iOS开发·runtime+KVC实现多层字典模型转换（多层数据：模型嵌套模型，模型嵌套数组，数组嵌套模型）

本文实验Demo传送门：DictToModelDemo

> **前言**：将后台JSON数据中的字典转成本地的模型，我们一般选用部分优秀的第三方框架，如SBJSON、JSONKit、MJExtension、YYModel等。但是，一些简单的数据，我们也可以尝试自己来实现转换的过程。

更重要的是，有时候在iOS面试的时候，部分面试官会不仅问你某种场景会用到什么框架，更会问你如果要你来实现这个功能，你有没有解决思路？所以，自己实现字典转模型还是有必要掌握的。有了这个基础，在利用运行时runtime的动态特性，你也可以实现这些第三方框架。

笔者的KVC系列为：

- iOS开发·KVC:字典转模型,防止因本地未定义字段（后台的字段与本地字符串名不一致）导致数据转换过程中的奔溃
- iOS开发·runtime+KVC实现字典模型转换

1. 字典转模型：KVC

当对象的属性很多的时候，我们可以利用KVC批量设置。

```
- (void)setValuesForKeysWithDictionary:(NSDictionary<NSString *,id>
*)keyedValues;
```

但是KVC批量转的时候，有个致命的缺点，就是当字典中的键，在对象属性中找不到对应的属性的时候会报错。解决办法是实现下面的方法：

```
//空的方法体也行
- (void)setValue:(id)value forUndefinedKey:(NSString *)key{}
```

> **需求**：有一个排名列表页面，这个页面的每个排名对应一个模型，这个模型从

> Plist转换得到。那么实现的代码如下所示：

- 头文件

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

@interface GloryListModel : NSObject

//图标
@property (nonatomic, copy) NSString *icon;
//标题
@property (nonatomic, copy) NSString *title;
//目标控制器
@property (nonatomic, copy) NSString *targetVC;
//菜单编号
@property (nonatomic, copy) NSString *menuCode;

+ (instancetype)gloryListModelWithDict:(NSDictionary *)dict;

+ (NSArray<GloryListModel *> *)gloryListModelsWithPlistName:
(NSString *)plistName;

@end
```

- 实现文件

```
#import "GloryListModel.h"

@implementation GloryListModel

//kvc实现字典转模型
- (instancetype)initWithDict:(NSDictionary *)dict{
    if (self = [super init]) {
        [self setValuesForKeysWithDictionary:dict];
    }
    return self;
}

//防止与后台字段不匹配而造成崩溃
- (void)setValue:(id)value forUndefinedKey:(NSString *)key{}

+ (instancetype)gloryListModelWithDict:(NSDictionary *)dict;{
    return [[self alloc]initWithDict:dict];
```

```
}

+ (NSArray<GloryListModel *> *)gloryListModelsWithPlistName:
(NSString *)plistName;{
    //获取路径
    NSString *path = [[NSBundle
mainBundle]pathForResource:plistName ofType:@"plist"];
    //读取plist
    NSArray *dictArr = [NSArray arrayWithContentsOfFile:path];
    //字典转模型
    NSMutableArray *modelArr = [NSMutableArray array];
    [dictArr enumerateObjectsUsingBlock:^(NSDictionary *dict,
NSUInteger idx, BOOL * _Nonnull stop) {
        [modelArr addObject:[self gloryListModelWithDict:dict]];
    }];
    return modelArr.copy;
}

@end
```

1.2 KVC字典转模型弊端

弊端：必须保证，模型中的属性和字典中的key一一对应。 如果不一致，就会调用[<Status 0x7fa74b545d60> setValue:forUndefinedKey:] 报key找不到的错。

分析:模型中的属性和字典的key不一一对应，系统就会调用`setValue:forUndefinedKey:`报错。

解决:重写对象的`setValue:forUndefinedKey:`,把系统的方法覆盖， 就能继续使用KVC，字典转模型了。

```
- (void)setValue:(id)value forUndefinedKey:(NSString *)key{
}
```

2. 字典转模型： Runtime

- **思路1**：利用运行时，首先要遍历参数字典, 如果我们获取得属性列表中包含了字典中的 key，就利用 KVC 方法赋值，然后就完成了字典转模型的操作。

- **思路2**：利用运行时，遍历模型中所有属性，根据模型的属性名，去字典中查找key，取出对应的值，给模型的属性赋值，然后就完成了字典转模型的操作。

至于实现途径，可以提供一个NSObject分类，专门字典转模型，以后所有模型都可以通过这个分类转。

2.1 先遍历被转换的字典

- 分类实现：NSObject+EnumDictOneLevel.m

```
#import "NSObject+EnumDictOneLevel.h"
#import <objc/runtime.h>

const char *kCMPropertyListKey1 = "CMPropertyListKey1";

@implementation NSObject (EnumDictOneLevel)

+ (instancetype)cm_modelWithDict1:(NSDictionary *)dict
{
    /* 实例化对象 */
    id model = [[self alloc]init];

    /* 使用字典,设置对象信息 */
    /* 1\. 获得 self 的属性列表 */
    NSArray *propertyList = [self cm_objcProperties];

    /* 2\. 遍历字典 */
    [dict enumerateKeysAndObjectsUsingBlock:^(id  _Nonnull key, id
_Nonnull obj, BOOL * _Nonnull stop) {

        /* 3\. 判断 key 是否字 propertyList 中 */
        if ([propertyList containsObject:key]) {

            // KVC字典转模型
            if (obj) {
                /* 说明属性存在,可以使用 KVC 设置数值 */
                [model setValue:obj forKey:key];
            }
        }

    }];

    /* 返回对象 */
    return model;
}
```

```
+ (NSArray *)cm_objcProperties
{
    /* 获取关联对象 */
    NSArray *ptyList = objc_getAssociatedObject(self,
kCMPropertyListKey1);

    /* 如果 ptyList 有值,直接返回 */
    if (ptyList) {
        return ptyList;
    }
    /* 调用运行时方法, 取得类的属性列表 */
    /* 成员变量:
     * class_copyIvarList(__unsafe_unretained Class cls, unsigned
int *outCount)
     * 方法:
     * class_copyMethodList(__unsafe_unretained Class cls, unsigned
int *outCount)
     * 属性:
     * class_copyPropertyList(__unsafe_unretained Class cls,
unsigned int *outCount)
     * 协议:
     * class_copyProtocolList(__unsafe_unretained Class cls,
unsigned int *outCount)
     */
    unsigned int outCount = 0;
    /**
     * 参数1: 要获取得类
     * 参数2: 雷属性的个数指针
     * 返回值: 所有属性的数组, C 语言中,数组的名字,就是指向第一个元素的地址
     */
    /* retain, creat, copy 需要release */
    objc_property_t *propertyList = class_copyPropertyList([self
class], &outCount);

    NSMutableArray *mtArray = [NSMutableArray array];

    /* 遍历所有属性 */
    for (unsigned int i = 0; i < outCount; i++) {
        /* 从数组中取得属性 */
        objc_property_t property = propertyList[i];
        /* 从 property 中获得属性名称 */
        const char *propertyName_C = property_getName(property);
        /* 将 C 字符串转化成 OC 字符串 */
        NSString *propertyName_OC = [NSString
stringWithCString:propertyName_C encoding:NSUTF8StringEncoding];
        [mtArray addObject:propertyName_OC];
```

```
    }

    /* 设置关联对象 */
    /**
     *  参数1 : 对象self
     *  参数2 : 动态添加属性的 key
     *  参数3 : 动态添加属性值
     *  参数4 : 对象的引用关系
     */

    objc_setAssociatedObject(self, kCMPropertyListKey1,
mtArray.copy, OBJC_ASSOCIATION_RETAIN_NONATOMIC);
    /* 释放 */
    free(propertyList);
    return mtArray.copy;

}

@end
```

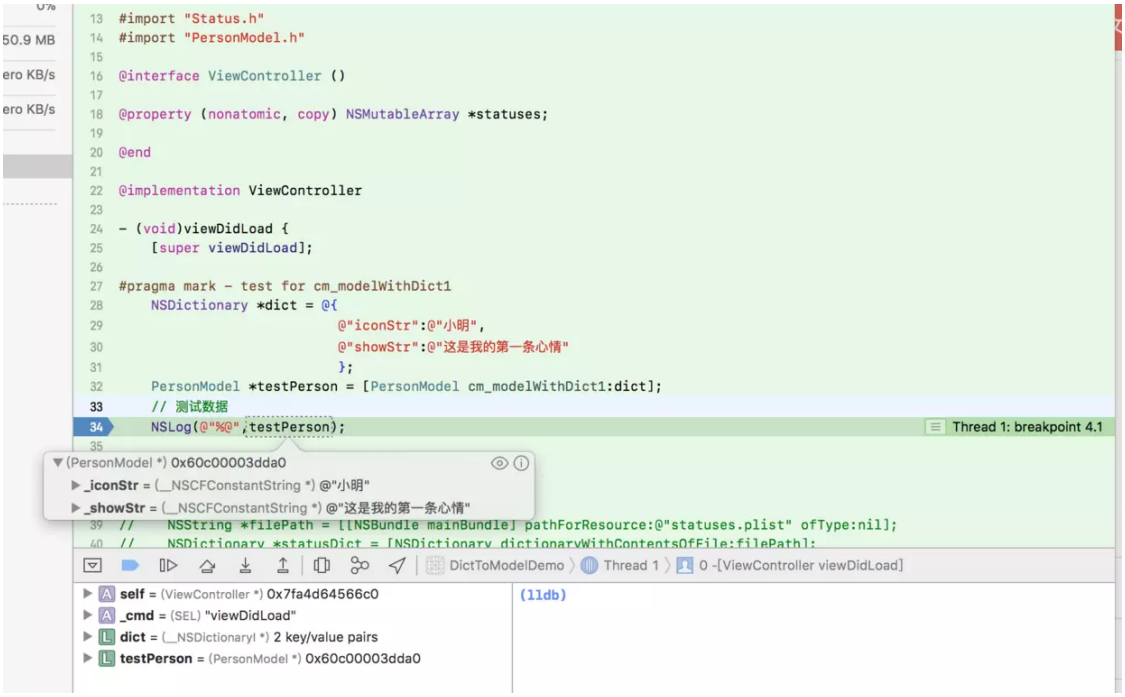
- 模型：PersonModel.h

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

@interface PersonModel : NSObject

@property (nonatomic, copy) NSString *iconStr;

@property (nonatomic, copy) NSString *showStr;

@end
```

- 调用

```
NSDictionary *dict = @{
                           @"iconStr":@"小明",
                           @"showStr":@"这是我的第一条心情"
                           };
    PersonModel *testPerson = [PersonModel cm_modelWithDict1:dict];
    // 测试数据
    NSLog(@"%@",testPerson);
```

- 运行验证



2.2 先遍历模型的成员变量数组

- 实现分类：NSObject+EnumArr.m

```
#import "NSObject+EnumArr.h"
#import <objc/message.h>

@implementation NSObject (EnumArr)

/*
 * 把字典中所有value给模型中属性赋值,
 * KVC:遍历字典中所有key,去模型中查找
 * Runtime:根据模型中属性名去字典中查找对应value,如果找到就给模型的属性赋值.
 */
// 字典转模型
+ (instancetype)modelWithDict:(NSDictionary *)dict
{
    // 创建对应模型对象
    id objc = [[self alloc] init];

    unsigned int count = 0;
```

```
    // 1.获取成员属性数组
    Ivar *ivarList = class_copyIvarList(self, &count);

    // 2.遍历所有的成员属性名,一个一个去字典中取出对应的value给模型属性赋值
    for (int i = 0; i < count; i++) {

        // 2.1 获取成员属性
        Ivar ivar = ivarList[i];

        // 2.2 获取成员属性名 C -> OC 字符串
        NSString *ivarName = [NSString
stringWithUTF8String:ivar_getName(ivar)];

        // 2.3 _成员属性名 => 字典key
        NSString *key = [ivarName substringFromIndex:1];

        // 2.4 去字典中取出对应value给模型属性赋值
        id value = dict[key];

        // 获取成员属性类型
        NSString *ivarType = [NSString
stringWithUTF8String:ivar_getTypeEncoding(ivar)];

        // 二级转换,字典中还有字典,也需要把对应字典转换成模型
        //
        // 判断下value,是不是字典
        if ([value isKindOfClass:[NSDictionary class]] && !
[ivarType containsString:@"NS"]) { //  是字典对象,并且属性名对应类型是自
定义类型
            // user User

            // 处理类型字符串 @\"User\" -> User
            ivarType = [ivarType
stringByReplacingOccurrencesOfString:@"@" withString:@""];
            ivarType = [ivarType
stringByReplacingOccurrencesOfString:@"\"" withString:@""];
            // 自定义对象,并且值是字典
            // value:user字典 -> User模型
            // 获取模型(user)类对象
            Class modalClass = NSClassFromString(ivarType);

            // 字典转模型
            if (modalClass) {
                // 字典转模型 user
                value = [modalClass modelWithDict:value];
            }
```

```
            // 字典,user
            //            NSLog(@"%@",key);
        }

        // 三级转换：NSArray中也是字典，把数组中的字典转换成模型.
        // 判断值是否是数组
        if ([value isKindOfClass:[NSArray class]]) {
            // 判断对应类有没有实现字典数组转模型数组的协议
            if ([self
respondsToSelector:@selector(arrayContainModelClass)]) {

                // 转换成id类型，就能调用任何对象的方法
                id idSelf = self;

                // 获取数组中字典对应的模型
                NSString *type =  [idSelf arrayContainModelClass]
[key];

                // 生成模型
                Class classModel = NSClassFromString(type);
                NSMutableArray *arrM = [NSMutableArray array];
                // 遍历字典数组，生成模型数组
                for (NSDictionary *dict in value) {
                    // 字典转模型
                    id model =  [classModel modelWithDict:dict];
                    [arrM addObject:model];
                }

                // 把模型数组赋值给value
                value = arrM;

            }
        }

        // 2.5 KVC字典转模型
        if (value) {

            [objc setValue:value forKey:key];
        }
    }

    // 返回对象
    return objc;

}

@end
```

- 第1层模型：Status.h，各个属性与字典对应

```
#import <Foundation/Foundation.h>
#import "NSObject+EnumArr.h"

@class PersonModel;

@interface Status : NSObject <ModelDelegate>

@property (nonatomic, strong) NSString *title;
@property (nonatomic, strong) PersonModel *person;
@property (nonatomic, strong) NSArray *cellMdlArr;

@end
```

- 第1层模型：实现文件需要指明数组里面装的类名
- Status.m

```
#import "Status.h"

@implementation Status

+ (NSDictionary *)arrayContainModelClass
{
    return @{@"cellMdlArr" : @"CellModel"};
}

@end
```

- 第2层模型：第2层模型作为第一层模型的自定义类的属性
- PersonModel.h

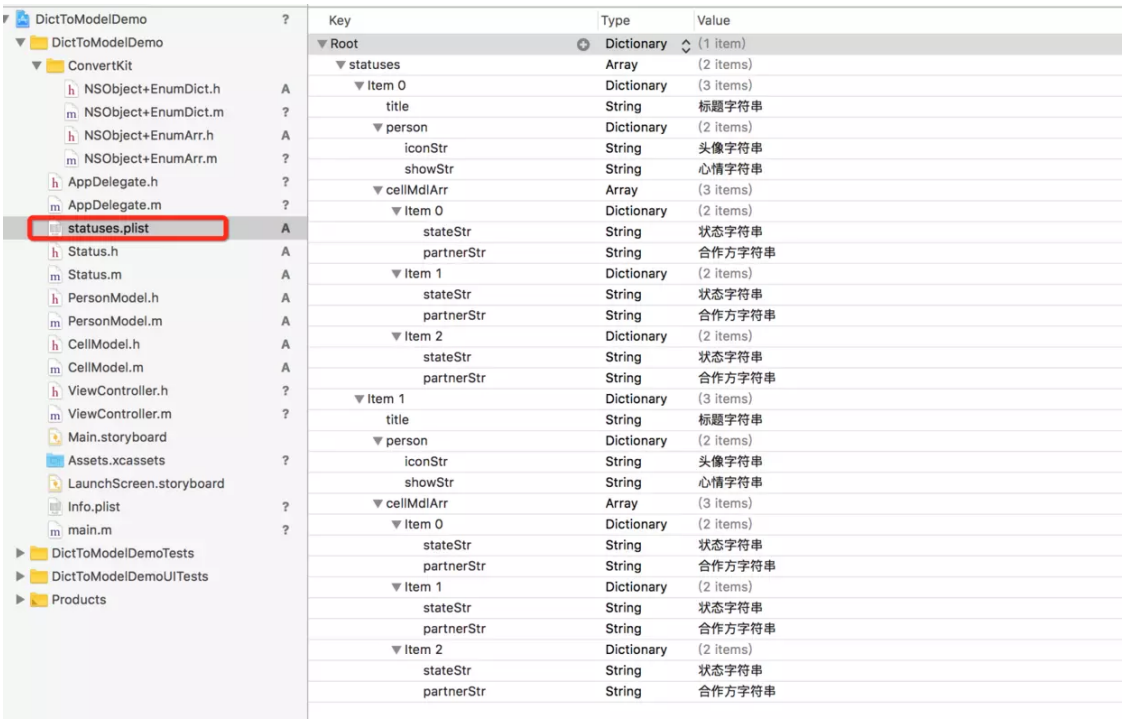
```
#import <Foundation/Foundation.h>

@interface PersonModel : NSObject
@property (nonatomic, copy) NSString *iconStr;
@property (nonatomic, copy) NSString *showStr;
```

```
@end
```

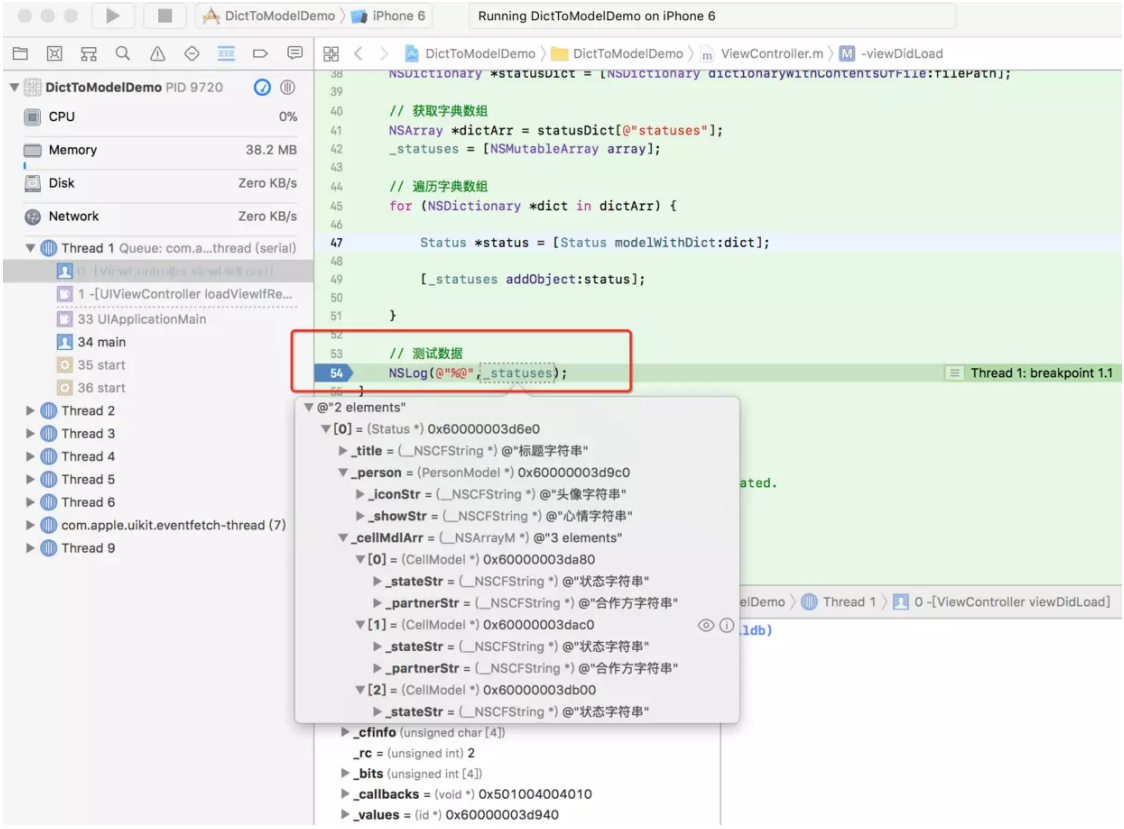
- 第2层模型：第2层模型作为第一层模型的数组类型的属性
- CellModel.h

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

@interface CellModel : NSObject

@property (nonatomic, copy) NSString *stateStr;
@property (nonatomic, copy) NSString *partnerStr;

@end
```

- 将被转换的字典



- 运行验证



2.3 对2.1的改进：2.1无法对多层数据进行转换

思路：可以模仿2.2中的递归，对2.1进行改进：模型中，除了为数组属性添加数组元素对应的类名映射字典，还要为模型属性对应的类名添加映射字典。这是因为，从字典遍历出来的key无法得知自定义类型的属性的类名。

- Status.m

```
+ (NSDictionary *)dictWithModelClass
{
    return @{@"person" : @"PersonModel"};
}
```

- NSObject+EnumDict.m

```
#import "NSObject+EnumDict.h"

//导入模型
#import "Status.h"
```

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

@implementation NSObject (EnumDict)

const char *kCMPropertyListKey = "CMPropertyListKey";

+ (instancetype)cm_modelWithDict:(NSDictionary *)dict
{
    /* 实例化对象 */
    id model = [[self alloc]init];

    /* 使用字典,设置对象信息 */
    /* 1\. 获得 self 的属性列表 */
    NSArray *propertyList = [self cm_objcProperties];

    /* 2\. 遍历字典 */
    [dict enumerateKeysAndObjectsUsingBlock:^(id  _Nonnull key, id
_Nonnull obj, BOOL * _Nonnull stop) {

        /* 3\. 判断 key 是否字 propertyList 中 */
        if ([propertyList containsObject:key]) {

            // 获取成员属性类型
            // 类型经常变，抽出来
            NSString *ivarType;

            if ([obj
isKindOfClass:NSClassFromString(@"__NSCFString")]) {
                ivarType = @"NSString";
            }else if ([obj
isKindOfClass:NSClassFromString(@"__NSCFArray")]){
                ivarType = @"NSArray";
            }else if ([obj
isKindOfClass:NSClassFromString(@"__NSCFNumber")]){
                ivarType = @"int";
            }else if ([obj
isKindOfClass:NSClassFromString(@"__NSCFDictionary")]){
                ivarType = @"NSDictionary";
            }

            // 二级转换,字典中还有字典,也需要把对应字典转换成模型
            // 判断下value,是不是字典
            if ([obj
isKindOfClass:NSClassFromString(@"__NSCFDictionary")]) { //  是字典对
象,并且属性名对应类型是自定义类型
                // value:user字典 -> User模型
                // 获取模型(user)类对象
```

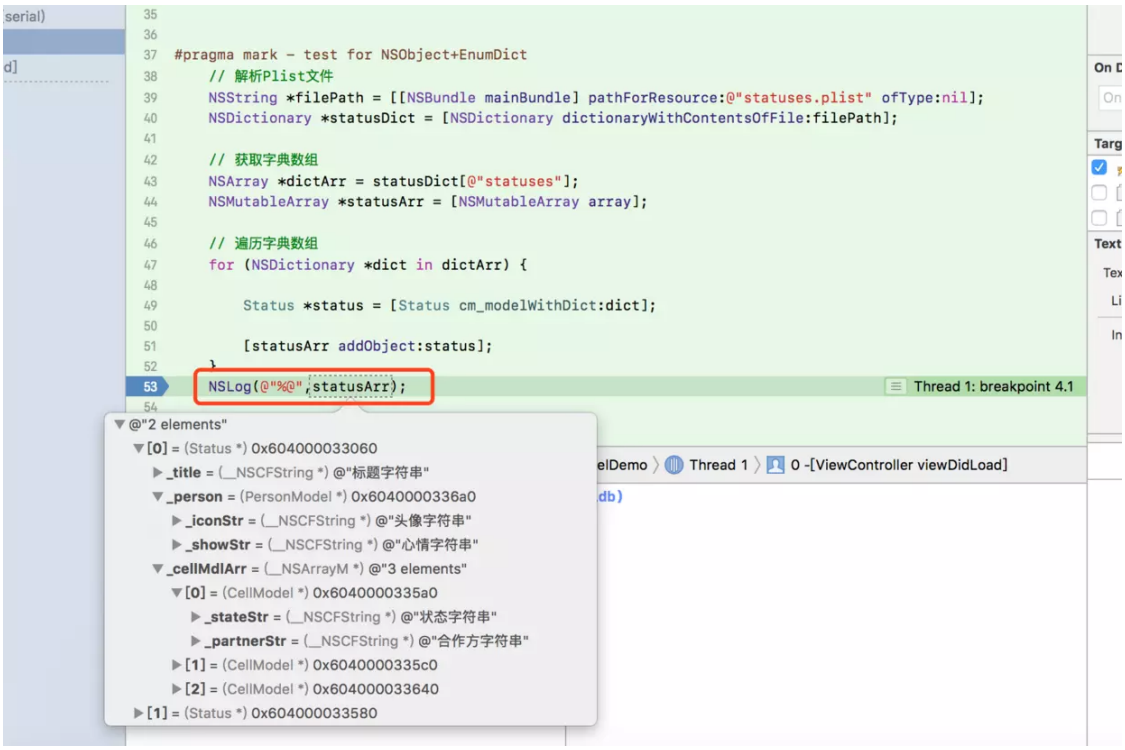
```
                NSString *ivarType = [Status dictWithModelClass]
[key];
                Class modalClass = NSClassFromString(ivarType);

                // 字典转模型
                if (modalClass) {
                    // 字典转模型 user
                    obj = [modalClass cm_modelWithDict:obj];
                }

            }

            // 三级转换：NSArray中也是字典，把数组中的字典转换成模型.
            // 判断值是否是数组
            if ([obj isKindOfClass:[NSArray class]]) {
                // 判断对应类有没有实现字典数组转模型数组的协议
                if ([self
respondsToSelector:@selector(arrayContainModelClass)]) {

                    // 转换成id类型，就能调用任何对象的方法
                    id idSelf = self;

                    // 获取数组中字典对应的模型
                    NSString *type =  [idSelf
arrayContainModelClass][key];

                    // 生成模型
                    Class classModel = NSClassFromString(type);
                    NSMutableArray *arrM = [NSMutableArray array];
                    // 遍历字典数组，生成模型数组
                    for (NSDictionary *dict in obj) {
                        // 字典转模型
                        id model =  [classModel
cm_modelWithDict:dict];
                        [arrM addObject:model];
                    }

                    // 把模型数组赋值给value
                    obj = arrM;

                }
            }

            // KVC字典转模型
            if (obj) {
                /* 说明属性存在,可以使用 KVC 设置数值 */
                [model setValue:obj forKey:key];
```

```
            }
        }

    }];

    /* 返回对象 */
    return model;
}
```

- 调用

```
// 解析Plist文件
    NSString *filePath = [[NSBundle mainBundle]
pathForResource:@"statuses.plist" ofType:nil];
    NSDictionary *statusDict = [NSDictionary
dictionaryWithContentsOfFile:filePath];

    // 获取字典数组
    NSArray *dictArr = statusDict[@"statuses"];
    NSMutableArray *statusArr = [NSMutableArray array];

    // 遍历字典数组
    for (NSDictionary *dict in dictArr) {

        Status *status = [Status cm_modelWithDict:dict];

        [statusArr addObject:status];
    }
    NSLog(@"%@",statusArr);
```

- 运行验证


#pragma mark - test for NSObject+EnumDict
// 解析Plist文件
NSString *filePath = [[NSBundle mainBundle] pathForResource:@"statuses.plist" ofType:nil];
NSDictionary *statusDict = [NSDictionary dictionaryWithContentsOfFile:filePath];
// 获取字典数组
NSArray *dictArr = statusDict[@"statuses"];
NSMutableArray *statusArr = [NSMutableArray array];
// 遍历字典数组
for (NSDictionary *dict in dictArr) {
Status *status = [Status cm_modelWithDict:dict];
[statusArr addObject:status];
}
NSLog(@"%@",statusArr);
Thread 1: breakpoint 4.1
@"2 elements"
[0] = (Status *) 0x604000033060
_title = (__NSCFString *) @"标题字符串"
_person = (PersonModel *) 0x6040000336a0
_iconStr = (__NSCFString *) @"头像字符串"
_showStr = (__NSCFString *) @"心情字符串"
_cellMdlArr = (__NSArrayM *) @"3 elements"
[0] = (CellModel *) 0x6040000335a0
_stateStr = (__NSCFString *) @"状态字符串"
_partnerStr = (__NSCFString *) @"合作方字符串"
[1] = (CellModel *) 0x6040000335c0
[2] = (CellModel *) 0x604000033640
[1] = (Status *) 0x604000033580
elDemo
Thread 1
0 -[ViewController viewDidLoad]