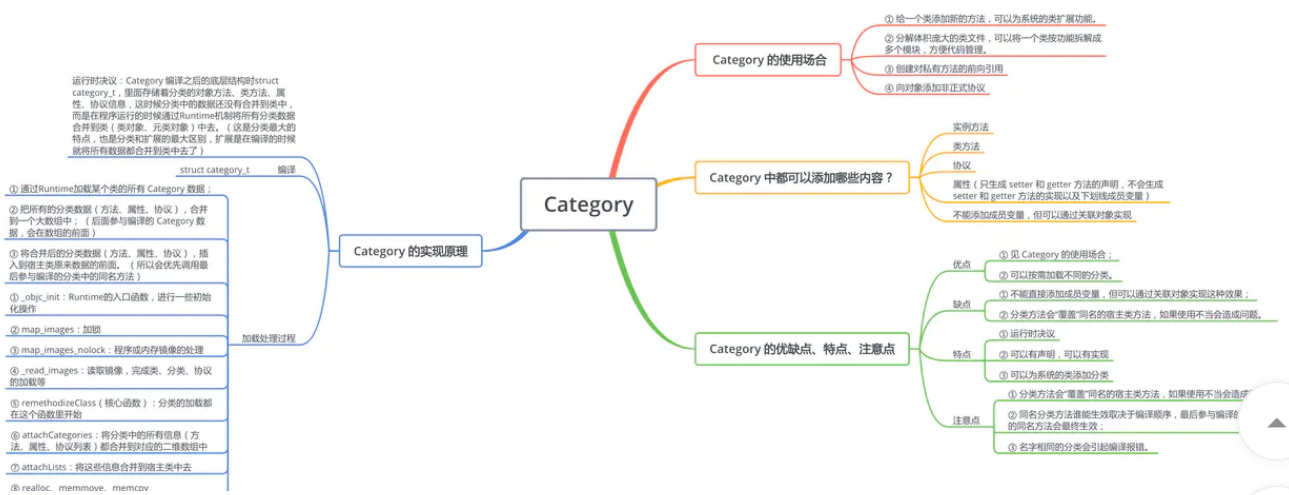
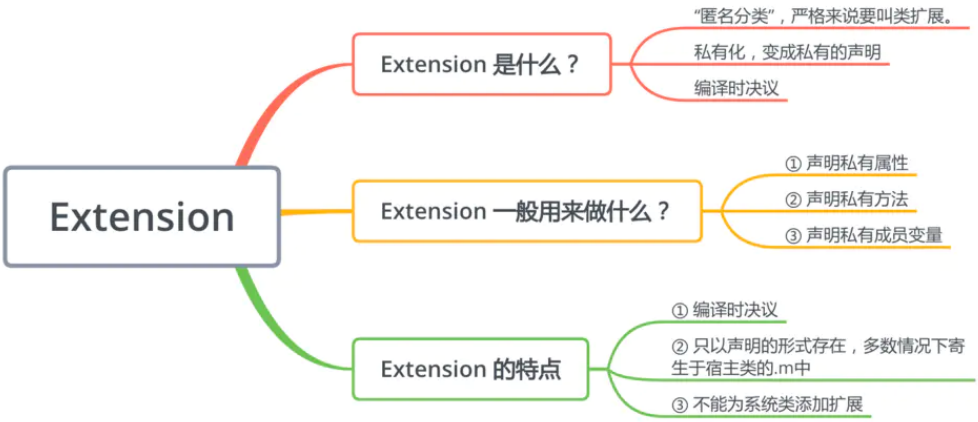
师大小海腾 Lv2
2020年02月19日 阅读 371

关注

OC 底层探索 - Category 和 Extension

大纲





1. Category 分类

1.1 Category 的使用场合

- ① 给一个类添加新的方法，可以为系统的类扩展功能。
- ② 分解体积庞大的类文件，可以将一个类按功能拆解成多个模块，方便代码管理。
- ③ 创建对私有方法的前向引用：声明私有方法，把 Framework 的私有方法公开等。直接调用其他类的私有方法时编译器会报错的，这时候可以创建一个该类的分类，在分类中声明这些私有方法（不必提供方法实现），接着导入这个分类的头文件就可以正常调用这些私有方法。
- ④ 向对象添加非正式协议：创建一个 NSObject 或其子类的分类称为 “创建一个非正式协议”。（正式协议是通过 protocol 指定的一系列方法的声明，然后由遵守该协议的类自己去实现这些方法。而非正式协议是通过给 NSObject 或其子类添加一个分类来实现。非正式协议已经渐渐被正式协议取代，正式协议最大的优点就是可以使用 泛型 约束，而非正式协议不可以。）

1.2 Category 中都可以添加哪些内容?

- 实例方法、类方法、协议、属性（只生成 setter 和 getter 方法的声明，不会生成 setter 和 getter 方法的实现以及下划线成员变量）；
- 默认情况下，因为分类底层结构的限制，不能添加成员变量到分类中，但可以通过关联对象来间接实现这种效果。

1.3 Category 的优缺点、特点、注意点

Category	描述
优点	① 见 Category 的使用场合； ② 可以按需加载不同的分类。
缺点	① 不能直接添加成员变量，但可以通过关联对象实现这种效果； ② 分类方法会“覆盖”同名的宿主类方法，如果使用不当会造成问题。
特点	① 运行时决议 ② 可以有声明，可以有实现 ③ 可以为系统的类添加分类 运行时决议：Category 编译之后的底层结构时 `struct category_t`，里面存储着分类的对象方法、类方法、属性、协议信息，这时候分类中的数据还没有合并到类中，而是在程序运行的时候通过 `Runtime` 机制将所有分类数据合并到类（类对象、元类对象）中去。（这是分类最大的特点，也是分类和扩展的最大区别，扩展是在编译的时候就将所有数据都合并到类中去了）
注意点	① 分类方法会“覆盖”同名的宿主类方法，如果使用不当会造成问题； ② 同名分类方法谁能生效取决于编译顺序，最后参与编译的分类中的同名方法会最终生效； ③ 名字相同的分类会引起编译报错。

1.4 Category 的实现原理

- ① 分类的实现原理取决于运行时决议；
- ② 同名分类方法谁能生效取决于编译顺序，最后参与编译的分类中的同名方法会最终生效；
- ③ 分类方法会“覆盖”同名的宿主类（原类）方法，这里说的“覆盖”并不是指原来的方法没了。消息传递过程中优先查找宿主类中靠前的元素，找到同名方法就进行调用，但实际上宿主类中原有同名方法的实现仍然是存在的。我们可以通过一些手段来调用到宿主类原有同名方法的实现，如可以通过 `Runtime` 的 `class_copyMethodList` 方法打印类的方法列表，找到宿主类方法的 `imp`，进行调用（可以交换方法实现）。

1.4.1 编译

源码分析

通过 Clang 将以下分类代码转换为 C++ 代码，来分析分类的底层实现。

```
// Clang
xcrun -sdk iphoneos clang -arch arm64 -rewrite-objc Person+Test.m
```

```objc
#import "Person.h"
@interface Person (Test)<NSCopying>
@property (nonatomic, copy) NSString *name;
@property (nonatomic, assign) int age;
- (void)eat;
- (void)sleep;
+ (void)run;
+ (void)walk;
@end

#import "Person+Test.h"
@implementation Person (Test)
- (void)eat {
    NSLog(@"eat");
}
- (void)sleep {
    NSLog(@"sleep");
}
+ (void)run {
    NSLog(@"run");
}
+ (void)walk {
    NSLog(@"walk");
}
@end
```

```objc
// Person+Test.cpp
struct _category_t {
        const char *name;
        struct _class_t *cls;
        const struct _method_list_t *instance_methods;
        const struct _method_list_t *class_methods;
        const struct _protocol_list_t *protocols;
        const struct _prop_list_t *properties;
};

// 实例方法列表
static struct /*_method_list_t*/ {
        unsigned int entsize;  // sizeof(struct _objc_method)
        unsigned int method_count;
        struct _objc_method method_list[2];
} _OBJC_$_CATEGORY_INSTANCE_METHODS_Person_$_Test __attribute__ ((used, section ("__DATA,__
        sizeof(_objc_method),
        2,
        {{(struct objc_selector *)"eat", "v16@0:8", (void *)_I_Person_Test_eat},
```

```
// 类方法列表
static struct /*_method_list_t*/ {
        unsigned int entsize;  // sizeof(struct _objc_method)
        unsigned int method_count;
        struct _objc_method method_list[2];
} _OBJC_$_CATEGORY_CLASS_METHODS_Person_$_Test __attribute__ ((used, section ("__DATA,__obj
        sizeof(_objc_method),
        2,
        {{(struct objc_selector *)"run", "v16@0:8", (void *)_C_Person_Test_run},
        {(struct objc_selector *)"walk", "v16@0:8", (void *)_C_Person_Test_walk}}
};

// 协议列表
static struct /*_protocol_list_t*/ {
        long protocol_count;  // Note, this is 32/64 bit
        struct _protocol_t *super_protocols[1];
} _OBJC_CATEGORY_PROTOCOLS_$_Person_$_Test __attribute__ ((used, section ("__DATA,__objc_co
        1,
        &_OBJC_PROTOCOL_NSCopying
};

// 属性列表
static struct /*_prop_list_t*/ {
        unsigned int entsize;  // sizeof(struct _prop_t)
        unsigned int count_of_properties;
        struct _prop_t prop_list[2];
} _OBJC_$_PROP_LIST_Person_$_Test __attribute__ ((used, section ("__DATA,__objc_const"))) =
        sizeof(_prop_t),
        2,
        {{"name","T@\"NSString\",C,N"},
        {"age","Ti,N"}}
};

// Person+Test 分类编译的底层结构
static struct _category_t _OBJC_$_CATEGORY_Person_$_Test __attribute__ ((used, section ("__
{
        "Person",
        0, // &OBJC_CLASS_$_Person,
        (const struct _method_list_t *)&_OBJC_$_CATEGORY_INSTANCE_METHODS_Person_$_Test,
        (const struct _method_list_t *)&_OBJC_$_CATEGORY_CLASS_METHODS_Person_$_Test,
        (const struct _protocol_list_t *)&_OBJC_CATEGORY_PROTOCOLS_$_Person_$_Test,
        (const struct _prop_list_t *)&_OBJC_$_PROP_LIST_Person_$_Test,
};
```

从以上可以看到，Category 编译之后的底层结构时 `struct _category_t`

下面我们进入 `Runtime` 的最新源代码 `objc4-756.2` 进行分析。在源代码中与 Category 相关的代码基本都放在 `objc-runtime-new.h` 和 `objc-runtime-new.mm` 两个文件中。我们先来看一下 Category 在源代码中的定义 `struct category_t` 。

从以上 Category 的底层结构来看，分类中可以添加实例方法、类方法、协议、属性，但是不能添加成员变量，因为没有存储成员变量对应的指针变量。

1.4.2 加载处理过程

在编译时，Category 中的数据还没有合并到类中，而是在程序运行的时候通过 `Runtime` 机制将所有分类数据合并到类（类对象、元类对象）中去。下面我们来看一下 Category 的加载处理过程。

- ① 通过 `Runtime` 加载某个类的所有 Category 数据；
- ② 把所有的分类数据（方法、属性、协议），合并到一个大数组中；（后面参与编译的 Category 数据，会在数组的前面）
- ③ 将合并后的分类数据（方法、属性、协议），插入到宿主类原来数据的前面。（所以会优先调用最后参与编译的分类中的同名方法）

源码分析

加载函数调用栈：

- objc-os.mm
  ① _objc_init：`Runtime`的入口函数，进行一些初始化操作
  ② map_images：加锁
  ③ map_images_nolock：程序或内存镜像的处理
- objc-runtime-new.mm
  ④ _read_images：读取镜像，完成类、分类、协议的加载等
  ⑤ **remethodizeClass（核心函数）**：分类的加载都在这个函数里开始
  ⑥ attachCategories：将分类中的所有信息（方法、属性、协议列表）都合并到对应的二维数组中
  ⑦ attachLists：将这些信息合并到宿主类中去
  ⑧ realloc、memmove、memcpy

下面我们通过⑤⑥⑦三个函数来分析分类中实例方法的添加逻辑：

remethodizeClass

attachLists

2. Extension 扩展

2.1 Extension 是什么?

- ① Extension 有一种说法叫“匿名分类”，因为它很像分类，但没有分类名。严格来说要叫类扩展。
- ② Extension 的作用是将原来放在 .h 中的数据放到 .m 中去，私有化，变成私有的声明。
- ③ Extension 是在编译的时候就将所有数据都合并到类中去了（编译时决议），而 Category 是在程序运行的时候通过 `Runtime` 机制将所有数据合并到类中去（运行时决议）。

2.2 Extension 一般用来做什么?

- ① 声明私有属性
- ② 声明私有方法
- ③ 声明私有成员变量

2.3 Extension 的特点以及 Extension 与 Category 的区别

- ① 编译时决议（在编译的时候就将扩展的所有数据都合并到类中去了）
- ② 只以声明的形式存在，多数情况下寄生于宿主类的.m中
- ③ 不能为系统类添加扩展

Category	Extension
运行时决议	编译时决议
可以有声明，可以有实现	只以声明的形式存在，多数情况下寄生于宿主类的.m中
可以为系统的类添加分类	不能为系统类添加扩展

3. 相关面试题

Q：Category 能否添加成员变量？如果可以，如何给 Category 添加成员变量?

因为分类底层结构的限制，不能直接给 Category 添加成员变量，但是可以通过关联对象间接实现 Category 有成员变量的效果。
传送门：OC - Association 关联对象

Q：为什么分类中属性不会自动生成 setter、getter 方法的实现，不会生成成员变量，也不能

因为类的内存布局在编译的时候会确定，但是分类是在运行时才加载，在运行时 `Runtime` 会将分类的数据，合并到宿主类中。

Q：为什么将以前的方法列表挪动到新的位置用 memmove 呢?

为了保证挪动数据的完整性。而将分类的方法列表合并进来，不用考虑被覆盖的问题，所以用 memcpy 就好。

Q：为什么优先调用最后编译的分类的方法?

`attachCategories()` 方法中，从所有未完成整合的分类取出分类的过程是倒序遍历，最先访问最后编译的分类。然后获取该分类中的方法等列表，添加到二维数组中，所以最后编译的分类中的数据最先加到分类二维数组中，最后插入到宿主类的方法列表前面。而消息传递过程中优先查找宿主类中靠前的元素，找到同名方法就进行调用，所以优先调用最后编译的分类的方法。

Q：objc_class 结构体中的 baseMethodList 和 methods 方法列表的区别?

回答此道问题需要先了解 `Runtime` 的数据结构 `objc_class` 。
传送门：深入浅出 Runtime（二）：数据结构。

- `baseMethodList` 基础的方法列表，是 `ro` 只读的，不可修改，可以看成是合并分类方法列表前的 `methods` 的拷贝；
- 而 `methods` 是 `rw` 可读写的，将来运行时要合并分类方法列表。

Q：Category 中有 +load 方法吗？+load 方法是什么时候调用的？+load 方法能继承吗?

1. 分类中有 `+load` 方法；
2. `+load` 方法在 `Runtime` 加载类、分类的时候调用；
3. `+load` 方法可以继承，但是一般情况下不会手动去调用 `+load` 方法，都是让系统自动调用。

传送门：OC - load 和 initialize

> **关注下面的标签，发现更多相似文章**

Objective-C

获得点赞 117 · 获得阅读 12,666

输入评论...

小小怪下士oy
mrak || mark
1月前 回复

执笔续春秋 Lv2 iOS @ 一只流浪者
已阅
2月前 回复

相关推荐

我是熊大 · 5天前 · Objective-C
🐻objc1.0 与 objc2.0 解析
4

EricStone · 9天前 · Objective-C
iOS-组件化实践（OC篇）
12

尧少羽 · 10天前 · Objective-C
Objective-C 之 Runtime 对象
10

HookLee · 13天前 · Objective-C
MGJRouter CTMediator BeeHive 组件化源码分析
5 3

KFAaron · 15天前 · Objective-C
浅谈Objective-C中的atomic和nonatomic

老司机技术周报 · 1月前 · Objective-C / WWDC

【WWDC20】10163 - iOS 14 苹果对 Objective-C Runtime 的优化

15

尧少羽 · 10天前 · Objective-C

Objective-C 之 Runtime 类

2

萌呆宝 · 22天前 · Objective-C

面试遇到RunLoop的第一天-原理

5

KFAaron · 13天前 · Objective-C

浅谈Objective-C中的weak那些事

2

iOShuyang · 11月前 · Objective-C

⚠2019年iOS面试反思总结--不断更新当中ing⚠

212 59