课程安排，请关注微信公众平台或者官方微博

编程语言： Golang 与 html5

编程工具： Goland 和 HBuilder

预计平均一周左右更新一或二节课程

授人以鱼，不如授人以渔业。

大家好，
欢迎来到 **字节教育** 课程的学习
社区论坛网址：**www.Golang.Ltd**
技术交流群 ： 221273219
微信公众号 ：**Golang 语言社区**
微信服务号 ：**Golang 技术社区**

第一季 Go 语言基础、进阶、提高课程

第二十二节 深入理解 interface

Go 不是一种典型的面向对象编程的语言，它在语法上不支持类和继承的概念。

没有继承是否就无法拥有多态行为了呢？答案是否定的，Go 语言引入了一种新类型—Interface，它在效果上实现了类似于 C++的"多态"概念，虽然与 C++的多态在语法上并非完全对等，但至少在最终实现的效果上，它有多态的影子。那么，Go 的 Interface 类型到底是什么呢？怎么使用呢？这正是本节说明的问题。

1、Method(s) in Go

在说明 Interface 类型前，不得不先用 Go 的 method(s)概念来热身，因为 Go 语言的 interface 与 method(s)这两个语法有非常紧密的联系。虽然 Go 语言没有类的概念，但它支持的数据类型可以定义对应的 method(s)。**本质上说，所谓的 method(s)其实就是函数，只不过与普通函数相比，这类函数是作用在某个数据类型上的**，所以在函数签名中，会有个 receiver 来表明当前定义的函数会作用在该 receiver 上。关于 methods 的精确语法规范，可以参考 **language specification** 或 **Effective Go** 中的说明，这里略过。注意：Go 语言支持的除 Interface 类型外的任何其它数据类型都可以定义其 method（而并非只有 struct 才支持 method），只不过实际项目中，method(s)多定义在 struct 上而已。在 struct 类型上定义 method(s)的语法特性与 C++中的 struct 支持的语法非常类似（c++中的 struct 定义了数据，此外也支持定义数据的操作方法），**从这一点来看，我们可以把 Go 中的 struct 看作是**

不支持继承行为的轻量级的“类”。

2、 what is Interface tyce in Go？

GoLang 官网 **language specification 文档**对 interface type 的概念说明如下：

An interface type specifies a method set called its interface. A variable of interface type **can store a value of any type** with a method set that is any superset of the interface. Such a type is said to **implement** the interface. The value of an uninitialized variable of interface type is nil.

说实话，这段说明对新手来说比较晦涩，这正是本节试图解释清楚的地方。

从语法上看，Interface 定义了一个或一组 method(s)，这些 method(s)只有函数签名，没有具体的实现代码（有没有联想起 C++中的虚函数？）。若某个数据类型实现了 Interface 中定义的那些被称为"methods"的函数，则称这些数据类型实现（implement）了 interface。举个例子来说明。

```
package main

import (

    "fmt"

    "math"

)

type Abser interface {

    Abs() float64

}

type MyFloat float64

func (f MyFloat) Abs() float64 {

    if f < 0 {

        return float64(-f)

    }

    return float64(f)

}

func main() {

    var a Abser

    f := MyFloat(-math.Sqrt2)

    a = f  // a MyFloat implements Abser

    fmt.Println(a.Abs())

}
```

上面的代码中，第 8-10 行是通过 type 语法声明了一个名为 Abser 的 interface 类型（Go 中约定的 interface

类型名通常取其内部声明的 method 名的 er 形式）。而第 12-19 行通过 type 语法声明了 MyFloat 类型且为该类型定义了名为 Abs()的 method。根据上面的解释，Abs()是 interface 类型 Abser 定义的方法，而 MyFloat 实现了该方法，所以，MyFloat 实现了 Abser 接口。

Interface 类型的更通用定义可归纳如下：

```
type Namer interface {

    Method1(param_list) return_type

    Method2(param_list) return_type

    ...

}
```

上面的示例用 type 语法声明了一个名为 Namer 的 interface 类型（但 Namer 不是个具体的变量，此时内存中还没有它对应的对象）。interface 类型是可以定义变量的，也即 interface type can have values，例如：

```
var ai Namer
```

此时，定义了一个变量名为 ai 的 Namer 类型变量，**在 Go 的底层实现中，ai 本质上是个指针，其内存布局如下**（内存布局图引用自<The Way to Go - A Thorough Introduction to the Go Programming Language>一书第 11.1 节）：

它的 method table ptr 是不是与 C++中类的虚函数表非常类似？**而这正是 interface 类型的变量具有多态特性的关键：**

ai 共占 2 个机器字，1 个为 receiver 字段，1 个为 method table ptr 字段。ai 可以被赋值为任何变量，只要这个变量实现了 interface 定义的 method(s) set，赋值后，ai 的 receiver 字段用来 hold 那个变量或变量副本的地址（若变量类型小于等于 1 个机器字大小，则 receiver 直接存储那个变量；若变量类型大于 1 个机器字，则 Go 底层会在堆上申请空间存储那个变量的副本，然后 receiver 存储那个副本的地址，即此时 receiver 是个指向变量副本的指针）。而由变量实现的接口 method(s)组成的 interface table 的指针会填充到 ai 的 method table ptr 字段。当 ai 被赋值为另一个变量后，其 receiver 和 method table ptr 会更新为新变量的相关值。

关于 interface 类型内部实现细节，可以参考 GoLang 官网 Blog 推荐过的一篇文章"**Go Data Structures: Interfaces**"，写的很清楚，强烈推荐。

所以，如果某个函数的入参是个 interface 类型时，任何实现了该 interface 的变量均可以作为合法参数传入且函数的具体行为会自动作用在传入的这个实现了 interface 的变量上，这不正是类似于 C++中多态的行为吗？

这正是 Interface 类型在 Go 语言中的威力。

引用<The Way to Go>一书第 11.5 节对 interface 类型的总结如下，值得每个 Go 学习者理解：

An interface is a kind of contract which the implementing type(s) must fulfill. Interfaces describe the behaviorof types, what they can do. They completely separate the definition of what an object can do from how it does it, allowing distinct implementations to be represented at different times by the same interface variable, **which is what polymorphism essentially is**.

Writing functions so that they accept an interface variable as a parameter makes them more general.

3、Interface “多态”特性实例

Go 语言自带的标准 Packages 提供的接口很多都借助了 Interface 以具备"可以处理任何未知数据类型"的能力。例如被广泛使用的 **fmt** 包，其功能描述如下：

Package fmt implements formatted I/O with functions analogous to C's printf and scanf. The format 'verbs' are derived from C's but are simpler.

它除了可以格式化打印 Go 的 built-in 类型外，还可以正确打印各种自定义类型，只要这些自定义数据类型实现了 fmt 的 Print API 入参所需的 interface 接口。

以 fmt 包的 Printf()函数为例，其函数签名格式如下：

```
func Printf(format string, a ...interface{}) (n int, err error)
```

它的入参除了用以描述如何格式化的'format'参数外，还需要 interface 类型的可变长参数。该函数在实现底层的打印行为时，要求传入的可变长参数实现了 fmt 包中定义的 Stringer 接口，这个接口类型定义及描述如下：

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

Stringer is implemented by any value that has a String method, which defines the "native" format for that value. The String method is used to print values passed as an operand to any format that accepts a string or to an unformatted printer such as Print.

所以，自定义类型想要调用 fmt.Printf()做格式化打印，那只需实现 Stringer 接口就行。

例如，下面是一段简单的打印代码：

```
package main

import "fmt"

type IPAddr [4]byte

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)

    }

}
```

其输出如下：

```
loopback: [127 0 0 1]

googleDNS: [8 8 8 8]
```

现在要求按规定的格式打印：IPAddr{1, 2, 3, 4}应该输出为"1.2.3.4"的格式，所以 IPAddr 这个自定义类型需要实现 Stringer 接口，实现代码如下：

```
// exercise-stringer.go

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])

}

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)

    }

}
```

执行结果符合需求：

```
googleDNS: 8.8.8.8

loopback: 127.0.0.1
```

4、interface 在游戏服务器开发中的应用采坑例子（网上整理）

GoWorld 作者开发框架的时候，碰到的一个实际的 bug。由于 GoWorld 支持多种不同的数据库（包括 MongoDB，Redis 等）来保存服务端对象，因此 GoWorld 在上层提供了一个统一的对象存储接口定义，而不同的对象数据库实现只需要实现 `EntityStorage` 接口所提供的函数即可。

```
// EntityStorage defines the interface of entity storage backends

type EntityStorage interface {

    List(typeName string) ([]common.EntityID, error)

    Write(typeName string, entityID common.EntityID, data interface{}) error

    Read(typeName string, entityID common.EntityID) (interface{}, error)

    Exists(typeName string, entityID common.EntityID) (bool, error)

    Close()

    IsEOF(err error) bool

}
```

以一个使用 Redis 作为对象数据库的实现为例，函数 `OpenRedis` 连接 Redis 数据库并最终返回一个 `redisEntityStorage` 对象的指针。

```
// OpenRedis opens redis as entity storage

func OpenRedis(url string, dbindex int) *redisEntityStorage {

    c, err := redis.DialURL(url)

    if err != nil {

        return nil

    }

    if dbindex >= 0 {

        if _, err := c.Do("SELECT", dbindex); err != nil {

            return nil

        }

    }

    es := &redisEntityStorage{

        c: c,

    }

    return es

}
```

在上层逻辑中，我们使用 `OpenRedis` 函数连接 Redis 数据库，并将返回的 `redisEntityStorage` 指针赋值个一个 `EntityStorage` 接口变量，因为 `redisEntityStorage` 对象实现了 `EntityStorage` 接口所定义的所有函数。

```
var storageEngine StorageEngine // 这是一个全局变量

storageEngine = OpenRedis(cfg.Url, dbindex)

if storageEngine != nil {

    // 连接成功

    ...

} else {

    // 连接失败

    ...

}
```

上面的代码看起来都很正常，`OpenRedis` 在连接 Redis 数据库失败的时候会返回 nil，然后调用者将返回值和 nil 进行比较，来判断是否连接成功。**这个就是 Go 语言少有的几个深坑之一，因为不管 `OpenRedis` 函数是否连接 Redis 成功，都会运行连接成功的逻辑。**

寻找问题所在

想要理解这个问题，首先需要理解 interface{}变量的本质。在 Go 语言中，一个 interface{}类型的变量包含了 2 个指针，一个指针指向值的类型，另外一个指针指向实际的值。 我们可以用如下的测试代码进行验证。

```
// InterfaceStructure 定义了一个 interface{}的内部结构

type InterfaceStructure struct {

    pt uintptr // 到值类型的指针

    pv uintptr // 到值内容的指针

}

// asInterfaceStructure 将一个 interface{}转换为 InterfaceStructure

func asInterfaceStructure (i interface{}) InterfaceStructure {

    return *(*InterfaceStructure)(unsafe.Pointer(&i))

}

func TestInterfaceStructure(t *testing.T) {

    var i1, i2 interface{}

    var v1 int = 0x0AAAAAAAAAAAAAA

    var v2 int = 0x0BBBBBBBBBBBBBB

    i1 = v1

    i2 = v2
```

```
    fmt.Printf("sizeof interface{} = %d\n", unsafe.Sizeof(i1))
    fmt.Printf("i1 %x %+v\n", i1, asInterfaceStructure(i1))
    fmt.Printf("i2 %x %+v\n", i2, asInterfaceStructure(i2))
    var nilInterface interface{}
    fmt.Printf("nil interface = %+v\n", asInterfaceStructure(nilInterface))
}
```

这段代码的输出如下：

```
sizeof interface{} = 16
i1 aaaaaaaaaaaaaaa {pt:5328736 pv:825741282816}
i2 bbbbbbbbbbbbbbb {pt:5328736 pv:825741282824}
nil interface = {pt:0 pv:0}
```

所以对于一个 interface{}类型的 nil 变量来说，它的两个指针都是 0。这是符合 Go 语言对 nil 的标准定义的。在 Go 语言中，nil 是零值（Zero Value），而在 Java 之类的语言里，null 实际上是空指针。关于零值和空指针有什么区别，这里就不再展开了。

当我们将一个具体类型的值赋值给一个 interface 类型的变量的时候，就同时把类型和值都赋值给了 interface 里的两个指针。如果这个具体类型的值是 nil 的话，interface 变量依然会存储对应的类型指针和值指针。

```
func TestAssignInterfaceNil(t *testing.T) {
    var p *int = nil
    var i interface{} = p
    fmt.Printf("%v %+v is nil %v\n", i, asInterfaceStructure(i), i == nil)
}
```

输入如下：

```
<nil> {pt:5300576 pv:0} is nil false    var i interface{} = p
```

可见，在这种情况下，虽然我们把一个 nil 值赋值给 interface{}，但是实际上 interface 里依然存了指向类型的指针，所以拿这个 interface 变量去和 nil 常量进行比较的话就会返回 `false`。

如何解决这个问题

想要避开这个 Go 语言的坑，我们要做的就是避免将一个有可能为 nil 的具体类型的值赋值给 interface 变量。以上述的 `OpenRedis` 为例，一种方法是先对 `OpenRedis` 返回的结果进行非-nil 检查，然后再赋值给 interface 变量，如下所示。

```
var storageEngine StorageEngine // 这是一个全局变量
redis := OpenRedis(cfg.Url, dbindex)
```

```
if redis != nil {
    // 连接成功
    storageEngine = redis // 确定 redis 不是 nil 之后再赋值给 interface 变量
} else {
    // 连接失败
    ...
}
```

另外一种方法是让 `OpenRedis` 函数直接返回 EntityStorage 接口类型的值，这样就可以把 `OpenRedis` 的返回值直接正确赋值给 EntityStorage 接口变量。

```
// OpenRedis opens redis as entity storage
func OpenRedis(url string, dbindex int) EntityStorage {
    c, err := redis.DialURL(url)
    if err != nil {
        return nil
    }
    if dbindex >= 0 {
        if _, err := c.Do("SELECT", dbindex); err != nil {
            return nil
        }
    }
    es := &redisEntityStorage{
        c: c,
    }
    return es
}
```

5、interface 实现游戏服务器开发过程中第三方包是如何处理存、取数据的

Go 语言 map 并发为安全的，推荐大家使用 go-concurrentMap ，社区的 github 也有相应的包

分析 interface 在 go-concurrentMap 中的使用 代码如下：

<1> 使用 put 成员函数

```
/**
 * Maps the specified key to the specified value in this table.
 * Neither the key nor the value can be nil.
 *
 * The value can be retrieved by calling the get method
 * with a key that is equal to the original key.
 *
 * @param key with which the specified value is to be associated
 * @param value to be associated with the specified key
 *
 * @return the previous value associated with key, or
 *         nil if there was no mapping for key
 */
func (this *ConcurrentMap) Put(key interface{}, value interface{}) (oldVal interface{}, err error) {
    if isNil(key) {
        return nil, NilKeyError
    }
    if isNil(value) {
        return nil, NilValueError
    }

    if hash, e := hashKey(key, this, false); e != nil {
        err = e
    } else {
        Printf("Put, %v, %v\n", key, hash)
        oldVal = this.segmentFor(hash).put(key, hash, value, false, nil)
    }
    //hash := hash2(hashKey(key, this, true))
    //Printf("Put, %v, %v\n", key, hash)
    //oldVal = this.segmentFor(hash).put(key, hash, value, false)
    return
}
```

<2>使用 get 成员函数

```
/**
 * Returns the value to which the specified key is mapped,
 * or nil if this map contains no mapping for the key.
 */
func (this *ConcurrentMap) Get(key interface{}) (value interface{}, err error) {
    if isNil(key) {
        return nil, NilKeyError
    }
    //if atomic.LoadPointer(&this.kind) == nil {
    //  return nil, nil
    //}
    if hash, e := hashKey(key, this, false); e != nil {
        err = e
    } else {
        Printf("Get, %v, %v\n", key, hash)
        value = this.segmentFor(hash).get(key, hash)
    }
    return
}
```

<3> 使用

① 初始化

M = concurrent.NewConcurrentMap()

② Put

M.Put("UPDATE", "0%")

③ Get

```
val, _ := M.Get("UPDATE")
```

④ 总结

```
func DuLiGX(w http.ResponseWriter, req *http.Request) {

    //  http://local.websocket.club:8765/BenGX?GX=&IType=
    // IType =1 更新  IType =2 获取进度  IType =3  更新进度
    if req.Method == "GET" {
        w.Header().Set("Access-Control-Allow-Origin", "*")
        req.ParseForm()
        // 获取函数
        StrBF, bGX := req.Form["GX"]
        StrIType, bIType := req.Form["IType"]
        if bGX && bIType {
            // 启动更新程序，
            if StrIType[0] == "1" {
                CallEXELP(StrBF[0])
                fmt.Fprint(w, "成功！！！")
                //BF = "0%"
                M.Put("UPDATE", "0%")
                return
            } else if StrIType[0] == "2" {
                // 返回web数据：已经更新文件，需要更新文件总数。
                val, _ := M.Get("UPDATE")
                // glog.Info("                ----update:", val)
                if val != nil {
                    fmt.Fprint(w, val)
                    return
                }
                fmt.Fprint(w, "0%")
                return
            } else if StrIType[0] == "3" {
                // 返回web数据：已经更新文件，需要更新文件总数。
                //BF = StrBF[0]
                M.Put("UPDATE", StrBF[0])
                //fmt.Println("返回web数据：已经更新文件，需要更新文件总数", StrBF[0])
```

6、interface 实现游戏服务器开发过程中自定义数据结构 如何处理存、取

<1> 定义：

```
LunBoMap := make(map[string]interface{})
```

<2> HTTP 服务器返回数据：

```
} else if strType[0] == "6" { // 获取账号下全部游戏  http://[redacted]DuliServer?Type=&LoginName=&LoginPW=
    // 1000: 账号不存在  1001: 密码错误
    strLoginName, bLoginName := req.Form["LoginName"]
    strLoginPW, bLoginPW := req.Form["LoginPW"]
    if bLoginName && bLoginPW {
        //1  首先验证用户名存在不
        brdt := dbif.ReadDuliServerYanZhengPayInfo(strLoginName[0], strLoginPW[0])
        if !brdt {
            fmt.Fprint(w, base64.StdEncoding.EncodeToString([]byte("1000")))
            return
        }
        //2  验证游戏存在不，时间到期没有
        data, _ := dbif.InitPayGameInfobak(strLoginName[0])
        b, _ := json.Marshal(data)
        // 获取卡卷基础信息
        fmt.Fprint(w, base64.StdEncoding.EncodeToString(b))
        return
    }

    fmt.Fprint(w, "获取失败！！！")
    return
}
```

```go
func InitPayGameInfobak(LoginName string) (mapGameInfo map[string]*Global_Define.StXianChangInfo, size int64) {

    if len(LoginName) == 0 {
        glog.Info("InitPayGameInfo Of LoginName is nil!!!")
        return nil, -1
    }
    mapGameInfo = make(map[string]*Global_Define.StXianChangInfo)
    strSql := "select * from t_PayGameInfo where SJName = " + "\"" + LoginName + "\""
    Rows, err := GetMySQL().Query(strSql)
    defer Rows.Close()
    if err != nil {
        glog.Info("InitPayGameInfo err:", err.Error())
        return nil, -1
    }
    var iTmp int64
    iTmp = 0
    for Rows.Next() {
        iTmp++
        gameinfo := new(Global_Define.StXianChangInfo)
        Rows.Scan(&gameinfo.ID, &gameinfo.SJName, &gameinfo.XianChangName, &gameinfo.GameID, &gameinfo.GameName, &gameinfo.GameTime, &gameinfo.InsertTime, &gameinfo.Firm
        //gameinfo.GameName = Global_Define.GBKConvertUTF8(gameinfo.GameName)
        mapGameInfo[strconv.Itoa(int(iTmp))] = gameinfo
    }
    return mapGameInfo, iTmp
}
```

```go
// 商家游戏现场的数据的结构
type StXianChangInfo struct {
    ID              uint32
    SJName          string
    XianChangName   string // 现场名字
    GameID          uint32 // 游戏的ID
    GameName        string // 游戏的名字
    GameTime        string // 游戏的结束时间
    InsertTime      string // 插入的时间
    FirmsLogo       string // 商家的LOGO
    FirmsErCode     string // 商家的二维码
    FirmsName       string // 商家的名字
    FirmsAward      string // 商品列表
    FirmsPublicize string // 宣传语列表
    //FirmsAward     map[string]*StAwardInfo      // 商品列表
    //FirmsPublicize map[string]*StPublicizeInfo // 宣传语列表
    ResPath    string // 生成二维码的URL
    IpAndPort string //ip 端口
}
```

<3> 存数据：

```
        url := "http://" + url1 + "&Type=6&LoginName=" + strLoginName[0] + "&LoginPW=" + strLoginPW[0]
        resp, err1 := http.Get(url)
        if err1 != nil {
            glog.Error("Send_FirmGetAward_JiLu_By_Hsttp get error", err1.Error())
            return
        }
        body, err := ioutil.ReadAll(resp.Body)
        if err != nil {
            glog.Error("body err:", err.Error())
            return
        }
        fmt.Fprint(w, string(body))
        // 写文件
        WriteFile(strLoginName[0])
        BShutGame = 0
        BShutGamebak = 0
        BShutGamebakre = 0
        IsPlayer = ""
        // 重置配置缓存
        if IniName != strLoginName[0] {
            IniName = strLoginName[0]
            // 关闭现在运行游戏
            KillEXE(StrGameName)
        }
        // 保存数据到内存，作为轮播游戏的数据
        enbyte, err := base64Decode(body)
        if err != nil {
            fmt.Println(err.Error())
        } else {
            //fmt.Println("                              ", string(enbyte))
            var r Requestbody
            r.req = string(enbyte)
            if req2map, err := r.Json2map(); err == nil {
                //fmt.Println(req2map["1"])
                //fmt.Println("-----------------------------", (req2map["1"].(map[string]interface{}))["XianChangName"])

                //------------------------------------------------------------
                // 保存数据
                LunBoMap = req2map
                //      for k, v := range req2map {
                //          fmt.Printf("%s %s\n", k, v)
                //      }
                //------------------------------------------------------------
            } else {
                fmt.Println("登陆数据解析错误@@@", err)
            }

        }
```

<4> 取数据：

```
//----------------------
for k, _ := range LunBoMap {
    iGameID, _ := strconv.Atoi(GameID)

    if int((LunBoMap[k].(map[string]interface{})["GameID"]).(float64)) == iGameID {
        // 判断是不是本身的游戏,如果是就不做处理

        if StrGameName == LunBoMap[k].(map[string]interface{})["GameName"].(string) {
            ListWZ++
            B2bet = true
            break
        }

        // 判断文件存在不
        strPath := getCurrentPath()
        strPath = strPath + "\\YULEGAME\\" + LunBoMap[k].(map[string]interface{})["GameName"].(string)
        dd := checkFileIsExist(strPath)
        if !dd {
            fmt.Println("轮播游戏不存在", strPath)
            ListWZ++
            B2bet = true
            break
        }
        bbret = true
    }
    if bbret {
        fmt.Println("开始执行轮播ID: ", GameID)
        // 关闭运行游戏
        if len(StrGameName) != 0 {
            KillEXE(StrGameName)
        }

        // 启动游戏
        go CallEXE(strLogin+"|"+LunBoMap[k].(map[string]interface{})["GameName"].(string)+"|"+LunBoMap[k].(map[string]interface{})["XianChangName"].(st
            LunBoMap[k].(map[string]interface{})["IpAndPort"].(string), LunBoMap[k].(map[string]interface{})["GameName"].(string))
        // 更新游戏启动状态
        strLogin := ReadFile()
        url := "http://www.yulegame.cn:81/API/AIO.ashx?method=updatagame&username=" + strLogin + "&gamerunstate=true" + "&gamerunname=" + LunBoMap[k].(
        _, Err := http.Get(url)
        if Err != nil {
            glog.Info("http://www.yulegame.cn:81/API/", Err.Error())
        }
        break
    }
```

（Golang 语言社区）

（Golang 技术社区）