Transactions et SPRING Framework

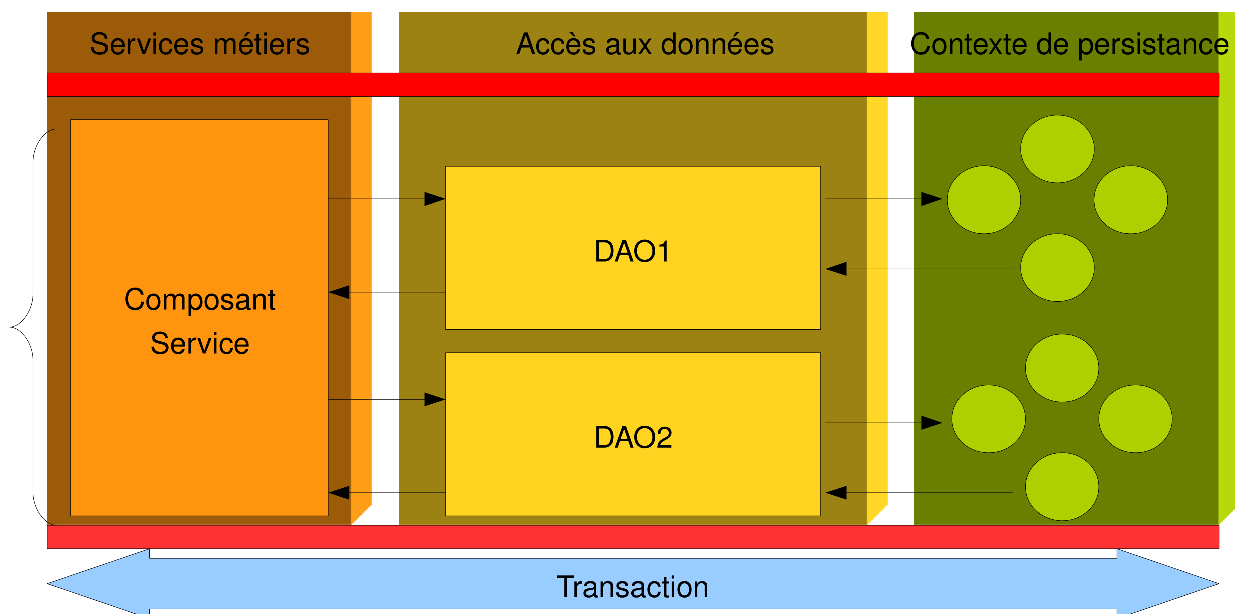
Définition d'une Transaction : Une transaction regroupe une série d'opérations dans un tout indivisible. La transaction n'est validé que si chacune des taches unitaires qu'elle regroupent se sont déroulées correctement (COMMIT). Dans le cas contraire (si une des taches unitaires échoue) l'ensemble des données traitées retrouvent leur état initial(ROLBACK).

Une transaction doit respecter les quatre contraintes suivantes dites ACID :

- ***Atomicité*** : une transaction doit s'effectuer en tout ou rien ;
- ***Cohérence*** : la cohérence des données doit être assurée dans tous les cas, même dans les cas d'erreur où le système doit revenir au précédent état cohérent ;
- ***Isolation*** : la transaction va travailler dans un mode isolé où elle seule peut voir les données qu'elle est en train de modifier, cela en attente d'un nouveau point de synchronisation ; le système garantit aux autres transactions, exécutées en parallèle sur le même système, une visibilité sur les données antérieures ;
- ***Durabilité*** : lorsque la transaction est achevée, le système est dans un état stable durable, soit à l'issu d'une modification transactionnelle réussie, soit à l'issue d'un échec qui se solde par le retour à l'état stable antérieur.

Overview d une transaction applicative

les transactions spring permettent d'encapsuler plusieurs appels de méthodes sur différentes couches et d'en assurer le caractère ACID.



SPRING permet de les gérer de manière déclarative et programmatique les transactions.

Typiquement nous pouvons déclarer des rollbacks en fonction d une exception qui est déclenchée.

Exemple d une annotation déclarant les règles transactionnelles à suivre pour une méthode:

@Transactional(readOnly = false, propagation = Propagation.MANDATORY, rollbackFor = {Exception.class},isolation=Isolation.REPEATABLE_READ)

Les niveaux d'isolation des transactions

Dans le cas d'une concurrence d'accès Il peut arriver que plusieurs transactions travaillent sur les mêmes informations sans que l'une ou l'autre ne soit terminée. Dans ce genre de cas il possible de perdre la cohérence des données.

Nous pouvons rencontrer 3 types de problèmes relatifs aux transactions concurrentes :

- Lecture sale ou " dirty read " : Une transaction lit des données contenant un changement non validé d'une autre transaction. Une partie des données peut se révéler fausse en fonction du commit ou du rollback de l'autre transaction.
- Lecture non répétable ou " nonrepeatable reads " : Une transaction lit une donnée, une seconde transaction change la même donnée et la première transaction relit la donnée et obtient une valeur différente. La donnée a changé et est incohérente sur la transaction.
- Lecture fantôme ou " phantom reads " : Dans une transaction on ré-exécute une requête, retournant un ensemble de données qui satisfont une condition de recherche. Un nouveau jeu de données apparaît entre les deux opération de lecture.

Pour pallier à ce genre de situation nous avons la possibilité de régler la manière dont chacune de nos transactions seront isolées les unes des autres.

TRANSACTION_READ_UNCOMMITTED : La transaction peut lire des données non validées, c'est à dire des données qui ont été modifiées et non validées par des transactions concurrentes. Toutes les erreurs vues ci-dessus peuvent arriver.

Ce niveau est très dangereux dans les environnements stratégiques où des transactions simultanées mettent à jour des données partagées. Il est également inapproprié pour tous les calculs sensibles, comme les opérations de débit/crédit sur comptes bancaires qui doivent adopter un mode d'isolation plus strict.

Ce niveau d'isolation reste approprié si vous savez à l'avance qu'une instance de votre composant ne fonctionne qu'en l'absence de toute autre transaction simultanée. Cependant, dans la plupart des contextes transactionnels, ce niveau d'isolation est insuffisant.

Son principal avantage est la performance, comme le système transactionnel sous-jacent n'a pas besoin de verrouiller les données partagées.

TRANSACTION_READ_COMMITTED : Ce niveau d'isolation fait que l'on ne peut pas lire les changements non validés des transactions concurrentes.

Ce niveau est certainement plus robuste que le mode précédent. Vous ne pouvez plus lire les données écrites, et non validées, ce qui signifie par conséquent que toutes les données que vous lisez sont cohérentes.

Ce mode est fréquemment utilisé pour les programmes qui effectuent des lectures en base pour constituer des rapports sur les valeurs des données.

TRANSACTION_REPEATABLE_READ : Ce niveau d'isolation assure en plus que lire les données plusieurs fois retourneront les mêmes valeurs même si une autre transaction modifie ces données. Les lectures sont répétables.

Ce mode est utile lorsque l'on doit mettre à jour un ou plusieurs éléments de données d'une ressource, comme un ou plusieurs enregistrement d'une BDDR. Il est nécessaire de lire toutes les lignes et de les mettre à jour, en sachant qu'aucune n'est modifiée par d'autres transactions simultanées. Si on choisit de relire l'une de ces lignes ultérieurement au cours de la transaction, on a la certitude qu'elle contient les mêmes données qu'au début de la transaction.

TRANSACTION_SERIALIZABLE : La transaction a les privilèges exclusifs en lecture/ecriture sur des données en les bloquant; les autres transactions ne peuvent ni lire ni écrire sur ces données. C'est le niveau d'isolation le plus contraignant empêchant également les " phantom reads ".

Ce mode doit être utilisé pour les systèmes stratégiques qui exigent une parfaite isolation transactionnelle. Il assure que les données lues ont été validées, que la lecture reste identique au fil du temps et que de mystérieuses données validées n'apparaissent pas dans la base en cours de traitement en raison de transactions simultanées.

Ce niveau d'isolation doit être utilisé avec parcimonie, car il induit un coût certain. Si toutes les opérations sont réalisées dans ce mode, les performances de la base de données ont tendance à se dégrader très rapidement jusqu'à l'immobilisation éventuelle.

Récapitulatif des différents niveaux d'isolation et de leurs effets respectifs			
Isolation Level	**Dirty Read**	**Non Repeatable Read**	**Phantom Read**
TRANSACTION_READ_UNCOMMITTED	possible	possible	possible
TRANSACTION_READ_COMMITTED	aucune	possible	possible
TRANSACTION_REPEATABLE_READ	aucune	aucune	possible
TRANSACTION_SERIALIZABLE	aucune	aucune	aucune

Les modes de propagation des transactions

Propagation : REQUIRED

Description : La méthode doit forcément être exécutée dans un contexte transactionnel existant. S'il n'existe pas lors de l'appel, il sera créé. **En cas d'erreur d'une des méthode composant la transaction elle seront toutes annulés**

Contexte transactionel appelant	Aucun contexte transactionel appelant	Existe dans la norme EJB3
S'execute dans le contexte existant	Créer un nouveaux contexte	oui

Propagation: SUPPORTS

Description : La méthode peut être exécutée dans un contexte transactionnel s'il existe. Dans le cas contraire, la méthode sera exécutée quand même mais hors d'un contexte transactionnel.

Contexte transactionel appelant	Aucun contexte transactionel appelant	Existe dans la norme EJB3
S'execute dans le context existant	Aucune transaction	oui

Propagation: MANDATORY

Description : La méthode doit forcément être exécutée dans un contexte transactionnel. Si tel n'est pas le cas, une exception sera levée.

Contexte transactionel appelant	Aucun contexte transactionel appelant	Existe dans la norme EJB3
S'execute dans le context existant	leve une exception	oui

Propagation: REQUIRES_NEW

Description : La méthode impose la création d'une nouvelle transaction pour la méthode. Si un contexte transactionnel existe il sera suspendu jusqu'à ce que la nouvelle transaction soit terminé.

Ce comportement est à utiliser si vous avez besoins que la méthode soit annulée en cas d'erreur sans propager l'annulation à la méthode appelante.

Attention entraîne des pertes de performances en cas d'utilisation abusive.

Contexte transactionel appelant	Aucun contexte transactionel appelant	Existe dans la norme EJB3
Créer un nouveau context	Créer un nouveau context	oui

Propagation: NOT_SUPPORT

Description :aucun contexte transactionnel n'est appliqué. Peut être util si vous êtes sur que la méthode n'effectue pas d'opération critique.

Contexte transactionel appelant	Aucun contexte transactionel appelant	Existe dans la norme EJB3
aucun contexte transactionel	Hors contexte transactionel	oui

Propagation: NEVER

Description : La méthode ne doit pas être exécutée dans un contexte transactionnel. Si tel n'est le cas, une exception sera levée.

Contexte transactionel appelant	Aucun contexte transactionel appelant	Existe dans la norme EJB3
leve une exeption	Hors du contexte transactionel	oui

Propagation: NESTED

Description : La méthode est exécutée dans une transaction imbriquée si un contexte transactionnel existe lors de son appel. Permet de mettre en place des savepoint si le drivers JDBC sous jacent le permet

Contexte transactionel appelant	Aucun contexte transactionel appelant	Existe dans la norme EJB3
Créer une nouvelle transaction et l'imbrique dans la transaction courante	Crée une nouvelle transaction	non

Comportement des transactions en fonction des Exceptions

Les RuntimeException seront rollabcké dans tous les cas, quand aux Exception applicatives ces dernières peuvent être traité et laissé la transaction finir son boulot en fonction des besoins fonctionnels de l'application.