Erlang 二十年，如何在编程语言中占据一席之地？

CSDN
已认证账号

1998 年开源发布的 Erlang 项目，在全球范围内只是个小众编程语言。

根据 TIOBE 最新发布的语言排行榜，Erlang 仅排名在第 65 位，市场份额占有小于 0.1%。但是作为一门函数编程语言，它拥有着强大的并行处理能力和容错机制，简单好用且易学。最开始，Erlang 之父 Joe Armstrong 编写的初衷针对的是类似于电话交换机那样的高可用性、高可靠性系统，但后来却成就了可靠性达到 99.9999999% 的目前世界上最复杂的 ATM 交换机！

须臾二十载，如今的 Erlang 项目仍然焕发着生机，并在技术淘汰中稳稳站住了脚跟。那么 Erlang 在这些年中经历了哪些发展阶段？Erlang 语言的知识阶梯如何影响了对它的采用？Erlang 及其社区未来的发展方向是怎样的？……下面就让我们一探究竟。

声明：本文已获作者 Fred T-H 翻译授权。

作者 | Fred T-H

译者 | 苏本如，责编 | 郭芮

出品 | CSDN（ID：CSDNnews）

以下为译文：

大约在10年前，我加入了Erlang社区，当时正值Erlang语言的第一个炒作阶段。我们都被告知，Erlang语言代表着并发性和并行性开发的未来，因为它能够以最轻松和最快速的方式完成开发。你还可以得到免费的Erlang分发包，因为它已经变成了一个开源软件。在那个年代，一切都开始变得不可思议。虚拟机也刚刚获得了SMP支持，而在此之前，要想真正利用同一台计算机上的所有CPU，你必须在那台机器运行多个虚拟机。

在这篇文章中，我想先花点时间对这十年的大部分时间作个反思。然后我会介绍一些这期间发生在Erlang上的事情，例如炒作阶段对Erlang的影响？Erlang语言的知识阶梯如何影响了对它的采用？以及我在Erlang社区的十年中自身发生了什么变化？最后，我将探讨一下我认为Erlang及其社区的发展方向。

炒作阶段

炒作周期（或技术成熟度曲线）在一个产品或技术的生命周期中引入了“阶段”这个概念。这是一个营销概念，而不是一个科学概念，但是用它来描述事情的发展往往会很有用。而我最感兴趣的部分是“炒作阶段”，它就像一股发生在编程社区的淘金热。你可能见过一个或多个这样的炒作阶段，通常他们似乎都依附于某个杀手级应用程序，吸引着每个人蜂拥而至。

我能想到的“炒作”的例子有：炒作Ruby on Rails的“如何在15分钟内构建一个博客引擎”的视频（youtube.com/embed/Gzj72...，其中的“看看我没有做的所有事情！”仍然是一句有趣的话），炒作Kubernetes 的“在Kubernetes下使用Go语言”的文章（Kubernetes在此以前已经有了大量使用，但在那之后确实进入了一个爆发期）。在某种程度上，对Elixir和Phoenix的炒作或许也可以列入这个名单。

在这样的一个炒作阶段，大量的新来者蜂拥而至，都想来看看能捞点什么好处。有些人会留下，但多数人会很快离开。大多数人的停留时间可能只是几个月或几年，能真正地安顿下来并且坚持几十年的人非常少见。绝大多数人都是一些络绎不绝的先期采用者，他们从一个技术冲浪到另一个技术，嗅探最佳的机会，希望先行采用某种框架、语言或工具包，来获取一定的竞争优势。

所以通常的想法是，首先你要拥有一个真正的杀手级应用程序，然后人们就会涌入你的生态系统。杀手级应用程序会驱动这股人潮，只要你把它构造出来，人们就会蜂拥而至。如果你能留住他们中的小部分并保持他们的活跃度，那么在可以预见的未来，你将会拥有一个活跃的社区。这用一种奇怪的方式，让我想起了“雨随犁至”这个理论：

> 上帝加快犁地的速度。……这是奇妙的“人类主宰自然”的理论假定，天上的云会分配取之不尽的雨水…而[犁]是让野蛮变成文明的工具；能把沙漠变成农场或花园。……或者更简洁的表达，就是“雨随犁至”。
> 这一理论的基本前提是，通过人类的定居和农业活动可以影响干旱和半干旱地区气候的永久变化，使这些地区更加湿润。这一理论在19世纪70年代被广泛推广，作为美国大平原（曾被称为“美国大沙漠”）开垦定居的理由。这也被用来证明南澳大利亚在同一时期的边界土地上小麦种植的扩张是合理的。

如果我们能让一个大项目一直进行，开发人员就会出现，然后它就会自我维系下去。我认为这种观点显然是错误的，因为尽管Erlang在其最热的炒作阶段拥有了数十个杀手级应用程序，但是，它的社区仍然维持在很小的规模。下面，就让我们看看在那个时代，Erlang都有哪些杀手级应用程序：

- **ejabberd**（诞生于2002年，第一个稳定版本发布于2005年）：它是迄今为止最具可扩展性的主机聊天服务器之一（即使不是之最）。Ejabberd曾经是一个巨大的成功，在某种程度上来说现在仍然是。到目前为止，在StackOverflow上仍然能看到关于它的模块的提问。在2011年左右，它衍生出了MongooseIM分支，现在这两个解决方案仍然处在维护中。
- **CouchDB**（2005）：是根据CAP定理，用Erlang编写的第一个流行数据库，也是当时新潮的多主文档存储解决方案之一。虽然现在MongoDB是该领域的王者，但CouchDB在存储引擎方面仍然有其精神继承者，最出名的是BarrelDB，它现在还在维护中。
- **RabbitMQ**（2007）：几乎占据整个AMQP市场的一个消息队列软件，它现在仍在用并且很有价值。在流式工作负载方面，它经常被和Kafka一起讨论，尽管它们具有明显不同的特性和用

例。

- **Facebook Chat**（2008）：Facebook Chat的初始版本是用Erlang编写的。由于许多内部因素，如系统稳定性的考虑，和内部C++工程师的强势地位，以及已经有了C++实现的一套解决方案等等，后来它用C++全部改写。
- **WhatsApp**（2009年，2014年被收购）：在Facebook的聊天系统摆脱了Erlang语言后，他们最终购买了WhatsApp这个众所周知的只用了50个工程师开发的服务于9亿用户的跨平台应用程序。它今天仍在使用，事实上，WhatsApp的开发者决定比以前更加深入地参与Erlang和Elixir社区中。
- **Riak**（2009）：是分布式系统世界展现自己优势的最佳例子之一。Riak是一个真正可靠的分布式键值数据库店，它是Basho Technology公司开发的一个当前仍在医疗保健系统和其他关键基础设施中运行的NoSQL数据库产品。在Basho Technology遭遇财务危机并被迫破产时（这在很大程度上是因为公司违反了信托责任，迅速走向了败亡），Bet365买下了它的所有知识产权，并优雅地将其开放源代码，现在Riak数据库仍然在开源世界中运行，尽管它的支持力度和过去最好的时期比稍稍不如。

以上提到的杀手级应该程序，很多都是在乔·阿姆斯特朗的《Erlang编程》一书问世的时候出现的。它们制造了一场对Erlang大量采用的完美风暴，同时也为Erlang吸引了大量的旁观者，甚至对黑客新闻网站都产生了明显的影响，以至于它强制要求在某一天所有讨论都必须是关于Erlang的内部机制。

然而，喧嚣之后，留下来的人很少。

我认为，现在的杀手级应用程序的出现和过去相反，它是出于人们想在最初的炒作阶段分一杯羹的贪婪。任何应用程序，总有一个弱小的、早期的阶段，有人嗅出了它技术的有趣之处，决定喜欢它，并且将它构建出来，如果碰巧它成了一个杀手级的应用程序，那么就会进入一个更大的炒作阶段。人人都盼望从天上掉馅饼，一个成功的故事会培养更多的模仿者。另一个常见的现象是“重塑世界”阶段，每个人都花时间重新实现现有的一切，所以你会看到很多很多“与语言无关”的公告。

但是杀手级的应用程序本身从来就不能够实现开源语言的自我维系。其中一个有趣的结果是，像RabbitMQ和Ejabberd这样的产品，尽管很受欢迎，但其用户社区比贡献者社区要大得多。成千上万的公司使用他们的产品，但他们不一定会参与到Erlang社区中来。

毫无疑问，部分原因在于Erlang的大多数杀手级应用程序都在专门的基础设施中：你创建了一个其他人都可以使用的高可靠性黑盒组件，如果它工作得足够好，他们就不需要查看盒子内部。现在好了，几十个开发人员已经为数千种其他产品和服务提供了基础设施。而按照定义，专业化的基础设施是一个不需要大量人员就能产生巨大影响的地方。所以它们的贡献者群体和社区总是比靠近最终产品的群体和社区要小得多。例如，拥有数不清的网页开发人员的Web开发框架，或者更通用的基础设施，都可以把它们用到小规模的部署项目上，因为任何企业都可能找到它们的用途。

但是，即使没有这些因素，也很容易让人觉得，Erlang错过了一个巨大的机会，那就是它在炒作阶段没有获得更大的蛋糕份额。

知识阶梯

我不想用诸如“过去可以发生或者应该发生”这样的描述来做反事实推理。相反，我想挖掘我在Erlang社区多年的教学和写作中所看到的常见学习模式。这些也是我现在正在Elixir社区中看到的模式，我觉得这可能是预示着Elixir可能有着相似未来的一种迹象。

我最钟爱的一个理论是，像编程语言（及其生态系统）这样的技术主题具有多层复杂性，需要学习和发现各种概念。我第一次开始表达这种理论是在Learn You Some Erlang的网站上，用一个我称之为Erlang第九圈的图表。

我认为学习一项技术并不是无休止的痛苦（至少，它不应该如此），这话是不是说得有点假心假意，我只是喜欢双关语。但简单地说，对任何新技术的学习，通常有一个更“核心”的主题路径或次序，这样就有了“知识阶梯”这样的东西，在这个知识阶梯上，越有价值的知识/概念被放得越高，也越难到达，所以实际情况是，到达越高的地方的人会越来越少。

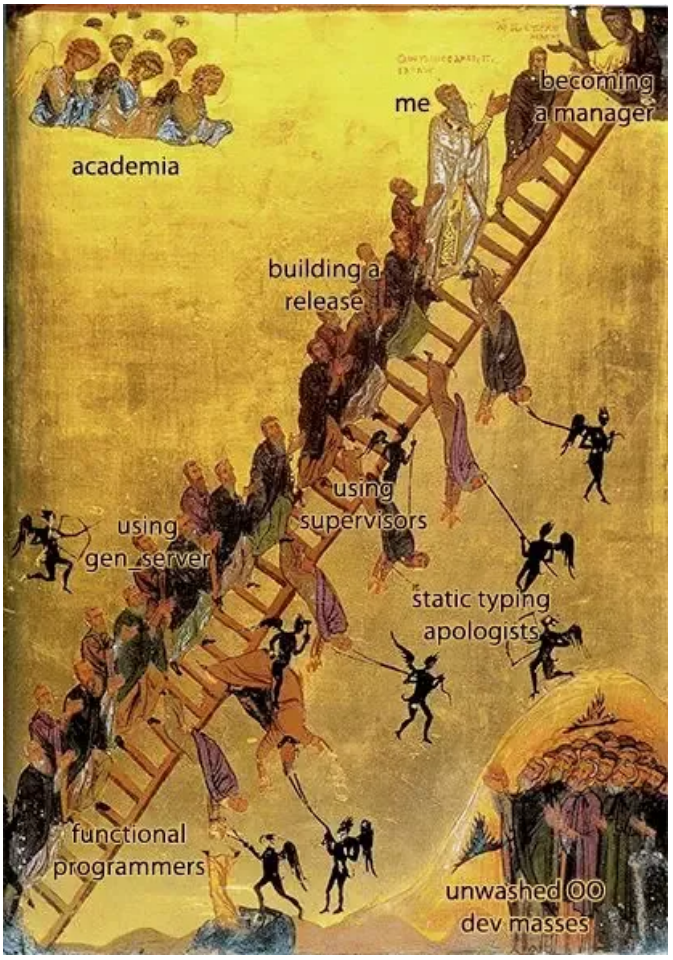
对Erlang来说, 我认为它的“知识阶梯”可能是这样的：

1. **函数式编程；**
2. **隔离进程和并发性；**
3. **可靠的并发性（links, monitors, timeouts）；**
4. **OTP行为模式和其他系统抽象；**
5. **如何构建OTP系统；**
6. **如何构建发布版本并处理其生命周期；**
7. **如何保持系统始终在线，以及如何运作。**

如果你是第一次接触Erlang，并且从一本初学者的书开始学习，你可能会在第一级阶梯上花费大量的时间：与函数式编程、不可变性、递归和类似的概念交朋友。迟早，你会接触到并发性和并行性、进程和消息传递。在那之后，你开始学习links和monitors进程监控，错误处理，以及Erlang的内部机制。在Erlang的大炒作阶段，第二级和第三级阶梯成了最让旁观者惊叹的卖点。如果你必须学会一些在未来的项目中必备之知识，那就是其中之一。

其他的梯级会在稍后跟进，尤其是OTP（第4级阶梯），但前提是你坚持完成编程这一阶梯。OTP被视为Erlang的真正价值所在——并发性和函数式编程确实不错，但是OTP代表的一般开发框架是你必须坚持和使用的真正独特的东西。很多人愿意使用这些框架，了解它们所做的美好的抽象，但是对于如何正确地构建每件事情可能会感到有点困惑。

事实上，像Ejabberd这样的应用程序的大部分开发几乎没有突破第四级阶梯。当时的生态系统有点像蛮荒的西部，对于爱立信的员工和最有动力的自学者来说， OTP知识也就是那么一回事。大多数人只有在有值得投入生产的东西出现问题、并且想寻找更好的解决方法时，才有可能到达第五级。直到2015年或2016年，当Relx的出现开始让整个发行体验变得更容易时，第6级才被重视。第7级几乎从未到达，事实上，很多人都觉得不应该热升级一个节点，理想情况下，你也不会在生产环境中使用SSH命令行进行调试操作。



在实践中，并非每个人都会按照相同的顺序来学习所有这些知识，有些书会调换它们的次序（这让我想到了Erlang and OTP in Action这本书）。我觉得这些次序都不是问题，“阶梯”用在这里只是为了说明问题。

社区波浪式向前发展。炒作阶段会使一个社区的规模增加十倍甚至百倍，大多数人会好奇地看一眼然后离开，所以一个社区中的大多数用户倾向于停留在第一个梯级。少部分人会到达高一级阶梯，更少部分的人会到达更高一级，依此类推，直到你拥有到达最高阶梯的内部专家圈为止。

对于Erlang来说，我认为前三个梯级可能是最容易到达的。第四个阶梯花了几年的时间来开发，最终被认为是有价值的。第五个阶梯非常复杂。Erlang缺乏工具和生态系统，Erlang社区留下来的都是自己选择的，愿意忍受这种贫瘠环境的人，因此对新来者的困境不敏感。为了保持这篇文章简短（好吧，长而不是荒谬的长），我就不在此赘述了。

在任何情况下，如果你是一个Elixir的用户，你可能会看到你在这个硬性定义的阶梯上的位置，你可以感觉到一个社区中的派系通常都在那里。很多人，可能他们只是在Phoenix上做得很好，很少能突破第四阶梯，而且在可预见的将来，他们中的许多会一直停留在第三阶梯或以下。在许多情况下，这些情形都是对的。这里我们只是观察，不作评判。作为一个通过了很多知识阶梯的人（可能在这种环境中，我的头上还有一些知识阶梯，比如“修补虚拟机”之类的），他们似乎错过了很多知识阶梯，但坦率地说，那些东西对他们可能永远不会有用。

但所有这些都是说：作为一个社区，我们可能会让人们很难超越基本水平，从而使我们自己陷入困境。经验教训的学习是不可以一蹴而就的，在某种程度上，Erlang社区是盲人在引导盲人，因为它实在太小了，没有足够的人来分享所有需要的经验。今天的情况比较容易，如果你在一个炒作的周期之外，你很可能会找到好的帮助，因为很少有人同时要求所有的帮助。

我想表达的是，如果明天Erlang有第二个炒作阶段，我们会比上一个炒作阶段表现得更好。希望这些经验，加上Erlang和Elixir社区之间更好的合作，让我们能够接触到更多的受众，让我们成功的机会加倍。

Erlang有哪些改变？

Erlang不是一具放在一个装满甲醛的玻璃容器里的尸体，等待在光天化日之下被带走——它一直在进化。部分原因是由于Elixir社区的压力和需求，幸运的是，他们对自己的工具的期望比Erlang用户已经习惯的要高。另外部分原因在于推动平台向前发展的实际工业需求，而不像学术界，他们只是按照他们自己喜欢的方式推动事情向前发展。

以下是我能想到的一些改变，大家可能很高兴知道有些变化在2009年或更早的时候就发生了：

- **多核支持现在工作得很好。**最初支持2-4个内核的时候，开发人员经常碰到各种超出自己控制的瓶颈问题，后来就可以很好地处理12-16个内核了。而现在我不太确定能支持的内核上限是多少，但有一点我很确定，我写的堆栈操作运行在超过32个内核的机器上没有任何问题。
- **Stacktrace异常跟踪报告支持行号。**回到没有行号的年代几乎是无法想象的，在那个年代，“写简短的自我描述的函数”不仅仅是一个设计问题，也是一个生存问题。现在，你不再需要超自然的调试技能，就可以调试Erlang程序。
- **Unicode支持现在可以接受。**string模块包含最重要的算法，Unicode模块可以很好地处理大多数转换和规范化，处理raw codepoint、utf-8、utf-16和utf-32的一般策略已经具备。本地化支持仍然缺乏，但现在一切都可行，诸如re（用于正则表达式）和所有更高级别的文件处理代码之类的模块也可以很好地处理unicode。
- **支持映射（作为HAMTs实现），具有明确的模式匹配语法。**使用Dialyzer对其进行的类型分析也可以将其替换为多个使用案例，在这些案例中，以前使用的记录非常痛苦。
- **虚拟机中的时间处理机制是世界级的，在处理时间规整、各种类型的时钟等问题时，都可以很好工作。**不过，时区和格式处理使用社区库仍然更好一些。
- **添加了atomics、counters和persistent terms等高性能工具，以帮助改进所有增强可观测性功能和较低级别核心库的底层机制。**
- **所有信号处理都是异步的，包括端口，大大减少了瓶颈。**
- **编译器正在重写中，以便通过SSA获得更高级别的分析和性能提高。**
- **运行NIF的脏调度器已经可用，使得与C甚至Rust代码的集成变得简单，同时支持IO密集型或CPU密集型工作负载。**因此，尽管该语言可能不会无限快，但它已经快过其它语言。在对运行时稳定性不造成太大影响的情况下，为获得更高性能的库而停机比以往任何时候都容易。
- **内存分配和管理的各种改进。**
- **更快速、更灵活的实时跟踪和微观状态分析，以保证正确运行和性能调查。**
- **更灵活的gen-statem OTP行为模式，以实现能够处理选择性接收的有限状态机。**
- **新的改进的日志框架，内置对结构化日志的支持。**
- **重写crypto加密模块以便使用NIF，而不是更复杂（通常更新速度较慢）的驱动程序。**
- **使用NIF对文件驱动程序进行整体重写，以获得巨大的性能提升。**
- **使用NIF对网络驱动程序进行重写的工作正在进行，以获得类似的性能提升。**
- **对用于TLS处理的SSL应用程序进行整体重写。**这让我想起我在HeloCu工作的日子，通过整体重写使其与C++解决方案在延迟（可能慢5%）方面具有竞争力，并且在可预测性方面总体上要好得多（大约10-30倍的提升）。
- **ETS性能的主要改进。**
- **我编写了一本关于如何使用Erlang VM操作和调试生产系统的手册（erlang-in-anger.com/）。**
- **全新的构建工具（rebar3），与Erlang生态系统的统一软件包管理器集成。**
- **在虚拟机上还提供多种新的编程语言，具有可替换的库用法。**包括（但不限于）Elixir、Efen、LFE、Luerl、Clojerl，以及至少两种使用Gleam和Alpaca进行类型推断的语言。
- **其它更多针对核心Erlang分发包的内部和外部的改进。**

如果你有兴趣了解更多信息，你可以查看版本发布说明的完整列表（erlang.org/news/tag/rel...）。但简而言之，如果OTP 版本13到16的发布时间对爱立信（Ericsson）OTP团队来说有点晚的话（我们现在已经是版本22！），那么他们使用Erlang对他们的旗舰产品中所做的最新投资确实是显而易见的。但即使在爱立信之外，一切都在发生变化。Erlang社区，以及Elixir社区和运行在Erlang VM上的其他语言的贡献者，都聚集在一起建立了Erlang生态系统基金会，现在它拥有一个活跃的工作组，帮助协调和解决有关构建和打包工具、可观察性工作、安全性、培训和采用等问题。

如果你像我一样，在大炒作初期加入Erlang，但又没有像我一样留下来，因为你觉得很多东西不可用或太棘手，你可能想再试一次，因为Erlang的语言的人类工程学及其生态系统已经大大改善。

Erlang将去向何方？

虽然没有必要像2007年到2009年那样突然出现大杀手级的应用程序，但这并不意味着没有任何项目显示出这种希望。Erlang仍然是许多公司的基础设施的不可缺失的部分，其最初的杀手级应用程序大多还在其上运行。我们也有很多有趣的新应用，就像每个BEAM配置文件显示的那样。我自己真的很喜欢基于属性的测试等概念，并且Erlang和Elixir拥有世界上最好的框架。尽管如此，迹象表明我们现在还没有进入炒作阶段。

会有另一个炒作阶段吗？也许有，也许没有。你可以说，Elixir会有下一个炒作阶段。生态系统都有足够的共同点，在一个地方学到的经验教训可以用到另一个地方。它们之间的相似之处多过不同之处。也许还有一个新的复兴时期，我个人不再那么在乎它了。我喜欢小社区，所以我对此感觉很好。Erlang不需要几何级增长来让我觉得乐在其中，它只需要可持续发展。

Erlang社区的规模大小也从来没有阻碍它在全世界发挥它的影响力。就我所知，Erlang一直处于这样一种状态，既没有足够的工作量满足Erlang开发人员的需求，同时没有足够的开发人员来完成Erlang的工作：这两个方面都有很多工作要做，但他们在地理位置上并不一致。面向偏远市场的公司和员工往往做得最好。而在Erlang之前无法轻易突破Webapp市场的地方，整个Elixir的就业市场现在只需稍加努力就能达到良性循环。

从一个更高的层面来看，你是否使用Erlang或类似的语言，这可能并不太重要。虽然我确实觉得它没有被充分利用而且它的价值被低估了，但最大的好处不是来自于运行一个使用它的系统。而是来自于学习可靠系统设计的基本原理，并在实际环境中吸收其经验教训。

我多年来听到的一类问题都和寻求指导有关。例如，我应该如何学习系统设计？关于构建分布式系统，你有什么好的建议吗？我应该做什么可以让系统变得更加健壮和容错？我怎么知道我的设计是模块化的，不会导致抽象泄漏？什么是良好的错误处理？有什么好方法可以让我知道优化工作还为时过早呢？声明（declarative）某个东西意味着什么？

我们总是喜欢简短易懂的解决方案，如食谱和最佳实践，但事实证明，大多数真正的答案都是“我花了很多时间学到的”或类似的东西。我可以坦诚地说，我的职业生涯中没有什么能比得上花时间在Erlang世界里，潜移默化地吸收社区里老手们的丰富经验。从数量上看，Erlang不是一个大的社区，但从任何其它指标来看，它肯定是富有的。几年后，我从一个初级开发人员变成了高级开发人员，在世界各地发表演讲，寻找方法把我获得的这些经验传授传大家，这其中大部分都归功于Erlang这个社区。

也许我在15分钟内还是写不出一个博客引擎（事实上，我不是一个快速开发人员），但我已经成为一个更加可靠的开发人员和系统架构师，我认为这是一种非常有效的方式。再说一次，我在这里讨论的不是使用系统，而是构建它们并使它们工作。无论如何，激励人们的东西并不随处可见。

我无法想象我能在其他社区能得到这么多，过去的10年里发生的一切令人惊叹。有趣的是，Erlang社区仍然很小，大部分还没有开发利用。这意味着你有足够的机会参与到任何事情中去，与那些充满智慧的人一对一地进行交流，并为自己争取一席之地。

原文：ferd.ca/ten-years-of-er...

【END】

发布于 2019-07-24 13:18

Erlang（编程语言）

写下你的评论...

还没有评论，发表第一个评论吧

文章被以下专栏收录

CSDN

推荐阅读

为什么需要这么多编程语言？

学编程的过程中，总是有小伙伴纠结我到底是该学C语言呢？还是Python呢？或者学Java？ 那到底为什么编程需要这么多编程语言呢？ 编程语言的起源是怎样的？ 其实编程语言并不是一开始就有这么…

张巧龙 发表于开源机器人

历经10年，我写了一个没人用的编程语言

头发还在呢

10门最佳编程语言各

如果你是一名具有前瞻或者想成为一名具有前发者，那么，是时候开了，至少在最好的编程一个，类似 Python, S JavaScript, C#, C, Rub

可乐瓶里的小辣椒