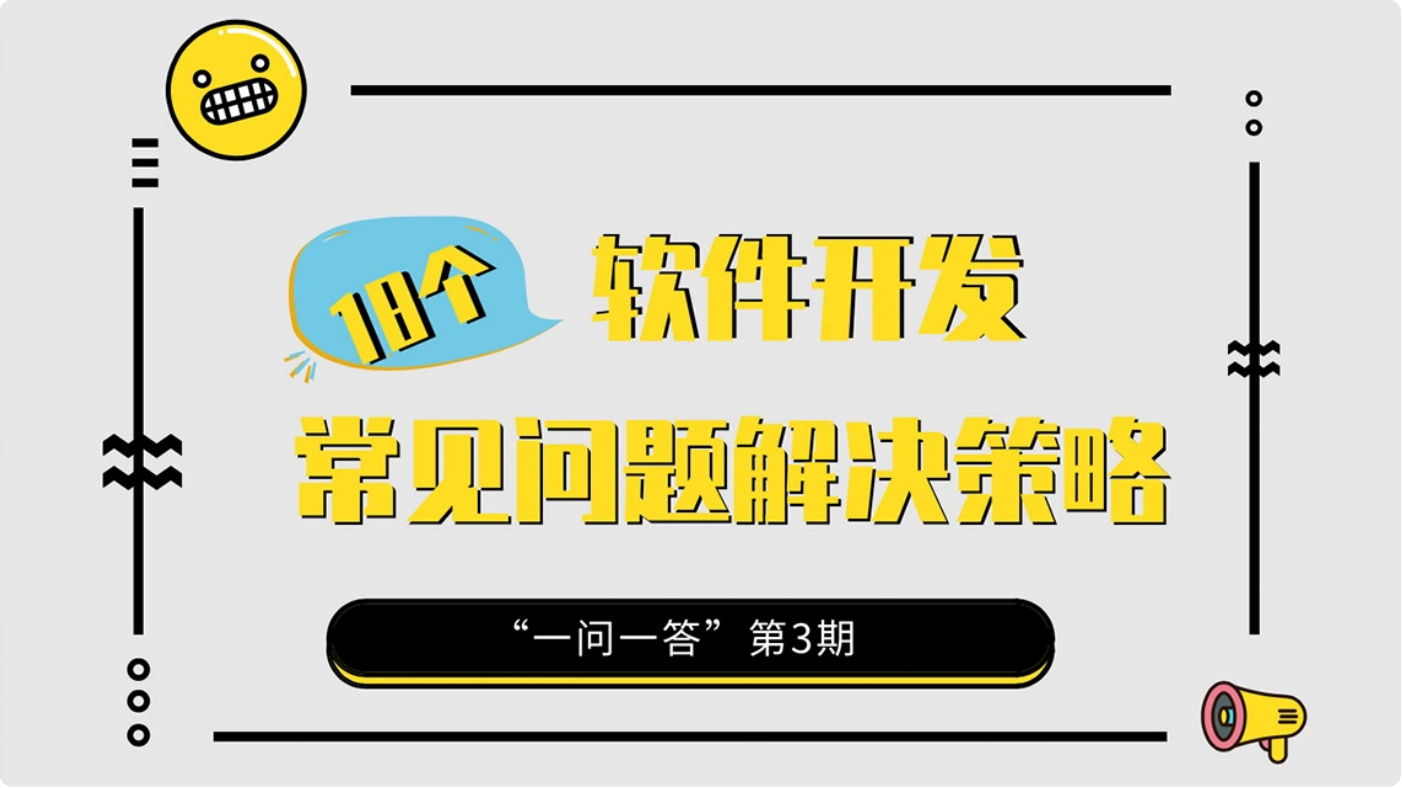
“一问一答”第3期 | 18个软件开发常见问题解决策略

2019-05-09 宝玉 来自北京

《软件工程之美》



你好，我是宝玉。我们专栏已经完成了架构设计和开发这两个模块的学习。这两个模块都是和技术有很大关系，也是很多人关心的内容。

希望你通过对架构设计内容的学习，能控制好软件项目中技术复杂的问题。遇到技术选型，能心中有数，通过一个科学的过程去完成选型；对于项目中的技术债务，能及早识别，及早应对。

通过对开发模块的学习，可以帮助你在项目中搭建持续集成环境，推行自动化测试，改进基于源代码管理工具的开发流程。借助工具和流程，让你项目的开发质量更上一个台阶。

本篇继续分享同学们的精彩留言，这些问答和同学们的分享都是对专栏内容的最好补充，希望可以帮助你更好地学习和理解软件工程知识。

一问一答

No.1

一路向北：每次看这些架构的思想方法的时候，总是和实际的应用没能很好的结合起来，原因是不是架构设计的实践不够？或者是对各种实现的分析和思考太少？

宝玉：我觉得不仅要有架构实践，还要有不同场景的实践。

举个例子来说，你平时做企业应用架构，没什么流量，没多少数据，复杂的地方都在业务逻辑，这时候你去看那些讲大数据、讲高并发的文章，很难带入到场景去。

还有就是一些架构，不自己搭一遍是很难了解其中的优缺点的，这也是另一个原因。

可以考虑有机会自己尝试，把看到的一些好的架构用一个原型程序搭一遍，造一点数据出来，用工具压测一下，这样会更有感觉。

和实际应用相结合的问题，一方面说明你现有的架构可能并没有什么大问题，没有那么迫切的需求要改造；另一方面可能还是因为缺少实践经验，心里没底，不知道真用上了有没有用。

No.2

小伟：比较规范的文档有哪些，他们功能分别是什么？

宝玉：对于瀑布模型，每个阶段结束后，都有相应的验收文档，而敏捷开发则没有那么多硬性的要求，而是根据项目需要，写必要的文档。

有些团队对于测试阶段，会有测试用例文档、测试验收报告，发布前还会有部署文档、维护手册，但现在这类文档基本上被测试工具、部署脚本替代了，也没有什么存在必要。

我觉得项目中必要的文档，主要包括这几类：

1. 设计类文档

这类文档主要用来说明、讨论需求设计、架构设计，可以用来了解、讨论和评审，以及记录后续结果。

2. 说明类文档
   这类文档用来对规范、API、配置、操作等做说明，便于规范和统一。
3. 报告类文档
   对事情结果的报告和说明，比如说验收报告、故障报告、调研等。

而这些文档的价值，在于帮助成员了解设计、参与讨论，记录项目成果，减少沟通成本。重要的不是文档多丰富，而是这些文档有没有价值，你能不能及时通过这些文档得到想要的答案。

所以你也可以对照一下你的项目中，现在的文档有哪些地方是可以简化的，哪些地方是要增强的。

比如说，概要设计 / 接口设计 / 详细设计是不是可以适当合并，减轻文档工作量？PRD 是不是够详细？会不会引起歧义不容易理解，要不要增加原型设计文档辅助？

No.3

邢爱明：项目团队的开发人员，基本都是从外包公司临时找的，水平参差不齐，稳定性差，因此技术选型更多考虑技术的普及度的和是否容易学习掌握，从这方面看基本不太可能选择比较小众、但在特定领域很高效的技术。

加上是企业内部管理的系统，数据量和用户数量可控，因此存在技术瓶颈的可能性很小，综合下来看，最好的选择就是最成熟和通用的技术，比如说选择 java 技术栈，web 开发的 ssm 框架等，但这样长远看团队和个人的技术能力很难提升，请问老师在这方面有什么建议？

宝玉：我觉得团队的技术提升和项目的技术选型要分开，不要总想着两个都兼顾，优先保证好项目稳定、低成本运行。

技术提升这种事，需要让一部分人先成长起来，然后带动其他人。我自己工作之外会做一些业余项目，然后在这些项目中体验新的技术，体会其中优缺点，然后再逐步应用到工作的项目中，传授给同事们。

我也鼓励其他同事这么做，去做一点自己的项目。但工作中的项目，我是很保守的。

No.4

alva_xu：对于开源技术方面，老师有没有什么经验来指导选型？

宝玉：开源技术选型，我的经验一般是这样的。

1. 先找朋友推荐，少走一点弯路。
2. 没有推荐的话，就去网上搜索，找几个满足需求的备选。
3. 对比以下几个指标：

   代码质量、有无测试；

   文档健全度；

   看 Issue 处理情况、最后更新时间（无人维护的项目后续恐怕有问题都没法解决）；

   看 Star 数量，通过 Google 和 StackOverflow 看使用情况。

4. 自己按照说明试试看。

No.5

alva_xu：有没有什么大的原则可以指导技术选型？比如技术成熟度等？

宝玉：我认为在满足设计目标的前提下，大的原则还是在于项目约束，尤其是成本和时间，然后就是看技术可行性和风险是不是可控，其他看团队风格，有的偏保守有的追新。

比如说我自己的原则：

1. 成熟的好过新酷的；
2. 流行的好过小众的；

3. 团队熟悉的好过陌生的；
4. 简单的好过复杂的；
5. 开源的好过商业的（有时候也视情况而定）。

No.6

Charles：有着正常职位或头衔的架构师，对一个全新的项目理解产品需求后进行架构设计，一般会产出哪些“东西”，来满足后续的架构讲解和项目开发过程中的沟通？

宝玉：互联网产品特点是用户多，企业产品特点是业务复杂，所以架构的侧重点不一样。

架构师在架构设计后，产出首先是架构设计文档，让大家理解架构。然后还要写架构开发的文档，比如如何基于这个架构开发功能模块，有哪些公共 API 可以调用，怎么样是最佳实践，要遵守哪些规范等。

再要帮助搭脚手架和基础模块或示例项目，也就是要搭建一个最基础的可运行项目，通过这个项目，大家可以直观地理解你的架构是怎么落地的，通过基础模块或者示例项目，可以知道如何基于框架开发，后面就也可以照葫芦画瓢照着实现。

还有就是在开发过程中，要答疑、解决架构中存在的问题，对架构做优化，还要做代码审查，对于不符合架构规范的地方要指出和修正。

No.7

Dora：互联网架构，要考虑互联网很快的迭代速度，所以对于扩展等特别注意。企业架构，内部 IT 系统相对稳定，对比互联网架构，更简单？

宝玉：挺好的分析。帮你补充几点：互联网架构不仅迭代会快一些，用户规模通常更大，但业务也会单一些；企业应用通常业务比较复杂，尤其是和行业会有一些结合，但是用户规模要小很多。这些特点，都会影响架构设计的选择。

No.8

WL：老师能不能具体讲讲重构有哪些原则和要注意的地方，感觉一直得不到要领。

宝玉：重构的要领我觉得两点。

第一：你要先写一部分自动化测试代码，保证重构后这些测试代码能帮助你检测出来问题；

第二：在重构模块的时候，老的代码先保留，写新的代码，然后指向新代码，或者用特定开关控制新旧代码的指向（这样上线后可以自己先测试，有问题也可以及时关闭），然后让自动化测试通过，再部署测试，新代码没问题了，删除旧代码。

No.9

bearlu：有没有事情管理的工具？因为如果不记录下来，一会儿就忘记了。

宝玉：留言区 McCree 同学推荐了滴答清单。我个人的话，一般就用系统自带的记事本记一下，或者贴一个便签纸在显示器。如果时间跨度长，我就记到 Calendars 上，加上提醒。工作中的任务，我则会创建成 Ticket。

No.10

W.T：现在还有一种说法：提倡基于主分支开发，效率更高；而不是您提到的每人基于自己的分支开发完再合并回主分支。您怎么看待这个问题？

宝玉：我认为对于软件工程来说，很多问题，并不是只有唯一解，即使是最佳实践，也得看适用的场景和团队。

无论是基于主干还是分支开发，有两点需要注意的：

1. 就是一定要有一个稳定的分支，可以随时发布的那种，至于是叫 master 还是叫 release 并不重要。
2. 合并之前要有代码审查和自动化测试（配合 CI）。

上面两点才是核心。

No.11

hua168：如果一个项目有 5 个开发做，持续集成怎么保证不乱？比如开发 A 刚刚修复的 bug1，开发 B 把自己修复的 bug2 上传，之前的代码 bug1 没修复，怎么办？如果采用分支怎么合并？如果是直接更新 master 分支，那 A 不是白做了？

宝玉：要注意是“合并”而不是“覆盖”。比如说 bug1 涉及 file1 和 file3 的修改，那么开发 A 合并的时候只合并 file1 和 file3。

等到开发 B 修复了 bug2，修改了 file1 和 file2，file2 直接合并，file1 需要手动去修复合并冲突才能合并。

每个人开发之前，都会从 master 获取最新版本，合并的时候，如果出现冲突，要先解决冲突才能合并进去。这些其实应该自己去动手试试，会体会更深刻。

No.12

dancer：在微服务架构中，一个服务在测试环境的交付验证，往往还依赖于其他相关服务的新版本，导致新的 feature 很难独立的交付。对于这种情况，有什么好的方法吗？

宝玉：我觉得对于大部分时候，微服务之间应该是独立的，而不是依赖过于紧密，如果每一个新功能都会这样，那架构设计一定是有问题的，需要重新思考服务划分的合理性。

但你需要有更多上线或者场景我才能针对性提出一些意见。对于有一些确实需要跨服务合作的大 Feature，这样也是正常的，就是需要一起协作，事先商量好通信协议，分头开发，再联调。

No.13

Gao：老师所讲排查生产问题的案例，首先回滚版本，再看日志。这会引发更多的系统功能不可用吧，两个版本之间的功能差异尚不清楚就直接回滚，系统风险是否被进一步扩大？

宝玉：这个确实要具体情况具体看，因为我日常的系统上线，都会有回滚方案，回滚也是自动化的很方便。有些跟数据库相关的，如果数据库结构发生变化又产生了新数据，确实没法直接回滚。

No.14

kirogiyi：团队成员的能力和素质参差不齐，如何有效的去组织和管理项目的自动化测试，自动化集成？

宝玉：首先，你要先搭建好自动化测试环境，让自动化测试代码能跑起来，最好要和 CI（持续集成工具）整合在一起，每次提交代码 CI 都会跑自动测试，然后能看到运行结果。

然后，把自动化测试作为开发流程的一部分，比如说要代码审查和自动化测试通过后才能合并代码。这部分工作如果和 CI 集成会容易很多。

再有就是要培训，比如遇到不会写的，开始先带着他写几个，确保他学会了自己能写，然后下次代码审查的时候，看到缺了就要求补上，还不会就继续教，来不及写的就创建个 Ticket 跟踪起来。

简单来说，就是代码审查 +CI+ 培训。

No.15

探索无止境：各种类型的测试覆盖率你们一般采用什么指标？个人感觉在理想的情况下最好是做到百分百覆盖率。

宝玉：100% 覆盖，这个我觉得可以作为一种理想追求，但是没必要追求极致，还是要在进度和质量之间有个平衡比较好，毕竟进度也很重要。

另外对于前端业务，我更重视集成测试的覆盖，对于主要业务场景集成测试覆盖到位后，单元测试也就有比较多的覆盖，相对性价比更高，然后再逐步补充单元测试的覆盖率。

No.16

起而行：持续集成怎么理解呢？我看知乎上说，有的团队成员在一天内多次进行编译，发布或自动化测试。

宝玉：狭义的持续集成不包括发布，主要指集成，持续的（每次提交代码变更都触发，频繁地提交）对代码进行集成（合并到主干），但集成前要确保自动化测试通过。广义的持续集成还包括部署，也就是集成后自动部署测试环境 (持续交付) 或者生产环境（持续部署）。

No.17

小小：请问下有没有介绍开发如何写好测试不错的书？

宝玉：推荐：《how we test software at microsoft》中文版《微软的软件测试之道》。不过没有书其实你也可以找到很多资料的。比如我平时写前端程序，那么我会去 GitHub 或者 Google，通过关键字、语言找跟我项目类似的开源项目，然后看其中有没有自动化测试写得好的。

找到了 (例如：reactstrap、electron-react-boilerplate、kitematic) 就照葫芦画瓢好了，因为都是真实项目，所以特别简单有效，建议你也可以试试。

另外耐心一点，你也可以看到很多关于测试知识分享的技术文章，多看一看也有收获。

No.18

hua168：代码审核是纯手工做的吗？没有好的工具？

宝玉：代码审查可以参考 GitHub 上一些开源项目的 PR Review，通常网页上可以清楚地标记出代码修改，针对代码行可以写 Review 的评论，这就已经很方便了。

其他工具主要是 Lint 检查代码规范、语法错误等，这个一般在 CI 里面就集成了。

精选留言

陈珙：

在没有特殊要求的情况下，项目中更加倾向选择更为熟悉的技术，因为我们需要对项目的质量与交付时间负责，可以做到可控的。而新技术有着新的设计思想与强大的功能，同时也伴随着无法预知的“坑”。在后续产品迭代的时间里，有针对性的升级或者选择更换同类技术里更优的。

相关阅读： 22 | 如何为项目做好技术选型?

Y024：

Appfuse（一个 Web 开发基础平台）的作者 Matt Raible 曾总结了选择 Web 框架的 20 条标准。

同时，他也整理了一份表格，你可以根据自己的权重进行调整，产生自己的分析。

但是现实情况，大家可能更遵循的是“经济适用原则”，比如：很多人提到的，负责人会啥就用啥，或者大公司、业界流行什么就用什么。

有位大佬说过，“这个世界是，你认为有很多选择，其实只是幻觉，大多数人只有很少的选项。技术研讨会，搞一个选型：hadoop + mysql + xx 时髦技术。架构师唾沫四溅吹一下午，结果老老实实上 Oracle 单例。”

相关阅读： 22 | 如何为项目做好技术选型?

kirogiyi：

架构师是一个概念性职位，没有明确的界定，需要具备的能力和素质也是千差万别，每个开发人员心目中的架构师画像也都不一样，神秘的 IT 牛人，高级的保姆，无休的恶魔…

在我看来，一名优秀的架构师应该具备良好的技术思维、产品思维和项目管理思维。技术思维是基础，评估技术难度、分析技术复杂度、准确把握技术方向，这些都是架构师在设计架构时面临的技术决策。

产品思维是骨架，在产品思维上构建起来的整体全面的产品意识，可以对业务、功能、模块进行明确的抽象、分治、迭代等等。

项目管理思维是方向，无论是敏捷管理模型还是瀑布管理模型，都需要在不同的时间、不同的环境条件下去关注金三角理论的取舍所带来的影响，降低技术以外对项目带来的局限性。

不过，架构师也不是想象中的那么神秘。开发人员和架构师的差别，最主要是层次和格局上的差别，导致最终产生了不同的结果而已。

试想，两个能力相同的开发人员，一个的目标是每年涨工资（80% 开发人员），他会去努力多做事，拓展技术的深度；一个的目标是 CTO（20% 开发人员），他会去努力多做事，多思考，多学习，多交流，尽力做到面面俱到。几年以后的结果就不言而喻，至少坚定的目标能够推动过程的发展。

相关阅读：23 | 架构师：不想当架构师的程序员不是好程序员

alva_xu ：

讲到架构，我想先得谈一下康威定律。康威在 1967 年曾说过：

> Organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations.

通俗地说，就是组织形式等同于系统设计。所以系统架构设计的进化，是和组织形式的变化结合的。

从 ITIL 来说，BIA，（business IT alignment）是 IT 的核心，所以充分认识组织的业务模式和运营方式，才能让架构师设计出适合于企业的系统架构，架构设计的最高境界就是适合企业业务的运营。

从单体架构到微服务架构，从前后端分离到中台，都是架构适应业务（功能与非功能需求）的体现。所以架构师首先必须要有业务思维、产品思维。TOGAF 把企业架构分成业务架构、应用架构、数据架构、技术架构四个子域，我觉得相当全面。

从程序员开始，如果能培养好老师讲的架构师能力模型中的四个思维和三个能力，我们可以给自己规划出一个架构师的成长路径，从单个业务应用开始，然后扩展到一个业务领域，最终到达企业架构师，甚至成为跨企业应用架构师的境界。

相关阅读：23 | 架构师：不想当架构师的程序员不是好程序员

纯洁的憎恶：

技术债务不全坏，与金融债务一样，需要具体问题具体分析。轻率 & 有意的债务要避免。谨慎 & 有意的债务有收益。

轻率 & 无意的债务要警惕。谨慎 & 无意的债务要改变。识别债务防患于未然。根据成本收益分析，决定重写（一次性还款）、维持（只还利息）还是重构（分期付款）。

相关阅读：24 | 技术债务：是继续修修补补凑合着用，还是推翻重来？

kirogiyi：

在研发过程中，产生技术债务的时候，稍微有点技术功底的人，或多或少都会有感觉的。

比如：有重复代码的时候，会意识到好像已经写过了；函数命名的时候，会意识到好像有个相似的名称已经命名过；函数行数过多的时候，自己心里会感觉不舒服等等。

更有甚者，你去整理这些问题还会被同事标上“强迫症”患者的称号，还是放弃吧。技术债务就这样在外部和内部双重压力下自然而然的产生了。

那么如何产生有利的技术债务呢？我觉得应该从公司制度、研发流程、个人素质培养三方面入手。

公司制度实际上是为领导层准备的，领导层以身作则去影响下面的员工，员工就没有冒犯的理由，比如合理的奖惩制度，要做到公平合理，一视同仁；

研发流程主要是让团队成员知道自己什么时候该做什么事情，如何去按照指定的约束去做好自己的事情，除此之外，还应该给予明确的成长上升空间；

员工素质的培养则需要从一个人的职业素质，技能优化，团队协作方面着手，让他们拥有积极努力的心态参与到工作中去，这基本上就能解决最基础的技术债务问题（领导决策错误产生的技术债务另当别论）。

在我遇到过的技术债务中，主要由领导决策、产品业务逻辑、技术最初选型、技术更新换代、团队综合素质中的一种或几种导致。对此，我只能说个人能力达到什么层次就应该去解决什么层次的技术债务，不能去推诿和落井下石，在你手中的技术债务就应该当成自己欠下的技术债务来解决，这样才能持续性的做好自己分内和分外的事情，工作起来才能得心应手。

相关阅读：24 | 技术债务：是继续修修补补凑合着用，还是推翻重来？

kirogiyi：

我觉得高效，意味着自律，自律的好坏是可以通过你散发出来的气息让周围的人感受到的，比如：说话有没有条理，做事拖不拖延等等。

生活自律，你会发现每一分每一秒都充满了希望和力量，用积极乐观的心态去完成每一件事，知道自己上一步做好什么，下一步才能做好什么。

工作也是一样，要想高效完成任务，需要利用前辈们总结的思想和方法，去长期实际应用，在使用的过程中就会体现出你的高效，不能说我知道单元测试，我知道 CI...，很少有人讲我一直在用。

如果我们注意观察，会看到身边的同事，有的很少加班（活蹦乱跳），有的经常加班（蓬头垢脑），做了一样的事情用了不一样的时间，此时就能真正的体会到高效做事的魅力了。

我不提倡加班的原因就在于此，但那是针对高效人士的，低效人士不加班，老板是不会答应的。而一般对自己的时间把握比较好的人，在估算工作时间或工作量的时候，都比较果断，不会支支吾吾，还会主动给出具体时间点和阶段性成果，让人觉得这才是真正做事的人应该有的态度。

我的看法是，积极、主动、自律是高效人士的必备素质。

相关阅读：25 | 有哪些方法可以提高开发效率?

nigel:

就学习能力而言，“祭海先河，尤务本原之学”，重要的是对基础知识的掌握。就像侯捷先生说的“基础的东西不易变，不易变的可重用”。

相关阅读：27 | 软件工程师的核心竞争力是什么？（上）

_CountingStars:

之前看到过一个关于 code review 的观点：在让别人 review 你的代码的之前，你要确保你的代码没有基础的问题，比如单元测试要通过，不能有代码风格问题，首先你要确保你的代码是能正常工作并解决需求的。当然这些基本都可以通过自动化来操作，比如提交 PR 的时候，自动化的检查代码风格，运行单元测试。保证邀请别人 review 你代码的时候，不要为这些小事费精力，提高 review 效率和积极性。

相关阅读：30 | 用好源代码管理工具，让你的协作更高效

bearlu：

其实我觉得用什么源代码管理工具都没关系，最重要是要了解工具，形成流程，按流程走，然后纠正流程。

相关阅读：30 | 用好源代码管理工具，让你的协作更高效

思辨时刻

Charles：

三四线城市，技术选型前期主要考虑：当地市场什么人才比较充足，比如后端 PHP 人多，那就 PHP，学习成本也低，几人团队协作起来也不是大问题，而且前期扩充人员也比较好招人；另外前期应该也不会在语言层面出现性能问题。

然后数据库基本就选 MySQL，够熟悉够成熟。前端的话，web、小程序、ios、android 之类的都统一 MVVM 思想，进行前后端分离开发，这样各个用户端都可以统一 API 提升效率，这个也会从产品角度思考。

如果产品经理就只是需要一个 PC 网站，而且短期也没升级计划，就选择传统的后端渲染 web 页面方案。可能会站在目前项目或经历过的项目经验去思考问题，期待老师回复指正。

宝玉：

我觉得你的选型思路在项目发展阶段，包括没有很大规模之前都是没有问题的。选最熟悉的、流行的往往也是风险比较低的。包括如果就是一个 PC 站也不做 SPA（单页应用），也没有必要前后端分离。还是看是不是能低成本满足好项目需求和业务发展。

有一点补充的，就是前端除了 MVVM，像 React 的 Flux 和 Redux 的架构模式，也是一种很好的架构模式，但在非 Rect/Vue 的项目中应用不多。

相关阅读：22 | 如何为项目做好技术选型？

好，今天的加餐就到这里，非常感谢同学们的用心留言，希望大家都能持续思考与总结，不断精进！

感谢阅读，如果你觉得这篇文章对你有一些启发，也欢迎把它分享给你的朋友。

精选留言 (5)

mgxian
2019-05-09

重构最重要的原则是小步快跑 甚至小到你觉得这一步没有必要 这样做的意义是可以随时停下来 不影响用来的功能正常运行 如果出错 由于步子小 你很容易能找到哪里修改出现了错误 重构之前一定要写测试 如果没有测试你将不知道自己的重构有没有破坏原来的功能。

测试可以使用BDD的测试形式 测试不是说为每一个函数 每一个类写一个单元测试 应该为那些public api 写测试，为使用场景写测试。想把测试写好 可以看看 那本测试驱动开发 tdd by example 重构强烈建议看看 重构第二版 注意看每一个重构小步骤 每一个小步骤都是为了可以让重构随时停下来 而不影响原来的功能

作者回复: 👍👍非常有价值的分享！

5

Sudouble
2019-05-10

以前天天追着某种开发语言的语法，最近渐渐地意识到，学好语法其实也仅仅起了个头，如同刚学会写字的小学生，要写出优美的散文，又是另外一回事了。

还有是测试驱动开发的强大之处。自己写了一个小软件，最近要加新功能和补bug，庆幸那时候忍着补齐了测试代码，不然在改动里真是寸步难行。

作者回复: 👍是呀，语言学会不难，要用好才行！

测试覆盖很重要，重构的时候就省心了！

3

易林林

2019-05-09

特别感谢宝玉老师准备如此精彩的专栏，使我们无论在技术、架构、产品、管理方面都得以全面成长。看到IT同行如此多独特的见解，受益匪浅，同时发现自己还有很多需要思考和践行的地方。

作者回复: 也谢谢你还有其他很多的精彩留言，补充了很多有用的分享👍

3

纯洁的憎恶

2019-05-10

学习到的知识一定要多用，而且要变着花样的用。

作者回复: 👍知识应用了才能变成自己的

2

ifelse

2022-07-03

学习了