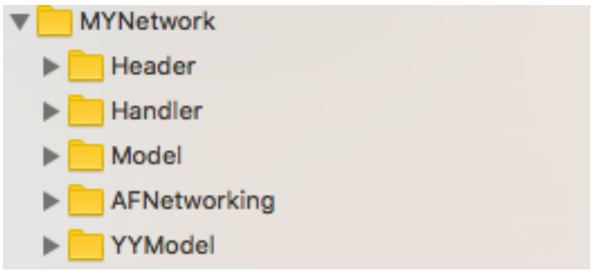
MYNetwork 网络库(基于 AFNetworking)

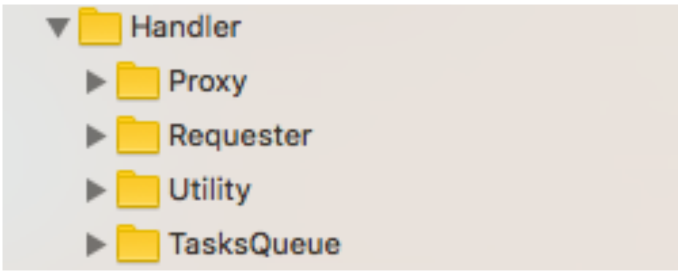
1. 简述：MYNetwork （基于 AFNetworking）是在 AFNetworking 框架基础上封装的一个网络库，功能包括：网络数据 get/post 请求、文件上传下载、图片加载缓存等，目标是做到 1 行代码完成一个网络请求事件，此次封装验证了 MYNetwork 框架是松耦合的，相比之前基于 NSURLSession 进行封装的 MYNetwork 网络库 1.0 版，仅需修改部分实现文件即可，以下将详细介绍修改细节。

2. 封装

(1) 目录结构：



```
▼ MYNetwork
   ▶ Header
   ▶ Handler
   ▶ Model
   ▶ AFNetworking
   ▶ YYModel
```



```
▼ Handler
   ▶ Proxy
   ▶ Requester
   ▶ Utility
   ▶ TasksQueue
```

(2) 相关文件修改：

1) HeaderFile

1>MYNetwork-prefix.pch

导入头文件：

```
#import "AFNetworking.h"
#import "UIKit+AFNetworking.h"
```

2>MYTypeDefinitions.h

添加枚举、定义 block：

```
typedef NS_ENUM(NSUInteger, MYHTTPResponseSerializerType){
    MYHTTPResponseSerializerTypeJson,
    MYHTTPResponseSerializerTypeData,
    MYHTTPResponseSerializerTypePlist,
    MYHTTPResponseSerializerTypeXMLParser,
    MYHTTPResponseSerializerTypeCompound
};
```

```
typedef void (^MYNetworkLinkStatusFetchBlock)(MYNetworkLinkStatus status);
```

```
typedef void (^MYNetworkImageFetchBlock)(UIImage *fetchImage, BOOL isCache);//fetchImage为nil说明缓存里没找着、图片url请求失
  败
```

2) Model

1>MYRequestObj.h

添加属性

```
@property (assign, nonatomic) MYHTTPResponseSerializerType responseSerializer;///返回数据解析类型
```

2>MYRequestObj.m

初始化

```
self.responseSerializer = MYHTTPResponseSerializerTypeJson;
```

3>MYURLSessionTask.m

添加 requestId 创建逻辑(使用 AFNetworking，requestId 不适合直接使用 session 的 taskIdentifier)

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

@interface NSURLSessionTask (RequestId)

@property (assign, nonatomic) NSInteger requestId;

@end

@implementation NSURLSessionTask (RequestId)

- (NSInteger)requestId{
    return [objc_getAssociatedObject(self, _cmd) integerValue];
}

- (void)setRequestId:(NSInteger)requestId{
    objc_setAssociatedObject(self, @selector(requestId), [NSNumber numberWithInteger:requestId],
        OBJC_ASSOCIATION_RETAIN_NONATOMIC);
}
```

```
static int myRequestId = 0;
static NSObject *obj;
```

```
- (void)setTask:(NSURLSessionTask *)task{
    if(!obj)
        obj = [[NSObject alloc] init];
    @synchronized (obj) {
        _task = task;
       task.requestId = (++myRequestId);
        _requestId = myRequestId;
        self.netWorkResponse.requestId = myRequestId;
    }
}
```

3) Handler(TaskQueue)

1>MYTasksOperationQueue.m

```
- (void)startSessionTaskFromTasksQueueWithTask:(MYURLSessionTask *)task  taskType:(MYNetworkTaskType)taskType{
//    [task.task resume];
```

Handler(Utility)

1>MYNetworkUtility.h

```
@class AFHTTPResponseSerializer;
```

添加方法

```
+ (AFHTTPResponseSerializer *)getHttpResponseSerializerFromRequestObject:(MYRequestObj *)requestObj;//从请求模型中获取请求响应的解析
  器
```

注掉方法

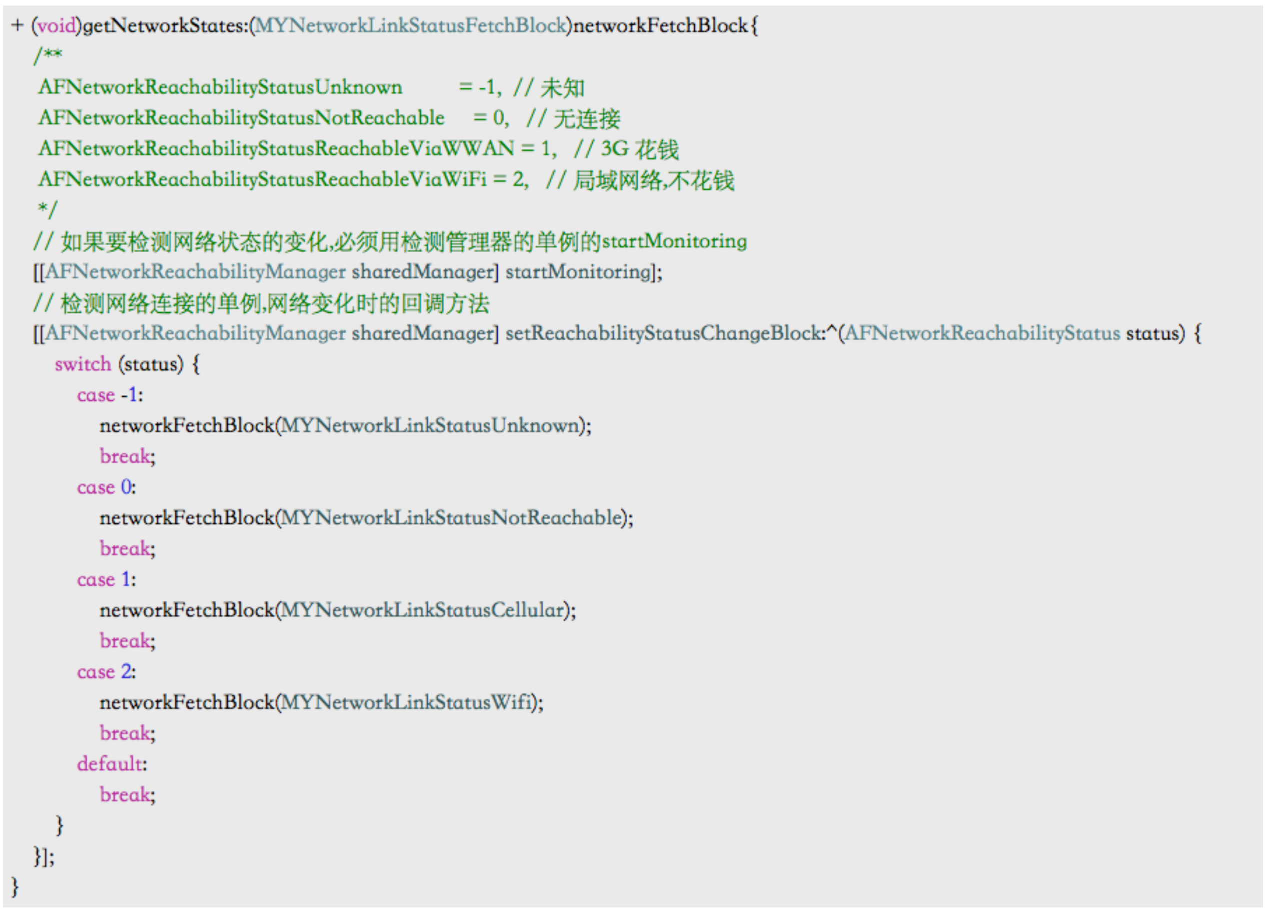
```
//+ (void)setHttpHeadersWithRequestObject:(MYRequestObj *)requestObj  request:(NSMutableURLRequest *)request;//设置http请求头部信息

//+ (NSString *)getEncodedParamsFromDictionary:(NSDictionary *)dic;//从参数字典中获取编码后的参数字符串

//+ (NSString*) mk_urlEncodedString:(NSString *)string;//url参数编码逻辑

//+ (NSString *)getContentTypeWithFilePath:(NSString *)filePath;//获取文件的MIME

//+ (NSMutableURLRequest *)getPostRequestWithRequestObj:(MYRequestObj *)requestObj;//构建post请求
//+ (NSMutableURLRequest *)getGetRequestWithRequestObj:(MYRequestObj *)requestObj;//构建get请求
//+ (NSMutableURLRequest *)getMultipartFormDataRequestWithRequestObj:(MYRequestObj *)requestObj filePath:(NSString *)filePath;//构建
   multipart/form-data请求
//+ (NSURLRequest *)getDownloadRequestWithUrl:(NSURL *)url;//构建download请求
```

修改方法

```
+ (void)getNetworkStates:(MYNetworkLinkStatusFetchBlock)networkFetchBlock;//获取网络状态
```

2> MYNetworkUtility.m

```
+ (AFHTTPResponseSerializer *)getHttpResponseSerializerFromRequestObject:(MYRequestObj *)requestObj{
    switch (requestObj.responseSerializer) {
        case MYHTTPResponseSerializerTypeJson:
            return [AFJSONResponseSerializer serializer];
            break;
        case MYHTTPResponseSerializerTypeData:
            return [AFHTTPResponseSerializer serializer];
            break;
        case MYHTTPResponseSerializerTypePlist:
            return [AFPropertyListResponseSerializer serializer];
            break;
        case MYHTTPResponseSerializerTypeXMLParser:
            return [AFXMLParserResponseSerializer serializer];
            break;
        case MYHTTPResponseSerializerTypeCompound:
            return [AFCompoundResponseSerializer serializer];
            break;
        default:
            return nil;
            break;
    }
}
```



```
+ (void)getNetworkStates:(MYNetworkLinkStatusFetchBlock)networkFetchBlock{
  /**
   AFNetworkReachabilityStatusUnknown          = -1,  // 未知
   AFNetworkReachabilityStatusNotReachable     = 0,   // 无连接
   AFNetworkReachabilityStatusReachableViaWWAN = 1,   // 3G 花钱
   AFNetworkReachabilityStatusReachableViaWiFi = 2,   // 局域网络,不花钱
   */
  // 如果要检测网络状态的变化,必须用检测管理器的单例的startMonitoring
  [[AFNetworkReachabilityManager sharedManager] startMonitoring];
  // 检测网络连接的单例,网络变化时的回调方法
  [[AFNetworkReachabilityManager sharedManager] setReachabilityStatusChangeBlock:^(AFNetworkReachabilityStatus status) {
      switch (status) {
          case -1:
              networkFetchBlock(MYNetworkLinkStatusUnknown);
              break;
          case 0:
              networkFetchBlock(MYNetworkLinkStatusNotReachable);
              break;
          case 1:
              networkFetchBlock(MYNetworkLinkStatusCellular);
              break;
          case 2:
              networkFetchBlock(MYNetworkLinkStatusWifi);
              break;
          default:
              break;
      }
  }];
}
```

3>Handler(Requester)

MYDataRequester.h