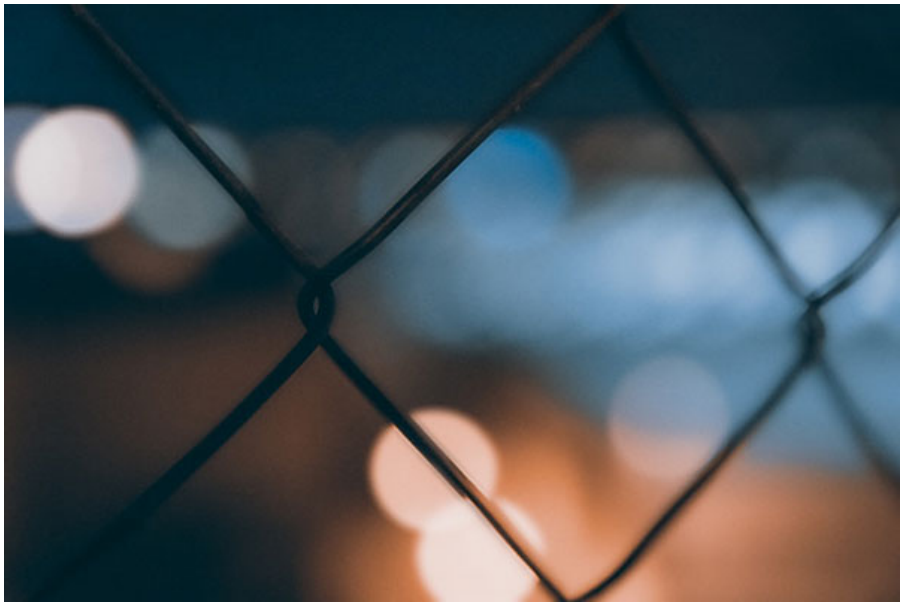
接口编程那些事（或者面向协议编程）

2015-12-14 06:18 编辑：lansekuangtu 分类：iOS开发 来源：庞海礁 投稿 0 237

接口编程

招聘信息：Android开发工程师



接口是一系列可调用方法的集合。何为接口编程？接口编程是指当写一个函数或一个方法时，我们应该更加关注具体的接口，而不是实现类。具体理解可以参考这篇文章

在OC中，接口又可以理解为Protocol，面向接口编程又可以理解为面向Protocol编程，或者面向协议编程。在Swift中，苹果大幅强化了 Protocol 在这门语言中的地位，整个 Swift 标准库也是基于 Protocol 来设计的，有兴趣的童鞋可以看看这篇文章。面向接口编程正逐步成为程序开发的主流思想

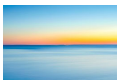
在实际开发中，大多数朋友都比较熟悉对象编程，比如，使用ASIHttpRequest执行网络请求

```
ASIHTTPRequest *request = [ASIHTTPRequest requestWithURL:url];
[request setDidFinishSelector:@selector(requestDone:)];
[request setDidFailSelector:@selector(requestWrong:)];
[request startAsynchronous];
```

request是请求对象，当发起请求时，调用者需要知道给对象赋哪些属性或者调用对象哪些方法。然而，使用AFNetworking请求方式却不尽相同

```
AFHTTPRequestOperationManager *manager = [AFHTTPRequestOperationManager manager]
[manager GET:@"www.olinone.com" parameters:nil success:^(AFHTTPRequestOperation
```

热门资讯


如何让iOS 保持界面流畅？这些技巧你知道吗
点击量 13120


如何设计一个 iOS 控件?(iOS 控件完全解析)
点击量 12307


培训机构毕业的程序员被歧视的背后逻辑
点击量 7169

苹果Swift编程语言开源 有望进一步普及
点击量 6210


对 Xcode 菜单选项的详细探索（干货）
点击量 6152


【资源集合】94个iOS开发资源推荐，帮你加
点击量 6072


Swift 3 API 设计准则
点击量 5504


一款Loading动画的实现思路（一）：拆分复
点击量 5411


iOS 事件处理机制与图像渲染过程
点击量 5358

Swift中编写单例的正确方式
点击量 4913

综合评论

mark

wooshare 评论了 【资源集合】94个iOS开发资源推荐，帮你加速应用...

mark

youkini 评论了 【资源集合】94个iOS开发资源推荐，帮你加速应用...

6666666

HAlice 评论了 深入理解RunLoop

mark

2yscw 评论了 快速搭建一个成熟，强

```
    NSLog(@"好网站，赞一个！");
} failure:^(AFHTTPRequestOperation *operation, NSError *error) {
    //to do
}];
```

同是请求对象，使用AFNetworking发起请求时，调用者可以不需要关心它有哪些属性，只有接口无法满足需求时才需要了解相关属性的定义。两种设计思路完全不同，当然，此处并不是想表明孰优孰劣，只是想通过两种截然不同的请求方式引出本文的思想——面向接口编程（或者面向协议编程）

接口比属性直观

在对象编程中，定义一个对象时，往往需要为其定义各种属性。比如，ReactiveCocoa中RACSubscriber对象定义如下

```
@interface RACSubscriber ()

@property (nonatomic, copy) void (^next)(id value);
@property (nonatomic, copy) void (^error)(NSError *error);
@property (nonatomic, copy) void (^completed)(void);

@end
```

参考AFNetworking的思想，以接口的形式提供访问入口

```
@interface RACSubscriber

+ (instancetype)subscriberWithNext:(void (^)(id x))next
                             error:(void (^)(NSError *error))error
                         completed:(void (^)(void))completed;

@end
```

通过接口的定义，调用者可以忽略对象的属性，聚焦于其提供的接口和功能上。程序猿在首次接触陌生的某个对象时，接口往往比属性更加直观明了，抽象接口往往比定义属性更能描述想做的事情

接口依赖

设计一个APIService对象

```
@interface ApiService : NSObject

@property (nonatomic, strong) NSURL        *url;
@property (nonatomic, strong) NSDictionary *param;

- (void)execNetRequest;

@end
```

正常发起Service请求时，调用者需要直接依赖该对象，起不到解耦的目的。当业务变动需要重构该对象时，所有引用该对象的地方都需要改动。如何做到既能满足业务又能兼容变化？抽象接口也许是一个不错的选择，以接口依赖的方式取代对象依赖，改造代码如下

```
@protocol ApiServiceProtocol
- (void)requestNetWithUrl:(NSURL *)url Param:(NSDictionary *)param;

@end

@interface NSObject (ApiServiceProtocol)
@end

@implementation NSObject (ApiServiceProtocol)

- (void)requestNetWithUrl:(NSURL *)url Param:(NSDictionary *)param {
    ApiService *apiSrevice = [ApiService new];
    apiSrevice.url = url;
    apiSrevice.param = param;
    [apiSrevice execNetRequest];
}

@end
```

通过接口的定义，调用者可以不再关心ApiService对象，也无需了解其有哪些属性。即使需要重构替换新的对象，调用逻辑也不受任何影响。调用接口往往比访问对象属性更加稳定可靠

抽象对象

定义ApiServiceProtocol可以隐藏ApiService对象，但是受限于ApiService对象的存在，业务需求发生变化时，仍然需要修改ApiService逻辑代码。如何实现在不修改已有ApiService业务代码的条件下满足新的业务需求?

参考Swift抽象协议的设计理念，可以使用Protocol抽象对象，毕竟调用者也不关心具体实现类。Protocol可以定义方法，可是属性的问题怎么解决？此时，装饰器模式也许正好可以解决该问题，让我们试着继续改造ApiService

```
@protocol ApiService
// private functions

@end

@interface ApiServicePassthrough : NSObject

@property (nonatomic, strong) NSURL        *url;
@property (nonatomic, strong) NSDictionary *param;

- (instancetype)initWithApiService:(id)apiService;
- (void)execNetRequest;

@end
```

```
@interface ApiServicePassthrough ()

@property (nonatomic, strong) id apiService;

@end

@implementation ApiServicePassthrough

- (instancetype)initWithApiService:(id)apiService {
    if (self = [super init]) {
        self.apiService = apiService;
    }
    return self;
}

- (void)execNetRequest {
    [self.apiService requestNetWithUrl:self.url Param:self.param];
}

@end
```

经过Protocol的改造，ApiService对象化身为ApiService接口，其不再依赖于任何对象，做到了真正的接口依赖取代对象依赖，具有更强的业务兼容性

定义一个Get请求对象

```
@interface GetApiService : NSObject
@end

@implementation GetApiService

- (void)requestNetWithUrl:(NSURL *)url Param:(NSDictionary *)param {
    // to do
}

@end
```

请求代码也随之改变

```
@implementation NSObject (ApiServiceProtocol)

- (void)requestNetWithUrl:(NSURL *)url Param:(NSDictionary *)param {
    id apiSrevice = [GetApiService new];
    ApiServicePassthrough *apiServicePassthrough = [[ApiServicePassthrough allo
    apiServicePassthrough.url = url;
    apiServicePassthrough.param = param;
    [apiServicePassthrough execNetRequest];
}
```

```
@end
```

对象可以继承对象，Protocol也可以继承Protocol，并且可以继承多个Protocol，Protocol具有更强的灵活性。某一天，业务需求变更需要用到新的Post请求时，可以不用修改 GetApiService一行代码，定义一个新的 PostApiService 实现Post请求即可，避免了对象里面出现过多的if-else代码，也保证了代码的整洁性

依赖注入

文章写到这里，细心的童鞋可能已经发现问题——GetApiService依然是以对象依赖的形式存在。如何解决这个问题？没错，那就是依赖注入！

依赖注入是什么？借用博客里面的一句话，其最大的特点就是：帮助我们开发出松散耦合、可维护、可测试的代码和程序。这条原则的做法是大家熟知的面向接口，或者说是面向抽象编程。 objc上这篇文章介绍的不错， 有兴趣的童鞋也可以看看，在此就不再累述

基于依赖注入objection开源库的基础上继续改造ApiService

```
@implementation NSObject (ApiServiceProtocol)

- (void)requestNetWithUrl:(NSURL *)url Param:(NSDictionary *)param {
    id apiSrevice = [[JSObjection createInjector] getObject:[GetApiService clas
    ApiServicePassthrough *apiServicePassthrough = [[ApiServicePassthrough allo
    apiServicePassthrough.url = url;
    apiServicePassthrough.param = param;
    [apiServicePassthrough execNetRequest];
}

@end
```

调用者关心请求接口，实现者关心需要实现的接口，各司其职，互不干涉

接口和实现分离的设计适用于团队协作开发，实现了系统的松散耦合，便于以后升级扩展。当然，接口编程对开发人员的要求也比较高，需要提前定义好接口，接口一变，全部乱套，这就是所谓的设计比实现难，但是设计接口的人工资都高啊！！！

后记：你可以在github找到我，也可以通过微博联系我，感谢你的来访！

上一篇：换一个姿势写界面样式
下一篇：源码推荐(12.11B)：视图动画，仿SegmentFault

相关资讯

我来说两句

你怎么看？快来评论一下吧！

发表评论

所有评论（0）