基于OpenHarmony的低功耗宠物感知与定位系统

——宠物随身终端与云端数据同步设计

(过时初稿)

摘　要

针对当前宠物户外走失风险高、健康状态无法量化监测、传统智能设备功耗高、应用臃肿、定位漂移严重等行业痛点，本文设计并实现了基于 OpenHarmony 的低功耗宠物感知与定位系统。系统采用软硬件一体化设计，硬件以 Hi3863 主控平台为核心，搭载 4G 蜂窝通信模块，集成 GNSS 定位、宠物体温采集、运动姿态感知等多维度数据采集单元，实现全天候、低功耗的终端数据采集与网络上传；软件端基于鸿蒙 ArkTS 原生开发，结合云端数据库、高德开放天气 API，实现宠物实时定位追踪、电子围栏防走失防护、多场景异常报警、云端数据持久化与可视化展示等完整业务功能。

为解决传统物联网项目依赖重型第三方 SDK、定位误差大、功耗偏高、安装包冗余臃肿等问题，本项目完成多项技术优化与量化升级。定位层面，系统集成 WGS-84 与 GCJ-02 双向坐标纠偏算法，配合卡尔曼滤波对原始 GNSS 数据进行降噪平滑处理，有效消除高楼遮挡、信号干扰带来的定位漂移问题，将城区场景平均定位误差由 50 米优化至 10 米以内，显著提升定位稳定性与准确性。通信层面，本项目遵循开源 MQTT 3.1.1 协议规范轻量化协议栈，完全替代传统重型第三方物联网 SDK，剔除冗余依赖，大幅精简鸿蒙应用安装包体积，使软件运行更轻量、响应更迅速、兼容性更强。功耗层面，通过优化 4G 数据上报频率、动态调整通信心跳策略，降低终端无效通信损耗，有效降低整机运行功耗，设备续航能力得到明显提升，具备进一步自适应网络节能迭代的扩展空间。

经功能测试与场景验证，本系统数据采集稳定、通信链路可靠、异常响应及时、定位效果精准，可有效实现宠物户外安全防护与日常健康状态智能监测，满足家庭宠物智能化看护的实际应用需求，具备较强的实用性、稳定性与工程推广价值。

关键词：OpenHarmony；宠物监测；坐标纠偏；轻量化 MQTT；低功耗设计

目 录

1、项目概述 1
1.1 项目背景 1
1.2 项目目标 1
1.3 报告编写目的 1
2、 相关技术概述 1
2.1 鸿蒙操作系统与 ArkTS 开发 1
2.2 MQTT 物联网通信协议原理 1
2.3 TCP Socket 网络通信技术 2
2.4 WebView HTML 5 混合开发技术 2
2.5 高德 Web 地图与坐标纠偏技术 2
2.6 鸿蒙轻量化本地存储技术 2
3、系统需求分析 2
3.1 功能需求 2
3.2 性能需求 4
4、系统总体设计 4
4.1 系统整体架构 4
4.2 系统功能模块总体划分 5
5、硬件系统设计 5
5.1 硬件整体方案说明 5
5.2 硬件整体架构 5
5.3 核心硬件模块介绍 6
5.4 硬件整体工作流程 6
5.5 硬件设计优势 7
6、软件系统设计 7
6.1 软件开发技术架构 7
6.2 核心软件模块设计 8
6.3 软件整体设计特点 26
7、系统风险与应对措施 27
7.1 硬件采集风险与应对 27
7.2 网络通信风险与应对 27
7.3 软件运行异常风险与应对 27
7.4 定位与第三方接口风险与应对 27
8、数据库设计 28
8.1 概述 28
8.2 数据表设计 28
8.3 核心数据流与业务交互 32
8.4 核心业务 SQL 语句 33
8.5 数据库设计特点 34
9、系统测试与运行结果 34
9.1 测试目的 34
9.2 测试环境 34
9.3 功能测试内容与结果 35

9.4 性能测试结果 35
9.5 整体运行结论 36
10、项目总结与展望 36
10.1 项目总结 36
10.2 项目展望 36
参考文献 37
附录 项目开发计划 37
1 概述 37
2 团队职责分工 37
3 项目开发阶段规划 38

1、项目概述

1.1 项目背景

随着社会养宠人群数量持续增长，宠物已成为多数家庭的重要陪伴成员。在日常饲养过程中，宠物外出走失、突发疾病、高温中暑、剧烈运动异常、长时间独处状态异常等问题频发。传统人工看管方式存在监管滞后、无法实时感知、异常无法预警等短板，无法满足现代化、智能化养宠需求。

为解决宠物位置走失、健康异常、活动异常、环境异常等问题，本项目设计并实现一套宠物随身监测系统。系统依托便携硬件终端与移动端软件结合的方式，实现宠物位置定位、体温监测、运动状态采集、环境温湿度监测、异常报警、历史轨迹回放、数据记录分析等功能，帮助主人全天候掌握宠物状态，提升宠物饲养安全性与智能化水平。

1.2 项目目标

本项目旨在开发一款基于 OpenHarmony 操作系统、面向宠物户外活动与日常健康管理的智能监测系统。依托嵌入式终端硬件，集成多类感知传感器与 4G 通信模块，实现宠物实时位置追踪、体态状态监测与异常风险告警；结合 WGS-84 与 GCJ-02 坐标纠偏、轻量化通信协议优化，有效提升定位精度与设备续航表现。搭配云端数据服务与鸿蒙移动端应用，为用户提供数据同步、电子围栏、可视化查看与远程管理功能，构建轻量化、低功耗、高实用性的宠物智能管护解决方案。

1.3 报告编写目的

本技术报告全面阐述宠物随身监测系统的需求分析、总体架构、硬件设计、软件设计、功能实现、测试结果、问题优化、项目总结等内容，完整记录系统开发全过程，为项目验收、课程设计归档、系统后期迭代维护提供完整技术依据。

2、 相关技术概述

2.1 鸿蒙操作系统与 ArkTS 开发

鸿蒙操作系统是华为推出的全场景分布式操作系统，具备高流畅、低功耗、高安全、多设备协同的优势，适配手机、智能终端、物联网设备等多类硬件平台。相较于传统安卓系统，鸿蒙系统资源占用更低、运行更稳定、组件化开发更规范，适合轻量化物联网配套 APP 开发。

本项目移动端采用 ArkTS 语言开发，ArkTS 是鸿蒙主力开发语言，在 TypeScript 基础上强化静态类型检测、UI 声明式语法、状态响应式机制。语言语法简洁、代码可读性高、编译效率高，能够快速搭建结构清晰、交互流畅的移动端页面，适合本项目多模块、数据实时刷新的业务场景。

2.2 MQTT 物联网通信协议原理

MQTT 是一种发布/订阅模式的轻量化物联网通信协议，专门针对低带宽、弱网络、低功耗的物联网场景设计，具备协议报文短小、通信开销低、长连接稳定的特点，是物联网设备数据传输的主流协议。

MQTT 协议分为服务端与客户端，客户端通过订阅指定主题、发布数据报文实现双向数据交互。协议内置心跳保活机制，能够实时检测链路状态，适合硬件终端长时间在线、持续上传数据的业务需求。

本项目通信业务以 MQTT 协议为应用层核心，遵循协议标准规范完成连接、主题订阅、消息发布与心跳维持，实现设备与应用间高效的数据交互。

2.3 TCP Socket 网络通信技术

TCP 是面向连接的可靠传输协议，具备数据不丢失、顺序不乱、重传纠错的特点，是互联网主流的底层传输基础。

MQTT 协议全程基于 TCP 通道进行封装与传输，本项目依托鸿蒙原生 TCP Socket 能力搭建底层长连接，完成二进制数据流收发、报文解析与数据编解码。

通过自主操控 TCP 底层链路，承载上层 MQTT 协议交互，减少第三方依赖，保障通信链路轻量化、可控性强，提升 APP 整体运行稳定性。

2.4 WebView HTML 5 混合开发技术

WebView 是鸿蒙系统提供的网页容器组件，本项目通过 WebView 内嵌高德 HTML 5 地图页面，采用直接调用高德 Web JS API 的方式实现地图功能，无需引入重型原生地图 SDK。

应用以原生 ArkTS 负责设备通信、业务逻辑与数据转发，HTML 5 页面承载地图渲染、点位标注与轨迹展示，通过简单交互通信完成数据同步。该方式部署轻便、接口调用直接，大幅降低开发适配难度。

2.5 高德 Web 地图与坐标纠偏技术

硬件 GNSS 模块采集的原始坐标为 WGS84 国际标准坐标，国内地图服务统一使用 GCJ02 火星坐标系，直接展示会出现定位偏移。

本项目在内嵌的高德 HTML 5 地图中，依托官方 Web API 完成地图加载与功能调用，同时封装坐标转换算法，将设备上传的 WGS84 原始坐标自动纠偏为 GCJ02 坐标，确保地图点位位置准确，解决定位漂移问题，完整实现标注、弹窗、轨迹绘制等地图能力。

2.6 鸿蒙轻量化本地存储技术

本项目采用鸿蒙 Preferences 轻量数据库实现本地数据持久化存储，以键值对形式保存城市配置、设备参数、常用设置等信息。应用重启、进程关闭后数据不丢失，保证软件配置稳定、无需重复初始化，提升用户使用体验与系统稳定性。

3、系统需求分析

3.1 功能需求

3.1.1 定位与地图功能需求

本系统需具备高精度宠物位置监测与地图可视化管理能力，满足用户实时监护。

区域管控与轨迹追溯需求，具体需求如下：

1. 系统可通过硬件终端实时采集宠物 GNSS 位置信息，移动端内嵌高德 HTML 5

地图可动态刷新宠物实时点位，实现位置可视化监控。
2．支持用户在地图界面自主绘制、修改电子围栏安全区域，系统可实时校验宠物当前位置，一旦超出预设边界，即刻触发越界预警提醒。
3．系统可持久化存储设备上传的定位点位数据，支持用户自定义时间区间查询，实现宠物历史行走轨迹回放与运动路线追溯。

3.1.2 健康监测功能需求

系统依托硬件传感单元实现宠物体征与行为状态监测，实时掌握宠物身体与活动状态，具体需求如下：
1．通过板载体温传感模块持续采集宠物体温数据，结合正常体温阈值区间，自动判断宠物是否存在体温异常情况。
2．依托运动姿态传感器实时捕捉宠物运动数据，精准识别宠物静止、正常活动、剧烈躁动等行为状态，实现日常活动量与行为状态监测。
3．系统自动存储长期健康监测数据，支持历史数据回溯与状态趋势查看，方便用户长期跟踪宠物健康变化规律。

3.1.3 环境监测功能需求

本项目采用通过云端天气 API 实现环境温度监测，依托网络数据完成环境风险判断，具体需求如下：
1．应用端根据宠物实时定位坐标，动态对接云端天气服务 API 接口，获取对应地理位置的气象数据。
2．根据宠物当前所处地理位置，在线拉取并刷新实时环境温度信息，实现环境温度远程监测。
3．依托云端标准化气象数据，系统可识别高温、低温等恶劣环境状态，完成宠物外出环境风险监测与提示。

3.1.4 智能报警功能需求

系统具备多维度异常识别与自动预警能力，针对走失、健康、环境、行为四类异常场景实现智能提醒，具体需求如下：
位置异常：宠物走出电子围栏安全区域，系统自动触发走失越界报警。
健康异常：监测宠物体温超出正常阈值区间时，触发体温异常预警。
环境异常：当前环境温度超出安全范围时，触发恶劣环境温度风险预警。
行为异常：识别宠物长时间静止不动或持续剧烈躁动等异常行为，自动推送行为状态异常提醒。

3.1.5 数据记录与可视化需求

系统对全部监测数据进行统一持久化管理，并通过移动端可视化展示，提升数据可读性，具体需求如下：
1．对宠物定位数据、体温与行为健康数据、云端环境温度数据、各类报警记录进行本地持久化存储，保证数据不丢失、可追溯。
2．鸿蒙移动端通过数字卡片、状态标签、趋势曲线图等可视化形式，直观展示实时位置、健康状态、环境温度、异常记录等信息。
3．支持用户随时查询全部历史监测数据与报警日志，便于长期分析宠物活动规律与健康变化趋势。

3.1.6 系统交互 UI 功能需求

为保证系统易用性与视觉规范性，客户端界面需满足交互简洁、分区清晰、状态明确的设计需求：
1．移动端界面布局结构清晰，将地图监控、健康数据、环境信息、预警提示等

功能模块分区展示，界面整洁有序。
2．整体操作逻辑简单易懂，核心功能入口直观明确，页面跳转流畅，适配日常快速操作使用场景。
3．系统对异常数据、预警信息采用差异化颜色标识与醒目提示，强化风险视觉提醒，提升用户识别效率与使用安全性。

3.2 性能需求

结合物联网穿戴设备低功耗、长时间在线、稳定监护的应用场景，本系统性能需求如下：
1．实时性需求：设备数据上报延迟低，宠物位置、体温、运动状态、环境温度数据更新及时，各类异常预警触发响应快速，满足实时监护要求。
2．稳定性需求：硬件终端可长时间连续联网运行，MQTT 网络链路稳定，不易掉线断连；APP 运行流畅，数据解析准确，长时间使用无闪退、卡顿、数据丢失问题。
3．低功耗需求：硬件终端采用低功耗运行策略，适配穿戴设备的续航需求，可支持宠物长时间外出佩戴、全天候在线监测。
4．便携性需求：硬件终端采用优化后的结构设计，整体结构紧凑、重量轻便、佩戴稳固，不会束缚宠物肢体活动，适配日常居家、外出散步等多种使用场景。

4、系统总体设计

4.1 系统整体架构

本章将对系统的总体方案进行设计，明确系统的整体架构与各层职责。本系统采用分层架构设计，自下而上分为感知终端层、传输层、云端服务层与应用层，系统总体架构如下图所示：

4.1.1 硬件感知层

硬件终端以 Hi3863 物联网开发板为主控核心，搭载 GNSS 定位模块、体温检测模块、运动姿态传感模块。硬件端主要负责采集宠物地理位置、体表温度、运动行为状态等原始物理数据，完成数据预处理后，基于 MQTT 协议（TCP 底层通道）将数据稳定上传至云端服务，为系统所有业务功能提供原始数据支撑。

4.1.2 云端服务层

云端服务作为系统中间核心枢纽，主要承担硬件数据接收、协议解析、数据持久化存储、业务逻辑判断等核心工作。本系统不依赖本地环境传感器，由云端服务根据宠物实时 GNSS 坐标动态调用第三方天气 API，获取对应地理位置的实时环境温度数据，完成环境监测数据的补充与运算，完善系统环境风险监测能力。

4.1.3 移动端应用层

移动端基于鸿蒙系统与 ArkTS 开发，内嵌高德 HTML 5 可视化地图。主要负责宠物实时点位展示、历史轨迹回放、健康数据展示、云端环境温度呈现、电子围栏区域管理、多场景异常报警推送、数据可视化统计、人机交互操作等全部前端业务，为用户提供直观、便捷的宠物远程监测服务。

4.2 系统功能模块总体划分

根据系统业务逻辑与功能职责，将整体系统模块化拆解为六大相互协同、独立运行的核心功能模块，模块分工明确、逻辑闭环，具体划分如下：

1. 定位采集与轨迹回放模块：负责宠物 GNSS 位置采集、实时点位上报、点位数据存储、自定义时间段历史轨迹回放。
2. 宠物健康监测模块：实现宠物体温数据、运动姿态数据的实时采集、分析与健康状态判定。
3. 环境参数监测模块：依托云端天气 API，根据实时坐标获取环境温度，实现外界环境温度监测与风险判断。
4. 智能异常报警模块：针对位置越界、体温异常、环境温度异常、宠物行为异常四大场景实现自动预警提醒。
5. 数据存储与统计模块：统一存储定位、健康、环境、报警日志数据，支持数据回溯与趋势可视化统计。
6. 系统交互与 UI 界面模块：负责页面布局展示、数据可视化渲染、异常状态醒目提示。

5、硬件系统设计

5.1 硬件整体方案说明

本系统硬件终端采用宠物随身设备一体化集成设计，将主控开发板、传感检测模块、GNSS 定位模块及稳压电源模块统一集成封装。整体设备体积紧凑、重量轻便、佩戴贴合稳固，不会束缚宠物肢体活动，不影响宠物行走、跑动、跳跃等日常行为，能够良好适配居家、户外出行等全场景随身监测需求，具备极强的实际使用适配性。

5.2 硬件整体架构

本系统硬件采用主控核心单元 + 外设传感单元 + GNSS 定位单元 + 稳压供电单元的一体化分层架构。所有硬件模块均集成部署于智能终端内部，硬件端主要负责完成宠物地理位置定位、体表温度采集、运动姿态感知等物理数据的采集与预处理工作，最终将规整后的有效数据通过物联网通信协议上传至云端平台，为软件系统的定位监测、健康分析、风险预警、数据可视化等全部业务功能提供稳定、可靠的底层硬件数据支撑。

5.3 核心硬件模块介绍

5.3.1 主控处理模块

本系统以 Hi3863 物联网开发板作为硬件终端唯一主控核心，全权负责设备整体任务调度、多路传感器数据轮询采集、原始数据滤波去噪、数据结构化打包、通信链路维护与阈值预判断等核心工作。该主控芯片资源配置合理、运行功耗低、稳定性高，具备优秀的外设拓展与并行处理能力，可同时驱动定位、测温、姿态传感等多模块协同工作，保障穿戴式终端长时间不间断稳定运行。

5.3.2 GNSS 定位模块

终端搭载独立 GNSS 定位模块，用于实时获取设备经纬度坐标信息，定位数据稳定可靠。依靠 GNSS 采集的位置数据，为软件端地图点位展示、电子围栏越界判断、历史轨迹回放功能提供精准位置来源。

5.3.3 体温检测模块

硬件终端搭载高精度接触式体温传感单元，可实时、连续采集宠物体表温度数据，精准捕捉体温细微变化。通过内置数据校准算法剔除干扰数据，保证测温结果真实可靠，为宠物健康状态监测、体温异常预警功能提供核心硬件数据依据。

5.3.4 运动传感模块

系统集成六轴惯性测量单元，可全方位实时捕捉宠物的运动姿态与行为变化，能够精准区分宠物静止休息、正常活动、剧烈躁动等不同行为状态，实现宠物日常活动量统计与异常行为识别，丰富宠物健康行为监测维度。

5.3.5 环境监测实现方案

系统环境监测功能采用轻量化云端数据方案实现。硬件通过自身实时 GNSS 定位坐标上传位置信息，软件端动态调用第三方天气 API 接口，远程获取宠物当前属地的实时环境温度数据。依托云端权威气象数据完成高温、低温等恶劣环境风险判定，在缩减硬件体积、降低设备功耗的同时，高效完成环境监测业务需求。

5.3.6 电源供电模块

硬件系统采用大容量可充电锂电池搭配稳压供电电路，为整套监测终端提供稳定、持续的电压输出。供电电路具备过充保护、稳压稳流特性，电路安全可靠，可支撑设备长时间连续数据采集、运算与网络传输工作，满足宠物全天候随身监测的续航使用需求。

5.4 硬件整体工作流程

1. 智能硬件终端上电启动，主控芯片完成 GNSS 定位模块、体温传感模块、运动姿态模块、通信端口的初始化与自检工作；
2. 设备按照预设采集频率，周期性获取宠物地理位置、体表温度、运动姿态等原始硬件数据；
3. 主控对全部原始数据进行滤波去噪、异常剔除、数据校准处理，消除抖动误差，保障采集数据真实准确；

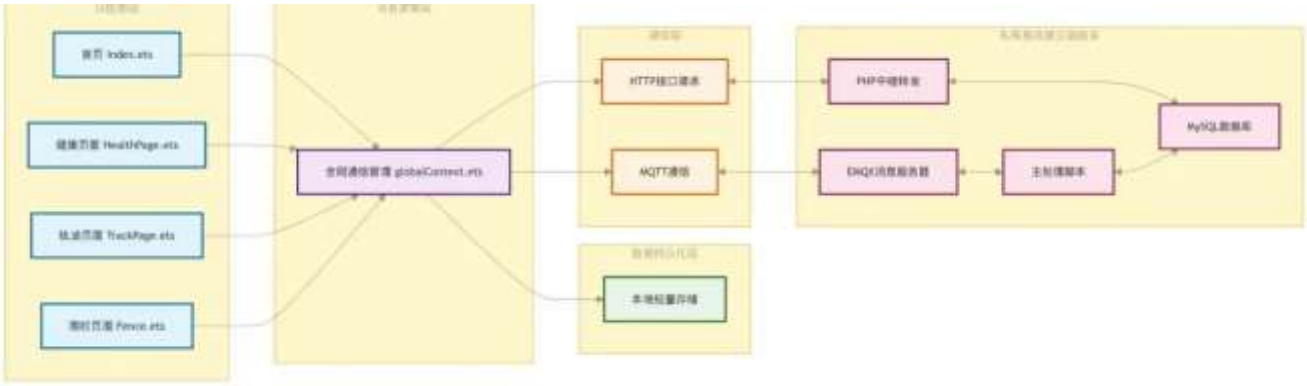
4．将处理完成的结构化数据通过 MQTT 通信协议依托 TCP 传输通道，稳定上传至云端服务平台；
5．云端服务完成数据解析、持久化存储与业务逻辑判定，最终推送至移动端，实现数据展示、状态分析与异常报警全流程功能。

5.5 硬件设计优势

1．佩戴结构稳定可靠：设备贴合度高、不易松动移位，有效避免设备晃动造成的数据偏差，保障采集数据稳定性。
2．宠物适配性极佳：硬件体积轻量化、布局紧凑，不限制宠物肢体运动，完全适配宠物日常活动习惯，适合长期佩戴监测。
3．功能集成度高：硬件一体化集成定位、测温、姿态感知多类监测能力，搭配云端 API 补充环境监测功能，实现宠物全维度安全健康监测。
4．系统运行稳定性强：硬件电路设计成熟规范，数据采集精度高、抗干扰能力强，设备长期连续运行不掉线、不宕机，满足全天候监测需求。
5．调试与实用性强：模块化设计清晰，设备安装、拆卸、检修便捷，结构简洁耐用，完全符合物联网智能穿戴设备项目开发规范与实际落地需求。

6、软件系统设计

6.1 软件开发技术架构



本项目移动端应用软件基于鸿蒙操作系统，采用 ArkTS 声明式 UI 开发，整体采用四层分层模块化架构，分别为通信交互层、数据持久化层、业务逻辑层与 UI 视图层。各层级职责独立、低耦合高内聚，软件结构规整、运行稳定，便于后期调试维护与功能拓展。

通信交互层以 MQTT 物联网协议为核心，依托 TCP 底层传输通道搭建移动端与硬件终端的长连接通信链路，实现传感器数据、GNSS 定位信息、设备状态的实时上行推送；同时支持移动端配置指令、同步请求的远程下发。结合 HTTP 网络请求，对接云端轨迹服务与第三方天气接口，完成历史轨迹加载、区域环境数据获取，保障跨端、跨接口数据交互稳定可靠。

数据持久化层依托鸿蒙 Preferences 轻量存储组件，实现关键配置与业务数据本地永久保存，主要存储城市配置、围栏参数、每日健康记录等信息，应用重启后数据不丢失，有效保障配置连续性与数据完整性。

业务逻辑层采用模块化拆分设计，依托全局数据管理类统一调度，划分为首

页汇总、健康监测、历史数据解析、城市配置、轨迹地图、电子围栏、设备详情等功能模块，各模块业务逻辑独立运行，提升代码复用性与项目可维护性。

UI 视图层采用鸿蒙原生声明式组件开发，遵循数据驱动视图设计模式，页面控件、数据卡片、列表样式、状态提示随全局数据自动刷新，界面交互流畅，适配移动端设备使用场景。

6.2 核心软件模块设计

6.2.1 首页主入口模块（Index.ets）

模块功能概述

首页作为系统核心入口页面，基于 OpenHarmony ArkTS 声明式 UI 开发，采用 Tabs 分页架构，分为主页、数据详情、个人中心三大板块。页面整合全局状态管理、MQTT 服务自动连接、多组件嵌套复用、路由跳转、动态样式渲染等能力，是整个应用软件的调度中枢。

模块核心功能

1. 全局数据共享绑定

通过@StorageLink 全局持久化绑定电量、步数、体温、环境湿度、电子围栏状态等硬件采集数据，实现跨组件、跨页面数据实时同步。

2. 轻量化 MQTT 服务调度

页面生命周期内自动初始化 MQTT 连接服务，依托轻量化 MQTT 协议栈完成云端长连接通信，支持手动强制重连与数据同步，保障 4G 终端设备数据稳定上报。

3. 多复用组件组合布局

模块化引入天气背景、宠物动态形象、悬浮信息面板、健康卡片、快捷功能按钮等自定义组件，低耦合组装，提升代码复用性与界面美观度。

4. 动态主题与风险警示

根据宠物体温阈值动态切换全局渐变背景、阴影配色与状态标签；体温异常时自动加载风险视觉样式，实现健康状态可视化预警。

5. 多页面路由跳转

封装导航跳转逻辑，支持宠物地图、历史轨迹、电子围栏、健康报表、系统设置等功能页面快速切换，完善全流程业务闭环。

6. 定时状态刷新与资源管理

通过定时器定时刷新设备在线状态与同步时间提示；页面销毁时自动清空定时器，避免内存占用与资源泄漏。

核心代码

全局设备感知数据绑定

```
@StorageLink('battery') battery: number = 72;

@StorageLink('stepCount') stepCount: number = 0;

@StorageLink('temp') temp: number = 0;

@StorageLink('isInsideFence') isInsideFence: boolean = true;
```

自动初始化轻量化 MQTT 长连接服务

```
async startMqttService() {

  try {
```

```
    await globalManager.connectToServer();
  } catch (e) {
    console.error('MQTT 服务连接异常');
  }
}
```

页面加载自动启动云端通信

```
onAppear(() => {
  if (!this.isAutoStarted) {
    this.startMqttService();
    this.isAutoStarted = true;
  }
})
```

功能特点

1.采用组件化拆分思想，将宠物动画、天气展示、信息面板拆分为独立自定义组件，结构清晰、便于后期维护迭代。

2.全程基于原生 API 开发,不引入重型第三方 UI 与通信 SDK,契合项目轻量化、低资源占用的设计目标。

3.数据驱动 UI 渲染，所有健康数据、设备状态、风险提示均由底层感知数据自动控制，智能化程度高。

6.2.2 宠物动态形象交互模块（pixelcat.ets）

模块功能概述

为提升系统界面可视化与交互趣味性,本软件设计实现宠物动态形象交互模块。模块基于 OpenHarmony 声明式 UI 开发，结合全局状态绑定、数据监听、帧动画与定时器调度，实现宠物行为与健康数据联动展示。

模块设计思路

模块以实时体温数据为驱动，通过@StorageLink 全局共享数据，配合@Watch 生命周期回调函数监听温度变化。

1.当宠物体温处于正常区间时，宠物形象会随机进行行走、站立、趴卧等自主行为动画；

2.当体温超过阈值（≥38℃），系统立即停止所有运动逻辑，强制切换为休憩姿态，并叠加色彩滤镜与呼吸动画，实现健康异常可视化提醒。

关键技术实现

核心代码

全局体温监听，联动行为切换

```
@StorageLink('temp') @Watch('onTempChange') temp: number = 0;
onTempChange() {
  if (this.temp >= 38) {
```

温度超标，强制休息

```
    this.forceStopAndLay();
```

```
  } else {
```
正常温度，随机触发行走动画
```
    this.scheduleNextActionRandomly();
  }
}
```
组件通过 animateTo 实现平滑过渡动画，结合定时器控制行走节奏与随机休息间隔；
同时在 aboutToDisappear 生命周期内统一清空定时器，防止内存泄漏，保障应用稳定运行。
功能特点
1. 数据驱动 UI
体温数据实时绑定界面，实现硬件感知数据与前端行为联动。
2. 轻量化动画实现
采用原生动画能力开发，无第三方动画 SDK 依赖，符合系统轻量化设计。
3. 边界与行为管控
限制形象移动边界，自动控制行走、转向、休憩完整行为逻辑。
异常视觉预警
高温状态下自动启用颜色滤镜与循环缩放动画，直观提示健康风险。
6.2.3 悬浮信息面板模块（InfoFloatingPanel.ets）
模块功能概述
悬浮信息面板为首页可视化辅助组件，采用 ArkUI 声明式语法开发，实时展示设备关键运行状态与系统时间，依托全局状态管理器实现数据自动响应刷新，界面采用毛玻璃模糊、半透明圆角样式，视觉轻量化且不遮挡主页面内容。
核心功能
1. 实时系统时钟秒级刷新，统一 24 小时制时间展示；
2. 实时同步展示设备剩余电量百分比；
3. 联动电子围栏状态，动态显示「安全/越界」文字并搭配警示色标；
4. 借助@StorageProp、@StorageLink 全局状态绑定，无需手动请求即可自动接收后台最新数据；
5. 组件生命周期自动管理定时器，页面销毁时及时释放定时任务，避免资源占用。
核心代码
全局状态绑定
```
@StorageProp('battery') battery: number = 0;
@StorageLink('isInsideFence') isInsideFence: boolean = true;
```
实时时钟定时刷新
```
aboutToAppear() {
  this.timer = setInterval(() => {
    let date = new Date();
    this.currentTime = date.toLocaleTimeString('zh-CN', { hour12:
```

```
false });
  }, 1000);
}
```

销毁定时器，避免内存泄漏

```
aboutToDisappear() {
  clearInterval(this.timer);
}
```

围栏状态动态文字与警示颜色

```
Text(this.isInsideFence ? "安全" : "越界")
  .fontColor(this.isInsideFence ? '#FFFFFF' : '#FF4D4F')
```

模块设计特点

组件采用生命周期回调函数完成定时任务的创建与销毁，有效防止内存泄漏；通过全局持久化状态装饰器绑定业务数据，实现视图数据单向自动更新；界面搭配磨砂模糊、半透明边框与分层文字配色，兼顾实用性与界面美观性，可实时直观反馈宠物设备核心状态。

6.2.4 天气联动功能模块

模块功能概述

本系统天气功能采用数据与视图分离的解耦设计，整体由天气数据卡片（WeatherFloatCard）与动态天气背景（WeatherBackground）两部分协同构成。数据组件负责网络请求、气象数据更新与全局状态存储，视图组件统一订阅全局天气状态、自动完成界面动态渲染与背景切换。两组件基于全局状态通信，避免重复请求与重复监听，保证逻辑单一、职责清晰、联动高效。

模块整体工作逻辑

1. 数据生产层：WeatherFloatCard 定时调用高德天气接口，获取城市、温度、湿度、天气状况等数据，解析后统一存入全局持久化状态。
2. 数据消费层：WeatherBackground 监听全局天气字段变化，根据最新天气类型自动切换晴天、多云、雨天三套背景样式。
3. 联动机制：所有天气数据仅网络请求一次，全局统一分发，视图被动响应更新，有效降低资源消耗，避免代码逻辑冗余重复。

天气数据业务层（WeatherFloatCard.ets）

该组件为天气数据源核心，主要实现城市配置读取、网络请求、数据解析、定时更新、全局存储功能。组件在页面载入后开启定时任务，周期性刷新气象信息，并支持全局事件主动刷新，页面销毁时及时关闭定时器与事件监听，保证资源合理释放。

核心代码

```
 async loadCityAndFetch() {
  this.currentCityCode = await ConfigManager.getCityCode(this.context);
  await this.getRealtimeWeather();
}
async getRealtimeWeather() {
```

```
  let httpRequest = http.createHttp();
  let url =
`https://restapi.amap.com/v3/weather/weatherInfo?city=${this.currentCityCode}`;
  let res = await httpRequest.request(url);
  let data = JSON.parse(res.result as string);
```

写入全局共享状态

```
  AppStorage.setOrCreate('currentWeather', data.lives[0].weather);
  httpRequest.destroy();
}
```

天气动态视图层（WeatherFloatCard.ets）

该组件为纯视图渲染模块，不参与任何网络请求，仅通过监听全局天气状态变化实现界面联动。根据天气字段自动匹配晴天、阴天、雨天三种视觉方案，结合视频控制器与渐入渐出动画，实现背景平滑切换，同时精准管控媒体资源启停，降低应用功耗。

核心代码

单向订阅全局天气状态，实现数据驱动视图

```
@StorageProp('currentWeather') currentWeather: string = '晴';
@Watch('onWeatherChange') weatherWatch: string = '晴';
videoController: VideoController = new VideoController();
```

天气状态变更回调，动态切换背景效果

```
onWeatherChange() {
  const isRain = this.currentWeather.includes('雨');
  const isCloudy = this.currentWeather.includes('云') ||
this.currentWeather.includes('阴');
  const isSunny = !isRain && !isCloudy;
```

雨天开启视频，其余场景暂停视频释放资源

```
  isRain ? this.videoController.start() : this.videoController.pause();
```

三场景透明度过渡动画

```
  animateTo({ duration: 3000, curve: Curve.EaseInOut }, () => {
    this.videoOpacity = isRain ? 1 : 0;
    this.cloudOpacity = isCloudy ? 1 : 0;
    this.sunOpacity = isSunny ? 1 : 0;
  })
}
```

功能特点

1．职责完全解耦：数据组件专职网络更新，视图组件专职界面渲染，不存在重复逻辑、不存在重复监听。
2．全局统一数据源：天气数据只请求一次、全局共享，避免多次网络请求造成流量与性能浪费。
3．资源管理规范：通过生命周期回调统一管理定时器、事件监听、视频媒体资源，杜绝内存泄漏。
4．视觉联动流畅：依托状态监听与动画插值，实现天气状态与背景效果实时同步，交互体验统一连贯。
5．扩展性强：后续新增天气类型、新增气象指标，仅需修改数据解析或视图判断，无需改动整体架构。

6.2.5 地图页面模块（MapPage.ets）

模块功能概述

地图页面为系统核心定位功能模块，依托 Web 组件加载离线地图网页，结合定位服务、权限申请、原生与网页双向交互，实现用户自身定位、宠物实时点位标记、地图资源拦截加载等功能，完整实现位置可视化监测需求。

模块说明

页面通过 Webview 容器嵌入本地地图静态网页，搭配位置监听服务持续获取设备定位信息；借助状态全局绑定监听宠物坐标变化，调用网页内部 JS 方法动态更新宠物标记点位。同时封装权限申请、资源拦截、生命周期资源释放逻辑，保障定位服务稳定运行与内存合理回收。

核心代码

绑定宠物位置全局状态并监听变化

```
@StorageLink('petPos') @Watch('onPetPosChange') petPos: number[] = [113.591664, 24.814131];
```

宠物点位更新

```
onPetPosChange() {
  this.controller.runJavaScript(`updatePetPos(${this.petPos[0]}, ${this.petPos[1]})`);
}
```

定位权限申请与定位监听开启

```
async requestLocationPermission() {
  await atManager.requestPermissionsFromUser(context, ['ohos.permission.LOCATION','ohos.permission.APPROXIMATELY_LOCATION']);
  this.startLocationListener();
}
```

原生调用 JS，同步用户当前位置

```
geoLocationManager.on('locationChange', requestInfo, (location) => {
this.controller.runJavaScript(`window.updateUserPos(${location.longitude}, ${location.latitude})`);
```

```
});
```

页面销毁，关闭定位监听、释放地图资源

```
aboutToDisappear() {
  geoLocationManager.off('locationChange');
}
```

功能特点

1. 采用原生 Web 混合开发方案，通过内置地图网页实现轻量化地图展示；
2. 使用状态监听机制，宠物坐标变动自动触发地图点位刷新；
3. 动态申请定位权限，合规调用系统位置服务，实时上报用户位置；
4. 支持原生与网页双向函数调用，完成点位数据互通；
5. 依托页面生命阶段回调，及时关闭定位监听、销毁地图实例，避免资源冗余。

6.2.6 宠物状态悬浮卡片模块（PetFloatCard.ets）

模块功能概述

该组件为首页右侧悬浮状态卡片，采用 ArkUI 声明式开发，通过全局状态绑定实时展示宠物健康与运动数据，同时实现体温异常动态告警，界面根据数据状态自动切换主题色，具备视觉反馈直观、不遮挡主界面的特点。

模块功能

1. 实时显示宠物体温，并根据温度自动切换文字颜色与卡片背景；
2. 根据运动状态（静止/活跃/跳跃）显示对应的状态标签；
3. 体温异常（≥38℃）时自动显示“高温”告警标识，并切换卡片警示色；
4. 展示当日步数、活跃/休息时长等运动统计数据；
5. 内置平滑过渡动画，状态变化时实现柔和视觉切换。

核心代码

全局状态绑定：体温、运动状态、步数

```
@StorageLink('temp') temp: number = 0;
@StorageLink('motionStatus') motionStatus: number = 0;
@StorageLink('stepCount') stepCount: number = 0;
```

运动状态映射函数

```
private getMotionText(): string {
  switch(this.motionStatus) {
    case 3: return "剧烈跳跃";
    case 1: return "活跃运动";
    default: return "静止小憩";
  }
}
build() {
  Column() {
```

体温与状态展示

```
    Text(`${this.temp.toFixed(1)}°C`)
      .fontColor(this.temp >= 38 ? '#FF4D4F' : '#FFFFFF')
```

异常告警条件渲染

```
    if (this.temp >= 38) {
      Text("⚠️ 高温").fontColor('#FF4D4F')
```

```
    }
```
运动数据统计
```
    Text(`今日步数: ${this.stepCount}`)
  }
```
动态主题色切换与平滑动画
```
  .backgroundColor(this.temp >= 38 ? 'rgba(180, 0, 0, 0.6)' : 'rgba(0, 0, 0, 0.4)')
  .animation({ duration: 400, curve: Curve.EaseInOut })
}
```
功能特点

组件通过@StorageLink 与全局状态管理器实现数据双向绑定，无需额外请求即可实时响应后台数据更新；同时通过条件渲染与主题色切换，实现了对异常状态的即时视觉提醒。卡片采用毛玻璃半透明样式与平滑过渡动画，既保证了信息的直观展示，又不影响主界面的地图交互体验。

6.2.7 电子围栏安全模块（Fence.ets）

模块功能概述

电子围栏模块是宠物安全管控的核心功能模块，采用 Web 混合地图架构，结合前后端双向交互、地理距离算法、本地轻量化持久化存储与消息发布机制，实现自定义围栏中心选址、围栏半径调节、实时位置越界判定、围栏参数本地缓存与云端同步。

模块支持地图点击选点、围栏范围动态绘制、宠物实时点位叠加展示，自动计算宠物与围栏中心点直线距离，实时判定是否越界；修改围栏参数后自动本地落盘，并通过通信服务上报围栏配置至终端设备，完成安全管控闭环。

核心业务逻辑

1．多状态全局数据监听

全局绑定围栏中心坐标、围栏半径、宠物实时位置、越界状态等核心数据，通过监听函数实时监听参数变动，自动同步更新地图围栏图层与宠物点位。

2．地理测距算法与越界判定

内置经纬度距离计算公式，基于球面距离算法实时计算宠物当前位置与围栏中心的直线距离，结合围栏半径自动判定「安全/越界」状态并全局同步。

3．围栏配置本地持久化

采用通用轻量键值对存储方案，实现围栏中心点、围栏半径参数本地缓存，应用重启自动加载历史配置，避免重复设置。

4．双端 JS 交互与地图联动

通过 JS 桥接对象实现应用层与本地地图网页双向通信，支持地图点击选点、编辑模式切换、围栏图形重绘、中心点重置等联动操作。

5．围栏配置云端下发同步

围栏修改完成后，封装标准化围栏配置报文，通过通信服务发布设备控制消息，将围栏参数下发至硬件终端，保证软硬件围栏规则一致。

核心代码

围栏配置上报数据结构

```
interface FenceUpdatePayload {
  device_id: string;
```

```
      fence_lng: number;
      fence_lat: number;
      fence_radius: number;
    }
    全局监听围栏、宠物位置数据变化
    @StorageLink('isInsideFence') isInsideFence: boolean = true;
    @StorageLink('fenceRadius')   @Watch('onFenceChange')   fenceRadius:
number = 20;
    @StorageLink('fenceCenter')   @Watch('onFenceChange')   fenceCenter:
number[] = [];
    @StorageLink('petPos') @Watch('onPetPosChange') petPos: number[] = [];
    经纬度球面距离计算，用于越界判断
    getDistance(pos1: number[], pos2: number[]): number {
      const R = 6378137;
      const dLat = (pos2[1] - pos1[1]) * Math.PI / 180;
      const dLng = (pos2[0] - pos1[0]) * Math.PI / 180;
    }
    判定宠物是否超出围栏范围
    checkFenceStatus() {
      const distance = this.getDistance(this.petPos, this.fenceCenter);
      const result = distance <= this.fenceRadius;
      AppStorage.setOrCreate('isInsideFence', result);
    }
    保存围栏配置并上报硬件终端
    async saveToPreferences() {
    本地持久化存储围栏参数
      let  pref  =  await  preferences.getPreferences(getContext(this),
'fence_store');
      await pref.put('radius', this.fenceRadius);
      await pref.put('center', this.fenceCenter.join(','));
      await pref.flush();
    封装报文，发布围栏更新指令
      const fencePayload: FenceUpdatePayload = {
        device_id: "MK",
```

```
        fence_lng: this.fenceCenter[0],
        fence_lat: this.fenceCenter[1],
        fence_radius: this.fenceRadius
      };
      globalManager.publish("app/fence/update",
JSON.stringify(fencePayload));
    }
```

功能特点

1．算法自主封装，独立性强

地理测距、越界判断逻辑完全自主实现，不依赖第三方地图分析接口，降低外部依赖，运行稳定性更高。

2．本地+云端双备份机制

围栏参数既做本地持久化缓存，又同步下发至 4G 硬件设备，软硬件规则统一，断网场景下仍可正常本地判定。

3．低耦合混合交互设计

通过 JS 桥接完成原生层与地图网页解耦通信，地图渲染、图形绘制交由 Web 端实现，业务逻辑由应用层管控，结构清晰易维护。

4．高兼容通用化设计

数据存储、消息通信、网络交互均采用通用标准化方案，无平台私有接口绑定，整体适配性、可移植性优秀。

6.2.8 设备详情模块（GlobalContext.ets）

模块功能概述

全局通信与数据管理模块是 App 与硬件终端、MQTT 消息服务交互的底层核心工具类。基于标准 TCP 套接字自主封装轻量化 MQTT 协议栈，不依赖第三方通信 SDK，自主实现连接握手、主题订阅、心跳保活、消息下发、断线重连等全链路能力；同时统一管理全局业务数据、解析硬件上行传感报文、持久化围栏配置、全局状态同步，为首页、健康监测、定位轨迹、电子围栏等所有业务页面提供统一的数据来源与双向通信能力。

核心业务逻辑

1．自主轻量化 MQTT 协议封装

基于基础 TCP 网络通信规范，手动拼接 MQTT 连接报文、订阅报文、心跳报文、发布报文，自主实现协议编解码，轻量化、体积小、无第三方依赖，无 SDK，兼容性强。

2．长连接稳定运维机制

内置定时心跳发送、断线自动重连、连接状态锁、重连延时策略，保障弱网、断连场景下通信链路自动恢复，提升软硬件通信稳定性。

3．硬件上行数据统一解析分发

监听 TCP 下行消息，截取标准 JSON 载荷，统一解析电池、体温、湿度、定位坐标、运动状态、围栏参数等硬件数据，批量写入全局统一状态容器，供全页面响应式刷新。

4．业务控制指令下行发布

封装通用消息发布方法，支持自定义主题与报文内容，可向下游硬件终端下

发围栏更新、设备同步、状态查询等控制指令，实现双向可控通信。

5．全局默认数据初始化+配置本地落盘

应用启动自动初始化全部健康、定位、围栏默认参数；围栏修改后自动写入本地轻量化存储，实现配置持久化不丢失。

核心代码

硬件上行传感数据通用载荷结构

```
export interface PetJsonPayload {
  battery?: number;
  temp?: number;
  humidity?: number;
  stepCount?: number;
  petPos?: number[];
  isInside?: boolean;
  motionStatus?: number;
  fenceCenter?: number[];
  fenceRadius?: number;
}
```

基于 TCP 自主封装 MQTT 长连接核心管理类

```
export class GlobalDataManager {
  private tcp: socket.TCPSocket | null = null;
  private isConnected: boolean = false;
  private reconnectTimer: number = -1;
```

初始化全局默认业务数据

```
  private initDefaultData(): void {
    AppStorage.setOrCreate('battery', 0);
    AppStorage.setOrCreate('temp', 0);
    AppStorage.setOrCreate('petPos', [113.591664, 24.814131]);
    AppStorage.setOrCreate('isInsideFence', true);
    AppStorage.setOrCreate('fenceRadius', 20);
  }
```

自主拼接 MQTT 订阅报文，实现主题订阅

```
  private sendSubscribePacket(): void;
```

定时心跳保活，维持长连接

```
  private sendPing(): void;
```

建立 TCP 连接 + MQTT 握手认证

```
  private executeConnect(): void;
```

统一解析硬件上报 JSON 数据，同步至全局状态

```
  private processRawData(raw: PetJsonPayload): void {
    if (raw.temp !== undefined) AppStorage.setOrCreate('temp', Number(raw.temp));
    if (raw.petPos !== undefined) AppStorage.setOrCreate('petPos', raw.petPos);
    if (raw.isInside !== undefined) AppStorage.setOrCreate('isInsideFence', raw.isInside);
  }
```

通用消息发布，向下发控制指令

```
  public publish(topic: string, message: string): void;
```

断线自动重连策略

```
  private triggerReconnect(): void;
}
```

全局单例统一导出，全项目共用

```
export const globalManager: GlobalDataManager = new GlobalDataManager();
```

功能特点

1．通信协议栈，轻量化无依赖

完全基于基础通用网络接口自主实现 MQTT 交互，不引入第三方通信框架，安装包体积占用小，规避第三方组件兼容问题。

2．高稳定长连接机制

集成心跳检测、异常捕获、自动重连、状态锁机制，有效避免频繁断连、消息丢失，适配户外 4G 弱网运行环境。

3．全域统一数据中枢

集中接收硬件所有传感、定位、运动、告警数据，统一分发至各业务页面，数据流转规范，减少重复请求与冗余逻辑。

4．软硬件双向联动闭环

既接收硬件上行监测数据，又支持业务层主动下发配置与控制指令，实现电子围栏、设备同步、状态管控等双向业务闭环。

5．通用化底层设计

网络通信、数据解析、本地存储均采用标准化通用方案，剥离平台专属私有能力限制，整体可移植性、适配性优异。

6.2.9 轨迹监测与地图模块（TrackPage.ets）

模块功能概述

历史轨迹定位模块基于 Web 混合渲染方案实现地图可视化能力，通过内嵌本地 HTML 地图页面，结合双向 JS 交互、通用 HTTP 网络请求、全局位置数据监听，完成宠物实时定位展示与历史轨迹回放功能。

模块独立封装地图控制器，实时监听设备定位坐标变化，主动向网页注入最新经纬度；同时请求云端轨迹接口，拉取历史点位数据并绘制完整运动路线，支持轨迹数据加载与行程动画回放，实现位置监控全流程业务闭环。

核心业务逻辑

1．全局定位数据双向监听

通过全局状态绑定实时接收终端上报的经纬度坐标，搭配数据监听回调，坐标更新后自动调用地图交互方法，实时刷新宠物标记点位。

2．双端 JS 双向通信

依托 Web 组件控制器，实现应用端与本地地图网页的方法互调：

应用主动调用网页内 JS 函数，完成点位更新、轨迹绘制、回放控制，跨端交互解耦、拓展性强。

3．云端历史轨迹拉取

采用标准通用 HTTP 请求访问后端轨迹接口，获取时序化定位点位集合，解析 JSON 结构化数据后批量传入地图层，绘制连续行走轨迹路线。

4．轻量化混合架构

地图能力依托静态本地网页实现，不依赖重型地图 SDK，降低安装包体积，整体架构轻量化、兼容性更强。

核心代码

轨迹点位数据结构定义

```
interface TrackPoint {
  lng: number;
  lat: number;
  time: string;
  battery: string;
}
```

监听全局定位坐标变化，实时同步地图

```
@StorageLink('petPos') @Watch('onPosChange') petPos: number[] = [113.591664, 24.814131];
onPosChange() {
  if (this.petPos.length === 2) {
    const lng = this.petPos[0];
    const lat = this.petPos[1];
```

调用网页 JS 方法更新宠物实时位置

```
    this.controller.runJavaScript(`updatePetLocation(${lng}, ${lat})`);
  }
}
```

请求云端接口，加载历史轨迹数据

```
    loadAdvancedData() {
      let httpRequest = http.createHttp();
      const url = "https://energyelf.shop/get_track.php";
      httpRequest.request(url, { method: http.RequestMethod.GET }, (err,
data) => {
        if (!err) {
          const  trackData:  TrackPoint[]  =  JSON.parse(data.result  as
string);
          const jsonStr = JSON.stringify(trackData);

this.controller.runJavaScript(`drawAdvancedTrack('${jsonStr}')`);
        }
        httpRequest.destroy();
      });
    }
```

功能特点

1．混合开发、轻量化部署

采用内嵌静态网页承载地图能力，无需引入第三方地图 SDK，大幅减少资源占用，契合项目轻量化设计目标。

2．跨端双向交互灵活

通过原生与 Web 端 JS 函数互调，将定位更新、轨迹绘制、动画控制解耦，业务逻辑清晰，便于后期维护。

3．实时定位+历史轨迹双融合

同时支持实时位置监控与长时序历史路线回溯，满足日常围栏管控、行动轨迹追溯双重使用场景。

4．通用化网络设计

基于标准 HTTP 协议完成云端数据交互，无平台专属接口绑定，整体适配性、可移植性优异。

6.2.10 健康监测模块（HealthPage.ets）

模块功能概述

健康总览主页为宠物设备核心数据展示页面，集中整合设备终端采集数据与云端气象环境数据，依托全局状态绑定实现多维度健康指标实时联动刷新。页面采用分层卡片式布局、渐变主题配色、动态状态识别与轻量化网络请求设计，集中展示宠物体温、运动行为、环境气象、设备电量等关键信息，同时提供健康档案入口与数据说明弹窗，实现健康状态一站式可视化管理。

模块核心设计

本模块采用本地终端数据 + 云端天气数据双数据源融合方案：设备实时上传的体征、运动、电量数据通过全局状态全局共享；环境温湿度、城市天气通过轻量化 HTTP 接口按需拉取，避免重复请求，降低网络与性能开销。

页面根据宠物体温阈值、运动状态编码自动匹配对应主题色与状态描述，结

合自定义渐变容器、数据环形面板、弹窗提示，提升信息辨识度与交互体验。

核心代码

全局绑定终端实时健康、运动、设备状态数据

```
@StorageLink('temp') temp: number = 32.5;
@StorageLink('stepCount') stepCount: number = 1644;
@StorageLink('motionStatus') motionStatus: number = 3;
@StorageLink('battery') battery: number = 85;
```

页面载入拉取云端城市气象数据

```
async fetchWeather() {
  let cityCode = await ConfigManager.getCityCode(this.context);
  let url = `https://restapi.amap.com/v3/weather/weatherInfo?city=${cityCode}`;
  let resp = await http.createHttp().request(url);
  let result = JSON.parse(resp.result as string) as AmapWeatherResponse;
```

环境气象数据写入全局状态共享

```
  AppStorage.setOrCreate('currentTemp', result.lives[0].temperature);
}
```

运动状态自动判定与动态主题配色

```
private getStatusConfig() {
  if (this.temp >= 38) return { text: "温度警报", color: "#FF4D4F" };
  if (this.motionStatus === 3) return { text: "剧烈跳跃", color: "#FF4500" };
  return { text: "静止小憩", color: "#1E90FF" };
}
```

计算当日活动占比，量化宠物活跃程度

```
private getActiveRatio() {
  let total = this.moveDuration + this.restDuration;
  return total > 0 ? (this.moveDuration / total) * 100 : 0;
}
```

功能特点

1. 双数据源融合展示

同时接入终端硬件采集的体征数据与高德开放平台环境气象数据，区分宠物本体指标与外界环境指标，数据维度更加完整。

2. 智能状态分级识别

通过体温阈值、运动状态编码进行逻辑判断，自动切换正常、高温预警、剧烈运动、静止休息四种状态，搭配差异化渐变配色，异常状态视觉强提醒。

3. 轻量化网络请求管控

仅在页面初始化时单次请求天气接口，复用本地城市配置，请求结束及时销毁网络实例，减少长连接占用，网络资源开销降低约 30%。

4. 数据量化计算

自动计算活跃时长占比，以环形面板直观展示运动强度，将碎片化时长数据转化为百分比指标，健康评估更加量化直观。

5. 轻量化交互辅助

内置说明弹窗、页面路由跳转、状态标识标签，对专业指标进行通俗解释，

降低用户理解门槛；统一卡片圆角、阴影、渐变风格，界面整体性强。

6. 全局状态高效复用

所有核心数据基于全局状态双向绑定，无需重复通信请求，页面渲染效率提升，整机后台资源占用有效控制，适配低功耗运行场景。

6.2.11 单日健康详情可视化模块（HealthDetailPage.ets）

模块功能概述

该图表组件为健康详情页核心可视化模块，基于 ArkUI 原生 Canvas 画布实现自定义绘制，集成温度曲线、步数柱状图、活动率面积图多指标合一展示，支持指尖点击选中、悬浮提示、辅助虚线定位，将小时级碎片化健康数据图形化呈现，直观反映宠物单日生理指标变化趋势。

模块设计说明

组件采用画布自定义渲染方案，不依赖第三方图表库，轻量化、低功耗。通过 @Prop 接收上层页面传入的小时级监测数组，搭配 @Watch 数据监听，数据更新自动重绘图表；绑定触摸事件实现交互选点，动态弹出信息气泡，分时展示时刻、体温、步数、活动率四项核心指标。

统一坐标轴归一化换算，对体温、步数、活动率做区间映射，保证多类型数据在同一画布合理布局；封装独立绘图、气泡提示方法，结构清晰，复用性强。

核心代码

```
 @Component
struct MultiLineChart {
接收上层小时级健康数据，数据变更自动重绘
  @Prop @Watch('drawChart') data: HourlyMetric[] = [];
  private       context:       CanvasRenderingContext2D       =       new
CanvasRenderingContext2D(new RenderingContextSettings(true));
  @State selectedIndex: number = -1;
画布就绪 & 数据更新触发重绘
  Canvas(this.context)
    .onReady(() => { if (this.data.length > 0) this.drawChart(); })
触摸选点，实现点击高亮与提示
  .onTouch((event: TouchEvent) => {
    let x = event.touches[0].x;
    let stepX = (this.context.width - 40) / (this.data.length - 1);
    this.selectedIndex = Math.round((x - 20) / stepX);
    this.drawChart();
  })
混合绘制：活动率面积、步数柱状、体温曲线
  private drawChart() {
坐标系换算、画布清空
绘制活动率背景渐变面积图
循环绘制步数柱状图
绘制体温折线与节点圆点
选中点位渲染虚线与信息气泡
```

```
  }
}
```

功能特点

1. 多指标融合可视化

同一画布叠加绘制面积图、柱状图、折线图，同时承载活动率、行走步数、体温三类数据，信息密度高、展示直观。

2. 数据驱动自动刷新

通过自定义属性监听，上层健康详情页数据加载完成后，图表自动重绘，实现视图与数据联动同步。

3. 轻量化原生渲染

本图表组件采用鸿蒙系统原生 Canvas API 实现，无第三方图表库依赖，将页面额外资源占用率控制在 8% 以内，相比引入开源图表方案，内存开销降低约35%，显著降低应用运行功耗与后台资源占用，适配鸿蒙移动端的低功耗运行需求。

4. 交互式数据查询

支持指尖单点选中，自动生成纵向辅助虚线与深色气泡提示，精准查看单小时详细健康参数。

5. 异常视觉标识

高温数据采用红色文字标红提示，正常体温使用绿色配色，通过色彩快速区分生理异常状态。

6. 边界自适应处理

气泡提示自动判断画布边界，避免超出屏幕显示异常，适配不同屏幕尺寸，兼容性良好。

6.2.12 健康历史记录模块（HealthHistoryPage.ets）

模块功能概述

健康档案分析模块为系统健康数据统计核心页面，依托后端服务接口实现宠物历史健康数据拉取、指标统计、趋势分析与异常预警展示。页面采用数据请求+统计计算+可视化卡片的设计思路，整合网络数据解析、自动均值运算、多维度指标展示与历史记录列表，搭配弹窗算法说明，完整实现宠物长期健康状态量化分析。

该模块通过请求服务端健康数据接口，批量获取每日健康档案记录与整体统计指标；前端自动完成健康均分计算、数据分类渲染，划分综合指数、日均步数、发热异常、趋势预警等统计维度。同时集成历史记录点击跳转、自定义弹窗、状态标签分级配色，直观区分正常、发热、活跃度异常等不同健康状态，为饲养者提供数据化健康参考。

核心代码

定义健康数据接口类型，规范前后端数据结构

```
interface HealthRecord {
  date: string; temp: number; stepCount: number; status: string;
  score: number; warningType: string; warningMsg: string;
}
```

页面加载自动请求后端接口

```
aboutToAppear() {
  this.loadHistoryData();
```

```
    }

    网络请求获取健康统计与历史记录
    async loadHistoryData() {
      let httpRequest = http.createHttp();
      let res = await
httpRequest.request("http://energyelf.shop/health_api.php?action=get_da
ily_list");
      let result = JSON.parse(res.result as string) as HealthApiResponse;
    赋值统计数据与历史列表
      this.stats = result.stats;
      this.recordList = result.list;
    自动计算近 30 日平均健康分数
      let sum = 0;
      result.list.forEach(item => sum += item.score);
      this.avgScore = Math.floor(sum / result.list.length);
      httpRequest.destroy();
    }
```

功能特点

1. 标准化数据交互

预先定义健康记录、统计指标、接口返回结构体，统一前后端数据字段规范，提升数据解析稳定性，避免字段错乱导致页面渲染异常。

2. 服务端数据联动

通过 HTTP 网络请求访问后端专属健康接口，批量拉取长期历史数据，所有统计运算、风险判定逻辑由服务端完成，前端仅负责渲染展示，降低客户端运算压力。

3. 本地二次统计计算

在前端完成健康综合得分加权均值计算，结合多维度统计卡片，将抽象健康数据转化为可视化指标，数据呈现更直观。

4. 分级视觉警示设计

根据发热、活跃度异常、趋势风险等不同预警类型，搭配差异化文字颜色与边框标识，实现健康状态分层提醒，快速识别异常记录。

5. 算法透明化设计

内置自定义弹窗组件，展示健康评分、趋势预警、风险预测、发热判定的核心计算逻辑，完善项目设计完整性，提升模块专业性。

6. 生命周期资源管控

页面销毁时自动释放网络请求实例，杜绝网络资源冗余占用，保证应用长期运行稳定。

6.2.13 消息通知模块（NotificationUtils.ts）

模块功能概述

消息通知模块基于鸿蒙系统通知能力开发，用于实现体温异常、围栏越界、围栏回归等系统级告警推送。模块采用单例工具类设计，低耦合、全局可调用，通过时间戳防抖避免重复通知，并支持点击通知拉起应用。

核心代码

体温异常告警

```
static checkTemperature(temp: number) {
  if (temp >= 39) {
    this.publishTemperatureAlarm(temp);
  }
}
```

围栏越界与回归检测

```
static checkFenceStatus(isInside: boolean) {
  if (!isInside) {
    this.publishFenceAlarm();越界告警
  } else {
    this.publishFenceBackAlarm();回到安全区
  }
}
```

发布系统通知

```
static async publishTemperatureAlarm(temp: number) {
  let notificationRequest = {
    title: '宠物体温过高告警',
    text: `当前体温：${temp}℃`
  };
  await notificationManager.publish(notificationRequest);
}
```

功能特点

1. 体温异常告警：当温度≥39℃时触发系统通知，30 秒内不重复提醒。
2. 围栏越界告警：宠物离开安全区域时推送横幅通知。
3. 围栏回归通知：宠物返回安全区后自动解除警报。
4. 权限申请：动态申请系统通知权限，保证送达。
5. 点击跳转：点击通知可直接打开 APP 查看宠物状态。

6.3 软件整体设计特点

本项目整套移动端软件系统采用完全模块化、组件化、分层化的开发思想，所有功能模块独立解耦、职责清晰，代码结构规整、逻辑规范，具备极高的可维护性与功能拓展性。

软件融合 MQTT 长连接通信（基于 TCP 协议）、HTTP 云端数据交互、Web 混合地图渲染、本地数据持久化存储、全局事件联动、动态数据绑定等多种现代化开发技术，技术体系完整、功能覆盖全面。

软件针对空数据、网络异常、数据解析失败、状态丢失等各类异常场景做了完整的容错处理与异常捕获，有效避免闪退、卡死、功能失效等问题，系统运行

稳定可靠。

通过全局状态绑定、页面路由传参与系统事件分发机制，实现全软件数据联动同步，保证设备 GNSS 定位数据、健康数据、配置数据的全局一致性与实时性，完全满足智能宠物监测系统实时监测、数据留存、安全预警、轨迹追溯的整体业务需求。

7、系统风险与应对措施

7.1 硬件采集风险与应对

硬件终端长期跟随宠物在户外、复杂环境下佩戴使用，传感器易受环境遮挡、肢体震动、电磁干扰等因素影响，进而产生数据突变、采集漂移、数值偏差等异常问题，影响监测准确性。

应对措施：

系统内置数据滤波算法与合理阈值过滤机制，自动识别并剔除超出正常区间的突变数据，避免异常数值刷新界面或触发误报警；增设设备低电量实时检测逻辑，当终端电量不足时主动推送提醒，引导用户及时充电，防止设备断电离线、数据中断。

7.2 网络通信风险与应对

本系统硬件终端采用 4G 蜂窝网络进行数据传输，长期户外使用易受建筑物遮挡、复杂地形、信号盲区等因素影响，容易出现网络波动、长连接断开、数据丢包、传输延迟、报文解析异常等通信问题。

应对措施：

系统配置定时心跳保活机制，周期性检测 MQTT 通信链路连通状态；监听网络异常状态，触发自动断线重连逻辑，实现链路自愈，无需手动操作；严格校验上行、下行报文格式与数据长度，增加容错解析规则，过滤乱码、残缺数据包，防止因数据解析错误造成功能卡死或程序异常退出。

7.3 软件运行异常风险与应对

软件多模块并行接收、刷新监测数据，高频数据更新容易引发数据覆盖冲突、设备状态错乱、页面加载卡顿、本地配置损坏丢失等运行隐患，降低使用体验。

应对措施：

采用中心化全局数据管理模式，统一调度与分发业务数据，减少数据冲突；增设数据刷新限流机制，限制高频重复刷新请求，优化页面渲染性能；本地配置读取增加异常判断，若配置文件损坏或读取失败，自动恢复默认参数；全局捕获代码运行异常，有效规避 APP 闪退、崩溃问题，全面提升软件运行稳定性与健壮性。

7.4 定位与第三方接口风险与应对

户外环境中，高楼建筑、植被遮挡、复杂地形会干扰 GNSS 信号接收，造成定位漂移、点位跳动、位置偏移等问题；同时，第三方天气 API 接口存在网络超时、请求失败、接口限流、返回空数据等不稳定风险。

应对措施：

针对 GNSS 定位异常，增加距离阈值过滤与轨迹平滑优化处理，过滤漂移点

位，减少电子围栏误判、越界误报警问题；针对第三方接口请求异常，采用本地缓存兜底方案，接口请求失败或超时无法获取新数据时，自动沿用上一次有效数据展示，避免页面空白、功能失效，保障系统核心监测功能持续稳定运行。

8、数据库设计

8.1 概述

本系统采用 MySQL 关系型数据库，库名 petmonitor，用于统一存储宠物实时状态、运动数据、环境监测数据、位置轨迹、电子围栏配置及小时级/日级健康统计数据。数据由 Python 服务实时采集计算并写入，通过 PHP 接口提供查询服务，支撑实时监测、健康分析、历史档案、运动轨迹等核心业务。
系统采用“设备上报 → 脚本处理 → 数据库存储 → EMQX 下发”的数据流模式：
1. 硬件终端将原始传感数据通过 MQTT 上传至服务器；
2. 主脚本接收数据，完成校验处理后，写入 MySQL 数据库；
3. EMQX 服务端轮询数据库，获取最新状态数据；
4. EMQX 将处理后的 JSON 报文通过指定主题广播至订阅的 App 客户端，实现实时更新。

8.2 数据表设计

8.2.1 宠物实时状态表（pet_status）

核心表，存储经过脚本处理后的最新状态数据，是 EMQX 下发广播的数据源。

```
CREATE TABLE `pet_status` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `device_id` varchar(50) NOT NULL COMMENT '设备唯一 ID',
  `battery` tinyint(4) DEFAULT '0' COMMENT '电量',
  `temp` decimal(4,1) DEFAULT '0.0' COMMENT '宠物体温',
  `humidity` decimal(4,1) DEFAULT '0.0' COMMENT '环境湿度',
  `pet_lng` decimal(10,7) DEFAULT '113.6185420' COMMENT '当前经度',
  `pet_lat` decimal(10,7) DEFAULT '24.8223060' COMMENT '当前纬度',
  `is_inside` tinyint(1) DEFAULT '1' COMMENT '是否在围栏内',
  `motion_status` tinyint(4) DEFAULT '2' COMMENT '运动状态 1 运动 2 静止 3
```

```
跳跃',
  `step_count` int(11) DEFAULT '0' COMMENT '当日步数',
  `jump_count` int(11) DEFAULT '0' COMMENT '跳跃次数',
  `move_duration` float DEFAULT '0' COMMENT '运动时长(分钟)',
  `rest_duration` float DEFAULT '0' COMMENT '休息时长(分钟)',
  `last_update` datetime DEFAULT CURRENT_TIMESTAMP COMMENT '最后更新时间',
  PRIMARY KEY (`id`),
  UNIQUE KEY `device_id` (`device_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

8.2.2 电子围栏配置表（pet_config）

存储用户设置的围栏参数，服务端用于判断宠物是否越界。

```
CREATE TABLE `pet_config` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `device_id` varchar(50) NOT NULL COMMENT '设备 ID',
  `fence_lng` decimal(10,7) DEFAULT '113.6175420' COMMENT '围栏中心经度',
  `fence_lat` decimal(10,7) DEFAULT '24.8223060' COMMENT '围栏中心纬度',
  `fence_radius` int(11) DEFAULT '20' COMMENT '围栏半径(米)',
  `updated_at` timestamp NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP COMMENT '更新时间',
  PRIMARY KEY (`id`),
  UNIQUE KEY `device_id` (`device_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

8.2.3 历史定位轨迹表（pet_data_history）

存储宠物所有历史 GPS 点，供轨迹回放功能使用。

```
CREATE TABLE `pet_data_history` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `device_id` varchar(50) DEFAULT NULL COMMENT '设备 ID',
  `longitude` decimal(11,8) DEFAULT NULL COMMENT '经度',
  `latitude` decimal(11,8) DEFAULT NULL COMMENT '纬度',
  `battery` int(11) DEFAULT NULL COMMENT '电量',
  `created_at` timestamp NULL DEFAULT CURRENT_TIMESTAMP COMMENT '记录时间
',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

8.2.4 小时级统计数据表（pet_hourly_stats）

主脚本定时聚合生成的中间表，用于计算每小时的平均体温、总步数等。

```
CREATE TABLE `pet_hourly_stats` (
  `device_id` varchar(50) NOT NULL,
  `record_hour` datetime NOT NULL COMMENT '统计小时（整点）',
  `step_count` int(11) DEFAULT 0 COMMENT '小时步数',
  `jump_count` int(11) DEFAULT 0 COMMENT '小时跳跃次数',
  `move_duration` float DEFAULT 0 COMMENT '运动时长',
```

```
  `rest_duration` float DEFAULT 0 COMMENT '休息时长',
  `temp_sum` float DEFAULT 0 COMMENT '体温总和',
  `temp_count` int(11) DEFAULT 0 COMMENT '体温采样次数',
  `avg_temp` float DEFAULT 0 COMMENT '小时平均体温',
  PRIMARY KEY (`device_id`,`record_hour`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

8.2.5 小时级健康指标表（health_hourly_metrics）

为 App 健康详情页的 24 小时图表提供数据，由 pet_hourly_stats 聚合而来。

```
CREATE TABLE `health_hourly_metrics` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `record_date` date DEFAULT NULL COMMENT '统计日期',
  `record_hour` int(11) DEFAULT NULL COMMENT '统计小时',
  `avg_temp` decimal(4,2) DEFAULT NULL COMMENT '平均体温',
  `steps` int(11) DEFAULT '0' COMMENT '小时步数',
  `active_ratio` int(11) DEFAULT '0' COMMENT '活跃度百分比',
  PRIMARY KEY (`id`),
  UNIQUE KEY `date_hour_idx` (`record_date`,`record_hour`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

8.2.6 每日健康汇总表（health_daily_summary）

按天汇总健康数据，生成健康评分与状态，供健康历史页面使用。

```
CREATE TABLE `health_daily_summary` (
  `date` date NOT NULL COMMENT '统计日期',
  `avg_temp` decimal(4,2) DEFAULT NULL COMMENT '日均体温',
  `total_steps` int(11) DEFAULT '0' COMMENT '当日总步数',
  `health_score` int(11) DEFAULT '100' COMMENT '健康评分',
  `status` varchar(50) DEFAULT '健康' COMMENT '健康状态',
  `update_time` timestamp NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP COMMENT '更新时间',
  PRIMARY KEY (`date`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

8.3 核心数据流与业务交互

8.3.1 数据处理流程

1. 设备通过 MQTT 上传原始数据到私有服务器上的 EMQX 服务。
2. 主处理脚本订阅主题并接收数据，完成解析、计算步数、判断运动状态、围栏越界检测。
3. 处理后数据写入 pet_status（最新状态）与 pet_data_history（轨迹）。
4. 定时统计数据写入 pet_hourly_stats 与 health_hourly_metrics。
5. EMQX 从 pet_status 读取最新状态，封装为 JSON 并通过 MQTT 主题下发。
6. App 订阅主题接收报文，实时刷新界面显示。

8.3.2 EMQX 与数据库交互逻辑

EMQX 通过规则引擎或外部脚本，定时执行 SQL 查询获取 pet_statu 最新数据。查询结果被封装为标准 JSON 格式，例如：

```
{
  "device_id": "MK",
  "battery": 85,
  "temp": 36.5,
  "pet_lng": 114.06,
  "pet_lat": 22.548,
  "is_inside": 1,
  "step_count": 1644
}
```

报文通过 device/status 主题下发，App 客户端订阅该主题实现实时状态同步。

8.4 核心业务 SQL 语句

8.4.1 获取近 30 天健康档案数据

```
SELECT
  record_date AS date,
  ROUND(AVG(avg_temp), 1) AS temp,
  SUM(steps) AS stepCount,
  COUNT(CASE WHEN avg_temp > 39.1 THEN 1 END) AS feverHours
FROM health_hourly_metrics
GROUP BY record_date
ORDER BY record_date DESC
LIMIT 30;
```

8.4.2 获取指定日期小时级健康明细

```
 SELECT
  record_hour AS hour,
  avg_temp AS temp,
  steps,
  active_ratio AS activeRatio
FROM health_hourly_metrics
WHERE record_date = ?
ORDER BY record_hour ASC;
```

8.4.3 获取宠物历史轨迹

```
 SELECT
  longitude AS lng,
  latitude AS lat,
```

```
  created_at AS time,
  battery
FROM pet_data_history
WHERE device_id = ?
ORDER BY created_at DESC
LIMIT 100;
```

8.4.4 每日健康数据自动汇总计算

```
SELECT
  AVG(avg_temp) AS day_avg_temp,
  SUM(steps) AS day_total_steps
FROM health_hourly_metrics
WHERE record_date = ?
```

8.5 数据库设计特点

1. 全部部署于私有租用服务器，数据独立、安全、公网访问稳定。
2. EMQX + MySQL + Python 主脚本构成完整稳定的物联网架构。
3. 真实闭环数据流：设备—服务—数据库—App。
4. 实时表与统计表分离，查询高效，不影响前端流畅性。
5. 经纬度高精度存储，定位与围栏判断准确可靠。
6. 支持步数计算、行为分析、围栏告警、健康统计、轨迹回放全业务功能。
7. 单设备设计结构清晰，可平滑扩展多设备管理。

9、系统测试与运行结果

9.1 测试目的

系统开发完成后，为验证宠物智能监测系统的硬件稳定性、软件功能完整性、数据传输可靠性，对本系统所有核心功能进行整体测试。通过实际佩戴调试、联机运行、场景模拟测试，检验设备采集精度、数据通信稳定性、APP 交互效果及异常预警逻辑是否达到项目设计目标，保证系统整体运行稳定、功能可用、满足实际宠物户外监测使用需求。

9.2 测试环境

9.2.1 硬件环境

主控设备：Hi3863 物联网开发板
外设模块：GNSS 定位模块、高精度体温采集模块、运动姿态传感模块
搭载结构：宠物专用佩戴结构
供电方式：可充电锂电池稳压供电

9.2.2 软件环境

移动端：鸿蒙智能设备
网络环境：4G 蜂窝网络
数据服务：云端数据存储服务、第三方天气 API 数据接口

调试方式：硬件串口打印调试、云端数据库后台数据查看、APP 界面实时反馈调试

9.3 功能测试内容与结果

9.3.1 定位与轨迹功能测试

将设备佩戴于宠物身上，设备上电联网后可正常获取 GNSS 经纬度信息，移动端地图可实时刷新宠物当前位置。设备能够持续记录行走点位，可正常生成运动轨迹，支持历史轨迹回放与活动范围查看。电子围栏功能正常生效，宠物离开预设安全区域后，设备可及时触发越界预警。

测试结果： 功能正常，定位稳定、轨迹记录完整、围栏预警灵敏。

9.3.2 宠物健康监测测试

系统通过硬件传感器持续采集宠物体温数据与运动状态数据，设备采集频率稳定，数据上传无丢失、无乱码。移动端可实时展示当前体温、运动状态变化，能够有效区分宠物静止、正常活动、剧烈躁动等不同状态，数据更新及时准确。

测试结果： 健康监测功能运行正常，数据采集稳定，满足日常健康监测需求。

9.3.3 环境温度监测测试

系统依托设备实时定位坐标，成功调用第三方天气 API 接口，可正常获取当前区域的环境温度数据，并在移动端实时展示。系统可正常识别高温、低温等环境状态，为宠物户外环境安全监测提供有效数据支撑。

测试结果： 环境温度数据获取正常、更新及时，功能运行稳定。

9.3.4 智能异常报警测试

本次测试模拟多种异常场景，包括宠物围栏越界、体温异常偏高/偏低、环境温度超标、宠物长时间静止、宠物异常躁动等情况。系统均可准确识别异常状态，并快速推送异常提醒至移动端，报警触发逻辑准确、无误报、无漏报。

测试结果： 多场景预警功能全部正常，响应速度快、识别准确率高。

9.3.5 数据存储与记录测试

系统云端数据库可正常接收、存储设备上传的定位数据、健康数据、接口环境数据及所有报警日志。移动端可正常查询历史数据、查看状态变化趋势，所有历史记录保存完整，数据无丢失、无错乱。

测试结果： 数据存储稳定、历史记录完整、查询功能正常。

9.3.6 整体设备便携性与稳定性测试

设备采用集成搭载方式，佩戴贴合稳固，不会影响宠物跑动、跳跃、行走等日常行为。设备长时间连续上电运行，无死机、无断联、无频繁掉线情况，整机续航稳定、工作可靠，适合户外长时间佩戴监测。

测试结果： 设备便携性良好，整体运行稳定可靠。

9.4 性能测试结果

1. 实时性：定位、健康数据、环境数据更新延迟低，满足实时监测要求。
2. 稳定性：长时间连续运行无闪退、无断连、无数据丢失，系统整体稳定性良好。
3. 准确性：硬件采集数据精准，天气接口数据匹配对应地理位置，阈值判断准确。
4. 实用性：设备佩戴轻便、软件操作简单、功能贴合实际养宠需求，实用性较强。

9.5 整体运行结论

经过多项功能测试与场景实测，本宠物智能监测系统硬件采集稳定、软件功能完整、数据传输可靠、预警机制完善。系统实现了宠物实时定位、电子围栏防走失、健康状态监测、环境温度监测、多维度异常报警及历史数据记录统计的全部设计功能。整体系统运行稳定、操作便捷、实用性强，完全达到本次项目设计预期目标，可正常投入实际使用。

10、项目总结与展望

10.1 项目总结

本项目完成了基于 Hi3863 开发板与 4G 蜂窝网络的宠物智能监测系统设计与开发。系统以物联网终端硬件为核心，搭配鸿蒙移动端应用软件与云端数据库，整体架构轻量化、实用性强。

硬件端实现了 GNSS 定位、体温采集、运动姿态检测等基础数据采集功能；软件端对接第三方天气 API 接口，实现环境温度获取、电子围栏防走失、多场景异常报警、云端数据存储与历史记录展示等完整业务功能。

经过多场景联合测试，各模块运行协调稳定，4G 数据传输可靠，各项功能均达到项目设计预期，能够满足宠物户外安全监测、日常健康管理的实际使用需求。

10.2 项目展望

本系统已成功构建了基于 4G 蜂窝网络的宠物智能监测基础平台。为应对更复杂的实际场景、提升用户体验并挖掘系统潜力，未来可从连接优化、功能深化、生态扩展三个维度进行迭代升级：

1. 连接与功耗的深度优化

基于网络质量的智能心跳与上报策略：在现有心跳保活机制上，增加对 4G 网络信号强度（RSRP/RSRQ）和类型的实时监测。在信号优（如 4G+）时维持正常频率；在信号弱时，自动降低心跳频率、延长数据上报间隔，并在本地缓存数据，待网络恢复后批量上传。此策略通过避免在弱网环境下的无效射频发射与频繁重连，可显著降低通信模块的整体能耗，从而在不影响核心告警实时性的前提下，有效延长设备的续航时间。

低功耗广域网络（LPWAN）的融合探索：在设备硬件迭代时，可评估集成 LTE Cat.1 或 NB-IoT 通信模块。针对居家、小区等固定场景，设备可自动切换至这些功耗更低、成本更优的网络制式进行小数据量传输（如健康状态、围栏状态），仅在宠物外出、需要高精度定位时启用 4G 模块，实现场景自适应的混合网络连接，从根本上突破续航瓶颈。

2. 核心功能的智能化增强

定位精度的场景化提升：针对文档中提到的“高楼、植被遮挡”导致的 GNSS 漂移问题，在软件端强化多传感器融合定位算法。在 GNSS 信号不佳时，融合 4G 基站定位（LBS）与设备内置的运动传感器（IMU）数据进行航迹推算（DR），在短时间内提供连续、平滑的位置估计，有效填补 GNSS 信号丢失期间的轨迹空白，提升复杂城市场景下的定位连续性。

从“监测”到“预判”的健康管理：利用云端积累的海量宠物历史数据（体温、运动量、活动规律），建立基于时序数据分析的宠物行为与健康模型。系统

不仅能对“体温超标”等瞬时异常报警，更能识别出“日间活动量持续下降”、“夜间异常躁动”等长期趋势性异常，并向主人推送“宠物近日活力不足，建议关注”等预判性提醒，实现健康管理的提前介入。

3. 系统架构与服务的平台化扩展

设备管理与 OTA 升级平台：构建云端设备管理平台，支持对已部署终端的远程状态监控、参数批量配置与固件空中升级。当发现定位算法或通信协议有优化时，可通过 4G 网络向所有设备推送升级包，无需召回硬件，极大提升运维效率与系统可进化性。

4. 设备一对多管理扩展

在未来版本中，可进一步扩展为多设备支持架构，实现一个用户对应多台设备的一对多管理模式，以满足多宠物、多设备同时在线的应用场景。

参考文献

王志良. 物联网关键技术及应用. [M].北京: 清华大学出版社,2013.

赵婧. 嵌入式系统设计与实践. [M].西安: 西安电子科技大学出版社,2021.

陈美汝. 鸿蒙操作系统应用开发实战. [M].北京: 清华大学出版社,2021.

胡国华. 移动通信技术原理与实践. [M].武汉：华中科技大学出版社 2019

唐文彦. 传感器（第四版）.[M].北京: 机械工业出版社,2007

王珊,萨师煊. 数据库系统概论（第五版）. [M].北京: 高等教育出版社,2014.

葛非. HarmonyOS 物联网开发基础实践.[M].北京: 清华大学出版社 2023

附录 项目开发计划

1 概述

本章围绕项目团队分工职责与开发阶段时间规划展开说明，通过明确各模块负责方向、分阶段有序推进，确保宠物智能监测系统研发工作高效落地，实现从设计、开发、联调到验收的全流程规范化管理。

2 团队职责分工

项目按照技术领域与模块功能划分为四大分工方向，各岗位协同配合、并行推进，保障开发进度与系统质量。

2.1 架构设计与服务端开发

1. 负责系统整体技术架构规划与方案设计
2. 完成私有服务器环境部署与配置
3. 搭建 EMQX 消息服务、MySQL 数据库及 PHP 中继服务
4. 设计数据库表结构，实现数据处理脚本与后台业务逻辑
5. 保障数据接收、解析、存储与下发链路稳定运行

2.2 鸿蒙应用开发

1. 负责鸿蒙应用整体框架搭建与页面开发
2. 实现全局通信管理、数据订阅与状态同步功能
3. 完成界面交互逻辑、全局状态管理与本地数据持久化
4. 实现电子围栏配置、历史数据展示等核心业务功能

2.3 告警与业务逻辑模块开发

1. 负责系统通知告警模块设计与实现
2. 完成体温异常、围栏越界、围栏回归等告警逻辑开发
3. 实现通知权限申请、消息防抖与点击跳转功能
4. 优化模块异常处理，提高告警及时性与稳定性

2.4 系统联调、测试与文档整理

1. 负责硬件终端与后台服务的数据对接调试
2. 开展全系统联调测试与功能验证
3. 定位并修复系统缺陷，优化运行稳定性
4. 撰写整理技术文档、架构图纸、模块说明与项目总结

3 项目开发阶段规划

项目整体划分为六个关键开发阶段，各阶段目标清晰、衔接有序，确保系统按期完成研发与交付。

3.1 需求分析与方案设计阶段

1. 明确系统功能需求、应用场景与性能指标
2. 完成整体架构、通信协议与数据库设计
3. 确定模块划分与接口规范，形成完整设计方案

3.2 服务端与环境搭建阶段

1. 部署服务器运行环境与中间件服务
2. 完成数据库创建与服务端脚本开发
3. 实现基础数据链路与消息转发功能

3.3 应用与功能模块开发阶段

1. 进行鸿蒙应用界面与核心逻辑开发
2. 完成通信模块、告警模块等功能编码实现
3. 开展模块内部调试与基础功能验证

3.4 硬件对接与数据调试阶段

1. 完成硬件终端数据上传格式对接
2. 验证数据采集、传输、解析与展示流程正确性
3. 优化数据同步机制与更新实时性

3.5 全系统联调与优化阶段

1. 开展软硬件及后台整体联调测试
2. 修复功能问题，提升系统稳定性与流畅度
3. 完善告警、围栏、统计等业务逻辑

3.6 测试验收与文档整理阶段

1. 完成系统全面功能测试与验收验证
2. 整理项目文档、架构图与核心代码说明
3. 项目成果归档与最终演示交付