项目申请书

项目名称：uORB 订阅发布机制组件

项目主导师：latercomer

申请人：褚仕成

邮箱：rosemake.csc@outlok.com

<!--

---

1. 项目背景

uORB (Micro Object Request Broker) 是 PX4 开源飞控项目中广泛使用的一种轻量级、实时的进程间/线程间通信消息机制。它通过发布/订阅模式，实现了模块解耦和高效的数据交换，是构建复杂嵌入式系统的关键组件。

RT-Thread是一款成熟、稳定、高度可裁剪的开源实时操作系统（RTOS），广泛应用于物联网、工业控制、消费电子等领域。在 RT-Thread 上实现一套功能完备、性能优异的 uORB 组件，将极大地便利开发者构建需要模块化设计和实时数据交互的应用，例如无人机、机器人控制系统、传感器网络等。

本项目旨在为 RT-Thread 生态系统贡献一个高质量的 uORB 组件，尽可能复制 PX4 uORB 的核心功能和用户体验，并与 RT-Thread 的内核特性深度融合。

导师意见参考： (LaterComer 导师)
希望能够复制 PX4 uORB 的绝大部分功能，包括：

- PX4 uORB 所有的用户接口 `orb_xxx_xx()` 函数。
- `uorb` 命令行工具（如 `uorb top` 等）。
- PX4 uORB 的 `.msg` 消息定义自动生成机制（采用 Python 脚本）。

- 对接 RT-Thread 的设备框架，将每一个 topic 注册为 device (建议)。
- (非必须) 对接 RT-Thread 的 DFS 系统，可以将 topic 作为文件访问。
- 增加 C++ 的兼容层。
- uORB 主题支持单实例和多实例 (instance)。
- 支持 uORB 写回调函数 (publish callback or notification)。

2. 项目目标

根据项目要求和导师意见，本项目的主要目标如下：

1. **设计并实现一个基于 RT-Thread 内核的 uORB 订阅发布机制组件**：
   - 实现核心的 C 语言 API，与 PX4 uORB 的 `orb_*` 系列函数接口保持一致（如 `orb_advertise`, `orb_subscribe`, `orb_publish`, `orb_copy`, `orb_check`, `orb_unadvertise`, `orb_advertise_multi`, `orb_subscribe_multi` 等）。
   - 实现完善的主题管理机制，包括主题的创建、元数据管理、多实例支持。
2. **消息定义与代码生成**：
   - 支持 `.msg` 文件格式定义消息结构。
   - 实现或适配 Python 脚本工具，根据 `.msg` 文件自动生成相应的 C 语言头文件和主题元数据。
3. **优化组件性能**：
   - 确保组件在 RT-Thread 的实时环境下具有低延迟和高吞吐量。
   - 支持多线程并发安全访问，使用 RT-Thread 提供的同步原语（如互斥锁、信号量）。
4. **RT-Thread 系统集成**：
   - 对接 RT-Thread 的设备框架，将每个 uORB 主题（topic）抽象为一个 RT-Thread 设备 (`rt_device_t`)。
   - (可选) 若实现设备框架对接，则可进一步考虑对接 DFS，使得 topic 能通过文件系统路径访问。
   - 集成组件到 RT-Thread 的构建系统中，支持通过 `menuconfig` 配置启用/禁用组件及其子功能。
5. **C++ 兼容层**：
   - 提供一层轻量级的 C++ API 封装，类似于 PX4 中的 `uORB::Publication` 和 `uORB::Subscription`，方便 C++ 应用开发。
6. **命令行工具**：
   - 实现一个 `uorb` 命令行工具，可通过 RT-Thread FinSH/MSH 进行交互。
   - 至少包含 `uorb top` (实时显示主题信息、发布者、订阅者、频率等)、`uorb status` 等子命令。
7. **示例应用**：
   - 提供至少两个示例应用（例如：传感器数据发布模块、控制命令订阅与处理模块），清晰展示组件的实际使用场景和 API 调用方法。
8. **详细的开发文档**：
   - 提交组件架构设计说明文档。
   - 提供完整的 API 参考手册（可通过 Doxygen 等工具从源码注释生成）。
   - 编写详细的使用教程和示例说明。

3. 技术方案与可行性分析

3.1. 整体架构

本 uORB 组件将借鉴 PX4 uORB 的设计思想，并结合 RT-Thread 的特性进行实现。其大致架构如下：

1. **消息定义层 (.msg)**：用户通过简单的 `.msg` 文件定义消息的数据结构。
2. **代码生成层 (Python)**：Python 脚本解析 `.msg` 文件，生成 C 语言头文件（包含消息结构体定义）和 uORB 主题元数据（`struct orb_metadata`）。
3. **uORB核心管理器 (C)**：
    - 负责主题的注册、查找、元数据管理。
    - 管理发布者和订阅者列表。
    - 处理消息的路由和队列。
    - 提供底层的线程安全机制。
4. **C API 层**: 实现 `orb_*` 系列函数接口，供应用程序直接调用。
5. **C++ API 封装层**: 在 C API 基础上提供面向对象的封装，提升易用性。
6. **RT-Thread 适配层**:
    - 使用 RT-Thread 的 IPC 机制（如 `rt_mutex_t`, `rt_sem_t`, `rt_event_t`）保证线程安全和实现阻塞/非阻塞订阅。
    - (建议) 实现 `rt_device_t` 接口，将 topic 注册为 RT-Thread 设备。
7. **命令行接口层**: 实现 `uorb` 命令，通过 FinSH/MSH 与核心管理器交互。

3.2. 核心模块设计

a. 主题元数据 (`orb_metadata`) 与管理

每个主题将关联一个 `orb_metadata` 结构体，这些元数据主要由代码生成脚本根据 `.msg` 文件生成。核心管理器将维护一个全局的主题列表，支持动态查找。

b. 消息发布 (`orb_publish`)

发布者通过 `orb_advertise` 或 `orb_advertise_multi` 声明其发布的主题，并获取一个句柄。之后调用 `orb_publish` 发布消息。核心管理器会将消息放入对应主题的队列中，并根据需要通知订阅者。对于队列已满的情况，可以采取覆盖旧消息的策略（类似 PX4）。

c. 消息订阅 (`orb_subscribe`)

订阅者通过 `orb_subscribe` 或 `orb_subscribe_multi` 订阅感兴趣的主题，并获取一个句柄。

写回调支持: 当新消息发布时，可以设计一种机制通知订阅者。这可以通过 RT-Thread 的事件（event）或信号量（semaphore）实现，或者允许订阅者注册一个回调函数，在发布时由发布路径（或一个专门的通知任务）调用。

d. 消息队列

每个主题实例将拥有一个消息队列。队列大小可在 `.msg` 文件中指定（例如 `# ORB_QUEUE_LENGTH 5`）。对于单元素队列，行为类似于 PX4 的默认情况，只保留最新消息。

e. 多实例支持

API 如 `orb_advertise_multi` 和 `orb_subscribe_multi` 将用于处理同一主题的多个实例。核心管理器需要能够区分和管理这些实例。在设备模型中，每个实例可以映射为一个独立的设备节点（如 `/dev/sensor_accel0`, `/dev/sensor_accel1`）。

f. Python 消息生成脚本

将参考 PX4 `Tools/msg/` 下的 Python 脚本，进行适配或重写，以生成符合本项目需求的 C 代码。脚本将负责：

- 解析 `.msg` 文件。
- 生成消息结构体定义（e.g., `struct topic_name_s {}`）。
- 生成 `orb_metadata` 实例。
- 生成主题 ID 枚举。

g. RT-Thread 设备框架集成

若将 topic 注册为 `rt_device_t`：

- `init`: 初始化 topic 相关的内部数据结构。
- `open`: 对应 `orb_subscribe`，返回一个文件描述符或句柄。
- `close`: 对应 `orb_unsubscribe`。
- `read`: 对应 `orb_copy`。
- `write`: 对应 `orb_publish`。
- `control`: 实现 `orb_check`, `orb_set_interval`, 获取 topic 状态等特定命令。

h. `uorb` 命令行工具

基于 RT-Thread 的 FinSH/MSH 组件，实现 `uorb` 命令。

- `uorb top`: 定期从核心管理器获取所有 topic 的状态（名称、实例数、队列使用情况、发布频率、订阅者数量等），并格式化输出到控制台。
- `uorb status [topic_name]`: 显示特定 topic 或所有 topic 的详细信息。
- `uorb test [topic_name]`: 用于简单的发布/订阅测试。

3.3. 关键技术点

- **线程安全**: 所有对共享数据（如主题列表、订阅者列表、消息队列）的访问都必须使用 RT-Thread 的互斥锁或其他同步原语进行保护。
- **低延迟**: 消息的发布和拷贝路径需要尽可能高效，避免不必要的内存拷贝和长时间的锁占用。
- **实时性**: 阻塞订阅的唤醒机制需要可靠且及时。
- **内存管理**: 组件自身的内存占用，以及消息队列的内存管理需要高效且可控。
- **与 RT-Thread 内核的契合度**: 充分利用 RT-Thread 提供的特性，如设备驱动模型、IPC、FinSH 等。

3.4. 参考项目

本项目将重点参考以下现有实现：

1. **PX4 uORB**:
    - 源代码: https://github.com/PX4/PX4-Autopilot/tree/main/src/lib/uORB (C++ wrappers) 和 https://github.com/PX4/PX4-Autopilot/tree/main/platforms/common/uORB (C core)
    - 消息定义和生成工具: https://github.com/PX4/PX4-Autopilot/tree/main/msg 和 https://github.com/PX4/PX4-Autopilot/tree/main/Tools/msg
    - 这些是本项目功能和 API 设计的主要蓝本。
2. **导师提供的 RT-Thread uORB 实现**:
    - 基于 C++ 的完整实现: https://gitee.com/nextpilot/nextpilot-flight-control/tree/main/pkgs/lib/uORB
    - 基于 C 的部分实现: https://gitee.com/latercomer/rt-thread/tree/uorb/components/utilities/uORB
    - 这些项目为在 RT-Thread 环境下实现 uORB 提供了宝贵的实践经验和代码基础，可以学习其与 RT-Thread 内核的集成方式。本项目目标是在此基础上，完成一个功能更全面、更接近 PX4 uORB 用户体验的 C 版本。

3.5. 可行性分析

uORB 的核心概念相对成熟，并且有 PX4 这一成功的开源项目作为参考。导师提供的现有 RT-Thread uORB 实现也证明了在 RT-Thread 上实现此功能是完全可行的。关键在于细致的设计、对 PX4 uORB 机制的深入理解，以及与 RT-Thread 系统特性的良好结合。通过分阶段开发和充分测试，可以确保项目的成功。

4. 项目开发规划

根据开源之夏的时间节点 (项目开发阶段为 07/01 - 09/30，共三个月)，初步规划如下：

第一阶段：核心功能实现与初步集成 (约 6 周：07/01 - 08/11)

- **Week 1-2: 准备与设计阶段**
    - [ ] 深入学习 PX4 uORB 源码 (C核心部分 `platforms/common/uORB` 和 C++封装 `src/lib/uORB`) 及消息生成机制 (`Tools/msg`)。
    - [ ] 详细研究导师提供的两个 RT-Thread uORB 参考实现。
    - [ ] 完成详细的 uORB 组件数据结构设计和模块接口设计文档。
    - [ ] 搭建 RT-Thread 开发和测试环境。
- **Week 3-4: 核心 C API 与主题管理实现**
    - [ ] 实现核心的主题元数据管理模块。
    - [ ] 实现基础的 `orb_advertise()`, `orb_subscribe()`, `orb_publish()`, `orb_copy()`, `orb_check()`, `orb_unadvertise()` C API (针对单实例)。
    - [ ] 实现消息队列机制（支持可配置队列深度，默认为1）。
    - [ ] 使用 RT-Thread 的互斥锁保证线程安全。
- **Week 5-6: 消息生成与多实例支持**
    - [ ] 适配或重写 Python 脚本，实现从 `.msg` 文件到 C 头文件和元数据的自动生成。

- [ ] 在核心 API 和管理器中加入对多实例的支持 (`orb_advertise_multi`, `orb_subscribe_multi`)。
- [ ] 编写初步的单元测试用例，验证核心功能的正确性。

第二阶段：功能完善与多方面集成 (约 6 周：08/12 - 09/22)

- **Week 7-9: RT-Thread 系统集成与示例应用**
  - [ ] 设计并实现 C++ 兼容层 (`uORB::Publication`, `uORB::Subscription`)。
  - [ ] 开始开发 `uorb` 命令行工具，首先实现 `uorb status` 和基础的 `uorb top` 功能。
  - [ ] (建议) 将 topic 集成到 RT-Thread 设备驱动框架，实现 `rt_device_t` 接口。
  - [ ] (可选) 若设备框架集成完成，尝试 DFS 对接。
  - [ ] 将 uORB 组件集成到 RT-Thread 构建系统，添加 `menuconfig` 配置项。
  - [ ] 开发两个演示 uORB 功能的示例应用（如传感器数据发布、控制命令订阅）。
- **Week 10-12: 测试、优化与文档完善**
  - [ ] 完善 `uorb top` 等命令行工具功能。
  - [ ] 进行多线程并发测试和性能测试，根据结果进行优化。
  - [ ] 编写详细的开发文档：组件架构说明、API 参考手册、使用教程。
  - [ ] 整理代码，添加完善的注释。

最后一周：总结与提交 (09/23 - 09/30)

- [ ] 完成所有代码的最终测试和 Bug 修复。
- [ ] 准备项目成果演示。
- [ ] 提交最终的项目代码、文档和相关产出物。

5. 预期成果与产出

1. **源代码**:
   - 一套完整的、基于 C 语言实现的、功能对齐 PX4 的 RT-Thread uORB 组件。
   - 包含 C++ 兼容层。
   - 包含 `.msg` 文件解析及代码生成的 Python 脚本工具。
   - 包含 `uorb` 命令行工具的实现。
2. **示例代码**:
   - 至少两个演示 uORB 组件使用方法的示例应用程序。
3. **文档**:
   - 详细的组件设计与架构说明文档。
   - 完整的 API 参考手册 (中英文)。
   - 用户使用教程，包含如何在 RT-Thread 项目中集成和使用 uORB 组件的步骤。
4. **项目报告**:
   - 符合开源之夏要求的项目总结报告。
