Introduction to ObjC

陈嘉杰

2018 年 10 月

啥是 ObjC

ObjC 是一门动态语言

ObjC 就是 ...Objective-C 鸭

Origin

调用函数全靠名字（selector）

```
1  // Example 1
2  NSString *harryChen = [[NSString alloc] init];
3  [harryChen isHarry];
4  // is equivalent to:
5  id harryChen =
   ↪  objc_msgSend(objc_msgSend(objc_getClass("NSString"),
   ↪  @selector(alloc)), @selector(init));
6  objc_msgSend(harryChen, @selector(isHarry));
7  // further:
8  id harryChen =
   ↪  objc_msgSend(objc_msgSend(objc_getClass("NSString"),
   ↪  sel_getUid("alloc")), sel_getUid("init"));
9  objc_msgSend(harryChen, sel_getUid("isHarry"));
```

随时给别的类添加一个方法

```
1  // Monkey patching
2  @implementation NSString
3  - (BOOL) isHarry {
4      return YES;
5  }
6  // is equivalent to:
7  BOOL isHarryIMP(id self, SEL _cmd) {
8      return YES;
9  }
10 class_addMethod(objc_getClass("NSString"),
   ↪  sel_registerName("isHarry"), (IMP) isHarryIMP,
   ↪  "v@:");
```

所以 objc_msgSend 到底是什么（1）

```
1  /********************************************************************
2   *
3   * id objc_msgSend(id self, SEL        _cmd,...);
4   * IMP objc_msgLookup(id self, SEL _cmd, ...);
5   *
6   * objc_msgLookup ABI:
7   * IMP returned in r11
8   * Forwarding returned in Z flag
9   * r10 reserved for our use but not used
10  *
11
    ↪  ********************************************************************
12 ...
13          GetIsaFast NORMAL // r10 = self->isa
14          CacheLookup NORMAL, CALL // calls IMP on
            ↪  success
15 ...
16          jmp         __objc_msgSend_uncached
```

所以 objc_msgSend 到底是什么（2）

```
1  /********************************************************************
2   *
3   * _objc_msgSend_uncached
4   * _objc_msgSend_stret_uncached
5   * _objc_msgLookup_uncached
6   * _objc_msgLookup_stret_uncached
7   *
8   * The uncached method lookup.
9   *
10
    ↪  ********************************************************************
11 ...
12          MethodTableLookup NORMAL // r11 = IMP
13          jmp         *%r11 // goto *imp
14 ...
15          MethodTableLookup STRET // r11 = IMP
16          jmp         *%r11 // goto *imp
```

首先要知道这是个什么类型的对象

```
1  .macro GetIsaFast
2  .if $0 != STRET
3          testb       $$1, %a1b
4          PN
5          jnz         LGetIsaSlow_f
6          movq        $$0x00007ffffffffff8, %r10
7          andq        (%a1), %r10
```

内存里对象是怎么样的?

```
1  struct objc_object {
2  private:
3    isa_t isa;
4    // ...
5  };
6  union isa_t {
7    Class cls;
8    uintptr_t bits;
9    // ...
10 #define ISA_MASK 0x00007ffffffffff8ULL
11   struct {
12     uintptr_t nonpointer : 1;
13     uintptr_t has_assoc : 1;
14     // ...
15   };
16 };
```

程序里是如何储存这些信息呢?

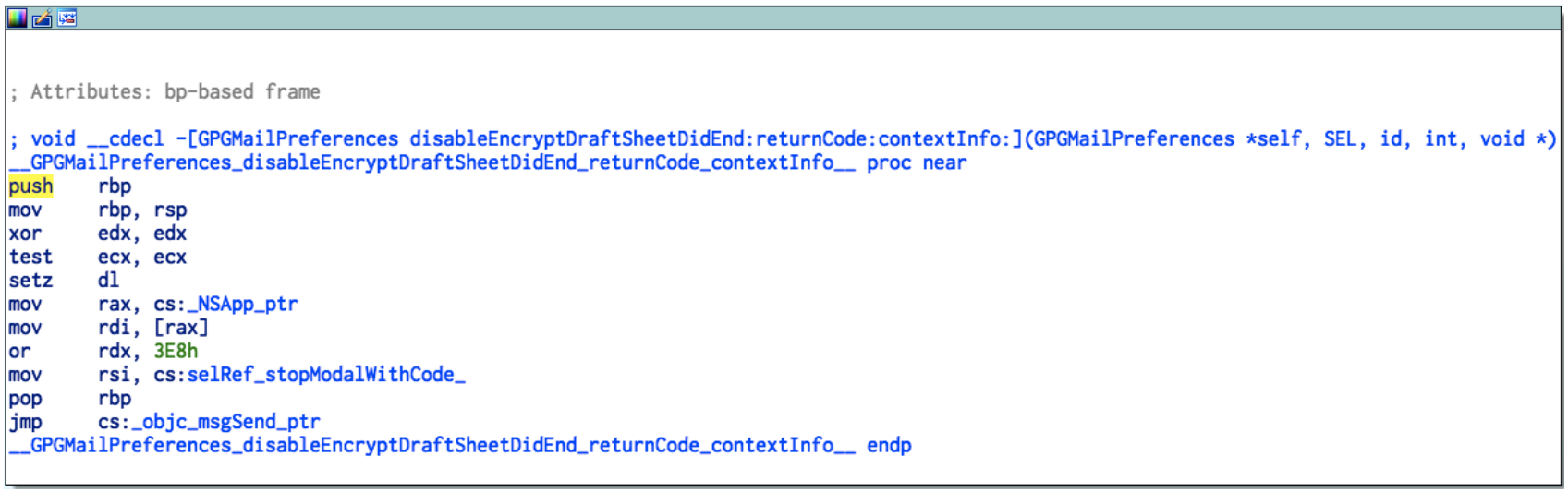
$ otool -ov /Applications/Maps.app/Contents/MacOS/Maps

```
Contents of (__DATA,__objc_classlist) section
0000000100275398 0x1002ec2e8
 ↪  _OBJC_CLASS_$_DirectionsPlan
        isa 0x1002ec310
         ↪  _OBJC_METACLASS_$_DirectionsPlan
superclass 0x0 _OBJC_CLASS_$_PBCodable
      data 0x10027cb18 (struct class_ro_t *)
        name 0x1001e07e0 DirectionsPlan
 baseMethods 0x10027cb60 (struct method_list_t *)
        count 82
         name 0x100207198 hasRouteRequestStorage
        types 0x10023e4fb c16@0:8
          imp -[DirectionsPlan
           ↪  hasRouteRequestStorage]
```

Type Encodings

举个例子

```
1  name 0x589df
    ↪  disableEncryptDraftSheetDidEnd:returnCode:contextInfo:
2  types 0x61d3e v36@0:8@16i24^v28
3    imp 0xc4b4
4
5  - (void)disableEncryptDraftSheetDidEnd:(NSWindow
    ↪  *)sheet returnCode:(int)returnCode
    ↪  contextInfo:(void *)contextInfo
```



```
; Attributes: bp-based frame

; void __cdecl -[GPGMailPreferences disableEncryptDraftSheetDidEnd:returnCode:contextInfo:](GPGMailPreferences *self, SEL, id, int, void *)
__GPGMailPreferences_disableEncryptDraftSheetDidEnd_returnCode_contextInfo__ proc near
push    rbp
mov     rbp, rsp
xor     edx, edx
test    ecx, ecx
setz    dl
mov     rax, cs:_NSApp_ptr
mov     rdi, [rax]
or      rdx, 3E8h
mov     rsi, cs:selRef_stopModalWithCode_
pop     rbp
jmp     cs:_objc_msgSend_ptr
__GPGMailPreferences_disableEncryptDraftSheetDidEnd_returnCode_contextInfo__ endp
```

CF, NS, or Neither?

大家如果用 Objective-C 写过 macOS 或者 iOS 上软件的话，要是遇到一些稍微底层一些的 API ，就可能用到 Core Foundation 的函数，需要传一个 CFBlink 类型的变量进去。
这时候你有一个 NSBlink* 的变量，然后发现，诶，直接转换一下类型就可以直接传进去，妙啊。
如果你特别有经验，甚至写过 Swift ，你会发现，你拿着一个 Blink 类型的变量，可以直接当成 NSBlink* 来用。

Toll (sometimes) free bridging

原理其实很简单：NSString 定义了接口，没有定义实现。我完全可以用 CFString 实现一个 NSString *:

```
[0x0001bb60]> pd
            ;-- -[__NSCFString length]:
            ;-- func.0001bb60:
            ;-- method.__NSCFString.length:
            0x0001bb60      55              push rbp
            0x0001bb61      4889e5          rbp = rsp
            0x0001bb64      5d
          < 0x0001bb65      e906000000      goto sym.__CFStringGetLength2
            0x0001bb6a      660f1f440000
```

As it turns out, there's no particular magic to make this work. It's just pure brute force. The implementation of `CFStringGetLength` looks like this:

```
CFIndex CFStringGetLength(CFStringRef str) {
    CF_OBJC_FUNCDISPATCH0(__kCFStringTypeID, CFIndex, str, "length");

    __CFAssertIsString(str);
    return __CFStrLength(str);
}
```

Origin

我知道你们想看什么

现在开始实践吧：
把 GPGMail 干了！