SCIENCES SORBONNE UNIVERSITÉ

SRCS : Systèmes Répartis Client/Serveur

TME 8 : Enterprise Java Bean

Objectifs pédagogiques :

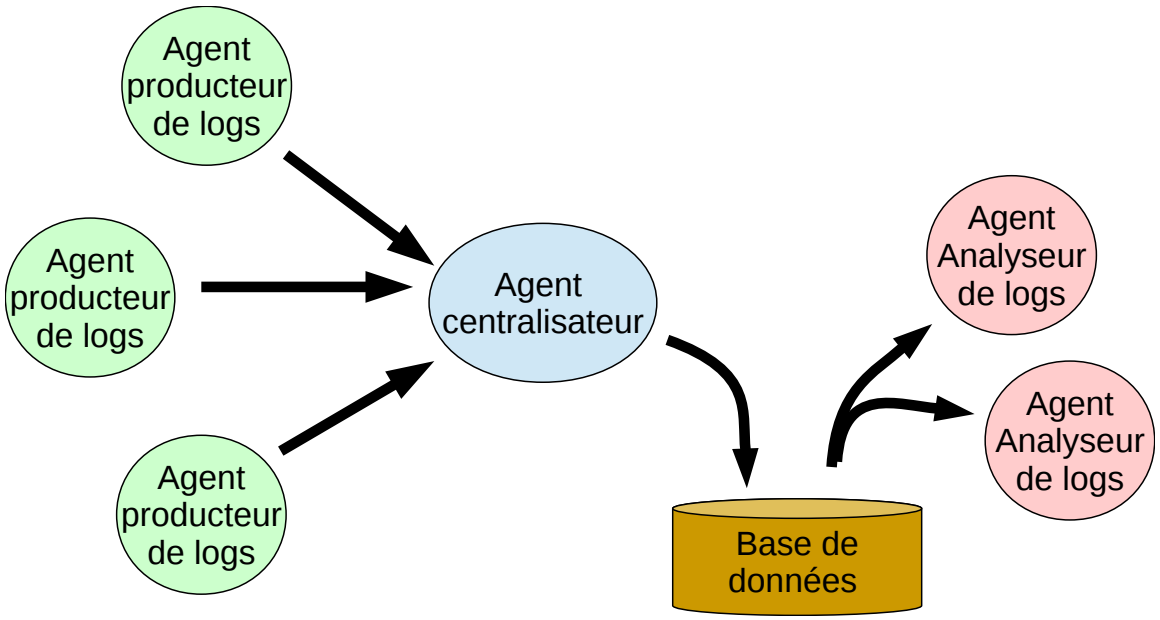
- **EJB session**
- **EJB Entity**

Objectif

L'objectif est de construire un système de log grâce à la technologie EJB. Ce système de log a la structure suivante :

- des agents vont collecter des informations et les envoyer à un agent centralisateur
- l'agent centralisateur sera chargé de les stocker dans une base de données
- d'autres agents seront chargés de l'analyse de données en interrogeant la base périodiquement et/ou ponctuellement.

La communication entre les agents clients collecteurs de données (sur les machines clientes) et le serveur centralisé de logs pourra se faire par l'intermédiaire de l'envoi de logs de manière synchrone ou asynchrone.



Ressources du TME

Pour simuler la production de log, les clients produiront leurs logs à partir de la lecture d'un fichier. Ainsi chaque client lira séquentiellement un fichier de log dont le nom lui sera passé en paramètre de la méthode main. Un fichier de log représentera les logs d'une machine particulière.

Exercice 1 – Création du modèle relationnel

Dans cet exercice, nous allons créer les entités nécessaires à schématiser les données de notre système. Comme vous pouvez le constater, dans les fichiers de log fournis, pour une machine donnée, un log est composé des informations suivantes :

- La date de production
- Niveau du log (DEBUG, INFO, WARN, ERROR)
- Le nom de la classe de l'objet qui a émis le log
- Le message du Log

À chaque Log nous souhaitons savoir depuis quelle machine il a été émis (dans notre cas le nom de la machine sera le nom du fichier d'entrée). Cependant nous souhaitons maintenir également une table des machines. Une machine sera caractérisée par un identifiant unique et un nom. Il existera donc une relation 1-N et N-1 entre les deux tables.

Question 1

Dans un premier temps, créez une Application JavaEE "LogApplication".

Question 2

Créer un projet JPA "LogJPA", l'affilier à l'EAR créée précédemment, créer le package `srcs.log` et y ajouter les classes suivantes

- la classe `Machine` : entité qui représentera une machine et dont les identifiants seront générés automatiquement
- la classe `DateLog` : classe annotée `@Embeddable` qui regroupera l'ensemble des attributs qui caractérisent la date d'un log (année, mois, jour, heure, minutes, secondes, millisecondes). Vous pourrez générer les méthodes `equals` et `hashCode` dans cette classe en faisant un clic droit sur votre code ⇒ *Source* ⇒ *Generate hashcode() and equals()*
- la classe `Log` : entité représentant un Log, identifié par un id généré automatiquement et qui comportera **entre autres** un attribut annoté `@Embedded` de type `DateLog`.

Définissez une méthode `toString()` pour chacune de ces classes

Question 3

Une fois tous les services nécessaires démarrés, générez vos tables dans la base de données à partir de vos entités

Exercice 2 – Agent centralisateur

Question 1

Créez un projet EJB *"LogTraitement"*

Question 2

Dans le package `srcs.log`, créez un EJB session `LogReceiver` et ajoutez-le à l'application JavaEE. Cet EJB comportera les services suivants accessibles à distance :

- `newLog` qui pour un log donné, l'ajoutera dans la base de données. Attention vous devez prendre en compte le fait que la machine associée au log donnée existe ou pas dans la base.
- `newLogAsync` qui fera le même traitement que le service précédent mais dont l'invocation se fera de manière asynchrone
- `getMachines` qui renverra un tableau de toutes les machines enregistrées dans la base
- `getLogs` qui renverra la liste de tous les logs de la base
- `getLogsWithLevel` qui, pour un level donné, renverra la liste de tous les logs associés.
- `mr_proper()` qui efface le contenu des tables `Machine` et `Log`

- `Log[] getLogsWithMachine(String name_machine)` qui pour une machine donnée, renvoie la liste des logs associés

Pour les requêtes JPQL, vous pouvez inclure une clause *WHERE* pour filtrer les résultats de la requête en fonction d'un prédicat.
Exemple : pour obtenir les tuples d'une table FOO, (mappé par l'entité Java `Foo`), il vous est possible de sélectionner les tuples où la colonne BAR (mappé par l'attribut `bar` dans la classe `Foo`) vous pouvez écrire :

```
Query q = em.createQuery("SELECT f FROM Foo f WHERE f.bar = :my_value_bar");
q.setParameter("my_value_bar", <votre_valeur_de_bar_ici>);
```

Vous pourrez remarquer qu'un paramètre dans une requête JPQL est un mot précédé de `":"`. La valeur du paramètre peut être spécifiée ensuite par la méthode `setParameter` de la classe `Query`.

Question 3
Déployez votre application et assurez-vous que tout se passe bien.

Exercice 3 – Client producteur de Log

Dans le cadre de cet exercice, nous allons rester dans un environnement Java SE classique.

Question 1
Créer un projet Java "LogProduction" client (n'oubliez pas de bien configurer le Build path).

Question 2
Dans un package `srcs.log.util` créez une classe `LogReader` qui un `FilterReader`. Ce décorateur permet de lire des `Log` à partir d'un flux de caractères pour une machine donnée. Cette classe offre :
- un constructeur qui prend une chaîne de caractères désignant le nom de la machine en question et un reader de ligne `BufferedReader` qui est le flux que l'on décore.
- une méthode `Log nextLog()` qui permet de transformer la ligne lue sur le flux décoré en objet Log. Pour transformer une ligne de caractères en `Log`, vous pourrez éventuellement vous aider de la classe utilitaire `Parser` fournie dans les ressources qui renvoie pour une ligne donnée une information du log correspondant.

Question 3
Testez votre classe avec le test JUnit `TestLogReader`.

Question 4
Testez l'implantation du service avec le test JUnit TestLogGenerator et les fichiers de logs qui vous sont fournis dans les ressources (il y en a 5 mais vous pouvez en utiliser moins dans les phases de debug en modifiant la méthode `beforeAllTest`).

Exercice 4 – Analyseur de Log en ligne

Nous souhaitons maintenant pendant la collection de Log afficher périodiquement sur la console du serveur des informations sur la liste des Logs. Ce traitement se fera de manière périodique grâce à la fonctionnalité des Timer.

Question 1
Créez un EJB *"LogWatcher"* (à vous de déterminer si il s'agit d'un Stateless, Stateful ou Singleton) que vous ajouterez à votre projet EJB créé précédemment.

Question 2

Ajoutez une dépendance de type `@EJB` sur l'EJB `LogReceiver`.

Question 3

Codez une méthode qui se déclenchera toutes les 10 secondes et qui enverra sur la sortie standard (avec `System.out.println`) :

- la taille des tables Machine et Log.
- tout nouveau log de type WARN ou ERROR. Attention un log ne doit être affiché qu'une seule fois.

Question 4

Pour tester votre nouveau composant, redéployer entièrement votre application afin que les timer soient bien réinitialisés (dans Eclipse, supprimez l'application du serveur et redéployez-là ensuite). La sortie standard du composant est redirigée dans le fichier *server.log* situé dans le répertoire *glassfish/-domains/domain1/logs*