iOS路由设计（三）带你一步步构建iOS路由

> 接上一篇移动端路由层设计,这一篇是实战篇，手把手的带你编写一个简单的路由组件。有朋友说很多人都收藏以后就再也没看过，其实这属于时间管理问题，在你忙碌的工作和生活的时候，有时候需要你稍微停顿一下，思考一下，例如，你可以把本篇文章收藏以后再在iPhone的提醒事项里加入到一个阅读清单里，不用设置提醒，只需要在你闲的时候抽出一两个小时，看一下。想象一下你自己动手从发现问题到解决问题再到做出一个解决问题的组件的过程给你带来的成就感和获取的进阶经验，再稍微改变一下你对每天需要处理的繁杂事物的管理方式，也许你的生活和工作就会豁然开朗。

这个路由究竟是什么鬼？能解决什么问题？

举一些场景来看看

场景1：一个App项目中团队人员比较多，不同的人负责不同的模块开发，有的人直接使用资源文件设计的，有的人用代码直接写的，有的人负责登录，有的人负责订单，突然有一天搞订单的开发A找搞登录的开发B说要调一下登录，登录成功以后你要再回调下我写的模块的方法告诉我成功登录，我要刷新一下订单页面，B傻傻的就答应了，找B的人C、D、F....越来越多，B负责的代码越写越多，同时A也不怎么开心，因为A发现调B写的登录要通过类实例化函数获取模块，调C写的支付使用工厂方法，调D写的计算器组件又是另外一种写法，结果A自己的代码也越来越丑。

场景2：一个App里面有很多内嵌的H5页面，缠品A对猿B说，我们的活动页面要调用一下我们的订单页面，用户如果下了一个订单成功以后H5要能够拿到反馈有欢迎语，猿B和H5的开发猿C经过很久很久的讨论，确定了H5如果调用App的订单页面，参数怎么传，订单提交以后怎么再调H5的接口，参数怎么定义，各自把代码写到各自的项目里，没过多久缠品A说另外的H5要调用原生的界面，怎么怎么个流程，推送点击要调用原生的某个页面，点完要反馈给后台统计，兄弟App要跳转到我们的App某个页面跳转完成某个动作以后要再跳转回去.....猿B每每接到这样的需求就紧紧握住自己中箭的膝

盖，收拾了一下写的那么多代码，深藏功与名.....🤫.

出了什么问题?

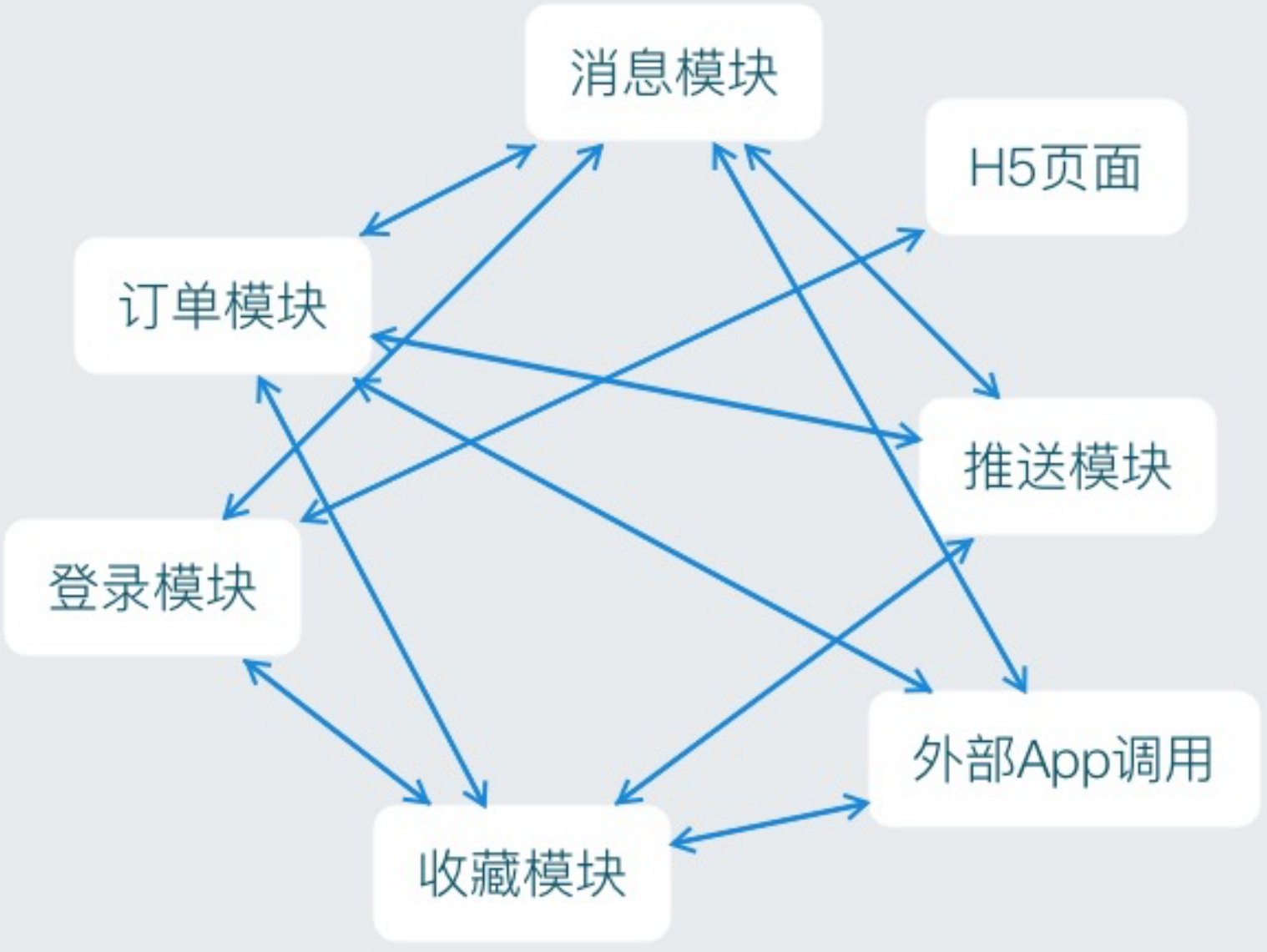
我想上面的两个场景出现的问题大家或多或少都会遇见，总结一下就是:

1. 因为不同人负责不同模块，调用他人必须了解他人编写的模块如何调用，对象是啥，初始化方式是啥，这违背了面向对象的封装原则
2. 引入不同的模块头文件，多了以后，所依赖的外部发生一丁点变化你就要跟着变，逻辑变得越来越耦合，不利于维护
3. 调用不同模块要反复与他人沟通传参、回调流程、接口定义等等，沟通效率低下
4. 产品提出各种需求，但是我写的代码都是差不多的，来一个页面我需要写一些相同逻辑的代码，而且产品还抱怨每次加相同的东西就要改代码发版，这显然不能满足复用的要求。

> 总结:
> 依赖多、耦合高、复用低。
> 可我们都知道有这么句话啊：高内聚、低耦合，职责单一逻辑清晰。

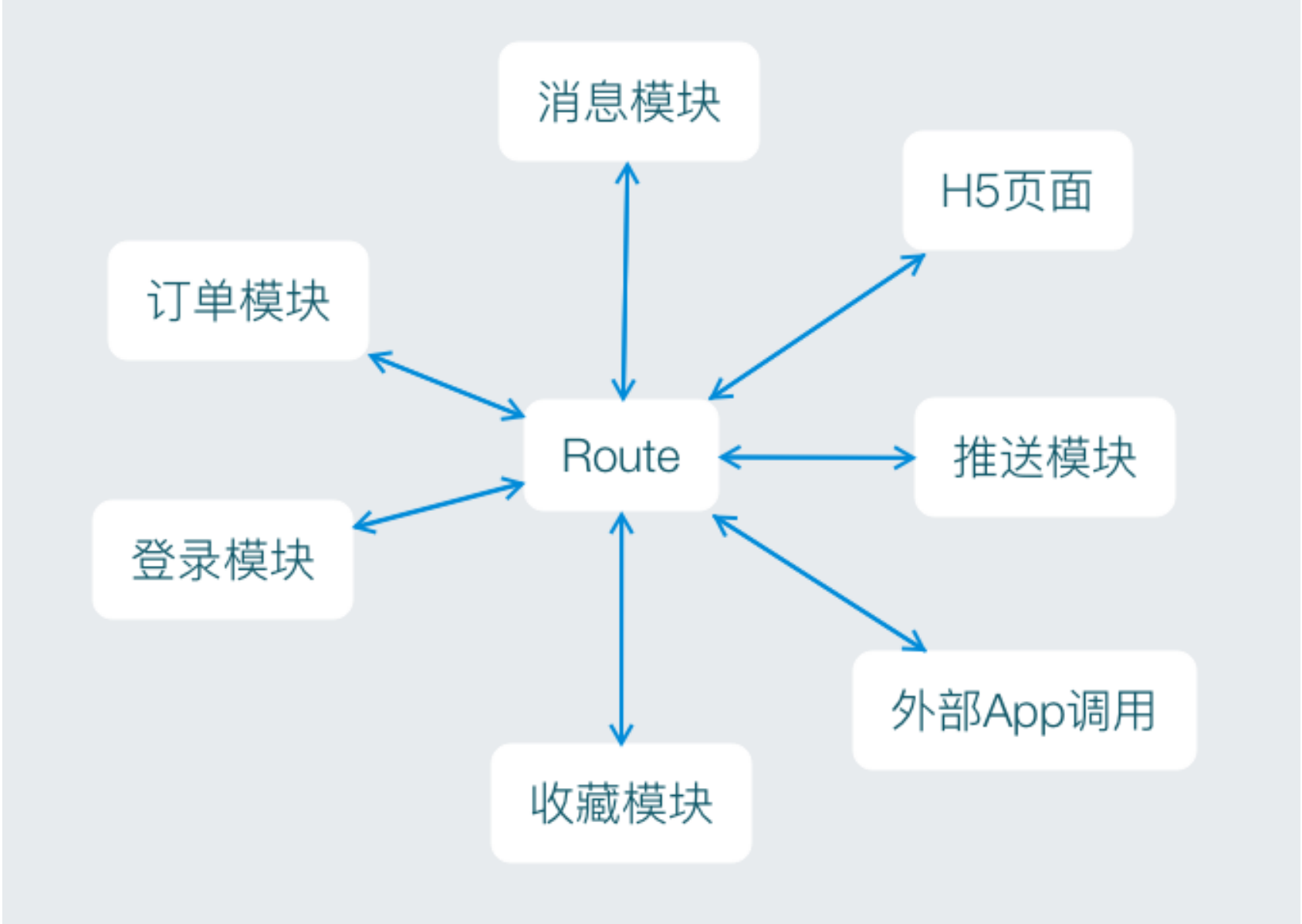
路由就是解决上面的问题

我们已经发现依赖比较大是因为要导入其他模块的头文件，了解其他模块的逻辑和定义，如果多了，你的代码中引入的头文件或者导入的包名越来越多，改一下牵一发而动全身啊。大概是这个样子:



相互引用

依赖的问题很严重，要想破除这样的依赖，我们能想到的办法就是找个调度中心去做这件事，其实各个业务模块并不关心其他模块具体的业务逻辑是什么，也不需要知道这个模块如何获取，我只关心怎么调用和反馈的结果，而这个有了调度中心这个东西，每个模块不需要依赖其他模块，只需要调度中心关心每个模块的调度。



引入中介者

有了Route这个调度中心，每个模块就不用写那么多重复的耦合代码了，也不需要在导入那么多头文件了和引入那么多包名了，这些蓝色的箭头代表着调用方式，如果调用方式再统一一下，沟通效率就提升上去了，因为我们可以用一套约定好的数据协议来代替重复沟通，有时候我们需要靠约定和协议来提高我们的工作效率。

> Tips:
> 发现问题这个环节很重要，你在工作中经常要反复做的，浪费时间的都是需要你去优化和花大力气去解决的，作为一个专业人士，不断改进你的代码，优化你的工作流程，带动团队向好的协作方式去转型，这是专业人士的习惯，更应该成为你的习惯。同时针对代码存在的问题，也许你经常会隐隐约约感到有问题，就是不知道问题在什么地方，那么需要问问自己有没有以下情况：哪些代码是经常写且重复度很高的，是不是可以抽象出来？哪些代码需要反复的变动，是不是可以做成配置或者是定义一套数据格式来满足动态兼容？有没有一些现成的设计模式可以解决这些问题？比方说，调度中心则使用的是中介者模式。我见过

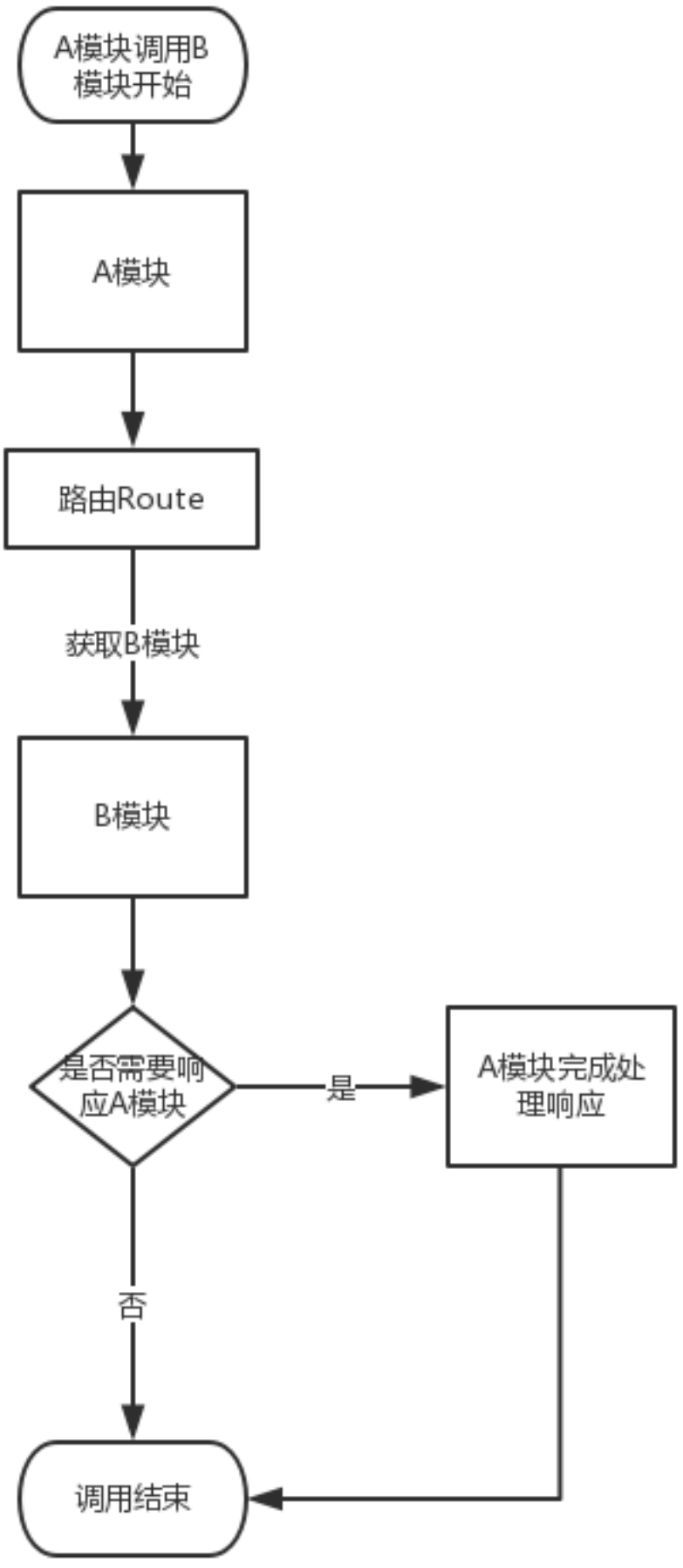
为啥要说iOS路由呢？

路由层其实在逻辑功能上的设计都是一样的，**很多人把App中的视图切换当做是路由组件的功能职责**，这点我持否定态度，从单一职责角度和MVC框架分析来看，视图切换属于View中的交互逻辑并不属于消息传递或者是事件分发的范畴，但路由请求、视图转场的实现部分与Android平台和iOS平台上的导航机制有着非常紧密的关系，Android操作系统有着天然的架构优势，Intent机制可以协助应用间的交互与通讯，是对调用组件和数据传递的描述，本身这种机制就解除了代码逻辑和界面之间的依赖关系，只有数据依赖。而iOS的界面导航和转场机制则大部分依赖UI组件各自的实现，所以如何解决这个问题，iOS端路由的实现则比较有代表性。
其实说白一点，路由层解决的核心问题就是原来界面或者组件之间相互调用都必须相互依赖，需要导入目标的头文件、需要清楚目标对象的逻辑，而现在全部都通过路由中转，只依赖路由或者某种通讯协议，或者依靠一些消息传递机制连路由都不依赖。其次，路由的核心逻辑就是目标匹配，对于外部调用的情况来说，URL如何匹配Handler是最为重要的，匹配就必然用到正则表达式。了解这些关键点以后就有了设计的目的性，let‘s do it~

总结一下这个路由都要有什么？(需求分析)

我们先根据上面的模糊的总结梳理一下：

1. 路由需要能够实现被其他模块调度，从而调度另外一个模块
2. 接入路由的模块不需要知道目标模块的实现
3. 调度发起方需要有目标的响应回调，类似于http请求，有一个request就要有一个response，才能实现双向的调用
4. 调用方式需要统一，统一而松散的调用协议和数据协议可以减少大量接入成本和沟通成本
   那一个完整的调度流程应该是这样的：



Route流程

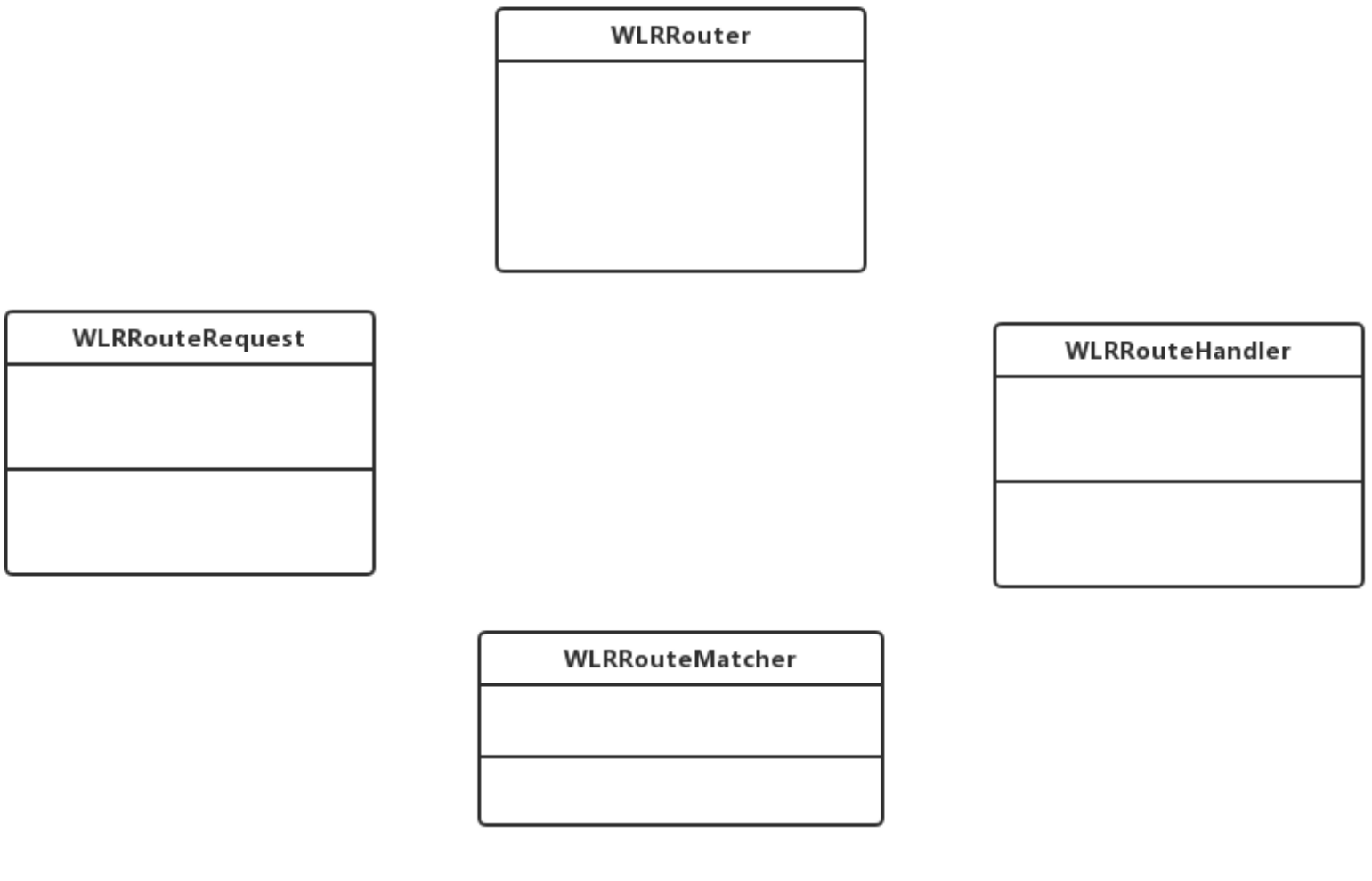
看到这个流程以后，可以确定以下几件事：

1. A模块调用路由，为表达自己需要调用的是B模块，考虑到H5、推送以及其他App的外部调用，可以使用URL这种方式来定义目标，也就是说用URL来表示目标B
2. 对一个URL的请求来说，路由需要有统一的回调处理，当然，如果不需

要回调也是可以的，回调是需要目标去触发的

3. 路由要有处理URL的功能，并调用其他模块的能力

根据以上粗略的定义一下路由的框架：



框架类图

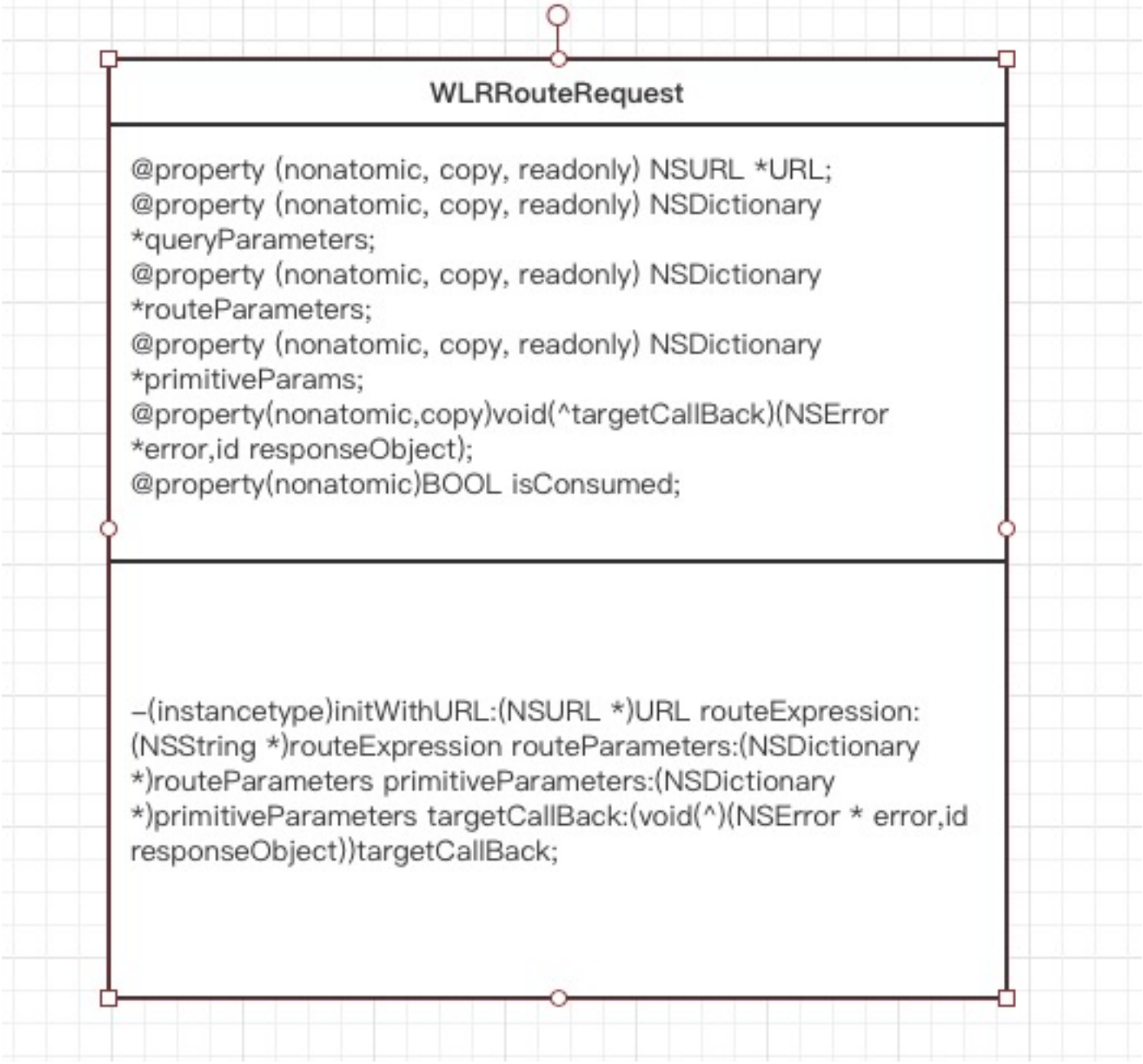
这里面以供有4部分：
WLRRouter就是一个实体对象，用来提供给其他模块调用。
WLRRouteRequest是一个以URL为基础的实体对象，为什么不直接用URL字符串？因为考虑到如果路由在内部调用其他模块的时候需要传入一些原生对象，而URL上只能携带类型单一的字符串键值对表示参数，所以需要使用这么一个对象进行包装。
WLRRouteHandler是一个处理某一个WLRRouteRequest请求的对象，当路由接收一个WLRRouteRequest请求，转发给一个WLRRouteHandler处理，处理完毕以后如果有回调，则回调给调用者。URL的请求与Handler的对应关系肯定需要匹配的逻辑，为了使得路由内部逻辑更加清晰单独使用WLRRouteMatcher来处理匹配的逻辑。

深入具体需求，细化功能实现(详细设计)

有了粗略的需求分析接下来就是细化需求并给出详细设计的阶段了，其实编写一个模块要有系统性思维，粗略的需求里面包含了整个模块要实现的主要核心功能，核心流程是什么，要有哪几个类才能实现这样的流程，不要妄图一下子深入到细枝末节上，让细节左右宏观上的逻辑架构，大脑不适合同时考虑宏观和微观的事情，尤其是对经验不太足的开发者来说，要逐渐学会大脑在不同的时期进行宏观和微观的无缝切换，这样才能专注目标和结果，在实现过程中再投入全部精力考虑细节，才能保证具体的实现是不偏离总体目标的。

WLRRouteRequest设计

路由层的请求，无论是跨应用的外部调用(H5调用、其他App调用)还是内部调用(内部模块相互调用)，最后都要形成一个路由请求，一个以URL为基础的request对象，首先需要有携带URL，再一个要携带请求所需要的参数，参数有三种，一种是Url上的键值对参数，一种是RESTFul风格的Url上的路径参数，一种是内部调用适用的原生参数，具体是:



WLRRouteRequest

这里说一下路径参数，很多有后端开发经验的人都知道，一个url上传递参数，或者是匹配后端服务的service，Url的路径对于表达转发语义十分重要，比方说：

http://aaaa.com/login

http://aaaa.com/userCenter

那Url中的login和userCenter可以代表是哪个后端服务，那路由就需要设置正则匹配表达式去匹配http://aaaa.com/ 这部分，截取login、userCenter部分，说回我们的路由，App的路由需要通过设置Url的正则表达式来获取路径参数，同时我们必须知道这些参数的值和名称，那么我可以这样定义Url匹配的表达式

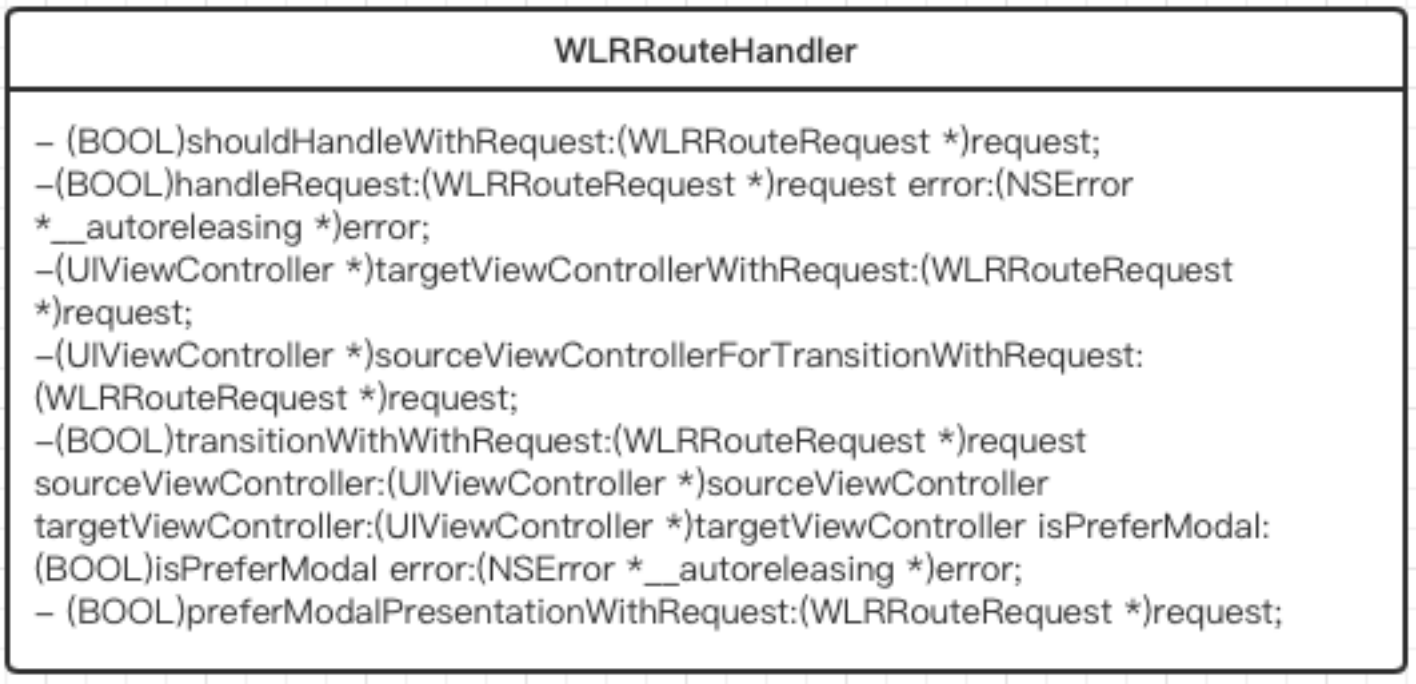
scheme://host/path/:name([a-zA-Z_-]+)

熟悉正则表达式的孩子都知道分组模式，path后name是key，([a-zA-Z_-]+)是规定name对应的value应该是什么格式的。那么routeParameters就是存放路径参数的

```
//url
@property (nonatomic, copy, readonly) NSURL *URL;
//url上? 以后的键值对参数
@property (nonatomic, copy, readonly) NSDictionary *queryParameters;
//url上匹配的路径参数
@property (nonatomic, copy, readonly) NSDictionary *routeParameters;
//原生参数，比方说要传给目标UIImage对象，NSArray对象等等
@property (nonatomic, copy, readonly) NSDictionary *primitiveParams;
//目标预留的callBack block,当完成处理以后,回到此Block，完成调用者的回调
@property(nonatomic,copy)void(^targetCallBack)(NSError *error,id responseOb
//是否消费掉，一个request只能处理一次，该字段反应request是否被处理过
@property(nonatomic)BOOL isConsumed;
```

WLRRouteHandler设计

handler对象要接收一个WLRRouteRequest对象来启动处理流程，前面经过我们的分析，这个handler应该担负起通过url和参数获取目标对象的职责，在一般的route处理中，目标往往是一个视图控制器，先实现这样一个通过url调用某一个视图控制器的并跳转处理的handler，那么应该是如下的：



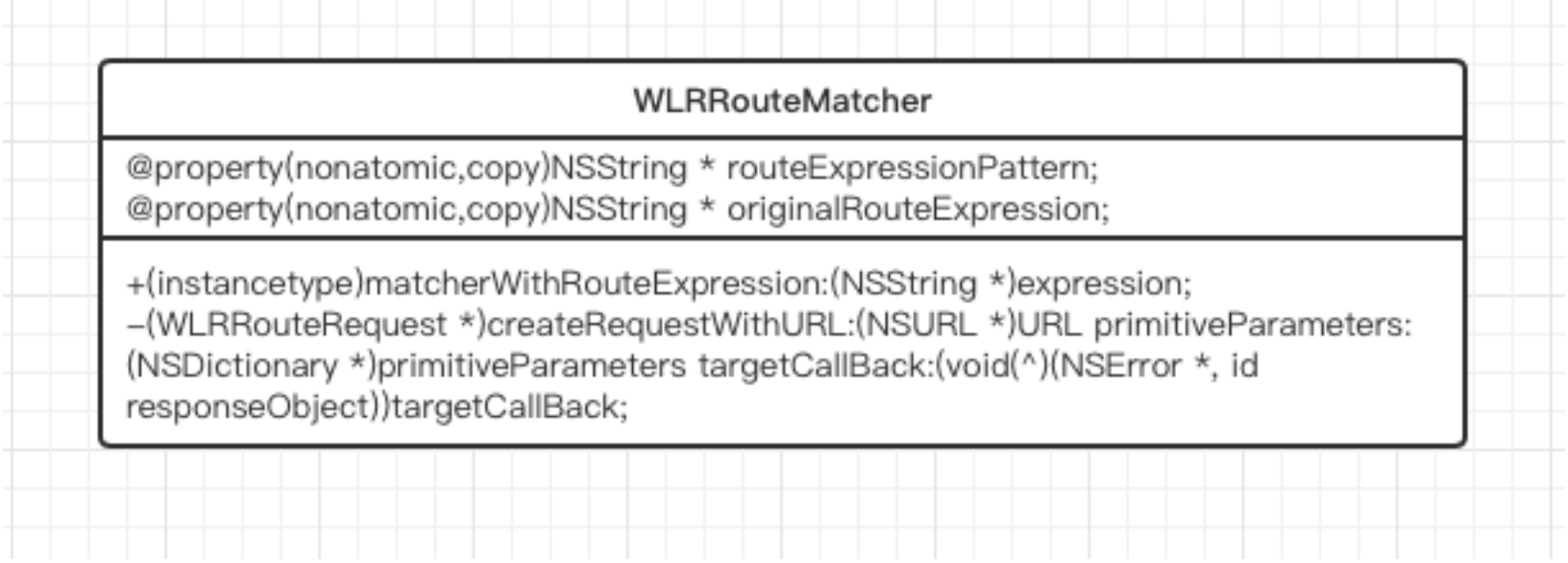
WLRRouteHandler

handler处理一个request请求是一个具有过程性的逻辑，WLRRouteHandler要作为一个基类，我们知道，这个handler在需要处理获取目标视图控制器->参数传递给目标视图控制器->视图控制器的转场->完成回调，那么我们需要设计这样的接口

```
//即将开始处理request请求，返回值决定是否要继续相应request
- (BOOL)shouldHandleWithRequest:(WLRRouteRequest *)request;
//开始处理request请求
-(BOOL)handleRequest:(WLRRouteRequest *)request error:(NSError *__autorelea
// 根据request获取目标控制器
-(UIViewController *)targetViewControllerWithRequest:(WLRRouteRequest *)req
//转场一定是从一个视图控制器跳转到另外一个视图控制器，该方法用以获取转场中的源视图控制器
-(UIViewController *)sourceViewControllerForTransitionWithRequest:(WLRRoute
//改方法内根据request、获取的目标和源视图控制器，完成转场逻辑
-(BOOL)transitionWithWithRequest:(WLRRouteRequest *)request sourceViewContr
//根据request来返回是否是模态跳转
- (BOOL)preferModalPresentationWithRequest:(WLRRouteRequest *)request;
```

WLRRouteMatcher设计

一个matcher应该具有根据url和参数判断是否匹配某个url表达式的逻辑



WLRRouteMatcher

matcher对象必须拥有url的匹配表达式，类似于scheme://host/path/:name([a-zA-Z_-]+)，也有拥有该表达式真正的正则表达式，^scheme://host/path/([a-zA-Z_-]+)$

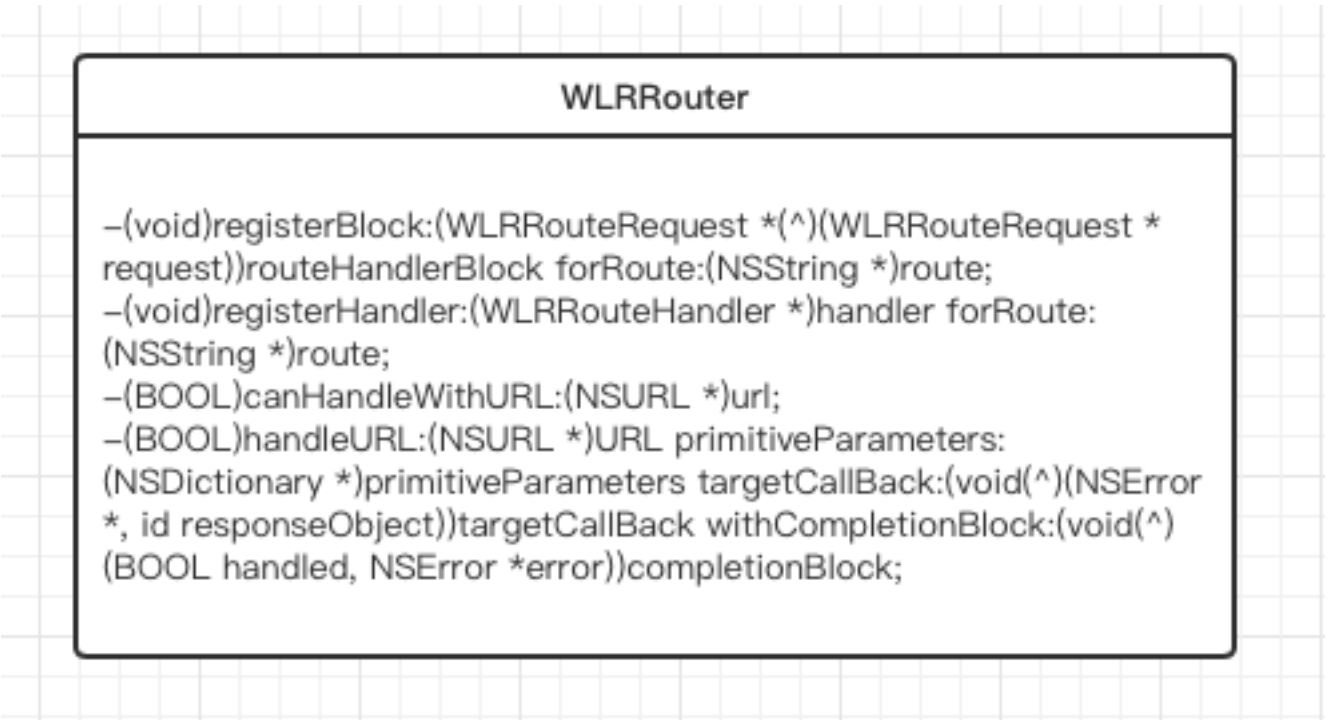
```
@interface WLRRouteMatcher : NSObject
//url匹配表达式
@property(nonatomic,copy)NSString * routeExpressionPattern;
//url匹配的正则表达式
@property(nonatomic,copy)NSString * originalRouteExpression;
+(instancetype)matcherWithRouteExpression:(NSString *)expression;
-(WLRRouteRequest *)createRequestWithURL:(NSURL *)URL primitiveParameters:(
```

设计`-(WLRRouteRequest *)createRequestWithURL:(NSURL *)URL primitiveParameters:(NSDictionary *)primitiveParameters targetCallBack:(void(^)(NSError *, id responseObject))targetCallBack;`这个方法，可以通过传入url和参数，检查是否返回request请求，来表示该WLRRouteMatcher对象所拥有的匹配表达式与url是否能够匹配，这句话有点绕，看不懂的多看几遍。

WLRRouter

WLRRouter是路由实体对象，后端开发者对于路由挂载的概念非常了解，其实这样一个路由实体对象可以完成对URL的拦截和处理并返回结果，事实上，根据前面的梳理和总结，WLRRouter对象内部应该保存了需要匹配拦截的URL表达式，而前面我们知道Url的匹配表达式是存储在WLRRouteMatcher对象中的，并且一个Url传入检查是否匹配也是Matcher对象提供的功能，对于匹配上的Url需要有对应的Handler处理，所以Router对

象的内部存在Machter对象和Handler对象一一对应的关系，并且拥有注册Url表达式对应到Handler的功能，也具有传入Url和参数就能匹配到Handler的功能，还要有一个检测Url是否能有对应Handler处理的功能，所以应该是：



WLRRouter

这里有两种注册的方法，注册handler的就不需再多描述，另外一个是注册Block的回调形式，因为有时候可能会需要一些简单的Url拦截，去做一些事情，这里面的Block需要返回一个request对象，这是因为，如果Block没有对request的回调做处理，Router应该处理调用者的回调问题，否则就会出现调用者设置了回调的Block而没有人调用回来，这样就尴尬了。

```
/**
 注册一个route表达式并与一个block处理相关联

 @param routeHandlerBlock block用以处理匹配route表达式的url的请求
 @param route url的路由表达式，支持正则表达式的分组，例如app://login/:phone({0,9+
 */
-(void)registerBlock:(WLRRouteRequest *(^)(WLRRouteRequest * request))route
/**
 注册一个route表达式并与一个block处理相关联

 @param routeHandlerBlock handler对象用以处理匹配route表达式的url的请求
 @param route url的路由表达式，支持正则表达式的分组，例如app://login/:phone({0,9+
 */
-(void)registerHandler:(WLRRouteHandler *)handler forRoute:(NSString *)rout

/**
 检测url是否能够被处理，不包含中间件的检查

 @param url 请求的url
```

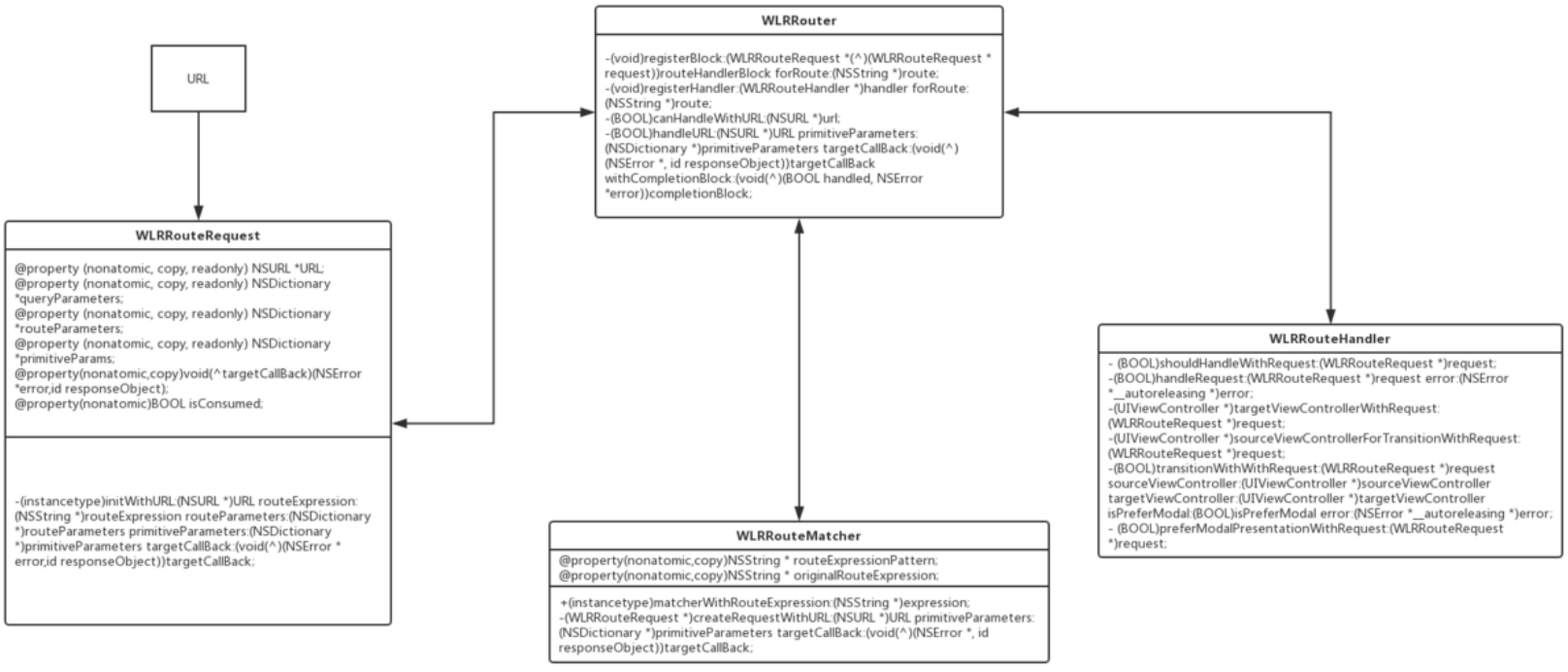
```
 @return 是否可以handle
 */
-(BOOL)canHandleWithURL:(NSURL *)url;
/**
 处理url请求

 @param URL 调用的url
 @param primitiveParameters 携带的原生对象
 @param targetCallBack 传给目标对象的回调block
 @param completionBlock 完成路由中转的block
 @return 是否能够handle
 */
-(BOOL)handleURL:(NSURL *)URL primitiveParameters:(NSDictionary *)primitive
```

梳理总结：

从以上我们规划的几个类的接口，我们可以清楚的看到Router工作的流程。

1. 首先实例化Router对象
2. 实例化Handler或者是Block，通过Router的注册接口使得一个Url的匹配表达式对应一个Handler或者是一个block
3. Router内部会将Url的表达式形成一个Matcher对象进行保存，对应的Handler或处理的Block会与Matcher一一对应，怎么对应呢？应该使用路由表达式进行关联
4. Router通过handle方法，接收一个Url的请求，内部遍历所有的Matcher对象，将Url和参数转换为Request对象，如果能转换为Request对象则说明能匹配，如果不能则说明该Url不能被路由实体处理
5. 拿到Request对象以后，则根据Matcher对应的路由表达式找到对应的Handler或者是Block
6. 根据Handler的几个关键方法，传入Request对象，按照顺序完成处理逻辑的触发，最后如果有request当中包含有目标的回调，则将处理结果通过回调的Block响应给调用方
7. Handler完成处理后，Router完成本次路由请求



WLRRoute

> Tips:
> 很多开发者把敏捷开发当做来了需求不管三七二十一，一把梭子就是干，不断写不断改。