Gestion des accès concurrents

Licence DANT

PITTON Olivier

Gestion des accès concurrents

Différents threads peuvent lire / écrire des références en parallèle.

Les deux points à prendre en compte :

- La visibilité : Quand une modification sera visible par les autres ?
- La cohérence : Ne pas corrompre les données

Gestion des accès concurrents

Java propose deux mécanismes pour le verrouillage :

- Les moniteurs avec le mot-clé synchronized
- Les verrous définis par l'interface Lock

Gestion des accès concurrents

Un verrou (lock) Empêche en lecture / écriture les accès concurrents.

Un moniteur Est un type de verrou en écriture permettant la coopération entre threads (wait / notify)

Un sémaphore Verrou autorisant N entrées avant de verrouiller

Gestion des accès concurrents

3 règles lors de la mise en place de verrous :

- Un objet immuable est thread-safe
- Doit être maintenu le moins longtemps possible
- Ne les utiliser que si nécessaire

Gestion des accès concurrents

Le mot-clé synchronized se place sur :

- une méthode
- une reference

Il garantit qu'un seul thread peut executer le code par référence

Gestion des accès concurrents

En live io.dant.synchro.cours.SynchronizedKeyword

Gestion des accès concurrents

Si une classe possède plusieurs méthodes définies avec le mot clé synchronized, alors une seule de ces méthodes pourra être exécutée en même temps par plusieurs threads, même si chaque thread exécute une méthode différente.

Il est possible d'imbriquer dans le même bloc de code plusieurs instructions synchronized sur des moniteurs différents. Attention cependant, l'ajout de verrous pour gérer des cas de gestion d'accès concurrents peut augmenter, sous certaines circonstances, le risque d'introduire des situations de deadlocks.

Gestion des accès concurrents

Le modèle mémoire Java ne garantit pas la visibilité des modifications.

Le mot-clé volatile sert à cela.

Se met uniquement sur une variable

Gestion des accès concurrents

volatile force l'écriture de la valeur d'une variable et sa relecture.

Une variable peut être mise en cache à plusieurs niveaux, un thread tourné sur différents CPU. Plusieurs copies peuvent exister.

Ne garantit pas l'atomicité

Gestion des accès concurrents

En live io.dant.synchro.cours.VolatileKeyword

Gestion des accès concurrents

Le deadlock est un état où plusieurs threads attendant des ressources indéfiniment

En live io.dant.synchro.cours.DeadlockExample

Gestion des accès concurrents

La section critique d'une méthode est un bloc de code qui doit être thread-safe.

Une section critique non protégée entraîne des races conditions.

Une race condition est une incoherence de résultat à cause de multi-threads.

Gestion des accès concurrents

Quelques conseils :

- Privilégier les objets immuables
- Penser à la copie défensive

Java – wait / notify

Licence DANT

PITTON Olivier

wait / notify

Le mot-clé Java synchronized permet la cooperation entre threads.

Les threads peuvent s’attendre (wait) et se réveiller (notify)

Les appels à ces méthodes doivent être dans des blocs synchronized, sur les mêmes références.

wait / notify

En live io.dant.synchro.cours.WaitNotify

Java – Les verrous

Licence DANT

PITTON Olivier

Les verrous

L’interface Lock permet la création de verrou avec plus de possibilité que synchronized :

- Attente bloquante et non bloquante
- Verrou en lecture et en écriture
- Support des conditions
- Gère la famine

Les verrous

Classe ReentrantLock Verrou en écriture

En live io.dant.synchro.cours.MyReentrantLock

Classe ReentrantRedWriteLock Verrou en lecture / écriture

En live io.dant.synchro.cours.MyReentrantReadWriteLock

Les verrous

L’interface Condition permet de mettre en attente un thread jusqu’à ce qu’il reçoive une notification

Remplace wait / notify

En live io.dant.synchro.cours.MyCondition

Java – Les operations atomiques

Licence DANT

PITTON Olivier

Les operations atomiques

Une operation atomique est une operation qui ne peut pas être exécutée partiellement.

Toutes les instructions ont la garantie d’être exécutées sans interruption

Les operations atomiques

Problème d’incrementation parallèle.

Une incrementation se fait en 3 instructions :

- la lecture en mémoire de la valeur courante
- son incrémentation
- l'écriture en mémoire de la nouvelle valeur

En live io.dant.synchro.cours.atomic.MyOldCompteur

Les operations atomiques

Utilisation des classes Atomic<Type>

AtomicInteger, AtomicLong, AtomicIntegerArray …

Evite l’utilisation de verrous qui est coûteux et bloquant.

En live io.dant.synchro.cours.atomic.MyNewCompteur

Les operations atomiques

Algorithme bloquant approche pessimiste (les verrous)

Algorithme non-bloquant approche optimiste (CAS)

Ce type d’algorithme est plus difficile à écrire et consiste à réaliser une operation jusqu’à ce qu’elle réussise.

Les operations atomiques

Compare-And-Swap : Opération atomique de mise à jour d'une valeur

Requiert une valeur courante et une valeur souhaitée.

Mise à jour de la valeur en mémoire si la valeur est celle souhaitée, sinon ne fait rien

Les operations atomiques

En live io.dant.synchro.cours.atomic.MyCasCompteur