Angular，工程之美

[](https://www.zhihu.com/people/alpha-gde)

**[汪志成](https://www.zhihu.com/people/alpha-gde" \t "https://zhuanlan.zhihu.com/p/_blank)**

angular.cn 译者，二十三年码农，现代儒生

328 人赞同了该文章

Angular，工程之美

汪志成

汪志成

angular.cn 译者，二十三年码农，现代儒生

328 人赞同了该文章

如果只能用一个词来概括 Angular 的优点，那我会选“工程化”；如果要换个文艺点的词，我会说 “大巧若工，大道至简”。这个诞生于一群 Google 工程师之手的框架，从一开始就打上了鲜明的“工程化”烙印。

什么是工程化开发呢？它是和手工作坊式的开发相对而言的。主要的特征是需要多人、多时段的分工与配合，而不是一个超级程序员写完90%的代码，其他人只负责打下手。

但大中型系统面对的挑战不是靠超级程序员就能解决的。如果一个系统的复杂性逐渐累积，迟早会让它的维护成本越来越高，变成鸡肋一般的存在。要解决复杂性的问题，首要办法就是从架构层面上隔离系统的各个部分，让它们彼此之间相互独立，互不干扰，也就是说，分而治之，各个击破。然而这并不容易。

多人开发最大的挑战就是如何分工协作。

在 Angular 中，用来支持分工协作的主要基础设施是模块。

比如，如果我们不通过模块来给应用划分出硬边界，那么 A 编写的组件就可能会随手引用到 B 编写的组件，但 B 写这个组件原本只是为了给自己用的，根本没有考虑过复用问题。那么 A 和 B 之间的工作就产生了意外耦合，而意外是工程化最大的敌人。实践证明，各种形式的意外耦合往往会埋下地雷。

有了模块，B 就可以把这个组件先私藏起来，让别人没法用它，这样 A 写的组件就不会再无意间依赖 B 写的组件了。如果将来多方达成共识，就可以收集多方需求，真正为了复用的目的而去开发一个新的组件，把原来的组件改写成这个可复用组件再加一些适配代码。

现在，终于可以为每个模块都指定一个明确的“责任人”，而不再“九龙治水”了。同时，团队分工也得到了优化，少量高手负责开发可复用模块和核心模块，而新手则负责开发那些“只需要对自己负责”的模块，无论他写的代码质量怎样，至少这些问题不会扩大化。让每个人都人尽其才，这是老板最喜欢的状态了，也是你架构水平和管理水平的体现。

除了意外耦合的问题之外，多人开发的另一个挑战就是契约。

手工作坊式的开发可能不需要显式的契约，因为超级程序员“心怀宇宙”，了解整个系统的每一处关键点。但一旦人多了，这种方式的弊端就会显露出来了，毕竟人类的心灵并不相通。这时候，就需要显式地描述契约，并借助工具来保障这些契约没有被误解。

在 Angular 中，用来应对契约问题的主要基础设施是类型与测试。

Angular 借助 TypeScript 来对类型提供支持。通过类型，Angular 可以对接口契约进行一定程度的表达，比如我只希望你给我传一个数字而不希望是字符串，那么我就可以把这个参数定义为 number 类型，如果调用者传了字符串进来，那么在编译阶段就能发现并阻止这个错误。我们都知道，发现错误越早，解决它的代价就越低。当然，TypeScript 的类型系统远比这强大多了，而作为 TypeScript 从初生到成熟期间的主要实践者，Angular 可谓把 TypeScript 的特性发挥到了极致。这块的内容非常庞大，这里就不展开讲了。

不过，类型只能对契约进行一定程度的表达，但它无法表达“我要一个大于10小于1000的数字”之类的契约，这时候就需要不同层次的测试出手了。从最早的 AngularJS 开始，测试就一直是重中之重，更不用说重写后的 Angular 了。不但 Angular 本身代码的测试覆盖率很高，而且还对你写自己的测试提供了全方位的支持。你可以对不依赖 Angular 的独立函数/独立类进行极其快捷轻便的测试，也可以对依赖 Angular 的服务等进行很容易的测试，还可以对组件进行 midway 测试，以验证它是否正确的生成了 DOM 节点，而不必动用 E2E 测试等比较沉重的方式。同时，别忘了 Angular 中的服务、组件、管道等都是纯类（POJO），只是附加在它们上面的注解让它们分化成了不同的用途，却并没有改变它们作为 POJO 的本质，因此，你也可以把它们当做不依赖 Angular 的 POJO 进行极其快捷的测试。这么多层次的测试支持，可以让你完全不用担心“能不能测”的问题，只需要担心某种场景下哪种测试方式最合适的问题，而这其中的运用之妙存乎一心，就是你能发挥最大作用的地方。

多人开发中一个稍微容易但更加繁琐的问题是代码风格。

一个团队要想形成统一的代码风格，“写得像一个人一样”其实相当麻烦，特别是在团队中还有人员流动的情况下。光靠制订代码规范是没用的，《规范.exe》胜于《规范.txt》。业界的常规实践是靠 Check Style 工具，JS 的世界中通常叫 lint。但是 lint 不认识具体的框架，因此它能保障的只是与框架无关的代码规范，而对像 Angular 这样的大型框架就无能为力了。

Angular 开发组想到了这一点，他们专门做了一个用于 Angular 风格检查的工具，并且集成进了 Angular CLI 中。它所依据的规范就是 Angular 开发组提供的官方风格指南，有了这份风格指南，天南海北、中国外国、现在未来的 Angular 程序员就可以写出风格大致相同的代码了。当然，你们团队可以根据需要对风格检查的规则进行调整，不过，如果你们不是非常资深的 Angular 团队，建议还是按照默认值来做吧。

时间是系统最大的客户，也是系统最大的敌人。

一个只为打印一次 Hello, world! 而写的程序不需要任何工程化。但那些真正的系统势必会随着时间的推移而不断演化，使用时间越长，说明这个系统越被认可，但另一方面，使用时间越长，这个系统腐化的可能性也越高，直到最后“杀死”这个系统。

Angular 如何抵抗岁月的侵蚀呢？

首先，Angular 的设计原则是每个部件只关注一件事（关注点分离 SoC）。

从宏观上说，core 模块只负责提供依赖注入等基础设施，forms 模块只提供表单支持，router 模块只管路由等等。这样明确的划分开，就让 Angular 同时具有了大而全和小而美这两种看似矛盾的特征。小而美保证了系统可以只用尽可能少的 Angular 模块，从而避免其它模块发生更改时带来的影响。大而全则保证了系统即使只用 Angular 自身提供的模块都能做很多事，让第三方依赖所带来的侵蚀也最小化。

从微观上说，服务拆分出了那些具有全局性的、要在多个组件之间共享的状态和逻辑，但它完全不关心界面展现；组件只负责展现用户界面，它自己的部分逻辑可以委托给服务；指令只负责对 DOM 元素或现有组件的能力进行简单的扩展；管道只负责从模型数据到显示数据的转换。这样分工明确之后，需求变更带来的影响也被最小化了。

其次，Angular 的版本发布策略在保持稳定和拥抱变化之间做出了较好的平衡。

事实上，这种版本发布策略并不是 Angular 首创和独有的，它叫做语义化版本（semver）。NodeJS 以及最新的 Java 等重量级选手都遵循着这种策略。

语义化版本的要点是：只在主版本号变更时引入不向后兼容的改动，次版本号变更只增加能向后兼容的新特性，修订号变更只用能向后兼容的方式修 bug。这里的要点是“向后兼容”只会出现在主版本号变更时。

但采用这种发布策略对作者自身的工程化能力要求很高，一般的作者是不容易做到的，不过 Angular 开发组对这一点倒是信心十足，而且也一直在兑现着承诺。事实上，Angular 开发组所做的远不止这些: 它的每一个主版本都会完全兼容上一个主版本，只是会把不兼容的部分标记出来给你一个警告，让你有半年的时间可以从容修改它。同时，Angular CLI 中还提供了 update 命令，它可以帮助你自动把程序从上一个主版本升级到当前主版本，我曾用两分钟的时间把一个花了一个人月开发的 Angular 程序从 5.0 升级到了 6.0。

语义化版本带来的好处是可以兼顾发展生态圈与追逐新技术。生态圈喜欢稳定，谁也不希望自己写的库过半年就没法用了，就算真没法用了，也不希望自己花大量时间迁移，却只为了跟随新版本。但不断跟上前沿却是一项技术能拥有长久生命力的根本保障。而真正的工程化，两者都想要，一个也不能少。

最后，Angular 既老且新。

说它老，是因为 Angular 采用了大量有十几年二十几年历史的成熟技术，比如依赖注入、接口、注解等。这些全都是经历过岁月的重重考验的最佳实践，既然那么多年都没被侵蚀，可以预期，除非将来出现了革命性的变化，否则照样拿它们没办法。难能可贵的是，Angular 从未给它们发明新的名词进行包装，要知道，前端圈最喜欢做的事可能就是发明新名词了。

说它新，是因为 Angular 热情拥抱其它上下游技术，并积极追随标准。从最初选择与少年时期的 TypeScript 合作，到跟 RxJS 的深度整合，再到对 PWA 的第一时间支持，再到对 Web Components 等标准草案的紧紧追随，无不体现着 Angular 的开放和与时偕行的心态。

结语

工程化的本质目的就是支持团队化开发并帮助系统抵抗时间的侵蚀，而 Angular 展现出的这些工程之美，让我对 Angular 的未来充满信心。

周虽旧邦，其命维新！

=======2018-06-13更新========

评论区惊现神逻辑，电脑用户请翻到第三页参观 Phi6 的发言，手机用户就别浪费流量了。

前端三大开发框架-angularjs创始人之一Misko Hevery

播报文章

**[互联网前世今生](https://author.baidu.com/home?from=bjh_article&app_id=1620838007405198" \t "https://baijiahao.baidu.com/_blank)**

2019-06-27 20:57科技领域创作者

关注

Misko Hevery 出生在斯洛伐克，在那里他度过了幸福的童年。在13岁的时候，他 移民到美国，开始了他全新的求学生涯。因为没有从小在美国学习，所以在学校里面他学习新语言很慢，很难和他人进行沟通，不过慢慢地，他就掌握了新语言，不到半年的时间，他就可以和同学们进行愉快地进行沟通了。



Misko Hevery 从小就对计算机感兴趣，这得益于他的父亲。虽然他的父亲是个医生，但是他却计算机非常感兴趣。在 Misko Hevery还是小时候的时候，他的父亲就买了第一台电脑，那时候的他在父亲的帮助下已经可以使用BASIC,PASSCAL进行编程了。那时候的他已经表现出在计算机方面的才华。在参加一个计算机编程比赛中，因为他做的太好，被评委取消了资格，因为评委不相信那是他做的，从侧面我们也能看得出来，那时候的他写的程序已经堪比大人写的了。

因为从小对计算机的热爱，他大学自然选择了计算机系，在1999年，他毕业于罗切斯特理工学院。在2005年，他从圣克拉拉大学得到了MBA硕士学位。



Misko Hevery第一份工作是在一家咨询公司做顾问。在当顾问的时候，他接触了很多新事物，很多不同的客户，他也因此学到了很多东西。然而生来喜欢折腾的他，工作了一段时间就离开了公司，他加入了一家初创公司SiliconX，他很喜欢初创公司的氛围，在那里他结识了很多好朋友和学到了很多初创公司的经验。

后来的他先后去了硅谷很多公司， Intel、Xerox（施乐）、Sun和Adobe公司他都有工作过，在这些公司他主要从事数据库/后端方面的工作。一直换工作的他，总是想要找到最好的一份工作，后来他找到了，那就是谷歌。2005年他加入了谷歌，他开始了有趣的工作，他认为谷歌是他最待过最棒的公司，在那里可以通过兴趣创造一个更美好的世界。



让Misko Hevery被世界认识的是他在谷歌做的angular,当初angularjs只是一个业余项目。他只是想让web设计师也可以使用html创建出简单的应用。后来他发现angular可以做的更多，如今，angular已经成为前端三大开发框架，而它的前端mvc,mvvm,mvw如今看来是给前端插上了翅膀，让前端不但可以飞，而且飞得更高。甚至vue的创造也是吸收了angular数据绑定的思想。



可以说Misko Hevery的后端开发经历给他开发angular提供了帮助，正是有了后端的思想，他想让前端数据的操作也想后端一样方便。纵观如今的前端框架，你会发现很多框架的创造者都或多或少会有后端的开发经历。或许将来，全栈工程师会变得越来越多。也许这就是天下大事分久必合合久必分吧。