模拟项目概要设计V2-郭韬

1. 需求描述

针对开发过程中有时缺少真实硬件外设测试的场景，开发一款测试APP，基于ios蓝牙外设开发模式，可任意选取设备类型并进行模拟，可以发出蓝牙广播并和vesync配网成功，针对用蓝牙进行数据交互的外设类型（如体脂秤），需要完整模拟其和vesync的数据交互，如上报当前称重数据和vesync同步设置时间到体脂秤等。

2. 整体架构

项目整体采用MVVM架构，其中底层为通用的蓝牙工具类，只负责发送蓝牙广播和建立蓝牙连接，此外还需要控制数据收发的解析类，用于解析vesync端下发数据和拼装本地上发数据包，再往上到model层，根据是否有蓝牙数据传输和产品功能分为几个大类，再往上的viewmodel和viewcontroller也根据model大类来分别设计，数据的双向绑定通过RxSwift实现，具体结构如下图所示：

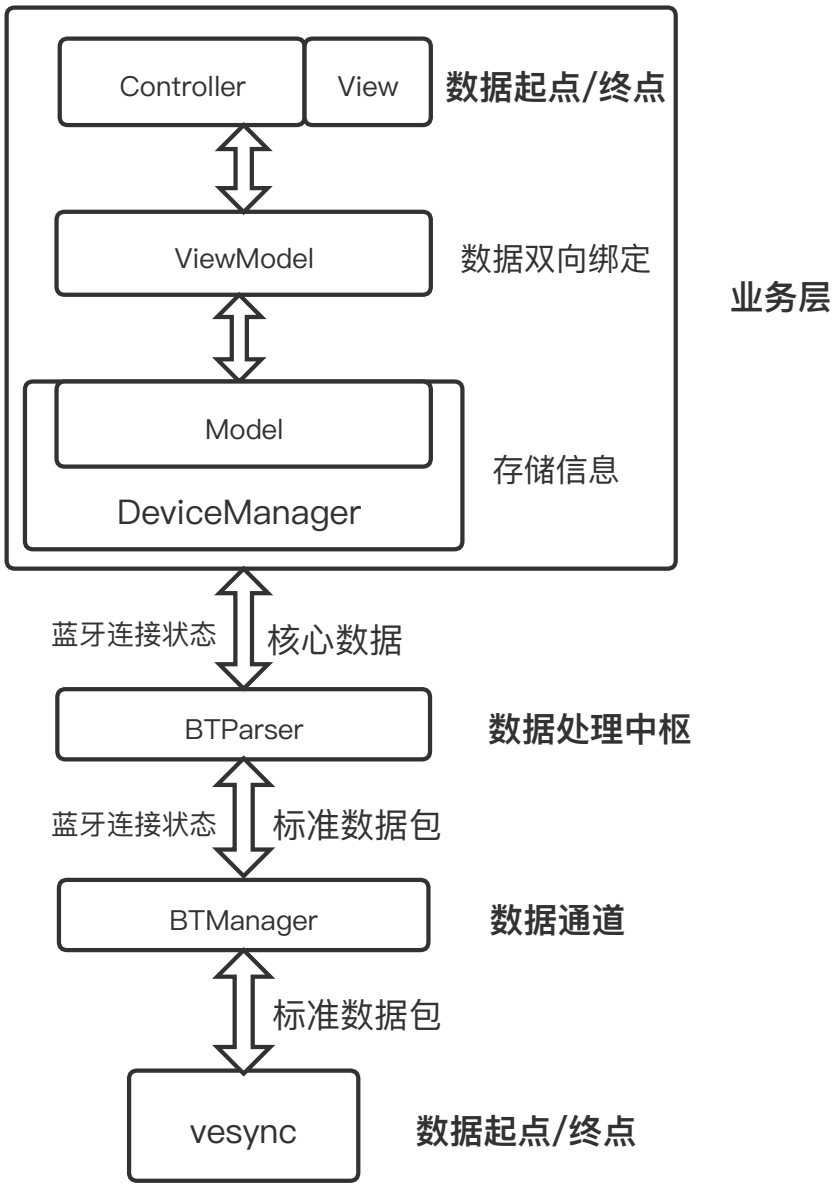


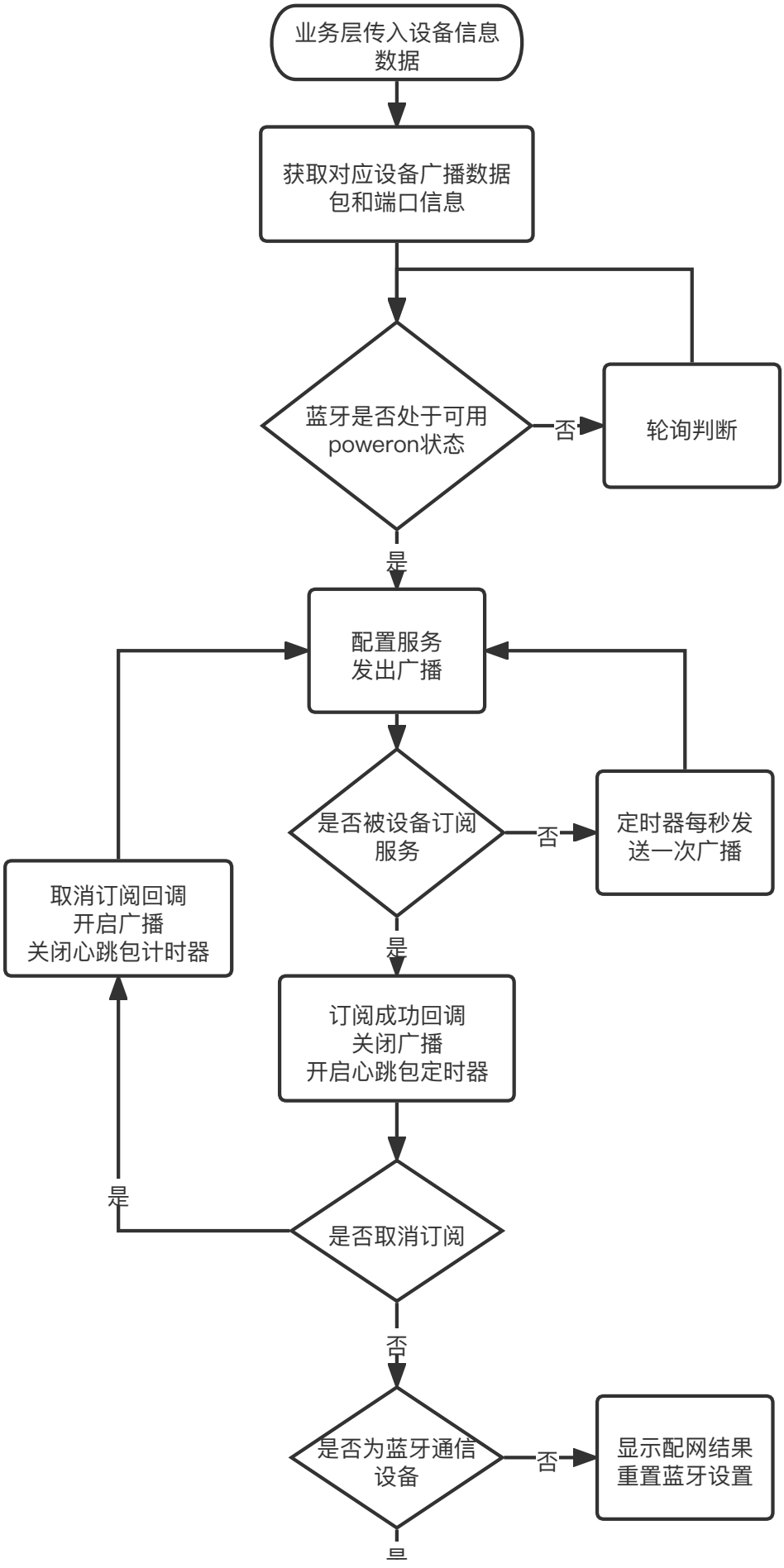
图2.1 整体架构图

3. 模块设计

3.1 蓝牙管理类（BTManager）

蓝牙管理类BTManager是继承自NSObject类的单例，通过在extension中加入CBPeripheralManagerDelegate实现蓝牙代理，蓝牙具体功能通过类型为CBPeripheralManager的成员变量peripheralManager来实现。BTManager在程序打开后全程保持运行，peripheralManager通过peripheralManagerDidUpdateState方法实时检测蓝牙状态，若本机蓝牙关闭则提示用户打开蓝牙，当蓝牙状态为可用即poweron时，开启定时器每秒发送一次蓝牙广播，设备链接订阅成功后回调方法中关闭广播定时器并开启心跳包定

时器，在自定义的服务端口上发空数据保证不断连。具体的蓝牙控制流程如图所示：



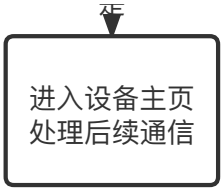


图3.1 蓝牙连接流程图

注意，BTManager本身不存储任何数据，只对蓝牙数据的收发做控制，数据来源是业务层用户的点击操作，包括设备信息和对应蓝牙指令信息，收发信息的解析和封装通过接口来交给BTParser类完成，BTManager只承担vesync端和BTParser间的数据通道功能，订阅成功、取消订阅、接受下发请求时会实时通知BTParser类传往业务层，业务层回复数据、主动上报时将原始数据传送至BTParser类处理，BTParser处理完毕后通知BTManager类发送加工后数据，大致数据交互如下图：

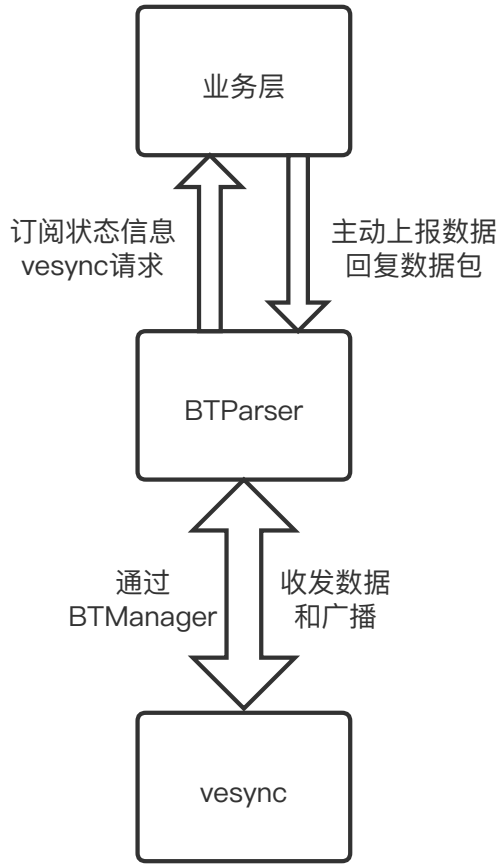


图3.2 主体数据交互

PS：发送广播时的广播包格式为公司id➕设备mac地址➕设备型号，其中公司id固定，设备型号信息从上层model中获取，设备mac地址由于ios无法获取本机mac信息，可以采取随机生成和用户修改相结合的方式，将当前模拟的mac地址显示在主页，正常来说不会重复，不会影响正常的连接和数据收发。

3.2 蓝牙数据解析类（BTParser）

相比于只负责蓝牙连接控制的BTManager，BTParser作为vesync和模拟APP业务层数据交互的中枢，承担了绝大多数的业务逻辑，但依然只存储少量和数据解析封装操作相关的字段信息，数据交互内容主要可以分为4块，蓝牙连接状态信息、设备模型状态信息、vesync请求和业务层封装数据指令，如下图所示：

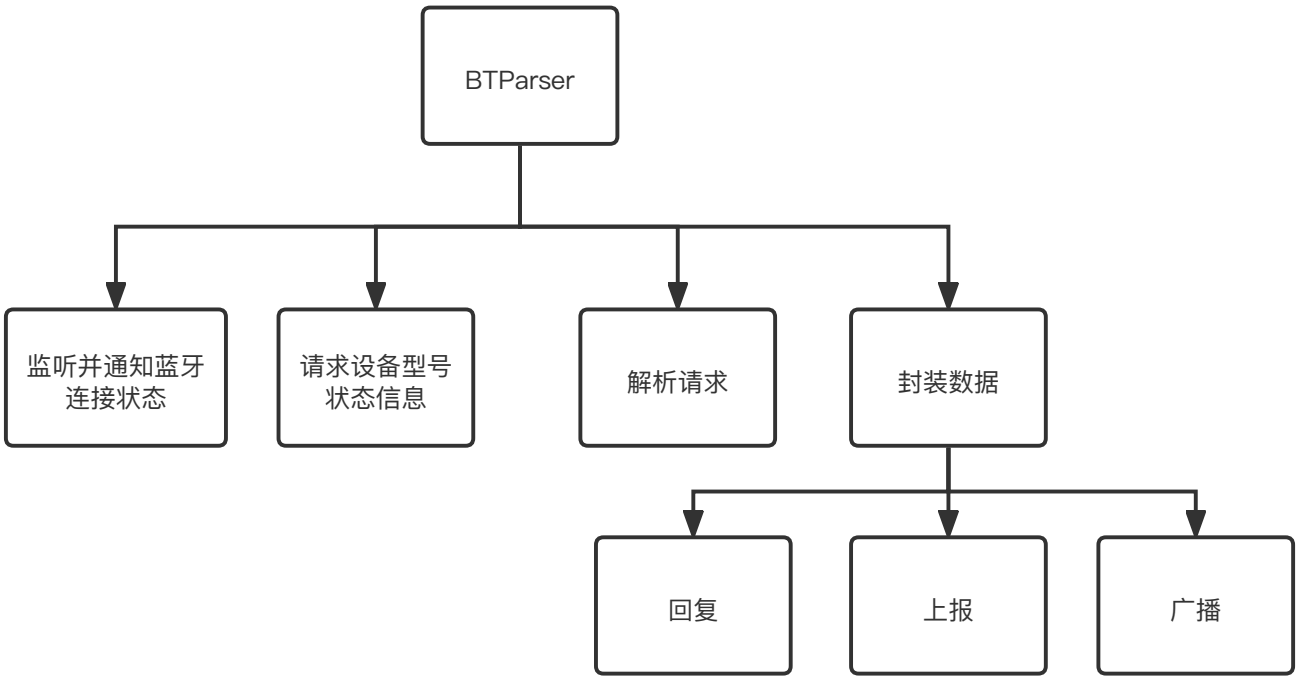


图3.3 BTParser功能结构

其中蓝牙连接状态信息是双向的，自下而上是订阅成功/失败的状态信息，自上而下是蓝牙连接和广播的开关指令；设备模型信息的获取主要来自于另一个单例工具类DeviceManager，在业务层进入设备主页时会将设备型号信息、广播服务信息、蓝牙指令信息和设备数据信息初始化为一个model并保存到DeviceManager中，方便BTParser后续获取设备信息。

对于vesync端的请求，BTParser先是判断数据包长度的合法性和对应指令是否存在，检验无误后按照统一格式进行解析，提取数据包中的重要信息即指令码、数据长度和包尾数据，将重要数据发送至viewModel层，viewModel层根据这些数据对model执行对应操作并将结果通知到viewcontroller中显示给用户。相对的，业务层的原始待封装数据只包括指令码、数据长度和数据包，传达到BTParser后再判断原始数据包长度合法性，检验无误后按照统一格式进行封装，目前的封装数据主要为回复、上报和广播3类，其中回复和上报十分相似，只不过回复要与请求一一对应，并且不用从vc层接受用户选定的指令码和数据。

一般情况下BTParser类不会有改动，除非新接入第三方产品的数据包格式跟vesync统一的数据包格式不同，则需要额外加一套解析规则。

3.3 Model类

Model信息可分为设备型号信息、广播服务信息、蓝牙指令信息和设备数据信息4类，顺序越靠后的数据信息越高级（只有一部分设备会有），最低级的设备型号信息则是每个设备都有，从下到上依次过滤不支持对应信息的设备，如下图所示：

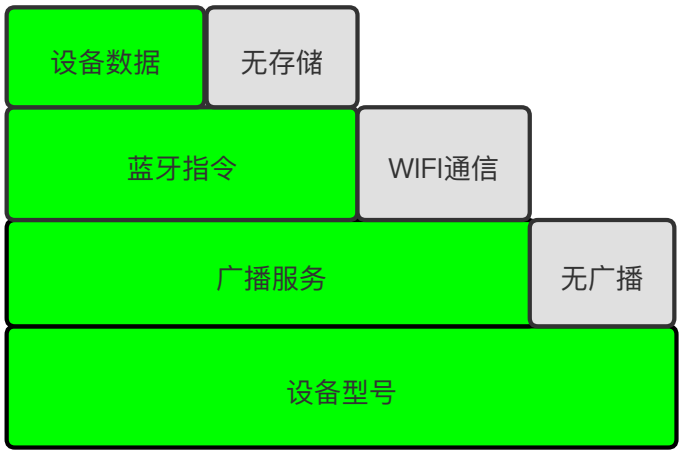


图3.4 Model数据层级

其中底层的设备型号信息在最基础的同时，也是最特别的信息，每款设备的型号信息都不完全一致，其中可以再次提取到设备大类信息和设备小类信息，这里的设备小类为自己定义的概念，指广播服务和蓝牙指令相似的设备可归为一类，共用一套广播服务和蓝牙指令的model，设备小类的设备数据目前也可作为一套共用，但后续的更新升级可能会加入特定设备数据的持久化存储，届时需要用设备的真实设备名作为主键存储对应设备数据，还可以考虑加入账号登陆的机制，让账号id和设备名联合作为主键，存在云端数据库中，登陆模拟APP后发起请求从云端拉取设备数据。

model的蓝牙指令模块可以将每条指令对model的设备数据操作也涵盖进去，简化viewmodel的逻辑，只要默认给蓝牙指令的model在传入指令的同时传入核心数据即可，viewmodel在这里只充当model和vc的数据双向桥梁，负责监听和通知数据的变化。

3.4 ViewController类

界面只涉及选择设备的主页和选择完设备后各设备的主页（配网信息或者指令数据传输），默认蓝牙开启，在部分设备主页中可以添加关闭蓝牙连接的按钮（如体脂秤的本地添加称重数据），只需对BTManager类的成员变量peripheralManager赋值nil和重新初始化即可，各设备的主页尽量用一页展示数据即可。

选择设备的APP主页参照vesync的Add Device页，左侧为产品大类，右侧为具体产品型号和图片，点击即可进入设备主页，界面上部区域用于显示和修改模拟设备的MAC地址，具体如下图所示：

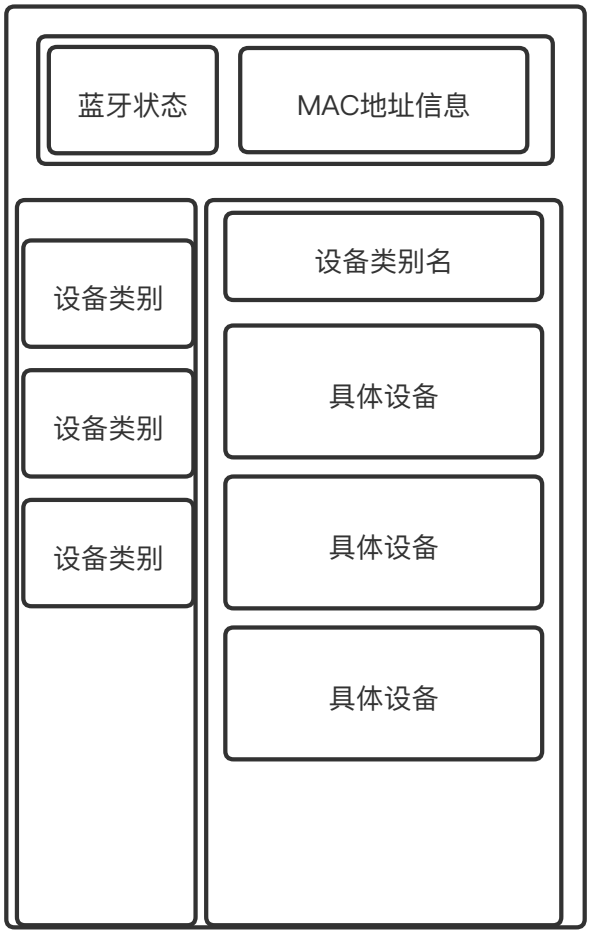


图3.5 APP主页示意图

配网结果提示页统一即可，即便没有提示页vesync端也可以查看到配网结果。设备主页则统一用tableview展示即可，每个cell代表一组设备数据，相似设备的cell可以复用，简化开发。

3.5 工具类（Tools）

整个类由静态方法组成，包括但不限于数据展示格式的转换、UTC时间戳的转换、随机MAC地址的生成和全机型的UI界面适配，还可以添加错误处理的方法、打印调试信息等。此外还将导入方家浩的日志抓取工具pod，实现脱离xcode真机测试的控制台日志显示，更加便于测试人员测试。