

インターネットプログラミング
教室2406
水曜日7限19:50-21:20

第9回
2013/11/20

岩井将行

授業資料

- <http://www.cps.im.dendai.ac.jp>
- <http://www.cps.im.dendai.ac.jp/index.php?Classes%2F2013InternetProg>

講師紹介

- <http://cps.im.dendai.ac.jp/index.php?Members%2Fiwai>
- 岩井研究室
- <http://cps.im.dendai.ac.jp>
- 岩井研究室の研究分野
- <http://cps.im.dendai.ac.jp/index.php?Research%2FTopics>
- 連絡先 1号館11F 11107b
- iwaiあっと im.dendai
- 研究室内線2844

講師

- 慶應義塾大学 卒
- 元東京電機大学 戸辺義人先生
OSOITEProject参加
- 東京大学生産技術研究所 助教
- 2013.4月より未来科学部情報メディア学科
准教授

- 詳しくはFacebook 岩井将行
- Twitter @masaiwai

TA・SA・副手

- 加藤 佳祐
- 鉄谷研究室
- 東京電機大学 未来科学部 情報メディア学科
- E-mail: Keisuke Kato
- <case.unl[atmark--].-gmail.com>

ソケット通信

プログラム同士の通信は

- ソケットを使ってデータの送受信
- ソケットを使った通信

ソケット通信

ソケット

意味：「接続の端点」

コンピュータとTCP/IPを
つなぐ出入り口

ソケット



ソケット通信

- ソケットを使って通信を行うには
2つのプログラムが必要

クライアントプログラム

ソケットを用意して
サーバに接続要求を行う

サーバプログラム

ソケットを用意して接続要求を待つ

ソケット通信の全体の流れ

クライアント

ソケット生成(socket)

サーバを探す
(gethostbyname)

接続要求(connect)

データ送受信(send/rcv)

ソケットを閉じる(close)

サーバ

ソケット生成(socket)

接続の準備(bind)

接続待機(listen)

接続受信(accept)

データ送受信(send/rcv)

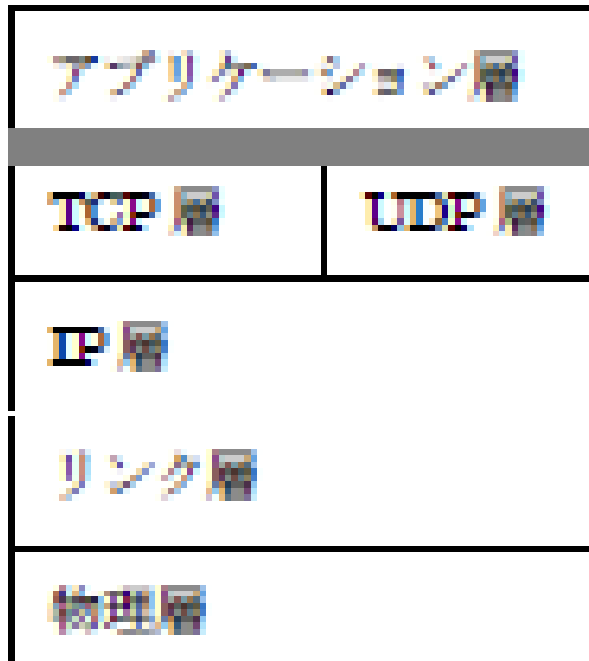
ソケットを閉じる(close)

識別情報

動かしてみよう。

- SwingAnimationBasic
- SwingAnimationFaceObj
- Swing より美しいグラフィックスを提供
java2D
- ちらつき防止
- http://www.java2s.com/Tutorial/Java/0240_Swing/Catalog0240_Swing.htm

Socket interface



ソケットインターフェイス

UDP通信

- UDP通信では接続という概念はない。
- ユーザは毎回データを送信し、また毎回自分のソケット、及び相手のソケットとアドレスを指定することになる。TCPでは最初に相手のソケットとの間の接続を行い、双方のソケット間の接続が確立されたら、互いの通信が可能になる。
- TCPではそのような制限はない。
- TCPでは接続が確立されたら2つのソケットはストリームのように振る舞う。TCPでは受信したデータの信頼性と順序が保障される。

DatagramPacket

- SocketやServerSocketでは、データのやりとりは入出カストリームに抽象化されていたため、パケットという概念が見えてこなかった。
- DatagramPacketとDatagramSocketの場合はUDPパケットはUDPパケットとして抽象化されており、パケットにデータも送信先アドレスも含める必要がある。
- SocketやServerSocketとは別物

UDP Server

```
int serverPort = 5000;
```

```
DatagramSocket socket = new  
    DatagramSocket(serverPort);
```

```
DatagramPacket receivePacket = new  
    DatagramPacket(new byte[DMAX], DMAX);
```

```
socket.receive(receivePacket);
```

```
socket.close();
```

Udp Client

```
String servhostname="localhost";

InetAddress serverAddress =
    InetAddress.getByName(servhostname);
String message="hello UDP from yourname";
byte[] bytesToSend = message.getBytes();
int serverPort = 5000;
DatagramSocket socket = new DatagramSocket();
DatagramPacket sendPacket = new
    DatagramPacket(bytesToSend, bytesToSend.length,
        serverAddress, serverPort);
socket.send(sendPacket);
socket.close();
```

2013/11/20

今週の課題[UDPの送受信をやってみよう]

- 提出フォルダは第9回
- DendaiUDPClient1 DendaiUDPServ1を改造して、「送信したメッセージを真逆にして返してくる」
- 送信時はDendaiUDPServer5000番ポート
返信時にはDendaiUDPClientの5001番ポートを指定すること

Innerクラスの整理 動かしてみよう

- ClassA

ソケット通信

プログラム同士の通信は

- ソケットを使ってデータの送受信
- ソケットを使った通信

ソケット通信

ソケット

意味：「接続の端点」

コンピュータとTCP/IPを
つなぐ出入り口

ソケット



ソケット通信

- ソケットを使って通信を行うには
2つのプログラムが必要

クライアントプログラム

ソケットを用意して
サーバに接続要求を行う

サーバプログラム

ソケットを用意して接続要求を待つ

ソケット通信の全体の流れ

クライアント

ソケット生成(socket)

サーバを探す
(gethostbyname)

接続要求(connect)

データ送受信(send/rcv)

ソケットを閉じる(close)

サーバ

ソケット生成(socket)

接続の準備(bind)

接続待機(listen)

接続受信(accept)

データ送受信(send/rcv)

ソケットを閉じる(close)

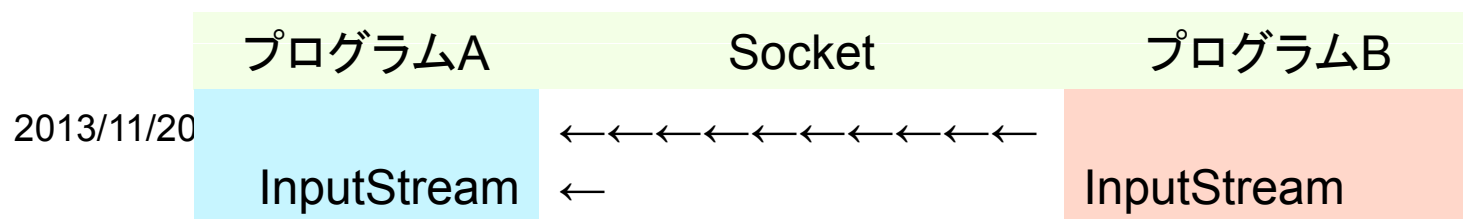
識別情報

Socket通信

- 双方がデータのやりとりを行う場合、双方を結んで情報を運ぶための「線」が必要となります。Javaにおいてはこの「線」のことを「**Socket** (ソケット)」という概念で表します。「Socket」は逆方向の情報の流れ (Stream) をもつ二本の通信線を一本に束ねたものだと考えられます。

InputStream, OutputStream

- 仮にAとBという二つのプログラムが通信を行うとすれば、一方の通信線がAにとっての「InputStream」でありかつBにとっての「OutputStream」、もう一方の通信線はAにとっての「OutputStream」かつBにとっての「InputStream」となる。
- この2本の通信線を束ねる「Socket」を介して情報のやりとりが行われる。

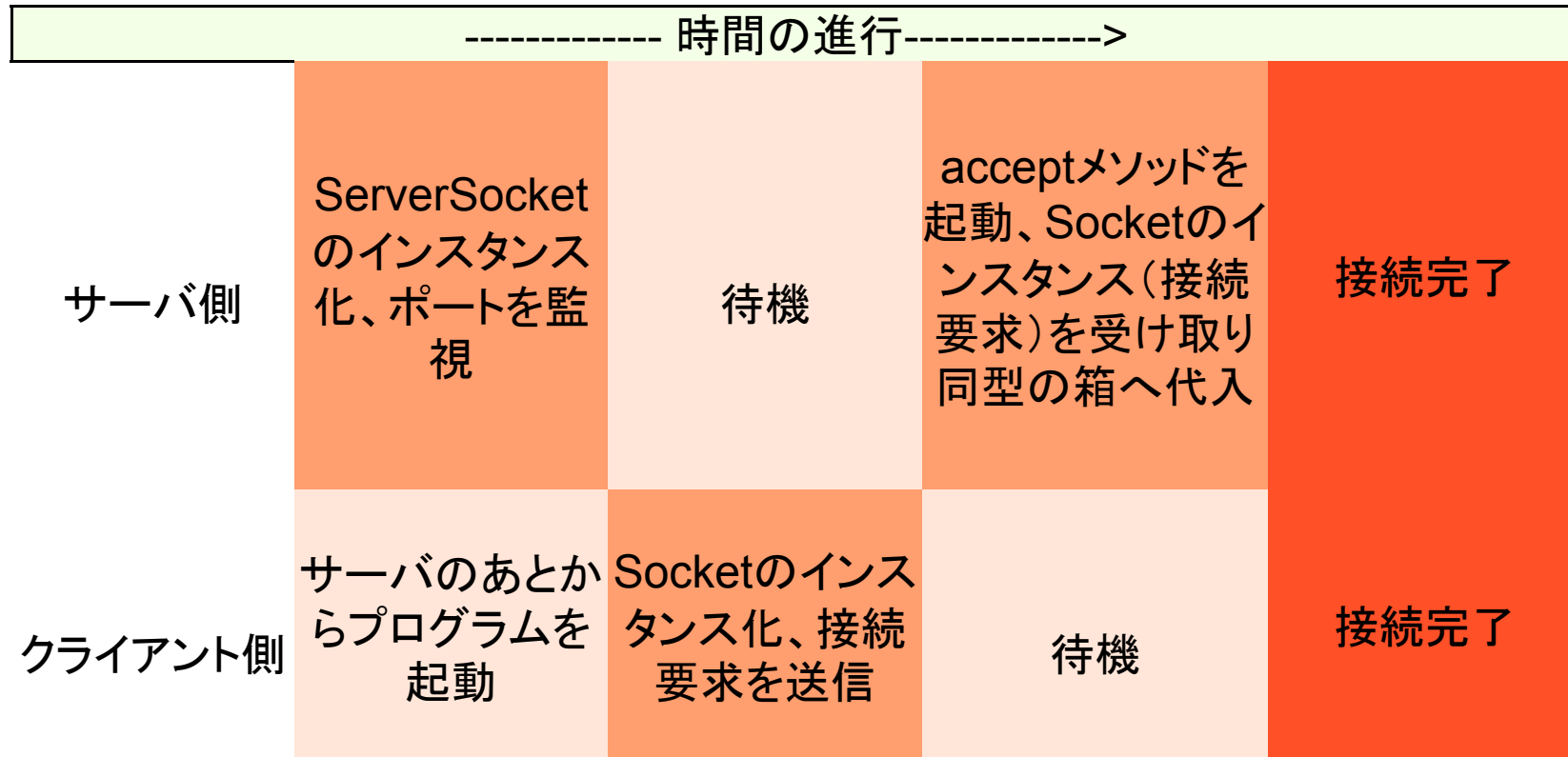


Socketとport番号

- クライアント側からサーバ側に向かって「Socket」が送られて、初めて通信が開通
- クライアントが「Socket」を送り込む際には、必ず「どのサーバコンピュータ」の「何番の通信口」に向かって送り込むのかを明らかにしなければなりません。
- その際の「どのサーバコンピュータ」のことを一般に「サーバ名」、「何番の通信口」のことを「ポート番号」、と呼びます。サーバには複数の「通信口=ポート」があり、番号で管理しています。サーバの側は、プログラムによって決まった番号のポートを監視しています。
- クライアントはサーバが監視するポート番号に「Socket」を送らなければなりません。

ServerSocketクラス、Socketクラス

- **ServerSocketクラス** サーバの側に用いられます。インスタンス化の際に引数として「ポート番号(整数型)」を要求します。インスタンス化が済んだ時点でポートの監視が始まります。そのとき、もしもクライアントから接続要求(「Socketクラス(後ほど説明)」のインスタンスが送られてくる)があれば、「acceptメソッド」で受け取ります。この時点で通信の準備が完了します。
- **Socketクラス** クライアントの側に用いられます。インスタンス化の際に引数として「サーバ名(文字列型)、ポート番号(整数型)」の順に二つの引数を要求します。インスタンス化が行われた時点で接続要求がサーバに送られます。



動かしてみよう1

- DendaiTCPServ1
- DendaiTCPClient1

Objectのやりとり

- OutputStreamへのデータの書き込み → データの送信
- InputStreamからのデータの読み込み → データの受信
- **OutputStream/InputStreamの取得** OutputStream/InputStreamの取得を行うためには、Socketクラスのメソッドである「**getOutputStream/getInputStream**」メソッドを利用します。これらのメソッドが起動されると、返値として、取得したOutputStream/InputStreamのインスタンスが返されます。
- このインスタンスを引数にして、Streamを用いてデータの通信を行うクラスをインスタンス化することで、OutputStream/InputStreamが取得されデータ送受信の準備が完了します。
- 通信の際にやりとりするデータの型が「任意のクラスのインスタンス」の場合、データ通信は「**ObjectOutputStream/ObjectInputStream**」クラスを用いて行います。従ってgetOutputStream/getInputStreamメソッドの返値であるOutputStream/InputStreamインスタンスを引数として、これらのクラスのインスタンス化を行うことで、OutputStream/InputStreamが取得されデータ送受信の準備が完了します。

ObjectOutputStream、 ObjectInputStream

- データ送信の準備 (但しObjectOutputStreamのインスタンス名をoos、Socketのインスタンス名をsocketとする)
- ObjectOutputStream oos =
new
ObjectOutputStream(socket.getOutputStream());
- データ受信の準備 (但しObjectInputStreamのインスタンス名をois、Socketのインスタンス名をsocketとする)
- ObjectInputStream ois = new
ObjectInputStream(socket.getInputStream());

OutputStreamへのデータの書き込み(送信)

- OutputStreamへのデータの書き込みは、「ObjectOutputStream」の「writeObject」メソッドを利用して行います
- データの書き込み(送信)
- `oos.writeObject(送信したいインスタンス名);`
- `oos.flush();`

InputStreamからのデータの読み込み(受信)

- InputStreamからのデータの読み込みも、ファイルからの読み込みと同様「ObjectInputStream」の「readObject」メソッドを利用します。
- データの読み込み(受信)(但し、ObjectInputStreamのインスタンス名をois、読み込むデータをVector型のインスタンスとする)
- `Vector vec = (Vector)ois.readObject();`

通信終了の合図

- 通信終了の合図(但し、ObjectOutputStreamのインスタンス名をoosとする)
- `oos.close();`
- `socket.close();`

動かしてみよう

- DendaiTCPServ2Age
- DendaiTCPClient2Age

Serializable

- Serializableを実装したら、
- **serialVersionUID**という定数を定義
- class Test implements Serializable {
**private static final long
serialVersionUID =
8531245739641223373L;** }
- Objectの送受信が可能になる。