《代码英雄》第二季（4）：更好的失败

代码英雄讲述了开发人员、程序员、黑客、极客和开源反叛者如何彻底改变技术前景的真实史诗。

什么是《代码英雄》

Command Line Heroes

代码英雄是世界领先的企业开源软件解决方案供应商红帽（Red Hat）精心制作的原创音频播客，讲述开发人员、程序员、黑客、极客和开源反叛者如何彻底改变技术前景的真实史诗。该音频博客邀请到了谷歌、NASA 等重量级企业的众多技术大牛共同讲述开源、操作系统、容器、DevOps、混合云等发展过程中的动人故事。

本文是《代码英雄》系列播客第二季（4）：更好的失败的音频脚本。

导语：失败是探索时的心跳。我们会在尝试新事物时会多次跌倒。其中秘诀是放弃快速失败，取而代之的是，更好地失败。

本期节目关注在科技领域如何拥抱失败。（对于科技领域来说）以好奇和开放的态度来对待失败是过程中的一部分。Jennifer Petoff 分享了 Google 是如何建立起一种从失败中学习和改进的文化；Jessica Rudder 通过视角的转变，展示了拥抱错误如何能带来意想不到的成功。而 Jen Krieger 则介绍了敏捷框架如何帮助我们为失败做计划。

失败未必是终点。它可以是迈向更伟大事物中的一步。

00:00:00 - Saron Yitbarek：

如果你没有听过这个笑话 —— 两个工程师在编译他们的代码。新手举手喊道：“哇，我的代码编译好了！”；老手则会眯着眼睛喃喃道：“唔，我的代码居然编译好了”。

00:00:18：

如果你已经做过一段时间编程，当你开始思考失败这件事，对有些事情的看法可能就会有所不同。那些过去无法解决的问题，如今开始看起来像一个更大的解决方案中的一个正常组成部分。那些你曾经称之为“失败”的东西，现在看起来像是变相的成功。

你开始希望你的代码无法通过编译。你希望可以一路摆弄和实验它们，调试和修订和重构这些代码。

00:00:37：

你正在收听的是红帽公司的原创播客节目《代码英雄》。我是主持人 Saron Yitbarek。

老实说，那句“快速失败”(fail fast)的口号经常被用来作为通往成功的捷径。但是，如果我们不是告诉彼此加快速度并快速失败，而是鼓励彼此更好地失败呢？

00:01:20：

《代码英雄》的第二季将介绍的是开发工作中真实的体验：“当我们生活在代码中，到底感觉如何？又是如何变化的？这也是为什么我们要用一整集的时间来讨论失败，因为正是这些失败时刻促使我们适应它。我们称之为“失败”的东西，是进化的心跳，而开源开发者正在拥抱这种进化。当然，这说起来容易做起来难。

00:01:59：

想象一下，如果一首全新的莎士比亚的十四行诗被发现了。网络上会兴起一阵热潮，每个人都想去搜索它。但这时，有个小小的设计缺陷导致了所谓的“文件描述符耗尽”。这会造成一连串的失败。突然之间，这所有的流量都在越来越少的服务器之间流动。很快，在 Google 上的“莎士比亚”搜索崩溃了，并崩溃了一个多小时。

00:02:33：

现在，你丢掉了 12 亿次搜索查询。这是一场莎士比亚式的悲剧，所有的一切，在网站可靠性工程师（SRE）四处补救的同时上演。

00:02:45 - 配音：

还有你吗，布鲁特？那就倒下吧，凯撒！

00:02:54 - Saron Yitbarek：

不好意思，我打断一下。但上面说的这个莎士比亚事件其实并不存在。事实上，这是一本书《SRE：Google 运维解密》中一系列灾难性场景的一部分。从这本书中学到的重要的一课就是你必须超越灾难本身。这就是我的意思。

00:03:13：

在这个莎士比亚的例子中，当流量被集火到一个被牺牲的单独集群时，这个死亡查询问题就解决了。这为团队赢得了扩充容量的足够时间。但你不能就此止步。尽管这个问题很糟糕，但解决它并不是真正的重点所在。因为失败不一定以痛苦告终，失败也可以引导你的学习。

00:03:38 - Jennifer Petoff：

嗨，我是 Jennifer Petoff。

00:03:41 - Saron Yitbarek：

Jennifer 在谷歌工作。她是 SRE（site reliability engineering 站点可靠性工程）团队的高级项目经理，领导谷歌的全球 SRE 教育计划，她也是这本描述了莎士比亚场景的书的作者之一。对于 Jennifer 来说，钻研这样的灾难才能使事情变得更好，但前提是你需要有一个拥抱错误和意外的文化。

00:04:08：

所以，让我们再拿莎士比亚举例子。有一个直接的办法，减少负载可以让你免于这种连锁故障。但，真正的工作将在一切恢复正常之后开始，重点在于事后分析报告。

00:04:25 - Jennifer Petoff：

事件解决后，我们会创建一个事后分析报告。谷歌的每一个事件都需要有一个事后分析和相应的行动项目，以防止将来再次出现问题，以及更有效地检测和缓解未来出现类似事件或整类问题的可能。

00:04:42 - Saron Yitbarek：

这是一个关键的区别。不仅仅是解决这个特定事件，而是看到这个事件告诉你的一系列问题。真正有效的事后分析，不只是告诉你昨天哪里出现了问题。而是让你对今天所做的工作以及对未来的计划有深刻的见解。这种更广泛的思想，灌输了对所有这些事故和失败的尊重，使它们成为日常工作生活中至关重要的一部分。

00:05:12 - Jennifer Petoff：

所以，一个真正好的事后分析不仅仅要解决手头的单个问题，它还解决了整个问题。事后分析的重点是什么地方作对了，什么地方做错了，在何处幸运的解决了问题，以及可以采取哪些优先行动来确保这种情况不会再次发生。如果你不采取行动，历史必将重演。

00:05:32 - Saron Yitbarek：

在谷歌，人们关注的是无责任(blameless)的事后分析(post-mortems)，这就造成了根本的不同。如果出了问题而没有人要责怪，那么每个人都可以诚实地挖掘错误，真正地从错误中吸取教训，而不必掩盖任何线索，也不必争吵。这些无责任的事后分析已经成为谷歌文化的一个重要组成部分，其结果是一个不必害怕失败的工作场所。这是一种正常情况。

00:06:01 - Jennifer Petoff：

谷歌如何看待失败？100% 的在线时间是一个不可能的目标。如果你认为这是可以实现的，那就是在自欺欺人。所以，失败会发生只是时间和方式的问题。在谷歌，失败是值得庆祝的，因为我们可以从中吸

取教训，而事后分析也会在团队中广泛分享，以确保学到的东西可以广泛使用。

00:06:23 - Jennifer Petoff：

错误是不可避免的，但你永远不想以同样的方式失败两次。犯错是人之常情，但反复犯错是可以避免的。

00:06:34 - Saron Yitbarek：

听到 Jennifer 讨论失败的方式，这真是太有趣了，因为就像她在犯那些错误一样。比如，当事情出错的时候，这意味着你已经走到了一个可以挖掘价值的地方。

00:06:50 - Jennifer Petoff：

你会现场处理这种情况，但事后花时间把发生的事情写出来，让别人可以从中学习。发生任何事件时，你都需要付出代价。如果你不写出事后分析，并真正从这个经验中吸取教训，你就不会重新收回解决问题所花费的成本。在我看来，这是至关重要的一课。在谷歌，我们坚信无责任文化。你不会因为指责别人而获得任何好处，那只会让人们去掩盖失败，而失败，总是会发生。

00:07:27 - Saron Yitbarek：

这里非常重要的一点是，要记住 Jennifer 之前说过的一些话，没有错误的工作是一种幻想，总会有出错的地方。归根结底这是思想的转变。我们可以抛弃那种认为只有一个单一的、可确定的最终目标，即一切最终都会按照我们想象的方式发展的想法。我们没有人试图达到这一目标，事实证明，这是非常强大和积极的东西。

谷歌拥抱失败的做法很有意义。超级实用。但我想知道，这只是口头上的么？我们是否有一些具体的让事情变得更好的失败例子，或者这只是一种当我们进行第 200 次编译时，让我们感觉更好的一种方法。

00:08:26：

事实证明，有人可以回答这个问题。

00:08:29 - Jessica Rudder：

我的名字叫 Jessica Rudder。我是 Github 的软件工程师。

00:08:33 - Saron Yitbarek：

Jessica 在 Github 经历过失败。从某种意义上说，这是一个失败的舞台，在这一过程中，她收集了一些关于失败是通往巨大成功的故事。比如这个：

00:08:50 - Jessica Rudder：

90 年代有个游戏开发公司正在开发一款全新的游戏。从本质上说，这是一款赛车游戏，但他们的转折之处在于将其改为街头赛车。所以当赛车手在街道上飙车时，他们不仅是在互相飙车，而且他们也是与在追赶他们的警车（非玩家角色）赛车。如果一辆警车抓住了你，它会让你靠边停车，然后你就输掉了比赛。然后他们把这些代码衔接起来，然后开始运行，他们发现他们完全调校错了算法：警车只是尖叫着从侧街冲出来，直接撞向玩家的车，而不是追赶玩家的车。

00:09:37：

所以这里简直是一团糟。他们想，不要惊慌，让我们继续前进，看看人们如何看待它的，这样我们就知道该怎么调整算法了。所以他们把它交给了游戏测试人员，他们发现游戏测试人员在逃离警察并试图躲避被这些流氓暴力警车抓捕的过程中获得了更多乐趣。而事实上，它是如此的有趣，以至于开发团队改变了他们为游戏打造的整个理念。

00:10:17 - Saron Yitbarek：

你能猜出这是怎么回事吗？

00:10:21 - Jessica Rudder：

所以我们才有了《侠盗猎车手（Grand Theft Auto）》。我的意思是，它确实是有史以来最畅销的电子游戏，它能存在的全部原因都是因为当时他们没有使用正确的算法时所导致的失误，他们想，好吧，让我们来试试；看看我们得到了什么，看看我们能从中学到什么。

00:10:41 - Saron Yitbarek：

很神奇吧？但这里有个技巧，《侠盗猎车手》团队在遭遇失败时必须保持宽容；他们必须保持好奇心。

00:10:52 - Jessica Rudder：

所以，如果这些开发者没有开放的思想，并决定从这个错误中去学到什么，我们将永远不会有《侠盗猎车手》，我们只能玩一些无聊的、普通的街头赛车游戏了。

00:11:07 - Saron Yitbarek：

让我们再就游戏主题讨论一分钟，类似的事情也发生在《寂静岭 Silent Hill》的制作过程中。这是一个大型的、3A 级的大制作游戏。但他们遇到了严重的弹出问题。局部景观的处理速度不够快，因此突然之间，你会突然发现一堵墙或一条小路突然冒出来。这是一个破坏性的问题，而且他们的开发已经到非常后期。他们是怎么做的？完全放弃游戏，举手投降？还是将错就错。

00:11:42 - Jessica Rudder：

他们所做的就是让这个世界充满了非常浓郁、诡异的雾气。因为事实证明，雾对处理器来说非常容易渲染，而且不会有任何延迟。而且另外，雾使你看不到远处的东西，所以在现实中，那些建筑物仍然会突然出现，但由于雾遮挡了你的视线，你看不到它们。所以当它们进入视野时，它们已经被渲染了，看起来它们是从雾中出来的。

00:12:15 - Saron Yitbarek：

雾是变得如此受欢迎，它基本上被认为是《寂静岭》系列中的一个特点。它限制了玩家的视野，使游戏变得更加恐怖。甚至当处理器的速度快到不需要再掩盖那些弹出的时候，他们也保留了雾气。

00:12:33 - Jessica Rudder：

你无法在没有雾的情况下玩《寂静岭》。而这些雾最初所做的一切都是在掩盖一个错误。

00:12:40 - Saron Yitbarek：

我喜欢这个故事！他们拥抱失败而不是逃避失败，从而挽救了一个重大的发展。这条关于不怕失败的原则也适用于个人的小事，而不仅仅是全公司的决策。从容面对失败是我们一点一点地变得更好的方法。

00:13:01 - Jessica Rudder：

很多时候人们脑子里想的太多了，他们认为失败意味着我不擅长某样东西。并不是代码坏了我还不知道如何修复它，而是“我不知道如何编写 JavaScript”。而且，你永远不会通过说“我不知道如何编写 JavaScript”来学习所需的知识。但是如果你能确定，“哦，我不知道如何在 JavaScript 中实现这个循环”，那么你可以通过 Google 找到答案，而且效果很好。我是说，你仍然需要努力，但你遇到的麻烦会少的多。

00:13:36 - Saron Yitbarek：

因此，无论你是新开发人员还是大型工作室的负责人，我们的错误将我们推向更大的领域，那些实验，那些失败，那些英勇的尝试，它们占据了旅程的大部分。在我所熟悉和喜爱的开源社区里，这是最真实的情况了。失败在开源中可能是一件美好的事情，这就是我们接下来的故事。

00:14:14：

我们在前面看到了失败是如何带来惊喜 —— 那些我们甚至不知道自己想尝试的事情。在最好的情况下，开源开发文化正好符合这一点。它让失败变得正常。为了理解这种愿意失败的想法是如何被引入开源开发的，我和 Jen Krieger 聊了聊。她是 Red Hat 的首席敏捷架构师。我们讨论了对开源失败的态度，以及这些态度是如何塑造无限可能的。请听：

00:14:47：

我想谈谈这个口号，我觉得这也许是一个很好的表达方式。“fail fast and break things 快速失败，打破现状”，这几乎是为我们社区所设计的一个巨大的召集口号。你怎么看？

00:15:04 - Jen Krieger：

我对此有很多想法。

00:15:06 - Saron Yitbarek：

我也觉得你会有。

00:15:06 - Jen Krieger：

快速失败，在失败中前进，所有这些都是一个意思。所以，在我刚刚参加工作的时候，我在一家没有失败空间的公司工作。如果你做错了什么事情，你就可以准备辞职了。任何人都不能做错事，没有任何空间、途径允许你犯错。这令人们困扰。你绝对没有失败的余地，导致我们几乎陷入一场文化运动。愿意的话，这会催生出一个很棒词 —— 敏捷，以及催生出另一个很棒的词 —— DevOps。当我看到这些词的时候，我看到的是我们只是要求团队做一系列非常小的实验，帮助他们修正方向。

00:16:02：

这是个，哦，你已经做出了选择，而这实际上是一件积极的事情。你可能会做一个冒险的决定，然后你赢了，因为你做出了正确的决定。或者反之，就是你做了错误的决定，然后你明白了，那不是正确的方向。

00:16:18 - Saron Yitbarek：

是的，这是有道理的。所以，当你把“快速失败，打破现状”当成这个运动的时候，感觉在如何失败，如何以正确的方式失败上还是有一些方式，有一些最佳的实践的。那么，如何以一种正确的方式失败，有哪些最佳实践和原则呢？

00:16:44 - Jen Krieger：

我总是喜欢告诉工程师，他们需要尽早和尽可能多地破坏构建。如果他们正在破坏他们的构建，并且他们意识到他们已经破坏了构建，他

们在当下还有机会真正修复它。而这一切都围绕着“反馈循环”这个概念，并确保你在工作中得到的反馈循环尽可能小。

00:17:08：

所以在开源开发中，我提交了一个补丁，然后有人说，“出于这九个原因，我不会接受你的补丁”，或者“我认为你的补丁很棒，继续吧”。或者，你提交了一个补丁，但是机器人告诉你它失败了，因为它没有正确构建。有各种不同类型的反馈。

00:17:25：

然后在开源开发中，你可能会遇到更长的反馈循环，你可能会说，“我想设计这个新功能，但我不确定所有的规则应该是什么。有人能帮我设计吗？”因此，你进入了一个漫长的过程，在这个过程中，你要进行长时间和详细的对话，而人们参与进来，提出最好的想法。

00:17:45：

所以有各种各样的反馈循环可以帮助你完成这个。

00:17:50 - Saron Yitbarek：

Jennifer 认为，每个公司的反馈循环看起来都不一样。它们是可定制的，人们可以使它们以 100 种不同的方式工作。但重点是，她甚至没有把它们称为失败或错误。她只是称它们为“反馈循环”。这是一个有机系统，这是一种思考整个过程的健康方式。

00:18:11：

与此同时，对这些小毛病的另外一种态度却产生了完全相反的效果。

00:18:18 - Jen Krieger：

有些组织所做的事情是完全错误的。

00:18:23 - Saron Yitbarek：

嗯是啊。

00:18:24 - Jen Krieger：

让你的领导团队（或者，在一个很高的层面上，比如组织）认为，羞辱做错事情的人，或者在绩效结果方面灌输恐惧；就像是，“如果你工作做得不好，就拿不到奖金”或者“如果你工作做得不好，我会把你列入绩效计划。”这些都是会产生敌意的事情。

00:18:50 - Saron Yitbarek：

她描述的是一个不正确的失败。不能接受失败就是失败。她也在呼应 Jennifer Petoff 的态度，对吧？就是我们在这集开头提到的那个无责任的事后分析？

00:19:07：

是的，这很有趣。就好像如果我们在如何一起工作上要求更严格一点，或者只是更用心，更有目的性的在一起工作，我们几乎就会被迫在失败中表现得更好。

00:19:23 - Jen Krieger：

是的。有一些公司已经学会了这一点，而且他们很久以前就学会了，丰田就是一个很好的例子，它接受了这种不断学习和改进的理念，这是我在其他公司很少看到的。就是这样一种想法，任何人在任何时候都可以指出某些东西不能正常工作。不管他们是谁，在公司的哪个级别。在他们的文化中，认为这是对的。这种持续学习和改进的环境，我想说，是一种领先的实践，这是我希望公司能够做到的事情，能够适应失败并允许它发生。

00:20:06 - Saron Yitbarek：

嗯，没错。

00:20:07 - Jen Krieger：

如果你问的是为什么事情进展不顺利，而不是指责或试图隐藏事情，或责怪别人，这就会造成完全不同的情况。那就是改变对话方式。

00:20:23 - Saron Yitbarek：

这很有趣，因为你之前提到过“快速失败，打破现状”这句话是这种文化，这种对过去做事方式的反击。 但这听起来似乎是一种口头禅，也许也创造了一种在公司内部、技术团队内部的不同的团队工作方式。再给我讲讲这个问题，它是如何改变了开发人员看待自己角色的方式，以及他们与公司其他人互动的方式?

00:20:55 - Jen Krieger：

我早期和工程师一起工作的时候差不多是这样的，工程师们都坐在一个小区域，他们互相交谈。他们从未真正与任何商业人士进行过交流。他们从来没有真正理解他们的任何需求，我们花了很多时间真正专注于成功所需的东西，而不一定是企业实际完成工作所需的东西。所以，它更像是，“我是一个工程师，我需要什么才能编写这个功能片段？”我观察到，今天在几乎每一个和我一起工作的团队中，对话方式已经发生了巨大的变化，“作为工程师我需要什么才能完成工作”变成了“客户是谁，或者用户需要什么才能真正感觉到这我做的这块功能对他们来说是成功的？他们如何使用产品？我该怎样做才能让他们更轻松？”

00:21:56：

很多这样的对话已经改变了，我认为这就是为什么如今公司在提供有意义的技术方面做得更好的原因。我还想说的是，我们发布的速度越快，我们就越容易知道我们的假设和决定是否真正实现了。所以，如果我们对用户可能想要什么做了假设，在此之前，我们需要等待，比如，一年到两年才能确定这是不是真的。

00:22:25：

而现在，如果你看看亚马逊或奈飞的模式，你会发现，他们每天会发布数百次假设的客户需求。他们从使用他们的应用程序的人们那里得到的反馈，会告诉他们他们是否在做用户需要他们做的事情。

00:22:46 - Saron Yitbarek：

是的，这听起来需要更多的合作，因为即使是你之前提出的关于构建、破坏构建、经常破坏它的建议，这就需要工程团队或开发人员与

DevOps 保持步调一致，以便他们能够破坏它，并了解尽早发布并经常发布是什么样子的。听起来这需要双方更多的合作。

00:23:15 - Jen Krieger：

是的，对于拥有敏捷教练这个头衔的人来说，或者以我作为首席敏捷架构师看来，总是很有趣，因为《敏捷宣言》的初衷是让人们从不同的角度来考虑这些事情。我们通过开发和帮助别人开发来发现更好的开发软件的方法。它确实是敏捷所要做的的核心、根本和基础。因此，如果你将 10 年，15 年以上的时间快速推进到 DevOps 的到来，并坚持我们需要持续进行集成和部署。我们有监控，我们开始以不同的方式思考如何将代码扔出墙外。

00:23:56：

所有这些东西都是我们最初开始讨论敏捷时应该想到的。

00:24:03 - Saron Yitbarek：

嗯。绝对是的。所以，不管人们如何实践这种失败的理念，我认为我们都可以接受失败，将失败规范化只是过程的一部分，是我们需要做的事情，是我们可以管理的事情，是我们可以用“正确的方式”做的事情，这是一件好事。它对开源有好处。跟我说说这个新运动的好处，这种接受失败是过程的一部分的新文化的一些好处。

00:24:36 - Jen Krieger：

看着这个过程发生是一件美妙的事情。对一个人来说，从一个他们害怕可能发生事情的环境，到一个他们可以尝试实验、尝试成长、尝试找出正确答案的环境。真的很高兴，就像它们已经盛开花朵。他们的士气提高了，他们真正意识到他们可以拥有的是什么，他们可以自己做决定，而不必等待别人为他们做决定。

00:25:05 - Saron Yitbarek：

失败即自由。啊，我喜欢! Jen Krieger 是红帽公司的首席敏捷架构师。

00:25:19：

并不是所有的开源项目都像 Rails、Django 或 Kubernetes 那样声名鹊起。事实上，大多数都没有。大多数都是只有一个贡献者的小项目，解决一小群开发人员面临的小问题的小众项目，或者它们已经被抛弃，很久没有人碰了。但它们仍然有价值。事实上，很多这样的项目仍然非常有用，可以被回收、升级，被其他项目蚕食。

00:25:54：

而另一些人通过他们的错误启发我们，教导我们。因为在一个健康的、开放的舞台上，失败会带给你比胜利更好的东西。它给了你洞察力。还有一点。尽管有那些死胡同，尽管有各种冒险的尝试和惊呼，但开源项目的数量每年都在翻倍；我们的社区正在繁荣，事实证明，尽管因失败我们没有繁荣，但因失败我们正在繁荣。

下一集预告，DevOps 世界中的安全性如何变化。持续部署意味着安全正在渗透到开发的每个阶段，这正在改变我们的工作方式。同时，如果你想了解更多关于开源文化的知识，以及我们如何改变围绕失败的文化，请访问 redhat.com/commandlineheroes ，免费资源等着你。

00:26:54 - Saron Yitbarek：

《代码英雄》是红帽的原创播客。你可以在 Apple Podcast、Google Podcast 或是其他你喜欢的途径免费收听。我是 Saron Yitbarek，坚持编程，下期再见。

什么是 **LCTT SIG** 和 **LCTT LCRH SIG**

Special Interest Group

LCTT SIG 是 LCTT 特别兴趣小组，LCTT SIG 是针对特定领域、特定内容的翻译小组，翻译组成员将遵循 LCTT 流程和规范，参与翻译，并获得相应的奖励。LCRH SIG 是 LCTT 联合红帽（Red Hat）发起的 SIG，当前专注任务是《代码英雄》系列播客的脚本汉化，已有数十位贡献者加入。敬请每周三、周五期待经过我们精心翻译、校对和发布的译文。

欢迎加入 LCRH SIG 一同参与贡献，并领取红帽（Red Hat）和我们联合颁发的专属贡献者证书。

via: https://www.redhat.com/en/command-line-heroes/season-2/fail-better

作者：Red Hat 选题：bestony 译者：bestony 校对：wxy

本文由 LCRH 原创编译，Linux中国 荣誉推出