

14 | 别有洞天：从后端到前端

2019-10-11 四火

全栈工程师修炼指南

[进入课程 >](#)



讲述：四火

时长 19:11 大小 15.38M



你好，我是四火。

前两章我们分别介绍了网络协议和 Web 接口的知识，以及网站等应用的 MVC 架构和其衍生发展而来的各种设计模式。以上希望你已经充分消化吸收了，今天我们将迈入第三大基于 Web 的全栈技能领域——前端。

为什么要学习前端技术？

“前端”简简单单两个字，背后却有着纷繁的故事和复杂的情感。这也促使我产生了想多聊一聊这个话题的想法，一般的技术在“为什么要学”的方面我往往简言述之，但对于前端技术我想为此破例。

前端一直以来是一个颇具争议的领域，有人极其喜爱，有人避而远之，和多数“天下太平”的技术相比，这确实有些令人费解，但我认为这其中的原因至少包括这样两点。

第一，某些技术人员或管理者单项技术进步，但思想却依然陈旧迂腐，停留在“前端就是改页面”这样老旧的思维程度，认为前端没有技术含量且无法创造显著价值。

第二，相对于软件领域的通用技术，前端极低的入门门槛，导致号称“懂前端”的工程师技术水平严重参差不齐，这反向影响了整个技术群体的形象。

如果你志在学习全栈工程，前端就自然是你无法避开，且还需努力驾驭的领域。但即便你的长期目标不在此，也应该学习前端技术，因为它能给你带来的好处是多方面，且是别的技术所不可替代的。具体包括这样几个方面。

首先，它可以帮你开阔眼界，为你的思维模式带来新的选项，整个全栈技术都有这样的特点，但是前端技术在这方面尤其明显。前端技术的结构和软件其它领域有着显著的不同，技术发展极其迅速，技术之水深不见底，开源社区百花齐放。我们也将在本章中体会到前端领域所需要的不同的思维模式。

其次，它可以帮你形成快速原型、即时验证和独立展示演示的优势，在迅捷的反馈中设计和编程。在我参与过的 Hackthon（黑客马拉松）数天的短期竞赛中，产品经理、程序员和数据科学家被认为是最合理的一组搭配，懂前端技术的程序员总是对互联网的用户交互、数据采集等方面特别有经验，在展示环节还可以快速地做出非常优秀的效果来。

再次，它可以帮你建立产品思维。有人认为它能用来解决用户的核心问题，但实际上往往不是，解决核心问题主要还是靠后端的代码，但是前端的代码却是和用户最贴近和交互的部分，一个优秀的前端工程师总是具备非常强烈的产品属性。

我记得以前在一个团队中负责一个 portal，有别的团队的同事私下里抱怨，说我们做的东西被 portal 一展示，用户都说好用，结果都默认是做 portal 的团队做的了。其实这是一个很现实的问题，无论产品的功和过，即便它的组件分层再深，用户的眼光往往只到很浅的层次就断了。有句话叫，“没有声音，再好的戏也出不来”，如果说，产品的功能性能是它的硬实力，是这出戏的画面，那么前端带来的用户体验在很多情况下就是这出戏的声音。

最后，前端技术是全栈工程的必备技能。它可以让你拍着胸脯对用户说，“这个可以做”，“这个不能做”，而不是说，“我去和前端确认一下这个交互能不能实现”。产品做

出来，也不至于成为一个号称装着高性能引擎，却裹着破布毯子的“豪车”。

遗憾的是，现实中有不少迈入职场没有几年，却已经给自己打上“前端工程师”“后端工程师”等标签的程序员朋友。我觉得他们可能是受到了某些万恶的职业生涯规划鸡汤的影响，这些标签会让他们在面对新技术和新机遇的时候，觉得身处“不属于自己的领域”而选择封闭自己。


因此我的建议是：**职业生涯不宜过早做过细的规划，除了技术深度，也需要在技术广度上积累，等到一定程度以后再来选择自己的发展分支路线。**而且，某些特定技术领域，在程序员给自己打标签的时候，压根还没有发展成熟，等到发展起来，时机真正到来的时候，只有那些原本“不偏食”的优秀程序员才能够脱颖而出。

思维模式的转变

如果你具备后端开发的经验，刚刚开始从后端转向前端，你可能会发现，有很多想当然的理解，不再适用，有很多想当然的解决方法，也不再有效。

1. 应用事件驱动编程

来看这样一段 JavaScript 代码：

 复制代码


```
1 console.log("1");
2
3 setTimeout(function timeout() {
4     console.log("2");
5 }, 0);
6
7 setTimeout(function timeout() {
8     console.log("3");
9 }, 5000);
10
11 console.log("4");
```

代码中有四处打印，setTimeout 接受两个参数，第一个参数表示调用逻辑，第二个参数表示等待多少毫秒后再来执行该调用逻辑。

你觉得打印结果应该是什么？

是 1 -> 2 -> 4 -> 3 吗？先别急着回答，我们好久没动手了，让我们来动手看看结果吧。

在 Chrome 中，任意一个页面打开浏览器的开发者工具，在 Console 标签下，把上面的代码复制粘贴进去，于是我们看到这样的输出：

 复制代码

```
1 1
2 4
3 undefined
4 2
5 3
```

这是为什么，上面的 undefined 又是什么？

为了回答上面的问题，我们需要了解 JavaScript 执行机制中的 Event Loop（事件循环）来理解上面的代码。

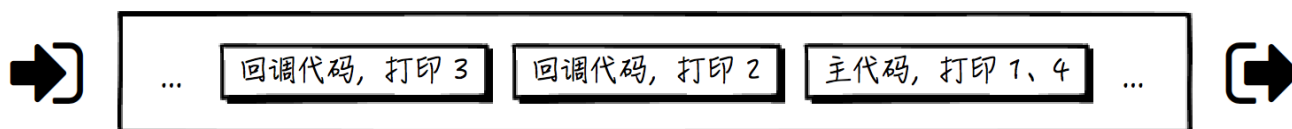
在写后端代码的时候，你可能已经习惯了使用进程（process）或者线程（thread）来对付需要并行处理的逻辑，Java 如此，Python 也如此。进程或线程可以说是“真并行”，虽然微观地看，它们可能会顺序占用 CPU 时间片，但宏观地看，代码在二者中执行互不阻塞，是并行执行的。

而在 JavaScript 中，在浏览器中，你看到眼花缭乱的效果和变化，却是“假并行”，是一个彻头彻尾的“骗局”。为什么这么说？

因为浏览器中 JavaScript 代码的执行通常是单线程的（对于 [Web Worker](#) 这样的“例外”我们暂不讨论）——一个线程，一个调用栈，一次只做一件事。

具体说来，在整个 JavaScript 的世界里，引起代码运行的行为是通过事件驱动的，并且**全部是通过这唯一的一个勤奋的工作线程来执行的。那么当有事件产生的时候，这个工作线程不一定空闲，这就需要一个机制来让新产生的事件排队“等一等”，等当前的工作完成之后，再来处理它。这个机制就是 Event Loop，这个等一等的事件，就被放在一个被称为事件（回调）队列的数据结构中。**

于是上面的代码，实际在运行的时候，从事件队列的角度看，是这样的：



工作线程不断地从整个事件队列的右侧取得新的事件来处理执行，而新的事件只会从左侧放入：

主代码最先被执行，从上往下顺序执行，因此顺序是：

先打印 1；

在遇到第一个 `setTimeout` 的时候，告知浏览器，请在 0 秒之后往事件队列内放入执行打印 2 的事件；

在遇到第二个 `setTimeout` 的时候，告知浏览器，请在 5 秒之后往事件队列内放入执行打印 3 的事件；

再打印 4；

主代码执行完毕，Chrome 的控制台打印这段代码的返回值，但因为它没有返回值，于是就打印 `undefined`。

浏览器老老实实在地按照要求放入了打印 2 的事件，虽然是第 0 秒就放入，但是因为放入的时候主代码还在执行，因此只能等待，它等到主代码执行完毕后才得到执行，打印了 2。

5 秒钟后，浏览器按照要求往队列里放入了打印 3 的事件，于是 3 被打印出来。

你看，通过这种方式，JavaScript 可以让不同的任务在一个线程中完成，而整个任务编排的机制，**从代码的角度看，所有的逻辑都是通过七七八八的“异步回调”来完成的；而从程序员思维方式的角**度看，以往基于线程的编程，变成了事件驱动的编程。

thread-based		event-driven
monitor	~	event handler
scheduling	~	event loop
exported functions	~	event types accepted by event handler
returning from a procedure	~	dispatching a reply
executing a blocking procedure call	~	dispatching a message, awaiting a reply
waiting on condition variables	~	awaiting messages

上图来自 [The Case of Threads vs. Events](#)，很好地对比了两者的不同之处，其中：

对于逻辑的触发，基于线程编程需要不断地由监视线程去查询被监视线程的某一个状态，如果状态满足某个条件，则触发相应的逻辑；而事件驱动则通过事件处理器，在事件发生时执行挂载的回调逻辑。不知你是否联想起了 [\[第 03 讲\]](#) 中我介绍的 push 和 pull，在这里，前者正类似于 pull 的形式，而后者则类似于 push 的形式。

基于线程的方式可以阻塞线程，等待时间或某个条件满足后再继续执行；而事件驱动则相反，发送一条消息后无阻塞等待回调的发生。阻塞线程的方式对资源的消耗往往更加显著，因为无论是否执行线程都被占用，但从人直观理解的角度来说，代码更直白，更符合人对顺序执行的理解；而事件驱动的方式则相反，资源消耗上更优，但是代码的执行顺序，以及由此产生的系统状态判断变得难以预知。

请注意的是，在 JavaScript 中我们通常无法使用基于线程的编程，但是在很多情况下，例如 Java 和 Python 这些传统的后端编程语言中，我们可以混合使用基于线程和事件驱动的编程，它们是互不矛盾的。

最后，为什么 JavaScript 要被设计成单线程的，多线程难道就不行吗？最重要的原因，就是为了让整个模型简单。如果引入多线程，这里有很多问题需要解决，例如事件处理的依赖关系（多线程的事件处理就不再是简单队列的挨个处理了），例如资源的共享和修改（无锁编程不再有效，必须要考虑同步等加锁机制了），整个系统会变得极其复杂，不只是对于浏览器的开发者而言，对前端的开发者也一样。

另外，需要说明的是，浏览器的 JavaScript 执行是单线程的，但不代表浏览器是单线程的。浏览器通常还包含其它线程，比如说：

界面（GUI）渲染线程，这个线程的执行和上述的 JavaScript 工作线程是互斥的，即二者不可同时执行；

事件触发线程，这个也很好理解，我们介绍过有一个神秘人物帮着往队列中放入事件（例子中的回调打印 2 和回调打印 3），这个神秘人物就是事件触发线程。

2. 学写声明式代码

习惯于设计和书写大量的声明式代码，也是一个很重要的思维转变。

我们在 [\[第 07 讲\]](#) 中讲过什么是声明式代码，为什么我们在写视图层的时候会大量使用声明式代码。HTML、CSS 和 JavaScript，前端的三驾马车，两架是用声明式代码写的，我们应当记得自己做的是前端开发，而不是一个单纯的 JavaScript 写手。

声明式代码和命令式代码一样，都需要设计，且都需要测试。我见过不少工程师能够写出优秀的命令式代码，甚至已经习惯了，但是在写声明式代码的时候，却缺乏条理。

举例来说，设计页面的时候，要先设计布局，抓住整棵 DOM 树中核心的部分，自上而下地去划分区域，哪些是静态的区域，哪些是动态生成的，并合理设计可重用组件。再比如说，使用声明式代码处理模板中呈现数据的格式转换，使得呈现部分的代码更纯粹、自然，具体请参看 [\[第 16 讲\]](#) 中的过滤器。

3. 培养交互思维

前端工程师必须具备敏感的交互思维。通常来说，前端的代码，兼具着“甲方”和“乙方”的角色：

对用户和前端的交互来说，客户是甲方，享受服务；前端就是乙方，提供服务。

对和服务端的交互来说，前端就是甲方，从服务端获得数据和服务；服务端就是乙方，提供数据和服务。

而无论是和用户，还是和服务端的交互，都是学习前端技术中需要领会的部分。**和用户的交互要求开发前端的程序员具备产品思维，而和服务端的交互则要求开发前端的程序员具备工程思维。**

一头是用户，另一头是后端工程师，前端的开发人员，在整个庞大的研发体系中，既像粘合剂，又像润滑剂，要从产品和工程两个视角去思考问题，作出判断；不但要交付实实在在的

功能，要引导好的工程架构，还要给用户带来优秀的产品体验。

总结思考

今天我们首先强调了对于基于 Web 的全栈学习来说，学习前端技术的重要性，接着我们介绍了前端思维的几个转变，特别是事件驱动编程。希望你已经了解了 JavaScript 的单线程运行机制，并能够慢慢习惯不断在代码中与异步和回调打交道。

下面留两个思考问题：

你在技术团队中主要扮演什么角色，你对前端技术的认识是怎样的？

为什么 JavaScript 中，没有像 Java 或 Python 一样的 sleep 方法？毕竟，我就是想让当前执行过程稍等一下，再继续后面的逻辑，有 sleep 的话多方便啊。

最后，我想说的是，以前有句话叫做，“狗拿耗子，多管闲事”，但是我们在学习前端技术的时候，却要反过来，我们不但要多管闲事，还要“越管越多”，要多去想想类似的后端技术是怎样实现的。那在学习后端技术的时候，道理也是一样的，也要联想。

无论是早些年的 [GWT](#)，后端 Java 程序员写出了优秀的基于 Ajax 的跨浏览器应用；还是这些年的 [Node.js](#)，利用强大的 V8 引擎把数不清的 JavaScript 异步回调也写到后端去.....技术是没有边界的，前端和后端的技术当然也包括在内。

好，今天的内容就到这里，欢迎你和我讨论，也欢迎你邀请你的朋友一起阅读、学习。

扩展阅读

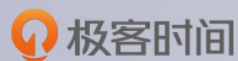
你可能听过这样一句话，“任何能用 JavaScript 写的应用，最终都会用 JavaScript 来实现。”这句话最初来自 [The Principle of Least Power](#) 这篇文章，写于 2007 年。

有位工程师做了一个[名为 Loupe 的网站](#)，用动画来形象地展示事件循环的过程，本文的例子也可以在它上面运行。

为什么浏览器中 JavaScript 代码的执行设计成单线程的，还有一个文中没有提到的原因，就是多线程的 GUI 特别容易死锁。这篇文章 [Multithreaded toolkits: A failed dream?](#) 描述了其中的缘由，大致是说 GUI 的行为大多都是从更高层的抽象一层一层往下调用到更低层的抽象、具体工具类实现，再到操作系统；而事件则是反过来，从下往上冒泡。结果就是两个方向相反的行为在碰头，给资源加锁的时候一个正序，一个逆序，极

其容易出现互相等待而饿死的情况，而这种情况下要彻底解决这一问题的难度无异于“逆转潮汐”。

[浏览器的工作原理：新式网络浏览器幕后揭秘](#)，这可能是在互联网上流传最广泛地介绍浏览器工作原理的中文文章，非常推荐。



全栈工程师修炼指南

从全栈入门到技能实战

熊燚

Oracle 首席软件工程师



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 13 | 特别放送：选择比努力更重要

下一篇 15 | 重剑无锋，大巧不工：JavaScript面向对象

精选留言 (6)

写留言



Carson

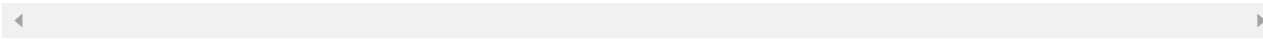
2019-10-11



Javascript 是单线程+事件驱动，单线程就是主线程。Javascript 必须让主线程一直处于运行当中，才能实现事件驱动。它无法通过简单的 sleep 主线程来达到 sleep 后继续执行某件事目的。

以上是我的理解，不确定是否有误。

展开 ▾

作者回复: 



  3



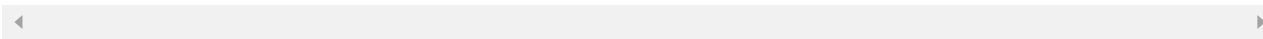
不记年



2019-10-12

思考题的第二个，我觉得应该是因为js是单线程的，如果设置sleep的话，就是阻塞唯一的线程，而又没有一个监视线程去pull。而且按照事件驱动编程的逻辑，如果想sleep当前正常执行的事件的话，应该也使用事件回调的方式，这样就只阻塞事件而不是线程。

展开 ▾

作者回复: 





不记年

2019-10-12

我所理解的js 执行过程分为两部分，第一部分是做一些初始化的工作，初始化工作包括初始化一些参数之类的，以及告知浏览器事件以及触发的逻辑。第二部分是事件循环，不断地往事件队列中抓取事件执行。在这整个过程中。

展开 ▾



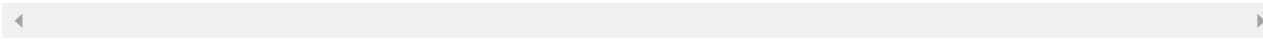
joker



2019-10-11

他是执行哪段代码出的underfined

展开 ▾

作者回复: `console.log("4");`



 1 



tt

2019-10-11

我一直从事后端开发，但是经过了几个Web项目后，对前端的认识发生了转变，真的就像老师文中所说的那样。这个感触在直接面向用户时尤其明显，感觉连原有的部门开发流程

都不适应了。

至于为什么没有sleep，是不是因为sleep里指定的时间只对事件分发线程有效，而真正...
展开 ∨

作者回复: 第二问不太正确，你可以联系文中介绍的 event loop 再想想。



許敲敲

2019-10-11

Web developer ,工作中就是开发公司的组件库，客户报defect 去fix 。想说前端水好深，
WebGL ,tensorflow.js ,webassembly 这些技术都好强悍

展开 ∨

