ネットワークプログラミング
ネットワークプログラミング演習
21001教室

TCP Server Client，thread
第10回
2013/11/25

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%2F2013netpro

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

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@gmail.com>

2限副手紹介

- 鉄谷研究室M2
  - 多田　真之　tada先輩 2限
  - 新井　駿　arashun先輩　2限

成績

- 出席
- 毎回課題
- 中間試験
- 最終試験＋最終課題

- ★演習は演習最終発表会を加味

講義内容

第1回	Java理解度チェック
第2回	TCP/IPの復習
第3回	TCPサーバ
第4回	TCPクライアント/サーバ通信　チャットプログラム
第5回	UDP通信
第6回	**中間試験（持ち込み不可　紙提出）**
第7回	スレッド基礎 サーバのスレッド化 マルチスレッド
第8回	デザインパターン ファクトリメソッド　シングルトン
第9回	ノンブロッキングI/O　　最終課題に向けた目標設定
第10回	マルチスレッド　スレッドプール　Twitter4J
第11回	Twitter4J, Webクライアント
第12回	WEBサーバ,プロジェクト設計
第13回	プロジェクト実装
第14回	予備
第15回	**学力考査（持ち込み可　プログラミング提出）**

授業予定日日程

- http://www.soe.dendai.ac.jp/kyomu/portal/2013_schedule_t.pdf
- (2)9/16 敬老の日【授業実施日】
- (3)9/23 秋分の日【授業実施日】
- (4)9/30
- (5)10/7
- **(6)10/14 体育の日【授業実施日】 中間試験**
- (7)10/21
- (8)10/28
- ~~休み　11/4 文化の日~~
- (9)11/11
- (10)11/18
- (11)11/25 thread+tcpchat
- (12)12/2 最終課題説明会 twitter4j Gui＋chatprogramming
- (13)12/9 最終課題へ向けた自主、およびテスト勉強　nio
- (14)12/16　最終課題へ向けた自主、およびテスト勉強
- (15)12/23 -*学力考査
- ~~休み1/13 成人の日~~
- (15)1/20 学力考査結果解説
- **※授業予定日に休講が有る場合は連絡します。**

概要

- クライアント／サーバモデル、TCP/IPネットワークのアプリケーションプログラミングインタフェースの基本および、ネットワークアプリケーションを効率的に動作させるためのマルチスレッドプログラミングを講義する。この基本の後、チャット等の対話型アプリケーション、Twitter4J等のアプリケーション開発の実例を講義する。

ゴール

- 通信ネットワークを利用したアプリケーションソフトウェアを、TCP/IP を意識したレベルで作成できる力を養成することを目標とする。

参考書

- 購入の必要はありません。
  - TCP/IPソケットプログラミング JAVA編
  - –ISBN4-274-06520-0
  - –オーム社
  - •[改訂第２版]JAVA言語プログラミングレッスン上
  - –ISBN4-7973-3211-5
  - –SoftBankCreative
  - •[改訂第２版]JAVA言語プログラミングレッスン下
  - –ISBN4-7973-3212-3
  - –SoftBankCreative

持ち物

- Laptop pc
  –Eclipse環境が整っていること
- PDF アクロバットリーダ
- LANでネットワークに接続できること。
- Macでもよい

日本語版eclipse

- http://mergedoc.sourceforge.jp/

Inner Class

```
public class ClassA {

	String name="classA";
	static InnerClassB b;
	static InnerClassC c;

	public static void main(String arsg[]){

			ClassA a=new ClassA();
			System.out.println(a.name);

			System.out.println((new ClassA()).name);

			System.out.println(b.name);
			System.out.println(c.name);

	}

	ClassA(){
			System.out.println("constractor ClassA");

			b=new InnerClassB();


			c=new InnerClassC();


	}

	class InnerClassB{

			String name="Inner ClassB";

			InnerClassB(){
					System.out.println("constractor ClassB");
			}

	}//inner ClassB end


	class InnerClassC{

			String name="Inner ClassC";

			InnerClassC(){
					System.out.println("constractor ClassC");
			}


			class InnerInnerClassD{
					InnerInnerClassD(){


					}
			}

	}//inner ClassC end

}//classA end
```

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

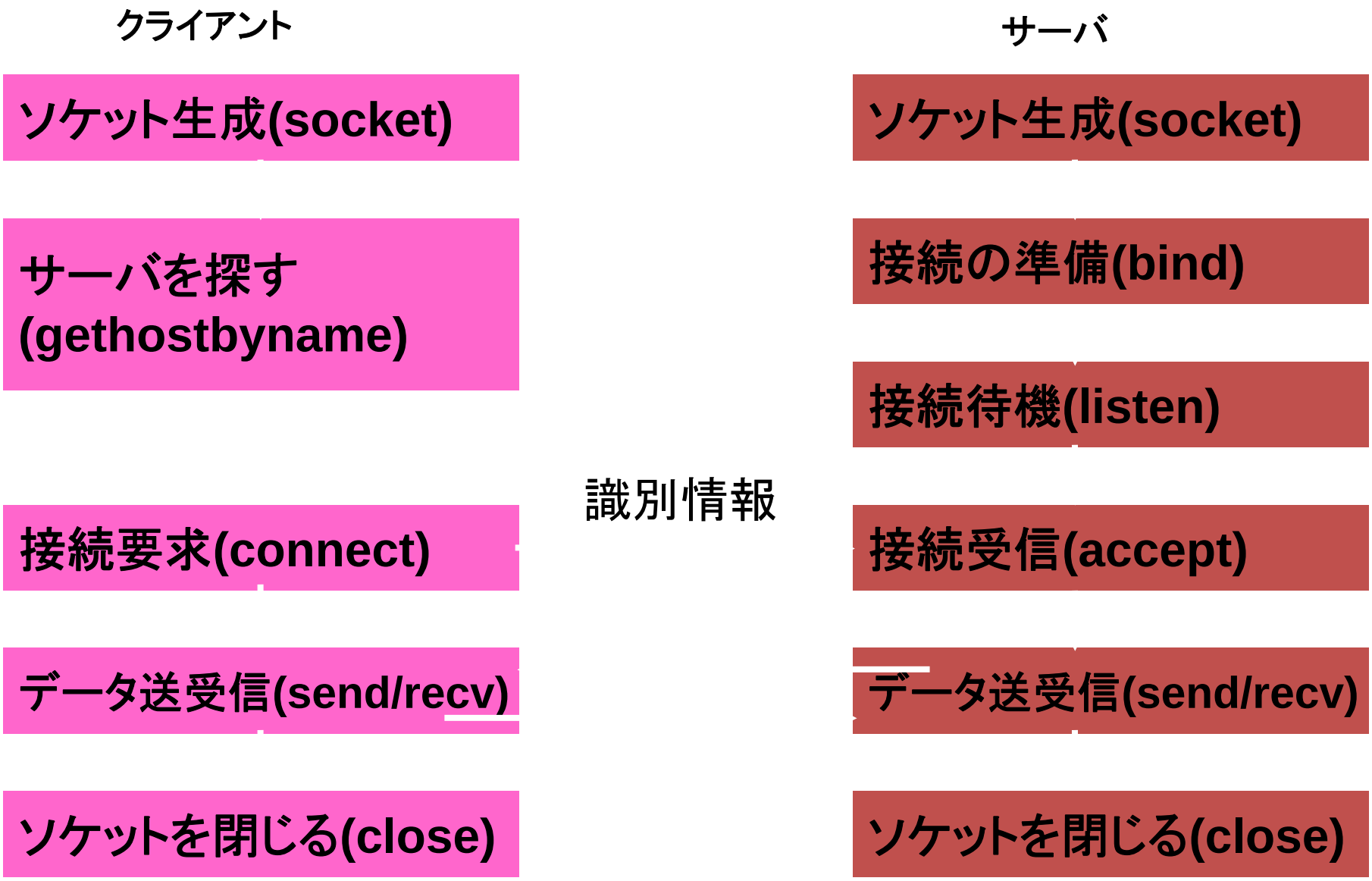
クライアントプログラム

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

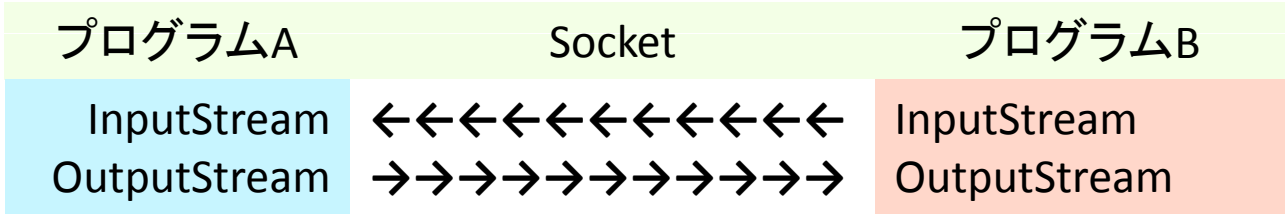


Sokect通信

- 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, OutptStream

- 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「OutptStream」、もう一方の通信線はAにとっての「OutputSteam」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。



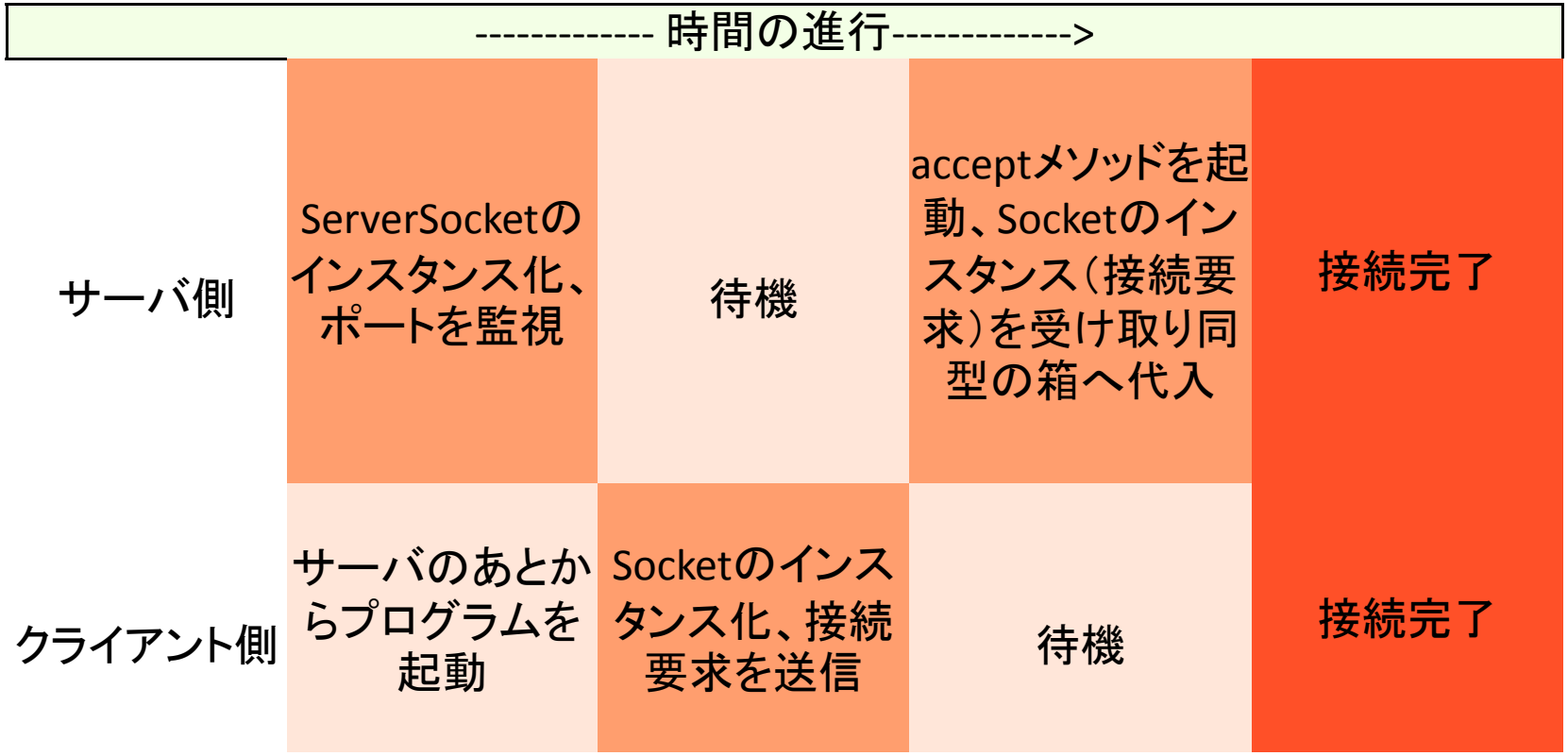
Socketとport番号

- ***クライアント側からサーバ側に向かって「Socket」が送り込まれて、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。


---------- 時間の進行---------->
サーバ側
ServerSocketのインスタンス化、ポートを監視
待機
acceptメソッドを起動、Socketのインスタンス（接続要求）を受け取り同型の箱へ代入
接続完了
クライアント側
サーバのあとからプログラムを起動
Socketのインスタンス化、接続要求を送信
待機
接続完了

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「***ObjectOutputSream/ObjectInputSream 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();

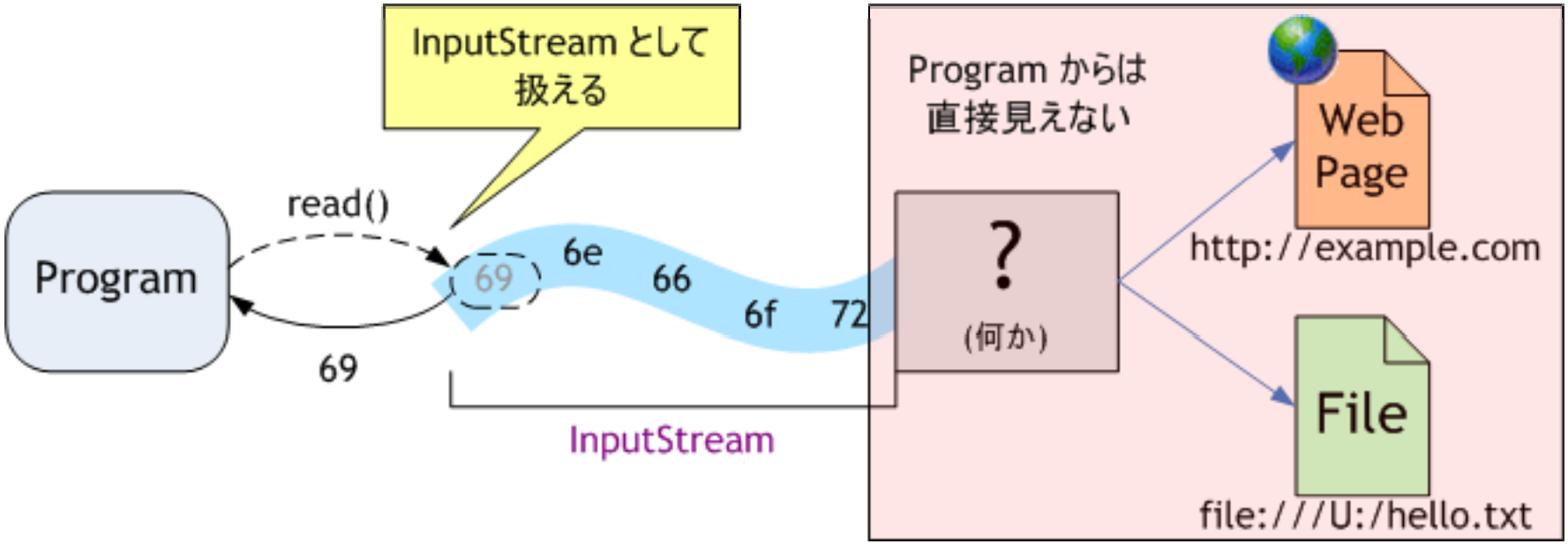
- socket.close();

先週動かしてみた

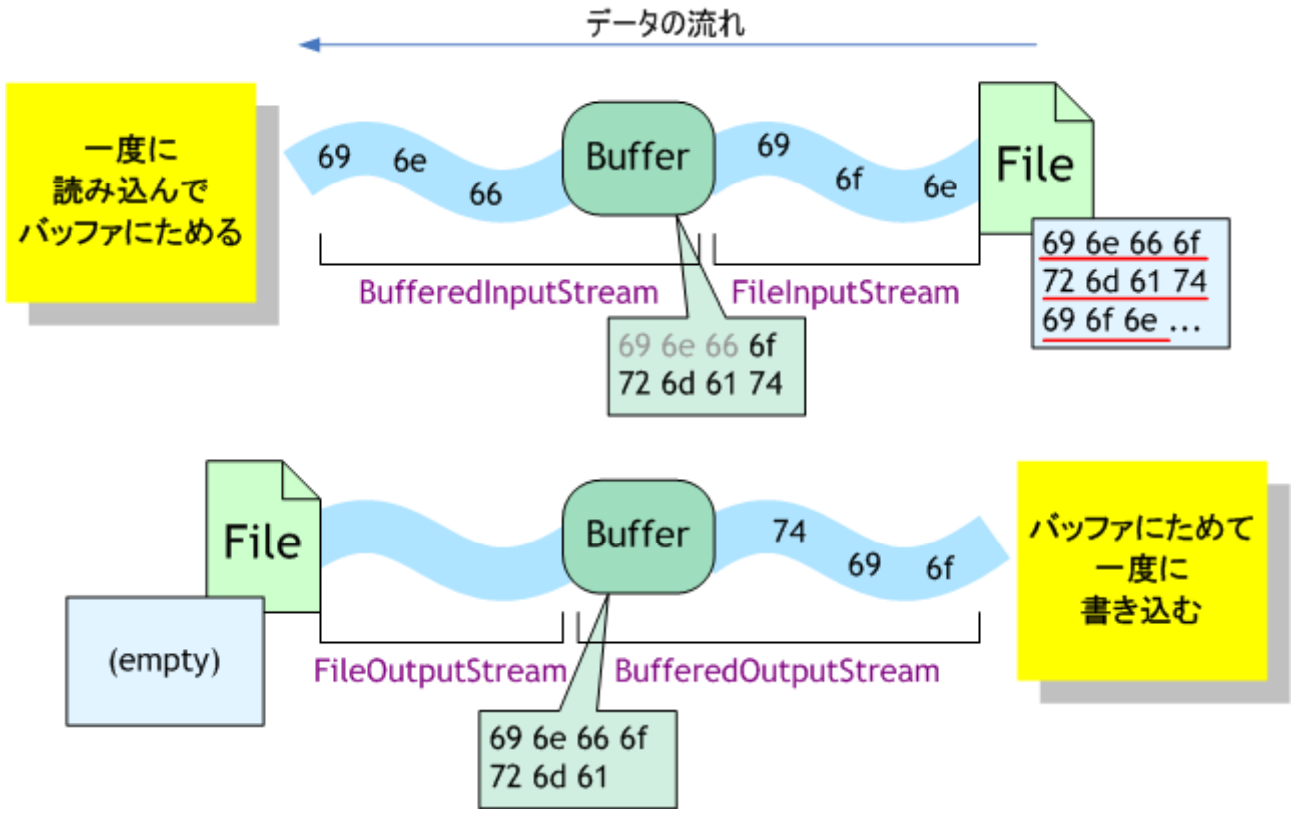
- DendaiTCPServ2Age
- DendaiTCPClient2Age

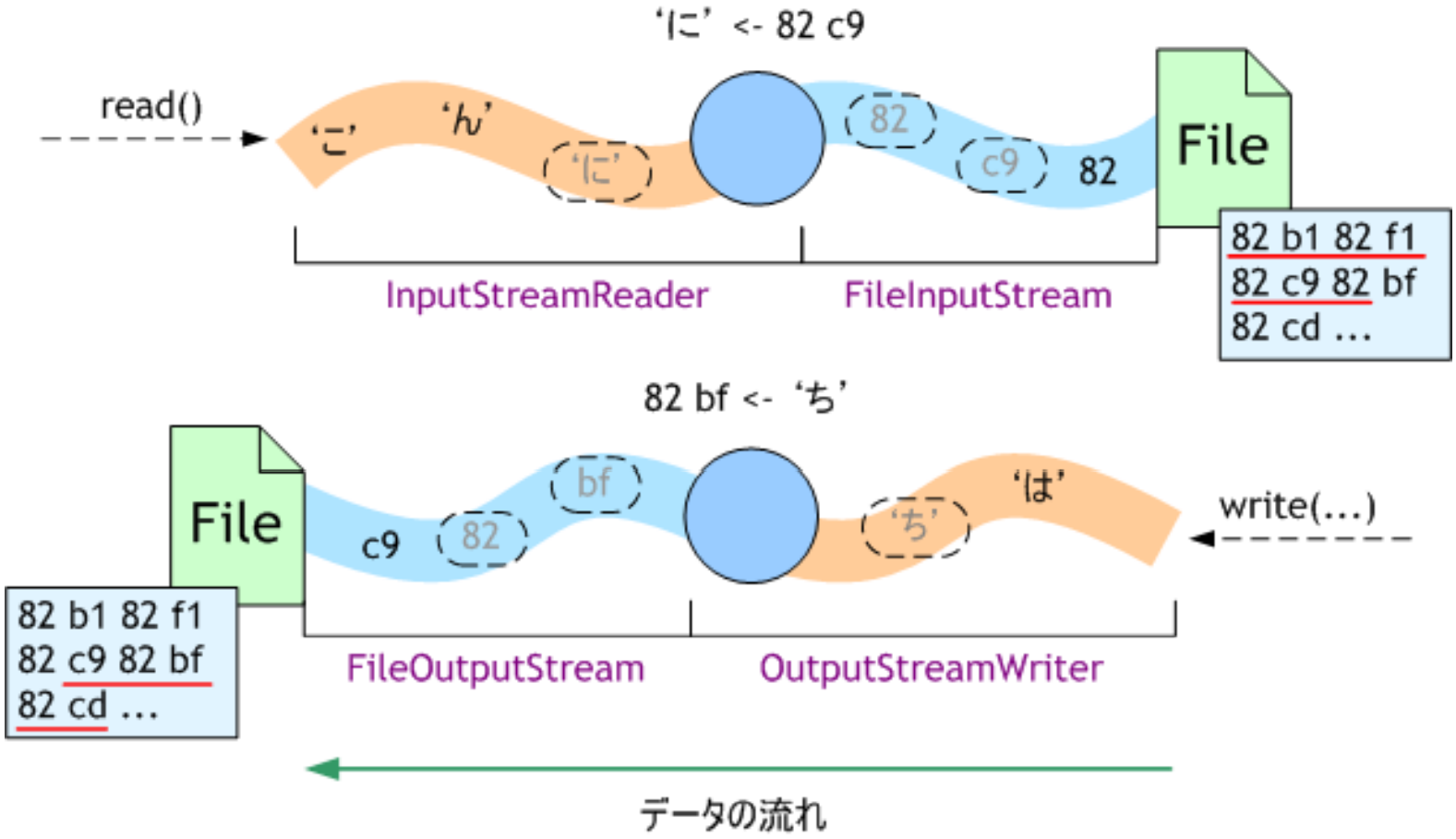
- 課題の答え合せ

IO Stream

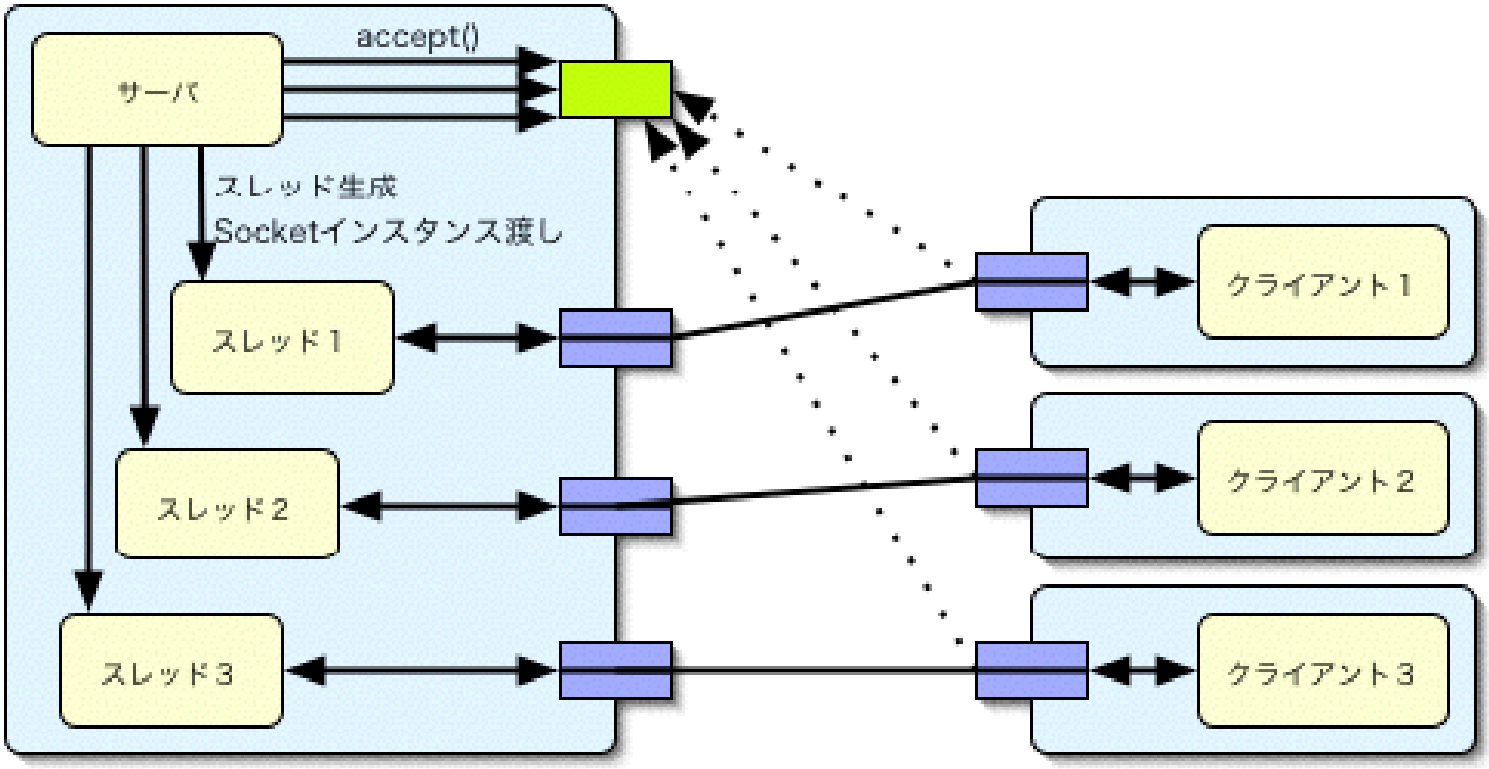

InputStream として
扱える
Program からは
直接見えない
Program
read()
69
6e
66
6f
72
69
InputStream
?
(何か)
Web
Page
http://example.com
File
file:///U:/hello.txt

BufferdI/OSream




‘に’ <- 82 c9
read()
‘こ’
‘ん’
‘に’
82
c9
82
File
82 b1 82 f1
82 c9 82 bf
82 cd …
InputStreamReader
FileInputStream
82 bf <- ‘ち’
File
c9
82
bf
‘ち’
‘は’
write(…)
82 b1 82 f1
82 c9 82 bf
82 cd …
FileOutputStream
OutputStreamWriter
データの流れ

MultiThread



動かしてみよう

- CountTenRunnable

try-catch

```
try {
    例外のスロー
} catch (Exception e ) {
    例外のキャッチ
}
```

例外を投げる

```
throw new Exception();

Public void method1 (int x) throws Exception {
    throw new Exception();
}
```

例外を投げる。

```
// NumberFormatException を捕捉するための try-catch
    try {
       n = Integer.parseInt(input);
    }
    catch (NumberFormatException e) {
       // 数値の形式が不正である場合は、入力自体が不正
       throw new IllegalInputException("不正な入力 " + input);
    }

    if (n < 0) {
       // 負の値が入力された場合は、不正な入力
       throw new IllegalInputException("不正な入力 " + input);
    }

    System.out.println("入力された正の値は" + n);
```

例外クラス

```
public class IllegalInputException extends
    Exception {
    public IllegalInputException(String message) {
        // 親クラスのコンストラクタにメッセージを
    渡す
        super(message);
    }
}
```

スタックトレース

- e.printStackTrace();　を使ってみる

うごかしてみよう。

- MultiServerSample.java
- MultiClientSample.java