Go语言面向对象编程

一、课前准备

> 说明：提前需要让学生做的课前准备，比如环境安装部署、工具安装、插件安装等，需要学生提前做的都放这，并且给出下载链接或信息源；

1. Golang开发环境环境安装就绪；
2. Golang-IDE开发环境安装就绪；
3. 练习Go语言基础语法代码；

二、课堂主题

> 说明：本堂课的总体概述，明确课堂主题和主体；

本节主要介绍Go语言当中的面向对象编程方法，介绍在Go语言当中如何体现面向对象的封装、继承以及多态，涉及到方法定义，结构化继承，接口封装等知识点。

三、课堂目标

> 说明：主要是让学生了解，学了本堂课后，能达到的一个期望值，要量化；

1. 掌握Go语言自定义结构方法；
2. 掌握Go语言方法封装要素；

3. 掌握Go语言的结构化继承；
4. 掌握Go语言接口的定义与封装；

四、知识点

Go语言也是面向对象编程的语言，而且Go语言没有像C++那样把语法发展成那样的庞然大物，让人望而生却，相反Go语言在面向对象上仍然保持其一贯简洁的风格。

1. 封装

1.1 自定义结构体

先来说说结构体，Go语言当中，使用type和struct关键字来定义结构体，比如定义一个人的结构，人的属性可以包含：姓名，年龄，性别，战斗力等。

```
type Person struct {
  Name  string
  Age   int
  Sex   string
  Fight int
}
```

我们来构造一个战斗力为5的人，就把他叫做：韩立。

```
p1 := Person{"韩立", 30, "man", 5}
fmt.Printf("%+v\n", p1) // %+v 可以清晰的打印结构体数据
```

完整运行代码如下：

```
package main

import (
  "fmt"
)

type Person struct {
  Name  string
  Age   int
  Sex   string
  Fight int
}

func main() {
  p1 := Person{"战五渣", 30, "man", 5}
  fmt.Printf("%+v\n", p1) // %+v 可以清晰的打印结构体数据
}
```

执行结果：

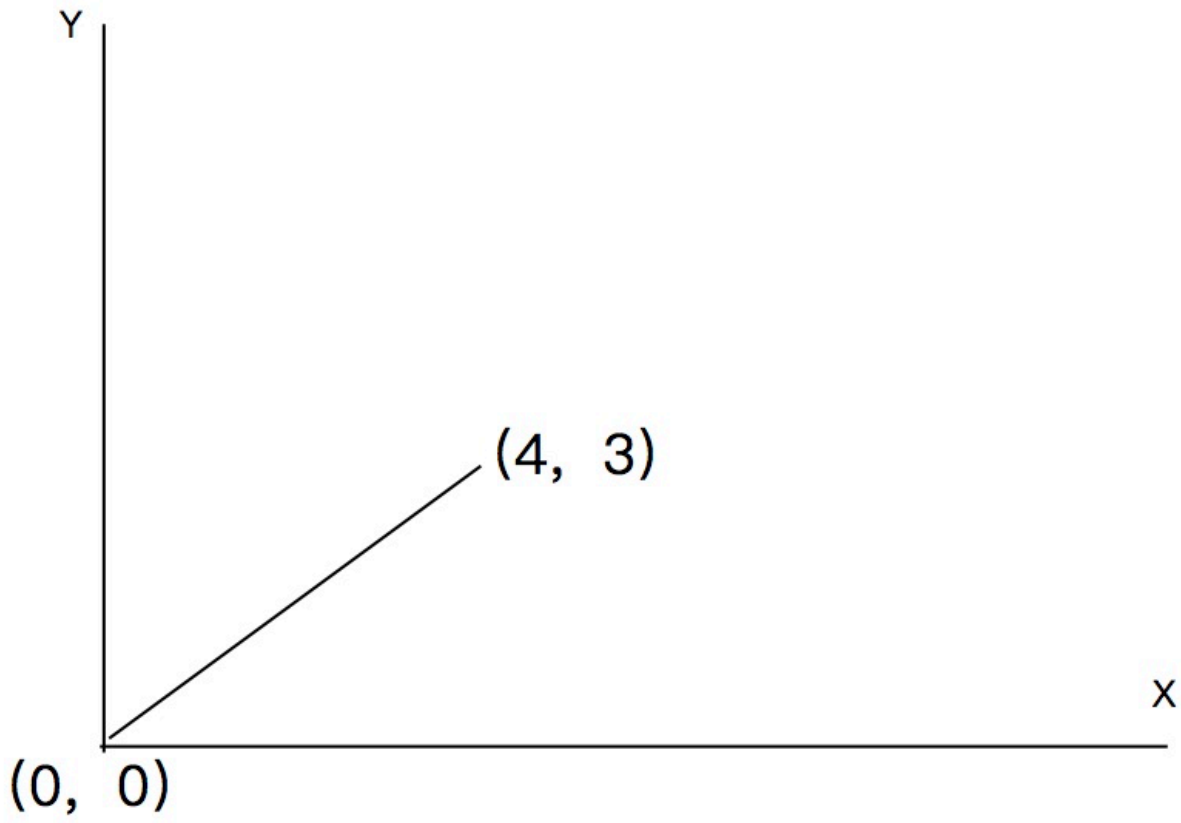
```
localhost: teacher$ go run 01-struct.go
{Name:韩立 Age:30 Sex:man Fight:5}
```

1.2 方法封装

介绍封装先要澄清一个概念就是方法和函数，函数之前我们介绍过，我们可以根据功能需要设计并实现我们想要的函数，那么什么又是方法呢？方法与函数的区别主要就是方法是特定属于某个结构体的，这就是类似于我们理解的类的概念。不过不像大多数语言那样定义，方法的定义与结构体定义是脱离的，但是作为方法必须要指定一个调用者（或接受者）。

```
func (obj ObjT) funcName([params list]) [return list] {
    do sth
}
```

我们来实现一个例子，如在平面直角坐标系内计算2个点之间的距离。



按照正常的方式，我们可以定义一个点的结构体Point(x,y)，然后定义2个节点p1,p2，之后利用数学公式：

$$dis = \sqrt{(p2.x - p1.x)^2 + (p2.y - p1.y)^2}$$

于是计算2个点之间的距离，我们可以实现这样的一函数：

```
type Point struct {
  x, y float64
}

func distance(p1, p2 Point) float64 {
  return math.Sqrt((p2.x-p1.x)*(p2.x-p1.x) + (p2.y-p1.y)*(p2.y-p1.y))
}
```

这和我们前面提到的封装和方法似乎没有关系，接下来我们考虑另外一种方式，之前的计算2个点之间的距离是以第三者的角度去观察，现在我们站在一个点的角度去考虑，这个点距离某个点的距离的计算方法只需要传入一个点就够了，于是我们可以定义下面的方法：

```
func (this Point) dis(p Point) float64 {
  return math.Sqrt((this.x-p.x)*(this.x-p.x) + (this.y-p.y)*(this.y-p.y))
}
```

(this Point) 这里的this只是点的名称，并非一定要叫this，这里也可以使用*的方式*，代表的是指针的方式，这时候对该对象是引用。distance2的调用必须是Point结构体对象才可以调用。

```
package main

import (
  "fmt"
  "math"
)

type Point struct {
  x, y float64
}

//第一种方法，传入2个点
func distance(p1, p2 Point) float64 {
  return math.Sqrt((p2.x-p1.x)*(p2.x-p1.x) + (p2.y-p1.y)*(p2.y-p1.y))
}

//第二种方法，以一个点作为参照点，再传入一个点
func (this Point) dis(p Point) float64 {
  return math.Sqrt((this.x-p.x)*(this.x-p.x) + (this.y-p.y)*(this.y-p.y))
}

func main() {
  p1 := Point{0.0, 0.0}
  p2 := Point{3.0, 4.0}
  fmt.Println(p2, "between", p1, "distance is ", distance(p1, p2))
  fmt.Println(p2.dis(p1))
}
```

执行结果如下：

```
localhost: teacher$ go run 02-enclosure.go
{3 4} between {0 0} distance is  5
call distance2 =  5
```

2. 继承

Go语言一直号称语法简洁，在继承上体现的真是淋漓尽致。我们之前定义过Person结构体，现在我们再来定义一个超人SuperMan的结构体，他继承Person的信息。可以这样写：

```
type Person struct {
  Name  string
  Age   int
  Sex   string
  Fight int
}

type SuperMan struct {
  Strength int
  Speed    int
  Person
}
```

结构体可以这样赋值：

```
    s1 := SuperMan{
    Strength: 100,
    Speed:    99,
    Person: Person{
      Name:  "hanli",
      Age:   40,
      Sex:   "man",
      Fight: 5000,
    },
  }
```

如果Person实现了函数，那么在SuperMan中可以直接调用。

```
func (p *Person) setAge(age int) {
  p.Age = age
}

func (s SuperMan) Print() {
  fmt.Printf("Name = %s, Age = %d, Sex = %s, Fight = %d\n", s.Name, s.Age,
s.Sex, s.Fight)
  fmt.Printf("strength = %d, fight = %d\n", s.Strength, s.Speed)
}
```

全部代码如下：

```
package main

import (
  "fmt"
)

type Person struct {
  Name  string
  Age   int
  Sex   string
  Fight int
}

func (p *Person) setAge(age int) {
  p.Age = age
}

type SuperMan struct {
  Strength int
  Speed    int
  Person
}

func (s SuperMan) Print() {
  fmt.Printf("Name = %s, Age = %d, Sex = %s, Fight = %d\n", s.Name, s.Age,
s.Sex, s.Fight)
  fmt.Printf("strength = %d, fight = %d\n", s.Strength, s.Speed)
}

func main() {
  s1 := SuperMan{
    Strength: 100,
    Speed:    99,
    Person: Person{
      Name:  "hanli",
      Age:   40,
```

```
        Sex:   "man",
        Fight: 5000,
    },
  }
  fmt.Println(s1)
  s1.setAge(41)
  s1.Print()
}
```

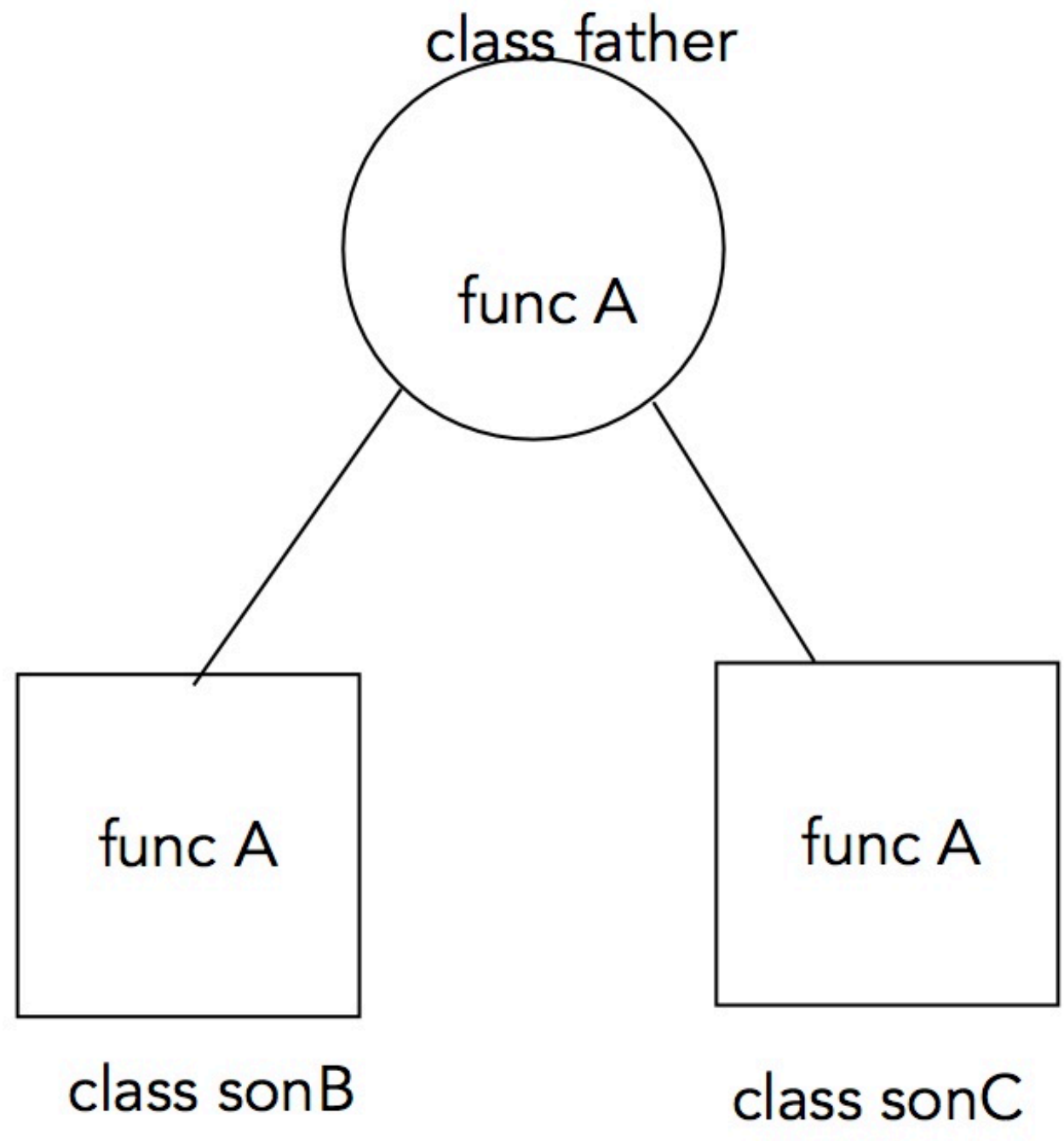
执行结果如下：

```
localhost: teacher$ go run 03-embed.go
{100 99 {hanli 40 man 5000}}
Name = hanli, Age = 41, Sex = man, Fight = 5000
strength = 100, fight = 99
```

Go语言当中，习惯给改种方式叫做内嵌（embed）

3. 接口

面向对象三要素：封装、继承、多态。我们已经介绍过了封装与继承，那么什么是多态呢?



Go语言同样支持多态，Go语言当中需要借助interface（接口）来实现。接下来我们定义一个动物接口，代码如下：

```
type Animal interface {
  Sleeping()
  Eating()
}
```

Animal接口定义了2个函数：Sleep和Eating，开发者如果想让Animal接口可以指向自己实现的对象，就必须支持这2个函数。比如我们定义一个小猫结构体和一个小狗结构体。

```
type Cat struct {
  Color string
}
```

```
type Dog struct {
  Color string
}
//小猫结构体的方法
func (c Cat) Sleeping() {
  fmt.Println(c.Color, "Cat is sleeping")
}
func (c Cat) Eating() {
  fmt.Println(c.Color, "Cat is Eating")
}

//小狗结构体的方法
func (c Dog) Sleeping() {
  fmt.Println(c.Color, "Dog is sleeping")
}
func (c Dog) Eating() {
  fmt.Println(c.Color, "Dog is Eating")
}
func (c Dog) Print() {
  fmt.Println("Dog's color is", c.Color)
}
```

我们再来实现一个工厂的方法，去构造一个接口对象，这个对象就可以动态指向Cat或者Dog了。

```
func Factory(color string, animal string) Animal {
  switch animal {
  case "dog":
    return &Dog{color}
  case "cat":
    return &Cat{color}
  default:
    return nil
  }
}

func main() {
  d1 := Factory("black", "dog")
  d1.Eating()
  c2 := Factory("white", "cat")
  c2.Sleeping()
}
```

执行结果如下：

```
localhost: teacher$ go run 04-factory.go
black Dog is Eating
white Cat is sleeping
```

五、拓展点

> 说明：
>
> 1. 典型面试题、笔试题；
> 2. 新技术 or 经验分享；
> 3. 未来计划、行业趋势分享；

六、总结

> 说明：
>
> 1. 回顾本堂课所有知识点；
> 2. 提示`注意点`和`重点`；
> 3. 提示学习方法；
> 4. 提示哪些需要记、哪些需要背、哪些代码需要敲；

- Go语言面向对象太简洁
- 方法是针对对象而言
- Go语言无需记忆面向对象的各种原则
- interface后面还会存在

七、大作业

> 说明：
>
> 1. 给出明确的作业要求，形成文字、图片或测试题等形式；
> 2. 给出明确的解答方式；
> 3. 频次低，可设计为：1次/周 或 1次/2周；

可从下面的检测题题库任意选择。

八、集中答疑

> 说明：完成本堂课所有知识点的讲解后，由学员集中提问，讲师逐一解答；

九、检测题

> 说明：
>
> 1. 针对本堂课，设计5-10个题，由学员课下完成（课下刷题）；
> 2. 出题范围可从检查点里挑选，可以是面试题或笔试题；
> 3. 题型要求是 单选、多选、判断、填空中的一种或多种；
> 4. 频次高，原则上每次课都要有检测题；

十、下节课预告

> 说明：
>
> 1. Go语言面向对象编程；
> 2. Go语言的interface；