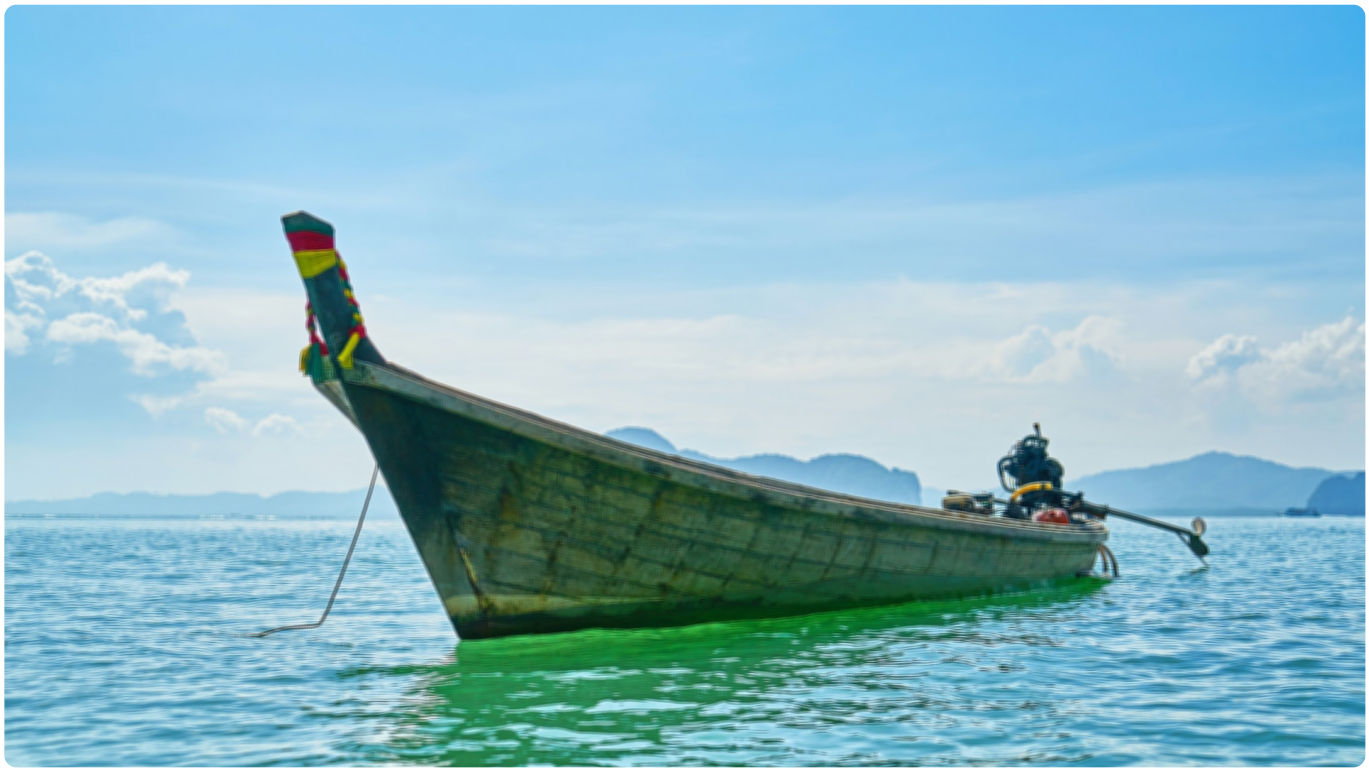
12 | 持续集成：你说的CI和我说的CI是一回事吗？

2019-11-07 石雪峰



讲述：石雪峰

时长 17:44　大小 16.26M

你好，我是石雪峰。今天我来跟你聊聊 CI。

之前，我曾应邀参加某公司的 DevOps 交流活动，他们质量团队的负责人分享了 DevOps 平台建设方面的经验，其中有一大半时间都在讲 CI。刚开始还挺好的，可是后来，我越听越觉得奇怪，以至于在交流环节，我只想提一个问题：“你觉得 CI 是个啥意思？”后来，为了不被主办方鄙视，话到嘴边我又努力憋回去了。

回来的路上，我就一直在思考这个问题。很多时候，人们嘴上总是挂着 CI，但是他们说的 CI 和我理解的 CI 好像并不是一回事。比如，有时候 CI 被用来指代负责内部工具平台建设的团队；有时候，CI 类似一种技术实践，间接等同于软件的编译和打包；有时候，CI 又成了一种职能和角色，指代负责版本的集成和发布的人。可见，CI 的定义跟 DevOps 一样，每个人的理解都千差万别。

可问题是，如果不能理解 CI 原本的含义，怎么发挥 CI 真正的价值呢？以 CI 的名义打造的平台又怎么能不跑偏，并且解决真正的问题呢？

所以，今天，就让我们一起重新认识下这个“最熟悉的陌生人”。

CI 是 Continuous Integration 的缩写，也就是我们熟悉的持续集成，顾名思义，这里面有两个关键的问题：集成什么东西？为什么要持续？要回答这两个问题，就得从 CI 诞生的历史说起了。

在 20 世纪 90 年代，软件开发还是瀑布模式的天下，人们发现，在很长一段时间里，软件是根本无法运行的。因为按照项目计划，软件的功能被拆分成各个模块，由不同的团队分别开发，只有到了开发完成之后的集成阶段，软件才会被真正地组装到一起。可是，往往几个月开发下来，到了集成的时候，大量分支合并带来的冲突和功能问题集中爆发，团队疲于奔命，各种救火，甚至有时候发现压根集成不起来。

我最初工作的时候，做的就是类似这样的项目。我们负责客户端程序的开发，到了集成的时候才发现，客户的数据库使用的是 Oracle，而我们为了省事，使用的是微软 Office 套件中的 Access，估计现在很多刚工作的年轻工程师都没听说过这个数据库，这就导致客户下发的数据没法导入到本地数据库中。结果，整整一个元旦假期，我们都在加班加点，好不容易赶工了一个数据中间层，这才把两端集成起来。

所以，软件集成是一件高风险的、不确定的事情，国外甚至有个专门的说法，叫作“集成地狱”。也正因为如此，人们就更倾向于不做集成，这就导致开发末端的集成环节变得更加困难，从而形成了一个恶性循环。

为了解决这个问题，CI 的思想应运而生。CI 本身源于肯特·贝克（Kent Beck）在 1996 年提出的极限编程方法（ExtremeProgramming，简称 XP）。顾名思义，极限编程是一种软件开发方法，作为敏捷开发的方法之一，目的在于通过缩短开发周期，提高发布频率来提升软件质量，改善用户需求响应速度。

不知道为什么，每次听到极限编程，我心中都热血沸腾。不管在任何时代，总有那么一群程序员走在时代前沿，代表和传承着极客精神，就像咱们平台的名字极客时间，就代表了不甘于平庸、追求极致的精神，特别好。

扯远了，让我们回归正题。极限编程方法中提出的实践，现在看来依然相当前沿，比如结对编程、软件重构、测试驱动开发、编程规范等，这些词我们都耳熟能详，但是真正能做到的却是凤毛麟角。其中还有一个特别有意思的实践规范，叫作每周 40 小时工作制，也就是一周工作 5 天，每天工作 8 小时。联想到前些日子在网络上引发激烈争论的“996”，就可以看出，极限编程方法在国内的发展还是任重而道远啊。

当然，在这么多实践中，持续集成可以说是第一个被广泛接受和认可的。

关于 CI 的定义，我在这里引用一下马丁·福勒（Martin Fowler）的一篇博客中的内容，这也是当前最为业界公认的定义之一：

> CI 是一种软件开发实践，团队成员频繁地将他们的工作成果集成到一起（通常每人每天至少提交一次，这样每天就会有多次集成），并且在每次提交后，自动触发运行一次包含自动化验证集的构建任务，以便尽早地发现集成问题。

CI 采用了一种反常规的思路来解决软件集成的困境，其核心理念就是：越是痛苦的事情，就要越频繁地做。很多人不理解为什么，举个例子你就明白了。我小时候身体非常不好，经常要喝中药，第一次喝的时候，每喝一口都想吐，可是连续喝了一个星期之后，我发现中药跟水的味道也没什么区别。这其实是因为人的适应力很强，慢慢就习惯了中药的味道。对于软件开发来说，也是这个道理。

如果开发周期末端的一次性集成有这么大的风险和不确定性，那不如把集成的频率提高，让每次集成的内容减少，这样即便失败，影响的也仅仅是一次小的集成内容，问题定位和修复都可以更加快速地完成。这样一来，不仅提高了软件的质量，也大大降低了最后阶段的返工所带来的浪费，还提升了软件交付效率。

你可能会说，这个道理我也懂啊，我们的持续集成就是这样的。别急，我们一起来测试一下。

假如你认为自己所在的项目和团队在践行 CI，那么你可以思考 3 个问题，看看你们是否做到了。

> 1. 每一次代码提交，是否都会触发一次完整的流水线？

> 2. 每次流水线是否会触发自动化的测试环节？

> 3. 如果流水线出现了问题，是否能够在 10 分钟之内修复？

我曾在现场做过很多次这个测试，如果参与者认为做到了，就会举手表示；如果没有做到，就会把手放下。每次面对一群自信满满的 CI“信徒们”，三连问的结果总会让人“暗爽”，因为最开始几乎所有人都会举手，他们坚信自己在实践持续集成。但接下来，我每问一个问题，就会有一半的人把手放下，坚持到最后的人寥寥无几，这几个人面对周边人的目光，内心也开始怀疑起来，如果我再适时地追问两下，基本就都放下了。

这么看来，CI 听起来简单易懂，但实施起来并没有那么容易。可以说 CI 涵盖了三个阶段，每个阶段都蕴含了一组思想和实践，只有把这些都做到了，那才是真正地在实施 CI。接下来，让我们逐一看下这三个阶段。

第一阶段：每次提交触发完整的流水线

第一个阶段的关键词是：**快速集成**。这是对 CI 核心理念的最好诠释，也就是集成速度做到极致，每次变更都会触发 CI。

当然，这里的变更有可能是代码变更，也有可能是配置、环境、数据变更。我之前强调过，**要将一切都纳入版本控制**，这样，所有的元数据变更都会被版本管理系统捕获，并通过事件或者 Webhook 的方式通知持续集成平台。

对于现代的持续集成平台，比如大家常用的 Jenkins，默认支持多种触发方式，比如定时触发、轮询触发或者 Webhook 触发。那么，**如果想做到每次提交都触发持续集成的话，首先就需要打通版本控制系统和持续集成系统**，比如 GitLab 和 Jenkins 的集成，网上已经有很多现成的材料，大家照着操作一般都不会有太多问题。但是，只要打通两个系统就足够了吗？显然没有这么简单。实施提交触发流水线，还需要一些前置条件。

1.**统一的分支策略**。

既然 CI 的目的是集成，那么首先就需要有一条以集成为目的的分支。这条分支可以是研发主线，也可以是专门的集成分支，一旦这条分支上发生任何变更，就会触发相应的 CI 过程。那么，可能有人会问，很多时候开发都是在特性分支或者版本分支上进行的，难道这些分支上的提交就不要经过 CI 环节了吗？这就引出了第 2 个前置条件。

2.**清晰的集成规则。**

对于一个大中型团队来说，每天的提交量是非常惊人的，这就要求持续集成具备足够的吞吐率，能够及时处理这些请求。而对于不同分支来说，持续集成的步骤和要求也不尽相同。不同分支的集成目的不同，相应的环节自然也不相同。

比如，对于研发特性分支而言，目的主要是快速验证和反馈，那么速度就是不可忽视的因素，所以这个层面的持续集成，主要以验证打包和代码质量为主；而对于系统集成分支而言，它的目的不仅是验证打包和代码质量，还要关注接口和业务层面的正确性，所以集成的步骤会更加复杂，成本也会随之上升。所以，**根据分支策略选择合适的集成规则，对于 CI 的有效运转来说非常重要。**

3.**标准化的资源池。**

资源池作为 CI 的基础设施，重要性不言而喻。

首先，**资源池需要实现环境标准化**，也就是任何任务在任何节点都具备可运行的能力，这个能力就包括了工具、配置等一系列要素。如果 CI 任务在一个节点可以运行，跑到另外一个节点就运行失败，那么 CI 的公信力就会受到影响。

另外，**资源池的并发吞吐量应该可以满足集中提交的场景，可以动态按需初始化的资源池就成了最佳选择。**当然，同时还要兼顾**成本因素**，因为大量资源的投入如果没有被有效利用，那么造成的浪费是巨大的。

4.**足够快的反馈周期。**

越是初级 CI，对速度的敏感性就越强。一般来讲，如果 CI 环节超过 10～15 分钟还没有反馈结果，那么研发人员就会失去耐心，所以 CI 的运行速度是一个需要纳入监控的重要指标。对于不同的系统而言，要约定能够容忍的 CI 最大时长，如果超过这个时长，同样会导致 CI 失败。所以，这就需要环境、平台、开发团队共同维护。

你看，一套基本可用的 CI 所依赖的条件远不止这些，核心还是为了能够在最短的时间内完成集成动作并给出反馈。如果你们公司已经实现了代码提交的 CI，并且不会有大量失败和排队的情况发生，那么，恭喜你，第一阶段就算通过了。

第二阶段：每次流水线触发自动化测试

第二个阶段的关键词是：**质量内建**。关于质量内建，我会在专栏后面的内容中详细介绍。实际上，CI 的目的是尽早发现问题，这些问题既包括构建失败，也包括质量不达标，比如测试不通过，或者代码规约静态扫描等不符合标准。

我见过的很多 CI 都是“瘸腿”CI，因为缺失了自动化测试的能力注入，或者自动化测试的能力很差，基本无法发现有效问题。这里面有几个重要的关注点，我们来看一下。

1.**匹配合适的测试活动**。

对于不同层级的 CI 而言，同样需要根据集成规则来确定需要注入的质量活动。比如，最初级的提交集成就不适合那些运行过于复杂、时间太长的测试活动，快速的代码检查和冒烟测试就足以证明这个版本已经达到了最基本的要求。而对于系统层的集成来说，质量要求会更高，这样一来，一些接口测试、UI 测试等就可以纳入到 CI 里面来。

2.**树立测试结果的公信度**。

自动化测试的目标是帮助研发提前发现问题，但是，如果因为自动化测试能力自身的缺陷或者环境不稳定等因素，造成了 CI 的大量失败，那么，这个 CI 对于研发来说就可有可无了。所以，**我们要对 CI 失败进行分类分级，重点关注那些异常和误报的情况，并进行相应的持续优化和改善**。

3.**提升测试活动的有效性**。

考虑到 CI 对于速度的敏感性，那么如何在最短的时间内运行最有效的测试任务，就成了一个关键问题。显然，大而全的测试套件是不合时宜的，只有在基础功能验证的基础上，结合与本次 CI 的变更点相关的测试任务，发现问题的概率才会大大提升。所以，根据 CI 变更，自动识别匹配对应的测试任务也是一个挑战。

当你的 CI 已经集成了自动化验证集，并且该验证集可以有效地发现问题，那么恭喜你，第二阶段也成功了。但这并不是“一锤子买卖”，毕竟，由于业务需求的不断变化，自动化测试要持续更新，才能保证始终有效。

第三阶段：出了问题可以在第一时间修复

到现在为止，我们已经做到了快速集成和质量内建，说实话，利用现有的开源工具和框架快速搭建一套 CI 平台并不困难，**真正让 CI 发挥价值的关键，还是在于团队面对持续集成的态度，以及团队内是否建立了持续集成的文化**。

硅谷的很多公司都有一种不成文的规定，那就是员工每天下班前要先确认持续集成是正常的，然后再离开公司，同时，公司也不建议在深夜或者周末上线代码，因为一旦出了问题，很难在第一时间修复，造成的影响难以估计。

其实，很多企业并不知道他们花费大量人力、物力建设 CI 的平均修复时长是多少，也缺乏这方面的数据统计。就现状而言，有些时候，他们可以做到在 10 分钟内修复，而有些时候就需要几个小时，原因可能是负责人出去开会了，或者是赶上了午休的时间。

当然，也有一些企业质疑 10 分钟这个时间长度，因为软件项目的特殊性，很有可能每次集成周期就远大于 10 分钟。如果你也是这样想的，那你可能就误解 CI 的理念和初衷了，毕竟我也不相信马丁·福勒能够保证在 10 分钟内修复问题。在这么短的时间里，人为因素其实并不可控，所以，**人不是关键，建立机制才是关键**。

什么是机制呢？**机制就是一种约定，人们愿意遵守这样的行为，并且做了会得到好处**。对于 CI 而言，保证集成主线的可用性，其实就是团队成员间的一种约定。这不在于谁出的问题谁去修复，而在于我们是否能够保证 CI 的稳定性，足够清楚问题的降级路径，并且主动关注、分析和推动问题解决。

另外，团队要建立清晰的规则，比如 10 分钟内没有修复则自动回滚代码，比如当 CI “亮红灯”的时候，团队不再提交新的代码，因为在错误的基础上没有办法验证新的提交，这时需要集体放下手中的工作，共同恢复 CI 的状态。

只有团队成员深信 CI 带给团队的长期好处远大于短期投入，并且愿意身体力行地践行 CI，这个“10 分钟”规则才有可能得到保障，并落在实处。

总结

在这一讲中，我们回顾了 CI 诞生的历史和 CI 试图解决的根本问题。同时，我们也介绍了 CI 落地建设的三个阶段和其中的核心理念，即快速集成、质量内建和文化建立。

最后，我特别想再提一点，很多人经常会把工具和实践混为一谈，一旦结果没有达到预期，就会质疑实践是否靠谱，工具是否好用，很容易陷入工具决定论的怪圈。实际上，CI 的核心理念从未有过什么改变，但工具却一直在升级换代。工具是实践的载体，实践是工具的根基，单纯的工具建设仅仅是千里之行的一小步，这一点，我们必须要明白。

思考题

可以说，一个良好的 CI 体现了整个研发团队方方面面的能力，那么，你对企业内部实践 CI 都有哪些问题和心得呢？

欢迎在留言区写下你的思考和答案，我们一起讨论，共同学习进步。如果你觉得这篇文章对你有所帮助，欢迎你把文章分享给你的朋友。

上一篇　11 | 分支策略：让研发高效协作的关键要素

下一篇　13 | 自动化测试：DevOps的阿克琉斯之踵

精选留言 (10)

写留言

Robert小七
2019-11-07

对于老师问的三个问题，我们公司可以算是做到了！但对于每一次提交触发集成，想请教下是否意味着每次提交都会触发持续集成流水线？如果是这样，也就是说持续集成是包含持续部署的？因为构建只能运行单元测试，但是大量的自动化还是需要先部署服务的！再就是老师所说的每次提交集成触发这个机制，很想请教下老师，您推荐监控主线分支还是特性分支？或者说有专门的集成分支？如果是主线分支，按照特性分支策略，其实已经...

展开

2

雷霹雳的爸爸
2019-11-07

赶巧了我是12讲发出来才看的11讲，刚好一起看的，所以难免想起好像在哪儿有一个论调，翻了下笔记，https://www.continuousdeliveryconsulting.com/blog/organisation-pattern-trunk-based-development/，这哥们提到Jez humble在 https://book.douban.com/subject/26702829/ 将TBD称为CI的synonymous

...

展开

> 作者回复: 你好，很多理念，比如TBD，CI，这些都是一以贯之的哈，关于CI的三个测试问题，你可以参考下这篇文章： https://martinfowler.com/bliki/ContinuousIntegrationCertification.html

2

1

桃子-夏勇杰
2019-11-10

周六雷涛老师在我们公司做devops培训，果然也问了这3个问题，真的是灵魂拷问啊。

石子头
2019-11-08

老师，有什么相对权威的DevOps培训可以推荐吗？网上很多，比较难分辨真正好的，能指明个方向也行的，谢谢！

展开

作者回复: 你好，又是一个让我比较纠结的问题啊，说实话，要看你自己对于培训的心理预期，如果你现在对于DevOps还属于门外汉的阶段，我觉得参加一个培训建立全局认知，多理解一些案例，是有帮助入门的，但是如果说入门之后，只靠培训就比较困难了，因为你面临的实际问题，并不是培训的主题，培训更多的是一些客观的技能提升，比如某个工具，语言，技能等等，还是要跟高手一起共事和思考才行，如果你们公司在做一些这方面的转型工作，你可以多关注一下参与一下，在解决实际问题的过程中积累经验。


Jxin

2019-11-07

ci是一套理念，随着它的推行会催生出工具平台和文化。工具平台的重心在于提高效率，也就是所有可以被自动化的操作最终都由自动化来解决，使得问题可以close，人力可以得到解放。文化则是倒逼着人们制定并执行ci的流程，同时开始切割集成工作，缩小每次集成的力度提高灵活性，规范每次提交的原子性提高版本管理和问题识别的精准度，以及核心流程单元测试的编写close风险。

展开

作者回复: 嗯，我觉得很多时候我们都在就事论事，比如ci有问题就谈ci，其实ci也是整个团队能力的体现，需求拆分是否合理，分支策略是否合适，构建速度是否足够快，自动化测试能力有没有，质量门禁的指标和规则是哪些，通知机制是否合理，反馈是否及时，想把ci做好，真不是工具的问题。


睡觉

2019-11-07

之前的公司，采用了gitlab+gerrit+jenkins组件了一条CI的流水线，做的也比较简单，每个人创建自己的分支进行提交，触发jenkins进行编译与自动化测试(测试没人写，只进行了简单的编译检查)。然后在gerrit上进行review，通过代码才能进master。现在的公司没有这套流水线，感觉特别low

展开

作者回复: 的确，这些也基本上属于研发标配了，有个小问题，你们为什么要结合Gitlab和Gerrit呢，我们公司也有这样的使用的，原因是公司要求统一使用Gitlab，实际上如果看重Gerrit强大的Review能力的话，为什么不直接使用Gerrit管理代码呢，毕竟多套环境不稳定因素也会增多，同步镜像备份都是麻烦事，不知道你们是怎么考量的呢?

Bun
2019-11-07

有什么好的自动化测试工具推荐下

展开

作者回复: 你好，不知道你具体指哪种测试类型呢，比如常见的单元测试、接口测试、UI测试，到典型的非功能性测试，如性能，兼容性，稳定性，安全性，再到端到端测试，压力测试等等，即便一种测试类型也有多重模式，比如面向接口的，面向契约的都不相同哈。
我推荐你看下专栏特别放送（下），里面有一幅DevOps工具罗盘图，里面罗列的典型的工具哈，如果对某一个具体的工具有问题，也欢迎给我留言，谢谢。

1

阿硕
2019-11-07

石老师，您好，关于每次提交触发构建中的方式，应该如何选择呢？能否提供下最佳实践参考，谢谢。

作者回复: 你好，我想这个问题首先取决于你们当前的业务形态，比如典型的包括APP类型，服务端类型，智能硬件类型和商业软件类型。为什么说跟业务形态有关，这就牵扯到了分支策略和分支模型，提交构建首先要明确监控哪条分支上的提交，在最开始我的建议是如果资源不够充裕或者集成时长普遍超过15分钟的话，可以优先在主干分支上跑起来，再逐步扩展到下游分支。从集成的操作来说，可以先跑一个最小集，如下载代码，构建，然后再逐步拓展到静态检查，自动化测试等，这样循序渐进的来进行。至于工具层面，最典型的就是Jenkins跟版本控制系统打通，你可以在Jenkins中搜索对应系统的插件，大多可以支持，你也可以留言说明你们现在使用的工具，我可以找到一些更详尽的资料给你哈。

1

manatee
2019-11-07

请问老师前端项目在ci中的测试有什么好的联系吗

展开

作者回复: 你好，这个必须有，我们团队自己就在使用前后端分离的开发模式，对于Vue.js来说单测可以使用Jest和Mocha，覆盖率采用的Istanbul，在以前的演示项目里面也用到过PhantomJS和CasperJS，当然各种Lint工具对于代码风格也很有用，具体看你的使用场景啦。

leslie

2019-11-07

老师在文章中的"工具是实践的载体，实践是工具的根基，单纯的工具建设仅仅是千里之行的一小步"其实深有感悟，在此基础上简单的扩充一下"方法/理论是实践的载体，实践是工具的根基，单纯的方法/理论建设仅仅是千里之行的一小步"。

落地性和实践性确实非常成问题：可能自己中小企业和大型企业的OPS的工作时间对半吧，期间其实碰过不少企业做自动化，可是2009就普遍使用的SVN其实很多时候都简单...

展开

作者回复: 呵呵，我觉得很多时候，文章就是抛砖引玉，而留言区的互动反而更加精彩，因为能看到不用形态、规模、岗位的同学对于同一个问题的不同思考，以及面对的困难，和自己的观点。虽然目标是确定的，但是实践的路径却有很多条，所以不要迷恋一家之言，有独立思考能力才更加重要，感谢leslie的补充和回复。就像一个心理咨询师，他不会替你解决问题，更多的是帮助你解决问题，企业的外部顾问也是如此，很多时候的确需要一个外部的推动力，或者目标性更强的事情来推动团队的变革。