# C/C++ 的严格别名规则

孔俊

2020-01-24

# Contents

C/C++ 中的变量占有一块内存，这时这个变量就是这块内存的别名，指针也可以指向内存，因此同一块内存可能会有多个别名。

```
int main()
{
    int i = 0;
    int *ip = &i;
}
```

其中`i`和`ip`是同一块内存，都是它的别名。

内存别名的存在会影响编译器生成的代码的行为。

考虑以下代码块（来自 *CSAPP* 5.1 节）：

```
void twiddle1(long *xp, long *yp)
{
    *xp += *yp;
    *xp += yp;
}

void tiwddle2(long *xp, long *yp)
{
    *xp *= 2 * *yp;
}
```

这两个函数的功能看起来是相同的，但其实不然。加入，`xp`和`yp`指向同一块内存，`twiddle1()`将`*xp`写为原来的四倍，而`twiddle2()`将`xp`写为原来的两倍。

编译器在进行优化时，要确保优化是安全的，即优化的程序和未优化的程序行为是一致的。在上面的例子中，编译器无法判断`xp`和`yp`是同一块内存的别名（指向同一块内存），只能保守地认为两个指针指向同一块内存，因此`twiddle()`要老老实实的进行两次`+=`。

而当指针指向的对象类型不同时，编译器可以放心地认为指针指向不同的内存，互相不为别名，这就是所谓的 *strict aliasing*：不同类型的指针指向不同的内存块。在这种情况下，编译器可以使用激进的优化策略。

但是 C/C++ 经常使用类型转换和指针直接操作内存，有时就会破坏 strict aliasing 规则，导致未定义行为。

```
#include <stdio.h>
int global = 2;
int test_strict_aliasing(int *arg)
{
    global = 1;
    *reinterpret_cast<float*>(arg) = 0;
    return global;
}

int
main()
{
    printf("%d\n", test_strict_aliasing(&global));
    printf("global: %d\n", global);
    return 0;
}

--------------------------------------------------------------------------
g++ -Wall -Wstrict-aliasing=1 -o strict-aliasing strict-aliasing.cpp
--------------------------------------------------------------------------
0
global: 0

--------------------------------------------------------------------------

g++ -Wall -Wstrict-aliasing=1 -O2 -o strict-aliasing strict-aliasing.
   cpp
--------------------------------------------------------------------------

0
global: 1
```

在上面的程序中`reinterpret_cast<float *>(arg)`创建了一个临时的`float *`指针，并且指向的内存块和`int *`类型指针`arg`相同，这是非法的内存别名，这反了 strict aliasing 规则，产生未定义行为。在 GCC O2 以下的优化级别，不假设 strict aliasing，在 O2 及以上优化级别假设 strict aliasing，因此两编译选项下程序的行为不同。

```
g++ -Wall -Wstrict-aliasing=1 -O2 -o strict-aliasing strict-aliasing.cpp 生成的汇编代码
  401170:   c7 05 aa 2e 00 00 01    movl   $0x1,0x2eaa(%rip)        # 404024 <global>
  401177:   00 00 00
  40117a:   b8 01 00 00 00          mov    $0x1,%eax
  40117f:   c7 07 00 00 00 00       movl   $0x0,(%rdi)
  401185:   c3                      retq
  401186:   66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
  40118d:   00 00 00

g++ -Wall -Wstrict-aliasing=1 -o strict-aliasing strict-aliasing.cpp 生成的汇编代码
  401126:   55                      push   %rbp
  401127:   48 89 e5                mov    %rsp,%rbp
  40112a:   48 89 7d f8             mov    %rdi,-0x8(%rbp)
  40112e:   c7 05 ec 2e 00 00 01    movl   $0x1,0x2eec(%rip)        # 404024 <global>
  401135:   00 00 00
  401138:   48 8b 45 f8             mov    -0x8(%rbp),%rax
  40113c:   66 0f ef c0             pxor   %xmm0,%xmm0
  401140:   f3 0f 11 00             movss  %xmm0,(%rax)
  401144:   8b 05 da 2e 00 00       mov    0x2eda(%rip),%eax        # 404024 <global>
  40114a:   5d                      pop    %rbp
  40114b:   c3                      retq
```

可以发现 O2 编译选项下生成的代码更少，性能更强。O2 下的代码直接将数字1当成返回值返回，而不是将`global`当成返回值返回，因此程序出现了错误的行为。

将指针转型为不相容（*imcompatible* 的指针类型，并进行读写违反了 strict aliasing，是严重的未定义行为。如果确实需要进行 *type punning*，必须将指针转换为相容的指针类型, 即通过合法的内存别名访问内存。C/C++ 标准规定了以下类型的指针是合法的别名：

1. 指针指向的类型相差`unsigned`、`signed`、**`volatile`**
2. **`char`** *和**`void`** *是所有指针的合法别名
3. 指向包含指针指向对象类型的聚合类或 union 的指针是合法别名

## 1 使用 `char *` 创建合法别名

**`char`**在 C/C++ 中实际上是字节类型，使用非常频繁，因此在标准中为它开了“后门”。

以下程序避免了未定义行为：

```
int test_strict_aliasing(int *arg)
{
    global = 1;
    *reinterpret_cast<float*>(arg) = 0;
    *reinterpret_cast<char *>(arg) = 0;
    *(reinterpret_cast<char *>(arg) + 1) = 0;
    *(reinterpret_cast<char *>(arg) + 2) = 0;
    *(reinterpret_cast<char *>(arg) + 3) = 0;
    return global;
}
```

## 2 使用 union *创建合法别名

```
union int2float
{
    int i;
    float f;
};
int test_strict_aliasing(int *arg)
{
    global = 1;
    *reinterpret_cast<float*>(arg) = 0;
    return global;
}
```

上面的程序定义了一个包含我们要修改的指针指向的对象类型（int）的联合体，然后将arg转型为union int2float *再通过 union 修改*arg。这种方法是 GCC 推荐的方法。

## 3 禁止编译器假设 strict aliasing

上面提到，在 O2 及以上优化等级才会 假设 strict aliasing ，有大量的 C/C++ 程序必须违反 strict aliasing，因此 GCC 提供了-fstrict-aliasing和-fno-strict-aliasing选项开启和关闭 strict aliasing。

GCC 还提供了-Wstrict-aliasing来警告违反 strict-aliaisng 的行为，这个选项被-Wall默认开启。GCC 虽然提供了警告选项，但该功能工作的并不好。

## 4 总结

GCC 对 strict aliasing 的处理让很多 C/C++ 程序不加上-no-strict-aliasing选项就无法正确运行，这引起了很多人的愤怒，Linus 还专门喷过 GCC，但是经过我通过实验发现，Clang 也和 GCC 一样烂。

总之，要避免直接对指针进行转型并读写，这是未定义的!!!