Découvrez Java EE 5 avec NetBeans - partie 2

par Patrice Secheresse

Date de publication : 17/06/06

Dernière mise à jour : 17/06/06

Cet article est la suite de notre tutoriel d'introduction aux bases de Java EE 5 avec l'utilisation de NetBeans 5.5. Le tutoriel continue sur les beans de type session pour montrer la simplicité apportée dans ce domaine. Durée prévue: 30 minutes

I - Logiciels nécessaire pour utiliser ce tutorial

Avant de commencer, vous devez installer les logiciels suivant sur votre ordinateur:

- NetBeans IDE 5.5 (télécharger). La version 5.5 Beta contient l'ensemble NetBeans IDE + Sun Java System Application Server 9 et est le meilleur choix pour un débutant car il suffit d'installer et tout est prêt.
- Java Standard Development Kit (JDK) version 5 (télécharger)

II - Notations utilisée dans ce tutorial

<GLASSFISH_HOME> - le répertoire d'installation du serveur d'application Sun Java System Application Server 9 (en général c:\Sun\AppServer) ou du serveur Glassfish

III - Une calculatrice pour tout le monde

Notre premier exemple utilisait un bean session avec une interface éloignée (remote). Les premiers EJB 1 ne possédaient que cette interface faite pour communiquer entre deux JVM différentes, souvent sur des systèmes différents. Avec une interface éloignée, une instance est transmise par valeur, il faut donc la sérialiser puis la dé sérialiser à la réception à travers le réseau. Nous avons pu voir que ce type d'opération peut être très long. Pour éviter ces opérations dans le cas de Beans destinés à communiquer à l'intérieur du même conteneur, il existe le type d'interface local, un des plus important apports de la version 2 des EJB. Une interface locale permet de communiquer sans passer par les services réseaux et est donc plus proche d'un classique appel de fonction avec passage de paramètre par référence. Les gains de performance sont donc énormes. Lorsque nous devons faire communiquer des centaines d'instance de beans résidentes dans le même conteneur, inutile de gaspiller les ressources en réalisant des opérations inutiles. Autre problème, notre Bean sans état à la mémoire un peu courte. Certaines opérations sont parfois réalisées en plusieurs étapes et nécessitent de mémoriser les états intermédiaires jusqu'à la finalisation de l'opération. Il existe un type de bean session à état pouvant conserver un état durant toute la session avec le client, ce qu'on appelle l'état conversationnel. Il ne s'agit pas de persistence car un bean à état n'est pas prévu pour survivre à un arrêt du serveur ou un crash, son existence est éphémère.

Illustrons par un nouvel exemple concret très proche du précédent : nous allons créer un EJB servant de calculatrice pour faire une addition, mais cette fois nous voulons que cette calculatrice soit utilisable par les autres composants du conteneur. Il voulons utiliser une interface locale pour que les opérations avec notre bean soient les plus performantes possible à l'intérieur de notre conteneur EJB car nous avons prévu dans le futur d'utiliser ce bean massivement.

Autre particularité, nous voulons additionner les nombres à un sous total au fur et à mesure qu'ils arrivent. Notre Bean va donc additionner le nombre qui lui est fourni avec le sous total qu'il a en mémoire et renvoyer le nouveau sous total. Il faut donc mémoriser le cumul en cours en utilisant un bean session à état.

IV - Création du projet

NetBeans 5.5 et le serveur d'application Sun Java System Application Server 9 sont installés. Nous démarrons NetBeans 5.5.

Nous commençons par créer un nouveau projet par le menu File > New Project, nous choisissons Entreprise et EJB Module puis Next. Laissez le nom par défaut (ce sera quelque chose comme EJBModule2), indiquez le répertoire où vous désirez placer votre projet et assurez vous que la version est bien Java EE 5 (si ce n'est pas le cas, aucun serveur Java EE 5 n'est enregistré, il faut alors choisir l'onglet Runtime, puis avec le clic droit sur Server choisir Add Server et indiquer le serveur et son répertoire d'installation) et cliquez sur Finish.

V - Création d'un Bean Session sans état à interface locale

Nous recommençons exactement la même chose que dans la partie 1 du tutoriel, mais cette fois, notre calculatrice est locale.

Dans l'onglet Projet, ouvrez le noeud EJBModule2 pour voir le noeud Source Package et avec le click droit de la souris sur ce dernier choisissez New puis Session Bean. Dans la zone EJB Name, indiquez Calculatrice et dans la zone package indiquez demo.ejb3.calculatrice. Choisissez un type de Session Stateless (sans état qui est l'opposé de Stateful soit à état) et la création d'une interface de type Local (désélectionnez Remote si celui ci est coché). Ce sont les valeurs par défaut de NetBeans. Cliquer sur Finish.

NetBeans créé alors deux sources, un pour l'implémentation de notre calculatrice, l'autre pour l'interface de notre calculatrice. Le source de notre Bean se nomme Calculatrice2Bean et est affiché : nous sommes prêt à saisir notre implémentation. Pour mieux comprendre les relations, ouvrons le source de l'interface CalculatriceLocal. Pour cela, dans l'onglet Projet, ouvrez les noeud EJBModule2, Source Package, puis notre package demo.ejb3.calculatrice.

Dans ce package, vous trouvez le source de l'interface CalculatriceLocal; double cliquez pour l'ouvrir. Pour le moment, l'interface est encore une enveloppe vide qui n'étend aucune autre interface et ne contient aucune méthode :

```
@Local
public interface CalculatriceLocal {

}
```

Vous pouvez remarquer simplement l'annotation @Local qui permet de deviner quel est le rôle.

Nous pouvons maintenant récapituler les annotations servant à définir une interface d'un bean de type session : @Remote pour une interface éloignée ou @Local pour une interface locale. Par défaut l'interface est locale si l'annotation n'est pas indiquée, rien alors trahit que l'interface est une interface EJB, comment faire plus simple !!!

Revenons à notre source CalculatriceBean. Nous remarquons qu'il est aussi très simple avec une simple annotation @Stateless() qui le différencie d'un bean Java SE et qu'il implémente l'interface CalculatriceLocal.

Cette annotation stateless est le marqueur pour que l'état du bean ne soit pas mémorisé.

Nous allons fournir un composant fonctionnellement identique à celui de la première partie, les manipulation sont exactement les mêmes.

Dans le source CalculatriceBean, cliquer avec le bouton droit pour avoir le menu contextuel et choisir EJB Methods puis Add Business Method...

Dans le haut de la fenêtre Add Business Method..., nous indiquons le nom additionner dans la zone Name et int dans le type de retour Return Type. Dans la partie basse, dans l'onglet Parameters, nous cliquons sur Add pour ajouter un paramètre x de type int en indiquant int dans la zone Type et x dans la zone name suivi de OK pour confirmer la fenêtre Enter Method Parameter. Nous cliquons à nouveau sur Add pour ajouter un paramètre y de type int en indiquant int dans la zone Type et y dans la zone Name suivi de OK pour confirmer la fenêtre Enter Method Parameter. Nous confirmons notre méthode en cliquant sur OK.

Nous allons maintenant terminer notre CalculatriceBean en remplaçant return 0 par return x + y. Le source CalculatriceBean.java résultant est :

```
ackage demo.ejb3.calculatrice;

import javax.ejb.Stateless;

/**
 *
 * @author PSe
 */
@Stateless
public class CalculatriceBean implements CalculatriceLocal {

    public int additionner(int x, int y) {
        return x + y;
    }

}
```

Pour l'interface CalculatriceLocal.java, la méthode a été automatiquement ajoutée :

```
package demo.ejb3.calculatrice;

import javax.ejb.Local;


/**
 * This is the business interface for Calculatrice enterprise bean.
 */
@Local
public interface CalculatriceLocal {
    int additionner(int x, int y);

}
```

VI - Ayons une belle facade

Jusque là, rien de bien extraordinaire et notre Bean est inaccessible depuis l'extérieur : l'interface étant locale, il n'est utilisable que dans le conteneur. La solution est de fournir un Bean éloigné chargé de communiquer avec le bean calculatrice. Ce bean va implémenter le patron de conception (design pattern) Facade très classique dans ce cas. Il fourni un accès aux possibilités internes de notre système pour les clients externes. Un bean Facade doit :

- masquer la complexité et l'implémentation du service
- être de granularité large (coarse grained) pour réaliser une opération complète en un seul appel. En effet, les accès éloignés étant coûteux en temps et ressource, il est largement plus performant de faire un seul appel avec l'ensemble des paramètres que de faire de nombreux appels avec un seul paramètre à chaque fois

Dans cet exemple, la simplicité de notre calculatrice ne nécessite pas une grosse recherche. Nous nous contenterons de faire suivre l'appel depuis le Bean Facade vers le Bean calculatrice. Cependant, ce bean doit présenter la particularité de conserver le sous total entre chaque appel. Pour cela, nous créons un bean session avec état nommé CalculatriceFacade dans lequel nous mettons une méthode additionner ayant pour objectif de récupérer le résultat du dernier calcul puis de passer l'appel en local.

Dans l'onglet Projet, ouvrez le noeud EJBModule2 pour voir le noeud Source Package et avec le clic droit de la souris sur ce dernier choisissez New puis Session Bean. Dans la zone EJB Name, indiquez CalculatriceFacade et dans la zone package indiquez demo.ejb3.calculatrice. Choisissez un type de Session Stateful et la création d'une interface de type Remote (dé sélectionnez Local si celui ci est coché). Cliquer sur Finish.

Pour conserver le sous total, nous définissons une variable de membre nommé sousTotal de type int dans la classe juste après la déclaration de la classe :

```
        int sousTotal;
```

La mise en place de la méthode additionner est identique à la précédente avec le clic droit dans le source puis choisir EJB Methods puis Add Business Method... je vous épargne la copie du paragraphe et vous demande d'essayer de la faire tout seul. La différence est que notre méthode ne contient qu'un seul paramètre :

```
            public int additionner(int x) {}
```

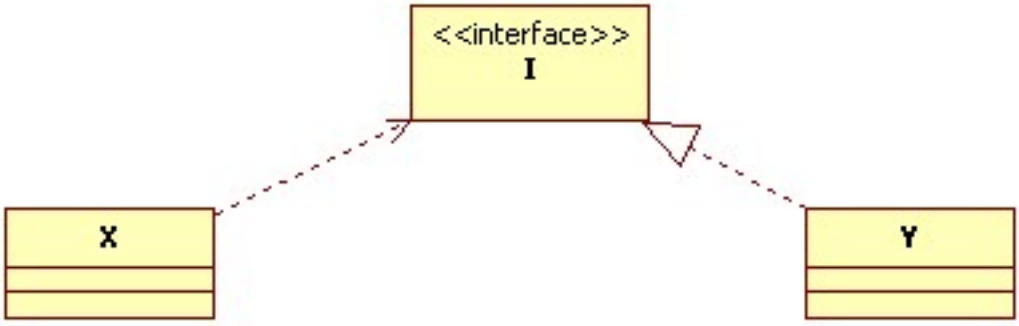
Cette méthode doit simplement faire appel à notre bean CalculatriceLocal au travers du conteneur pour garder les bénéfices de la sécurité, des transactions et bien d'autres services dispensés par celui-ci. Grâce à l'injection de dépendance, une des grandes nouveautés de Java EE 5, cela se fait de façon presque aussi naturelle qu'un appel de méthode hors du conteneur.

VI - L'inversion de dépendance

Un peu de théorie car le terme injection de dépendance est largement utilisé dans la littérature concernant Java EE 5. Une classe X est dite dépendante de Y si X ou une instance x de la classe X utilise des méthodes ou des champs de la classe Y ou une instance y de la classe Y. Si de plus Y est dépendant de X, alors la dépendance est cyclique. Une dépendance cyclique peut être amenée par une chaîne de dépendance (X dépend Y qui dépend de Z qui dépend de X et vice versa).

Dépendance de X avec Y

Pour casser cette dépendance, on introduit une interface I qui devra contenir toutes les méthodes de y que x peut appeler. De plus, Y doit être changée de façon à ce qu'elle implémente l'interface I. X et Y sont maintenant dépendant de l'interface I et la classe X ne dépend plus de la classe Y, en admettant que x n'instancie pas Y.



Casser la dépendance

Cette élimination de la dépendance de la classe X sur Y en introduisant une interface I est dite inversion de contrôle (*Inversion of Control- IOC*) ou **inversion de dépendance**.

Il faut noter que Y dépend d'autres classes. Avant l'application de la transformation, X dépendait de Y et par conséquent X dépendait de toutes les classes dont Y dépendait. En appliquant l'inversion de contrôle, toute ces dépendances indirectes ont été complètement éliminées aussi, pas seulement la dépendance de X avec Y. La nouvelle interface I dépend d'aucune autre.

Les programmeur ont appliqué l'inversion de contrôle dans des conteneurs (Inversion of Control Container - Fowler 2004). Le logiciel réclame un objet au conteneur et le conteneur construit l'objet et ses dépendances. Des exemples de ces conteneurs sont HiveMind, PicoContainer, Spring et Excalibur. A noter que Spring est une plate forme entreprise complète, pas seulement un conteneur IOC et apporte une grande facilité d'utilisation qui l'on conduit à une grande popularité. Les ingrédients de ce succès ont donc été repris pour améliorer de façon radicale Java EE.

VIII - L'injection de dépendance dans Java EE 5

Le principe d'utiliser une interface est la base de beaucoup de patrons de conception (design pattern) et permet de créer une couche d'abstraction entre différents composants. L'interface présente le contrat qui doit être rempli et le fournisseur de composant implémente les classes pour remplir ce contrat. L'utilisateur de l'interface peut choisir n'importe quel fournisseur de composant implémentant cette interface tant que le contrat est respecté. C'est ce principe qui va permettre de créer des applications utilisant les services Java EE de n'importe quel fournisseur s'il respecte le contrat Java EE. Les spécifications (JSR) sont pilotées par des comités (JCP) comprenant de nombreux fournisseurs et permettent d'utiliser n'importe quel conteneur JavaEE conforme aux spécifications sans modifier ou recompiler son application.

La question à régler dans notre exemple est comment va-t-on obtenir une classe Y en provenance de notre conteneur lors de l'exécution de notre classe X qui a besoin d'une instance de la classe concrète Y pour réaliser le contrat? Le conteneur Java EE utilise le service JNDI qui permet de retrouver un objet à partir d'un nom. Cependant, comme nous avons vu lors dans les tests unitaires, l'utilisation des services JNDI est rébarbatif, nécessite de nombreuses lignes de code et peut apporter des erreurs. Cela a amené de nombreuses critiques, en particulier du conteneur EJB dans les versions J2EE 1.4 et antérieures. La grande nouveauté de Java EE 5.0 est l'injection de dépendance grâce aux annotations qui allègent la corvée du lookup, du narrowing, des Services Locators et de la jungle des descripteurs de déploiement.

L'utilisation des annotations permet de désigner les champs ou les propriétés correspondants aux ressources dont nous avons besoin. Le conteneur se charge lors de l'utilisation de la classe de mettre en place la référence à la ressource demandée dans le champ ou la propriété avant l'exécution de la première méthode. Auparavant l'EJB demandait une ressource par l'intermédiaire d'une recherche grâce au service JNDI, maintenant c'est le conteneur qui fournit la référence à la ressource : c'est l'**injection de dépendance**.

IX - Notre bean à injection

L'appel de la méthode de notre composant calculatrice doit se faire après avoir obtenu une référence à la classe qui implémente l'interface calculatrice. Il suffit de déclarer un champ du type attendu précédé d'une annotation @EJB. Dans notre cas, nous voulons une classe qui implémente l'interface CalculatriceLocal, le code va être :

```
        @EJB
        private CalculatriceLocal calculatriceBean;
```

Remarque importante : la version 5.5 propose la possibilité de générer automatiquement la ligne permettant d'obtenir la référence au bean appelé. Cette option fonctionne sur les version NetBeans de développement récentes, pour cela, il suffit de faire un clic droit dans le source, choisir 'entreprise ressource' puis 'call Entreprise Bean'. La liste des beans du projet apparait. Cliquez sur CalculatriceBean et la ligne est automatiquement insérée.En version 5.5Beta, il faut enlever dans le fichier de configuration sun-ejbjar.xml le bloc concernant l'EJB.

La tache du conteneur sera de fournir la référence par injection dans la variable calculatriceBean. Pour cela, le type de cette variable permet au conteneur de connaître quel bean est demandé. Là encore, le code est d'une extrême simplicité.

Il ne reste plus qu'à ajouter l'appel à la méthode additionner de notre bean local. Le code complet de notre bean CalculatriceFacadeBean.java est celui-ci :

```
    package demo.ejb3.calculatrice;

    import javax.ejb.EJB;
    import javax.ejb.Stateful;

    /**
     *
     * @author PSe
     */
    @Stateful
    public class CalculatriceFacadeBean implements CalculatriceFacadeRemote {

        @EJB
        private CalculatriceLocal calculatriceBean;
        int sousTotal;

        public int additionner(int x) {
            sousTotal = calculatriceBean.additionner(x, sousTotal);
            return sousTotal;
        }

    }
```

Une fois le code complet en place, vous pouvez alors déployer le projet par la touche F6 qui va démarrer le serveur d'application et installer le bean sous forme packagée.

X - Le test unitaire de notre calculatrice

Comme dans la première partie, le principe est simple : depuis le source de CalculatriceFacadeBean appuyez sur les touches Ctrl+Maj+U ou utilisez le menu Tools puis Create JUnit Test (le source de CalculatriceFacadeBean doit être sélectionné pour que cette option fonctionne). Laissez les valeurs par défaut et cliquer sur OK. NetBeans va créer tout le squelette nécessaire à faire un test de notre Bean.

Nous mettons tout de suite les quelques tests qui permettent de vérifier que les additions sont correctes, mais surtout que l'appel successif du même bean cumule le nombre à additionner au résultat précédent, prouvant que le bean conserve l'état précédent. Saisir le contenu de la méthode testAdditionner :

```
    /**
     * Test of additionner method, of class demo.ejb3.calculatrice.CalculatriceFacadeBean.
     */
    public void testAdditionner() {
        System.out.println("additionner");

        int x = 0;
        int expResult = 0;
        InitialContext ctx;
        try {
            ctx = new InitialContext();
            Object ref = ctx.lookup(CalculatriceFacadeRemote.class.getName());
            CalculatriceFacadeRemote calc =
(CalculatriceFacadeRemote)PortableRemoteObject.narrow(ref,CalculatriceFacadeRemote.class);
            // Première addition + 0 = 0
            int result = calc.additionner(x);
            assertEquals(expResult, result);
            // Deuxième + 2 = 2, le bean a en mémoire le précédent résultat qui est 0
            x= 2;
            expResult = 2;
            result = calc.additionner(x);
            // Troisième + 2 = 4, le bean a en mémoire le précédent résultat qui est 2
            expResult = 4;
            result = calc.additionner(x);
            assertEquals(expResult, result);
        } catch (NamingException ex) {
            fail(ex.getMessage());
        }
    }
```

Utilisez les touches Alt+Maj+F pour résoudre les imports et le code est prêt. Par rapport au test vu dans la partie 1, une petite astuce permet de se rappeler que si aucun paramètre ne fixe le nom JNDI du bean dans le conteneur, c'est le nom de l'interface qui est pris : plutôt que de donner une chaîne de caractère pouvant comporter une erreur de saisie, le nom de l'interface est fourni par l'instruction CalculatriceFacadeRemote.class.getName(). Le compilateur ne fera pas de faute de frappe.

Surtout, ne pas oublier d'ajouter les bibliothèques nécessaires : les interfaces JavaEE et le run time du serveur d'application. Pour cela, cliquez avec le bouton droit sur Test Libraries dans le projet et sélectionnez Add Jar/Folder. Dans le répertoire lib du répertoire d'installation du serveur d'application <GLASSFISH_HOME>, sélectionnez les deux fichiers jar suivant : j2ee.jar et appserv-rt.jar.

Testez le bean, par l'option 'Run' > 'Test EJBModule2' (ou Alt + F6). Le test devrait être un succès.

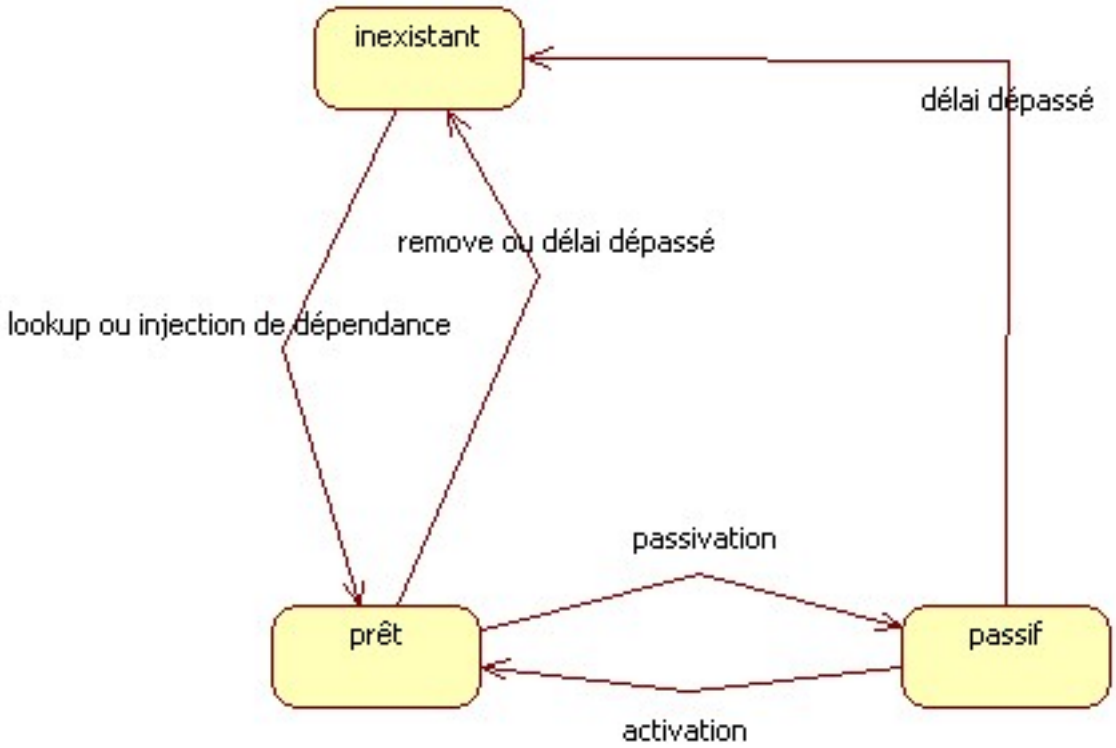
XI - Un bean à état dans tous ses états

Une dernière digression pour terminer sur les beans à état concernant leur cycle de vie. Le conteneur à pour mission d'optimiser la gestion mémoire et même de pouvoir transférer un bean d'un serveur à un autre. Ce dernier cas correspond à des serveurs travaillant en groupe avec gestion d'équilibrage de charge : il est possible dans les serveurs évolués de faire migrer automatiquement les beans avec leur état depuis les serveurs en surcharge vers les serveurs en sous activité de façon transparente.

Le cas le plus courant est lorsque la mémoire commence à saturer. Le serveur va écrire sur disque les beans peu utilisés ou déjà en mémoire depuis pas mal de temps. Au prochain appel du bean, le conteneur va récupérer le bean sur le disque et de réintégrer en mémoire. Seul un petit groupe de beans est réellement présent simultanément en mémoire et ils peuvent servir des centaines d'utilisateurs simultanés. L'action de transférer l'état sous une certaine forme de mémoire secondaire est appelée passivation, la retour en mémoire principale est appelé activation. Contrairement aux beans sans état qui peuvent être réutilisés indéfiniment et indistinctement à partir d'un groupe limité, les beans à état ont chacun leur propre identité et sont associés au client. Ils ne peuvent pas être interchangés, il n'y a recyclage que lorsque qu'un bean n'est plus utile.

Un bean à état donc 3 étapes dans son cycle de vie :

- le bean n'existe pas : il n'a pas d'existence ou est en attente de recyclage
- il est prêt : le bean est en mémoire et est prêt à accepter un appel
- il est passif : le bean a été mis en mémoire secondaire pour libérer des ressources mémoire



Cylce de vie d'un bean session à état

Nous pouvons remarquer dans ce diagramme d'état que le bean va disparaître après un certain délai ou si explicitement sa suppression est demandée par la méthode remove. Le diagramme est en réalité plus complexe, il existe aussi un état prêt en cours de transaction car un bean ne peu pas être passivé ou éliminé si son état fait partie d'une transaction. Il faut savoir qu'une autre caractéristique de Java Entreprise que nous n'avons pas abordée est la gestion des transactions. Le bean est inclu dans une transaction si une de ses méthodes est appelée lors d'une transaction, ceci jusqu'à la fin de la transaction.

Voir la démo

XII - Conclusion

Nous venons d'avancer encore de quelques pas et couvert les Beans entreprises de type session :

- nous pouvons prévoir des opérations réalisées en plusieurs étapes nécessitant de mémoriser l'état entre chaque appel
- nous pouvons utiliser des beans internes au conteneur qui apportent à Java EE l'efficacité depuis les EJB2 et la simplicité depuis les EJB3
- la notion d'injection de dépendance est souvent citée et une petite explication a pu éclairer ce terme

Ce ne sont que quelques aspects des beans entreprise et bien que cette partie soit un peu touffue, il reste encore beaucoup de chemin à parcourir. Certains détails ont été volontairement omis et les exemples sont peu réalistes mais il faut avancer pas à pas pour arriver au but.

Pour résumer les bean session, les tableaux suivant récapitulent les différentes annotations.

L'interface métier définit l'accès (à noter qu'un bean peut posséder les deux annotations) :

Chaque bean implémente l'interface métier et possède la capacité de conserver l'état conversationnel ou pas :