# go的接口和面向对象编程

kodo后台开发：刘坚君

# go的起源

## 起源于使用C++开发带来的种种不爽

Google首席工程师、Go语言之父Rob Pike：

回想2007年9月的时候，我正在为Google庞大的C++程序做一些比较琐碎但是核心的工作，我工作在Google庞大的分布式编译集群上都需要花45分钟。之后有几个C++标准委员会的Google员工为我们做了个演讲，他们给我们介绍了下C++0x（现在被叫做C++11）里会有什么新东西。

会上，Rob的内心在各种吐槽。。。

演讲结束后，我们又回到了办公室。我又开始了编译工作，我转过椅子好面向Robert，然后开始问一些关键性的问题。在编译结束之前，我们说服了Ken，决定一起做一些事情。我们再也不想用C++了，同时也很希望能在写Google的代码时能用到并行性特性。

# 一个值得思考的问题

在很多方面，go提供的编程机制，是传统语言的进一步优化

但是在面向对象机制上，却是全新的设计

# 进一步的优化

| 语言特性 | go | c/c++ | 优化点 |
| --- | --- | --- | --- |
| 语法 | 更简洁的语法，如去掉;号，defer机制，函数多个返回值 | 语法更复杂 | 一切必须简化得不能再简化 |
| 垃圾回收 | 自动回收 | 手工回收 | 自动解决琐碎但有重要的内存管理问题 |
| 异常处理 | panic/defer/recover | try/catch/finally | 异常处理和控制逻辑分离，使得异常处理不干扰正常逻辑 |
| 项目管理和编译速度 | package | Makefile | 自动完成Makfile要描述的事情<br>闪电般的编译速度 |
| 并发编程 | goroutine&channel | 多线程&共享内存 | 比线程更小的开销，更合理的数据共享机制 |
| more。。。 | | | |

# 面向对象机制的全新设计

## why?

c++/java的面向对象两大机制：

1.强大的类：封装，继承，重载，构造/析构函数，多态

2.侵入式接口机制：一个类要实现某个接口，必须在定义时显式说明

go：

1.没有类的概念，只有结构体的概念。结构体=成员变量+成员方法。结构体之间没有继承，没有构造/析构函数。

2.非侵入式接口：一个结构体只要实现了某个接口的所有函数，就认为实现了该接口

# 内容

1.理论：go面向对象的语法
2.实践：一起动手写一个程序（采用面向对象方法）
3.探讨：go语言面向对象的设计哲学
4.锻炼：练习题

# go语言
# 面向对象的语法详解

其实非常简单，4页ppt说完

# 结构体

1.语法上，go的结构体对应java的类,有成员变量，有成员方法
2.go的结构体没有构造函数，提供了一种简洁的结构体初始化方式。没有析构函数，结构体对象由go的垃圾回收器自动回收
3.go的结构体可以匿名组合。组合原则：组合了一个结构体，就拥有该结构体的变量和方法
4.如果结构体中的变量，首字母大写，那么这个变量能够被外面的pkg访问到，否则，只能在本pkg内访问

```go
type GoCoder struct{
    Name string
}

func (coder *GoCoder)WriteCode()(sf SoftWare, err error){
    //write code and software
}

func main(){
    gc := GoCoder{
        Name: "xiaomin",
    }
    gc.WriteCode()
}
```

```go
type Tester struct {
        Name string
        Sex  string
}
func (tester *Tester) TestCode() (err error) {
        //test code and software
        return
}
type GoCoder struct {
        Name string
        Tester
}
func (coder *GoCoder) WriteCode() (err error) {
        //write code and software
        return
}
func main() {
        gc := GoCoder{
                Name: "xiaomin",
        }
        gc.Name = "xiaoli"
        gc.Tester.Name = "laoA"
        gc.Sex = "male"
        gc.WriteCode()
        gc.TestCode()
        fmt.Println(gc) //result:{xiaoli {laoA male}}
```

# 接口

接口的概念：
和java的接口一样，是一组方法的集合。接口是一种规范，描述了要做什么。而实现了该接口的结构体，则描述了怎么做。

侵入式和非侵入式的区别：
java的接口是侵入式的，如果一个类要实现该接口，需要显式声明。
而go语言的接口是非侵入式的，在定义go的结构体时，不需要显式声明要实现哪个接口。只要实现了该接口所有的方法，就实现了该接口。

```java
public interface Coder {
    int WriteCode();
}

class JavaCoder implements Coder {
    Name string;
    int WriteCode(){
        //...
    }
}
```

```go
type Coder interface{
    WriteCode()(err error)
}

type GoCoder struct{
    Name string
}

func (c *GoCoder)WriteCode()(err error){
}

var coder Coder
goCoder := GoCoder{
    Name:"xiaomin",
}

coder = goCoder //ok
```

# any类型： interface{}

任何类型的变量，都可以赋值给interface{}变量。如：

```
var v1 interface{} = 1
var v1 interface{} = "abc"

type A struct{
    Name string
}
var v1 interface{} = A{}
var v1 interface{} = &A{}

funcA := func(i int) (err error) {
        fmt.Println(i)
        return
}
var v1 interface{} = funcA
```

# 接口查询

1.查询某个接口对象，是否是某个结构体对象
2.查询某个接口对象指向的结构体对象，是否实现了另外一个接口

```go
type Coder interface{
    WriteCode()(err error)
}

type BusinessMan interface{
    GetMoney()(err error)
}

type GoCoder struct{
    Name string
}

func (c *GoCoder)WriteCode()(err error){
    return
}

func (c *GoCoder)GetMoney()(err error){
    return
}
```

```go
goCoder := GoCoder{
    Name:"xiaomin",
}

var c Coder
c = goCoder

//查询coder是否指向的是一个GoCoder对象
gc, ok := c.(GoCoder)
if !ok{
    //如果失败，说明c并不是GoCoder的对象
}

//GoCoder实现了BusinessMan接口，为此可以赋值
var bm BusinessMan
bm = goCoder

//查询这个BusinessMan接口对象，是否实现了Coder接口
c, ok = bm.(Coder)
if ok{
    fmt.Println("this businessman is also a coder")
}
```

# 问题?

# 实践：
# 一起动手写一个程序

分别用java和go来实现
来对比一下，go和java面向对象机制的不同之处

# 虚拟一家软件开发公司的运作流程

老Y开了一家软件开发公司，公司的运作规则很简单：
1.总投资100万
2.老Y自己跑业务，获取一个订单，订单有价格（如果失败就休息一天然后继续）
3.通过人才市场，雇佣一个程序员编写代码（如果没找到，就退回订单并赔偿2倍）
4.该程序员编写好代码（如果开发失败，则公司倒闭，创业失败）
5.交付订单，获得收入
6.付给程序员money，然后结束雇佣（money=订单价格/10）
7.获取下一个订单

一直到100万投资耗尽，或者公司资产超过1000万，结束创业。

# java的实现

请一位同学来写一个程序，模拟该软件公司的运作流程

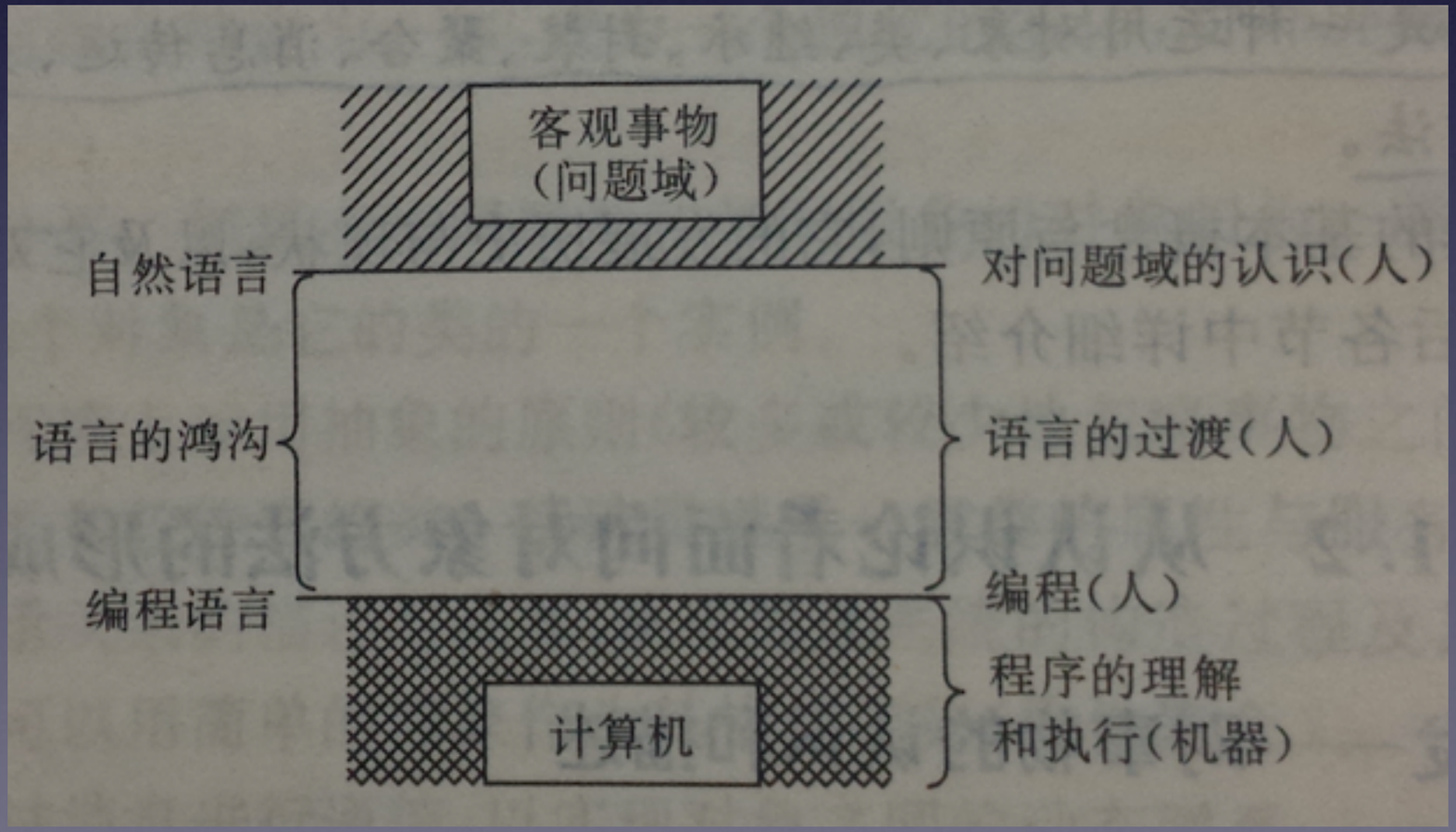
假设现在市场上招聘不到程序员了，但发现一个牙医很会写代码，
该怎么办?

# go的实现

https://github.com/jianjunliu/qiniu-study/blob/master/jianjunliu/software_company.go

# 非侵入式接口背后的编程哲学

编程的本质
是将某个问题的解
从现实世界转换到机器世界



程序员使用的语言至关重要，
它决定了程序员的看待问题和
分析问题的方式

手里拿着锤子,看什么都像是钉子

# 语言的变革

1.机器语言： 用0和1编程是什么感觉？活着的地球人没多人少知道
2.汇编语言： 从机器语言提炼了助记符，编写程序方便一点，减轻心智负担
3.结构化编程语言（c/fortran）： 有了数据抽象和结构抽象，更易于地球人理解了，但是描述能力还是偏机器世界
4.面向对象语言（c++/java）：能够直接描述问题域中，客观存在的事物，以及他们之间的关系。使开发人员能够运用人类认识事物的一般思维方法来进行软件开发。

面向对象语言和人类自然语言之间的鸿沟是最小的。

# 为什么会有类、继承、封装、多态等概念

跟现代人类认识世界的一般思维方法有关。这种思维方法，就是对知识进行分类。

学科分类：物理 化学 生物 地理，每个学科又有更细的划分
医院科室分类：内科 外科 骨科 肝胆科

分类思想的历史渊源：
在亚里士多德之前，科学家和哲学家都力求提出一个完整的世界体系，来解释自然现象。但是自亚里士多德之后，科学和哲学出现分野，许多科学家放弃提出完整体系的企图，转而研究具体问题。

而正是亚里士多德在历史上第一个提出了对知识进行分类的思想。从此后科学脱离哲学和神学，开始独立发展

作为现代科学发展顶峰的数理逻辑和计算机科学，在解决填补语言鸿沟的问题时，自然而然地想到了面向对象，然后自然而然地使用了分类的思想。于是，类这个概念就诞生了。由此演化出继承、重载、多态。

有了面向对象这个利器，程序员开始能够对现实世界的问题，基于严谨的逻辑，进行更直接和透彻的建模，带来的好处是由于事先精确的分析和建模，项目施工流程更可控，返工现象更少。由此软件开发行业，进入了飞速发展的阶段。

从1980到2010的三十年，是传统面向对象语言和建模方法，高速发展的三十年。在金融系统、企业ERP、电子政务上效果尤为显著。

但是最近几年来，传统面向对象语言逐渐力不从心了。

以分类思想为基础的传统面向对象语言，更适合：
静态的、物理的、传统行业的，以项目为单位进行开发的世界
但是对于：
动态变化的、字节的、互联网行业的，以产品为单位进行开发的世界
传统面向对象语言分析和建模理念，是一种不必要负担

因为分类的方式：

1.过早地施加了假定和约束（牙医和程序员的例子），使得后面调整的成本过高

2.互联网开发模式下，强调的是产品的抢先推出，动态生长。开发模式是小步快跑的迭代式开发。在每个小迭代里，不需要复杂的OO建模；而由于迭代粒度都很小，所以每次都很容易推翻前面的架构重新开发，使得系统越来越健壮

3.互联网服务天生存在分布式问题。而传统的面向对象方法，为解决分布式问题提供的机制有限

分类的软件建模方法，面对动态变化的世界不管用了。
怎么办?

# go语言尝试突破

## 基于连接和组合，而不是基于分类

使用简单的几条法则:

1.放弃分类：分类过分强调对象和对象之间是什么关系。而：“真正重要的是这些类能为你做什么而不是它们之间是什么关系！(Rob Pike)”

2.非侵入式接口：一个对象实现了某个接口的所有函数，就能够成为这个接口的对象

3.对象A聚合了对象B，就具备对象B的能力

带来的好处是:

1.对象和对象能够更自由地连接。只要实现了对方所要求的方法，就可以能够接入到对方

2.放弃对问题域完整的，深入的建模，完美地抽象，走务实路线，希望代码容易写，容易读，效率高，go的最终目标是解放程序员，而不是让程序员建构一切和控制一切。

# go语言的未来

go语言代表的实用派 VS c++/java代表的学院派

动态生长（自由组合连接&小步快跑，快速迭代&快速简单地完成编码）

能否胜于

一次到位（对问题域的精确建模&稳步实施&把代码写到完美）

# 写一个程序，更真实地模拟一家软件公司

老Y决定创业，打算开一家软件公司。公司的运作很简单：

1.总投资100万

2.老Y自己去跑业务拉订单。拉到的订单，都有报价和交付时间

3.如果连续3个月都没有拿到订单，那么创业失败

4.如果拿到订单，那么从人才市场上，招聘一个程序员，负责写软件完成订单，招聘时，双方协商好工资

5.程序员开发软件，完成订单。如果完成，则交付订单，并向程序员支付之前协商好的工资+订单报价/10。如果按期没有完成，那么这次生意失败，公司需要按订单报价的两倍进行赔偿，程序员只能获得工资

6.订单交付完成后，老Y和程序员一起决定，是离开公司还是继续留下。如果继续留下，那么下次拿到订单后，就不需要再招聘程序员

7.回到第2步：老Y去跑市场，获取下一个订单，开始下一个循环

按照以上流程进行循环，直到公司资金超过1000万，或者100万投资耗尽，老Y结束创业。

一些规则：

1.订单的金额和交付时间：从市场上获取到的订单的金额和交付时间，可以由程序随机产生。订单金额单位为万，交付时间单位为月

2.程序员的雇佣：雇佣程序员需要一定的时间（可以随机产生），这个时间要算到订单开发时间里面

3.程序员的工资：招聘程序员时，程序员工资可以随机产生，但不能超过老Y指定的上限。

4.程序员开发软件的时间：开发软件的时间可以随机产生，单位为月，但不能超过订单交付时间，超过后则开发失败