Go!

xuli@qiniu.com

What?

- ~~Godaddy（“去你爹”，X）~~
- ~~Go a head（“去个头”，X）~~
- Golang (Go语言，YES)

Go a head...

Golang is

一个在语言层面实现了
并发机制的类C通用型
编程语言

Why Golang?

- 云计算时代，多核化、集群化、分布式是趋势
- 硬件发展很快，软件发展太慢
- 传统编程语言并发支持比较繁琐或不支持多核
- 既不损失性能，又能保证生产效率，何乐不为

Go的前世今生

- 1995 年，贝尔实验室九号计划在开发的分布式操作系统 Inferno 包含了一个用于编写分布式系统的编程语言 Limbo，这是能追溯到最早接近于 Go 的雏形。
- 2007/09，作为 Google 里边 20% 自由时间里的项目
- 2008/05，发展为 Google 里边 100% 全时项目
- 2009/11，Google 官方首次对外公开透露
- 2012/03，Go 1.0 正式发布

Go 的一些开源项目

- docker - 基于 Linux 容器技术的一个虚拟化工具，能够轻易实现 PAAS 平台的组建
- packer - vagrant 的作者开源的用来生成不同平台的镜像文件，例如QEMU、KVM、Xen、VM、vbox、AWS等
- drone - 基于 docker 构建的持续集成测试平台，类似 jenkins-ci
- libcontainer - Docker 官方开源的用 Go 实现的 Linux Containers
- tsuru - 开源的PAAS平台，类似 GAE、SAE
- groupcache - memcahe 作者(Brad Fitzpatrick) 写的用于 dl.google.com 的缓存系统
- nsq - bit.ly 开源的高性能消息队列系统，用以每天处理数十亿条的消息
- influxdb - 开源分布式时序、事件和指标数据库
- heka - Mozilla 开源的日志处理系统
- doozer - 分布式同步工具，类似 ZooKeeper
- etcd - 高可用的 Key/Value 存储系统，主要用于分享配置和服务发现。etcd 的灵感来自于 ZooKeeper 和 Doozer
- goandroid - 使之用 Go 编写动态库，在原生的 Android 应用中运行
- mandala - 基于 goandroid 的工具链，用 Go 编写原生的 Android 应用的一个便捷框架
- 更多：https://code.google.com/p/go-wiki/wiki/Projects

Go 的一些实践公司

- 国外：Google、YouTube、Dropbox、dotCloud、10gen、Apcera、Mozilla、Heroku、Github、Bitbucket、bitly、CloudFlare、Cloud Foundry、Flipboard、Disqus、SendGrid、Tumblr、Zynga、Soundcloud
- 更多：https://code.google.com/p/go-wiki/wiki/GoUsers
- 国内：七牛云存储、京东云平台、盛大云CDN、仙侠道、金山微看、Weico、西山居、美团、豆瓣、小米商城、360
- 更多：https://github.com/qiniu/go/issues/15

主题：Go 语言之美

常见编程范式

- 过程式
  - C
- 面向对象
  - Java、C#、Ruby
- 面向消息
  - Erlang
- 函数式
  - Haskell

Go 的编程范式?

范式	YES / NO
过程式	YES
面向对象	YES
面向消息	YES（支持但不纯粹）
函数式	YES（支持但不纯粹）

Go 的编程范式

Go 无门无派

- Go 支持过程，但只是语言基础特性。

- Go 支持对象，但将特性最小化，只作为语言基础特性。Go 甚至反对继承，拒绝提供继承语法。

- Go 支持消息(Channel)，但并没有杜绝共享内存，只是将消息作为语言基础特性。

- Go 支持闭包(Closure)，但并没有试图成为纯粹的函数式语言，只是将其作为语言基础特性。

Go语言之美

1. 设计哲学——大道至简

2. 语言层面天然支持并发（goroutine & channel）

3. 优雅的错误处理规范（内置error类型、defer、func多返回值）

4. 极度简化但完备的面向对象表达（OOP）

5. 类型系统的纲——非侵入式接口（Interface）

6. 连接与组合（Pipeline、UNIX编程艺术）

一、大道至简

- 基础哲学：继承自 C、C— 而不是 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
- 少就是指数级的多
  - 最少特性原则
  - 25 个关键字表达所有
  - 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

二、并发编程

```java
// in Java (简化，用标准库中的线程模拟并发)

public class MyThread implements Runnable {

    String arg;

    public MyThread(String a) {
        arg = a;
    }

    public void run() {
        // ...
    }

    public static void main(String[] args) {
        new Thread(new MyThread("test")).start();
        // ...
    }
}
```

```go
// Go 的并发

func run(arg string) {
    // ...
}

func main() {
    go run("test")
    ...
}
```

并发的单位 - goroutine

```
//启动一个异步过程

func foo(arg1 T1, arg2 T2) {
    // ...
}
go foo(arg1, arg2)
```

goroutine:
- 轻量级执行体 (类比：协程/纤程)
- 无上限 (只受限于内存)、创建/切换成本低
- 但注意不要当做是零成本,还是应该留心创建 goroutine 频度与数量

并发之间通信 - channel

```
c := make(chan int)

c<-1      // 发送整数 1 到 channel c
<-c       // 从 channel c 接值，并丢弃
i := <-c  // 接值，并初始化赋值给变量 i
```

channel:
- Go 的内建类型
- 本质上是一个 MessageQueue
- 非常正统的执行体间通讯设施

Go 的并发特性

- 无需共享内存，更不用内存锁，并发靠语言自带的 goroutine
- goroutine 之间的通信靠语言自带的 channel 来传递消息

并发 vs. 并行

- 并发(Concurrency) 不等于 并行(Parallelism)
- 并发是指程序的逻辑结构；并行是指程序的运行状态；并行依赖硬件支持（多核）。
- 并发是并行的必要条件；但并发不是并行的充分条件。并发只是更符合现实问题本质的表达，目的是简化代码逻辑，而不是使程序运行更快。要让程序运行更快必是并发程序 +多核并行。

并发编程(Concurrency)

```
// 用 channel 等别人的结果

done := make(chan Result, 1)
go func() {
    ...
    done <- Result{}
}()
...
result := <-done
fmt.Println(result)
```

并发编程(Concurrency)

```
// 不永久等别人的结果，对方可能有异常 (考虑timeout)

done := make(chan Result, 1)
go func() {
    ...
    done <- Result{}
}()
...
select {
case result := <-done:
    fmt.Println(result)
case <- time.After(3 * time.Second):
    fmt.Println("timeout")
}
```

并发编程(Concurrency)

```
// 给多个人同样的活,谁先干完要谁的结果(考虑timeout)

done := make(chan Result, 3)
for i := 0; i < 3; i++ {
    go func() {
        ...
        done <- Result{}
    }()
}
...
select {
case result := <-done:
    fmt.Println(result)
case <- time.After(3 * time.Second):
    fmt.Println("timeout")
}
```

并发编程(Concurrency)

- 生产者 / 消费者模型

  - 并行编程中，多个 goroutine 间符合生产者/消费者模型非常常见。

  - channel 用于生产者和消费者间的通信，并适配两者的速度。

  - 如果生产者速度过快，那么它会 channel 缓冲区满时停下来等待；如果消费者速度过快，则在 channel 缓冲区空时停下来等待。

并发编程(Concurrency)

- 误区 / 陷阱

    - 用 channel 来做互斥 (正常应该让 Mutex 做)，比如多个 goroutine 访问一组共享变量。

- channel的成本

    - 作为消息队列,channel 成本原高于 Mutex

    - 成本在哪?

        - channel内部有Mutex,因为它本身属于共享变量

        - channel内部可能有Cond,用来等待或唤醒满足条件的 goroutine

        - 出让 cpu 并且让另一个 goroutine 获得执行机会,这个切换周 期不低,远高于 Mutex 检查竞争状态的成本(后者通常只是一 个原子操作)

三、优雅的错误处理规范

```java
// In Java (普通资源释放)

Connection conn = ...;
try {
    Statement stmt = ...;
    try {
        ResultSet rset = ...;
        try {
            ... // 正常代码
        }
        finally {
            rset.close();
        }
    }
    finally {
        stmt.close();
    }
}
finally {
    conn.close();
}
```

```go
// in Go (Go的资源释放)

conn := ...
defer conn.Close()

stmt := ...
defer stmt.Close()

rset := ...
defer rset.Close()

... // 正常逻辑代码
```

Go的错误处理范式

```
// 文件操作
file, err := os.Open(fileName)
if err != nil {
    return
}
defer file.Close() // 有事没事，defer 一下
... // 操作已经打开的 f 文件


// 锁操作
var mutex sync.Mutex
// ...
mutex.Lock()
defer mutex.Unlock()
... // 正常代码逻辑
```

内建error, 多值返回

```
type error interface {
    Error() string
}

// 将错误抑制在它所出现的地方，避免层层堆叠
result, err := someExpression()
if err != nil {
      // …
}
```

四、极度简化但完备的OOP

- 废弃大量的 OOP 特性
  - 如:继承、构造/析构函数、虚函数、函数重载、 默认参数等。
- 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。

结构体（Struct）

```
// 是类，不只是结构体
type Foo struct {
    a int
    B string // 可作为类的成员变量导出
}

func (this *Foo) Bar(arg1 T1, arg2 T2, ...) (out1 RetT1, ...) {
    // ...
}
```

从 Struct 定义 OOP

```
type Point struct {
    x, y int
}

func (p *Point) Get() (int, int) { // 公开Public
    return p.x, p.y
}

func (p *Point) Put(x, y int) { // 公开Public
    p.x = x
    p.y = y
}

func (p *Point) add(x, y int) int { // 私有private
    return p.x + p.y
}
```

通过组合模拟继承

```
type YetAnotherPoint struct {
    Point  // 继承 Point 类
    ...
}
```

```
type YetAnotherPoint struct {
    *Point // 虚拟继承 Point 类
    ...
}

type YetAnotherPoint struct {
    Pointer // 继承 Pointer 接口
    ...
}
```

五、非侵入式接口 (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。

非显示声明接口实现

```
type Pointer interface {
    Get() (int, int)
    Put(x, y int)
}

type Point struct  {
    x, y int
}

func (p *Point) Get() (int, int) {
    return p.x, p.y
}

func (p *Point) Put(x, y int) {
    p.x = x
    p.y = y
}
```

接口查询

```
var a interface{} = ...
if w, ok := a.(io.Writer); ok {
    // ...
}
if foo, ok := a.(*Foo); ok {
    // ...
}
```

六、连接与组合（Pipeline）

- Pipeline 与并行模型
  - 在 Go 中实施 Pipeline 非常容易
  - 在 Go 中让任务并行化非常容易
- 连接：组件之间的耦合方式
  - 非侵入式的 interface, 组件间的协议由 interface 描述, 松散耦合
  - 抽象的 io.Reader, io.Writer 和 Pipe
- 组合：形成复合对象的基础
  - 强大的组合能力（匿名组合、指针组合、接口组合）
  - 不支持继承，却胜过继承
  - 不是 COM，但更胜 COM
- http://open.qiniudn.com/thinking-in-go.mp4

源于 Unix 哲学

- Unix 的连接和组合
    - app1 params1 | app2 params2
- App 接口
    - 输入：stdin, params
    - 输出：stdout
    - 协议：text (data stream)
- Pipeline
    - 将一个 app 的输出(stdout) 转为另一个 app 的输入 (stdin)

Pipeline 关键点

- 多个 app 是并行执行的

  上游每产生一段output，会立即交由下游处理。

- app 间的协议是松散耦合的

  上游 app 的 ouput 是 xml 还是 json，下游 app 需要知晓，但是属于一种松散的耦合关系，并无任何强制的约束。

Go 对 Unix Pipeline 的仿真

```
func pipe(
    app1 func(in io.Reader, out io.Writer),
    app2 func(in io.Reader, out io.Writer)
) func(in io.Reader, out io.Writer) {
    return func(in io.Reader, out io.Writer) {
        pr, pw := io.Pipe()
        defer pw.Close()
        go func() {
            defer pr.Close()
            app2(pr, out)
        }()
        app1(in, pw)
    }
}
```

Go 对 Unix Pipeline 的仿真

```
func pipe(apps ...func(in io.Reader, out io.Writer)) func(in io.Reader, out io.Writer) {
	if len(apps) == 0 { return nil }
	app := apps[0]
	for i := 1; i < len(apps); i++ {
		app1, app2 := app, apps[i]
		app = func(in io.Reader, out io.Writer) {
			pr, pw := io.Pipe()
			defer pw.Close()
			go func() {
				defer pr.Close()
				app2(pr, out)
			}()
			app1(in, pw)
		}
	}
	return app
}
```

结论

- 在 Go 中实施 Pipeline 非常容易
- 在 Go 中让任务并行化非常容易

Go在七牛的实践

- 分布式键值存储系统 (Distributed Key/Value Storage)
- 数据处理服务 (Data Processing)
- 网络接口服务 (RESTful API Service)
- 消息队列服务（Message Queue Service）
- 日志处理系统 (Log Service)
- Web 网站 (不含前端 JavaScript)
- CLI 命令行和 GUI 图形界面工具
- 其他辅助工具

- 90% 的代码都是 Go, 剩余 10% 主要覆盖 web 前端和多达十几种编程语言的 SDK。

链接

- Collison预言：Go语言将在两年内制霸云领域!
- Hacker News: Google Trends: Golang is popular in China
- 七牛首席布道师：Go不是在颠覆，就是在逆袭
- Rob Pike: Concurrency is not Parallelism (it's better)
- 许式伟：Go，基于连接与组合的语言

鸣谢

- Google (Rob Pike、Ken Thompson, ..)
- Qiniu (许式伟、七牛云存储)
- Go Communities
- Github

Q & A

- Twitter: why404
- Wechat: why404