MCU 实时数据读取架构分析报告

==========

MCU 实时数据读取架构分析报告

项目: qsjnic (car/qsjnic)
分析时间: 2026-05-17

一、整体数据流（MCU → APP）

```
[MCU 串口]
    ↓ UART 原始字节
[mcu_transport.c]  ← reader thread (select + read)
    ↓ mcu_parser_feed (按 0xFF 0xAA 帧头解析)
[EventQueue] ← C层消息队列 (容量128, mutex+cond)
    ↓ + notify_cb 回调 (DATA/TRANSFER/BOOT 帧)
[jni_bridge.cpp] ← 在 reader 线程中直接 JNI 回调
    ↓ CallStaticVoidMethod
[FrameRouter.onMcuFrame()] ← Java 层入口
    ↓ payload.clone() + Handler.post → 转到 FrameRouter 工作线程
[DomainBus] → 按 msgType 分发给 Feature
    ↓
[VehicleFeature] / [RadioFeature] / [UpgradeFeature]
    ↓ 更新 Snapshot + RemoteCallbackList 广播
[AIDL 客户端] ← IMcuVehicleListener / IMcuFrameListener
```

==========

二、C/Native 层：消息队列已存在

关键文件: service/src/main/cpp/mcu_transport.c / .h

组件	说明
Reader 线程	live_reader_thread()，独占 UART RX，通过 select() 阻塞等待数据
帧解析	mcu_parser_feed() 流式解析，基于 0xFF 0xAA 帧头和 checksum
消息队列	EventQueue，容量 128，基于 pthread_mutex_t + pthread_cond_t 环形缓冲区
入队	event_queue_push()，满时覆盖最旧数据（不是阻塞）
出队	event_queue_wait_pop()，支持超时等待（ACK/SetupAck waiters 使用）

双路径通知机制（live_reader_consume 函数内）：

```
    event_queue_push(&ctx->events, frame);          // 路径1：C层内部waiters使用

    if (ctx->notify_cb != NULL && frame_is_follow_up_data(frame)) {
        ctx->notify_cb(frame, ctx->notify_user);    // 路径2：JNI回调到Java层
    }
```

结论: C 层已经有消息队列，但主要用于 C 层内部的 ACK 等待、SetupAck 等待等同步操作；
实时数据走的是 notify_cb 直接 JNI 回调。

==========

三、JNI 桥接：从 C 直接回调 Java

关键文件: service/src/main/cpp/jni_bridge.cpp

- jni_notify_cb() 在 reader 线程上下文中执行
- 通过缓存的 jmethodID 调用:
  FrameRouter.onMcuFrame(int frameType, int msgType, int cmd, byte[] payload)
- 注意: 这里做了 AttachCurrentThread，reader 线程会长期 attach 到 JVM

==========

四、Java Service 层: FrameRouter + DomainBus

关键文件: service/src/main/java/.../FrameRouter.java

FrameRouter 的工作方式:

```
    // 被 native RX 线程直接调用（非主线程！）
    public static void onMcuFrame(int frameType, int msgType, int cmd, byte[] payload) {
        final byte[] copy = (payload == null) ? null : payload.clone();  // 立即复制
        HANDLER.post(() -> dispatch(frameType, msgType, cmd, copy));      // 抛到工作线程
    }
```

- HANDLER 绑定在独立的 HandlerThread("FrameRouter") 上
- 这样 native reader 线程不会被阻塞

dispatch 做两件事:

1. broadcastRawFrame() → 通过 RemoteCallbackList<IMcuFrameListener> 广播原始帧给所有 AIDL 客户端
2. bus.onFrame() → 交给 DomainBus 做业务级分发

DomainBus（简单轮询分发器）:

```
    public void onFrame(int ft, int mt, int cmd, byte[] body) {
        for (Feature f : features) {
            if (f.accepts(mt)) f.onFrame(ft, mt, cmd, body);
        }
    }
```

==========

五、VehicleFeature: ACC / 车速 / 电压等实时数据

关键文件: service/src/main/java/.../feature/VehicleFeature.java

数据项	cmd 常量	解析函数	广播方式
ACC/线路检测	M2A_LINE_DETECT (0x01)	new LineDetectInfo(body[0])	broadcastVehicle() → onVehicleStateChanged()
车速	M2A_CAR_SPEED	Decoders.parseCarSpeed(body)	broadcastCarSpeed() → onCarSpeed()
电压	M2A_VOLTAGE (0x0D)	Decoders.parseVoltage(body)	broadcastVehicle() → onVoltage()
里程	M2A_MILEAGE	Decoders.parseMileage(body)	broadcastMileage()
背光	M2A_TFT_DUTY	Decoders.parseBacklight(body)	broadcastVehicle()
MCU版本	M2A_MCU_VERSION	Decoders.parseMcuVersion(body)	broadcastVehicle()

VehicleFeature 内部状态:

```
    private final VehicleSnapshot snapshot = new VehicleSnapshot();
```

- 收到帧后更新 snapshot 对应字段
- 通过 RemoteCallbackList<IMcuVehicleListener> 广播给已注册的监听者
- 同时通过 frameListeners（IMcuFrameListener）广播统一状态变更

==========

六、当前架构的总结

层面	队列/线程模型	说明
C 层 UART 读取	独立 reader 线程 + select()	阻塞等待串口数据
C 层消息队列	EventQueue (128容量)	供 C 层 waiters 使用（ACK等待等）
C → Java 跨层	JNI 直接回调	reader 线程直接调用 Java，无队列缓冲
Java 层接收	HandlerThread("FrameRouter")	把 JNI 回调从 native 线程转到 Java 工作线程
业务分发	DomainBus 轮询	按 msgType 匹配 Feature，无优先级队列
客户端通知	RemoteCallbackList	AIDL 广播给所有绑定客户端

==========

七、潜在问题与观察

==========

1. C 层已有消息队列，但 Java 层的实时数据流实际上是:
   reader 线程 → JNI 同步回调 → Handler.post → FrameRouter 工作线程

2. 这个路径中:
   - C 层的 EventQueue 不缓冲给 Java 的实时数据（走的是 notify_cb 旁路）
   - Java 层只有 HandlerThread 的一个消息队列（Android Looper），但没有按数据优先级或类型隔离的队列
   - 如果 VehicleFeature 处理耗时，会阻塞 FrameRouter 工作线程，导致后续 MCU 帧堆积在 Handler 的消息队列中

3. 结论:
   - C 层确实有消息队列（EventQueue，128 容量，pthread mutex/cond）
   - ACC、车速等实时数据已经能完整解析，路径清晰
   - Java 层没有独立的消息队列隔离不同优先级的 MCU 数据，所有帧共享一个 FrameRouter HandlerThread

==========

八、关键文件清单

C/Native 层:
  - service/src/main/cpp/mcu_transport.c/h   (L2 传输层 + EventQueue)
  - service/src/main/cpp/mcu_protocol.c/h   (L1 协议层 + 帧解析)
  - service/src/main/cpp/jni_bridge.cpp   (JNI 桥接)

Java Service 层:
  - service/src/main/java/.../FrameRouter.java   (帧路由 + HandlerThread)
  - service/src/main/java/.../McuLink.java   (Java Facade)
  - service/src/main/java/.../QsMcuSystemService.java (系统服务入口)
  - service/src/main/java/.../feature/DomainBus.java (Feature 分发器)
  - service/src/main/java/.../feature/VehicleFeature.java (车辆数据 Feature)
  - service/src/main/java/.../feature/RadioFeature.java   (收音机 Feature)

==========

报告结束