The past, present, and future of Go 2

QCon Shanghai 2018

GO

GO

GO

GO

GO

GO

Today

Where Go came from?

How Go has evolved since it was launched?

What’s happening in Go 2?

The past

2007–2009

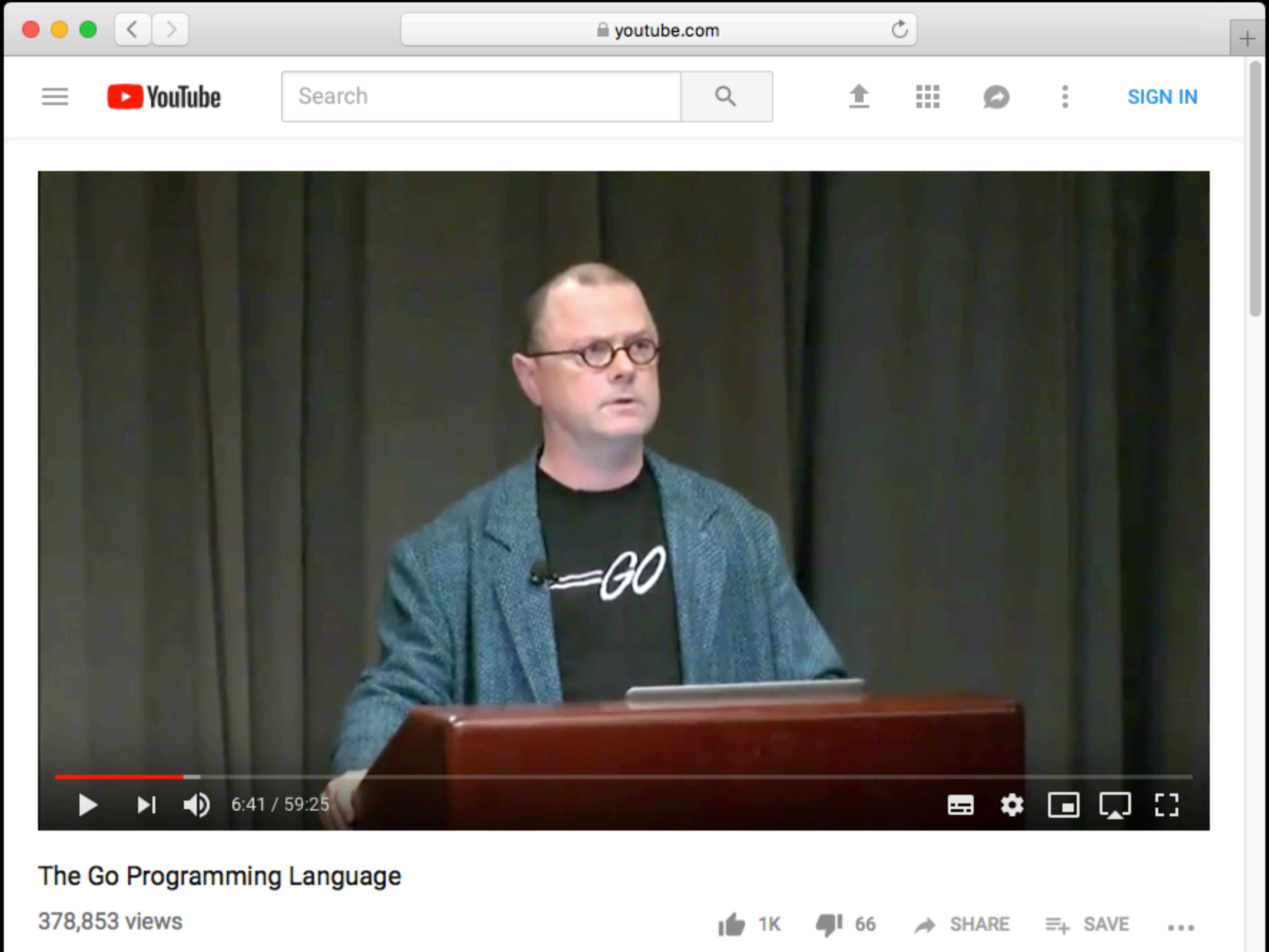
Why Go?

Why is their a language called Go?

We have C++, Java, C#, Python, Ruby, PHP, and Javascript

Why did Rob Pike, Ken Thompson, and Robert Griesemer decide to write a new language?

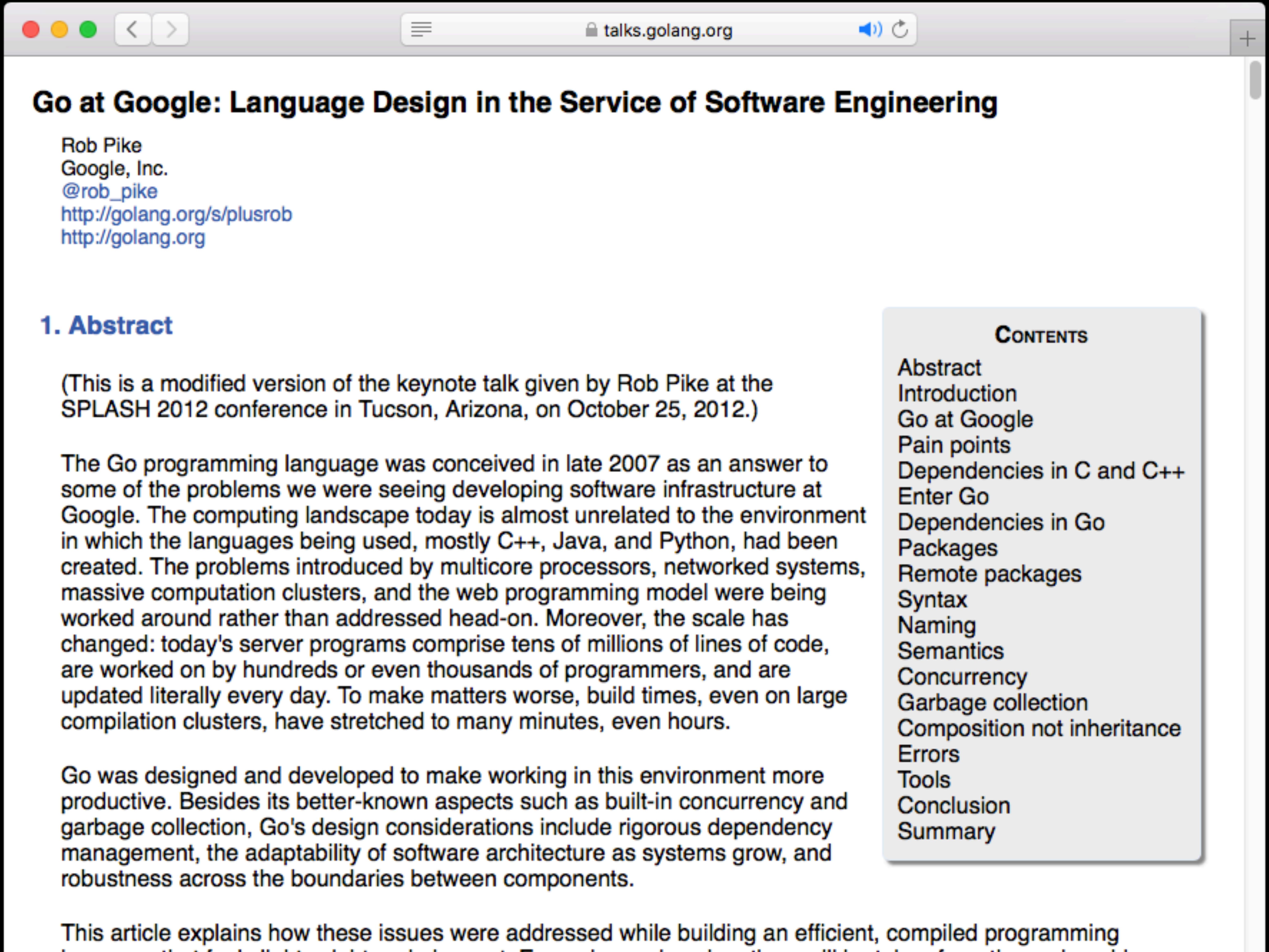
The Go programming language, 2009



The Go Programming Language

378,853 views

Language Design in the Service of Software Engineering



talks.golang.org

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 keynote talk given by Rob Pike at the SPLASH 2012 conference in Tucson, Arizona, on October 25, 2012.)

The Go programming language was conceived in late 2007 as an answer to some of the problems we were seeing developing software infrastructure at Google. The computing landscape today is almost unrelated to the environment in which the languages being used, mostly C++, Java, and Python, had been created. The problems introduced by multicore processors, networked systems, massive computation clusters, and the web programming model were being worked around rather than addressed head-on. Moreover, the scale has changed: today's server programs comprise tens of millions of lines of code, are worked on by hundreds or even thousands of programmers, and are updated literally every day. To make matters worse, build times, even on large compilation clusters, have stretched to many minutes, even hours.

Go was designed and developed to make working in this environment more productive. Besides its better-known aspects such as built-in concurrency and garbage collection, Go's design considerations include rigorous dependency management, the adaptability of software architecture as systems grow, and robustness across the boundaries between components.

This article explains how these issues were addressed while building an efficient, compiled programming

Contents

A language for developer productivity

Together these presentations provide a rationale for a new language, originally designed for Google's software development needs.

As it turns out—because we all need software—Go has become a pretty good fit for anyone writing large scale server software.

Because, at its core, the goal of Go is to improve developer productivity.

The difference between programming and software engineering

"Software engineering is what happens to programming when you add time and other programmers."
—Russ Cox

The difference between programming and software engineering

The difference between software programming and software engineering is not the size of the program, but how long the program will live for.

Sitting down and writing a script or a throw away program for a single computation is software programming.

That's totally fine, sometimes that is all the problem calls for.

The difference between programming and software engineering

On the other hand, Software engineering is a more deliberate, considered, act.

It requires a broader view of the software development lifecycle than just focusing on lines of code, syntax, and algorythms.

When Go launched it was with the explicit intent to improve the life of the software engineer.

The PRESENT

2009–2018

The platforms

When Go was open sourced on the 11th of November 2009 it supported Linux, Mac OS X, on 386, amd64, and if you were running Linux, ARMv5 and v6.

By the time Go 1.0 was launched in March of 2012 we added support for Windows, FreeBSD, OpenBSD

The platforms

In Go 1.3 we added support for FreeBSD, DragonfyBSD, OpenBSD, and NetBSD, Plan 9 on 386 and Native Client (NaCl), and Solaris on amd64

Go 1.4 added support for cross compling to Android, NaCl on ARM, and Plan 9 amd64

Go 1.5 added support for arm64 on Linux and OS X.

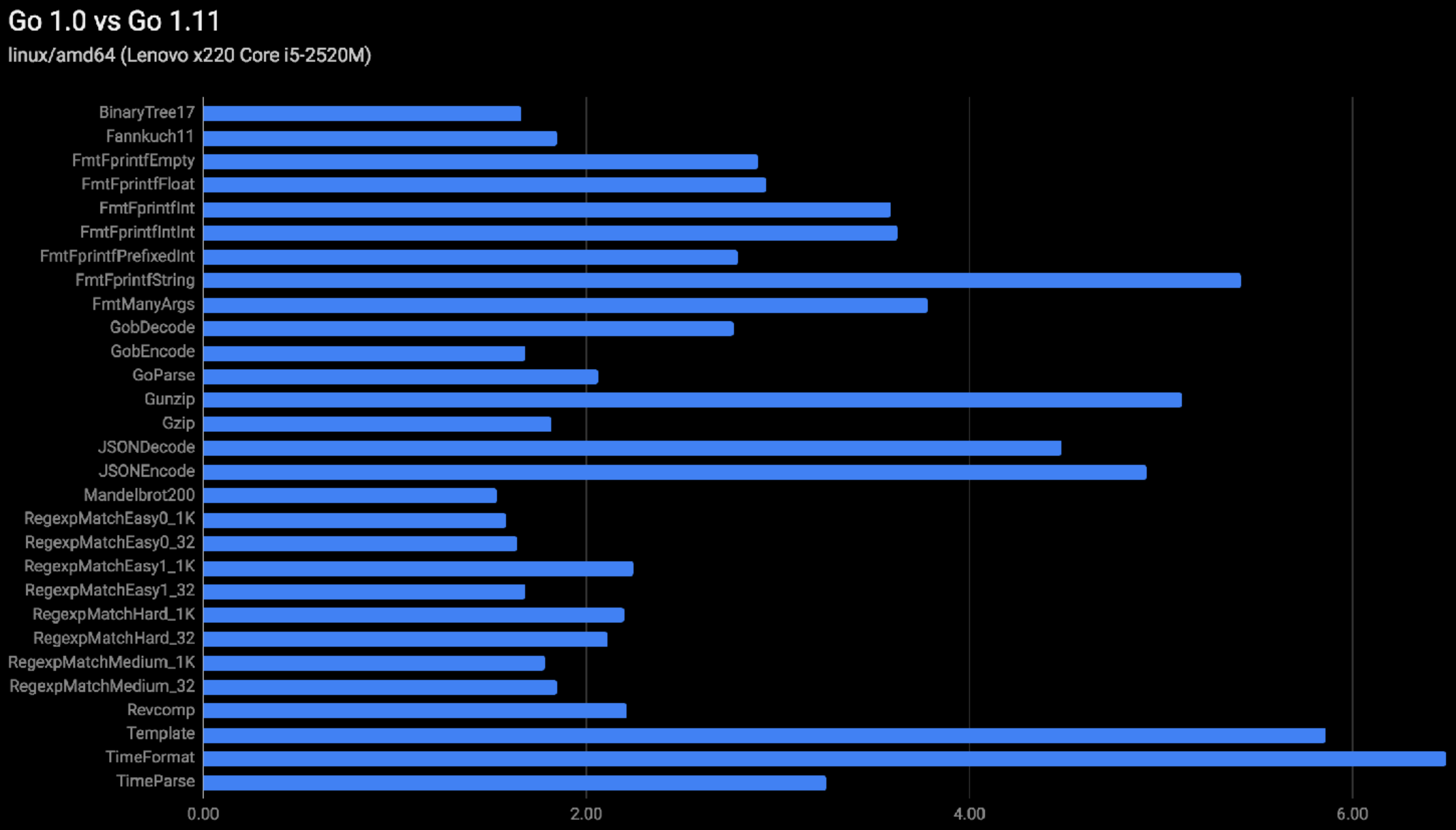
Go 1.6 added support for 64bit MIPS on Linux, and Android on 386

Go 1.7 added support for IBM System/z and 64 bit PowerPC

Go 1.8 added support for 32 bit MIPS

Go 1.11 added support for web assembly and plans are in the works for a RISC-V port

The performance



The companies

Atlassian, Heptio, Digital Ocean, Netflix, Pulimi, Twitch, Google, Microsoft, Reddit, Cloudflare, MongoDB, InfluxDB, Datadog, bookings.com, Rakuten, GitHub, GitLab, Freelancer, Fastly, Netlify, Pivotal, Couchbase, Lyft, Monzo, Uber, Source{d}, srcgraph, …

The companies

QINIU 七牛云存储

Baidu 百度

头条 今日头条

PingCAP

阿里云

Grab

bilibili

腾讯云

网易云

DiDi

英语流利说

Tantan

罗辑思维

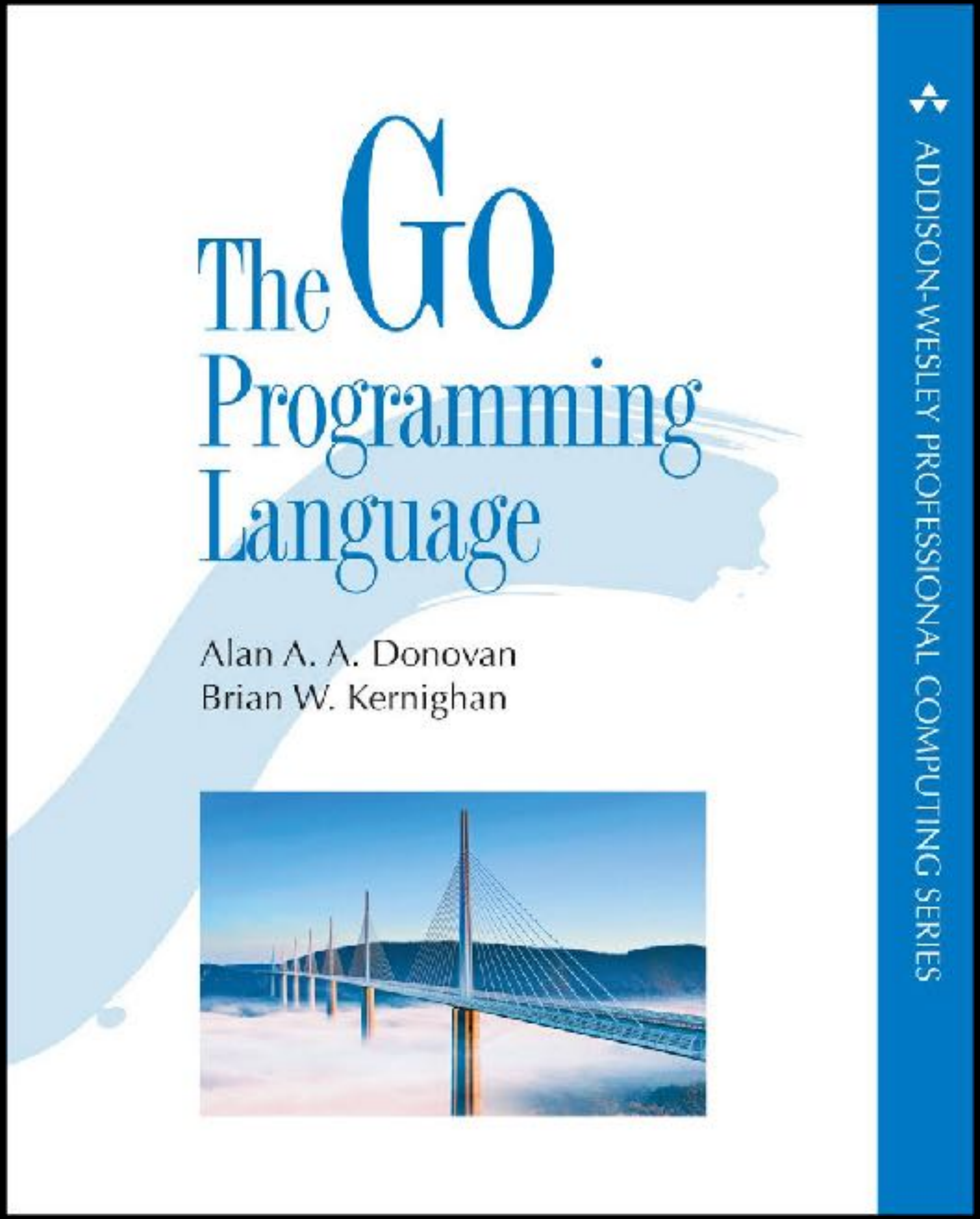
The projects

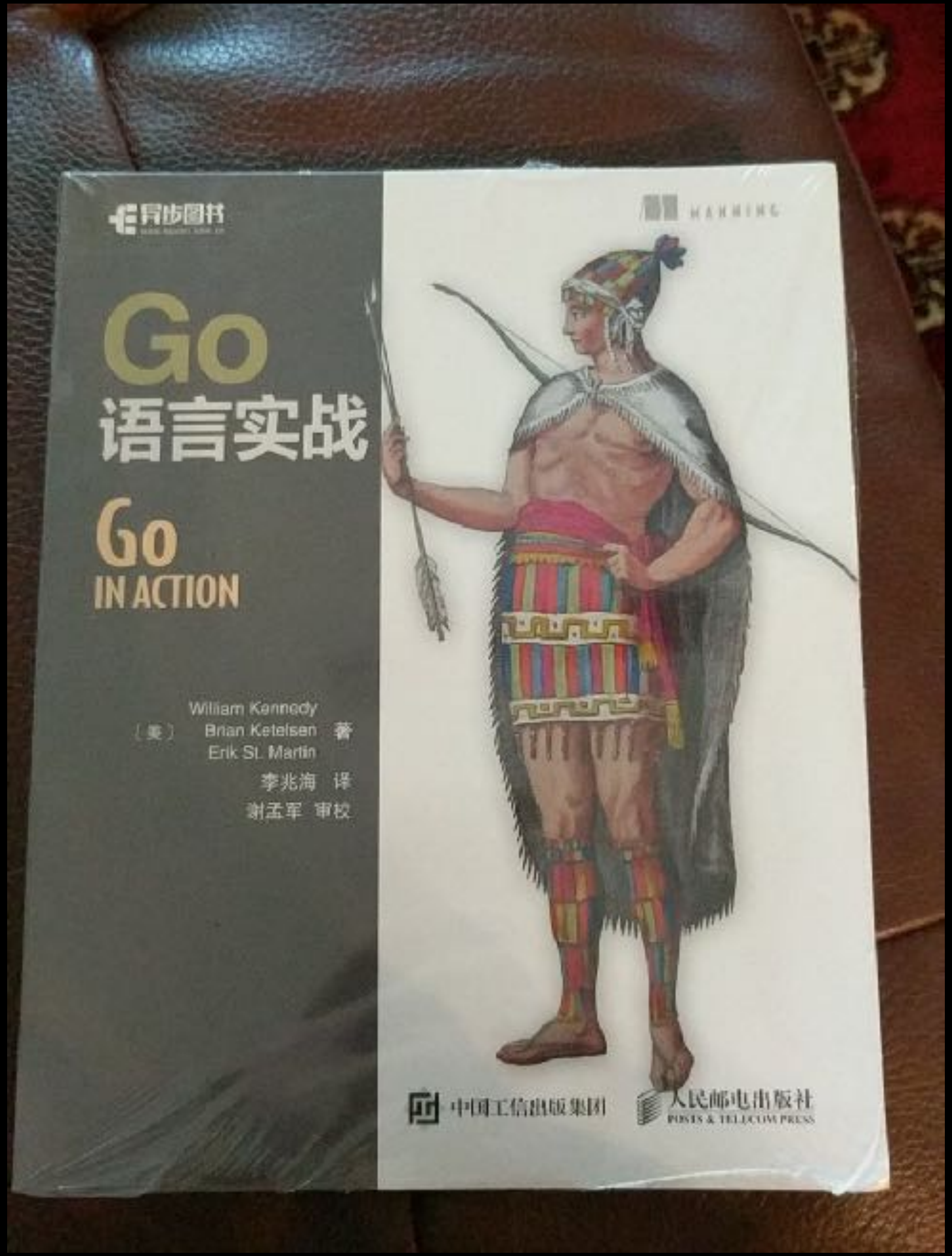
Vitess, Docker, Traefic, Kubernetes, Istio, GitLab, Vault, Consol, Terraform, CockroachDB, CloudFoundry, Gobot, Beego, …

The books

HTTPS://GITHUB.COM/GOLANG/GO/WIKI/BOOKS

The books


The Go Programming Language
Alan A. A. Donovan
Brian W. Kernighan
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES



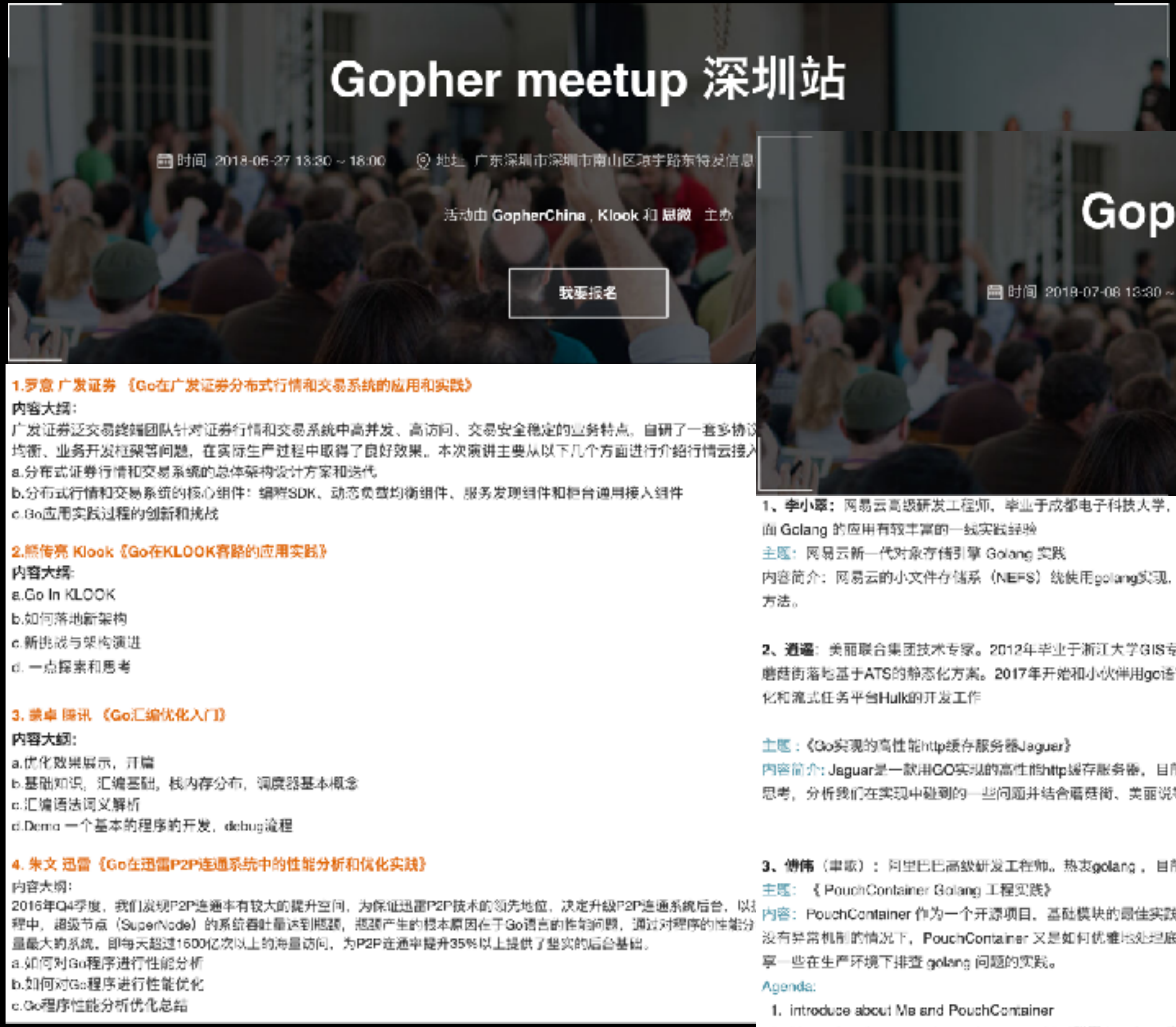
HTTPS://GITHUB.COM/GOLANG/GO/WIKI/BOOKS

The community

Greenland
Iceland
Northwestern Passages
Canada
Hudson Bay
Labrador Sea
United States
Gulf of Mexico
Mexico
Cuba
Puerto Rico
Guatemala
Nicaragua
Caribbean Sea
Venezuela
Colombia
Guyana
Suriname
Ecuador
Peru
Brazil
Bolivia
Paraguay
Chile
Argentina
Uruguay
North Atlantic Ocean
South Atlantic Ocean
Sweden
Norway
Finland
Denmark
United Kingdom
Ireland
Germany
Poland
Ukraine
France
Spain
Italy
Romania
Turkey
Syria
Iraq
Iran
Russia
Kazakhstan
Mongolia
Uzbekistan
Kyrgyzstan
Turkmenistan
Afghanistan
Pakistan
Nepal
India
China
South Korea
Sea of Japan
Sea of Okhotsk
East China Sea
Myanmar (Burma)
Thailand
Vietnam
Philippines
Philippine Sea
South China Sea
Gulf of Thailand
Malaysia
Indonesia
Banda Sea
Arafura Sea
Papua New Guinea
Morocco
Tunisia
Algeria
Libya
Egypt
Western Sahara
Mauritania
Mali
Niger
Chad
Sudan
Burkina Faso
Guinea
Ghana
Nigeria
South Sudan
Ethiopia
Somalia
Saudi Arabia
Oman
Yemen
Gulf of Aden
Arabian Sea
Bay of Bengal
Laccadive Sea
Gabon
DRC
Tanzania
Angola
Zambia
Mozambique
Madagascar
Namibia
Zimbabwe
Botswana
South Africa
Indian Ocean
Australia
Great Australian Bight
Coral Sea
Tasman Sea
19
37
17
31
21
12
11
9
2
3
3
3
21
10
5
4
43
46
10
12
5
7
5
2
2
3
2
10
20
2
4
3
2
5
5
4

Go meetups in China

Go meetups in China



Gopher meetup 深圳站

时间 2018-06-27 13:30 ~ 18:00　地址 广东深圳市深圳市南山区琼宇路东特发信息

活动由 GopherChina , Klook 和 慧微 主办

我要报名

1.罗意 广发证券 《Go在广发证券分布式行情和交易系统的应用和实践》

内容大纲:

广发证券泛交易终端团队针对证券行情和交易系统中高并发、高访问、交易安全稳定的业务特点，自研了一套多协议
均衡、业务开发框架等问题，在实际生产过程中取得了良好效果。本次演讲主要从以下几个方面进行介绍行情云接入

a.分布式证券行情和交易系统的总体架构设计方案和迭代

b.分布式行情和交易系统的核心组件：编解SDK、动态负载均衡组件、服务发现组件和柜台通用接入组件

c.Go应用实践过程的创新和挑战

2.熊传亮 Klook《Go在KLOOK落地的应用实践》

内容大纲:

a.Go In KLOOK

b.如何落地新架构

c.新挑战与架构演进

d. 一点探索和思考

3. 崇卓 腾讯 《Go汇编优化入门》

内容大纲:

a.优化效果展示，开篇

b.基础知识，汇编基础，栈内存分布，调度器基本概念

c.汇编语法词义解析

d.Demo 一个基本的程序的开发，debug流程

4. 朱文 迅雷《Go在迅雷P2P连通系统中的性能分析和优化实践》

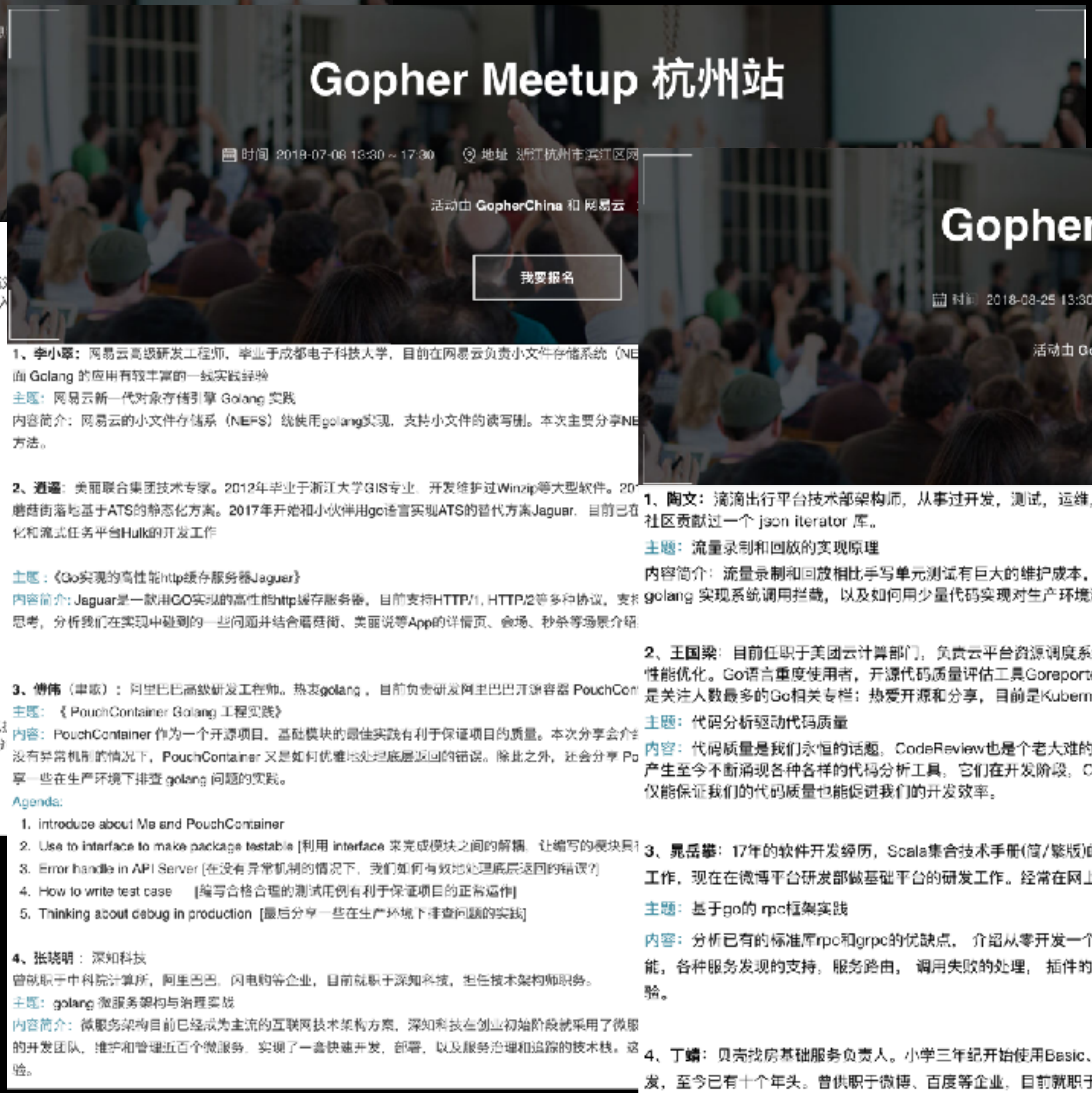
内容大纲:

2016年Q4季度，我们发现P2P连通率有较大的提升空间，为保证迅雷P2P技术的领先地位，决定升级P2P连通系统后台，以
程中，超级节点（SuperNode）的系统吞吐量达到瓶颈，瓶颈产生的根本原因在于Go语言的性能问题，通过对程序的性能分
量最大的系统，即每天超过1600亿次以上的海量访问，为P2P连通率提升35%以上提供了坚实的后台基础。

a.如何对Go程序进行性能分析

b.如何对Go程序进行性能优化

c.Go程序性能分析优化总结



Gopher Meetup 杭州站

时间 2018-07-08 13:30 ~ 17:30　地址 浙江杭州市滨江区网

活动由 GopherChina 和 网易云

我要报名

1、李小翠: 网易云高级研发工程师，毕业于成都电子科技大学，目前在网易云负责小文件存储系统（NE
面 Golang 的应用有较丰富的一线实践经验

主题: 网易云新一代对象存储引擎 Golang 实践

内容简介: 网易云的小文件存储系（NEFS）统使用golang实现，支持小文件的读写删。本次主要分享NE
方法。

2、逍遥: 美丽联合集团技术专家。2012年毕业于浙江大学GIS专业，开发维护过Winzip等大型软件。201
蘑菇街落地基于ATS的静态化方案。2017年开始和小伙伴用go语言实现ATS的替代方案Jaguar，目前已在
化和流式任务平台Hulk的开发工作

主题 :《Go实现的高性能http缓存服务器Jaguar》

内容简介: Jaguar是一款用GO实现的高性能http缓存服务器。目前支持HTTP/1，HTTP/2等多种协议。支持
思考，分析我们在实现中碰到的一些问题并结合蘑菇街、美丽说等App的详情页、会场、秒杀等场景介绍

3、傅伟（串欢）：阿里巴巴高级研发工程师。热爱golang，目前负责研发阿里巴巴开源容器 PouchCon

主题: 《PouchContainer Golang 工程实践》

内容: PouchContainer 作为一个开源项目，基础模块的最佳实践有利于保证项目的质量。本次分享会介
没有异常机制的情况下，PouchContainer 又是如何优雅地处理底层返回的错误。除此之外，还会分享 Po
享一些在生产环境下排查 golang 问题的实践。

Agenda:

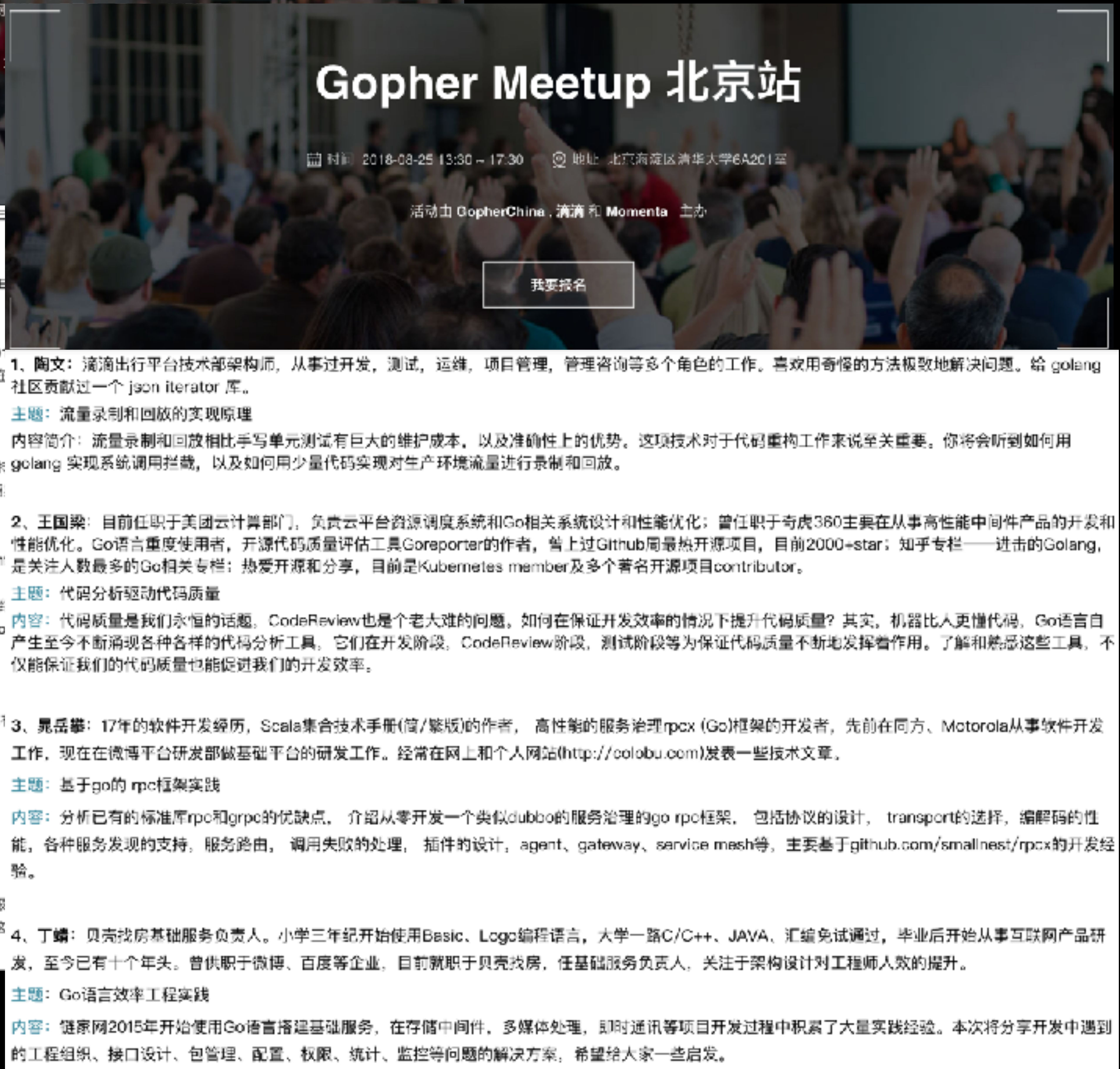
1. introduce about Me and PouchContainer
2. Use to interface to make package testable [利用 interface 来完成模块之间的解耦，让编写的模块具
3. Error handle in API Server [在没有异常机制的情况下，我们如何有效地处理底层返回的错误?]
4. How to write test case　[编写合格合理的测试用例有利于保证项目的正常运作]
5. Thinking about debug in production [最后分享一些在生产环境下排查问题的实践]

4、张晓明：深知科技

曾就职于中科院计算所，阿里巴巴、闪电购等企业，目前就职于深知科技，担任技术架构师职务。

主题: golang 微服务架构与治理实战

内容简介: 微服务架构目前已经成为主流的互联网技术架构方案，深知科技在创业初始阶段就采用了微服
的开发团队，维护和管理近百个微服务，实现了一套快速开发，部署，以及服务治理和追踪的技术栈。这
验。



Gopher Meetup 北京站

时间 2018-08-25 13:30 ~ 17:30　地址 北京海淀区清华大学6A201室

活动由 GopherChina，滴滴 和 Momenta 主办

我要报名

1、陶文: 滴滴出行平台技术部架构师，从事过开发，测试，运维，项目管理，管理咨询等多个角色的工作。喜欢用奇怪的方法极致地解决问题。给 golang 社区贡献过一个 json iterator 库。

主题: 流量录制和回放的实现原理

内容简介: 流量录制和回放相比手写单元测试有巨大的维护成本，以及准确性上的优势。这项技术对于代码重构工作来说至关重要。你将会听到如何用 golang 实现系统调用拦截，以及如何用少量代码实现对生产环境流量进行录制和回放。

2、王国梁: 目前任职于美团云计算部门，负责云平台资源调度系统和Go相关系统设计和性能优化；曾任职于奇虎360主要在从事高性能中间件产品的开发和性能优化。Go语言重度使用者，开源代码质量评估工具Goreporter的作者，曾上过Github周最热开源项目，目前2000+star；知乎专栏——进击的Golang，是关注人数最多的Go相关专栏；热爱开源和分享，目前是Kubernetes member及多个著名开源项目contributor。

主题: 代码分析驱动代码质量

内容: 代码质量是我们永恒的话题，CodeReview也是个老大难的问题。如何在保证开发效率的情况下提升代码质量? 其实，机器比人更懂代码，Go语言自产生至今不断涌现各种各样的代码分析工具，它们在开发阶段，CodeReview阶段，测试阶段等为保证代码质量不断地发挥着作用。了解和熟悉这些工具，不仅能保证我们的代码质量也能促进我们的开发效率。

3、晁岳攀: 17年的软件开发经历，Scala集合技术手册(简/繁版)的作者，高性能的服务治理rpcx (Go)框架的开发者，先前在同方、Motorola从事软件开发工作，现在在微博平台研发部做基础平台的研发工作。经常在网上和个人网站(http://colobu.com)发表一些技术文章。

主题: 基于go的 rpc框架实践

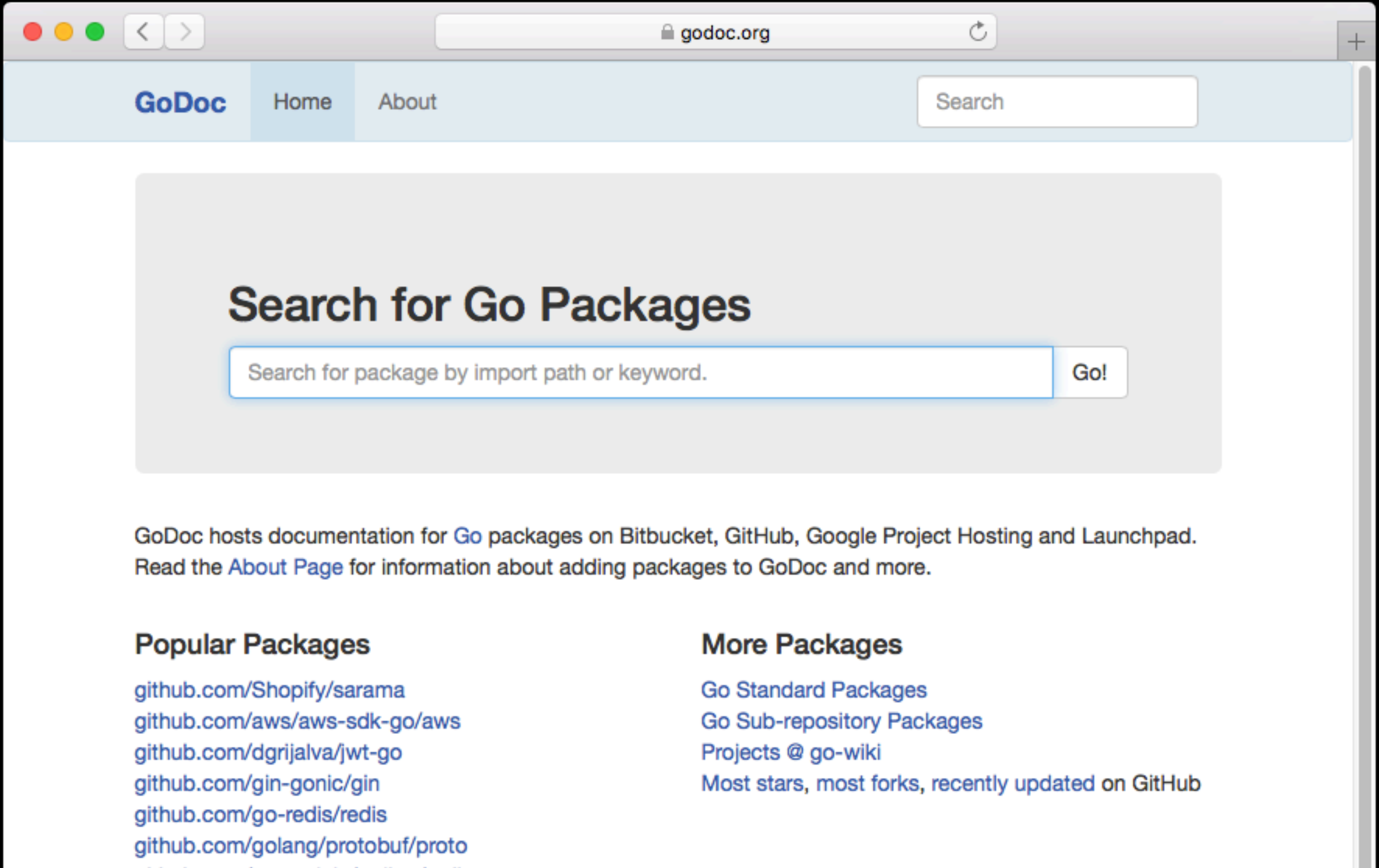
内容: 分析已有的标准库rpc和grpc的优缺点，介绍从零开发一个类似dubbo的服务治理的go rpc框架，包括协议的设计，transport的选择，编解码的性能，各种服务发现的支持，服务路由，调用失败的处理，插件的设计，agent、gateway、service mesh等，主要基于github.com/smallnest/rpcx的开发经验。

4、丁靖: 贝壳找房基础服务负责人。小学三年纪开始使用Basic、Logo编程语言，大学一路C/C++、JAVA、汇编免试通过，毕业后开始从事互联网产品研发，至今已有十个年头。曾供职于微博、百度等企业，目前就职于贝壳找房，任基础服务负责人，关注于架构设计对工程师人效的提升。

主题: Go语言效率工程实践

内容: 链家网2015年开始使用Go语言搭建基础服务，在存储中间件，多媒体处理，即时通讯等项目开发过程中积累了大量实践经验。本次将分享开发中遇到的工程组织、接口设计、包管理、配置、权限、统计、监控等问题的解决方案，希望给大家一些启发。

GODOC.ORG, 800,000+ PACKAGES



Search for Go Packages

Search for package by import path or keyword.

GoDoc hosts documentation for Go packages on Bitbucket, GitHub, Google Project Hosting and Launchpad. Read the About Page for information about adding packages to GoDoc and more.

Popular Packages

github.com/Shopify/sarama
github.com/aws/aws-sdk-go/aws
github.com/dgrijalva/jwt-go
github.com/gin-gonic/gin
github.com/go-redis/redis
github.com/golang/protobuf/proto

More Packages

Go Standard Packages
Go Sub-repository Packages
Projects @ go-wiki
Most stars, most forks, recently updated on GitHub

The conferences

GopherCon Denver

dot Go

dotGo Paris

GopherCon Singapore

GopherCon EU

GopherCon Brazil

GopherCon UK

Go conferences in China

GopherChina 2015

第一届 GopherChina 大会

GopherChina 2016

第二届 GopherChina 大会

GopherChina 2017

第三届 GopherChina 大会

GopherChina 2018

第四届 GopherChina 大会

Next year, GopherChina Beijing

THE GOPHER

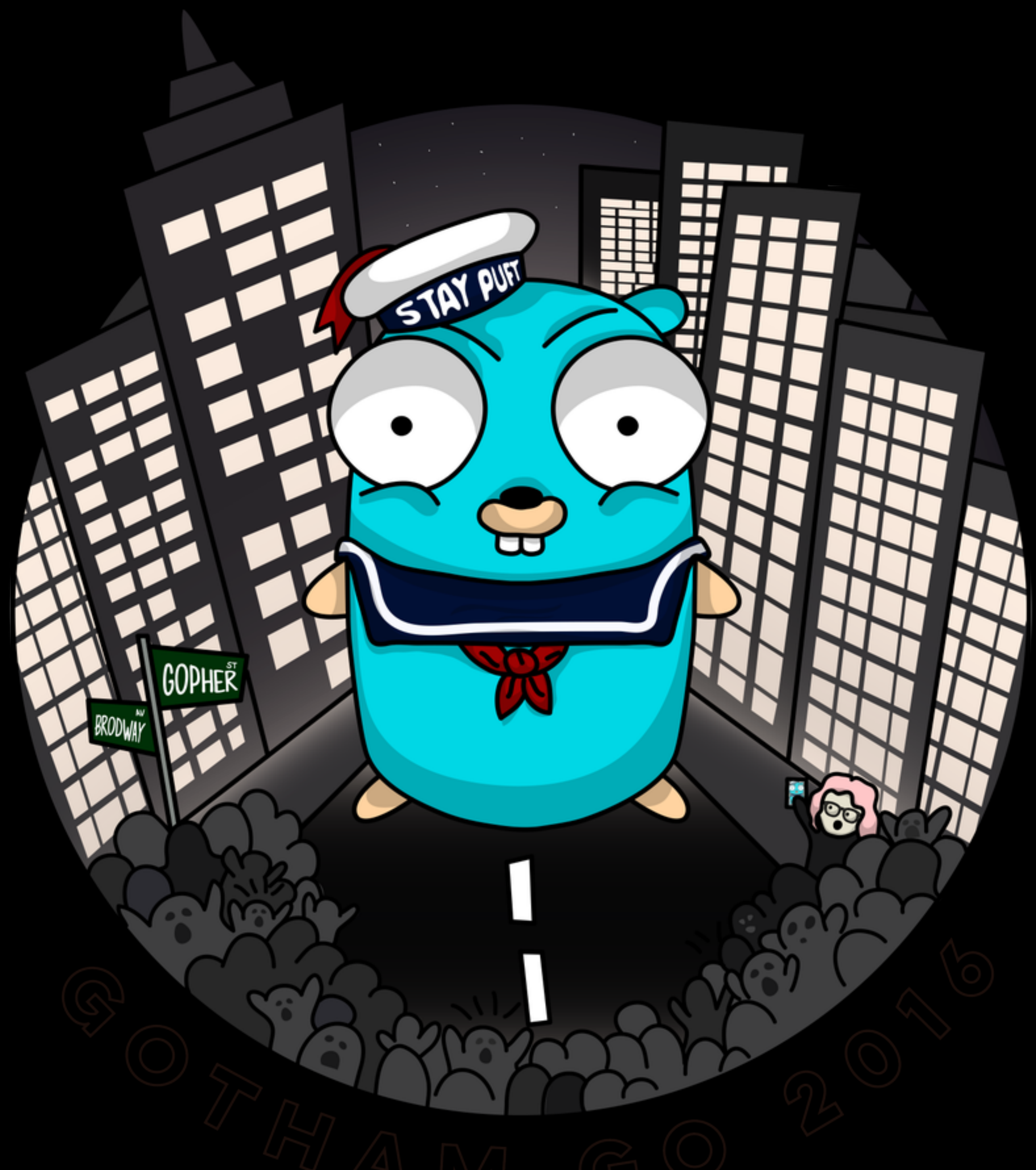
The Go gopher was designed by Renee French.

The gopher

The Go gopher was designed by Renee French.

The design is licensed under the Creative Commons 3.0 Attributions license.

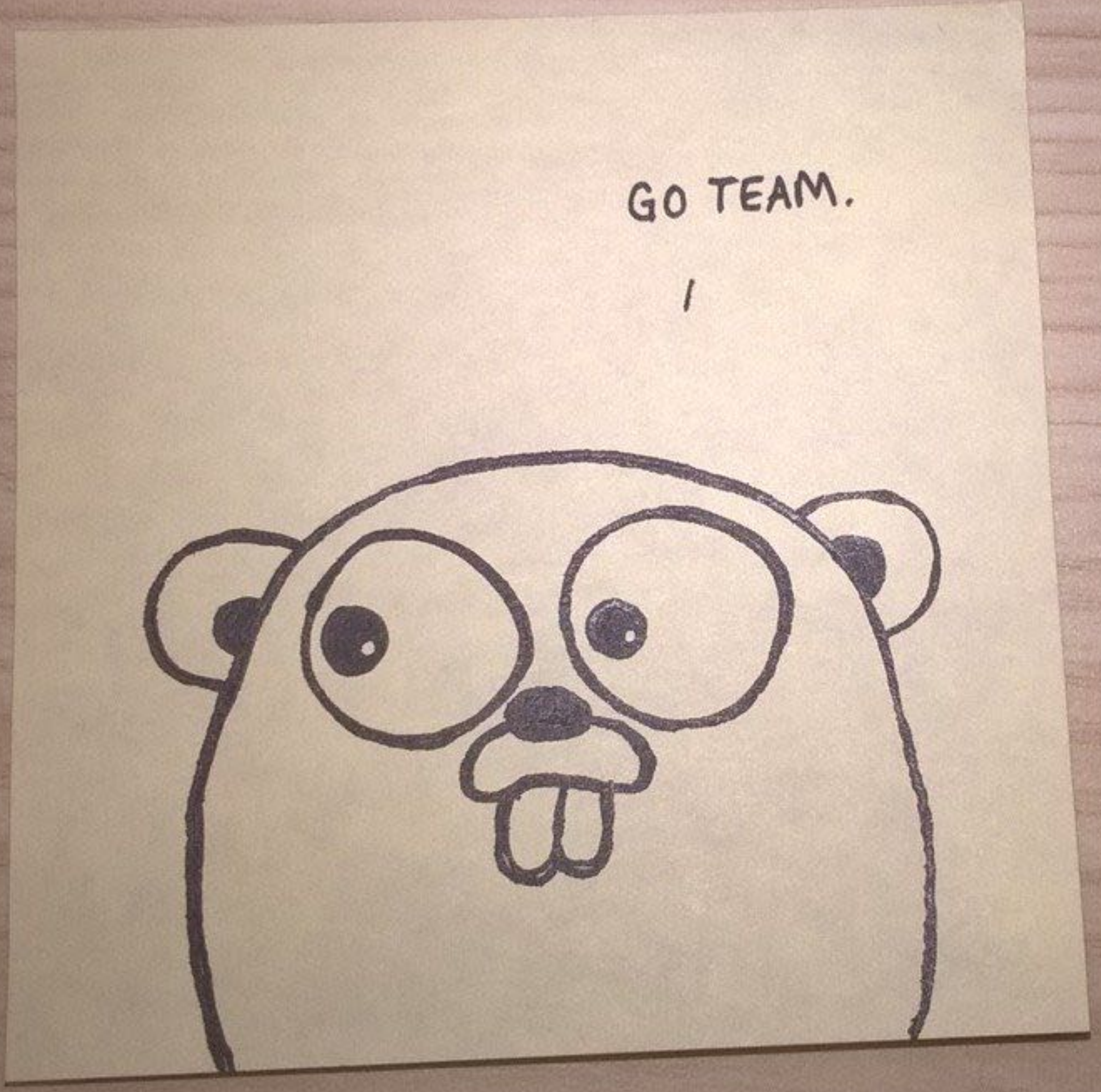
Egon Elbre

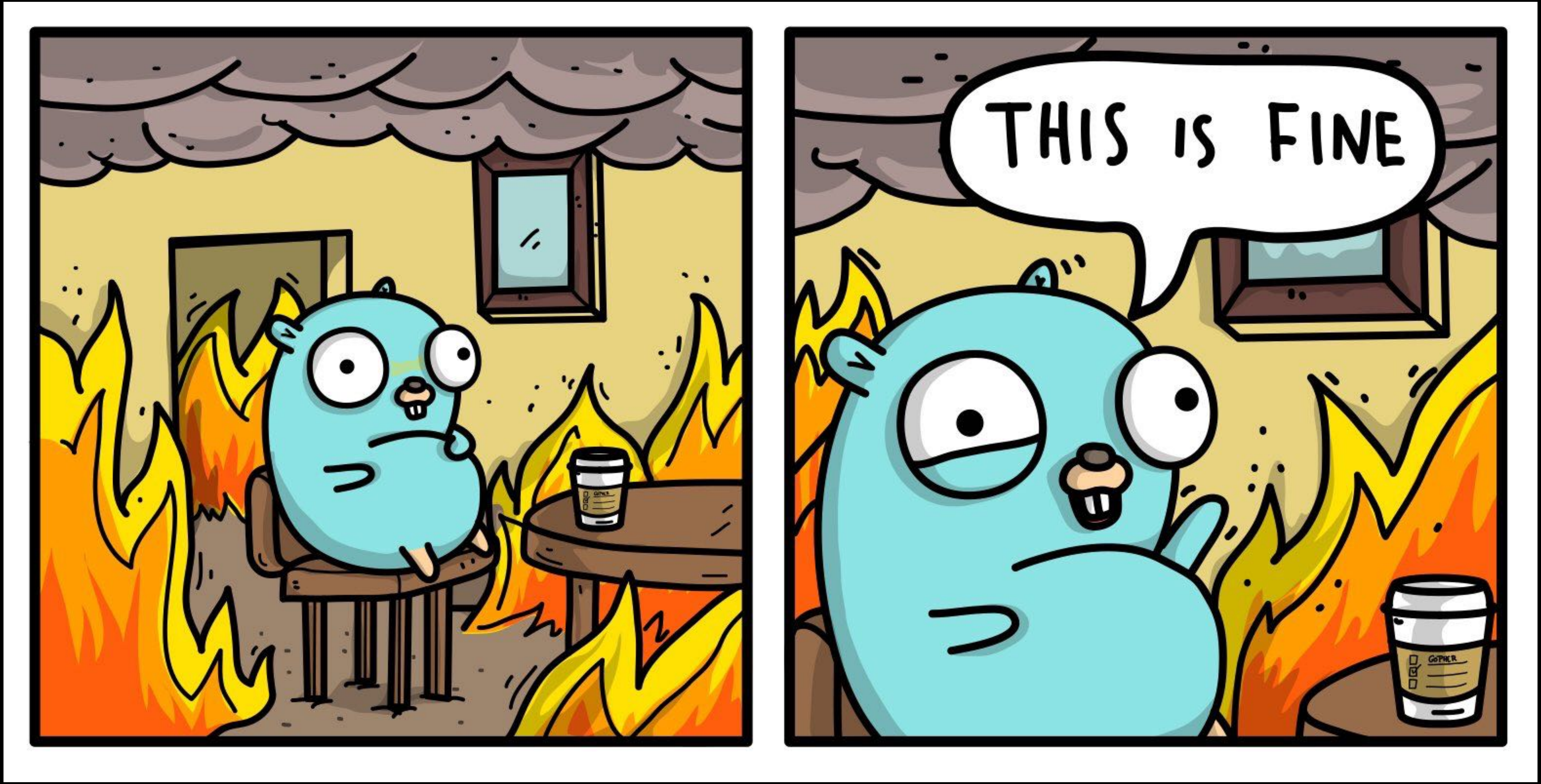

STAY PUFT
GOPHER ST
BRODWAY AV
GOTHAM GO 2016

Takuya Ueda

//LINUX.PICTURES
IDGAF LICENSE

GITHUB.COM/GENGO/GOSHIP


GO TEAM.



Ashley Macnamara

The growth of the language

"My best estimate is now between 0.8 and 1.6 million. It seems to me likely that we've crossed a million Go developers."

—Russ Cox, July 2018

The future

2018–

The beginning of Go 2

Gophercon 2017, Russ Cox announced it was time to start talking about Go 2

https://blog.golang.org/toward-go2

Go 2 would not be an opportunity to redesign the language from scratch.

Instead, Go 2 would be an evolution of Go 1, designed to address pain points Go developers worldwide have felt for a decade.

The beginning of Go 2

"Our goal for Go 2 is to **fix** the most significant ways Go **fails to scale.**"
—Russ Cox, GopherCon 2017

How shall we discover where Go failed to scale?

In his presentation at Gophercon 2017 Russ discribed the methodology for how the large issues which caused Go to fail at scale will be identified.

Specifically Russ called on the users of Go to write experience reports; gists, blog posts, issues, that demonstrated clearly the issues that developers were having using Go for increasingly larger and larger projects.

https://github.com/golang/go/wiki/ExperienceReports

Now it’s a year later, what did the Go team discover?

Towards Go 2

Towards Go 2

Top three pain points for Go developers:

Towards Go 2

Top three pain points for Go developers:

- Dependency management – modules

Towards Go 2

Top three pain points for Go developers:

- Dependency management – modules
- Error handling – check, handle, and error values

Towards Go 2

Top three pain points for Go developers:

- Dependency management – modules
- Error handling – check, handle, and error values
- Generics

Dependency management

Go modules

Go modules

The first improvement is the addition of a new concept to the Go tool, a module.

A module is a collection of packages.

Just as we have .go source files grouped into a package, so too can a collection of packages with shared prefix be considered a module.

Now, this probably looks pretty close to a concept that you aready know, a git repository. But there is an important difference, modules have an explicit understanding of versions.

Why do we need Go modules?

Prior to Go modules, `go get` only knew how to fetch whatever revision happened to be current in your repository at the time.

If you already had a copy of a package in your `$GOPATH` then `go get` would skip it, so you might end up building against a really old version.

If you used the `go get -u` flag to force it to download a fresh copy, you might find that you now had a much newer version of a package than the author.

Go get does not provide reproducible builds

Put simply, go get doesn't guarentee reproducible builds. We've had many people propose solutions, tools like:

- godep
- gopkg.in
- govendor
- gb

Promoted the idea of a `vendor/` directory, a self contained gopath that could be checked in with the code so that your program had a copy of each of the dependencies it needed.

The package management working group

In 2016 Peter Bourgon formed a working group to focus on solving the dependency management problem and called on the go team to join him in this effort.

From that working group grew a tool we know as dep.

dep drew much of its inspiration from the authors experience with their previous tools glide.

dep encouraged the use of semver, semantic versioning, using tags on your git repos, to provide tools like dep with a way of managing the contents of your vendor/ directory.

The Go team introduce modules

In early 2018 the Go team proposed their own tool, at the time given the working title vgo, now known as go modules.

Go modules are integrated into the Go tool. The notion of modules is baked in as a first class citizen.

This makes it possible for Go developers to build their code anywhere they want.

Go modules don’t require a vendor/directory, and if you use modules you no longer need to use GOPATH to hold all your Go source code.

Go modules live demo

You can use Go modules today

Go 1.11, which shipped in August, includes full support for modules.

It's opt-in at the moment, because we realise there is a large change, not just for package authors but for the ecosystem of tool authors

Experiment with converting your projects to use go.mod and please give the Go team feedback via Github.

Error handling

Check, handle, and error values

Error handling in Go

Unlike Java, Ruby, Python, or C#, Go does not use exceptions for control flow.

Instead Go's error handling takes advantage of the language's native support for multiple return values.

```
func Open(path string) (*File, error)
```

By convention, if a function returns an error value, then the caller should check if that error value to see if the operation succeeded or failed.

Error handling in Go

By convention, if a function returns an error value, then the caller should check the error value to see if the operation succeeded or failed.

```
f, err := os.Open("/etc/passwd")
if err != nil {
        return err
}
```

Go developers believe that by being forced to think about failure case before the success case, leads to more robust programs.

However, this form of error checking means it can feel repetitive to write this error checking code by hand.

```go
func CopyFile(src, dst string) error {
	r, err := os.Open(src)
	if err != nil {
		return err
	}
	defer r.Close()

	w, err := os.Create(dst)
	if err != nil {
		return err
	}
	defer w.Close()

	if _, err := io.Copy(w, r); err != nil {
		return err
	}
	if err := w.Close(); err != nil {
		return err
	}
	return nil
}
```

```
func CopyFile(src, dst string) error {
        r, err := os.Open(src)
        if err != nil {
                return err
        }
        defer r.Close()

        w, err := os.Create(dst)
        if err != nil {
                return err
        }
        defer w.Close()

        if _, err := io.Copy(w, r); err != nil {
                return err
        }
        if err := w.Close(); err != nil {
                return err
        }
        return nil
}
```

```
func CopyFile(src, dst string) error {
	r, err := os.Open(src)
	if err != nil {
		return err
	}
	defer r.Close()

	w, err := os.Create(dst)
	if err != nil {
		return err
	}
	defer w.Close()

	if _, err := io.Copy(w, r); err != nil {
		return err
	}
	if err := w.Close(); err != nil {
		return err
	}
	return nil
}
```

```go
func CopyFile(src, dst string) error {
        r, err := os.Open(src)
        if err != nil {
                return err
        }
        defer r.Close()
        …
}
```

```go
func CopyFile(src, dst string) error {
        r, err := os.Open(src)
        if err != nil {
                return err
        }
        defer r.Close()
        …
}

func check(rc io.ReadCloser, err error) io.ReadCloser {
        if err == nil {
                return rc
        }
        panic(err)
}
```

```
func CopyFile(src, dst string) error {
        r := check(os.Open(src))
        defer r.Close()
        …
}




func check(rc io.ReadCloser, err error) io.ReadCloser {
        if err == nil {
                return rc
        }
        panic(err)
}
```

```
func CopyFile(src, dst string) error {
        r := check(os.Open(src))
        defer r.Close()
        …
}
```

Two values go into check

```
func check(rc io.ReadCloser, err error) io.ReadCloser {
        if err == nil {
                return rc
        }
        panic(err)
}
```

```
func CopyFile(src, dst string) error {
        r := check(os.Open(src))
        defer r.Close()
        …
}
```

Two values go into check

```
func check(rc io.ReadCloser, err error) io.ReadCloser {
        if err == nil {
                return rc
        }
        panic(err)
}
```

One value comes out

```
func CopyFile(src, dst string) error {
        r := check(os.Open(src))
        defer r.Close()
        …
}
```

check must return a value

```
func check(rc io.ReadCloser, err error) io.ReadCloser {
        if err == nil {
                return rc
        }
        panic(err)
}
```

```
func CopyFile(src, dst string) error {
        r := check(os.Open(src))
        defer r.Close()
        …
}
```

check must return a value

```
func check(rc io.ReadCloser, err error) io.ReadCloser {
        if err == nil {
                return rc
        }
        panic(err)
}
```

Crashes the whole program ☹️

```
func CopyFile(src, dst string) error {
	r := check(os.Open(src))
	defer r.Close()
	…
}



func check(rc io.ReadCloser, err error) io.ReadCloser {
	if err == nil {
		return rc
	}
	return err
}
```

```
func CopyFile(src, dst string) error {
        r := check(os.Open(src))
        defer r.Close()
        …
}



func check(rc io.ReadCloser, err error) io.ReadCloser {
        if err == nil {
                return rc
        }
        return err
}
```

We want to return the error here

Check is added to the language

Go programmers cannot write the own check functions today as we cannot return to the caller of the caller of check.

So the Go team are adding a new `check` keyword to the language which does exactly this.

```go
func CopyFile(src, dst string) error {
	r, err := os.Open(src)
	if err != nil {
		return err
	}
	defer r.Close()

	w, err := os.Create(dst)
	if err != nil {
		return err
	}
	defer w.Close()

	if _, err := io.Copy(w, r); err != nil {
		return err
	}
	if err := w.Close(); err != nil {
		return err
	}
	return nil
}
```

```
func CopyFile(src, dst string) error {
        r := check os.Open(src)
        defer r.Close()

        w := check os.Create(dst)
        defer w.Close()

        check io.Copy(w, r)
        check w.Close()
        return nil
}
```

```go
func CopyFile(src, dst string) error {
	r, err := os.Open(src)
	if err != nil {
		return err
	}
	defer r.Close()

	w, err := os.Create(dst)
	if err != nil {
		return err
	}
	defer w.Close()

	if _, err := io.Copy(w, r); err != nil {
		return err
	}
	if err := w.Close(); err != nil {
		return err
	}
	return nil
}
```

```
func CopyFile(src, dst string) error {
	r, err := os.Open(src)
	if err != nil {
		return err
	}
	defer r.Close()

	w, err := os.Create(dst)
	if err != nil {
		return err
	}
	defer w.Close()

	if _, err := io.Copy(w, r); err != nil {
		return err
	}
	if err := w.Close(); err != nil {
		return err
	}
	return nil
}
```

Will say "couldn't open file", but why the file was being opened is lost

```go
func CopyFile(src, dst string) error {
	r, err := os.Open(src)
	if err != nil {
		return err
	}
	defer r.Close()

	w, err := os.Create(dst)
	if err != nil {
		return err
	}
	defer w.Close()

	if _, err := io.Copy(w, r); err != nil {
		return err
	}
	if err := w.Close(); err != nil {
		return err
	}
	return nil
}
```

Will say "couldn't open file", but why the file was being opened is lost

Should cleanup failed copy destination on failure

```go
func CopyFile(src, dst string) error {
	r, err := os.Open(src)
	if err != nil {
		return err
	}
	defer r.Close()

	w, err := os.Create(dst)
	if err != nil {
		return err
	}
	defer w.Close()

	if _, err := io.Copy(w, r); err != nil {
		return err
	}
	if err := w.Close(); err != nil {
		return err
	}
	return nil
}
```

Will say "couldn't open file", but why the file was being opened is lost

Should cleanup failed copy destination on failure

And remove copy if close fails

```go
func CopyFile(src, dst string) error {
	r, err := os.Open(src)
	if err != nil {
		return fmt.Errorf("copy %s %s: %v", src, dst, err)
	}
	defer r.Close()

	w, err := os.Create(dst)
	if err != nil {
		return fmt.Errorf("copy %s %s: %v", src, dst, err)
	}

	if _, err := io.Copy(w, r); err != nil {
		w.Close()
		os.Remove(dst)
		return fmt.Errorf("copy %s %s: %v", src, dst, err)
	}

	if err := w.Close(); err != nil {
		os.Remove(dst)
		return fmt.Errorf("copy %s %s: %v", src, dst, err)
	}
	return nil
}
```

```go
func CopyFile(src, dst string) error {
	r, err := os.Open(src)
	if err != nil {
		return fmt.Errorf("copy %s %s: %v", src, dst, err)
	}
	defer r.Close()

	w, err := os.Create(dst)
	if err != nil {
		return fmt.Errorf("copy %s %s: %v", src, dst, err)
	}

	if _, err := io.Copy(w, r); err != nil {
		w.Close()
		os.Remove(dst)
		return fmt.Errorf("copy %s %s: %v", src, dst, err)
	}

	if err := w.Close(); err != nil {
		os.Remove(dst)
		return fmt.Errorf("copy %s %s: %v", src, dst, err)
	}
	return nil
}
```

Add context to the error so we know what failed

```go
func CopyFile(src, dst string) error {
	r, err := os.Open(src)
	if err != nil {
		return fmt.Errorf("copy %s %s: %v", src, dst, err)
	}
	defer r.Close()

	w, err := os.Create(dst)
	if err != nil {
		return fmt.Errorf("copy %s %s: %v", src, dst, err)
	}

	if _, err := io.Copy(w, r); err != nil {
		w.Close()
		os.Remove(dst)
		return fmt.Errorf("copy %s %s: %v", src, dst, err)
	}

	if err := w.Close(); err != nil {
		os.Remove(dst)
		return fmt.Errorf("copy %s %s: %v", src, dst, err)
	}
	return nil
}
```

```go
func CopyFile(src, dst string) error {
        r, err := os.Open(src)
        if err != nil {
                return fmt.Errorf("copy %s %s: %v", src, dst, err)
        }
        defer r.Close()

        w, err := os.Create(dst)
        if err != nil {
                return fmt.Errorf("copy %s %s: %v", src, dst, err)
        }

        if _, err := io.Copy(w, r); err != nil {
                w.Close()
                os.Remove(dst)
                return fmt.Errorf("copy %s %s: %v", src, dst, err)
        }

        if err := w.Close(); err != nil {
                os.Remove(dst)
                return fmt.Errorf("copy %s %s: %v", src, dst, err)
        }
        return nil
}
```

If we use check to remove the if err != nil block, there will be nowhere to put the cleanup code

Check and handle

The solution the Go team are proposing a new statement called `handle`.

You can think of `handle` as being similar to `defer`. Control will transfer to the handle block if `err != nil`.

Just like defer, handle functions can appear anywhere during the function. If a `check` fails, it transfers control to the innermost handler, which transfers control to the next handler above it, and so on, until a handler executes a return statement.

```
func CopyFile(src, dst string) error {
        handle err {
                return fmt.Errorf("copy %s %s: %v", src, dst, err)
        }

        r := check os.Open(src)
        defer r.Close()

        w := check os.Create(dst)
        handle err {
                w.Close()
                os.Remove(dst) // (only if a check fails)
        }

        check io.Copy(w, r)
        check w.Close()
        return nil
}
```

```
func CopyFile(src, dst string) error {
        handle err {
                return fmt.Errorf("copy %s %s: %v", src, dst, err)
        }

        r := check os.Open(src)
        defer r.Close()

        w := check os.Create(dst)
        handle err {
                w.Close()
                os.Remove(dst) // (only if a check fails)
        }

        check io.Copy(w, r)
        check w.Close()
        return nil
}
```

```
func CopyFile(src, dst string) error {
        handle err {
                return fmt.Errorf("copy %s %s: %v", src, dst, err)
        }

        r := check os.Open(src)
        defer r.Close()

        w := check os.Create(dst)
        handle err {
                w.Close()
                os.Remove(dst) // (only if a check fails)
        }

        check io.Copy(w, r)
        check w.Close()
        return nil
}
```

```
func CopyFile(src, dst string) error {
        handle err {
                return fmt.Errorf("copy %s %s: %v", src, dst, err)
        }

        r := check os.Open(src)
        defer r.Close()

        w := check os.Create(dst)
        handle err {
                w.Close()
                os.Remove(dst) // (only if a check fails)
        }

        check io.Copy(w, r)
        check w.Close()
        return nil
}
```

```
func CopyFile(src, dst string) error {
	handle err {
		return fmt.Errorf("copy %s %s: %v", src, dst, err)
	}

	r := check os.Open(src)
	defer r.Close()

	w := check os.Create(dst)
	handle err {
		w.Close()
		os.Remove(dst) // (only if a check fails)
	}

	check io.Copy(w, r)
	check w.Close()
	return nil
}
```

```
func CopyFile(src, dst string) error {
        handle err {
                return fmt.Errorf("copy %s %s: %v", src, dst, err)
        }

        r := check os.Open(src)
        defer r.Close()

        w := check os.Create(dst)
        handle err {
                w.Close()
                os.Remove(dst) // (only if a check fails)
        }

        check io.Copy(w, r)
        check w.Close()
        return nil
}
```

```
func CopyFile(src, dst string) error {
        handle err {
                return fmt.Errorf("copy %s %s: %v", src, dst, err)
        }

        r := check os.Open(src)
        defer r.Close()

        w := check os.Create(dst)
        handle err {
                w.Close()
                os.Remove(dst) // (only if a check fails)
        }

        check io.Copy(w, r)
        check w.Close()
        return nil
}
```

Inspecting errors

Go programmers have two main techniques for providing information in errors. If the intent is only to describe a unique condition with no additional data, a variable of type error suffices

```
var ErrUnexpectedEOF = errors.New("unexpected EOF")
```

Programs can act on such sentinel errors by a simple comparison:

```
if err == io.ErrUnexpectedEOF { ... }
```

Inspecting errors

To provide more information, the programmer can define a new type that implements the `error` interface. For example, `os.PathError` is a struct that includes a pathname.

Programs can extract information from these errors by using type assertions:

```
if pe, ok := err.(*os.PathError); ok { ... pe.Path ... }
```

Inspecting errors

We could instead create a new type to hold additional details along with the underlying error

```
if err != nil {
        return &WriteError{Database: "users", Err: err}
}
```

However, this could break the caller as type of the error has changed to our WriteError type.

Either way, wrapping breaks both equality checks and type assertions looking for the original error. This discourages wrapping, leading to less useful errors.

ERRORS.UNWRAP

The first part of the design is to add a standard, optional interface implemented by errors that wrap other errors:

```
package errors

// A Wrapper is an error implementation
// wrapping context around another error.
type Wrapper interface {
	// Unwrap returns the next error in the error chain.
	// If there is no next error, Unwrap returns nil.
	Unwrap() error
}
```

Programs can inspect the chain of wrapped errors by using a type assertion to check for the Unwrap method and then calling it.

Is and As

Wrapping errors breaks the two common patterns for acting on errors, equality comparison and type assertion.

To reestablish those operations, the second part of the design adds two new functions: `errors.Is`, which searches the error chain for a specific error value.

```
// instead of err == io.ErrUnexpectedEOF
if errors.Is(err, io.ErrUnexpectedEOF) { ... }
```

The `errors.Is` function is used instead of a direct equality check

Is and As

The second helper is `errors.As`, which searches the chain for a specific type of error.

The `errors.As` function is used instead of a type assertion:

```
// instead of pe, ok := err.(*os.PathError)
if pe, ok := errors.As(*os.PathError)(err); ok {
        ... pe.Path ...

}
```

Error handling

`check` and `handle` for cleaning up error handling boilerplate

`errors.Is` and `errors.As` for error inspection

Generics

🎉

Why do Go programmers want generics?

Why do Go programmers want generics?

```
func Max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

Why do Go programmers want generics?

```
func Max(a, b int) int {
        if a > b {
                return a
        }
        return b
}
```

Only works with ints

GoDoc
Home
About
Search

math: github.com/pkg/math

Index | Files

package math

```
import "github.com/pkg/math"
```

Package math provides helper functions for mathematical operations over all integer Go types.

Almost all files in this package are automatically generated.

To regenerate this package

```
make -B
```

This package relies on github.com/davecheney/godoc2md.

Index

func EqualBigInt(a, b *big.Int) bool

func MinInt32

```
func MinInt32(a, b int32) int32
```

MinInt32 returns the smaller of two int32s.

func MinInt32N

```
func MinInt32N(v ...int32) int32
```

MinInt32N returns the smallest int32 in the set provided. If no values are provided, MinInt32 returns 0.

func MinInt64

```
func MinInt64(a, b int64) int64
```

MinInt64 returns the smaller of two int64s.

func MinInt64N

```
func MinInt64N(v ...int64) int64
```

Generic Max implementation

Generic Max implementation

```
func Max(a, b T) T {
        if a > b {
                return a
        }
        return b
}
```

Generic Max implementation

```
func Max(a, b T) T {
        if a > b {
                return a
        }
        return b
}

func main() {
        var A, B uint8 = 50, 90
        result := Max(A, B)
        fmt.Println(result)
}
```

Generic Max implementation

```
func Max(a, b uint8) uint8 {
        if a > b {
                return a
        }
        return b
}

func main() {
        var A, B uint8 = 50, 90
        result := Max(A, B)
        fmt.Println(result) // 90
}
```

The problem with template substitution

```
func Max(a, b T) T {
        if a > b {
                return a
        }
        return b
}

func main() {
        var A, B = "Hello", "QCon"
        result := Max(A, B)
        fmt.Println(result)
}
```

The problem with template substitution

```
func Max(a, b string) string {
        if a > b {
                return a
        }
        return b
}

func main() {
        var A, B = "Hello", "QCon"
        result := Max(A, B)
        fmt.Println(result) // "QCon"
}
```

The problem with template substitution

```
func Max(a, b T) T {
        if a > b {
                return a
        }
        return b
}

func main() {
        var A, B = []byte("Hello"), []byte("QCon")
        result := Max(A, B)
}
```

The problem with template substitution

```
func Max(a, b []byte) []byte {
        if a > b {
                return a
        }
        return b
}

func main() {
        var A, B = []byte("Hello"), []byte("QCon")
        result := Max(A, B)
}
```

The problem with template substitution

```go
func Max(a, b []byte) []byte {
        if a > b {
                return a
        }
        return b
}

func main() {
        var A, B = []byte("Hello"), []byte("QCon")
        result := Max(A, B)
}
```

Compiler complains here

The problem with template substitution

```
func Max(a, b []byte) []byte {
        if a > b {
                return a
        }
        return b
}

func main() {
        var A, B = []byte("Hello"), []byte("QCon")
        result := Max(A, B)
}
```

Compiler complains here

But the bug is actually here

The problem with template substitution

```
func Max(a, b T) T {
        if a > b {
                return a
        }
        return b
}

func main() {
        var A, B float64 = 3.1417, math.NaN()
        result := Max(A, B)
        fmt.Println(result)
}
```

The problem with template substitution

```
func Max(a, b float64) float64 {
        if a > b {
                return a
        }
        return b
}

func main() {
        var A, B float64 = 3.1417, math.NaN()
        result := Max(A, B)
        fmt.Println(result) // ????
}
```

The problem with template substitution

We need a way of applying a constraint on which types can be substituted for T

In other languages, like Java, this is called a type bound.

```
public static <T extends Number> T max(T a, T b) { ... }
```

Go doesn't have type inheretence, and we don't want to add it, we see not having inheretence as a feature, not a bug.

Contracts

The suggestion the Go team have come up with is called a contract.

A contract is a way to write down a list of requirements for a type implementing T

```
contract comparable(t T) {
        t > t
        t << 1
}
```

Comparable contract

```
contract comparable(t T) {
        t > t
        t << 1
}
```

Comparable contract

```
contract comparable(t T) {
          t > t
          t << 1
}
```

T must be a type with a greater than operator. This excludes slices, maps channels, or structs.

Comparable contract

```
contract comparable(t T) {
        t > t
        t << 1
}
```

T must be a type with a greater than operator. This excludes slices, maps channels, or structs.

T must be a type that can be shifted, this excludes float64.

How do we use a contract?

```
contract comparable(t T) {
        t > t
        t << 1
}

func Max(type T comparable)(a, b T) T {
        if a > b {
                return a
        }
        return b
}
```

How do we use a contract?

```
contract comparable(t T) {
        t > t
        t << 1
}

func Max(type T comparable)(a, b T) T {
        if a > b {
                return a
        }
        return b
}
```

Formal parameters

How do we use a contract?

```
contract comparable(t T) {
        t > t
        t << 1
}

func Max(type T comparable)(a, b T) T {
        if a > b {
                return a
        }
        return b
}
```

Return values

Formal parameters

How do we use a contract?

```
contract comparable(t T) {
        t > t
        t << 1
}

func Max(type T comparable)(a, b T) T {
        if a > b {
                return a
        }
        return b
}
```

Type parameter

Formal parameters

Return values

THE GENERIC DILEMMA

The generics debate in Go is not new. Years ago Russ Cox wrote a short post called the Generics Dilemma, on the three approaches to adding generics to *any* language

1. Don't do it. This is the approach C tool, and, until now, the approach Go chose.

2. Compile-time specialisation or template expansion. This is the C++ approach, It generates a lot of code, much of it redundant, and needs a good linker to eliminate duplicate copies. This slows down compliation

3. Box everything and insert casts at runtime. This is the Java approach.

Go generics don't dictate how the compiler will implement them

The important thing to recognise in this proposal is the syntax I shown in the previous slides does not dictate how the feature will be implemented.

Unlike the C++ implementation which is explicitly defined to rely on template text substitution, or the java solution which requires boxing every patameter into an object, this proposal does not specify how the compiler should implement this feature.

The Go compiler may choose to specialise a generic function at compile-time or use run time boxing and casting. The decision becomes purely a compiler optimization, not one of semantic significance.

Would you like to know more?

If you'd like to know more, read the design documents, and importantly contribute your feedback on these proposals at this page

https://blog.golang.org/go2draft

Go modules implementation is much further along, as I mentioned, its available to try in Go 1.11 today, so feedback and experience reports are best directed to the issue tracker.

There will be no Go 2

And that's OK

Stability

Go developers recognise that over the last 9 years the value Go has bought to you is not what has been added to go, but what has not changed.

The value in Go is the huge base of software written in the language that was defined in 2012, and which we’ve been using productively since then.

Backwards compatibility

The value of Go is in the commitment to backwards compatibility that the Go 1 contract bought us for the last nine years.

ADOPTION

The value in language is all of you in this room today. Because ultimately a programming language is only successful if it has a large user base of people who are happy to continue to use it.

Towards Go 2

In a few months it will be December, then January 2019.

2019 is a whole new year, distinct and separate from the previous 365 days of 2018.

Yet, except for changing the year, January 1st 2019 will be in every other respect just a continuation of December 31st, 2018.

Towards Go 2

For all of the Go users today, Go 2 is not a single release we're working towards.

Just like one day following the next, the progress of small, frequent, releases will continue, adding these features that I discussed today,—and maybe a few other small tweaks—until one day we decide to call it Go 2.

Thank you!

Enjoy QCon Shanghai 2018

Egon Elbre