第1章　Servletの作成

1.1　Webサーバーサイドプログラムとは

Web サーバーサイドプログラムとは、WWW のサーバー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 サーバー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 ブラウザーに渡すための仕組み（プログラムと Web サーバー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, PHP 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 ブラウザーが実行されているコンピューター）側でプログラムが実行されるのに対し、CGI を含むサーバーサイドプログラムはサーバー（HTML ファイルが置かれているコンピューター）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ー・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ーサイドプログラムが必要になる。

1.2　Servletとは

本実験ではサーバー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 サーバー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。

- まず、約束事は Java のクラス・インタ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（または JVM ベースの言語）に限定される。
- 呼出しごと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、サーバー側で接続の情報を記憶しておくことができるなど、サーバー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 によるサーバー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。

Servlet を実行するためには、Web アプリケーションサーバーが必要である。Web アプリケーションサーバーは、コンテナーとも呼ばれ、Web ブラウザー（あるい

は Apache などの Web サーバー）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。Web アプリケーションサーバーとして有名なものに、Apache Tomcat や Jetty などがある。

なお、Apache Tomcat, Jetty 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。

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
- Apache Tomcat （`http://tomcat.apache.org/`）
- Jetty （`http://www.eclipse.org/jetty/`）

1.3　本実験の位置づけ

本実験では次のような 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。
  - クラスフィールド・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://docs.oracle.com/javase/jp/8/api/`
  Java(tm) Platform, Standard Edition 8（Java の標準 API）— 以降、上記を単に ***(J2SEAPI)*** と記す。
- `http://docs.oracle.com/javaee/7/api/`
  Java(TM) EE7 Specification APIs（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:　本実験の主目的は、Servlet などの Web サーバーサイドプログラムの作成方法を修得するというよりも、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[1]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。

ファイル MyDate.java

```
1  import java.io.IOException;
2  import java.io.PrintWriter;
3  import java.util.Calendar;
4
5  import javax.servlet.ServletException;
6  import javax.servlet.annotation.WebServlet;
7  import javax.servlet.http.HttpServlet;
8  import javax.servlet.http.HttpServletRequest;
9  import javax.servlet.http.HttpServletResponse;
10
11 @WebServlet("/MyDate")
12 public class MyDate extends HttpServlet {
13   String[] youbi = {"日", "月", "火", "水", "木", "金", "土"};
14
15   @Override
16   protected void doGet(HttpServletRequest request,
17                        HttpServletResponse response)
18                 throws ServletException, IOException {
19     response.setContentType("text/html;␣charset=UTF-8");
20     PrintWriter out = response.getWriter();
21     out.println("<html><head></head><body>");
22
23     Calendar cal = Calendar.getInstance();
24     out.printf("%d年%d月%d日%s曜日%d時%d分%d秒%n",
25         cal.get(Calendar.YEAR),
26         cal.get(Calendar.MONTH)+1, cal.get(Calendar.DAY_OF_MONTH),
27         youbi[cal.get(Calendar.DAY_OF_WEEK)-1],
28         cal.get(Calendar.HOUR_OF_DAY),
29         cal.get(Calendar.MINUTE), cal.get(Calendar.SECOND));
30
31     out.println("</body></html>");
32     out.close();
33   }
34 }
```

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.http.〜
```

[1] 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。

という形の 5 つの 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。

@WebServlet("/MyDate") は Servlet の URI のパスを指定するためのアノテーションである。クラスファイルに/MyDate というパスの情報が書き込まれ、コンテナーがその情報を読み出して、そのパスに Servlet を配備する。

Servlet の処理は基本的に doGet（または後述の doPost）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 ServletException, IOException
```

からわかるように、doGet/doPost は HttpServletRequest クラス、HttpServletResponse クラスの 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 ServletException, IOException の部分は、この doGet というメソッドが、ServletException や 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=UTF-8");
```

は、以下に続くデータが HTML のデータで文字コードが UTF-8 であるということをブラウザーに伝える役割を持つ。

また、

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

は、ブラウザー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 は書式制御の機能つきの出力メソッドである。C 言語の printf 関数に相当する。%d や%s などの書式制御は C 言語の printf のときとほぼ同じ意味である。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、ブラウザーに表示されるときの文字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 アプリケーションルートに、画像ファイル（例えば images/foo.png）を用意しておいて、サーブレットでと出力すると画像を表示できる。

現在の秒によって、ブラウザー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 body タグで <body background='images/foo.png'> ... </body>のように指定する。
- 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

以下では、Eclipse 等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。このパスは使用する Web アプリケーションサーバーにより異なる。

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く。

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、Web アプリケーションサーバーのデフォルトのディレクトリがある。このデフォルトディレクトリの子として例えば *JouhouKankyouJikken2* というディレクトリを作成し、このデ

ィレクトリを Web アプリケーションルートとする（つまり、*JouhouKankyouJikken2* の子として WEB-INF ディレクトリを作成する）と、

http://*hostname*:8080/JouhouKankyouJikken2/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 Web アプリケーションサーバー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 ブラウザーは同一のコンピューター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、アクセスカウンターにせよ、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　アクセスカウンター

アクセスカウンターはもっとも代表的なサーバー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 簡易版アクセスカウンター Servlet では、アクセスの回数はサーバー上のファイルに記録しておく。

ファイル Counter.java

```
1  import java.io.BufferedReader;
2  import java.io.File;
3  import java.io.FileNotFoundException;
4  import java.io.FileReader;
5  import java.io.FileWriter;
6  import java.io.IOException;
7  import java.io.PrintWriter;
8
9  import javax.servlet.ServletException;
10 import javax.servlet.annotation.WebServlet;
11 import javax.servlet.http.HttpServlet;
12 import javax.servlet.http.HttpServletRequest;
13 import javax.servlet.http.HttpServletResponse;
14
15 @WebServlet("/Counter")
16 public class Counter extends HttpServlet {
17   @Override
```

```
18   protected void doGet(HttpServletRequest request,
19                        HttpServletResponse response)
20                 throws ServletException, IOException {
21     response.setContentType("text/html;␣charset=UTF-8");
22     PrintWriter out = response.getWriter();
23     out.println("<html><head></head><body>");
24     int i;
25
26     File f = new File(getServletContext()
27                       .getRealPath("/WEB-INF/counter.txt"));
28     BufferedReader fin = null;
29     try {
30       fin = new BufferedReader(new FileReader(f));
31       i =  Integer.parseInt(fin.readLine());
32     } catch (FileNotFoundException      // ファイルがなければ
33           | NullPointerException       // ファイルが空なら
34           | NumberFormatException e) { // 数でないならば
35       i = 0;  // 0に
36     } finally {
37       if (fin!=null) {
38         fin.close();  // closeを忘れない
39       }
40     }
41
42     PrintWriter fout = new PrintWriter(new FileWriter(f));
43     fout.println(++i);
44     fout.close(); // closeを忘れない
45
46     out.printf("あなたは␣%d番目の来訪者です。%n", i);
47     out.println("</body></html>");
48     out.close();    // closeを忘れない
49   }
50 }
```

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、***(J2EEAPI)***/javax/servlet/http/HttpServlet.html）を参照すること。

（参考）次のような簡単なサーブレットを実行してみると、getRealPath

メソッドの結果を確認することができる。

ファイル GetRealPathExample.java

```
 1 import java.io.IOException;
 2 import java.io.PrintWriter;
 3
 4 import javax.servlet.ServletException;
 5 import javax.servlet.annotation.WebServlet;
 6 import javax.servlet.http.HttpServlet;
 7 import javax.servlet.http.HttpServletRequest;
 8 import javax.servlet.http.HttpServletResponse;
 9
10 @WebServlet("/GetRealPathExample")
11 public class GetRealPathExample extends HttpServlet {
12   protected void doGet(HttpServletRequest request,
13                        HttpServletResponse response)
14                   throws ServletException, IOException {
15     response.setContentType("text/plain");
16     PrintWriter out = response.getWriter();
17     out.println(getServletContext().getRealPath("/WEB-INF"));
18     out.close();
19   }
20 }
```

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 クラスはファイルから文字ストリームを読み込むためのクラス、BufferedReader クラスは文字をバッファリングすることによって、文字ストリームから文字を効率良く読み込むためのクラスである。FileReader クラス、BufferedReader クラスの一般的な使用法は API 仕様で確認することができる。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。

FileWriter クラスはファイルに文字ストリームを書き込むためのクラス、PrintWriter クラスは文字ストリームのフォーマットされた出力のためのクラスである。FileWriter クラス、PrintWriter クラスの一般的な使用法は API 仕様で確認することができる。

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 1,1 | ... | 例外型 1,k1 変数 1) {
  文の並び 1
}
  ...
  catch (例外型 n,1 | ... | 例外型 n,kn 変数 n) {
  文の並び n
}
```

文の並び $_0$ で例外が起こらなかった場合は、そのまま次へ行く。

文の並び $_0$ の中で、例外が起こったときには 文の並び $_0$ の間の残りの文は無視され、例外が例外型 $_{i,1}$〜例外型 $_{i,k_i}$（ただし $i = 1 \cdots n$）にマッチするならば、文の並び $_i$ が実行される。マッチする例外がなかった場合には、文の並び $_i$（$i = 1 \cdots n$）は実行されない。

この場合、現在実行しているメソッドを呼び出した式を囲んでいる try〜catch 文を探す。それもなければ、さらにメソッド呼出しの履歴をさかのぼって、メソッド呼出しを囲んでいる try〜catch 文を探す。それでもなければプログラムを終了する。

また、最後に “**finally** { 文の並び }” という形（finally 節）がつく場合もある。その場合、finally 節の文の並びは例外が起こったか否かにかかわらず、必ず最後に実行される。

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、カウンターの値を 0 に設定している。

問 1.8.1 カウンターの値が特別な値（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。（整数の割算の剰余を求める演算子は、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ーや時刻表示 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ーができる。

数字を画像として表示するアクセスカウンターを作成せよ。

参考: 数字画像データ

- Digit Mania（http://www.digitmania.holowww.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ー 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、Web アプリケーションサーバーの中で保持され 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ーを保持しておけば、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ーを実現できる。

ファイル Counter0.java

```
1  import java.io.IOException;
2  import java.io.PrintWriter;
3
4  import javax.servlet.ServletException;
5  import javax.servlet.annotation.WebServlet;
6  import javax.servlet.http.HttpServlet;
7  import javax.servlet.http.HttpServletRequest;
8  import javax.servlet.http.HttpServletResponse;
9
10 @WebServlet("/Counter0")
11 public class Counter0 extends HttpServlet {
12   int i=0;          // フィールドとして宣言する
13
14   @Override
15   protected void doGet(HttpServletRequest request,
16                        HttpServletResponse response)
17                 throws ServletException, IOException {
18     response.setContentType("text/html;␣charset=UTF-8");
19     PrintWriter out = response.getWriter();
20     out.println("<html><head></head><body>");
21     out.printf("あなたは␣%d番目の来訪者です。", i++);
22     out.println("</body></html>");
23     out.close();    // close を忘れない
24   }
25 }
```

ただし、ファイルに書き込まないので、Web アプリケーションサーバーを再起動すると、カウンターが 0 に戻ってしまう。完全なアクセスカウンターにするためには、Web アプリケーションサーバーの終了時にカウンター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ー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、Web アプリケーションサーバー起動・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ーに改良せよ。

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

```
1  import java.io.File;
2  import java.io.IOException;
3  import java.io.PrintWriter;
4
5  import javax.servlet.ServletException;
6  import javax.servlet.annotation.WebServlet;
7  import javax.servlet.http.HttpServlet;
8  import javax.servlet.http.HttpServletRequest;
9  import javax.servlet.http.HttpServletResponse;
10
11 @WebServlet("/DirIndex")
12 public class DirIndex extends HttpServlet {
13   protected void doGet(HttpServletRequest request,
14                        HttpServletResponse response)
15                 throws ServletException, IOException {
16     response.setContentType("text/html;␣charset=UTF-8");
17     PrintWriter out = response.getWriter();
18
19     String path = getServletContext()
20                   .getRealPath("/"); //適切なパスに変更する
21     File dir = new File(path);
22     String[] files = dir.list();  // dirにあるファイル名の配列を得る
23
24     out.println("<html><head></head><body>");
25     out.println("<pre>");
26
27     int i;
28     out.printf("%sのファイル一覧%n%n", path);
29     for (i=0; i<files.length; i++) {
30       out.println(files[i]);      // filesの各要素を順に出力
31     }
32
33     out.println("</pre>");
```

```
34      out.println("</body></html>");
35      out.close();
36    }
37  }
```

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 メソッドを用いる。

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）