微服务和Go语言的应用

AstaXie

关于我

- 《Go Web 编程》作者
- 开源框架 beego 作者
- GopherChina大会发起人兼组委会主席
- Go Team中国区的社区负责人
- Go基金会发起人和董事会成员
- EGO上海分会会长
- 积梦智能CEO

OUTLINE

- 什么是Go?
- 什么是微服务?
- Go适合微服务
- 总结

什么是Go

```
Date: Sun, 23 Sep 2007 23:33:41 -0700
From: "Robert Griesemer" <gri@google.com>
To: "Rob 'Commander' Pike" <r@google.com>, ken@google.com
Subject: prog lang discussion
...
*** General:
Starting point: C, fix some obvious flaws, remove crud, add a few missing features
  - no includes, instead: import
  - no macros (do we need something instead?)
  - ideally only one file instead of a .h and .c file, module interface
    should be extracted automatically
  - statements: like in C, though should fix 'switch' statement
  - expressions: like in C, though with caveats (do we need ',' expressions?)
  - essentially strongly typed, but probably w/ support for runtime types
  - want arrays with bounds checking on always (except perhaps in 'unsafe mode'-see section on GC)
  - mechanism to hook up GC (I think that most code can live w/ GC, but for a true systems
    programming language there should be mode w/ full control over memory allocation)
  - support for interfaces (differentiate between concrete, or implementation types, and abstract,
    or interface types)
  - support for nested and anonymous functions/closures (don't pay if not used)
  - a simple compiler should be able to generate decent code
  - the various language mechanisms should result in predictable code
...
```

Go 的三大作者

Go的特性

- 静态类型
- 编译—-fast!
- 二进制
- GC
- C Like
- 丰富的标准库
- 内置并发

Go集大成者

- A breath of fresh air from "kitchen sink" languages (cf. Scala)
- Simple, orthogonal features that aren't surprising (cf. Node)
- Efficient by default (cf. Python, Ruby)
- Predictable runtime behavior, fast lifecycles (cf. all JVM languages)
- Familiar heritage, syntax, and paradigm (cf. Haskell, Elixir)

设计初衷

Go at Google: Language Design in the Service of Software Engineering

Rob Pike
Google, Inc.
@rob_pike
http://golang.org/s/plusrob
http://golang.org

1. Abstract

(This is a modified version of the keyno
Rob Pike at the SPLASH 2012 confere
Arizona, on October 25, 2012.)

The Go programming language was co
2007 as an answer to some of the prob
seeing developing software infrastructu
computing landscape today is almost u
environment in which the languages be
C++, Java, and Python, had been creat

19. Summary

Software engineering guided the design of Go. More than most general-purpose programming languages, Go was designed to address a set of software engineering issues that we had been exposed to in the construction of large server software. Offhand, that might make Go sound rather dull and industrial, but in fact the focus on clarity, simplicity and composability throughout the design instead resulted in a productive, fun language that many programmers find expressive and powerful.

The properties that led to that include:

- Clear dependencies
- Clear syntax
- Clear semantics
- Composition over inheritance
- Simplicity provided by the programming model (garbage collection, concurrency)
- Easy tooling (the `go` tool, `gofmt`, `godoc`, `gofix`)

https://talks.golang.org/2012/splash.article

Go 发展趋势


TIOBE Index for Go
Source: www.tiobe.com
Ratings (%)
2.5
2
1.5
1
0.5
0
2010
2011
2012
2013
2014
2015
2016
2017

Go 发展趋势

TECOSYSTEMS

The RedMonk Programming Language Rankings: January 2017

By Stephen O'Grady | March 17, 2017

- **Go**: While Go also benefitted from the new ranking model, jumping four spots in the GitHub portion of our ranking system, that wasn't enough to keep up with Swift which leapfrogged it. To some extent, this isn't a surprise, as Go had neither the built in draw of iOS mobile app development nor is it generally positioned as a front and back end language as Swift increasingly is. More to the point, while it might have held static, a ranking of 15 is impressive for an infrastructure runtime.

The eigenvector of "Why we moved from language X to language Y"

2017-03-15

Go is the future of programming (?)

Without further ado, here is the top few languages of the stationary distribution:

16.41%	Go
14.26%	C
13.21%	Java
11.51%	C++
9.45%	Python

I took the stochastic matrix sorted by the *future popularity* of the language (as predicted by the first eigenvector).

异教徒

Why Go Is Not Good

Now you may say "But why is Go not good? This is just a list of complaints; you can complain about any language!". This is true; no language is perfect. However, I hope my complaints reveal a little bit about how
- Go doesn't really do anything new.
- Go isn't well-designed from the ground up.
- Go is a regression from other modern programming languages.

对 Go 语言的综合评价

以前写过一些对 Go 语言的负面评价。现在看来，虽然那些评价大部分属实，然而却由于言辞激烈，没有点明具体问题，难以让某些人信服。在经过几个月实际使用 Go 来构造网站之后，我觉得现在是时候对它作一些更加“客观”的评价了。

定位和优点

Go 比起 C 和 C++ 确实有它的优点，这是很显然的事情。它比起 Java 也有少数优点，然而相对而言更多是不足之处。所以我对 Go 的偏好在比 Java 稍低一点的位置。

Go 语言比起 C，C++ 的强项，当然是它的简单性和垃圾回收。由于 C 和 C++ 的设计有很多历史遗留问题，所以 Go 看起来确实更加优雅和简单。比起那些大量使用设计模式的 Java 代码，Go 语言的代码也似乎更简单一些。另外，Go 的垃圾回收机制比起 C 和 C++ 的全手动内存管理来说，大大降低了程序员的头脑负担。

但是请注意，这里的所谓“优点”都是相对于 C 之类的语言而言的。如果比起另外的一些语言，Go 的这种优点也许就很微不足道，甚至是历史的倒退了。

ksimka / go-is-not-good

Code | Issues 5 | Pull requests 2 | Projects 0 | Wiki

A curated list of articles complaining that go (golang) isn't good enough

generator.go — Use sort.Slice from go1.8 (#28)

Go 发展趋势

Orchestration

Monitoring

Distributed Tracing API

Logging

Service Mesh

Remote Procedure Call

Service Discovery

Container Runtime

Container Runtime

Networking API

Service Mesh

Distributed Tracing

Security

Software Update Spec

微服务

什么是微服务

- 1000 行代码或者更少代码?
- 使用Docker部署?
- 使用Node写的?

基于size的定义

“A single programmer can design, implement, deploy, and maintain”

– Fred George

“Software that fits in your head.”

– Dan North

基于data的定义

"A microservice implements a single bounded context."

- Martin Fowler, Sam Newman

"A single logical database per service."

– Chris Richardson

基于operation的定义

"Built & deployed independently. Stateless; state as backing services."

- 12Factor.net

"Addressable through a service discovery system."

– Chris Richardson

微服务的价值和问题

Microservices

solve

organizational problems

✳

Microservices

cause

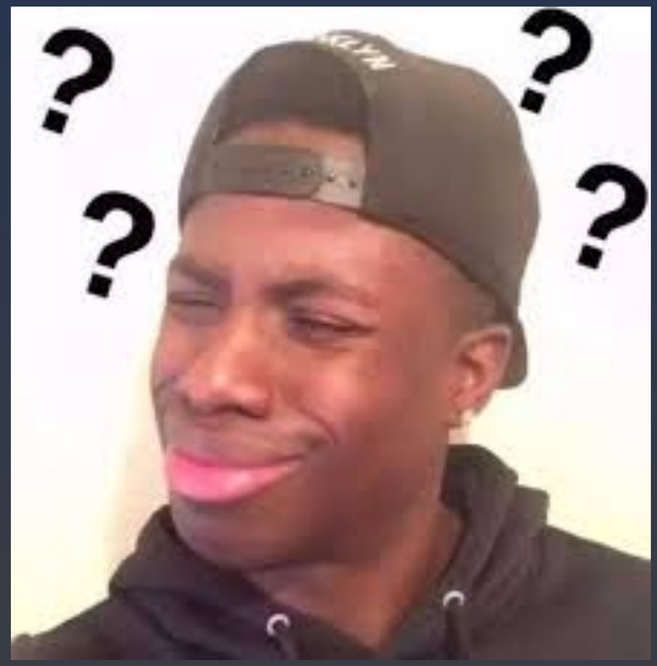
technical problems

微服务解决的问题

- 大团队无法高效的协作
- 团队之间相互依赖无法推进工作
- 沟通变得非常困难
- 产品迭代速度慢

微服务引起的问题

- 团队之间API定义需要非常稳定
- 分布式事务处理变得极其困难
- 微服务的集成测试
- 服务发现
- 监控和日志
- 分布式tracing
- 微服务之间的安全性



Go适合微服务

微服务关心什么?

- 高可用
- 易扩展
- 可实施
- 易扩容
- 可追踪

高可用

- 负载均衡
- 服务注册与发现
- 应用无状态

负债均衡

- seesaw
- caddy

服务注册与发现

- etcd
- consul
- serf

服务网关

- tyk
- fabio
- valcand
- traefik

易扩展

- 无状态
- 异步消息
- 标准接口

异步消息队列

- NSQ
- NATS

标准接口

- gRPC
- HTTP

可实施

- 团队基础
- 入门容易
- 部署方便

入门容易

- 类C语法
- GC内置
- 工程工具

部署容易

- docker化
- 二进制

微服务框架多

- beego
- go-kit
- go-micro
- gin
- Iris

易扩展

- 分布式调度
- 扩容机器

分布式调度

- k8s
- swarm

可追踪

- 服务器监控/告警
- APM

监控

- Prometheus
- open-falcon
- grafana

日志分析

- Beats
- Heka

APM

- opentracing
- cloudinsight
- appdash

总结

微服务不适合你

- 引起很多问题
- 如果你们是小团队，那么坚决不要用
- 如果你选择了，就准备好擦屁股

Go适合微服务

- 容易学习
- 天生并发，为服务而生
- http默认性能就非常高效
- 丰富的标准库
- 简单方便的部署
- 快速发展的生态圈

谢谢