

DJI 智能电机驱动设计文档

作者：Gemini 3.0 Pro

审查修改：余浩

适用硬件：RoboMaster C610, C620, GM6020

核心特性：O(1) 调度、静态内存池、自动分组、严格冲突检测、零动态内存分配

第一部分：设计方法

1.1 设计背景与电机协议分析

根据 RoboMaster C610/C620/GM6020 电机使用手册，DJI 智能电机采用了一种“**组包控制**”的 CAN 通信协议。其核心特征如下：

1. **一拖四机制**：一个 CAN 控制报文（Standard ID）拥有 8 字节数据负载，每 2 字节控制一个电机。因此，一个 CAN ID 可以同时控制 4 个电机。
2. **ID 强绑定**：电机的物理 ID（电调上设定的 1-8）决定了它必须监听哪个 CAN ID 以及处于报文的哪个位置（Slot）。
 - **C6x0 ID 1-4**: 监听 `0x200`
 - **C6x0 ID 5-8**: 监听 `0x1FF`
 - **GM6020 ID 1-4**: 电压模式下监听 `0x1FF` (**冲突高发区**)
 - **GM6020 ID 5-7**: 电压模式下监听 `0x2FF`

问题：传统的“一个电机一个对象”的面向对象写法容易忽略“组包”特性，导致总线利用率低，且难以检测 C620 与 GM6020 在 `0x1FF` 频段的物理冲突。

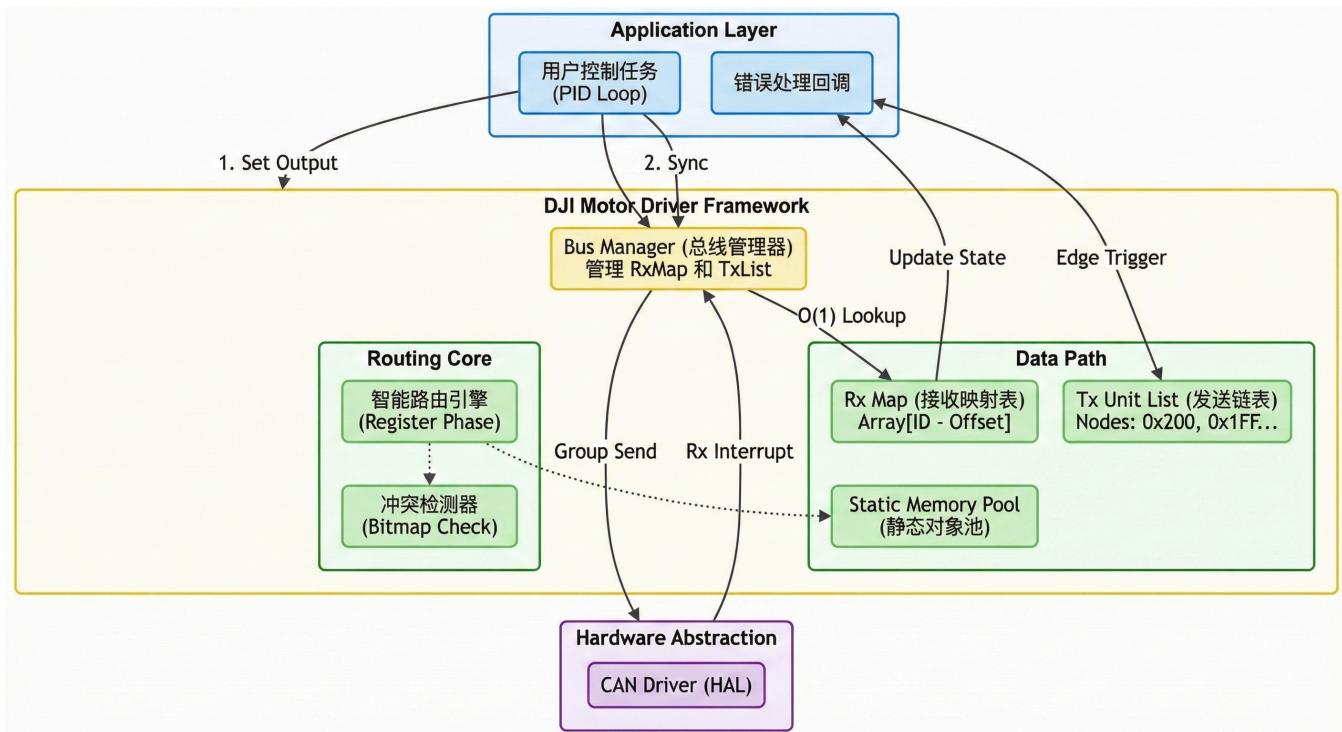
1.2 设计思想

本驱动框架基于以下四大设计哲学构建：

1. **O(1) 极速调度**：
 - 放弃传统的链表遍历查找接收方式。
 - 采用 **直接映射表**，利用 CAN ID 的连续性，通过数组下标直接定位电机对象，接收耗时为常数级。
2. **零动态内存**：
 - 为了满足嵌入式（尤其是车规级/航天级）对确定性的要求，移除 `malloc/free`。
 - 引入 **静态内存池 + 链表** 的管理方式，既保留了动态配置的灵活性，又保证了内存安全。
3. **转运行为与控制行为分离**：
 - `set_output` 仅操作 RAM，不产生 IO。
 - `sync` 负责物理发送。这允许上层算法在计算完所有电机 PID 后，一次性同步发送，最大化利用 CAN 带宽。
4. **防呆与冲突检测**：

- 在注册阶段利用 **位图** 技术检测发送槽位冲突，将物理连接错误拦截在编译/初始化阶段。

1.3 系统架构图



1.4 数据流图

数据流分为 **控制流 (Tx)** 和 **反馈流 (Rx)** 两条主线：

- 控制流 (Tx):** User -> Motor Object -> TxUnit Buffer (RAM) -> Sync -> CAN Bus
- 反馈流 (Rx):** CAN ISR -> Callback -> RxMap (查表) -> Motor Object -> User

第二部分：微观模块设计细节

本驱动主要划分为三个模块：配置模块、核心数据结构、驱动实现逻辑。

2.1 配置模块 (`dji_motor_conf.h`)

作用：提供编译时的资源裁剪能力，使用户能根据实际硬件调整内存占用。

- `DJI_MOTOR_RX_ID_MIN / MAX`:
 - **设计思想：**并非所有机器人都会用到 1-8 号所有 ID。允许用户定义最小和最大 ID，从而缩减 RxMap 指针数组的大小，节省 RAM。
- `DJI_MOTOR_MAX_TX_UNITS`:
 - **设计思想：**这是静态内存池的大小。它决定了系统能同时支持多少个**独立的 CAN 控制帧**（而非电机数）。例如 4 个 C620 共用 1 个 Unit。这避免了为每个电机分配发送缓存的浪费。

2.2 核心数据结构 (`dji_motor_drv.h`)

1. 发送单元 `DJIMotorTxUnit_s`

这是“组包发送”的物理载体，对应 CAN 总线上的一帧报文。

```
struct DJIMotorTxUnit {
    struct list_head list; // 链表节点，用于挂载到总线管理器
    uint32_t canId; // CAN 标识符（如 0x200）
    uint8_t txBuffer[8]; // 8字节数据负载，存放 4 个电机的控制量
    uint8_t isDirty; // 脏标记：用于 Sync 时判断是否有数据更新，减少总线负载
    uint8_t usageMask; // [核心] 资源位图 (Bitmap)
};
```

- `usageMask` **设计细节**：这是一个 4 位的掩码 (Bit 0-3)。
 - Bit 0 = Slot 0 (Byte 0-1)
 - Bit 1 = Slot 1 (Byte 2-3)
 - ...
 - **作用**：当注册电机时，如果计算出的 Slot 对应的 Bit 已经是 1，说明发生了**物理冲突**（两个电机抢同一个字节位置），直接报错。

2. 电机对象 `DJIMotor_s`

这是用户持有的句柄，实现了数据的封装。

```
struct DJIMotor {
    // 1. 静态路由链接 (Link) - 初始化后只读
    struct {
        uint16_t rxId; // 反馈报文 ID (用于 Rx 匹配)
        uint8_t txBufIdx; // 在 TxUnit 中的字节偏移 (0, 2, 4, 6)
        DJIMotorTxUnit_s* txUnit; // 指向所属的发送单元 (指针引用)
        // ...
    } link;

    // 2. 运行时测量数据 (Measure) - 仅通过 API 访问
    struct {
        int16_t angle, velocity, current; // 原始数据
        uint8_t errorCode, lastErrorCode; // 错误状态管理
        int32_t totalAngle; // 软件累加的多圈角度
        // ...
    } measure;

    // 3. 错误回调
    DJIMotorErrorCb_t errorCallback; // 边沿触发回调
};
```

- **设计思想**：将“路由信息”(Link) 与“业务数据”(Measure) 分离。用户通常只关心 Measure，而驱动核心关心 Link。

3. 总线管理器 DJIMotorBus_S

管理一条物理 CAN 总线上的所有资源。

```
typedef struct {
    Device_t canDev;      // 底层 CAN 设备句柄
    uint8_t filterBank;   // 硬件过滤器组编号

    // O(1) 接收表
    DJIMotor_s* rxMap[DJI_MOTOR_RX_MAP_SIZE];

    // 动态(伪)发送链表
    struct list_head txList;
} DJIMotorBus_S;
```

- rxMap **设计细节**: 这是一个指针数组。当收到 CAN ID 为 0x201 的报文时，直接访问 rxMap[0x201 - MIN_ID] 即可拿到电机对象，无需 for 循环遍历。

2.3 驱动实现逻辑 (dji_motor_drv.c)

1. 静态内存分配器 (__alloc_tx_unit_static)

- 实现: 使用一个静态数组 s_tx_pool 和一个标志位数组 s_pool_usage_map。
- 目的: 模拟 malloc 的行为 (按需分配)，但消除了堆内存碎片和分配失败的不确定性，符合高可靠性代码规范。

2. 智能路由注册 (dji_motor_register)

这是整个驱动最复杂的逻辑部分，它充当了“交警”的角色。

- 逻辑流程:

- 输入: 电机类型 (Type)、ID、模式 (Mode)。
- 计算目标车辆 (CAN ID): 根据 DJI 手册规则，判断该电机应该坐哪辆车 (0x200? 0x1FF?)。
 - 细节: 这里处理了 GM6020 电压模式下 ID 1-4 占用 0x1FF 的特殊情况。
- 计算座位号 (Slot): 根据 ID 计算在 8 字节中的偏移。
- 获取/创建 TxUnit: 在链表中查找是否已有该 ID 的车辆。如果没有，从静态池中分配一辆。
- 冲突检测: 检查 usageMask，如果座位已被占，返回 AWLF_ERR_CONFLICT。
- 绑定: 将电机对象的 txUnit 指针指向该车辆。

3. 边沿触发错误回调 (__dji_rx_callback)

- 实现:

```
if (motor->errorCallback && (motor->measure.errorCode != motor->measure.lastErrorCode))
{
    motor->errorCallback(...);
}
```

- 设计思想：**Edge Triggered** (边沿触发)。
 - 原因：电机反馈频率极高 (1kHz)。如果采用电平触发（只要有错就回调），一旦报错，回调函数会瞬间占满 CPU，可能会导致系统卡死。边沿触发保证了只在“出错瞬间”和“恢复瞬间”通知应用层，极大地降低了系统负载。
 - 核心：系统只需要知道“你什么时候坏的”和“你什么时候恢复”即可

4. 物理量转换 Getter

- 实现：`dji_motor_get_current` 等函数。
- 细节：不同电机的原始电流值量程不同 (C610: 10A, C620: 20A)。驱动层内部通过 `scale` 系数封装了这些硬件差异，对上层提供统一的安培 (A) 单位接口，实现了**硬件抽象**。