Sink型DAQコンポーネント開発について

素核研

千代浩司

DAQコンポーネント分類

- Source型

  （代表例） Gatherer
- Sink型

  （代表例） Logger, Monitor
- その他

  Dispatcher, Filter, Merger

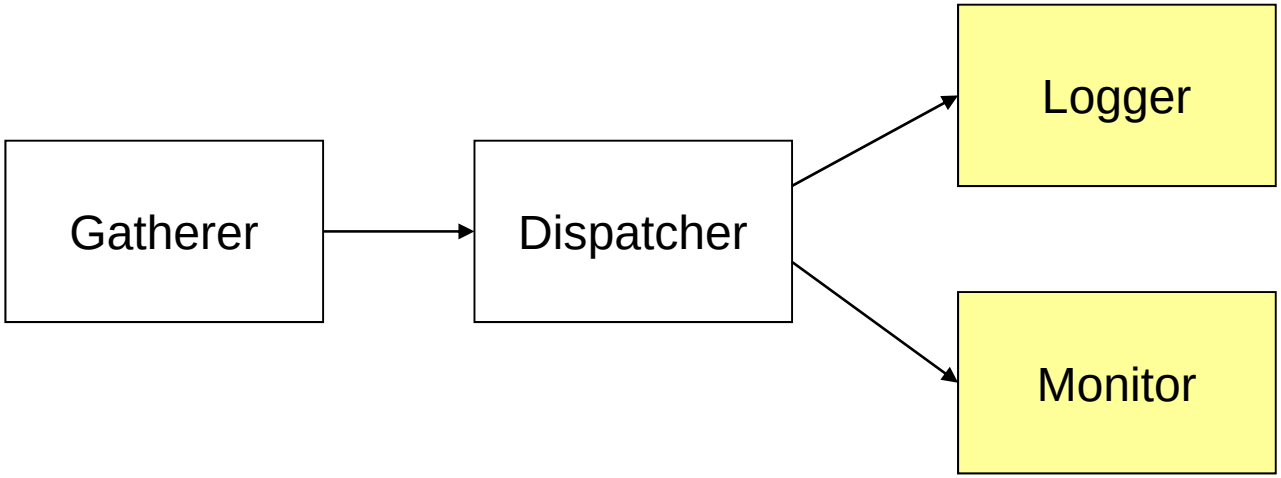
  ここではSink型の開発について話します。

マニュアル

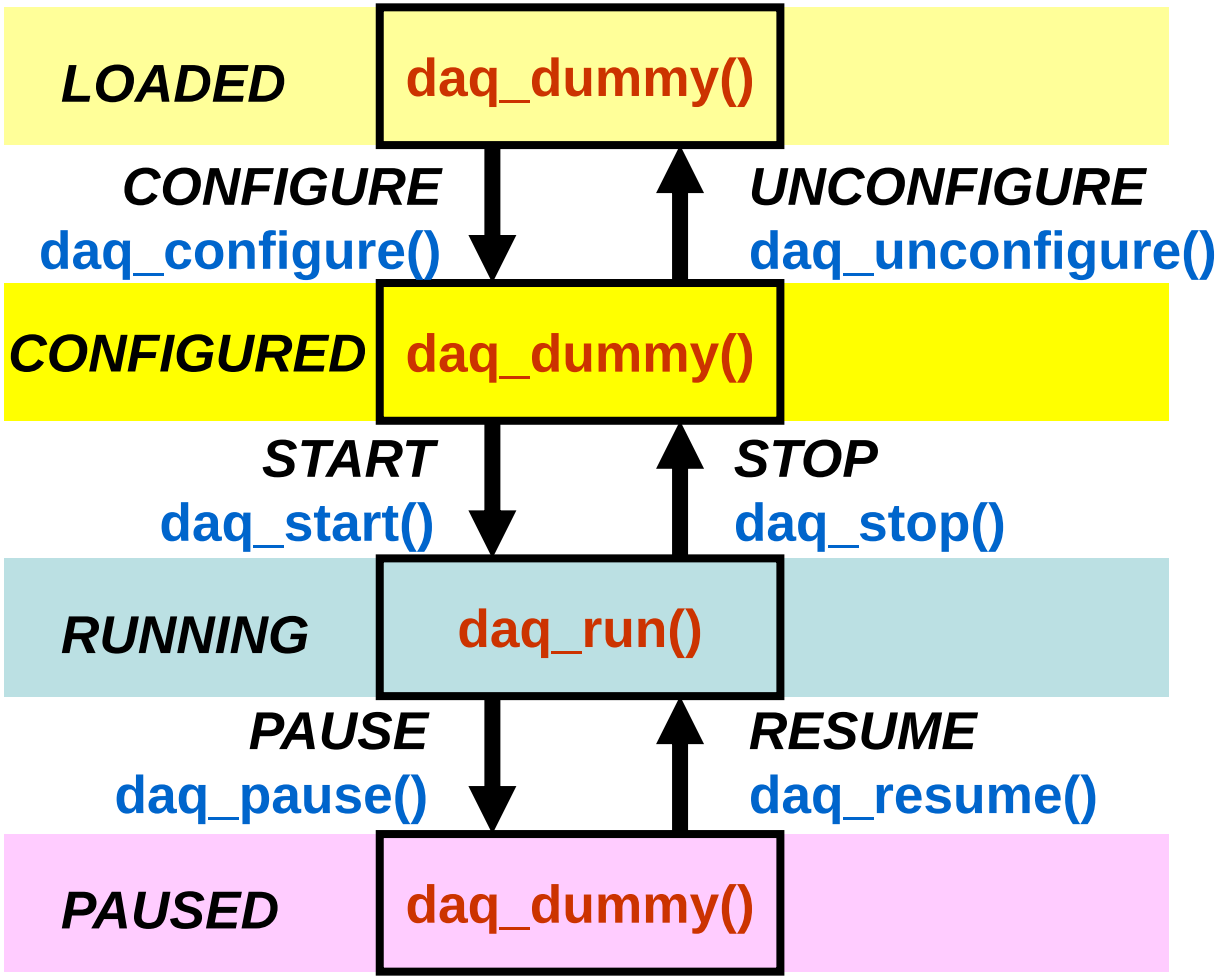
- http://greentea.kek.jp/daqm/docs/sink-comp.pdf

Sink型

- 上流からデータを受け取って仕事をする
  - データのセーブ
  - ヒストグラムの作成　など



ステートチャート



データの読み出し、ヒストグラムへのインクリメント、
ヒストグラム図の作成はdaq_run()内に実装する。

sink型でのdaq_run()の流れ

1. データの受け取り
2. 上流がヘッダー、フッター付で送ってきているならそれらの確認
3. イベントデータのとりだし
4. 取り出したイベントデータをヒストグラムにインクリメントしたりディスクに書いたりする
5. STOPコマンドがきていたかどうか確認

上流DAQコンポーネントからのデータの受け取り

```
// daq_run()
```

m_in_status = **m_InPort.read(m_in_data);**

タイムアウトした場合（データがなかった場合）
BUF_TIMEOUTが返る

エラーの場合にはBUF_FATALが返る

それ以外はデータが読めた。
1回のread()で読めるデータは、上流DAQコンポーネントがそのOutPortに書いた1回分の全部

上流DAQコンポーネントからのデータの受け取り（2）

```
// daq_run()

m_in_status = m_InPort.read(m_in_data);

if (m_in_status == BUF_TIMEOUT) { // Timeout.  No data
    if (check_trans_lock()) { // got STOP command
        set_trans_unlock();
    }
    return 0;
}
else if (m_in_status == BUF_FATAL) { // Fatal Error
    fatal_error_report(USER_ERROR1, -1);
    return 0;
}
```

上流DAQコンポーネントが送ってきたデータ長の取得

m_in_status = m_InPort.read(**m_in_data**);

len = m_in_data.**data.length();**

上流コンポーネントがヘッダー、フッターをつけてOutPortに書いたならlenはヘッダー、フッターの長さを含んだ長さになる。

上流コンポーネントから
送られてきたデータ： m_in_data.data[ ]

m_in_status = **m_InPort.read(m_in_data);**
len = m_in_data.**data.length();**

unsigned char *ptr;
ptr = &m_in_data.data[0]

m_in_data.data[0]
m_in_data.data[HEADER_BYTE_SIZE - 1]

m_in_data.data[HEADER_BYTE_SIZE]
m_in_data.data[len - HEADER_BYTE_SIZE - FOOTER_BYTE_SIZE - 1]

m_in_data.data[len - FOOTER_BYTE_SIZE]
m_in_data.data[len - 1]

上流コンポーネントから受け取れたデータを処理

- あとはヒストグラムを作ったり、ディスクに書いたりすればよい。

EchoMonitor(1)

```
m_in_status = m_InPort.read(m_in_data);
 if ((m_in_status == BUF_TIMEOUT) && check_trans_lock()) {
      set_trans_unlock();
      return 0;
  }
  else if (m_in_status == BUF_TIMEOUT) {
      return 0;
  }
  else if (m_in_status == BUF_FATAL) {
      fatal_error_report(USER_ERROR1, -1);
      return 0;
  }
```

EchoMonitor(2)

```
/////////////// Get Data Length ///////////////

unsigned int block_byte_size = m_in_data.data.length();

unsigned int event_byte_size = block_byte_size - HEADER_BYTE_SIZE -
                                                 FOOTER_BYTE_SIZE;
```

EchoMonitor(3)
check_header(), check_footer()

```
/////////////// Check Header and Footer ///////////////

unsigned char header[HEADER_BYTE_SIZE];

//// header copy
for (unsigned int i = 0; i < HEADER_BYTE_SIZE; i++) {
        header[i] = m_in_data.data[i];
}

//// header check
if (check_header(header, event_byte_size) == false) {
        std::cerr << "### ERROR: header invalid in EchoMonitor" << std::endl;
        fatal_error_report(HEADER_DATA_MISMATCH, -1);
        return 0;
}
```

FOOTERも同様
check_footer(footer, m_loop)を使う

EchoMonitor (4)
単純に取り出し

```
////////// Extract each event data and print to STDERR //////////

unsigned int *event_data;

for (unsigned int i = HEADER_BYTE_SIZE;
        i < block_byte_size - FOOTER_BYTE_SIZE;
        i += EVENT_BYTE_SIZE)  {

        event_data = (unsigned int *) &m_in_data.data[i];

        std::cerr << "Event Data: " << *event_data << std::endl;
}
std::cerr << "*** Data Extraction End" << std::endl;
```

EchoMonitor(4')
ヒストグラムにフィル、ヒストグラムを書く

```
////////// Extract each event data and fill to histogram data //////////

unsigned int *event_data;

for (unsigned int i = HEADER_BYTE_SIZE;
        i < block_byte_size - FOOTER_BYTE_SIZE;
        i += EVENT_BYTE_SIZE) {
        event_data = (unsigned int *) &m_in_data.data[i];
        m_histo->Fill(*event_data);
}

if (m_loop % m_monitor_update_rate == 0) {
        m_histo->Draw();
        m_canvas->Update();
}
```

m_loop, m_total_event

```
m_loop++; //   daq_run()でデータが読めた回数
          //  上流コンポーネントがOutportに書いた
          //   回数と一致するはず

m_total_event += event_byte_size / EVENT_BYTE_SIZE;
// DAQオペレータから何イベント処理したか聞かれたらこの
// 値を答える
```

STOPコマンドがきていたかどうかの確認

```
if (check_trans_lock()) {  // got stop command
    set_trans_unlock();
    return 0;
}

return 0; // end of daq_run()
```