来自饿了么移动团队

概述

关于KVC/KVO的实现原理，网上的相关介绍文章很多，但大部分说的比较抽象，难以真切的理解，下面我们直接撸代码来实地探讨下。

演示代码地址：https://github.com/Assuner-Lee/KVC-KVO-Test.git

KVC 演示代码

ASClassA.h

```
#import <Foundation/Foundation.h>

@interface ASModel : NSObject

@property (nonatomic, strong) NSString *_modelString;

@end

@interface ASClassA : NSObject

@property (nonatomic, strong) NSString *stringA;

@property (nonatomic, strong) ASModel *modelA;

@end
```

ASClassA.m

```
#import "ASClassA.h"

@implementation ASModel

- (void)set_modelString:(NSString *)_modelString {
    __modelString = _modelString;
    NSLog(@"执行 setter _modelString");
}

- (void)setModelString:(NSString *)modelString {
    NSLog(@"执行 setter modelString");
}

- (void)setNoExist1:(NSString *)noExist {
    NSLog(@"执行 setter noExist1 ");
}

@end

@implementation ASClassA

- (void)setStringA:(NSString *)stringA {
     NSLog(@"执行 setter stringA");
    _stringA = stringA;

- (instancetype)init {
    if (self = [super init]) {
        self.modelA = [[ASModel alloc] init];
    }
    return self;
}

@end
```

main.m

```
#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wunused-variable"

#import <Foundation/Foundation.h>
#import <objc/runtime.h>
#import "ASClassA.h"

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        ASClassA *objectA = [[ASClassA alloc] init];
        objectA.stringA = @"stringA setter";
// setter
       ① [objectA setValue:@"stringA KVC" forKey:@"stringA"];
// kvc
       ② [objectA setValue:@"_stringA KVC" forKey:@"_stringA"];
// kvc _

        NSLog(@" objectA.stringA 值: %@", objectA.stringA);

        NSLog(@"-----------------------------------------------------
------");

       ③ [objectA setValue:@"_modelString kvc"
forKeyPath:@"modelA._modelString"];    //setter
       ④ [objectA setValue:@"modelString kvc"
forKeyPath:@"modelA.modelString"];      // kvc  不存在的属性
       ⑤ [objectA setValue:@"__modelString kvc"
forKeyPath:@"modelA.__modelString"]; //kvc _

       ⑥ [objectA setValue:@"noExist1"
forKeyPath:@"modelA.noExist1"];              //kvc  不存在的属性
        NSLog(@"objectA.modelA._modelString 值: %@",
objectA.modelA._modelString);

        NSLog(@"-----------------------------------------------------
------");

       ⑦ NSString *s1 = [objectA
valueForKeyPath:@"modelA._modelString"];
       ⑧ NSString *s2 = [objectA
valueForKeyPath:@"modelA.modelString"];
       ⑨ NSString *s3 = [objectA
valueForKeyPath:@"modelA.__modelString"];
}
    return 0;
}
```

运行结果

①->⑨全部执行成功; 其中①③④⑥ 执行了setter方法，⑦⑧执行了getter方法，②⑤⑨直接访问的实例变量。

原因

其实点进去valueForKey: 或setValueForKey: 帮助文档已经讲得很清楚了

valueForKey:

```
The default implementation of this method does the following:
    1\. Searches the class of the receiver for an accessor method
whose name matches the pattern -get<Key>, -<key>, or -is<Key>, in
that order. If such a method is found it is invoked. If the type of
the method's result is an object pointer type the result is simply
returned. If the type of the result is one of the scalar types
supported by NSNumber conversion is done and an NSNumber is
returned. Otherwise, conversion is done and an NSValue is returned
(new in Mac OS 10.5: results of arbitrary type are converted to
NSValues, not just NSPoint, NRange, NSRect, and NSSize).
    2 (introduced in Mac OS 10.7). Otherwise (no simple accessor
method is found), searches the class of the receiver for methods
whose names match the patterns -countOf<Key> and -
indexIn<Key>OfObject: and -objectIn<Key>AtIndex: (corresponding to
the primitive methods defined by the NSOrderedSet class) and also -
<key>AtIndexes: (corresponding to -[NSOrderedSet
objectsAtIndexes:]). If a count method and an indexOf method and at
least one of the other two possible methods are found, a collection
proxy object that responds to all NSOrderedSet methods is returned.
Each NSOrderedSet message sent to the collection proxy object will
result in some combination of -countOf<Key>, -
indexIn<Key>OfObject:, -objectIn<Key>AtIndex:, and -<key>AtIndexes:
messages being sent to the original receiver of -valueForKey:. If
the class of the receiver also implements an optional method whose
name matches the pattern -get<Key>:range: that method will be used
when appropriate for best performance.
    3\. Otherwise (no simple accessor method or set of ordered set
access methods is found), searches the class of the receiver for
methods whose names match the patterns -countOf<Key> and -
objectIn<Key>AtIndex: (corresponding to the primitive methods
defined by the NSArray class) and (introduced in Mac OS 10.4) also
-<key>AtIndexes: (corresponding to -[NSArray objectsAtIndexes:]).
If a count method and at least one of the other two possible
methods are found, a collection proxy object that responds to all
NSArray methods is returned. Each NSArray message sent to the
collection proxy object will result in some combination of -
countOf<Key>, -objectIn<Key>AtIndex:, and -<key>AtIndexes: messages
being sent to the original receiver of -valueForKey:. If the class
of the receiver also implements an optional method whose name
matches the pattern -get<Key>:range: that method will be used when
appropriate for best performance.
    4 (introduced in Mac OS 10.4). Otherwise (no simple accessor
method or set of ordered set or array access methods is found),
searches the class of the receiver for a threesome of methods whose
names match the patterns -countOf<Key>, -enumeratorOf<Key>, and -
memberOf<Key>: (corresponding to the primitive methods defined by
the NSSet class). If all three such methods are found a collection
proxy object that responds to all NSSet methods is returned. Each
```

```
NSSet message sent to the collection proxy object will result in
some combination of -countOf<Key>, -enumeratorOf<Key>, and -
memberOf<Key>: messages being sent to the original receiver of -
valueForKey:.
    5\. Otherwise (no simple accessor method or set of collection
access methods is found), if the receiver's class'
+accessInstanceVariablesDirectly property returns YES, searches the
class of the receiver for an instance variable whose name matches
the pattern _<key>, _is<Key>, <key>, or is<Key>, in that order. If
such an instance variable is found, the value of the instance
variable in the receiver is returned, with the same sort of
conversion to NSNumber or NSValue as in step 1.
    6\. Otherwise (no simple accessor method, set of collection
access methods, or instance variable is found), invokes -
valueForUndefinedKey: and returns the result. The default
implementation of -valueForUndefinedKey: raises an
NSUndefinedKeyException, but you can override it in your
application.
```

简而言之：

1.访问器匹配：先寻找与key，isKey， getKey (实测还有_key)同名的方法，返回值为对象类型。

2.实例变量匹配：寻找与key， _key，isKey，_isKey同名的实例变量

setValueForKey:

```
The default implementation of this method does the following:
    1\. Searches the class of the receiver for an accessor method
whose name matches the pattern -set<Key>:. If such a method is
found the type of its parameter is checked. If the parameter type
is not an object pointer type but the value is nil -
setNilValueForKey: is invoked. The default implementation of -
setNilValueForKey: raises an NSInvalidArgumentException, but you
can override it in your application. Otherwise, if the type of the
method's parameter is an object pointer type the method is simply
invoked with the value as the argument. If the type of the method's
parameter is some other type the inverse of the NSNumber/NSValue
conversion done by -valueForKey: is performed before the method is
invoked.
    2\. Otherwise (no accessor method is found), if the receiver's
class' +accessInstanceVariablesDirectly property returns YES,
searches the class of the receiver for an instance variable whose
```

```
name matches the pattern _<key>, _is<Key>, <key>, or is<Key>, in
that order. If such an instance variable is found and its type is
an object pointer type the value is retained and the result is set
in the instance variable, after the instance variable's old value
is first released. If the instance variable's type is some other
type its value is set after the same sort of conversion from
NSNumber or NSValue as in step 1.
    3\. Otherwise (no accessor method or instance variable is
found), invokes -setValue:forUndefinedKey:. The default
implementation of -setValue:forUndefinedKey: raises an
NSUndefinedKeyException, but you can override it in your
application.
```

简而言之：

1.存取器匹配：先寻找与setKey同名的方法,且参数要为一个对象类型

2.实例变量匹配：寻找与key，_isKey，_key，isKey同名的实例变量，直接赋值。

其他

当我们使用 `id objectA = ObjectB.value2` ;时是否代表objectB有一个value2的属性呢，实际上不一定，"."操作只是去寻找一个名称匹配参数匹配的方法，我们习以为常的引用属性只是因为属性刚好有getter，setter符合要求而已，属性的实质为一个实例变量加存取方法(有些实例变量没有存取方法，而有些存取方法并没有对应的实例变量)... 例如 `object.class` ，`NSObject` 中 并没有 `class` 属性，只有一个 `class` 方法。 KVC是一种高效的取设值的方法，而无论这个键是否暴露出来。

KVO演示代码

ASClassA.h

```
#import <Foundation/Foundation.h>
#import <objc/runtime.h>

@interface ASClassA : NSObject

@property (nonatomic, assign) NSUInteger value;
```

```
@property (nonatomic, assign) IMP imp;

@property (nonatomic, assign) IMP classImp;

@end
```

ASClassA.m

```
#import "ASClassA.h"

@implementation ASClassA

- (void)setValue:(NSUInteger)value {
    _value = value;
}

- (IMP)imp {
    return [self methodForSelector:@selector(setValue:)];
}

- (IMP)classImp {
    return [self methodForSelector:@selector(class)];
}

@end
```

ASClassB.h

```
#import <Foundation/Foundation.h>

@interface ASClassB : NSObject

- (NSString *)classssss;

- (void)setClassssss;

@end
```

ASClassB.m

```
#import "ASClassB.h"

@implementation ASClassB

-(void)observeValueForKeyPath:(NSString *)keyPath ofObject:
(id)object change:(NSDictionary<NSString *,id> *)change context:
(void *)context{
    NSLog(@"B接收到变化");
}

- (NSString *)classssss {
    return @"classssss";
}

- (void)setClassssss {
}

@end
```

ASClassC.h

```
#import <Foundation/Foundation.h>

@interface ASClassC : NSObject

@end
```

ASClassC.m

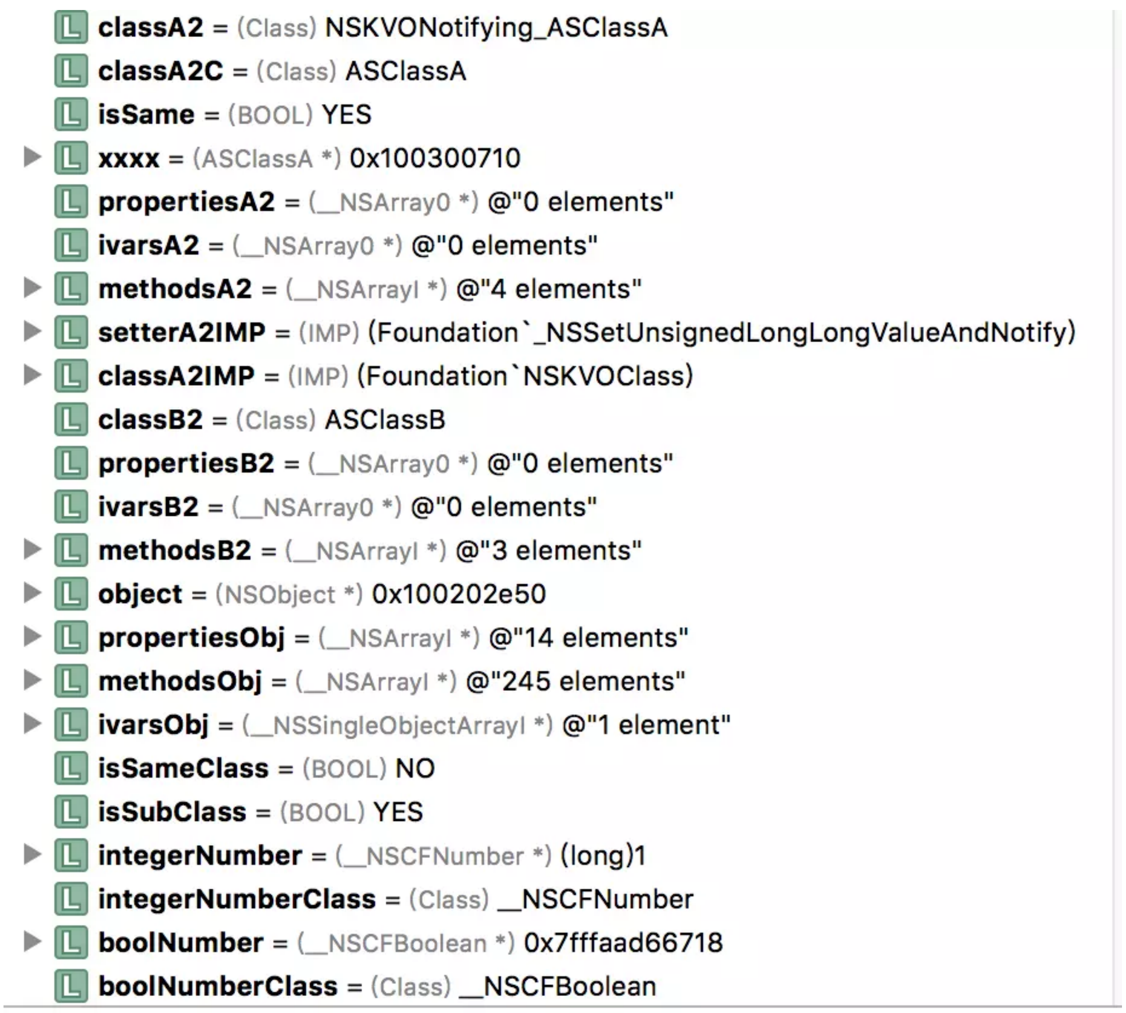
```
#import "ASClassC.h"

@implementation ASClassC

-(void)observeValueForKeyPath:(NSString *)keyPath ofObject:
(id)object change:(NSDictionary<NSString *,id> *)change context:
(void *)context{
    NSLog(@"C接收到变化");
}

@end
```

main.m

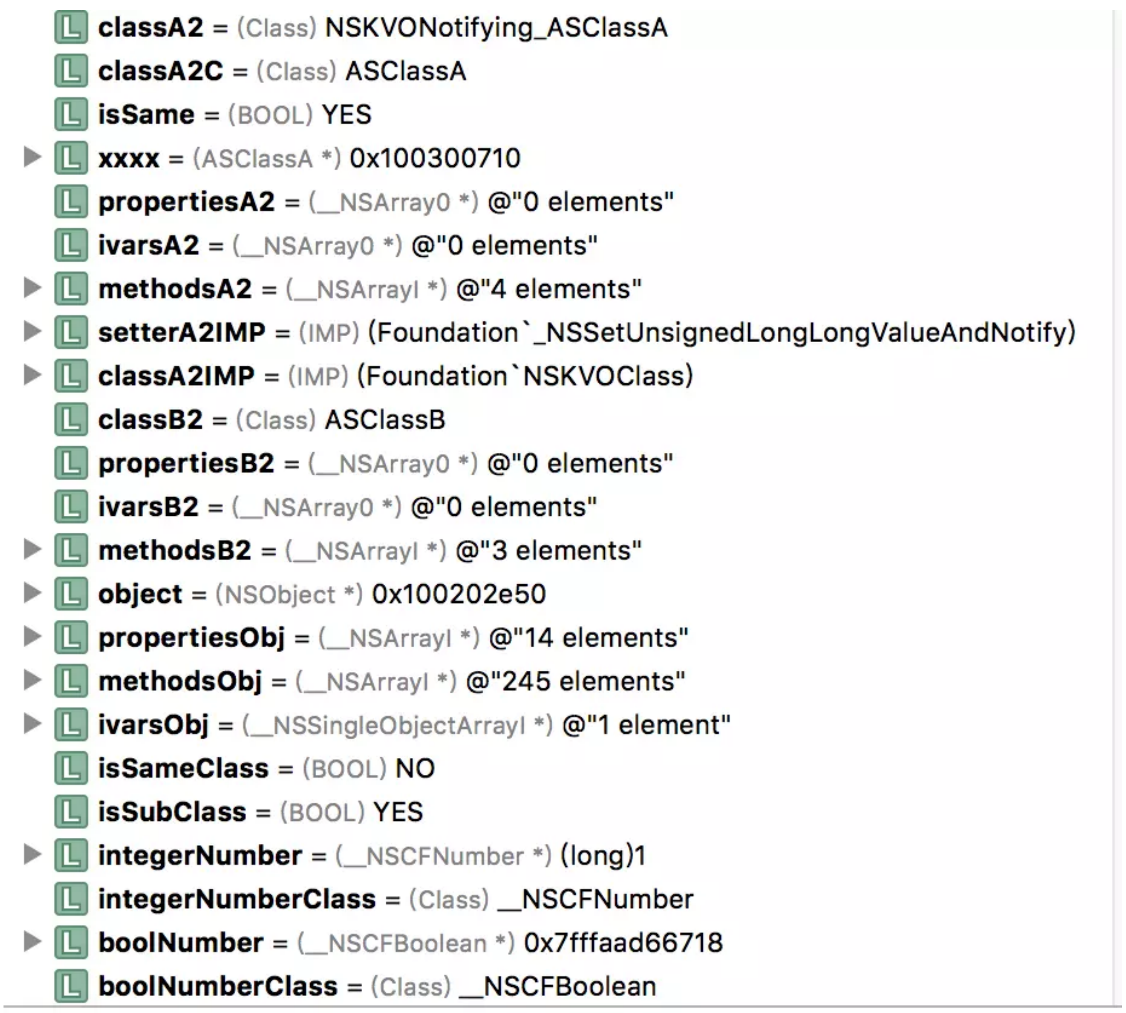
```
#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wunused-variable"

#import <Foundation/Foundation.h>
#import "ASClassA.h"
#import "ASClassB.h"
#import "ASClassC.h"

NSArray<NSString *> *getProperties(Class aClass) {
    unsigned int count;
    objc_property_t *properties = class_copyPropertyList(aClass,
&count);
    NSMutableArray *mArray = [NSMutableArray array];
    for (int i = 0; i < count; i++) {
        objc_property_t property = properties[i];
        const char *cName = property_getName(property);
        NSString *name = [NSString stringWithCString:cName
encoding:NSUTF8StringEncoding];
        [mArray addObject:name];
    }
    return mArray.copy;
}

NSArray<NSString *> *getIvars(Class aClass) {
    unsigned int count;
    Ivar *ivars = class_copyIvarList(aClass, &count);
    NSMutableArray *mArray = [NSMutableArray array];
    for (int i = 0; i < count; i++) {
        Ivar ivar = ivars[i];
        const char *cName = ivar_getName(ivar);
        NSString *name = [NSString stringWithCString:cName
encoding:NSUTF8StringEncoding];
        [mArray addObject:name];
    }
    return mArray.copy;
}
```

```
NSArray<NSString *> *getMethods(Class aClass) {
    unsigned int count;
    Method *methods = class_copyMethodList(aClass, &count);
    NSMutableArray *mArray = [NSMutableArray array];
    for (int i = 0; i < count; i++) {
        Method method = methods[i];
        SEL selector = method_getName(method);
        NSString *selectorName = NSStringFromSelector(selector);
        [mArray addObject:selectorName];
    }
    return mArray.copy;
}

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        ASClassA *objectA = [[ASClassA alloc] init];
        ASClassB *objectB = [[ASClassB alloc] init];
        ASClassC *objectC = [[ASClassC alloc] init];
        NSString *bbb = objectB.classssss;
        //objectB.classssss = @"";

        Class classA1 = object_getClass(objectA);
        Class classA1C = [objectA class]; // objectA.class;
        NSLog(@"before objectA: %@", classA1);
        NSArray *propertiesA1 = getProperties(classA1);
        NSArray *ivarsA1 = getIvars(classA1);
        NSArray *methodsA1 = getMethods(classA1);
        IMP setterA1IMP = objectA.imp;
        IMP classA1IMP = objectA.classImp;

           Class classB1 = object_getClass(objectB);
           NSLog(@"before objectA: %@", classB1);
           NSArray *propertiesB1 = getProperties(classB1);
           NSArray *ivarsB1 = getIvars(classB1);
           NSArray *methodsB1 = getMethods(classB1);

        [objectA addObserver:objectB forKeyPath:@"value"
options:NSKeyValueObservingOptionOld|NSKeyValueObservingOptionNew
context:nil];
        [objectA addObserver:objectC forKeyPath:@"value"
options:NSKeyValueObservingOptionOld|NSKeyValueObservingOptionNew
context:nil];

        Class classA2 = object_getClass(objectA);
        Class classA2C = [objectA class];
        BOOL isSame = [objectA isEqual:[objectA self]];
        id xxxx = [[classA2 alloc] init];
        NSLog(@"after objectA: %@", classA2);
        NSArray *propertiesA2 = getProperties(classA2);
        NSArray *ivarsA2 = getIvars(classA2);
        NSArray *methodsA2 = getMethods(classA2);
        IMP setterA2IMP = objectA.imp;
        IMP classA2IMP = objectA.classImp;

          Class classB2 = object_getClass(objectB);
          NSLog(@"before objectA: %@", classB2);
          NSArray *propertiesB2 = getProperties(classB2);
          NSArray *ivarsB2 = getIvars(classB2);
          NSArray *methodsB2 = getMethods(classB2);

             NSObject *object = [[NSObject alloc] init];
             NSArray *propertiesObj = getProperties([object
class]);
             NSArray *methodsObj = getMethods([object class]);
             NSArray *ivarsObj = getIvars([object class]);

        BOOL isSameClass = [classA1 isEqual:classA2];
        BOOL isSubClass = [classA2 isSubclassOfClass:classA1];

        objectA.value = 10;
        [objectA removeObserver:objectB forKeyPath:@"value"];
        [objectA removeObserver:objectC forKeyPath:@"value"];

        NSNumber *integerNumber = [NSNumber numberWithInteger:1];
        Class integerNumberClass = object_getClass(integerNumber);
        NSNumber *boolNumber = [NSNumber numberWithBool:YES];
        Class boolNumberClass = object_getClass(boolNumber);
    }
    return 0;
}

#pragma clang diagnostic pop
```

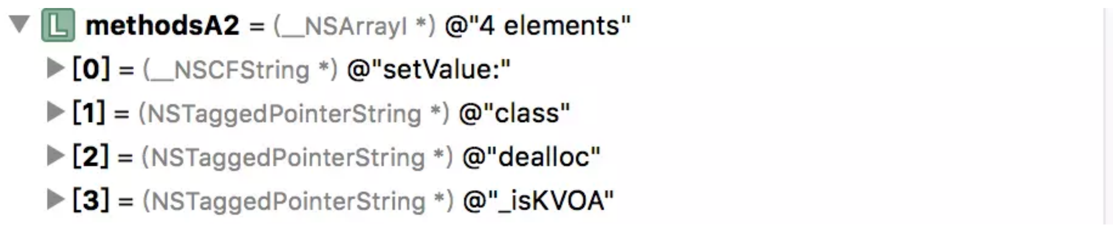
运行结果



```
classA2 = (Class) NSKVONotifying_ASClassA
classA2C = (Class) ASClassA
isSame = (BOOL) YES
xxxx = (ASClassA *) 0x100300710
propertiesA2 = (__NSArray0 *) @"0 elements"
ivarsA2 = (__NSArray0 *) @"0 elements"
methodsA2 = (__NSArrayI *) @"4 elements"
setterA2IMP = (IMP) (Foundation`_NSSetUnsignedLongLongValueAndNotify)
classA2IMP = (IMP) (Foundation`NSKVOClass)
classB2 = (Class) ASClassB
propertiesB2 = (__NSArray0 *) @"0 elements"
ivarsB2 = (__NSArray0 *) @"0 elements"
methodsB2 = (__NSArrayI *) @"3 elements"
object = (NSObject *) 0x100202e50
propertiesObj = (__NSArrayI *) @"14 elements"
methodsObj = (__NSArrayI *) @"245 elements"
ivarsObj = (__NSSingleObjectArrayI *) @"1 element"
isSameClass = (BOOL) NO
isSubClass = (BOOL) YES
integerNumber = (__NSCFNumber *) (long)1
integerNumberClass = (Class) __NSCFNumber
boolNumber = (__NSCFBoolean *) 0x7fffaad66718
boolNumberClass = (Class) __NSCFBoolean
```



```
classA2 = (Class) NSKVONotifying_ASClassA
classA2C = (Class) ASClassA
isSame = (BOOL) YES
▶ xxxx = (ASClassA *) 0x100300710
propertiesA2 = (__NSArray0 *) @"0 elements"
ivarsA2 = (__NSArray0 *) @"0 elements"
▶ methodsA2 = (__NSArrayI *) @"4 elements"
▶ setterA2IMP = (IMP) (Foundation`_NSSetUnsignedLongLongValueAndNotify)
▶ classA2IMP = (IMP) (Foundation`NSKVOClass)
classB2 = (Class) ASClassB
propertiesB2 = (__NSArray0 *) @"0 elements"
ivarsB2 = (__NSArray0 *) @"0 elements"
▶ methodsB2 = (__NSArrayI *) @"3 elements"
▶ object = (NSObject *) 0x100202e50
▶ propertiesObj = (__NSArrayI *) @"14 elements"
▶ methodsObj = (__NSArrayI *) @"245 elements"
▶ ivarsObj = (__NSSingleObjectArrayI *) @"1 element"
isSameClass = (BOOL) NO
isSubClass = (BOOL) YES
▶ integerNumber = (__NSCFNumber *) (long)1
integerNumberClass = (Class) __NSCFNumber
▶ boolNumber = (__NSCFBoolean *) 0x7fffaad66718
boolNumberClass = (Class) __NSCFBoolean
```



```
▼ methodsA2 = (__NSArrayI *) @"4 elements"
  ▶ [0] = (__NSCFString *) @"setValue:"
  ▶ [1] = (NSTaggedPointerString *) @"class"
  ▶ [2] = (NSTaggedPointerString *) @"dealloc"
  ▶ [3] = (NSTaggedPointerString *) @"_isKVOA"
```

分析以上结果

我们通过抓取objectA在被objectB, objectC观察前和观察后的 类的类型，属性列表，变量列表，方法列表，得出：

① class: 被观察前， `objectA` 为 `ASClassA` 类型， 被观察后，变为了 `NSKVONotifying_ASClassA` 类型，且这个类为ASClassA的子类(通过isa指向改变，事实上， `object_getClass(objectA)` 和 `objectA->isa` 方法等价)。

② 属性，实例变量：无变化。

③ 方法列表： `NSKVONotifying_ASClassA` 出现了四个新的方法，

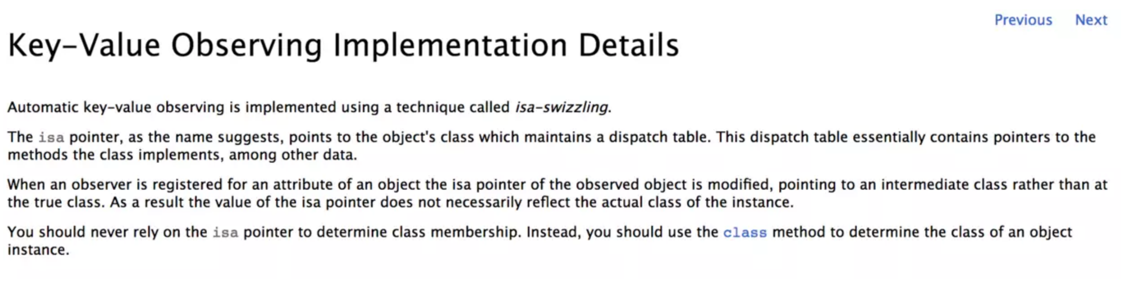
我们可以注意到，被观察的值setValue:方法的实现由 `([ASClassA setValue:] at ASClassA.m)` 变为了 `(Foundation_NSSetUnsignedLongLongValueAndNotify)`。这个被重写的setter方法在原有的实现前后插入了 `[self willChangeValueForKey:@“name”];` 调用存取方法之前总调 `[super setValue:newName forKey:@”name”]; [self didChangeValueForKey:@”name”];` 等，以触发观察者的响应。 然后 `class` 方法由 `(libobjc.A.dylib -[NSObject class])` 变为了 `(Foundation_NSKVOClass)`，

这也解释了我们在被观察前被观察后执行 `[objectA class]` 方法得到结果不同的原因， `-(Class)class` 方法的实现本来就是 `object_getClass` ，但在被观察后 `class` 方法和 `object_getClass` 结果却不一样，事实是 `class` 方法被重写了， `class` 方法总能得到 `ASClassA`

`dealloc` 方法: 观察移除后使class变回去ASClassA(通过isa指向)， `_isKVO` : 判断被观察者自己是否同时也观察了其他对象

事实上

苹果开发者文档


Previous Next

Key-Value Observing Implementation Details

Automatic key-value observing is implemented using a technique called *isa-swizzling*.

The `isa` pointer, as the name suggests, points to the object's class which maintains a dispatch table. This dispatch table essentially contains pointers to the methods the class implements, among other data.

When an observer is registered for an attribute of an object the isa pointer of the observed object is modified, pointing to an intermediate class rather than at the true class. As a result the value of the isa pointer does not necessarily reflect the actual class of the instance.

You should never rely on the `isa` pointer to determine class membership. Instead, you should use the `class` method to determine the class of an object instance.

简而言之，苹果使用了一种isa交换的技术，当 `objectA` 被观察后， `objectA` 对象的isa指针被指向了一个新建的 `ASClassA` 的子类 `NSKVONotifying_ASClassA` ，且这个子类重写了被观察值的setter方法和 `class` 方法， `dealloc` 和 `_isKVO` 方法，然后使 `objectA` 对象的isa指针指向这个新建的类，然后事实上 `objectA` 变为了NSKVONotifying_ASClassA的实例对象，执行方法要从这个类的方法列表里找。(同时苹果警告我们，通过isa获取类的类型是不可靠的，通过class方法总是能得到正确的类=_=!!).

更多关于OC对象，类，isa, 属性， 变量， 方法的介绍请参考我的另一篇博文 Runtime简介

思考

由于研究方法有限，并不能知道被观察者的值改变后，以何种方式去通知观察者，并使其执行实现的对应的方法的，我们可以猜想，也许是苹果惯用的，像维护对象们的引用计数，和weak修饰的对象的存亡 一样，建立了一张hash表去对应观察者，被观察者的地址或其他。能力有限，尚不能得知。

谢谢观看

欢迎大家指出文中的错误！

演示代码地址：https://github.com/Assuner-Lee/KVC-KVO-Test.git