1.目的

テンプレートエンジンの考え方を理解するとともに、java のテンプレートエンジンである「Velocity」の使い方を理解することを目的とする。※また、ウィキペディアやブログの情報も含んでいるため、このレポートに関してすべてを鵜呑みにしてはならない。

2.テンプレートエンジン

2.1 テンプレートエンジンとは

Fig.1 にあるように、テンプレートとデータを分離して管理し、それをまとめることで、ひとつのドキュメントを作成するいわゆるコンパイラのようなものである。テンプレートとデータを分離すること言う考え方は、MVC モデルで言うところの View に相当している[1]。(MVC モデルでは、プログラムを Model,View,Controller という3つの要素に分割してお互いに呼び出しあって処理が実行されていきます。)

テンプレートは、共通部分を抽象化して効率化を図るという設計思想がある。このため、似たような構成のページである web ページの設計は楽になるはずである。なぜなら、ヘッダ・ログイン名など共通して作成できるからである。しかし、テンプレート内で、for/foreach などの繰り返し文や if 文などの条件分岐文を記述しなければならない点で、MVC モデルの View に似ているといえど、controller が混ざっていることになることになる[2]。

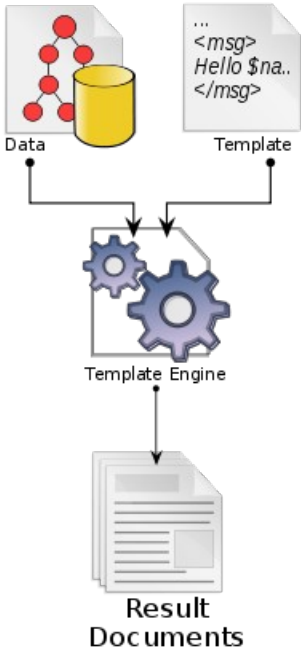

Fig. 1: テンプレートエンジンの処理フロー
**[1]より参照*

2.2 テンプレートエンジンの有名所？

本当に多くのテンプレートエンジンが存在している。

Table 1: テンプレートエンジンの一例

テンプレートエンジン名	対応言語	ライセンス
Smarty	php	LGPL
SimpleTemplate	php	BSD License
Symfony2	php	MIT License
velocity	java	Apache License 2.0
FreeMarker	java	BSD-like License
Mustache(マスタッシュ)	多言語	Apache License 2.0

Table 1に示してあるのは、ほんの一例で、perl・ruby・python のテンプレートエンジンも数多く存在する。php では Smarty が有名で、java ベースのテンプレートエンジンは、velocity と FreeMarker が有名らしい。

2.3 ライセンスと MVC モデルについて

今回のレポートの内容とは異なるため割愛するが、オープンソースのライセンスについてと MVC モデルは知っておいたほうがいいと思う。MVC モデルについては[3]の web ページがわかりやすいかもしれない。

---

[1]ウィキペディア「テンプレートエンジン」:http://ja.wikipedia.org/wiki/
[2]テンプレートの考え方:http://pukiwiki.tuntunkun.com/index.php
[3]MVC モデルを絵でわかりやすく説明:http://hijiriworld.com/web/mvc-concept/

3.Velocity

Velocity の使い方は、[4]で非常にわかりやすくまとめられている。よって、このレポートの内容を読んだ後に復習として、読んで実践してみると良いだろう。

まず、次の2つの言葉を覚えます。

・VelocityContext

・Velocity Template Language(VTL)

VelocityContext とは、テンプレート内で書かれた変数名と java の中での変数名の対応表のようなもの。

VTL とは、テンプレートを作るときに使用する言語で、変数名の使用や繰り返し文/条件分岐文などが使用できる。

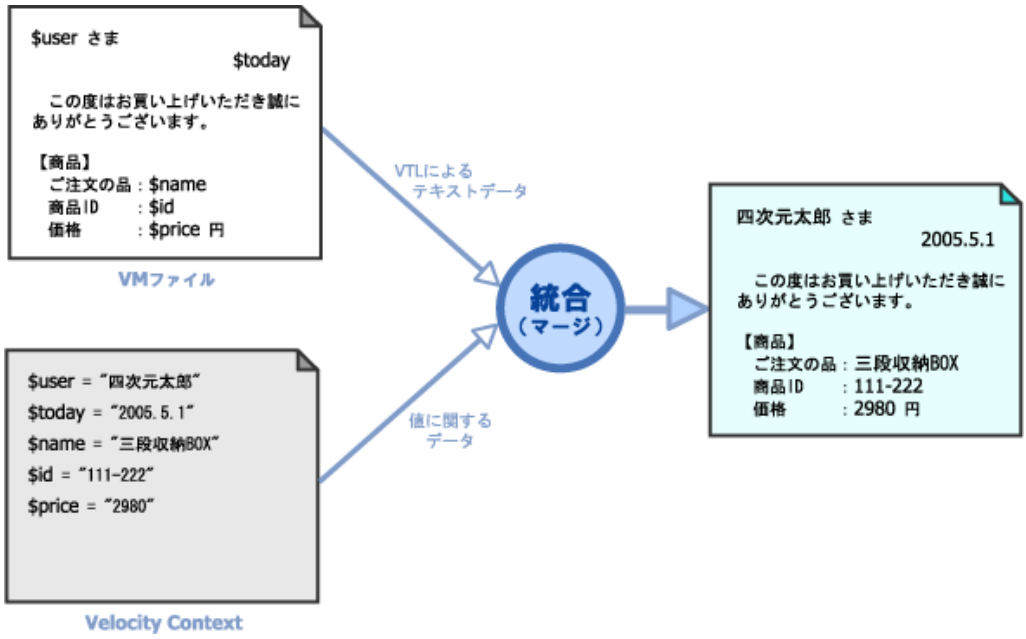

*Fig. 2: 簡単な例　*[4]より参照*

VelocityContext と VTL を使った.vm ファイルを、統合してひとつのドキュメントが出来上がる。

ここで、java がどこで使われているかというと、VelocityContext を生成する部分と、それらを統合する部分である。VelocityContext を生成するのは手書きでもおkだと思うのだけれども、それだと、例えば Fig.2 の $user の値が常に「四次元太郎」になってしまう。よって、VelocityContext は java で動的に作る必要がある。

3.1 Velocity を使ってみよう[5]

1. ダウンロード

Velocity の公式サイトのダウンロード http://velocity.apache.org/download.cgi から最新のバージョンの Velocity Engine を選択してダウンロードし適当に解凍します（現時点では velocity-1.7 が最新版）。この時に Engine 以外にも tools などがあるので、Engine をダウンロードします。

---

[4]Velocity：http://www.techscore.com/tech/Java/ApacheJakarta/Velocity/index/
[5]Velocity で Helloworld：http://wdsdx.com/node/55

2. プロジェクトの作成

解凍したファイルのうち、以下の JAR ファイルを外部 JAR として追加する。logkit 以外は基本的に必須です。

・velocity-1.7.jar
・lib/commons-collections-3.2.1.jar
・lib/commons-lang-2.4.jar
・lib/oro-2.0.8.jar
・lib/avalon-logkit-2.1.jar

次に Velocity の設定ファイルを作ります。

velocity.properties

```
runtime.log.logsystem.avalon.logger = velocity
input.encoding = UTF-8
output.encoding = UTF-8
```

プロジェクトの構成は進めていくと、右の Fig.3 のようになります。

あとは、扱う情報を保存するクラス（ここでは src/entity/に入れる）を作り、その情報をどのように表示するかを VTL で記述した.vm ファイルを作り、それらを統合（マージ）してドキュメントを作るクラスを作れば良い。

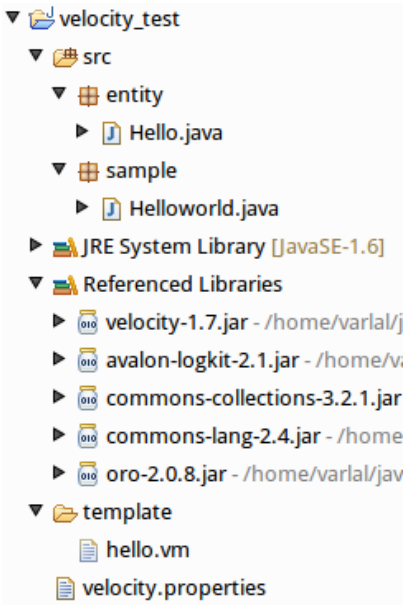

Fig. 3: プロジェクトの構成

3. エンティティの作成

表示したい情報（データ）を保持するクラスを作成します（これをエンティティ=実体と呼ぶことにする）。

src/entity/Hello.java

```
package entity;
import java.util.List;

public class Hello {

    private String name;
    private List<String> list;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public List<String> getList() {
        return list;
    }
    public void setList(List<String> list) {
        this.list = list;
    }
}
```

4.テンプレートの作成

さっき作成したHelloクラスをどのように表示させるかを記述したテンプレートを作ります。

template/hello.vm

```
$hello.name
---------------------
#foreach($s in $hello.list)
    リスト：$s
#end
---------------------
```

５．サンプルクラスの作成

０．エンティティにデータを入力する。

１.エンティティからVelocityContextを作成。

２.vmファイルを読み込む。

３．VelocityContextと.vmファイルをマージする。

４．マージ結果を標準出力する。

という動作を行うクラスを作成する。

src/sample/Helloworld.java

```
package sample;
import java.io.StringWriter;
import java.util.Arrays;
import org.apache.velocity.Template;
import org.apache.velocity.VelocityContext;
import org.apache.velocity.app.Velocity;
import entity.Hello;

public class Helloworld {
    public static void main(String[] args) {
        Hello hello = new Hello();
        hello.setName("Hello World!!");
        hello.setList(Arrays.asList("h", "e", "l", "l", "o"));

        try {
            Velocity.init("velocity.properties");
            VelocityContext context = new VelocityContext();
            context.put("hello", hello);
            StringWriter writer = new StringWriter();
            Template template = Velocity.getTemplate("template/hello.vm");

            template.merge(context, writer);
            System.out.println(writer.toString());
            writer.flush();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

５．実行

実行すると以下のようになる。

```
Hello World!!
--------------------
    リスト：h
    リスト：e
    リスト：l
    リスト：l
    リスト：o
--------------------
```

６．サンプルのクラスのソースを読む【理解】

さっき書いた src/sample/Helloworld.java にコメントをつけることで、どこで何をしているのかを理解します。まずは自分でやってみた後、下のコメント例を参考にしてください。

```
public static void main(String[] args) {
        //データを保存するインスタンスを作る。（エンティティの作成）
        Hello hello = new Hello();
        //インスタンスにデータを入れる。
        hello.setName("Hello World!!");
        hello.setList(Arrays.asList("h", "e", "l", "l", "o"));

        try {
            //設定ファイルを読み込む。
            Velocity.init("velocity.properties");

            //VelocityContextを作成する。
            VelocityContext context = new VelocityContext();
            //テンプレート内の変数helloとjava内のインスタンス（エンティティ）hello　を関連付ける。
            context.put("hello", hello);

            //ドキュメントを保存するwriterを作る。
            StringWriter writer = new StringWriter();

            //テンプレートを読み込む。
            Template template = Velocity.getTemplate("template/hello.vm");

            //テンプレートとVelocityContextをマージしてwriterに保存する。
            template.merge(context, writer);

            //マージした結果のwriterを出力する。
            System.out.println(writer.toString());

            //flushする。
            writer.flush();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

7．どうやって繋がっているかを把握する【理解】

サンプルのクラスにコメントをつけることで、VelocityContextと.vmファイルをマージする流れは理解できたと思います。次にどのように繋がっているかを見ていきます。src/entity/Hello.java の中のgetは使用していないので消して、実行してみます。

src/entity/Hello.java

```
package entity;
import java.util.List;

public class Hello {
    private String name;
    private List<String> list;

    public void setName(String name) {
        this.name = name;
    }
    public void setList(List<String> list) {
        this.list = list;
    }
}
```

結果:

```
$hello.name

----------------------
----------------------
```

となってしまい。読み込めていないことがわかります。
よって、**テンプレートのプロパティ参照には、getterを使っている**ことがわかります。

また、setやgetを作るのが面倒なとき、例えば以下のような例では、
右クリック － Sorce － Generate getters and setters で自動生成できます（Eclipse）。
velocityのエンティティを作る際に活躍するでしょう。

```
import java.util.Date;

public class document {
    public String DocumentName;//題名
    public Date   RegistDate;    //登録日付
    public String Authur;        //著者
    public String Keyword;       //キーワード
    public String Abstruct; //概要・アブストラクト
}
```

8．マージした時の出力先【理解】

今回、マージしたドキュメントを保存してある変数writerは、StringWriterクラスです。これは、java.io.StringWriterで使えるようになる、いわゆる標準(画面)出力用のバッファです。

```
//テンプレートとVelocityContextをマージしてwriterに保存する。
template.merge(context, writer);
```

の第二引数はWriterクラスが指定されています。Writerクラスの直系のサブクラスは以下のようなものがあります。単に画面に出力したい時はStringWriterを使い、ファイルに出力したい時はBufferedWriterやPrintWriterを使うと良いでしょう。

Writerクラスの直系のサブクラス
BufferedWriter, CharArrayWriter, FilterWriter, OutputStreamWriter, PipedWriter, PrintWriter, StringWriter