14 | 如何做到构建的提速，再提速！

2018-08-04 王潇俊

14 | 如何做到构建的提速，再提速！
朗读人：王潇俊　10′58″ | 5.03M

在前面几篇文章中，我分享了很多关于构建的观点，然而天下武功唯为快不破，构建的速度对于用户持续交付的体验来说至关重要。

在实施持续交付的过程中，我们经常会遇到这样的情况：只是改了几行代码，却需要花费几分钟甚至几十分钟来构建。而这种情况，对于追求高效率的你我来说，是难以容忍的。

那么，今天我就带你一起看看，还有哪些手段可以帮助构建提速。

升级硬件资源

构建是一个非常耗时的操作，常常会成为影响持续交付速度的瓶颈。原因是，构建过程，会直接消耗计算资源，而且很多构建对硬件的要求也非常高。那么，升级硬件资源就是构建过程提速的最为直接有效的方式。

需要注意的是，这里的硬件资源包括 CPU、内存、磁盘、网络等等，具体升级哪一部分，需要具体情况具体分析。

比如，你要构建一个 C 语言程序，那么 CPU 就是关键点。你可以增加 CPU 的个数或者提升 CPU 主频以实现更快的编译速度。

再比如，你要用 Maven 构建一个 Java 应用，除了 CPU 之外，Maven 还会从中央仓库下载依赖写在本地磁盘。这时，网络和磁盘的 I/O 就可能成为瓶颈，你可以通过增加网络带宽提升网络吞吐，使用 SSD 代替机械硬盘增加磁盘 I/O ，从而到达提升整个构建过程速度的目的。

总之，当你使用成熟的构建工具进行构建时，如果无法通过一些软件技术手段提升软件本身的构建速度，那么根据构建特点，有针对性地升级硬件资源，是最简单粗暴的方法。

搭建私有仓库

构建很多时候是需要下载外部依赖的，而网络 I/O 通常会成为整个构建的瓶颈。尤其在当前网络环境下，从外网下载一些代码或者依赖的速度往往是瓶颈，所以在内网搭建各种各样的私有仓库就非常重要了。

目前，我们需要的依赖基本上都可以搭建一套私有仓库，比如：

- 使用 createrepo 搭建 CentOS 的 yum 仓库；
- 使用 Nexus 搭建 Java 的 Maven 仓库；
- 使用 cnpm 搭建 NodeJS 的 npm 仓库；
- 使用 pypiserver 搭建 Python 的 pip 仓库；
- 使用 GitLab 搭建代码仓库；
- 使用 Harbor 搭建 Docker 镜像仓库
- ……

除了提升构建时的下载速度外，更重要的是，你还可以用这些工具存储辛勤工作的成果，保护知识产权。

总之，搭建私有仓库一定物超所值。当然，维护和管理这一大批工具需要投入不少人力和经济成本，在公司 / 团队没有成一定规模的前提下，会有一定的负担。

所以，如果你的团队暂时没有条件自己搭建私有仓库的话，可以使用国内已有的一些私有仓库，来提升下载速度。当然，在选择私有仓库时，你要尽量挑选那些被广泛使用的仓库，避免安全隐患。

使用本地缓存

虽然搭建私有仓库可以解决代码或者依赖下载的问题，但是私有仓库不能滥用，还是要结合构建机器本地的磁盘缓存才能达到利益最大化。

如果每次依赖拉取都走一次网络下载，一方面网络下载的速度通常会比本地磁盘慢很多，另一方面在构建量很大时，并发请求会导致私有仓库出现网卡打爆或者出现莫名其妙的异常，从而导致

所有的构建过程变得不稳定，甚至影响其他工具的使用。

所以，妥善地用好本地缓存十分重要。这里说的“妥善”，主要包括以下两个方面：

1. 对于变化的内容，增量下载；
2. 对于不变的内容，不重复下载。

目前，很多工具都已经支持这两点了。

对于第一点，项目的源码是经常变化的内容，下载源码时，如果你使用 Git 进行增量下载，那么就不需要在每次构建时都重复拉取所有的代码。Jenkins 的 Git 插件，也默认使用这种方式。

对于第二点，Maven 每次下载依赖后都会在本地磁盘创建一份依赖的拷贝，在构建下载之前会先检查本地是否已经有依赖的拷贝，从而达到复用效果。并且，这个依赖的拷贝是公共的，也就是说每个项目都可以使用这个缓存，极大地提升了构建效率。

如果你使用 Docker，那么你可以在宿主机上 mount 同一个依赖拷贝目录到多个 Slave 容器上，这样多个容器就可以共享同一个依赖拷贝目录。你可以最大程度地利用这一优势，但要注意不要让宿主机的磁盘 I/O 达到瓶颈。

规范构建流程

程序员的祖训说：Less is More，Simple is Better，这与大道至简的含义不谋而合。

程序的追求是简约而不简单，但随着业务越来越复杂，构建过程中各种各样的需求也随之出现，虽然工具已经封装了很多实用的功能，但是很多情况下，你都需要加入一些自定义的个性化功能，才能满足业务需求。

在携程，Java 构建过程中就有大量的额外逻辑，比如 Enforcer 检查、框架依赖检查、Sonar 检查、单元测试、集成测试等等，可以说是无所不用其极地去保证构建产物的质量。

因此，当前复杂的构建过程再也回不到仅仅一条 mvn 或者 gcc 命令就能搞定的年代。而这一套复杂的流程下来必定会花费不少时间，让程序员们有更多喝茶和去厕所的时间。

追求高效的同时，又不舍弃这些功能，是一个现实而又矛盾的命题，我们能否做到二者兼顾呢？答案，当然肯定的。

以 Java 构建为例，Enforcer 检查、框架依赖检查、Sonar 检查、单元测试、集成测试这些步骤，并没有放在同一个构建过程中同步执行，而是通过异步的方式穿插在 CI/CD 当中，甚至可以在构建过程之外执行。

比如， Sonar 扫描在代码集成阶段执行，用户在 GitLab 上发起一个合并请求（Merge Request），这时只对变更的代码进行对比 Sonar 扫描，只要变更代码检查没有问题，那么就

可以保证合并之后主干分支的代码也是没问题的。

所以，用户发布时就无需再重复检查了，只要发布后更新远端 Sonar Qube 的数据即可，同时，这个过程完全不会影响用户的构建体验。

通过以上一些规范构建流程的做法，可以进一步提高构建速度。

善用构建工具

正如我前面所说的，目前很多构建工具已经具备了非常多的功能来帮助我们更好地进行构建，因此，充分理解并用好这些功能就成了我们必须要掌握的武林绝学。

以 Maven 为例，我来带你看看有哪些提速方式，当然其他的构建工具，如 Gradle 等也都可以采用类似的方法：

1. 设置合适的堆内存参数。 过小的堆内存参数，会使 Maven 增加 GC 次数，影响构建性能；过大的堆内存参数，不但浪费资源，而且同样会影响性能。因此，构建时，你需要反复试验，得到最优的参数。

2. 使用 -Dmaven.test.skip = true 跳过单元测试。 Maven 默认的编译命令是 mvn package，这个命令会自动执行单元测试，但是通常我们的构建机器无法为用户提供一套完整的单元测试环境，特别是在分布式架构下。因此如果单元测试需要服务依赖，则可以去掉它。

3. 在发布阶段，不使用 Snapshot 版本的依赖。 这就可以在 Maven 构建时不填写 -U 参数来强制更新依赖的检查，省下因为每次检查版本是否更新而浪费的时间。

4. 使用 -T 2C 命令进行并行构建。 在该模式下 ，Maven 能够智能分析项目模块之间的依赖关系，然后并行地构建那些相互间没有依赖关系的模块，从而充分利用计算机的多核 CPU 资源。

5. 局部构建。 如果你的项目里面有多个没有依赖关系的模块，那么你可以使用 -pl 命令指定某一个或几个模块去编译，而无需构建整个项目，加快构建速度。

6. 正确使用 clean 参数。 通常情况下，我们建议用户在构建时使用 clean 参数保证构建的正确性。clean 可以删除旧的构建产物，但其实我们大多数时间可能不需要这个参数，只有在某些情况下（比如，更改了类名，或者删除了一些类）才必须使用这个参数，所以，如果某次变更只是修改了一些方法，或者增加了一些类，那么就不需要强制执行 clean 了。

总之，如果你能熟练运用各种构建工具，那么你的效率一定会比其他人高，你的构建速度一定比其他人快。

总结

我介绍了五种常见的构建提速的方式，分别是：

1. 升级硬件资源，最直接和粗暴的提速方式；
2. 搭建私有仓库，避免从外网下载依赖；
3. 使用本地缓存，减少每次构建时依赖下载的消耗；
4. 规范构建流程，通过异步方式解决旁支流程的执行；
5. 善用构建工具，根据实际情况合理发挥的工具特性。

然而，每个公司持续交付的构建流程不太一样，面临的问题与挑战也都不太一样，所以在优化前，一定要先了解问题原因，再对症下药。

思考题

你所在公司的构建流程是什么样的？是否也面临性能的问题？你又是是如何解决这些问题的？

欢迎你给我留言。

通过留言可与作者互动