B

Annotations Seam

Nous présentons ici les principales annotations Seam, dont un certain nombre sont définies dans la spécification EJB 3.0.

Annotations au niveau composant

Les annotations suivantes permettent de définir un composant Seam. Elles apparaissent au niveau de la classe du composant.

@Name

Définit le nom du composant pour la classe (requise pour tous les composants Seam).

```
@Name (" MonComposant" )
```

@Scope

Définit le contexte par défaut du composant. Les valeurs possibles sont définies par l'énumération ScopeType, soit EVENT, PAGE, CONVERSATION, SESSION, BUSINESS_PROCESS , APPLICATION, STATELESS. Lorsque la portée n'est pas explicitement spécifiée, la valeur par défaut dépend du type de composant. Pour les beans session stateful, la valeur par défaut est CONVERSATION. Pour les composants JavaBeans la valeur par défaut EVENT.

```
@Scope(ScopeType.CONVERSATION)
```

@Role

Permet à un composant Seam d'être relié à plusieurs variables contexte.

Propriétés

- name : nom de la variable contexte.
- scope : portée de la variable contexte.

```
@Role (name=" monRole" , scope=ScopeType.SESSION)
```

@BypassInterceptors

Désactive tous les intercepteurs Seam sur un composant particulier ou une méthode d'un composant.

@JndiName

Spécifie que la portée du composant est conversationnelle, ce qui signifie qu'aucune méthode du composant ne peut être appelée à moins qu'une conversation longue ne soit active.

```
@JndiName("ejb/session/MonBeanSession")
Public class MonBeanSession{
}
```

@Startup

Utilisée en combinaison avec @Scope, spécifie qu'un composant de portée application est démarré immédiatement lors de l'initialisation de l'application (utile par exemple pour démarrer certaines ressources critiques, comme les sources de données).

Paramètres

- depends : spécifie le nom du composant qui doit être démarré en premier.

```
@Scope (SESSION) @Startup (depends= "org.jboss.seam.bpm.jbpm")
```

@Install

Spécifie si le composant doit être ou non installé par défaut. L'absence de cette annotation indique que le composant doit être installé.

Propriétés

- dependencies : spécifie que le composant doit être installé si les composants listés comme dépendants sont aussi installés.
- genericDependencies : spécifie qu'un composant doit être installé si un composant implémenté avec une classe particulière est installé.
- classDependencies : spécifie la précédence du composant. S'il existe plusieurs composants de même nom, c'est celui ayant la plus haute priorité qui est installé. Les valeurs de priorités sont les suivantes (ordre ascendant) :
  - BUILT_IN : priorité sur tous les composants Seam installés.
  - FRAMEWORK : priorité d'utilisation pour les composants des frameworks qui étendent Seam.
  - APPLICATION : priorité sur les composants de l'application (priorité par défaut).
  - DEPLOYMENT : priorité d'utilisation pour les composants qui surchargent les composants de l'application dans un déploiement spécifique.
  - MOCK : priorité pour les objets simulés (MOCK) utilisés pour les tests.

```
@Install(precedence=BUILT_IN)
```

@Synchronized

Spécifie qu'un composant est accédé de manière concurrente par plusieurs clients et que Seam doit sérialiser les requêtes. Si la requête est dans l'impossibilité d'obtenir un verrou sur le composant pendant la période spécifiée par le paramètre timeout, une exception est levée.

Propriétés

- timeout : durée en seconde de l'attente du verrou à partir de laquelle une exception est levée.

```
@Synchronized (timeout=2000)
```

@ReadOnly

Spécifie qu'un composant JavaBean ou une méthode d'un composant ne nécessite pas de réplication de l'état à la fin de l'invocation.

@AutoCreate

Spécifie qu'un composant sera automatiquement créé, même si le client ne spécifie pas create=true.

Annotations pour la bijection

@In

Spécifie que l'attribut d'un composant est injecté dans une variable de contexte au début de chaque invocation du composant. Si la variable de contexte est nulle, une exception est levée.

Propriétés

- value : spécifie le nom de la variable de contexte qui peut être substituée par une expression JSF EL, délimitée par le caractère #{…}.
- create : spécifie que Seam doit instancier le composant avec le même nom que la variable de contexte si cette dernière est indéfinie pour tous les contextes.
- required : spécifie que Seam doit lever une exception si la variable de contexte est indéfinie pour tous les contextes.

```
@In(required=false)
@In
private User user
```

@Out

Spécifie que l'attribut du composant est *outjected* vers une variable de contexte une fois la méthode appelée (principe d'injection de dépendances). La variable spécifiée par l'annotation dans le contexte est automatiquement substituée après chaque appel. Si l'attribut est nul, une exception est levée.

Propriétés

- value : spécifie le nom de la variable de contexte, par défaut le nom de l'attribut du composant.

- required : spécifie que l'attribut du composant Seam est *outjected* vers la variable de contexte à la fin de l'invocation. L'attribut peut être null (required=false) durant l'outjection.
- scope : spécifie qu'un attribut du composant qui n'est pas du type de composant Seam doit être outjected vers un scope spécifique à la fin de l'invocation. Si aucun scope n'est explicitement spécifié, le scope du composant avec l'attribut @Out est utilisé (ou le scope EVENT si le composant est stateless).

```
@In(create=true) @Out private User currentUser;
```

@Factory

Spécifie que la méthode du composant est utilisée pour initialiser la valeur de la variable de contexte spécifiée, lorsque la variable de contexte n'a pas de valeur.

Propriétés

- value : spécifie le nom de la variable de contexte, par défaut le nom de l'attribut du composant.
- required : spécifie que l'attribut du composant Seam est *outjected* vers la variable de contexte à la fin de l'invocation. L'attribut peut être null (required=false) durant l'outjection.
- scope : spécifie qu'un attribut du composant non-Seam doit être outjected vers un scope spécifique à la fin de l'invocation. Si aucun scope n'est explicitement spécifié, le scope du composant avec l'attribut @Out est utilisé (ou le scope EVENT si le composant est stateless).

```
@Factory("items")
    public void getItems() {
        if ((items==null)  || (index != firstItem) ){
            getNextItems();
        }
```

Annotations pour les méthodes du cycle de vie du composant

Ces annotations permettent au composant de réagir à son propre cycle de vie des événements sur les méthodes du composant.

@Create

Spécifie que la méthode doit être invoquée lorsqu'une instance du composant est instanciée par Seam. Les méthodes create sont uniquement applicables pour les JavaBeans et les beans session stateful.

```
    @Name("DatabaseUtils")
Public class DatabaseUtils{

    @Create()
    public void initDatabase(){
    }
}
```

@Destroy

Spécifie que la méthode doit être appelée lorsque le contexte se termine et que les variables de contexte associées au contexte sont supprimées. Les méthodes destroy sont uniquement supportées par les JavaBeans et les beans session stateful.

```
@Name("DatabaseUtils")
Public class DatabaseUtils{

    @Destroy()
    public void closeDbResources(){
    }
}
```

@Observer

Spécifie que la méthode doit être appelée lorsqu'un événement sur un composant survient.

Paramètres

- value : spécifie le nom de la variable de contexte, par défaut le nom de l'attribut du composant.
- create : si l'instance n'existe pas et que create est à false, l'événement produit est ignoré. Par défaut à true.

```
@Observer(value="ModificationEffectuee",create=false)
```

Annotations pour la démarcation du contexte

Ces annotations s'appliquent à la gestion déclarative du contexte conversationnel.

@Begin

Marque la méthode comme démarrant une conversation « longue » Seam.

Paramètres

- join : valeur true ou false. Spécifie que si une conversation longue est déjà en progression, le contexte conversationnel est simplement propagé. Si false (défaut), l'invocation de la méthode begin dans le scope d'une conversation en cours provoque une exception.
- Nested : valeur true ou false. Si actif (valeur true), et si une conversation est en cours, démarre une conversation imbriquée, au lieu de continuer dans le contexte de la conversation existante.
- Flushmode : positionne le mode flush pour chaque session Hibernate Seam managée ou contexte persistant JPA créé durant la conversation (par défaut AUTO).
- Pageflow : nom de la définition du processus jBPM définie pour le flux de la page pour une conversation particulière.

```
// Specify that this method starts a long running conversation
    @Begin
    public String find() {
        hotel = null;
        String searchPattern = searchString == null ? "%" : '%' +
        ➥searchString.toLowerCase().replace('*', '%') + '%';
```

```
        hotels = em.createQuery("from Hotel where lower(name) like :search or
        ➥lower(city) like :search or lower(zip) like :search or lower(address)
        ➥like :search")
                .setParameter("search", searchPattern)
                .setMaxResults(50)
                .getResultList();

        log.info(hotels.size() + " hotels found");

        return "main";
    }
```

@End

Spécifie qu'une conversation « longue » Seam se termine et supprime les variables de contexte associées.

Paramètres

- Beforeredirect : valeur true ou false. Par défaut (valeur=false), la conversation n'est pas réellement détruite, à moins qu'une redirection n'ait été effectuée. Sinon (valeur true), spécifie que la conversation est détruite à la fin de la requête courante et que la redirection est traitée dans un nouveau contexte de conversation temporaire.

```
@End @Destroy @Remove
    public String confirm() {
        if (booking == null || hotel == null) return "main";
        em.persist(booking);
        log.info("booking confirmed");
        return "confirmed";
    }
```

@StartTask

Démarre une tâche jPBM.

Paramètres

- taskIdParameter : nom du paramètre de la requête sous lequel localiser l'identifiant de la tâche qui doit être lancée.
- flushmode : positionne le mode flush pour n'importe quel gestionnaire d'entité utilisé pour cette conversation.

@BeginTask

Reprend la tâche jBPM précédemment démarrée.

Propriétés

- taskIdParameter : nom du paramètre de la requête sous lequel localiser l'identifiant de la tâche qui doit être reprise.
- flushmode : positionne le mode flush pour n'importe quel gestionnaire d'entité utilisé pour cette conversation.

@EndTask

Termine une tâche jBPM.

Propriétés

- transition : nom de la transition jBPM déclenchée lorsque la tâche est terminée (par défaut, nom de la transition par défaut).
- beforeRedirect (valeur true ou false) : permet de spécifier si la conversation est détruite avant toute redirection. Le comportement par défaut propage la conversation à travers le redirect et la détruit à la fin de la requête de redirection.

@CreateProcess

Crée une nouvelle instance de processus jBPM.

Propriétés

- definition : nom de la définition du processus jPBM déployé *via* org.jboss.seam.bpm.jbpm.processDefinitions.

@ResumeProcess

Entre à nouveau dans le scope d'une instance jBPM existante.

Propriétés

- processIdparameter : nom du paramètre de la requête sous lequel localiser l'identifiant du processus qui doit être repris.

@Transition

Spécifie une méthode pour la transition jBPM après que celle-ci retourne un résultat non null et sans exception.

Propriétés

- value : nom de la transition (par défaut, nom de la méthode).

```
@Transition("cancel")
```

Annotations pour l'utilisation des composants JavaBean Seam dans un environnement J2EE

Les annotations suivantes permettent de spécifier la stratégie de démarcation transactionnelle.

@Transactional

Spécifie qu'un composant JavaBean doit avoir un comportement transactionnel similaire au comportement par défaut d'un composant bean session. Les invocations de méthodes doivent prendre place dans la transaction. Si aucune transaction n'existe lorsque la méthode est invoquée, une transaction est démarrée juste pour cette méthode.

Propriétés

- value : type de programmation transactionnelle.

Annotations pour le support des exceptions

Ces annotations permettent de spécifier comment le framework Seam doit gérer une exception qui se propage en dehors d'un composant Seam.

@Redirect

Provoque un débranchement vers la page spécifiée par le paramètre viewId.

Propriétés

- viewId : spécifie l'identifiant de la page JSF pour la redirection. Vous pouvez également utiliser une expression JBoss EL (Expression Language).
- message : message qui sera affiché.
- end : spécifie qu'une conversation longue doit se terminer (par défaut, false).

```
@Redirect (viewId="error.jsp")
```

@HttpError

Provoque une erreur HTTP.

Propriétés

- errorcode : le code erreur http (par défaut, 500).
- message : message qui sera envoyé avec l'erreur HTTP (par défaut, le message d'exception).
- end : spécifie qu'une conversation longue doit se terminer (par défaut, false).

```
@HttpError (errorCode=404)
```

Annotations pour Seam Remoting

Le framework Seam fournit un mécanisme rapide et puissant pour l'accès aux composants à partir d'une page Web, en utilisant AJAX (Asynchronous Javascript and XML). Cela s'effectue avec le framework Seam Remoting, dont l'emploi nécessite que l'interface locale d'un bean session soit annotée avec @WebRemote.

@WebRemote

Spécifie que la méthode distante annotée peut être appelée à partir d'un client JavaScript.

Propriétés

- exclude : propriété optionnelle permettant aux objets d'être exclus du graphe d'objets produit.

```
@Local
  public interface HelloLocal {
    @WebRemote
    public String sayHello(String name);
  }
```

Annotations pour le support des intercepteurs Seam

L'annotation @Interceptor n'est pas une annotation Seam spécifique. Elle est utilisée par Seam pour mettre en œuvre les mécanismes de bijection, de validation et d'interception des invocations de composant.

@Interceptor

Met en œuvre le mécanisme d'interception Seam.

Propriétés

- Stateless : valeur true ou false. Spécifie que l'intercepteur est stateless.
- type : spécifie que l'intercepteur est de type client et qu'il est invoqué avant le conteneur EJB (type=CLIENT).
- around : spécifie l'ordre de précédence de l'intercepteur.
- within : spécifie que l'intercepteur est positionné plus profondément dans la pile que les intercepteurs spécifiés.

```
@Interceptor(around={unIntercepteur.class, unAutreIntercepteur.class})

@Interceptor(within={unIntercepteur.class, unAutreIntercepteur.class})
```

Annotations pour la gestion des méthodes asynchrones

L'annotation @Asynchronous est utilisée pour déclarer des méthodes asynchrones.

@Asynchronous

Spécifie que l'appel de la méthode est effectué de manière asynchrone. Ce mécanisme s'adapte parfaitement à l'utilisation du framework AJAX lorsque le client peut interroger le serveur de manière automatique pour recevoir le résultat d'une tâche fonctionnant en arrière-plan.

Propriétés

- Les annotations @Duration, @Expiration, @IntervalDuration s'utilisent conjointement avec @Asynchronous pour permettre une planification des méthodes asynchrones associées.

```
@Local
public interface PaymentHandler
{
    @Asynchronous
    public void processScheduledPayment(Payment payment, @Expiration Date date);

    @Asynchronous
    public void processRecurringPayment(Payment payment, @Expiration Date date,O
    ➥@IntervalDuration Date interval)'
}
```

Annotations pour l'utilisation avec JSF

Les annotations suivantes facilitent l'implémentation de listes sélectionnables qui s'appuient sur les beans session stateful.

@DataModel

Outjecte une propriété (la propriété devenant ainsi accessible aux variables JSF) de type List, Map, Set ou Objects, comme type DataModel JSF dans la portée du composant propriétaire (ou dans la portée event si le composant propriétaire est stateless). Dans le cas particulier d'un type Map, chaque ligne de DataModel est une Map.Entry.

Propriétés

- value : nom de la variable de contexte de conversation (par défaut, nom de l'attribut).
- scope : si scope=ScopeType.PAGE est explicitement spécifiée, DataModel est conservé dans le contexte PAGE.

```
@DataModel
private List<Book> books;
```

@DataModelSelection

Injecte la ligne de donnée sélectionnée de l'objet DataModel à utiliser conjointement avec l'annotation @DataModel.

Propriétés

- value : nom de la variable de contexte de conversation (non requis s'il existe une annotation @DataModel dans le composant).

```
@DataModel
private List<Book> bookList;
@DataModelSelection
Book book;
```

@DataModelSelectionIndex

Injecte la ligne de donnée sélectionnée de l'objet DataModel à utiliser conjointement avec l'annotation @DataModel.

Propriétés

- value : nom de la variable de contexte de conversation (non requis s'il existe une seule annotation @DataModel dans le composant).

```
@DataModel
private List<Book> bookList;
@DataModelSelection
Book book;
```

Annotations pour l'intégration avec le conteneur de servlets

Ces méta-annotations permettent d'implémenter des fonctionnalités similaires à @DataModel et @dataModelSelection pour les autres structures de données indépendamment des listes.

@DataBinderClass

Spécifie que l'annotation est de type databinding, c'est-à-dire qu'elle résulte de l'outjection d'une représentation « enveloppe » de la valeur de l'attribut du composant annoté.

```
@DataBinderClass(DataModelBinder.class)
```

@DataSelectorClass

Spécifie que l'annotation est une annotation de sélection de donnée, c'est-à-dire qu'elle résulte de l'injection de l'élément sélectionné de type de donnée databound.

```
@DataSelectorClass(DataModelSelector.class)
```

Annotations pour le packaging

Ces annotations fournissent un mécanisme de déclaration pour un ensemble de composants qui seront packagés ensemble. Peut être appliqué à chaque package Java.

@Namespace

Spécifie la configuration de l'espace de nom du package Java contenant l'ensemble des composants Seam.

Propriétés

- value : espace de nom de la configuration pour le packaging.
- prefix (en option) : spécifie le préfixe du nom du composant à appliquer pour les noms de composants spécifiés dans le fichier de configuration XML.

```
@Namespace(value="http://jboss.com/products/seam/core", prefix="org.jboss.seam.core")
```