【TCP/IPを用いた複数ロボット通信用ライブラリ】

RobotClient、RobotServerはTCP/IPを利用した複数端末間での通信をサポートするライブラリである。RobotServerを用いてサーバーを構築し、RobotClientを用いてサーバーにデータの保存と問い合わせを行うサーバークライアントモデルで、複数端末間での通信を実現する。

両ライブラリは簡略化されているため、ここの使い方を見るだけでTCP/IPを利用した安定したサーバークライアントモデルの通信を実装することが出来る。これらのライブラリを扱うには、Boostのライブラリと同梱されたヘッダーファイルをインクルードしなければならない。

RobotClient

# 関数の解説

・network::RobotClient(std::string address,int port,int timeout,std::string matchCondition, boost::asio::io\_service ioService)

戻り値 無し

第一引数 サーバーのアドレス。”127.0.0.1”のような形式。std::string型

第二引数 サーバーの接続するポート。int型

第三引数 タイムアウトするまでの時間(秒単位)。int型

第四引数 通信を打ち切る文字列。std::string型

第五引数 ライブラリ内で使用するboost::asio::io\_service型の変数

コンストラクタ。

・void connect()

戻り値 無し

引数 無し

サーバーと接続する

・void set(std::map<std::string,int>)

戻り値 無し

第一引数 サーバーに保存したい変数。std::map<std::string,int>型

サーバーにstd::map<std::string,int>型の変数を自身が接続しているポート番号をkeyとして保存する。

・std::map<std::string,int> get(std::string address)

戻り値 サーバーに保存されていたデータ

第一引数 欲しいデータのkeyとなっているアドレス。std::string型

サーバーに欲しいデータのkeyを送り、そのkeyに対するvalueとして保存され

ているstd::map<std::string,int>型の変数を返す。

・void get(std::string address, std::map<std::string,int>& data)

戻り値 なし

第一引数 欲しいデータのkeyとなっているアドレス。std::string型

第二引数 サーバーに保存されていたデータを格納したい変数map型

サーバーに欲しいデータのkeyを送り、そのkeyに対するvalueとして保存され

ているデータを第二引数で与えられた変数に格納して返す。ひとつ前の関数と

較してこちらはいちいちstd::map<std::string,int>型の変数を生成しないので効率が良い。

# 例

1 #include "RobotClient.hpp"

2

3 int main(){

4 boost::asio::io\_service ioService;

5 network::RobotClient client("127.0.0.1",5000,5,"\n",ioService);

6 client.connect();

7 std::map<std::string, int>setData;

8 int i = 0;

9 while (1) {

10 setData["x"] = i++;

11 client.set(setData);

12 std::cout << client.get(192.168.0.1)["a"] << std::endl;

13 std::cout << client.get(192.168.0.2)["b"] << std::endl;

14 std::cout << client.get(192.168.0.3)["c"] << std::endl;

15 }

16 return 0;

17 }

4行目 ライブラリで使用するboost::asio::io\_service型の変数を定義

5行目 サーバーのアドレスを127.0.0.1、接続先のポートを5000番、5秒でタイムアウト、”\n”で受信打ち切りとしてRobotClientのインスタンスを生成する

6行目 サーバーと接続する

11行目 サーバーに自身の持つ変数setDataを保存する

12行目 サーバーに192.168.0.1から接続している端末が保存した変数の”a”keyに対応

するvalueを出力する

13行目 サーバーに192.168.0.2から接続している端末が保存した変数の”b”keyに対応

するvalueを出力する

14行目 サーバーに192.168.0.3から接続している端末が保存した変数の”c”keyに対応

するvalueを出力する

RobotServer&MultipleTransfer

# 関数の解説

RobotServerは単独では1:1の通信しか出来ないため、複数の通信を実現するためにはマルチスレッドの知識が必要になる。なので、そのような手間を省くために、MultipleTransferと言うRobotServerのラッパーライブラリを作成した。基本的にはRobotServerを直接扱わず、こちらを利用して欲しい。

・network::MultipleTransfer(int size,int port, std::map<std::string,std::map<std::string,int>>& data, boost::mutex& mtx)

戻り値 無し

第一引数 接続先の受入数。

第二引数 通信を行うポートの指定。

第三引数 クライアントからのデータを保存する変数

第四引数 ミューテックス変数。

コンストラクタ。

・MultipleTransfer::start()

戻り値 無し

引数 無し

通信を開始する。

・MultipleTransfer::stop()

戻り値 無し

引数 無し

通信を終了する。

基本的に、インスタンスを生成したらstart()関数を呼び出すだけで後は放置していれば動く。

# 不具合

MultipleServerでは切断の際のエラー出力を出さないように例外処理を上書きしているが、エラー出力が出てしまう場合がある。

●ライブラリの詳細

・両ライブラリの共通事項

　両ライブラリ共に、TCPというライブラリのラッパーとなっている。このライブラリは私が作成したBoostのラッパーライブラリである。これについて解説するとキリがないので自分でソースコードを読んで理解して欲しい。TCP通信を実装することが出来るこのライブラリを用い、その上に各種文字列変換、解析用の関数を実装したものがRobotClient、RobotServerとなっている。

　両ライブラリともにSimpleMapSerializationというネームスペースに属する関数を使用している。これは与えられたmap型の変数と文字列を、指定された区切り文字を使用して相互に変換する関数である。

・RobotClient

・コンストラクタ

引数で与えられたパラメータを元にTCPクラスのインスタンスを作成する。

・set

引数で与えられたmap型の変数をSimpleMapSerialization::simpleMapDeSerializer()関数を用いて文字列に変換、変換した文字列の先頭に、文字列setを付与してサーバーに送信する。

・connect

サーバーと接続しTCP通信を開始する。

・get

　サーバーに、先頭に文字列get、クライアントのIPアドレスで構成された文字列を送信する。その後サーバーから受信した文字列をmap型の変数に変換し戻り値として返す。又は、参照渡しされた引数に格納する。

・RobotServer&MultipleTransfer

　RobotServerから解説をする

　・コンストラクタ

引数で与えられたパラメータを元にTCPクラスのインスタンスを作成する。

・waitingClient

クライアントからの通信を待つ。通信が行われた時、受信した文字列の先頭を確認し、setであればset関数、getであればget関数に受信した文字列から命令を削除した文字列を変数として渡し、実行する。

・set

変数として与えられた文字列をmap型に変換し、クライアントのIPアドレスと紐付けて保存する。

・get

変数として与えられた文字列からIPアドレスと紐付いたmap型の変数を取得し、文字列に変換、クライアントに送信する。

・getPrefix

受信した文字列の先頭についた命令文を取得する。

・deletePrefix

受信した文字列の先頭についた命令文を除去する

MultipleTransfer

・コンストラクタ

与えられたパラメータを用いてメンバ変数の初期化を行う

・start

通信を開始するために、通信受け入れスレッドを実行する。

・stop

全てのスレッドを終了し、通信を終了する。

・accepter

通信受け入れスレッド。通信を受け入れたRobotServerクラスのインスタンス一つ一つ　　に通信用のスレッドを割り当て実行する。

・transfer

accepterで割り当てられ、実行するスレッド。通信の切断を感知すると切断されたクライアントの保存したデータを消去する。

●ライブラリの改良点

もし、もっと時間があればこんな改良をしても良かったなぁと反省している部分。

・変数の文字列変換部分の改良

　　このライブラリで行ったような変数を文字列に変換しテキストデータとして保存したり、通信に利用できる形にすることを直列化(シリアライズ)という。このシリアライズには幾つか形式や方法があり、その為にXMLやJSONと言った規格がある。しかし私はそれらを使わずに、独自にカンマ区切り形式のシリアライザを行う関数を作った。理由として、元々RaspberryPiはWiFi通信のみを行い、通信内容をそのままマイコンにシリアル通信で送信するためのブリッジとして利用していた。また、マイコン間の通信にもこれを適用し、nbitが0or1の時はこういう処理をして…といったわかりにくい実装から”stop”キーの値が1の時に止まるといったようなより分かりやすい実装が行いたかったのである。しかし、両者ともに通信速度がWiFiと比較して非常に遅いシリアル通信を使うため、XMLでは無視できない遅延が発生した。その為に、データ数のより少ないカンマ区切り形式でシリアライズを行ったのである。しかし、データ数が少なくなった一方で、自由度は減り、XMLやJSONは型をあまり考えずにどんな値でも送信できたが、単純なカンマ区切り形式では、型が一つに固定されてしまう。ところで、XMLでは遅延が発生したと書いたが、実はJSONについては検証をしていない。JSONはXMLよりも軽量な言語なので、もしかしたら遅延を許容範囲内に収めつつ自由度を増すことが出来るかもしれない。Boost.FusionとBoost.PropertyTreeを利用して、より良く出来たのではないかと思う。

・演算子のオーバーロードの活用

　　　RobotClientはget関数を利用して変数を取得しているが、[]演算子をオーバーロードして インスタンス名[IPアドレス] でデータを取得できたほうが直感的だったのではないかと思う。