这节课我们将进入前端效率工程化的第三个模块——部署效率篇。本模块主要讨论两个方面的问题：第一个是在前端项目的构建部署流程里，除了使用构建工具执行构建之外，还有哪些因素会影响整个部署流程的工作效率？第二个是在部署系统中进行项目构建时，又会面临哪些和环境相关的问题和优化方案？

这节课我们先来讨论为什么要使用部署系统，而不是在本地环境下部署代码？

在分析这个问题之前，我们先对前端项目的部署流程进行界定。

前端项目的一般部署流程

在前端项目中，通常可以把在一个全新环境下的代码部署过程分为以下几个环节：

1. **获取代码**：从代码仓库获取项目代码，并切换到待部署的分支或版本。
2. **安装依赖**：安装项目构建所需要的依赖包。
3. **源码构建**：使用构建工具对项目源代码进行构建，生成产物代码。
4. **产物打包**：将部署所需的代码（通常指的是构建后的产物代码，如果是部署 Node 服务则还需要其他目录与文件）打成压缩包。
5. **推送代码**：将待部署的文件或压缩包推送至目标服务器的特定目录下，如果是推送压缩包的情况，还需执行解压。
6. **重启服务**：在部署 Node 服务的情况下，在代码推送后需要进行服务重启。

本地部署相比部署系统的优势

对于使用部署系统的项目而言，除了重启服务这一步骤在普通静态服务部署中不需要执行外，上述其他环节通常是每次构建都需要经历的。

而如果使用本地开发环境进行部署，则可以根据情况**对前两个环节进行简化**：

1. 在获取代码的环节中，本地开发环境已经包含了项目的本地代码，同拉取完整的代码仓库相比，直接获取更新内容并切换分支或版本的处理要更快一些。
2. 在安装依赖的环节中，本地开发环境通常已包含了构建所需的最新依赖包，即使切换到待部署版本后发现依赖版本有变更，更新依赖包的时间也比在空目录下完整安装依赖包的时间更短。

此外，本地部署还有另外两点优势是使用部署系统所不具备的：

1. 增量构建：我们之前分析过增量构建的实现原理。在构建配置与项目依赖不发生变化的情况下，理论上，本地部署可以让构建进程长时间地驻留，以达到增量构建的效果。
2. 快速调试：本地部署时，构建过程会直接在本地进行，因此有任何构建问题时可以第一时间发现并处理。相比之下，远程的部署系统则需要将一定的时间消耗在链路反馈和本地环境切换上。

因此，如果单从上面的部署环节来看，本地部署的效率一般优于部署系统，那么为什么在企业中通常不建议这样做呢？

本地部署的劣势

同远程部署系统相比，不管从安全性还是人员效率上看，本地部署都存在诸多问题：

流程安全风险

环境一致性

本地部署的第一个问题在于无法保证环境的一致性：

- 同一个项目，不同开发人员的本地环境（操作系统、NodeJS 版本等）都可能存在差异。

- 由于 NodeJS 语义化版本（Semantic Version）在安装时自动升级的问题，不同开发人员的本地 node_modules 中的依赖包版本也可能存在差异。
- 开发人员的本地环境和部署代码的目标服务器环境之间也可能存在差异。

这些差异会导致项目代码的稳定性无法得到保障。例如对于一个 Node 项目而言，在一个 NodeJS 低版本环境下构建的产物，在 Node 高版本环境下就有可能启动异常。

因此，如果项目都由开发人员各自在本地部署，无疑会降低项目的稳定性，增加部署风险。

而使用远程统一的部署系统，一方面避免了不同开发人员的本地环境差异性，另一方面，部署系统的工作环境也可以与线上服务环境保持一致，从而降低环境不一致的风险。

过程一致性
同环境一致性的问题相似，本地部署的第二个问题是无法保证部署过程的一致性。所谓过程的一致性，就是尽可能地让每次部署的流程顺序、各环节的处理过程都保持一致，从而打造规范化的部署流程。本地部署依赖人工操作，这就可能因为操作中的疏漏，导致过程一致性无法得到保障。尽管可以通过将部署流程写入脚本等方式减少人工误操作的风险，但是这和通过部署系统将完整处理过程写入代码的方式相比，仍然不够安全可靠。同时，系统可以记录每次部署操作的细节日志，便于当出现问题时快速解决。

工作效率问题

可回溯性
可回溯性的问题可以从日志和产物两方面来看。

- **日志**：在部署过程中我们可能遇到各种问题，例如构建失败、单元测试执行失败、推送代码失败、部署后启动服务失败等。遇到这些问题时，需要有相应的日志来帮助定位。尽管在本地部署执行时也会输出日志，但是这些日志是临时的，查阅不便，且本地部署的日志至多只能保留当前一次的处理日志，如果希望对历史部署过程进行查看分析，更不能使用这种方式。
- **产物**：通常，部署系统中会留存最近几次部署的构建产物包，以便当部署后的代码存在问题时能够快速回滚发布。而本地部署在项目的开发目录下执行，因此通常只会保留最近一次的构建产物，这就阻碍了上述快速回滚的实现。

相对的，一个规范化的部署系统，则可以记录和留存每一次部署操作的细节日志，以及保留最近若干次的部署代码包，因此在可回溯性上又胜一筹。

人员分工
工作效率的第二个问题是人员分工问题，这个问题又可以从以下几个侧面来分析：

首先部署过程需要耗费时间。在本地部署当前项目的某一个分支时，无法同时对该项目进行继续开发，往往只能中断当前的工作，等待部署完成。

在这个前提下，一个项目中的多名开发人员如果各自在电脑中**进行部署**，无疑增大了上述流程安全的风险系数。但反过来，如果一个项目里只有**个别开发者的本地环境拥有部署权限**，则所有人的部署需求都会堆积到一起，大大增加对有权限的开发者的工作时间的占用。如果不能及时响应处理，也会延误其他人的后续工作。

此外由于分工角色的不同，在许多情况下，部署流程会主动由测试人员而非开发人员发起。当部署在开发人员的本地环境中进行时，会像上面多人开发集中部署那样彼此影响，也增加了相应的沟通成本。

CI/CD
持续集成（Continuous Integration，CI）和持续交付（Continuous Delivery，CD），是软件生产领域提升迭代效率的一种工作方式：开发人员提交代码后由 CI/CD 系统自动化地执行合并、构建、测试和部署等一系列管道化（Pipeline）的流程，从而尽早发现和反馈代码问题，以小步快跑的方式加速软件的版本迭代过程。

这个过程通常是各系统（版本管理系统、构建系统、部署系统等）以自动化的方式协同完成的。而本地部署依赖人工操作，所以并不支持这种自动化的处理过程。

总结

作为部署优化的开篇，这节课我们主要讨论了相比远程部署系统，本地部署的优缺点：尽管本地部署有着流程简化、快速调试等优点，但是相对应的也带来了流程安全风险和人员效率下降等问题。因此一般在规范化的企业技术研发流程中，通常都不使用本地人工操作这样的部署方式。

本节课的思考题是：回顾一下本地部署同远程部署系统相比有哪些优势呢？关于这两者的差异，还可以从其他维度进行分析吗？如果有的话，你可以在留言区回复，期待你的想法。