Struct的使用

1. 定义（type的两种用法）

示例代码

```
// 类型别名
type t = int64  // t和int64 指向同一个类型元数据
// 定义类型
type t int64    // t和int64 指向两个类型元数据 即使他们的类型相同

type t1 = int64
type t2 int64
var a t1
var b t2
fmt.Printf("t1 type is %T \n", a)
fmt.Printf("t2 type is %T \n", b)
// t1 type is int64
// t2 type is main.t2
```

2. 书写（内存对齐）

CPU把内存当成是一块一块的，块的大小可以是2，4，8，16字节大小，因此CPU在读取内存时是一块一块进行读取的

示例代码

```
type t1 struct {
 int8
 int64
 int16
 int32
 bool
}
```

```
type t2 struct {
 int8
 bool
 int16
 int32
 int64
}

fmt.Printf("int8 size is %v \n", unsafe.Sizeof(int8(0)))
fmt.Printf("int16 size is %v \n", unsafe.Sizeof(int16(0)))
fmt.Printf("int32 size is %v \n", unsafe.Sizeof(int32(0)))
fmt.Printf("int64 size is %v \n", unsafe.Sizeof(int64(0)))
// int8 size is 1
// int32 size is 4
// int64 size is 8

fmt.Printf("float32 size is %v \n", unsafe.Sizeof(float32(0)))
fmt.Printf("float64 size is %v \n", unsafe.Sizeof(float64(0)))
// float32 size is 4
// float64 size is 8

fmt.Printf("bool size is %v \n", unsafe.Sizeof(bool(true)))
// bool size is 1

fmt.Printf("string size is %v \n", unsafe.Sizeof("string"))
fmt.Printf("string size is %v \n", unsafe.Sizeof("string,string"))
// string size is 16
// string size is 16

fmt.Printf("slice size is %v \n", unsafe.Sizeof(make([]int,100,1000)))
fmt.Printf("slice size is %v \n", unsafe.Sizeof(make([]string,100,1000)))
// slice size is 24
// slice size is 24

fmt.Printf("map size is %v \n", unsafe.Sizeof(make(map[string]int)))
fmt.Printf("map size is %v \n", unsafe.Sizeof(make(map[string]string)))
// map size is 8
// map size is 8

s :="string"
p :=&s
fmt.Printf("pointer size is %v \n", unsafe.Sizeof(p))
// pointer size is 8

fmt.Printf("t1 size is %v \n", unsafe.Sizeof(t1{}))
fmt.Printf("t2 size is %v \n", unsafe.Sizeof(t2{}))
// t1 size is 32
// t2 size is 16
```

- 结构体的成员变量，第一个成员变量的偏移量为 0。往后的每个成员变量的对齐值必须为编译器默认对齐长度或当前成员变量类型的长度，取最小值作为当前类型的对齐值。其偏移量必须为对齐值的整数倍

- 结构体本身，对齐值必须为编译器默认对齐长度或结构体的所有成员变量类型中的最大长度，取最大数的最小整数倍作为对齐值

3. 初始化（代码规范）

3.1. 使用字段名初始化结构体

应该始终指定字段名

Bad Good

```
k := User{"John", "Doe", true}
```

```
k := User{
    FirstName: "John",
    LastName: "Doe",
    Admin: true,
}
```

3.2. 省略结构中的零值字段

初始化具有字段名的结构时，除非提供有意义的上下文，否则忽略值为零的字段

Bad Good

```
user := User{
  FirstName: "John",
  LastName: "Doe",
  MiddleName: "",
  Admin: false,
}
```

```
user := User{
  FirstName: "John",
  LastName: "Doe",
}
```

3.3. 对零值结构使用 var

如果在声明中省略了结构的所有字段，使用 var 声明结构

Bad Good

```
user := User{}
```

```
var user User
```

3.4. 初始化 Struct 引用

在初始化结构引用时，使用&T{}代替new(T)，以使其与结构体初始化一致

Bad Good

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

sptr := new(T)
sptr.Name = "bar"
```

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

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

4. 方法的定义

Go 方法是作用在接收者（receiver）上的一个函数，接收者是某种类型的变量。因此方法是一种特殊类型的函数

示例代码

```
type A struct {
    name string
}

func (a A) Name() string {
    return a.name
}

func Name(a A) string {
    return a.name
}

fmt.Printf("A.Name() type is %T \n",A.Name)
fmt.Printf("Name(A) type is %T \n",Name)
```

```
// A.Name() type is func(main.A) string
// Name(A) type is func(main.A) string
```

接收者中的参数变量名在命名时,应使用接收者类型名的第一个小写字母，而不是self、this之类的命名

Bad Good

```
type A struct {
    name string
}

func (this A) Name() string {
    return this.name
}
// 等价
func Name(this A) string {
    return this.name
}
```

```
type A struct {
    name string
}

func (a A) Name() string {
    return a.name
}
```

虽然方法和类型关联，但是方法的本质是函数，使用self、this在函数之中的实际意义是指函数本身，而不是类型实例

5. 方法的调用

```
type A struct {
    name string
}

func (a A) Name() string {
    return a.name
}

func Name(a A) string {
    return a.name
}

func (a *A)SetName(name string){
    a.name = name
```

```
}

a := A{name:"golang"}
p := &a
a.SetName("gin") // 编译阶段 (&a).SetName("gin")
fmt.Println(p.Name()) // 编译阶段 (*a).Name()
// gin

A{name:"golang"}.SetName("gin")
// 编译错误 cannot call pointer method on A{...}

(&A{name:"golang"}).SetName("gin")
```

6. 将方法赋给变量

```
type A struct {
    name string
}

func (a A) Name() string {
    return a.name
}

a := A{name:"golang"}
f1 := A.Name
f1(a)

f2 := a.Name
f2()
```

6.1. function value

将函数赋值给变量或者作为参数及返回值时，这种变量、参数、返回值成为function value

```
f1 :=A.Name
// f1是一个指针 指向funcval
type funcval struct{
    fn uintptr
}
// fn指向函数的入口地址
```

7. 匿名字段与内嵌结构体

```
type A struct {
    int1 int
    int2 int
```

```
}

type B struct {
    int0    int
    int  // 匿名字段
    A //内嵌结构体
}

b := new(B)
b.int=10
b.int1=20
```