程序设计语言的 OO 特性调研报告

本报告中，作者调研了两种之前了解甚少的语言—— Go & Ruby ——的面向对象特性并与作者之前比较熟悉的 Java 语言进行对比。

- Go 语言

Go 语言是 Google2009 年发布的一款编程语言，可以在不损失应用程序性能的情况下降低代码的复杂性，且可以快速编译，更加安全，支持并行进程。

♦ 类、对象、属性、操作、封装

Go 不支持传统意义上的面向对象。类、对象、属性是使用类似于 C 语言 struct 的方式实现，而方法(操作)通过限定只能应用于某类 type 的实例的函数实现。例如：

```go
package try
import "strconv"

type Student struct {
    Name string
    Age  int
}
func (s *Student) SetName(name string) {
    s.Name = name
}
func (s *Student) GetName() string {
    return s.Name
}
func (s *Student) SetAge(age int) {
    s.Age = age
}
func (s *Student) GetAge() int {
    return s.Age
}
func (s *Student) String() string {
    return "name is " + s.Name + ",age is " + strconv.Itoa(s.Age)
}
```

```go
package main
import (
    "fmt"
    "try"
)
func main() {
    ss := new(try.Student)
    ss.SetName("name")
    ss.SetAge(20)
    fmt.Println(ss.String())
}
```

上面代码示例中，Student 是一个“类”，实际上用 struct 实现。在 struct 中可以定义对象的属性。而用一些限定调用实例 type 的函数定义了该“类”的一系列 get 和 set 操作。而在 java 中用 class 声明类并在类内定义方法和属性。

java 中为类、方法、字段声明 private、public 等来控制其可访问性，起到封装的作用。而在 Go 语言中，函数或字段名称的首字母大小写是用来声明其封装特性的：只有首字母大写的字段和方法才可以在包外被调用。

- 继承和多态

java 中类之间可以继承，并通过在基类中定义抽象方法，通过基类引用调用子类实例来实现多态。不过 java 中没有多继承，但类可以实现多个接口，在功能上相当于多继承。

在 Go 中，没有继承的概念，有接口的概念，但不需要显式地声明 struct 实现了什么接口，而是只要在事实上实现了这个接口的所有方法就隐式地认为实现了该接口。在 Go 语言中，通过接口可以类似 java 地实现多态的特性。

可以通过在定义的新 struct 中声明一个匿名的 strcut 类型的字段来变通地实现一些继承的特性。

例如：

```
package try
import "strconv"
type IPeople interface {
    SetName(string)
    GetName() string
}
type Student struct {
    Name string
    Age  int
}
type Teacher struct {
    Name   string
    Course string
}
func (s *Student) SetName(name string) {
    s.Name = name
}
func (t *Teacher) SetName(name string) {
    t.Name = name
}
func (s *Student) GetName() string {
    return s.Name
}
func (t *Teacher) GetName() string {
    return t.Name
}
func (s *Student) SetAge(age int) {
    s.Age = age
}
func (s *Student) GetAge() int {
    return s.Age
}
func (s *Student) String() string {
    return "name is " + s.Name + ",age is " + strconv.Itoa(s.Age)
}
func (t *Teacher) SetCourse(course string) {
    t.Course = course
}
func (t *Teacher) GetCourse() string {
    return t.Course
}
```

```go
package main
import (
    "fmt"
    "try"
)
type director struct {
    try.Student
    Name string
}
func (di *director) GetName() string {
    fmt.Println("get director name")
    return di.Name
}
func (di *director) SetName(name string) {
    di.Name = name
}
func main() {
    ss := new(try.Student)
    ss.SetName("name")
    ss.SetAge(20)
    dd := new(director)
    dd.Name = "director"
    dd.Student = *ss
    fmt.Println(dd.GetName())
    fmt.Println(dd.Student.GetName())
    fmt.Println(dd.GetAge())

    var ii lxy.IPeople
    ii = dd
    ii.SetName("test")
    fmt.Println(ii.GetName())
}
```

在上例中，Student 和 Teacher 都隐式地实现了 IPeople 的接口。director 中匿名地声明了 student，所以在主函数时可以直接调用 GetAge 方法，也包含 Student 中的字段，间接地实现了一些继承的功能。同时也可以通过接口的引用来间接实现多态，如最后一行的 GetName。

（聚合、关联、消息等机制应该是完全在语义层面上的，和语言语法关系不大，在各个语言中的实现方式都比较类似，在此不再讨论。后同。）

- Ruby 语言

  Ruby 是 1995 年发布的一门完全面向对象的脚本语言。思想源于 Lisp 和 Perl 语言。

  - 类、对象、属性、操作、封装

    Ruby 中所有东西都是对象，不像 java 中还保留有 int 等基本数据类型。

    Class 关键字在 Ruby 中用于声明类，类的 new 方法用于产生该类的实例。在 Ruby 中方法和字段的定义和 Java 中的类似，不过因为 Ruby 中没有变量类型的概念，所以字段不需要在赋值之前声明。

    Ruby 中变量有 4 种封装类型，局部变量以小写字母或_开头，实例变量以@为开头，类变量即 java 中的静态属性以@@为开头，全局变量以$为开头。

    代码如下例：

```ruby
class Box
   # Constructor
   def initialize(w,h)
      @width, @height = w, h
   end

   # default : public
   def getArea
      getWidth() * getHeight
   end

   # private getters
   def getWidth
      @width
   end
   def getHeight
      @height
   end
   # make them private
   private :getWidth, :getHeight

   def printArea
      @area = getWidth() * getHeight
      puts "Big box area is : #@area"
   end
   protected :printArea
end

box = Box.new(10, 20)

a = box.getArea()
puts "Area of the box is : #{a}"
```

上例中，Box 是一个类。定义有一些属性和方法。Ruby 类中的方法可以和 java 中类似地声明 public、private 等等限制。

- 继承和多态

Ruby 中有继承和函数重写，可以和 java 一样实现多态。如下例：

```ruby
class Person
  def initialize( name,age=18 )
    @name = name
    @age = age
    @motherland = "China"
  end

  def talk
    puts "my name is "+@name+", age is "+@age.to_s
    if @motherland == "China"
      puts "I am a Chinese."
    else
      puts "I am a foreigner."
    end
  end

  attr_writer :motherland

end

class Student < Person
  def talk
    puts "I am a student. my name is "+@name+", age is "+@age.to_s
  end
end
```

```ruby
class Worker < Person
  def talk
    puts "I am a worker. my name is "+@name+", age is "+@age.to_s
  end
end

p3=Student.new("kaichuan",25)
p3.talk

p4=Student.new("Ben")
p4.talk

p5=Worker.new("kaichuan",30)
p5.talk
p6=Worker.new("Ben")
p6.talk
```

上例中，Worker 类与 Student 类同样继承自 Person 类，当他们 talk 时，能准确表明自己身份，因为他们都重写了各自的 talk 方法。

而在 Ruby 中，虽然没有接口（java=）和多继承（C++），但可以使用 Mixin 来实现多继承。例如：

```ruby
module WriteStream
    def write(str)
        puts str
    end

    def conflict
        puts "conflict"
    end
end

module ReadStream
    def read
        puts "read data"
    end

    def conflict
        puts "conflict-read"
    end
end

class Stream
    def getstream
        puts "get stream"
    end
end

class ReadWriteStream < Stream
    include WriteStream
    include ReadStream
end

rw = ReadWriteStream.new
rw.getstream
rw.read
rw.write("haha")
rw.conflict
```

上面代码利用 module 的 include 来变相的实现了 C++中需要菱形继承关系才能够实现的 iostream 的逻辑。

参考资料：

http://baike.baidu.com/item/Go 语言

https://golang.org 《Learning Go》

http://blog.csdn.net/lxy15329/article/details/8423525

http://baike.baidu.com/item/Ruby/11419

http://www.ziqiangxuetang.com/ruby/ruby-class.html

http://www.educity.cn/wenda/350512.html

http://www.cnblogs.com/Jerry-Chou/archive/2011/11/17/2252230.html