# Day6 数据库&接口编程

tony

## 目录

## MYSQL开发

1. sqlx库介绍与使用

   A. 使用更简单

   B. 支持多数据库，mysql、postgresql、oracle、sqlite

## MYSQL开发

2. sqlx库介绍与使用

A. 查询， sqlx.DB.Get和sqlx.DB.Select

B. 更新、插入和删除，sqlx.DB.Exec

C. 事务，sqlx.DB.Begin()、sqlx.DB.Commit、sqlx.DB.rollback

## MYSQL开发

### 3. sql注入分析

A. Select *from user where name = ‘%s’，构造name=“1 ' or 1 = 1 or ‘”

B. 构造name=123' and (select count(*) from user ) > 10#

C. 构造name=123' union select *from user #

***避免手动拼接sql， 使用占位符或预处理！***

Redis使用和开发

4. Redis开发

A. 使用第三方的redis库， github.com/garyburd/redigo/redis

B. 连接redis

```
package main
import (
 "fmt"
 "github.com/garyburd/redigo/redis"
)
func main() {
 c, err := redis.Dial("tcp", "localhost:6379")
 if err != nil {
 fmt.Println("conn redis failed,", err)
 return
 }
 defer c.Close()
}
```

## 5. Redis开发

### A. Set操作， 设置key-value

```
package main
import (
 "fmt"
 "github.com/garyburd/redigo/redis"
)
func main() {
 c, err := redis.Dial("tcp", "localhost:6379")
 if err != nil {
 fmt.Println("conn redis failed,", err)
 return
 }
 defer c.Close()
 _, err = c.Do("Set", "abc", 100)
 if err != nil {
 fmt.Println(err)
 return
 }
 r, err := redis.Int(c.Do("Get", "abc"))
 if err != nil {
 fmt.Println("get abc failed,", err)
 return
 }
 fmt.Println(r)
}
```

## 6. Redis开发

### A. Hash表操作

```
package main
import (
 "fmt"
 "github.com/garyburd/redigo/redis"
)
func main() {
 c, err := redis.Dial("tcp", "localhost:6379")
 if err != nil {
 fmt.Println("conn redis failed,", err)
 return
 }
 defer c.Close()
 _, err = c.Do("HSet", "books", "abc", 100)
 if err != nil {
 fmt.Println(err)
 return
 }
 r, err := redis.Int(c.Do("HGet", "books", "abc"))
 if err != nil {
 fmt.Println("get abc failed,", err)
 return
 }
 fmt.Println(r)
}
```

Redis使用和开发

## 7. Redis开发

### A. Hash表操作

```
package main
import (
 "fmt"
 "github.com/garyburd/redigo/redis"
)
func main() {
 c, err := redis.Dial("tcp", "localhost:6379")
 if err != nil {
 fmt.Println("conn redis failed,", err)
 return
 }
 defer c.Close()
 _, err = c.Do("HSet", "books", "abc", 100)
 if err != nil {
 fmt.Println(err)
 return
 }
 r, err := redis.Int(c.Do("HGet", "books", "abc"))
 if err != nil {
 fmt.Println("get abc failed,", err)
 return
 }
 fmt.Println(r)
}
```

Redis使用和开发

## 8. Redis开发

### A. Mset操作

```
package main
import (
 "fmt"
 "github.com/garyburd/redigo/redis"
)
func main() {
 c, err := redis.Dial("tcp", "localhost:6379")
 if err != nil {
 fmt.Println("conn redis failed,", err)
 return
 }
 defer c.Close()
 _, err = c.Do("MSet", "abc", 100, "efg", 300)
 if err != nil {
 fmt.Println(err)
 return
 }
 r, err := redis.Ints(c.Do("MGet", "abc", "efg"))
 if err != nil {
 fmt.Println("get abc failed,", err)
 return
 }
 for _, v := range r {
 fmt.Println(v)
 }
}
```

## 9. Redis开发

### A. 设置过期时间

```
package main
import (
 "fmt"
 "github.com/garyburd/redigo/redis"
)
func main() {
 c, err := redis.Dial("tcp", "localhost:6379")
 if err != nil {
 fmt.Println("conn redis failed,", err)
 return
 }
 defer c.Close()
 _, err = c.Do("expire", "abc", 10)
 if err != nil {
 fmt.Println(err)
 return
 }
}
```

## 10. Redis开发

### A. 队列操作

```
package main
import (
 "fmt"
 "github.com/garyburd/redigo/redis"
)
func main() {
 c, err := redis.Dial("tcp", "localhost:6379")
 if err != nil {
 fmt.Println("conn redis failed,", err)
 return
 }
 defer c.Close()
 _, err = c.Do("lpush", "book_list", "abc", "ceg", 300)
 if err != nil {
 fmt.Println(err)
 return
 }
 r, err := redis.String(c.Do("lpop", "book_list"))
 if err != nil {
 fmt.Println("get abc failed,", err)
 return
 }
 fmt.Println(r)
}
```

## 11. Redis开发

### A. Redis连接池

```
//初始化一个pool
func newPool(server, password string) *redis.Pool {
    return &redis.Pool{
        MaxIdle:     64,
        MaxActive:   1000,
        IdleTimeout: 240 * time.Second,
        Dial: func() (redis.Conn, error) {
            c, err := redis.Dial("tcp", server)
            if err != nil {
                return nil, err
                }
                /*
            if _, err := c.Do("AUTH", password); err != nil {
                c.Close()
                return nil, err
            }*/
            return c, err
        },
        TestOnBorrow: func(c redis.Conn, t time.Time) error {
            if time.Since(t) < time.Minute {
                return nil
            }
            _, err := c.Do("PING")
            return err
        },
    }
}
```

## 接口介绍和定义

1. 接口定义了一个对象的行为规范

    A. 只定义规范，不实现

    B. 具体的对象需要实现规范的细节

## 接口介绍和定义

2. Go中接口定义

   A. type 接口名字 interface

   B. 接口里面是一组方法签名的集合

```go
type Animal interface {
    Talk()
    Eat() int
    Run()
}
```

## 接口介绍和定义

3. Go中接口的实现

   A. 一个对象只要包含接口中的方法，那么就实现了这个接口

   B. 接口类型的变量可以保存实现该接口的任何具体类型的实例

```go
type Animal interface {
    Talk()
    Eat() int
    Run()
}
```

## 接口介绍和定义

4. 接口实例

   A. 一个公司需要计算所有职员的薪水

   B. 每个职员的薪水计算方式不同

```
type Animal interface {
    Talk()
    Eat() int
    Run()
}
```

## 接口介绍和定义

5. 接口类型变量

   A. var a Animal

   B. 那么a能够存储所有实现Animal接口的对象实例

```
type Animal interface {
    Talk()
    Eat() int
    Run()
}
```

# 空接口和类型断言

6. 空接口

   A. 空接口没有定义任何方法

   B. 所以任何类型都实现了空接口

```
interface {

}
```

# 空接口和类型断言

7. 空接口

```go
package main

import (
    "fmt"
)

func describe(i interface{}) {
    fmt.Printf("Type = %T, value = %v\n", i, i)
}

func main() {
    s := "Hello World"
    describe(s)
    i := 55
    describe(i)
    strt := struct {
        name string
    }{
        name: "Naveen R",
    }
    describe(strt)
}
```

# 空接口和类型断言

8. 类型断言

A. 如何获取接口类型里面存储的具体的值呢?

```go
package main

import (
	"fmt"
)

func assert(i interface{}) {
	s := i.(int) //get the underlying int value from i
	fmt.Println(s)
}
func main() {
	var s interface{} = 56
	assert(s)
}
```

# 空接口和类型断言

## 9. 类型断言

### A. 类型断言的坑!

```go
package main

import (
    "fmt"
)

func assert(i interface{}) {
    s := i.(int)
    fmt.Println(s)
}
func main() {
    var s interface{} = "Steven Paul"
    assert(s)
}
```

# 空接口和类型断言

## 10. 类型断言

A. 如何解决，引入 ok判断机制!　　　　v, ok := i.(T)

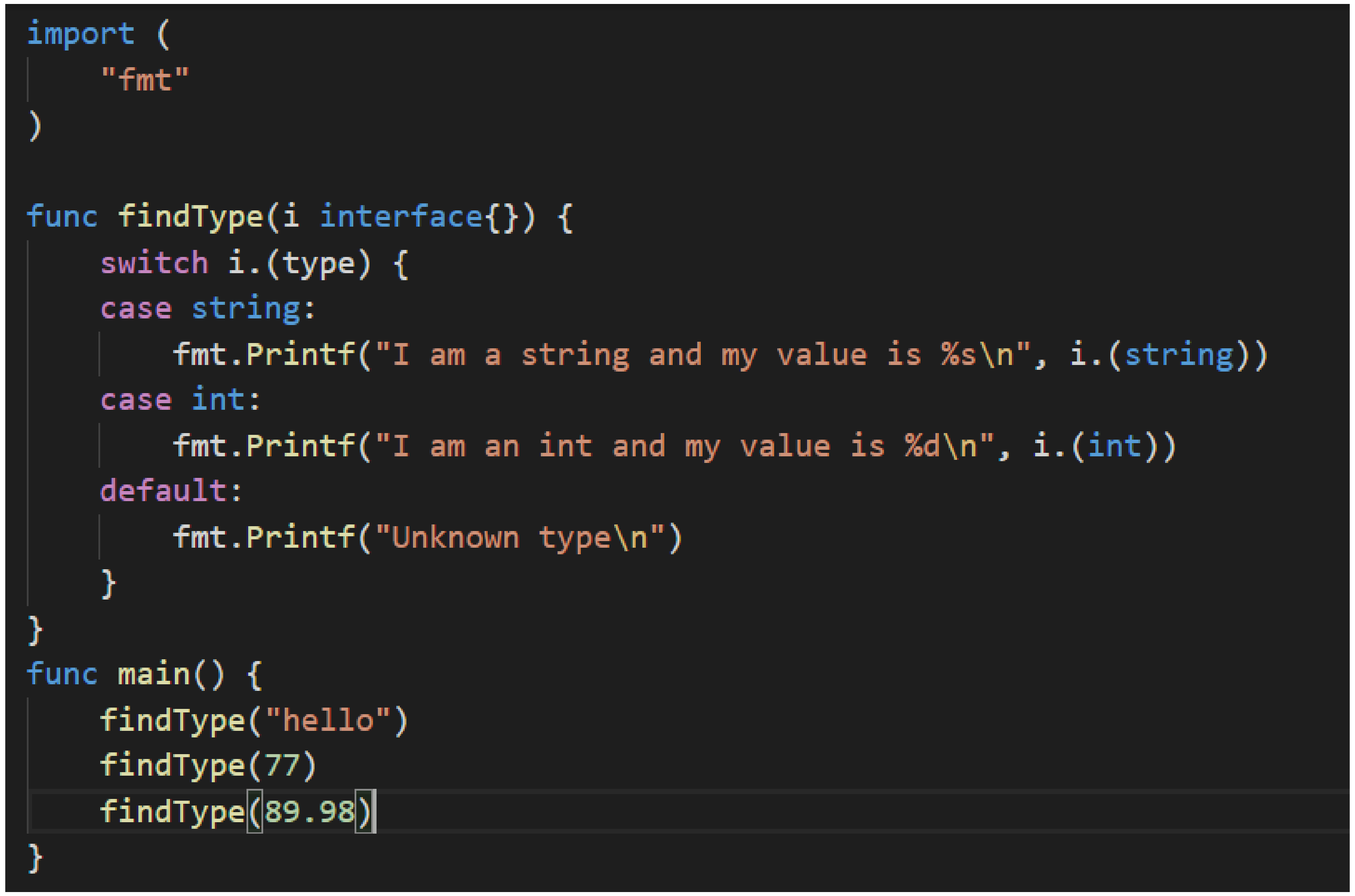
```go
package main

import (
    "fmt"
)

func assert(i interface{}) {
    v, ok := i.(int)
    fmt.Println(v, ok)
}
func main() {
    var s interface{} = 56
    assert(s)
    var i interface{} = "Steven Paul"
    assert(i)
}
```

# 空接口和类型断言

## 11. 类型断言

A. type switch。

**问题需要转两次?**

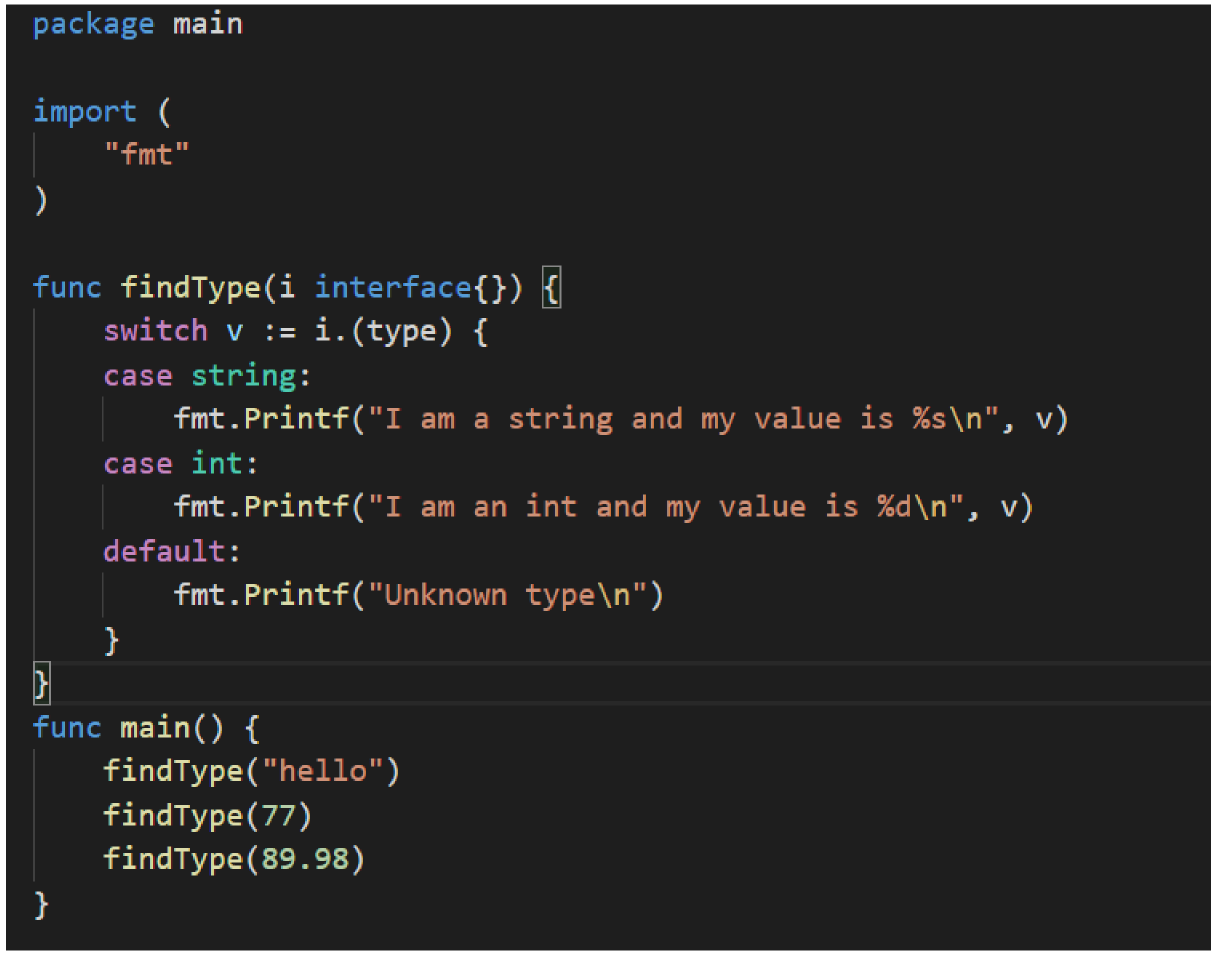


```go
import (
	"fmt"
)

func findType(i interface{}) {
	switch i.(type) {
	case string:
		fmt.Printf("I am a string and my value is %s\n", i.(string))
	case int:
		fmt.Printf("I am an int and my value is %d\n", i.(int))
	default:
		fmt.Printf("Unknown type\n")
	}
}
func main() {
	findType("hello")
	findType(77)
	findType(89.98)
}
```

## 空接口和类型断言

12. 类型断言

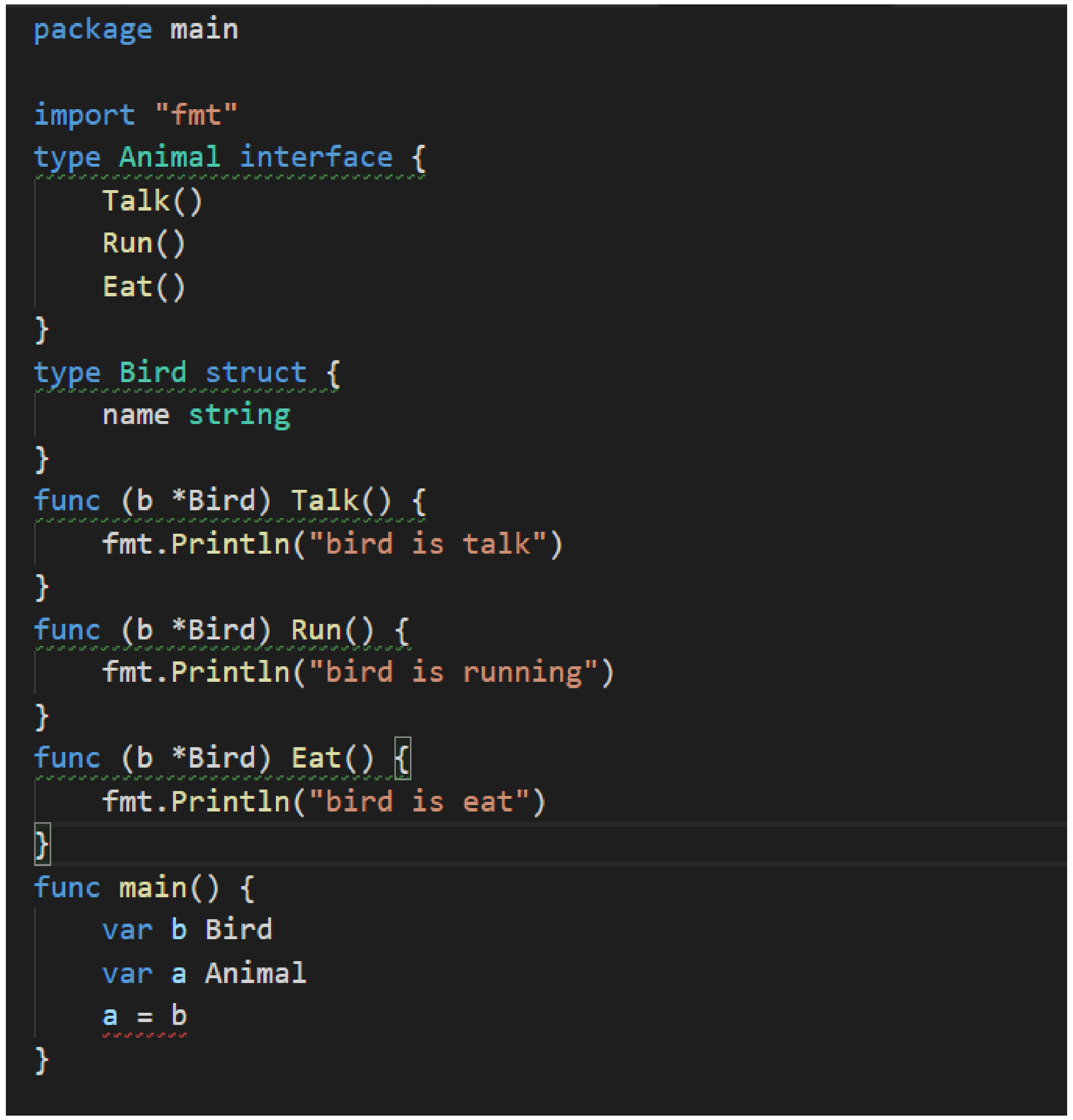
A. type switch另外一种写法，解决转两次的问题



```go
package main

import (
	"fmt"
)

func findType(i interface{}) {
	switch v := i.(type) {
	case string:
		fmt.Printf("I am a string and my value is %s\n", v)
	case int:
		fmt.Printf("I am an int and my value is %d\n", v)
	default:
		fmt.Printf("Unknown type\n")
	}
}
func main() {
	findType("hello")
	findType(77)
	findType(89.98)
}
```

## 指针接收和值接收

### 13. 指针接收



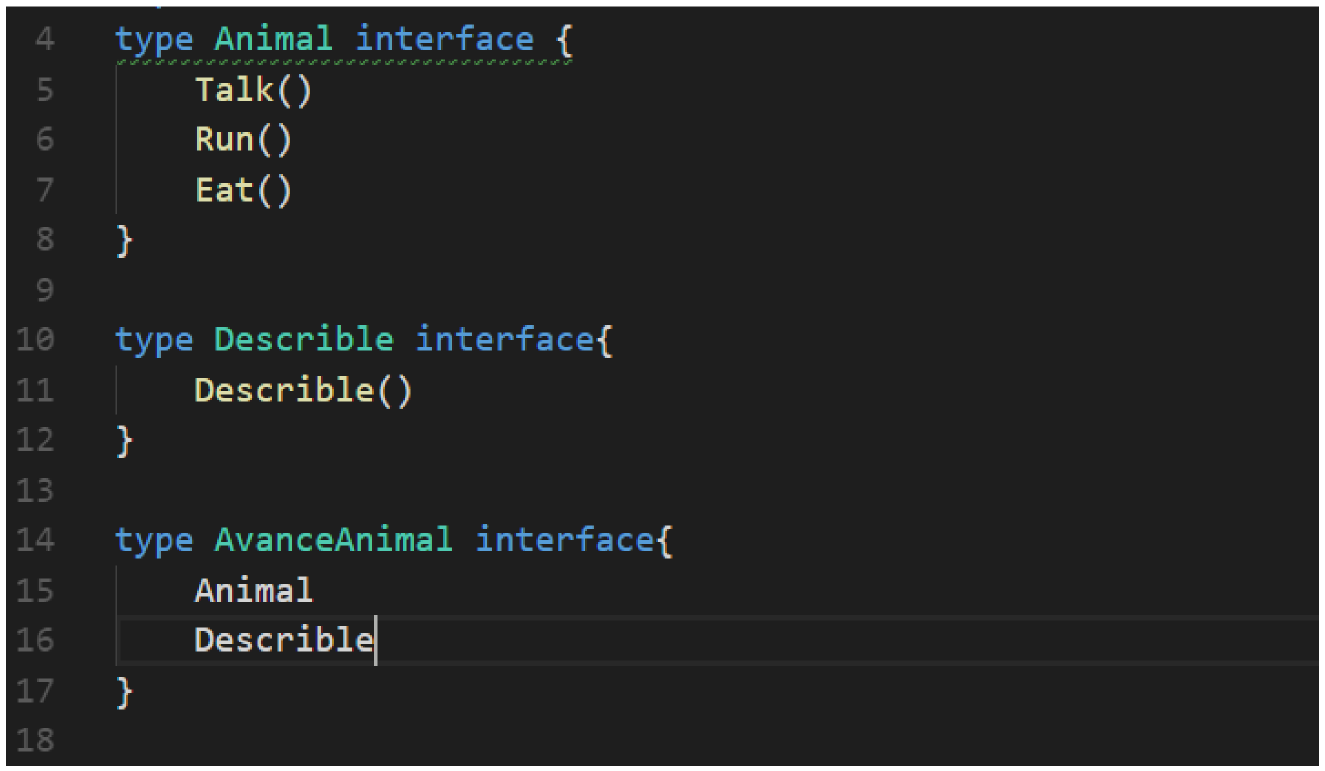
```go
package main

import "fmt"
type Animal interface {
    Talk()
    Run()
    Eat()
}
type Bird struct {
    name string
}
func (b *Bird) Talk() {
    fmt.Println("bird is talk")
}
func (b *Bird) Run() {
    fmt.Println("bird is running")
}
func (b *Bird) Eat() {
    fmt.Println("bird is eat")
}
func main() {
    var b Bird
    var a Animal
    a = b
}
```

## 实现多接口

14. 同一个类型可以实现多个接口

## 实现多接口

15. 接口嵌套，和结构体嵌套类似



```go
type Animal interface {
    Talk()
    Run()
    Eat()
}

type Describle interface{
    Describle()
}

type AvanceAnimal interface{
    Animal
    Describle
}
```

## 接口实例讲解

1. io包中的writer接口

```
package main

import (
    "fmt"
    "os"
)

func main() {
    var w Writer

    // os.Stdout 实现了 Writer
    w = os.Stdout

    fmt.Fprintf(w, "hello, writer\n")
}
```

## 接口实例讲解

2. fmt包中的Stringer接口

```go
type Stringer interface {
     String() string
}
```

```go
package main

import "fmt"

type Person struct {
    Name string
    Age  int
}

func (p Person) String() string {
    return fmt.Sprintf("%v (%v years)", p.Name,
p.Age)
}

func main() {
    a := Person{"Arthur Dent", 42}
fmt.Printf("Person 类型: %T\n", a)
    z := Person{"Zaphod Beeblebrox", 9001}
    fmt.Println(a,"|", z)
}
```

## 接口实例讲解

3. fmt包中的Stringer接口

```
package main

import "fmt"

type IPAddr [4]byte

// TODO: Add a "String() string" method to IPAddr.
func (ip IPAddr) String() string{
    return fmt.Sprintf("%v,%v.%v.%v", ip[0], ip[1], 
ip[2], ip[3])    //Sprintf:格式化返回数据
}

func main() {
    addrs := map[string]IPAddr{
        "loopback":  {127, 0, 0, 1},
        "googleDNS": {8, 8, 8, 8},
    }
    for n, a := range addrs {
        fmt.Printf("%v: %v\n", n, a)
    }
}
```

## 接口实例讲解

4. error包中的error接口

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

```
package main
import (
    "fmt"
    "time"
    "errors"
    "strconv"
)
type MyError struct {
    When time.Time
    What string
}
func (e MyError) Error() string {
    str := fmt.Sprintf("at %v, %s",e.When, e.What)
    fmt.Printf("1:%T\n", str)
    return str
}
func run() error{
    fmt.Println("0")
    str := MyError{time.Now(),"it didn't work",}
    fmt.Printf("2:%T\n", str)
    fmt.Println(MyError{time.Now(),"it didn't work",})
    return str
}
func main() {
    if err := run(); err != nil {
        fmt.Printf("3:%T\n", err)
        fmt.Println(err)
    }
}
```

## 接口实例讲解

5. error包中的error接口

```
package main

import (
    "fmt"
    "math"
)
type ErrNegativeSqrt float64

func (e ErrNegativeSqrt) Error() string{
    return fmt.Sprintf("cannot Sqrt negative number:%v", float64(e))
}

func Sqrt(x float64) (float64, error) {
    if x < 0 {
        return 0, ErrNegativeSqrt(x)
    }
    //牛顿法求二次根
    z := float64(1)
     for {
          y := z - (z*z-x)/(2*z)
          if math.Abs(y-z) < 1e-10 {
               return y, nil
          }
          z = y
     }
     return z, nil
}

func main() {
    fmt.Println(Sqrt(2))
    fmt.Println(Sqrt(-2))
}
```

## 接口实例讲解

### 6. Reader接口

```
package main

import (
    "golang.org/x/tour/reader"
    "time"
    "fmt"
    _"strings"
)

type MyReader struct{}

// TODO: Add a Read([]byte) (int, error) method to MyReader.
func (r MyReader) Read(b []byte) (int, error){
    b[0] = 'A'
    return 1, nil
}

func main() {

    var myre MyReader
    b := make([]byte, 1)
    //for{
        //r := strings.NewReader(b)
        myre.Read(b)
        fmt.Printf("%c\n", b[0])
        time.Sleep(1 *time.Second)
        myre.Read(b)
        fmt.Println(b[0])
    //}

}
```

## 接口实例讲解

7. Image接口

```
type Image interface {
ColorModel() color.Model
Bounds() Rectangle
At(x, y int) color.Color
}
```

## 接口实例讲解

### 7. Image接口练习

```go
package main

import (
    "golang.org/x/tour/pic"
    "image"
    "image/color"
    //"fmt"
)
type Image struct{
    weight int
    height int

}

func (c Image) ColorModel() color.Model{
    return color.RGBAModel
}

func (b *Image) Bounds() image.Rectangle{
    return image.Rect(0, 0, b.weight, b.height)
}

func (a *Image) At(x, y int) color.Color{
    //fmt.Println(x, y)
    return color.RGBA{uint8(x), uint8(y), 255, 255}
}

func main() {
    m := &Image{700,50}
    //m.At(225, 0)
    pic.ShowImage(m) //m.At(x, y)的参数由pic传入，传入了所有情况
}
```

## 课后工作

1. 实现一个图书管理系统v2，具有以下功能：

    a. 增加用户登录、注册功能

    b. 增加借书过期的图书界面

    c. 增加显示热门图书的功能，被借次数最多的top10

    d. 增加查看某个人的借书记录的功能