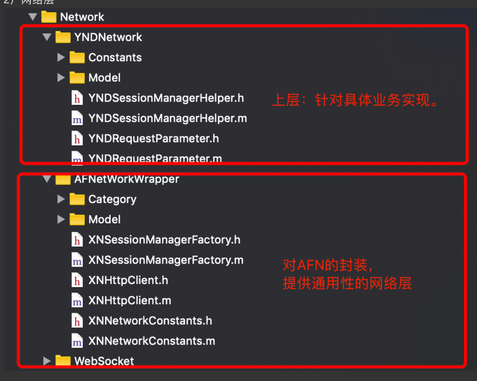
1）组件化方案：

如图，当开发一个新增业务模块时，分三部分：
1）业务的具体实现；
2）该业务对外的协议接口；
3）路由类对该业务模块的协议接口的分类实现。

该组件化方案有如下特点：
1）简单，逻辑清晰。
2）组件之间相互调用基本0硬编码，便于排查调用问题。
3）向上可升级为阿里开源的beehive，改造工作量小。
4）vc锚点功能可抽取为单独组件。

2）网络层



如图：
网络层具体分为三层：
第一层：AFNetworking库；
第二层：在AFNetworking库上面的一个通用层包装，和业务无关；
第三层：在通用层上面进一步实现。主要包括：
具体请求的数据参数封装，例如各种header参数（有可能是http header中传给后端，也有可能是在body中传给后端， 数据是否加密处理，时间戳等等， 以及response的解析。

优势：
这样设计可以应对不同业务团队（或跨子公司的业务接入）的不同接口协议格式。分层清晰，便于扩展。

网络请求调用demo：

```
- (void)sendAddOrderRequest {

    //api
    NSString *api = kAPI_otc_addorder;
    //参数
    NSString *ad_type = self.tradeInfoDic[@"ad_type"];
    NSString *volume = self.count;
    NSString *ads_id = [self.tradeInfoDic[@"id"] stringValue];
    NSString *sec_pwd = self.password;
    NSString *account_types = self.curPayMethod[@"type"];

    NSMutableDictionary *paramDic = [NSMutableDictionary dictionary];
    [paramDic ss_setObject:ad_type forKey:@"ad_type"];
    [paramDic ss_setObject:volume forKey:@"volume"];
    [paramDic ss_setObject:ads_id forKey:@"ads_id"];
    [paramDic ss_setObject:sec_pwd forKey:@"sec_pwd"];
    [paramDic ss_setObject:account_types forKey:@"account_types"];

    Class entityCls = nil;

    //request
    XNLRequest *request = XNLPostRequest(api, paramDic);
    [self sendRequest:request
          showLoading:YES
          entityClass:entityCls
         successBlock:^(XNLBaseResponse *response) {
             //nothing
         }];
}
```

3）MVVM设计
定义好代码规范：
1）例如view，viewController subview的创建，添加，约束设置，RAC绑定 分别写在什么方法内。
2）viewModel 网络请求，数据处理等分别怎么写。

然后通过自动代码生成脚本 生成各个类的骨架代码，再结合代码段（Code Snippet）可一定程度上提高开发效率~