xquant： broadcast.cpp源码-结构

多个APP的情况

https://github.com/uNetworking/uWebSockets/issues/1631

uqant/src/broadcast.cpp

defer

`uWS::Loop::get()` 可以在APP 之前，也可以在之后。 app.run() 实际上是 loop.run()

loop.h中如下

```
1   /* Can be called from any thread to run the thread local loop */
2   inline void run() {
3       Loop::get()->run();
4   }
```

在broadcast.cpp中可以这样写： `uWS::run();`

```
1       // app.run();
2       uWS::run();
```

一个线程，可以有多个app，在一个loop.run()，在一个loop循环里。 loop循环里的回调函数，不可阻塞。

```C++
    struct us_loop_t *loop = (struct us_loop_t *)uWS::Loop::get();
    struct us_timer_t *delayTimer = us_create_timer(loop, 0, 0);
```

https://github.com/uNetworking/uWebSockets/issues/1592

```C++
void
LoopFunc(uWS::WebSocket<false, true, PerSocketData> *ws)
{
    std::string name;
    while(true) {
        std::cout << " Loop Thread: " << std::this_thread::get_id()
                  << std::endl;
        name = ws->getUserData()->name;

        /* Check break condition. */

        sleep(5); // Simulate calculating.
        std::string result = "Mission Complete! " + name;
        ws->getUserData()->loop->defer([ws, result]() {
            ws->send(result, uWS::TEXT, true);
        });
    }
}
```

https://github.com/uNetworking/uWebSockets/discussions/1120

```C++
    void removeAllClients()
    {
    std::unique_lock<std::mutex> lock(mMutex);
        for (auto & it : mClients)
        {
            auto ws = it.second;
            mLoop->defer([ws]()
                {
                    ws->close();
                });
        }

        mClients.clear();
    }
```

写法：

C++

```cpp

struct PerSocketData {
    /* Fill with user data */
    uWS::Loop *loop;
    std::string name;
};

void LoopFunc(uWS::WebSocket<false, true, PerSocketData> *ws) {
    std::string name;
    while (true) {
        std::cout << " Loop Thread: " << std::this_thread::get_id() << std::endl;
        name = ws->getUserData()->name;
        std::cout << " Loop  get: " << std::this_thread::get_id() << name << std::endl;

        /* Check break condition. */

        std::this_thread::sleep_for(std::chrono::milliseconds(1000));
        // sleep(1);
        std::cout << " Loop  get: " << std::this_thread::get_id() << name << std::endl;
        std::string result = "Mission Complete! " + name;
        std::cout << " Loop Thread defer B: " << std::this_thread::get_id() << std::endl;
        ws->getUserData()->loop->defer([ws, result]() { ws->send(result, uWS::TEXT, true); });
        std::cout << " Loop Thread defer E: " << std::this_thread::get_id() << std::endl;
    }
}
```

```C++
                .open =
                    [](auto *ws) {
                        /* Open event here, you may access ws->getUserData() which points to a
                         * PerSocketData struct */
                        // ws->subscribe("broadcast");
                        auto x = ws->getUserData();
                        x->name = "tttt";
                        x->loop = uWS::Loop::get();
                        std::thread th1(LoopFunc, ws);
                        th1.detach();
```

向多线程发送publish

https://github.com/uNetworking/uWebSockets/discussions/1050

```cpp
/*
   This example demonstrates basic idea of handling multi-threaded uWebSockets sever.
   There may be better ways to capture and use uWS::Loop and uWS::App objects.
*/

#include <thread>
#include <vector>
#include <memory>
#include "uWebSockets/App.h"

/* ws->getUserData returns one of these */
struct PerSocketData {
  std::string_view user_secure_token;
};

/* uWebSocket worker runs in a separate thread */
struct worker_t
{
  void work();

  /* uWebSocket worker listens on separate port, or share the same port (works on Linux). */
  struct us_listen_socket_t *listen_socket_;

  /* Every thread has its own Loop, and uWS::Loop::get() returns the Loop for current thread.*/
  struct uWS::Loop *loop_;

  /* Need to capture the uWS::App object (instance). */
  std::shared_ptr<uWS::App> app_;

  /* Thread object for uWebSocket worker */
  std::shared_ptr<std::thread> thread_;
};

/* uWebSocket workers. */
std::vector<worker_t> workers;

/* uWebSocket worker thread function. */
void worker_t::work()
{
  /* Every thread has its own Loop, and uWS::Loop::get() returns the Loop for current thread.*/
```

```
41    loop_ = uWS::Loop::get();
42
43    /* uWS::App object / instance is used in uWS::Loop::defer(lambda_functi
      on) */
44    app_ =  std::make_shared<uWS::App>();
45
46    /* Very simple WebSocket broadcasting echo server */
47    app_->ws<PerSocketData>("/*", {
48      /* Settings */
49      .compression = uWS::SHARED_COMPRESSOR,
50      .maxPayloadLength = 16 * 1024 * 1024,
51      .idleTimeout = 0,
52      .maxBackpressure = 1 * 1024 * 1204,
53      /* Handlers */
54      // Refer to  6735bad  commit. Don't pass req in open handler
55      // https://github.com/uNetworking/uWebSockets/commit/625efbc499460a21
      d27c5811ce7c948a69a04eea
56      .open = [](auto *ws) {
57        /* Let's make every connection subscribe to the "broadcast" topic.
58           Since our example server runs multiple worker threads, the clien
      t connection
59           may be served by any of the worker thread.
60           However, this example demonstrates publishing message(s) to all
      subscribers
61           served by all workers (threads).
62        */
63        ws->subscribe("broadcast");
64      },
65      .message = [this](auto *ws, std::string_view message, uWS::OpCode opC
      ode) {
66        /* Exit gracefully if we get a closedown message (ASAN debug) */
67        if (message == "closedown") {
68          /* Bye bye */
69          us_listen_socket_close(0, listen_socket_);
70          ws->close();
71        }
72        /* When an "alert" message is recieved on any worker thread, it sim
      ply invokes
73           uWS::Loop::defer(funcion) of each thread.
74           Loop::defer(function) is the only thread-safe function here; yo
      u use it to defer
75           the calling of a function to the thread that runs the Loop. Use
      this to signal
76           every Loop that it should publish some message.
77        */
78        if (message == "alert") {
79          /* Simply broadcast alert message we get */
80
```

```
            std::for_each(::workers.begin(), ::workers.end(), [message, opCode](worker_t &w) {
              /* uWs::Loop of each worker. */
              w.loop_->defer([w, message, opCode]() {
                  /* uWs::App of each worker. uWs::App object has knowledge of all socket contexts. */
                  w.app_->publish("broadcast", message, opCode);
              });
            });
          }
          /* Simple echo the message to the client. */
          else {
            ws->send(message, opCode);
          }
        },
        .drain = [](auto *ws) {
          /* Check getBufferedAmount here */
        },
        .ping = [](auto */*ws*/, std::string_view) {
              /* Not implemented yet */
        },
        .pong = [](auto */*ws*/, std::string_view) {
              /* Not implemented yet */
        },
        .close = [](auto *ws, int code, std::string_view message) {
          /* We automatically unsubscribe from any topic here */
        }
    }).listen(9001, [this](auto *token) {
        listen_socket_ = token;
        if (listen_socket_) {
          std::cout << "Thread " << std::this_thread::get_id() << " listening on port " << 9001 << std::endl;
        }
        else{
          std::cout << "Thread " << std::this_thread::get_id() << " failed to listen on port " << 9001 << std::endl;
        }
    }).run();

    std::cout << "Thread " << std::this_thread::get_id() << " exiting" << std::endl;
}

/* Main */
int main() {

    workers.resize(std::thread::hardware_concurrency());
```

```
    std::transform(workers.begin(), workers.end(), workers.begin(), [](work
er_t &w) {
        w.thread_ = std::make_shared<std::thread>([&w]() {
          /* create uWebSocket worker and capture uWS::Loop, uWS::App object
s. */
          w.work();
        });
        return w;
    });

    std::for_each(workers.begin(), workers.end(), [](worker_t &w) {
        w.thread_->join();
    });

    return 0;
}
```

1. 一个APP可以监听多个端口，共用一个loop

```
    app.listen(9001, [](auto *listen_socket) {
        if (listen_socket)
            std::cout << "Listening on port " << 9001 << std::endl;
    });

    app.listen(5001, [](auto *listen_socket) {
        if (listen_socket)
            std::cout << "Listening on port " << 5001 << std::endl;
    });
```

1.1. 旧版uws的client同一个hub也可以连接多个服务进程

```cpp
void Base::MainHubConnection() { // 这里是进行实际TCP连接
    if (forweb_) {
        main_hub_->connect("ws://localhost:5001", (void *)6); //  data server 将数据转给显示服务进程，由其转发给web端。
        std::this_thread::sleep_for(std::chrono::milliseconds(1000));
    }

    switch (this->mode_) {
    case Mode::BACK_TEST:
        std::cout << " Base::MainHubConnection()   Mode::BACK_TEST" << std::endl;
        // 后面这个  (void *)2 ,  可以
        main_hub_->connect("ws://localhost:9001", (void *)2); //  data server HubBck //回测
        break;
    case Mode::REAL_CTP:
        main_hub_->connect("ws://localhost:3003", (void *)3); //  data server HubCtp  //实盘
        break;
    case Mode::SIMULATION:
        main_hub_->connect("ws://localhost:3004", (void *)4); //  data server HubSim  //模拟
        break;
    default:
        main_hub_->connect("ws://localhost:3004", (void *)5); //  data server HubSim  //模拟
        break;
    }
}
```

1.2. mainhub 建立连接过程 【base.cpp】

`void Base::Run()` ==> `Base::MainHubConnection();`

问题：断线重连时，那些没有断线的，也会重新连接？如何处理？

请看： `Base::MainHubOnDisconnInit()` 这里有一个 `for (;;)`

还有： `Base::MainHubOnErrorInit()` 这里也有一个 `for (;;)`