Go vs. GoPlus(Go+)

×

许式伟

x@goplus.org

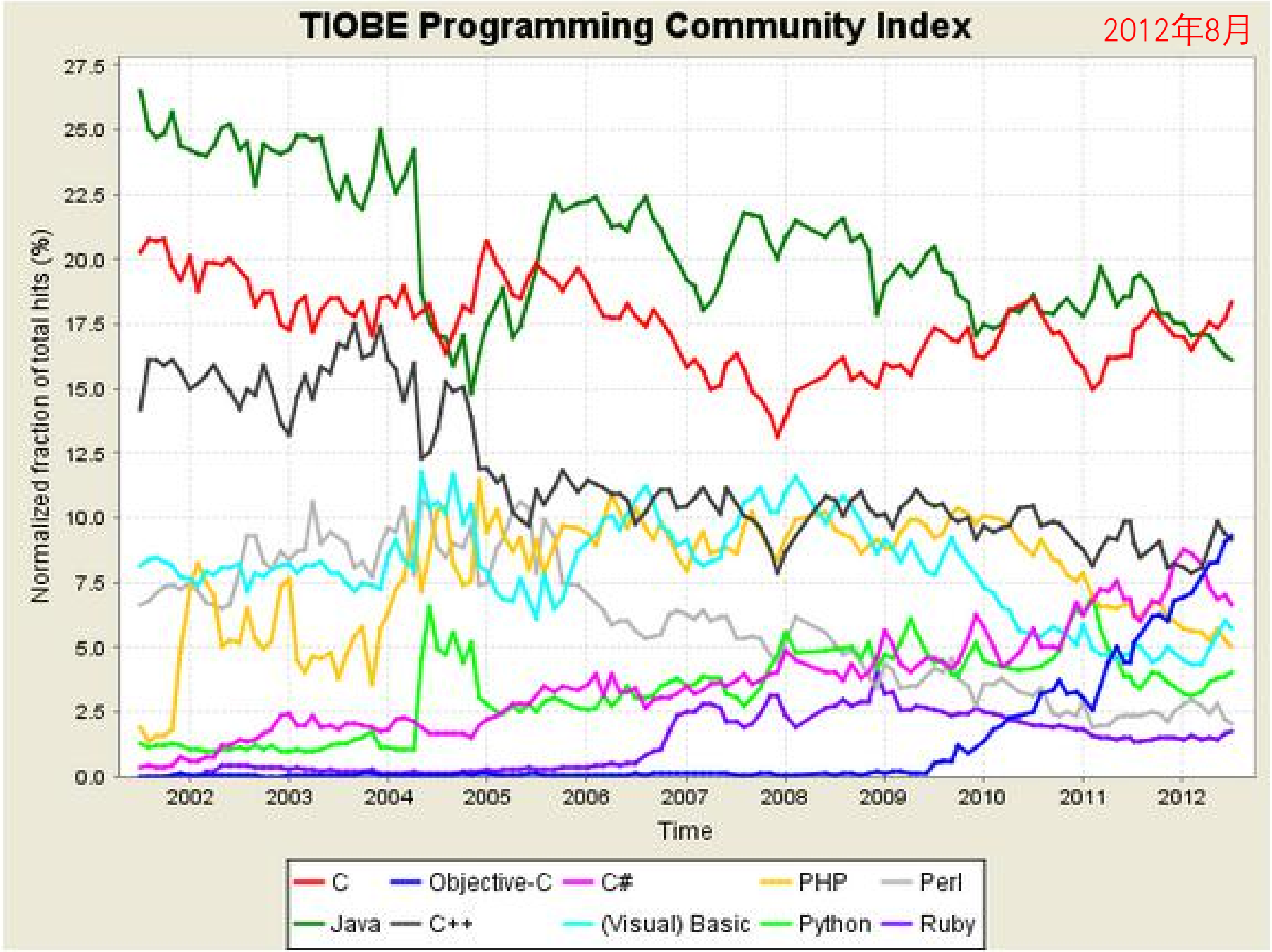
2021-6-27 北京

话外: 模板

Go 篇

谁是最成功的语言？

- 1970 – 至今
  - 什么语言是最成功的?
- C (1970)
- C++ (1979)
- Objective-C (1986)
- Java (1994)
- C# (2002)
- Go (2009)


TIOBE Programming Community Index
2012年8月
Normalized fraction of total hits (%)
27.5
25.0
22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
Time
C
Objective-C
C#
PHP
Perl
Java
C++
(Visual) Basic
Python
Ruby

Position Aug 2012	Position Aug 2011	Delta in Position	Programming Language	Ratings Aug 2012	Delta Aug 2011	Status
1	2	↑	C	18.937%	+1.55%	A
2	1	↓	Java	16.352%	-3.06%	A
3	6	↑↑↑	Objective-C	9.540%	+4.05%	A
4	3	↓	C++	9.333%	+0.90%	A
5	5	=	C#	6.590%	+0.55%	A
6	4	↓↓	PHP	5.524%	-0.61%	A
7	7	=	(Visual) Basic	5.334%	+0.32%	A
8	8	=	Python	3.876%	+0.46%	A
9	9	=	Perl	2.273%	-0.04%	A
10	12	↑↑	Ruby	1.691%	+0.36%	A
11	10	↓	JavaScript	1.365%	-0.19%	A
12	13	↑	Delphi/Object Pascal	1.012%	-0.06%	A
13	14	↑	Lisp	0.975%	+0.07%	A
14	26	↑↑↑↑↑↑↑↑↑↑↑↑	Visual Basic .NET	0.877%	+0.41%	A
15	15	=	Transact-SQL	0.849%	+0.03%	A
16	18	↑↑	Pascal	0.793%	+0.13%	A
17	11	↓↓↓↓↓↓	Lua	0.726%	-0.64%	A--
18	16	↓↓	Ada	0.649%	-0.05%	B
19	22	↑↑↑	PL/SQL	0.610%	+0.08%	B
20	29	↑↑↑↑↑↑↑↑↑	MATLAB	0.533%	+0.09%	B

May 2021	May 2020	Change	Programming Language	Ratings	Change
1	1		C	13.38%	-3.68%
2	3	↑	Python	11.87%	+2.75%
3	2	↓	Java	11.74%	-4.54%
4	4		C++	7.81%	+1.69%
5	5		C#	4.41%	+0.12%
6	6		Visual Basic	4.02%	-0.16%
7	7		JavaScript	2.45%	-0.23%
8	14	⇈	Assembly language	2.43%	+1.31%
9	8	↓	PHP	1.86%	-0.63%
10	9	↓	SQL	1.71%	-0.38%
11	15	⇈	Ruby	1.50%	+0.48%
12	17	⇈	Classic Visual Basic	1.41%	+0.53%
13	10	↓	R	1.38%	-0.46%
14	38	⇈	Groovy	1.25%	+0.96%
15	13	↓	MATLAB	1.23%	+0.06%
16	12	⇊	Go	1.22%	-0.05%
17	23	⇈	Delphi/Object Pascal	1.21%	+0.60%
18	11	⇊	Swift	1.14%	-0.65%
19	18	↓	Perl	1.04%	+0.16%
20	34	⇈	Fortran	0.83%	+0.51%

C 是 50 年来最成功的语言

- C 语言的黄金时间
  - 1970 至今，>50年，经久不衰

Go, Next C!

- Go 语言要学习的是 C，不是 Java
  - 多数的系统级语言包括Java、C#，其根本的编程哲学来源于C++，将C++的面向对象进一步发扬光大
  - 但是Go语言的作者们很清楚，C++ 真的没啥好学的，他们要学的是C语言。C语言经久不衰的根源是它足够简单。Go语言也要足够得简单
- Go，互联网时代的C，下一个C
  - 要再创 C 的辉煌

Go至少领先其他语言10年

- 尽管是 50 年来出现的语言非常之多，各有各的特色，让人眼花缭乱。但是我个人固执地认为，谈得上突破了 C 语言思想，将编程理念提高到一个新高度的，仅有 Go 语言而已
- 10 年后才会有语言试图去突破Go语言已经达到的新高度
  - 不会是现有的任何语言，而会是一门新兴语言

Go，超越我的想象

- Go 语言的各种语法特性显得那么深思熟虑、卓绝不凡，其对软件架构与工程的理解，让我深觉无法望其项背

- 处处带给我惊喜的语言

惊喜1：大道至简

- 基础哲学：继承自 C
  - 大道至简
- 显式表达
  - 任何封装都是有漏洞的
  - 最佳的表达方式就是最直白的表达方式
  - 不试图去做任何包装
  - 所写即所得的语言
- 少就是指数级的多
  - 最少特性原则
  - 如果一个功能不对解决任何问题有显著价值，那么就不提供

惊喜2：最对胃口的并行支持

- 我的并行编程历程
  - Erlang
  - CERL 1.0 (Erlang 风格并行的模仿)
  - CERL 2.0 (对 Erlang 风格并行的修正)
    - 后来发现，CERL 2.0 的并行编程理念，与 Go 完全一致（雏形版的 Go）
  - Go 语言
- 参考资料
  - ECUG 2011 讲座
    - 《从Erlang 到 CERL 到 Golang》
  - Collison预言：Go语言将在两年内制霸云计算领域
    - http://www.csdn.net/article/2012-09-14/2809984-will-go-be-the-new-go-to-programming-lan

惊喜3：interface

- 非侵入式接口
  - 只要某个类型实现了接口要的方法，那么我们说该类型实现了此接口。该类型的对象可赋值给该接口
  - 任何 Go 语言的内置对象都可以赋值给空接口 interface{}
- 接口查询
  - Windows COM 思想优雅呈现

惊喜4：极度简化但完备的OOP

- 废弃大量的 OOP 特性
  - 继承、构造/析构函数、虚函数、函数重载等
- 简化的符号访问权限控制
- 取消隐藏的 this 指针
  - 改为显式定义的 receiver 对象
- OOP 编程核心价值原来如此简单
  - 只是多数人都无法看透

惊喜5：错误处理规范

- 函数多返回值
- 内置的 error 类型
- defer

- 例子

```
f, err := os.Open(file)
if err != nil {
    ... // error processing
    return
}
defer f.Close()

... // process file data
```

惊喜6：功能内聚

- 用组合实现继承(包括虚拟继承)

```
type Foo struct { // 继承
        Base
        ...
}
type Foo struct { // 虚拟继承
        *Base
        ...
}
```

- 直达问题的本质，清晰易懂

惊喜7：消除了堆与栈的边界

- Go 语言里面你不需要关心，也并不清楚，变量在堆上还是栈上
- 与 Go 语言的显式表达并不矛盾
  - Go 语言强调的是对开发者的程序逻辑（语义）的显式表达，而非对计算机硬件结构的显式表达
  - 对计算机硬件结构的高度抽象，将更有助于 Go 语言适应未来计算机硬件发展的变化

惊喜8：C 语言的支持

- Go 语言是除了 Objective-C、C++ 这两门以兼容 C 为基础目标的语言外的所有语言中，对 C 语言支持最友善的一个
  - 什么语言可以直接嵌入 C 代码？没有，除了 Go
  - 什么语言可以无缝调用 C 函数？没有，除了 Go
- 对 C 语言的完美支持，是 Go 快速崛起的关键支撑

Go 语言小结

- 少就是指数级的多
  - 最精简
  - 学习门槛低
  - 心智负担低

- 关注焦点
  - 服务端开发
  - 大型软件工程

Go 语言的发展瓶颈

- 服务端开发不是一个大市场
  - 成也云计算，败也云计算

- Go 语言需要开辟新战场
  - 桌面开发（程序员最多的市场）
    - PC桌面开发
    - Mobile开发
    - Web开发（含小程序及轻应用）
    - IoT开发
  - 数据科学（当前最火的市场，推动Python到语言排行榜第二）
    - 大数据、人工智能
    - 数学软件

Go+ 篇

CONTENTS

01 语言的发展

02 数据科学的发展

03 Go+的设计理念

04 Go+实现的迭代

01

语言的发展

静态语言发展史 (TOP20)

- C (1970)
- C++ (1979)
- Objective-C (1986)
- Java (1994)
- C# (2002)
- Go (2009)
- Swift (2014)
- Go+ (2020)

大约每 6–8 年会出现一门新的影响力语言

脚本语言发展史 (TOP20)

- Visual Basic (1991)
- Python (1991)
- PHP (1994)
- JavaScript (1995)
- Ruby (1995)

脚本语言是集中性大爆发的
大概也就是在 Java 出现的那个年代

数据科学语言发展史 (TOP50)

- SQL (1973)
- SAS (1976)
- MATLAB (1984)
- Python (1991)
- R (2000)
- Julia (2009)
- Go+ (2020)

数据科学的发展古老而漫长
但开始进入加速期

语言发展史的启发

- 脚本语言是特定历史阶段下的产物，长远看静态语言更有生命力
- 数据科学是计算机的最初需求，历史悠久但进步缓慢
  - 因为数据大爆发的时代一直没有到来

02

数据科学的发展

数据科学的原始时期：数学软件时代

- SQL (1973)
- SAS (1976)
- MATLAB (1984)
- Excel (1985)

- Limited Domains (有限领域)，比如 BI (Business Intelligence)
- Limited Data (有限数据规模)

- 数据科学不是基础设施，而是数学应用软件
- 全能力：统计/预测/洞察/规划/决策/…

数据科学的基建时期：大数据的兴起

- Map/Reduce (2004)
- Hadoop (2006)
- Spark (2009)

- 大数据的兴起，是数据科学基础设施化的开始
- 以大规模处理能力为优先
- 功能上相对局限

数据科学的基建时期：深度学习的兴起

- TensorFlow/Python (2015)
- Torch/Python (2017)

- 迭代的是 y=F(x) 中的 F 预测
- 和大数据并不是互相取代的关系，而是能力加强

经济发展的体系要素

科学与技术	科技发展体系与范式		经济发展阶段		经济发展体系环境		人类活动	能源与信息
科学： 人类知识探索体系 解释自然现象的理论	思考	个人	农业时代		简单交易: 1 + 1 = 2 土地是核心产能，相应地理环境为主		劳力	太阳能 简单流程
技术 基于自然现象 有相应的科学理论 可 "编程" (programmable) 可 "执行" (executable) 用信息组织能源转化 目的是服务人类需求	实验	大学 系统化 职业化	工业时代	1.0 2.0 3.0	开放市场: 1 + 1 > 2 自由交易 无限分工 国际贸易 技能(人)是核心产能	英国 欧洲 美国	技能	化石能转化成电能 复杂系统与流程
	计算	研究型大学 国家实验室 新模式	信息时代	工业 4.0	市场 + 技术 + 数字化： 1 + 1 > 4 创新(人才) + 技术(资本)是核心产能 探索新的经济发展体系	美国 重心向亚洲转移	研发创新	新能源 数字化：辅助，替代，扩充人的能力
	数据 第四范式							

数据科学的大爆发时期：DT 时代

- 从前
  - Limited Domains (有限领域): 比如 BI (Business Intelligence)
  - Limited Data (有限数据规模): 比如 Excel、Matlab

- 未来
  - Full Domains (全领域): 智能应用 (Intelligent Application)
    - 典型代表：抖音、快手
  - Big Data (大规模数据)
  - Any Where (随处): 云 (Cloud)、智能手机 (Smart Phone)、嵌入式设备 (IoT)

数据科学的大爆发时期：DT 时代

- 互联网平民化
  - Internet Application (互联网应用)
  - 诞生了 BAT
- 数据科学平民化
  - Intelligent Application (智能应用)
  - 眼界局限了我的想象力，但不可否认它催生了字节跳动这样的新巨头
  - 而这，只是个开始
- 数据地位的变化
  - 数据是副产品 => 数据是原材料（石油），无处不在，深植于业务流

DT时代对数据科学的期待

- 数学软件
- 通用语言
- 大数据
- 深度学习

- 数据科学的未来，一定是通用语言与数学软件的融合
  - 完成数据科学的基础设施化

为什么有了 Python 还不够？

- 因为，Python 成不了基础设施（Infrastructure）
- 数据科学本质上是算力革命，是计算密集型的业务

- 数据科学进一步下沉，终局会是什么？

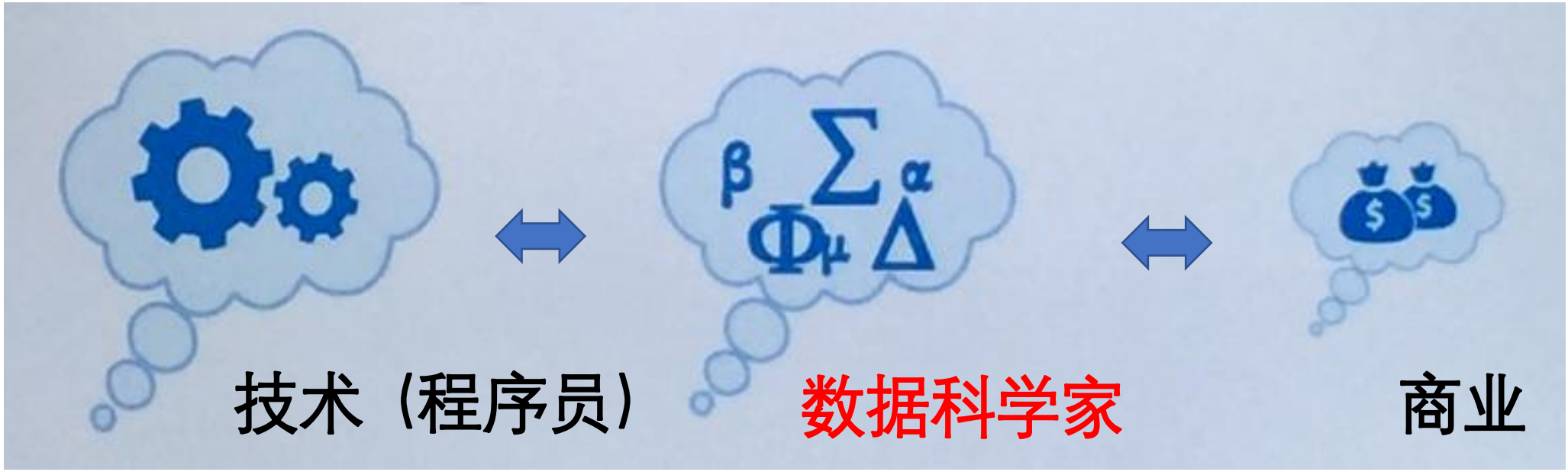
- 所以 Go+ 来了！

03

Go+的设计理念

数据科学家，连接技术与商业

- 培养程序员易，培养数据科学家难



Go+ 的核心设计理念

- Go+ 统一了程序员与数据科学家的语言，让双方自然对话

Go+ 的基础设计理念

- 静态语言，且语法完全兼容 Go
- 形式上比 Go 更像脚本，有更低的学习门槛（和 Python 相当）
- 更简洁的数学运算上的语法支持（相比 Go）
- 双引擎，既支持静态编译为可执行文件，也支持编译成字节码方式解释执行

静态语言，且语法完全兼容 Go

- 相比脚本语言，静态语言将拥有更强的生命力
- 静态语言中，Go 的语法最为精简，学习门槛也最低

要比 Go 有更低的学习门槛

- 形式上比 Go 更像脚本，有更低的学习门槛（和 Python 相当）
- 更简洁的数学运算上的语法支持（相比 Go）

- 当前 Go+ 已经拥有 3 个种子用户（13-14岁）

For example, the following is legal Go+ source code:

```
a := [1, 2, 3.4]
println(a)
```

How do we do this in the Go language?

```
package main

func main() {
    a := []float64{1, 2, 3.4}
    println(a)
}
```

- 有理数

```
a := 1r << 65   // bigint, large than int64
b := 4/5r       // bigrat
c := b - 1/3r + 3 * 1/2r
println(a, b, c)
```

- Map

```
x := {"Hello": 1, "xsw": 3.4} // map[string]float64
y := {"Hello": 1, "xsw": "Go+"} // map[string]interface{}
z := {"Hello": 1, "xsw": 3} // map[string]int
empty := {} // map[string]interface{}
```

- Slice

```
x := [1, 3.4] // []float64
y := [1] // []int
z := [1+2i, "xsw"] // []interface{}
a := [1, 3.4, 3+4i] // []complex128
b := [5+6i] // []complex128
c := ["xsw", 3] // []interface{}
empty := [] // []interface{}
```

- List comprehension

```
a := [x*x for x <- [1, 3, 5, 7, 11]]
b := [x*x for x <- [1, 3, 5, 7, 11], x > 3]
c := [i+v for i, v <- [1, 3, 5, 7, 11], i%2 == 1]
d := [k+","+s for k, s <- {"Hello": "xsw", "Hi": "Go+"}]

arr := [1, 2, 3, 4, 5, 6]
e := [[a, b] for a <- arr, a < b for b <- arr, b > 2]

x := {x: i for i, x <- [1, 3, 5, 7, 11]}
y := {x: i for i, x <- [1, 3, 5, 7, 11], i%2 == 1}
z := {v: k for k, v <- {1: "Hello", 3: "Hi", 5: "xsw", 7: "Go+"}, k > 3}
```

- For range

```
sum := 0
for x <- [1, 3, 5, 7, 11, 13, 17], x > 3 {
    sum += x
}
```

双引擎：既可静态编译，也可解析执行

- 既支持静态编译为可执行文件来执行，也支持编译成字节码方式进行解释执行

- 数据科学家喜欢单步执行（为什么？这并不是因为懒）
  - 请回忆一下所有数学软件的 UI

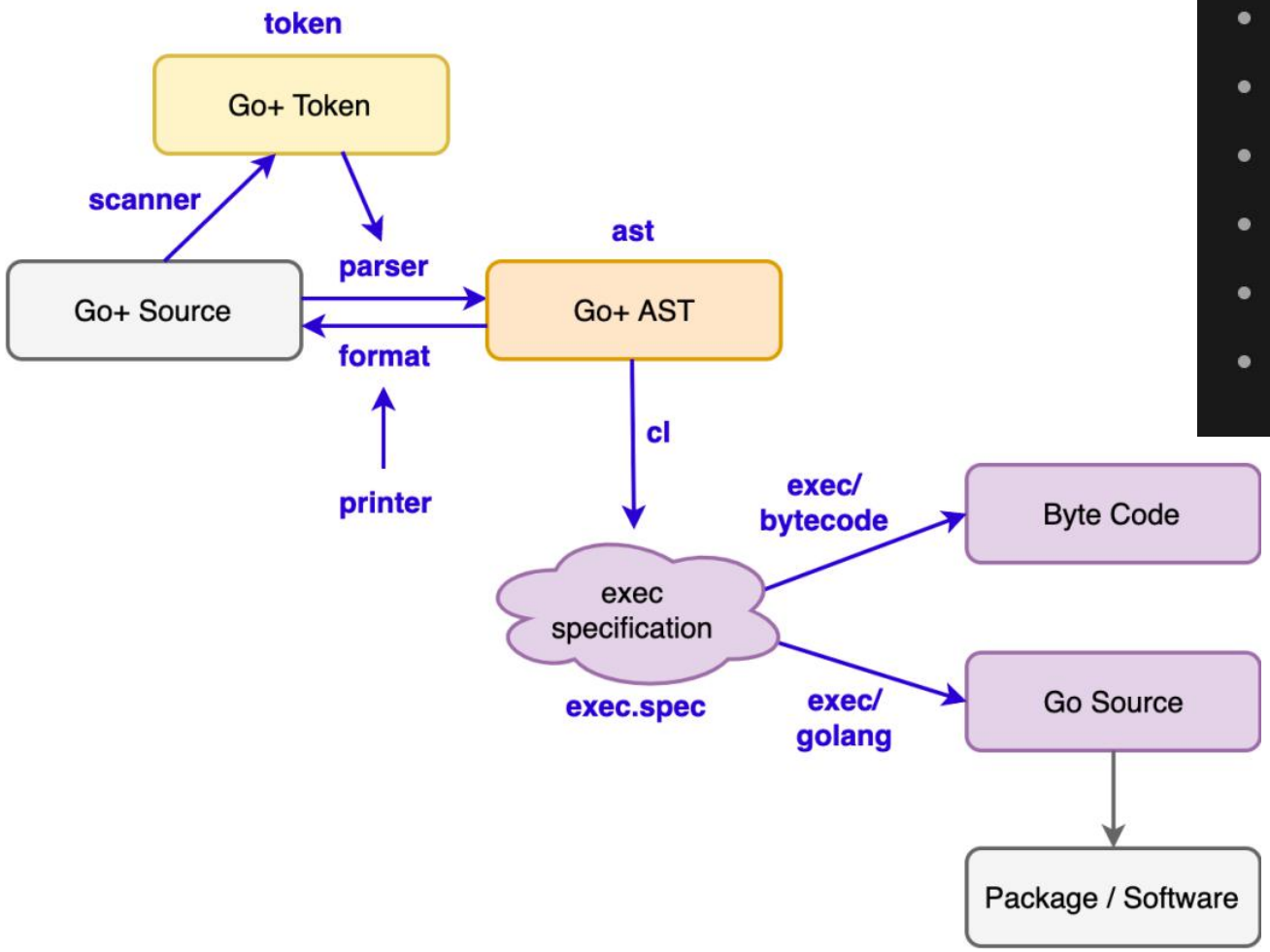
- 但最终交付仍然需要最大化的执行效率！
  - 因为：数据科学本质上是算力革命，是计算密集型的业务

04

Go+实现的迭代

Go+当前的架构设计

- github.com/goplus/gop/**token**
- github.com/goplus/gop/**scanner**
- github.com/goplus/gop/**parser**
- github.com/goplus/gop/**ast**
- github.com/goplus/gop/**format**
- github.com/goplus/gop/**printer**
- github.com/goplus/gop/**cl**
- github.com/goplus/gop/**exec.spec**
- github.com/goplus/gop/**exec/bytecode**
- github.com/goplus/gop/**exec/golang**



Go+进行中的重构

- exec.spec 不再是抽象的 SAX 接口，而是某个标准实现的 DOM
  - github.com/goplus/gox

Go+下一步的重心

- 用户使用范式最大化的确定
- 在 1.0 版本中尽可能大部分语法都稳定下来

Go+下一步的重心

- 所以我们决定：先单引擎迭代，先做好静态编译执行
- 等 1.0 发布后再发展脚本引擎

Go+团队成员持续寻找中

- Go+ 统一了程序员与数据科学家的语言，让双方自然对话
- Go+ 会是数据科学领域的下一场巨大变革
- 我很兴奋能够参与其中
- 你呢?

THANKS

https://github.com/goplus

jobs@qiniu.com

@xushiwei