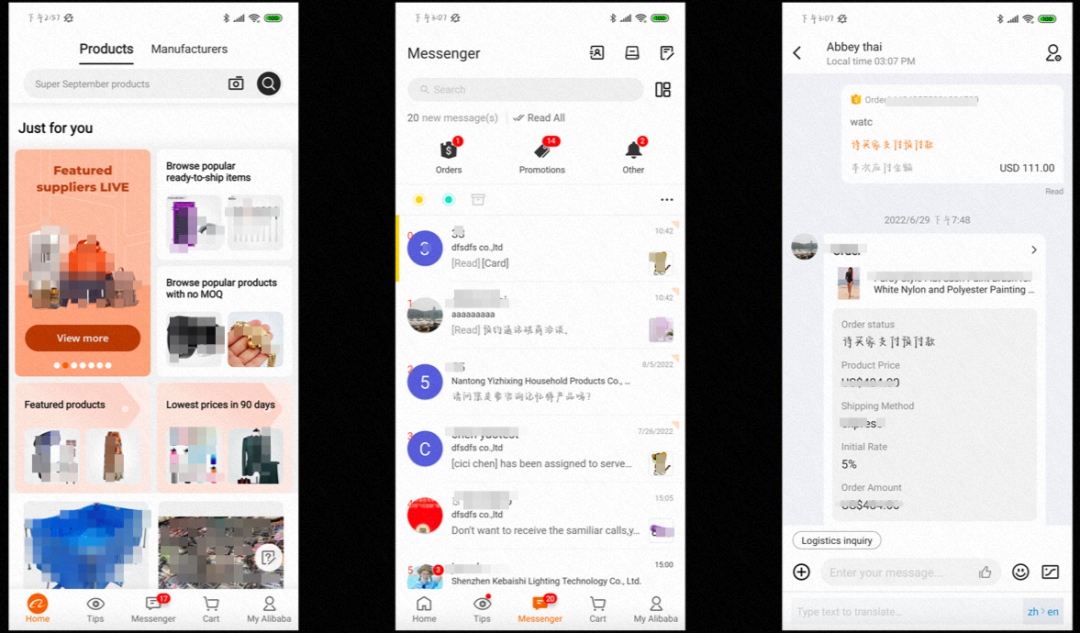
# 研发视角：一个需求应该怎么拆解与实现？

研发过程中，开发同学在接到一个需求后，必须要回答两个问题：做什么（WHAT）、怎么做（HOW）。本文就开发与测试在拆解需求时面临的共性问题，结合自己过往的经验，总结的一个实用的方法。本文不讨论技术选型，仅从思考逻辑上总结应该如何拆解与实现一个给定的需求。欢迎讨论。

理解需求拆解的关注点

以带UI的需求为示例，来看拆解需求过程中的关注点。看下图，停留20秒，思考两个问题：（1）从无到有实现以下需求对应的功能页面，需要做什么？（2）以下页面中只修改指定区域的元素，又需要做什么？﻿﻿﻿上文两个问题代表两种需求类型：第一个问题对应新功能的实现，第二个问题对应已有功能的修改与维护。无论是新功能，还是历史功能，都需要关注如下点：

* 需要多少UI（UI组件、功能页面）（UI=User Interface）
* 数据从哪里来？
* 数据与UI如何关联？（数据到哪里去、怎么去）
* 用户行为响应
* 用户行为采集
* 发布后，如何运维与监控（出现问题的时候，可以有哪些方法帮助问题的排查）

﻿将上文提及的关注点进行抽象与分层后，会得到如下一张图：



﻿﻿抽象后的架构层次图可以帮助我们理解实现的层次划分与结构设计。但是，一般来说，抽象意味着抹平细节。对于一个新手开发或者新手测试来说，需求实现或者测试过程，不仅要关注抽象层面的架构设计，更需要关注实现过程中的细节。接下来对于上文提及到的关注点，下文会展开说明。（在一个多角色的团队中，实现一个需求，不仅有自己，还有其他角色的开发、测试、PD、运营、数据等，理解关注细节才能保障交付质量）。﻿

通过关注点看实现需求要做些什么

**关注点1：需要多少UI**

要绘制多少UI组件、要增加多少UI页面，首先取决于“需求文档（PRD）”、“视觉交互设计稿”。对于开发来说，请重视需求评审、视觉交互评审。对于质量良好的需求文档或者视觉设计稿，文档中给出的UI组件/页面范围一般就是开发要实现的UI范围；对于质量差的需求甚至是一句话的需求，UI范围则需要研发“梳理确定”。无论需求文档或视觉设计稿质量如何，研发人员在需求评审、视觉交互评审、技术设计、编码实现过程中都需重点关注“异常逻辑”。什么是异常逻辑？比如，一个查询类的需求，“查询结果为空”“查询时无网络”这类场景就属于“异常逻辑”。很多时候，新手容易出问题或者遗漏的地方就是异常逻辑的处理与显示。

﻿需求拆解阶段，关注UI如何实现的同时，也请务必关注UI的展示对象，包括：图片与多媒体内容（CDN图片、OSS图片文件、base64编码后的图片）、文案（文案的长度、多语言翻译的来源）、RTL（Right to Left）适配。已一个支持17种语言甚至是更多语言的海外APP开发为示例，这些内容虽不影响开发进度，但是会影响需求在交付时的质量、或者是到了需求实现后期补作业。﻿实操方法：需求拆解的时，对于UI范围，可以拆解出如下内容用于辅助后续研发实现：

* UI范围：功能页面范围、页面状态及状态的变化逻辑、界面的异常逻辑态等等
* 页面元素的拆解：列表元素、卡片元素、动画元素、UI特效等（这个也是沉淀通用组件的依据）
* UI展示的对象：文案的范围、多语言翻译的来源、RTL适配的范围、图片、短视频等。

**关注点2：数据从哪里来？**

这里的数据是包括用户直接感知的数据与富媒体内容流，也包括用户不感知但是用于支撑功能的研发类型的数据。数据的来源包括：网络数据、本地存储的数据、内存数据。不同的数据类型，需求实现的处理有差异，例如：

* 网络数据：需求拆解时候重点关注接口文档、接口名称、接口返回的数据字段（包括字段类型、范围、字段名称等等）、接口对应的团队或具体的开发、接口的上下游链路（示例：搜索的接口会涉及广告、算法等）、接口测试与联调方式等。对于新手，需求拆解的时候还需要确认：数据网关类型（MTOP网关、Node编排后的数据、Web网站对应的数据网关等），明确网络数据调度所用的域名（比如，部分APP交易功能的域名与非交易功能的域名是分开的，调度方式也会有差异）。
* 本地存储数据：需求拆解阶段，要明确本地存储数据还是自己负责还是使用他人的能力。如果是自己负责则需要进一步判断数据是写本地数据库，还是写本地文件；如果是他人负责，数据的读写方式与接口是什么。
* 内存数据：需求拆解重点关注内存数据的读写方式，数据使用后的释放方式。
* 富媒体内容流：图片、短视频、直播等多媒体流。需求拆解时关注这些内容存储的形式（以图片为示例：图片是CDN的URL，还是OSS的文件地址或者ID，甚至是base64处理后的编码结果）；同时还需要判断这些展示能力是已有的还是需要新引入等。

﻿总结来说，“数据从哪里来”很好记忆，但是，在真实的研发过程中，“数据从哪里来”往往是研发过程中最容易出问题的地方。以网络数据为例，典型的问题有：联调环境不稳定、接口返回的字段未遵守约定、接口上下游数据链路不通导致无法测试等。即使需求完成并且交付上线，也还是会有各类问题，比如：引入了新CDN导致现有的图片/短视频加载过程的优化策略不支持，导致线上反馈性能或者是稳定性问题等。﻿对于“数据从哪里来”，在拆解需求的时候，一定要关注到每一个字段。这么强调，既因为这个点既影响实际投入工作量，还特别影响交付的需求的可用性与质量。**﻿实操方法：**进行需求拆解的时候，围绕数据从哪里来，可以拆解出以下内容：

* 数据逻辑的分层结构图（根据实际需要）
* 功能与数据接口的对照关系、接口文档、接口的承接平台、接口的Mock方式等
* 功能的数据流转与逻辑流程图

**关注点3：数据与UI如何关联？**

“数据要到哪里去，怎么去”，也可以理解为数据绑定。对于有一定复杂度的需求，“数据与UI如何关联”很大程度上会决定这个功能的实现与维护难易。需求拆解的时候，无论是选择MVC、MVP、还是MVVM，数据绑定这个过程比较考验开发人员的设计能力。我个人将数据绑定总结为如下几种：

* 单向数据绑定：数据变化自动驱动UI刷新，代码实现会体现为各种Observer或者Observable等。
* 双向数据绑定：数据改变的同时使视图刷新，而视图改变也可以同时改变数据。
* 无所谓数据绑定，数据关联到UI是过程式的编程，体现为：加载数据、加载成功手工代码上屏等。

﻿当然，以上总结是简化后的总结。对于新功能，建议需求拆解的时候，反复理解所需的数据源、充分理解UI范围，结合控制逻辑，仔细提取关键特征后，充分设计后再进行编码等操作。设计的时候多借鉴团队内或行业内的最佳实践。﻿对于维护类型的需求，特别是有浓重祖传特质的功能（比如2013年功能迭代至2022年的需求），则要花点时间理解原有的关联关系，这个很有可能是一个不大不小的难点。比如，有的历史功能在实现的时候，数据与UI的关系是通过EventBus/消息广播类的事件触发；还有的功能数据在独立进程加载，加载完成后通过进程间通信再通过事件通知机制绑定数据到UI。总结来说，部分历史功能在最开始的实现，会用到各种酷爽的方案，但是到了后期维护，这类酷爽方法则会以一种超长技术链路或者拗口的技术链路呈现给维护者，变成一座需要咬牙才能翻阅的“山”。本质上来说，编码不是人与机器的交流，而是人与人之间通过写作的交流。对于一个维护性质的功能，掌握原有关系的一种方式就是植入各种测试性的代码辅助自己理解。**﻿实操方法：**拆解时，数据与UI关联，可拆解出的内容（数据与UI的关联，建议将“克制”作为自己的原则）：

* 功能的数据流转与逻辑流程图
* 代码设计、数据关系流转设计等等
* 数据状态变化、UI刷新时机等
* 关联所用的绑定能力、数据变化后的通知机制等

**关注点4：用户行为响应**

相对于上文提及的三个点，这一点相对容易。对于带UI的需求，按照如下范围拆解与实现即可：

* 归纳需求文档或设计文档中的用户行动点，包括：点击、上下滑动、左右滑动、长按、开锁屏、虚实键盘响应、缩放手势等等。
* 行动点的常见处理：页面跳转、tips/toast/弹窗等展示、动画处理、界面缩放、界面关闭等
* 需约定的接口或规范：页面跳转的schema/传参、二三方能力唤起的方式等﻿

**关注点5：用户行为采集**

也就是收集用户行为的数据，简称“数据埋点”。比如，用户点击按钮、进入页面、在某区域停留一定时长之类的行为数据。这些行为数据是产品、数据、BI等角色关心并用于分析用户特征的数据。用户行为采集与用户行为响应密切相关。数据埋点作为独立的关注点强调，是因为即使研发在每个需求都会与“页面点”“点击点”“曝光点”“自定义事件点”打交道，同时也是最容易出问题的地方，问题示例：：

* 数据漏采集：有可能是产品/数据同学无明确的数据要求，也有可能是开发漏写数据埋点代码等。
* 数据点错误：包括事件类型错误、数据点名称错误，采集时机错误、多采集、少采集、数据参数错误（数据参数错误包括字段名称不对、字段实际传递的值不对）等等。

﻿**实操方法：**研发人员在拆解需求的时候，针对用户行为采集，check以下点：

* 需求是否需要采集埋点数据
* 点的名称是否符合产品/数据等消费方的要求
* 参数范围与参数值是否有要求
* 需求完成后，数据采集谁验证？

**关注点6：发布后，如何运维与监控**

线上一旦出问题，开发或测试可以用什么工具或者方法进行排查分析。这就涉及到：需求如何发布、交付后的功能如何监控、开发过程中是否要提前布点以支撑排查分析工具的使用。﻿功能发布形式有多种，例如：灰度、Beta、A/B测试形式发布、动态配置下发、直接全量等。不同发布形式对于功能发布之初的观测有差异，要支撑不同的发布形式，具体的实现也有差异：比如，通过Google Play进行APP灰度测试不需要开发人员针对功能做额外处理；以A/B测试形式上线的功能则需要在编码之初就要考虑通过什么平台能力进行，同时还需要编写对应代码。﻿关于如何监控：需要区分是技术指标监控，还是业务类指标的监控。不同的监控范围，在需求拆解的时候就要决定哪个平台进行，比如：业务类的指标可以在xflush平台上进行（需求拆解时要考虑数据回流xflush是已有的能力，还是需要新建能力）；性能类指标可以在魔兔/iTrace上观测；研发问题的排查可以通过SLS进行，还可以自建排查能力。监控虽可以跟随功能的交付逐步补充完善，考虑需求的完整性，建议是在需求拆解的时候明确监控范围与形式。毕竟，实现监控也是需要工作量的。﻿上线后有哪些工具可以用于排查分析问题，在需求实现的时候就需要将能力预置好的。例如，对于“用户下单”这类容错较低的需求，研发在需求实现过程中如果没有写入足够的日志，一旦线上用户反馈“同一个产品在同一个时间下两个订单”，大概率这个问题就是无头无解问题。﻿总结来说：需求发布到线上后如何运维与监控，研发人员在拆解需求的时候需要思考明白：需求发布形式、上线后期望的监控方式、出问题时可用的排查方式与与排查数据。这些明确后，具体的实现，很容易通过历史功能、咨询、查资料等方式学习到。﻿

总结

无论是开发还是测试，对于新手来说，都可以尝试基于“需要多少UI”、“数据从哪里来”、“数据与UI如何关联”、“用户行为响应”、“用户行为采集”、“发布后，如何运维与监控”六个方面进行需求拆解，并且根据拆解后的内容进行需求实现或者是需求测试。这个方法也适用于去分析其他功能的实现。这个方法是基于有UI的需求进行的总结，还有更多的需求是无UI的：比如，从某个位置同步数据到指定位置并且提供给到其他场景使用；再比如，研发需要一套新的路由框架或需要一套新的资源调度框架。这类需求的拆解与本文总结的方法有相同点，也有不同点，避免本文冗长，不铺开陈述。以上方法，是我个人在工作过程中，在接手需求的时候，用来判断需求影响范围、评估合作方、评估工作量的方法。欢迎实践、欢迎讨论，欢迎补充与更正。