情報システムとWeb技術'15 ＃1 補足資料

久野　靖 *

2015.11.17

1　Javaによるネットワークプログラミング

1.1　情報システムとネットワーク

今日の情報システムにおいて、ネットワークは不可欠な要素となっています。それには大きく分けて 2 つの理由があります。

1. 情報システムを「情報の発生するその場所で」使えるようにするために (これは事業所などの現場という意味と、顧客が直接自宅などから使うという意味とがあります)、ネットワークが不可欠だから。
2. 情報システムの「内部の構造として」複数の要素がネットワークでつながって全体として動作する、という形のものが一般的になってきたから。

前者についてはとくに説明するまでも無いと思いますが、後者についてはネットワークが広く普及してそのコストが低下し、またコンピュータシステムも 1 台ずつが安くなって、多数のシステムをつなげて動かす方が性能上も構築の易しさからも好ましくなったことによります。

たとえば、先にやった DFD を用いてシステムの構造を記述したとして、それを実際に動くシステムにするときに、これまでは各処理をモジュールとかオブジェクトに変換して組み合わせて動くようにしていたわけですが、そうするかわりに個々の○ (プロセス、処理) をシステム上でもプロセスとして自律実行させ、矢線に対応するネットワーク接続でデータを送受する形でシステムを組み合わせることも可能です (やや極論ですが)。

また、もう 1 つの面として、今日のネット上にはさまざまな有用なサービスが (おもにブラウザから呼び出すために) 提供されていますが、自分のシステムを作るさいにこれらのサービスを利用することで、少ない労力で豊富な機能を持たせることができます。そうなると、自分で作成する部分もいくつかの個々のサービスに分け、それら全てを組み合わせることでシステムとして動かすことが自然です。このような枠組みをサービス指向アーキテクチャ(SOA、service oriented architecture) と呼びます。

以下では、いくつかの例題をもとにこのようなシステムの作り方に少しだけ触れてみたいと思います。

1.2　Java による URL アクセス

とりあえず、今日ネット上で最も多くのリソースにアクセスできるのは Web ですから、Web サイトから情報を取得する簡単なプログラムから始めてみましょう。次のプログラムは、URL を指定するとその URL の中身を取り出して出力するものです。

*経営システム科学専攻

```
import java.net.*;
import java.util.*;

public class Sample11 {
  public static void main(String[] args) throws Exception {
    System.setProperty("http.proxyHost", "utogw"); // GSSM only
    System.setProperty("http.proxyPort", "8080");  // GSSM only
    System.setProperty("http.nonProxyHost", "w3in|10.*.*.*"); // GSSM only
    URL url1 = new URL(args[0]);
    Scanner sc = new Scanner(url1.openStream(), "JISAutoDetect");
    while(sc.hasNextLine()) {
      System.out.println(sc.nextLine());
    }
  }
}
```

main() の冒頭の 3 行は GSSM 内から外側のサイトを参照できるようにするためのプロキシ設定と、プロキシ使用から除外するサイトの指定です。たとえば自宅など直接インターネットにつながっている場所で使用する場合はこの 3 行をコメントアウトしてください。文字コードは上記のほかに「JIS」「EUCJIS」「Shift_JIS」が指定できます (が、やってみないとうまく行くかよく分からない…)。

その次からが本体で、まずコマンド引数として渡したも文字列をもとに URL オブジェクトを生成し、そのメソッド openStream() によって生成した読み込みストリームを 1 行単位で読めるように Scanner に持たせます。あとはストリームの終りまで 1 行ずつ読んで出力するだけです。

動かし方は次の通り (Java のプログラムはクラス名と同じ名前のファイル「Sample11.java」に入れる必要があることに注意)。

```
% javac Sample11.java            ←コンパイル
% java Sample11 http://w3in/     ← URL を指定して実行
<HTML>                    ← いきなり HTML が出力される
...
%
```

演習 このプログラムをそのまま打ち込んで、動かしなさい。実際にさまざまなサイトの URL を指定して、どんな感じのものが取れて来るか様子をつかみなさい。納得したら、次のような改造をしてみなさい。

a. 取り寄せた HTML 中の「href="...."」という部分だけを抜き出して表示するようにしてみなさい。(ヒント: String のメソッド indexOf()、substring() などを活用する。)

b. 取り寄せた HTML 中に埋められたリンク先を全部取り寄せるようにしてみなさい。(ヒント: 絶対 URL でない URL を解釈するためには、元の URL を指定できる形の URL のコンストラクタを使用するとよい。)

c. GSSM のサイト内のすべてのページの URL 一覧を作ってみなさい。

d. 興味ある外部 URL からつながっているページを一定数 (たとえば最大 100 ページぶん) 収集するようにしてみなさい。

e. 次の URL により、指定した ISBN の書籍に関する情報が取得できる (日本書籍出版協会のサービス)。

```
http://www.books.or.jp/ResultList.aspx?scode=
&searchtype=1&isbn=番号&showcount=1&startindex=0
```

これを利用して、ISBN(13 桁) を指定したらその書籍の名前を表示するプログラムを作ってみよ。

f. その他、適当な外部サイトのサービスを便利に活用するプログラムを作ってみなさい。

質問 このような、Web サービスを下請けとして使用するシステムの利点と弱点について、思い付くものをあげなさい。

注意: 外部サイトにプログラムアクセスするときは相手に迷惑を掛けないように注意すること。岡崎図書館事件にならないように。

1.3 Java によるネットワークサーバ

サービスを利用する側に続いて、サービスを提供する側についてもちょっとだけやってみましょう。最低限の機能だけでブラウザから情報を読める Web サーバのおもちゃを示します。

```
import java.net.*;
import java.io.*;
import java.util.*;

public class Sample12 {
  public static void main(String[] args) throws Exception {
    ServerSocket ss = new ServerSocket(4192);
    System.out.println("starting...");
    int count = 1;
    try {
      while(true) {
        Socket cs = ss.accept();
        PrintWriter out = new PrintWriter(cs.getOutputStream(), true);
        Scanner sc = new Scanner(cs.getInputStream());
        while(true) {
          String line = sc.nextLine();
          System.out.println(line);
          if(line.equals("")) { break; }
        }
        out.print("HTTP/1.1 200 OK\r\n");
        out.print("Content-type: text/html\r\n");
        out.print("connection: close\r\n");
        out.print("\r\n");
        out.printf("<body><i>Hello %d</i></body>\r\n", count++);
        cs.close();
      }
    } finally {
      ss.close();
    }
  }
}
```

先の例題よりは少し長いので分けて説明して行きます。

- TCP でアクセスされるサーバを作る場合は `ServerSocket` オブジェクトを作って使います。ここで引数はポート番号であり、1024 未満の番号は特権がないと作れませんが、大きい番号のは自由に作れます。なんとなく 4192 を使っています。

- 何回アクセスしても番号が同じだとつまらないので、1 回ごとにカウンタが増えるようにするので、そのカウンタが count。
- 何かエラーがあって終了するときも必ずサーバソケットが close() されるように try 文で囲んであります。
- while ループの内側が 1 つのアクセスにつき 1 回実行される部分。
- まずサーバソケットに接続されてくるのを accept() で待ちます。接続されると Socket オブジェクトが返されるので、そこから読むためのストリームとそこに書くためのストリームを生成。
- まず、空行が来るまで読みます。この部分は HTTP 要求ヘッダであり、ブラウザからサーバにいろいろ情報を渡しているのですが、ここではあっさり画面に表示するだけであとは無視しています。
- 次に、こちらから HTTP 応答ヘッダとして最低限の情報である状態行と Content-type:ヘッダを送信し、空行を送信してから本体部分 (ここでは 1 行の HTML) を返しています。

では、これを動かしている様子を見てみましょう (エラーが出ない限り終らないのでサーバを止めるには Control-C で停止させてください)。

```
% javac Sample12.java
% java Sample12
starting...
GET / HTTP/1.0
Host: smr06:4192
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.1.22)
Gecko/20090606 SeaMonkey/1.1.17
Accept: text/xml,application/xml,application/xhtml+xml,text/html;
q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: ja,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-2022-JP,utf-8;q=0.7,*;q=0.7
Via: 1.1 utogwgw.gssm.otsuka.tsukuba.ac.jp:8080 (squid/2.6.STABLE17)
X-Forwarded-For: 10.2.2.6
Cache-Control: max-age=0
Connection: keep-alive

...

^C  ← Control-C で停止
%
```

ちなみにブラウザ側からは図 1 のように「http://ホスト名:ポート番号/」という URL でアクセスします。再読み込みするごとにカウントが増えて行くのが分かるはずです。

演習 このプログラムをそのまま打ち込んで動かしなさい。動いたら次のような観察を行ってみなさい。

a. 指定する URL をもっと長くしたらどのような違いが観測されるか調べてみなさい。
b. Content-type:として text/html を通知していますが、これを text/plain にするとどうなるか試してみなさい。
c. 今は出力が 1 行だけですが、もっと行数を増やしてみなさい。また、1 行出力するごとに 1 秒、時間をあけるとどうなるかやってみなさい。(ヒント: 1 秒時間待ちをするのには「Thread.sleep(1)」を呼べばよいです。)

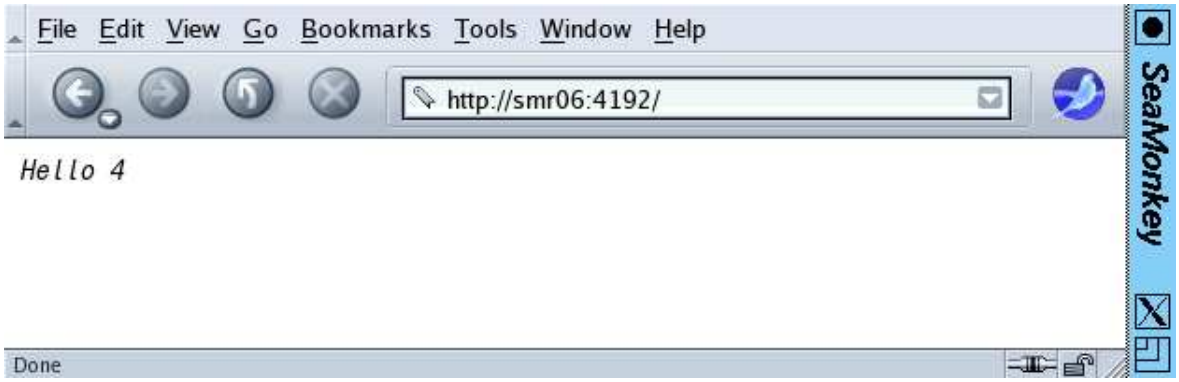

図 1: ブラウザから Sample12 に接続

d. 返送する HTML に IMG 要素を含めてみて、その結果どういうことが起きるか観察しなさい。

1.4 HTTP プロトコルと Web サーバ

別資料として HTTP 1.0(現在主に使われている 1.1 より 1 つ前のバージョン) の RFC(仕様書) を配布しました。1 つ前のにしたのは、あんまり長いと読む気が失せるからです。

ここまでで (なんとなく) 分かったように、HTTP ではブラウザはサーバに接続し、まずメソッド (GET が代表的) とプロトコルを指定し、続いて HTTP 要求ヘッダによって複数の付加情報を渡します。その後空行があり、もしブラウザからサーバに本体データを渡す場合はその後に本体データが続きます。

サーバ側はこれらの情報を受け取り、まず状態行 (200 OK が代表的を返します。続いて HTTP 応答ヘッダによって複数の付加情報を渡します。その後空行があり、それに続いてブラウザからサーバに渡す本体データが来ます。

これらのプロトコルによって必要な付加情報と本体データを渡し合う、というのが HTTP の全てです (もちろん細かい制御は色々ありますが)。そして、Web サーバは渡された URL に応じて「どこから」「どんな」データを取って来るとか、または必要に応じて内部でどのようなプログラムを動かしてデータを処理させるかなどを全部決めて実装するわけです。本ものの Web サーバはこの部分が非常に複雑で大規模になっていますが、土台となる HTTP の原理は非常に簡単であることがお分かり頂けるかと思います。

演習 先の例題サーバを修正して次のような実験をやってみよ。

a. 200 OK 以外のレスポンスにどのようなものがあるか調べ、実際に使ってみよ。(ヒント: リダイレクト系は Location: ヘッダにより転送先を指定すること。)
b. 画像データを生成して送り返す機能を追加してみなさい。渡されたパスに応じてさまざまな画像になるとなおよいです。
c. URL のパス部分をそのままパス名と解釈して、そのファイルを直に送り返すようなサーバを作る。(注意! どのファイルでも取れるというのは実際にはとても危険なので本番運用してはいけません。なぜ危険なのか分からない人は分からせておくこと。)
d. HTML にフォームを入れて、そのフォームのデータを受け取って処理できるような機能を加えてみなさい。
e. その他、ブラウザから使ってみて面白い機能を何か入れてみなさい。