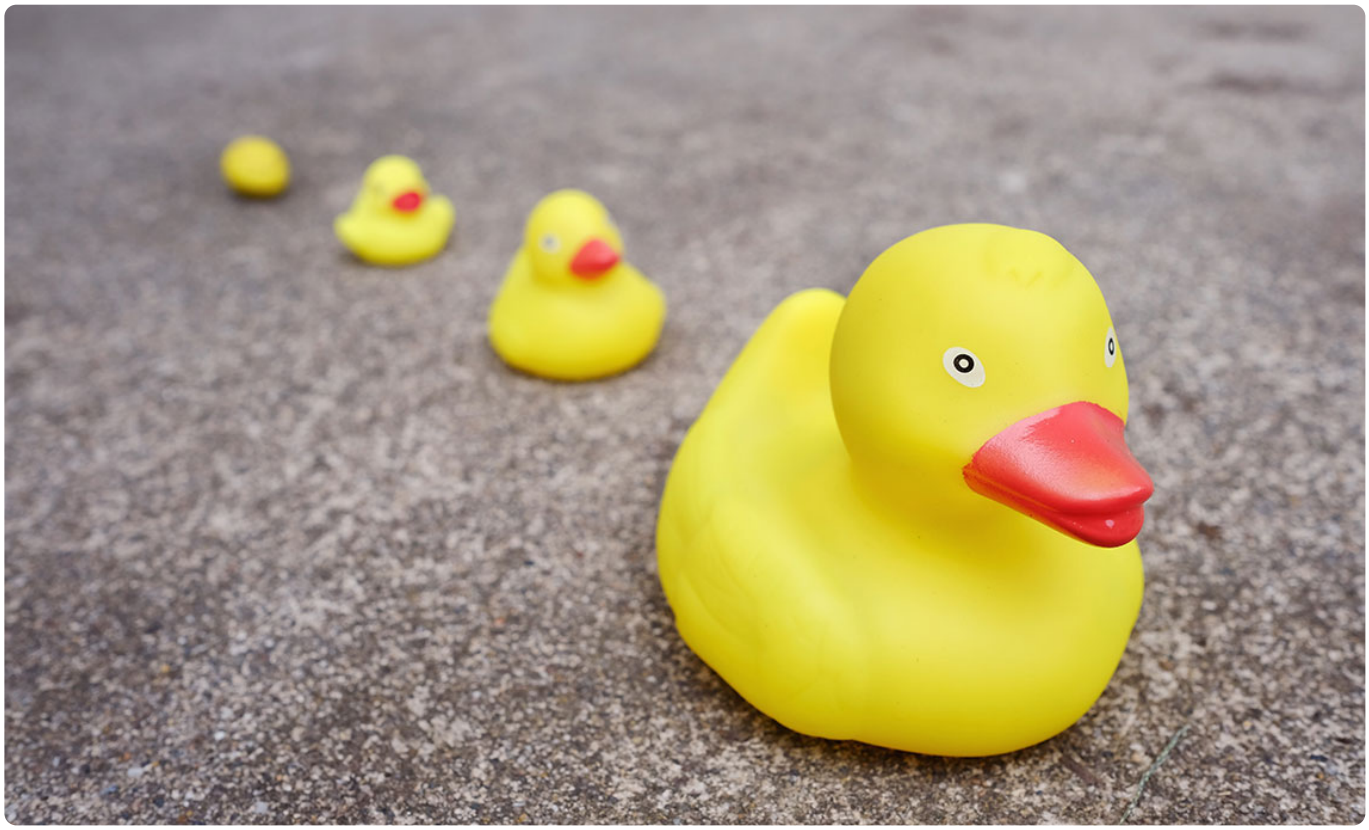
加餐 | 再八卦几门语言！

2020-06-19 郑晔

软件设计之美

进入课程 >



讲述：郑晔
时长 17:23　大小 15.92M

你好，我是郑晔！

软件设计是一个比较烧脑的话题，对于一些同学来说，学起来还是有一些辛苦的。所以，我准备了这次加餐，让大家在前面高密度地狂奔了一段时间之后，稍微休息一下。

我在第 8 讲中讲了程序设计语言的发展，有同学觉得不过瘾，想了解其他语言的发展过程。那好，我们就来谈谈几门比较吸引眼球的程序设计语言。

C#

当年 Java 开始起势的时候，微软还处于自己的巅峰，它当然不想错过 Java 这么有前景的东西。但是，微软从来就不会老老实实按照标准做事，所以，你会看到微软手中的 Basic 已经很不像 Basic 了，微软的 C++ 也有着自己的扩展。

于是，微软也想做出一个自己的 Java，J++ 就出现了。但是，这不是一个正常的 Java，引发了 SUN 的不满，将微软告上法庭。最终，双方庭外和解，微软不再祸害 Java，J++ 停止更新。

但有一点不得不承认，微软在 Windows 上的 JVM 性能是当时最好的，因为操刀 J++ 的是 Anders Hejlsberg，他是全世界最顶级的程序员。微软为了不与 Java 开启的受控（Managed）代码浪潮擦肩而过，于是，转身又推出了 C# 和.NET。

C# 的初版本简直和 Java 一模一样，一个 Java 程序员几乎不用培训就可以成为一个 C# 程序员。所以，从语言的角度来说，最初的 C# 并没有对行业做出什么贡献。

不过，既然有 Anders Hejlsberg 在背后，事情当然不会就这么简单收尾。C# 在语言特性上开始一路狂奔，一个更强大的 C# 崭露头角。像 Lambda、类型推演这些特性早早就落户 C# 了。

然而，C# 时运不济，它的上升期遇到了微软的下降期。越来越多的公司选择了 Java，越来越多的程序员拥抱了 Java，而语言模型上表现优秀的 C# 则遭遇了冷落。

当年 Java 号称“一个语言，多个平台”，而.NET 则是“一个平台，多个语言”。结果呢，.NET 的“一个平台”并不足以吸引更多的公司和程序员的投入，除了微软自己，其他在上面开发语言的尝试通常都是浅尝辄止。而 JVM 虽然目标不是为了多语言，但丝毫不妨碍很多人在上面开发新语言，比如，Groovy、Scala、Clojure 等等。

最终，**JVM 成了“多个语言，多个平台”**。随着微软的逐步开放，.NET 也开始迈向了多平台，**C# 也成了一门跨平台的语言**，遗憾的是，为时已晚，Java 已经成就了一番霸业。

如果出于学习的目的，C# 绝对是值得一学的程序设计语言，毕竟微软在语言设计上还是很有一套的。Java 语言的进化是非常缓慢的，尤其是 SUN 的衰退又耽误了很多年。所以，从语言特性上来看，说 C# 领先 Java 十年并不夸张。

JavaScript

JavaScript 从诞生之日起就扮演着一个不受待见的角色，Brendan Eich 发明 JavaScript 完全是为了应付工作，因为他当时供职的 Netscape 需要让网页上的元素动起来。

雷锋和雷锋塔有什么关系？Java 和 JavaScript 有什么关系？这是一个经常被人提起的段子，但实际上，JavaScript 和 Java 真的有关系，关系就是蹭热度。当时的 Java 给世界描绘了一个美好的未来，让无数人心潮澎湃，JavaScript 就想借一下 Java 的东风。

JavaScript 仅仅用 10 天就设计出来，所以，在它的实现中，包含了各种奇怪的问题。不过，它还是体现出了 BrendanEich 的功底，比如，JavaScript 提供了对各种编程范式的支持。其实，他真正想做的是一门函数式编程的语言，但向现实妥协的结果就是，借助了 C 风格的语法，函数式编程的底子却留在了 JavaScript 里。

虽然在今天看来，在浏览器上 JavaScript 一枝独秀，但当年它也是有竞争对手的，那个年代无处不在的微软，出手做了一个 VBScript。但是，如同微软错过了互联网时代一样，与 Windows 结合更加紧密的 VBScript 也在这场竞争中败下阵来。

当年，与 JavaScript 联系在一起的，更多的是像走马灯之类的页面特效。让 JavaScript 真正第一次得到重视是 Ajax 这门技术。Ajax 的出现，让页面的元素可以与远程的服务器进行交互，JavaScript 开始由一个小玩具变成了一门值得研究的技术，前端的表现力得到了大幅度的提升。

但很长一段时间里，JavaScript 一直都不是一门正式的语言，对于很多人来说，它只是要做前端时顺便学习的语言。这种现象一直持续到 Node.js 的诞生。

Node.js 其实是一个集成商，它之所以能有良好的表现要归功于 V8 这个 JavaScript 引擎。而 V8 的出现则要归因于 Google 对于网络应用前景的格局。

想当年的浏览器大战，Netscape 和 IE 拼得你死我活，最终 IE 凭借 Windows 的优势成了赢家，Netscape 也退出了历史舞台。然而，胜利后的微软认为天下已平，竟然解散了 IE 的团队，导致了程序员们要在很长时间内忍受 IE 这个既不标准又慢的浏览器。

这就给后来居上者留下了空间。最有名的两个后来者，一个是 Netscape 的转世 Firefox，另外一个就是 Google 出品的 Chrome。

Chrome 认为未来的页面一定要有更强的表现力，所以，一个高效强大的浏览器是必需的。既然慢是个大问题，Chrome 就着力解决慢这个问题，甚至不惜开发了一个新的 JavaScript 引擎，也就是 V8，它的重点就是解决 JavaScript 执行慢的问题。可等微软看懂 Google 的操作，幡然悔悟，重新投入浏览器的开发之时，大势已去。Chrome 成了新的霸主。

Chrome 有一点做得很好，V8 一开始就是一个独立的 JavaScript 引擎。所以 Node.js 才可以很方便地把它借鉴过去。除了 V8 的性能优势，Node.js 还引入了异步 IO 的模型，这刚好与 JavaScript 事件驱动的特点相吻合。

Node.js 刚一登场便赢得了满堂喝彩。因为，人们认识到，JavaScript 原来不只能在浏览器中运行，也可以跑在服务器端。很快，**NPM 这个包管理器登场，降低了众多开发者参与的门槛，JavaScript 迎来了属于自己的爆发**，各种各样的程序库让人眼花缭乱。

前端开发也由少数人的爱好，成为了一个专属的职位，像 React、Angular、Vue 等框架的出现，更是让前端开发有了工程的味道，而不再是小打小闹了。

一旦 JavaScript 突破了浏览器的限制，给人们的想象空间就大了许多。除了服务器端，有人想把 JavaScript 用在嵌入式开发中，有人想把它用在手机开发中。**JavaScript 成了一门全平台覆盖的语言**，大有一统天下的架势。

不过，JavaScript 作为一门语言，其问题之多也是由来已久的。虽然 JavaScript 本身也在不断进化，但沉重的历史包袱让很多人都想开发出新的语言去替代它。所以，在 JavaScript 社区中，很多人把它看成了一种 Web 上的汇编语言，把新的语言编译成 JavaScript，这样，就可以在浏览器上运行了。从早先的 CoffeeScript 到现在的 TypeScript，甚至新一代的 JavaScript 标准都是以这种方式进行开发的。

当然还有人有更高的追求，他们认为仅仅在语言层面屏蔽 JavaScript 是不够的。**WebAssembly 就是想成为 Web 上真正的汇编**，真正取 JavaScript 而代之，事实上，它也得到了很多人的支持。不过，这种努力至今仍在继续中，还有很长的路要走。

JavaScript 就是这样，从一出生就不受待见，到今天，很多人仍想把它干掉。但这并不妨碍它在软件开发的历史中写下浓墨重彩的一笔。

Go 和 Rust

在系统编程方面，C 语言是当之无愧的霸主，然而，C 语言已经快 50 岁了。在计算机这个快速变化的行业里，50 年长得令人发指。在这 50 年中，C 从被人质疑发展到如日中天，再到应用开发的地位逐步被取代。如今，它只在系统编程有着无可替代的作用。事实上，人们也一直想着替代它。

C 的强项是对于计算机模型的适度抽象，弱项却是在程序的组织上。因为在 C 诞生那个年代，程序的规模还不算太大。然而 C 的成功却让程序的规模越来越大，大到超出了 C 语言的能力范畴。于是，有人想着把面向对象加到 C 语言里，扩大程序的组织规模。这方面的尝试，我们都熟悉的是 C++。

不过，C++ 只风光了一段时间，就被 Java 盖了过去。C++ 本身有一段时间变成了语言特性的试验田，泛型编程，尤其是模板元编程的出现，一度让人怀疑人生。它成了高手极度喜爱，普通人一脸懵硬着头皮写的程序语言。

但更重要的是，C++ 背负了 C 语言所有的历史负担。所以，很多 C 的问题在 C++ 里面依然存在，比如，内存管理。虽然 C++ 有各种补丁方案，但你必须对 C++ 极其了解，才能写好 C++，然而，这个要求对于一个工程化的语言来说，实在是太高了。

所以，无论是 C 还是 C++，都是在执行性能上无可挑剔，在代码编写上一地鸡毛，人们还是需要一门更有开发效率的系统编程语言。

时间来到新千年，又有人出手想代替 C 语言，这回出手的人物背景强大，他就是 Ken Thompson，C 语言的亲爹。2009 年，如日中天的 Google 推出了 Go 语言，再加上 Ken Thompson 和 Rob Pike 这样早期的 Unix 先驱站在它背后，Go 语言的前景给人无限的遐想。

Go 语言的语法设计是简单的，基本上，你花一个晚上就可以把 Go 语言完整地学习一遍。**它在接口设计和并发上的处理方式都给人眼前一亮的感觉**。人们热切地期盼着它成为下一个系统编程语言的霸主。

但事实并没有像人们想象地那样发生，除了初生之时引起了一片欢呼，Go 语言很长一段时间都在低位徘徊。比较有趣的是，中国有很多开发者对于 Go 的喜爱程度极高，一度让 Go 语言在中国的热度远远超过了全球的平均水平。之所以 Go 没有很快赢得人们的关注，因为

它关注的系统编程领域并没有太多的机会留给它，人们嘴上喊着热爱，手里还依然用 C 写着代码。

不过，机会总是留给做好准备的人，语言也不例外。随着 Docker 这套虚拟化软件登上历史舞台，Go 语言终于有了用武之地。人们开始意识到，**原来云计算领域还有一些基础设施要写，用 C 的话，不好维护；用 Java 的话，浪费资源；Go 恰如其分地解决了大部分问题**。

一批新生代的基础设施纷纷出炉，除了 Docker 之外，还有帮助人们实现容器部署的 Kubernetes，也就是 k8s，还有辅助 Service Mesh 的 istio 等等。

虽然在云计算基础设施中，Go 赢得了一席之地，这属于开辟了一片蓝海。在传统系统编程的红海中，Go 语言其实并没有做出什么特别的成绩，对于实时性和性能要求极高的领域，Go 语言有一个拿不出手的弱项，也就是它的 GC。

自动的内存管理固然是简化程序员工作的一项重要手段，但对于系统编程这个领域而言，GC 显然还没有表现得能够赢得大家的信任，而且，在可见的未来，也不会有明显的起色。

所以，在系统编程领域替代 C 的征程上，大家都还有机会。这条赛道上目前最有力的竞争者是 Rust。

Rust 出自 Mozilla，这是浏览器 Firefox 背后的公司，它原本是 Mozilla 员工 Graydon Hoare 的个人项目，后来得到了公司赞助，由一个练手的项目成为了一个正式的项目。

Rust 对初学者并不友好，对于习惯“少废话、先动手”的程序员而言，Rust 的初体验可能一点都不好，按照习惯方式写出来的代码很可能是无法编译的。比如，Rust 的“变”量缺省是不变的，再有，想写好 Rust 程序，先要了解所有权的概念。不过，也恰恰是因为这些限制，让 Rust 写出来的程序犯下低级错误的概率大大降低了。

如果你理解系统编程面临的问题，以及现代软件开发的趋势，你会发现，Rust 提供的选项很好地规避了许多问题。比如，之所以要用不变性，是因为它可以规避掉很多因为“变”带来的问题，这是函数式编程给软件开发贡献的一个重要思路。再比如，所有权的概念也是为了防止一块内存不同的人去改，造成各种问题，同时，也给内存管理提供了新的思路。

内存不能让程序员管，这已经成了共识，但主流的 GC 方案又不能满足系统编程的需要，Rust 则给出了第三种方案，把内存当作一种资源，申请下来就初始化好，出了生命周期就销毁掉。之所以能够做到这点，还是要拜 Rust 强大的编译器所赐，因为所有权的存在，编译器可以很好地分析出内存到底该什么时候释放。

Rust 成为系统编程语言的有力竞争者还有一个原因，它背靠着 LLVM。LLVM 是一套编译器的基础设施，它的出现是因为传统的工具链 GCC 太过沉重。LLVM 把编译器的前端和后端分离开来，语言开发者只要关注前端，设计好各种语言特性，就可以利用 LLVM 的后端进步的优势，比如，不断优化带来的性能提升。对系统编程语言来说，一个重点就是可移植性。

系统编程一个重要的战场就是各种嵌入式设备，而绝大多数设备都只支持 C/C++ 语言。一个重要的原因就是谁来移植编译器，C/C++ 的后端常常是厂商提供支持的，而其他语言则多半无人理睬。现在有了 LLVM 的基础设施，一个芯片厂商只要支持了 LLVM 的后端，用 LLVM 前端开发出的语言也就都得到了支持。这对于新兴语言来说，绝对是一个巨大的好消息。

Rust 在语言层面表现出来的安全特性，帮它赢得了像微软、亚马逊这样大厂的注意；占用资源少的内存管理方式，让一些人开始尝试使用它编写 Linux 驱动；更多的移植可能，也让它成为了嵌入式开发的一种考虑。在这场 C 语言替代者的竞争中，Rust 值得期待！

总结时刻

今天的内容主要是为了让大家放松一下，所以，我们也不做内容上的总结了。

每个程序员除了学习当下要用到的知识之外，一般都会对自己的未来做一些技术储备，其中，判断技术趋势就是我们在投资未来时的一个重要参考。

如何才能更好地判断未来技术发展趋势呢？就是去知道一些技术的发展历史。

思考题

最后，我想请你分享一下，你看好哪门语言未来的发展？为什么？欢迎在留言区分享你的想法。

感谢阅读，如果你觉得这一讲的内容对你有帮助的话，也欢迎把它分享给你的朋友。

更多课程推荐

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 11 | DSL：你也可以设计一门自己的语言

下一篇 12 | 编程范式：明明写的是Java，为什么被人说成了C代码？

精选留言 (12)

写留言

Kafka²⁰²⁰

2020-06-19

语言的流行通常需要一个杀手级的应用，比如RoR之于Ruby，Docker之于Go，Spring之于Java

作者回复: 是这样的，杀手级应用助力语言爆发。

6

明20-06-22

看了老师的文章 我决定去学学rust

展开

作者回复: 加油，希望你有收获！

2

hello world

2020-06-21

python应该会因为AI继续焕发第二春下去

展开

作者回复: 这点还真是，Python找到了AI这个增长点。

2

escray

2020-06-23

从语言本身来说，我还是比较喜欢 C# 的，多年以前也写过 ASP.NET 的程序。不过 C# 确实“时运不济”，而且替技术人员背了不少黑锅，比如 “ASP.NET 就是慢”之类的。我还是挺喜欢 C# 后来的泛型、Lamda 表达式、LINQ 之类的“语法糖”。

从就业市场的角度讲，C# 的程序员的平均薪水一直不怎么高（可能是因为上手比较容...

展开

1

有学识的兔子

2020-06-20

很感谢老师的分享，开阔了不少眼界。
由于自己的行业相对比较传统，没有接触特别前沿的领域和技术，不好判定语言的走势。
就程序设计语言，又让我回想起那句：语言即是程序库，程序库即是语言。语言的出现是伴随着工程问题的出现，通过新的语言特性更好地解决现有的工程问题。
前段时间阅读了《c++的程序设计与演化》，较直观地看到语言的前世今生，了解语言的...

展开

作者回复: 一个程序员，保持对技术趋势的关注很重要。

1

qinsi
2020-06-20

Wasm最初的规范中不支持GC，所以Rust和C/C++天生都可以"直出"Wasm，Go和Java等就不行。但要让前端开发去掌握系统编程语言还是要求高了点且没必要（其他的选择像是AssemblyScript虽然门槛低一些但还不完整；Yew可以重用一些Elm的经验，但Elm本身也太小众了）。有GC的话虽然影响性能，但确实可以降低语言的门槛。

展开

作者回复: 感谢你的补充信息！

1

再来二两杜康酒
2020-06-19

个人以为语言应差异化竞争，百花齐放，多出些最佳实践和案例相互成就。
我比较看好微软的.NET 5.0，立足实际又不乏野心，公司本身也具备做好的条件和能力。
不要太在意短期吸粉与排名，做好自我完善和迭代也许就水到渠成了。

展开

作者回复: .NET 本身的能力只是一方面，还要看微软怎么让它摆脱单一平台的刻板印象。

1

迈步
2020-06-19

看好JavaScript，感觉它的出生起，就奠定了它以无声胜有声，无意胜有意的大道。我走我的路，让别的语言说去吧。

展开

作者回复: 等着看未来的发展吧！

1

阳仔
2020-06-19

每种语言都有自己解决问题的范畴
哪个语言更好其实没有很强的可比性，因为每个语言解决的问题不一样

从发展潜力上看，个人还是比较看好golang和rust
虽然我自己使用的语言主要是JAVA，kotlin等jvm语言

展开

1

梦醒十分
2020-06-19

好文章，的多读几遍。

展开

作者回复: 也欢迎分享啊！

1

Jxin
2020-06-19

1.微软早期本就是屡战屡败，屡败屡战顽强存活下来的...失利和失败在那个时候也算是家常便饭了。

2.很遗憾，语言这个话题不敢乱说。虽然也会几门语言，但也就看得懂语法，能写点玩具的程度。也就java熟悉点。只能说，无论以后各种语言多么璀璨。java都挺难退出历史舞...

展开

作者回复: Java 在可见的未来都会长期存在，它的生态太庞大了。

1

NIU
2020-06-19

一度认为能前后端通吃的语言吧，比如Javascript, 更多新潮需要特性支持的Swift。

作者回复: JS 能力上没问题，语言拖后腿了。

1