Golang Defer

# defer的运行过程

Golang defer是用来执行一个func的，它将func函数先放到systemstack中，等主函数执行完毕后再出栈执行func函数。主函数最后返回return。

Defer语句在运行时会生成两条重要的指令，一是runtime.deferproc(SB), 在该函数中，是将fn入栈到systemstack中去，见代码：

而在主函数执行return语句之前，将执行runtime.deferreturn(SB), 在该函数中，将fn出栈并执行。

由此可见，如果多个defer时，将按照先进后出的方式执行。这里可以从汇编的角度来看一下它的内部是如何做的。使用如下命令：

*func main() {*

*a := TestDefer()*

*fmt.Println(a)*

*}*

*func TestDefer() (i int) {*

*defer func() {*

*i++*

*}()*

*defer func() {*

*i++*

*}()*

*i = +10*

*return*

*}*

**go tool compile –S main.go > main.o.txt**

其汇编代码如下：

"".main t=1 size=240 value=0 args=0x0 locals=0x70

//定义main函数，$112-0: 栈大小为112个字节，0个参数

0x0000 00000 (main.go:5) TEXT "".main(SB), $112-0

//线程本地存储(thread local store)的结构体G的地址保存到CX

0x0000 00000 (main.go:5) MOVQ (TLS), CX

//16(CX)存储的是stack base的值，这里是和SP(stack pointer)比较，确保栈分配的大小为112

0x0009 00009 (main.go:5) CMPQ SP, 16(CX)

0x000d 00013 (main.go:5) JLS 229

0x0013 00019 (main.go:5) SUBQ $112, SP

//gc用到的标识

0x0017 00023 (main.go:5) FUNCDATA $0, gclocals·7d2d5fca80364273fb07d5820a76fef4(SB)

0x0017 00023 (main.go:5) FUNCDATA $1, gclocals·5ef976c2593056b9243adf402ae9d952(SB)

0x0017 00023 (main.go:6) PCDATA $0, $0

**//调用TestDefer函数，其中**TestDefer(SB)表示TestDefer的地址

**0x0017 00023 (main.go:6) CALL "".TestDefer(SB)**

0x001c 00028 (main.go:6) MOVQ (SP), BX

0x0020 00032 (main.go:7) MOVQ BX, "".autotmp\_0000+48(SP)

0x0025 00037 (main.go:7) MOVQ $0, BX

0x0027 00039 (main.go:7) MOVQ BX, "".autotmp\_0004+72(SP)

0x002c 00044 (main.go:7) MOVQ BX, "".autotmp\_0004+80(SP)

0x0031 00049 (main.go:7) LEAQ "".autotmp\_0004+72(SP), BX

0x0036 00054 (main.go:7) CMPQ BX, $0

0x003a 00058 (main.go:7) JEQ $1, 222

0x0040 00064 (main.go:7) MOVQ $1, "".autotmp\_0001+96(SP)

0x0049 00073 (main.go:7) MOVQ $1, "".autotmp\_0001+104(SP)

0x0052 00082 (main.go:7) MOVQ BX, "".autotmp\_0001+88(SP)

//创建int型变量，地址保存到BX

**0x0057 00087 (main.go:7) LEAQ type.int(SB), BX**

0x005e 00094 (main.go:7) MOVQ BX, (SP)

//将**TestDefer**返回值赋值给int变量

**0x0062 00098 (main.go:7) LEAQ "".autotmp\_0000+48(SP), BX**

0x0067 00103 (main.go:7) MOVQ BX, 8(SP)

0x006c 00108 (main.go:7) MOVQ $0, 16(SP)

0x0075 00117 (main.go:7) PCDATA $0, $1

0x0075 00117 (main.go:7) CALL runtime.convT2E(SB)

0x007a 00122 (main.go:7) MOVQ 24(SP), CX

0x007f 00127 (main.go:7) MOVQ 32(SP), AX

0x0084 00132 (main.go:7) MOVQ "".autotmp\_0001+88(SP), BX

0x0089 00137 (main.go:7) MOVQ CX, "".autotmp\_0005+56(SP)

0x008e 00142 (main.go:7) MOVQ CX, (BX)

0x0091 00145 (main.go:7) MOVQ AX, "".autotmp\_0005+64(SP)

0x0096 00150 (main.go:7) CMPB runtime.writeBarrier(SB), $0

0x009d 00157 (main.go:7) JNE $0, 202

0x009f 00159 (main.go:7) MOVQ AX, 8(BX)

0x00a3 00163 (main.go:7) MOVQ "".autotmp\_0001+88(SP), BX

0x00a8 00168 (main.go:7) MOVQ BX, (SP)

0x00ac 00172 (main.go:7) MOVQ "".autotmp\_0001+96(SP), BX

0x00b1 00177 (main.go:7) MOVQ BX, 8(SP)

0x00b6 00182 (main.go:7) MOVQ "".autotmp\_0001+104(SP), BX

0x00bb 00187 (main.go:7) MOVQ BX, 16(SP)

0x00c0 00192 (main.go:7) PCDATA $0, $2

0x00c0 00192 (main.go:7) CALL fmt.Println(SB)

0x00c5 00197 (main.go:8) ADDQ $112, SP

0x00c9 00201 (main.go:8) RET

0x00ca 00202 (main.go:7) LEAQ 8(BX), R8

0x00ce 00206 (main.go:7) MOVQ R8, (SP)

0x00d2 00210 (main.go:7) MOVQ AX, 8(SP)

0x00d7 00215 (main.go:7) PCDATA $0, $1

0x00d7 00215 (main.go:7) CALL runtime.writebarrierptr(SB)

0x00dc 00220 (main.go:7) JMP 163

0x00de 00222 (main.go:7) MOVL AX, (BX)

0x00e0 00224 (main.go:7) JMP 64

…

"".TestDefer t=1 size=176 value=0 args=0x8 locals=0x18

0x0000 00000 (main.go:10) TEXT "".TestDefer(SB), $24-8

0x0000 00000 (main.go:10) MOVQ (TLS), CX

0x0009 00009 (main.go:10) CMPQ SP, 16(CX)

0x000d 00013 (main.go:10) JLS 161

0x0013 00019 (main.go:10) SUBQ $24, SP

0x0017 00023 (main.go:10) FUNCDATA $0, gclocals·5184031d3a32a42d85027f073f873668(SB)

0x0017 00023 (main.go:10) FUNCDATA $1, gclocals·33cdeccccebe80329f1fdbee7f5874cb(SB)

**//初始化函数参数i为0，i+32(FP)表示函数的第4个参数i（64bit）**

**0x0017 00023 (main.go:10) MOVQ $0, "".i+32(FP)**

0x0020 00032 (main.go:10) MOVQ $0, "".i+32(FP)

0x0029 00041 (main.go:13) LEAQ "".i+32(FP), BX

0x002e 00046 (main.go:13) MOVQ BX, 16(SP)

//runtime.deferproc的第一个参数size=8，将第一个参数入栈

0x0033 00051 (main.go:13) MOVL $8, (SP)

// runtime.deferproc的第二个参数fn的地址，将第二个参数入栈

**0x003a 00058 (main.go:13) LEAQ "".TestDefer.func1·f(SB), AX**

0x0041 00065 (main.go:13) MOVQ AX, 8(SP)

0x0046 00070 (main.go:13) PCDATA $0, $0

//调用runtime.deferproc, 将第一个fn入栈到systemstack中

0x0046 00070 (main.go:13) CALL runtime.deferproc(SB)

0x004b 00075 (main.go:13) CMPL AX, $0

0x004e 00078 (main.go:13) JNE $1, 150

//以下是重复将第二个fn加入到systemstack中

0x0050 00080 (main.go:17) LEAQ "".i+32(FP), BX

0x0055 00085 (main.go:17) MOVQ BX, 16(SP)

0x005a 00090 (main.go:17) MOVL $8, (SP)

**0x0061 00097 (main.go:17) LEAQ "".TestDefer.func2·f(SB), AX**

0x0068 00104 (main.go:17) MOVQ AX, 8(SP)

0x006d 00109 (main.go:17) PCDATA $0, $0

0x006d 00109 (main.go:17) CALL runtime.deferproc(SB)

0x0072 00114 (main.go:17) CMPL AX, $0

0x0075 00117 (main.go:17) JNE $1, 139

**//将i参数赋值为10**

**0x0077 00119 (main.go:18) MOVQ $10, "".i+32(FP)**

0x0080 00128 (main.go:19) PCDATA $0, $0

0x0080 00128 (main.go:19) XCHGL AX, AX

//执行runtime.deferreturn, 将fn出栈并执行

0x0081 00129 (main.go:19) CALL runtime.deferreturn(SB)

0x0086 00134 (main.go:19) ADDQ $24, SP

0x008a 00138 (main.go:19) RET

0x008b 00139 (main.go:17) PCDATA $0, $0

0x008b 00139 (main.go:17) XCHGL AX, AX

0x008c 00140 (main.go:17) CALL runtime.deferreturn(SB)

0x0091 00145 (main.go:17) ADDQ $24, SP

0x0095 00149 (main.go:17) RET

0x0096 00150 (main.go:13) PCDATA $0, $0

0x0096 00150 (main.go:13) XCHGL AX, AX

0x0097 00151 (main.go:13) CALL runtime.deferreturn(SB)

0x009c 00156 (main.go:13) ADDQ $24, SP

0x00a0 00160 (main.go:13) RET

0x00a1 00161 (main.go:13) NOP

//栈不足，扩栈操作

0x00a1 00161 (main.go:10) CALL runtime.morestack\_noctxt(SB)

0x00a6 00166 (main.go:10) JMP 0

…

"".TestDefer.func1 t=1 size=16 value=0 args=0x8 locals=0x0

0x0000 00000 (main.go:11) TEXT "".TestDefer.func1(SB), $0-8

0x0000 00000 (main.go:11) NOP

0x0000 00000 (main.go:11) NOP

**//参数i的地址赋值给CX**

**0x0000 00000 (main.go:11) MOVQ "".&i+8(FP), CX**

0x0005 00005 (main.go:11) FUNCDATA $0, gclocals·87d20ce1b58390b294df80b886db78bf(SB)

0x0005 00005 (main.go:11) FUNCDATA $1, gclocals·33cdeccccebe80329f1fdbee7f5874cb(SB)

0x0005 00005 (main.go:12) MOVQ (CX), BP

**//加1**

**0x0008 00008 (main.go:12) INCQ BP**

0x000b 00011 (main.go:12) MOVQ BP, (CX)

0x000e 00014 (main.go:12) NOP

0x000e 00014 (main.go:13) RET

0x0000 48 8b 4c 24 08 48 8b 29 48 ff c5 48 89 29 c3 cc H.L$.H.)H..H.)..

"".TestDefer.func2 t=1 size=16 value=0 args=0x8 locals=0x0

0x0000 00000 (main.go:15) TEXT "".TestDefer.func2(SB), $0-8

0x0000 00000 (main.go:15) NOP

0x0000 00000 (main.go:15) NOP

0x0000 00000 (main.go:15) MOVQ "".&i+8(FP), CX

0x0005 00005 (main.go:15) FUNCDATA $0, gclocals·87d20ce1b58390b294df80b886db78bf(SB)

0x0005 00005 (main.go:15) FUNCDATA $1, gclocals·33cdeccccebe80329f1fdbee7f5874cb(SB)

0x0005 00005 (main.go:16) MOVQ (CX), BP

0x0008 00008 (main.go:16) INCQ BP

0x000b 00011 (main.go:16) MOVQ BP, (CX)

0x000e 00014 (main.go:16) NOP

0x000e 00014 (main.go:17) RET

0x0000 48 8b 4c 24 08 48 8b 29 48 ff c5 48 89 29 c3 cc H.L$.H.)H..H.)..

…

# defer的使用

1. 对于需要回收或者关闭的资源，可以在创建资源的代码之后使用defer语句来保证关闭。如：
2. 如果要捕获panic, 那么可以在可能出现panic的代码前使用defer语句，这样如果出现了panic， 主函数立即停止执行，但systemstack栈中的defer 函数依然会展开执行，如果在defer函数中有recover，那么就可以阻止panic继续向上传播。如：

func TestDefer() (i int) {

defer func() {

if r=recover(); r!= nil {

...

}

}()

//可能出现panic

…

return

}