

Jakarta Commons Digester

par [Sébastien Le Ray](#)

Date de publication : 28/09/2005

Dernière mise à jour : 28/09/2005

Dans ce tutoriel, nous allons voir comment utiliser l'API Jakarta Commons Digester qui permet de faire un mapping XML/Objet de façon relativement simple.

- I - Introduction
- II - Principes Généraux
 - II.A - Organisation
 - II.B - Règles
- III - Utilisation des Digesters
 - III.A - Ajout des règles par programmation
 - III.B - Définition des règles dans un fichier XML
 - III.C - Lancement de l'analyse
- IV - Règles Standards
 - IV.A - ObjectCreateRule
 - IV.B - SetNextRule, SetRootRule & SetTopRule
 - IV.C - FactoryCreateRule
 - IV.D - SetPropertyRule
 - IV.E - SetPropertyRule
 - IV.F - CallMethodRule, CallParamRule & ObjectParamRule
 - IV.G - BeanPropertySetterRule & SetNestedPropertiesRule
- V - Jeux de règles
- VI - Validation du document
- VII - Conclusion
- VIII - Remerciements
- IX - Téléchargements

I - Introduction

L'API commons digester, initialement partie intégrante du framework Struts, a pour but de créer des objets à partir de données contenues dans des fichiers XML, c'est ce que l'on appelle plus communément un *mapping* XML/Objet. Les commons digester vous offrent la possibilité de créer des objets à la volée lorsqu'un certain enchaînement de balises est détecté ou encore d'appeler des méthodes spécifiques sur ces objets en leur passant des paramètres issus du fichier XML.

A l'origine, les commons digester n'étaient paramétrables que par la programmation, il fallait ajouter les règles une par une au moyen d'appels de méthodes Java, à partir de la version 1.2, les règles peuvent être spécifiées au sein d'un fichier XML (qui est lu par les commons digester en utilisant... un *Digester*), ce qui offre un plus grande souplesse et une configuration non intrusive.

Dans cet article, basé sur la version 1.7, nous présenterons les principes de fonctionnement généraux de l'API ainsi que la plupart des règles.

II - Principes Généraux

II.A - Organisation

Comme le suggère la structure d'un document XML, les commons digester fonctionnent selon une logique arborescente. Le résultat de l'analyse d'un fichier est un élément racine à partir duquel on doit pouvoir accéder aux autres. Le raccordement des objets entre eux et à l'objet racine est à la charge de l'utilisateur. Comme nous le verrons, il est possible de demander au *Digester* de créer des objets ; ceux-ci sont temporairement placés sur une pile (la *pile d'objets*) afin de pouvoir être utilisés mais à l'issue de l'analyse du fichier XML, seul l'élément racine est accessible (à moins d'avoir empilé manuellement d'autres objets au préalable).

Il faut donc vous arranger pour pouvoir récupérer tous les objets à partir de l'objet racine, celui-ci peut tout simplement être une collection à laquelle on ajoute les différents éléments au fur et à mesure de leur création.

II.B - Règles

La façon dont va être interprété le fichier XML est spécifié par un ensemble de règles. Une règle est une combinaison d'un pattern (voir ci-dessous) et d'un type de règle. La création de règles personnalisées dépasse le cadre de cet article et ne sera donc pas abordée ici. Nous nous contenterons de décrire les règles basiques fournies avec les commons digester qui offrent déjà une grande variété de possibilités. Il faut garder à l'esprit que les règles sont appelées dans l'ordre où elles ont été ajoutées au *Digester* sur les ouvertures de balises et dans l'ordre inverse sur les fermetures, le fonctionnement de certaines règles repose sur ce comportement.

Lors du paramétrage d'une règle quelle qu'elle soit, il est nécessaire d'indiquer sur quel pattern elle doit être déclenchée. La définition d'un pattern est relativement simple, il s'agit du chemin pour accéder à l'élément. En supposant que l'on ait un fichier XML de la forme suivante :

```
<?xml version="1.0"?>
  <racine>
    <premier-niveau>
      <data/>
      <data/>
    </premier-niveau>
  </racine>
```

La définition d'une règle devant se déclencher pour les deux premiers **data** devra se faire en indiquant le pattern **racine/premier-niveau/data**. Si vous souhaitez que la règle soit déclenchée quel que soit l'emplacement de **data**, vous pouvez indiquer ***/data**. Dans le cas où un pattern générique et un pattern particulier correspondent tous les deux à un chemin, c'est la règle correspondant au pattern particulier qui est utilisée (principe du *best matching pattern*). En reprenant l'exemple ci-dessus, si l'on définit une règle **R1** associée au pattern **racine/premier-niveau/data** et une règle **R2** associée au pattern ***/data**, c'est **R1** qui sera utilisée pour les deux premiers **data** et **R2** pour le second.

Par défaut, les commons digester supportent les deux types de règles cités ci-dessus. Il est possible d'étendre les possibilités en appelant la méthode **setRules** du *Digester* et en lui passant une instance de **ExtendedBaseRules** :

```
Digester digester = new Digester();
digester.setRules(new ExtendedBaseRules());
```

L'utilisation de telles règles donne accès à plus de fonctionnalités :

- Si le pattern est préfixé par **!**, le principe du *best matching pattern* ne s'applique plus, les règles

correspondant au pattern exact sont appliquées mais également celles correspondant au pattern générique, ce qui peut être souhaitable dans certain cas ;

- Les patterns génériques offrent plus de possibilités, il est possible de spécifier une règle pour tous les enfants directs de certains éléments *via* le caractère ? (***/premier-niveau/?**) ou pour tous les descendants (directs ou non) de certains éléments (**racine/premier-niveau/***).

Les types de règles disponibles sont présentés dans la section Règles Standards

III - Utilisation des Digesters

III.A - Ajout des règles par programmation

L'ajout des règles par programmation est apparue la première, elle est peu souple mais évite que l'utilisateur de l'application n'aille modifier un fichier de configuration par erreur. L'ajout des règles se fait en deux étapes :

```
// 1) Déclaration du Digester
Digester digester = new Digester();
// 2) Ajout des règles
// Directement
digester.addRule("racine", new ObjectCreateRule(Root.class));
// Ou en utilisant les méthodes de raccourci pour les règles standards
digester.addObjectCreate("racine/premier-niveau", Child.class);
```

Comme on le voit dans cet exemple, le *Digester* dispose de méthodes de raccourci pour l'ajout des règles standards. Les différents types de règles fournis avec les commons digesters sont décrits dans la section Règles standards.

III.B - Définition des règles dans un fichier XML

La configuration en utilisant un fichier XML permet d'utiliser de façon simple les règles standards mais également des règles personnalisées. Chaque règle standard a sa propre balise. Pour les cas simples, vous pouvez spécifier le pattern en entier pour chaque règle comme dans l'exemple suivant :

```
<?xml version="1.0"?>
<digester-rules>
  <object-create-rule pattern="racine" classname="Root"/>
  <object-create-rule pattern="racine/premier-niveau" classname="Child"/>
  <set-next-rule pattern="racine/premier-niveau" methodname="addChild"/>
</digester-rules>
```

Cette configuration crée un objet **Root** lorsqu'elle rencontre la racine et lui ajoute des objets **Child** qu'elle crée au fur et à mesure lorsqu'elle rencontre des balises **premier-niveau**.

Lorsque le nombre de règles augmente, il peut être utile de structurer le document de façon plus forte, pour cela, on peut utiliser la balise **pattern** :

```
<?xml version="1.0"?>
<digester-rules>
  <pattern value="racine">
    <object-create-rule classname="Root"/>
    <pattern value="premier-niveau">
      <object-create-rule classname="Child"/>
      <set-next-rule methodname="addChild"/>
    </pattern>
  </pattern>
</digester-rules>
```

Chaque pattern est concaténé à ceux de niveaux supérieurs, ainsi, les règles de création et d'ajout des enfants seront déclenchées sur le pattern **racine/premier-niveau**.

L'utilisation de règles personnalisées peut se faire en modifiant la DTD ainsi que les mécanismes de chargement du fichier. Un moyen plus pratique peut être d'utiliser la balise **include**. Celle-ci dispose d'un attribut **class** qui désigne une classe implémentant **org.apache.commons.digester.xmlrules.DigesterRulesSource**. Cette interface définit une seule méthode, **getRules(Digester)** qui doit ajouter ses propres règles au *Digester* passé en

paramètre. Lors de l'ajout de ces règles, il faut bien avoir en tête que si l'include est effectué au sein d'une balise **pattern**, celui-ci est ajouté à tous les patterns précisés dans le *DigesterRulesSource*. Ainsi, si l'on a un include au sein d'un pattern **racine** et que l'on ajoute des règles avec le pattern **premier-niveau** au sein du *DigesterRulesSource*, le pattern de ces règles sera en réalité **racine/premier-niveau**.

Le chargement du fichier de configuration peut se faire de deux façons, une version simple :

```
Digester digester = DigesterLoader.createDigester(getClass().getResource("/rules.xml"));
```

où **rules.xml** est le fichier XML décrivant les règles. Une version plus complexe mais permettant de bénéficier des règles étendues en effectuant les initialisations appropriées est également disponible :

```
Digester digester = new Digester();  
// Cette ligne peut être supprimée si l'on n'utilise pas de règles étendues  
digester.setRules(new ExtendedBaseRules());  
digester.addRuleSet(new FromXmlRuleSet(getClass().getResource("/rules.xml")));
```

III.C - Lancement de l'analyse

Une fois le *Digester* correctement configuré, il faut lancer l'analyse du fichier XML, ceci se fait au moyen de la méthode **parse** :

```
root = (Root)digester.parse(getClass().getResourceAsStream("/") + FILE_NAME));
```

Une exception est levée si un problème survient durant l'analyse du fichier. La méthode **parse** renvoie l'élément racine. Lorsqu'elle se termine, la pile d'objets est vide, il est donc nécessaire d'avoir combiné les éléments de la façon voulue au moyen des règles.

IV - Règles Standards

Pour chaque règle, le nom de la classe est donné, suivi de la méthode raccourci définie au niveau du digester et de la balise correspondante pour une configuration XML.

IV.A - ObjectCreateRule

- Classe : org.apache.commons.digester.ObjectCreateRule
- Méthode : addObjectCreate
- Balise : object-create-rule

La règle **ObjectCreateRule** est une des plus utiles. Comme son nom l'indique, elle va créer un objet puis le placer sur la pile d'objets lorsqu'elle rencontre la balise ouvrante de son pattern. Lorsque la balise fermante est rencontrée, l'objet du haut de la pile, qui correspond normalement à celui créé, est dépilé et donc perdu s'il n'a pas été rattaché à un autre. Ce comportement permet de créer un objet, de le rattacher au précédent puis de traiter les balises filles afin de définir des propriétés de cet objet. Notez que cette règle utilise le constructeur vide de la classe à créer.

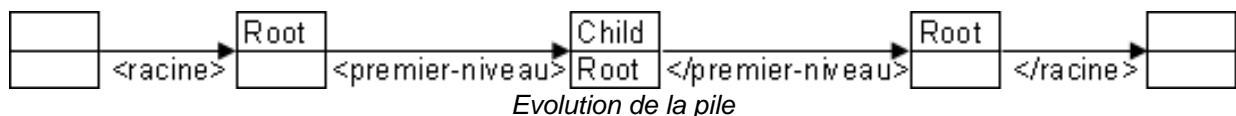
Considérons l'ensemble de règles suivant :

```
Digester d = new Digester();
d.addObjectCreate("racine", Root.class);
d.addObjectCreate("racine/premier-niveau", Child.class);
// Cette règle est expliquée plus tard
d.addSetNext("racine/premier-niveau", "addChild");
Root r = d.parse(getClass().getResourceAsStream("/test.xml"));
```

Il demande au *Digester* de créer un objet **Root** (la classe serait indiquée par l'attribut **type** pour une configuration XML) lorsqu'il rencontre la balise **racine**. On suppose, comme son nom l'indique, que cette balise est la racine du fichier XML et qu'elle n'apparaît donc qu'une fois, au début. L'objet ainsi créé sera donc renvoyé par la méthode **parse**. La seconde règle demande à créer un objet **Child** lorsque la balise **premier-niveau** est rencontrée au sein de **racine**, l'objet est mis sur la pile puis traité. La dernière règle prend l'objet au sommet de la pile et le passe en argument à une méthode de l'objet "en dessous" (ici la racine).

Voici l'évolution de la pile lors de l'analyse du fichier suivant :

```
<racine>
  <!-- Note : La balise premier-niveau est écrite sous cette
        forme pour une question de facilité d'explication -->
  <premier-niveau></premier-niveau>
</racine>
```



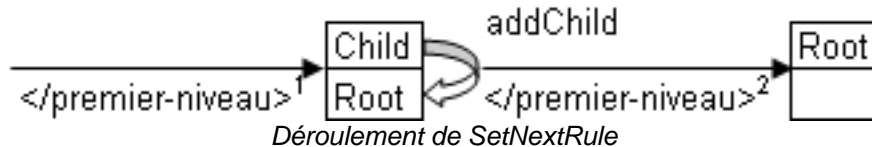
Les balises **premier-niveau** auraient pu s'écrire **<premier-niveau/>** cela n'aurait rien changé.

IV.B - SetNextRule, SetRootRule & SetTopRule

- Classe : org.apache.commons.digester.SetNextRule
- Méthode : addSetNext
- Balise : set-next-rule

Cette règle est également très fréquemment utilisée en collaboration avec **ObjectCreateRule** (voir exemple précédent). Elle permet d'appeler une méthode de l'objet suivant celui du haut de la pile en lui passant en paramètre l'objet du haut de pile (on appelle donc une méthode de l'objet parent en lui passant l'objet enfant). La méthode à appeler est spécifiée dans le constructeur (ou *via* l'attribut **methodname** en XML). Elle est déclenchée sur la balise fermante afin que l'objet au sommet de la pile soit totalement initialisé lorsque la méthode est appelée.

Si l'on reprend l'exemple d'ObjectCreateRule on peut y rajouter la séquence suivante :



Dans la mesure où **SetNextRule** a été ajoutée après **ObjectCreateRule** et que les règles sont appelées dans l'ordre inverse de leur ajout sur les balises fermantes, on a bien l'appel de méthode sur l'objet parent qui est effectué avant le dépileage de l'objet enfant ce qui est le comportement attendu.

La règle **SetRootRule** est similaire à **SetNextRule** mais la méthode est appelée sur l'objet *racine* de la pile et non pas sur le second objet. **SetTopRule** quant à elle effectue l'inverse de **SetNextRule** puisqu'elle appelle une méthode sur l'objet du sommet de la pile en lui passant l'objet suivant.

IV.C - FactoryCreateRule

- Classe : org.apache.commons.digester.FactoryCreateRule
- Méthode : addFactoryCreate
- Balise : factory-create-rule

Cette règle est utilisée d'une façon similaire à **CreateObjectRule** mais permet d'invoquer des constructeurs disposant de paramètres. Les paramètres doivent être obtenus *via* des attributs de la balise pour laquelle est définie la règle.

Afin de pouvoir récupérer ces attributs et de les passer au constructeur de l'objet, vous devez créer une *Factory* qui implémente l'interface **org.apache.commons.digester.ObjectCreationFactory** (vous pouvez étendre la classe **org.apache.commons.digester.AbstractObjectCreationFactory** qui définit déjà les méthodes de base). La méthode centrale de celle-ci est **objectCreate(Attributes)**, c'est elle qui renvoie le nouvel objet, en voici un exemple :

```
public Object createObject(Attributes attributes) throws Exception {
    String name = attributes.getValue("name");
    if(name == null) {
        return new Child();
    }
    return new Child(name);
}
```

La *Factory* contenant cette méthode crée une nouvelle instance de la classe **Child** en lui donnant un nom si l'attribut **name** est présent. En supposant que la règle est rattachée au même pattern que dans l'exemple de **ObjectCreateRule**, la rencontre de la balise

```
<premier-niveau name="UnEnfant">
```

crée une nouvelle instance de la classe **Child** avec le nom **UnEnfant** et l'ajoute à la pile des objets ce qui la rend éventuellement disponible pour d'autres règles.

Lors de la définition de la règle, il est nécessaire de préciser au moins le nom de la classe *Factory* (via le constructeur ou par le biais de l'attribut **classname** en XML). Vous pouvez également préciser le nom d'un attribut qui, s'il est présent, indique la classe de la *Factory* à utiliser à la place de celle passée au constructeur (le nom de l'attribut est spécifié dans le constructeur ou au moyen de l'attribut **attrname** en XML).

IV.D - SetPropertyRule

- Classe : org.apache.commons.digester.SetPropertyRule
- Méthode : addSetProperty
- Balise : set-property-rule

Cette règle positionne une propriété (via une méthode **setXXX**) sur l'objet du haut de la pile (généralement créé par une **ObjectCreateRule** ayant le même pattern que la **SetPropertyRule**). Le nom et la valeur de la propriété à positionner sont précisés dans des attributs de la balise pour laquelle est déclenchée la règle.

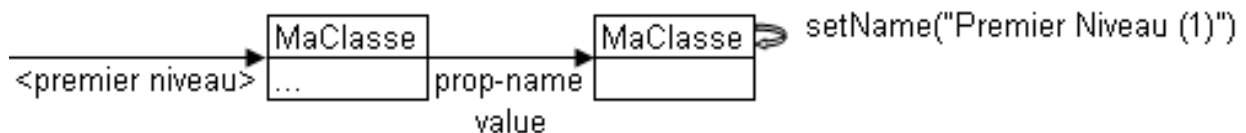
Prenons la séquence d'ajout de règles suivante :

```
Digester digester = new Digester();
// Règles omises pour concision
d.addObjectCreate("racine/premier-niveau", MaClasse.class);
d.addSetProperty("racine/premier-niveau", "prop-name", "value");
```

Elle demande la création d'un objet de type **MaClasse** lorsque la balise **premier-niveau** est rencontrée puis le positionnement de la propriété dont le nom est donné dans l'attribut **prop-name** et la valeur dans l'attribut **value**. Ainsi, lors de l'analyse du fichier XML suivant :

```
<racine>
  <premier-niveau prop-name="name" value="Premier Niveau (1)"/>
</racine>
```

La séquence d'événements aura l'aspect suivant :



Déclenchement de la règle SetPropertyRule

IV.E - SetPropertyRule

- Classe : org.apache.commons.digester.SetPropertiesRule
- Méthode : addSetProperties
- Balise : set-properties-rule

Cette règle permet de considérer les attributs d'une balise comme étant des propriétés de l'objet situé au sommet de la pile (généralement créé par une **ObjectCreateRule** ayant le même pattern). Elle prend chaque attribut de la balise et appelle la méthode **setNomAttribut** sur l'objet du sommet de la pile. Les attributs ne correspondant à aucune propriété sont ignorés par défaut.

En supposant que l'on définisse les règles suivantes :

```
Digester digester = new Digester();
```

```
// Règles omises pour concision
d.addObjectCreate("racine/premier-niveau", MaClasse.class);
d.addSetProperties("racine/premier-niveau");
```

Avec ce fichier XML :

```
<racine>
  <premier-niveau name="Premier Niveau (1)" priority="low"/>
</racine>
```

Une fois l'objet créé, les méthodes **setName("Premier Niveau (1)")** et **setPriority("low")** seront appelées.

Il est possible de faire en sorte que les noms d'attributs ne soient pas utilisés directement comme noms de propriétés, pour cela, il faut passer deux tableaux au constructeur, le premier indique les noms d'attributs, le second les propriétés correspondantes (dans le même ordre). Les attributs non déclarés dans ce tableau sont mappés directement. Si vous utilisez un fichier XML pour définir les règles, il faudra passer par la balise **alias** :

```
<set-properties-rule pattern="racine/premier-niveau">
  <alias attr-name="name" prop-name="nom">
</set-properties-rule>
```

Si l'on avait ce genre de définition dans notre exemple, les méthodes **setNom** et **setPriority** seront appelées.

IV.F - CallMethodRule, CallParamRule & ObjectParamRule

- Classe : org.apache.commons.digester.CallMethodRule
- Méthode : addCallMethod
- Balise : call-method-rule

Utilisée seule, la règle **CallMethodRule** présente assez peu d'intérêt, elle se contente d'appeler une méthode sur l'objet situé au sommet de la pile. Le nom de la méthode est passé en paramètre au constructeur ou *via* l'attribut **methodName** pour une configuration XML. Notez que l'appel de la méthode est effectué lorsque la balise fermante du pattern est rencontrée ce qui permet de mettre en place des paramètres (voir ci-dessous).

- Classe : org.apache.commons.digester.CallParamRule
- Méthode : addCallParam
- Balise : call-param-rule

CallMethodRule offre plus d'intérêt lorsqu'elle est combinée à **CallParamRule**. Cette dernière va placer des paramètres sur la pile *des paramètres* qui est distincte de celle où sont empilés les objets créés au fur et à mesure de l'analyse. Ces paramètres peuvent provenir d'attributs de l'élément du pattern, de son contenu ou d'objets présents sur la pile d'objets. Lorsque la balise fermante du pattern de la **CallMethodRule** est rencontrée, les paramètres nécessaires sont dépilés et passés à la méthode spécifiée.

Nous ne donnons un exemple ici que pour la forme plus complexe de la règle. Voici l'ensemble de règles utilisé :

```
d.addObjectCreate("racine", Root.class);
d.addCallMethod("racine/premier-niveau", "afficher", 2, new Class[]{Child.class, String.class});
// Création d'un enfant
d.addObjectCreate("racine/premier-niveau", Child.class);
// Définition des propriétés (pour affichage)
d.addSetProperties("racine/premier-niveau");
d.addCallParam("racine/premier-niveau", 0, true);
d.addCallParam("racine/premier-niveau", 1, "message");
```

L'ordre de déclaration des règles est très important. Tout d'abord, on déclare une règle qui crée un objet **Root** lorsque la balise **racine** est rencontrée. La **CallMethodRule** étant déclarée en premier, elle sera exécutée en dernier pour le pattern **racine/premier-niveau** (puisqu'elle s'exécute sur la balise fermante). C'est le comportement attendu puisqu'il faut que l'on puisse créer tous les paramètres à passer à la méthode avant que celle-ci soit appelée.

Les deux dernières règles, **CallParamRule**, présentent les deux utilisations possibles de ce type de règles. Tout d'abord, on indique que le premier paramètre est l'objet situé au sommet de la pile d'objets (il est possible d'en indiquer un autre en spécifiant son indice dans la pile à la place du booléen), il s'agit donc de l'instance de **Child** qui vient juste d'être créée. Dans le second cas, on donne le nom de l'attribut du pattern qui sera transmis en paramètre. Il existe une autre forme pour cette règle, elle ne prend qu'un pattern et l'indice du paramètre dans la méthode ; dans ce cas, c'est le contenu de la balise du pattern qui est passé en paramètre.

En considérant ce fichier XML :

```
<racine>
  <premier-niveau name="Premier niveau 1" message="Message 1"/>
  <premier-niveau name="Premier niveau 2" message="Message 2"/>
</racine>
```

On aura successivement, création d'un objet **Root** et mise sur la pile d'objet. Ensuite, pour chaque balise **premier-niveau**, création d'un objet **Child** et mise sur la pile d'objets, définition de ses propriétés, mise de cet objet sur la pile des paramètres (indice 0), récupération de la valeur de l'attribut **message** de la balise **premier-niveau** et mise sur la pile des paramètres (indice 1). Sur la fermeture de la balise **premier-niveau**, suppression de l'objet au sommet de la pile (objet **Child** créé précédemment) appel de la méthode **afficher** sur l'objet de la pile (qui est désormais le **Root** créé à l'origine) qui récupère ses paramètres sur la pile des paramètres (le **Child** et le contenu de l'attribut **message**).

Lorsque vous désirez passer un objet constant défini par programmation à la méthode (par exemple, une chaîne de version), vous pouvez passer par la règle **ObjectParamRule** qui demande l'indice du paramètre et l'objet à passer. Pour une configuration XML, il faut préciser la classe de l'objet et en fournir la valeur initiale (attributs **type** et **value**). Vous pouvez conditionner le passage de ce paramètre à la présence d'un attribut dans la balise pour laquelle est déclenchée la règle (attribut **attrname** pour une configuration XML).



La DTD correspondant au fichier de définition des règles indique un attribut **param** requis pour **object-param-rule**, néanmoins, celui-ci n'est pas utilisé. Cette erreur sera corrigée dans la prochaine version des commons digester.

IV.G - BeanPropertySetterRule & SetNestedPropertiesRule

- Classe : org.apache.commons.digester.BeanPropertySetterRule
- Méthode : addBeanPropertySetter
- Balise : bean-property-setter-rule

La règle **BeanPropertySetterRule** positionne une propriété de l'objet au sommet de la pile des objets au moyen d'une méthode **setXXX**. Le nom de la propriété correspond au nom de la balise définie dans le pattern ou à celui passé au constructeur (propriété **property-name** en XML). La valeur de la propriété est le contenu de la balise.

Ainsi la règle

```
d.addBeanPropertySetter("racine/premier-niveau/name");
```

appellera la méthode **setName** de l'objet au sommet de la pile en lui passant le contenu de la balise **name**.

- Classe : org.apache.commons.digester.SetNestedPropertiesRule
- Méthode : addSetNestedProperties
- Balise : set-nested-properties-rule

La règle **setNestedPropertiesRule** agit d'une façon similaire à **BeanPropertySetterRule** mais appelle les setters appropriés pour toutes les balises filles du pattern pour lequel elle est définie. Il faut être prudent lors de son utilisation car elle prend en compte l'objet du haut de pile *courant*.

V - Jeux de règles

Il arrive fréquemment que l'on dispose d'ensembles de règles liées entre elles. Afin de simplifier l'application, il peut être souhaitable, plutôt que d'effectuer une configuration règle par règle, d'ajouter chaque ensemble de règles sans se soucier des détails. C'est le but du jeu de règles (spécifié par l'interface **org.apache.commons.digester.RuleSet**). Vous créez votre propre implémentation de **RuleSet** (ou étendez **RuleSetBase**) qui est en charge d'ajouter les règles une par une. Cet ajout se fait au sein de la méthode **addRuleInstances** en appelant les méthodes adéquates (**addRule** et autres) sur le *Digester* passé en paramètre. Au niveau de l'initialisation du *Digester*, on aura alors à la place de la série d'appels aux méthodes de type **addRule** un nombre réduit d'appels à la méthode **addRuleSet** pour chaque *RuleSet* créé. Le code ainsi produit est plus clair.

VI - Validation du document

Il peut être utile de valider le document analysé par le *Digester* vis-à-vis d'une DTD (Document Type Definition). Par défaut, les incohérences sont journalisées au cours de l'analyse mais n'interrompent pas le processus. Pour modifier ce comportement, il faut créer sa propre implémentation de l'interface **org.xml.sax.ErrorHandler**. Celle-ci définit trois méthodes qui sont appelés selon le niveau de gravité de l'erreur rencontrée (la non conformité à une DTD est de niveau erreur). Pour interrompre l'analyse, vous devez lever une **SaxException**.

VII - Conclusion

Cet article vous a fourni les bases pour pouvoir utiliser la plupart des fonctionnalités des commons digester. Vous trouverez la javadoc complète ainsi que divers exemple sur le [site du projet Apache Jakarta Commons](#).

VIII - Remerciements

Merci à Bestiol pour sa relecture.

IX - Téléchargements

- Le site des [commons digester](#) ;
- L'article au format [HTML Zippé](#) ([mirroir](#)) ;
- L'article au format [PDF](#) ([mirroir](#)).