

# 1.2 fdcan协议解析

## 1. 协议解析

### 1.1 CAN-FD 相关说明

#### 1. CAN-FD 波特率：

- 仲裁段：1 Mbps
- 数据段：最高支持 5 Mbps，也可用 1Mbps（可变波特率）。

#### 2. ID：由 16 位构成，其中 `0x7F` 是广播地址。

- 高 8 位：表示**源地址**：
  - 最高位为 1：需要回复，相当于一个**总开关**，开启但不发生查询指令会返回一个数据段长度为 0 的帧。
  - 最高位位 0：无需回复。
  - 其余 7 位：信号源地址。
- 低 8 位：表示**目的地址**：
  - 最高为 0。
  - 其余 7 位表示目的地址。

例如：

#### 1. ID： `0x8001`

- 信号源地址为 0。
- 目的地址为 1。
- 最高位为 1，表示需要回复，即打开回复**总开关**。

#### 2. ID： `0x100`

- 信号源地址为 1。
- 目的地址为 0。
- 最高位为 0，表示无需回复，即关闭回复**总开关**。

### 1.2 协议基本说明

1. 最小单位为子帧，每条子帧可以向**一个或多个**寄存器写入值，或读取数据。
2. 一条 CAN-FD 帧可由**一个或多个**子帧组成。

3. 电机各个功能实现通过在一条 FDCAN 帧中写入一个或多个对应功能的寄存器的值实现。
4. 可向任意寄存器写入 `int8_t`、`int16_t`、`int32_t`、`float` 四种基本数据类型。
5. 在**同一子帧**中的数据类型必需相同，**不同子帧**中的数据类型可以不同。
6. 所有基本类型的传输都为**小端模式**，即先发送低字节的数据，再发送高字节的数据。
7. 在一条 CAN-FD 中需要在**尾部填充**字节是应使用 `0x50`，表示无任何操作（NOP）。
8. 各种数据类型的**无限制**:
  - `int8_t` : `0x80`
  - `int16_t` : `0x8000`
  - `int32_t` : `0x80000000`
  - `float` : `NAN`
9. 模式三（一拖多模式）用 FDCAN 的 ID 来识别。
  - ID 低 8 位大于等于 `0x80` 时，为模式三（一拖多模式）。
  - ID 低 8 位小于 `0x80` 时，为模式一或模式二（普通模式）。

## 1.3 子帧解析

### 1.3.1 发送协议

#### 1.3.1.1 模式一

```
1  uint8_t tdata[] = {cmd, addr, a1, a2, b1, b2...};
```

1. `cmd` : 表示读写、数据类型和个数。
  - a. 高四位 `cmd[7:4]` 表示读、写、回读。
    - i. `0000` : 写
    - ii. `0001` : 读
    - iii. `0010` : 回复
  - b. 低四位 `cmd[3:0]` 表示数据类型和个数:
    - i. 高两位 `cmd[3:2]` 表示数据类型:
      1. `00` : `int8_t`
      2. `01` : `int16_t`
      3. `10` : `int32_t`

4. 11 : float

ii. 低两位 cmd[1:0] 表示数据个数。

1. 01 : 一个数据。

2. 10 : 两个数据。

3. 11 : 三个数据。

4. 00 : 模式二标志。

2. addr : 起始寄存器地址。

3. a1, a2, b1, b2... : 向寄存器中写入的数据, 注意: 需满足 1.2 协议基本说明的 4、5 项。

a. a1, a2 : 需要向 addr 中写入的数据。

b. b1, b2 : 需要向 addr + 1 中写入的数据。

c. ... : 需要向 add + n 中写入的数据。

### 1.3.1.2 模式二

```
1 uint8_t tdata[] = {cmd, num, addr, a1, a2, b1, b2...};
```

1. cmd : 表示读写、数据类型和个数。

a. 高四位 cmd[7:4] 表示读、写、回读。

i. 0000 : 写

ii. 0001 : 读

iii. 0010 : 回复

b. 低四位 cmd[3:0] 表示数据类型和个数:

i. 高两位 cmd[3:2] 表示数据类型:

1. 00 : int8\_t

2. 01 : int16\_t

3. 10 : int32\_t

4. 11 : float

ii. 低两位 cmd[1:0] 固定为 00 , 表示模式二, 后一字节表示数据个数。

2. num : 数据个数。

3. addr : 起始寄存器地址。

4. a1, a2, b1, b2... : 向寄存器中写入的数据, 注意: 需满足 1.2 协议基本说明的 4、5 项。

- a. `a1, a2` : 需要向 `addr` 中写入的数据。
- b. `b1, b2` : 需要向 `addr + 1` 中写入的数据。
- c. `...` : 需要向 `addr + n` 中写入的数据。

**模式二实质：**在模式一的 `xxx` 中数据个数写为 0 时，使用后一字节表示数据个数，其余字节都往后移动一位。

### 1.3.1.3 模式三（一拖多模式）

- 模式三由 FDCAN-ID 的**低 8 位**决定，FDCAN-ID 大于 **0x80** 即为**一拖多模式**（具体模式见例程）
- 协议格式为：电机 1 数据、电机 2 数据、电机 3 数据... **最后 2 字节**为查询电机状态代码。
- 电机 1、电机 2...：指 id 为 1 的电机、id 为 2 的电机...。
- 一拖多模式发送的数据类型均为 **int16 类型**。

#### 例 1：ID 为 0x8090

```
1  uint8_t cmd[] = {pos11, pos12, val11, val12, tqe11, tqe12, pos21, pos22,
    val21, val22, tqe21, tqe22, 占位字节, 0x17, 0x01}
```

- ID 为 0x8090 时，为一拖多的位置、速度、力矩控制。
- 第 1 字节到第 6 字节分别为位置、速度、力矩控制。
- 第 7 字节到第 12 字节分别为位置、速度、力矩控制。
- 第 13 字节到第 18 字节分别为位置、速度、力矩控制。
- 以此类推...
- 最后 2 字节为查询电机状态的指令。

#### 例 2：ID 为 0x8093

```
1  uint8_t cmd[] = {pos11, pos12, val11, val12, tqe11, tqe12, rkp11, rkp12,
    rkd11, rkd12, pos21, pos22, val21, val22, tqe21, tqe22, rkp21, rkp22, rkd21,
    rkd22, 占位字节, 0x17, 0x01}
```

- ID 为 0x8093 时，为一拖多的位置、速度、力矩、PD 控制。
- 第 1 字节到第 10 字节分别为位置、速度、力矩、PD 控制。
- 第 11 字节到第 20 字节分别为位置、速度、力矩、PD 控制。
- 第 21 字节到第 30 字节分别为位置、速度、力矩、PD 控制。
- 以此类推...

- 最后 2 字节为查询电机状态的指令。

### 1.3.2 接收协议

- 接受协议模式是由发送协议的模式决定的。
- 接收协议模式和发送协议模式是相同的。

#### 1.3.2.1 模式一

假设获取的数据是 `int16_t`

```
1  uint8_t rdata[] = {cmd, addr, a1, a2, b1, b2, ..., cmd1, addr1, c1, c2, c3, c4}
```

- `cmd` :
  - 高四位 `cmd[7, 4]` : `0010` 表示回复。
  - 2~3 位 `cmd[3, 2]` : 表示类型。
    - `00` : `int8_t` 类型。
    - `01` : `int16_t` 类型。
    - `10` : `int32_t` 类型。
    - `11` : `float` 类型。
  - 低 2 位 `cmd[1, 0]` : 表示数量。
    - `00` : 表示模式二。
    - `01` : 一个数据。
    - `10` : 两个数据。
    - `11` : 三个数据。
- `addr` : 开始获取的地址。
- `a1, a2` : 数据 1, 小端模式。
- `b1, b2` : 数据 2, 小端模式。

#### 1.3.2.2 模式二

```
1  uint8_t rdata[] = {cmd, addr, num, a1, a2, b1, b2, ..., cmd1, addr1, c1, c2, c3, c4}
```

- `cmd` :

- 高四位 `cmd[7, 4]` : `0010` 表示回复。
- 2~3 位 `cmd[3, 2]` : 表示类型。
  - `00` : `int8_t` 类型。
  - `01` : `int16_t` 类型。
  - `10` : `int32_t` 类型。
  - `11` : `float` 类型。
- 低 2 位 `cmd[1, 0]` : 表示数量。
  - `00` : 表示模式二, 后一个字节表示数据数量。
- `num` : 数据个数。
- `addr` : 开始获取的地址。
- `a1, a2` : 数据 1, 小端模式。
- `b1, b2` : 数据 2, 小端模式。

### 1.3.3 示例

#### 1.3.3.1 发送协议示例

##### 1.3.3.1.1 模式一

```
1  uint8_t cmd[] = {0x01, 0x00, 0x0A, 0x0A, 0x20, 0x00, 0x00, 0x00, 0x80, 0x10,
    0x27, 0x00, 0x00, 0x50, 0x50, 0x50};
```

该 CAN-FD 帧由两个子帧构成:

- 子帧 1: 整体意思是电机进入位置模式。
  - `0x01` : 第一个子帧的开头
    - 高四位为 `0000` , 表示写操作。
    - 低四位:
      - 高 2 位为: `00` , 表示 `int8_t` 类型。
      - 低 2 位为: `01` , 表示 1 个数据。
  - `0x00` : 起始寄存器地址: 查表可知, `0x00` 寄存器表示电机模式设置。
  - `0x0A` : 往 `0x00` 寄存器写入 `0x0A` 。
- 子帧 2: 整体意思是位置不限制, 速度为 0.1 转/秒
  - `0x0A` : 第 2 个子帧的开头

- 高 8 位为 `0x0`，表示写操作。
  - 低 8 位：
    - 高 2 位为： `10`，表示 `int32_t` 类型。
    - 低 2 位为： `10`，表示 2 个数据。
  - `0x20`：起始寄存器地址：查表可知，`0x20` 寄存器表示位置，`0x21` 寄存器表示速度。
  - `0x00`、`0x00`、`0x00`、`0x80`：小端模式，即 `0x80000000` 写入 `0x20` 寄存器，即表示电机位置无限制。
  - `0x10`、`0x27`、`0x00`、`0x00`：小端模式，即 `0x2710` 写入 `0x21` 寄存器，表示电机速度设置为 0.1 转/秒。
3. `0x50`，`0x50`，`0x50`：由于 CAN-FD 发送字节数最近的两个是 12 和 16，固还需加上 3 个字节的占位字节。

### 1.3.3.1.2 模式二

```
1  uint8_t cmd[] = {0x01, 0x00, 0x0A, 0x08, 0x02, 0x20, 0x00, 0x00, 0x00, 0x80,
    0x10, 0x27, 0x00, 0x00, 0x50, 0x50};
```

#### 该 CAN-FD 帧由两个子帧构成：

1. 子帧 1：整体意思是电机进入位置模式。
  - `0x01`：第一个子帧的开头
    - 高 8 位为 `0001`，表示写操作。
    - 低 8 位：
      - 高 2 位为： `00`，表示 `int8_t` 类型。
      - 低 2 位为： `01`，表示 1 个数据。
  - `0x00`：起始寄存器地址：查表可知，`0x00` 寄存器表示电机模式设置。
  - `0x0A`：往 `0x00` 寄存器写入 `0x0A`。
2. 子帧 2：整体意思是位置不限制，速度为 0.1 转/秒
  - `0x08`：第 2 个子帧的开头
    - 高 8 位为 `0x0`，表示写操作。
    - 低 8 位：
      - 高 2 位为： `10`，表示 `int32_t` 类型。
      - 低 2 位为： `00`，表示模式二，后一个字节表示数据数量。

- `0x02` : 2 个数据。
  - `0x20` : 起始寄存器地址: 查表可知, `0x20` 寄存器表示位置, `0x21` 寄存器表示速度。
  - `0x00`、`0x00`、`0x00`、`0x80` : 小端模式, 即 `0x80000000` 写入 `0x20` 寄存器, 即表示电机位置无限制。
  - `0x10`、`0x27`、`0x00`、`0x00` : 小端模式, 即 `0x2710` 写入 `0x21` 寄存器, 表示电机速度设置为 0.1 转/秒。
3. `0x50`, `0x50` : 由于 CAN-FD 发送字节数最近的两个是 12 和 16, 固还需加上 2 个字节的占位字节。

### 1.3.3.2 接收协议示例

- 接受协议模式是由发送协议的模式决定的。
- 接收协议模式和发送协议模式是相同的。

#### 1.3.3.2.1 模式一

```
1  uint8_t rdata[] = {0x2B, 0x01, 0x96, 0x1D, 0x04, 0x00, 0x54, 0x40, 0x02,
    0x00, 0x86, 0x01, 0x00, 0x00, 0x50, 0x50};
```

该帧由一个子帧构成:

#### 1. 子帧 1: 整体意思是电机进入位置模式。

- `0x2B` :
  - 高 8 位为 `0x2` , 表示回复操作。
  - 低 8 位:
    - 高 2 位为: `10` , 表示 `int32_t` 类型。
    - 低 2 位为: `11` , 表示 3 个数据。
- `0x01` : 起始寄存器地址
- `0x96`, `0x1D`, `0x04`, `0x00` : 小端模式, 十进制为 269718, 即电机当前位置是 2.69718 转处。
- `0x54`, `0x40`, `0x02`, `0x00` : 小端模式, 十进制为 147540, 即当前电机速度为 1.4754 转/秒。
- `0x86`, `0x01`, `0x00`, `0x00` : 小端模式, 十进制为 390, 即当前实际输出力矩为: 0.0039 NM。
- `0x50`, `0x50` : 占位符。

由前三个字符可知: 此 CAN-FD 是回复 `0x1B`, `0x01` 的。



1.3.3.2.2 模式二

```
1  uint8_t rdata[] = {0x28, 0x04, 0x00, 0x0A, 0x00, 0x00, 0x00, 0x96, 0x1D,
    0x04, 0x00, 0x54, 0x40, 0x02, 0x00, 0x86, 0x01, 0x00, 0x00, 0x50};
```

该帧由一个子帧构成：

1. 子帧 1：整体意思是电机进入位置模式。

- 0x28：
  - 高 8 位为 0010，表示回复操作。
  - 低 8 位：
    - 高 2 位为：10，表示 int32\_t 类型。
    - 低 2 位为：00，表示模式二，后一个字节表示数据数量。
- 0x04：4 个数据。
- 0x00：起始寄存器地址
- 0x0A, 0x00, 0x00, 0x00：0x00 寄存器的值，查表可知为电机模式，十进制为 10，查表可知为位置模式。
- 0x96, 0x1D, 0x04, 0x00：小端模式，十进制为 269718，即电机当前位置是 2.69718 转处。
- 0x54, 0x40, 0x02, 0x00：小端模式，十进制为 147540，即当前电机速度为 1.4754 转/秒。
- 0x86, 0x01, 0x00, 0x00：小端模式，十进制为 390，即当前实际输出力矩为：0.0039 NM。
- 0x50：占位符。

由前三个字符可知：此 CAN-FD 是回复 0x18, 0x04, 0x00 的。

2. 常用类型（单位）说明

2.1 电流（A）

数据类型	LSB	实际 (A))
int8	1	1
int16	1	0.1
int32	1	0.001

float	1	1
-------	---	---

## 2.2 电压（V）

数据类型	LSB	实际 (V))
int8	1	0.5
int16	1	0.1
int32	1	0.001
float	1	1

## 2.3 扭矩（Nm）

- 真实扭矩 =  $k * tqe + d$
- **d项**：相当于电机本身的摩擦力，对精度要求不高的可以不用加，fdcan\_h730工程从v3.0.5开始不再加入此项。
- 扭矩系数以fdcan\_h730中的 motor\_tqe\_adj 为准。

电机型号	扭矩系数
M3536_32	0.458105
M4438_30	0.5256
M4438_32	0.485565
M4538_19	0.493835
M5043_20	0.966
M5046_20	0.533654
M5047_09	0.547474
M5047_36	0.803
M6056_36	0.677
M7256_35	0.676524
M60SG_35	0.7942

M60BM_35	0.7942
----------	--------

2.4 温度（℃）

数据类型	LSB	实际 (℃)
int8	1	1
int16	1	0.1
int32	1	0.001
float	1	1

2.5 时间（s）

数据类型	LSB	实际 (s)
int8	1	0.01
int16	1	0.001
int32	1	0.000001
float	1	1

2.6 位置（转）

数据类型	LSB	实际 (转)	实际 (°)
int8	1	0.01	3.6
int16	1	0.0001	0.036
int32	1	0.00001	0.0036
float	1	1	360

2.7 速度（转/秒）

数据类型	LSB	实际 (转/秒)
int8	1	0.01

int16	1	0.00025
int32	1	0.00001
float	1	1

## 2.8 加速度（转/秒^2）

数据类型	LSB	实际 (转/秒^2)
int8	1	0.05
int16	1	0.001
int32	1	0.00001
float	1	1

## 2.9 PWM 标度（无单位）

数据类型	LSB	实际
int8	1	1/127 - 0.007874
int16	1	1/32767 - 0.000030519
int32	1	(1/2147483647) - 4.657^10
float	1	1

## 2.10 rKp、rKd 标度（无单位，寄存器0x23和0x24的）

数据类型	LSB	实际
int8	1	1/127 - 0.007874
int16	1	1/32767 - 0.000030519
int32	1	(1/2147483647) - 4.657^10
float	1	1

## 2.11 Kp、Kd 标度（无单位，例程中用的是这个）

数据类型	LSB	实际
int8	1	1
int16	1	0.1
int32	1	0.001
float	1	1

### 3. 函数示例

说明：

- 下面介绍示例程序中提供的电机控制模式的简要说明。
- 详情请看提供的示例工程。
- 本电机所能实现的功能不止于此，如需自定义特殊功能，可根据寄存器表发挥想象。
- 下列为封装函数，可以直接调用

#### 3.1 普通模式

##### 3.1.1 DQ 电压模式

说明：

1. 通过给定的 Q 相电压控制电机运行，并让电机返回状态信息。
2. 参数解析：
  - portx :CAN 通道，PORT1、PORT2、PORT3 对应通道 1、2、3
  - type ：通信协议的数据类型，影响数据的精度和量程，TFLOAT、TINT32、TINT16 对应 float、int32、int16
  - id ：电机 ID
  - volt ：Q 相电压，单位：伏特（V）。

代码块

```
1 void motor_set_dq_vlot(port_t portx, const data_type_t type, const uint8_t id,
  const float volt);
```

##### 3.1.2 DQ 电流模式

说明：

1. 通过给定的 Q 相电流控制电机运行，并让电机返回状态信息。

## 2. 参数解析：

- `portx` :CAN 通道，`PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
- `type`：通信协议的数据类型，影响数据的精度和量程，`TFLOAT`、`TINT32`、`TINT16` 对应 `float`、`int32`、`int16`
- `id`：电机 ID
- `cur`：Q 相电流，单位：安培（A）。

代码块

```
1 void motor_set_dq_current(port_t portx, const data_type_t type, const uint8_t id, const float cur);
```

### 3.1.3 位置模式

说明：

1. 电机将以最大速度和最大加速度运动到指定目标位置，并让电机返回状态信息。

## 2. 参数解析：

- `portx` :CAN 通道，`PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
- `type`：通信协议的数据类型，影响数据的精度和量程，`TFLOAT`、`TINT32`、`TINT16` 对应 `float`、`int32`、`int16`
- `id`：电机 ID
- `pos`：目标位置，单位：转（r）。

注意：

1. 该模式下速度与力矩均为最大，运动过程较为激烈。
2. 瞬时电流可能会飙到 5A~10A，若电源响应不够快，或电源电流限制过小，可能导致电机在瞬间获得的电流不足，从而报错。
3. 此模式适用于对响应速度有极端要求的场合，一般不建议使用，如需进行位置控制，推荐使用**梯形控制**模式。

代码块

```
1 void motor_set_pos(port_t portx, const data_type_t type, const uint8_t id, const float pos);
```

### 3.1.4 速度模式

说明：

1. 电机以最大加速度加速到指定目标速度，并让电机返回状态信息。

2. 参数解析：

- `portx` :CAN 通道，`PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
- `type`：通信协议的数据类型，影响数据的精度和量程，`TFLOAT`、`TINT32`、`TINT16` 对应 `float`、`int32`、`int16`
- `id`：电机 ID
- `vel`：目标速度，单位：转/秒 (r/s)

代码块

```
1 void motor_set_vel(port_t portx, const data_type_t type, const uint8_t id,
  const float vel);
```

### 3.1.5 力矩模式

说明：

1. 电机按照设定的目标力矩进行转动，并让电机返回状态信息。

2. 参数解析：

- `portx` :CAN 通道，`PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
- `type`：通信协议的数据类型，影响数据的精度和量程，`TFLOAT`、`TINT32`、`TINT16` 对应 `float`、`int32`、`int16`
- `id`：电机 ID
- `tqe`：目标力矩，单位：牛·米 (N·m)。

代码块

```
1 void motor_set_tqe(port_t portx, const data_type_t type, const uint8_t id,
  const float tqe);
```

### 3.1.6 位置、速度模式

说明：

1. 电机以目标速度运动至指定的目标位置，并让电机返回状态信息。

2. 参数解析：

- `portx` :CAN 通道，`PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
- `type`：通信协议的数据类型，影响数据的精度和量程，`TFLOAT`、`TINT32`、`TINT16` 对应 `float`、`int32`、`int16`

- `id` : 电机 ID
- `pos` : 目标位置, 单位: 转 (r) 。
- `vel` : 目标速度, 单位: 转/秒 (r/s)

代码块

```
1 void motor_set_pos_vel(port_t portx, const data_type_t type, const uint8_t id,  
2   const float pos, const float vel);
```

### 3.1.7 位置、速度、最大力矩模式

说明:

1. 电机以目标速度运动至指定的目标位置, 同时限制最大输出力矩, 并让电机返回状态信息。

2. 参数解析:

- `portx` :CAN 通道, `PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
- `type` : 通信协议的数据类型, 影响数据的精度和量程, `TFLOAT`、`TINT32`、`TINT16` 对应 `float`、`int32`、`int16`
- `id` : 电机 ID
- `pos` : 目标位置, 单位: 转 (r) 。
- `vel` : 目标速度, 单位: 转/秒 (r/s) 。
- `tqe` : 目标力矩, 单位: 牛·米 (N·m) 。

注意:

1. 该模式对输出力矩有限制, 若设置的最大力矩过小, 电机可能无法达到目标速度。

代码块

```
1 void motor_set_pos_vel_MAXtqe(port_t portx, const data_type_t type, const  
   uint8_t id,  
2   const float pos, const float vel, const float tqe);
```

### 3.1.8 位置、速度、加速度模式 (梯形控制)

说明:

1. 电机按照恒定加速度运动, 实现先加速 → 匀速 → 减速的梯形速度控制, 并让电机返回状态信息。

2. 参数解析:

- `portx` :CAN 通道, `PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3



- `type`：通信协议的数据类型，影响数据的精度和量程，`TFLOAT`、`TINT32`、`TINT16` 对应 `float`、`int32`、`int16`
- `id`：电机 ID
- `pos`：目标位置，单位：转（r）。
- `vel`：目标速度，单位：转/秒（r/s）。
- `acc`：加速度，单位：转每秒平方（rps<sup>2</sup>）。

代码块

```
1 void motor_set_pos_velmax_acc(port_t portx, const data_type_t type, const
  uint8_t id,
2  const float pos, const float vel, const float acc);
```

### 3.1.9 速度、加速度模式

说明：

1. 以目标加速度加速到目标速度，并让电机返回状态信息。

2. 参数解析：

- `portx`：CAN 通道，`PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
- `type`：通信协议的数据类型，影响数据的精度和量程，`TFLOAT`、`TINT32`、`TINT16` 对应 `float`、`int32`、`int16`
- `id`：电机 ID
- `vel`：目标速度，单位：转/秒（r/s）。
- `acc`：加速度，单位：转每秒平方（rps<sup>2</sup>）。

代码块

```
1 void motor_set_vel_acc(port_t portx, const data_type_t type, const uint8_t id,
2  const float vel, const float acc);
```

### 3.1.10 运控模式

说明：

1. 电机输出力矩的计算公式为：

- 输出力矩 = （目标位置-当前位置） \* `kp` + （目标速度-当前速度） \* `kd` + `torque`。

2. 参数解析：

- `portx`：CAN 通道，`PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3

- `type`：通信协议的数据类型，影响数据的精度和量程，`TFLOAT`、`TINT32`、`TINT16` 对应 `float`、`int32`、`int16`
- `id`：电机 ID
- `pos`：目标位置，单位：转（r）。
- `vel`：目标速度，单位：转/秒（r/s）。
- `tqe`：目标力矩，单位：牛·米（N·m）。
- `kp`：位置比例系数。
- `kd`：速度比例系数。

#### 注意：

1. 需要设置合适的 `kp` 和 `kd`，否则控制效果可能较差。
2. `motor_set_pos_vel_tqe_kp_kd` 函数的位置是不断积分得到的，所以在低频控制下会有意外情况。
3. 推荐使用运控模式 2 `motor_set_pos_vel_tqe_kp_kd_2`。

#### 代码块

```
1  /* 不建议使用 */
2  void motor_set_pos_vel_tqe_kp_kd(port_t portx, const data_type_t type, const
   uint8_t id,
3  const float pos, const float vel, const float tqe, const float kp, const float
   kd);
4
5  /* 建议使用 */
6  void motor_set_pos_vel_tqe_kp_kd_2(port_t portx, const data_type_t type, const
   uint8_t id,
7  const float pos, const float vel, const float tqe, const float kp, const float
   kd);
```

### 3.1.11 运控模式 2

#### 说明：

1. 电机输出力矩的计算公式为：
  - 输出力矩 = （目标位置-当前位置） \* `kp` + （目标速度-当前速度） \* `kd` + torque。
2. 参数解析：
  - `portx`：CAN 通道，`PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
  - `type`：通信协议的数据类型，影响数据的精度和量程，`TFLOAT`、`TINT32`、`TINT16` 对应 `float`、`int32`、`int16`

- `id` : 电机 ID
- `pos` : 目标位置, 单位: 转 (r) 。
- `vel` : 目标速度, 单位: 转/秒 (r/s) 。
- `tqe` : 目标力矩, 单位: 牛·米 (N·m) 。
- `kp` : 位置比例系数。
- `kd` : 速度比例系数。

#### 注意:

1. 真正的运控模式, 和 `motor_set_pos_vel_tqe_kp_kd` 区别是:
  - `motor_set_pos_vel_tqe_kp_kd` 的位置是不断积分得到的, 在低频控制下容易出现意外情况。
  - `motor_set_pos_vel_tqe_kp_kd_2` 则避免了此问题, 控制更稳定。
2. 该需要设置合适的 `kp` 和 `kd` , 否则控制效果可能较差。

#### 代码块

```

1  /* 不建议使用 */
2  void motor_set_pos_vel_tqe_kp_kd(port_t portx, const data_type_t type, const
    uint8_t id,
3  const float pos, const float vel, const float tqe, const float kp, const float
    kd);
4
5  /* 建议使用 */
6  void motor_set_pos_vel_tqe_kp_kd_2(port_t portx, const data_type_t type, const
    uint8_t id,
7  const float pos, const float vel, const float tqe, const float kp, const float
    kd);

```

### 3.1.12 查询电机状态信息

#### 说明:

1. 发送查询电机状态信息的指令。
2. 电机返回的状态信息包括, 模式、错误码、位置、速度、力矩。
3. 电机状态信息解析在 `motor.c` 的 `motor_process_state` 函数。
4. 参数解析:
  - `portx` : CAN 通道, `PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
  - `type` : 通信协议的数据类型, 影响数据的精度和量程, `TFLOAT`、`TINT32`、`TINT16` 对应 `float`、`int32`、`int16`

- `id`：电机 ID

代码块

```
1 void motor_get_state_send(port_t portx, const data_type_t type, const uint8_t id);
```

### 3.1.13 查询电机固件版本

说明：

1. 发送查询电机固件版本号指令。
2. 电机固件版本解析在 `motor.c` 的 `motor_process_state` 函数。
3. 参数解析：
  - `portx`：CAN 通道，`PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
  - `id`：电机 ID

代码块

```
1 void motor_get_version(port_t portx, const uint8_t id);
```

### 3.1.14 停止模式

说明：

1. 电机进入停止模式，电机三相都悬空，使电机可以自由转动。
2. 参数解析：
  - `portx`：CAN 通道，`PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
  - `id`：电机 ID

代码块

```
1 void motor_set_stop(port_t portx, const uint8_t id);
```

### 3.1.15 刹车模式

说明：

1. 将电机所有相短接到地，实现“阻尼刹车”效果。
2. 刹车阻力与电机转速成正相关。
3. 参数解析：

- `portx`:CAN 通道, `PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
- `id`: 电机 ID

代码块

```
1 void motor_set_brake(port_t portx, const uint8_t id);
```

### 3.1.16 电机软重启

说明:

1. 对电机执行软件重启。 , 重启后进入停止模式。
2. 不会有任何反馈。
3. 参数解析:

- `portx`:CAN 通道, `PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
- `id`: 电机 ID

代码块

```
1 void motor_set_reset(port_t portx, const uint8_t id);
```

## 3.2 一拖多控制模式 `motor_many.c`

- 此文件内, 只有 `motor_many_send` 是用于发送指令的, 其他的函数都是向缓存写入指令。
- 一拖多模式下, 同一条 CAN 通道上的电机模式是相同的。
- 一拖多模式大致原理, 发送一条通用的 FDCAN 帧, 帧上不同的字节控制不同电机, 其中由 ID 确定电机模式, 详细说明请看 02-fdcan 协议解析。
- 一拖多模式下, 在仲裁段为 1M, 数据段为 5M 的情况下, 一个 CAN 总线 1ms 内最多控制 10 个电机, 即 1KHz 控制频率下最多控制 10 个电机, 如需控制更多电机, 需降低控制频率。
- `TINT16` 对应的是 `int16` 的数据类型, 控制只能到 3.2 圈。
- 一拖多模式使用说明:

代码块

```
1  /* 写入缓存 */
2  motor_many_vel(PORT1, 1, 0.1);
3  motor_many_vel(PORT1, 2, 0.1);
4  motor_many_vel(PORT1, 3, 0.1);
5
6  /* 发送指令 */
```

```
7 motor_many_send(PORT1);
```

### 3.2.1 DQ 电压模式

#### 说明：

1. 通过给定的 Q 相电压控制电机运行，并让电机返回状态信息。
2. 参数解析：
  - `portx` :CAN 通道，`PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
  - `type` : 通信协议的数据类型，影响数据的精度和量程，`TFLOAT`、`TINT32`、`TINT16` 对应 `float`、`int32`、`int16`
  - `id` : 电机 ID
  - `volt` : Q 相电压，单位：伏特（V）。

#### 注意：

1. 此函数仅将指令写入缓冲区，不会立即发送，需要调用 `motor_many_send` 才会生效。

#### 代码块

```
1 void motor_many_dq_volt(port_t portx, const uint8_t id, const float volt);
```

### 3.2.2 DQ 电流模式

#### 说明：

1. 通过给定的 Q 相电流控制电机运行，并让电机返回状态信息。
2. 参数解析：
  - `portx` :CAN 通道，`PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
  - `type` : 通信协议的数据类型，影响数据的精度和量程，`TFLOAT`、`TINT32`、`TINT16` 对应 `float`、`int32`、`int16`
  - `id` : 电机 ID
  - `cur` : Q 相电流，单位：安培（A）。

#### 注意：

1. 此函数仅将指令写入缓冲区，不会立即发送，需要调用 `motor_many_send` 才会生效。

#### 代码块

```
1 void motor_many_dq_current(port_t portx, const uint8_t id, const float cur);
```

### 3.2.3 位置模式

#### 说明：

1. 电机将以最大速度和最大加速度运动到指定目标位置，并让电机返回状态信息。
2. 参数解析：
  - `portx` :CAN 通道，`PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
  - `type` : 通信协议的数据类型，影响数据的精度和量程，`TFLOAT`、`TINT32`、`TINT16` 对应 `float`、`int32`、`int16`
  - `id` : 电机 ID
  - `pos` : 目标位置，单位：转（r）。

#### 注意：

1. 该模式下速度与力矩均为最大，运动过程较为激烈。
2. 瞬时电流可能会飙到 5A~10A，若电源响应不够快，或电源电流限制过小，可能导致电机在瞬间获得的电流不足，从而报错。
3. 此模式适用于对响应速度有极端要求的场合，一般不建议使用，如需进行位置控制，推荐使用**梯形控制**模式。
4. 此函数仅将指令写入缓冲区，不会立即发送，需要调用 `motor_many_send` 才会生效。

代码块

```
1 void motor_many_pos(port_t portx, const uint8_t id, const float pos);
```

### 3.2.4 速度模式

#### 说明：

1. 电机以最大加速度加速到指定目标速度，并让电机返回状态信息。
2. 参数解析：
  - `portx` :CAN 通道，`PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
  - `type` : 通信协议的数据类型，影响数据的精度和量程，`TFLOAT`、`TINT32`、`TINT16` 对应 `float`、`int32`、`int16`
  - `id` : 电机 ID
  - `vel` : 目标速度，单位：转/秒（r/s）

#### 注意：

1. 此函数仅将指令写入缓冲区，不会立即发送，需要调用 `motor_many_send` 才会生效。

```
1 void motor_many_vel(port_t portx, const uint8_t id, const float vel);
```

### 3.2.5 力矩模式

#### 说明:

1. 电机按照设定的目标力矩进行转动，并让电机返回状态信息。
2. 参数解析：
  - `portx`: CAN 通道, `PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
  - `type`: 通信协议的数据类型, 影响数据的精度和量程, `TFLOAT`、`TINT32`、`TINT16` 对应 `float`、`int32`、`int16`
  - `id`: 电机 ID
  - `tqe`: 目标力矩, 单位: 牛·米 (N·m)。

#### 注意:

1. 此函数仅将指令写入缓冲区, 不会立即发送, 需要调用 `motor_many_send` 才会生效。

#### 代码块

```
1 void motor_many_tqe(port_t portx, const uint8_t id, const float tqe);
```

### 3.2.6 位置、速度模式

#### 说明:

1. 电机以目标速度运动至指定的目标位置, 并让电机返回状态信息。
2. 参数解析：
  - `portx`: CAN 通道, `PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
  - `type`: 通信协议的数据类型, 影响数据的精度和量程, `TFLOAT`、`TINT32`、`TINT16` 对应 `float`、`int32`、`int16`
  - `id`: 电机 ID
  - `pos`: 目标位置, 单位: 转 (r)。
  - `vel`: 目标速度, 单位: 转/秒 (r/s)

#### 注意:

1. 此函数仅将指令写入缓冲区, 不会立即发送, 需要调用 `motor_many_send` 才会生效。

#### 代码块

```
1 void motor_many_pos_vel(port_t portx, const uint8_t id, const float pos, const
```



```
float vel);
```

### 3.2.7 位置、速度、最大力矩模式

#### 说明：

1. 电机以目标速度运动至指定的目标位置，同时限制最大输出力矩，并让电机返回状态信息。

#### 2. 参数解析：

- `portx` :CAN 通道，`PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
- `type` : 通信协议的数据类型，影响数据的精度和量程，`TFLOAT`、`TINT32`、`TINT16` 对应 `float`、`int32`、`int16`
- `id` : 电机 ID
- `pos` : 目标位置，单位：转 (r) 。
- `vel` : 目标速度，单位：转/秒 (r/s) 。
- `tqe` : 目标力矩，单位：牛·米 (N·m) 。

#### 注意：

3. 该模式对输出力矩有限制，若设置的最大力矩过小，电机可能无法达到目标速度。
4. 此函数仅将指令写入缓冲区，不会立即发送，需要调用 `motor_many_send` 才会生效。

#### 代码块

```
1 void motor_many_pos_vel_MAXtqe(port_t portx, const uint8_t id,  
2 const float pos, const float vel, const float tqe);
```

### 3.2.8 位置、速度、加速度模式（梯形控制）

#### 说明：

1. 电机按照恒定加速度运动，实现先加速 → 匀速 → 减速的梯形速度控制，并让电机返回状态信息。

#### 2. 参数解析：

- `portx` :CAN 通道，`PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
- `type` : 通信协议的数据类型，影响数据的精度和量程，`TFLOAT`、`TINT32`、`TINT16` 对应 `float`、`int32`、`int16`
- `id` : 电机 ID
- `pos` : 目标位置，单位：转 (r) 。
- `vel` : 目标速度，单位：转/秒 (r/s) 。
- `acc` : 加速度，单位：转每秒平方 (rps<sup>2</sup>) 。

## 注意：

1. 此函数仅将指令写入缓冲区，不会立即发送，需要调用 `motor_many_send` 才会生效。

### 代码块

```
1 void motor_many_pos_vel_acc(port_t portx, const uint8_t id,  
2 const float pos, const float vel, const float acc);
```

## 3.2.9 运控模式

### 说明：

1. 电机输出力矩的计算公式为：
  - 输出力矩 = (目标位置-当前位置) \* `kp` + (目标速度-当前速度) \* `kd` + torque。
2. 参数解析：
  - `portx` :CAN 通道，`PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
  - `type` : 通信协议的数据类型，影响数据的精度和量程，`TFLOAT`、`TINT32`、`TINT16` 对应 `float`、`int32`、`int16`
  - `id` : 电机 ID
  - `pos` : 目标位置，单位：转 (r) 。
  - `vel` : 目标速度，单位：转/秒 (r/s) 。
  - `tqe` : 目标力矩，单位：牛·米 (N·m) 。
  - `kp` : 位置比例系数。
  - `kd` : 速度比例系数。

### 注意：

3. 需要设置合适的 `kp` 和 `kd` ，否则控制效果可能较差。
4. `motor_many_pos_vel_tqe_kp_kd` 函数的位置是不断积分得到的，所以在低频控制下会有意外情况。
5. 推荐使用运控模式 2 `motor_many_pos_vel_tqe_kp_kd_2`。
6. 此函数仅将指令写入缓冲区，不会立即发送，需要调用 `motor_many_send` 才会生效。

### 代码块

```
1 /* 不建议使用 */  
2 void motor_many_pos_vel_tqe_kp_kd(port_t portx, const uint8_t id,  
3 const float pos, const float vel, const float tqe, const float kp, const float  
kd);  
4
```

```

5  /* 建议使用 */
6  void motor_many_pos_vel_tqe_kp_kd_2(port_t portx, const uint8_t id,
7  const float pos, const float vel, const float tqe, const float kp, const float
    kd);

```

### 3.2.10 运控模式 2

#### 说明：

##### 1. 电机输出力矩的计算公式为：

- 输出力矩 = (目标位置-当前位置) \* kp + (目标速度-当前速度) \* kd + torque。

##### 2. 参数解析：

- `portx` :CAN 通道, `PORT1`、`PORT2`、`PORT3` 对应通道 1、2、3
- `type` : 通信协议的数据类型, 影响数据的精度和量程, `TFLOAT`、`TINT32`、`TINT16` 对应 `float`、`int32`、`int16`
- `id` : 电机 ID
- `pos` : 目标位置, 单位: 转 (r) 。
- `vel` : 目标速度, 单位: 转/秒 (r/s) 。
- `tqe` : 目标力矩, 单位: 牛·米 (N·m) 。
- `kp` : 位置比例系数。
- `kd` : 速度比例系数。

#### 注意：

##### 3. 真正的运控模式, 和 `motor_many_pos_vel_tqe_kp_kd` 区别是：

- `motor_many_pos_vel_tqe_kp_kd` 的位置是不断积分得到的, 在低频控制下容易出现意外情况。
- `motor_many_pos_vel_tqe_kp_kd_2` 则避免了此问题, 控制更稳定。

##### 4. 该需要设置合适的 `kp` 和 `kd` , 否则控制效果可能较差。

##### 5. 此函数仅将指令写入缓冲区, 不会立即发送, 需要调用 `motor_many_send` 才会生效。

#### 代码块

```

1  /* 不建议使用 */
2  void motor_many_pos_vel_tqe_kp_kd(port_t portx, const uint8_t id,
3  const float pos, const float vel, const float tqe, const float kp, const float
    kd);
4
5  /* 建议使用 */
6  void motor_many_pos_vel_tqe_kp_kd_2(port_t portx, const uint8_t id,

```

```
7  const float pos, const float vel, const float tqe, const float kp, const float
    kd);
```

## 4. 示例程序说明

- 电机示例函数都在 `livelybot_fdcan.c` 和 `livelybot_fdcan.h` 文件内。
- 示例程序默认不会运行任何控制代码，需测试某一功能请在 `main.c` 文件内解注释或自行编写。

需要注意的宏定义：

- `MOTOR_TORQUE_RATIO`：用于选择电机型号，用于修正输入力矩。
- `POS_FLAG`：用于**测试位置模式**，共三种模式，三种模式只是数据类型不同，效果都是 -0.5 转 ~0.5 转来回旋转。
  - `POS_FLAG = 1`：float
  - `POS_FLAG = 2`：int32
  - `POS_FLAG = 3`：int16
- `READ_MOTOR_FLAG`：用于改变读取**电机状态的数据类型**。
  - `READ_MOTOR_FLAG = 1`：float
  - `READ_MOTOR_FLAG = 2`：int 32
  - `READ_MOTOR_FLAG = 3`：int 16
- `data_type_t`：用于改变读取**电机状态的数据类型**。
  - `TFLOAT`：float
  - `TINT32`：int 32
  - `TINT16`：int 16
- `port_t`：用于选择**电机通道**
  - `PNULL`：0
  - `PORT1`：1
  - `PORT2`：2
  - `PORT3`：3
- `POS_REZERO`：用于测试电机的**重置零点**功能，
  - 效果是电机上电 3 秒后将当前位置设为零点。
  - 注意：需让电机停止后再重置零位，否则无效
  - 需测试此功能将 `POS_REZERO` 解注释即可。
- `MOTOR_STOP`：用于测试电机**停止**功能。

- a. 需测试此功能将 MOTOR\_STOP 解注释即可。
  - b. 效果是电机上电 3 秒后将停止。
  - c. 注意：需配合电机控制函数使用，启用 MOTOR\_STOP 宏不会改变任何电机控制函数。
8. `MOTOR_BRAKE`：用于测试电机**刹车**功能功能。
- a. 需测试此功能将 MOTOR\_BRAKE 解注释即可。\*\*\*\*