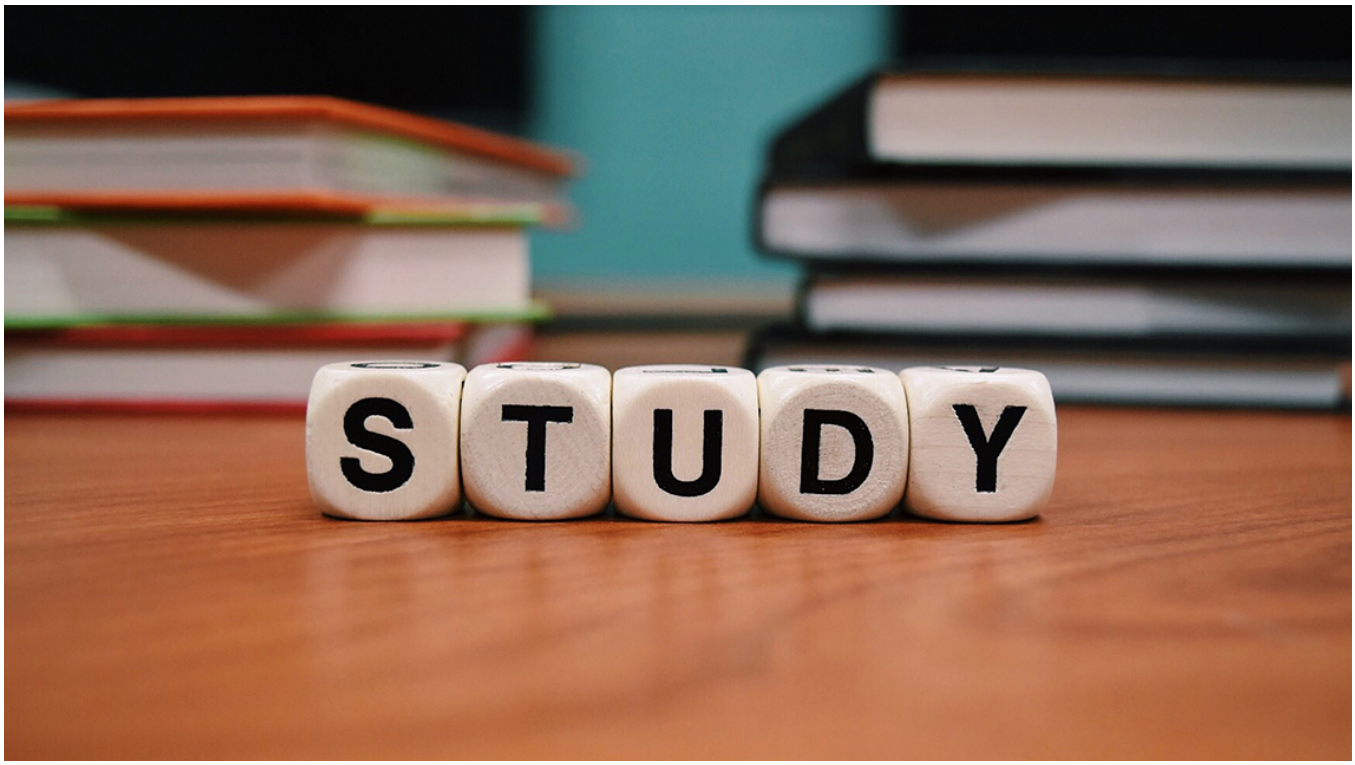
推荐阅读：每个程序员都该知道的事

2017-10-24 陈皓，杨爽



在专栏开篇词中，我提到，在每个月月中，我会推荐一些有价值的内容，供你参考。这个月，我将为你推荐五篇阅读文章，这五篇文章我觉得都是比较不错的经验总结，是我们每一个技术人员都需要知道的东西。它们分别是：

- Stack Overflow 上开出来的一个经典书书单；
- 美国某大学教授给计算机专业学生的一些建议，其中有很多的资源；
- LinkedIn 的高效代码复查实践，很不错的方法，值得你一读；
- 一份关于程序语言和 bug 数相关的有趣的报告，可以让你对各种语言所有了解；
- 最后是一本 C++ 性能优化的非常有价值的电子书。

每个程序员都应该要读的书

在 Stack Overflow 上有一个问题 What is the single most influential book every programmer should read，网址为： https://stackoverflow.com/questions/1711/what-is-the-single-most-influential-book-every-programmer-should-read

虽然这个问题被关闭了，但是这是一个非常热门的问题。排在第一个的人给了一大串书的列表，看上去着实吓人，不过都是一些相当经典相当有影响力的书，在这里我重罗列一些我觉得你必需要看的。

- 《代码大全》虽然这本书有点过时了，而且厚到可以垫显示器，但是这是一本绝对经典的书。
- 《程序员修练之道》这本书也是相当的经典，我觉得就是你的指路明灯。
- 《计算机的构造和解释》经典中的经典，必需读的书。
- 《算法导论》美国的本科生教材，这本书应该也是中国计算机学生的教材。
- 《设计模式》这本书是面向对象设计的经典书。
- 《重构》代码坏味道和相应的代码的最佳实践。
- 《人月神话》这本书可能也有点过时了。但还是经典书。
- 《代码整洁之道》细节之处的效率，完美和简单。
- 《Effective C++》/《More Effective C++》C++ 中两本经典得不能再经典的书。也许你觉得 C++ 复杂，但这两本书中带来对代码稳定性的探索方式让人非常受益，因为这种思维方式同样可以用在其它地方。以至于各种模仿者，比如《Effective Java》也是一本经典书。
- 《Unix 编程艺术》、《Unix 高级环境编程》也是相关的经典。

还有好多，我就不在这里一一列了。你可以看看其它的答案，我发现自己虽然读过好多书，但也有好些书没有读过，这个问答对我也很有用。

每个搞计算机专业的学生应有的知识

What every computer science major should know，每个搞计算机专业的学生应有的知识，网址为：http://matt.might.net/articles/what-cs-majors-should-know/

本文作者马修·迈特（Matthew Might）是美国犹他大学计算机学院的副教授，2007 年于佐治亚理工学院取得博士学位。计算机专业的课程繁多，而且随着时代的变化，科目的课程组成也在不断变化。如果不经过思考，直接套用现有的计算机专业课程列表，则有可能忽略一些将来可能变得重要的知识点。为此，马修力求从四个方面来总结，得出这篇文章的内容。

1. 要获得一份好工作，学生需要知道什么？
2. 为了一辈子都有工作干，学生需要知道什么？
3. 学生需要知道什么，才能考进研究生院？
4. 学生需要知道什么，才能对社会有益？

这篇文章不仅仅对刚毕业的学生有用，对有工作经验的人同样有用，这里我把这篇文章的内容摘要如下。

首先，对于我们每个人来说，作品集（Portfolio）会比简历（Resume）更有参考意义。所以，在自己的简历中应该放上自己的一些项目经历，或是一些开源软件的贡献，或是你完成的软件的网址等。最好有一个自己的个人网址，上面有一些你做的事，自己的技能，经历，以及你的一些文章和思考会比简历更好。

其次，计算机专业工作者也要学会与人交流的技巧，包括如何写演示文稿，以及面对质疑时如何与人辩论的能力。

最后，他就各个方面展开计算机专业人士所需要的硬技能：工程类数学、Unix 哲学和实践、系统管理、程序设计语言、离散数学、数据结构与算法、计算机体系结构、操作系统、网络、安全、密码学、软件测试、用户体验、可视化、并行计算、软件工程、形式化方法、图形学、机器人、人工智能、机器学习、数据库，等等。详读本文可以了解计算机专业知识的全貌。

这篇文章的第三部分简直就是一个知识资源向导库，给出了各个技能的方向和关键知识点，你可以跟随着这篇文章里的相关链接学到很多东西。

LinkedIn 高效的代码复查技巧

LinkedIn's Tips for Highly Effective Code Review，LinkedIn 的高效代码复查技巧，网址为：https://thenewstack.io/linkedin-code-review/

对于 Code Review，我有一篇文章 《从 Code Review 谈如何做技术》，讲述了为什么 Code Review 是一件很重要事情。今天推荐的这篇文章是 LinkedIn 公司的相关实践。

这篇文章介绍了 LinkedIn 公司内部实践的 Code Review 形式。具体来说，LinkedIn 的代码复查有以下几个特点。

- 从 2011 年开始，强制要求在团队成员之间做代码复查。Code Review 带来的反馈意见让团队成员能够迅速提升自己的技能水平，这解决了 LinkedIn 各个团队近年来因迅速扩张带来的技能不足的问题。
- 通过建立公司范围的 Code Review 工具，这就可以做跨团队的 Code Review。既有利于消除 bug，提升质量，也有利于大家对代码的学习和技能的传播。
- Code Review 的经验作为员工晋升的参考因素之一。
- Code Review 的一个难点是，Reviewer 可能不了解某个修改的背景和目的。所以 LinkedIn 要求代码签入版本管理系统前，就对其做清晰的说明，以便复查者了解其目的，促进 Review 的进行。

我认为，这个方法实在在太赞了。因为，我看到很多时候，Reviewer 都会说不了解对方代码的背景或是代码量比较大而无法做 Code Review，然而，却没有找到相应的方法解决这个问题。

LinkedIn 对提交代码写说明文档这个方法是一个非常不错的方法，因为代码提交人写文档的过程其实也是重头梳理的过程。我的个人经验是，写文档的时候通常会发现自己把事儿干复杂了，应该把代码再简化一下，于是就会回头去改代码。是的，写文档就是在写代码。

- 有些 Code Review 工具所允许给出的反馈只是代码怎样修改以变得更好，但长此以往会让人觉得复查提出的意见都表示原先的代码不够好。为了提高员工积极性，LinkedIn 的代码复查工具允许提出“这段代码很棒”之类的话语，以便让好代码的作者得到鼓励。我认为，这个方法也很赞，正面鼓励的价值也不可小看。

- 为 Code Review 的结果写出有目的性的注释。比如“消除重复代码”，“增加了测试覆盖率”，等等。长此以往也让团队的价值观得以明确。

- Code Review 中，不但要 Review 提交者的代码，还要 Reivew 提交者做过的测试。除了一些单元测试，还有一些可能是手动的测试。提交者最好列出所有测试过的案例。这样可以让 Reviewer 可以做出更多的测试建议，从而可以提高质量。

- 对 Code Review 有明确的期望，不过分关注细枝末节，也不要炫技，而是对要 Review 的代码有一个明确的目标。

编程语言和代码质量的研究报告

A Large-Scale Study of Programming Languages and Code Quality in GitHub，编程语言和代码质量的研究报告，网址为：https://cacm.acm.org/magazines/2017/10/221326-a-large-scale-study-of-programming-languages-and-code-quality-in-github/

这是一项有趣的研究。有四个人从 Github 上分析了 728 个项目，6300 万行代码，近 3 万个提交人，150 万个 commits，以及 17 种编程语言（如下图所示），然后，他们想找到编程语言对软件质量的影响。

	Project details		Commits		
Language	#Projects	#Devs (K)	#Commits (K)	#Insertion (MLOC)	#BugFixes (K)
C	220	13.8	447.8	75.3	182.6
C++	149	3.8	196.5	46.0	79.3
C#	77	2.3	135.8	27.7	50.7
Objective-C	93	1.6	21.6	2.4	7.1
Go	54	6.6	19.7	1.6	4.4
Java	141	3.3	87.1	19.1	35.1
CoffeeScript	92	1.7	22.5	1.1	6.3
JavaScript	432	6.8	118.3	33.1	39.3
TypeScript	14	2.4	3.3	2.0	0.9
Ruby	188	9.6	122.1	5.8	30.5
Php	109	4.9	118.7	16.2	47.2
Python	286	5.0	114.2	9.0	41.9
Perl	106	0.8	5.5	0.5	1.9
Clojure	60	0.8	28.4	1.5	6.0
Erlang	51	0.8	31.4	5.0	8.1
Haskell	55	0.9	46.1	2.9	10.4
Scala	55	1.3	55.7	5.3	12.9
Summary	728	28	1574	254	564

然后，他们还对编程语言做了一个分类，想找到不同类型的编程语言的 bug 问题。如下图所示：

Language classes	Categories	Languages
Programming paradigm	Imperetive procedural	C, C++, C#, Objective-C, Java, Go
	Imperetive scripting	CoffeeScript, JavaScript, Python, Perl, Php, Ruby
	Functional	Clojure, Erlang, Haskell, Scala
Type checking	Static	C, C++, C#, Objective-C, Java, Go, Haskell, Scala
	Dynamic	CoffeeScript, JavaScript, Python, Perl, Php, Ruby, Clojure, Erlang
Implicit type conversion	Disallow	C#, Java, Go, Python, Ruby, Clojure, Erlang, Haskell, Scala
	Allow	C, C++, Objective-C, CoffeeScript, JavaScript, Perl, Php
Memory class	Managed	Others
	Unmanaged	C, C++, Objective-C

We omit TypeScript from language classification as it allows both explicit and implicit type conversion.

以及，他们还对这众多的开源软件做了个聚类，如下图：

Domain name	Domain characteristics	Example projects	Total projects
(APP) Application	End user programs	bitcoin, macvim	120
(DB) Database	SQL and NoSQL	mysql, mongodb	43
(CA) CodeAnalyzer	Compiler, parser, etc.	ruby, php-src	88
(MW) Middleware	OS, VMs, etc.	linux, memcached	48
(LIB) Library	APIs, libraries, etc.	androidApis, opencv	175
(FW) Framework	SDKs, plugins	ios sdk, coffeekup	206
(OTH) Other	–	Arduino, autoenv	49

对 bug 的类型也做了一个聚类，如下图：

	Bug type	Bug description	Search keywords/phrases	Count	% count
Cause	**Algorithm** (Algo)	Algorithmic or logical errors	Algorithm	606	0.11
	Concurrancy (Conc)	Multithreading/processing issues	Deadlock, race condition, synchronization error	11,111	1.99
	Memory (Mem)	Incorrect memory handling	Memory leak, null pointer, buffer overflow, heap overflow, null pointer, dangling pointer, double free, segmentation fault	30,437	5.44
	Programming (Prog)	Generic programming errors	Exception handling, error handling, type error, typo, compilation error, copy-paste error, refactoring, missing switch case, faulty initialization, default value	495,013	88.53
Impact	**Security** (Sec)	Runs, but can be exploited	Buffer overflow, security, password, oauth, ssl	11,235	2.01
	Performance (Perf)	Runs, but with delayed response	Optimization problem, performance	8651	1.55
	Failure (Fail)	Crash or hang	Reboot, crash, hang, restart	21,079	3.77
	Unknown (Unkn)	Not part of the above categories		5792	1.04

其中分析的方法我不多说了。我们来看一下相关的结果。

首先，他们得出来的第一个结果是，从查看 bug fix 的 commits 的个数情况来看，C、C++、Objective-C、PHP 和 Python 中有很多很多的 commits 都是和 bug fix 相关的，而 Clojure、Haskell、Ruby、Scala 在 bug fix 的 commits 的数明显要少很多。

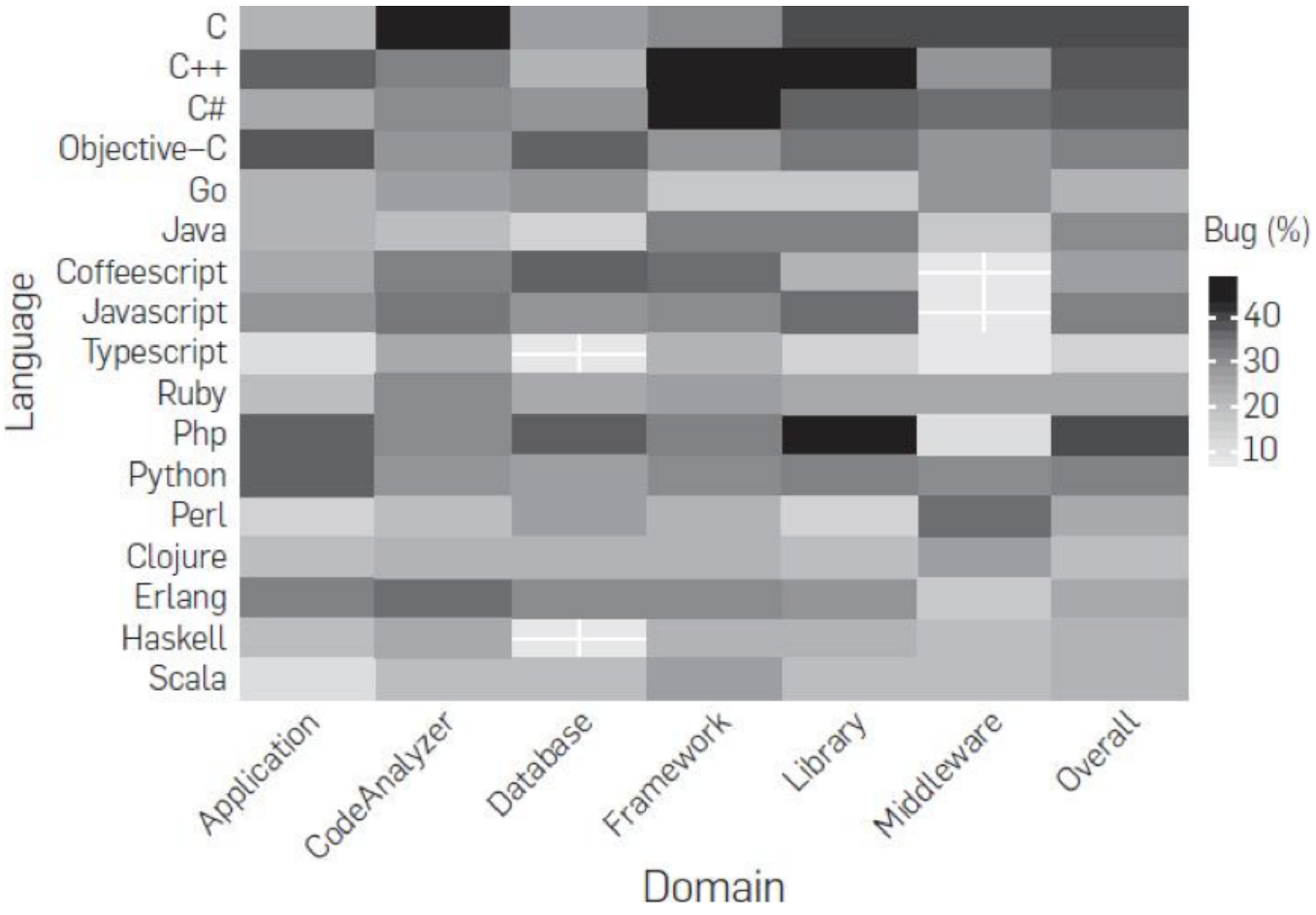
下图是各个语言出 bug 的情况。如果你看到是正数，说明高于平均水平，如果你看到是负数，则是低于平均水平。

Defective commits model	Coef. (Std. Err.)
(Intercept)	−2.04 (0.11)***
Log age	0.06 (0.02)***
Log size	0.04 (0.01)***
Log devs	0.06 (0.01)***
Log commits	0.96 (0.01)***
C	0.11 (0.04)**
C++	0.18 (0.04)***
C#	−0.02 (0.05)
Objective-C	0.15 (0.05)**
Go	−0.11 (0.06)
Java	−0.06 (0.04)
CoffeeScript	0.06 (0.05)
JavaScript	0.03 (0.03)
TypeScript	0.15 (0.10)
Ruby	−0.13 (0.05)**
Php	0.10 (0.05)*
Python	0.08 (0.04)*
Perl	−0.12 (0.08)
Clojure	−0.30 (0.05)***
Erlang	−0.03 (0.05)
Haskell	−0.26 (0.06)***
Scala	−0.24 (0.05)***

Response is the number of defective commits. Languages are coded with weighted effects coding. *AIC=10432, Deviance=1156, Num. obs.=1076.*
***$p < 0.001$, **$p < 0.01$, *$p < 0.05$

第二个结论是，函数式编程语言的 bug 明显比大多数其它语言的要好很多。有隐式类型转换的语言明显产生的 bug 数要比强类型的语言要少很多。函数式的静态类型的语言要比函数式的动态类型语言的程序出 bug 的可能性要小很多。

第三，研究者想搞清是否 bug 数会和软件的领域相关。比如，业务型的，中间件型、框架、lib，或是数据库。研究表明，并没有什么相关性。下面这个图是各个语言在不同领域的 bug 率。



第四，研究人员想搞清楚 bug 的类型是否会和语言有关系。的确如此，bug 的类型和语言是强相关性的。下图是各个语言在不同的 bug 类型的情况。如果你看到的是正数，说明高于平均水平，如果你看到的是负数，则是低于平均水平。

	Memory	Concurrency	Security	Failure
(Intercept)	−7.49 (0.46)***	−8.13 (0.74)***	−7.29 (0.58)***	−6.21 (0.41)***
Log commits	0.99 (0.05)***	1.09 (0.09)***	0.89 (0.07)***	0.88 (0.05)***
Log age	0.15 (0.06)*	0.19 (0.10)	0.30 (0.08)***	0.07 (0.06)
Log size	0.01 (0.04)	−0.08 (0.07)	−0.01 (0.05	0.14 (0.04)***
Log devs	0.07 (0.04)	0.09 (0.07)	0.07 (0.06)	−0.11 (0.04)*
C	1.71 (0.12)***	0.39 (0.22)	0.28 (0.18)	0.43 (0.13)**
C#	−0.12 (0.17)	0.81 (0.24)***	−0.42 (0.23)	−0.07 (0.16)
C++	1.08 (0.10)***	1.07 (0.18)***	0.40 (0.16)*	1.05 (0.11)***
Objective-C	1.40 (0.15)***	0.41 (0.28)	−0.14 (0.24)	1.10 (0.15)***
Go	−0.05 (0.25)	1.62 (0.30)***	0.35 (0.28)	−0.49 (0.24)*
Java	0.53 (0.14)***	0.80 (0.22)***	−0.07 (0.19)	0.15 (0.14)
CoffeeScript	−0.41 (0.23)	−1.73 (0.54)**	−0.36 (0.27)	−0.05 (0.19)
JavaScript	−0.16 (0.10)	−0.21 (0.16)	0.02 (0.12)	−0.15 (0.09)
TypeScript	−0.58 (0.62)	−0.63 (1.02)	0.37 (0.51)	−0.42 (0.41)
Ruby	−1.16 (0.19)***	−0.89 (0.29)**	−0.18 (0.21)	−0.32 (0.16)*
Php	−0.69 (0.17)***	−1.70 (0.34)***	0.11 (0.21)	−0.62 (0.17)***
Python	−0.48 (0.14)***	−0.25 (0.22)	0.36 (0.16)*	0.04 (0.12)
Perl	0.15 (0.35)	−1.23 (0.83)	−0.62 (0.45)	−0.64 (0.38)
Scala	−0.47 (0.18)**	0.63 (0.24)**	−0.22 (0.22)	−0.93 (0.18)***
Clojure	−1.21 (0.27)***	−0.01 (0.30)	−0.82 (0.27)**	−0.62 (0.19)**
Erlang	−0.60 (0.23)**	0.63 (0.28)*	0.62 (0.22)**	0.59 (0.17)***
Haskell	−0.28 (0.20)	−0.27 (0.32)	−0.45 (0.26)	−0.49 (0.20)*
AIC	2991.47	2210.01	3328.39	4086.42
Deviance	895.02	665.17	896.58	1043.02
Num. obs.	1081	1081	1073	1077
Residual deviance (NULL)	5065.30	2124.93	2170.23	3769.70
Language deviance	522.86	139.67	42.72	240.51

For all models the deviance explained by language type has $p < 0.0003076$.
***$p < 0.001$, **$p < 0.01$, *$p < 0.05$.

也许，这份报告可以在你评估语言时有一定的借鉴作用。

电子书：《C++ 软件性能优化》

Optimizing Software in C++ - Agner Fog - PDF，C++ 软件性能优化，http://agner.org/optimize/optimizing_cpp.pdf

这本书是所有 C++ 程序员都应该要读的一本书。这本书从事无巨细地从语言层面，编译器层面，内存访问层面，多线程层面，CPU 层面……讲述了如果对软件性能的调优。实在是一本经典的电子书。

Agner Fog 还写了其它几本和性能调优相关的书（你可以到这个网址下载：http://www.agner.org/optimize/ ）

- Optimizing subroutines in assembly language: An optimization guide for x86 platforms
- The microarchitecture of Intel, AMD and VIA CPUs: An optimization guide for assembly programmers and compiler makers
- Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs
- Calling conventions for different C++ compilers and operating systems

我今天推荐的内容比较干，需要慢慢吸收体会，最好能借鉴到实践中用用，相信会有更多的感悟和收获。你还对哪些方面的内容感兴趣，欢迎留言给我。我后面收集推荐内容的时候，会有意识地关注整理。

精选留言

陈俊鸿 37

一上来就做酷炫的项目，learn by doing，学得快，容易有成就感，但是基础不牢，这是Top-Down的学习方式。看书，学概念，啃理论，比较枯燥，容易忘记，学得慢，费时间，但可以修炼内功，这是Bottom-Up的学习方式。对于这两种学习方式的结合，想听听老师的见解。

2017-10-25

Chris 9

老师你好，希望有时间可以具体谈谈架构师（Java）方面的知识体系与进阶之路，谢谢！

2017-10-25

yeyuliunian 9

关于数据库方面的资料

2017-10-24

songyy 5

在语言项目 bug数量分析那里，某些static typing的语言（比如Java）比non static type的语音（比如ruby）bug多，还是蛮让人惊讶的。我觉得可能是因为，ruby这类的语言因为重构困难，让人不得不重视测试吧。

此外，cpp的bug那么多...除了语言设计本身的问题，我想和使用数量比较大也有关系：主要是library层级的代码，很容易发现一些bug

2017-10-26

Wilson_qqs 2

就是需要这么干的好文章，谢谢耗子哥

2018-01-12

作者回复

谢谢认可

2018-01-12

菡萏如佳人 2

满满干货，特别是代码复查部分受益良多

2017-10-24

李志博 2

对大数据存储和查询方面有兴趣

2017-10-24

whhbbq 1

"这段代码很棒"，很少听到review的人这样说的，学到了，自己会这么去做的。

2018-03-02

孔令南 1

希望老师可以谈谈闭源开发者（如peoplesoft、ebs等套件二次开发）转型之路该怎么走？

2017-10-25

阿兄 0

今天的内容是很干的，收获不少，解决一个计算机类推进书单的问题，还给出一个对语言选型的评判参考。

2018-05-07

shane 0

感谢，code review部分有所启发。

2018-01-14

夜漆黑 0

想要了解数据库方面的知识

2018-01-05

Tech 0

有没有c语言性能优化的推荐呢？

2017-11-21