sucess_flower
2020年3月7日

Runtime学习笔记

Talk is cheap , Show me the code

1.0 消息转发机制

```
/**
 消息转发机制
    动态方法解析
    快速转发
    慢速转发
 */

//MARK : - 面试题
/**
    你为App崩溃做过什么事情
    增强健壮性，用动态解析
 */
int main(int argc, const char * argv[]) {
    @autoreleasepool {

        objc_msgSend([Person new],
@selector(sendMessage:), @"Hello World");
        //'-[Person sendMessage:]: unrecognized selector
    }
    return 0;
}
```

```
//- (void)sendMessage:(NSString *)text{
//    NSLog(@"_______   %@", text);
//}

//MARK : - 动态方法解析
void sendMessage(id self, SEL _cmd, NSString *msg){
    NSLog(@" 动态解析 ： %@", msg);
}

+ (BOOL)resolveInstanceMethod:(SEL)sel{
    /// >.  匹配方法
//    NSString *methodName = NSStringFromSelector(sel);
//    if ([methodName isEqualToString:@"sendMessage:"]) {
//
//      return class_addMethod(self, sel,
(IMP)sendMessage, "v@:@");
//    }
    return NO;
}
```

```
//MARK : - 快速转发
- (id)forwardingTargetForSelector:(SEL)aSelector{

    /// >. 找一个接收者
//    NSString *methodName =
NSStringFromSelector(aSelector);
//    if ([methodName isEqualToString:@"sendMessage:"]) {
//
//      return [Student new];
//    }
    return [super forwardingTargetForSelector:aSelector];
}

//MARK : - 慢速转发
/// >1. 方法签名
/// >2. 消息转发
- (NSMethodSignature
*)methodSignatureForSelector:(SEL)aSelector{

    NSString *methodName =
NSStringFromSelector(aSelector);
    if ([methodName isEqualToString:@"sendMessage:"]) {
      return [NSMethodSignature
signatureWithObjCTypes:"v@:@"];
    }
    return [super methodSignatureForSelector:aSelector];
}

- (void)forwardInvocation:(NSInvocation *)anInvocation{
    SEL sel = [anInvocation selector];
    Student *s = [Student new];
    if ([s respondsToSelector:sel]) {/// >. 如果该实例响应这
个方法
        return [anInvocation invokeWithTarget:s]; /// >.
将接受者传递过去
    }else{
        [super forwardInvocation:anInvocation];
    }
    [super forwardInvocation:anInvocation];
}
```

```
- (void)doesNotRecognizeSelector:(SEL)aSelector{
    NSLog(@"找不到此方法");
}
@end

po object_getClass(d2)
```

2.0 方法交换

```
+ (void)load{
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        Method originM = class_getInstanceMethod(self,
@selector(reloadData));
        Method swizzledM = class_getInstanceMethod(self,
@selector(sf_reloadData));
        method_exchangeImplementations(originM,
swizzledM);
    });
}

- (void)sf_reloadData{
    /// >. 调用 reloadData 方法
    [self sf_reloadData];
    [self fillDefaultView];
}

- (void)fillDefaultView{
    id<UITableViewDataSource> dataSource =
self.dataSource;
    NSInteger section = [dataSource
respondsToSelector:@selector(numberOfSectionsInTableView:
)]
        ?[dataSource numberOfSectionsInTableView:self] :
1;
    NSInteger rows = 0;
    for (NSInteger i = 0; i < section; i++) {
        rows = [dataSource tableView:self
numberOfRowsInSection:section];
    }
```

```
    if (!rows) {
        self.delfaultView = [[UIView alloc]
initWithFrame:CGRectMake(0, 0, self.frame.size.width,
self.frame.size.height)];
        self.delfaultView.backgroundColor = [UIColor
redColor];
        [self addSubview:self.delfaultView];
    }else{
        self.delfaultView.hidden = YES;
    }
}

//MARK : - Setter and Getter
- (void)setDelfaultView:(UIView *)delfaultView{
    objc_setAssociatedObject(self,
@selector(setDelfaultView:), delfaultView,
OBJC_ASSOCIATION_RETAIN_NONATOMIC);
}

- (UIView *)delfaultView{
   return objc_getAssociatedObject(self,
@selector(setDelfaultView:));
}
```

3.0 模型字典互转

```
/**
 Key - Value
 set方法
 */
- (instancetype)initWithDic:(NSDictionary *)dic{
    self = [super init];
    if (self) {
        for (NSString *key in dic.allKeys) {
            id value = dic[key];
            NSString *methodName = [NSString
stringWithFormat:@"set%@:", key.capitalizedString];
            SEL sel = NSSelectorFromString(methodName);
            if (sel) {
                objc_msgSend(self, sel, value);
```

```
            }
        }
    }
    return self;
}

- (NSDictionary *)convertModelToDic{

    unsigned int count ;
    objc_property_t *propertyList =
class_copyPropertyList([self class], &count);
    if (count > 0) {
        NSMutableDictionary *dic = [NSMutableDictionary
dictionary];
        for (NSInteger i = 0; i < count; i++) {
            const void *propertyName =
property_getName(propertyList[i]);
            NSString *name = [NSString
stringWithUTF8String:propertyName];
            SEL sel = NSSelectorFromString(name);
            if (sel) {
              id value = objc_msgSend(self, sel);
                if (value) {
                    dic[name] = value;
                }else{
                    dic[name] = @"";
                }
            }
        }
        free(propertyList);
        return dic;
    }

    return nil;
}
```

4.0 自定义KVO

```
void myMethod(id self, SEL _cmb, NSString *text){

    /// >1. 这个是子类，调用父类的方法
    /// >2. 调用父类的方法
    /// >3. 通知观察者
    /// >4. 直接有一个父类的结构体
    /**
     struct objc_super {
         /// Specifies an instance of a class.
         __unsafe_unretained _Nonnull id receiver;

         /// Specifies the particular superclass of the
instance to message.
     #if !defined(__cplusplus)  &&  !__OBJC2__
         /* For compatibility with old objc-runtime.h
header */
//          __unsafe_unretained _Nonnull Class class;
//     #else
//          __unsafe_unretained _Nonnull Class
super_class;
//     #endif
         /* super_class is the first class to search */
   //     };

    struct objc_super superClass = {
        self,
        class_getSuperclass([self class])
    };
    /// >5. 给结构体发送消息
    objc_msgSendSuper(&superClass, _cmb, text);
    /// >6. 获取监听者
    id observer = objc_getAssociatedObject(self,
(__bridge const void *)@"objc");
    /// >7. 通知改变
    NSString *methodName = NSStringFromSelector(_cmb);
    NSString *key = getValueKey(methodName);
    objc_msgSend(observer,
@selector(observeValueForKeyPath:ofObject:change:context:
),
                 key, self, @{key: text} ,nil);
}
```

```
NSString * getValueKey(NSString *setter){
    NSRange range = NSMakeRange(3, setter.length - 4);
    NSString *key = [setter substringWithRange:range];
    NSString *letter = [[key substringToIndex:1]
lowercaseString];
    key = [key
stringByReplacingCharactersInRange:NSMakeRange(0, 1)
withString:letter];
    return key;
}

- (void)sf_addObserver:(NSObject *)observer
forKeyPath:(NSString *)keyPath
options:(NSKeyValueObservingOptions)options  context:(void
*)context{

    /// >1. 获取当前类的名字
    NSString *oldName = NSStringFromClass([self class]);
    NSString *newName = [NSString
stringWithFormat:@"CustomKVO_%@", oldName];
    /// >2. 创建一个类
    Class customClass = objc_allocateClassPair([self
class], newName.UTF8String, 0);
    /// >3. 注册类
    objc_registerClassPair(customClass);
    /// >4. 修改指针的指向
   // object_isClass(customClass);
    object_setClass(self, customClass);
    /// >5. 重写set 方法
    NSString *setterName = [NSString
stringWithFormat:@"set%@:", keyPath.capitalizedString];
    SEL sel = NSSelectorFromString(setterName);
    /// >6. 添加方法实现
    class_addMethod(customClass, sel, (IMP)myMethod,
"v@:@");
    /// >7. 关联一个属性
    objc_setAssociatedObject(self, (__bridge const void
*)@"objc", observer, OBJC_ASSOCIATION_ASSIGN);
}
```

```
    Dog *d1 = [Dog new];
    Dog *d2 = [Dog new];
```

```
    d2.name = @"123";

    NSLog(@"监听之前 p1: %p  p2 : %p",
          [d1 methodForSelector:@selector(setName:)],
          [d2 methodForSelector:@selector(setName:)]);

//    [d1 sf_addObserver:self forKeyPath:@"name"
options:NSKeyValueObservingOptionNew |
NSKeyValueObservingOptionOld context:nil];
    [d1 sf_addObserver:self forKeyPath:@"name"
options:NSKeyValueObservingOptionNew |
NSKeyValueObservingOptionOld  context:nil];

    NSLog(@"监听之后 p1: %p  p2 : %p",
          [d1 methodForSelector:@selector(setName:)],
          [d2 methodForSelector:@selector(setName:)]);
    d1.name = @"aaa";
    d1.name = @“bbb”;
```

```
- (void)observeValueForKeyPath:(NSString *)keyPath
ofObject:(id)object
change:(NSDictionary<NSKeyValueChangeKey,id> *)change
context:(void *)context {

    NSLog(@"change = %@", change);
}
```