Javaの イケてるコード 残念なコード

そのコーディングスタイルはもう古い？ Javaの新定石を学ぶ（後編）

Acroquest Technology
谷本 心

前回は、Javaの新しい書き方について紹介してきました。今回は、その第2弾として、前回で紹介しきれなかった便利なライブラリなどを紹介していきます。

さらに便利なファイル読み込み

前回、Java SE 7で導入されたtry-with-resourcesを紹介しました。try-with-resources機能を使うことで、リソースのクローズ処理をより安全にかつ簡単にできるようになりました。具体的には、**リスト1**のコードです。ポイントは、try節の中でリソースを宣言することで、自動的にリソースをクローズしてくれるというものです。

ただ、ファイルの読み込み処理については、もっと便利なクラスがJava SE 7に用意されています。それが「java.nio.file.Files」クラスです。このクラスを利用すると、ファイル読み込みは、**リスト2**のように1行で書けます。

これまでは、リスト1のようなファイル読み込み処理を記述するか、またはApache Commons IO[*1]やGoogle Guava[*2]のような外部ライブラリを利用する必要がありました。しかし、Java SE 7以降では、Filesクラスを利用することで外部ライブラリを使うことなく、簡単にファイルの読み込みができるようになりました。ファイルの読み込み処理だけならば、java.nio.file.Filesクラスを使うことが、新しい定石となるでしょう。

なお、このFilesクラスには、ファイルをコピーするcopyやシンボリックリンクを作成するcreateSymbolicLinkなど、便利なメソッドが用意されています。これまでは冗長な処理を書いたり、システムコール[*3]を行なわなければならなかった処理を、簡単に書けるようになったのは、うれしいところですね。

リスト1●try-with-resourcesを使ったプログラム

```
try (InputStream in = new FileInputStream("test.txt");
  Reader inReader = new InputStreamReader(in);
  Reader reader = new BufferedReader(inReader)) {
    String line;
    while ((line = reader.readLine()) != null) {
      lines.add(line);
    }
  } catch (IOException e) {
    // 例外処理をする
  }
}
```

try節にリソースを宣言する

リスト2●java.nio.file.Filesクラスを使ったファイルの読み込み処理

```
public static List<String> readFile3(String fileName) throws IOException {
  return Files.readAllLines(Paths.get(fileName), Charset.forName("UTF-8"));
}
```

1行でファイルの読み込み処理が書ける

プロパティファイルで日本語を使う

Javaには、Propertyクラスというプロパティファイルを読み込むクラスがあります。プロパティファイルとは、定数やファイルパスなどの情報をまとめたもので、プログラムとは別のファイルにまとめておくことが推奨されています。使い方は、**リスト3**のようになります。

このプロパティファイルですが、残念なことに日本語を直接扱うことができず、native2asciiというコマンドを用いて、コードを変換する必要がありました。例えば、「message=テスト」というプロパティファイルは、native2asciiで変換することによって「message=¥u30c6¥u30b9¥u30c8」のような形式になります。これは、コンピュータが読むための形式であり、人間が読むことは難しいといえます。実際のコーディングでは、毎回native2asciiコマンドで変換するのは面倒なので、native2asciiコマンドを実行するのではなく、Eclipseのプラグインなどを利用して自動的に変換することがほとんどでした。

このnative2asciiコマンドやEclipseプラグインによる変換も、今となっては昔の話です。Java SE 6以降では、プロパティファイルに普通の日本語を利用できるようになったのです。プロパティファイルを読み込む処理は、**リスト4**のようになります。

リスト3のソースコードと見比べてもあまり違いがわからないのですが、Propertyクラスのloadメソッドの引数が変わっています。Java SE 6より前では、文字エンコーディングが指定できないInputStreamクラスしか引数に渡すことができませんでした。しかし、Java SE 6以降では、文字エンコーディングが指定できるReaderクラス（インタフェース）を引数として渡せるようになったのです。おかげでプロパティファイルの文字エンコーディングを指定できるようになり、プロパティファイルに日本語が使えるようになったのです。もはやnative2asciiやプロパティエディタなどのプラグインも古いツールと言えるでしょう。

リスト3●プロパティファイルから値を読み込むプログラム

```
Properties props = new Properties();
InputStream in = getClass().getResourceAsStream("/test.properties");
props.load(in);
in.close();

// 以下のように記述してプロパティファイルの値を取得できる
System.out.println(props.getProperty("message"));
```

リスト4●Java SE 6以降のプロパティファイルの読み込み

```
Properties props = new Properties();

try (InputStream in =  getClass().getResourceAsStream("/test.properties");
  Reader reader = new InputStreamReader(in, "Windows-31j")) {
  props.load(reader);
}
```

文字エンコーディングが指定できるReaderクラスを渡せる

```
System.out.println(props.getProperty("message"));
```

リスト5●タイムスタンプを取得して確認するプログラム

```
private Map<String, String> cache = new HashMap<>();
private Map<String, Long> lastLoadedMap = new HashMap<>();
private File dir = new File("/etc/files/");
private long interval = 1000L;

void checkFiles() throws InterruptedException {
  while (true) {
    for (File file : dir.listFiles()) {
      try {
        String path = file.getAbsolutePath();
        long lastModified = file.lastModified();
        Long lastLoaded = lastLoadedMap.get(path);
        if (lastLoaded == null || lastModified > lastLoaded.longValue()) {
          // ファイルが更新されていた場合、中身を読み込んで保持する
          cache.put(path, readFile(file));
          // 読み込んだファイルのタイムスタンプを保持する
          lastLoadedMap.put(path, lastModified);
        }
      } catch (IOException ex) {
        System.err.println("ファイルの読み込みに失敗したが処理を継続");
        ex.printStackTrace();
      }
    }
    Thread.sleep(interval);
  }
}
```

更新状況を確認

1秒間処理を止める

ファイルの変更や更新を監視したい

ファイル操作関連のイマドキのコードをもう一つ紹介します。Javaでシステムを開発していると、「ファイルが変更されたことを検知して処理を実行したい」という要望が出てくることがあります。例えば、設定ファイルなどが更新されたタイミングで読み込んで、システムに反映させるという

*1 Webサイト（http://commons.apache.org/io/）を参照。
*2 Webサイト（http://code.google.com/p/guava-libraries/）を参照。
*3 システムコールとは、OSの関数でアプリケーションから呼び出せるものです。この関数が呼び出されると、CPUに割り込み要求が発生して、あらかじめ決められた特定の割り込みルーチンが実行されます。