05 | 代码入库前：Facebook如何让开发人员聚焦于开发?

2019-09-02 葛俊

研发效率破局之道

进入课程 >

讲述：葛俊

时长 15:12　大小 13.94M

你好，我是葛俊。今天，我将与你分享优化流程中，代码入库前的开发流程。

代码入库之前的开发活动，主要包括编码、调测调优、静态检查、自动化测试、代码审查等。这是开发者编写代码的步骤，自然是提高研发效能的关键环节。

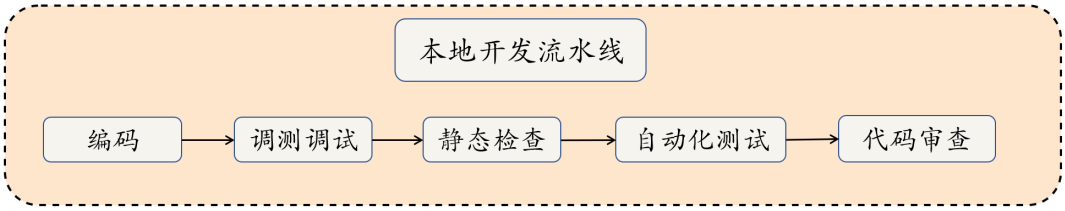

图 1 本地开发流水线

提高开发者编写代码的效能，关键在于让开发者不受阻塞、不受不必要的干扰，从而全身心地聚焦在产品开发上。我把这种不受阻塞的开发状态叫作**持续开发**。

一个团队如果能够做到持续开发，那么它的有效产出自然会很好。而对于个人开发者而言，持续开发能够帮助我们把精力集中在技术本身，对技术和个人能力的提升都大有裨益，所以是一种很好的开发体验。

在我看来，持续开发的基本原则主要包括两条：

1. 规范化、自动化核心步骤；
2. 快速反馈，增量开发。

接下来，我们就一起看看这两条核心原则吧。

规范化、自动化核心步骤

要让开发者聚焦于开发，就必须把研发流程中可以自动化的步骤尽量自动化。因为一般不可能完成所有步骤的自动化，所以我推荐的方式是：分析关键路径上的活动，以及耗时较长的活动，然后投入精力优化这些步骤。

首先，我们需要明确具体的开发步骤有哪些。我将其归纳为以下三大步：

1. 获取开发环境，包括获取开发机器、配置环境、获取代码等。
2. 在本地开发机器上进行开发，包括本地的编码、调测、单元测试等。
3. 代码入库前，把改动提交到检查中心（比如 Gerrit），再进行一轮系统检查，主要包括代码检查、单元测试、代码审查等，通过之后再入库。

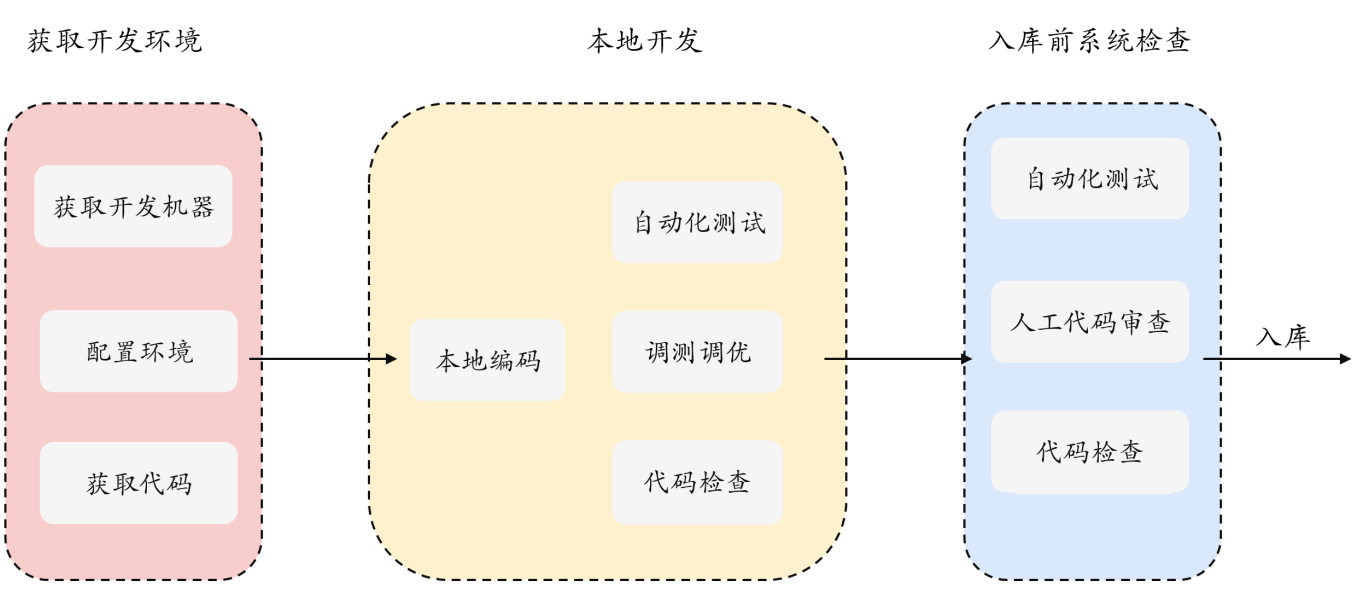

图 2 代码入库前的 3 大开发步骤

针对这三大步骤，我们可以有以下 3 个工程实践：

提高开发环境的获取效率；

规范化、自动化本地检查；

建设并自动化代码入库前的检查流程。

接下来，我们分别看看这 3 个工程实践如何落地。

提高开发环境的获取效率

开发环境的设置，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。这些操作的频率不是特别高，但如果步骤复杂、耗时长，就会对新成员加入、成员切换团队或者项目，产生比较大的影响。所以，开发环境的获取效率，通常是值得优化的。

有一个可以采用的优化方式是，**把整个开发环境的获取，进行服务化、自助化**。也就是说，开发者可以自助地申请获取环境，不需要 IT 部门的人员介入，从而既节省了开发者的时间，又降低了 IT 部门的人力成本。

比如，我之前在 Facebook 工作的时候，采用虚拟机作为个人开发机。内部工具团队开发了一个基于共享机器池的开发环境服务系统，让开发者可以在网页上申请和释放机器。机器返还之后，开发环境服务系统会自动对它进行清理，配置之后再重新放回机器池中。这就使得开发者可以在 5 分钟之内拿到一套干净的环境。

而至于开发机器上的代码，这个服务系统可以克隆获取团队常用的代码仓，并定时拉取最新的代码。这就使得开发者拿到一台机器之后，只需要再额外拉取很少的代码就可以进行开发。

上面这种方法定制性很强，Facebook 并没有开源。如果你准备进行这方面尝试的话，在机器的生成和配置方面，我推荐两种方式。

第一种方式，借助基础设施即代码（Infrastructure As Code，IaC）系统。比如，HashiCorp 公司的Terraform 工具。它支持声明式的方式快速产生自定义配置的机器，并在上面运行脚本进行配置。TerraForm 使用插件机制支持许多底层平台，比如 AWS、阿里云或者本地系统。

这种方式的优点是使用方便、功能强大，但前期投入大。

第二种方式是，提供机器镜像和配置脚本。通过镜像让每一台新机器拥有最基本的设置，比如 CPU、操作系统、基本软件，然后通过脚本实现基本配置，比如网络设置、软件更新等。这种方式的优点就是，前期投入小。我在 Stand 时，就使用了这种方法，效果不错。不过它的缺点就是不够灵活。

规范化、自动化化本地检查

本地检查是指，开发者在开发机器上进行的验证，比如语法检查、规范检查、单元测试、沙盒搭建等。我推荐的方式是，**根据团队实际情况，找到合适的工具和配置进行这些检查，并让团队成员统一使用。**

在这个方面，Facebook 的方法是，把很多工具都放到一个网盘上，挂载到每台开发机器的 Linux 文件系统上，让开发者们不用安装就可以直接使用。

挂载共享网盘的方法非常方便，因为用户不用操心工具的升级。但如果你们的系统没有这样的网盘的话，也可以通过脚本让开发人员一键安装工具和完成配置，效果也不错。缺点就是软件更新比较麻烦，因为要通知用户手动更新或者设计自动更新机制。

至于检查中使用的工具，我们需要根据具体的语言和框架去选择。

建设并自动化代码入库前的检查流程

建设并自动化代码入库前的检查流程，是持续集成前的必要工作，也可以看作是持续集成的一部分。它对入库代码质量起到一个门禁作用，对提高质量用处很大。**我认为，除了人数非常少的初创公司以外，其他开发团队都应该进行这个配置。**

这个流程一般可以使用代码仓管理系统作为中心，直接使用或者通过钩子集成其他工具和系统来实现。比如，使用 GitLab 提供的 GitLab CI/CD 框架。基本方法是，在项目的根目录里创建一个.gitlab-ci.yml 文件，来描述你的检查环境设置和步骤。你可以点击这个链接查看具体的方法。

在 Facebook，这一步使用的是开源版 Phabricator 的一个内部 Fork。Phabricator 在工作流中使用单元测试和 Linter 的方法，你可以参考帮助文档。

以上内容就是持续开发的第一个原则，也就是规范化、自动化核心步骤。这个原则，可以帮助开发者尽量减少非开发工作的耗时，从而把更多的时间、精力投入到本职的开发工作中。接下来，我们再来看看持续开发的第二个原则，即提供快速反馈，促进增量开发，这样能及早暴露问题，从而保证将来的工作不会因为实现错误，或者方向调整而进行昂贵的修改。

提供快速反馈，促进增量开发

提供快速反馈，进行增量开发指的是，能够快速验证已经完成的开发工作，说白了就是边开发边验证。具体的工程实践主要包括以下 3 个：

灵活使用各种 Linter 和测试；

建设并优化沙盒环境；

使用实时检查工具。

接下来，我们分别看看这 3 个工程实践如何落地。

灵活使用各种 Linter 和测试

最常用的快速验证方法就是，提高运行静态检查和测试的方便性、灵活性。各种语言、框架都有自己的测试框架和 Linter，这里我就不再一一列举了。接下来，我会与你分享**两种通用的有效使用 Linter 和测试的方法。**

首先，用命令行的工具来封装各种检查。命令行工具特别适用于自动化，方便开发人员使用。比如，我们可以通过命令行脚本，来实现简单的工作流。

举一个具体的例子，我希望团队在开发中，在运行公司提供的统一检查之外，还可以运行一些适应团队自身特点的检查，每个开发人员也可以添加自己希望使用的检查。这样就可以通过一个 Shell 脚本，依次调用公司的、团队的、个人的检查来实现，很方便。

其次，以服务化的方式，把这些检查的能力提供出来。比如，Facebook 的基础平台团队提供了在云上运行单元测试的能力，并把这个能力通过服务的方式提供给开发者，以方便他们在自己的开发机器上调用。也就是说，开发者可以调用云上资源运行大量的测试而不占用本地资源，从而在运行测试的同时可以高效地进行开发工作。

建设并优化沙盒环境

沙盒也是一个高频使用的、提高质量的工具。开发者如果能够方便地在本地搭建沙盒进行验证，那么进行开发自测的频率和质量就会大大提高，进而提高产品质量。所以，我推荐你在沙盒环境的搭建上进行投入。

在沙盒环境搭建中，有两个常见的优化点：

本地构建。因为我们必须把改动构建成产品才能进行本地验证，而这个步骤通常耗时较长。我推荐的优化方法是，不要使用全量构建，尽量只进行最小范围的增量构建。

测试数据的产生。产生贴近生产环境的数据往往比较费劲，Facebook 的做法是，开发环境直接使用生产环境的数据，不过这个方法比较激进，使用的公司比较少。另一个常见方法是，进行生产数据的导出并脱敏，然后使用到沙盒环境中。

使用实时检验工具

快速提供检查反馈，做到极致就是开发者无需手动触发检查，工具就会自动探测到改动、自动运行检查。最常见的是，IDE 中的实时语法检查。我们可以花一些时间来配置 IDE。另外，有些工具可以自动监视文件系统的变化，文件有变化时自动重启服务。这对于开发者来说，非常便利。

举个例子。使用 Node.js 进行开发时，nodemon 就是不可或缺的工具，你只要在原来的命令前，加上 nodemon 就可以使用。比如，启动服务的语句是./bin/www，使用

nodemon 的形式就是 nodemon ./bin/www。这样运行服务之后，如果你的文件有修改，nodemon 就会自动重新运行。

你可以在下面这个动图中看到，在第一次保存时，有语法错误，nondemon 重新启动失败；第二次保存时修复了语法错误，nodemon 成功重启服务。通过 nodemon，我减少了两次手动重启服务的繁琐操作。

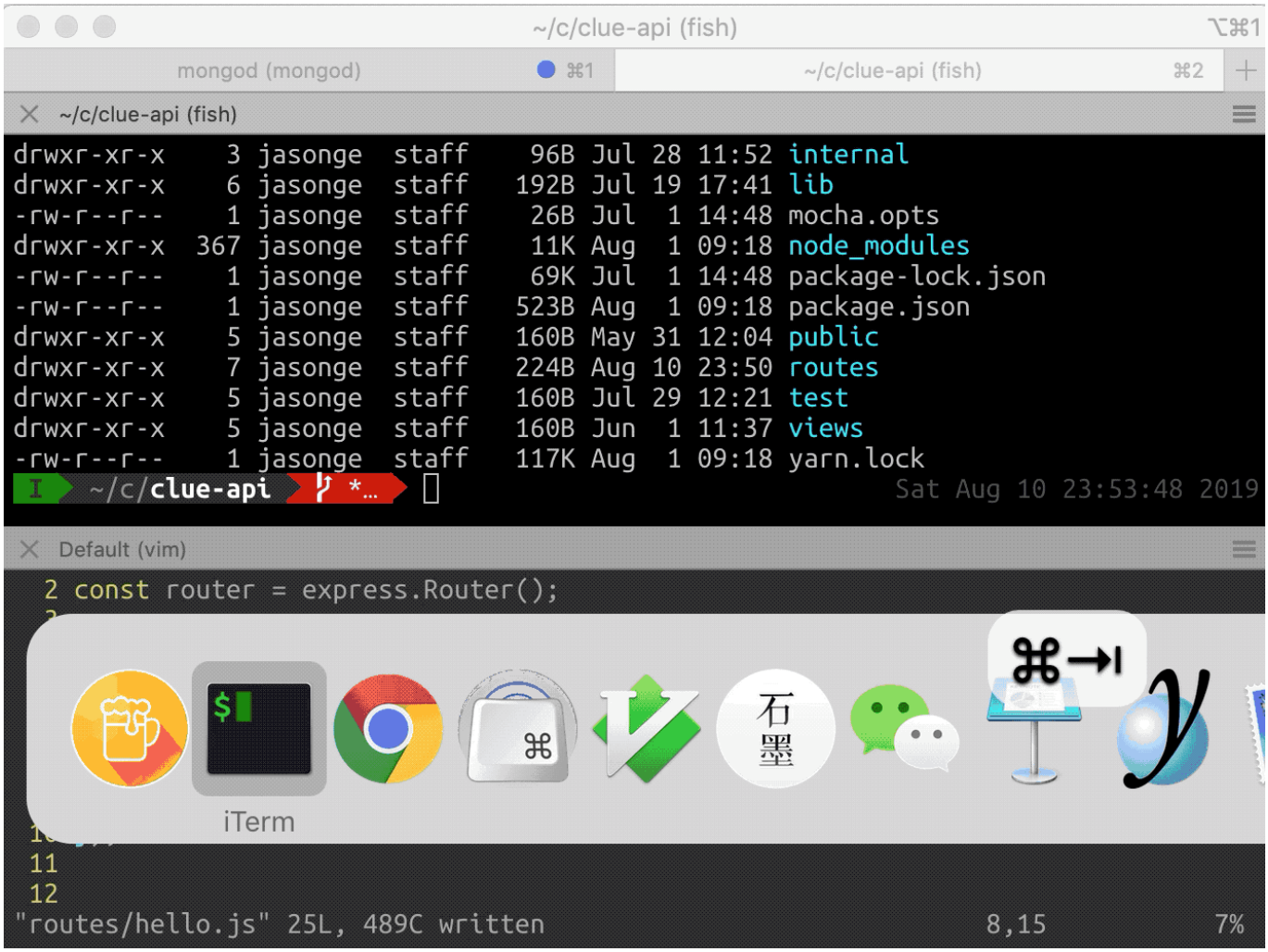

图 3 nodemon 使用示例

类似的工具，SpringBoot 有 Spring-Boot-Devtools，你可以点击这个链接查看详细描述。针对 Python，可以直接使用 nodemon，这里也有一个链接供你参考。

如果你使用的框架、语言没有直接可以使用的工具，帮助你进行实时重启服务的话，可以使用类似watchdog/watchmedo的工具来实现自动化。比如

复制代码

```
watchmedo shell-command \
```

```
  --patterns="*.py" \
  --command='python "${watch_src_path}"' \
  .
```

会监控所有的 Python 文件改动并自动重启。

提供快速反馈，边开发边验证，虽然只是一个简单的原则，但可以带来很多好处。最直接的收益就是，能够大大提高开发者对当前代码的信心，从而促进代码尽早入仓、尽早集成。

可能你也注意到了，**代码集成越晚发现问题就越晚。这正是产品上线的最后关头合并混乱，产品质量差、返工率高的一个重要原因**。所以，我建议在你的工作流程中，要尽量提高实时验证的能力。如果你这么做了，很快就会看到效果。

小结

在今天这篇文章中，我和你分享了两条持续开发的基本原则，来帮助开发者在代码入库前聚焦于开发工作：一是，规范化、自动化代码入库前的核心步骤；二是，提供快速反馈，帮助开发者边开发边验证，以促进增量开发。

我将今天的内容，总结为了一幅图，帮助你复习。

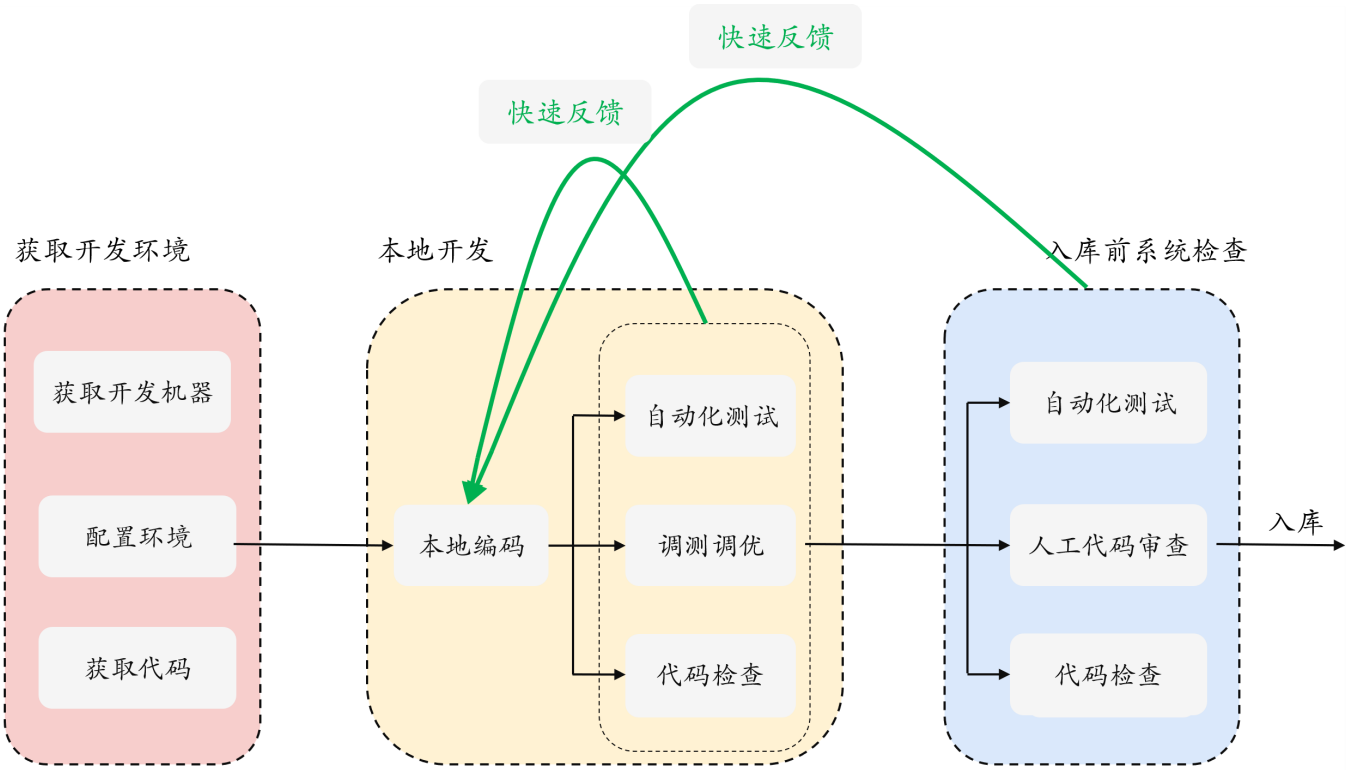

图 4 代码入库前流程优化

这些原则和实践，是我根据自己的经验总结出来的。如果能直接适用于你的团队当然最好了，但我更加希望的是，你能从这些原则和实践的讨论中，理解它们背后的思路，从而找到合适的方法和实践，去有优化代码入库前这个环节中，最需要优化的地方，让开发者能够真正聚焦于开发。

另外，我给你一个落地持续开发的小贴士：持续开发很适合用自上而下和自下而上相结合的方式来推动。因为开发者最了解自己工作的痛点，所以也能比较准确地找到需要优化的地方。在 Facebook，很多工具和流程都是由开发者自发开发或者引入，后来逐步推广至团队和公司使用的。

所以我推荐，**作为开发者，你可以自己抽一点时间提高自己的工作流程，自动化繁琐的工作；而作为管理者，你可以有意识地奖励这样的优化行为，并对适用于团队的部分进行推广。**

思考题

最后，我来给你留下两个思考题吧。

1. 在开发环境方面，你有没有尝试过在 Docker 里面进行开发？你觉得这种方式的好处是什么，弊端又是什么呢？
2. 有些开发者喜欢写好一个比较大的功能单元，然后再一口气调测。你觉得这样做的好处和坏处，各是什么呢？

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！

上一篇　04 | 流程优化：怎样才能让敏捷、精益真正为我所用?

精选留言 (3)

写留言

许童童
2019-09-02

老师，数据那一块怎么管理呢，比如初始数据

展开

Geek_1988
2019-09-02

如果工程比较大，编译需要很久的话，自动编译并重启服务是不是就不太合适了?

作者回复: 如果你能够做到很快时间自动编译完成并重启服务，那就很棒。当然这可能需要做比较多的优化，以及可能使用一些Hacky的办法。当然如果做不到，或者投入太大，那也就确实不合适了。

Marvin
2019-09-02

1、使用docker对前端或者硬件需求较苛刻的开发不是很友好，好处是开发环境搭建迅速零维护。2、较大功能单元，不利于单元测试，不利于后期维护，不利于工作拆解，不利于发现问题，好处是功能相对集中，持续开发时间较长。

展开

作者回复: 👍👍👍