



图1: YLENGINE FRAMEWORK

通用缓存器模块用于引擎的数据读写分离，并保证读线程和写线程在操作该模块的时候是线程安全的，以此大幅度提高程序的运行效率。模块允许多个读线程和多个写线程同时对其操作。

如 图1 所示，在 YLENGINE(跃) 视觉引擎中，模板类 `template<DataType> DataBuf` 会被具体化为两个类 `CamDataBuf` 和 `SerialDataBuf` 分别用于处理视觉数据和串口数据。

其中 `DataType` 分别为模板类 `template<RawDataType> Wrapped` 具体化的两个类，即 `Wrapped<ImageData>` 和 `Wrapped<SerialData>`。

version: 2021.1.20

## 1. template <RawDataType> Wrapped API

### 1.1 初始化一个包装器类

### 1.2 包装器对象包装一个数据 (可用于更新包装器内容)

## 2. template <DataType> DataBuf API

### 2.1 初始化一个数据缓存器

### 2.2 从缓存 BUF 中获得最新且未被读过数据

(只会读最新写入的数据，且该数据之前没有被成功读取过)

### 2.3 更新缓存 BUF 中的数据

## 3. ImageData API

### 3.1 实例化 ImageData 方式一

### 3.2 实例化 ImageData 方式二

### 3.3 实例化 ImageData 方式三

### 3.4 实例化 ImageData 方式四

### 3.5 实例化 ImageData 方式五

### 3.6 设置 (更新 ImageData) cv::Mat 数据字段

## 4. SerialData API

### 4.1 实例化 SerialData 方式一

### 4.2 实例化 SerialData 方式二

### 4.3 设置(更新) SerialData 值

## 5. 文件依赖

## 6. 实例代码

附录：稳定性测试

关于：作者联系方式

关于：自空间科技

# 1. template <RawDataType> Wrapped API

包装器模板类 `template<RawDataType> Wrapped` 的作用为给被包装的类加上时间戳。这个时间戳是一个 `short` 值，只要满足长度大于 `DataBuf` 即可，避免出现冲突。同时提供了一个结构 `wrap` 可以更新包装器类内部的包装的数据。

该包装器模板类被设计提供给模板类 `template<DataType> DataBuf` 一个具体化类型参数。

## 1.1 初始化一个包装器类

```
1 template <class RawDataType>
2 Wrapped<RawDataType>::Wrapped(const RawDataType& data_, short
  time_stamp_=clock());
```

初始化的同时自动加上时间戳。

## 1.2 包装器对象包装一个数据 (可用于更新包装器内容)

```
1 template<class RawDataType>
2 void wrap(RawDataType& raw_data_);
```

# 2. template <DataType> DataBuf API

模板类 `template <DataType> DataBuf` 的作用是数据缓存，实现读写数据分离，能够支持多线程读写安全的操作。

## 2.1 初始化一个数据缓存器

```
1 template <class DataType>
2 DataBuf<DataType>::DataBuf(int size=16);
```

## 2.2 从缓存 BUF 中获得最新且未被读过数据

(只会读最新写入的数据，且该数据之前没有被成功读取过)

```
1 template<class DataType>
2 bool DataBuf<DataType>::get(DataType& data, validator v = nullptr)
3
```

```

4 [note]
5 template<class DataType>
6 using validator = bool (*) (const DataType&); // validate data func

```

该缓存模块可以容许多个线程操作写入，但是每次读取数据的时候，即调用 `get` 方法的时候，只会尝试读取最新的数据，如果最新的数据已经被其他线程成功读取过，则这次读取不会成功，不会修改传入参数 `DataType& data` 的内容，并返回 `false`。只有读取的数据没有被成功读取过，才会将该数据写入 `DataType& data`，并会返回 `true`。

validator `v` 是一个数据校验函数指针，可以传入对数据进行校验（下同）。

## 2.3 更新缓存 BUF 中的数据

```

1 template <class DataType>
2 bool DataBuf<DataType>::update(const DataType& data, validator v=
  nullptr);
3
4 [note]
5 template<class DataType>
6 using validator = bool (*) (const DataType&); // validate data func

```

# 3. ImageData API

ImageData 数据包括两个部分：

```

1 uint8_t camera_id; // 1. 该数据来自哪个相机，编号 0, 1, 2, 3, ...
2 cv::Mat mat; // 2. 图像数据 cv::Mat mat;

```

`camera_id` 为 `int` 值时，会自动截取最低的 8 位。

## 3.1 实例化 ImageData 方式一

```

1 ImageData::ImageData();

```

初始化一个相机数据实例，并初始化相机编号为 `uint8_t(0)`，即初始化该数据来自 0 号相机。

## 3.2 实例化 ImageData 方式二

```
1 ImageData(int camera_id_, const cv::Mat& mat_);
```

## 3.3 实例化 ImageData 方式三

```
1 ImageData(const int& camera_id_, const cv::Mat& mat_);
```

## 3.4 实例化 ImageData 方式四

```
1 ImageData(const int& camera_id_);
```

## 3.5 实例化 ImageData 方式五

```
1 ImageData(const uint8_t& camera_id_, const cv::Mat& mat_);
```

## 3.6 设置 (更新 ImageData) cv::Mat 数据字段

```
1 void set(const uint8_t& camera_id_, const cv::Mat& mat_);
```

# 4. SerialData API

SerialData 数据段内容:

```
1 uint8_t camera_id; // 相机编号
2 float pitch; // pitch 角度
3 float yaw; // yaw 角度
4 float distance; // distance 距离
```

## 4.1 实例化 SerialData 方式一

```
1 SerialData();
```

实例化一个 SerialData, 并将 camera\_id, pitch, yaw, distance 均设置为 0。

## 4.2 实例化 SerialData 方式二

```
1 SerialData(uint8_t cam_id, float pitch_, float yaw_, float distance_);
```

实例化一个 SerialData, 设置 cam\_id, pitch, yaw, distance。

## 4.3 设置(更新) SerialData 值

```
1 void set(uint8_t cam_id, float pitch_, float yaw_, float distance_);
```

# 5. 文件依赖

三个文件

```
1 DataBuf, ImageData, SerialData
```

文件路径

```
1 kiko@idiospace:~/robomaster/YLENGINE/Components$ tree DataBuf/  
2 DataBuf/  
3 |— DataBuf  
4 |   └─ DataBuf.hpp // DataBuf 头文件  
5 |— DataIO  
6 |   └─ DataIO.pro  
7 |   └─ DataIO.pro.user  
8 |   └─ main.cpp // 测试源文件  
9 |— ImageData  
10 |   └─ ImageData.hpp // 图像数据文件  
11 |— SerialData  
12 |   └─ SerialData.hpp  
13  
14 4 directories, 6 files
```

文件打包



DataBuf.21.1.20.20...r.lz  
5.04KB

# 6. 实例代码

该代码也可以在以下文件目录获得。

```
1 Components/DataBuf/DataIO/main.cpp
```

```
1  /* kiko@idiospace.com 2021.01*/
2  #include <iostream>
3  #include <opencv.hpp>
4  #include <thread>
5  #include "/home/kiko/robomaster/YENGINE/Components/DataBuf/SerialData/SerialData.hpp"
6  #include "/home/kiko/robomaster/YENGINE/Components/DataBuf/ImageData/ImageData.hpp"
7  #include "/home/kiko/robomaster/YENGINE/Components/DataBuf/DataBuf/DataBuf.hpp"
8
9  using namespace std;
10 using namespace cv;
11 using namespace hnum;
12
13
14 DataBuf <Wrapped<ImageData>> w_img_db;
15 DataBuf <Wrapped<SerialData>> w_serial_db;
16
17 void show_serial_info(const SerialData& serial_data)
18 {
19     std::cout << "cam_id = " << static_cast<int>(
20         static_cast<uchar>(serial_data.camera_id)) << std::endl
21         << "pitch = " << serial_data.pitch << std::endl
22         << "yaw = " << serial_data.yaw << std::endl
23         << "distance = " << serial_data.distance << std::endl;
24 }
25
26 void insert_data()
27 {
28     Mat mat1 = imread("/home/kiko/Pictures/cat.png");
29     Mat mat2 = imread("/home/kiko/Pictures/cat.jpg");
30     Mat mat3 = imread("/home/kiko/Pictures/ice-berg.jpg");
31
32     for (uint i = 0; ; ++i)
33     {
34         Wrapped<ImageData> w_img; // create new wrapped img instance with new timestamp
```

```

35  Wrapped<SerialData> w_serial; // create new wrapped serial instance with new time_stamp
36
37  if (i % 3 == 0)
38  {
39      w_img.raw_data.set(1, mat1);
40      w_serial.raw_data.set(1, 11, 21, 3001);
41  }
42  else if(i % 3 == 1)
43  {
44      w_img.raw_data.set(2, mat2);
45      w_serial.raw_data.set(2, 12, 22, 3002);
46  }
47  else if(i % 3 == 2)
48  {
49      w_img.raw_data.set(3, mat3);
50      w_serial.raw_data.set(3, 13, 23, 3003);
51  }
52
53  if (w_img_db.update(w_img, [&](const Wrapped<ImageData>& w_img_)->bool
54  {
55      return !w_img_.raw_data.mat.empty();
56  })))
57  {
58      std::cout << "[update] times: " << i << " succeeded" << std::endl;
59      std::this_thread::sleep_for(std::chrono::milliseconds(3));
60  }
61  else
62  {
63      std::cout << "[update] failed, lock failed or data corrupted" << std::endl;
64      std::this_thread::sleep_for(std::chrono::milliseconds(5));
65  }
66  if(w_serial_db.update(w_serial))
67  {
68      //std::cout << "[update] times: " << i << " succeeded" << std::endl;
69      std::this_thread::sleep_for(std::chrono::milliseconds(2));
70  }
71  else
72  {

```



```

73  std::cout << "[update] times: " << i << " failed, lock failed or data c
    orrupted" << std::endl;
74  std::this_thread::sleep_for(std::chrono::milliseconds(5));
75  }
76  }
77  }
78
79  void get_data()
80  {
81  Wrapped<ImageData> w_img;
82  Wrapped<SerialData> w_serial;
83
84  for(uint i = 0 ; ; ++i)
85  {
86  w_img_db.get(w_img);
87  if (w_serial_db.get(w_serial))
88  {
89  std::cout << "[get] times: " << i << " succeed" << std::endl;
90  show_serial_info(w_serial.raw_data);
91  std::this_thread::sleep_for(std::chrono::milliseconds(2));
92  }
93  else
94  {
95  //std::cout << "[get] time: " << i << " failed, lock failed or no new d
    ata to get" << std::endl;
96  std::this_thread::sleep_for(std::chrono::milliseconds(2));
97  }
98  }
99  }
100
101  int main()
102  {
103
104  // single thread testing
105  // DataBuf <Wrapped<SerialData>> serial_buf_1; // For Output of Serial 1
106  // DataBuf <Wrapped<SerialData>> serial_buf_2; // For Output of Serial 2
107  // DataBuf <Wrapped<SerialData>> serial_buf_3; // For Output of Serial 3
108
109  // DataBuf <Wrapped<ImageData>> image_buf_1; // For Input of CAM 1
110  // DataBuf <Wrapped<ImageData>> image_buf_2; // For Input of CAM 2
111  // DataBuf <Wrapped<ImageData>> image_buf_3; // For Input of CAM 3

```

```
112
113 // Wrapped <SerialData> wrapped_serial_data;
114 // wrapped_serial_data.raw_data.set(1, 10, 20, 30);
115 // serial_buf_1.update(wrapped_serial_data);
116 // wrapped_serial_data.raw_data.set(2, 20, 30, 40);
117 // serial_buf_2.update(wrapped_serial_data);
118 // wrapped_serial_data.raw_data.set(3, 40, 50, 60);
119 // serial_buf_3.update(wrapped_serial_data);
120
121 // serial_buf_1.get(wrapped_serial_data);
122 // show_serial_info(wrapped_serial_data.raw_data);
123
124 // serial_buf_2.get(wrapped_serial_data);
125 // show_serial_info(wrapped_serial_data.raw_data);
126
127 // serial_buf_3.get(wrapped_serial_data);
128 // show_serial_info(wrapped_serial_data.raw_data);
129
130 // Mat mat1 = imread("/home/kiko/Pictures/cat.png");
131 // Mat mat2 = imread("/home/kiko/Pictures/cat.jpg");
132 // Mat mat3 = imread("/home/kiko/Pictures/ice-berg.jpg");
133
134 // Wrapped<ImageData> wrapped_image_data;
135
136 // wrapped_image_data.raw_data.set(1, mat1);
137 // image_buf_1.update(wrapped_image_data);
138 // wrapped_image_data.raw_data.set(2, mat2);
139 // image_buf_2.update(wrapped_image_data);
140 // wrapped_image_data.raw_data.set(3, mat3);
141 // image_buf_3.update(wrapped_image_data);
142
143 // image_buf_1.get(wrapped_image_data);
144 // std::cout << "id = " << static_cast<int>(static_cast<uchar>(wrapped_i
image_data.raw_data.camera_id)) << std::endl;
145 // imshow("img_data_1", wrapped_image_data.raw_data.mat);
146 // waitKey(0);
147
148 // image_buf_2.get(wrapped_image_data);
149 // std::cout << "id = " << static_cast<int>(static_cast<uchar>(wrapped_i
image_data.raw_data.camera_id)) << std::endl;
150 // imshow("img_data_2", wrapped_image_data.raw_data.mat);
```

```
151 // waitKey(0);
152
153 // image_buf_3.get(wrapped_image_data);
154 // std::cout << "id = " << static_cast<int>(static_cast<uchar>(wrapped_i
mage_data.raw_data.camera_id)) << std::endl;
155 // imshow("img_data_3", wrapped_image_data.raw_data.mat);
156 // waitKey(0);
157
158 // multi-thread testing
159 std::thread insert_thread1(insert_data);
160 std::thread insert_thread2(insert_data);
161
162 std::thread get_thread1(get_data);
163 std::thread get_thread2(get_data);
164
165 insert_thread1.join();
166 insert_thread2.join();
167 get_thread1.join();
168 get_thread2.join();
169
170 std::cout << "hi" << std::endl;
171 }
```

## 附录：稳定性测试

两条读线程+两条写线程同时操作缓存池，无死锁。

```
Terminal
[update] times: 168233 succeeded
[get] times: 416478 succeed
[update] times: 168230 succeeded
[get] times: 416519 succeed
[update] times: 168234 succeeded
[get] times: 416520 succeed
[update] times: 168231 succeeded
[get] times: 416482 succeed
[update] times: 168235 succeeded
[get] times: 416523 succeed
[update] times: 168232 succeeded
[get] times: 416524 succeed
[get] times: [update] times: 416485 succeed
168236 succeeded
[update] times: 168233 succeeded
[get] times: 416527 succeed
[update] times: 168237 succeeded
[get] times: 416528 succeed
[update] times: 168234 succeeded
[get] times: 416489 succeed
[update] times: 168238 succeeded
[get] times: 416490 succeed
[update] times: 168235 succeeded
```

## 关于：作者联系方式

作者：陈汉轩@自空间科技

邮箱： [chenhanxuan@idiospace.com](mailto:chenhanxuan@idiospace.com)

微信：



手机：13040958736

## 关于：自空间科技

自空间科技是一家专注于机器人和物联网开发的科技公司，为各行业提供机器人+的全套创新方案。

公司官网： [idiospace.com](http://idiospace.com)

公司地址：中国，湖南省，长沙市，湖南大学机器人学院 C2 204