iOS面试题：为什么Objective-C中有MetaClass这个设计?

iOS小小...
喜欢思考，热爱技术，喜欢交友

前置知识

首先简单分析下在Objective-C中，对象是什么。下面源码基于Runtime-709分析。

```
typedef struct objc_object *id;//id其实是一个object结构体的指针，所以id不用加*
typedef struct objc_class *Class;//Class是class结构体的指针

struct objc_object {
    Class isa;
};

struct objc_class : objc_object {
    Class superclass;
    cache_t cache;      // 用来缓存指针和虚函数表
    class_data_bits_t bits;  //方法列表等
    //...
}
```

可以看到，对象最基本的就是有一个isa指针，指向他的class，而Class本身是继承自object。isa指针的理解诶就是英文is a，代表“xxx is a （class）”。那么也就是说，一个对象的isa指向哪个class，代表它是那个类的对象。那么对于class来说，它也是一个对象，它的isa指针指向什么呢?

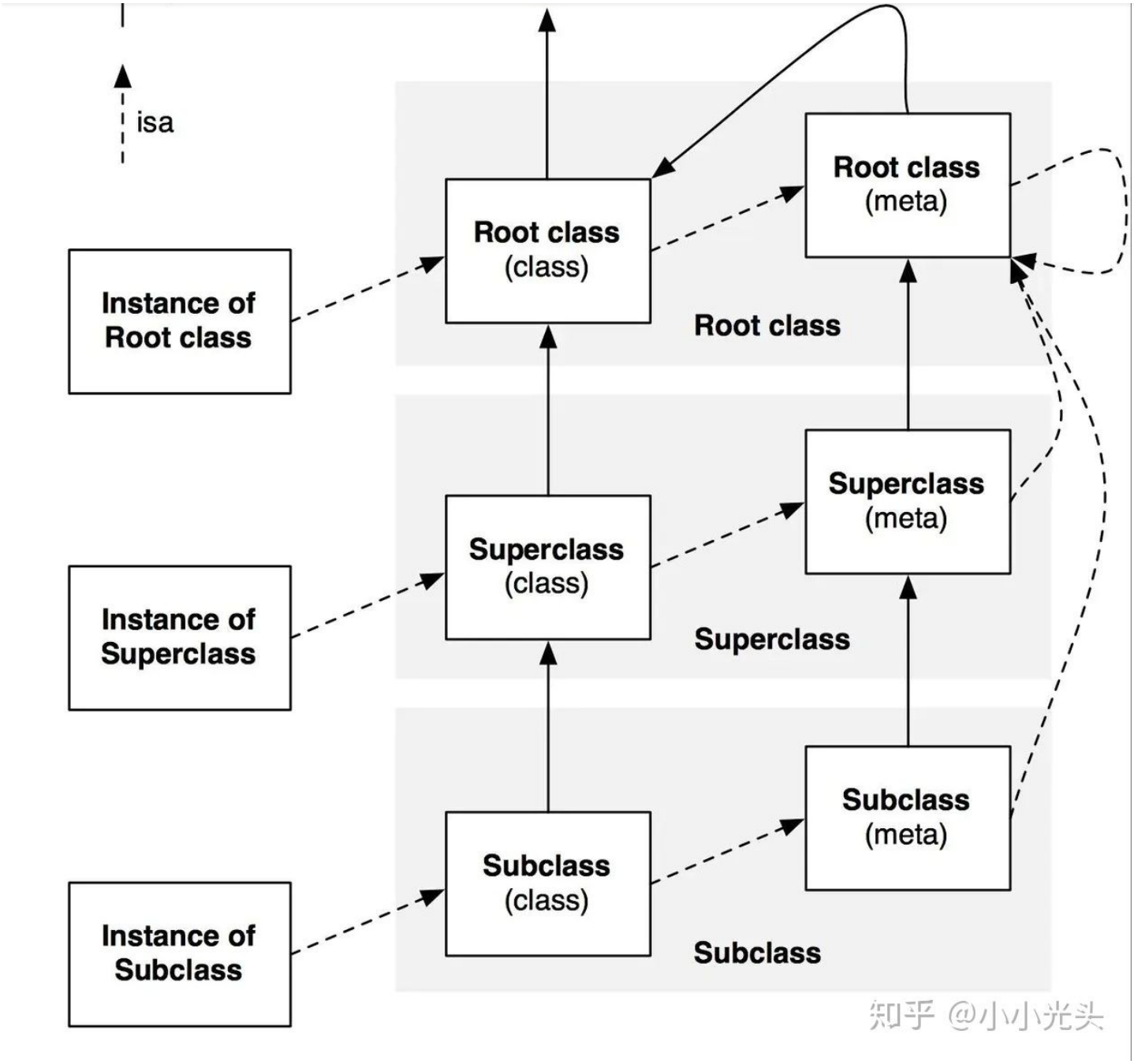
对于Class来说，也就需要一个描述他的类，也就是“类的类”，而meta正是“关于某事自身的某事”的解释，所以MetaClass就因此而生了。

而从runtime动态生成一个类的Api的方法中，我们也可以发现metaClass的踪迹。

```
Class objc_allocateClassPair(Class superclass, const char *name,
                             size_t extraBytes)
{
    Class cls, meta;
```

```
    if (getClass(name)  ||  !verifySuperclass(superclass, true/*rootOK*/)) {
        return nil;
    }

    //分配空间
    cls  = alloc_class_for_subclass(superclass, extraBytes);
    meta = alloc_class_for_subclass(superclass, extraBytes);

    //构建meta和class的关系
    objc_initializeClassPair_internal(superclass, name, cls, meta);

    return cls;
}
```

通过这个方法生成后，就成了大家熟悉的那张图。



从这张图上，我们可以看到通过这么一层继承关系，Objective-C的对象原型继承链就完整了。

同时，实例的实例方法函数存在类结构体中，类方法函数存在metaclass结构体中，而Objective-C的方法调用（消息）就会根据对象去找isa指针指向的Class对象中的方法列表找到对应的方法。

Python中的MetaClass

先看看Python中一个对象结构是怎么样的，以下源码基于CPython 3.7.0 alpha 1。

```
//object.h
typedef struct _object {
    _PyObject_HEAD_EXTRA
    Py_ssize_t ob_refcnt;//引用计数
    struct _typeobject *ob_type;//类型
} PyObject;
```

和Objective-C中类似，ob_type其实就是一个isa指针，代表是什么类型。

而再看看PyTypeObject是怎么样的。

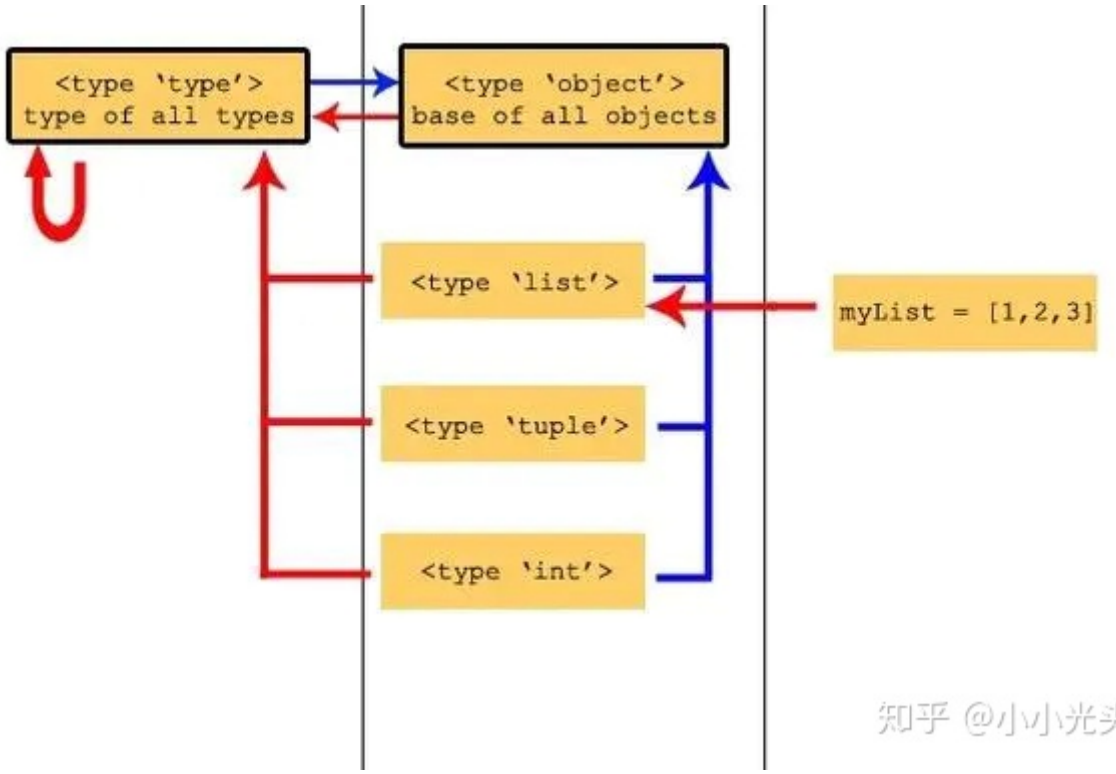
```
//object.h
typedef struct _typeobject {
    PyObject_VAR_HEAD
    const char *tp_name; /* For printing, in format "<module>.<name>" */
    Py_ssize_t tp_basicsize, tp_itemsize; /* For allocation */
   //....
} PyTypeObject;

#define PyObject_VAR_HEAD      PyVarObject ob_base;

typedef struct {
    PyObject ob_base;
    Py_ssize_t ob_size; //对象长度
} PyVarObject;
```

PyVarObject是一种可变长度对象，是在PyObject基础上加上了对象的长度。而开始的内存包括了ob_base这个PyObject，就代表可以用PyObject指针进行引用。所以可以说，结构体中刚开始的部分是一个PyObject对象，在Python中引用就是一个对象。那么PyTypeObject开头是一个PyVarObject，也就是一个对象。也就是说，Python里的Class，也是一个对象。

```
#在python中生成一个Class
MyClass = type('MyClass', (), {})
```

```
//typeobject.c
PyTypeObject PyType_Type = {
    PyVarObject_HEAD_INIT(&PyType_Type, 0)
    "type",                                     /* tp_name */
    //.....
    type_init,                                  /* tp_init */
    //....
    type_new,                                   /* tp_new */
    //....
};
```

可以发现type关键字是PyType_Type的一个引用，而PyType_Type是返回一个PyTypeObject，生成类的对象。而PyVarObject_HEAD_INIT递归引用了自己（PyType_Type）作为它的type，所以可以得知 `type(class) == type` 。也就是说，Python中类的isa指针指向type，也就说type其实就是MetaClass，而同时 `type(type) == type` ，也就是type的isa指针指向type自身。那么Python的对象链就如下图。



中**类的制造者的权限**。而同时，根据StackOverFlow这个问答，Python的类的设计是借鉴于Smalltalk这门语言。

Smalltalk！！Objective-C的特性基本上是照搬的Smalltalk，看来Smalltalk里可以找到一些线索。

Smalltalk-面向对象的前辈

Smalltalk，被公认为历史上第二个面向对象的语言，其亮点是它的**消息发送**机制。

Smalltalk中的MetaClass的设计是Smalltalk-80加入的。而之前的Smalltalk-76，并不是每个类有一个MetaClass，而是所有类的isa指针都指向一个特殊的类，叫做 `Class`（这种设计之后也被Java借鉴了）。

而每个类都有自己MetaClass的设计，加入的原因是，因为Smalltalk里面，类是对象，而对象就可以响应消息，那么类的消息的响应的方法就应该由类的类去存储，而每个MetaClass就持有每个类的类方法。

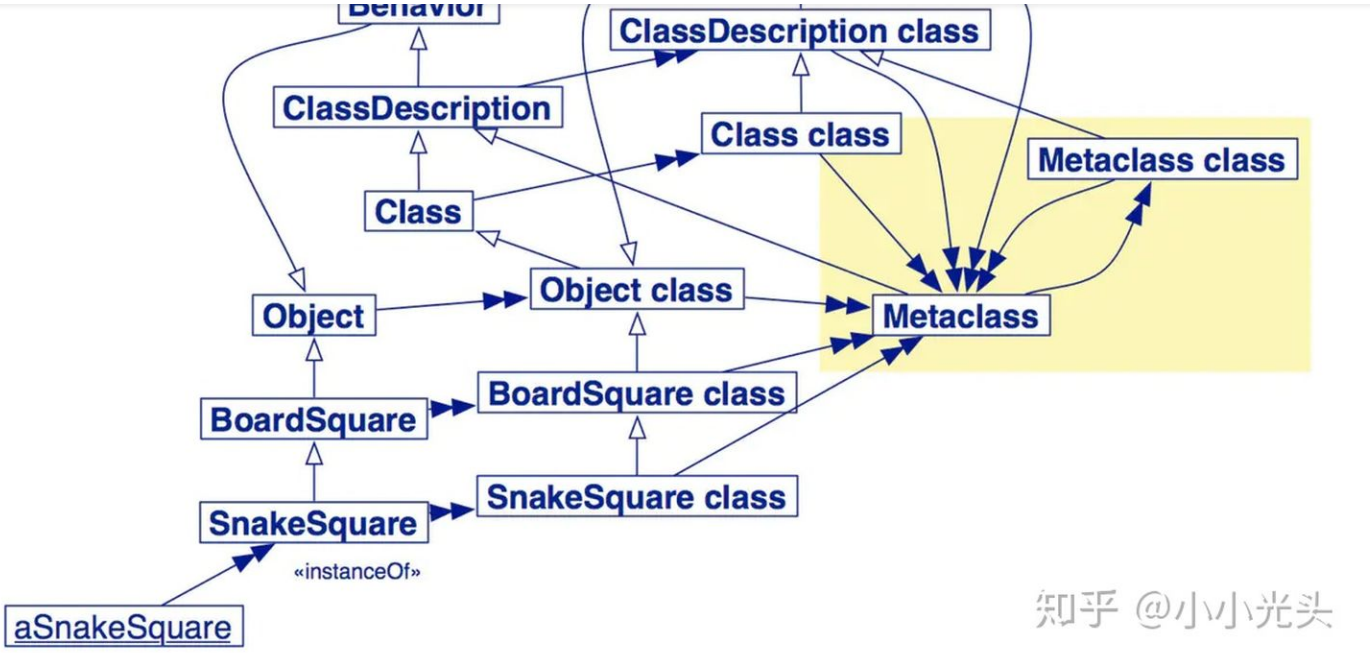
问题1：每个MetaClass的isa指针指向什么?

如果MetaClass再有MetaClass，那么这个关系将无穷无尽。Smalltalk里的解决方案是，指向同一个叫 `MetaClass` 的类。

问题2： `MetaClass` 的isa指针指向什么?

指向他的实例，也就是实例的isa指向 `MetaClass` ，同时 `MetaClass` isa指向实例，相互指着。

那么Smalltalk的继承关系，其实和Objective-C的很像了（后面有class的是前者的MetaClass）。



这时候产生了一个重要的问题，假如去掉MetaClass，把类方法放到也类里面是否可行?

这个问题，我思索许久，发现其实是一个对面向对象的哲学思想问题，要对这个问题下结论，不得不重新讲讲面向对象。

从Smalltalk重新认识面向对象

以前谈到面向对象，总会提到，面向对象三特征：**封装、继承、多态**。但其实，面向对象中也分流派，如C++这种来自Simula的设计思想的，更注重的是**类的划分**，因为方法调用是静态的。而如Objective-C这种借鉴Smalltalk的，更注重的是**消息传递**，是动态响应消息。

而面向对象三种特征，更基于的是类的划分而提出的。

这两种思想最大的不同，我认为是**自上而下**和**自下而上**的思考方式。

- 类的划分，要求类的设计者是以一个很高的层次去设计这个类，提取出**类的特性和本质**，进行类的构建。知道类型才可以去发送消息给对象。
- 消息传递，要求的是类的设计者以消息为起点去构建类，也就是对**外界的变化进行响应**，而不关心自身的类型，设计接口。尝试理解消息，无法处理则进行特殊处理。

关心接口，关心组合而非类本身。其实之所以有MetaClass这种设计，我的理解并不是先有MetaClass，而是在万物都是对象的Smalltalk里，向对象发送消息的基本解决方案是统一的，希望复用的。而实例和类之间用的这一套通过isa指针指向的Class单例中存储方法列表和查询方法的解决方案的流程，是应该在类上复用的，而MetaClass就顺理成章出现罢了。

最后

回到一开始那个问题，**为什么要设计MetaClass，去掉把类方法放到类里面行不行?**

我的理解是，可以，但不Smalltalk。这样的设计是C++那种自上而下的设计方式，类方法也是类的一种特征描述。而Smalltalk的精髓正在于消息传递，**复用消息传递**才是根本目的，而MetaClass只不过是因此需要的一个工具罢了。

PS：笔者这个问题从MetaClass入手去思考，是百思不得其解的。后来看了很多面向对象的东西，才发现这不过是一个产物，而并不是一个重点。

PSS：对于类的实现，Javascript中那种使用Protocol实现的方式也很有意思，受限于篇幅，暂不展开

更多：iOS面试题大全

收录：原文地址

参考链接

- What is a meta-class in Objective-C?
- StackOverFlow-What is a metaclass in Python?
- Design Principles Behind Smalltalk
- CSE 341: Smalltalk classes and metaclasses
- Understanding Classes and Metaclasses
- function/bind的救赎（上）

编辑于 2020-10-22

Objective-C Class 源代码

知乎 首发于 iOS面试题合集

文章被以下专栏收录

iOS面试题合集
收集的iOS面试题，答案不唯一，仅供参考！

推荐阅读

【iOS底层原理】OC对象的本质（一）

一个 NSObject 对象占用多少内存。 系统分配16个字节给NSObject对象，但是NSObject 对象内部只使用了8个字节（64位环境下）。我们平时编写的Object-C代码，底层实现其实都是C\C++代码。 所...

ney

IOS学习笔记之Object-C（一）

南山伐木

【学习笔记】Objective-C_语言基础（一）

Barry

还没有评论

写下你的评论...