yijingjing

2023年6月22日 星期四 19:07

IODevice

- 包含一个Location和Socket
- open_writer: 在Location目录下创建一个dest_id的writer
- open_reader
- iodevicemaster/iodeviceclient才有publisher_, observer_

Journal

- location ,dest_id
- 包含当前的frame_, 和当前的page
- Page
  - Frame

writer

- 关联对象
  - 一个location
  - 一个destination
  - 一个Journal
- 可出项相同的location,不同的writer及dest_id
- 用到一个写保护锁
- open_frame, 获取一个数据大小的frame地址，用户用它写数据
- close_frame，指针移到下一个frame的开始处
  - 写完后，如果是非low_lantency模式，调用nanomsg模块publish空的json
- write: 直接写数据
  - 基于open_frame, close_frame
  - 写的时候，如果数据超过页剩余容量大小，翻一页。
    - 当前frame_指针总是指向下一个空的frame header

reader

- 可以join多个Journal, 比如拼接行情和下单委托
- data_available: 是否有数据
  - frame.has_data()
    - header->length > 0

page

- return page_ids.front() -> rolling page
- 一个page对应一个文件
- seek_to_time: 找到nanotime能插入的地方

frame

- Header
  - length
  - header_length
  - gen_time: close_frame的时间，无限接近收到数据的时间
  - trigger_time: use for latency stats
  - source
  - dest
  - msgtype
- 对应一个object
- 包含：
  - 事件的时间
  - 消息类型

location

- 相当于一个目录
- uid: location的标识, 也被用作app的dest_id。 uid = hash_id（category, group, name,mode)
- locator基于home, category, group, name, "journal", mode创建目录，它下面的page就是文件
- 定义了组，用户名，用于对外暴露
- locator属性定义了真正的地址

locator: 根据location的参数获取对应地点的工具类

- env: 自定义的环境变量
- layout_dir: 可能对应着目录，实际并没什么用
- layout_file: 返回参数对应的内存映射的文件路径。每个返回值对应一个journal
  - 文件名: (dest_id + pageid).layout_name
- page id相当于一个文件,对应一个page
- default_to_system_db: 数据库文件地址，系统有个基于它的sql应用, 见TraderCtp中的ordermapper
- page_id: journal的具体页面索引

Hero

- writer_: 多个writer的字典，key为destination_id
- reader_: 一个reader，
- 一个location一定包含一个locator
  - location似乎通过locator属性来获取属性
  - locator应该就是定义了事件的地址
  - 不同的location可以使用相同的locator
- timer, interval的回调注册
- io_device_: iodeviceclient
  - 创建writer
- io_device_client
  - publisher --- PUSH模式
  - observer --- SUBSCRIBE模式
  - 其父类还有一个REPLY模式, 原理不确定

- run: 不间断轮询reader，并publish到events_中
  - Live 模式下，每一轮会检查socket的publish和replay通知，然后读reader_, 100% cpu
  - lazy + live配合，可以不用轮询
  - 也可同时支持2种模式
  - 客户端可以直接从网络上发送数据过来。
  - react: 监听events_
- channels
- locations: 注册地点
- require_write_to/require_read_from: 让某个应用发出/监听请求

apprentice

- 收到requestStart事件后，调用on_start
- live模式下, react -> checking发送Register消息

master: 一个writer, 监听多个app的

- register_app: 以location为参，注册app
  - writers_[0]: master的writer
  - 如果location已经注册，直接返回
  - 创建app location, 用的还是同一个locator
  - 注册app location
  - 基于app location，创建一个成对的master location
  - 创建一个writer, 使用app location.uid做为dest_id
  - reader_监听 (app location, master location)
  - 给writers_[0]写入Register消息
  - 往app location中写入SessionStart
  - 调用require_write_to往app location中写
  - 要求app往master中写消息：app -> master location写
  - 往app location中写入：
    - 注册的location
    - 注册的channel
    - RequestStart
  - 调用on_register(app location)
- deregister_app
  - 往app location中写入session end
- app_locations_:
- 处理事件：
  - Register
  - RequestWriteTo
    - source -> dest_id
    - dest_id <- source -- channel
    - reader_ <- dest_id 监听
  - RequestReadFrom
  - RequestReadFromPublic

事件:

RequestWriteTo: 要求e.source 往 e.dest_id中写入数据
RequestReadFrom: 要求e.dest_id 监听 e.source_id中的数据。

TODO

1. 长时间没收到register
2. requestStart事件
3. Command