原 golang: 利用unsafe操作未导出变量

陈亦 发表于 3年前 阅读 2556 收藏 13 点赞 8 评论 10

*摘要: unsafe.Pointer其实就是类似C的void *，在golang中是用于各种指针相互转换的桥梁。uintptr是golang的内置类型，是能存储指针的整型，uintptr的底层类型是int，它和unsafe.Pointer可相互转换。uintptr和unsafe.Pointer的区别就是：unsafe.Pointer只是单纯的通用指针类型，用于转换不同类型指针，它不可以参与指针运算；而uintptr是用于指针运算的，GC 不把 uintptr 当指针，也就是说uintptr 无法持有对象，uintptr类型的目标会被回收。golang的unsafe包很强大，基本上很少会去用它...*

看了 @喻恒春 大神的利用unsafe.Pointer来突破私有成员，觉得例子举得不太好。而且不应该简单的放个demo，至少要讲一下其中的原理，让看的童鞋明白所以然。see：http://my.oschina.net/achun/blog/122540

unsafe.Pointer其实就是类似C的void *，在golang中是用于各种指针相互转换的桥梁。uintptr是golang的内置类型，是能存储指针的整型，uintptr的底层类型是int，它和unsafe.Pointer可相互转换。uintptr和unsafe.Pointer的区别就是：unsafe.Pointer只是单纯的通用指针类型，用于转换不同类型指针，它不可以参与指针运算；而uintptr是用于指针运算的，GC 不把 uintptr 当指针，也就是说 uintptr 无法持有对象，uintptr类型的目标会被回收。golang的unsafe包很强大，基本上很少会去用它。它可以像C一样去操作内存，但由于golang不支持直接进行指针运算，所以用起来稍显麻烦。

切入正题。利用unsafe包，可操作私有变量(在golang中称为“未导出变量”，变量名以小写字母开始)，下面是具体例子。

在$GOPATH/src下建立poit包，并在poit下建立子包p，目录结构如下：

$GOPATH/src

----poit

--------p

----------v.go

--------main.go

以下是v.go的代码：

```
package p

import (
    "fmt"
)

type V struct {
    i int32
    j int64
}

func (this V) PutI() {
    fmt.Printf("i=%d\n", this.i)
}

func (this V) PutJ() {
    fmt.Printf("j=%d\n", this.j)
}
```

意图很明显，我是想通过unsafe包来实现对V的成员i和j赋值，然后通过PutI()和PutJ()来打印观察输出结果。

以下是main.go源代码：

```
package main

import (
    "poit/p"
    "unsafe"
)

func main() {
    var v *p.V = new(p.V)
    var i *int32 = (*int32)(unsafe.Pointer(v))
    *i = int32(98)
    var j *int64 = (*int64)(unsafe.Pointer(uintptr(unsafe.Pointer(v)) + uintptr(unsafe.Sizeof(int32(0)))))
    *j = int64(763)
    v.PutI()
    v.PutJ()
}
```

当然会有些限制，比如需要知道结构体V的成员布局，要修改的成员大小以及成员的偏移量。我们的核心思想就是：结构体的成员在内存中的分配是一段连续的内存，结构体中第一个成员的地址就是这个结构体的地址，您也可以认为是相对于这个结构体偏移了0。相同的，这个结构体中的任一成员都可以相对于这个结构体的偏移来计算出它在内存中的绝对地址。

具体来讲解下main方法的实现：

```
var v *p.V = new(p.V)
```

new是golang的内置方法，用来分配一段内存(会按类型的零值来清零)，并返回一个指针。所以v就是类型为p.V的一个指针。

```
var i *int32 = (*int32)(unsafe.Pointer(v))
```

将指针v转成通用指针，再转成int32指针。这里就看到了unsafe.Pointer的作用了，您不能直接将v转成int32类型的指针，那样将会panic。刚才说了v的地址其实就是它的第一个成员的地址，所以这个i就很显然指向了v的成员i，通过给i赋值就相当于给v.i赋值了，但是别忘了i只是个指针，要赋值得解引用。

```
*i = int32(98)
```

现在已经成功的改变了v的私有成员i的值，好开心^_^

但是对于v.j来说，怎么来得到它在内存中的地址呢？其实我们可以获取它相对于v的偏移量(unsafe.Sizeof可以为我们做这个事)，但我上面的代码并没有这样去实现。各位别急，一步步来。

```
var j *int64 = (*int64)(unsafe.Pointer(uintptr(unsafe.Pointer(v)) + uintptr(unsafe.Sizeof(int32(0)))))
```

其实我们已经知道v是有两个成员的，包括i和j，并且在定义中，i位于j的前面，而i是int32类型，也就是说i占4个字节。所以j是相对于v偏移了4个字节。您可以用uintptr(4)或uintptr(unsafe.Sizeof(int32(0)))来做这个事。unsafe.Sizeof方法用来得到一个值应该占用多少个字节空间。注意这里跟C的用法不一样，C是直接传入类型，而golang是传入值。之所以转成uintptr类型是因为需要做指针运算。v的地址加上j相对于v的偏移地址，也就得到了v.j在内存中的绝对地址，别忘了j的类型是int64，所以现在的j就是一个指向v.j的指针，接下来给它赋值：

```
*j = int64(763)
```

好吧，现在貌视一切就绪了，来打印下：

```
v.PutI()
v.PutJ()
```

如果您看到了正确的输出，那恭喜您，您做到了！

但是，别忘了上面的代码其实是有一些问题的，您发现了吗？

在p目录下新建w.go文件，代码如下：

```
package p

import (
```

```
    "fmt"
    "unsafe"
)

type W struct {
    b byte
    i int32
    j int64
}

func init() {
    var w *W = new(W)
    fmt.Printf("size=%d\n", unsafe.Sizeof(*w))
}
```

需要修改main.go的代码吗？不需要，我们只是来测试一下。w.go里定义了一个特殊方法init，它会在导入p包时自动执行，别忘了我们有在main.go里导入p包。每个包都可定义多个init方法，它们会在包被导入时自动执行(在执行main方法前被执行，通常用于初始化工作)，但是，最好在一个包中只定义一个init方法，否则您或许会很难预期它的行为)。我们来看下它的输出：

```
size=16
```

等等，好像跟我们想像的不一致。来手动计算一下：b是byte类型，占1个字节；i是int32类型，占4个字节；j是int64类型，占8个字节，1+4+8=13。这是怎么回事呢？这是因为发生了对齐。在struct中，它的对齐值是它的成员中的最大对齐值。每个成员类型都有它的对齐值，可以用unsafe.Alignof方法来计算，比如unsafe.Alignof(w.b)就可以得到b在w中的对齐值。同理，我们可以计算出w.b的对齐值是1，w.i的对齐值是4，w.j的对齐值也是4。如果您认为w.j的对齐值是8那就错了，所以我们前面的代码能正确执行(试想一下，如果w.j的对齐值是8，那前面的赋值代码就有问题了。也就是说前面的赋值中，如果v.j的对齐值是8，那么v.i跟v.j之间应该有4个字节的填充。所以得到正确的对齐值是很重要的)。对齐值最小是1，这是因为存储单元是以字节为单位。所以b就在w的首地址，而i的对齐值是4，它的存储地址必须是4的倍数，因此，在b和i的中间有3个填充，同理j也需要对齐，但因为i和j之间不需要填充，所以w的Sizeof值应该是13+3=16。如果要通过unsafe来对w的三个私有成员赋值，b的赋值同前，而i的赋值则需要跳过3个字节，也就是计算偏移量的时候多跳过3个字节，同理j的偏移可以通过简单的数学运算就能得到。

比如也可以通过unsafe来灵活取值：

```
package main

import (
    "fmt"
    "unsafe"
)

func main() {
    var b []byte = []byte{'a', 'b', 'c'}
    var c *byte = &b[0]
    fmt.Println(*(*byte)(unsafe.Pointer(uintptr(unsafe.Pointer(c)) + uintptr(1))))
}
```

关于填充，FastCGI协议就用到了。

分类：golang 字数：2014

标签： golang unsafe uintptr Pointer 指针

打赏 点赞 收藏 分享

陈亦

高级程序员 浦东

粉丝 176 | 博文 23 | 码字总数 53187

相关博客

GoLang 强制类型转换:unsafe.Pointer
喻恒春
6809 2

Java Magic. Part 4: sun.misc.Unsafe
dexterman
89 0

【实战Java高并发程序设计】1–Java中对指针的操作：Unsafe类
安小乐
0 0

评论 (10)

Ctrl+Enter 发表评论

喻恒春
1楼 2014/01/17 08:30
很详细啊. 原来 uintptr 可以这样用, 本来我不知道的. 学习到了.

wwwquan
2楼 2014/01/17 09:27

lidashuang
3楼 2014/01/23 11:19
写的很明白

leevainter
4楼 2014/01/23 18:05
知其然而知其所以然！！

Kyli
5楼 2014/02/23 23:43
从来没看过unsafe包，还有好多东西要学

妹子说名字长丁丁长
6楼 2014/06/08 21:25
研究这种用法就是入魔道

mr_franklin
7楼 2015/08/05 10:54
楼主是32位机器上编译的吧？64位机器，int64类型的alignof应该是8。

陈亦
8楼 2015/08/05 13:08
> 引用来自“mr_franklin”的评论
> 楼主是32位机器上编译的吧？64位机器，int64类型的alignof应该是8。

写此文的时候是32位机器上编译的。谢谢指出！

golang_yh
9楼 2015/12/28 00:05
这个特性真不错,而且unsafe.Pointer来突破私有成员这点很重要

咚往咚来

10楼 2016/08/20 17:08

赞~