Velocity Template Engine

Avagy miért ne írjunk saját template engine-t Java nyelven?

Viczián István (viczus@freemail.hu)

E cikk olyan haladó Java programozóknak szól, akik már kerültek szembe olyan problémával, hogy dinamikus tartalmat kell statikus szöveg közé szúrni. Ilyen eset lehet például körlevél küldése, HTML vagy XML oldal előállítása akár különálló alkalmazásban, akár szervletekben. Ebben a cikkben az ingyenes Velocity Template Engine alapjait mutatom be, mely megkönnyíti ezen a feladatok megoldását.

Bizonyára sokunk került már szembe azzal a problémával, hogy adott egy szöveges formátumú *sablon* (továbbiakban *template*), melyet ki kell tölteni az üzleti rétegből nyert szöveges adatokkal. Valószínűleg erre sokunk válasza az volt, hogy írjuk meg magunk. Ezen próbálkozások általában átláthatatlan kódot, speciális feladatot megoldó *template engine*-t, és az egyre több igény kielégítésére egy saját template nyelv kialakítását eredményezték.

Tipikus példa erre webes alkalmazások fejlesztésekor a statikus HTML tartalom kitöltése dinamikusan generált, vagy adatbázisból nyert adatokkal. Másik példa levelek küldésekor egy általános szöveges tartalom feltöltése konkrét megszólításokkal, értékekkel. Harmadik példaként megemlíthető XML fájlok generálása, ahol a template tartalmazhatja a tag-eket, attribútumok neveit, és az attribútumok értékeit vagy a tag-ek közti részeket kell tartalommal feltölteni.

Eredményesebb megoldás lehet egy konkrét megvalósítás beépítése az alkalmazásunkba. Ezekből rengeteg található az Interneten, mind szabad forráskódú, mind kereskedelmi termékként.

Én elsősorban az ingyenes termékek között kutakodtam, bemutatásra a Jakarta Project keretein belül fejlesztett Velocity Template Engine-t választottam, hiszen az Apache Software Foundation név már bizonyított a szabad szoftverek világában, a Jakarta Project-es termékeket megbízhatóság, megfelelő komplexitás, kidolgozottság, dinamikus fejlesztés jellemzi.

Szerencsére a Velocity is az Apache Software License v1.1 hatálya alá tartozik, mely kevés megszorítást tartalmaz, így azon kívül, hogy alkalmazásunk dokumentációjában megemlítünk egy mondatot ezzel kapcsolatban, és mellékeljük a licence fájlt, semmit nem kell tennünk a felhasználásához és tovább terjesztéséhez.

E cikk nem csak ezen termék megismeréséhez lehet hasznos, hanem bemutatja az alapfogalmakat és megoldásokat, körvonalazódhat, hogy milyen elvárásaink lehetnek egy hasonló alkalmazással szemben.

A cikk írásakor az aktuális verzió a 1.3.1-rc2 volt, tehát az implementációs részek ezen verzió meglétét igénylik, bár az általam említett példákban nem mutatkoznának a verziók közti különbségek.

A Velocity tehát egy Java nyelven írt, szabad forráskódú szoftvercsomag, mely template engine funkciókat lát el, rendelkezik egy jól használható API-val, illetve egy átgondolt, jól dokumentált, minden vezérlési struktúrával rendelkező template nyelvvel (Velocity Template Language - VTL). A cikk egyikben sem mélyül el, hanem konkrét példán keresztül mindkettő alapjait párhuzamosan bemutatja.

A Velocity legjelentősebb felhasználási területe a webes alkalmazások készítése, ezen belül is a szervlet technológia kiegészítőjeként terjedt el.

Előnye, hogy alkalmazásával külön választható a HTML és a Java kód, ami nagyobb rendszerek fejlesztésekor különösen fontos, hiszen sem a webdesigner nem akar megtanulni Java-ban programozni, sem a programozó nem akar órákat tölteni a megálmodott oldal beillesztésével. A Velocity válasza a problémára, hogy mindketten tanuljanak meg egy egyszerűsített, de mégis sok funkcióval rendelkező köztes nyelvet, mely összeköti a template-et és a kóddal előállított adatokat. (Tapasztalatom szerint a gyakorlatban a programozó írja meg a köztes kódot is, és a designer hamarabb megérti, kevésbé rontja el.)

A Velocity beleillik az objektumorientált szemléletbe is, hiszen a template-ben kulcsokat deklarálhatunk, melyekhez kódból objektum példányokat rendelhetünk, azaz változóként hivatkoznak egy objektumra. Az objektumok metódusait is meghívhatjuk, akár paraméterezve is, ha `Vector`, `Hashtable` vagy `Array`, akkor végigiterálhatunk az elemein, vagy egyszerűen kiírhatjuk az objektumot, vagy a metódus visszatérési értékét (ekkor a `toString` metódus, primitív típus esetén a hozzá tartozó osztály `toString` metódusát hívja meg a Velocity).

És végre elérkeztünk a kódhoz, lássuk, hogy is néz ki mindez a gyakorlatban. Példánk legyen egy webes CD katalógus, melyben listázni, lekérni, felvinni lehet a lemezeket. A lista XML-ben is elérhető legyen, és egy lemez felvitelekor generáljunk egy szöveges levelet. A példa így bemutatja, hogyan használható a Velocity HTML, XML és általános szöveges stream vagy fájl generálására.

A Velocity installálása nagyon egyszerű, használatba vételéhez csupán a JAR fájlt kell a szervlet konténer számára elérhetővé tenni.

Általánosan használt objektum a `Disc`, forráskódja (`Disc.java`):

```
public class Disc {
  private String myArtist;
  private String myTitle;
  private String myInfo;

  Disc(String artist; String title; String info)
  {
    myArtist = artist;
    myTitle = title;
    myInfo = info;
  }

  public String getArtist() {
    return myArtist;
  }

  public String getTitle() {
    return myTitle;
  }

  public String getInfo() {
    return myInfo;
  }
}
```

Az első példa írja ki egy album adatait. Ehhez szükségünk vagy egy template-re, `disc.vm`:

```
<html>
  <body>
    <h1>$disc.artist - $disc.title</h1>
    <p>$disc.info</p>
  </body>
</html>
```

Látható, hogy a valtozók előtt $ karaktert kell használnunk, a tulajdonságok minősítésére pontot.

Az ehhez tartozó szervlet, mely a disc kulcshoz hozzárendeli az objektumot (`InfoServlet.java`):

```
public class InfoServlet extends VelocityServlet
{
  public Template handleRequest(
       HttpServletRequest request,
       HttpServletResponse response,
       Context context )
  {

    Disc disc = new Disc("Jewel", "This way",
       "Blahblah.");

    context.put("disc", disc);

    Template template = null;

    try {
      template =  getTemplate("discinfo.vm");
    }
    catch( ResourceNotFoundException rnfe )
    {
      // nem találja a template fájlt
    }
    catch( ParseErrorException pee )
    {
      // a template szintaktikai hibás
    }
    catch( Exception e )
    {
      // egyéb hiba
    }

    return template;
  }
}
```

A szervlet ősosztálya a `VelocityServlet` lesz, melynek `handleRequest` metódusa szolgálja ki a http kéréseket. Visszatérési értéke a `Template` osztály egy objektuma, mely a nyers template-et tartalmazza.

A visszaadott HTML pedig a következő lesz (`Disc.html`):

```
<html>
  <body>
    <h1> Jewel - This way </h1>
    <p>Blahblah.</p>
  </body>
</html>
```

A példában láthatjuk, hogy az `artist` és `title` tulajdonságot a Velocity feloldja `getArtist` és `getTitle` metódushívásokra.

Most lássuk a Velocity ennél bonyolultabb szolgáltatását, méghozzá az iterációt. A lemezek listázásához a következő template fájl szükséges (`disclist.vm`):

```
<html>
  <body>
    <table>
      <tr>
        <td>Előadó</td>
       <td>Cím</td>
      </tr>
      #foreach ($disc in $discs)
        <tr>
         <td>$disc.artist</td>
         <td>$disc.title</td>
        </tr>
      #end
    </table>
  </body>
</html>
```

Látható, hogy a vezérlési szerkezeteket # karakterrel kell jelölni.

A szervletben ekkor a `handleRequest` metódus első két sorát a következőre kell kicserélnünk:

```
Vector discs = new Vector();
Disc disc;
disc = new Disc("Jewel", "This way",
       "Blahblah.");
discs.add(disc);
disc = new Disc("Jewel", "Spirit",
       "Blahblah.");
discs.add(disc);

context.put("disc", discs);
```

Ekkor a szervlet a következő HTML fájlt fogja visszaadni (`Discs.html`):

```
<html>
  <body>
    <table>
      <tr>
        <td>Előadó</td>
         <td>Cím</td>
      </tr>
      <tr>
        <td>Jewel</td>
         <td>This way</td>
        </tr>
         <tr>
        <td>Jewel</td>
         <td>Spirit</td>
        </tr>
       </table>
  </body>
</html>
```

Ha XML-ben szeretnénk viszontlátni kedvenc adatainkat, használjuk ugyanezen kódot más template-tel:

```
<?xml version="1.0"?>
<discs>
  #foreach ($disc in $discs)
    <disc>
      <artist>$disc.artist</artist>
      <title>$disc.title</title>
    </disc>
  #end
</discs>
```

Látható, hogy ugyanazt az üzleti logikát alkalmazva különböző formátumú kimeneteket gyárthatunk, csupán más template használatával. Az adat és a formátum így teljesen szétválik.

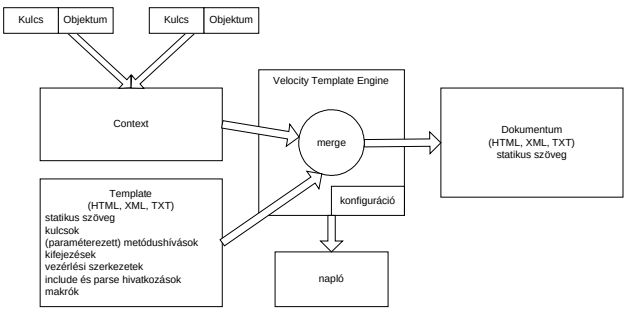
Abban az esetben, ha a Velocity-t nem szervletből, hanem külön alkalmazásból, vagy más beállításra lenne szükségünk, és a feldolgozás eredményét `String`-be szeretnénk látni, akkor használjuk a következőféleképpen. Először a template:

```
Örömmel jelentjük, hogy az adatbázisunkba
bekerült $disc.artist előadótól a $disc.title
című album.
```

Majd a forrás:

```
VelocityEngine ve = new VelocityEngine();
ve.init();

Disc disc = new Disc("Jewel", "This way",
       "Blahblah.");

VelocityContext context = new VelocityContext();

context.put("disc", disc);

Template t = ve.getTemplate(
       "discinfomail.vm" );

StringWriter writer = new StringWriter();

t.merge(context, writer);
```

Ez a forrás mutatja azon alapvető lépéseket, melyek egy részét a `VelocityServlet` osztály elrejt. Először inicializálni kell az `VelocityEngine`-t, majd létrehozni egy `Context` objektumot, melybe a kulcs és objektum párokat kell beletenni, végül ki kell választani a template-et, és az adatokat behelyettesíteni.



1. ábra Velocity működési elve

Persze a Velocity VCL ennél sokkal bonyolultabb, lehetőség van megjegyzések beszúrására, egyszerűbb kifejezések kiértékelésére, metódusok paraméterezésére, feltételek, ciklusok definiálására, template-ek másik template-be illesztésére (akár feldolgozva, akár feldolgozatlanul), makrók definiálására.

Lehetőség van `Context` objektumok összefűzésére is, ilyenkor a később létrehozott `Context` konstruktorában kell megadni az előző `Context`-et.

A Velocity-t a fentiekben két módon használtuk. Az első esetben a szervleteknél a JVM-ben egy példány helyezkedett el (Singleton Model), ami lehetővé teszi a szervletek számára a naplózó mechanizmus és a template-ek megosztását. Ha ezt nem szervleteknél használjuk, akkor a Velocity osztály statikus metódusait kell hívnunk az inicializáláshoz, szolgáltatásai eléréséhez. A második esetben egy `VelocityEngine`-t példányosítottunk (Separate Instance), ami lehetővé teszi, hogy más konfigurációval használjuk a template engine-t ugyanazon alkalmazásban.

A Velocity-t konfigurálhatjuk kódból `Properties` osztály használatával, vagy megadhatjuk közvetlenül a konfigurációs állomány útvonalát, szervleteknél a `WEB.XML` fájlban helyezzük el a következő részletet:

```
<context-param>
  <param-name>properties</param-name>
  <param-value>/velocity.properties</param-
value>
</context-param>
```

A konfigurációval megadhatjuk a naplózó rendszert (pl. használható a szintén Jakarta Project-es Log4J), karakterkódolást, template fájlok helyét.

Magyar nyelvű alkalmazások készítőinél gyakran komoly probléma a template-ben szereplő magyar ékezetes karakterek helyes megjelenítése. A Velocity-nél ez a következő konfigurációs név és érték párokkal egyszerűen megoldható:

```
input.encoding=ISO-8859-2
output.encoding=ISO-8859-2
```

`VelocityServlet` osztály használatakor használjuk a

```
default.contentType="text/html;encoding=ISO-
8859-2
```

értékpárt, míg `Velocity` osztály használatakor a `mergeTemplate` metódusok közül azt használjuk, melynél második paraméterként meg kell adni a kódolást.

A Velocity jó alternatíva lehet azon Java programozóknak, akik statikus szövegbe szeretnének dinamikus adatokat illeszteni futásidőben, illetve webes alkalmazások esetén, akiknek valamilyen okból nem megfelelő a JSP technológia, és a Java kódot külön akarják választani a statikus szövegtől.

Természetesen a Velocity csupán alapvető template engine funkciókat lát el, viszont ennek ismerete is szükséges lehet, egy bonyolultabb, modern, valamilyen template engine-re épülő, Model-View-Controller felépítésű framework használatakor (ld. Jakarta Project-es Turbine, Jetspeed), esetleg egy hasonló fejlesztésekor.

A címben felvetett kérdésre tehát az a válasz, hogy minek írjunk, hiszen van egy egyszerűen használható, jól átgondolt, szabad forrású megoldás, mely szinte minden igényt kielégít, de ha nekünk több kell, akár tovább is fejleszthetjük.

Budapest, 2002. szeptember 16.

Hivatkozások

The Apache Software Foundation http://www.apache.org

The Jakarta Project http://jakarta.apache.org

Velocity Template Engine http://jakarta.apache.org/velocity

Szerzőről

Viczián István a Debreceni Egyetem programtervező matematikus szakán végzett 2001 nyarán, ahol most levelező PhD. hallgató az elosztott rendszerek, middleware-ek témakörében. Jelenleg szoftver fejlesztőként dolgozik Budapesten, melynek keretében middleware szoftverekkel, alkalmazásintegrációval, webes technológiákkal valamint application mining-gal és reverse enginering-gel foglalkozik. Szabadidejében internetes portált fejleszt, és Java technológiákkal ismerkedik, valamint Java blog-ot ír (JTechLog).

E-mail: viczus@freemail.hu

Honlap: http://dragon.unideb.hu/~vicziani