ネットワークプログラミングを利用したアプリケーション開発

日本大学 文理学部情報科学科
5419045 高林 秀

2022 年 2 月 4 日

概要

本稿は、今年度発展プログラミングの課題研究として Processing を用いたネットワークプログラミングを使用したアプリケーション開発を行うものである。本稿前半部では、開発の際に利用した技術やコードに関して説明を行う。本稿後半部では、実際に説明した技術を用いて、Processing 上で実行可能なアプリケーションを作成する。

目次

1　目的

本稿の目的は、今年度発展プログラミングの課題研究として Processing を用いたネットワークプログラミングを使用したアプリケーション開発を行うものである。前半部にて基本的な技術用語の説明を通して、プログラミングの際に必要な知識の復習を行う。後半部では、実際に Processing 上で実行可能なアプリケーション開発を通してネットワークプログラミングを自身のプログラムに実装する。

2　Processing の概要

まず、本稿の開発言語である Processing に関して軽く触れる。

Processing は、オープンソースで開発されているプログラミング言語である。主に電子アートと、グラフィック・ビジュアルデザインに特化した機能を持ち合わせており、他のプログラミング言語よりも視覚的な表現とフィードバックを簡単に得れるという特徴を持つ。

この言語は、米国出身の Casey Reas 氏と Benjamin Fry 氏によって開発・設計された。Reas 氏は、マサチューセッツ工科大学の MIT メディアラボの美学および計算グループの一部として、メディアアートと科学の理学修士号を取得している人物であり、Fry 氏は、データの視覚化に関する専門知識を持つアメリカ人デザイナーで、カーネギーメロン大学でコンピューターサイエンスの副専攻であるコミュニケーションデザインの BFA を取得し、修士号と博士号を取得している人物である。

Processing の構文は、Java に似ており、Java をより単純化した様な構文で構成される。開発の際には、skechbook と呼ばれる統合開発環境（IDE）を利用して行う。

図 1　Processing ロゴ

出典：`https://ja.wikipedia.org/wiki/Processing`

3　ネットワークプログラミング概要

この章では、開発に関連する基本的な技術用語の説明を簡単に行う。

3.1　プロトコル

コンピュータ同士がネットワークを介して通信を行うには、データの形式や送受信の手順を定めた「プロトコル」と呼ばれるルールのようなものが必要である。プロトコルにはいくつか種類があり同一のプロトコルに従っているコンピュータ同士のみネットワーク通信を行うことが可能だ。

単純なアプリケーション開発でも、ネットワーク通信を行うには様々な決め事（通信先の機器の指定、デー

タ型の指定等）を定める必要があり、異常な手間がかかる。したがって実際にネットワークプログラミングを行う際には「プロトコルスタック」と呼ばれる、プロトコルを階層化した概念が取り入れられる。この理論的モデルが「OSI 参照モデル」と呼ばれるもので、それを簡略化したモデルが「TCP/IP」と呼ばれるプロトコルスタックである。

<table>
<tr><th>階層</th><th>OSI参照モデル</th><th>TCP/IPの階層</th><th>主なプロトコル</th><th>接続機器</th></tr>
<tr><td>第7層</td><td>アプリケーション層</td><td rowspan="3">アプリケーション層</td><td rowspan="3">HTTP・POP3・SMTP</td><td rowspan="3">ゲートウェイ</td></tr>
<tr><td>第6層</td><td>プレゼンテーション層</td></tr>
<tr><td>第5層</td><td>セッション層</td></tr>
<tr><td>第4層</td><td>トランスポート層</td><td>トランスポート層</td><td>TCP・UDP</td></tr>
<tr><td>第3層</td><td>ネットワーク層</td><td>インターネット層</td><td>IP・ICMP</td><td>ルータ・L3スイッチ</td></tr>
<tr><td>第2層</td><td>データリンク層</td><td rowspan="2">ネットワーク
インタフェース層</td><td rowspan="2">Ethernet・PPP</td><td>ブリッジ・L2スイッチ</td></tr>
<tr><td>第1層</td><td>物理層</td><td>リピータ</td></tr>
</table>

図 2　OSI 参照モデルと TCP/IP、その他プロトコルの関係図

出典：https://ansl-blog.hatenablog.com/entry/2019/05/22/OSI%E5%8F%82%E7%85%A7%E3%83%A2%E3%83%87%E3%83%AB%E3%81%A8TCP%EF%BC%8FIP%E3%81%AB%E3%81%A4%E3%81%84%E3%81%A6

3.2　TCP/IP

TCP/IP は一般的にネットワークプロトコル群の総称を指し、接続先のコンピュータと通信を行うアプリケーションを指定するため、IP アドレスとポート番号を利用する。IP アドレスは、コンピュータの住所のようなもので、個別にコンピュータごとに割り当てられた文字列である。ポート番号は、通信を行うアプリケーションを指定する際に使用する。

前項の図の通り、OSI 参照モデルでは細かく 7 層に通信機能が分類されているが、プログラミングを行う際には細かすぎるため実用的ではなかった。したがって、よりシンプルかつ実用的な階層構造にしたものが TCP/IP である。現代ののネットワーク通信ではこの TCP/IP が用いられている。

3.3　Processing におけるネットワークプログラミング

Processing には、ネットワーク上のコンピュータ間でデータを読み書きするための機能が標準ライブラリとして準備されている。そのため、Processing の実行環境以外新たに外部からインストールするライブラリやモジュールは必要ない。

ネットワーク通信を行うためには当然だが、クライアントサイド（接続する側）とサーバーサイド（接続される側）双方にプログラムを用意する必要がある。そのための関連機能として、Processing には「processing.net」と呼ばれるライブラリが用意されている。

それぞれのサイドのプログラムを作成する際はこのライブラリをインポートする必要がある。

```
import processing.net.*;
```

各サイドで利用するメソッドやクラスの使用法は後述する章を参照いただきたい。

4　クライアントサイド

4.1　Processing の Client クラス

Processing にてクライアントサイドプログラムを実装するには、Client クラスを使用する。この Client クラスを用いることで、後述する Server クラスを持つプログラムと TCP 通信ができる。

Client クラスは、3 引数のコンストラクタを持っている。以下引数の概要を示す。

`Client(PApplet インスタンス，サーバーの IP アドレス or ホスト名，サーバーのポート番号)`

1. PApplet インスタンス：自身のインスタンスを示す「this」。
2. サーバーの IP アドレス or ホスト名：文字列 (String) 型。例：192.168.xx.x。
3. サーバーのポート番号：サーバー側のアプリケーションを識別するためのポート番号。整数 (int) 型。

この Client クラスは以下の様なメソッドを持つ。※説明文は公式リファレンスより出典。

5　サーバーサイド

5.1　Processing の Server クラス

前章の Client クラスと同様に、Processing でサーバーサイドプログラムを実装するには、Server クラスを利用する。

このクラスは、2 引数のコンストラクタを持っている。

`Client(PApplet インスタンス，待機するポート番号)`

1. PApplet インスタンス：自身のインスタンスを示す「this」。
2. 待機するポート番号：クライアント側の通信を待機するポート番号。整数 (int) 型。

6　Processing のイベントハンドラ

TCP 通信を用いるプログラミングでは、サーバー接続時や、切断時など様々な場面のときに行う処理である「イベントハンドラ」を定義することができる。特に Processing では、次のイベントを利用することができる。

- serverEvent：serverEvent() 内のコードは、プログラム内に作成されたサーバーに新しいクライアントが接続されたときに実行される。
- clientEvent：サーバーが既存のクライアントオブジェクトに値を送信する際に呼び出される。
- disconnectEvent：クライアントとの接続が切れた際に呼び出される。

各イベントハンドラは、上記名前の関数を定義しその中に処理を記述する。

```
void serverEvent(Server s, Clinet c) {
  //Do something....
}
```

7　アプリケーション実装

7.1　開発環境

今回の開発は仮想マシン上で行った。下記に当時の環境を示す。

- ホスト OS : Window10 Home 20H2
- 仮想 OS : Ubuntu 20.04.2 LTS
- CPU : Intel(R)Core(TM)i7-9700K @ 3.6GHz
- GPU : Nvidia Geforce RTX2070 OC @ 8GB
- ホスト RAM : 16GB
- 仮想 RAM : 4GB
- Processing version : 3.5.3

7.2　制作内容

7.3　挙動説明

7.4　工夫点

8　巻末資料

本稿で使用した画像、プログラムコード等はすべて以下のリンク先に掲載している。必要に応じてご覧頂きたい。

- GoogleDrive:
- GitHub:`https://github.com/tsyu12345/advancedPrg/tree/master/No15-report3`