进程间通信（**uORB**）

参考链接：http://blog.arm.so/docs/183-0503.html

uORB 是 Pixhawk 系统中非常重要且关键的一个模块，它肩负了整个系统的数据传输任务，所有的传感器数据、GPS、PPM 信号等都要从芯片获取后通过 uORB 进行传输到各个模块进行计算处理。

uORB 的架构简述：uORB 全称为 micro object request broker (uORB),即“微对象请求代理器”，实际上 uORB 是一套跨进程的 IPC 通讯模块。在 Pixhawk 中，所有的功能被独立以进程模块为单位进行实现并工作。而进程间的数据交互就由为重要，必须要能够符合实时、有序的特点。

Pixhawk 使用 NuttX 实时 ARM 系统，而 uORB 对于 NuttX 而言，它仅仅是一个普通的文件设备对象，这个设备支持 Open、Close、Read、Write、Ioctl 以及 Poll 机制。通过这些接口的实现，uORB 提供了一套“点对多”的跨进程广播通讯机制，“点”指的是通讯消息的“源”，“多”指的是一个源可以有多个用户来接收、处理。而“源”与“用户”的关系在于，源不需要去考虑用户是否可以收到某条被广播的消息或什么时候收到这条消息。它只需要单纯的把要广播的数据推送到 uORB 的消息“总线”上。对于用户而言，源推送了多少次的消息也不重要，重要的是取回最新的这条消息。

uORB 实际上是多个进程打开同一个设备文件，进程间通过此文件节点进行数据交互和共享。

uORB 的系统实现：

uORB 的实现位于固件源码的 src/modules/uORB/uORB.cpp 文件，它通过重载 CDev 基类来组织一个 uORB 的设备实例。并且完成 Read/Write 等功能的重载。

uORB 的入口点是 uorb_main 函数，在这里它检查 uORB 的启动参数来完成对应的功能，uORB 支持 start/test/status 这 3 条启动参数，在 Pixhawk 的 rcS 启动脚本中，使用 start 参数来进行初始化，其他 2 个参数分别用来进行 uORB 功能的自检和列出 uORB 的当前状态。

在 rcS 中使用 start 参数启动 uORB 后，uORB 会创建并初始化它的设备实例，其中的实现大部分都在 CDev 基类完成。这个过程类似于 Linux 设备驱动中的 Probe 函数，或者 Windows 内核的 DriverEntry，通过 init 调用完成设备的创建，节点注册以及派遣例程的设置等。

********** 下面是官网资料**********

参考链接：http://pixhawk.org/dev/shared_object_communication

进程（process）/程序（application）间通信（如将传感器信息从传感器 app 传送到姿态滤波 app）是 PX4 程序结构的核心部分。进程（process，在此处被称作 nodes）通过被命名的总线（buses，在此处被称作 topic）交换信息。在 PX4 中，一个 topic 只包含一种信息类型，比如，vehicle_attitude 这个 topic 将一个包含姿态结构体（roll、pitch、yaw）的信息传送出去。Nodes 可以在 bus/topic 上 publish（发布）一个信息（即发送数据），也可以向一个 bus/topic subscribe（订阅）信息（即接收数据）。它们（Nodes）并不知道它们在跟谁通信。一个 topic 可以面向多个 publishers（发布者）和多个 subscribers（订阅者）。这种方式可以避免死锁问题，在机器人中很常见。为达到有效率，在 bus/topic 中，永远只有一个信息被传送，没有保持队列之说（即新来的信息会覆盖之前的信息，不存在有一串信息排队的情况）。

这个发布/订阅（publisher / subscriber）机制是通过微对象请求代理（micro

object request broker,简称 uORB）来实现的。

- ✧ 系统已存的 topics 通过 Doxygen 工具自动生成了文档，其链接为：

  https://pixhawk.ethz.ch/docs/group__topics.html

下面是关于 publisher / subscriber 的一个简单的例子，这个 publisher（发布者）advertises（通告）一个名叫 *random_integer* 的 topic，在这个 topic 中更新入随机数。subscriber（订阅者）检查并打印出更新值。

topic.h

```
/* declare the topic */
ORB_DECLARE(random_integer);

/* define the data structure that will be published where subscribers can see it */
struct random_integer_data {
        int r;
};
```

publisher.c

```
#include <topic.h>

/* create topic metadata */
ORB_DEFINE(random_integer);

/* file handle that will be used for publishing */
static int topic_handle;

int
init()
{
        /* generate the initial data for first publication */
        struct random_integer_data rd = { .r = random(), };

        /* advertise the topic and make the initial publication */
        topic_handle = orb_advertise(ORB_ID(random_integer), &rd);
}

int
update_topic()
{
        /* generate a new random number for publication */
        struct random_integer_data rd = { .r = random(), };

        /* publish the new data structure */
        orb_publish(ORB_ID(random_integer), topic_handle, &rd);
}
```

subscriber.c

```c
#include <topic.h>

/* file handle that will be used for subscribing */
static int topic_handle;

int
init()
{
        /* subscribe to the topic */
        topic_handle = orb_subscribe(ORB_ID(random_integer));
}

void
check_topic()
{
        bool updated;
        struct random_integer_data rd;

        /* check to see whether the topic has updated since the last time we read it */
        orb_check(topic_handle, &updated);

        if (updated) {
                /* make a local copy of the updated data structure */
                orb_copy(ORB_ID(random_integer), topic_handle, &rd);
                printf("Random integer is now %d\n", rd.r);
        }
}
```

✧ 发布（Publishing）

发布(Publishing)包含三个独立但是相关的过程 :定义(defining)一个 topic，通告（ advertising ）这个 topic，发布（ publishing ）更新。

1. 定义一个 topic（Defining a Topic）

系统已经定义了许多标准的 topic 来为模块间提供通信接口。如果一个发布者（ Publisher ）想要使用这些 topic 及其相关数据结构，不需要做额外的工作。

用户 topic：要定义一个用户 topic，发布者（ publisher ）需要创建一个对订阅者（ subscriber ）可见的头文件（ header file)(可以参考上面的 topic.h 文件)。这个头文件（ header file ）中必须包含如下内容：

- 以这个 topic 的名字为参数的 ORB_DECLARE()宏的一个实例。

- 一个描述将要 publish（发布）的数据结构的结构体定义。

Topic 名字必须是有意义的；PX4 的惯例是“类别_name”；比如，原始传感器数据通过 sensors_raw 这个 topic 来发布。

除了头文件（header file），发布者（publisher）还需在源文件（source file）中添加一个 ORB_DEFINE()宏的实例，这个实例将会在固件被 bulid 的时候被编译和链接（参考上面的 publisher.c 文件）。这个宏创建了一个数据结构体，这个结构体将由 ORB 用来唯一地识别一个 topic 的身份。

如果一个 topic 是由一个软件组件发布的，并且是可选（optional）的，可能不会在固件中出现，那么头文件中可以代替使用 ORB_DECLARE_OPTIONAL（ ）宏。这种方法声明的 topic 可需要专门的句柄（handling），但是下面将会讲到，订阅者在订阅这类 topic 时有一些额外需要考虑的地方。

2. 通告一个 topic（Advertising a Topic）

在数据可以被发布（publish）到一个 topic 之前，这个 topic 必须被通告（advertise）一下。使用的是 orb_advertise（ ）函数，这个函数也将初始化数据发布到了 topic 中。这个函数的原型如下：

```
  /**
 * Advertise as the publisher of a topic.
 *
 * This performs the initial advertisement of a topic; it creates the topic
 * node in /obj if required and writes the initial data.
 *
 * @param meta          The uORB metadata (usually from the ORB_ID() macro)
 *                      for the topic.
 * @param data          A pointer to the initial data to be published.
```

```
 *                      For topics published by interrupt handlers, the
advertisement
 *                      must be performed from non-interrupt context.
 * @return              ERROR on error, otherwise returns a handle
 *                      that can be used to publish to the topic.
 *                      If the topic in question is not known (due to an
 *                      ORB_DEFINE_OPTIONAL with no corresponding
ORB_DECLARE)
 *                      this function will return -1 and set errno to ENOENT.
 */
    externint    orb_advertise(conststruct orb_metadata
*meta,constvoid*data);
```

meta 变量是指向由“ORB_DEFINE（ ）”宏定义的数据的一个指针。通常一个是使用“ORB_ID（ ）”宏来提供的，这个宏起到了一个将 topic name 转换为 topic 原数据结构体 name 的作用。

注意，由于更新可以在中断中被发布，通告（advertise）一个 topic 必须在正常进程中进行。

多个发布者（Multiple Publishers）:一次只能有一个发布者来将一个 topic 通告（advertise），但是 topic handle（句柄）（句柄是一个 file descriptor,可以直接传递给 close（ ）函数）可以由一个 publisher（发布者）关闭，然后由另一个发布者通告（advertise）。

3. 发布更新（Publishing Updates）

一个 topic 被通告（advertise）后，一个由通告函数返回的 handle（句柄）可以用来向 topic 中发布更行。

```
   /**
 * Publish new data to a topic.
 *
 * The data is atomically published to the topic and any waiting subscribers
```

```
 * will be notified.  Subscribers that are not waiting can check the topic
 * for updates using orb_check and/or orb_stat.
 *
 * @handle              The handle returned from orb_advertise.
 * @param meta          The uORB metadata (usually from the ORB() macro)
 *                      for the topic.
 * @param data          A pointer to the data to be published.
 * @return              OK on success, ERROR otherwise with errno set
accordingly.
 */
    externint    orb_publish(conststruct orb_metadata *meta,int
handle,constvoid*data);
```

注意ORB不缓存多个更新，所以当一个订阅者检查一个topic的时候，它只能看到最新的更新。

✧ 订阅（Subscribing）

订阅（subscribling）一个 topic 需要如下步骤：

- 一个 ORB_DEFINE() 或者 ORB_DEFINE_OPTIONAL()宏（比如在由订阅者包含的一个头文件中）
- 一个发布给 topic 的数据结构体的定义（通常也是在那个头文件中）

满足上述条件后，使用如下函数来订阅一个 topic:

```
    /**
 * Subscribe to a topic.
 *
 * The returned value is a file descriptor that can be passed to poll()
 * in order to wait for updates to a topic, as well as orb_read,
 * orb_check and orb_stat.
 *
 * Subscription will succeed even if the topic has not been advertised;
 * in this case the topic will have a timestamp of zero, it will never
 * signal a poll() event, checking will always return false and it cannot
 * be copied. When the topic is subsequently advertised, poll, check,
 * stat and copy calls will react to the initial publication that is
```