

## Erlang 之父 Joe Armstrong 逝世，如何评价他在计算机领域的贡献？

关注问题

写回答

邀请回答

好问题 16

1 条评论

分享 ...

17 个回答

默认排序

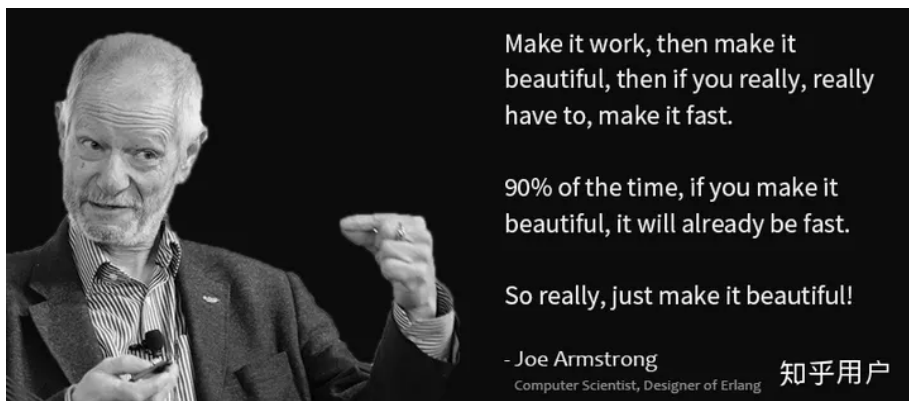


知乎用户

阿里巴巴 后端高级工程师

1,010 人赞同了该回答

RIP, 哀悼，第一次因为名人去世而感到痛心。



Joe Armstrong 不必说 erlang 与 OTP，光他的论文《面对软件错误构建可靠的分布式系统》[kaiyuanba.cn/content/ot...](http://kaiyuanba.cn/content/ot...) 就足以载入史册——领先现在几十年，提出了 OOP 等思想本质上不是并发的正确处理办法。

Joe Armstrong 在论文中是这样认为的：几乎所有传统的编程语言对真正的并发都缺乏有力的支持——本质上是顺序化的，而语言的[并发性](#)都仅仅由底层操作系统而不是语言提供。

而用对并发提供良好支持的语言（也就是作者说的[面向并发](#)的语言 Concurrency Oriented Language）来编写程序，则是相当容易的：

1. 从真实世界的活动中识别出真正的并发活动
2. 识别出并发活动之间的所有[消息通道](#)
3. 写下能够在不同消息通道中流通的所有消息

程序的结构要严格保持与问题的结构一致，即每一个真实世界里的活动都严格映射到我们编程语言中的一个[并发进程](#)上。如果从问题到程序的映射比例为 1:1，我们就说程序与问题是同构 (isomorphic) 的。

映射比例为 1:1 这一点非常重要。因为这样可以使得问题和解之间的概念隔阂最小化。如果比例不为 1:1，程序就会很快退化而变得难以理解。在使用非面向并发的编程语言来解决并发问题时，这种退化是非常常见的。在非面向并发的编程语言中，为了解决一个问题，通常要由同一个语言级的线程或进程来强制控制多个独立的活动，这就必然导致清晰性的损失，并且会使程序滋生复杂的、难以复现的错误。

在分析问题的时候，我们还必须为我们的模型选择一个合适的粒度。比如，我们在编写一个[即时通信系统](#) (instant messaging system) 时，我们使用每个用户一个进程的方式，而不是将用户身上的每一个原子对应到一个进程。

其次，通过定下的九条原则性思想设计，写出来天然支持[分布式系统](#)的 erlang 以及 OTP 框架，真的做到了他说的实现面向并发的语言（Concurrency Oriented Language）。

- 一切皆进程
- 进程强隔离
- 进程的生成与销毁都是轻量的操作
- 消息传递是进程交互的唯一方式

- 每个进程有唯一的名字
- 你若知道进程的名字，就可以向他发消息
- 进程之间不共享资源
- 错误处理[非本地化](#)<sup>Q</sup>
- 进程要么正常跑着，要么马上挂掉

就以上九条的观念，设计出的 erlang 语言，成就了可靠性达到99.9999999%的目前世界上最复杂的 ATM 交换机。

其三，[let it crash](#)<sup>Q</sup> 思想的提出与实现。

程序不可能处理一切错误，因此程序员只要力所能及的处理显然易见的错误就好了，而那些隐藏的，非直觉性的错误，就让他崩掉吧——本来就很有可能是极少见的错误，经常出现的？就需要程序员人工处理了，这是紧急情况，就算 try catch 所有错误也无法避免，因为系统已经陷入崩溃边缘了，苟延残喘下去只是自欺欺人。

要注意的是 let it crash，并不是说你用 go lang, C++ 等语言打包个[二进制](#)<sup>Q</sup>，用 supervisorctl 等工具监控，错误就让他马上崩，挂了自动重启 --，参考我之前的答案[应该如何理解 Erlang 的「任其崩溃」思想](#)？

其四，一切进程都是轻量级的，都可以被监控（monitor），有 Supervisor 专门做监控。

你可以方便的用一个进程（supervisor）去管理[子进程](#)<sup>Q</sup>，supervisor 会根据你设定的策略，来处理意外挂掉的子进程（这种情况不多的是，错误处理稍微做不好就会挂），策略有：

- one\_for\_one：只重启挂掉的子进程
- one\_for\_all：有一个子进程挂了，重启所有子进程
- rest\_for\_one：在该挂掉的子进程 创建时间之后创建的子进程都会重启。

老夫敲代码就是一把梭！可不，只要重启就行。

实质上，这是有论文支持的：在复杂的产品系统中，几乎所有的故障和错误都是暂态的，对某个操作进行重试是一种不错地解决方法 —— [Jim Gray 的论文](#)中指出，使用这种方法处理[暂态故障](#)<sup>Q</sup>，系统的[平均故障间隔时间](#)<sup>Q</sup> (MTBF) 提升了 4 倍。

因此，你就可以创建一课[监控树](#)<sup>Q</sup>，根节点就是啥事都不做，只负责监控的进程。其他都是它的子进程，如果不是 coredump（几乎不发生），那么根节点就不可能会挂；因此其他子进程就会正确的被处理。

当然，这有前提：5 秒内重启超于 3 次，就会不再重启，让进程挂掉。为什么呢？因为重启是为了让进程回到当初启动时的稳定态，既然稳定态都不稳定了，重复做重启是没有意义的，这时迫切需要人来处理。

---

Joe Armstrong 实在太多真知灼见了，比如 type system 才兴起不是很久，他就说，比起 完备且看起来完美的 type system，更重要的是你如何去做这件事情，如果从一开始就考虑错了（如用了非并发式语言来处理并发问题），不管代码多正确，也不过是正确的走在错误的方向上。

另外可以看看我写的关于 elixir 的介绍，[elixir](#)<sup>Q</sup> 是 Joe 看好的孙子

为什么你一定需要学习 Elixir?

[mp.weixin.qq.com/s/Q2XdmY4XHL\\_tnapp...](https://mp.weixin.qq.com/s/Q2XdmY4XHL_tnapp...)



erlang 的开发者 Joe Armstrong 去世了

[mp.weixin.qq.com/s/CzrH9330xvWaThc\\_C...](https://mp.weixin.qq.com/s/CzrH9330xvWaThc_C...)

RIP

编辑于 2019-04-21 23:25

赞同 1010

6 条评论

分享

收藏

喜欢

收起



知乎用户

132 人赞同了该回答

我只是一个半路出身的程序员，可能没有能力评价Joe老爷子在计算机领域的贡献，不过可以从自己的角度出发讲讲Erlang究竟带给了我什么。

在大二的时候我被[模拟电路](#)、数字电路、[信号与系统](#)、通信原理、[数字信号处理](#)……这一大堆专业课程折磨得要死要活，自己对它们一点也不感兴趣，产生了“叛逃”的念头。那个时候学校的ACM/ICPC集训队正好在面向全校举办讲座，过去听了几场而且跟着练了一阵子的题，发现原来写代码的感觉是如此地酸爽。虽然没有继续加入校队跟神牛们一起搞竞赛，但是后面的几年时间学习之余我都一直“不务正业”去各种OJ刷题。当时其实并不知道将来能做什么，隐隐约约觉得以后可能会写代码吧，但具体是个什么目标我也说不上来，整个人活得就像个[梦虫儿](#)一样。

到了毕业季才发现自己的菜，除了会刷几道简单或者中等的算法题以外，别的什么都不会。不懂什么是[进程线程](#)，不懂什么是[网络编程](#)，不懂[数据库](#)的基本使用……我都怀疑自己是不是选错了路自己压根就不适合。这些问题其实很基础并不难搞懂，当初自己为什么这么菜确实是不可思议，但是对于一个半路出家而且技能树点偏的人来说，它确实就发生了。最后一家眼瞎公司收了我，公司的手游服务器是用Erlang来写的。于是一边学Erlang一边码业务，我慢慢开启了程序员的入门之旅。

Erlang一开始给我的印象是“自然”。什么是进程什么是线程？不管，反正我先spawn出来一个东西，它能够执行你写的一个函数，你可以给它发消息它可以给你发消息。然后你可以不停地spawn一大堆的东西它们之间靠着消息传递相互协作最后完成任务。这就是我这种菜鸟一开始的认识，虽然很浅显，但是完全无障碍而且很自然。好像你什么都不需要做，自然而然就会写[并发程序](#)了，代码是有些烂，但是上线以后你发现它竟然就神奇的支撑起了单机七八千的玩家在线。总之我还是不懂什么是[并发编程](#)，但是以Erlang这个点为突破口，后面再去了解什么是Actor，什么是CSP，怎么fork怎么new thread，怎么做[互斥同步](#)，就发觉不像上学时候那么懵逼了。再到后面我学会了很多写并发的套路，但还是觉得Erlang写起来最自然。

Erlang接下来给我灌输的观念是“容错”。在接触Erlang以前，我也尝试着思考过代码出错该怎么办，然而越想越发现这是个极其复杂的问题。我该返回[错误码](#)还是抛异常呢，即使返回了错误，我的调用者又该怎么处理呢？甚至更可恶的是还有一些错误压根就不在你的掌控范围以内，那么它们发生了又该怎么办呢？而我从Erlang学到的并不只是表面上的"let it crash"，而是你在设计任何系统的时候都要考虑以什么样的方式监控每一个[子系统](#)，它们出问题的时候又应该以什么样的策略重启。而我一开始纠结的那些问题并不是真正的核心，当你有心里对系统的容错有具体方案以后，你就很容易知道在哪些地方应该做检查哪些地方不应该，哪些地方应该抛出异常哪些地方不应该。

Erlang然后也将我引进了“FP”的大门。一开始写Erlang的时候是很别扭的：凭什么不能让我修改一个变量的值？连循环都没有你让我怎么写代码？不过写得多了也就慢慢习惯了，觉得本来就不该如此吗？其实Erlang里面真正的[函数式](#)特性并不算多，不过顺着这个藤再摸后面的瓜就容易多了。学Erlang以前我也看Haskell，看不懂，学了以后再去，至少能看懂那么一点点了。

一晃工作了四五年，项目也做了四五个，工作之余也会挑一部分源码来读，在这个过程中越来越觉得Erlang就是一个取之不尽用之不竭的大宝库。也看过一些比较出名的项目比如Nginx和Redis之类，发现还是Erlang的源码难啃啊，毕竟说Erlang是个微型操作系统也不为过。

现在已经不写Erlang了。后面我接触过不少概念：Golang、区块链、[微服务](#)、docker、k8s……发现想要上手它们其实并不难，而且能够很自然地找到具体的路径，不像刚毕业时那种懵逼状态了。我想我这个半路出身的程序员应该算是真正入门了吧，而这个过程是在Erlang的陪伴下完成的。

感谢Joe老爷子创造了Erlang。

编辑于 2019-12-02 20:08



下载知乎客户端

与世界分享知识、经验和见解



## 相关问题

谁是最悲惨的计算机科学家？ 0 个回答

仍然在世的最伟大的计算机科学家是谁？ 2 个回答

什么样的人可以被称作计算机科学家？ 0 个回答

知乎上有哪些计算机科学家？ 0 个回答

你知道哪些伟大的计算机科学家？ 0 个回答

## 相关推荐



Haskell 函数式编程入门

张淞

236 人读过

阅读



Learning Akka

Jason Goodwin

0 人读过

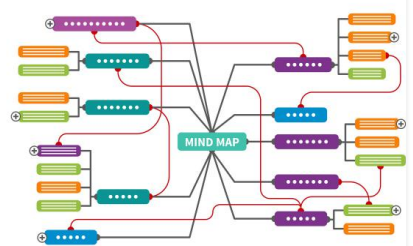
阅读



七周七语言：理解多种编程范式

666 人读过

阅读



## 思维导图软件

刘看山 · 知乎指南 · 知乎协议 · 知乎隐私保护指引

应用 · 工作 · 申请开通知乎机构号

侵权举报 · 网上有害信息举报专区

京 ICP 证 110745 号

京 ICP 备 13052560 号 - 1

京公网安备 11010802020088 号

写下你的评论...

## 5 条评论

默认

最新



一苇千帆

我也尝试着思考过代码出错该怎么办，然而越想越发现这是个极其复杂的问题。我该返回错误码还是抛异常呢，即使返回了错误，我的调用者又该怎么处理呢？

已经深深陷入这个坑，可以详细的谈一下这个问题的应对吗

2021-10-29

● 回复

👍 1



云末

题主现在是在做什么？我现在还是在用erlang写游戏服务端

2019-05-29

● 回复

👍 赞



thomas

我看了几十页的Erlang入门教程就看不下去了，觉得还是C语系的语法习惯。

2019-04-22

● 回复

👍 2



David Hobbes

这肯定的，函数式的语法实现很多东西都比较困难的，譬如一个简单的KMP字符串匹配算法，用Erlang或者Haskell就难写。

2019-10-29

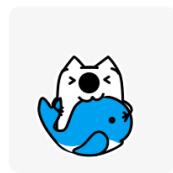
● 回复

👍 2



张凯

哎。。。这么好的语言我竟然没发现



2019-04-21

● 回复

👍 赞



程墨Morgan

知乎十年新知答主

+ 关注

17 人赞同了该回答

我对Erlang也不是太懂，只是感叹，计算机领域的前辈们也开始仙去了，这感觉就和看新闻上老一辈艺术家一个一个离开一样，伤感呐。

发布于 2019-04-22 07:49

写下你的评论...

## 1 条评论

默认

最新



黄宝臣

不奇怪，人类历史上一次文化井喷和硅谷文化创建大概都是60年代底到70年代。恰好一算，这群人年纪到了

2019-04-22

● 回复

👍 4

Baidu

广告 ▼

i.hao61.net/d.js?cid=40308

京网文[2022]2674-081 号

药品医疗器械网络信息服务备案

(京)网药械信息备字(2022)第00334号

广播电视节目制作经营许可证:(京)字第06591号

服务热线: 400-919-0001

违法和不良信息举报: 010-82716601

举报邮箱: jubao@zhihu.com

儿童色情信息举报专区

互联网算法推荐举报专区

仿冒诈骗专区

MCN 举报专区

信息安全漏洞反馈专区

内容从业人员违法违规举报

网络谣言信息举报入口

网络传播秩序举报专区

涉企虚假不实信息举报专区

证照中心 · Investor Relations

联系我们 © 2023 知乎

北京智者天下科技有限公司版权所有





绅士猫

用心写给我限流，我只能输出垃圾了

+ 关注

27 人赞同了该回答

Elixir 开发者路过。Elixir 是 EVM（更多的时候指 Beam）上的方言，与 Erlang 一脉相承，早期 Joe 还学过 Elixir： [A week with Elixir](#)

上面的文章算不上对 Elixir 的深度剖析，但仍然是一篇非常有参考价值的 Erlang 开发者开始使用 Elixir 的真实感受。更何况这名 Erlang 开发者同时还是一直活跃在技术领域的 Erlang 的创始人之一。强烈建议有相关想法的 Erlang 开发者去拜读一番。

顺便有不少人跟我类似是通过 Elixir 学习 Erlang 再了解到 Joe 的：

▲ feniv 18 hours ago [-]

I've only recently started learning Erlang (via [Elixir](#)), but I'm absolutely amazed by the underlying technology and the brilliant minds behind it. I'll remember Joe by the several insightful, entertaining talks he's given in recent years. <https://www.youtube.com/watch?v=IKXe3HUG2I4> in particular. [rsrly](#).

并且 Joe 近来越来越关注 Elixir 社区，前不久还成了官方论坛的管理员：

<https://elixirforum.com/t/introducing-our-new-moderators-and-a-new-admin-joe-...>

[elixirforum.com/t/introducing-our-new-moderators-an...](#)

可惜的是不久他就去世了。

Twitter 上 Joe 的帐号在这个月 10 号还发表了跟技术有关的言论：

JOE ARMSTRONG



Joe Armstrong @joeerl · 4月10日

Once upon a time my boss asked me to study if we should use C++ or Erlang for a specialist XML parser to be used in a product (for reasons of speed not energy).

My recommendations was an FPGA

We built an FPGA.

Relative speed of C++/Erlang was irrelevant compared to FPGA.

Joe Armstrong @joeerl

Should also add that all significant energy gains in the last 50 odd years are result of new hardware NOT software. [twitter.com/joeerl/status/...](https://twitter.com/joeerl/status/...)

🌐 翻译推文

💬 46

↺ 187

❤️ 778



知乎 @绅士猫

有一些人分享了跟 Joe 的合照：



🔄 Elixir Fountain 转推了



**Ju @arkh4m** · 17小时

Goodbye Joe! We all, transient children of a simple\_one\_for\_one supervisor, salute you 🙏

🌐 翻译推文



💬 1    🔄 13    ❤️ 109    ✉️

知乎 @绅士喵

表达对 Joe 去世的惋惜：

🔄 Code BEAM 转推了



**Herbert Daly @HaloedPayload** · 17小时

Really sad and shocking news. The significance of his contribution has grown and will continue to grow over time. Many thanks to a practical Computer Scientist of the Old School. A link to the times when all in tech was up for grabs. We owe Joe and his generation a great debt.

**Code BEAM @CodeBEAMio**

We are devastated to hear the news about Joe, who was a wonderful leader and collaborator within the #Erlang and #Elixir ecosystem right to the end. @joeerl you'll be greatly missed ❤️ [twitter.com/FrancescoC/sta...](https://twitter.com/FrancescoC/sta...)

🌐 翻译推文

💬    🔄 10    ❤️ 15    ✉️

知乎 @绅士喵

因为我 Twitter 上关注了不少 Elixir 和 Erlang 相关的帐号，昨天的动态充斥着 Joe 去世的消息。

如果你是一名资深的 Erlang 开发者，经常查看甚至参与 Erlang 邮件列表上的讨论的话，那肯定是非常了解 Joe 的那一批人。因为 Joe 长期活跃在那里。

很可惜，我不是，我并不是一名 Erlang 开发者，没能足够了解他。

附带 Joe 生前的一些精彩内容：

- 至今还保留在 Erlang 文档页的著名论文《[Making reliable distributed systems in the presence of software errors](#)》
- 有关计算、编程、行业复杂性的演讲《[The Mess We're In](#)》
- Joe 的[博客](#)

当然最精彩的毫无疑问是服务于世界一半的电信系统的可靠基石 Erlang/OTP 技术了。

```
+ ~ erl
Erlang/OTP 21 [erts-10.3.1] [source] [64-bit] [smp:8:8] [ds:8:8:10] [async-threads:1] [hipe]

Eshell V10.3.1 (abort with ^G)
1> io:format("Goodbye Joe~n").
Goodbye Joe
ok
2> |
```

知乎 @绅士喵

编辑于 2019-04-21 17:29

▲ 赞同 27 ▼

● 添加评论

🔗 分享

★ 收藏

♥ 喜欢

收起 ^

 知乎用户

13 人赞同了该回答

震惊、哀悼、缅怀。

先生的博士论文拜读过多遍，process、messaging、let it crash、OTP 的设计让人醍醐灌顶，眼界大开。

无奈 Erlang 的技术太领先于时代了，最终止步于一门小众的编程语言。

说实话，Erlang/OTP 的设计如果能用在航天、核电、武器装备这种对系统可靠性和容错性要求极高的领域，那是再合适不过了。只可惜国内这些行业的从业者基本都没听说过 Erlang 的大名，遑论用于型号产品，用 C/C++ 重新造轮子都忙不过来呢……一声长叹。

编辑于 2019-04-21 14:46

▲ 赞同 13 ▼

● 收起评论

🔗 分享

★ 收藏

♥ 喜欢

写下你的评论...

7 条评论

默认 最新

 左岸

遑论

2020-02-29

● 回复

👍 赞

 欲三更

erlang在国内基本上都用来写游戏了。。。

2019-05-11

● 回复

👍 赞

 Qiqi

emqx听说了吗

2019-06-05

● 回复

👍 赞

 Houhsiaohui

刚接触时，感觉太不习惯了，没有循环只有递归，而且那时没有分布式的概念，只是囫囵吞枣的把趣学erlang看了一遍，看了你的评论很想抽空再读一遍



2019-04-21

回复 赞



Houhsiaohui ▸ X Zhang

有推介的关于分布式的书？

2019-04-21

回复 赞



Houhsiaohui ▸ X Zhang

好的，谢谢

2019-04-21

回复 赞

展开其他 1 条回复 >



网易数帆

已认证账号

+ 关注

6 人赞同了该回答

1. “一件事情如果过于复杂，那么一定是哪里出问题了——大部分情况下是对问题的理解出现偏差。”来自朋友圈的引用，大师的思想对编程的影响，还会持续很久。
2. 有从 Python 到 Erlang 再到 Golang 的大咖表示：想把 OTP 搬到 Go 里。
3. 在整个互联网，基于 RabbitMQ 的消息队列实例，仍然在发挥着重要作用。

发布于 2019-04-22 15:11

赞同 6

收起评论

分享

收藏

喜欢

写下你的评论...

4 条评论

默认

最新



一浪又一浪

erlang的OTP搬到k8s里面了，这样就和语言无关了

2019-04-23

回复 2



网易数帆 作者 ▸ 一浪又一浪

说明 OTP 对云原生编程确实很有用。

2019-04-25

回复 赞



一浪又一浪 ▸ 网易数帆

一定是k8s，或者类似与k8s的东西，Erlang的OTP是开发了一个和Erlang语言强行绑定的分布式操作系统，而k8s是和编程语言无关的OTP，Erlang/OTP很多概念在k8s都能找到类似的东西。

2019-04-24

回复 赞

展开其他 1 条回复 >



大宽宽

飞书任务诚招人才中-有意者私信

+ 关注

30 人赞同了该回答

缅怀。

最喜欢他的观念：解决问题总是要先理解问题，然后要找到“自然”的方案。进程、线程、Actor、messaging、容错、并行……这些都是我们在解决真实问题的方式在计算机领域的抽象。我们先要理解问题，然后找到“自然而然的方案”，再找到合适的工具去处理它们。Erlang/OTP就是这种思想的产物。

“一件事情如果过于复杂，那么一定是哪里出问题了——大部分情况下是对问题的理解出现偏差。”

大师走好。

发布于 2019-04-22 15:46



赞同 30 收起评论 分享 收藏 喜欢

写下你的评论...

- 3 条评论 默认 最新
- 

avalonyzhily

于是，如何定义复杂，又成了一个问题。

2019-04-23

回复

1
- 

metalmoon

感觉答主很多观点都和我相同啊

2019-04-22

回复

赞
- 

Chau

问题清楚了答案自然就清晰了

2019-04-22

回复

赞

知乎用户

1 人赞同了该回答

虽然没有真正用过Erlang，但是Erlang在并发编程上的思维曾经给我很多启发。感谢Joe Armstrong先生。

想到了前些年去世的Dennis Ritchie和乔布斯，想到那一代的先驱者们都已经到了垂暮之年，想到自己也已过了而立，不觉有了川上之思。

R.I.P.

发布于 2021-11-07 18:01

赞同 1 添加评论 分享 收藏 喜欢

知乎用户00U450

+ 关注


16 人赞同了该回答

上帝想学并发编程了。

发布于 2019-04-21 12:23

赞同 16 收起评论 分享 收藏 喜欢

写下你的评论...

- 1 条评论 默认 最新
- 

玄冰尘灵秀

2019-04-21

回复

赞

毯毯毯毯

+ 关注

10 人赞同了该回答

坦白讲，我是没资格评论人家的。不过，我认为他创造的erlang的每一处，都深得我心——我知道这是在表扬我自己。在我觉得我的erlang大概是懂了之后，我又去看了看scheme，我用的是racket。我越发确定，我懂不了scheme，它搞的过于复杂了。你用一下函数式的优点就行了，干嘛搞得那么复杂，让人头晕呢？erlang就好多了，没有宏，没有什么continuation，没有delay，这些垃圾，整个世界清爽多了。我看racket感觉是走进了一片热带丛林，到处荆棘，到处陷阱，视线混

乱，动物繁多。而erlang，就像是一个简化了的清纯世界，像我的校园生涯，清楚而纯粹。一句话就是——爱上了。也许这和人的本身素质有关，我搞不了太复杂的东西，就像打王者荣耀，我也只会用那些简单粗暴的英雄。我属于典型的copy程序员，你让我写个树，写个冒泡，我的得搞好几天。没错我就是这么的垃圾。但是erlang让我觉得，世界也可以没那么复杂的。好歹也给头脑简单的人一点空间。没错！我认为erlang是简单的！它就是质朴的做基本的事，而把复杂的事交给vm，让聪明的人，专业的人，集中精力去解决。而我们就像去下馆子，吃就是了。是joe这个老头搞出了这个东西，我得感谢他，让我觉得编程世界还是有happy的可能性。这世界本来就是多样的，有的人适合使用刀，有的人适合用剑，有的人适合用枪。而c语言就是我摆弄不了的武器，指针到底是指哪里了？啥时候该释放？边界咋判断？我头都炸了！我他妈，你给我把手枪，让我直接撸火不行吗？我听过一句话，大概是：人的愤怒，基本上全都是因为自己的无能！没错，我要哭了。

发布于 2020-02-10 08:12

赞同 10



添加评论

分享

收藏

喜欢



匿名用户

4 人赞同了该回答

思想永存。

编辑于 2019-04-23 01:04

赞同 4



添加评论

分享

收藏

喜欢



知乎用户

3 人赞同了该回答

说到Erlang 就想安利一波Akka…

老先生的思想非常惊艳，忍不住安利一波Erlang程序设计

发布于 2019-04-21 07:48

赞同 3



添加评论

分享

收藏

喜欢



知乎用户

2 人赞同了该回答

程序员改变世界！！哀悼

发布于 2019-04-22 11:12

赞同 2



添加评论

分享

收藏

喜欢



留看

+ 关注

1 人赞同了该回答

乔爷一路走好，这是我最爱的编程语言，没有之一。

论起计算机领域上能有此贡献相比较的，也就是缓存的发明。

发布于 2019-05-04 03:25

赞同 1



添加评论

分享

收藏

喜欢



可能事件

不脱发的搬砖工

+ 关注

erlang让并发编程变得如此简单。

发布于 2019-04-24 10:52

赞同



添加评论

分享

收藏

喜欢



周一维

+ 关注

Joe对现实世界的思考形成了Erlang的哲学。Erlang超前的技术应当被发扬光大，但Joe的思想更是难得宝贵，用**自然的方式**去思考并处理问题，才是正确的。

发布于 2019-04-23 00:08

[▲ 赞同](#) [▼](#) [● 1 条评论](#) [🚩 分享](#) [★ 收藏](#) [♥ 喜欢](#)



知乎用户3j0giO

[+ 关注](#)

用了近10年erlang了，无限哀悼，感谢他做的贡献。真没想到，老头在twitter如此活跃，竟突然离世。唉，谢谢！

发布于 2019-04-22 11:15

[▲ 赞同](#) [▼](#) [● 2 条评论](#) [🚩 分享](#) [★ 收藏](#) [♥ 喜欢](#)

[✎ 写回答](#)

1 个回答被折叠（为什么？）