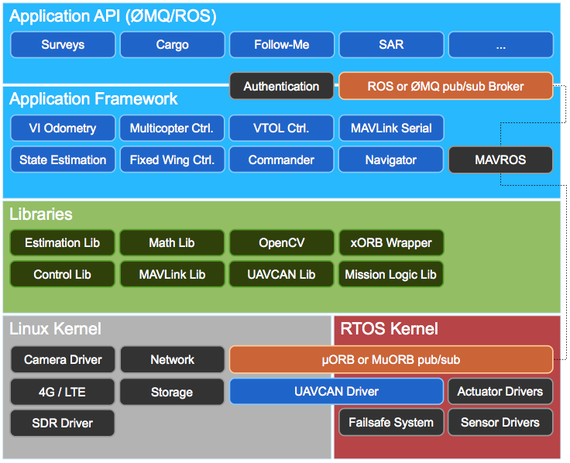
谈谈源代码中频频出现的orb\_xxxxxxxx函数到底是什么系列~

首先，让我们了解一下pixhawk的软件体系结构：



应用程序的API: 这个接口提供给应用程序开发人员，此API旨在尽可能的精简、扁平及隐藏其复杂性。

应用程序框架: 这是为操作基础飞行控制的默认程序集（节点）。

库: 这一层包含了所有的系统库和基本交通控制的函数。

操作系统: 最后一层提供硬件驱动程序，网络，UAVCAN和故障安全系统。

ORB——object request broker 对象请求代理器，主要负责整个系统的数据传输任务（如所有的传感器数据、gps、ppm信号都要从芯片获取吼通过ORB(uORB)进行传输到各个模块进行计算处理）。

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。

还有一些其他的名词——如主题（topic）、订阅（subscribe）、发布（publish）……

主题：进程通过命名的总线交换的信息称之为主题，pixhawk中一个主题仅包含一种消息类型（数据类型）。

订阅&发布：每个进程可以订阅或者发布主题。

具体关系如下 （应用程序框架）：



接下来就是重点了！常用函数功能解析！！！！！！！！！

**int poll(struct pollfd fds[], nfds\_t nfds, int timeout)**

功能：监控文件描述符（多个）；

说明：timemout=0,poll()函数立即返回而不阻塞；timeout=INFTIM(-1),poll()会一直阻塞下去，直到检测到return > 0；

参数：

fds:struct pollfd结构类型的数组；

nfds:用于标记数组fds中的结构体元素的总数量；

timeout:是poll函数调用阻塞的时间，单位：毫秒；

返回值：

>0：数组fds中准备好读、写或出错状态的那些socket描述符的总数量；

==0:poll()函数会阻塞timeout所指定的毫秒时间长度之后返回;

-1:poll函数调用失败；同时会自动设置全局变量errno；

**int orb\_subscribe(const struct orb\_metadata \*meta)**

功能：订阅主题（topic）;

说明：即使订阅的主题没有被公告，但是也能订阅成功；但是在这种情况下，却得不到数据，直到主题被公告；

参数：

meta:uORB元对象，可以认为是主题id，一般是通过ORB\_ID(主题名)来赋值；

返回值：

错误则返回ERROR;成功则返回一个可以读取数据、更新话题的句柄；如果待订阅的主题没有定义或声明则会返回-1，然后会将errno赋值为ENOENT;

eg:

int fd = orb\_subscribe(ORB\_ID(topicName));

**int orb\_copy(const struct orb\_metadata \*meta, int handle, void \*buffer)**

功能：从订阅的主题中获取数据并将数据保存到buffer中；

参数：

meta:uORB元对象，可以认为是主题id，一般是通过ORB\_ID(主题名)来赋值;

handle:订阅主题返回的句柄；

buffer:从主题中获取的数据；

返回值：

返回OK表示获取数据成功，错误返回ERROR;否则则有根据的去设置errno;

eg:

struct sensor\_combined\_s raw;

orb\_copy(ORB\_ID(sensor\_combined), sensor\_sub\_fd, &raw);

**orb\_advert\_t orb\_advertise(const struct orb\_metadata \*meta, const void \*data)**

功能：公告发布者的主题；

说明：在发布主题之前是必须的；否则订阅者虽然能订阅，但是得不到数据；

参数：

meta:uORB元对象，可以认为是主题id，一般是通过ORB\_ID(主题名)来赋值;

data:指向一个已被初始化，发布者要发布的数据存储变量的指针；

返回值：错误则返回ERROR;成功则返回一个可以发布主题的句柄；如果待发布的主题没有定义或声明则会返回-1，然后会将errno赋值为ENOENT;

eg:

struct vehicle\_attitude\_s att;

memset(&att, 0, sizeof(att));

int att\_pub\_fd = orb\_advertise(ORB\_ID(vehicle\_attitude), &att);

**int orb\_publish(const struct orb\_metadata \*meta, orb\_advert\_t handle, const void \*data)**

功能：发布新数据到主题；

参数：

meta:uORB元对象，可以认为是主题id，一般是通过ORB\_ID(主题名)来赋值;

handle:orb\_advertise函数返回的句柄；

data:指向待发布数据的指针；

返回值：OK表示成功；错误返回ERROR；否则则有根据的去设置errno;

eg:

orb\_publish(ORB\_ID(vehicle\_attitude), att\_pub\_fd, &att);

**int orb\_set\_interval(int handle, unsigned interval)**

功能：设置订阅的最小时间间隔；

说明：如果设置了，则在这间隔内发布的数据将订阅不到；需要注意的是，设置后，第一次的数据订阅还是由起初设置的频率来获取，

参数：

handle:orb\_subscribe函数返回的句柄；

interval:间隔时间，单位ms;

返回值：OK表示成功；错误返回ERROR；否则则有根据的去设置errno;

eg:

orb\_set\_interval(sensor\_sub\_fd, 1000);

**orb\_advert\_t orb\_advertise\_multi(const struct orb\_metadata \*meta, const void \*data, int \*instance, int priority)**

功能：设备/驱动器的多个实例实现公告，利用此函数可以注册多个类似的驱动程序；

说明：例如在飞行器中有多个相同的传感器，那他们的数据类型则类似，不必要注册几个不同的话题；

参数：

meta:uORB元对象，可以认为是主题id，一般是通过ORB\_ID(主题名)来赋值;

data:指向一个已被初始化，发布者要发布的数据存储变量的指针；

instance:整型指针，指向实例的ID（从0开始）；

priority:实例的优先级。如果用户订阅多个实例，优先级的设定可以使用户使用优先级高的最优数据源；

返回值：

错误则返回ERROR;成功则返回一个可以发布主题的句柄；如果待发布的主题没有定义或声明则会返回-1，然后会将errno赋值为ENOENT;

eg:

struct orb\_test t;

t.val = 0;

int instance0;

orb\_advert\_t pfd0 = orb\_advertise\_multi(ORB\_ID(orb\_multitest), &t, &instance0, ORB\_PRIO\_MAX);

**int orb\_subscribe\_multi(const struct orb\_metadata \*meta, unsigned instance)**

功能：订阅主题（topic）;

说明：通过实例的ID索引来确定是主题的哪个实例；

参数：

meta:uORB元对象，可以认为是主题id，一般是通过ORB\_ID(主题名)来赋值;

instance:主题实例ID;实例ID=0与orb\_subscribe()实现相同；

返回值：

错误则返回ERROR;成功则返回一个可以读取数据、更新话题的句柄；如果待订阅的主题没有定义或声明则会返回-1，然后会将errno赋值为ENOENT;

eg:

int sfd1 = orb\_subscribe\_multi(ORB\_ID(orb\_multitest), 1);

**int orb\_unsubscribe(int handle)**

功能：取消订阅主题；

参数：

handle:主题句柄；

返回值：

OK表示成功；错误返回ERROR;否则则有根据的去设置errno;

eg:

ret = orb\_unsubscribe(handle);

**int orb\_check(int handle, bool \*updated)**

功能：订阅者可以用来检查一个主题在发布者上一次更新数据后，有没有订阅者调用过orb\_copy来接收、处理过；

说明：如果主题在在被公告前就有人订阅，那么这个API将返回“not-updated”直到主题被公告。可以不用poll，只用这个函数实现数据的获取。

参数：

handle:主题句柄；

updated:如果当最后一次更新的数据被获取了，检测到并设置updated为ture;

返回值：

OK表示检测成功；错误返回ERROR;否则则有根据的去设置errno;

eg:

if (PX4\_OK != orb\_check(sfd, &updated)) {

return printf("check(1) failed");

}

if (updated) {

return printf("spurious updated flag");

}

//or

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);

}

**int orb\_stat(int handle, uint64\_t \*time)**

功能：订阅者可以用来检查一个主题最后的发布时间；

参数：

handle:主题句柄；

time:存放主题最后发布的时间；0表示该主题没有发布或公告；

返回值：

OK表示检测成功；错误返回ERROR;否则则有根据的去设置errno;

eg:

ret = orb\_stat(handle,time);

**int orb\_exists(const struct orb\_metadata \*meta, int instance)**

功能：检测一个主题是否存在；

参数：

meta:uORB元对象，可以认为是主题id，一般是通过ORB\_ID(主题名)来赋值;

instance:ORB 实例ID;

返回值：

OK表示检测成功；错误返回ERROR;否则则有根据的去设置errno;

eg:

ret = orb\_exists(ORB\_ID(vehicle\_attitude),0);

**int orb\_priority(int handle, int \*priority)**

功能：获取主题优先级别；

参数：

handle:主题句柄；

priority:存放获取的优先级别；

返回值：

OK表示检测成功；错误返回ERROR;否则则有根据的去设置errno;

eg:

ret = orb\_priority(handle,&priority);

订阅和发布主题的流程图如下：

