Spring Batch par l'exemple

Par David HASSOUN, le 26.01.2012

Qu'est-ce que Spring-Batch?

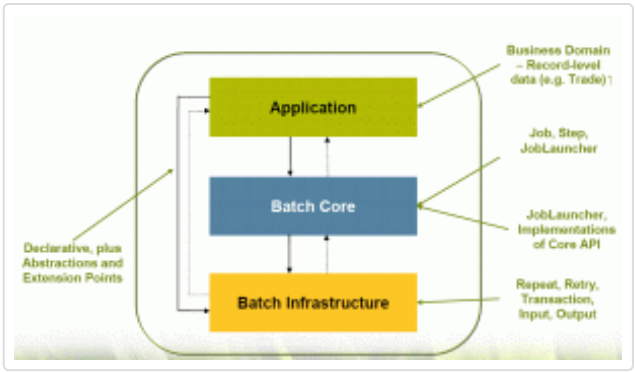
Spring-Batch est un framework développé en collaboration par SpringSource et Accenture.

Il permet de palier à des problème récurrents lors de développement de batchs:
- productivité
- gestion de gros volumes de données
- fiabilité
- reinvention de la roue .

Il s'agit donc d' un framework fournissant un outillage complet, adapté à l'écriture des batchs.

Pour rappel dans le monde informatique, un batch est un programme fonctionnant en StandAlone, réalisant un ensemble de traitements sur un volume de données.

Architecture de Spring-Batch



L'architecture de Spring-Batch est constituée en 2 couches:

- la couche Batch Core
- la couche Batch Infrastructure

La couche « Batch Core » contient une API permettant de lancer, monitorer et de gérer un batch.
Les principales interfaces et classes que contient l'API sont: Job, JobLauncher et Step.
Schématiquement un batch correspond à un job qui va être lancé via un JobLauncher.

Un job est constitué d'une ou plusieurs Step. Une Step correspond à une étape dans un batch.

La couche « Batch Infrastructure » contient un API fournissant comme principales interfaces: ItemReader, ItemProcessor et ItemWriter.

Etude de cas

Admettons que nous voulions traiter le fichier texte suivant:

```
1,DUPONT,Philippe,M
2,DUPUIS,Marie,Mme
3,DURAND,Guillaume,M
4,MARTIN,Cyril,M
5,MICHEL,Sophie,Mme
6,MERCIER,Muriel,Mme
7,VIDAL,Thomas,M
8,JACKSON,Michael,M
9,BROWN,Michele,Mme
10,WILLIAMS,Venus,Mme
```

Il s'agit d'un fichier plat contenant 10 enregistrements de personnes, dont les informations sont séparées par le caractère « , » .

Les informations sont dans l'ordre: l'id de la personne, le nom, le prénom et la civilité.

Nous voulons que notre batch enregistre les personnes en base de données uniquement si leur civilité à la valeur « M ».

Mise en place du projet

Base de données

Pour cela, je vais utiliser une base MySQL, et je vais créer une table PERSONNE ayant la structure suivante:

```
CREATE TABLE PERSONNE (ID INTEGER,

NOM VARCHAR (50),

PRENOM VARCHAR (50),

CIVILITE VARCHAR (3),

PRIMARY KEY (ID)

);
```

Récupérer Spring Batch

Pour récupérer les différentes bibliothèques, je vous conseille de passer par un projet Maven.

Avec Maven 2, il faudra récupérer les dépendances de Spring Batch de la façon suivante dans le fichier pom.xml:

```
<dependency>
```

```
<groupId>org.springframework.batch</groupId>
<artifactId>spring-batch-core</artifactId>
<version>2.1.8.RELEASE</version>
</dependency>

<dependency>
<groupId>org.springframework.batch</groupId>
<artifactId>spring-batch-infrastructure</artifactId>
<version>2.1.8.RELEASE</version>
</dependency>
```

Implémentation du batch

La classe Personne

Cette classe est un JavaBean qui va encapsuler les données d'une personne lue à partir du fichier texte.

Cette classe s'écrit de la façon suivante:

```
public class Personne {

    private int id;
    private String nom;
    private String prenom;
    private String civilite;
    public String getNom() {
    return nom;
    }
    public void setNom(String nom) {
    this.nom = nom;
    }
    public String getPrenom() {
    return prenom;
    }
    public void setPrenom(String prenom) {
    this.prenom = prenom;
    }
    public String getCivilite() {
    return civilite;
    }
    public void setCivilite(String civilite) {
    this.civilite = civilite;
    }
    public int getId() {
    return id;
    }
    public void setId(int id) {
```

```
    this.id = id;
    }
```

Définition de l'ItemReader

Spring Batch fournit des implémentation de l'interface ItemReader notement pour parser des fichier plats et des fichiers XML.

Notre reader va être déclaré dans le fichier de configuration de la façon suivante:

```
<bean id="personneReaderCSV" class="org.springframework.batch.item.file.FlatFileItemReader" >
 <property name="resource" value="input/personnes.txt" />
 <property name="lineMapper">
 <bean class="org.springframework.batch.item.file.mapping.DefaultLineMapper">
 <property name="lineTokenizer">
 <bean class="org.springframework.batch.item.file.transform.DelimitedLineTokenizer">
 <property name="delimiter" value=","/>
 <property name="names" value="id,nom,prenom,civilite" />
 </bean>
 </property>
 <property name="fieldSetMapper">
 <bean class="org.springframework.batch.item.file.mapping.BeanWrapperFieldSetMapper">
 <property name="targetType" value="ideo.springbatch.poc.Personne" />
 </bean>
 </property>
 </bean>
 </property>
 </bean>
```

Dans cette configuration nous précisons que:

- la classe FlatFileItemReader (pour lire un fichier plat) sera utilisée comme ItemReader pour lire notre fichier,

- le séparateur de champ est le caractère « , »

- chaque ligne est composée des champs: id,nom,prénom et civilité

- toutes ces informations seront stockées dans la classe Personne.

Nous avons donc définit notre reader sans rien coder! Bien sûr, il est possible de définir notre propre implémentation de l'interface ItemReader.

Définition de l'ItemProcessor

C’est dans l’ItemProcessor qu’il y aura la place pour implémenter les règles de gestion de notre batch.

Notre classe d’implémentation sera écrite de la façon suivante:

```
public class PersonProcessor implements ItemProcessor<Personne, Personne>{
 public Personne process(final Personne personneInput) throws Exception {
 Personne personneOutput = null;


 //si la civilite a la valeur M la personne sera ecrite en base sinon on la  rejette
 if ("M".equals(personneInput.getCivilite())) {
 personneOutput = new Personne();
 personneOutput.setCivilite(personneInput.getCivilite());
 personneOutput.setId(personneInput.getId());
 personneOutput.setNom(personneInput.getNom());
 personneOutput.setPrenom(personneInput.getPrenom());
 }
 return personneOutput;
 }
```

Puis dans le fichier de configuration, l’ItemProcessor sera définit de la façon suivante:

```
   <bean id="personProcessor" class="ideo.springbatch.poc.PersonProcessor" />
```

Définition de l’ItemWriter

L’ItemWriter va persister les données qui ont été traitées via l’ItemProcessor. Dans cet exemple je vais sauvegarder les données en base de données dans la table PERSONNE, en utilisant le template jdbcTemplate fournit par Spring.

Ce qui va donner dans la classe d’implémentation:

```
@Transactional (propagation=Propagation.REQUIRED, rollbackFor=Exception.class)
```

```
{Transactional (propagation Propagation.REQUIRED, rollbackFor Exception.class)
public class PersonJdbcWriter implements ItemWriter<Personne>{

 private JdbcTemplate jdbcTemplate;

 private static final String REQUEST_INSERT_PERSONNE = "insert into PERSONNE (id,nom,prenom,civilite) values (?,?,?,?)";
 private static final String REQUEST_UPDATE_PERSONNE = "update PERSONNE set nom=?, prenom=?, civilite=? where id=?";

 public void write(List<? extends Personne> items) throws Exception {
 for (Personne personne : items) {
 final Object object [] = {personne.getNom(),personne.getPrenom(), personne.getCivilite(),personne.getId()};

 //on tente un update
 int nbLigne = jdbcTemplate.update(REQUEST_UPDATE_PERSONNE, object);

 //si le nombre de ligne mise a jour vaut 0, on fait un insert
 if (nbLigne == 0) {
 final Object object2 [] = {personne.getId(),personne.getNom(),personne.getPrenom(), personne.getCivilite()};
 jdbcTemplate.update(REQUEST_INSERT_PERSONNE, object2);
 } else {

 }
 }
 }
 public JdbcTemplate getJdbcTemplate() {
 return jdbcTemplate;
 }

 public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
 this.jdbcTemplate = jdbcTemplate;
 }
}
```

}

Dans le fichier de configuration Spring, l'ItemWriter sera définit de la façon suivante:

```
<bean id="personDaoWriter" class="ideo.springbatch.poc.PersonJdbcWriter" >
 <property name="jdbcTemplate" ref="idJdbcTemplate" />
```

```
</bean>
```

Définition du job

Le job sera défini de la façon suivante:

```
<job id="importPersonnes"
 xmlns="http://www.springframework.org/schema/batch">
 <step id="readWritePersonne">
 <tasklet>
 <chunk reader="personneReaderCSV" writer="personDaoWriter" processor="personProcessor" commit-interval="2" />
 </tasklet>
 </step>
 </job>
```

Notre job sera donc constitué d'une seule Step dont la tasklet est de type « Chunk ».

Le chunk est un concept important dans Spring Batch qui définit l'enchainement qui va s'executer dans une transaction.
Le chunk aura comme propriétés l'item reader, l'item processor et l'item writer définit plus haut.
Le paramètre commit-interval du chunk définit le nombre d'items qui va être stocké en mémoire avant d'être persistés.

Pour executer notre batch il suffit de lancer le programme main suivant:

```
public class BatchPersonne {


 public static void main (String [] args) throws Exception {

 ClassPathXmlApplicationContext cpt = new ClassPathXmlApplicationContext("batch-context.xml");

 cpt.start();

 JobLauncher jobLauncher = (JobLauncher) cpt.getBean("jobLauncher");

 Job job = (Job) cpt.getBean("importPersonnes");

 JobParameters parameter = new JobParametersBuilder().addDate("date", new Date())

 .addString("input.file", "C:/envdev/travail/in/personnes.txt").toJobParameters();

 jobLauncher.run(job, parameter);

 }
```

```
}
```

Sources

Les sources complètes du projet sont disponible via l'URL suivante:http://public.ideotechnologies.com/blog/SpringBatch-POC.zip

Conclusion

Nous avons vu à travers cet exemple peut paraître complexe au premier abord, surtout sur l'aspect configuration.

Il a néanmoins le mérite d'apporter les avantages suivants:

- fourniture d'un framework ouvert adaptée dans l'écriture de batch
- moins de code à écrire, donc une meilleur productivité et une meilleur maintenabilité
- on récupère l'outillage de Spring (transaction, injection de dépendance, test unitaire etc …)

De plus Spring Batch présente des fonctionnalités avancées assez intéressantes:

- gestion de la reprise sur erreur
- possibilité de « skipping »
- possibilité de parallélisation

Spring Batch est donc un framework complet, robuste et stable.
Malgré sa complexité, il peut s'avérer interessant dans le cadre de batchs devant traiter de gros volumes de données.

Référence

- Documentation officielle de Spring Batch
- Spring Batch In Action (Thierry Templier, Arnaud Cogoluègnes, Gary Gregory, Olivier Bazoud)

26.01.2012 | Tags: Batch, J2EE, Spring | Categorie: Architecture, J2E

8 comments to Spring Batch par l'exemple

s.revel

26 Janvier 2012 at 16 h 23 min

J'ai quand même une remarque sur ce post, ayant effectué une étude de ce framework, il pose quand même une problématique.

Dans le cas de traitement de fichiers, ou d'une base MySql ou d'un traitement peu gourmand, spring Bacth permet en effet à un dévelopeur Java d'écrire rapidement son traitement.

Mais dans le cas d'un batch base de données, Spring Batch permet d'écrire le même code que ce soit pour un traitement oracle ou DB2 par exemple.
Or ces deux SGBD permettent de faire des traitement particulièrement optimisés (Oracle pour des traitements ensembliste et DB2 pour des traitements répétitifs).
Le fait d'utiliser Spring Batch empêche l'utilisation de ses optimisations et réduit le SGBD à leur plus petit dénominateur commun, un moteur SQL.

Les batchs sont très souvent utilisés pour des traitements lourds et les perfs sont un point non négligeable.
Donc réduire son Oracle ou son DB2 à un MySQL pour écrire son traitement batch en Java avec Spring Batch peut devenir très couteux en perf.

Il y avait aussi dans notre étude un petit bémol quand à l'administration d'un batterie de batch et la reprise en erreur de ceux-ci.

Hugo Capocci
26 Janvier 2012 at 18 h 34 min

Utilisant spring-batch sur mon projet actuel, et continuant de monter en compétences dessus, je vous remercie de faire partager ce framework vraiment salvateur dans bien des cas.

Néanmoins, je trouve un peu dommage que vos examples ne soient pas recopiables en l'état (qu'avez-vous fait avec les doubles quotes ? on dirait du word…)

Pour aller un peu plus loin, je dirais que ce framework est intéressant pour les batchs réguliers (plus que pour les one shots) à cause du coût d'apprentissage, mais spring propose également une interface web pour piloter ces batchs, et un système de sauvegarde « pas à pas » des étapes du batch, en base. Cette fonctionnalité à elle seule peut faire pencher en son utilisation !

Egalement, pour les tests, il y a encore plus simple :

utiliser les annotations
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = {« classpath:batch-context-test.xml »})

dans l'entête de votre classe de tests, et l'objet « JobLauncherTestUtils » fourni dans la librairie spring-batch-test

En vous souhaitant une très bonne continuation dans l'exploration de ce framework puissant !

saouti
9 Février 2012 at 19 h 29 min

Salut, merci pour cette explication c'est vraiment très intéressant.

Mais j veux savoir juste il est ou la déclaration de votre base de donnée ? il s'appel comment la base car dans l'exemple j'ai trouvé que la table.

Olivier Bazoud
30 Septembre 2012 at 14 h 14 min

Deux remarques:
- ce n'est pas utile de créer un nouvelle objet Person dans le cas exposé
- PersonJdbcWriter ne doit pas avoir d'annotation @Transactional car Spring Batch gère lui même les transactions

click here
29 Octobre 2012 at 0 h 25 min

I will immediately seize your rss feed as I can't find your email subscription link or newsletter service. Do you've any? Kindly permit me understand so that I could subscribe. Thanks.

author
3 Décembre 2012 at 14 h 34 min

comment

Quelques liens autour de Spring et son ecosysteme | Veille de Xoff
4 Avril 2013 at 20 h 45 min

[...] http://blog-rd.ideotechnologies.com/?p=2021 [...]

Clayton
15 Mai 2013 at 0 h 13 min

salut,bonjour,ça va ? Ici Avril
Je souffle mes 19 bougies dans

un mois j'assume totalement mon age !
Mon travail billettiste … Mon caractère est plutôt timide.

Take a look at my homepage: Clayton