

# 加餐 | 增强篇思考题答疑

2023-03-08 杨文坚 来自北京

《Vue 3 企业级项目实战课》

课程介绍 >



讲述：杨文坚

时长 01:27 大小 1.33M



你好，我是杨文坚。

课程的最后一个板块“增强篇”，我们就学完了，核心是跳出“前端”和“服务端”固有技术领域的禁锢，发散性地分析项目维护知识点，更好地迭代项目，保证项目有持续稳定的生命力。

在搭建平台的**功能扩展**方面，首先，我们了解了功能扩展相关的技术，掌握全栈项目如何做服务端单元测试，跳出了纯前端领域的单元测试范围，优先保证存量代码的功能稳定；然后，学习了全栈化的综合功能扩展，大思路就是要找到扩展的线索脉络，沿着线索的方向，做扩展的设计和实现。

在搭建平台的**功能维护**方面，我们学习了一些延伸技术，主要是系统运维相关的知识点。先借助 Node.js 的多线程和多进程案例，入门了提高服务器资源利用率的方法；然后深入学习了全栈项目的系统运维工作，站在运维工程师的视角上，学会如何独当一面，管理项目，从“全栈型工程师”蜕变成“全能型工程师”。

今天我们对增强篇做统一答疑，如果对课程中的其他知识点还有疑问，也欢迎你留言。

## 32

课程：🔗 [32 打造 Vue.js 和 Node.js 全栈项目的单元测试](#)

提问：我们提到了服务端做基准测试的必要性，那为什么前端项目代码很少见到做基准测试？

前端代码的基准测试，主要在前端 JS 框架或 JS 库等基建场景运用比较多，目的是验证 JS 基建代码性能效果。例如，用 JS 实现的时间日期格式化基础库，把时间戳转成指定格式的日期，就会用基准测试来验证性能。

但是，**前端业务场景很少做基准测试，主要是因为业务需求变化频繁**，基准测试的性能标准也会跟着频繁改动；**另外，客户使用的浏览器很难统一**，导致页面代码运行环境不一致；而且，不同厂商浏览器，有不同版本，基准测试很难得到统一的测试标准。

而服务端必须做基准测试，有个原因是服务端能被“自主掌控”，能统一开发或生产环境的服务器配置，在相同条件的服务端环境里，可以比较准确验证基准测试结果。

理论上，前端也能“自主掌控”统一浏览器，但主要是业务层面的操作。比如能引导或规范客户使用指定的浏览器，例如 **Chrome**，那就可以减少浏览器碎片化程度，集中在指定浏览器的主流使用版本上做基准测试。

## 33

课程：🔗 [33 对 Vue.js 全栈项目做优雅的页面功能扩展](#)

提问：如果扩展成兼容 Java 服务端提供的页面，这些页面也包括了多页面和单页面应用形式，要如何进行底座设计？

按照我们课程里的思路，扩展兼容 Java 服务页面，等于是“跨技术体系”的项目页面整合，这个时候**可以考虑使用“微前端”技术方案来解决**。

如果不使用微前端技术，硬要用课程里的页面扩展方式来实现的话。按照我们课程的思路，就先做业务和技术两个层面的归类分析，业务上，需要处理受众用户，判断是否有登录态等页面

状态的业务逻辑，技术上，就要考虑页面扩展整合问题，可以考虑使用 `<iframe>` 标签来内嵌页面。

最后，根据选择技术方案，约定扩展页面规范，实现对应的技术底座，方便统一扩展 Java 服务的页面。

## 34

课程：🔗 [34 对 Vue.js 全栈项目做服务端功能扩展](#)

提问：如果 Node.js 服务功能需要扩展功能，使用其他服务端语言实现的服务功能，比如 Java 服务提供的 API，怎么做功能扩展？

Node.js 服务调用其他服务端语言的服务，其实就是跨服务端通信，或者是跨服务调用。

而跨服务通信或调用，跟所用语言关系不大，**主要是选择什么 RPC 方案。**

RPC，全称是 Request Remote Call，中文称“远程过程调用”。RPC 是一种技术的统称，主要是解决从一个服务节点去请求另一个服务节点的问题。实现 RPC 通信的具体技术方案有很多种，主流以 TCP 通信方式为主，也有部分用 HTTP 来作为实现技术。

用 RPC 进行跨服务间通信，有很多种技术框架，例如技术社区开源的 gRPC、Thrift 等 RPC 开源框架，都可以实现 Node.js 服务和其他语言服务之间的通信，最终的通信都是以 TCP 等计算机网络技术来进行。

总的来说，跨服务通信，跟用什么语言关系不大，核心是你要了解不同 RPC 技术框架的通信协议。

## 35

课程：🔗 [35 多进程部署，最大限度利用服务器资源运行 Node.js 服务](#)

提问：服务端的性能优化措施，除了多线程、多进程的优化措施外，还有其它优化方案吗？

服务端的性能优化还有很多优化方案。首先我们要明白性能瓶颈问题的原因，表层原因是同时时间大量请求，或者是代码存在 CPU 密集计算的阻塞等等。底层原因是同时间执行程序出现

CPU 和内存资源使用的瓶颈。

因此，无论怎么设计解决方案，都可以在这两个层面展开：**表层问题一般是具体场景，具体分析设计；底层问题解决思路比较统一，就是解决机器资源瓶颈问题。**解决资源瓶颈问题有两个方向，压榨同机器资源，添加新机器资源进行多机器部署。

- 压榨同机器资源，其实就是多进程操作，那么除了多进程外，其实可以用多端口多次部署服务，用 Nginx 服务做负载均衡，统一用一个端口，管理多个服务端口。
- 多机器部署服务，就是同机器资源没有优化空间的情况下，新增多个服务机器，最后用一个服务机器的 Nginx 服务做统一的负载均衡服务，对外暴露统一 IP 地址和端口。

## 36

课程：[🔗 36 日志收集与问题排错，守护好 Vue.js 和 Node.js 的全栈项目](#)

提问：业界经常提到，用云服务、服务容器化、Serverless 等方案能解决运维成本问题，那么 Node.js 全栈项目如何选择相关服务方案呢？

首先，我们要了解各种方案的概念。

**云服务**，可以称为“云服务器”，你可以简单地理解成是一种“技术商品”。主要是技术厂商将大量服务器进行“资源集约化管理”，然后提供“虚拟化服务器”，也就是说，云服务器配置是数字化虚拟的，可以按照需要，选择升级或者降级配置，从而不影响实际服务功能。同时，很多云服务厂商能基于此类虚拟化机器，提供数据备份、防攻击保护等增值扩展功能。

总的来说，云服务优点就是资源扩展方便、操作相对简单，大量运维工作可以购买相关增值服务来解决。

**服务容器化**，是一种技术概念，指将项目代码和项目环境一起打包管理，进行统一化的部署流程。比如我们课程的运营搭建平台项目，需要依赖 Node.js 环境，那么我们就可以把项目代码和 Node.js 环境一起打包到容器配置（例如 Docker 容器配置）里，最后用这份容器配置，或者容器配置生成的镜像，直接部署到有容器环境的操作系统（例如 Linux 系统或者 Windows 系统）里。

总的来说，服务容器化优点是减少了部署和管理成本，不用考虑不同系统环境配置，因为都统一打包到一个容器里。


**Serverless**，可以称为“无服务架构”“无服务服务”或者“无服务计算”，也算是一种云服务厂商的“技术商品”，可以让使用者不用关心服务所在操作系统，只关注自己项目代码。系统资源问题都会自动按需扩容或缩容，无需关心系统资源不够用，或者利用不充分的问题。

了解了这三者的概念，对应解决的运维痛点是什么？我们如何选择呢？

- 云服务，主要能解决服务器资源动态扩容问题。系统运维可以通过购买服务，实现半自动化管理，这里的“半自动化”，是指云服务厂商提供的增值服务，不一定能满足所有运维诉求，还是需要人工方式来解决一些系统运维问题。
- 服务容器化，主要是解决部署和管理问题，方便实现一套配置部署多种操作系统，或者一个系统部署多个容器来“超量”使用资源。但是，这只是一种技术方案，需要人工来执行。
- **Serverless**，主要解决绝大部分运维成本问题。在正常情况下，开发者只需要花钱购买相关技术服务商品就好，但是服务的底层是“黑盒”的，如果出现问题，可能会比较棘手。

课程的所有思考题，我们就解答完成了，感谢你的参与。如果对课程中的知识点还有其他疑问，也欢迎你留言。

分享给需要的人，Ta购买本课程，你将得 18 元

 生成海报并分享

 赞 1  提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 36 | 日志收集与问题排错：如何守护好Vue.js和Node.js的全栈项目？

下一篇 结束语 | 时刻做好“军备竞赛”，机会只留给有准备的人



# 精选留言

 写留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。