第1章　Servletの作成

1.1　Webサーバサイドプログラムとは

Webサーバサイドプログラムとは、WWWのサーバ側で実行されて動的に（つまり要求のあるたびに異なる内容の）HTMLなどのコンテンツを生成するプログラムのことである。このなかでポピュラーなCGI（**C**ommon **G**ateway **I**nterface）は、Webサーバ上でプログラムを実行し、動的にHTML形式などのデータなどを作成してWebブラウザに渡すための仕組み（プログラムとWebサーバの間のデータのやりとりの約束事）である。またCGIに従って実行されるプログラム自体のこともCGIと呼ばれる。CGIを記述する言語は何でも構わないが、PerlやCを使うことが比較的多いようである。

Java AppletやJavaScriptはクライアント（Webブラウザが実行されているコンピュータ）側でプログラムが実行されるのに対し、CGIを含むサーバサイドプログラムはサーバ（HTMLファイルが置かれているコンピュータ）側でプログラムが実行されるという違いがある。例えばアクセスカウンタ・掲示板・オンラインショッピングサイトなどはクライアント上のプログラムだけでは実現できないのでサーバサイドプログラムが必要になる。

1.2　Servletとは

この演習ではサーバサイドプログラムの作成にJava Servletを用いる。Servletとは、CGIと同じようにWebサーバ側でプログラムを実行するための仕組み（約束事）である。しかしCGIとはいくつかの点で区別される[1]。

- まず、約束事はJavaのクラス/インターフェースとして提供されるので、プログラミング言語は当然Javaに限定される。
- 呼出し毎にいちいちプロセスを生成せず、スレッドとして実行するので効率が良い。
- ある程度の期間、サーバ側で接続の情報を記憶しておくことができるなど、サーバサイドプログラミングを支援するためのライブラリが充実している。

このためJavaによるサーバサイドのプログラミングとしては、CGIではなくServletを使用することが多い。例えばオンラインショッピングのためのWebサイトなどはServletが得意とする分野である。

本演習ではServletを実行するためのWebアプリケーションサーバとして、Apacheの Jakarta Tomcatを使用する。Webアプリケーションサーバは、コンテナとも呼ばれ、Webブラウザ（あるいはApacheなどのWebサーバ）からの要求を受け付け、Servlet（やJSP）を起動するプログラムである。

なお、TomcatやJavaの開発環境（JDK, Eclipse）などのインストール方法は別ドキュメントで解説する。

[1]ただし、Webサーバサイドプログラムの仕組みとしてCGIがもっとも代表的なので、 Servletを含めたWebサーバサイドプログラムの総称としてCGIという言葉を用いることもある。

有用なリンク

- 初めての ホームページ講座（http://www.hajimeteno.ne.jp/）— HTML のまとめ
- Java Tips（http://www.asahi-net.or.jp/~dp8t-asm/java/tips/）
- Apache Tomcat（http://tomcat.apache.org/）

1.3 本演習の位置づけ

本演習では次のような Java のごく初歩的な知識だけを仮定する。

- 制御構造（if〜else 文、for 文、while 文）の書き方がわかること。（C 言語の制御構文と同じ。）
- 継承（class〜extends）を利用してクラスを定義できること。
- メソッドの定義の書き方がわかること。（C 言語の関数定義とほとんど同じ。）
- クラス・オブジェクトの概念を理解していること。つまり、
  - （. 演算子を使って）フィールド参照・メソッド呼出しができること。
  - オブジェクトの生成（new）ができること。
  - クラス変数・クラスメソッドが使用できること。
- import 文が書けること。（C の#include に似ている。）

言い替えれば、if, else, for, while, class, extends, .（ドット）, new, import などのキーワード・演算子の使い方を理解していればよい。

あとは Java の API 仕様のドキュメント:

- http://java.sun.com/javase/ja/6/docs/ja/api/
  Java Platform, Standard Edition 6（Java の標準 API）— 以降、上記を単に ***(J2SEAPI)*** と記す。
- http://java.sun.com/javaee/5/docs/api/
  Java Platform, Enterprise Edition, v 5.0（Servlet 関連の API）

などで必要に応じてメソッドの使い方などを調べる必要がある。

DISCLAIMER: 本演習の主目的は、Servlet などの Web サーバサイドプログラムの作成方法を修得することではない。むしろ、Servlet を題材として Java の API の使用法に習熟することにある。

1.4 Servlet の作成

CGI も Servlet も、単純に言ってしまえば、HTML のデータ[2]を生成するプログラムである。

次に示すのは現在の時刻を表示する Servlet である。

[2]JPEG や PNG など HTML 以外のデータを出力する CGI や Servlet も考えられるが、はじめは簡単のために HTML を生成するプログラムのみ扱う。

ファイル MyDate.java

```
import java.io.*;
import java.util.Calendar;
import javax.servlet.*;
import javax.servlet.http.*;

public class MyDate extends HttpServlet {
  String[] youbi = {"日", "月", "火", "水", "木", "金", "土"};

  @Override
  public void doGet(HttpServletRequest req, HttpServletResponse res)
              throws IOException {
    res.setContentType("text/html; charset=Windows-31J");
    PrintWriter out = res.getWriter();
    out.println("<html><head></head><body>");

    Calendar cal = Calendar.getInstance();
    out.printf("%d 年%d 月%d 日%s 曜日%d 時%d 分%d 秒%n",
                cal.get(Calendar.YEAR), cal.get(Calendar.MONTH)+1,
                cal.get(Calendar.DAY_OF_MONTH),
                youbi[cal.get(Calendar.DAY_OF_WEEK)],
                cal.get(Calendar.HOUR_OF_DAY),
                cal.get(Calendar.MINUTE), cal.get(Calendar.SECOND));

    out.println("</body></html>");
    out.close();
  }
}
```

Servlet は HttpServlet というクラスを継承して作成する。このクラスに Servlet として必要なほとんどの機能が実装されているので、必要なところのみ書き換えれば Servlet が実行できるようになっている。

```
import javax.servlet.*;
import javax.servlet.http.*;
```

の import 文は大抵の Servlet で必要で、javax.servlet および javax.servlet.http というパッケージに属するクラスを利用するためにある。

Servlet の処理は基本的に doGet（または doPost — 後述）というメソッドの中に記述する。上のメソッド定義の最初の部分:

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws IOException
```

からわかるように、doGet/doPost は HttpServletRequest 型、HttpServletResponse 型の 2 つの引数を取る。それぞれ要求 (request) と応答 (response) を表すデータである。

最後の throws IOException の部分は、この doGet というメソッドが、IOException という例外（後述）を発生するかも知れないということを宣言する Java の構文である。

このメソッドの最初の文の

```
response.setContentType("text/html; charset=Windows-31J");
```

は、以下に続くデータが HTML のデータで文字コードが Windows-31J であるということをブラウザに伝える役割を持つ。

また、

```
PrintWriter out = response.getWriter();
```

は、ブラウザにデータを送るための**出力ストリーム**を取得する。これ以降 out オブジェクトの printf（あるいは、println, print）メソッドを呼び出すことにより、データを出力することができる。

PrintWriter クラスの printf は C 言語の printf 関数に相当する書式制御の機能つきの出力メソッドである。%d や%s などの書式制御は C 言語の printf のときと同じ意味である。一方、%n はプラットフォーム固有の改行コード（つまり、Unix では ¥n、Windows では ¥r¥n）を挿入する Java の printf 特有の書き方である。

println あるいは print はやはり出力のためのメソッドだが、%d や%s のような書式制御の機能はない。println は最後に改行を出力し、一方 print は改行しない。

なお出力の最後に out.close() を呼び出してストリームを閉じておく。

問 1.4.1 上の *Servlet* プログラムで現在の秒によって、ブラウザに表示されるときの色が変わるようにせよ。例えば、*0〜19* 秒が黒、*20〜39* 秒が青、*40〜59* 秒が赤など。

ヒント:

- *Calendar* クラスの使い方については ***(J2SEAPI)****/java/util/Calendar.html* を参照すること。
- 色を変えるには *HTML* のタグ * ... *などを用いる。（*HTML* の規格では、上の *red* の周りの引用符は上のように一重引用符「'」でも二重引用符「"」でも良い。*Java*（*C* でも同じ）のプログラムで、二重引用符「"」自体を出力したいときは、*out.println("<font color=¥"red¥">")* のように、「"」の前に バックスラッシュ「¥」をつける必要がある。

問 1.4.2 現在の秒によって、ブラウザに表示されるページの背景画像が変わるようにせよ。

ヒント:

- 背景画像を変えるには *HTML* のタグ *<body background=' ... '> ... </body>*などを使用する。
- 素材: *http://www.3776m.com/sozai/*（素材の館）
  http://www.ushikai.com/（牛飼いとアイコンの部屋）

1.5 （参考）Servlet の設置

本演習では Servlet のコンパイルと設置には Eclipse の WTP プラグインを使用する。このプラグインの使用法は別のドキュメントで説明する。以下では、Eclipse を使用しないで手作業でコンパイル・設置する方法を、概略だけ述べる。

Servlet を実行するには、まずコンパイルが必要である。次のコマンドで .class ファイルを作成する。

```
javac -classpath servlet-api.jar MyDate.java
```

“servlet-api.jar” の部分は、実際には ServletAPI が含まれている JAR ファイル（Java のライブラリファイル）へのパスに置き換える。これは通常、***(TOMCAT)***/common/lib/servlet-api.jar[3]となる。

[3] ***(TOMCAT)*** は Tomcat をインストールしたディレクトリのことを指す。 これは Tomcat のバージョンにより異なる。また、パスの区切りは Windows では通常 “¥” だが、このプリントでは一般的な “/” を使用する。

Servlet を実際に設置するには、生成されたクラスファイル（ソースファイルの名前が MyDate.java の場合、MyDate.class）を、Servlet の仕様で定められたディレクトリ構成:

```
-（ Web アプリケーションルート）
    - WEB-INF（ディレクトリ）
        - web.xml（設定ファイル）
        - classes（ディレクトリ）
            -（class ファイル）
        - lib（ディレクトリ）
            -（JAR ファイル）
```

の classes というディレクトリの下に置き、さらに、web.xml という設定ファイルにパスを記述する必要がある。

web.xml の設定例:

```
<web-app>
  ...
  <servlet>
    <servlet-name>MyDate</servlet-name>
    <servlet-class>MyDate</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>MyDate</servlet-name>
    <url-pattern>/MyDate</url-pattern>
  </servlet-mapping>
  ...
</web-app>
```

この例は、MyDate クラスに MyDate というサーブレットの名前をつけ、さらに MyDate という名前のサーブレットを/MyDate という URL のパスでアクセスできるようにするための設定である。これは冗長に見えるが、同じクラスを別の名前のサーブレットとして起動することができるようになっているためである。

ただし、開発中にはいちいち web.xml ファイルを書き換えるのは面倒なので、クラスファイルを classes フォルダに置くだけで Servlet が実行できるように、invoker サーブレットというものを有効にすることもできる。これは Tomcat 全体の設定ファイルの web.xml に設定することで有効化できる。*(TOMCAT)*/conf/web.xml のなかで、次のようになっている部分を探して、下線の部分を付け加える。

```
<!-- -->
    <servlet>
        <servlet-name>invoker</servlet-name>
        <servlet-class>
          org.apache.catalina.servlets.InvokerServlet
        </servlet-class>
        <init-param>
            <param-name>debug</param-name>
            <param-value>0</param-value>
        </init-param>
        <load-on-startup>2</load-on-startup>
    </servlet>
<!-- -->
```

```
<!-- -->
    <servlet-mapping>
        <servlet-name>invoker</servlet-name>
        <url-pattern>/servlet/*</url-pattern>
    </servlet-mapping>
<!-- -->
```

こうしておくと、例えば/servlet/MyDate というパスで、MyDate クラスにアクセスすることができる。

> **重要:** invoker サーブレットはセキュリティ上問題となることがあるので、開発時だけ使用するようにする。

Web アプリケーションルートは任意の場所に置くことができるが、Tomcat の設定ファイルの server.xml（*(TOMCAT)*/conf/server.xml）にその場所を記述する必要がある。

```
...
<Server ... >
  ...
   <Service ... >
  ...
     <Engine ... >
       ...
       <Host ... >
         ...
          <Context path="/SoftEngEnshu" reloadable="true"
                   docBase="C:¥somewhere¥SoftEngEnshu" />
         ...
       </Host>
     </Engine>
   </Service>
</Server>
```

この例では、C:¥somewhere¥SoftEngEnshu というフォルダをルートフォルダとする Web アプリケーションが SoftEngEnshu というパスでアクセスできることになる。さきほどの web.xml の設定例とあわせると、http://*hostname*:8080/SoftEngEnshu/MyDate（invoker サーブレットを使っている場合は、http://*hostname*:8080/SoftEngEnshu/servlet/MyDate）という URL で MyDate サーブレットの実行結果を見ることができる。*hostname* の部分は Tomcat を実行しているホストの名前または IP アドレスである。

> Servlet の開発中はサーブレットと WWW ブラウザは同一のコンピュータで実行していることが多いので、その場合は *hostname* は localhost（あるいは 127.0.0.1）となる。

1.6 ファイル・ディレクトリ操作

Servlet のようなサーバーサイドプログラムは、アクセスカウンタにせよ、掲示板にせよファイルやデータベースにアクセスする必要がある場合が多い。(でなければ、クライアントサイドのプログラムで実現できることがことが多い。)

以下では Java のファイルやディレクトリ操作の API を使用し、サーバーサイドでファイルアクセスを行なう Servlet を作成する。

1.7 アクセスカウンタ

アクセスカウンタはもっとも代表的なサーバサイドプログラムで、Web ページに対するアクセスの回数を記録し、表示するものである。以下に紹介する Servlet では、アクセスの回数はサーバ上のファイルに記録しておく。

ファイル Counter.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Counter extends HttpServlet {
  @Override
  public void doGet(HttpServletRequest request, HttpServletResponse response)
              throws IOException {

    int i;
    response.setContentType("text/html; charset=Windows-31J");
    PrintWriter out = response.getWriter();
    out.println("<html><head></head><body>");

    File f = new File(getServletContext().getRealPath("/counter.txt"));
    try {
      BufferedReader fin = new BufferedReader(new FileReader(f));
      i =  Integer.parseInt(fin.readLine());
      fin.close();  // close を忘れない
    } catch (FileNotFoundException e) {
      i = 0;        // ファイルがなければ 0 に
    }

    PrintWriter fout = new PrintWriter(new FileWriter(f));
    fout.println(++i);
    fout.close();   // close を忘れない

    out.printf("あなたは %d 番目の来訪者です。%n", i);
    out.println("</body></html>");
    out.close();    // close を忘れない
  }
}
```

この例では counter.txt というファイルにアクセス回数を記録している。このファイルは、Web アプリケーションルートフォルダの下の WEB-INF というフォルダに置かれている。counter.txt の中身は 1 行のみの数字だけのファイルである。

プログラム中の

```
getServletContext().getRealPath( ... )
```

という式は、WEB アプリケーションルートからの相対パスを受け取り、ファイルシステム中の絶対パスを返す。getServletContext は HttpServlet クラスのメソッドであり、getRealPath は SevletContext クラス（正確にはインタフェース）のメソッドである。これらのメソッドの詳細は Java の API 仕様のドキュメント（例えば HttpServlet クラスの場合は、http://java.sun.com/javaee/5/docs/api/javax/servlet/http/HttpServlet.html ）を参照すること。

また、

```
File f = new File(path);
BufferedReader fin = new BufferedReader(new FileReader(f));
...
fin.close();
```

は、ファイルから入力するときの常套句である。FileReader クラス、BufferedReader クラスの一般的な使用法は API 仕様で確認しておくこと。上記の ... の部分では、上で用意された fin というオブジェクトに対して、標準入力（System.in）からの入力と同じようにファイルからの入力が可能になる。最後の close() を忘れるとファイルの内容が消えてしまったりするので、注意が必要である。

また、Integer.parseInt は Java で文字列（String）を整数（int）に変換するためのクラスメソッドである。（C 言語の atoi 関数に相当する。）

同様に、

```
PrintWriter fout = new PrintWriter(new FileWriter(f));
...
fout.close();
```

は、ファイルへ出力するときの常套句である。ここで用意された fout というオブジェクトに対して、標準出力（System.out）に対するのと同じメソッドである print や println が使用できる。

ここまでの部分で、ファイルから数字を読み込み、一つ増やした数字をファイルに書き込んでいる。

1.8 Java の例外処理

Counter.java では、ファイルからの入出力処理の周りを try〜catch〜という形で囲っている。これは Java の例外処理の構文である。

try〜catch 文のもっとも基本的な使い方は次のような形である。

```
try {
  文の並び 0
} catch (例外型 1 変数 1) {
  文の並び 1
}
   ...
  catch (例外型 n 変数 n) {
  文の並び n
}
```

文の並び $_0$ の中で、例外が起こった時には try { 〜 }の間の残りの文は無視され、例外が型 $_k$（ただし k=1･･･n）にマッチするならば、文の並び $_k$ が実行される。文の並び $_0$ で例外が起こらなかった場合、および例外が起こってもマッチする例外がなかった場合には、文の並び $_k$（k=1･･･n）は実行されない。

先ほどのプログラム例では、counter.txt というファイルが見つからなかった場合、FileNotFoundException という例外が起こり、これに対応する catch 節の中で、カウンタの値を 0 に設定している。

問 1.8.1 カウンタの値が特別な値（例えば *10* の倍数など）になったときは、メッセージを変えたり、色を変えたりするように改造せよ。（割算の余りを求める演算子は、*C* 言語と同様 %である。）

問 1.8.2 アプリケーションルートの *images* ディレクトリに、*1.png, 2.png* などの名前で数字画像ファイルを用意しておいて、このアクセスカウンタや時刻表示 *CGI* で *1, 2,* と表示する代わりに*<img src='../images/1.png'>, <img src='../images/2.png'>*などにしておくと、数字を画像で表示するアクセスカウンタができる。
数字を画像として表示するアクセスカウンタを作成せよ。

参考: 数字画像データ

- *Digit Mania*（*http://www.digitmania.holowww.com*）
- *Counter Art*（*http://www.counterart.com/*）

問 1.8.3 数字を画像で表示する時刻表示プログラムを作成せよ。

（参考）ファイルを利用しない簡易アクセスカウンタ Servlet のインスタンスは、ページのアクセス毎に生成されるのではなく、いったん生成されると、Tomcat の中で保持され 2 回目以降のアクセスでは、以前に生成された Servlet のインスタンスが再利用される。このため、フィールドにデータを保持しておけば、ファイルを使用しなくても次のようなプログラムで簡易アクセスカウンタを実現できる。
ファイル Counter0.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Counter0 extends HttpServlet {
  int i=0;          // フィールドとして宣言する

  @Override
  public void doGet(HttpServletRequest request, HttpServletResponse response)
              throws IOException {
    response.setContentType("text/html; charset=Windows-31J");
    PrintWriter out = response.getWriter();
    out.println("<html><head></head><body>");
    out.printf("あなたは %d 番目の来訪者です。%n", i++);
    out.println("</body></html>");
    out.close();    // close を忘れない
  }
}
```

ただし、ファイルに書き込まないので、Tomcat を再起動すると、カウンタが 0 に戻ってしまう。完全なアクセスカウンタにするためには、Tomcat の終了時にカウンタの値をファイルに保存し、起動時にファイルからカウンタの値を読み込むように改造する必要がある。

問 1.8.4 *HttpServlet* クラスのメソッドを調べて、*Tomcat* 起動・終了時の保存・読み込み操作を追加し、*Counter0* を完全なアクセスカウンタに改良せよ。

1.9 ディレクトリ操作

つぎの Servlet はあるディレクトリのインデックス（ファイルの一覧）を生成する。
ファイル DirIndex.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class DirIndex extends HttpServlet {
  @Override
  public void doGet(HttpServletRequest request, HttpServletResponse response)
              throws IOException {
    response.setContentType("text/html; charset=Windows-31J");
    PrintWriter out = response.getWriter();

    String path = getServletContext().getRealPath("/"); // 適切なパスに変えておく
    File dir = new File(path);
    String[] files = dir.list();  // dir にあるファイル名の配列を得る

    out.println("<html><head></head><body>");
    out.println("<pre>");

    int i;
    out.printf("%s のファイル一覧%n%n", path);
    for (i=0; i<files.length; i++) {
      out.println(files[i]);      // files の各要素を順に出力
    }

    out.println("</pre>");
    out.println("</body></html>");
    out.close();
  }
}
```

ディレクトリの中のファイル名の一覧は、File クラスの list メソッドで String の配列として得ることができる。また、配列の要素の数は length というフィールドを調べることによってわかる。

問 1.9.1 *DirIndex.java* で、*3* 日前より変更された日付が新しいファイルには *"NEW!"* というマークをつけるようにせよ。例えば、ディレクトリに *old.txt* という *4* 日前に変更されたファイルと *new.txt* という *1* 日前に変更されたファイルがあるときは *DirIndex* は次のような *HTML* を出力する。

```
<html><head><title>ディレクトリ</title></head><body><ul>
<li>new.txt NEW!</li>
<li>old.txt</li>
</ul></body></html>
```

ヒント : *java.io.File* クラスの *lastModified* メソッドと *java.util.Calendar* クラスの *getTimeInMillis* を用いる。

キーワード :

HttpServlet クラス, doGet メソッド , throws, getServletContext メソッド , getRealPath メソッド , File クラス, FileReader クラス, BufferedReader クラス, FileWriter クラス, PrinterWriter クラス, 例外処理, try〜catch 文 , length（配列）