Media Programing 1 - openFrameworks
- proc.04 - 2010.05.18 hrr

今回のテーマ

インターネットの（超）基礎
ネットワークプログラミング

インターネットの（最低限の）基礎

用語から解説

インターネット：
インターネットは、世界中のたくさんのネットが相互接続されて構成されている。
インターネット自体は誰かが管理しているものではなく、各ネットは各々管理/運営
され、それらが相互に接続していることで機能している。
網の目状につながっている為、常に同じ経路で通信が行なわれるわけではないし、
すべてのネットワークが正常に稼働しているとも限らない。
そのため通信におけるデータの信頼性は低くなるが、あるネットにトラブルが
発生して通信できなくなっても別の経路で通信を確保できるため、
障害に強く柔軟性が高いともいえる。

IPアドレス：
ＩＰ（Internet Protocol)とはインターネットを使う上での約束事であり、
これに従ってインターネット上にIPネットワークが構成されている。
ＩＰアドレスはネットワークに繋がっているものを識別するための番号になる。
ネットワークに繋がっている各マシンやルータなど、インターネットにつなぐ
ために必須である。
ＩＰアドレスには、各組織に割り当てられたユニークな番号（グローバルアドレス）と、
自由に使ってよい番号（プライベートアドレス）がある。
グローバルアドレスはインターネット上でユニークなので通信先を一意に決定できるが、
プライベートアドレスは誰でも使えるので番号が重なってしまう。そのためローカルな
LAN内だけで利用できる。
現在のIP v4では大量のコンピュータや機器のすべてにグローバルアドレスは
与えられないので、対外的にはグローバルアドレスを持ち、内部では
プライベートアドレスを使うことが多くなっている。この場合、インターネットに
出る際にプライベートアドレスは、グローバルアドレスに変換される。
IP v6ではアドレス空間が広がりすべての機器にグローバルアドレスを

割り当てることも可能になっている。
ちなみに、DNSによってグローバルアドレスをドメイン名が割り当てられている為,
アドレスではなくドメイン名でインターネットにアクセスできるようになっている。
例： 202.43.240.2 （グローバル）, 192.168.*.* や　61.*.*.* など、8ビット（0-255)までの
４つの数値の組み合わせ（計32ビット）で示す.

イーサネット：
ethernet はネットワークの物理的な規格で、コネクタの形状や物理的なデータ通信の
規格が決められている。イーサネットによる物理的な規格とTCP/IPによる通信規約に
よってインターネットができているともいえる。

MACアドレス：
Media Access Controlアドレスはネットワークカードなどネットワーク機器をユニークに
識別するための番号である。ネットワークカードなどのの製造メーカーの番号と
製品一つ一つに違う番号が割り当てられるため、MACアドレスから機器を
特定できるようになっている。
例：01-1E-8C-72-*-* など計48ビットで表す.上位24ビットが機器メーカー。

TCP/IP
TCP(Transmission Control Protocol)とIPを総称した通信プロトコルで、
インターネット上でもっとも一般的に使われているプロトコル。
通信するためには、データをある程度の大きさのパケットに分けて送信する。
パケットには宛先のアドレスやデータ自身が入っている。
インターネット自体ではデータの信頼性を確保できないので、パケットを送り出しても
正しい順番で到着しないこともあるし、途中でパケットが失われることもある。
そのため、TCP/IPではデータやり取りの手順を決め、データが失われたら
再送信の要求をしたり、パケットの順番を正しい順番にしたりすることで
通信の信頼を確保している。
このようにお互いにやり取りしながら通信するので、仮想的に通信路を確保した安定した
通信となる。

UDP：
User Datagram Protocol はTCPのようにセッションを確保しないで、
垂れ流し的にデータを送る通信方法。
そのため、パケットが失われたり、順番が違う可能性がある。
ただし、セッションを確保しないのでLAN内の多数の機器に一斉送信するbroadcastが
できる。

ポート番号：

IPアドレスで通信先の情報はわかるが、どのプログラムがデータを受け取るかを指定するのがポート番号になる。お互いに番号を決めておいて、そのポート番号でデータのやり取りを行なう。ポート番号は0から65535まで指定できるが,若い番号はmailやhttpなど有名なプロトコル用となっているため、1024以降の大きい番号（10000以上とか）を使うようにする。

プロトコル：
httpやftp,mailなどはアプリケーションプロトコル、一般的にはプロトコルと呼ばれ、アプリケーションやサービスに対応したデータの形式や手順を定めたものである。
たとえば、httpならクライアントであるブラウザからWebサーバに対して、
GET /
などと送ると、トップページのhtmlが返ってくる。
クライアントとサーバの間のリクエストや応答などを取り決め、その手順を守ることで,ブラウザやhttpサーバが実現されている。

詳しくは、ネットワーク概論でやるはず。
自分で調べるならOSI参照モデルとかを調べると良いかも。

ネットワークプログラミング

openFrameworksのadd-onにあるofxNetworkを使ってプログラミングを行う。
このライブラリには、TCP/IPやUDPによる通信を簡単に行なえるクラスが用意されている。
ネットワークのプログラミングでは、データの読み込み時にデータが来るまで読み込み待ち状態（ブロッキング）になることに注意する。データが来ないとそこで処理が止まってしまい何もできなくなることがある。そのためスレッドを使ってメインとなるループと処理を分けたり、ノンブロッキングを指定したりする必要がある。

UDPによる通信

UDPは、簡単な手順で通信が行われるが、通信中にデータが失われたりする可能性がある。
LAN内のローカルな通信で多少のデータ落ちが発生しても問題にならないような場合やLAN内で複数のマシンに一斉送信したい場合などには適している。

仕組みと手順

UDPによる通信では、送り先のIPアドレスとポート番号を指定してソケットを作成し、データを送る。
受け取る側は、ポート番号を指定してソケットを作成し、データが来るのを待っている。
単純に送りつけ、来た物を受け取るだけである。途中でデータが失われてもわからない。

必要に応じてデータを管理する処理を自分で行う事で、データの信頼性を確保する必要がある。

準備：
XCodeの左側（グループとファイル）で右クリック「追加」から新規グループを選択し、"addons"と名前を付ける。
addons を右クリックして追加/既存のファイルを選択し、openFrameworksのフォルダのaddons/ofxNetworkおよび ofxThreadを追加する。

送信側プログラム：
マウスがクリックされた座標を文字列にして送信。
マウス座標が100,200ならば、"100|200\"　が送られる。

```
#include "testApp.h"
#include "ofxNetwork.h"

ofxUDPManager udpConnect;    // UDP接続用オブジェクト
float mx, my                 // マウス座標の保存用

//--------------------------------------------------------------
void testApp::setup(){
   ofSetWindowShape(400, 400);
   ofBackground(0, 0, 0);
   ofSetVerticalSync(true);
   ofSetFrameRate(30);

   udpConnect.Create();                          // UDP用ソケットの作成
   udpConnect.Connect("127.0.0.1", 22518);   // 接続先をアドレスと
                                                 // ポート指定
}

void testApp::draw(){
   ofRect(mx, my, 50, 50);          // マウス座標に四角を描画
}

void testApp::mousePressed(int x, int y, int button){
   string message = "";             // 送信用メッセージの初期化
   mx = mouseX;                     // マウス座標を記録
   my = mouseY;
```

```
   message = ofToString(mx, 0) + "|" + ofToString(my, 0) + "/";
   // 送信用文字列の作成

   int sent = udpConnect.Send(message.c_str(), message.length());
   // 文字列をc用の文字列に変換し、サイズを指定して送信。
}
```

受信側プログラム：

マウスの位置が入ったメッセージを受信し、文字列から座標を解析し、
その座標に四角を描画する。

```
----
#include "testApp.h"
#include "ofxNetwork.h"

ofxUDPManager udpConnect;
float mx,my;

void testApp::setup(){
   ofSetWindowShape(400, 400);
   ofBackground(0, 0, 0);
   ofSetVerticalSync(true);
   ofSetFrameRate(30);

   udpConnect.Create();              // UDPソケットの作成
   udpConnect.Bind(22518);           // ポート番号にバインドする
   udpConnect.SetNonBlocking(true);  // ノンブロッキングモードにする
                                     // しなくても動きます。
                                     // ただし、メッセージを受信するまで
                                     // ウインドウが表示されません。
}

//--------------------------------------------------------------
void testApp::update(){

   char recvmessage[100];            // メッセージを入れる文字配列

   udpConnect.Receive(recvmessage,100);// メッセージを受信し配列に入れる
   string str = recvmessage;           // 文字配列から文字列へ変換
   vector<string> strPnt = ofSplitString(str, "/");
                                       // 文字列を / 文字で分割する
```

```
    // 文字 / は、x座標とy座標のペア毎の区切り文字
    // 分割した文字列は可変長配列(vector型) として返る。
    // この場合、strPnt[0], strPnt[1] に２つに分割された文字列が各々入っている。

    for(int i = 0; i < strPnt.size(); i++){
        vector<string> point = ofSplitString(strPnt[i], "|");
        mx = atof(point[0].c_str());
        my = atof(point[1].c_str());
    }
    // 同様にx座標とy座標の区切り文字である | によって文字列を分割する
    // 分割した文字列を数値（float型)に変換する。
}

//--------------------------------------------------------------
void testApp::draw(){
    ofRect(mx, my, 50, 50);       // 受取った座標に描画する
}
```

練習問題１：
受信側のプログラムを改良して、マウスをクリックした場所を線で結んでいくような
プログラムにする。
vector<ofPoint> pnt;
などの可変長配列を利用して　pnt.push_back()を利用してクリックした座標を
保存していくようにする。
ofxNetworkのサンプル（networkUdpReceiveExample)を見ると非常に参考になります。

練習問題２：
送信側のプログラムのConnect部分を変更するとブロードキャストができます。

```
    udpConnect.SetEnableBroadcast(true);
    udpConnect.Connect("255.255.255.255",22518);
```

に変更して、プログラムを作り直し、
受信側のプログラムを他のマシンにコピーして、複数のマシンで起動しておくと
送信側でクリックすると、受信側全部に反映されます。

TCP/IPによる通信

TCP/IPによる通信では、お互いに接続しコネクションを確保して通信します。
つまり、UDPの用に送信側と受信側という区別ではなく、接続を要求する側と接続を

受け入れる側という区別になります。この関係がクライアントとサーバという役割になります。
サーバでは、接続要求が来るのを待っています。
クライアントが接続要求を出すと、サーバ側で受付け接続されます。
繋がった後は、お互いにデータのやり取りを行います。
データは送った順番に送ったデータがきちんと届くようになっています。

仕組みと手順
クライアント側
1　ポート番号とサーバのアドレスを指定してクライアント側ソケットを作成し接続する
2　接続されていたら、データの送受信を行う。
3　接続に失敗したら、再接続を試みる（ことを繰り返す）
4　必要なくなったら閉じる

サーバ側
1　ポート番号を指定してサーバ側ソケットをし接続を待つ
2　接続要求があると勝手に接続してくれる。各クライアントの情報は記録されている
3　各クライアントとデータの送受信を行う
4　必要なくなったらクライアントは切断する

準備：
UDPと同様。

クライアント側(testApp.cpp)

```
#include "testApp.h"
#include "ofxNetwork.h"

ofxTCPClient TCPClient;    // TCPクライアントオブジェクト
bool isConnected;
string message;

int connectTime;
int deltaTime;

void testApp::setup(){
   ofSetWindowShape(400, 400);
   ofBackground(0, 0, 0);
   ofSetVerticalSync(true);
   ofSetFrameRate(30);
```

```
   // TCPのソケットを作成し、サーバに接続する。
   // 接続に成功すると、trueを返す、失敗するとfalseを返す
   isConnected = TCPClient.setup("127.0.0.1", 22519);

   // サーバへ再接続する際のタイマー
   connectTime = 0;
   deltaTime = 0;

   // 詳細なメッセージを出力する（デバッガのコンソールで確認できます）
   TCPClient.setVerbose(true);
}

void testApp::update(){

   if(isConnected){
      // ソケットからデータを読み込む
      string str = TCPClient.receive();
      if(str.length() > 0){
         message = str;
      }
   } else {
      // まだサーバに接続してなかったら、再接続を試みる
      // この例では10秒(10000msec)ごとに試みる
      deltaTime = ofGetElapsedTimeMillis() - connectTime;
      if(deltaTime > 10000){
         isConnected = TCPClient.setup("127.0.0.1", 22519);
         connectTime = ofGetElapsedTimeMillis();
      }
   }

}

void testApp::draw(){

   int mx, my;

   // サーバから受け取った文字列から、座標を取り出し描画する
   vector<string> strPnt = ofSplitString(message, "/");
   for(int i = 0; i < strPnt.size(); i++){
      vector<string> point = ofSplitString(strPnt[i], "|");
      mx = atof(point[0].c_str());
      my = atof(point[1].c_str());
   }
   ofRect(mx, my, 50, 50);
```

```
}
```

サーバ側(testApp.cpp)

```
#include "ofxNetwork.h"

ofxTCPServer TCPServer;      // TCPサーバオブジェクト

void testApp::setup(){

   ofSetWindowShape(400, 400);
   ofBackground(0, 0, 0);
   ofSetVerticalSync(true);
   ofSetFrameRate(30);

   // TCPサーバソケットを用意し、接続を待ち受ける
   TCPServer.setup(22519);
}

void testApp::draw(){

   // 接続されているすべてのクライアントからデータを受け取り
   // その情報を描画する（
   // クライアント数の取得：getNumClients()
   // 各クライアントのポートの取得：getClientPort(クライアントID)
   // 各クライアントのIPアドレスの取得：getClientIP(クライアントID)
   for(int i=0; i < TCPServer.getNumClients(); i++){
      string port = ofToString(TCPServer.getClientPort(i));
      string ip = TCPServer.getClientIP(i);
      string info = "client " + ofToString(i) + ": from " + ip + " on
port: " + port;

      ofDrawBitmapString(info, 10, 50 * (i + 1));
}

void testApp::mousePressed(int x, int y, int button){

   string message;

   // マウスをクリックした座標を文字列にして、すべてのクライアントに送信
   message = ofToString(mouseX) + "|" + ofToString(mouseY) + "/";
   for(int i =0; i < TCPServer.getNumClients(); i++){
      TCPServer.send(i, message);
   }
```

}

練習問題３：
このサンプルでは、サーバ側でクリックした場所にクライアント側のウインドウの
その座標に四角が描画された。
クライアント側でクリックした場所にサーバ側のウインドウのその座標に四角を
描画するように変更する。その際にはクライアント毎に描画する色を変更する。

サーバークライアント型のプログラム

よくあるサーバ―クライアント型のシステムでは、サーバは常駐型のプログラムとして
常にクライアントからの要求を待ち続け、要求があると接続しサービスを提供する。
このようなタイプのプログラムを試してみる。
サーバでは、クライアントの状況を管理し、複数のクライアントが
連携して動いているように見せる。

授業内課題１：
あたかも2つのウインドウが繋がっているように、円が跳ね返りながら
行き来するようなプログラムを作る。
サーバでは状態の管理と状態の表示を行い、2つのクライアントのプログラムでは
各々ウインドウを作り、2つのウインドウが座標的に繋がっているかのように見せる。
クライアントの数を２つまでに制限し、右側と左側でほんのちょっとだけ違う2つの
プログラムを作る。(２つ以上を無視するか、２つ以上あったら切断する)
座標値の管理は、サーバで行い各クライアントに座標値を送る。
サーバのウインドウをクリックすると開始するようにすると良い。
なお、Zukeiクラスやそのサブクラスをうまく使って実現すること。
これまでやってきたように特定の方向に移動するには、x方向、y方向の移動量と
壁まで達したときの反射を考える必要がある。

参考：ほとんど同じ内容のプログラムを２つ作るときには、１つがある程度完成したら
複製して、違うプロジェクトにすると便利です。
・複製したいXCodeのプロジェクトのフォルダを複製する。
・複製したフォルダに新しいプロジェクト名を付ける。
・.xcodeをクリックしてプロジェクトを開く。
・右のペインの「ターゲット」にあるターゲットの名前をプロジェクト名と同じに変更する
・上にあるプルダウンメニューから、「ベースSDK｜Debug」をreleaseなどに一度変更し、
再度元に戻す。

課題：

授業内課題 1 を拡張し、ネット対戦型のPONゲームにしてみる。
各クライアントでは、Itaクラスを使って四角を描画しキーボード操作で左右に移動できるようにする。
円の座標はサーバで管理し、ラケットに当たったらクライアントからサーバにメッセージを送るなどの方法が考えられる。
また、ウインドウの外に出たら跳ね返すのに失敗したことになる。
今回は、得点計算までは必要ないので、ラケットを操作でき、当たったら跳ね返るようにする。