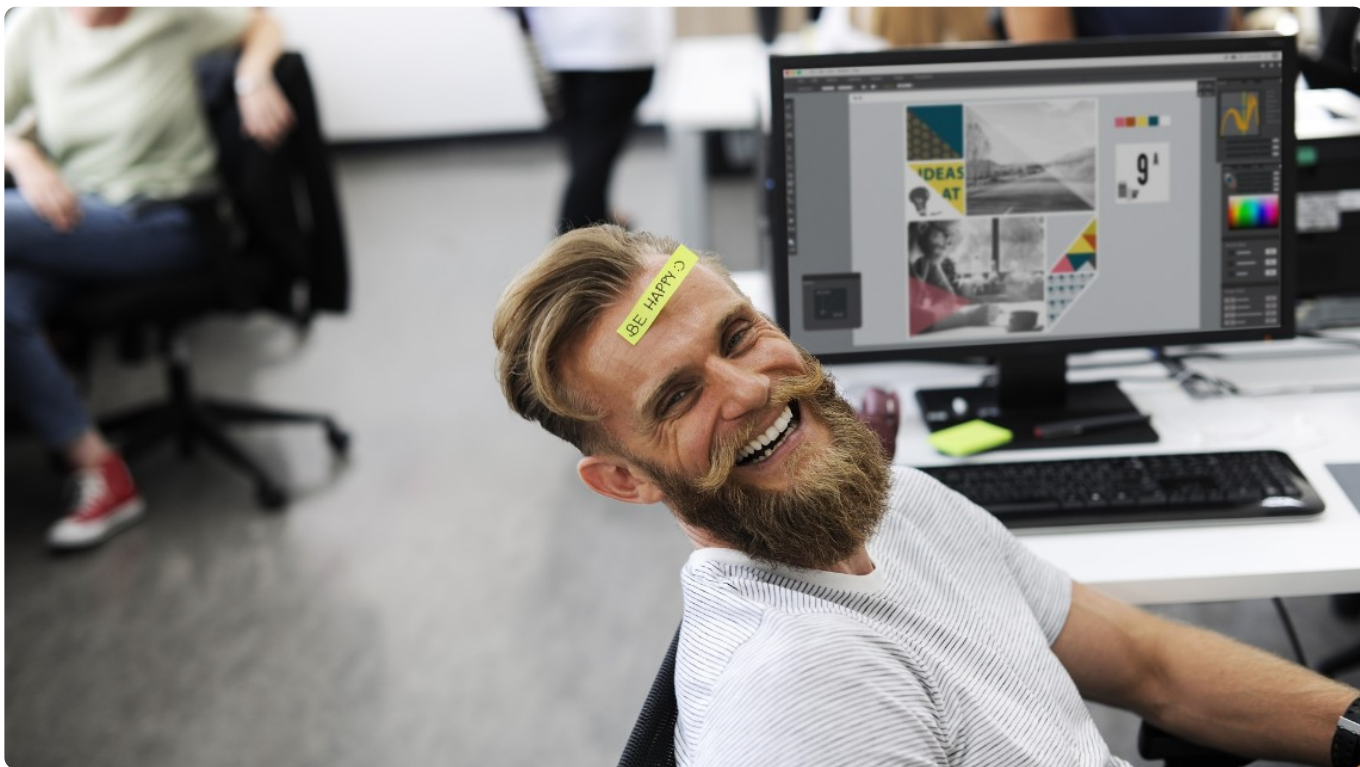


## 11 | 研发环境：Facebook怎样让开发人员不再操心环境？

2019-09-16 葛俊

研发效率破局之道

[进入课程 >](#)



讲述：葛俊

时长 17:57 大小 16.44M



你好，我是葛俊。今天，我来和你聊聊研发环境的问题，也就是如何才能让开发人员少操心环境，甚至不操心环境。从这篇文章开始，我们就一起进入工程方法模块了。

在[第 5 篇文章](#)关于“持续开发”的讨论中，我与你介绍了获取开发环境是开发工作的核心步骤之一，对提高研发效能来说是非常值得优化的环节。当时，我与你提到了开发环境服务系统以及沙盒环境，解决了开发环境的准备工作。

而这里的“开发环境”只是研发环境的一部分，特指开发机器环境，包括开发机器的获取、网络配置、基本工具以及代码的获取和配置。今天，我们就来看看整体的研发环境的配置问题，从全局上切实解决开发人员因为操心环境而导致的效能低下。

在此之前，你可以先回忆下是否对以下问题似曾相识呢？

开发人员使用的电脑配置太低，运行太慢，编译一次要 10 分钟；

测试环境不够，上线时熬夜排队等环境；

工具零散，不成系统，很多步骤需要手动操作，开发思路常常因为流程不连贯而被打断；

团队成员的环境设置参差不齐，有个别开发人员环境配置得比较好，效率比较高，但没有固化下来，其他团队成员也用不上。

这些问题，实际上都可以归结为研发环境不够高效。就像低劣的空气质量 and 食物质量会影响我们的身体健康一样，不理想的研发环境会严重降低研发效能。

具体来说，按照开发流程，软件研发需要以下几种环境：

开发机器；

IDE；

开发过程中使用的各种工具、数据和配置；

本地环境、联调环境；

测试环境、类生产环境。

接下来，我会按照这个顺序，分别与你介绍这 5 套环境如何配置。

因为 Facebook 在环境配置上做得非常好，比如说，他们可以保证开发人员在 5 分钟之内拿到一套干净的开发机器环境。所以，我会与你着重分享 Facebook 的实践，然后根据这些实践，总结出一套提供高效研发环境的原则。同时，我也会与你介绍一些我在其他公司的落地实践，帮助你掌握与环境配置相关的内容。

## 如何配置高效的研发环境？

### 开发机器

Facebook 从不吝啬在开发机器上的投资。每个开发人员除了有一台笔记本外，还在远程数据中心有一台开发机器。数据中心的机器用于后端以及网站的开发，为方便描述，我称之为后端开发机。而笔记本有两个用处：一是用来做移动端 App 的开发；二是作为终端，连接到后端开发机做后端和网站开发。

两台机器的配置都非常强劲。后端开发机一开始是实体机器，后来为了便于管理和提高资源利用率，逐步转为了虚拟机，但不变的是配置依然强大。在 2015 年的时候，绝大部分机器配置都能达到 16 核 CPU、144G 内存。笔记本有两种选择，一种是苹果 MacBook Pro，另一种是联想 ThinkPad，都是当时市场上顶配或者顶配下一级的配置。

开发机器的获取和释放，则是通过共享机器池以服务化自助化的方式完成的。更多细节，你可以再复习下[第 5 篇文章](#)的相关内容。

## IDE

IDE，指的是集成开发环境。Facebook 持续在 IDE 上投入，以完善其功能、一致性及一体化体验。接下来，我会以后端和网站开发为例来说明。

因为代码存放并且运行在后端开发机上，所以在 2012 年以前，绝大部分开发人员都是使用 SSH 远程登录到后端开发机，在那里使用 VIM/Emacs 这类能在命令行里运行的编辑器进行开发工作。也有人尝试在笔记本上运行 GUI 的 IDE，远程挂载后端开发机上的文件系统，但总是有卡顿，效果不好。

在这种情况下，公司首先采用的办法是，尽量提高命令行编辑器的体验，把 VIM/Emacs 配置成类似 IDE 的工具，将最常见的功能，比如代码搜索、代码跳转、Debugger 等集成进去，效果还算可以。

不过，与 GUI 形式的 IDE 相比，这种命令行的工作方式还是有一些局限性，比如，和其他工具服务集成不方便，需要记忆大量快捷键，在显示形式上不够丰富导致用户体验不够好，等等。所以，Facebook 持续投入研究 GUI 形式的 IDE。

第一个成型的 GUI IDE 是一个 Web IDE，也就是在数据中心运行一些 Web IDE 服务。这些 IDE 服务连接到开发者的后端开发机上获取代码，同时提供网页界面，供开发人员用浏览器登录，进行开发工作。这种专门的 IDE 服务解决了远程文件夹挂载的卡顿问题，同时跟其他服务集成起来也很方便。

比如，它可以与 Phabricator 的代码审查功能集成在一起，在代码中用 inline 的方式显示其他人对你的代码的 comments，也就是说，在两行代码之间显示一个额外区域，用于展示 comments。类似的，它还可以方便地与代码搜索服务、代码跳转服务、Debugger 集成，甚至可以跟 CI/CD 工具链的发布工具集成，所以非常方便。

到 2014 年的时候，Facebook 的大部分开发人员都逐渐转移到这个 Web IDE 上去了。如果你想深入了解 Web IDE，可以参考下[Apache Che](#)项目。

这个 Web IDE 已经很方便了，但基于网页的 IDE 在易用性和安全性还是有一些局限，所以 Facebook 继续加大在 GUI 形式的 IDE 方面的投入，最后使用 Electron 框架实现了一个原生的 IDE，也就是[Nuclide](#)。

Nuclide 的工作原理和 Web IDE 基本一致，都是在数据中心运行 IDE 服务，IDE 的前端则运行在本地笔记本上，通过与 IDE 服务联通实现代码编辑等功能。只不过 Nuclide 的前端，是运行在开发者笔记本上的一个原生应用，而 Web IDE 的前端是运行在笔记本上的网页浏览器而已。Nuclide 的功能比 Web IDE 更强大，易用性也更好，同时因为没有浏览器的依赖，安全性也更好一些。

## 本地环境、联调环境

关于 Facebook 的本地环境，我已经在[第 5 篇文章](#)中介绍过了，主要就是加快本地构建，使用生产环境的数据，从而使得本地环境更加快捷、方便。

而至于联调环境，我曾在[第 6 篇文章](#)中简单提过。在代码提交到 Phabricator 上进行审查的时候，Phabricator 会调用一个沙盒系统创造出一个沙盒环境，运行正在被审查的代码。这个系统也是用机器池实现的，同时也是一个自助式服务，也就是说开发人员可以不通过 Phabricator，直接调用 API 来生成沙盒环境。

本地环境和联调环境是开发中最高频使用的环境，对持续开发很重要。接下来，我再与你分享**两个我在其他公司的实施案例**吧。

**第一个例子是，我在 Stand 公司搭建本地环境。**Stand 的业务规模小，因此并没有使用微服务，主要的服务只有单体的网站后端服务、数据库服务 MySQL、缓存服务 Redis，以及一些数据监控服务，相对来说比较简单。

我们的做法是，把这些依赖服务尽量在本地开发机器（也就是笔记本）上都运行一个实例。实在不能在本地运行的服务，要么在本地环境运行时不调用它，要么就在调用它的时候传递额外的参数，表明这个调用来自开发环境，而被调用的服务则针对这样的调用进行特殊处理，从而达到不污染线上环境的效果。

这是一个很常见的办法，简单有效。不过，它的缺点是本地环境数据跟线上环境有区别。

如果你的系统采用的是微服务，则可以采用以下 3 种常见办法。

第一种方法是，直接在自己的机器上把所有的依赖服务都跑起来。这种方法的优点是方便，但要求开发机器配置要求高。

第二种方法是，团队维护一个环境，让大家的开发环境接入。也就是，开发者自己的机器上只运行自己开发的服务，调用其他服务时就使用这个共享环境中运行的服务实例。这种方式对开发机器要求不高，但需要团队维护一个环境，并且一般来说大家需要经常更新这个环境中的服务，整个环境容易不稳定。

第三种方法是，使用服务虚拟化工具，来模拟依赖的服务，比如[Mountbank](#)、[WireMock](#)。你可以参考[这篇文章](#)来了解 WireMock 的使用方法。

**第二个例子是，我为一个云产品团队提供联调环境。**这个云产品结构非常复杂，有十多个服务，至少需要 3 台服务器；不但有软件，还有数据、组网等复杂的设置，部署很困难；更严重的问题是，这个环境一旦损坏就很难修复，需要从头再来，所以开发人员自己配置基本不可能，运维人员也是忙于维护，应接不暇。

针对这个问题，我们也是使用了机器池的办法。不过，这个机器池里面的单元不再是单个机器，而是由 3 台机器组成的服务。这个服务自助化提供给开发者使用，使用之后自动回收销毁。

同时，确保机器池中有两套空闲环境。不够就补充，多了就删除，以保证获取环境时可以马上得到，同时又不会因为太多空闲环境而造成资源浪费。

另外，每次有了新的稳定版本，运维人员都会更新脚本，并重新安装和配置系统，保证开发人员能够在稳定版本上进行联调。

这样一来，就解决了团队开发人员的环境问题，将获取环境的时间由 2~4 个小时缩短到了几分钟。

## 开发过程中使用的各种工具、数据和配置

除了 IDE，开发过程中还会用到其他工具，比如代码搜索、发布部署，以及日志查看工具等。这些工具的组合，可能与你平时理解的开发环境不大一样，但事实上它也是一种广义的开发环境，对开发效率影响很大。



这部分环境的优化，主要是使用工具之间的网状互联来提高效率。关于工具的网状互联，我在[第 4 篇文章](#)中有介绍。这里，我想强调 Facebook 在的另一个重要实践：**重视开发体验，将开发流程中常用步骤的自动化做到极致。**

我用一个具体的例子来说明。我们都知道，Git 的 Commit Message（代码提交描述）是提供信息的重要渠道。但它有一个局限，就是只能存储文本，而图片在描述问题时常常比文字有效得多，也就是“A picture is worth a thousand words”，翻译为中文就是我们常说的“一图胜千言”。

为解决这个问题，Facebook 采用了以下方式：

1. 提供了一个图片存储服务，并为上传的图片提供永久 URL；
2. 开发了一个端测的截屏工具，截屏之后自动上传到图片存储服务，而且，在上传成功之后自动把图片 URL 保存到本地笔记本的系统剪贴板中；
3. 提供了一个内部使用的 URL 缩短工具，避免 URL 太长占用太多文字空间。

这三个工具集成起来，一个具体的使用场景是这样的：我在写 Commit Message 的时候，如果要截屏描述修改效果，就使用一个快捷键（比如 Cmd+Alt+4）激活截屏工具，随后拖动鼠标截屏，然后使用 Cmd+v 就可以直接把图片的 URL 粘贴到 Commit Message 里面了。

这个 workflow 非常顺畅、高效，不仅被大量用于 Commit Message 的书写中，也被经常被用在聊天工具中。后来，我到了其他公司，都会先配置这样一套工具流程。

## 测试环境、类生产环境

在测试环境、类生产环境的管理上，Facebook 使用一个叫作[TupperWare](#)的内部系统，以 IaC 的方式进行管理。不过，Facebook 并没有开源这个系统。如果你所在公司使用的是 Docker，那可以使用 Kubernetes 实现类似的功能。

如果你们没有使用 Docker，可以试试 HashiCorp 公司的[Terraform](#)，或者使用[Ansible](#)、[Chef](#)之类的配置管理工具，来产生一套干净的环境供团队成员使用，之后再销毁，既方便又不浪费资源。

这里，我再与你分享一个我在**Stand 公司使用 AWS 管理压测环境的例子**。当时我们没有使用 Docker，于是，我们选择了 AWS 的 OpsWorks 框架。它是 AWS 基于 Chef-Solo

开发的一个应用程序管理解决方案，同时支持基础设施的建模和管理。

OpsWorks 这个框架的用法也是声明式的，只不过这个声明不是纯代码，而是在 AWS 的网页上配置的，使用效果和 TupperWare 差不多，就是**首先**定义一个压测环境需要几台机器，需要运行什么操作系统、需要什么负载均衡器、需要什么数据库等。


**然后**，通过 OpsWorks 上暴露的钩子，使用代码来管理应用的生命周期，从而实现系统和应用的初始化。通过这种方式，我们可以很方便地使用 OpsWorks 产生一个云主机集群用于压测，结束之后马上删除，方便而且划算。

其实，使用 Ansible 或者 Chef 也可以实现类似功能，但需要自己开发些东西，这里我就不详细讨论了。

## 提供高效研发环境的原则

通过以上实践可以看出，配置高效的研发环境主要包括以下几条原则：

1. 舍得投入资源，用资源换取开发人员时间。Facebook 之所以从不吝啬在开发机器硬件上的投入，是因为人力成本更高。
2. 对环境的获取进行服务化、自助化。这一点可以在开发机器、联调环境的获取上，得到很好的体现。同时常常使用 IaC、配置管理系统（比如 Chef）和机器池的方法来实现，同时利用弹性伸缩来节约资源。
3. 注重环境的一体化、一致性，也就是要把团队的最佳实践固化下来。比如 Facebook 一个常见的操作是，配置文件统一处理。以 VIM 为例，将 VIM 的配置文件存放到网络共享文件夹中，开发人员只要在自己的.bashrc 文件中加上一行

 复制代码

```
1 source /home/devtools/vimconfig
```

就可以搞定。

## 小结

今天，我首先按照开发流程，也就是开发机器、IDE、本地环境和联调环境、开发过程中使用的各种工具及配置，以及测试环境和类生产环境的顺序，与你讲述了高效研发环境的具体

实践。然后，基于这些实践，总结了 3 个基本原则：一是，用资源换时间；二是，服务化、自助化环境的获取；三是，实现环境的一体化、一致性。

我觉得，这些原则和实践的背后有一个重要思路，就是 Facebook 重视环境，并持续优化环境。这一点在 IDE 的演化上尤为明显，从命令行到 Web IDE 再到 Nuclide，一直在进步。另外，在去年年底，Facebook 停止了对 Nuclide 的开源项目的维护，这也意味着后面他们可能还会有对 IDE 的一轮新的优化。

以上种种做法，使得我在 Facebook 做开发的时候，对研发环境的感觉就是不用操心，需要使用的时候直接到网站上申请就可以使用；配置也方便，团队的配置都已经在那里了；同时，环境中的各种工具、流程都很顺畅，让我能够静下心来做开发、写算法，做我最能提供价值的事情。

## 思考题

我在“开发过程中使用的各种工具、数据和配置”这一章节中提到的截屏工具流程，你觉得价值大吗？值得引入你所在的公司吗？如果值得的话，可以怎么来实现？

感谢你的收听，欢迎你在评论区给我留言分享你的观点，也欢迎你把这篇文章分享给更多的朋友一起阅读。我们下期再见！

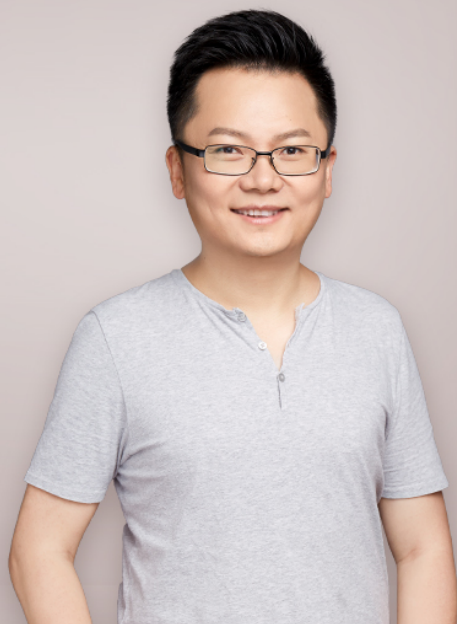


# 研发效率破局之道

Facebook 研发效率工作法

葛俊

前 Facebook 内部工具团队 Tech Lead



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。



上一篇 10 | 答疑篇：反对996并不是反对奋斗

## 精选留言 (2)

写留言



小名叫大明

2019-09-16

我看到了个人认为比较重要信息: Facebook认为人力成本更高，这个意识比较重要，很多公司没这个意识，有这个意识的确是通过延长工作时长来补，加班及长期加班又降低了非工具和环境造成编码效率，恶性循环。

慢慢理解招聘时多研发自驱，兴趣等能力的重视的原因了。哈哈

展开 ∨



1



Weining Cao

2019-09-16

我们现在遇到的一个问题是线上测试资源不够。因为我们发布的软件要支持mac系统，可是目前没有找到合适的提供mac虚拟机的服务商，可是买mac硬件又很贵。。。所以现在大部分mac环境提测的job都会长时间排队。。。

展开 ∨

