# WatchYourBehind 系统架构设计文档 (V1.0.1)

## 1. 系统概述 (Overview)

本系统构建了一个从 毫米波感知 到 云端可视化 的完整物联网闭环。

V3.2 版本引入了 “状态驱动 (State-Driven)” 架构，核心目标不仅仅是数据的单向采集，更实现了端到端的 指令闭环控制 和 按需加速 机制。系统能够根据前端用户的活跃状态，动态调整边缘设备的采样频率，并支持基于区域的“守卫模式”预警。

## 2. 系统分层架构

系统分为四层架构：

1. **感知层 (Sensing Layer)**: HLK-LD2450 毫米波雷达模块 (负责原始信号处理)。
2. **边缘层 (Edge Layer)**: ESP32 智能网关 (负责协议转换、指令解析、边缘过滤)。
3. **服务层 (Service Layer)**: Nginx + API Backend + MySQL (负责业务逻辑、状态管理、告警触发)。
4. **应用层 (App Layer)**: Web 前端实时仪表盘 (负责可视化渲染、指令下发)。

## 3. 边缘层：ESP32 智能网关逻辑

ESP32 在本架构中不再只是透明传输，而是具备**指令缓存**和**状态执行**能力的智能节点。

### 3.1 核心固件逻辑 (Firmware Logic)

* **指令闭环控制 (Control Loop)**:
  + **机制**: 由于 ESP32 位于内网，无法被动接收公网请求。因此采用 **“捎带响应 (Piggyback)”** 机制。
  + **流程**: ESP32 发起 POST /sync 上报数据 -> 服务器在 HTTP Response 中返回 pending\_cmd -> ESP32 解析并执行指令（如切换模式、重启、写入配置）。
  + **0x00C2 协议支持**: 针对“守卫模式”，固件新增了对 0x00C2 (设置区域过滤) 指令的解析逻辑。当收到 API 下发的 JSON 坐标数组时，ESP32 将其转换为 26 字节的 Hex 补码流写入雷达，在硬件层过滤掉非关注区的干扰源。
* **动态频率控制 (Active Sync)**:
  + **待机模式 (Standby)**: 默认 **1Hz** (1000ms/次)。当 API 返回 next\_interval: 1000 时，系统处于低功耗、低流量状态。
  + **加速模式 (Turbo)**: 当 API 返回 next\_interval: 100 时，ESP32 立即切换至 **10Hz** (100ms/次) 上报频率，以支持前端的丝滑实时渲染。

## 4. 服务层与通信协议

### 4.1 数据流向与控制流

graph TD  
 %% 物理层  
 Radar[HLK-LD2450] -- UART(256000) --> ESP32  
  
 %% 边缘层  
 subgraph Edge\_Layer  
 ESP32 -- 1. POST Data --> Nginx  
 ESP32 -- 3. Execute Cmd --> Radar  
 end  
  
 %% 云端  
 subgraph Cloud\_Server  
 Nginx -- Proxy --> API[API Backend]  
 API -- Write --> DB[(MySQL)]  
 API -- Read Cmd --> CmdQueue[指令缓存区]  
 API -- 2. Response (Next\_Interval + Cmd) --> ESP32  
 end  
  
 %% 前端  
 User[Web Dashboard] -- WebSocket (10Hz Stream) --> Nginx  
 User -- POST Config (Zone/Mode) --> API  
 API -- Enqueue --> CmdQueue

### 4.2 边缘-云端通信接口 (HTTP Interface)

* **上报与心跳接口**: POST /api/v1/device/sync
* **请求体 (Request)**: 包含雷达解析后的 3 个目标坐标 (X, Y, Speed, Resolution)。
* **响应体 (Response)**: 状态驱动的核心载体。  
  {  
   "code": 200,  
   "data": {  
   "next\_interval": 100, // 核心控制：告诉 ESP32 下次多久后联系我 (100ms 或 1000ms)  
   "server\_time": 1715000000,  
   "pending\_cmd": { // 指令下发通道 (可选)  
   "type": "SET\_ZONE", // 对应 0x00C2  
   "payload": [ // 预警区坐标 (将被固件转换为 Hex)  
   {"x1": 100, "y1": 200, "x2": 500, "y2": 600},  
   ...  
   ]  
   }  
   }  
  }

### 4.3 历史数据聚合 API

为了减轻前端在渲染“空间占用热力图”时的压力，API 层新增聚合接口：

* **热力图聚合**: GET /api/v1/analytics/heatmap
  + **逻辑**: 不返回数万个原始点位。API 根据请求的 grid\_size (如 500mm)，在数据库层面（或内存中）将历史坐标映射到网格中，返回每个网格的 density\_count。

## 5. 守卫模式 (Guardian Mode) 逻辑架构

“守卫模式”的实现采取 **边缘-云端协同** 的策略：

1. **边缘层 (过滤)**:
   * **职责**: 执行雷达内部指令 0x00C2 (Set Zone Filtering)。
   * **作用**: 在硬件源头屏蔽掉“预警区”以外的信号（如窗帘摆动、宠物活动），确保上报的数据聚焦于核心区域。
2. **服务层 (判定与告警)**:
   * **职责**: 维护“警戒时间轴”和“报警记录”。
   * **逻辑**:
     + 当数据上报时，API 检查当前是否处于用户设定的 Alert Time Window (如 22:00-06:00)。
     + 如果数据点落入 device\_shadow 中存储的 zone\_config 区域，且速度/停留时间符合阈值，则触发 **Alarm Event**。
     + 生成一条记录写入 guard\_events 表，并通过 WebSocket 向前端推送红色警报状态。