

学习心得

结构体 (Struct)

分享人：Roy

建议：

- 代码片段中可能包含与本次分享主题无关的辅助代码，可以酌情理解和记忆
- 如果有不理解的部分，可以先记下笔记，之后再慢慢理解
- 分享过程中可随时打断提问，但是否回答由分享人斟酌，可在Q&A部分回答
- 先理解编程思想(比如结构体的定义、逻辑含义)，再记忆语法细节
- 减少被动记忆，主动思考为什么语法这样设计

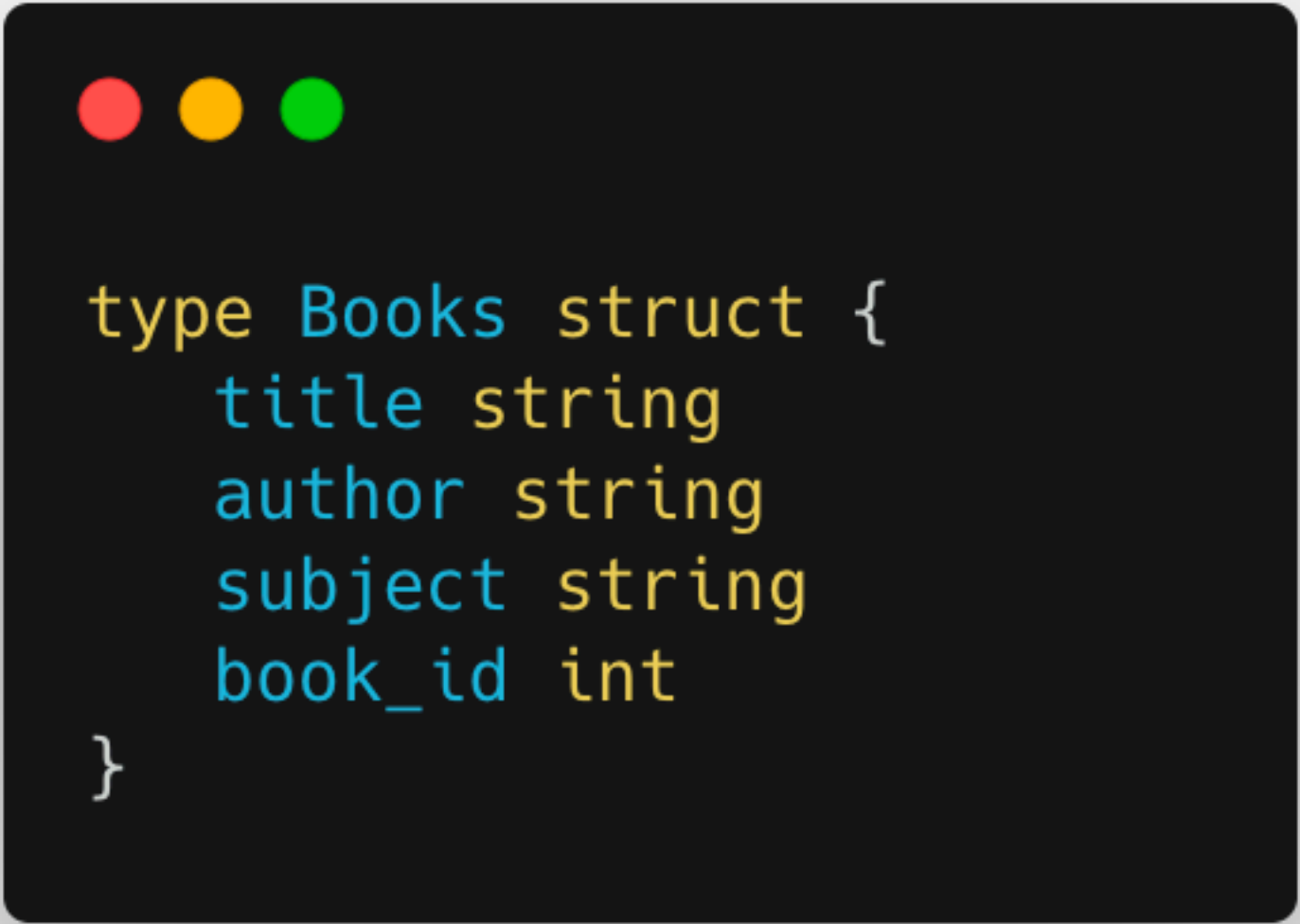
Agenda

- 定义
- 用法
 - 声明（标签、嵌套）
 - 实例化
 - 字段访问
 - 方法（简略）
 - 接口（简略）
 - 指针（简略）
 - 匿名结构体（省略，请自行探索）
- 与其他语言中类的区别
- 最佳实践
- 使用场景

定义

- 字段组成结构体，每个字段包含名称和类型
- 结构体表示现实世界中的实体，比如人、衣服、玩具等
- 结构体表示虚拟世界中的实体，比如订单、充值卡、游戏装备等
- 结构体的字段表示实体的属性，比如订单号，人的身高
- 结构体可以实现方法(非结构体类型也可以实现方法)
- 结构体的实现的方法可以用接口类型代替表示(设计模式、SOLID原则)
- 结构体的方法表示实体的行为，比如人说话，订单修改状态
- 结构体可以嵌套，多个结构体不能相互循环嵌套
- 函数和结构体是组成程序逻辑的常用的代码元素，类似对象和方法

定义



```
type Books struct {  
    title string  
    author string  
    subject string  
    book_id int  
}
```

示例：用结构体表示图书这个实体

用法



```
//talk is cheap, show me the code
```

声明

```
type DemoStruct struct {  
    Foo string `json:"foo"`  
    Bar string  
    lowerStr string  
    Number int  
    string  
    BaseStruct  
    AnotherBase AnotherBaseStruct  
}
```

- DemoStruct: 结构体名称
- Bar: 结构体字段（同变量）
- `json:"foo"`: 字段标签，用于json包解析
- lowerStr: 包私有字段
- string: 匿名字段
- AnotherBase: 嵌套结构体
- BaseStruct: 匿名嵌套结构体

声明（结构体名称）

- 表达结构体的逻辑含义
- 可以包含字母、数字、下划线、`unicode`字符(中文)
- `unicode`字符很少使用
- 不能以数字开头

声明（结构体名称）

```
type 1BaseStruct struct {  
    Name string  
}
```

错误示例(IDE提示报错)

声明（结构体字段）

- 字段命名规则同结构体名
- 字段名不能重复，不包括嵌套字段

声明（字段标签）

- 字段的附加属性
- 可以通过反射API获取
- 具体用法和具体场景有关，比如定义json编码中结构体字段对应的json字段

声明（字段标签）

```
func TestTag(t *testing.T) {  
    jsonStr, errJsonEncoding := json.Marshal(&DemoStruct{  
        Foo:        "",  
        Bar:        "",  
        lowerStr:    "",  
        Number:      0,  
        string:      "",  
        BaseStruct:  BaseStruct{},  
        AnotherBase: AnotherBaseStruct{},  
        Age:         "",  
    })  
    if errJsonEncoding != nil {  
        t.Log(errJsonEncoding)  
        t.Fail()  
    } else {  
        t.Log(string(jsonStr))  
    }  
}
```

声明（字段标签）


```
luoxiaojundeMacBook-Pro:tests luoxiaojun$ go test -v
=== RUN   TestTag
    TestTag: struct\_test.go:43: {"foo":"","Bar":"","Number":0,"Name":"","AnotherBase":{"AnotherName":"","Age":""}}
--- PASS: TestTag (0.00s)
PASS
ok      _/Users/luoxiaojun/php/jingsocial/learning-go/roy/practices/tests    0.423s
luoxiaojundeMacBook-Pro:tests luoxiaojun$
```

- 结构体字段Foo对应的json字段名为foo
- json字段名与结构体声明时的字段标签定义一致

声明（匿名字段）

- 直接用类型声明结构体字段
- 相同类型的匿名字段只能声明一个

声明（匿名字段）

```
type DemoStruct struct {  
    Foo string `json:"foo"`  
    Bar string  
    lowerStr string  
    Number int  
    string  
    string  
     BaseStruct  
    AnotherBase AnotherBaseStruct  
    Age string  
}
```

错误示例(IDE提示报错)

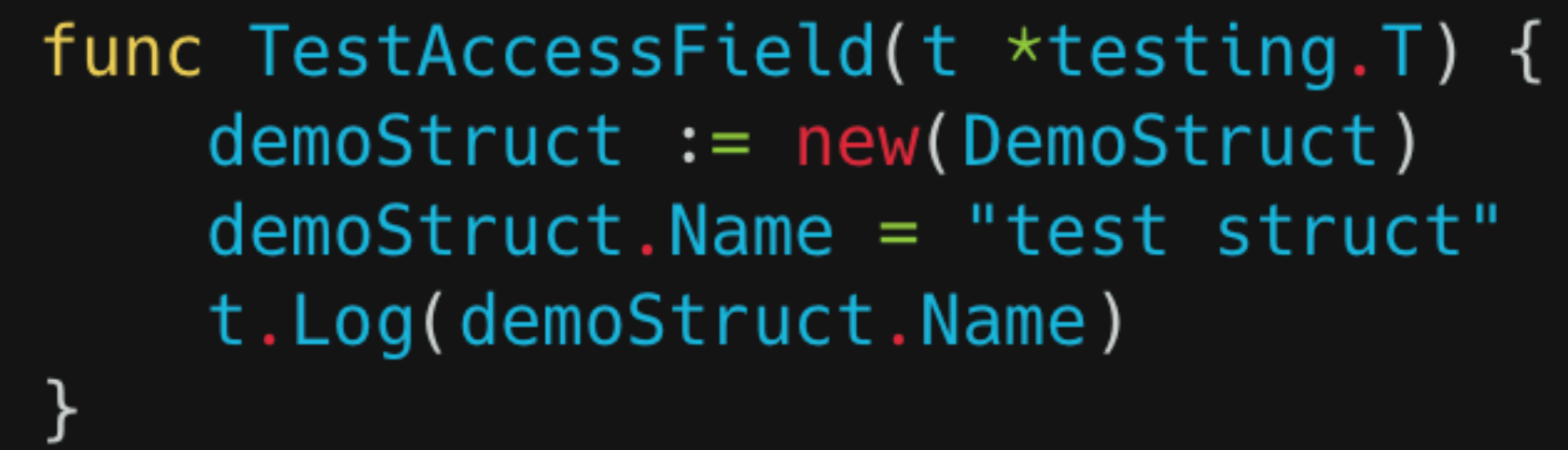
声明（嵌套）

- 嵌套的结构体的字段可以与外层结构体重名
- 多个结构体不能相互循环嵌套

实例化

- new函数实例化
- 字面量实例化
- 字面量指针实例化
- 不指定字段名实例化

实例化 (**new**函数)



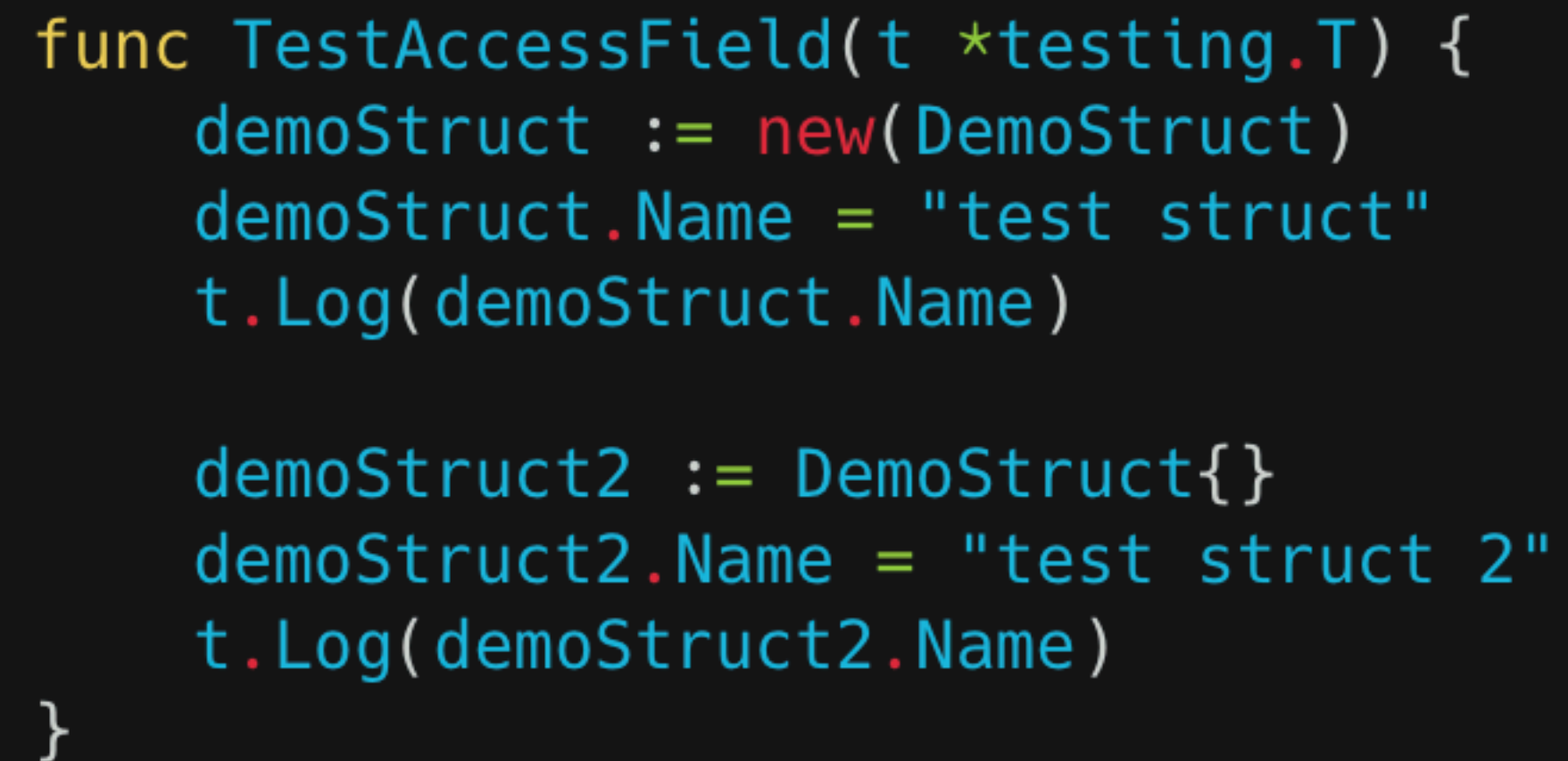
```
func TestAccessField(t *testing.T) {  
    demoStruct := new(DemoStruct)  
    demoStruct.Name = "test struct"  
    t.Log(demoStruct.Name)  
}
```

实例化（new函数）

```
luoxiaojundeMacBook-Pro:tests luoxiaojun$ go test -v
=== RUN   TestAccessField
    TestAccessField: struct\_test.go:31: test struct
--- PASS: TestAccessField (0.00s)
```

new函数返回值类型为结构体的指针

实例化（字面量）



```
func TestAccessField(t *testing.T) {  
    demoStruct := new(DemoStruct)  
    demoStruct.Name = "test struct"  
    t.Log(demoStruct.Name)  
  
    demoStruct2 := DemoStruct{}  
    demoStruct2.Name = "test struct 2"  
    t.Log(demoStruct2.Name)  
}
```

实例化（字面量）

```
luoxiaojundeMacBook-Pro:tests luoxiaojun$ go test -v
=== RUN   TestAccessField
    TestAccessField: struct\_test.go:31: test struct
    TestAccessField: struct\_test.go:35: test struct 2
--- PASS: TestAccessField (0.00s)
```

实例化（字面量指针）

```
func TestAccessField(t *testing.T) {  
    demoStruct := new(DemoStruct)  
    demoStruct.Name = "test struct"  
    t.Log(demoStruct.Name)  
  
    demoStruct2 := DemoStruct{}  
    demoStruct2.Name = "test struct 2"  
    t.Log(demoStruct2.Name)  
  
    demoStruct3 := &DemoStruct{}  
    demoStruct3.Name = "test struct 3"  
    t.Log(demoStruct3.Name)  
}
```

实例化（字面量指针）

```
luoxiaojundeMacBook-Pro:tests luoxiaojun$ go test -v
=== RUN   TestAccessField
    TestAccessField: struct\_test.go:31: test struct
    TestAccessField: struct\_test.go:35: test struct 2
    TestAccessField: struct\_test.go:39: test struct 3
--- PASS: TestAccessField (0.00s)
```

实例化（不指定字段名）

```
demoStruct4 := DemoStruct{
    "test struct foo 2",
    "",
    "",
    0,
    "",
    BaseStruct{},
    AnotherBaseStruct{},
    "",
}
t.Log(demoStruct4.Foo)
```


实例化（不指定字段名）

```
TestAccessField: struct\_test.go:88: test struct foo 2
```

- 实例化时可以不指定字段名
- 指定字段名和不指定字段名的值不能同时存在
- 不指定字段名时必须提供所有字段的初始化值
- 指定字段名时不需要提供所有字段的初始化值
- 可以不提供所有字段的初始化值，默认为类型零值
- 匿名字段也可以指定字段名，即类型名称
- 匿名字段的初始化规则也必须遵守上面的规则

字段访问

- 普通字段访问
- 匿名字段访问
- 嵌套结构体字段访问
- 匿名嵌套结构体字段访问

实例化时没有赋值，并且没有单独赋值的字段，字段值为字段类型的零值

字段访问（普通）



```
demoStruct3.Foo = "test struct foo"  
t.Log(demoStruct3.string)
```

字段访问（普通）

```
luoxiaojundeMacBook-Pro:tests luoxiaojun$ go test -v
=== RUN    TestAccessField
    TestAccessField: struct\_test.go:31: test struct
    TestAccessField: struct\_test.go:35: test struct 2
    TestAccessField: struct\_test.go:39: test struct 3
    TestAccessField: struct\_test.go:42: test struct foo
```

字段访问（匿名）



```
demoStruct3.string = "test struct anonymous field"  
t.Log(demoStruct3.string)
```

字段访问（匿名）

```
luoxiaojundeMacBook-Pro:tests luoxiaojun$ go test -v
=== RUN   TestAccessField
    TestAccessField: struct\_test.go:31: test struct
    TestAccessField: struct\_test.go:35: test struct 2
    TestAccessField: struct\_test.go:39: test struct 3
    TestAccessField: struct\_test.go:42: test struct anonymous field
--- PASS: TestAccessField (0.00s)
```

- 匿名字段使用类型名称访问，比如 string

字段访问（嵌套）



```
demoStruct3.AnotherBase.AnotherName = "test another base struct name"  
t.Log(demoStruct3.AnotherBase.AnotherName)
```

字段访问（嵌套）

```
luoxiaojundeMacBook-Pro:tests luoxiaojun$ go test -v
=== RUN   TestAccessField
    TestAccessField: struct\_test.go:31: test struct
    TestAccessField: struct\_test.go:35: test struct 2
    TestAccessField: struct\_test.go:39: test struct 3
    TestAccessField: struct\_test.go:42: test struct foo
    TestAccessField: struct\_test.go:45: test struct anonymous field
    TestAccessField: struct\_test.go:48: test another base struct name
--- PASS: TestAccessField (0.00s)
```

- 非匿名的嵌套字段通过二级选择器(.符号)访问

字段访问（匿名嵌套）



```
demoStruct3.Name = "test base struct name"  
t.Log(demoStruct3.Name)
```

字段访问（匿名嵌套）

```
luoxiaojundeMacBook-Pro:tests luoxiaojun$ go test -v
=== RUN   TestAccessField
    TestAccessField: struct\_test.go:31: test struct
    TestAccessField: struct\_test.go:35: test struct 2
    TestAccessField: struct\_test.go:39: test struct 3
    TestAccessField: struct\_test.go:42: test struct foo
    TestAccessField: struct\_test.go:45: test struct anonymous field
    TestAccessField: struct\_test.go:48: test another base struct name
    TestAccessField: struct\_test.go:51: test base struct name
--- PASS: TestAccessField (0.00s)
```

- 匿名嵌套字段访问方式可以与普通字段相同
- 匿名嵌套字段也可以用普通嵌套字段的方式访问

字段访问（匿名嵌套）



```
demoStruct3.BaseStruct.Name = "test base struct name 2"  
t.Log(demoStruct3.BaseStruct.Name)
```

字段访问（匿名嵌套）

```
TestAccessField: struct_test.go:54: test base struct name 2
```

- 匿名嵌套字段访问方式可以与普通字段相同
- 匿名嵌套字段也可以用普通嵌套字段的方式访问
- 匿名嵌套字段的字段名为匿名嵌套结构体的名称，比如BaseStruct

字段访问（匿名嵌套）

```
demoStruct3.AnotherName = "test another base name"
```

- 非匿名的嵌套字段只能通过二级选择器(.符号)访问
- 非匿名的嵌套字段不能使用匿名嵌套字段直接访问的方式
- 非匿名的嵌套字段不能使用普通字段直接访问的方式

字段访问（匿名嵌套字段覆盖）

```
demoStruct3.Age = 1  
t.Log(demoStruct3.Age)
```

- 错误示例，IDE报错
- 嵌套的Age字段被最外层结构体Age字段覆盖
- 访问的Age字段为最外层结构体的Age字段
- Age字段的类型为最外层结构体的Age字段类型(string)

字段访问（匿名嵌套字段覆盖）

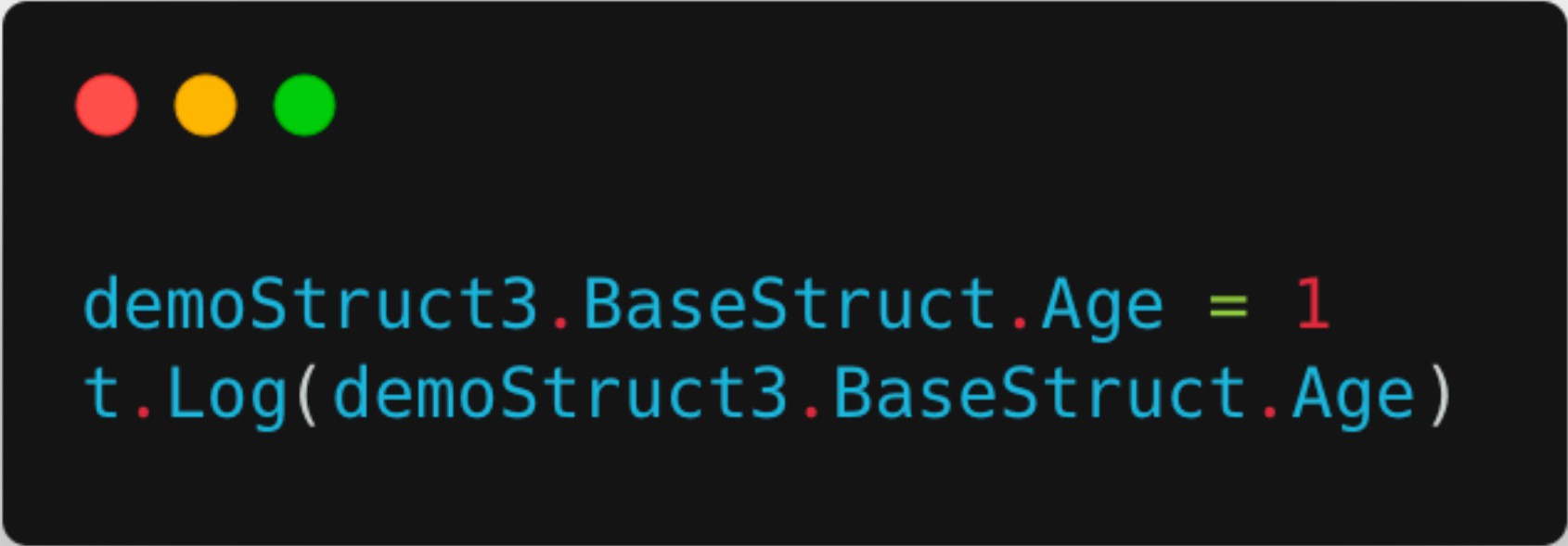


```
demoStruct3.Age = "test struct age"  
t.Log(demoStruct3.Age)
```

字段访问（匿名嵌套字段覆盖）

```
TestAccessField: struct\_test.go:54: test struct age  
--- PASS: TestAccessField (0.00s)
```


字段访问（匿名嵌套字段覆盖）



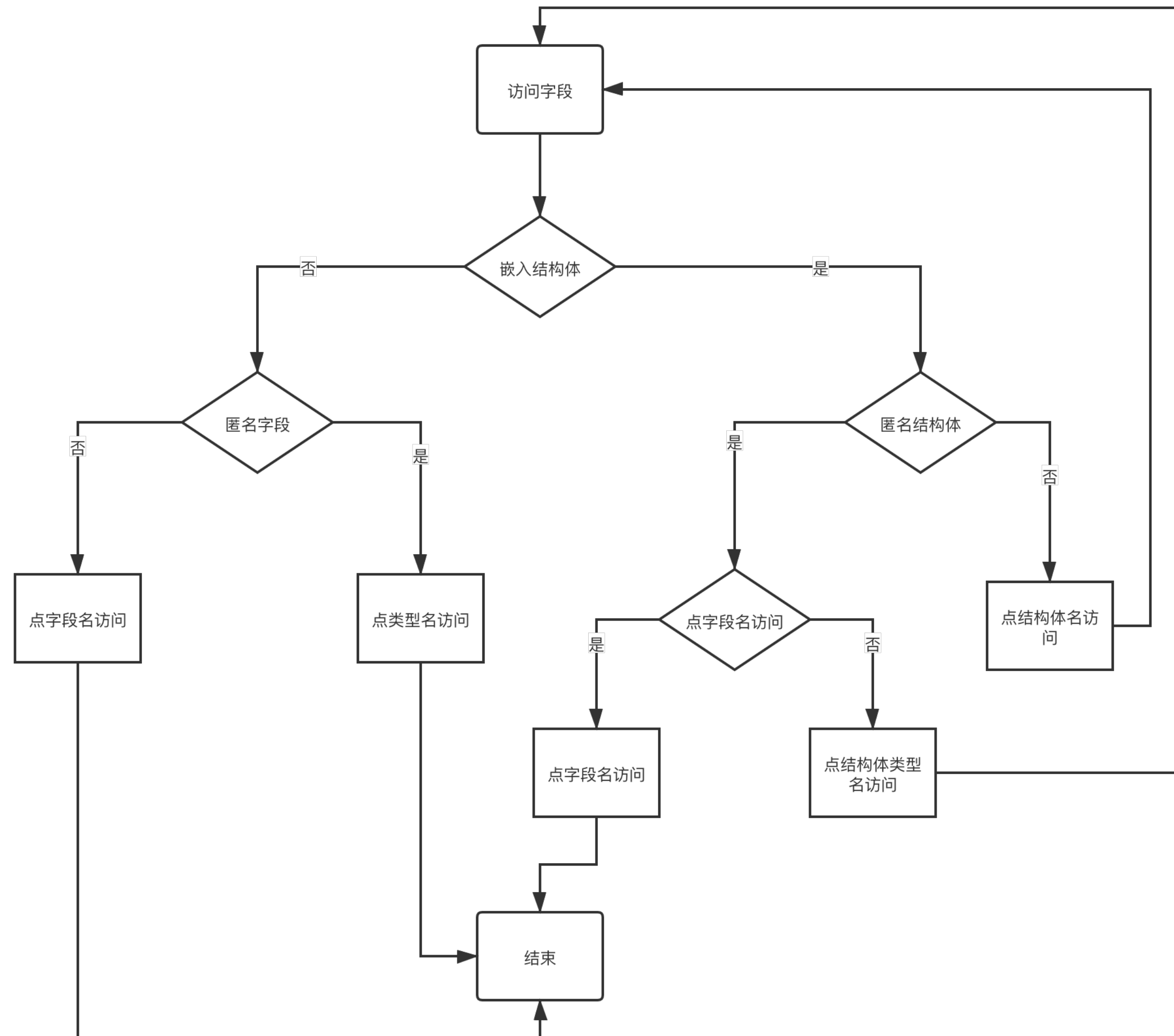
```
demoStruct3.BaseStruct.Age = 1  
t.Log(demoStruct3.BaseStruct.Age)
```

字段访问（匿名嵌套字段覆盖）

```
TestAccessField: struct_test.go:60: 1  
--- PASS: TestAccessField (0.00s)
```

可以看到，通过普通嵌套字段访问方式访问匿名字段时，嵌套字段不会被父结构体的字段覆盖。

字段访问（流程图）

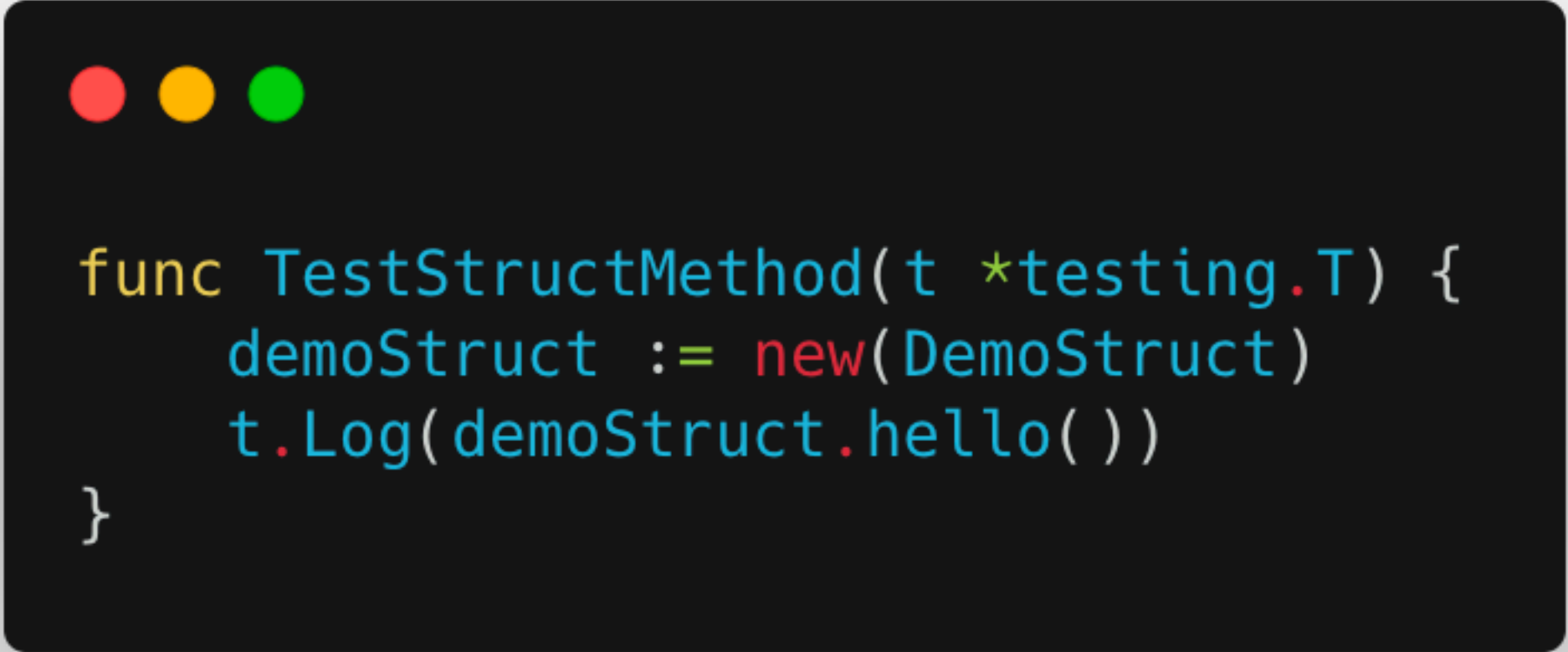


方法



```
func (demo *DemoStruct) hello() string {  
    return "Hello"  
}
```

方法

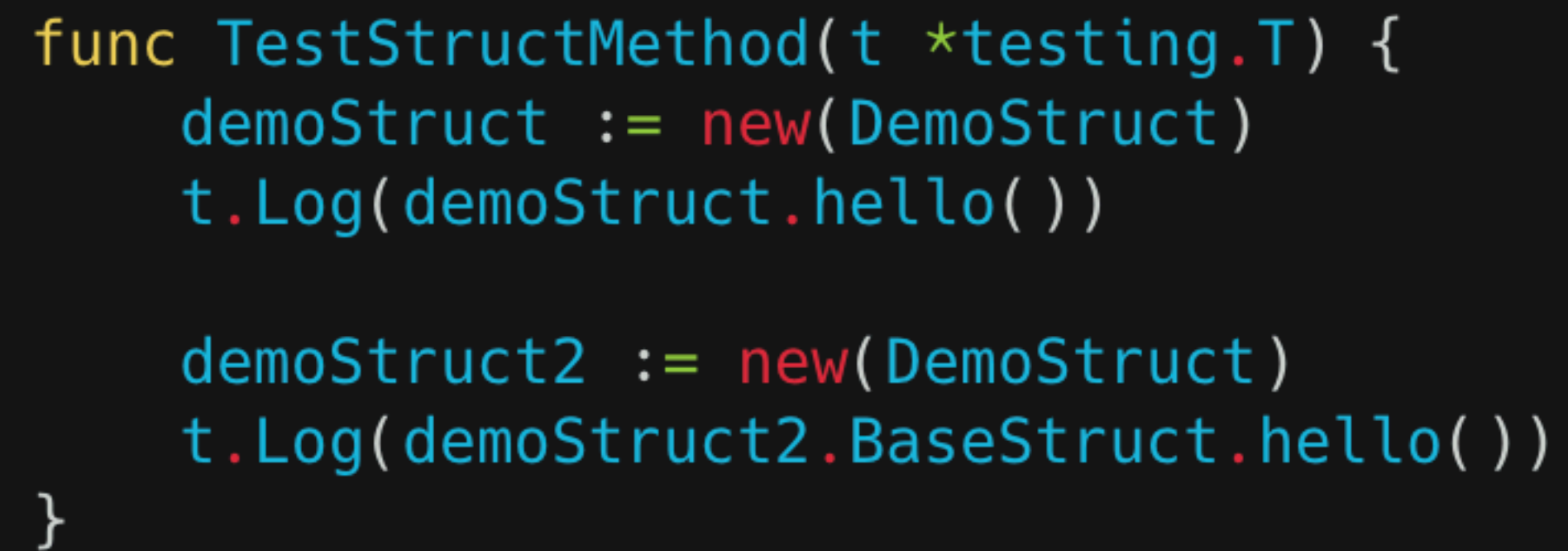


```
func TestStructMethod(t *testing.T) {  
    demoStruct := new(DemoStruct)  
    t.Log(demoStruct.hello())  
}
```

方法

```
=== RUN   TestStructMethod
    TestStructMethod: struct\_test.go:88: Hello
--- PASS: TestStructMethod (0.00s)
PASS
```

方法(匿名嵌套结构体方法覆盖)



```
func TestStructMethod(t *testing.T) {  
    demoStruct := new(DemoStruct)  
    t.Log(demoStruct.hello())  
  
    demoStruct2 := new(DemoStruct)  
    t.Log(demoStruct2.BaseStruct.hello())  
}
```

方法(匿名嵌套结构体方法覆盖)

```
=== RUN   TestStructMethod
    TestStructMethod: struct\_test.go:100: Hello
    TestStructMethod: struct\_test.go:103: Hello Base
--- PASS: TestStructMethod (0.00s)
```

匿名嵌套结构体方法的覆盖与字段覆盖规则相同

接口




```
type DemoInterface interface {  
    hello() string  
}
```

接口



```
func GeneralHello(demo DemoInterface) string {  
    return demo.hello()  
}
```

接口



```
func TestStructInterface(t *testing.T) {  
    t.Log(GeneralHello(new(DemoStruct)))  
}
```

接口

```
=== RUN   TestStructInterface
    TestStructInterface: struct\_test.go:100: Hello
--- PASS: TestStructInterface (0.00s)
```

与其他语言中类的区别

- 结构体可以理解为轻量级的面向对象中的类
- 结构体的方法可以单独定义，类的方法通常与类同时定义
- 结构体与接口可以单独定义，类的接口也可以单独定义
- 结构体与接口的实现关系在使用结构体的地方进行判断
- 结构体的实现者可以不知道接口的存在
- 类与接口的实现关系需要显式声明(`implements`关键字)

与其他语言中类的区别

- 结构体与结构体之间的关系通过嵌套实现
 - 结构体可以嵌套多个结构体
 - 类与类的关系可以通过继承实现，也可以嵌套组合
 - 类不可以继承多个类
-
- 直接访问嵌套的结构体的字段，不会读取外层结构体的字段
 - 读取父类的属性会根据访问权限判断是否读取子类的属性

与其他语言中类的区别

- 嵌入的结构体是独立的结构体
- 类继承父类后实例化的对象中没有独立的父类对象
- 结构体没有抽象结构体的概念
- 类包含抽象类和可实例化的类
- 结构体没有常量字段的概念
- 类有常量属性的概念

与其他语言中类的区别

欢迎补充！

最佳实践（嵌入）



匿名嵌入的结构体应位于结构体内的字段列表的顶部，并且用一个空行将其和常规字段分隔开

最佳实践（嵌入）

内嵌的作用是在符合语义（逻辑）上属于一体的前提下添加或增强结构体的功能

不应该在结构体中嵌入结构体的场景：

- 纯粹为了代码调用方便(比如与外层结构体有关联，但逻辑上并不属于一体的结构体)
- 导致结构体实例化困难或者使用困难

最佳实践（实例化）



```
sval := T{Name: "foo"}
```

```
sptr := &T{Name: "bar"}
```

实例化结构体指针时，用`&T{}`代替`new(T)`，与非指针结构体实例化方式保持一致

使用场景

代码看累了，聊点轻松的。。。。

使用场景

- 数据模型(Model)
- 领域实体
- 依赖注入容器
- 其他

使用场景

欢迎补充！

参考资料

- 代码示例：[代码示例](#)
- 入门教程：[入门指南](#)

参考资料

打个广告，个人技术分享公众号，欢迎订阅交流！



Q & A

谢谢！