LISE 3

Template pour les mails

Version 1.0 du 13/04/2010

Etat : Validé

SUIVI DES MODIFICATIONS

Version	Rédaction	Description	Vérification	Date
1.0	A. Lesuffleur	création du document		13/04/10
		Document validé dans sa version xxx		

LISTE DE DIFFUSION

Organisation	Nom	Info	Commentaire	Validation
		☐	☐	☐
		☐	☐	☐
		☐	☐	☐

SOMMAIRE

DOCUMENTS DE REFERENCE

Version	Titre

1 PRESENTATION

1.1 OBJECTIF

Les Template de mail permettent de modifier le contenu d'un courriel en incorporant des données métiers sans avoir à modifier le code de l'application.

Avantages :

- A partir du moment où un champ est dans le modèle métier, il peut être affiché dans le Mail via la Template
- Pas d'impact lors d'évolution : si un champ est ajouté, il sera automatiquement disponible dans les modèles
- Possibilité de générer du HTML ou du texte brut ou autre format
- Ne nécessite pas de nouvelles librairies (à partir de LISE 3.2.0)

1.2 DESCRIPTION

La solution repose sur l'intégration du moteur de Template JAVA Velocity. Ce moteur prend en entrée un fichier Template VTL (Velocity Template Language) et un ensemble de variable en JAVA (Contexte).

Un fichier Template VTL permet de définir la mise en page et le contenu statique désirés.

Le contexte permet lors de chaque génération de prendre en compte des données métiers différentes.

Le moteur de Template Velocity récupère le fichier VTL grâce à un « resource loader » et fusionne le fichier VTL avec les données métiers (objets JAVA).

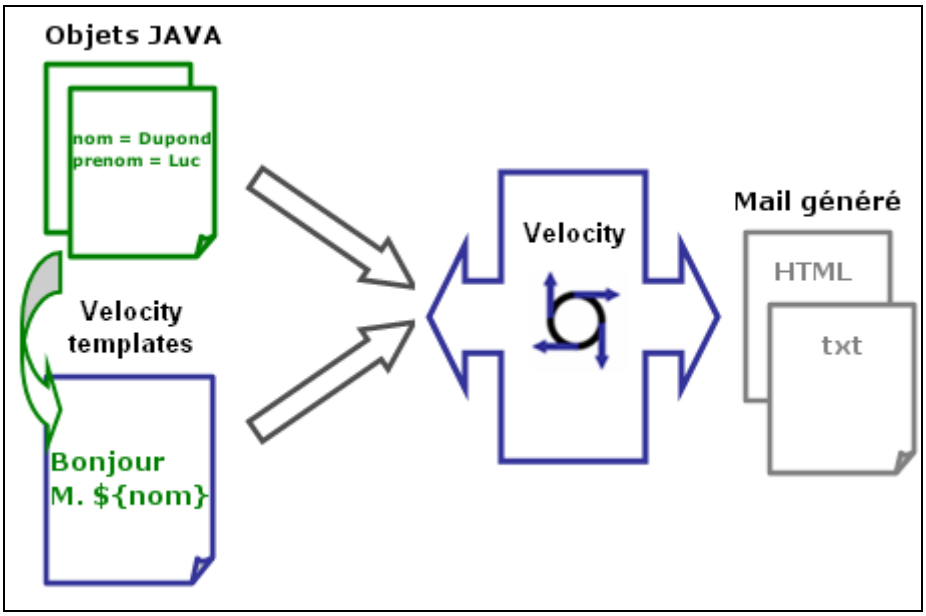

Schéma de fonctionnement de Velocity

1.3 NORMES ACUBE-VELOCITY

- Le moteur de Template Velocity est réservé à la génération de mails personnalisés.
- Le moteur Velocity ne doit pas être utilisé pour un autre but : générateur de code, remplacement des JSP ou feuilles XSL.
- Le serveur de mail doit être déclaré dans le fichier context.xml comme une ressource JEE JNDI, les propriétés doivent être tagguées pour la livraison avec les tâches ANT.
- Dans la documentation du projet, un fichier référencera toutes les données métiers disponibles pour les Template

  Exemple :

Code	Description
${personne.nom}	Nom du destinataire
${personne.prenom}	Prénom du destinataire
${personne.adresse.codePostal}	Code postal de l'adresse du destinataire
${personne.adresse.ville}	Ville de l'adresse du destinataire
${correspondant}	Prénom + nom du correspondant

2 MISE EN ŒUVRE

2.1 PRE REQUIS :

Pour mettre en place le moteur de template, la bibliothèque Velocity est requise ainsi que les dépendances ci-dessous :

Dépendance	Requis	Condition
commons-collections	Oui	Nécessaire au fonctionnement de Velocity
commons-lang	Oui	Nécessaire au fonctionnement de Velocity
commons-logging	Non	Seulement si les logs sont configurés pour utiliser le logger CommonsLogLogChute .
oro	Non	Seulement si les instructions VTL #include et #parse sont utilisées.
log4j	Non	Seulement si log4j est utilisé comme logger.

Ces bibliothèques sont incluses dans le Framework LISE V3 à partir de la version 3.2.0.

2.2 CONFIGURATION

2.2.1 CONFIGURATION SPRING

2.2.1.1 CONFIGURATION MINIMALE

Créer un fichier nommé « spring-appcontext-mail.xml » avec au minimum le contenu suivant :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Configuration de Spring Mail -->
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="
            http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
            http://www.springframework.org/schema/jee
http://www.springframework.org/schema/jee/spring-jee-2.5.xsd
            http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

<!-- Connexion à la ressource JNDI -->
<bean id="mailSession" class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName"><value>java:comp/env/mail/Session</value></property>
</bean>
```

```
<!-- Déclaration du bean JavaMail qui envoie les mails  -->
<bean id="mailSender"
class="org.springframework.mail.javamail.JavaMailSenderImpl">
    <property name="session"><ref bean="mailSession"/></property>
</bean>

<!-- Instanciation du moteur de template avec un ressource loader de type
classpath  -->
<bean id="velocityEngine"
    class="org.springframework.ui.velocity.VelocityEngineFactoryBean">
    <property name="velocityProperties">
        <props>
            <prop key="resource.loader">class</prop>
            <prop key="class.resource.loader.class">
            org.apache.velocity.runtime.resource.loader.ClasspathResourceLoader
            </prop>
        </props>
    </property>
</bean>

<!-- Instanciation du service applicatif contenant l'algorithme d'envoi des mails
-->
<bean id="mailService" class="acube.projet.business.service.MailServiceImpl">
    <property name="mailSender" ref="mailSender" />
    <property name="velocityEngine" ref="velocityEngine" />
</bean>

</beans>
```

Avec cette configuration ci-dessus, le projet disposera d'un service « MailService » contenant :

- un moteur de Template Velocity instancié avec un chargeur de ressources capable de récupérer les fichiers VTL dans le classpath JAVA.
- un objet MailSender, contenant les classes JavaMail instanciées avec les paramètres JNDI (contenu dans le fichier context.xml).

2.2.1.2 CONFIGURATION DU LOADER

Un « ressource loader » permet de récupérer un fichier VTL à partir d'une clé, ou un chemin.
Ci-dessous la liste des loaders disponibles avec leurs paramètres :

Loader	spécificité	paramètres
ClasspathResourceLoader	chargement des fichiers VTL dans le classpath	-file.resource.loader.cache = true/false -file.resource.loader.modificationCheckInterval = Intervalle en seconde pour vérifier les modifications
DataSourceResourceLoader	chargement des fichiers VTL dans une table d'une base de données	-ds.resource.loader.resource.datasource = datasource JNDI -ds.resource.loader.resource.table = nom de la table -ds.resource.loader.resource.keycolumn = nom de la colonne contenant la clé -ds.resource.loader.resource.templatecolumn = nom de la colonne contenant le VTL -ds.resource.loader.resource.timestampcolumn = nom de la colonne contenant la date du VTL

		-ds.resource.loader.cache = false/true -ds.resource.loader.modificationCheckInterval = intervalle en seconde
FileResourceLoader	chargement des fichiers VTL sur le système de fichier local	-file.resource.loader.path = chemin vers les fichiers VTL -file.resource.loader.cache = true/false -file.resource.loader.modificationCheckInterval = Interval en seconde pour vérifier les modifications
JarResourceLoader	chargement des fichiers VTL contenu dans une archive JAR	-jar.resource.loader.path= chemin du jar contenant les VTL jar:file:/test2.jar
StringResourceLoader	chargement d'une Template VTL à partir d'une clé et d'une chaine de caractère au format VTL en paramètre	
URLResourceLoader		-url.resource.loader.root = URL vers les fichiers VTL -url.resource.loader.cache = true/false -url.resource.loader.modificationCheckInterval = Intervalle en seconde pour vérifier les modifications

Chacun de ces loaders peut être déclarés dans le fichier Spring.

Exemple de configuration Spring pour DataSourceResourceLoader

```
<!-- Déclaration du dataSourceResourceLoader avec la datasource JNDI (SQL)  -->
<bean id="templateLoader"
    class="org.apache.velocity.runtime.resource.loader.DataSourceResourceLoader">
    <property name="dataSource">
        <ref local="dataSource" />
    </property>
</bean>

<!-- Instanciation du moteur avec les paramètres nécessaires au loader -->
<bean id="velocityEngineDS"
    class="org.springframework.ui.velocity.VelocityEngineFactoryBean">
    <property name="velocityPropertiesMap">
        <map>
            <entry key="resource.loader">
                <value>ds</value>
            </entry>
            <entry key="ds.resource.loader.instance">
                <ref bean="templateLoader" />
            </entry>
            <entry key="ds.resource.loader.resource.table">
                <value>template</value>
            </entry>
            <entry key="ds.resource.loader.resource.keycolumn">
                <value>name</value>
            </entry>
            <entry key="ds.resource.loader.resource.templatecolumn">
                <value>content</value>
            </entry>
```

```
            <entry key="ds.resource.loader.resource.timestampcolumn">
                <value>last_modified</value>
            </entry>
        </map>
    </property>
</bean>
```

2.2.2 IMPLEMENTATION JAVA

Coté Java, il faut créer le service qui doit envoyer des mails.
Ci-dessous l'interface correspondant à la configuration Spring avec une méthode métier « sendAlerteMail » :

```
public interface MailService {

    /**
     * @param velocityEngine
     *            the velocityEngine to set
     */
    public abstract void setVelocityEngine(
            VelocityEngine velocityEngine);

    /**
     * @param mailSender
     *            the mailSender to set
     */
    public abstract void setMailSender(
            MailSender mailSender);

    /**
     * Envoi du mail
     */
    public abstract void sendAlerteMail();

}
```

Pour l'implémentation, la seule méthode a exécuté pour fusionner la template VTL avec les données métiers est :

```
String VelocityEngineUtils.mergeTemplateIntoString (VelocityEngine, String VTL,Map<String,Object> model);
```

Cette méthode prend trois paramètres :

- Le moteur de template instancié par Spring,
- La clé permettant d'identifier le fichier VTL à récupérer,
- Une Map<String,Object> contenant les objets JAVA métier

Exemple d'implémentation de l'interface ci-dessus :

```
public class MailServiceImpl implements MailService {

    /**
     * Template engine
     */
    private VelocityEngine velocityEngine;
```

```
    /**
     * mailSender
     */
    private MailSender mailSender;

    /**
     *
     */
    public MailServiceImpl() {

    }

    /**
     * @param velocityEngine
     *            VelocityEngine
     * @param mailSender
     *            MailSender
     */
    public MailServiceImpl(
            VelocityEngine velocityEngine, MailSender mailSender) {

        this.velocityEngine = velocityEngine;
        this.mailSender = mailSender;
    }

    /**
     * @param velocityEngine
     *            the velocityEngine to set
     */
    public void setVelocityEngine(
            VelocityEngine velocityEngine) {

        this.velocityEngine = velocityEngine;
    }

    /**
     * @param mailSender
     *            the mailSender to set
     */
    public void setMailSender(
            MailSender mailSender) {

        this.mailSender = mailSender;
    }



    /**
     * Envoi du mail
     */
    public void sendAlerteMail() {

        //Model
        Map<String, Object> model = new HashMap<String, Object>();
        model.put("user", "Mon Utilisateur");
        model.put("destinataire", "M. Destin");
```

```
        //Transform
        String text = VelocityEngineUtils.mergeTemplateIntoString(
            this.velocityEngine, "mailTemplate.vm", model);

        //Send mail
        SimpleMailMessage msg = new SimpleMailMessage();
        String tos[] =
                {"M. Destin<m.destin@maee.com>",
                        "D2 <d2@maee.com>"};

        msg.setTo(tos);
        msg.setFrom("Expediteur" + "<Expedit@acube.com>");
        msg.setText(text);

        this.mailSender.send(msg);

    }

}
```

2.2.3 TEMPLATE VTL

2.2.3.1 SYNTAXE

Une Template VTL est un fichier texte sans en-tête particulier, avec l'extension .vm.

Pour faire référence à une variable, il faut utilise la syntaxe suivante :

```
${cléVariable.sousClé}
```

Pour faire référence à une méthode d'une variable, il faut utilise la syntaxe suivante :

```
${cléVariable.nomMéthode()}
```

Toutes les autres instructions utiliser le caractère #, exemple :

```
 #foreach( $criterion in $criteria )

    #set( $result = $query.criteria($criterion) )

    #if( $result )
        Query was successful
    #end

#end
```

Pour en savoir plus sur la syntaxe, consulter le guide d'utilisation disponible en français sur le site de Velocity : http://velocity.apache.org/engine/releases/velocity-1.6.2/translations/user-guide_fr.html

2.2.3.2 EXEMPLE HTML DE TEMPLATE

```
<html>
<body>
<h3>Bonjour ${user.userName},et bienvenue sur le site ${site}!</h3>
<div>
   L'application utilisera désormais l'adresse mail suivantes<a
href="mailto:${user.emailAddress}">${user.emailAddress}</a>.
</div>
</body>
</html>
```

3 PERSONNALISATION

3.1 CREER SON RESOURCELOADER

Il est possible de créer son propre Resource Loader. Pour se faire, il suffit d'implémenter la classe abstraite ResourceLoader.

Les méthodes à surcharger sont les suivantes :

Signatures des méthodes	Description
`public void init(ExtendedProperties configuration);`	Méthode d'initialisation permettant de récupérer des paramètres propres au loader
`public InputStream getResourceStream (String source )`	Méthode qui va récupérer le fichier VTL selon le paramètre source
`public boolean isSourceModified (Resource resource)`	Méthode permettant de savoir si une source a été modifiée, (nécessaire pour la gestion d'un cache)
`public long getLastModified (Resource resource)`	Méthode permettant de savoir depuis quand la ressource a été modifiée

Exemple d'un resourceLoader basique qui charge une seule template VTL sous forme de chaine de caractères, sans gérer de cache :

```
public class OneStringResourceLoader extends ResourceLoader {

    @Override
    public long getLastModified(
            Resource arg0) {
        return 0;
    }

    @Override
    public InputStream getResourceStream(
            String arg0) throws ResourceNotFoundException {

        return new ByteArrayInputStream(arg0.getBytes());
    }

    @Override
    public void init(
            ExtendedProperties arg0) {
    }

    @Override
    public boolean isSourceModified(
            Resource arg0) {
        return false;
    }
}
```

4 PERFORMANCE

4.1 CACHE

Certains Resource Loader permettent d'activer un gestionnaire de cache. Ce cache peut améliorer les performances lors de traitement de masse faisant appel à la même template un nombre de fois important.

Lors de l'utilisation du cache, il faut veiller à bien paramétrer l'intervalle de synchronisation du cache (paramètre : resource.loader.cache)

4.2 FONCTIONNEMENT ET PERFORMANCE

- Les Template VTL sont interprétées, il n'y a pas de phase de compilation, la vitesse d'exécuter est linéaire.
- Le moteur pour accéder aux variables utilise les getters n fois le nombre d'occurrences trouvé dans le fichier VTL.
  Pour éviter de dégrader les performances, il est préférable de rendre disponibles aux Template des objets métiers de type POJO sans algorithmes lourd et éviter les accès aux ressources extérieures (bases de données, web services …).