MACO

发布 *1*

Darwin Yuan

2020 年 09 月 13 日

Contents

注解： MACO 是一个图灵完备的 C/C++ macro 库

C/C++ 的宏一直被人诟病：难以理解，容易出错，名字悄然替换带来的麻烦，等等。

的确，只要 C++ 有正式的语法手段可以解决你的问题，你就应该避免使用宏。C++11 之后，越来越多的设施，比如 constexpr ，模版变参支持，让宏的用武之地越来越小。

但是，在 C++ 23 支持编译时反射这种强大的元编程能力之前，宏依然有着不可替代的作用。但是，C/C++ 宏对于需要它的程序员来讲，能力看起来又显得太弱。

不过，C/C++ macro 的能力或许比很多程序员所知的，要强大许多。事实上，在一定程度上，它甚至可以做到 **图灵完备**。

而这种能力，可以让程序员在不得不使用宏时，可以拥有更为强大的能力，解决更为棘手的问题。

我们来看看，通过 maco ，如何可以做到图灵完备的计算。

CHAPTER 1

1. __MACO_if

```
auto result = __MACO_if(__MACO_eq(3, 2))(
  10, // return 10 if 3 == 2
  20  // return 20 otherwise
);

ASSERT(result == 20);
```

CHAPTER 2

2. natural comparison

```
ASSERT(__MACO_eq(10, 10));
ASSERT(!__MACO_eq(9, 10));

ASSERT(__MACO_ne(9, 10));
ASSERT(!__MACO_ne(9, 9));

ASSERT(__MACO_gt(10, 9));
ASSERT(!__MACO_gt(9, 9));
ASSERT(!__MACO_gt(9, 10));

ASSERT(__MACO_gte(10, 9));
ASSERT(__MACO_gte(9, 9));
ASSERT(!__MACO_gte(9, 10));
```

CHAPTER 3

3. __MACO_simple_repeat_from_0

```
#define __f(n) case n: return __MACO_succ(n);

switch(number) {
  __MACO_simple_repeat_from_0(3, __f)
  default:  return 0;
}
```

其被展开后，则变为：

```
switch(number) {
  case 0: return 1;
  case 1: return 2;
  case 2: return 3;
  default:  return 0;
}
```

而它的实现，背后则是使用了递归：

```
__MACO_while(n)(next, (__MACO_prev(n), f))   f(n)
```

你可以清晰的看到其中的逻辑：尾部的 `f(n)` 是对用户指定的宏 `f` 以 `n` 为参数展开。而 `__MACO_while(n)` 会根据 `n > 0` 是否成立，决定继续递归调用 `next(n-1, f)`，或终止递归。

CHAPTER 4

4. __MACO_make_index_seq

```
int a[] = { __MACO_make_index_seq(4) };

ASSERT(sizeof(a)/sizeof(a[0]) == 4);
ASSERT(a[0] == 0);
ASSERT(a[1] == 1);
ASSERT(a[2] == 2);
ASSERT(a[3] == 3);
```

CHAPTER 5

5. __MACO_make_token_seq

```
int a_0 = 10;
int a_1 = 11;
int a_2 = 12;

int a[] = { __MACO_make_token_seq(a_, 3) };

ASSERT(sizeof(a)/sizeof(a[0]) == 3);
ASSERT(a[0] == a_0);
ASSERT(a[1] == a_1);
ASSERT(a[2] == a_2);
```

CHAPTER 6

6. __MACO_map

通过递归机制，我们就可以实现 map ，其原型为 __MACO_map(f, ...) ，通过它就可以对列表中每一个元素通过用户指定的宏 f 进行展开。比如：

```
#define __f(x) int x;

struct Foo { __MACO_map(__f, a, b, c) };
// struct Foo { int a; int b; int c; };
```

__MACO_map 是一个强大的，应用非常广泛的宏。其中最著名的应用：是生成结构体的反射信息。

CHAPTER 7

7. __MACO_map_i

`__MACO_map_i(f, ...)` 会给每个 `f` 传递其索引。比如：

```
#define __f(n, x) , n + x

auto array[] = { 0 __MACO_map_i(__f, 1, 2, 3) };

ASSERT(array[1] == 1);
ASSERT(array[2] == 3);
ASSERT(array[3] == 5);
```

CHAPTER 8

8. __MACO_num_of_args

求一个宏展开参数的个数。比如：

```
ASSERT(__MACO_num_of_args() == 0);
ASSERT(__MACO_num_of_args(a) == 1);
ASSERT(__MACO_num_of_args(a, b) == 2);
ASSERT(__MACO_num_of_args(a, b, c, d, e) == 5);
```

这个宏对于处理变参问题极为有用。

还有很多其它的宏，具体请参见：moco github 。

关于具体的宏展开原理，请参考 宏展开。

最后，再次提醒：不要滥用宏!!! 优先使用 C++ 语法元素解决你的问题，除非宏是你解决同等复杂度问题的最后手段。