

结构struct

- Go 中的struct与C中的struct非常相似，并且Go没有class
- 使用 `type <Name> struct{}` 定义结构，名称遵循可见性规则
- 支持指向自身的指针类型成员
- 支持匿名结构，可用作成员或定义成员变量
- 匿名结构也可以用于map的值
- 可以使用字面值对结构进行初始化
- 允许直接通过指针来读写结构成员
- 相同类型的成员可进行直接拷贝赋值
- 支持 `==` 与 `!=` 比较运算符，但不支持 `>` 或 `<`
- 支持匿名字段，本质上是定义了以某个类型名为名称的字段
- 嵌入结构作为匿名字段看起来像继承，但不是继承
- 可以使用匿名字段指针

struct在go语言当中，就代替了class的位置，但是并没有代替class的功能。因为在go语言没有继承的概念。

go语言没有指针运算。

go语言struct结构代码：

定义struct结构：

定义了一个空的结构。使用 `type` 这个关键字，它也是一种类型。

```
1. package main
2.
3. import (
4.     "fmt"
5. )
6.
7. type test struct{} //定义名为test的struct结构
8.
9. func main(){
10.
11. }
```

```
1. package main
2.
3. import (
4.     "fmt"
5. )
6.
7. type test struct{ //定义名为test的struct结构
8.
9. }
10.
11. func main(){
```

```
12.
13. }
```

设定一个变量，把struct类型test的赋值给这个变量

```
1. package main
2.
3. import (
4.     "fmt"
5. )
6.
7. type test struct { //定义名为test的struct结构
8.
9. }
10.
11. func main() {
12.     a := test{} //
13.     fmt.Println(a)
14. }
```

```
/home/jiemin/code/GOlang/go/src/struct/struct [/home/jiemin/code/GOlang/go/src/struct]
{}
成功: 进程退出代码 0.
```

输出了一对大括号，说明这个一个空的struct

那么怎么才能为这个struct里面添加一些东西呢？就像声明变量一样类似的，比如说这个person(人)。人的这个属性有Name sting和Age int。

```
1. package main
2.
3. import (
4.     "fmt"
5. )
6.
7. type person struct { //定义名为person的struct结构
8.     Name string //定义struct结构属性，string类型的name
9.     Age int //定义struct结构属性，int类型的age
10. }
11.
12. func main() {
13.     a := person{} //
14.     fmt.Println(a)
15. }
```

```
/home/jiemin/code/GOlang/go/src/struct/struct [/home/jiemin/code/GOlang/go/src/struct]
{ 0}
成功: 进程退出代码 0.
```

打印还是空，但是多了一个零。为什么呢，因为Age int 这个int类型，默认是零值。每一种值类型，它都有一个零值。int类型的零值是0，所以它就是输出了0。那么Name string 这个sting类型的

零值就是一个空字符串。所以说它就是空，感觉上是什么都没有输出。

这样就定义了一个 struct 结构，那么怎么样对它进行一个属性的操作呢？


和其它语言一定，就是这个点(.)来操作。Go语言没有指针运算，所以说它就是 这个操作符，来和贴近其它的面向对象的编程语言的用法。

```
package main

import (
    "fmt"
)

type person struct { //定义名为person的struct结构
    Name string //定义struct结构属性,string类型的name
    Age int //定义struct结构属性,int类型的age
}

func main() {
    a := person{} //
    a.
}
```



先设定一个name和age的值

```
1. package main
2.
3. import (
4.     "fmt"
5. )
6.
7. type person struct { //定义名为person的struct结构
8.     Name string //定义struct结构属性, string类型的name
9.     Age int //定义struct结构属性, int类型的age
10. }
11.
12. func main() {
13.     a := person{} //先定义了一个空的struct结构,然后赋值给了a
14.     a.Name = "joe" //名为person的struct结构属性Name定义值
15.     a.Age = 19 //名为person的struct结构属性Age定义值
16.     fmt.Println(a)
17. }
```

```
/home/jiemin/code/Golang/go/src/struct/struct [/home/jiemin/code/Golang/go/src/struct]
{joe 19}
```

成功: 进程退出代码 0.

这种就是其它 class 当中以及其它面向对象语言class的使用几乎是一模一样了。

在上面代码

```
a := person{} //先定义了一个空的struct结构,然后赋值给了a
```

```
a.Name = "joe"
```

```
a.Age = 19
```

这么定义特别繁琐,那么有没有其它简便的方法呢?由于go语言没有class,显然就没有构造函数。但是依旧有一个非常简便的方法,什么方法呢?就像之前使用slice/map,它都可以使用一个字面值,来进行一个初始化。那么怎么使用这个字面值呢?

比如说这个person当中的Name,要对这个Name要进行一个字面值的初始化。就可以输入一个

Name,然后是一个冒号(:) 冒号之后呢,在输入字面值,最后要加上一个逗号(,)

```
1. package main
2.
3. import (
4.     "fmt"
5. )
6.
7. type person struct { //定义名为person的struct结构
8.     Name string //定义struct结构属性, string类型的name
9.     Age int //定义struct结构属性, int类型的age
10. }
11.
12. func main() {
13.     a := person{
14.         Name: "joe", //对person的struct结构中的Name属性进行字面值初始化
15.     }
16.     a.Age = 19 //名为person的struct结构属性Age定义值
17.     fmt.Println(a)
18. }
```

```
/home/jiemini/code/Golang/go/src/struct/struct [/home/jiemini/code/Golang/go/src/struct]
{joe 19}
```

成功: 进程退出代码 0.

同样也对Age进行字面值初始化

```
1. package main
2.
3. import (
4.     "fmt"
5. )
6.
7. type person struct { //定义名为person的struct结构
8.     Name string //定义struct结构属性, string类型的name
9.     Age int //定义struct结构属性, int类型的age
10. }
11.
12. func main() {
13.     a := person{
14.         Name: "joe", //对person的struct结构中的Name属性进行字面值初始化
15.         Age: 19, //对person的struct结构中的Age属性进行字面值初始化
```

```
16.     }
17.     fmt.Println(a)
18. }
```

```
/home/jiemin/code/Golang/go/src/struct/struct [/home/jiemin/code/Golang/go/src/struct]
{joe 19}
```

成功: 进程退出代码 0.

这样就完全达到了要简便初始化的要求，这样一种字面值的初始化在go语言的开发过程中是非常常见的。这种方法是太方便了。

那么要注意的是 `a := person{ Name: "joe", Age: 19, }` 这个 `struct` 结构也是一种值类型，想要对它进行一个传递的时候也是一个值拷贝。是否是这样的呢？来测试一下：

```
1. package main
2.
3. import (
4.     "fmt"
5. )
6.
7. type person struct { //定义名为person的struct结构
8.     Name string //定义struct结构属性，string类型的name
9.     Age  int    //定义struct结构属性，int类型的age
10. }
11.
12. func main() {
13.     a := person{
14.         Name: "joe", //对person的struct结构中的Name属性进行字面值初始化
15.         Age:  19,    //对person的struct结构中的Age属性进行字面值初始化
16.     }
17.     fmt.Println(a)
18.     A(a)          //调用A()函数
19.     fmt.Println(a) //调用之后在打印这个a
20. }
21.
22. func A(per person) { //创建一个A函数，A函数接收一个参数per 是一个person类型
23.     per.Age = 13      //对名为per的struct结构中Age的值进行修改
24.     fmt.Println("A", per) //
25. }
```

```
/home/jiemin/code/Golang/go/src/struct/struct [/home/jiemin/code/Golang/go/src/struct]
{joe 19}
A {joe 13}
{joe 19}
```

成功: 进程退出代码 0.

上面的例子就是很显然得到了值拷贝。那么怎样才能进行一个不是值拷贝呢？很显然就是通过一个指针来进行一个传递。

```
1. package main
2.
3. import (
4.     "fmt"
```

```


5. )
6.
7. type person struct { //定义为person的struct结构
8.     Name string //定义struct结构属性, string类型的name
9.     Age int //定义struct结构属性, int类型的age
10. }
11.
12. func main() {
13.     a := person{
14.         Name: "joe", //对person的struct结构中的Name属性进行字面值初始化
15.         Age: 19, //对person的struct结构中的Age属性进行字面值初始化
16.     }
17.     fmt.Println(a)
18.     A(&a) //调用A()函数,取a的内存地址
19.     fmt.Println(a) //调用之后在打印这个a
20. }
21.
22. func A(per *person) { //创建一个A函数, A函数接收一个参数per 是一个person类型的指针
23.     per.Age = 13 //对名为per的struct结构中Age的值进行修改
24.     fmt.Println("A", per) //
25. }

```

```

/home/jiemin/code/Golang/go/src/struct/struct [/home/jiemin/code/Golang/go/src/struct]
{joe 19}
A &{joe 13}
{joe 13}
成功: 进程退出代码 0.

```



这个时候它就成功将这个person的Age改为13

那么再有一个B()函数也要取a的内存地址, 那么出于性能考虑还是使用指针传递。

```

1. package main
2.
3. import (
4.     "fmt"
5. )
6.
7. type person struct { //定义为person的struct结构
8.     Name string //定义struct结构属性, string类型的name
9.     Age int //定义struct结构属性, int类型的age
10. }
11.
12. func main() {
13.     a := person{
14.         Name: "joe", //对person的struct结构中的Name属性进行字面值初始化
15.         Age: 19, //对person的struct结构中的Age属性进行字面值初始化
16.     }
17.     fmt.Println(a)
18.     A(&a) //调用A()函数,取a的内存地址
19.     B(&a) //调用B()函数,取a的内存地址
20.     fmt.Println(a) //调用之后在打印这个a
21. }
22.

```

```

23. func A(per *person) { //创建一个A函数，A函数接收一个参数per 是一个person类型的指针
24.     per.Age = 13          //对名为person的struct结构中Age的值进行修改
25.     fmt.Println("A", per) //
26. }
27.
28. func B(per1 *person) { //创建一个B函数，B函数接收一个参数per1 是一个person类型的指针
29.     per1.Age = 15         //对名为per1的struct结构中Age的值进行修改
30.     fmt.Println("B", per1) //
31. }

```

/home/jiemin/code/Golang/go/src/struct/struct **[/home/jiemin/code/Golang/go/src/struct]**

```

{joe 19}
A &{joe 13}
B &{joe 15}
{joe 15}

```

成功: 进程退出代码 0.

上面也成功进行了修改，最终结果是B()函数的per1中Age值进行了修改。

那么我有100个怎么办呢？有非常多的时候，那么出于性能考虑都是使用指针进行传递，但是每次都要取地址符号，不是很麻烦吗，那么一般情况都是用指针来保存，那么怎么样把A(&a) 中的a 变成一个指向这个a := person{ Name: "joe", Age: 19, }结构的指针呢？那么很显然，在初始化的时候，把它a := &person{ Name: "joe", Age: 19, }这个地址取出来。来试一下

```

1. package main
2.
3. import (
4.     "fmt"
5. )
6.
7. type person struct { //定义名为person的struct结构
8.     Name string //定义struct结构属性，string类型的name
9.     Age int    //定义struct结构属性，int类型的age
10. }
11.
12. func main() {
13.     a := &person{ //初始化时候，把person的内存地址取出来
14.         Name: "joe", //对person的struct结构中的Name属性进行字面值初始化
15.         Age: 19,    //对person的struct结构中的Age属性进行字面值初始化
16.     }
17.     fmt.Println(a)
18.     A(a)          //调用A()函数，把a传递进去
19.     B(a)          //调用B()函数，把a传递进去
20.     fmt.Println(a) //调用之后在打印这个a
21. }
22.
23. func A(per *person) { //创建一个A函数，A函数接收一个参数per 是一个person类型的指针
24.     per.Age = 13          //对名为person的struct结构中Age的值进行修改
25.     fmt.Println("A", per) //
26. }
27.
28. func B(per1 *person) { //创建一个B函数，B函数接收一个参数per1 是一个person类型的指针
29.     per1.Age = 15         //对名为per1的struct结构中Age的值进行修改

```



```
30.     fmt.Println("B", per1) //
31. }
```

```
/home/jiemin/code/Golang/go/src/struct/struct [/home/jiemin/code/Golang/go/src/struct]
&{joe 19}
A &{joe 13}
B &{joe 15}
&{joe 15}
```

成功: 进程退出代码 0.

前面都有一个取地址的符号(&) 说明这个 a 是一个指针。

那么这个时候又用一个问题，如果说这里是一个指针的话呢，需要像那种class对这个 a 属性进行的一个操作，是不是需要这么样子呢？ *a.Name

在Go语言不需要这样操作，可以这样子 a.Name = "Ok" 看一下行不行：

```
1.  package main
2.
3.  import (
4.      "fmt"
5.  )
6.
7.  type person struct { //定义名为person的struct结构
8.      Name string //定义struct结构属性，string类型的name
9.      Age  int    //定义struct结构属性，int类型的age
10. }
11.
12. func main() {
13.     a := &person{ //初始化时候,把person的内存地址取出来
14.         Name: "joe", //对person的struct结构中的Name属性进行字面值初始化
15.         Age:  19,    //对person的struct结构中的Age属性进行字面值初始化
16.     }
17.     a.Name = "Ok" //修改a中Name的值
18.     fmt.Println(a)
19.     A(a)          //调用A()函数,把a传递进去
20.     B(a)          //调用B()函数,把a传递进去
21.     fmt.Println(a) //调用之后在打印这个a
22. }
23.
24. func A(per *person) { //创建一个A函数，A函数接收一个参数per 是一个person类型的指针
25.     per.Age = 13      //对名为person的struct结构中Age的值进行修改
26.     fmt.Println("A", per) //
27. }
28.
29. func B(per1 *person) { //创建一个B函数，B函数接收一个参数per1 是一个person类型的指针
30.     per1.Age = 15     //对名为per1的struct结构中Age的值进行修改
31.     fmt.Println("B", per1) //
32. }
```



```
/home/jiemin/code/GOlanguo/src/struct/struct [/home/jiemin/code/GOlanguo/src/struct]
&{Ok 19}
A &{Ok 13}
B &{Ok 15}
&{Ok 15}
成功: 进程退出代码 0.
```

发现Name的值变为了Ok

在开发过程中，都推荐在进行初始化时候都直接对 `a := &person{ Name: "joe", Age: 19, }` 这个结构初始化的时候，都习惯性的使用取地址符号(**&**)，这样这个 `a` 就变成了指向某一个结构的指针。然后更加方便的是，在Go语言当中并不需要特意的去取地址。可以还是像之前一样来对它的属性，它的字段进行一个操作。它是完全兼容的。然后在进行一个传递，那么就不需要每次都取地址，只需要在定义函数的时候，设定接收参数类型的地方加上星号(*****)，变成指针类型。在接收到依旧可以像使用类似class一样 `per.Age = 13` 对他进行一个字段的操作。

匿名struct结构

怎么样才是一个匿名struct结构呢？显然就是没有名称的结构。

```
a := &person{ //初始化时候,把person的内存地址取出来
    Name: "joe", //对person的struct结构中的Name属性进行字面值初始化
    Age: 19,    //对person的struct结构中的Age属性进行字面值初始化
}
```

这里的 `a / person` 已经是有一个结构了。

来创建一个匿名结构，没有名称，这里就是一个 `struct{}`。那么既然它匿名，然后我们之前没有定义的话，那么我们就需要 先对这个结构它本身的一个结构 进行一个定义，像刚才一样，定一个Name和Age，然后要注意：

```
a := struct { //匿名结构对变量a进行赋值
    Name string //对匿名结构中的结构属性Name进行定义
    Age int     //对匿名结构中的结构属性Age进行定义
}{
    Name: "joe", //对person的struct结构中的Name属性进行字面值初始化
    Age: 19,    //对person的struct结构中的Age属性进行字面值初始化
}
```

详细代码如下：

```
1. package main
2.
```

```

3. import (
4.     "fmt"
5. )
6.
7. func main() {
8.     a := struct { //匿名结构对变量a进行赋值
9.         Name string //对匿名结构中的结构属性Name进行定义
10.        Age  int    //对匿名结构中的结构属性Age进行定义
11.    }{
12.        Name: "joe", //对person的struct结构中的Name属性进行字面值初始化
13.        Age:  19,    //对person的struct结构中的Age属性进行字面值初始化
14.    }
15.    fmt.Println(a)
16. }

```

/home/jiemine/code/GOLang/go/src/struct/struct [/home/jiemine/code/GOLang/go/src/struct]
{ joe 19 }

成功: 进程退出代码 0.

这个时候就没有为struct结构进行命名，这个时候它就是一个匿名结构，在对

```

1. a := struct { //匿名结构对变量a进行赋值
2.     Name string //对匿名结构中的结构属性Name进行定义
3.     Age  int    //对匿名结构中的结构属性Age进行定义
4. }{
5.     Name: "joe", //对person的struct结构中的Name属性进行字面值初始化
6.     Age:  19,    //对person的struct结构中的Age属性进行字面值初始化
7. }

```

这个 a 进行一个初始化时候，就临时定义了声明了

```

1. struct { //匿名结构对变量a进行赋值
2.     Name string //对匿名结构中的结构属性Name进行定义
3.     Age  int    //对匿名结构中的结构属性Age进行定义
4. }

```

这么一个结构，然后通过刚才的方法对

```

1. {
2.     Name: "joe", //对person的struct结构中的Name属性进行字面值初始化
3.     Age:  19,    //对person的struct结构中的Age属性进行字面值初始化
4. }

```

它进行一个字面值的初始化，最后我们打印它的值。

那么怎么得到这样一个结构的地址呢？将 a 变成一个指针呢？同样在

```

1. a := &struct { //匿名结构对变量a进行赋值
2.     Name string //对匿名结构中的结构属性Name进行定义
3.     Age  int    //对匿名结构中的结构属性Age进行定义
4. }{
5.     Name: "joe", //对person的struct结构中的Name属性进行字面值初始化
6.     Age:  19,    //对person的struct结构中的Age属性进行字面值初始化
7. }

```

这个前面加上取地址符号(&)就可以了

代码如下：

```
1. package main
2.
3. import (
4.     "fmt"
5. )
6.
7. func main() {
8.     a := &struct { //匿名结构去除地址对变量a进行赋值,将a变为一个指针
9.         Name string //对匿名结构中的结构属性Name进行定义
10.        Age  int    //对匿名结构中的结构属性Age进行定义
11.    }{
12.        Name: "joe", //对person的struct结构中的Name属性进行字面值初始化
13.        Age:  19,    //对person的struct结构中的Age属性进行字面值初始化
14.    }
15.    fmt.Println(a)
16. }
```

```
/home/jiemini/code/Golang/go/src/struct/struct [/home/jiemini/code/Golang/go/src/struct]
&{joe 19}
```

成功: 进程退出代码 0.

那么这个匿名结构可不可以嵌套在其它结构当中呢？答案是可以的。

代码如下：

Phone, City string 这就是为什么Go把类型放在变量名的后面，因为它很多地方变的非常便利，而且不会产生歧义。

让a 等于一个空的person。输出一下结果：

```
1. package main
2.
3. import (
4.     "fmt"
5. )
6.
7. type person struct {
8.     Name    string //对结构属性Name进行定义
9.     Age     int    //对结构属性Age进行定义
10.    Contact struct { //定义一个匿名结构
11.        Phone, City string //对匿名结构中的结构属性Phone,City进行定义
12.    }
13. }
14.
15. func main() {
16.     a := person{}
17.     fmt.Println(a)
18. }
```

```
/home/jiemine/code/Golang/go/src/struct/struct [/home/jiemine/code/Golang/go/src/struct]
{ 0 { }}
```

成功: 进程退出代码 0.

看到首先是一个外面的大括号，大括号后面有一个0，那就是int类型Age的零值。然后里面又有一对大括号，显然就是Contact的匿名结构。这个时候怎么对person结构进行初始化呢？那么对于这个Name和Age这两个字段还是可以经行一个字面值的初始化：

```
a := person{Name: "joe", Age: 19}
```

但是这个里面的Contact这个结构，我们要怎么初始化呢？它是一个匿名结构，没有名称。像这里person{Name: "joe", Age: 19}要经行一个字面值初始化，我们要告诉系统，我这里是person结构。可是里面的匿名结构Contact不是结构名称，而是我这里的person结构字段名称，这个时候只能够通过这种方法：

```
1. a.Contact.Phone = "13210019876" //对person结构中的Contact字段属性匿名结构里面的Phone
2. a.Contact.City = "Beijing"      //对person结构中的Contact字段属性匿名结构里面的City
```

看一下代码运行结果：

```
1. package main
2.
3. import (
4.     "fmt"
5. )
6.
7. type person struct {
8.     Name    string //对结构属性Name进行定义
9.     Age     int     //对结构属性Age进行定义
10.    Contact struct { //定义一个匿名结构
11.        Phone, City string //对匿名结构中的结构属性Phone,City进行定义
12.    }
13. }
14.
15. func main() {
16.     a := person{Name: "joe", Age: 19} //对person结构里面的Name 和 Age 字段进行初始化
17.     a.Contact.Phone = "13210019876" //对person结构中的Contact字段属性匿名结构里面的
18.     a.Contact.City = "Beijing"      //对person结构中的Contact字段属性匿名结构里面的
19.     fmt.Println(a)
20. }
```

```
/home/jiemine/code/Golang/go/src/struct/struct [/home/jiemine/code/Golang/go/src/struct]
{joe 19 {13210019876 Beijing}}
```

成功: 进程退出代码 0.

匿名字段

匿名字段是怎么回事呢？就是定义好的结构中，没有字段参数名称，只有字段参数类型。

代码如下：

```
1. package main
2.
3. import (
4.     "fmt"
5. )
6.
7. type person struct {
8.     string //person结构的匿名字段
9.     int    //person结构的匿名字段
10. }
11.
12. func main() {
13.     a := person{"joe", 19} //对person结构进行字面值初始化,赋值给a
14.     fmt.Println(a)        //打印a
15. }
```

```
/home/jiemine/code/Golang/go/src/struct/struct [/home/jiemine/code/Golang/go/src/struct]
{joe 19}
```

成功: 进程退出代码 0.

```
1. package main
2.
3. import (
4.     "fmt"
5. )
6.
7. type person struct {
8.     string //person结构的匿名字段
9.     int    //person结构的匿名字段
10. }
11.
12. func main() {
13.     a := person{19, "joe"} //对person结构进行字面值初始化,赋值给a
14.     fmt.Println(a)        //打印a
15. }
```

```
/usr/local/go/bin/go build -i [/home/jiemine/code/Golang/go/src/struct]
```

```
# _/home/jiemine/code/Golang/go/src/struct
```

```
./struct.go:13: cannot use 19 (type int) as type string in field value
```

```
./struct.go:13: cannot use "joe" (type string) as type int in field value
```

错误: 进程退出代码 2.

程序异常，为什么呢？代码中的person结构中的字段都是匿名的，没有办法像刚才那样使用Name, Age 这样明确指定这个Name名称赋予它什么值，定义person结构中的类型有string和int类型。之后在字面值初始化时候顺序又是int类型在前，string类型在后。因为是匿名的，唯一可以依靠的是什么呢？就是定义person结构本身的顺序。所以说如果使用了匿名字段的时候，在进行一个字面值初始化的时候。**必须严格按照定义的结构中这个字段声明的顺序。**

像我们这里定一个person结构

```

1. type person struct {
2.     string //person结构的匿名字段
3.     int    //person结构的匿名字段
4. }

```

第一个用了string类型的，那么 `a := person{"joe", 19}` 它就会把第一个数值赋予string类型。然后我们用的是int类型的，那么它就赋给int类型。而如果当我们把int类型和string类型对调 `a := person{19, "joe"}` 的话，它显然这个结构就对不上号了，然后就会爆出这个

```

/usr/local/go/bin/go build -i [/home/jiemin/code/Golang/go/src/struct]
# _/home/jiemin/code/Golang/go/src/struct
./struct.go:13: cannot use 19 (type int) as type string in field value
./struct.go:13: cannot use "joe" (type string) as type int in field value
错误: 进程退出代码 2.

```

错误

结构也是一种类型，那么显然在相同类型之间，它的变量可以进行一个相互之间的赋值。

```

1. package main
2.
3. import (
4.     "fmt"
5. )
6.
7. type person struct {
8.     Name string //person结构的Name字段定义类型
9.     Age  int    //person结构的Age字段定义类型
10. }
11.
12. func main() {
13.     a := person{Name: "joe", Age: 19} //对person结构进行字面值初始化,赋值给a
14.     var b person                      //定义一个相同类型的结构b
15.     b = a                             //在把a的person结构的值赋值给相同person结构b
16.     fmt.Println(b)                   //打印b
17. }

```

```

/home/jiemin/code/Golang/go/src/struct/struct [/home/jiemin/code/Golang/go/src/struct]
{joe 19}
成功: 进程退出代码 0.

```

它也输出了这个 `{joe 19}`，那么这个就是相互的赋值。

那既然可以赋值，那就可以相互之间进行一个比较了

```

1. package main
2.
3. import (
4.     "fmt"
5. )
6.
7. type person struct {

```

```

8.     Name string //person结构的Name字段定义类型
9.     Age  int    //person结构的Age字段定义类型
10. }
11.
12. type person1 struct {
13.     Name string //person结构的Name字段定义类型
14.     Age  int    //person结构的Age字段定义类型
15. }
16.
17. func main() {
18.     a := person{Name: "joe", Age: 19} //对person结构进行字面值初始化,赋值给a
19.     b := person1{Name: "joe", Age: 19} //对person1结构进行字面值初始化,赋值给b
20.     fmt.Println(a == b)               //比较a,b的值
21. }

```

`/usr/local/go/bin/go build -i [/home/jiemine/code/Golang/go/src/struct]`
`# _/home/jiemine/code/Golang/go/src/struct`
`./struct.go:20: invalid operation: a == b (mismatched types person and person1)`
错误: 进程退出代码 2.

这时候它就报错了，为什么呢？因为

```

1. type person struct {
2.     Name string //person结构的Name字段定义类型
3.     Age  int    //person结构的Age字段定义类型
4. }

```

和

```

1. type person1 struct {
2.     Name string //person结构的Name字段定义类型
3.     Age  int    //person结构的Age字段定义类型
4. }

```

尽管它的内容包含相同，但是因为是不同的结构名称，所以说它实际上上不同的结构类型。既然是不同结构类型，它们之间就没有可比性，所以说这段代码就不可以相互比较了。那么进行相互比较，就必须是相同的结构类型。比如下面代码：

```

1. package main
2.
3. import (
4.     "fmt"
5. )
6.
7. type person struct {
8.     Name string //person结构的Name字段定义类型
9.     Age  int    //person结构的Age字段定义类型
10. }
11.
12. func main() {
13.     a := person{Name: "joe", Age: 19} //对person结构进行字面值初始化,赋值给a
14.     b := person{Name: "joe", Age: 19} //对person结构进行字面值初始化,赋值给b
15.     fmt.Println(a == b)               //比较a,b的值
16. }

```



```
/home/jiemin/code/Golang/go/src/struct/struct [/home/jiemin/code/Golang/go/src/struct]
```

true

成功: 进程退出代码 0.

所以结果就输出了true，表示值是相同的。那么修改一下赋值，会有什么结果呢？

```
1. package main
2.
3. import (
4.     "fmt"
5. )
6.
7. type person struct {
8.     Name string //person结构的Name字段定义类型
9.     Age  int    //person结构的Age字段定义类型
10. }
11.
12. func main() {
13.     a := person{Name: "joe", Age: 19} //对person结构进行字面值初始化,赋值给a
14.     b := person{Name: "joe", Age: 20} //对person结构进行字面值初始化,赋值给b
15.     fmt.Println(a == b)              //比较a,b的值
16. }
```

```
/home/jiemin/code/Golang/go/src/struct/struct [/home/jiemin/code/Golang/go/src/struct]
```

false

成功: 进程退出代码 0.

这时候就输出了false，那么和之前讲过的所有的等号和不等号的比较都是一样的。这个比较只能在相同类型的比较，当发生一些奇怪的错误。像这里

```
./struct.go:20: invalid operation: a == b (mismatched types person and person1)
```

说明是类型搞错了，这两个类型完全不一样。那么这时候要检查错误的认为这两个是相同类型。但实际上是不同的类型。这也是一个查找BUG当中要注意的一点。

在Go语言当中没有class，没有继承，那怎么样才能做成比较像class，继承呢？

比如在代码中有一个 human 和 teachre 和 student结构。teachre(译:老师) 和 student(译:学生)结构类型它们俩都是human(译:人类)。那么它们肯定有一些固定的属性。但是又不想每次在这个 teacher和student当中进行一个声明，如果有一些属性要增加或者删除的话，显然就是非常的不方便。这时候就需要用到 **嵌入结构**。嵌入结构就是接一个结构嵌入到另一个结构当中，这个结

构在go语言当中称之为 **组合**，而不是称为**继承**，它是不存在继承的。这一点要明确。

比如说在human中有int类型的Sex，用数值0和1来表示男和女。那么怎么使用呢？

下面代码如下：

```
1. package main
2.
3. import (
4.     "fmt"
```

```

5.  )
6.
7.  type human struct {
8.      Sex int //数值0和1来表示男和女
9.  }
10.
11. type teacher struct {
12.     human      //把human结构嵌入到teacher结构中
13.     Name string //person结构的Name字段定义类型
14.     Age  int    //person结构的Age字段定义类型
15. }
16.
17. type student struct {
18.     human      //把human结构嵌入到student结构中
19.     Name string //person结构的Name字段定义类型
20.     Age  int    //person结构的Age字段定义类型
21. }
22.
23. func main() {
24.     a := teacher{Name: "joe", Age: 19} //对teacher结构进行字面值初始化,赋值给a
25.     b := student{Name: "joe", Age: 20} //对学生结构进行字面值初始化,赋值给b
26.     fmt.Println(a, b)                  //分别打印a,b的值
27. }

```

```

/home/jiemn/code/Golang/go/src/struct/struct [/home/jiemn/code/Golang/go/src/struct]
{{0} joe 19} {{0} joe 20}
成功: 进程退出代码 0.

```

我们可以看到，human结构中的Sex成功的嵌入进去了，然后我们怎么操作这个Sex呢？

比如这个teacher 是一个男的

```

a := teacher{Name: "joe", Age: 19, Sex: 0}
b := student{Name: "joe", Age: 20, Sex: 1}

```

代码如下：

```

1.  package main
2.
3.  import (
4.      "fmt"
5.  )
6.
7.  type human struct {
8.      Sex int //数值0和1来表示男和女
9.  }
10.
11. type teacher struct {
12.     human      //把human结构嵌入到teacher结构中
13.     Name string //person结构的Name字段定义类型
14.     Age  int    //person结构的Age字段定义类型
15. }
16.
17. type student struct {

```

```


18.     human        //把human结构嵌入到student结构中
19.     Name string //person结构的Name字段定义类型
20.     Age  int     //person结构的Age字段定义类型
21. }
22.
23. func main() {
24.     a := teacher{Name: "joe", Age: 19, Sex: 0} //对teacher结构进行字面值初始化,赋值给
25.     b := student{Name: "joe", Age: 20, Sex: 1} //对student结构进行字面值初始化,赋值给
26.     fmt.Println(a, b)                          //分别打印a,b的值
27. }

```

```

/usr/local/go/bin/go build -i [/home/jiemini/code/Golang/go/src/struct]
# _/home/jiemini/code/Golang/go/src/struct
./struct.go:24: unknown teacher field 'Sex' in struct literal
./struct.go:25: unknown student field 'Sex' in struct literal
错误: 进程退出代码 2.

```



这怎么又报错了呢？看到这个

```

1. type student struct {
2.     human        //把human结构嵌入到student结构中
3.     Name string //person结构的Name字段定义类型
4.     Age  int     //person结构的Age字段定义类型
5. }

```

嵌入结构human，这个human结构当中包含Sex这样的字段，那么

在teacher{Name: "joe", Age: 19, Sex: 0}或者student{Name: "joe", Age: 20, Sex: 1}这里直接使用Sex，那么很显然我们这里是有的。那怎么样才可以调用这个human呢？human结构当中的Sex的属性呢？这里面有两种方法：

第一种方法:

```


1. a := teacher{Name: "joe", Age: 19, human{Sex: 0}} //对teacher结构进行字面值初始化,赋值给
2. b := student{Name: "joe", Age: 20, human{Sex: 1}} //对student结构进行字面值初始化,赋值给

```

```

/usr/local/go/bin/go build -i [/home/jiemini/code/Golang/go/src/struct]
# _/home/jiemini/code/Golang/go/src/struct
./struct.go:24: mixture of field:value and value initializers
./struct.go:25: mixture of field:value and value initializers
错误: 进程退出代码 2.

```



为什么还会报错呢？

Go语言知识点:

我们这个

```

1. type student struct {
2.     human        //把human结构嵌入到student结构中
3.     Name string //person结构的Name字段定义类型
4.     Age  int     //person结构的Age字段定义类型
5. }

```

嵌入结构，这结构是一个类型，但是像刚才那样如果像

```
1. type student struct {
2.     human      //把human结构嵌入到student结构中
3.     string      //person结构的Name字段定义类型
4.     Age int      //person结构的Age字段定义类型
5. }
```

刚才那样只有类型，没有名称，这是一个匿名结构，就是一个匿名字段。那么这个匿名字段怎么才能够操作它呢？所以说这里呢，如果是一个嵌入结构作为匿名字段，**它本质上就是将这个结构的名称，作为嵌入到这个结构中的字段名称**。所以说这里，要使用这个字面值来进行一个初始化呢，我们就需要这样子：

这个例子中的human嵌入结构，它是匿名字段，所以系统默认就把human结构名称作为我们的字段名称。

```
1. a := teacher{Name: "joe", Age: 19, human: human{Sex: 0}} //对teacher结构进行字面
2. b := student{Name: "joe", Age: 20, human: human{Sex: 1}} //对student结构进行字面
```

具体golang代码如下：

```
1. package main
2.
3. import (
4.     "fmt"
5. )
6.
7. type human struct {
8.     Sex int //数值0和1来表示男和女
9. }
10.
11. type teacher struct {
12.     human      //把human结构嵌入到teacher结构中
13.     Name string //person结构的Name字段定义类型
14.     Age  int    //person结构的Age字段定义类型
15. }
16.
17. type student struct {
18.     human      //把human结构嵌入到student结构中
19.     Name string //person结构的Name字段定义类型
20.     Age  int    //person结构的Age字段定义类型
21. }
22.
23. func main() {
24.     a := teacher{Name: "joe", Age: 19, human: human{Sex: 0}} //对teacher结构进行字面
25.     b := student{Name: "joe", Age: 20, human: human{Sex: 1}} //对student结构进行字面
26.     fmt.Println(a, b) //分别打印a,b的值
27. }
```

```
/home/jiemin/code/Golang/go/src/struct/struct [/home/jiemin/code/Golang/go/src/struct]
{{0} joe 19} {{1} joe 20}
成功: 进程退出代码 0.
```

要想这种class一样，比较简单的方法来操作这个属性，那要怎么操作呢？
如果我们要操作Name，就可以这样操作

```
1. a.Name = "joe2"
2. a.Age = 13
```

那操作Sex字段怎么操作呢？
应该这样调用吗？

```
1. a.human.Sex = 100
```

```
/home/jiemin/code/Golang/go/src/struct/struct [/home/jiemin/code/Golang/go/src/struct]
{{100} joe2 13} {{1} joe 20}
成功: 进程退出代码 0.
```

发现这样操作成功变成100。这样麻烦，human不是嵌入结构么？这么操作一点都没有嵌入。那么把human去掉能不能成功？

```
1. a.Sex = 100
```

```
/home/jiemin/code/Golang/go/src/struct/struct [/home/jiemin/code/Golang/go/src/struct]
{{100} joe2 13} {{1} joe 20}
成功: 进程退出代码 0.
```

发现成功打印100。所以说进行一个结构的嵌入，实际上它默认就把它的嵌入结构当中的所有字段都给了这个外层这个结构。这个代码中human这个结构中的Sex字段都默认给了 teacher和student结构中。所以说 teacher和student结构中都有Sex，直接有Sex属性。那么为什么go要保留像这种 a.human.Sex = 100 调用方法呢？这个就涉及到一个名称的冲突。

课堂作业

- 如果匿名字段和外层结构有同名字段，应该如何进行操作？
- 请思考并尝试。

作业：

一、先做的俩个同样名称，不同结构的定义：

```
1. package main
2.
3. import (
4.     "fmt"
```

```

5. )
6.
7. type person struct {
8.     Career string
9.     Sex    int
10. }
11.
12. type person struct {
13.     Number int
14.     Name   string
15.     Age    int
16.     person // person结构嵌入到外层名称为person结构中
17. }
18.
19. func main() {
20.     a := person{Number: 1111, Name: "Lilei", Age: 31, person: person{Career: "Police", Sex: 1}}
21.     b := person{Number: 2222, Name: "Hanmeimei", Age: 29, person: person{Career: "Teacher", Sex: 2}}
22.     fmt.Println(a, b)
23. }

```

```

/usr/local/go/bin/go build -i [/home/jiemini/code/GOLang/go/src/struct/work_struct]
# _/home/jiemini/code/GOLang/go/src/struct/work_struct
./work_struct.go:12: person redeclared in this block
previous declaration at ./work_struct.go:7
./work_struct.go:17: invalid recursive type person
./work_struct.go:20: unknown person field 'Career' in struct literal
./work_struct.go:20: unknown person field 'Sex' in struct literal
./work_struct.go:21: unknown person field 'Career' in struct literal
./work_struct.go:21: unknown person field 'Sex' in struct literal

```

错误: 进程退出代码 2.

修改一下代码，使用另外一种方式调用person嵌入结构：

```

func main() {
    a := person{Number: 1111, Name: "Lilei", Age: 31}
    b := person{Number: 2222, Name: "Hanmeimei", Age: 29}
    // 使用类似class方式调用
    a.Career = "Police"
    a.Sex = 1
    fmt.Println(a, b)
}

```

发现没有发现person外层结构中定义的

```

1. type person struct {
2.     Number int
3.     Name   string
4.     Age    int
5.     person // person结构嵌入到外层名称为person结构中
6. }

```


这些字段名称。而且无法执行代码

```
/usr/local/go/bin/go build -i [/home/jiemin/code/GOlang/go/src/struct/work_struct]
# _/home/jiemin/code/GOlang/go/src/struct/work_struct
./work_struct.go:12: person redeclared in this block
    previous declaration at ./work_struct.go:7
./work_struct.go:17: invalid recursive type person
错误: 进程退出代码 2.
```

为什么？

因为编译器不知道使用那个结构，即使是把点一个person结构嵌入到了第二个结构person结构中。编译器也默认只是找到了第一个person结构中的属性。第二个person结构中，编译器认为在main函数中，使用了对person结构进行字面值初始化对a,b进行赋值。提示找不到相应的字段属性。

二：匿名字段和外层字段有相同名称的字段

```
1. package main
2.
3. import (
4.     "fmt"
5. )
6.
7. type A struct {
8.     B    // B结构嵌入到A结构
9.     Name string
10. }
11.
12. type B struct {
13.     Name string
14. }
15.
16. func main() {
17.     a := A{Name: "A", B: B{Name: "B"}} //声明一个结构体a，对A结构中属性进行初始化
18.     fmt.Println(a.Name, a.B.Name)      //输出结构A中Name和结构B中Name的值
19. }
```

输出结构B当中的Name，一般情况下要输出 `fmt.Println(a.Name)` 中的 `a.Name` 是结构A

```
1. type A struct {
2.     B    // B结构嵌入到A结构
3.     Name string
4. }
```

当中的 `Name string`，因为从这个级别来讲，A的级别要高于结构B，所以这时候输出结果就是一个A，那怎么才能娶到结构B当中的 `Name string`，是这样 `fmt.Println(a.Name, a.B.Name)` 中的 `a.B.Name`。就会输出A和B，这就成功取到不同级别当中的同名字段。

```
/home/jiemin/code/GOlang/go/src/method/method [/home/jiemin/code/GOlang/go/src/method]
A B
成功: 进程退出代码 0.
```


如果结构A当中不存在Name string字段

```
1. package main
2.
3. import (
4.     "fmt"
5. )
6.
7. type A struct {
8.     B // B结构嵌入到A结构
9. }
10.
11. type B struct {
12.     Name string
13. }
14.
15. func main() {
16.     a := A{B: B{Name: "B"}} //声明一个结构体a，对A结构中属性进行初始化
17.     fmt.Println(a.Name, a.B.Name) //输出结构A中Name和结构B中Name的值
18. }
```

```
/home/jiemin/code/Golang/go/src/struct/work_struct/work_struct [/home/jiemin/code/Golang/go/src/struct/work_struct]
B B
成功: 进程退出代码 0.
```

输出两个都是B，也就是说当这个结构A，最高级别当中不存在字段的时候，它就会往下一级别找，找下一级别结构当中，会看存在不存在 Name string 字段，在这种情况下fmt.Println(a.Name, a.B.Name)中a.Name等于a.B.Name

还有一种情况

```
1. package main
2.
3. import (
4.     "fmt"
5. )
6.
7. type A struct {
8.     B // B结构嵌入到A结构
9.     C // C结构嵌入到A结构
10. }
11.
12. type B struct {
13.     Name string
14. }
15.
16. type C struct {
17.     Name string
18. }
19.
20. func main() {
21.     a := A{B: B{Name: "B"}, C: C{Name: "C"}} //声明一个结构体a，对A结构中属性进行初始化
```

```
22.     fmt.Println(a.Name, a.B.Name)           //输出结构A中Name和结构B中Name的值
23. }
```

```
/usr/local/go/bin/go build -i [/home/jiemin/code/Golang/go/src/method]
# _/home/jiemin/code/Golang/go/src/method
./method.go:22: ambiguous selector a.Name
```

错误: 进程退出代码 2.

提示有重复名的字段，因为B和C都是作为嵌入式结构，嵌入在结构A当中。所以说B和C结构级别是相同的。编译器没有办法选择要用到的是B当中的Name，还是C当中的Name。

如果把C结构嵌入到B结构中

```
1.  package main
2.
3.  import (
4.      "fmt"
5.  )
6.
7.  type A struct {
8.      B // B结构嵌入到A结构
9.  }
10.
11. type B struct {
12.     C    // C结构嵌入到B结构
13.     Name string
14. }
15.
16. type C struct {
17.     Name string
18. }
19.
20. func main() {
21.     a := A{B: B{Name: "B"}}           //声明一个结构体a，对A结构中属性进行初始
22.     a.B.C.Name = "C"                 // 对嵌入到B结构中的C结构中Name字段进行
23.     fmt.Println(a.Name, a.B.Name, a.B.C.Name) //输出结构A中Name和结构B中Name的值
24. }
```

```
/home/jiemin/code/Golang/go/src/struct/work_struct/work_struct [/home/jiemin/code/Golang/go/src/struct/work_struct]
B B C
成功: 进程退出代码 0.
```