获取Polar数据流3

开发计划

Step 0｜项目与权限“地基”（新增，放最前）

- 任务：Xcode 真机跑通；开启 Background Modes→Uses Bluetooth LE accessories；Info 里加入蓝牙权限说明键；确认 UDP 心跳能发到 Mac；LabRecorder 能看到 `PB_UDP` 与 `PB_MARKERS`。
- 验收：在真机上看到“每秒发送/标记”可用；`qa_check.py` 对一段 10–20 秒录制给出 PASS。

**Step 1｜建立整体 UI 框架

阶段性交付物与验收标准

交付物

1. **信息架构图**（简单即可）：页面区块、导航方式、各区数据来源。
2. **状态模型草图**：`AppState / SessionState / DeviceState` 字段清单与状态转换表。
3. **事件与标记词典 v1.1**：常量表与说明。
4. **UI 原型**（可用 SwiftUI 静态界面即可）：
   - 顶部计时区：总计时/诱导/干预三块数字时钟
   - 中部设备区：Verity/H10 两卡片，显示“已发现/未发现/已连接/未连接”的占位状态
   - 控制区：开始/标记组（基线/诱导/干预）/结束
   - 底部调试区：IP、端口、发送开关、丢包计数、最近心跳速率、日志窗口
5. **设置与持久化**：IP/端口/默认设备的保存与还原。
6. **一键自检**：按钮可发送测试 UDP/Marker，并在调试区显示结果。
7. **会话摘要落盘**：结束后在沙盒写入一条 JSONL 摘要。

验收标准

- 启动应用后，在**无设备**情况下仍可完整浏览 UI，各区展示“占位状态”，无崩溃。
- 配置项（IP/端口/默认设备）修改后，**杀掉应用重启仍能恢复**。
- 点击“开始”后，总计时走秒；点击“诱导开始/结束”“干预开始/结束”能独立启动/停止对应计时，而总计时不中断；点击“结束”三者全部停止。
- 开启“发送数据”后，**不依赖任何真实设备**，能向 UDP 端口发送测试心跳；LabRecorder 开启时，可看到 `PB_UDP_TEST` 与 `PB_MARKERS_TEST`。

- 结束后生成 `session_*.jsonl`，字段完整。
- 运行 `qa_check.py` 对应的 XDF 文件，若事件齐全且时长达标，则返回 PASS。

子任务顺序

UI框架搭建可分为三个层次。第一个层次是底层机制，包括App运行的全局信息、数据广播/发送/打包/标记等服务。这一层的程序服务所有的App页面。第二层是UI框架，包括信息框架、操作流程、功能模块和交互方法。第三层是数据层，建立数据模型，并在各页面之间通信。它们被标注为三个子任务。

第一层：底层机制（与 UI 脱钩）

手机端

- **ContentView.swift**
  这是界面入口
- **HomeView.swift**
  app主界面，展示所有主要功能模块，以及各功能的入口
- **CollectView.swift**
  信号采集和生成页面
- **AppStore.swift**
  一个全局的小本子，用来记“当前目标地址、端口、是否在连续发送、累计条数、最近一次发送时间”等状态。按钮改变状态时会在这里记一笔，也打印日志，方便确认流程。
- **UdpSender.swift** (已经启用，传送服务并入UDPSenderService) “数据发送器”。专门把 CollectView 里产生的“心跳”这类普通数据，用 UDP 方式发到指定主机和端口。
  UDP 是一种“扔纸飞机”的网络发送方式：不确认对方收没收，但速度快、开销小，适合我们这类高频、容错的实时数据。
- **MarkerBus.swift**
  “标记总线”。是一个很简单的广播器：界面上的标记按钮只需要说“我这儿有个标记事件，标签叫 baseline_start（或 stim_start 等）”，广播出去，不管谁来接。
- **UdpMarkerBridge.swift**
  “标记桥”。它订阅“标记总线”的广播，一旦听到某个标记事件，就把它打包成一条更完整的 JSON：
  - `label`：标记名
  - `packet_id`：递增的编号，方便日后排查是否丢了某一条
  - `t_device`：手机本地时间戳（用于跨设备对时对齐）
- **UDPSenderService.swift**
  “标记发送服务”。和数据发送器分开，单独维护一条 UDP 连接，专门负责发**标记**。它会打

印：目标地址、连接状态（ready 就绪）、每次发包的字节数、是否成功等。
这样“标记”和“数据”互不影响，也便于分别定位问题。

> 小结： 普通“心跳/数据”走 `CollectView → UDPSenderService` 。 “标记”走 `CollectView → MarkerBus → UdpMarkerBridge → UDPSenderService` 。
> 两路都发到同一个目标 IP:Port（在“应用设置”时会同时更新两路的目标）。

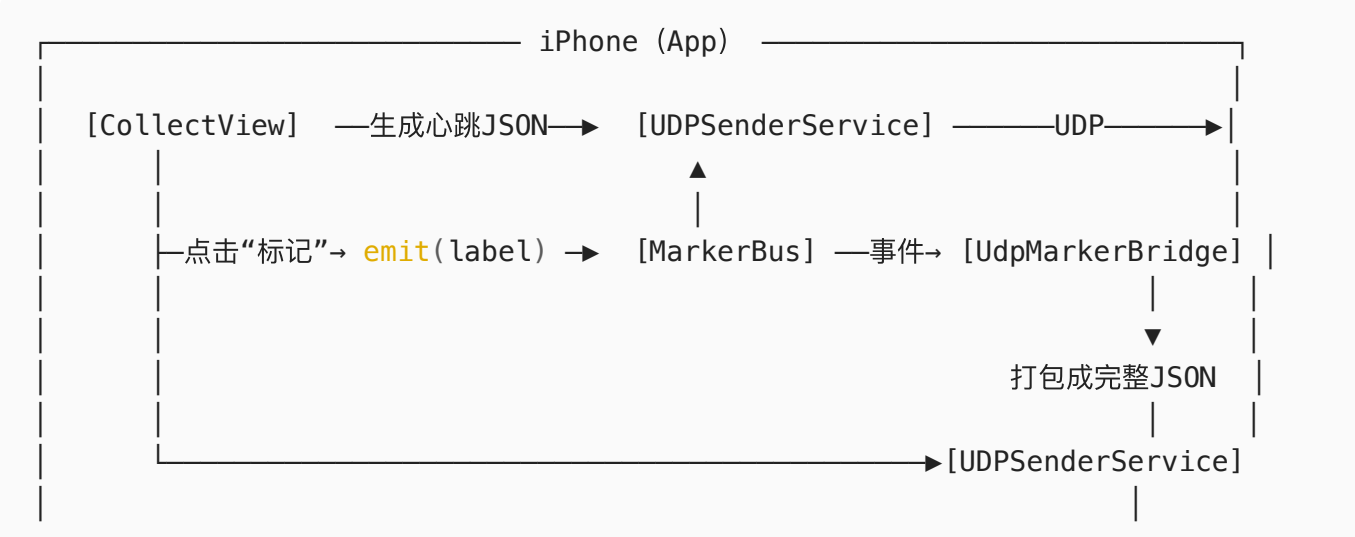
流程图

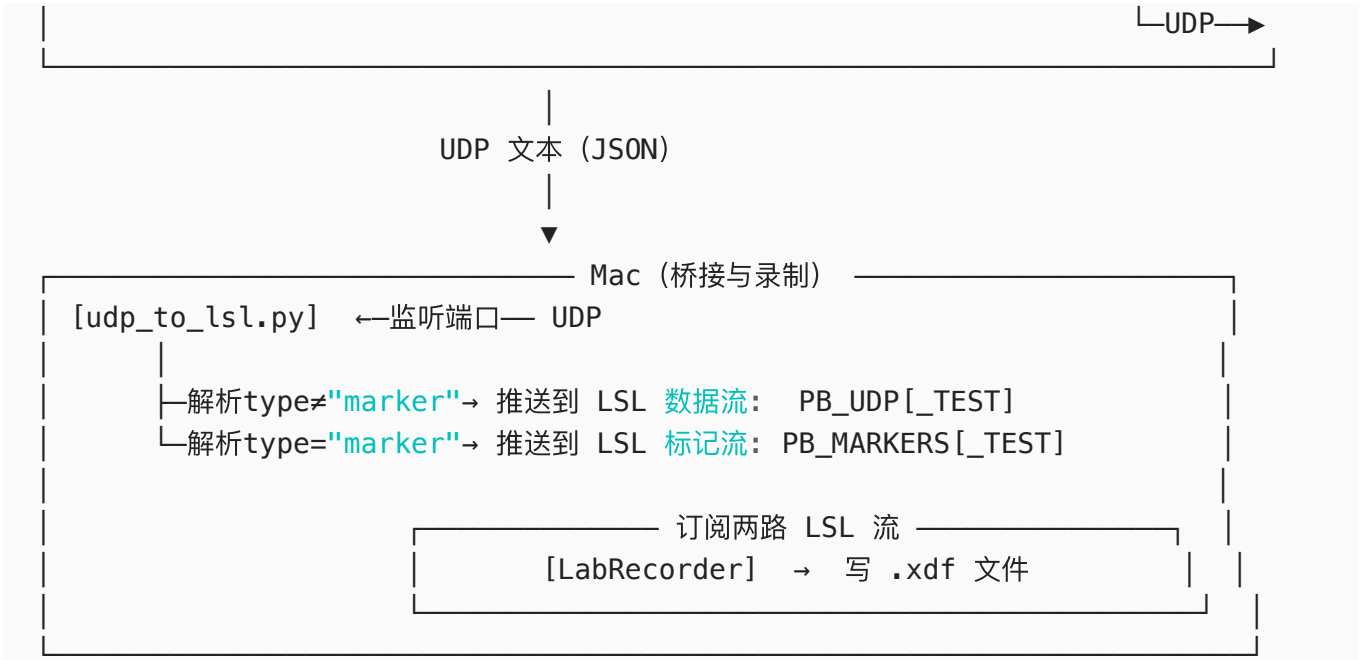
CollectView 是用户采集数据的交互界面。AppStore在背后进行总体调度，获悉用户采集行为后，通过 `startCollect()` 、 `markStarted()` 、 `emitMarker()` 等行为向系统所有服务/功能模块发布命令或信号，开启全局的采集/传输任务流程。同时AppStore还负责定义任务条件（如指定发送地址--目标UPD的IP地址）、记录采集任务流程中的关键事件，例如采集了几次 `collectCount` 、标注到了第几步 `markerStep` 等。

采集工作包含两个不同性质的数据采集流程。第一是采集生理数据本身，即“数据流”。第二是研究者根据实验阶段对数据的标记，如基线、诱导和干预的时间点，用于后续分析中的”锁时“，这个流程定义为“控制流”。两个采集都会执行向UDP发送数据并在UDP所在的局域网进行广播，并由LSL接受广播并记录广播内容（即数据）的流程。只不过“数据流”的广播内容是生理信号，“控制流”的广播内容是时间标记。本阶段的任务是先忽略具体的广播内容，先打通广播的通道。

打通通道的流程包含UDP与LSL两个基本步骤。本App应用负责第一个步骤。其原理是App主动地找到需要数据的设备所载的位置，也就是该设备的网络地址，即UDP目标，然后向该地址发送数据。目前这个UDP地址需要用户自己查阅目标设备 -- 也就是负责运行LSL的电脑 -- 在网络中的IP地址，然后手动输入到App中。UDP需要一系列的任务环节，包括定位UDP发送地址、连接目标地址、向地址发送信号等工作，这些工作都由UDPSenderService（UDPSS) 执行。

以生理数据采集为例，用户在 `CollectView` 点击采集数据，系统把这个事件告诉 AppStore。AppStore 通过 `stopCollect()` 来指挥后续一系列的工作。 AppStore 通知 `PolarManager` 开始工作（通过 `start(..` 来执行）。PolarManager 提取到生理信号，打包发出(`sendPacket()`)。接下来的事就是电脑端的工作了。





之后（离线）：
.xdf → [check_xdf.py / qa_check.py] → 打印样本数、时长、标签、PASS/NOT PASS

连线含义说明：

- 直线箭头 `→` 表示“把数据递交给谁”或“谁调用谁”。
  - 在 iPhone 内部，比如 `ContentView → MarkerBus` 是“按钮告诉总线：有个标记”；`UdpMarkerBridge → UDPSenderService` 是“桥把整理好的标记交给发送服务”。
- 带 “UDP” 的连线表示“用 UDP 把一段文本发到网络上的某台机器某个端口”。
- 从 `udp_to_lsl.py` 指向 “LSL 流”的描述，表示“脚本把收到的东西发布到 LSL 网络”，供 LabRecorder 订阅。
- 最后一段 “.xdf → 检查脚本” 是文件输入输出的关系。

Console信息解读

- `APPLY host=… port=…` 、 `[AppStore] target -> …`
  来自 HomeView 与 AppStore：说明你点了“应用设置”，目标地址被更新了。
- `TICK {"type":"heartbeat", …}` 、 `SEND-ONCE …`
  来自 CollectView：说明心跳/单次发送确实在产生数据。
- `[MarkerBus] emit -> baseline_start #N`
  来自 MarkerBus：说明按钮点击已广播出一个标记事件，第 N 次。
- `[UdpMarkerBridge] sent -> {...}`
  来自 UdpMarkerBridge：桥收到了标记事件，已经把它打包成完整 JSON 并交给发送服务。
- `[UDPSenderService] state=ready …` 、 `send to … bytes=… payload=…` 、 `send ok`
  来自标记发送服务：UDP 连接已就绪，刚刚发了多少字节，是否成功。

- `[udp_to_lsl] listening on …`、`[DATA #k] …`、`[MARK #k] …`、`[SUMMARY] data=…, markers=…`
  来自电脑端桥 `udp_to_lsl.py`： 正在监听；已经收到第 k 条数据/标记；每隔几秒打印一次累计统计。
- `PB_UDP[_TEST] / PB_MARKERS[_TEST]`
  你在 LabRecorder 里看到的流名； `check_xdf.py` 里也会打印对应的样本数与时长。

> 第一层所有任务已经完成

第二层：UI 框架


20:43
生理信号记录
设备
Polar Verity Sense
未连接
Polar H10
未连接
选择任务
受试者信息
记录被测基本信息
生理数据采集
采集实验数据
模拟数据测试
基于模拟数据调试设备
历史记录
记录历史
查看历史记录信息
当前 UDP 目标：192.168.31.22:9,001


20:44
Back
生理数据采集
采集状态
0次
试次数
0号
被试 ID
未开始
采集状态
0个
标记数
选择数据
未检测到可订阅数据
请先在首页连接 Verity / H10 等设备
采集操作
开始采集
持续时间 00:00
00:00
基线
00:00
诱导
00:00
干预
基线
开始
诱导
开始
诱导
结束
干预
开始
干预
结束
采集进度
说明：此区显示最小化的实时进度/统计；根据实
验需求继续开发。

T0｜项目结构整理

- 目标：把界面按“首页 / 采集 / 调试”拆成独立 View 文件，后续便于迭代。
- 要做：
  - 规划文件夹：`UI/Home`，`UI/CollectData`，UI/Debug，UI/Shared`.
  - 视图占位：`HomeView`，CollectView，DebugView，以及共享组件占位文件（空壳即可）：DeviceCard，DataPill，TimerPanel，MarkerBar，StatusBanner`. 等等

T1｜导航骨架与入口（按钮导航）

- 目标：建立首页导航布局，首页的选择任务栏中进入各主要页面的按钮。
- 要做：
  - 每个页面顶部保留导航栏标题（`navigationTitle`），右上角预留齿轮/帮助图标位（先空）。

T2｜首页信息架构（设备区 + 任务区 + 最近数据占位）

- 目标：主页层次与草图一致。
- 要做：
  - 顶部“设备状态区”：两张 `DeviceCard`（Verity/H10），显示多个连接状态（如：未发现 / 可发现未连 / 已连接等等）。
  - 中部“选择任务”：三项卡片（受试者信息 / 生理数据采集 / 模拟数据调试），点击“生理数据采集”跳到 CollectView`。
  - 底部“最近数据”：一张占位卡（显示最近一次录制的文件名与时间，先用假数据）。

T3｜设备卡（DeviceCard）状态语义与可达性

- 目标：设备卡状态清晰，不只靠颜色。
- 要做：
  - `DeviceCard` 支持状态：`not_found / discovered / connecting / connected / failed / permission_missing`。
  - 提供状态点 + 次行文案（例如“已连接 / 可发现 / 权限未开”），颜色仅作辅助。
  - 点击已连接的卡片，应在采集页的“可采集数据区”点亮相应数据源（后续再真接 Polar）。
- 验收：手动切换 `AppStore` 的设备状态（在debugView模拟连接状态）后，卡片根据模拟指令显示与设备的连接状态。

T4｜采集页骨架（状态 → 数据选择 → 操作 → 计时 → 进度）

- 目标：落地草图的整体框架。
- 任务概要：
  - 顶部“采集状态”区：用于显示采集编号（或者叫trialID）、被测ID。开始采集后，UDP发送状态、计数等等。用一排横向排列可滑动的图标+文字的样式显示。
  - “数据选择”区：一排 `DataPill`（PPI / PPG / HR / …），仅显示已经连接设备中所有可以订阅的数据。试验员可从其中选择一个或多个
  - “采集操作”区：一大片卡片。上方为大按钮（开始/停止），与数据采集计时。下方是 `MarkerBar`（基线开始/诱导开始/诱导结束/干预开始/干预结束）。
  - “采集进度区”：基本的数据采集过程显示，如已发送条数、已录时长等，取自 Store 的 counters。
  -
- T4-0｜模型与状态准备（Store 最小字段）

- 在 `AppStore` 增加/确认这些字段（只列名称与含义，不写实现）：
  - `subjectID: String?` 被测者编号（可空，先手输）
  - `trialID: String` 本次采集编号（进入采集页生成一次，如 `TYYYYMMDD-HHMMSS`）
  - `isCollecting: Bool` 是否处于采集中（驱动按钮与标记可用性）
  - `sessionStart: Date?` 采集开始时间（用于总计时）
  - `sentCount: Int` 已发送数据条数（已有）
  - `markerCount: Int` 已发送标记数（可选，方便进度区展示）
  - `availableSignals: Set<SignalKind>` 根据设备状态动态推导（如 Verity: PPI/PPG/HR；H10: ECG/HR）
  - `selectedSignals: Set<SignalKind>` 用户在“数据选择区”的勾选结果
- 事件方法（仅声明目标行为）：
  - `startCollect()`：置位 `isCollecting=true`、记录 `sessionStart`、清理统计、广播“开始”。
  - `stopCollect()`：置位 `isCollecting=false`、广播“停止”。
  - `toggleSelect(_ kind: SignalKind)`：切换某数据类型勾选。
  - `emitMarker(_ label: MarkerLabel)`：通过 `MarkerBus` 发标记，并+1 `markerCount`。
- **验收**
  - 这些字段/方法在 Store 可被 `CollectView` 读写；`isCollecting` 改变会触发 UI 切换。

- T4-1｜采集页布局骨架（不带交互）
  - `CollectView` 分 4 块：
    A. 顶部“采集状态区”
    B. “数据选择区”
    C. “采集操作区”（大按钮 + 计时 + 标记条）
    D. “采集进度区”
  - 先用占位内容撑出结构（固定高度，确保开始/停止切换不抖动）。
  - 验收
    - 四区块纵向排列，上下间距一致；切换明暗模式不破版。
- T4-2｜A 区｜状态条（横向可滑动 chips）
  - 按“图标 + 文字”做成一排可横滑的小标识（chips）：
    - `trialID` 这个标识标识某个被试的某次测试，以 `subjectID-testIndex`（例如 `001-1`）标识。`subjectID` 与 `testIndex` 未来在“受试者信息”页面由实验员手工铁血
    - `subjectID`（未填显示“未设置”）
    - `已发送条数`
    - `标记条数`（可选）

    - 数据由 `AppStore` 提供。
    - 验收
        - 横向可滑动；文字超长有省略号；进入采集后条数会动态更新。
- T4-3｜B 区｜数据选择（DataPill 动态生成）
    - 依据 `availableSignals` 生成 Pills：PPI / PPG / HR / ECG …
    - 支持选中/取消（点击变样式），结果存入 `selectedSignals`。
    - 若当前无可用设备：整块置灰 + 提示“未检测到可订阅数据”。
    - 验收
        - 连接状态切换会刷新可见的 Pills；选中状态持久显示；再次进入页面仍保持。
- T4-4｜C 区｜采集操作卡（大按钮 + 计时）
    - 卡片上半部分：一个的大按钮，包含按钮文字（开始/停止）与计时区，布局在T4-1已经完成。
        - 大按钮字样：未开始采集时，显示“开始采集”；点击开始后则显示“停止采集”（红色）。
        - 计时区在字体下方，默认只有持续时间。如果实验员打标注，会增加标注持续时间，总共最多4条时间显示：持续时间、基线时间、诱导时间、干预时间。
        - 当增加多条时间标注时，按钮的高度自适应加高。
        - 按钮点击“开始/停止”分别调用 `store.startCollect()` / `store.stopCollect()`。
        - 将测试部分的UDP发送模拟迁移到此按钮，让CollectView 直接获取同一份状态与计数，以后接 Polar 也不需要再改调用处。
    - **禁用规则**：当 `selectedSignals` 为空或无设备时，“开始采集”禁用。
    - 验收
        - 点击开始后计时立即走动；停止后计时冻结；按钮切换无布局跳动。
- T4-5｜C 区｜标记条（MarkerBar）
    - 一行 5 个标记按钮：基线开始 / 诱导开始 / 诱导结束 / 干预开始 / 干预结束。（骨架已经建好）
    - **可用性**：
        - 大按钮“开始采集”启动后才能使用。
        - 每次只能激活一个标记，一个标记激活后，之前激活的标记关闭
        - 标记的点击必须按照基线开始 / 诱导开始 / 诱导结束 / 干预开始 / 干预结束的顺序，不允许用户随意点击。
    - 点击调用 `store.emitMarker(...)`，并在 Console 打印 `MARK …`。
        - 此处会产生线程问题（在 `UdpMarkerBridge.swift` 中会看见此错误）需要把这些需求放在主线程。通过把 `AppStore` 标注为 `@MainActor`，并用 `Task { @MainActor in … }` 明确把**读** `isCollecting`、**调用** `markSent()` 等 UI/状态更新行为放回主线程执行
    - 验收

        - 未开始时按钮置灰不可点；开始后可点；每点一次 `markerCount` +1，Console 有日志。
- T4-6｜D 区｜采集进度
    - 显示实验中可能需要的信息，具体做什么还没想好，先空着
- T4-7 | 主页底部 ｜ 最近数据
    - 考虑修改为历史记录入口，点击后进入记录历史页面（先做占位，不实现）

T5｜小结与回归清单

- 目标：这一层结束的验收标准。
- 验收脚本（人工）：
    1. 打开 App，首页三块布局正确；两张设备卡显示“未发现设备”；点击“生理数据采集”能进入采集页。
    2. 在调试页改 UDP 目标，回首页与采集页显示同步更新。
    3. 采集页点击“开始每秒发送”，总计时开始走、标记按钮解锁；依次点“基线开始→诱导开始→诱导结束→干预开始→干预结束”，按钮互斥与节流生效。
    4. 点“停止”，总计时停止、标记按钮禁用。
    5. LabRecorder 录 10 秒，`qa_check.py` 通过（≥2 个标记、时长足够、数据密度 OK）。
    6. 无权限/无设备时的空态与黄条能正确出现（用假状态触发）。

说明：这些任务与底层/后续的关系

- **这批任务不接 Polar SDK**，仅用已有的 UDP 发送与 Store 状态驱动 UI，保证骨架扎实。
- 完成后，再进入“功能对接层”（把 Verity/H10 的真实状态与数据流接进来）：设备卡读真状态、DataPill 跟随实际能力、开始/停止触发真实订阅、标记广播到两路 LSL。

> 第二层所有任务已经完成

第三层：功能对接（把线插上）

目标：把底层服务注入到 UI，最小闭环通路达成。

- 把 `AppState` 改为**真实**状态（来自 Permission/Device/Session/Timer 各服务的 Combine 流）。
- 首页设备卡显示真实扫描/连接状态；采集页“开始/停止/标记”驱动 Session 与 UDP；计时显示来自 `TimerEngine`。
- 调试页“发送一次/每秒发送”调用真 `UDPSender`。
- 对接验收：
    - 真机操作一遍“开始 → 三个标记 → 停止”，LabRecorder 能录到 PB_UDP 与 PB_MARKERS；`qa_check.py` PASS。

> 第三层所有任务已经完成
>
> Step 1 完成

Step 2｜集成 Polar BLE SDK（iOS）

- 任务：用 Swift Package Manager 引入 `PolarBleSdk`；工程能在真机关联编译；不写业务逻辑，先保证链接没问题。
- 根据官方文档，文件需要 import PolarBleSdk、import RxSwift才行

Step 3｜设备扫描与识别（H10）

- 任务：实现最小扫描，**只在 Xcode 控制台打印**发现的 H10，识别后HomeView的设备卡更新状态，最终可显示“已连接”。
  - PolarManager.swift（简称PM） 扫描(startScan)，连接(connect) polar 设备，探测可订阅数据(describeSettings)，订阅数据(startHr)，用户交互式地选择需要录制的数据(applySelection)，启动数据传输到UDP(start)等功能。App中需要用到polar数据的服务和功能均从PM中获取
    - 注：扫描设备时，如果10秒内仍旧没发现设备，告诉用户排查问题的方法
    - Polar SDK 每次发来的是一“批”样本，HR取最后一个，RR需要迭代样本
    - UI更新需要在主线程操作
  - AppStore.swift 初始化时建立与PM的绑定(bindPolar)，更新设备状态（枚举DeviceStatus），根据用户“建立连接”的操作(tapDeviceCard)通知PM连接设备
  - HomeView.swift 首页启动后(.onAppear)调用PM启动扫描(startScan)
- SDK 搭建好，实现了扫描和订阅数据后，结合之前做的UI框架，在首页的设备卡马上能看到效果。佩戴好H10，显示“可发现”，点击卡片则建立连接。

Step 4｜订阅 数据UI更新

- 任务：根据 deviceId 建立连接；订阅数据。根据设备型号，UI中要提供所有可供录制的数据，例如H10能记录HR, RR, ECG, 以及加速度ACC等信息。这些信息应该都能显示在采集页(CollectView)的“选择数据”卡片中。用户可以自己决定记录哪个数据，从中挑选一个或多个进行采集。
  - UI层面，为了展示所有数据，需要在AppStore中把这些数据列出来(SignalKind)，并显示在UI上(refreshSourcesByConnectedDevices)。为了让用户选择数据，AppStore提供按键行为(toggleSelect)。上述的这些服务都在CollectView.swift中来应用。
  - 结合数据通讯，任务流程为：在 Home 点击 H10 卡片 → `AppStore.tapDeviceCard("h10")` → `PolarManager.connect(...)` → 连接成功 → `startHrStreaming` 开始在本地产生 HR/IBI 数据（日志持续打印）。

- 当用户对数据打标记：采集中点击阶段按钮 → `AppStore.emitMarker(label)` → `MarkerBus` → `UdpMarkerBridge` → `UDPSenderService` UDP → 桌面端 LSL 的 Markers 流。

Step 5 | 订阅 数据

- 任务：开始订阅数据。能够让“生理数据采集（CollectView）”在用户操作下完成端到端采集：Polar SDK → UDP → LSL/XDF。代码层面流程是在AppStore中，isCollecting通知开始后，PolarManager查询用户选择的数据，然后发送UDP。这个流程由CollectView的按键行为控制。
- 订阅数别识别流程为：
  - 在DeviceState.swift中补全所有可订阅的数据 `enum DataSource { ... }`。
  - AppStore 的 `enum SignalKind { ... }`、`availableSignals: Set<SignalKind>` 等信息也要更新。
  - PolarManager 建立新方法 `PolarManager.shared.applySelection(deviceId: <已连接的H10 id>, kinds: selectedSignals)` 对用户选择的数据建立“信号集合”
    - 这个方法中一个关键点是要实现“所见即所得”。也就是：研究者在「选择数据」里勾选了什么，就只订阅并只发送什么；取消勾选就停止相应订阅。
    - 为了能正确识别用户当前选择和取消的数据，需要考虑到用户已经选择了哪些数据（`activeStreams`），新开的订阅需要排除已有的选择(例如，`toStart = kinds - activeStreams`），这是所谓的“集合差分”。
    - SDK 的 HR 流同时携带 HR 与 RR，可以开一次订阅，但在回调里用两个布尔开关 `wantHR/wantRR` 决定是否各自发送，从而既符合“所见即所得”，又避免重复订阅。
    - 在Start中 识别 用户选择(SignalKind)，分别进行发送。此处要注意的是对各项数据的订阅/记录必须符合Polar SDK的API规范。目前的6.5.0版本要求首先进行传感器设置（`PolarSensorSetting`），如采样率、分辨率等，然后才能订阅：`.subscribe(onNext: { [weak self] ecg **in** ...`
    - 对于运动加速度ACC是否拆成三个数据的思考：记录ACC是为了后续处理 HRV/ECG 的运动伪迹。根据这个需求决定不拆，保留一个 `acc` 流，3 轴作为同一流的多通道数据。理由：伪迹识别用到的是三轴的组合特征（模、能量、jerk），保持同一流的多通道在 LSL/下游更容易保证严格对齐。
  - AppStore 从PM处获取deviceId（在 `startCollect()` 中操作），并指挥PM对哪些信号建立集合，实现集中订阅。
  - 用户停止采集后，所有的订阅行为都停止
  - TelemetryModels.swift 专门记录数据，保持一条记录一个 JSON，事件名与字段统一，只发所选流。集成时 `PolarManager` 构造对应 `*Packet`，`try TelemetryEncoder.json.encode(packet)` 后交给 `UDPSenderService` 发送。数据格式/属性计划为：

    - HR：`{"type":"hr","bpm":<Int>,"t_device":<Double>,"device":"H10"}`
    - RR：`{"type":"rr","ms":<Int>,"t_device":<Double>,"device":"H10"}`
    - ECG：批量样本可以打包成一条消息，避免 UDP 包过多：
      `{"type":"ecg","fs":130,"uV":[s1,s2,...],"t_device":<Double>,"device":"H10"}`
    - ACC：同理用批量：
      `{"type":"acc","fs":50,"mG":[[x,y,z],[x,y,z],...],"range_g":4,"t_device":<Double>,"device":"H10"}`
      说明：`t_device` 先用 iPhone 当前 `Date().timeIntervalSince1970` 近似，后续可探讨使用 Polar 的时间戳做更严谨的对齐。H10 的 ECG/ACC 均为批量回调，批量发送比逐点发送更稳妥，也更利于下游 LSL 的消费者控制缓冲。
- 任务流程为：
  - AppStore开启`startCollecting()`，要求PM工作
  - PolarManager （在startHr(id:) → subscribe(onNext:)流程中）听到 AppStore 的需求通知，开始`startHrStreaming`，并发送UDP
  - CollectView.swift 指挥 `AppStore.startCollecting()/stopCollecting()`
- 数据精度：
  - **H10 ECG**：采样率固定 **130 Hz**（Polar 固件层限制），分辨率固定 14 bit。不能提高到更高频率，也没有可选档位。
  - **H10 ACC（加速度计）**：采样率可选 **25 / 50 / 100 / 200 Hz**；量程可选 **±2g / ±4g / ±8g**；分辨率 16 bit。这个是可以在应用里让用户选择的。
  - **HR / RR**：由心率特征流产生，不存在“采样率设置”。HR 大约 1 Hz，RR 是逐拍事件（毫秒整数），只能订阅或过滤，不能“调精度”。

Step 6｜记录数据标记(markers)

- 将“基线-诱导-干预”的数据标记发送到UDP并由LSL记录
- 该流程 从CollectView 发起。AppStore 接收到消息后开始执行向 UDP 发送标记信息任务（`emitMarker`）。这个任务要经过三道手。首先是 MarkerBus 第一手，它把Mark建构为一个结构化的数据（叫做 marker event ），为Marker添加各中必要的属性。第二道手是UDPMarkerBridge，它进一步对marker数据封装，把数据封装为始于UDP渠道流通的样式。接下来才是由UDPSenderService.swift 负责把信息发送到UDP目标地址。
- 而第一、二两道手还依赖一个IOS提供的底层机制`PassthroughSubject`，它是那个“事件总线--bus ”或“广播喇叭”的实体。
- 最后为了让 UDPMarkerBridge 始终在线，在AppStore开始的时候就要初始化 UDPMarkerBridge

Step 7｜端到端一轮标准实验流程

- 任务：佩戴 Verity；走“基线→诱导→干预→结束”手动标记流；同步录制到 LabRecorder。

- 检查电脑端IP，在手机端的调试页面将UDP的目标IP修改为电脑IP
- 连接 H10 后进入 CollectView，“选择数据”区域应出现 4 个药丸：HR、RR、ECG、ACC。
- 点选全部四个数据，开始采集：
  - 控制台应出现 `[POLAR][HR] started` 之类的启动日志，每 0.5–1 s 打印一次批量到达摘要：`[POLAR][HR] batch n=<样本数> lastHR=... rrCount=...`。
  - UDP 抓包（或在你的 UDP→LSL 控制台）只看到 `type="hr"` 与 `type="rr"` 两类消息，不出现 `ecg/acc`。
- 在采集中实时关闭/选择某个数据信号，5 秒内应在终端看到 `type=".."` 的实时报告。
- 点选 ECG，再开始采集或采集中追加 ECG：
  - 控制台出现 `ECG settings chosen: fs=130` 与批量到达摘要。
  - UDP 只出现新增 `type="ecg"` 的包，且 `uV` 数组长度与批次一致，fs=130。
- ACC 同理，默认 `fs=50, range_g=4`。
- 进入 LabRecorder：应能创建 2–4 条 LSL 流（取决于用户选择的药丸），采样率显示与上面一致。Kubios/Kaj的快速检验应与数据特性一致（HR 与 RR 曲线一致性、ECG 形态稳定、ACC 与体动相关）。

Step 8｜硬件自检与外部对照

- 任务：用 Polar Flow 或 Kubios App 独立验证 H10 工作正常（只做“硬件好坏”的 sanity check，不参与主数据流）。
- 验收：Flow/Kubios 上 HR 曲线与你 App 中 HR 变化方向一致（不要求数值完全相同，重点是设备功能正常）。

Step 9｜结果检查与归档

- 任务：测试App是否能正常工作。测试流程：

采集前核对

1. 检查电脑端的IP地址，当前的 `udp_to_lsl.py` 就有这个功能。
2. Home 页顶部“设置 UDP”：Host 填 Mac 的局域网 IP，Port 9001，点“应用”，首页状态行更新为该地址。
3. 连接 H10；Polar Flow 等会抢占蓝牙的应用全部关闭。
4. 进入 CollectView，在“选择数据”勾选：**HR**、**RR**、**ECG**、**ACC** 四项。
5. Mac 端先运行 `udp_to_lsl.py`，看到：
   - `listening on 0.0.0.0:9001`
   - `LSL outlets: PB_UDP_TEST, PB_MARKERS_TEST`

采集步骤（约 40–45 秒）

- 点“开始采集”。等待 2–3 秒，控制台应出现：
  - `[STREAM] ECG started (fs=130Hz)`
  - `[STREAM] ACC started (fs=50Hz, range=±4G)`
  - 以及 HR/RR 的回调打印。
- 打标记：基线开始→10s→基线结束→诱导开始→10s→诱导结束→干预开始→10s→干预结束。
- 点“停止采集”。（建议你已在 `stopCollect()` 里补发 `collect_end` 标记。）

期望的“数量级”结果（40±5 s 为例）

这是用于快速判断是否四路都在录，不是严格判分值：

- **ECG**：130 Hz，Polar 回调每批约 73 点，批频约 1.78 Hz
  预期批条数 ≈ 70–90
- **ACC**：50 Hz，每批 36 点，批频 1.39 Hz
  预期批条数 ≈ 45–65
- **HR**：~1 Hz
  预期条数 ≈ 30–50
- **RR**：随心搏，~1 Hz
  预期条数 ≈ 30–50
- **Markers**：至少 5 个
  `['baseline_start','baseline_end','stim_start','stim_end','intervention_start','intervention_end','collect_end']` 中的若干

合并后，`PB_UDP_TEST` 的总样本条数通常 > 180；`PB_MARKERS_TEST` ≥ 5。

Lab Recorder

- LabRecorder 勾选 `PB_UDP_TEST` 和 `PB_MARKERS_TEST`，录 40–45 s，结束。
- 跑 `qa_check.py`，理想看到的结构类似：

```
[STREAMS]
    - PB_UDP_TEST ... samples=2xx–3xx span≈40s
      - PB_MARKERS_TEST ... samples=5–7 span≈30s
[DATA COUNT BY TYPE] {'ecg': ~80, 'acc': ~55, 'hr': ~40, 'rr': ~40}
[LABEL]
['baseline_start','baseline_end','stim_start','stim_end','intervention_start','intervention_end','collect_end']
[RESULT] PASS`
```

如果 `qa_check.py` 里还没有“按 type 计数”，用这段最小补丁（放在它读取数据流的遍历处即可）：

```
from collections import Counter
type_counter = Counter()
for ts, sample in data_samples:  # 你现有遍历变量名可能不同
    try:
        obj = json.loads(sample[0])
        type_counter[obj.get("type","?")] += 1
    except Exception:
        pass
print("[DATA COUNT BY TYPE]", dict(type_counter))
```

Step 10｜数据后处理

数据后处理包括将Lab Recorder记录的 XDF 转变为 CSV 格式、绘制数据图以及评估数据质量等工作。这些工作在电脑端执行。详情见py脚本，这里只说结果

转化 csv 文件

`xdf_to_csv.py` 有五个 CSV 输出（四路数据 + 标记），CSV 文件结构为：

- `file_hr.csv`：`time_lsl, t_device, bpm, device`
- `file_rr.csv`：`time_lsl, t_device, ms, device, seq`
- `file_ecg.csv`：`time_lsl, t_device, uV, device, seq, fs`
- `file_acc.csv`：`time_lsl, t_device, x_mG, y_mG, z_mG, device, seq, fs, range_g`
- `file_markers.csv`：`time_lsl, label, note, packet_id`

> 注：若某字段在原始 JSON 中缺失，CSV 中留空。

运行 `xdf_to_csv.py` 后，会对文件给出简要报告：

- `[SUMMARY]` 每类数据的行数、时间跨度与采样率/量程集合
- `[FILES]` 输出 CSV 的路径
- `[SAMPLE HEAD]` 每个 CSV 的前 3 行
- `[DATA DICTIONARY]` 字段定义与单位

绘制数据图

`data_plot_validity.py` 会读取之前生成的 CSV 文件：`*_hr.csv / *_rr.csv / *_ecg.csv / *_acc.csv / *_markers.csv`，并逐类绘制曲线图：ECG 电位、ACC（支持幅值或三轴）、HR、RR，以及在时间轴上叠加 Marker 垂直线并标注标签。

运行脚本后，`plots_<csv目录名>/` 下应出现：

- `ecg.png`：ECG uV 曲线，标题包含采样率信息。
- `acc.png`（或 `acc_xyz.png`）：加速度幅值或三轴曲线，标题包含 fs 与量程。
- `hr.png`、`rr.png`：心率与 RR 间期。
- 若存在 `*_markers.csv`，各图上能看到若干条垂直标记线，并在顶端标注文本（如 baseline_start、stim_start 等）
- 如数据量巨大（ECG/ACC），脚本会自动按步长抽样，保证单图不超过 `--max-points` 指定的点数，避免卡顿或内存问题。

该 py 的参数包括

- `--csv-dir`：CSV 文件所在目录；不传则弹窗选择
- `--what`：`all|ecg|acc|hr|rr`，默认 `all`
- `--acc-mode`：`mag|xyz`（幅值或三轴），默认 `mag`
- `--max-points`：单图最大点数，超出则等步长抽样降采样，默认 `120000`
- `--show`：生成图片后弹出窗口预览（不传则只保存 PNG）

报告数据质量

检查以下数据项目：

- **采样完整性（ECG）**：统计样本数 `n` 与 `fs × 时长` 的比值，判定是否在 **±1%** 以内（PASS/FAIL）。
- **时间一致性（ECG）**：相邻样本 Δt 的**均值**与**理想步长** `1/fs` 的偏差（1%门限）与 **Δt 的标准差**。
- **序号单调性（ECG）**：若有 `seq` 列，统计**回退**与**跳号**次数（容忍极少量）。
- **HR vs RR**：把 `HR_from_RR = 60000/RR` 插值到 HR 时间轴，给出**均值误差**与**MAE**（目标 ≤2 bpm）。
- **事件对齐烟雾测试**：对预定义标签（`baseline_start / stim_start / stim_end / intervention_start / intervention_end`），计算**事件前后 2–5s** 的 **ECG |abs| 均值**与 **HR 均值**，报告 `pre/post/Δ`，用于粗检段落差异与标记落点是否合理。

结果解读（qa_report.txt 的关键条目）

- `completeness = observed/expected`
  接近 1.00 最理想；脚本按**±1%** 判定 PASS。
- `Δt mean` vs `1/fs`
  两者差距越小越好；偏差≤1% 视为 PASS，`std` 越小越稳。
- `HR_from_RR vs HR`
  给出平均误差与 MAE；**≤2 bpm** 一般说明同步正确、转换合理。
- `EVENT alignment`
  每个标记点前后 2~5 秒的 HR/ECG |abs| 均值（如果两个窗口都能取到数据）。从直觉上看，

诱导/干预的后窗与前窗应当出现一些差异，这只是**烟雾测试**，不是统计检验。