45 | 持续集成：几十个前端一起工作，如何保证工作质量？ | 极客时间

winter 2019-05-14

你好，我是 winter。今天我们来聊聊持续集成。

持续集成是近现代软件工程中的一个非常重要的概念。它是指在软件开发过程中，以定期或者实时的方式，集成所有人的工作成果，做统一的构建和测试。

与持续集成相对的做法是：独立开发各个模块，在软件开发的最终阶段才做集成。持续集成的优势是及早处理集成阶段的问题，使软件质量和开发进度可控。

现在持续集成还有升级版本：持续交付和持续部署，这些因为需要更为完善的基础设施，目前很少有公司前端团队可以用上，我们暂且不谈。

传统的持续集成概念诞生于桌面客户端开发，在 Web 前端领域，由于技术和产品形态的差别，我们需要构建的持续集成体系也有一些区别。

持续集成总论

传统软件的持续集成主要有以下措施。

- daily build：每日构建，开发者每天提交代码到代码仓库，构建一个可运行的版本。
- build verification test（BVT）：构建验证测试，每日构建版本出来后，运行一组自动化的测试用例，保证基本功能可用。

对于前端来说，有一些现实的区别：

- 前端代码按页面自然解耦，大部分页面都是单人开发；
- 前端构建逻辑简单，一般开发阶段都保证构建成功，不需要构建；
- 前端代码一般用于开发界面，测试自动化成本极高；
- 前端页面跳转，是基于 url，没有明确的产品边界。

基于以上分析，传统的持续集成方案放在前端，要么不需要，要么不适用，要么实施成本高，因此我们不能套用传统的持续集成理论，而需要重新思考前端领域的持续集成体系。

持续集成的目标

前面我们已经分析过，每日构建不需要，前端构建验证测试成本过高难以实施，那么我们是不是可以有一些代替的措施呢？

首先我们要确定前端持续集成的目标，我们回到持续集成的根本理念，一是要及早集成代码形成可测试的版本，二是通过一定的测试来验证提交的代码的有效性。

持续集成的方案

我们进一步思考，前端持续集成如何完成这两个目标呢？

前端代码不需要构建，或者说只需要单页面构建，但是页面与页面之间的跳转是用 url 构成的，所以我们的可测试的版本，不可能通过“构建”来获得。

我们只能通过“发布”来获得一个前端代码的可执行版本，在传统语境中，“发布”的目标是线上生产环境，这显然不行。于是，我们就需要一个预览环境，来做一种“虚拟发布”的操作。

我们再来考虑一下，为界面编写自动化测试用例成本很高，那么如何代替构建验证测试呢？

我们回忆一下，在性能一课，我有讲过，页面的性能可以通过一些自动化工具来分析，还可以通过一些数据采集方案来发现性能问题，对于预览环境前端页面，我们可以采用同样的措施。

除了基于页面结构的分析和数据采集，我们还可以扫描代码。

综上，我认为前端的持续集成的措施应该是这样的：

- 预览环境，代替每日构建，前端每次（或指定次）提交代码到仓库都同步到预览环境，保证预览环境总是可用；
- 规则校验，代替构建验证测试，通过数据采集（如前面提到的性能数据）和代码扫描，保证提交的代码满足一定的质量要求。

接下来，让我来详细介绍一下预览环境的设计和规则校验的设计。

预览环境

前端代码发布到线上生产环境需要有线上的机器和域名，而预览环境同样需要机器和域名，不过，只需要在公司内网即可。

所以建立预览环境的第一步就是申请机器和域名，我们需要运维协助，在预览环境的机器上部署 Web 应用服务器。

有了预览环境的机器，下一步就是建立预览环境发布机制。

有些公司使用脚本发布，有些公司使用 git hook，有些公司则使用一个 Web 应用平台，进行白屏操作，因为各个公司的发布机制千差万别，我这里没办法讲解具体的方案。这里我建议，预览环境的机器发布流程应该跟线上发布保持一致，这样可以最大程度降低成本和降低心智负担。

预览环境的部署和发布机制建立是最基本的需求，在实际应用中，情况要复杂的多，可能需要多个预览环境同时存在。

比如，测试工程师可能要求一个相对稳定的环境来测试，这是一个合理的诉求，比如，全公司大部分业务都可能依赖登录页面，一旦登录页面在频繁发布导致一些预览环境的故障，可能全公司都没办法工作了。

又比如，当服务端工程师联调时，会希望前端的预览环境跟服务端的预览环境对接，而当服务端的代码部署到线上生产环境后，可能又需要前端的预览环境跟服务端线上环境对接。

这些问题都是我曾经遇到过的非常现实的问题，如果今天回过头来设计，我认为应该设计一套带参数和版本号的预览环境，为测试提供特定版本的预览环境，用参数解决那些跟服务端 API 对接问题，但是任何系统都不可能从一开始就设计完善，所以，建议你把重心放到建立预览环境的基本需求上来。

规则校验

接下来我们讲讲规则校验，规则校验可以分成三种措施：

- 页面结构扫描；
- 运行时数据采集；
- 代码扫描。

页面结构扫描可以使用无头浏览器（如 phantomjs）配合一些 JavaScript 代码编写的规则来完成。

运行时数据采集，可以通过在页面插入公共 js 文件的方式来完成，最基本的是用 Performance API 来采集性能数据，用 window.onerror 来采集 js 错误。

代码扫描，社区有一些现成的方案，比如 JSHint，你可以根据实际需要，选择社区方案或者自研。

持续集成的实施

持续集成的实施，是必须严格做到自动化和制度化的。我们可以通过上节课讲的工具来完成持续集成。其它部分，都可以通过工具和制度来完成，这里需要重点讲的是规则校验中的规则部分。

我们刚刚讲解的规则校验仅仅是搭建好了平台，而规则本身，我们需要先形成一个共识，然后在前端团队内部形成一定的更新机制。

这里，我建议用 issue 的方式来管理规则的提案，可以在周会或者月会上讨论，充分保证整个团队对校验规则的一致意见。

这里，我们必须警惕三种错误：

- 少数人拍脑袋决定校验规则；
- 一成不变的校验规则；
- 频繁无规律变化的校验规则。

只有经过民主讨论、定期更新的校验规则，才能在团队中起到积极作用。校验规则决定了整个前端团队的开发体验，所以必须非常慎重。

持续集成的结果

持续集成机制的建立本身就可以视为一种结果，它能够让整个团队的代码质量有一个基本的保障，提前发现问题，统一代码风格，从而带来开发体验和效率的提升。

此外，持续集成的结果也能够以数据的方式呈现出整个开发团队的健康状态，这是管理者会非常关注的一个点。

总结

今天我们讲解了持续集成，持续集成这个概念最早来自桌面客户端软件开发，应用到前端领域，会有一定的变化。这里我提出了一个预览环境 + 规则校验的前端持续集成体系。

预览环境需要申请机器和域名、部署和建立发布机制，规则校验有三种方法：结构扫描、数据采集和代码扫描。

持续集成的实施需要重点关注校验规则部分，要建立一个民主讨论、定期更新的校验规则。持续集成机制的建立就是其结果本身，此外，系统中产生的数据也可以有一定管理价值。

最后留一个问题，你所在的团队，是否有做持续集成呢？请你设计或者改进这个持续集成方案。

bd2star