

CDI

Olivier Cailloux

LAMSADE, Université Paris-Dauphine

Version du 25 octobre 2016

Utilité de l'injection

CDI ?

Utilité de l'injection

CDI ? Context Dependency *Injection*

Utilité de l'injection

CDI ? Context Dependency *Injection*

- Dépendance envers un service
- Pourquoi éviter de la coder en dur ?

Utilité de l'injection

CDI ? Context Dependency *Injection*

- Dépendance envers un service
- Pourquoi éviter de la coder en dur ?
- Tests : indépendance et robustesse ; câbler une alternative

Utilité de l'injection

CDI ? Context Dependency *Injection*

- Dépendance envers un service
- Pourquoi éviter de la coder en dur ?
- Tests : indépendance et robustesse ; câbler une alternative
- Comment éviter de la coder en dur ?

Utilité de l'injection

CDI ? Context Dependency *Injection*

- Dépendance envers un service
- Pourquoi éviter de la coder en dur ?
- Tests : indépendance et robustesse ; câbler une alternative
- Comment éviter de la coder en dur ?
- L'appelant indique la dépendance (référence dans constructeur...)

Utilité de l'injection

CDI ? Context Dependency *Injection*

- Dépendance envers un service
- Pourquoi éviter de la coder en dur ?
- Tests : indépendance et robustesse ; câbler une alternative
- Comment éviter de la coder en dur ?
- L'appelant indique la dépendance (référence dans constructeur...)
- MAIS crée une dépendance encombrante
- Et si classe créée automatiquement ? (exemple ?)

Utilité de l'injection

CDI ? Context Dependency *Injection*

- Dépendance envers un service
- Pourquoi éviter de la coder en dur ?
- Tests : indépendance et robustesse ; câbler une alternative
- Comment éviter de la coder en dur ?
- L'appelant indique la dépendance (référence dans constructeur...)
- MAIS crée une dépendance encombrante
- Et si classe créée automatiquement ? (exemple ? Servlet !)
- Solution Java EE : CDI
- Requiert gestion du cycle de vie

Vue d'ensemble

- Bean géré (*managed bean*) : classe dont le cycle de vie des instances peut être géré par le conteneur (y compris ressources)
- Pour ce faire, préciser sa portée contextuelle (*contextual scope*) : `@RequestScoped` ; `@SessionScoped` ; `@ConversationScoped` ; `@ApplicationScoped`...
- Conteneur crée *instance contextuelle* à la demande
- Instance contextuelle détruite à la fin de son scope
- Injection : `@Inject` sur champ ou méthode ou constructeur

Portées contextuelles

- `@RequestScoped` : détruit après traitement de la requête
- `@SessionScoped` : référence liée à la session HTTP ; détruit avec la session
- `@ApplicationScoped` : créé au besoin, un par application
- `@ConversationScoped` : gestion manuelle de la conversation
- `@Dependent` : pas un normal scope ; scope lié à la destination de l'injection
- Session et conversation-scoped : implémenter `Serializable`

Objets gestionnaires de contexte

- Le conteneur utilise un objet gestionnaire de contexte (*context object*) pour chaque contexte
- Exemple : quatre gestionnaires de contexte si le conteneur gère actuellement une application, une session, deux requêtes
- Chaque objet gestionnaire de contexte associe un type de bean à maximum une instance contextuelle
- Exemple : l'objet gestionnaire du contexte « Session 1 » associe le type `SayHelloService` à zéro instance
- Le gestionnaire du contexte crée une instance contextuelle de `SayHelloService` à la demande d'un servlet servant « Session 1 »
- Le gestionnaire du contexte sert *la même* instance à un autre servlet servant la même session
- À la destruction de la session, le gestionnaire du contexte détruit l'instance

Conversations

- `@ConversationScoped` : gestion manuelle de la conversation
- `Conversation` : objet `@RequestScoped` (donc injectable) pour démarrer et terminer une conversation
- Une conversation a un identifiant unique
- Joindre une conversation : envoyer paramètre de requête `cid`
- Voir application [JavaEE-Inject-Servlets-Conversation](#)

Portées et EJB

- Stateless session bean doit avoir portée `@Dependent`
- Singleton doit avoir `@ApplicationScoped` ou `@Dependent`
- Stateful : tout est permis

Portées et concurrence

- Le développeur doit protéger les objets dans un contexte contre les accès concurrents
- Sauf `ConversationScoped`, garanti par le conteneur actif pour une requête à la fois

Désambiguation et production

Comment préciser quelle classe injecter ?

- Un bean a des *qualifieurs* : annotations spécialisant un type
- Qualifieur : type annotation annoté `@Retention`
`RetentionPolicy.RUNTIME` et `@Qualifier`
- Préciser `@Inject @Qualifier Type truc;`
- Qualifieurs prédéfinis : `@Default` (tout type sans qualifieur ou slmt `@Named`), `@Any` (tout sauf `@New`)

Producteurs

- Méthode `@Produces` : produit une instance à la demande
- Méthode peut aussi être annotée d'un scope et de qualifieurs (s'appliquent au produit)

Divers

- `@Transactional` démarcation automatique de transaction
- On peut injecter : managed beans et Java EE Resources
- Ressource : bean représentant référence vers environnement composant (`@EJB`, `@Resource`, `@WebServiceRef`, `slmt scope @Dependent` standard)

Tests

- Comment faciliter les tests unitaires (par exemple) ?

Divers

- `@Transactional` démarcation automatique de transaction
- On peut injecter : managed beans et Java EE Resources
- Ressource : bean représentant référence vers environnement composant (`@EJB`, `@Resource`, `@WebServiceRef`, `slmt scope @Dependent` standard)

Tests

- Comment faciliter les tests unitaires (par exemple) ?
- Ajouter des méthodes `setTruc(Truc)`, visibilité package
- Test sans CDI : initialiser la classe avec `new`

Références

- The Java EE Tutorial: [Contexts and Dependency Injection](#)
- [JSR 346](#) (Context and Dependency Injection 1.1 et 1.2) ([direct](#)).
- [JSR 330](#) (Dependency Injection) ([direct](#)) : simplement `@Inject` et cie
- JSR 330, [section 4](#) : description courte de l'intérêt du DI
- [JSR 342](#) (Java EE 7) ([direct](#))
- [JSR 345](#) (EJB 3.2) ([direct](#))

Exercices I

- Programmer un bean `@RequestScoped`
- L'injecter et l'utiliser dans un servlet ou autre objet avec portée requête
- Programmer un bean `@RequestScoped` avec un état (un simple compteur, par exemple, qui s'incrémente lors de chaque utilisation)
- Observer son comportement lors de l'utilisation dans un servlet
- Rendre votre bean `@ApplicationScoped` ; observer ce que ça change
- Rendre votre bean `@SessionScoped` ; observer ce que ça change

Exercices II

- * Simuler un traitement long (`Thread.sleep`) dans le bean CDI. Avec deux navigateurs différents, envoyer deux GET simultanées. On souhaite qu'elles soient traitées en parallèle (la deuxième n'attend pas la fin de la première avant de s'exécuter). Comment annoter le bean CDI pour ce faire ? Quelle différence prévoyez-vous par exemple entre `@ApplicationScoped` et `@RequestScoped` ?
- * Pour le vérifier, modifiez le comportement de votre bean CDI. Il possède maintenant un compteur `i`. Il incrémente `i` en entrée d'appel, puis il renvoie la valeur de `i` à l'appelant, puis il décrémente `i`. Le bean renverra-t-il toujours la valeur d'initialisation de `i + 1`, lors d'appels parallèles de clients web différents ? Prédire son comportement en fonction de la portée CDI qui vous lui affectez. Réfléchir aux avantages de différents

Exercices III

choix. Pour vérifier vos prédictions, utiliser `Thread.sleep` judicieusement.

- Programmer un SLSB (stateless session bean), l'injecter avec `@EJB` : quelles différences avec un bean `@RequestScoped` ? Et quelles différences avec d'autres portées ?

Licence

Cette présentation, et le code LaTeX associé, sont sous [licence MIT](#). Vous êtes libres de réutiliser des éléments de cette présentation, sous réserve de citer l'auteur.
Le travail réutilisé est à attribuer à [Olivier Cailloux](#), Université Paris-Dauphine.