objc runtime

Tangtang

何为Runtime?

官方解释

The Objective-C language defers as many decisions as it can from compile time and link time to runtime. Whenever possible, it does things dynamically. This means that the language requires not just a compiler, but also a runtime system to execute the compiled code. The runtime system acts as a kind of operating system for the Objective-C language; it’s what makes the language work.

强行翻译

OC将一些静态语言在编译链接时做的事推迟到了编译链接之后，也就是运行时，这使得其更加灵活。这意味着OC不仅需要一个编译器，还需要一个运行时系统来执行编译的代码。运行时机制就像一个操作系统一样，它让所有的工作能够正常的运行

4个大点

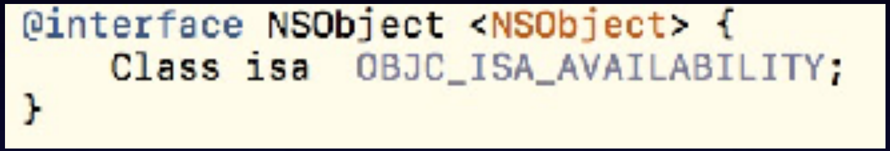
Runtime如何构建类的数据结构

Runtime如何构建消息转发机制

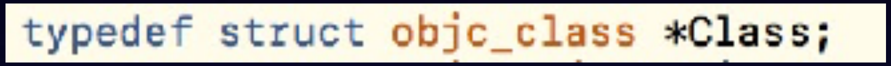
self和super的区别

Runtime的简单应用

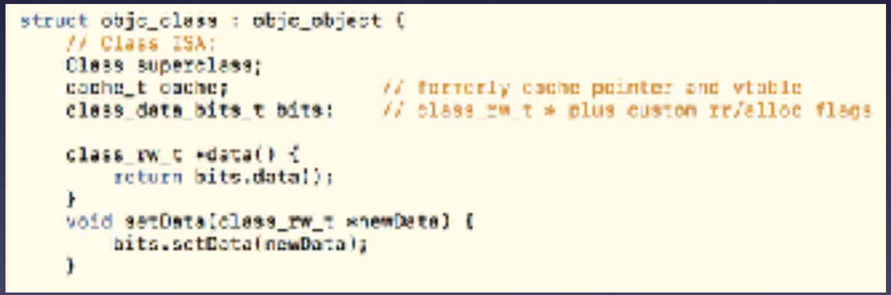
从NSObject开始



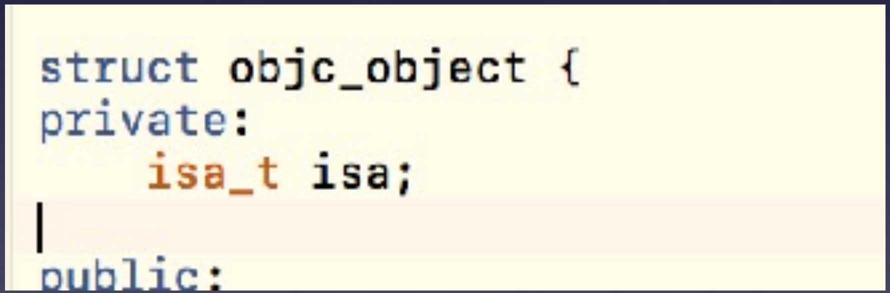
```
@interface NSObject <NSObject> {
    Class isa  OBJC_ISA_AVAILABILITY;
}
```



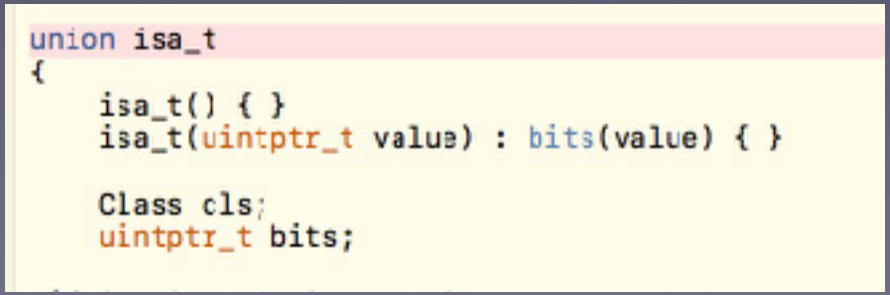
```
typedef struct objc_class *Class;
```



```
struct objc_class : objc_object {
    // Class ISA;
    Class superclass;
    cache_t cache;             // formerly cache pointer and vtable
    class_data_bits_t bits;    // class_rw_t * plus custom rr/alloc flags

    class_rw_t *data() {
        return bits.data();
    }
    void setData(class_rw_t *newData) {
        bits.setData(newData);
    }
```

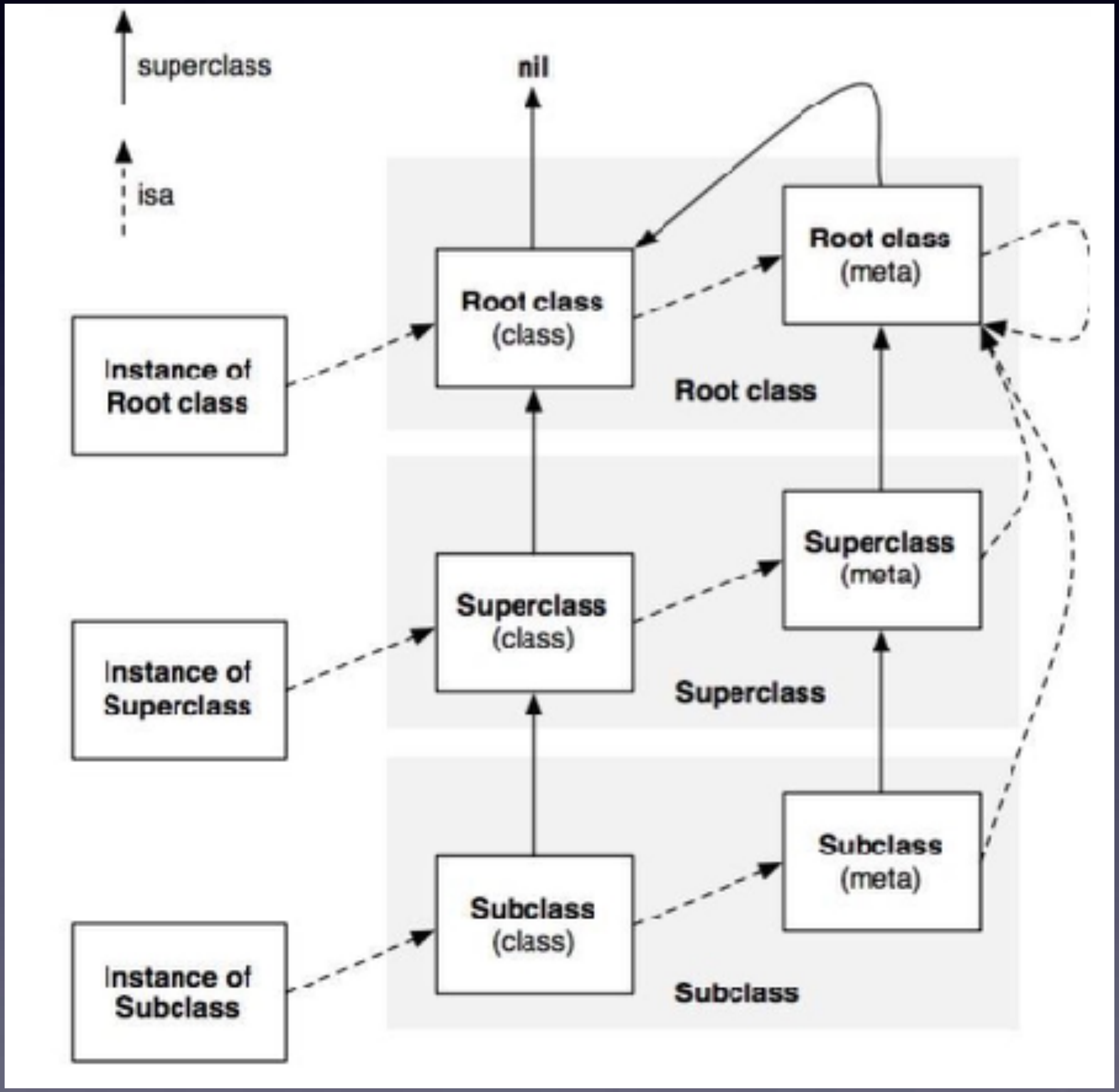


```
struct objc_object {
private:
    isa_t isa;

public:
```

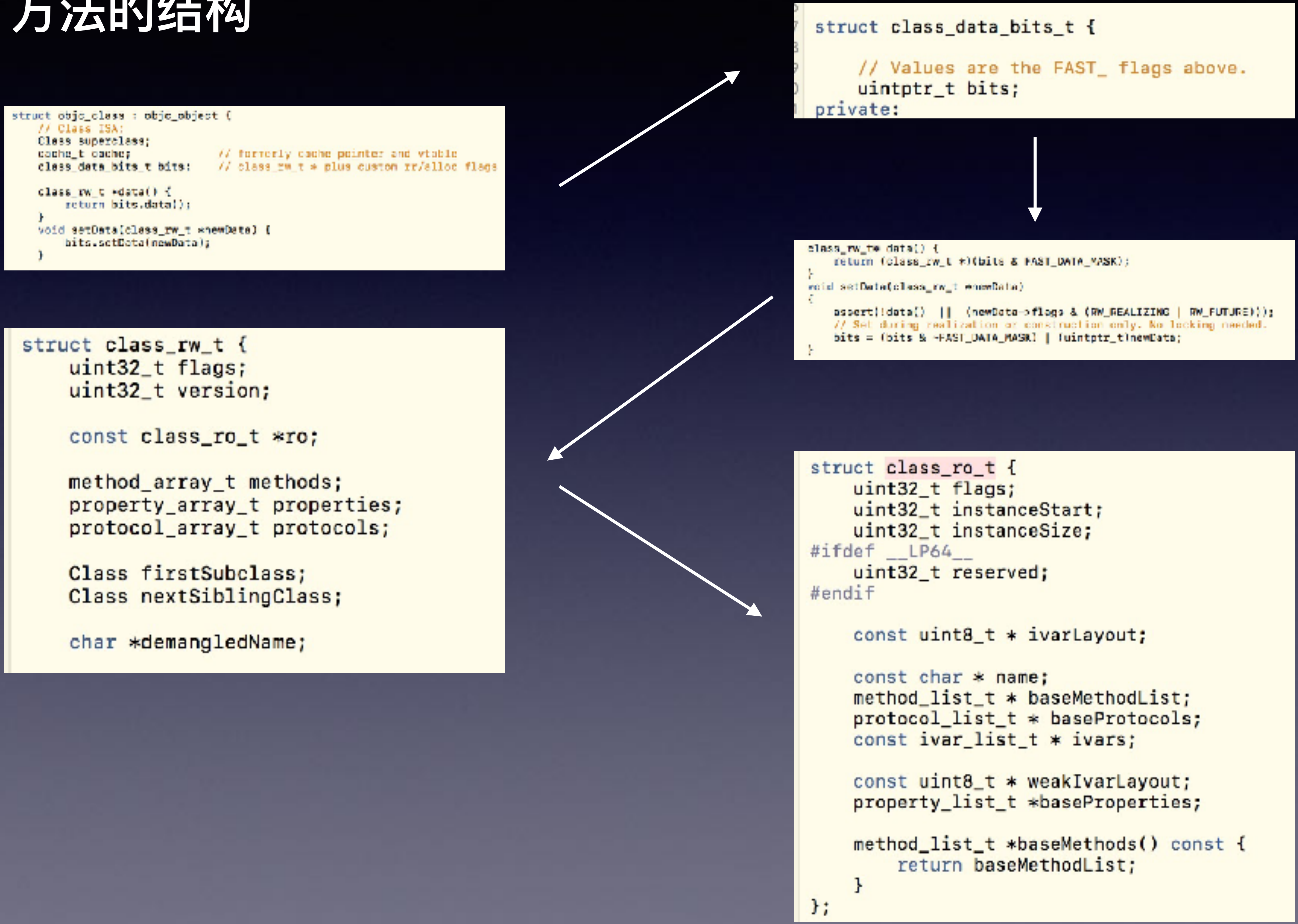


```
union isa_t
{
    isa_t() { }
    isa_t(uintptr_t value) : bits(value) { }

    Class cls;
    uintptr_t bits;
```

类、类对象、元类


superclass
isa
nil
Root class
(class)
Root class
(meta)
Instance of
Root class
Root class
Superclass
(meta)
Superclass
(class)
Instance of
Superclass
Superclass
Subclass
(meta)
Subclass
(class)
Instance of
Subclass
Subclass

方法的结构



```
struct objc_class : objc_object {
    // Class ISA;
    Class superclass;
    cache_t cache;             // formerly cache pointer and vtable
    class_data_bits_t bits;    // class_rw_t * plus custom rr/alloc flags

    class_rw_t *data() {
        return bits.data();
    }
    void setData(class_rw_t *newData) {
        bits.setData(newData);
    }
```

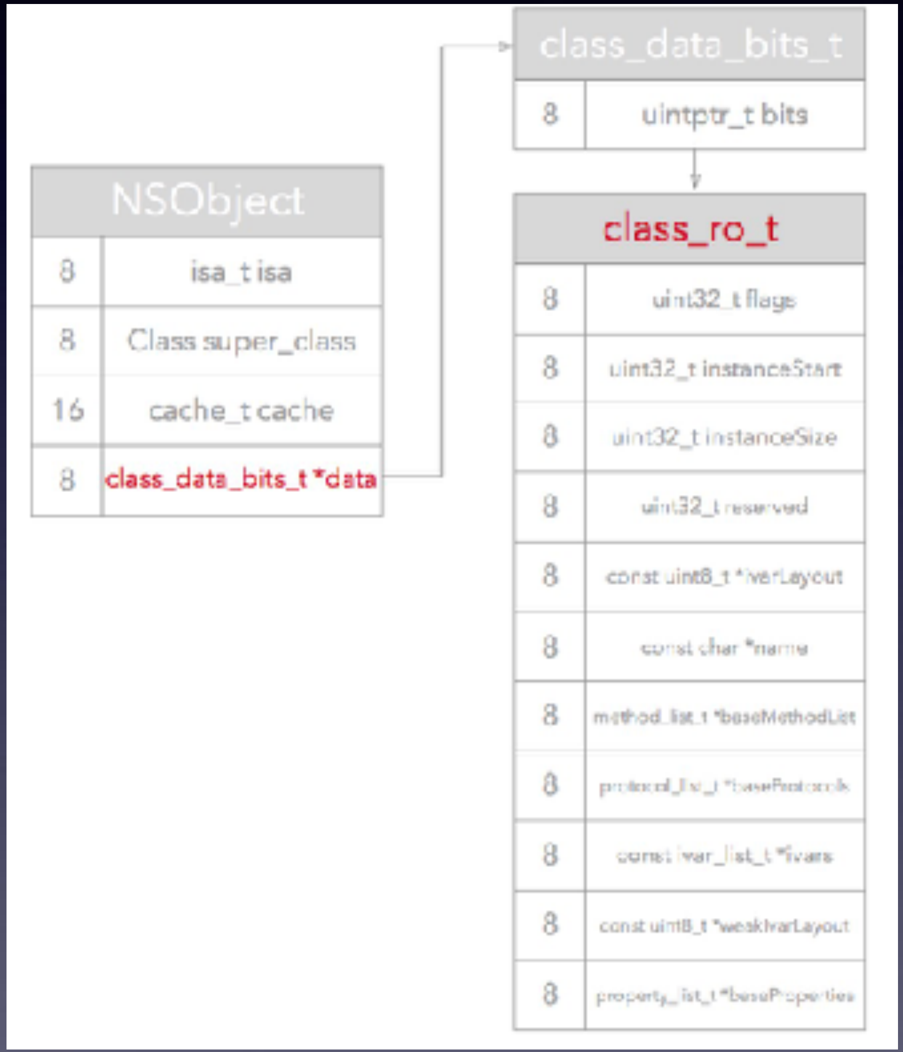
```
struct class_data_bits_t {

    // Values are the FAST_ flags above.
    uintptr_t bits;
private:
```

```
class_rw_t* data() {
    return (class_rw_t *)(bits & FAST_DATA_MASK);
}
void setData(class_rw_t *newData)
{
    assert(!data()  ||  (newData->flags & (RW_REALIZING | RW_FUTURE)));
    // Set during realization or construction only. No locking needed.
    bits = (bits & ~FAST_DATA_MASK) | (uintptr_t)newData;
}
```

```
struct class_rw_t {
    uint32_t flags;
    uint32_t version;

    const class_ro_t *ro;

    method_array_t methods;
    property_array_t properties;
    protocol_array_t protocols;

    Class firstSubclass;
    Class nextSiblingClass;

    char *demangledName;
```

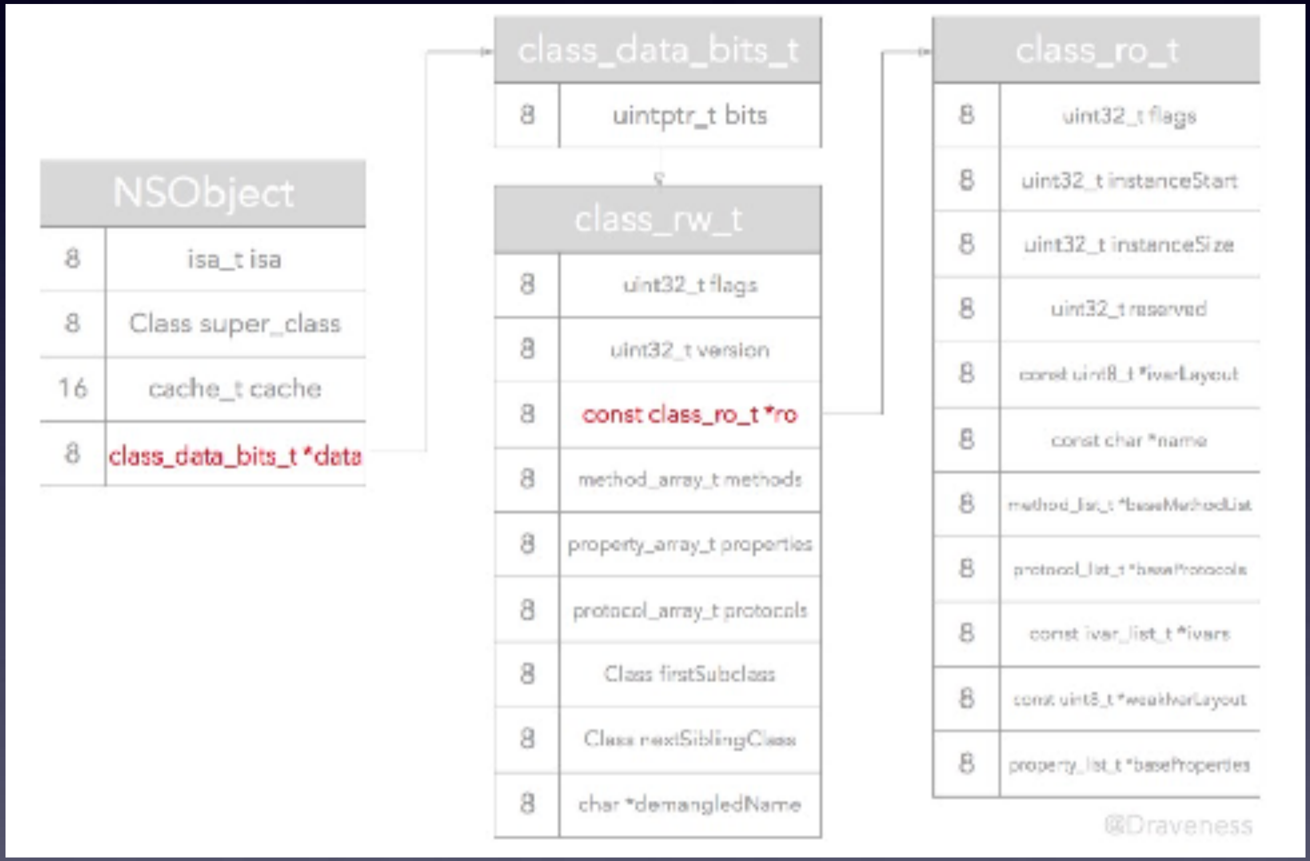
```
struct class_ro_t {
    uint32_t flags;
    uint32_t instanceStart;
    uint32_t instanceSize;
#ifdef __LP64__
    uint32_t reserved;
#endif

    const uint8_t * ivarLayout;

    const char * name;
    method_list_t * baseMethodList;
    protocol_list_t * baseProtocols;
    const ivar_list_t * ivars;

    const uint8_t * weakIvarLayout;
    property_list_t *baseProperties;

    method_list_t *baseMethods() const {
        return baseMethodList;
    }
};
```

方法的结构

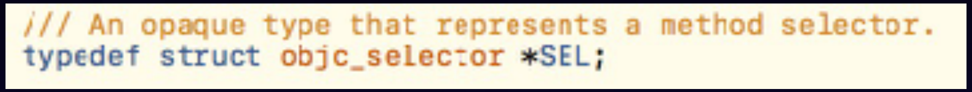


class_data_bits_t
8 uintptr_t bits
NSObject
8 isa_t isa
8 Class super_class
16 cache_t cache
8 class_data_bits_t *data
class_ro_t
8 uint32_t flags
8 uint32_t instanceStart
8 uint32_t instanceSize
8 uint32_t reserved
8 const uint8_t *ivarLayout
8 const char *name
8 method_list_t *baseMethodList
8 protocol_list_t *baseProtocols
8 const ivar_list_t *ivars
8 const uint8_t *weakIvarLayout
8 property_list_t *baseProperties

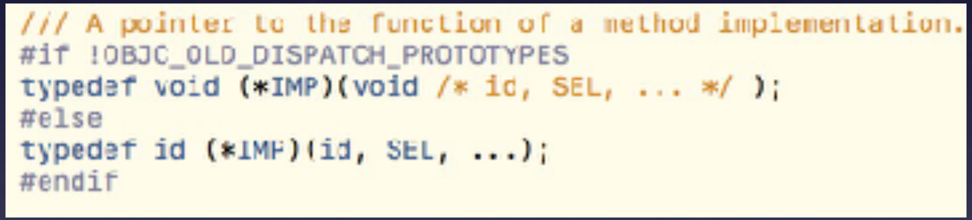
$\longrightarrow$



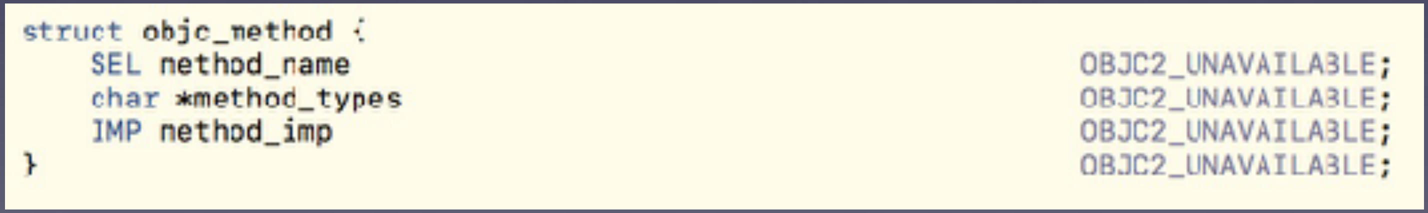
SEL、IMP、Method



```
/// An opaque type that represents a method selector.
typedef struct objc_selector *SEL;
```

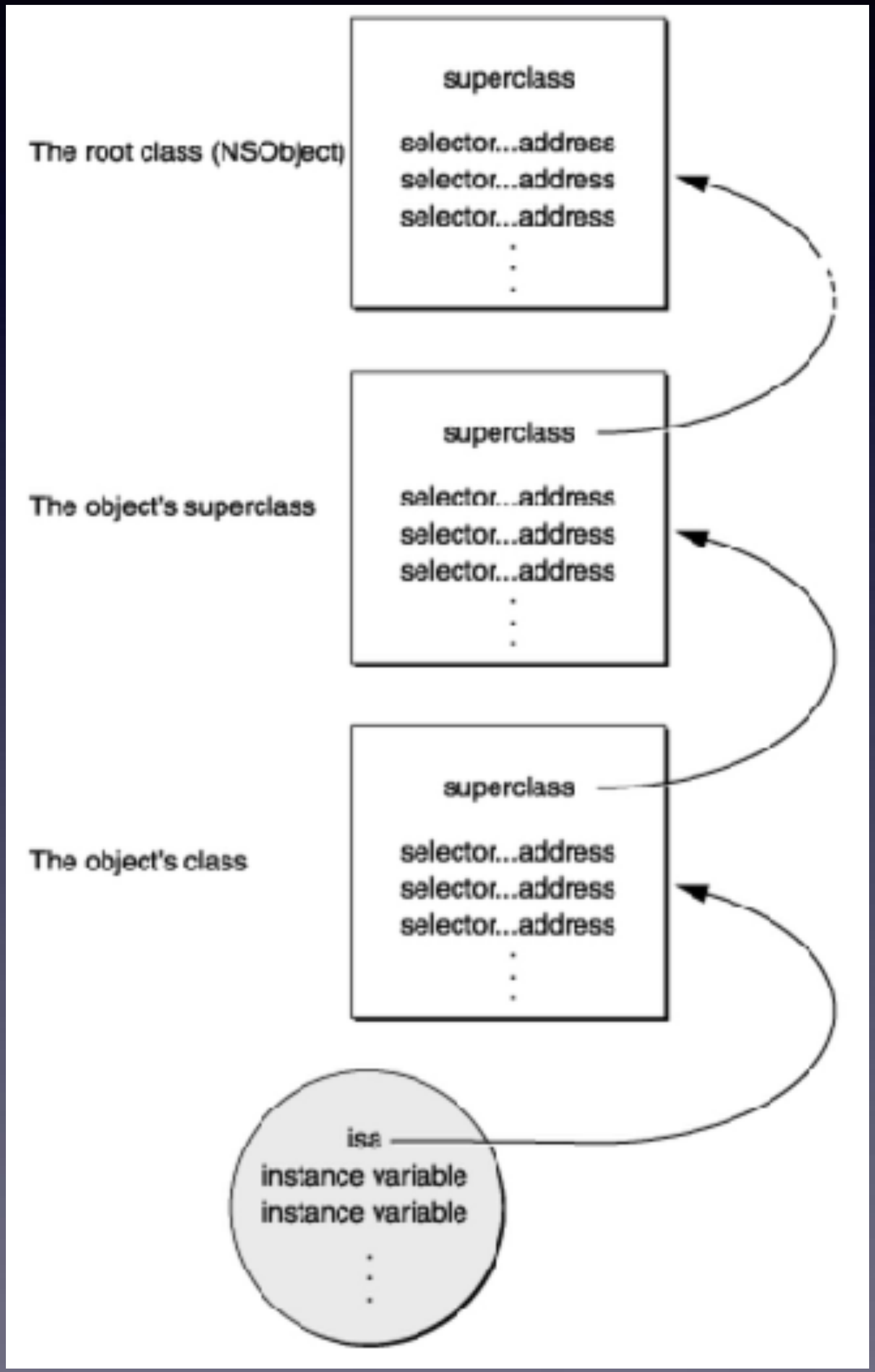


```
/// A pointer to the function of a method implementation.
#if !OBJC_OLD_DISPATCH_PROTOTYPES
typedef void (*IMP)(void /* id, SEL, ... */ );
#else
typedef id (*IMP)(id, SEL, ...);
#endif
```

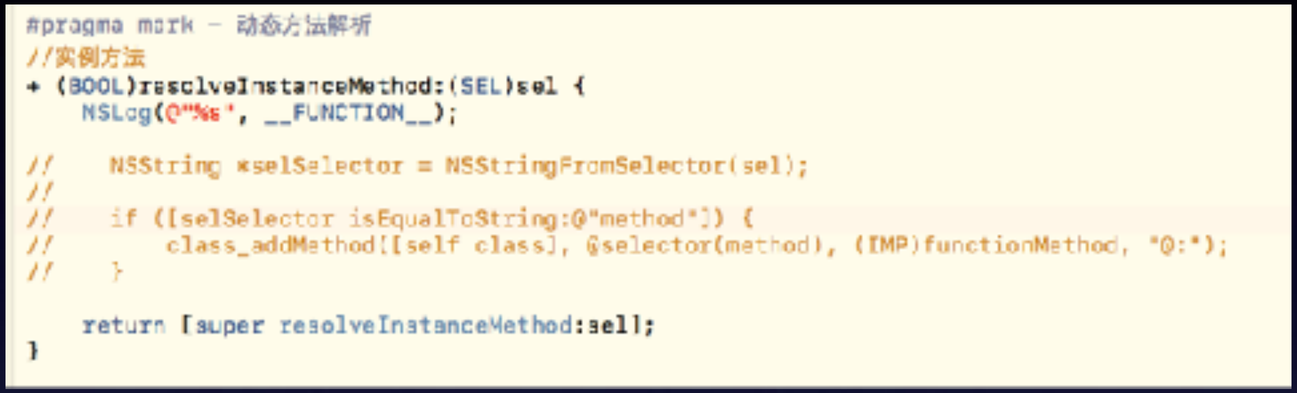


```
struct objc_method {
    SEL method_name                                         OBJC2_UNAVAILABLE;
    char *method_types                                      OBJC2_UNAVAILABLE;
    IMP method_imp                                          OBJC2_UNAVAILABLE;
}                                                           OBJC2_UNAVAILABLE;
```

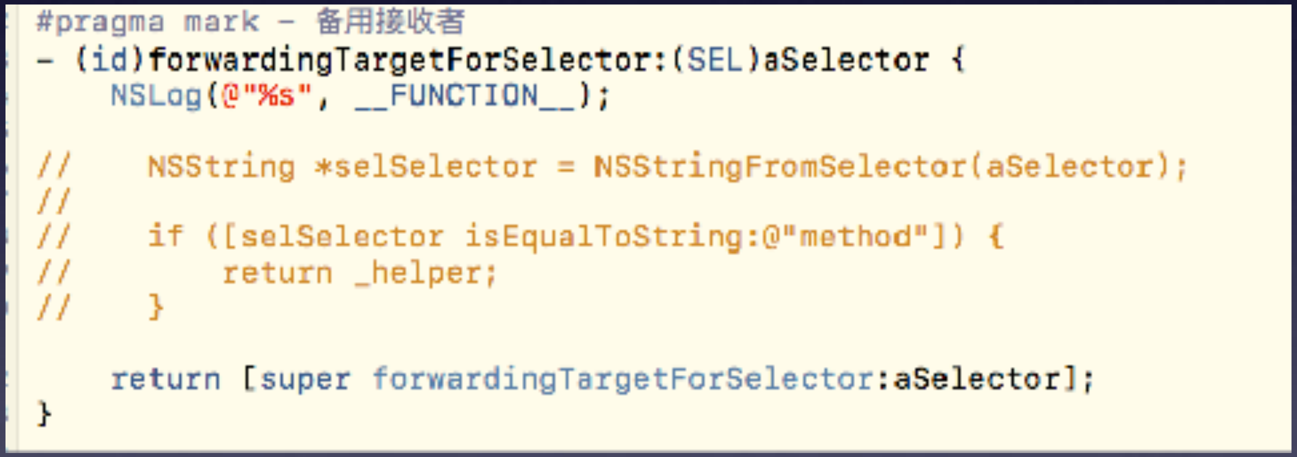
方法查找


The root class (NSObject)
superclass
selector...address
selector...address
selector...address
The object's superclass
superclass
selector...address
selector...address
selector...address
The object's class
superclass
selector...address
selector...address
selector...address
isa
instance variable
instance variable

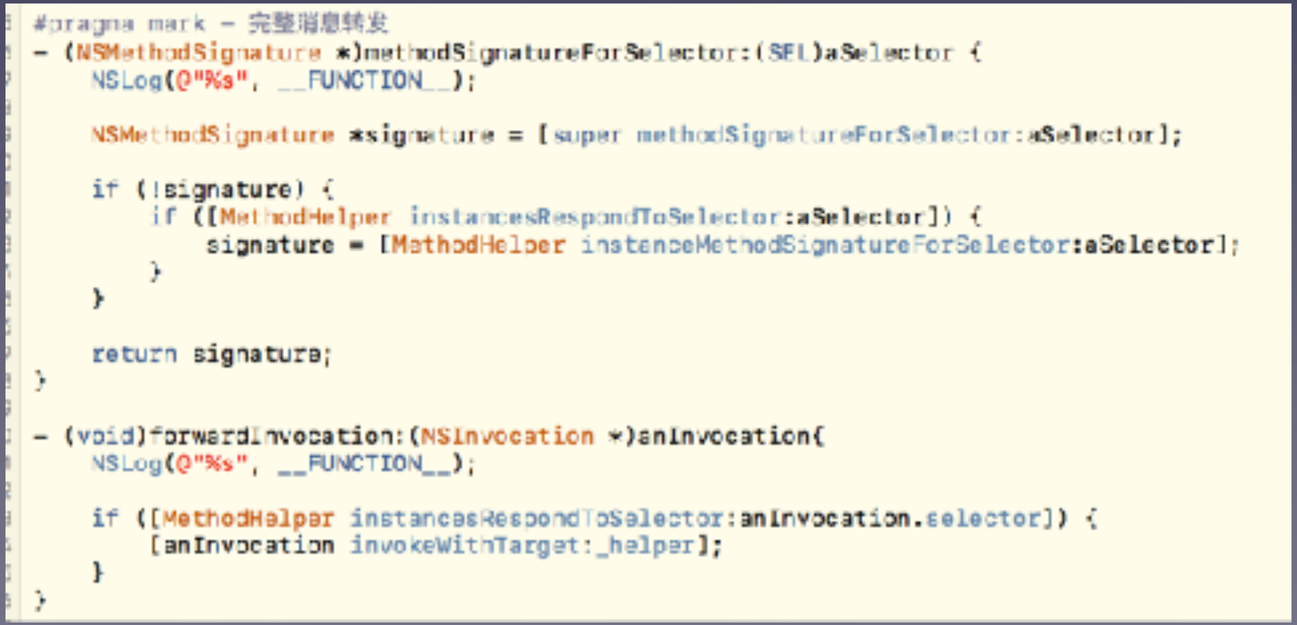
消息转发



```
#pragma mark - 动态方法解析
//实例方法
+ (BOOL)resolveInstanceMethod:(SEL)sel {
    NSLog(@"%s", __FUNCTION__);

//    NSString *selSelector = NSStringFromSelector(sel);
//
//    if ([selSelector isEqualToString:@"method"]) {
//        class_addMethod([self class], @selector(method), (IMP)functionMethod, "@:");
//    }

    return [super resolveInstanceMethod:sel];
}
```



```
#pragma mark - 备用接收者
- (id)forwardingTargetForSelector:(SEL)aSelector {
    NSLog(@"%s", __FUNCTION__);

//    NSString *selSelector = NSStringFromSelector(aSelector);
//
//    if ([selSelector isEqualToString:@"method"]) {
//        return _helper;
//    }

    return [super forwardingTargetForSelector:aSelector];
}
```



```
#pragma mark - 完整消息转发
- (NSMethodSignature *)methodSignatureForSelector:(SEL)aSelector {
    NSLog(@"%s", __FUNCTION__);

    NSMethodSignature *signature = [super methodSignatureForSelector:aSelector];

    if (!signature) {
        if ([MethodHelper instancesRespondToSelector:aSelector]) {
            signature = [MethodHelper instanceMethodSignatureForSelector:aSelector];
        }
    }

    return signature;
}

- (void)forwardInvocation:(NSInvocation *)anInvocation{
    NSLog(@"%s", __FUNCTION__);

    if ([MethodHelper instancesRespondToSelector:anInvocation.selector]) {
        [anInvocation invokeWithTarget:_helper];
    }
}
```

深入objc_msgSend

缓存是否命中

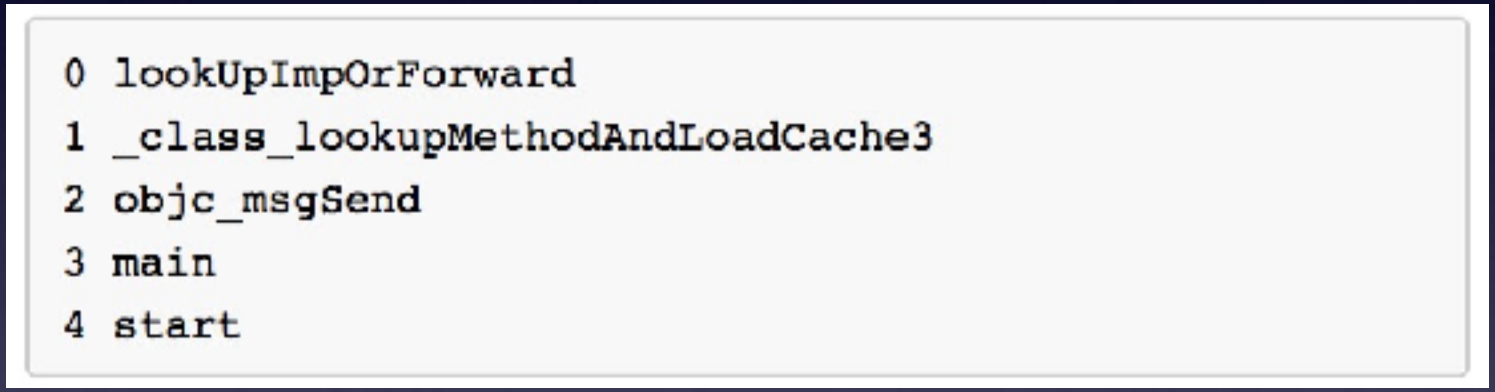
查找当前类的缓存及方法

查找父类的缓存及方法

方法决议(即消息转发机制的第一步，动态方法解析)

消息转发

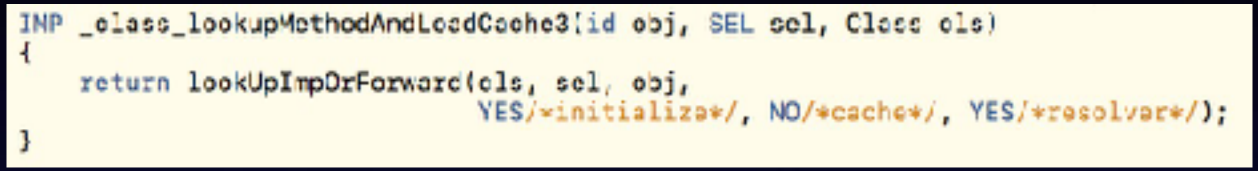
无缓存



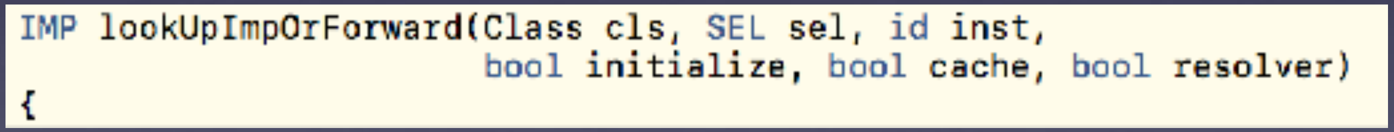
```
0 lookUpImpOrForward
1 _class_lookupMethodAndLoadCache3
2 objc_msgSend
3 main
4 start
```

objc_msgSend调用栈

无缓存

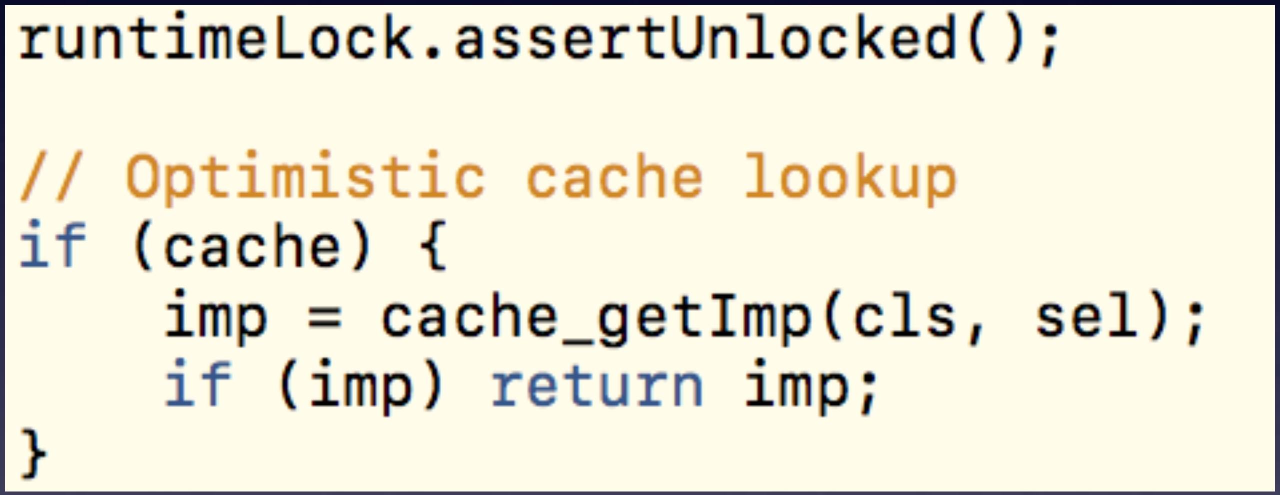


```
IMP _class_lookupMethodAndLoadCache3(id obj, SEL sel, Class cls)
{
    return lookUpImpOrForward(cls, sel, obj,
                              YES/*initialize*/, NO/*cache*/, YES/*resolver*/);
}
```



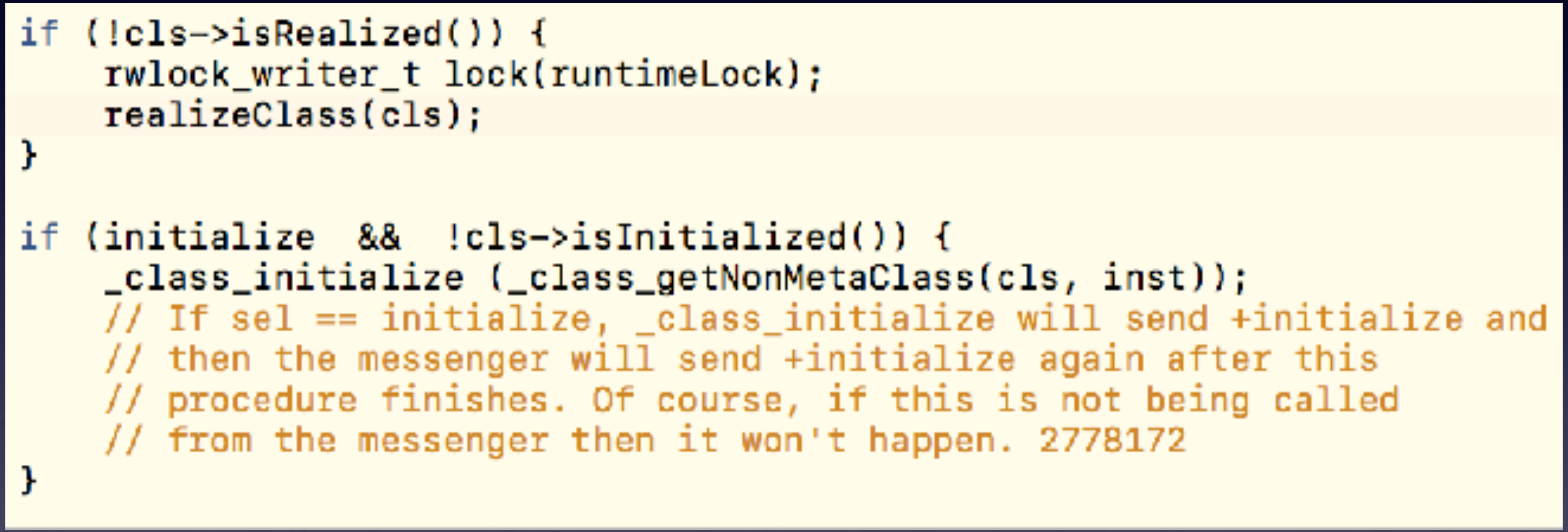
```
IMP lookUpImpOrForward(Class cls, SEL sel, id inst,
                       bool initialize, bool cache, bool resolver)
{
```

无锁的缓存查找



```
runtimeLock.assertUnlocked();

// Optimistic cache lookup
if (cache) {
    imp = cache_getImp(cls, sel);
    if (imp) return imp;
}
```

类的实现和初始化



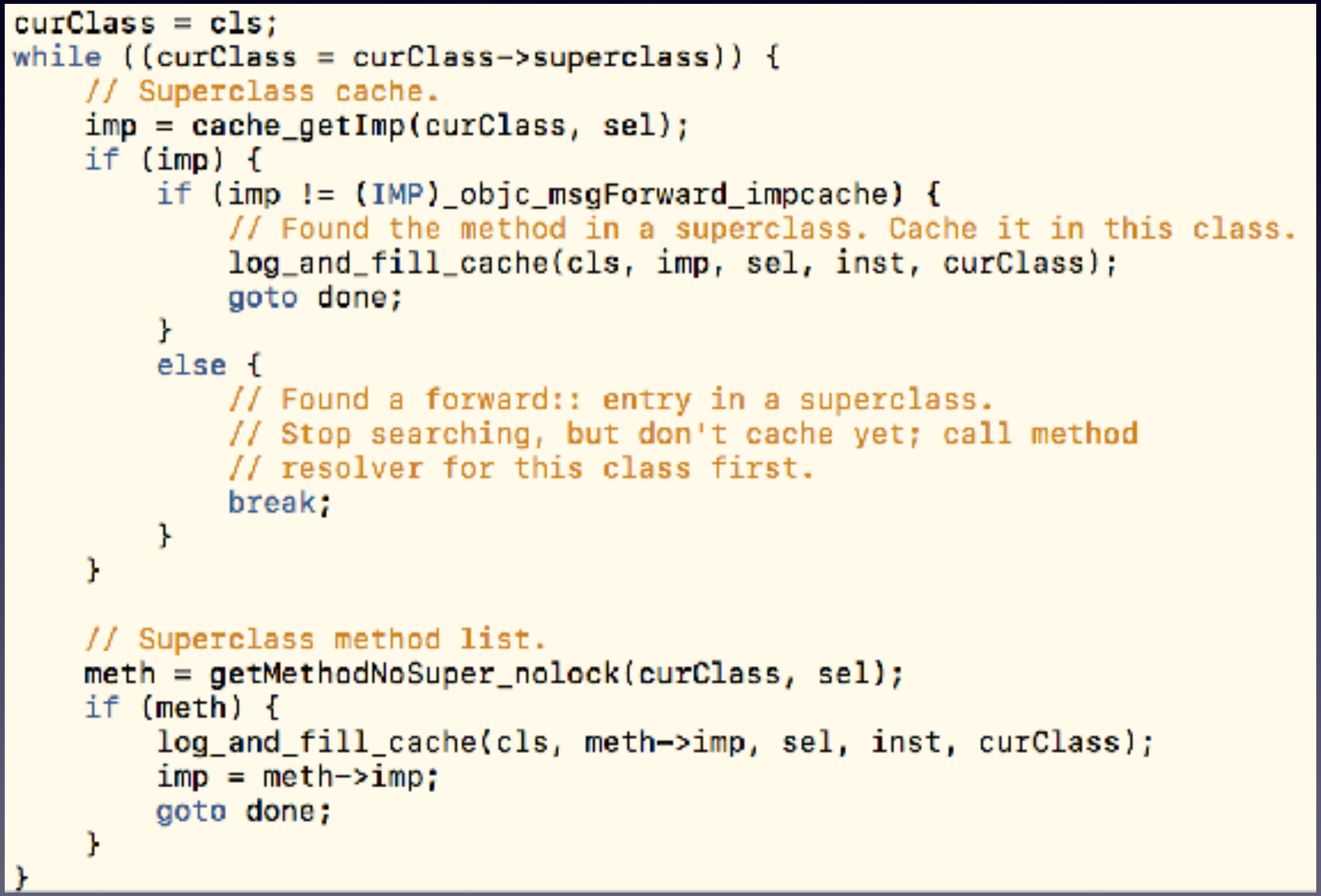
```
if (!cls->isRealized()) {
    rwlock_writer_t lock(runtimeLock);
    realizeClass(cls);
}

if (initialize  &&  !cls->isInitialized()) {
    _class_initialize (_class_getNonMetaClass(cls, inst));
    // If sel == initialize, _class_initialize will send +initialize and
    // then the messenger will send +initialize again after this
    // procedure finishes. Of course, if this is not being called
    // from the messenger then it won't happen. 2778172
}
```

加锁以及查找当前类

```
runtimeLock.read();
```

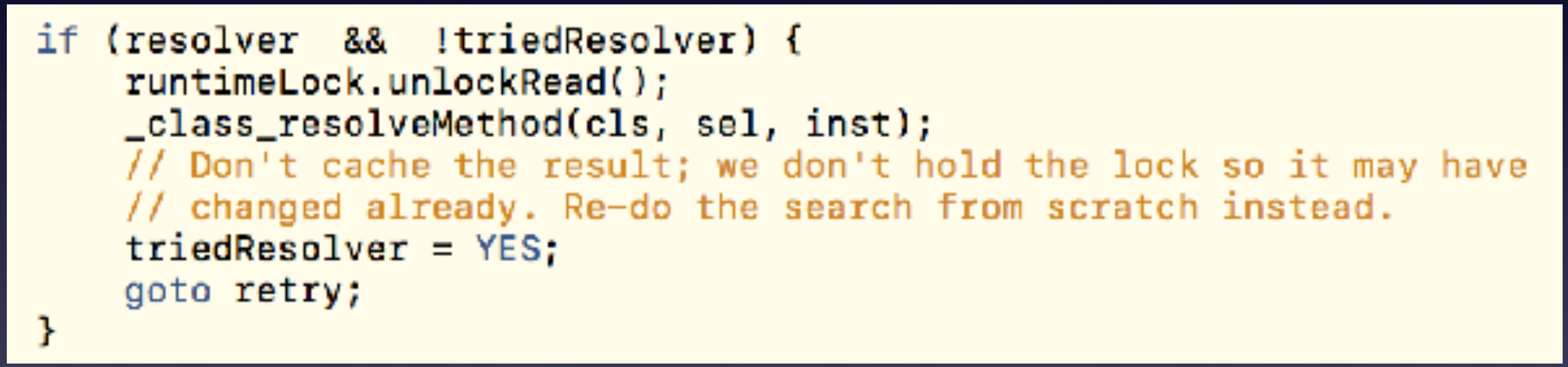
```
// Try this class's cache.

imp = cache_getImp(cls, sel);
if (imp) goto done;

// Try this class's method lists.

meth = getMethodNoSuper_nolock(cls, sel);
if (meth) {
    log_and_fill_cache(cls, meth->imp, sel, inst, cls);
    imp = meth->imp;
    goto done;
}
```

查找当前类的父类



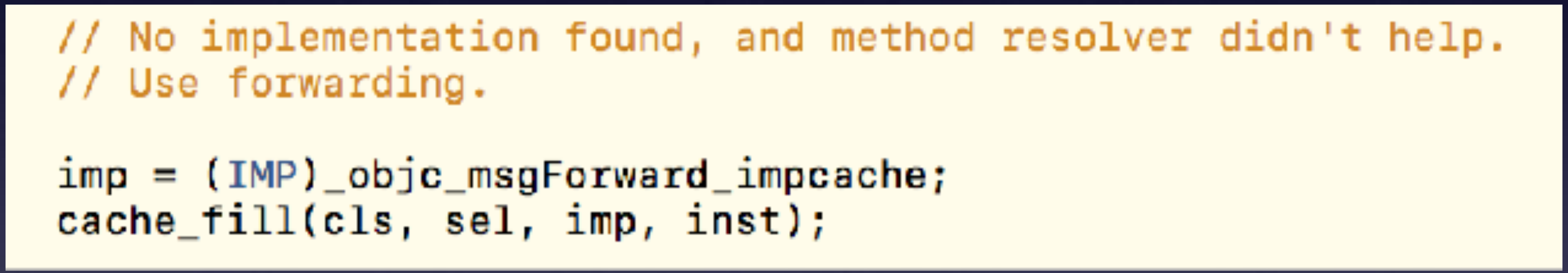
```
curClass = cls;
while ((curClass = curClass->superclass)) {
    // Superclass cache.
    imp = cache_getImp(curClass, sel);
    if (imp) {
        if (imp != (IMP)_objc_msgForward_impcache) {
            // Found the method in a superclass. Cache it in this class.
            log_and_fill_cache(cls, imp, sel, inst, curClass);
            goto done;
        }
        else {
            // Found a forward:: entry in a superclass.
            // Stop searching, but don't cache yet; call method
            // resolver for this class first.
            break;
        }
    }

    // Superclass method list.
    meth = getMethodNoSuper_nolock(curClass, sel);
    if (meth) {
        log_and_fill_cache(cls, meth->imp, sel, inst, curClass);
        imp = meth->imp;
        goto done;
    }
}
```

方法决议



```
if (resolver  &&  !triedResolver) {
    runtimeLock.unlockRead();
    _class_resolveMethod(cls, sel, inst);
    // Don't cache the result; we don't hold the lock so it may have
    // changed already. Re-do the search from scratch instead.
    triedResolver = YES;
    goto retry;
}
```

消息转发



```
// No implementation found, and method resolver didn't help.
// Use forwarding.

imp = (IMP)_objc_msgForward_impcache;
cache_fill(cls, sel, imp, inst);
```

有缓存

直接在objc_msgSend中使用汇编完成缓存查找

简单应用

Method Swizzling

Associated Object关联对象

动态的增加方法

字典和模型互相转换

讨论时间

github: https://github.com/iosTangtang/RuntimeDemo.git