iOS端组件化整套技术方案

- 你负责的项目是否需要组件开发？
- 组件开发实现方式三种技术方案
- 组件生命周期管理
- 组件通信历程解析
- 组件通信方案（中间件）
- 移动端 APP 通用型 架构 示意图及概述说明
- 公司多产品线APP组装示意图
- iOS通用型多项目整体结构图
- iOS基石示例组件模板仓库结构图
- 多项目组件化简化示意结构图（二进制，源码，源码映射等相关）
- 整体流程示例演示
- 三方公用库统一方案
- 其他补充等

https://github.com/DevDragonLi

2021年6月 DevDragonLi

你负责的项目是否需要组件开发？

> 综合公司现状及项目实际情况来判断

- 人员：随着人员的增加，过多人对同仓库的并行修改概率增加，这时候需要对人员和其维护的功能需要进行重新分配。
- 项目：项目的业务线超过2条以上，需要独立拆分。随着业务的拆分，对应的业务组件也就很自然的独立出来。
- 测试维度：随着项目的业务量增大，**不做组件化，就很难做单元测试**。每个小功能修改，测试都需要对App进行测试，严重增加测试工作量。

综上：当你的App业务之间交叉耦合，bug率难以下降，测试每天做大量重复工作。开发人员之间修改项目影响时，你需要考虑进行组件化。

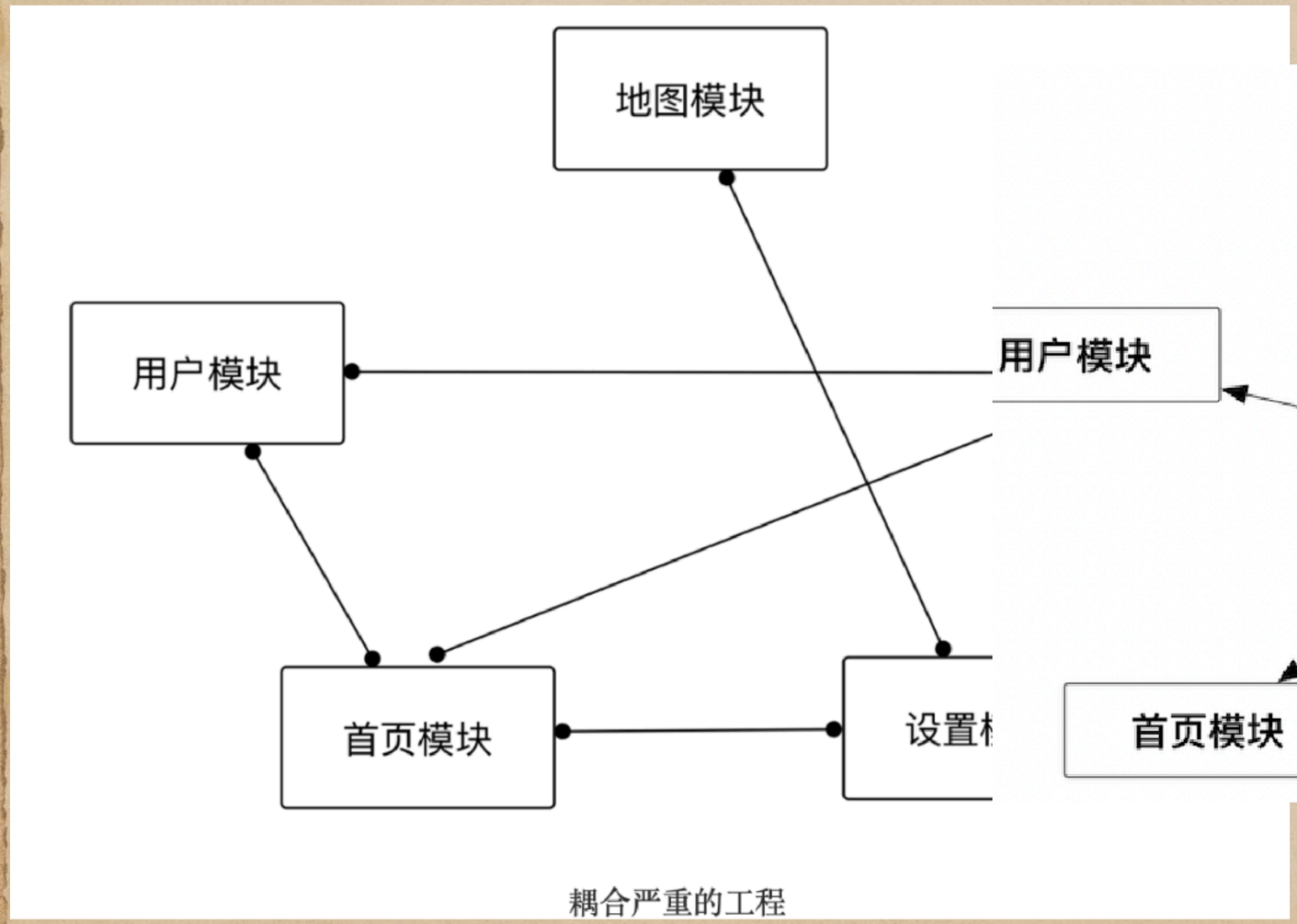
组件开发实现方式三种技术方案

- CocoaPods
- EasyBox【未开源】
- Target- Target 【Xcconfig】
- ...

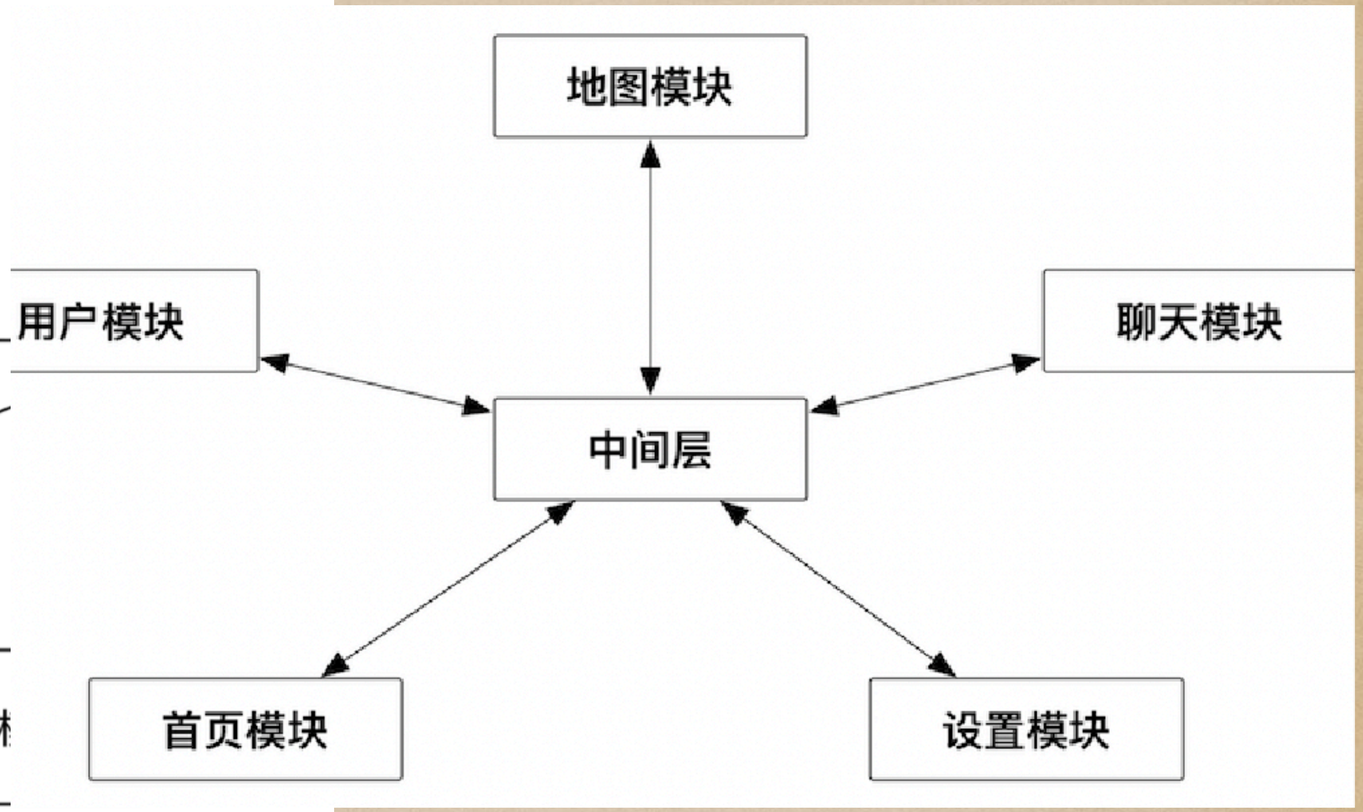
组件生命周期管理

- 组件的生命周期，与App的生命周期应该保持一致。
- AppDelegate 作为程序级状态变化的 delegate【iOS13 +SceneDelegate】，应该只做路由和分发的作用，具体逻辑实现代码还是应该在分别的模块中，这个文件应该保持整洁，除了的方法外不应该出现其他方法
- 解决项目组件化解耦宿主工程派分事件:https://github.com/DevDragonLi/ZDModuleKit
- 图示说明：https://github.com/DevDragonLi/ZDModuleKit/blob/master/images/image.jpg

组件通信历程解析



耦合严重的工程



组件通信方案（中间件）

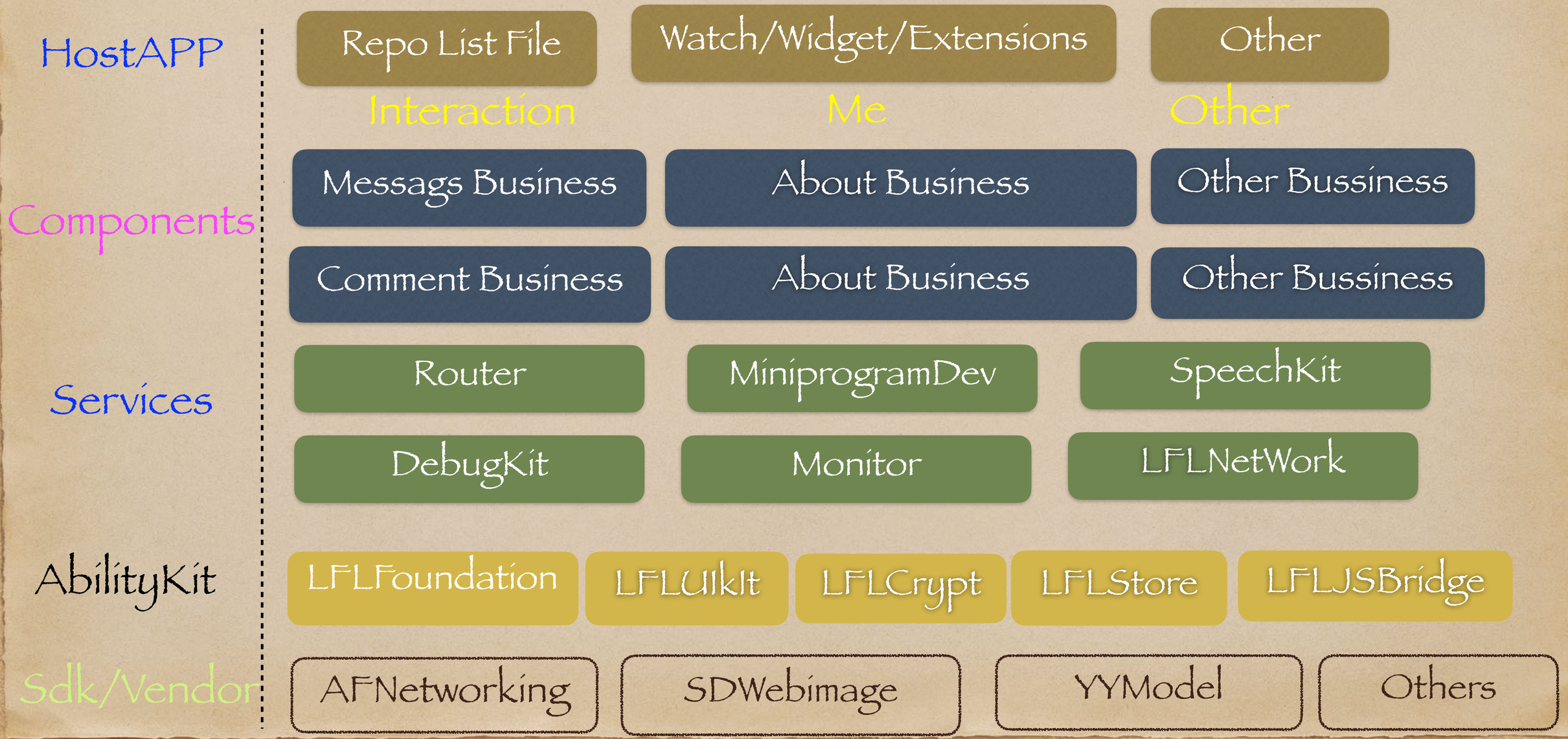
- URL Router
- Target-Action
- Protocol-Class
- https://github.com/DevDragonLi/ProtocolServiceKit【star点一点，follow !follow 一下。😄】

1.各个页面和组件之间的跳转问题。

2.各个组件之间相互调用。

https://github.com/DevDragonLi

移动端 APP 通用型 架构 示意图

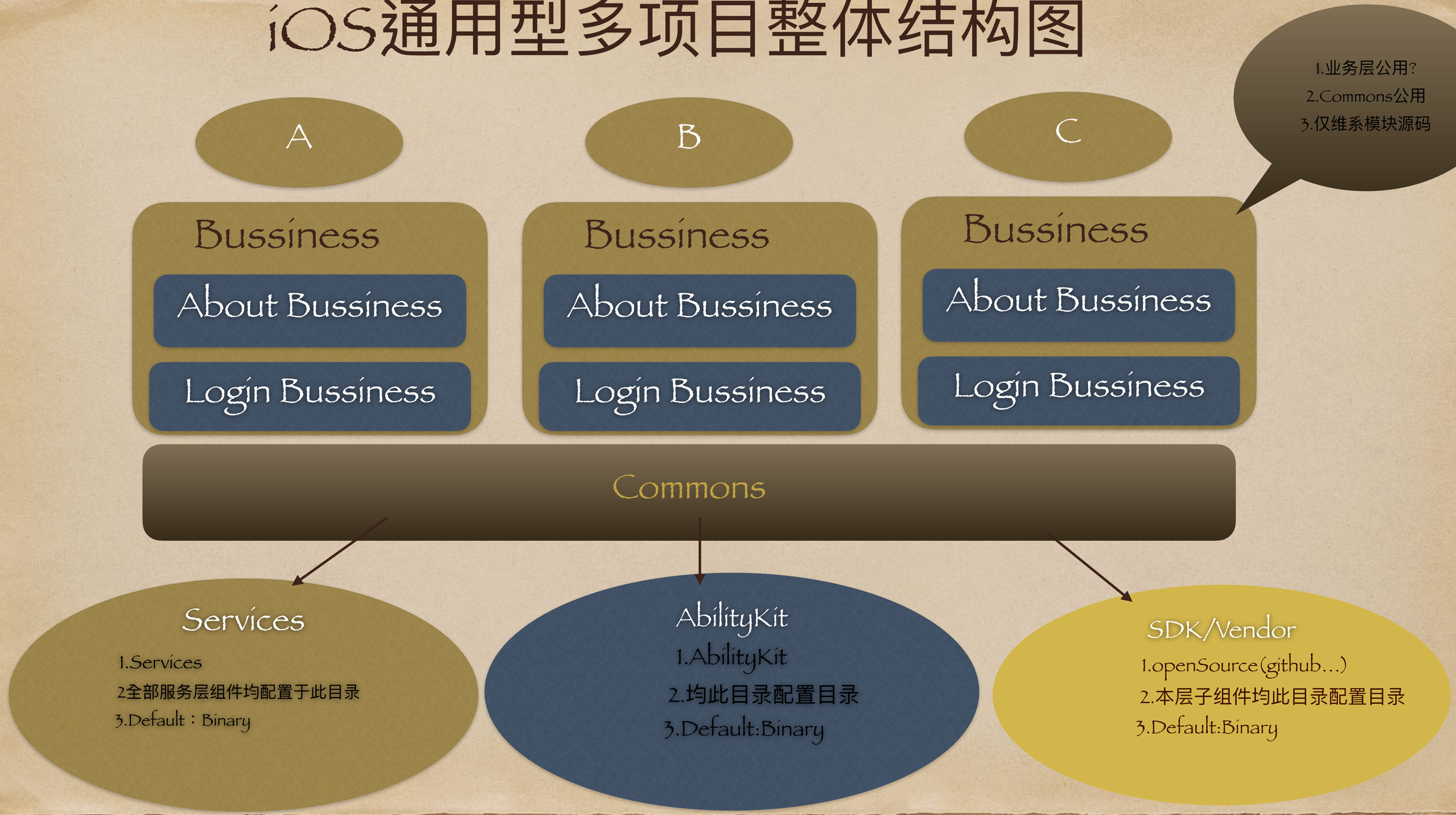


移动端 APP 通用型 架构 概述

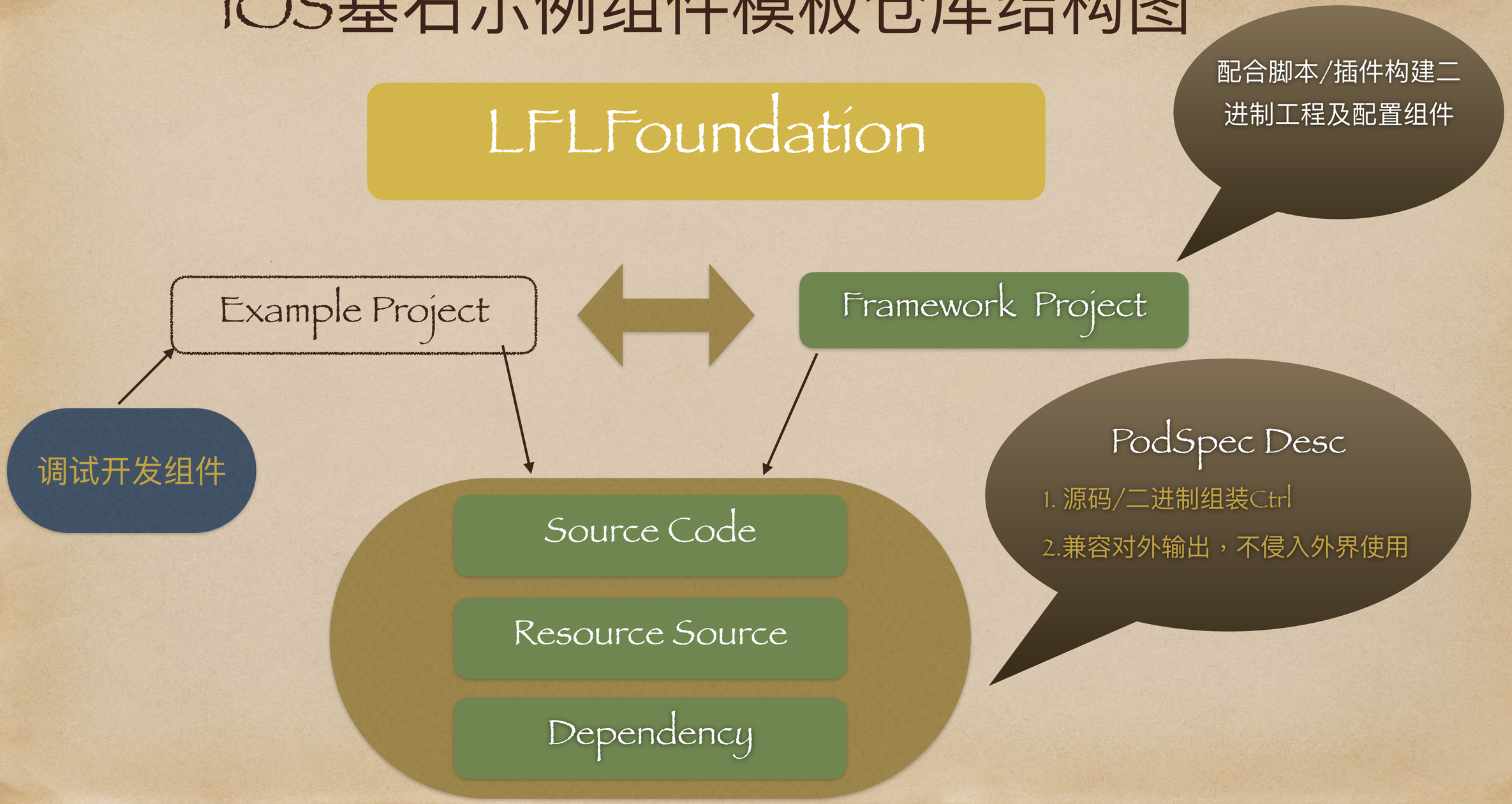
- 自上而下严格分层，不建议跨层依赖【Messags Business > AFNetworking❌,Messags Business > LFLNetWork✅ ...】
- Components：任意业务模块不可相互依赖(Router)，模块内自维系依赖，可多个业务组件组合为任意`APP`
- Services：AbilityKit之上服务层，可被任意`APP`及`Components`层引用。【Example：DebugKit】

公司多产品线APP组装示意图

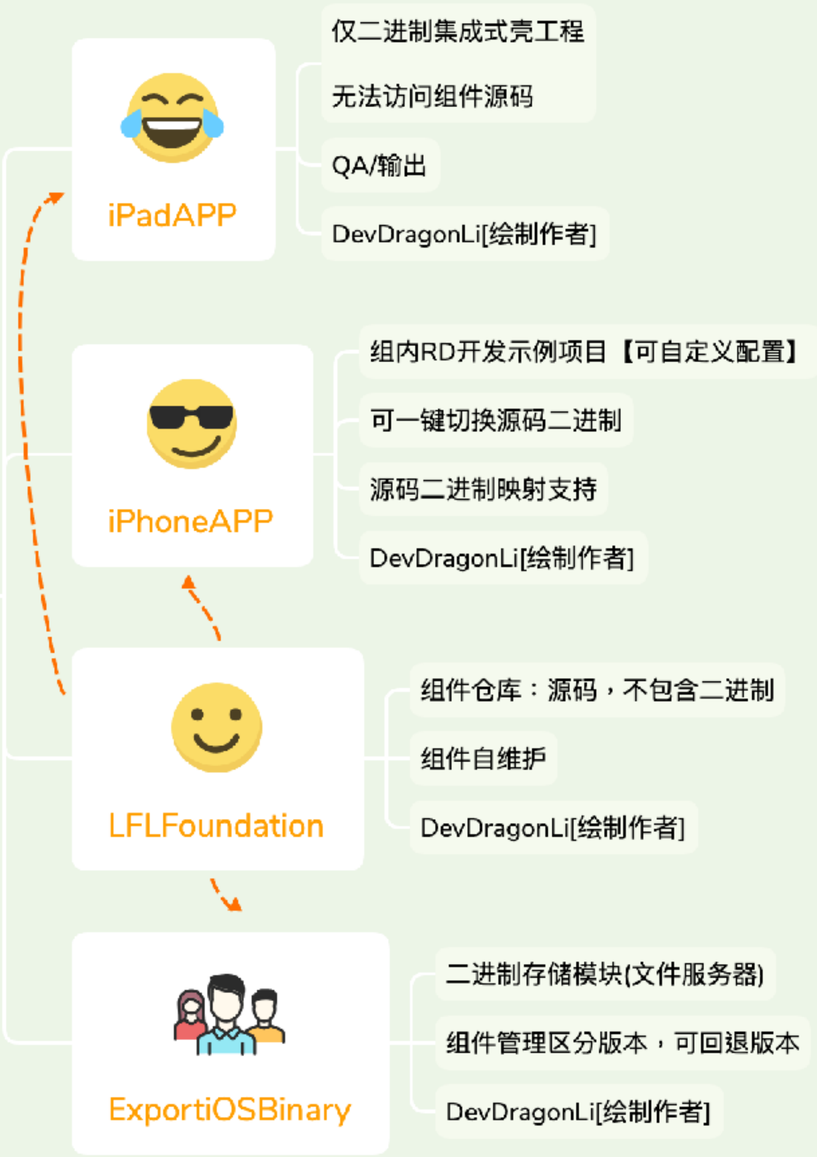
iOS通用型多项目整体结构图



iOS基石示例组件模板仓库结构图



多项目组件化简化示意结构图



整体流程示例演示

- 多项目工程及组件示例
- 构建二进制插件[公版]
- 更新组件脚本[已开源]
- 多仓库管理[已开源]
- 详细版本后续专题讲解

iOS示例组件简要说明

- 团队内按照WIKI 新增组件相关`基石`工作【WIKI】
- 调试及开发阶段可依托于 Example Project/Host APP【均支持】
- 可支持对外输出及打包上线。
- 资源文件：`use_frameworks`【兼容支持是否开启】
- 架构支持区别：线上模式：Arm64+x86_64，线下模式：ARM64(x86_64?)
- 二进制说明：(1)条件允许均公司内部文件服务器托管，结构参考后文 (2)条件有限,则：非三方组件暂不支持远程托管，均本地构建，三方参考后文
- 组件内完全解耦，不存在跨级依赖！

iOS APP示例项目补充说明

- 任何APP 均配置多仓脚本统一托管【一一对应配置托管仓库】，构建`APP` 脚本自动化。
- RD开发模式：默认除核心业务外(配置列表)，大部分二进制参与。
- 具备源码二进制一键调试能力及对外输出二进制组装模式。
- 组装：公用库更新->业务层更新->构建二进制 -> 自动打开壳工程
- 更新构建规则：方案细节

iOS端组件化收益

- 组件颗粒度细化，适应组合配置及中台输出能力。
- 工程编译组装速度明显提升及构建流程完善。
- 开发效率提升，并行开发能力。
- 具备`APP`快速构建组装能力
- ...

三方公用库统一方案【Beta】

- 开源三方库迁移公司内部部署【统一组件源】
  - 备注 1 ：可以做到一键切换【公/私源】 &&【版本对齐】
    - **对外输出/打包则切换为全源码，内部开发期间使用二进制版本**
    - 脚本桥接函数实现。
  - 备注2 ：开源库均需要选择Stable 版本。
  - 备注3 ：部分高频更新，存在严重问题解决版本的仓库，定期维护更新即可 【底成本】
- 无用三方依赖移除
  - 较低使用，可更优替代方案。
  - 重量级框架，项目仅使用较少/较浅等。
  - 问题较多/不再维护等。

Thanks

投币，点赞，关注

1V1服务

2021年6月 DevDragonLi