

获取Polar数据流3-开发计划

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。
-

建议的任务拆解顺序（只列标题与结果）

第一层：底层机制（与 UI 脱钩）

手机端

- **ContentView.swift**

这是界面和按钮逻辑的入口。它负责三件事：

1. 让你填目标地址与端口，并把这个目标应用到发送端。
2. “开始每秒发送/停止/发送一次”这三个调试按钮，会生成小的 JSON 文本（心跳），通过“数据发送器”发到电脑。
3. “基线开始/刺激开始/刺激结束/干预开始/干预结束”等标记按钮，会触发“标记总线”（见下）对外广播一个“我被点了”的事件。

- **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**

“数据发送器”。专门把 ContentView 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）

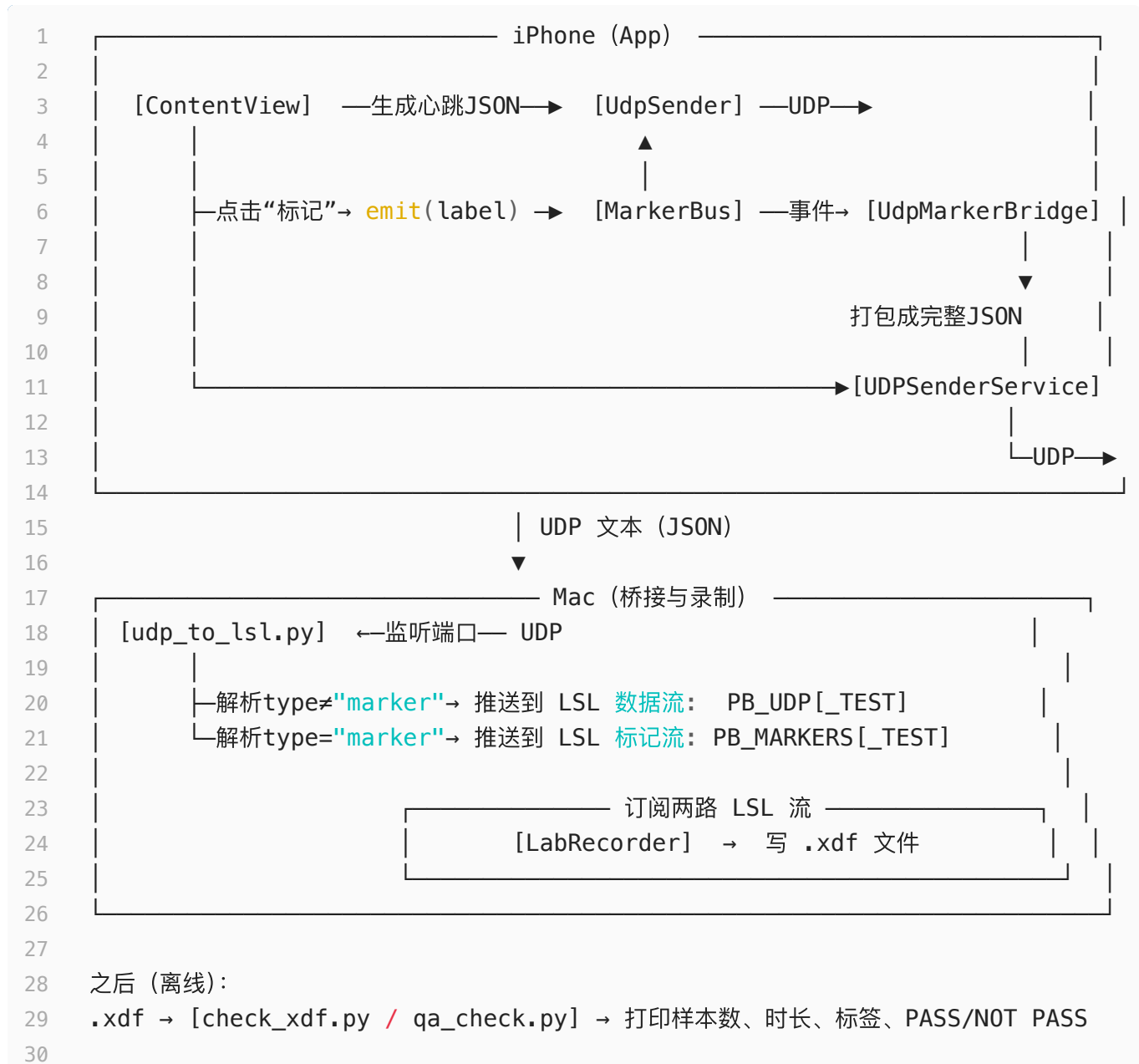
- **UDPSenderService.swift**

“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 ContentView → UdpSender。“标记”走 ContentView → 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



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 ContentView 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 ContentView：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。

第二层：UI 框架（纯静态/假数据）

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

| 已经完成

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）。

| 已经完成

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）。

已经完成

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”），颜色仅作参考。
 - 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。

已经完成

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。
- 要做：
 - 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。
- 验收：页面布局稳定，开始/停止切换不抖动；未开始时标记按钮禁用，开始后启用。

T5 | 交互约束与可用性 (禁用/互斥/节流/触觉)

- 目标：避免误触与“状态混乱”。
- 要做：

- 未开始：禁用所有标记按钮；开始后启用。
- 互斥示例：触发“诱导开始”后，在“诱导结束”前禁用再次“诱导开始”；结束后恢复。
- 节流：标记按钮 500ms 内只响应一次；触发时短促触觉反馈（Haptics）。
- 验收：连续狂点不会刷屏；互斥链条正确；停用/启用呈现一致。

T6 | 调试页（DebugView）最小化工具

- 目标：把“网络与日志”从主流程隔离。
- 要做：
 - 显示当前 UDP 目标（host/port）与一键“应用默认”按钮。
 - 最近 20 条发送日志（来自 Store 或临时内存），一键清空。
 - “重置为默认配置”按钮（仅改 UI 层状态，不改底层逻辑）。
- 验收：切换目标后首页与采集页同步显示；日志能实时看到心跳与标记行为。

T7 | 可访问性与空态（第一轮）

- 目标：首版就把“看不见 / 看不懂”风险降到最低。
- 要做：
 - 所有关键控件添加可读标签（VoiceOver 文案），确保“仅图标”不会迷惑。
 - 空态：未发现设备、无最近数据、无权限，都有文案和引导（不跳出弹窗）。
- 验收：系统设置里打开“旁白/VoiceOver”，逐控件朗读能说清楚作用；空态都能触发并显示。

T8 | 状态与持久化（只改 UI 层绑定）

- 目标：UI 与我们已有 AppStore 状态正确绑定。
- 要做：
 - HomeView 的设备卡从 AppStore 读状态；CollectView 的计数/计时读 AppStore 的 counters 与 isSending`。
 - @AppStorage 用于 UDP 目标；切换后三页一致显示。
- 验收：不接 Polar 的情况下，靠现有“心跳+标记”即可让 UI 跟着动；重启 App 后 UDP 目标保留。

T9 | 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。

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。
- 验收：真机构建成功，无 “Missing package product 'PolarBleSdk'” 等错误。

Step 3 | 设备扫描与识别 (Verity)

- 任务：实现最小扫描，只在 Xcode 控制台打印发现的 Verity (deviceId、RSSI、电量如可得)；UI 不必显示。
- 验收：打开 Verity 后，若干秒内稳定打印到它的 deviceId。

Step 4 | 连接与订阅 HR (先做易的)

- 任务：根据 deviceId 建立连接；订阅 HR (bpm)，如 SDK 同步给出 RR/IBI 也一并接入；封装为 JSON (version、session、device、stream、packet_id、t_host、payload)。

- 验收：Xcode 中能连续看到 HR 数字；Mac 桥接器每秒打印 [DATA]；短录 20–30 秒后，PB_UDP 样本数与时长匹配，qa_check.py PASS。

Step 5 | 订阅 PPI（逐拍间期）

- 任务：接入 PPI 流；同一路 UDP JSON 发送（stream:"ppi"，payload 含 ppi_ms、如有 hr_est）；必要时控制发送频率与批量打包。
- 验收：短录后能在 PB_UDP 看到 PPI 条目，时间推移合理；与 HR 同时存在时，检查两类样本交错记录是否正常。

Step 6 | 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。
- 验收：PB_MARKERS 至少包含 baseline_start / stim_start / stim_end（若做干预则有其成对标签）；计时显示与实际标记时刻一致；qa_check.py PASS。

Step 7 | 硬件自检与外部对照

- 任务：用 Polar Flow 或 Kubios App 独立验证 Verity 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 8 | 结果检查与归档

- 任务：check_xdf.py 查看流名、样本数、时长；qa_check.py 严格把关；保存 session.jsonl 旁路日志与 .xdf 成对归档（目录按日期/受试编号命名）。
- 验收：两脚本输出都正常；归档结构固定、可快速检索。

关键评述与注意点

- 你原先的“3. 佩戴 Verity，通过 Flow/Kubios 检验”很有必要，但**不要依赖它作为数据源**。它只用于确认传感器本身没有坏，主数据链路始终是“SDK→UDP→LSL→XDF”。
- **PPG 原始波形**暂时不纳入第一轮目标。先把 HR 与 PPI 打通，PPG raw 体量更大、对传输与绘制要求更高，等 H10 ECG 打通后再统一考虑“高带宽流”策略。
- UDP 仍然是**同一通道**发送不同 stream 的 JSON；区分靠 stream 与 device 字段，桥接器不需要开新端口。
- 计时器第一版仅做“显示与手动标记”，是正确的取舍。你已经在 XDF 中拿到“事件时间窗”，后续再把“自动倒计时、自动标记、自动切段”外接到这套 UI，很顺。

每步的最小“通过线”清单（Go/No-Go）

- Step 0: 真机心跳与 marker 能录进 XDF, `qa_check.py` PASS。
- Step 1: 计时显示正确, 四个段落按钮与 marker 一一对应。
- Step 2: 能在真机关联构建, 不报包管理错误。
- Step 3: 控制台能稳定打印 Verity 的 deviceId。
- Step 4: HR 连续、无明显中断; UDP 与 XDF 中样本数随时长增长。
- Step 5: PPI 有序、数值在合理范围 (600–1200 ms 为常见静息范围, 运动会更低)。
- Step 6: 标记序列完整、计时读数与标记时刻对应。
- Step 7: Flow/Kubios 中设备表现正常。
- Step 8: 脚本检查通过, 归档规范。

按这个版本推进, 就能先得到**可靠的最小可用系统**, 同时为后续接入 H10 ECG、倒计时自动化和外部刺激联动保留清晰的扩展点。