基于MVVM的实践

GomePlus移动端架构

刘继坤

国美互联网生态（分享）有限公司

01 Agenda

- MVVM的核心概念
- GomePlus的困境
- 重构目标与整体架构
- 实施方案与经验总结
- Q&A

02 MVVM相关的概念

Model

业务逻辑的处理及对应状态的保存

View

业务数据的可是化展示及用户输入信息获取

Controller

根据用户输入协调Model完成相应业务逻辑

Presenter

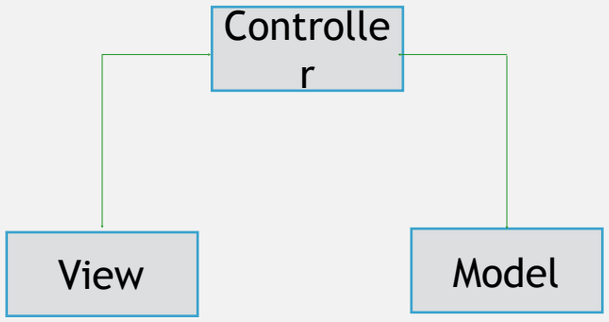
根据用户输入协调Model完成业务逻辑，并将Model中数据的变化更新到View中

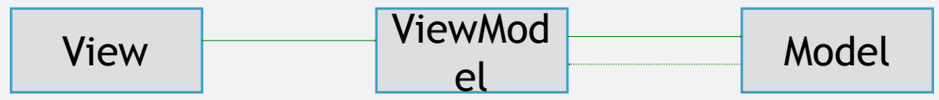
ViewModel

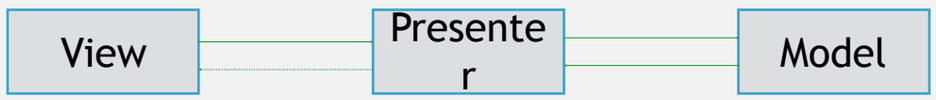
根据用户输入协调Model完成业务逻辑，根据**绑定**自动更新View到相应状态

03 MVVM相关的概念

MVC → MVP → MVVM







04 困境

开发测试周期变长

随着产品功能的增加，单个功能引入的人力成本越来越高，任何一个小的需求都需要大量的回归测试才能保证产品质量

内部沟通成本变高

不合理的设计导致团队成员需要深刻理解非本人业务内部逻辑才可以准确实现跨业务模块的需求，带来极大的沟通成本

代码复用度变低

成员担心相关的改动影响已有的业务，项目中存在大量的copy代码

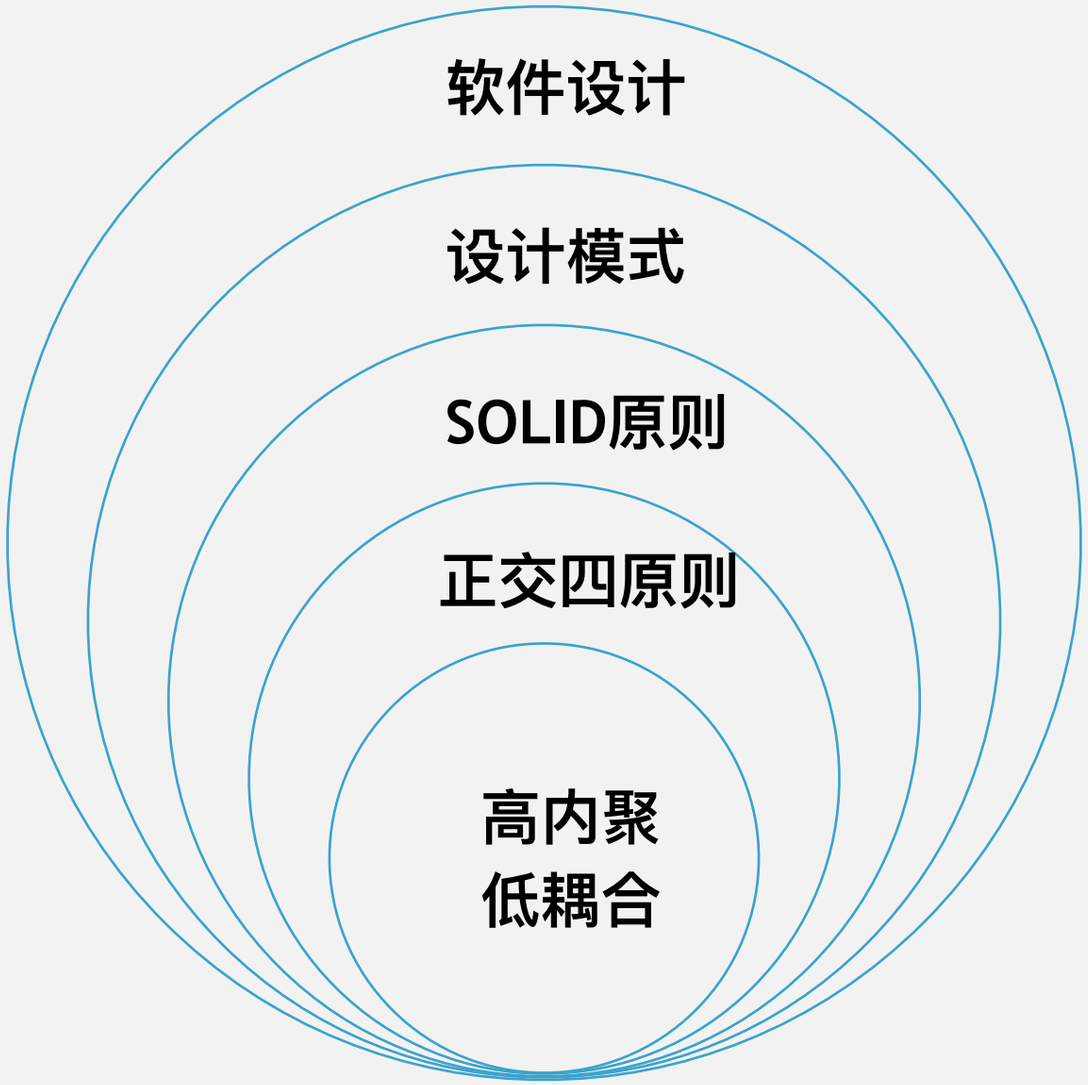
可测性变差

业务间和业务内部边界不明显，导致项目很难进行有效的单元测试

05 重构目标

- 可测性
  - 界面
  - 功能
  - 业务
- 可维护性
  - 职责明确
  - 风格统一
  - 结构统一
- 可复用性
  - 业务逻辑
  - 功能模块
  - 组件化

06 重构目标-方案选型-模块间



正交四原则

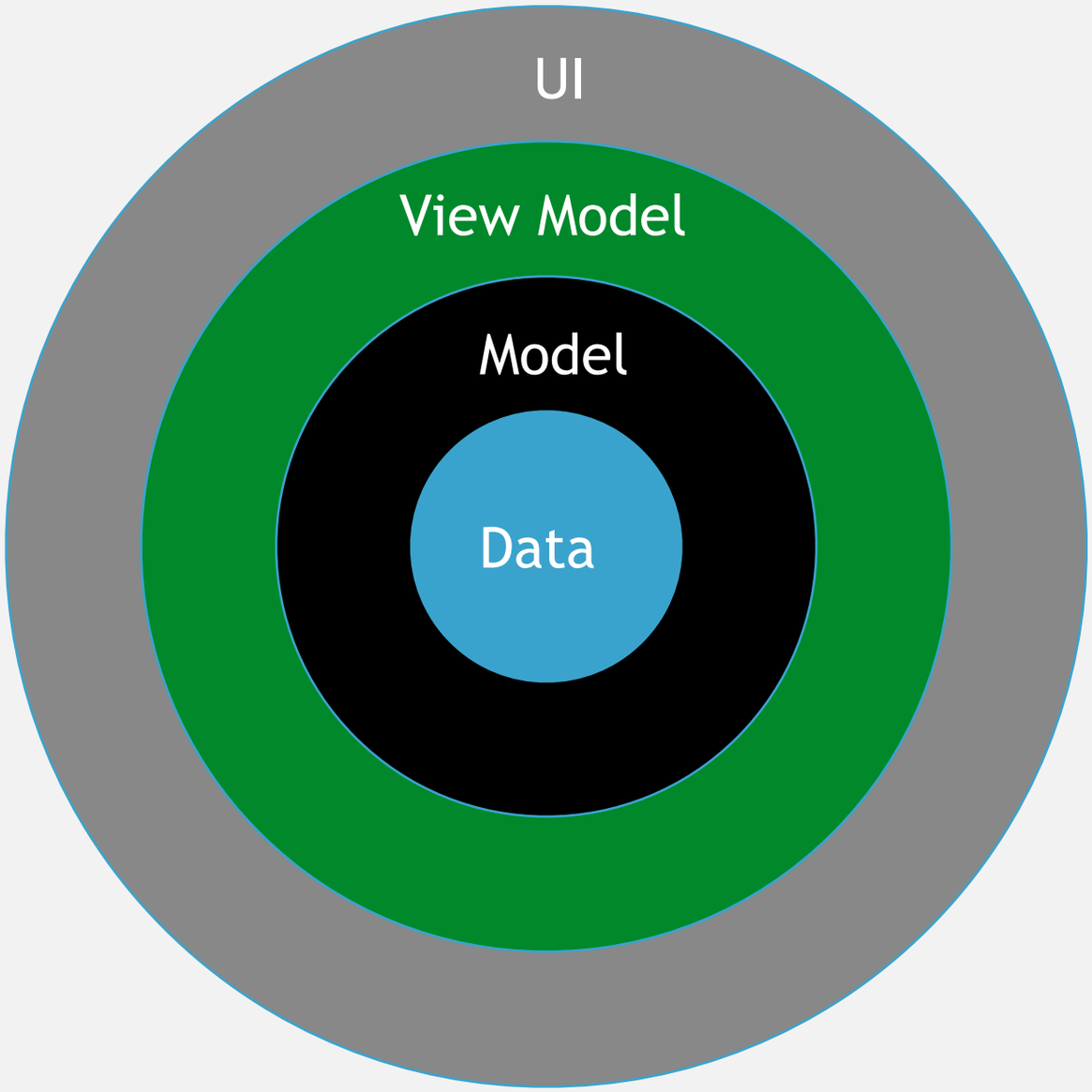
消除重复

分离不同方向的变化

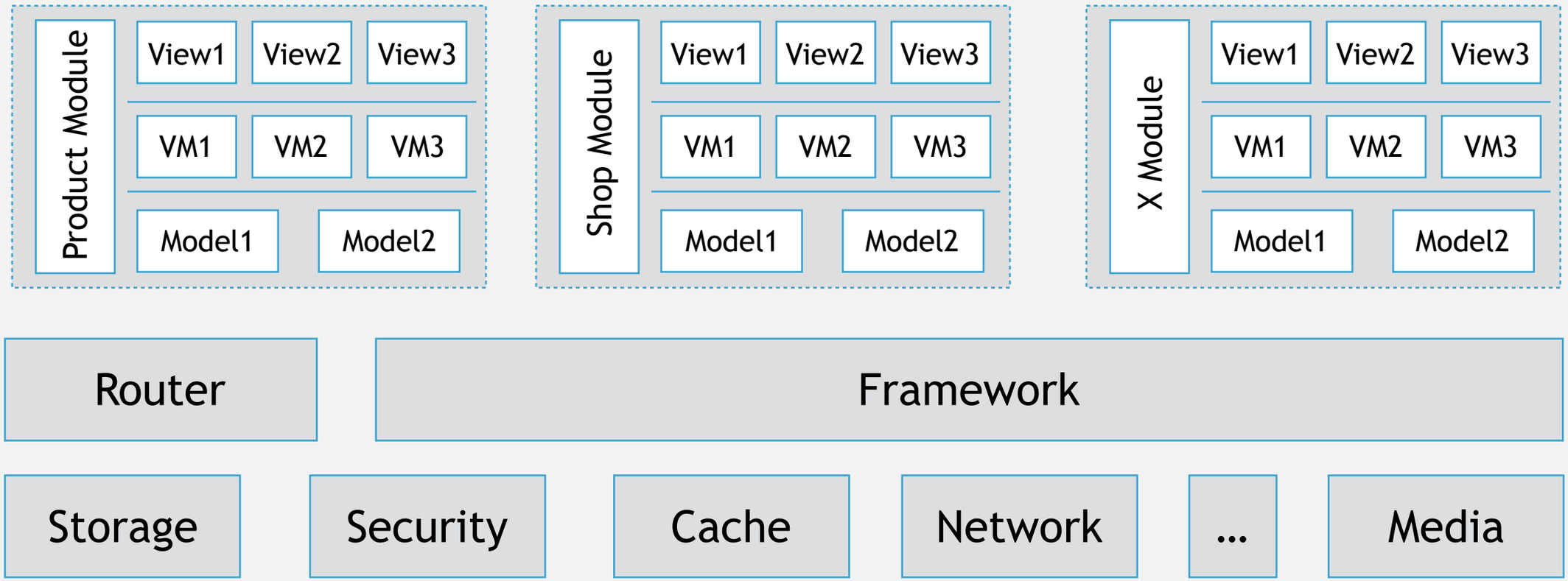
缩小依赖范围

向着稳定的方向依赖

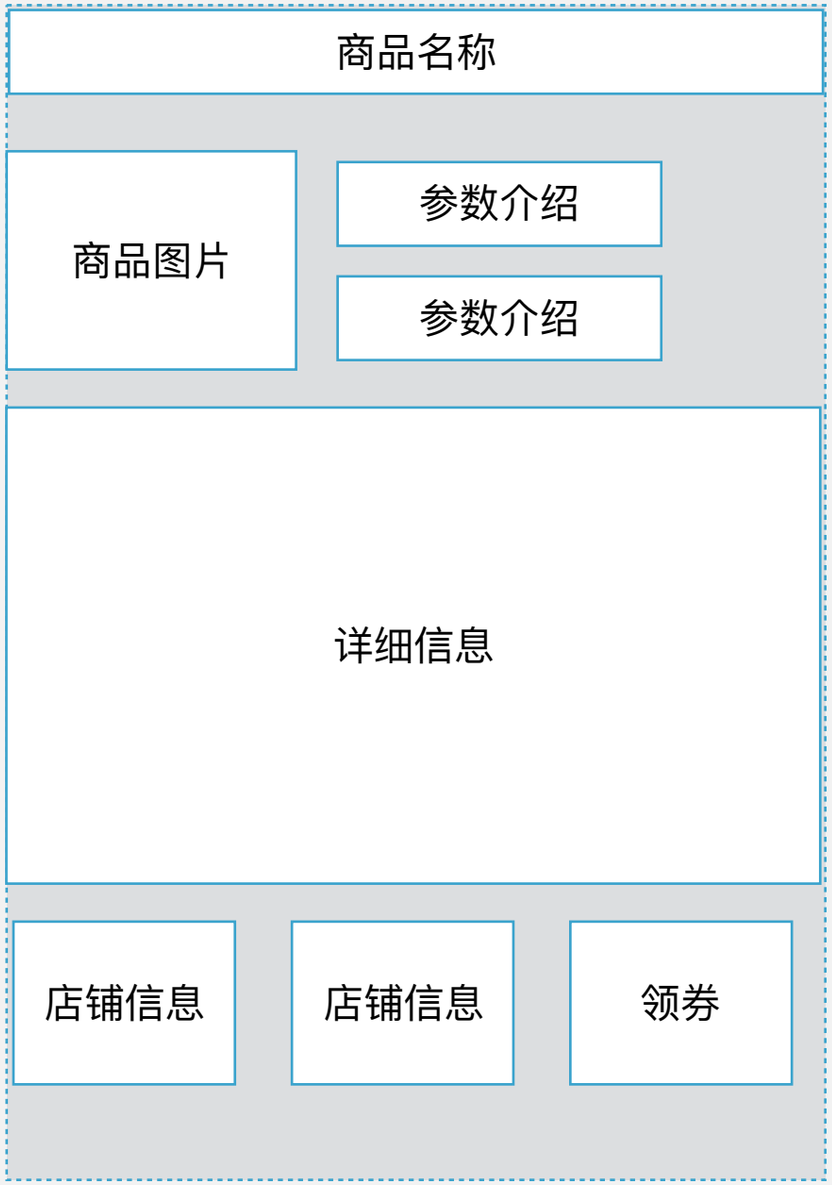
08 重构目标-方案选型-模块内



09 整体架构


Product Module
View1
View2
View3
VM1
VM2
VM3
Model1
Model2
Shop Module
View1
View2
View3
VM1
VM2
VM3
Model1
Model2
X Module
View1
View2
View3
VM1
VM2
VM3
Model1
Model2
Router
Framework
Storage
Security
Cache
Network
...
Media

10 整体架构



11 实施方案

1. 定义模块边界
2. 切分模块
3. MVVM重构

12 MVVM经验总结

VM/Model生命周期

VM关联UI的生命周期

Model常驻内存与引用计数

VM职责界定

响应View事件

Model层数据翻译

内部状态恢复

依赖于UI上下文的通用操作

业务严格遵守MVVM模型

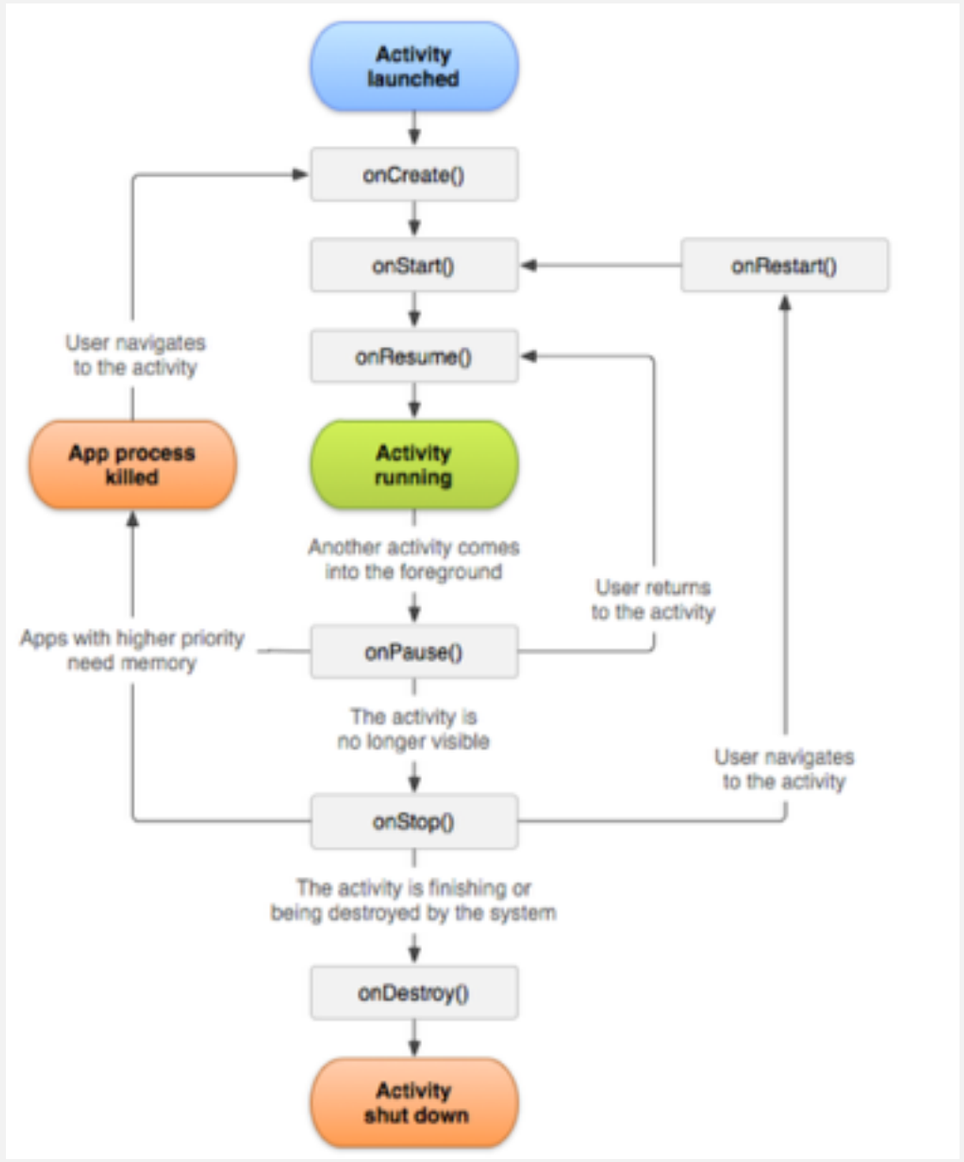
Data Binding Proxy

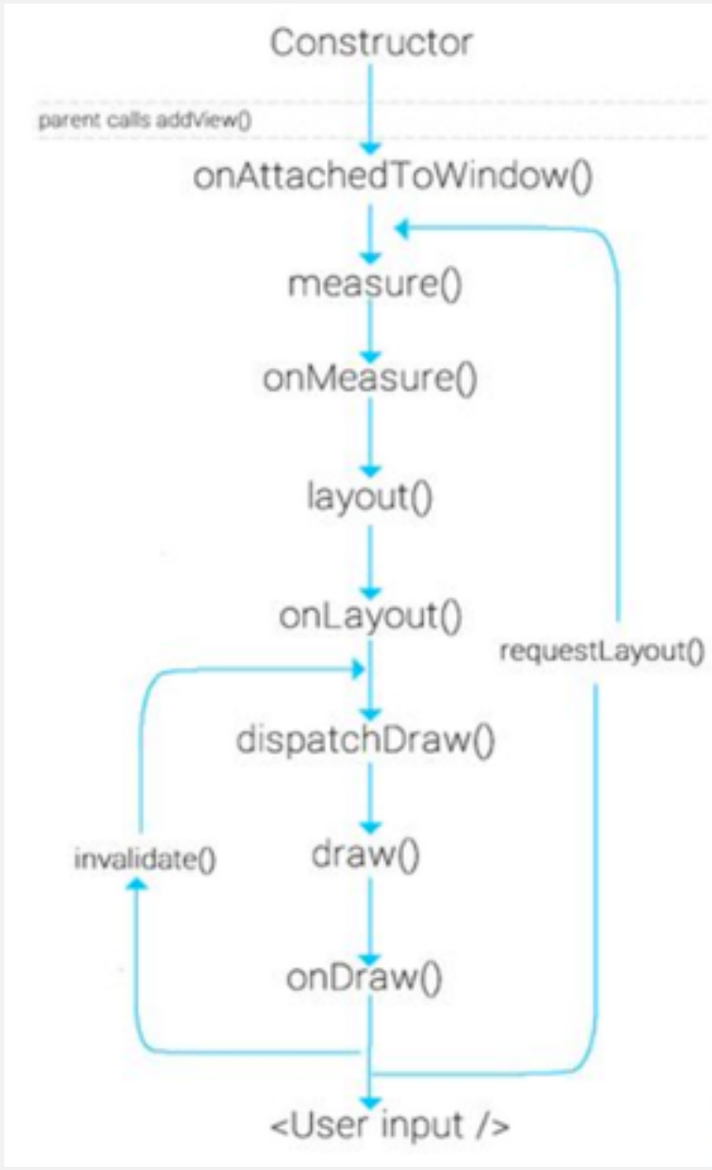
VM/Model数据对象隔离

VM层的数据对象严格对应View的元素和状态

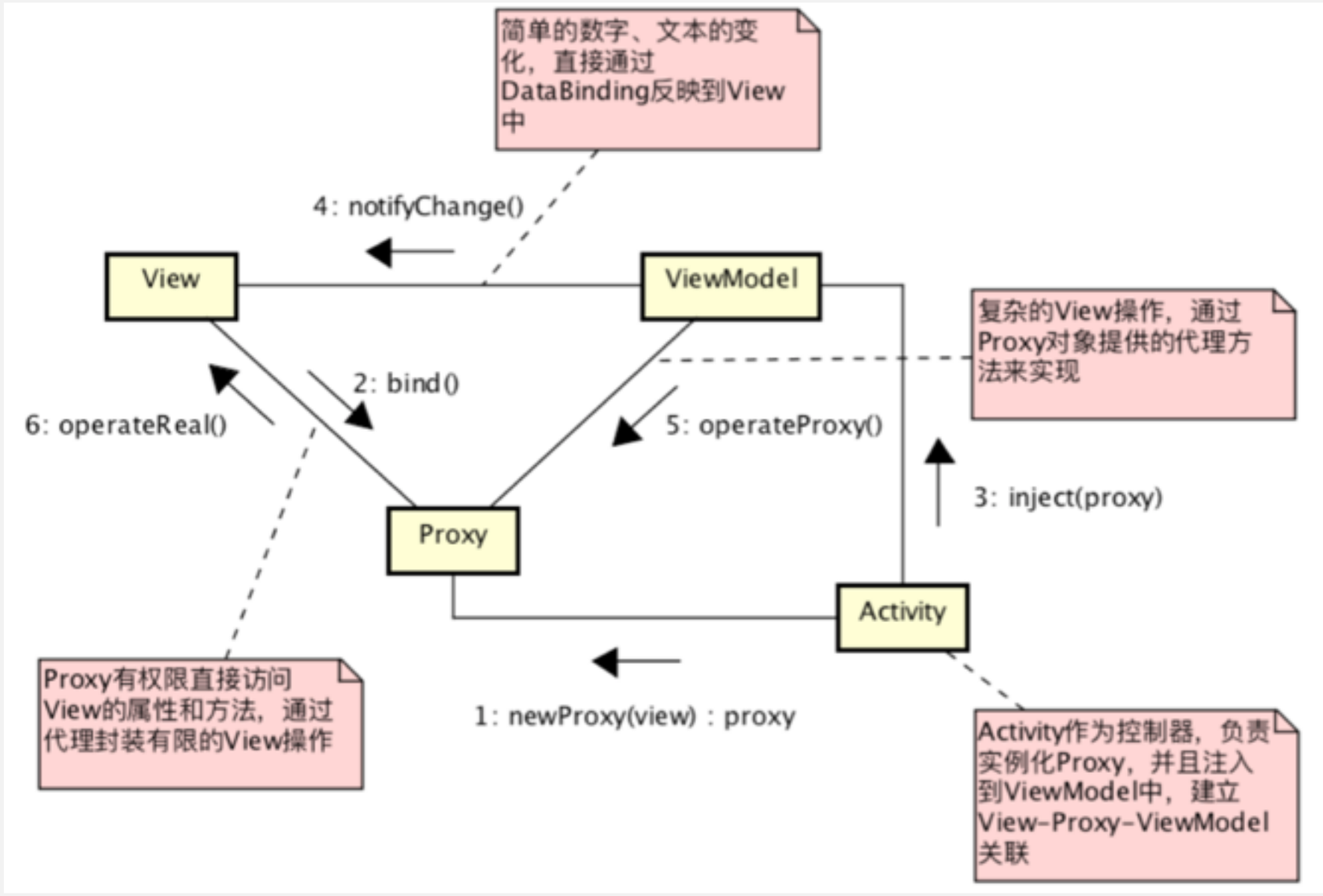
Model的数据对象对应业务的流转及持久化

13 VM生命周期


Activity launched
onCreate()
onStart()
onRestart()
onResume()
Activity running
User navigates to the activity
App process killed
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down


Constructor
parent calls addView()
onAttachedToWindow()
measure()
onMeasure()
layout()
onLayout()
requestLayout()
dispatchDraw()
invalidate()
draw()
onDraw()
<User input />

14 Databinding Proxy



15 MVVM优缺点

优点

1. UI和业务的彻底分离
2. 将UI状态的切换明确定义
3. 比较易于单元测试
4. 跨端一致性设计

缺点

1. 不适合小的App
2. 开发设计要求较高
3. 不适弱交互的App
4. 需要底层框架支撑

Q&A

联系方式

邮箱: jackyliusir@gmail.com

微信/手机: 18611999430