**iOS 组件化方案分析**

一 组件化背景

随着项目的越来越大和开发人员的增加，多人协作在同一个项目中开发遇到的问题也越来越突出：多人修改同意文件导致冲突；开发一个模块，程序员要等待整个项目编译完成；代码交互错杂，开发一个模块可能要修改其他模块代码，导致测试任务加大…

二 原理

组件化就是通过代码模块化拆分，降低业务模块之间的耦合度，使多个模块可以单独开发测试，模块之间通过特定的方式进行通信。

三 方案类型(主要针对模块之间的通信展开讨论，因为这一块是任何项目组件化的共性)

1 URL注册方案

2 Protocol注册方案

3 Target-Action runtime 调用方案

1 URL注册方案

通过URL硬编码，web与 app内部定义一套统一的url跳转规则，各个模块之间可以不用通过头文件引入就可以达到通信和页面跳转功能。

典型SDK：JLRoutes

优点:可以将web和app内部跳转规则统一，代码里不用区分来源

缺点:参数是一个典型的字典，调用方不知道服务方需要哪些参数。

2 Protocol注册方案

通过创建代理接口的方式为模块之间搭建通信的桥梁，每个模块可以创建一个与外部通信的接口，同时创建一个代理管理模型，负责模块获取代理对象，（例如：接口:<orderDetailProtocol> 代理管理模型:protocolServiceManager 代理: orderDetailProtocolService 执行过程: protocolServiceManager里保存orderDetailProtocol的代理orderDetailProtocolService ,orderDetailProtocolService 实现了接口，模块之间调用时只需要依赖 接口orderDetailProtocol和代理管理模型：protocolServiceManager 从而达到了模块之间的通信 ）

优点：参数由接口显示提供给调用者，调用方明确服务方需要哪些参数

缺点: 需要提前注册代理管理中心，代理常驻app生命周期，仅使用于本地调用

方案图

模块2

模块3

模块1

代理3

代理2

代理1

接口3

接口2

接口1

代理管理Model

3 Target-Action runtime 调用方案

为每个模块创建与外界交互的Target类，创建一个公共的动态执行Target的执行类，通过运行时方式负责底层调用模块中的Target类，达到通信功能。

（例如通过NSClassFromString，NSSelectorFromString和performSelector等运行时方法进行模块之间的通信） 为了方便各个模块之间的通信，需要建立一个执行类的扩展类，模块之间通过依赖扩展类建立联系

优点：参数由接口显示提供给调用者，调用方明确服务方需要哪些参数，运行时动态调用，无需提前注册，避免内存驻留问题。

缺点: 仅使用于本地调用，无法远程调用。

方案图

模块2

模块3

模块1

Target3

Target2

Target1

执行类（Runtime）

执行类扩展1

执行类扩展1

执行类扩展1

四 方案组合优化

以上三种方案只是模块之间相互通信的机制，每种都有利弊，一套好的组件化方案需要多种通信机制的配合使用。

例如：Target-Action runtime 调用方案 与URL注册方案结合使用，远程调用用URL注册方案，（先解析url规则，再转化成本地调用）本地调用用Target-Action runtime

好处：远程调用使用url注册机制，符合web传参方式，本地调用采用Target-Action runtime

方式符合本地传参方式。

本人建议远程调用与本地调用分开，因为一个app中大部分都是本地调用，业务开发者在调用其他模块时需要明确知道服务方所需参数，这样方便业务开发，也易于后期代码维护。远程调用传参方式过于隐式，不方便业务开发者开发

五 总结与引用

1 整体架构

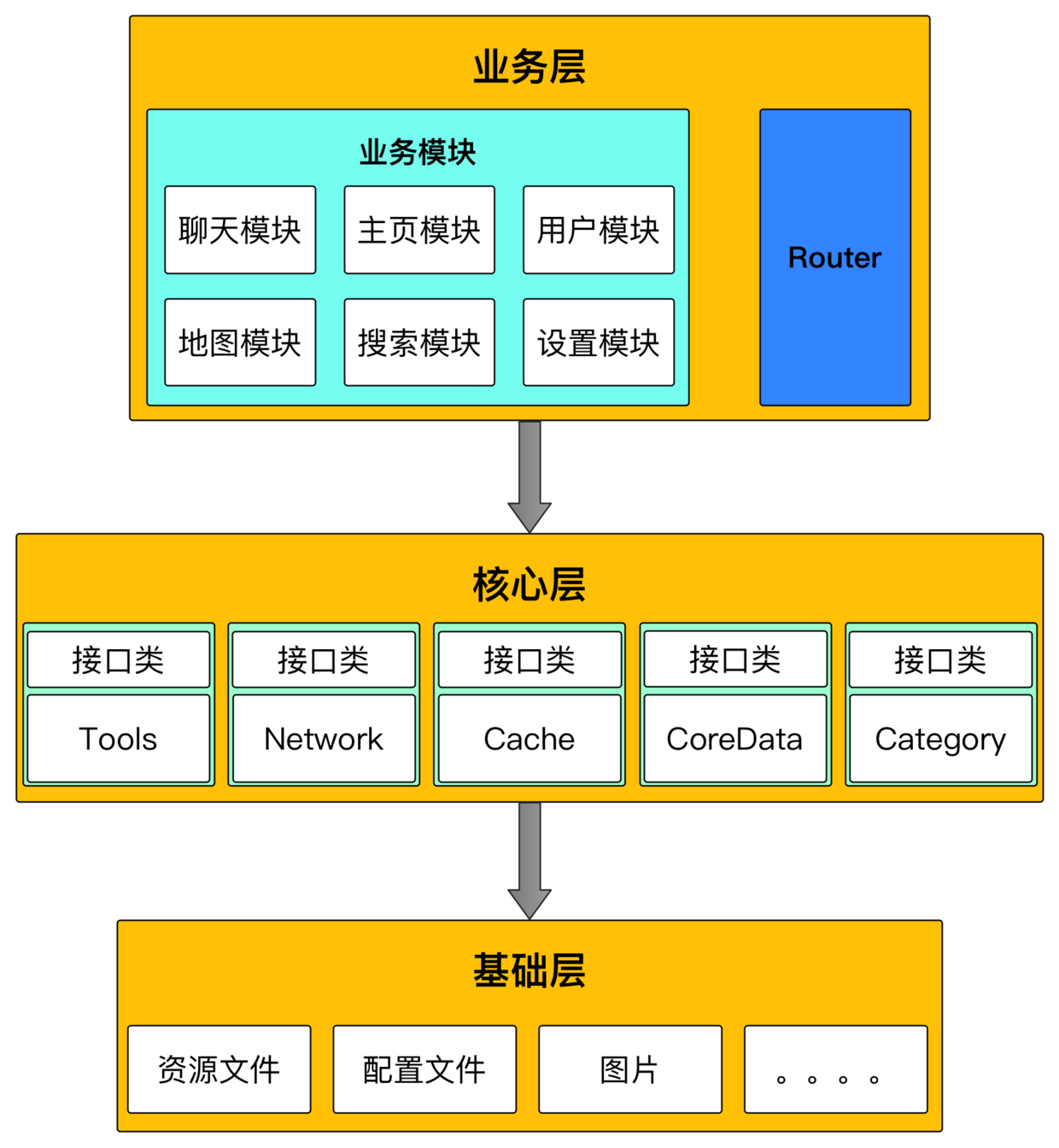
组件化架构中，首先有一个主工程，主工程负责集成所有组件。每个组件都是一个独立的工程，创建不同的私有仓库来管理，每个组件都有对应的开发人员负责开发，开发人员只需要关注与其相关的组件代码，其他业务代码与其无关，来新人也好上手。

组件划分需要注意组件粒度，粒度根据业务可大可小。组件划分后属于业务组件，对于一些多个组件共同的东西，例如网络，数据库之类的，应该划分到单独的组件或基础组件中，对于图片或配置表这样的资源文件，应该再单独划分一个资源组件，这样避免资源的重复性。

服务方组件对外提供服务，由中间件调用或发现服务，服务对当前组件无侵入性，只负责对传递过来的数据进行解析和组件内调用的功能。

通过这样的组件划分，组件的开发进度不会受其他业务的影响，**可以多个组件单独的并行开发**。组件间的通信都交给中间件来进行，需要通信的类只需要接触中间件，而中间件不需要耦合其他组件，这就实现了组件间的解耦。**中间件负责处理所有组件之间的调度**，在所有组件之间起到控制核心的作用。

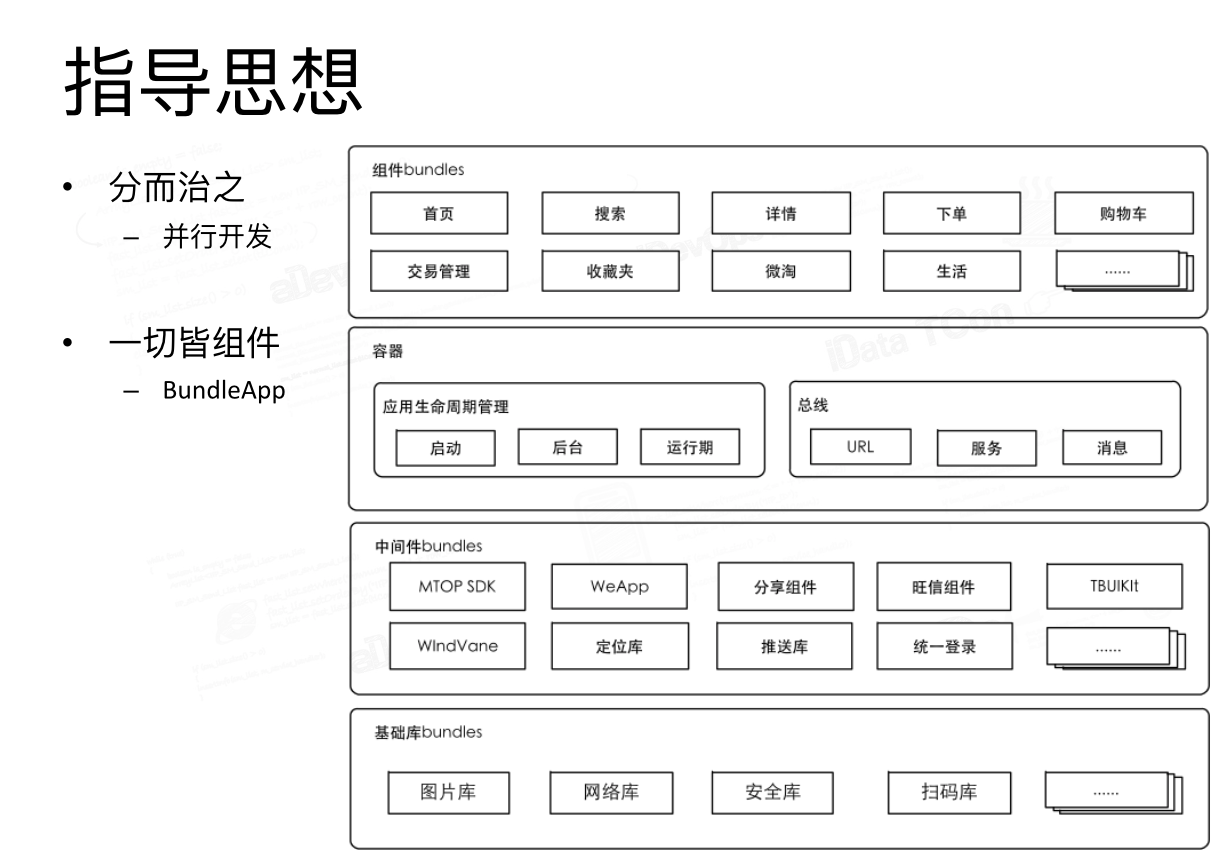
引入别人的整体架构图

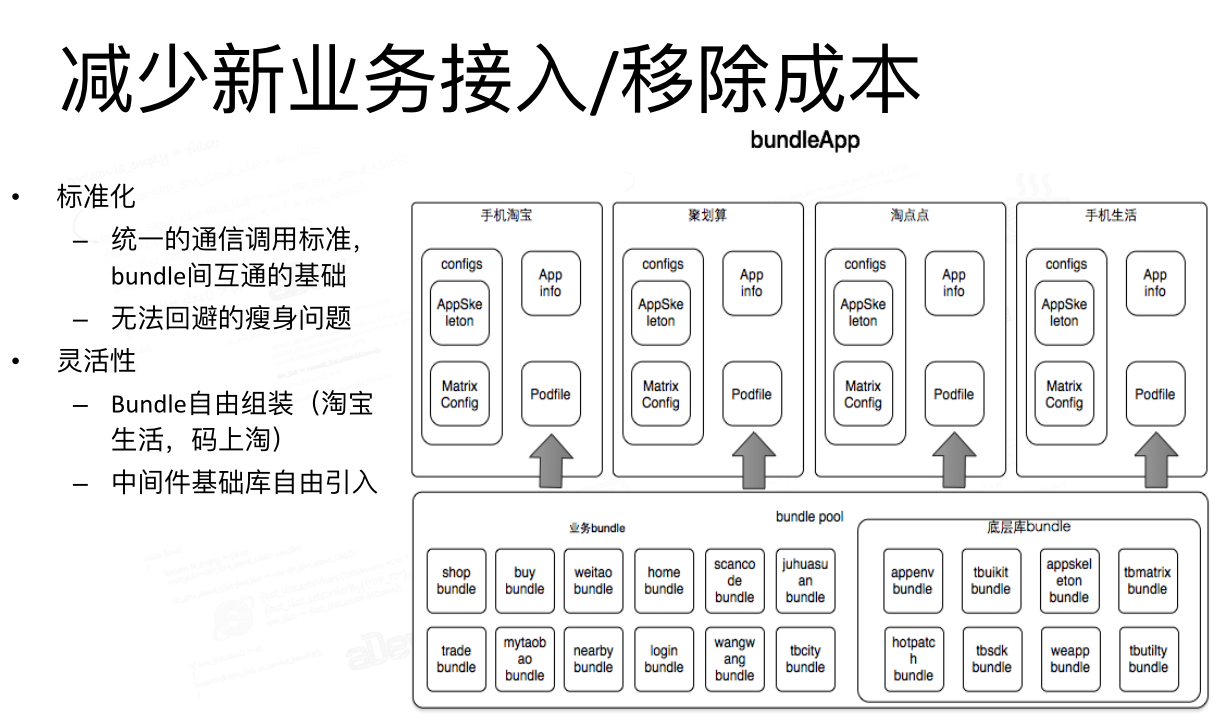


问题：业务组建的模型应放在哪里？

如果每个模块之间的通信涉及到了自定义模型参数，如果引入模型，势必会带来代码耦合，建议1 模型下沉到基础组建 2 业务组建通信传参时进行去model化处理（传递字典或分离model中的属性为多个参数，建议多参数传递，因为用字典又会进入传参隐式的尴尬局面）

淘宝的组件化拆分





阿里的BusBundle方式实际为远程调用navtive组建与本地调用native组建定义了统一的 URL规则（与正常https和http请求做了区分，但是也兼容了普通网络请求），在调用上却做了区分。本地URL根据不同的组建存储于不同的xml文件中，app启动将所有xml结构化读取到内存中，即URL注册，调用远程URL优先检查是否符合web规则(兼容https与http请求)，符合：直接跳转到UIwebIVew界面。不符合：再去内存的URL注册map中查找符合规则的元素，做对应的native组建跳转。

六 讨论

1 本地调用与远程调用分开好还是统一好？

2 业务组建的模型应放在哪里（与组件间传参相结合）？

3 如何减小各组件间的依赖关系？（或者什么方法可以将依赖关系不对外暴露，方便业务组建独立开发和调试）

4 图片资源，数据库资源放在各自的组件中好还是放在基础组件中好，还是做区分（公用的放在基础组件中，不共用的放在各自的业务组件中）？（注：把后期维护考虑在内）

5 如何做一套完善的组件版本管理体系，不影响各业务线开发?（例如：基础组件A 目前各业务线正在依赖于基础组件A 1.0版本开发新的业务，同时，基础组件A也在开发2.0版本，如何只让一个业务模块适用基础组件A的2.0版本，其他业务线组件仍用1.0版本）

6 如果用cocoapod管理组件，太多的依赖关系会导致install一遍pod很慢，如何解决这个问题，采用本地化组件方式更新如何？（这样版本管理又是个问题）

7 讨论得出一个合理的组件拆分结构图

Demo： https://github.com/jxzjxz/-Demo.git