如果你在关注 W3C 小组的动向，会发现，那些被认可的，并且被广泛重复定义的东西，都被纳入了标准。最开始的 jQuery/prototype，前者主要是对浏览器做兼容处理，让开发者不再把精力放到浏览器的差异上；后者是对语言本身的拓展，对 JavaScript 各种类型做拓展，并且提供了一套拓展任何对象的功能集。而现在的开发，我们很大程度上不再依托这些类库。规范和标准已经把这些差异都统一了，String 中自带了 includes/startsWith/endsWith/repeat/padStart/padEnd 等函数，Array 自带了 from/forEach/of/keys/values/find/findIndex 函数... 我要说话

规范的标准是为了让开发者得到更好的编程体验，编程不是目标，目标是将编程生产力转化成实际效益，越少的阻碍对开发者越有利。各浏览器厂商当然也认识到了这一点，他们不断地提升自己产品的体验，将标准中的新特性都融合进去，比如 ES6 中的 Promise/Generator/Class/Module 等等。在这些内容普及之前，我们不需要加入 jQuery/prototype 这些「不纯粹」的东西，而是添加两个 shim 和 polyfill，如 es5-shim，html5shiv 等等。待到山花烂漫时，再轻松删掉这些补丁程序。 我要说话

这两年工程化很热，W3C 小组也看到了，这就是市场的需求，为了完成一个大型应用的编程，就必须模块化、组件化，于是在规范中也出现了 Module & Module Loader；Node.js 的到来，让很多前端工程师开始接触数据库操作，面对巨量的异步，我们忍气吞声写了无数的回调地狱，尽管使用了很多 Promise 相关的操作，程序结构依然松散难以阅读，于是规范中也开始出现了 async/await 等对 Generator 的上层封装。文字已经不能满足当代人的沟通需求，音视频等富媒体传输走进了我们的生活，于是规范中也出来了 WebRTC/WebAudio 等规范。 我要说话

只要规范出来了，后续市面上就会根据规范来实现一套 shiv，这些 shiv 提供了同样的 API，提供了同样的编程体验。当浏览器自我进化完成之后，这些 shiv 也将成为历史，被开发者遗弃在代码的注释之中。这些都是规范和标准的魅力，它的存在，就是让开发者把精力投入到自己的业务之中，编程和范式的工作交给它。 我要说话

在 这里 可以看到，W3C 各个小组最近都在干啥。标准不能囊括一切。 我要说话

生态的自我完善和自我拓展

技术的更迭过于频繁，我们能够清晰地看到，很多人还在用更迭前一波甚至是前好几波的产品。

我要说话

当年的 IE6，在战场上鏖战了 10 多个年头，依然屹立不到，而现在它在市面上依然有百分之一左右的占有率，这种小强精神不得不让人肃然起敬。“只要用户在，我们就得追随”，这可能是很多公司的服务理念，因为用户就是潜在的利润。正是因为这种服务理念，成就了 IE6 一个又一个的 5 年！然而低本版的 IE 已经不仅仅是被前端从业人员抵制和排斥了，网络安全、网络运维、QA 等等，各个技术岗位的人员都开始对他不屑，它的存在对工作效率、对安全、对很多方面产生了极为不良的影响，甚至影响到一些核心内容的推广，所以 2016 将是低版本 IE 消亡的一年，我也呼吁业界所有的朋友举起义旗反抗起来！ 我要说话

庆幸的是，也有人开始吃螃蟹了。从支付宝到天猫到淘宝，阿里巴巴在很多业务上已经主（bei）动（bi）地放弃了对 IE6 和 IE7 的支持，甚至在统一接入层直接做了 302 跳转，提示用户更新浏览器或者引导流量到无线端。这是一个好的开始，我们期望这也是业界达成共识的开始！

我要说话

HTTP 协议，从 1.0 快速过度到了 1.1，整个互联网的上层建筑变的十分稳固。当然，我也了解到依然有很多产品还是保持了 1.0 的状态，据说电信公司的很多产品就是使用 HTTP/1.0 进行通讯，这无疑让人惊愕。为了追求更高的效率，减少网络传输中的无效流量，W3C 工作组对 HTTP 协议也做了重新的定义，SPDY 就是 13 年比较火热的一个话题，Firefox 和 Chrome 都陆续开始支持 SPDY，后来在 SPDY 的基础上做了升级，正式定义为 HTTP/2.0，它的一个很大特点就是多路复用，这个小小的特点改变了我们前端编程的很多优化模式，比如 我要说话

- 域名不是越多越好，为了能够充分利用浏览器的连接数，我们给 JS 和 CSS 开一个域名，给 img 开好几个域名，网页打开的时候，恰到好处的利用浏览器的连接数上限限制。HTTP/2.0 的多路复用，就是可以在一个 HTTP 请求中进行多个资源的传输，如果域名散列，反而不能利用这个特性
- 资源合并没有任何优势，以前的资源合并是为了减少请求数以节约建立 TCP 链接的网络开销和头部传输的流量开销，而在 HTTP/2.0 中，一个 HTTP 请求上完全可以把所有的资源全部推送过来，如果合并了资源，反而不能良好运用浏览器对资源的缓存。

当然，除了多路复用，还有很多其他的优化，比如传输的数据为二进制流，HEAD 头会被压缩处理，服务器可以向客户端推送内容等。在这个技术水平指数式增长的年代，我相信以后的革新不会比消灭 IE6 痛苦。 我要说话

模块加载上，经过了各派系的争论之后，流传下来几个不错的产品 SeaJS、RequireJS 等，那么那个模块加载器将成为工具平台中短暂的终点呢？似乎这些都不是。当我们按照规范中的方式进行模块定义，按照规范中的方式加载定义的模块时，加载这个流程就显得不那么重要了，因为这些事情最后都会变成 shiv/polyfill 的事情，最终会变成浏览器的固有属性。 我要说话

当一个东西在社区中被暴力追捧的时候，会有很多衍生的产品出来，当这些衍生物根深蒂固时，可能又会出现一个更加原生更加符合开发习惯的东西出来。就像 jQuery，我们为它编写的插件不计其数，而在工程化的需求冲击下，它却显得那么的弱不禁风，因为它关注的点和当前的发展态势不太吻合，仅此而已。 我要说话

Mobile 的发展驱动着战场的转移

记得当年拿着 Nokia5230 学完了 HTML 和 JavaScript 的入门，那屏幕尺寸也就是三个手指的宽度，紧紧攥在手里看着页面混排效果极差的网页文档。 我要说话

现如今，iPhone 都出到 6s 了，一个版本一个尺寸，而且尺寸越来越大，还有各种宽高不一的 Android 机器，种类繁多。以前的触屏是电阻式，只支持单点触碰；而现在电容式的触屏精度更高，还支持多指触控，这如丝般顺滑的体验在三四年前是完全体会不到的。曾经手机开一个程序久了就会卡，动不动还会自动重启；而现在的手机开一堆程序，完全无感知，这就是硬件发展前后的差异。 我要说话

手机已经成为了人们生活中不可或缺的一部分，甚至成为了一些人身体的一部分，淘宝今年双十一的数据显示，国内移动端的消费比例已经远远超过了 PC 端，占比 68%。面对庞大的用户，我们的技术是否做好了充足的准备，这里还得打一个问好。 我要说话

PC 上那一套经验不是直接搬到移动端就可以使用了，在移动端还需要解决更多的问题： 我要说话

- 多分辨率问题，这里涉及到了响应式设计和前端响应式技术
- 不同网络环境的网页加载优化问题，2g/3g/4g/wifi
- 手指交互带来的一系列体验问题
- 为了提升用户体验，将 Web Native 化 —— 类 React 技术带来的一系列问题
- 远程调试问题
- 移动安全问题等等

上面提到的问题很多已经有了优秀的解决方案，当然也有很多未提及的。WebApp 的性能、流畅度和稳定性远远不如原生应用，同时它也无法良好地运用设备提供的原生功能，这些都是大家转投 Native 的原因。 我要说话

端的融合

不同分辨率的手机，不同物理尺寸的终端，为了保持良好的视觉体验和用户体验，我们不得不为每一个尺寸写一份 Media Query 代码，那么对应的，设计师也需要设计多套版式供前端使用，这给设计师、前端和测试带来了无尽的麻烦。为此，我们通过前端技术重塑屏幕，重新定义像素尺寸，使用流式布局，通过百分比来响应不同的终端尺寸。这是端的融合。 我要说话

后续的 Mobile 的技术发展方向上，应该是相当明确的。很多公司都是三套人马维护三端的程序，iOS、Android 和 Web，而这三端做的事情都是一样的，一样的界面，一样的后端接口，一样的交互方式。为了能够快速响应业务的变更，我们不得不将三端合并为一端对待，用一套程序编程成三端代码，然后发布到三个平台上。这也是端的融合。React 系列技术发展到此，绝对不是终点，它只是一个探路灯，给我们照明了方向。 我要说话

技术需要为业务做保障，而好的技术是能够及时响应业务的变化，我们不可能投入大量的人力在 Web 的修补工作上，通过开发统一工具，屏蔽端和端之间的差异，统一开发模式和开发体验，这才是 Mobile 的未来。 我要说话

当然，回到我们之前说的规范和标准，我们目前所做的「屏蔽差异」工作，今后，也会有统一的标准来规范，目前手机厂商没有这个共识，是因为还处于当年 Chrome、Firefox 抢占 IE6 市场份额的阶段。端的最终融合在于一个统一的标准，以及强有力的执行。 我要说话

栈的融合

我刚接触前端的时候，还没有听说「全栈」，Web 技术栈往小里说，包含了从前端设计、交互、前端实现、网络数据传输、后端实现、后端运维和数据库等几个方面，能短时间内从无到有实现这么一套系统，并且能够抗得住一定流量冲击的人，我们可以称之为全栈工程师。能够有架构有条理地实现这套系统，并且抗得住大流量、有集成测试、有监控的，这种我们可以称之为资深全栈工程师。现在不乏这种人才，也不乏自吹为这种人。 我要说话

栈的融合得益于 Node.js 的出现，作为前后端分离的桥梁，它拉近了前端工程师与后端的距离，有的人在这座桥梁上卖力行走，渐渐的也从前端走进 了后端，甚至走进了后端的运维。至此，前端也拥有了部署和发布整个应用的能力，这是一个质的突破。 我要说话

使用 Node.js，简单几行程序便能实现一个 web 服务器、便能搭建一个多人聊天的网页，它的便捷性可见一斑。NPM 社区的发展，沉淀了成千上万的组件包，一行命令即可获取，这种组件拼凑式的开发，任何功能的实现都不会显得太复杂，而这里的「不复杂」也蕴含了无数的坑坑洼洼，在这一层的融合上也会遇上不少阻碍： 我要说话

- 冗余的庞大的包内容，为了使用一个小功能，我们从网络上拉取下来一个巨大的包，而且这里的「巨大」对很多人来说都是无感知的，很少会有人进入 node_modules 去查看依赖的第三方包是如何实现的，实际情况可能会相当震撼，第三方包还引用了一堆第三方包，这些包都会在 Node.js 执行的时候被收纳进去，放在内存中。
- 猛烈的迭代，今年的 Node.js 被人嫌弃迭代太慢了（当然，这是表面原因），走出了一个分支 io.js，发展了一会儿，进度赶超了 Node.js，后来觉得一家人不干两家活，又合并回去了。虽说上层 API 几乎没有变化，但是底层却被翻了一个天。
- 偶尔的巨大漏洞，每隔一端时间就会暴露 Node.js 存在漏洞，这些漏洞的补救措施就是立即升级版本号，比较让人担心受怕。
- 后端意识不强烈，前端占领了中间层的开发，有的时候还干这后端的活儿，然而却没有后端沉淀多年固有的意识，测试和监控做的相当潦草。

JavaScript 从客户端的脚本语言纵身跃进进入了后端行列，而今也开始深入到移动端 Native 领域，确实是无孔不入，这可能就是语言的特性，也可能是技术本身就在寻求融合点，把有差异的地方全部躺平，然后用统一的方式去关注业务，关注用户。端和栈也在融合。 我要说话

后端服务化，云数据，云安全

用户体验变得越来越重要，响应式技术的发展也是后续网页应用的一大特点，端和端之间的差异只是在表现上，数据这一层差异不是特别打，很多应用 PC 和 Mobile 共用一套接口，或者 Mobile 的接口在 PC 接口的基础上做了一层包装，对接口字段做了些许删减。后端为了响应各个端之间的数据需求，也需要关注数据的可利用性，接口包装的拓展性等，这是后端服务化的一个表现。移动端的开发上，前后端间隙十分明显，越来越多移动端应用的发布已经脱离了后端，前端完全通过异步方式获取数据。 我要说话

业务变化很快很快快，今天这个产品被并购，明天那个业务被砍掉，每个人负责的业务线可能冷不丁地就变了。很多大公司的决策是由上往下的，上面微动，下面可能就是大动，可能某个部门就不存在了，也可能被划分成几个产品部门。 我要说话

所以「大后台，小前台」的趋势必然形成。前端，毫无疑问，在这个前台之中。前台的特点是灵活的，多变的，可快速重组的。对后台而言，为了响应前台的变化，需要提供更细粒化的 API，将数据打散，打得更加零碎，零碎的数据易于重组，这是在考验后端的架构能力。如今，很多前端也都是半栈工程师，盘踞在前后端中间层上，然而如何迎接这种后端服务化的模式，似乎这个准备还是不够充足的。 我要说话

GraphQL 的出现场景跟 React 类似，React 是前端应对不同场景的一种强有力手段，而 GraphQL 则是后端应对不同需求场景的一次尝试，Web APIs 将会成为 Web App 和 Mobile App 的一个中心点，前端基于后端的 RESTful 服务构建应用，这里面存在太多未知的问题需要探索，这是一个大数据下探索的新起点，也给前端开发者创造了无数的可能。 我要说话

这几年各类网盘，各个云服务商都在抢占市场，有提供图片储存的，有提供 CDN 静态资源缓存的，有提供大文件储存的，也有卖数据库服务的。种类繁多，而归根到底都是，你付钱给我，我提供储存和安全，还提供方便的 SDK 让你获取自己的数据。云服务卖的是一套服务，它是把所有人的数据风险集于一身，用强硬的技术做安全防御。云，赋予了我们无穷的想象空间。 我要说话

三辆马车，我们还差一辆

开发功能对很多人来说是轻松活儿，基本的前端语言加些复杂的特效，实现成本不会很高；即便是搭建一个网站，使用 Node.js 社区中的框架也能够轻松实现。然后极少人会去关注每个功能点的测试，一个项目下来基本看不到测试用例，更不用说会去做监控相关的事情。结果就是，踏过了无数的坑洼之后终于上线了，而后续加功能的时候发现，加了东西就跑不通，新内容影响了之前的逻辑，只好去修复之前的逻辑，修好之后发现更早之前的逻辑又不通了，整个修复过程就像玩多米诺骨牌。 我要说话

程序开发三板斧：功能、测试和监控。在 github 上可以看到很多程序都加入了持续集成，这是一个好兆头，以为着我们写的程序也越来越健壮，至少贡献给世人使用的程序是健壮的。很多程序的代码覆盖率也达到了 90%+，这些数据都是重视测试的证据。 我要说话

然而，三辆马车，我们最后一辆依然没有开动起来。很多公司都会有自己的 log 平台，每个用户访问页面中的任何一个链接都会将用户信息和访问信息以 log 日志形式收集到 log 平台上，然后通过监控平台或者离线分析的方式，获取业务数据或者技术数据，进行分析和二次开发。这些东西在大公司见的很多，而这方面的东西在前端，尤其是使用 Node.js 做程序开发的前端身上，看到的并不多。 我要说话

最后

2016 年，我觉得技术上的新创造会稍微缓和些，这两年很多人已经被新技术冲击得有些找不着方向了，同一类东西，前者还没学完，后者就开始火爆了，紧接着又是一阵技术的凋零和新技术的出现，这样搞久了也会有一丝的疲倦。而更多的会关注，如何更好地服务多端，如何更大幅度地提升开发体验和用户体验，很多技术都会往性能、往极致这个方向上钻研。 我要说话

写长文真不轻松。写到这里，感觉说的不通透，还有很多想说的，但是个人理解力有限，也难以表达全面。技术的变化很快，今天说过的东西，到了明天就可能过时了。我们猜不透未来，只能把现有的东西好好消化吸收下，留下一个话柄，给读者吧。