[go-nuts] Variable declaration inside loop 792 views

Péter Szilágyi May 20, 2013, 7:26:16 AM

to Tobia, golang-nuts

Since you're declaring the x inside a block, by the language specs those 5 x-es will be completely different. The optimization is actually noticing that you can reuse it (not in this case).

On Sun, May 19, 2013 at 8:05 PM, Tobia <tobia.c...@gmail.com> wrote:

> Hi
>
> I was playing around with closures and I noticed something curious.
>
> First, the obvious behavior. This prints five 5, as expected*:
>
> ```
> for i := 0; i < 5; i++ {
>     go func() {
>         time.Sleep(time.Millisecond)
>         fmt.Println(i)
>     }()
> }
> time.Sleep(2 * time.Millisecond)
> ```
>
> http://play.golang.org/p/WlCESYk7UP
>
> (* There is only one "i" variable in memory, the closure merely captures it. By the time the five goroutines stop sleeping, the single "i" already contains 5.)
>
> This prints the numbers from 0 to 4 (in unpredictable order), again as expected:
>
> ```
> for i := 0; i < 5; i++ {
>     go func(x int) {
>         time.Sleep(time.Millisecond)
>         fmt.Println(x)
>     }(i)
> }
> time.Sleep(2 * time.Millisecond)
> ```
>
> http://play.golang.org/p/Hsf9yib9Cc
>
> But the following surprised me. I expected it to "fail" (print five 5), instead it prints from 0 to 4:
>
> ```
> for i := 0; i < 5; i++ {
>     x := i
>     go func() {
>         time.Sleep(time.Millisecond)
>         fmt.Println(x)
>     }()
> }
> time.Sleep(2 * time.Millisecond)
> ```
>
> http://play.golang.org/p/CE6q9Bs9xi
>
> Why does a single (lexical) variable declaration cause many memory allocations? Would it always do so, or is this a case of the compiler being "smart" and noticing that the variable is being captured?
> More importantly, is this specified in the reference docs? Can it be relied upon?

JavaScript - Avoiding variable declaration inside a loop

JavaScript is a language with a dynamic garbage collection. Memory allocated even if inaccessible, will not be given free before the garbage collector deallocates it.

If a variable is declared inside a loop, JavaScript will allocate fresh memory for it in each iteration, even if older allocations will still consume memory. While this might be acceptable for variables of scalar data types, it may allocate a lot of memory for SCFile variables, or huge arrays or objects.

Unrecommended implementation

The following implementation, in which a variable is declared inside a loop, is not recommended.

```
var arr = new Array();

arr = [ "IM10001", "IM10002", "IM10003", "IM10004" ];

for(i=0;i<arr.length;i++)

{

var file = new SCFile("probsummary");

file.doSelect('name="'+arr[i]+'"');

// do something with file

}
```

Recommended implementation

The following implementation, in which a variable is declared before a loop, is recommended.

```
var arr = new Array();

arr = [ "IM10001", "IM10002", "IM10003", "IM10004" ];

var file = new SCFile("probsummary");

for(i=0;i<arr.length;i++)

{

file.doSelect('name="'+arr[i]+'"');

// do something with file

}
```

c++ for循环中的定义的局部变量地址总是不变

原创 _DCG_ 于 2022-12-29 10:14:32 发布 148 收藏 版权

在开发过程中发现一个问题，当在一个for循环中定义一个局部变量，打印该局部变量的地址，地址总是相同的，地址相同的原因是这是一个局部变量，for循环每次循环执行完毕总是会释放该局部变量，从 堆栈 的角度看总是会先释放堆栈然后压栈，所以才会造成每次的局部变量的值总是一样。

```
for(int i = 1; i < 5; i++)
{
    int a;
    a = i+10;
    printf("a addr = %p\n", &a); //这里的a的地址总是一样的
}

汇编如下
fun():
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-4], 1
        jmp     .L2
.L3:
        mov     eax, DWORD PTR [rbp-4]
        add     eax, 10
        mov     DWORD PTR [rbp-8], eax
        add     DWORD PTR [rbp-4], 1
.L2:
        cmp     DWORD PTR [rbp-4], 5
        jle     .L3
        nop
        pop     rbp
        ret
```

编程 C（编程语言） 内存（RAM） C++ 变量

for循环中局部变量地址为什么不变?

for(int i=0;i<10;i++) {

int a=i;

std::cout<<&a;

}

匿名用户

10 人赞同了该回答

“每次都使用相同的内存地址”和“每次都使用不同的内存地址”都是对的。

- “每次都使用不同的内存地址”是对的吗?

你觉得，每次都应当申请一遍内存空间并得到不同的内存地址，是吧？那样做当然也是正确的实现方法，但效率上讲就比 只申请一遍 差了。如果有编译器作者宁愿在这一点上忍受，编译出的程序有一个较差的效率的话，每次都申请一个不同的内存地址也可以。

- 为什么你遇到（也最常见）的“每次都使用相同的内存地址”也是对的?

从语言上讲，

在for循环内定义的变量，生命周期仅限当轮循环，所以每一轮循环中是不同的对象a。而更重要的是多个a对象根本不会同时存在，前一轮迭代结束后前一轮的a就不存在了，下一轮才有新的a（逻辑上如此）。所以根本不会有任何

一个程序可以合法地对不同轮次中a的地址做比较操作（使用已被释放的对象的地址是非法的）。

“所有的a使用不同的地址，或是所有的a使用同一个地址？”**这个选择，这没有影响对a的合法操作（在a的生命周期内对其访问），影响的是在a的生命周期之后（对象a已经没了）还试图对它或者它的地址访问（比方说我们现在人为比较他们的地址、或者写一个程序试图比较他们）。**

而在对象的生命周期之后还试图对它访问是未定义的行为，未定义也就意味着结果是什么都可以。

你现在人为比较每一轮循环中a的地址，从语言上对程序自身来讲是没有意义的，只是开了上帝视角才觉得奇怪。

- 为什么多数选择了“每次都使用相同的内存地址”而不是“每次都使用不同的内存地址”

既然同一个循环中，同时存在的只会有一个a（而不会所有都一直存在）。为什么不把前一个a的空间再拿来重复利用作为下一次a的空间呢？**这样可以节省一笔申请空间的开销。**继而在一个for循环中，所有轮次的a使用了同一个空间。前文已经说过**这个优化没有影响对a的合法操作。**

优化只要不影响程序的合法操作，就是正确的。优化对于程序的未定义行为，可以任意改变（极端情况甚至于把未定义操作改成关机都符合标准，当然，没有编译器作者真的这么干）。

包括把int替换成类对象，它也只是在同一个地址上反复执行构造函数和析构函数而已。

```
struct t{
    t() {
        cout << "constructed" << endl;
    }
    ~t() {
        cout << "destroyed" << endl;
    }
};

int main() {
    for (int i = 0; i < 10; i++) {
        t a;
        cout << &a << endl;
    }
    return 0;
}
```

```
constructed
0x7fffd33a057b
destroyed
constructed
0x7fffd33a057b
destroyed
constructed
0x7fffd33a057b
destroyed
constructed
0x7fffd33a057b
destroyed
constructed
0x7fffd33a057b
```

```
destroyed
constructed
0x7fffd33a057b
destroyed
constructed
0x7fffd33a057b
destroyed
constructed
0x7fffd33a057b
destroyed
constructed
0x7fffd33a057b
destroyed
constructed
0x7fffd33a057b
destroyed
```

编辑于 2020-03-09 16:38

匿名用户

编译一下看看就知道了。

为了简单一点，可以把题目中的输出语句去掉：

```
// for-loop.cc
// #include <iostream>
// using std::cout;
// using std::endl;

int main() {
```

```
    for (int i = 0; i < 10; i++) {
        int a = i;
        // cout << &a << ' ' << &i << endl;
    }
    return 0;
}
```

汇编:

```
g++ -S ./for-loop.cc
```

结果:

```
        .file   "for-loop.cc"
        .text
        .def    ___main;        .scl    2;      .type   32;     .endef
        .globl  _main
        .def    _main;  .scl    2;      .type   32;     .endef
_main:
LFB0:
        .cfi_startproc
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        andl    $-16, %esp
        subl    $16, %esp
        call    ___main
        movl    $0, 12(%esp)
```

```
; here is the for loop
L3:
        cmpl    $9, 12(%esp)     ; compare
        jg      L2               ; jump if greater than 9
        movl    12(%esp), %eax   ;
        movl    %eax, 8(%esp)    ; local variable assinment
        addl    $1, 12(%esp)     ; increament
        jmp     L3
L2:
        movl    $0, %eax
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
LFE0:
        .ident  "GCC: (MinGW.org GCC-8.2.0-5) 8.2.0"
```

赋值用的都是 `8(%esp)`，地址当然不会变。

至于为什么。。。。因为没有换地址的必要。。。

其实我不太看得懂汇编，不过这么简单的程序能猜个大概啦。

发布于 2020-03-08 20:39

C/C++ 循环内还是循环外定义变量更好?

问:

```
// 方法 A:
Widget w;
for (int i = 0; i < n; ++i) {
    w = 取决于 i 的某个值;
}

// 方法 B:
for (int i = 0; i < n; ++i) {
    Widget w(取决于 i 的某个值);
}
```

Effective C++ 条款 26 有一段话大概是：当一个赋值成本低于一组构造+析构成本，定义在循环外（即A）比较高效，否则定义在循环内（即B）比较好。

我的问题是：哪些情况 赋值成本低于一组构造+析构成本?

答:

我觉得这个更应该思考两种情况下，变量的作用域，这个更为关键。A情况，w的生存周期会在循环外继续存在，而B不会，不要过早的考虑优化的问题，现在的编译器比你想象的还要聪明。

而对于C++来说，更崇尚RAII(Resource Acquisition Is Initialization)，**当你要使用时，就直接再声明定义。**比如在A情况下，如果我在赋值w前有一段是检测逻辑，如果不满足，我就直接抛出异常，那么你就需要承担构造w的代价，即使它没有用到，如下所示:

```
Widget w;
for (int i = 0; i < n; ++i) {
    if(some condition)
```

```
    {
        throw std::exception("haha");
    }
    w = 取决于 i 的某个值;
}
```

然而，对于在使用时再直接声明并定义就不会有这样的问题：

```
for (int i = 0; i < n; ++i) {
    if(some condition)
    {
       throw std::exception("haha");
    }
    Widget w(i);
}
```

这样，即使抛出异常，也不会承担构造无谓的构造w的代码。

所以，在编写C++代码时，应该忘记C89的先声明，后面再来定义的做法，而是考虑RAII。

原文链接：循环内还是循环外定义变量？ - 蓝色的回答 - 知乎 https://www.zhihu.com/question/36125544/answer/66031885