Entreprise Java Beans

Java EE 6 - Sommaire

- Introduction
- Comprendre les EJBs
  - Les types d'EJBs
  - Anatomie d'un EJB
  - Conteneur d'EJB
  - Conteneur embarqué
  - Injection de dépendance et JNDI
  - Méthodes de Callbacks et Interceptors
  - Packaging

Java EE 6 - Sommaire

- Les nouveautés des EJBs 3.1
  - EJB Lite
  - Implémentation de référence
- Exemple complet

Introduction

- JPA a permis des développer les objets du domaine, la couche de persistence, c'est à dire les données (les noms) (ex: livre, cd, artiste, ...)
- Mais les traitements et la logique métier, c'est à dire les actions (les verbes), ne sont dans ces objets, mais dans une couche séparée, appelée couche métier (ex: créer un livre, acheter un livre, imprimer une commande, ...)
- En Java EE, cette couche est implémentée en utilisant les Entreprise Java Beans (EJB)

Introduction

- La couche métier s'occupe notamment aussi des transactions et de la sécurité
- La couche métier intéragit souvent aussi avec des web services externes (SOAP ou RESTful), envoie des messages asynchrones à d'autres systèmes (via JMS), envoie des mails
- C'est donc la couche qui orchestre plusieurs systèmes, de la base de données à des systèmes externes, et sert de point central pour la sécurité et les transactions

Introduction

- La couche métier est donc le point d'entrée pour tout type de client, que ce soit une interface, un traitement batch ou un système externe

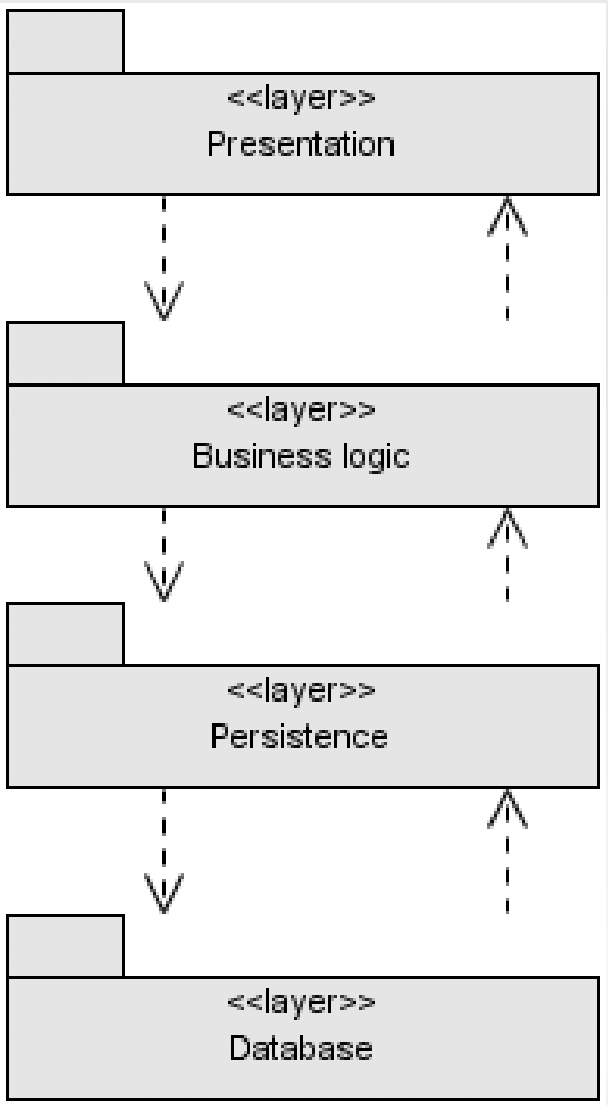
Comprendre les EJBs

- Les EJBs sont des composants serveur qui encapsulent la logique métier et gèrent les transactions et la sécurité
- Ils ont des facilités pour l'envoi de messages, la planification de tâches, les accès distants, la présentation de web services, l'injection de dépendance, la gestion du cycle de vie, l'AOP avec les intercepteurs, …

Comprendre les EJBs

- Ils s'intègrent aussi très simplement avec les autres technologies Java SE et Java EE comme JDBC, JavaMail, JPA, JTA, JMS, Java Authorization and Authorization API (JAAS), JNDI, RMI
- C'est pourquoi ils sont utilisés pour construire la couche métier en s'appuyant sur la couche de persistence

Comprendre les EJBs



Comprendre les EJBs

- Les EJBs sont devenus une modèle de composants à la fois très faciles à utiliser et très robustes
- La complexité de développement a disparu au profit de simple POJO avec des annotations déployés dans un conteneur
- Le conteneur d'EJBs s'occupe des services comme les transactions, la concurrence d'accès, le cache et la sécurité

Comprendre les EJBs

- Plus que jamais, avec la version 3.1, les EJBs peuvent être écrits un fois et déployés dans n'importe quel conteneur conforme à la spécification

Les types d'EJBs

- Pour gérer tous les cas de figure, plusieurs types d'EJBs et services ont été définis :
  - Session Beans : encapsulent la logique métier de haut niveau. Ils sont de plusieurs sortes :
    - Stateless : il n'y a pas d'état conservé entre deux appels
    - Stateful : entre deux appels, le client retrouve son état
    - Singleton : une seule instance partagée
  - Timer service : permettent de plannifier des tâches à date fixe ou de façon régulière pour les sessions beans

Les types d'EJBs

- Message Driven Beans : gèrent des messages asychrones reçus via JMS
- A noter que les EJBs peuvent aussi être utilisés comme web services

- A noter que la spécification 3.1 des EJBs continue de mentionner les EJBs de type Entity Beans. Mais ce mode de persistence est ”pruned” et pourrait disparaître de Java EE 7, puisque c'est JPA qui est désormais la technologie privilégiée pour la persistence

Anatomie d'un EJB

- Les session beans encapsulent la logique métier, sont transactionnels et leur conteneur gère le pooling, multithreading, la sécurité, …
- La façon la plus simple d'écrire un EJB de type session est de placer une annotation sur une simple classe Java (POJO)

Anatomie d'un EJB

```
@Stateless
public class BookEJB {

    @PersistenceContext(unitName = "chapter06PU")
    private EntityManager em;

    public Book findBookById(Long id) {
        return em.find(Book.class, id);
    }

    public Book createBook(Book book) {
        em.persist(book);
        return book;
    }
}
```

Anatomie d'un EJB

- Les précédentes version de JEE obligeaient à générer des artefacts, à définir des interfaces local ou remote (ou les deux), des interfaces local home ou remote home (ou les deux) ainsi qu'un fichier de déploiement
- Java EE 5 et les EJB 3.0 ont grandement simplifié le modèle puisque seulement une classe et une interface étaient nécessaires
- Java EE 6 et les EJB 3.1 vont plus loin puisqu'il n'y a plus besoin d'interface, seulement une classe annotée

Anatomie d'un EJB

- La simplicité a également été appliquée côté client. Il n'y a besoin que d'une annotation pour obtenir une référence sur un EJB en utilisant l'injection de dépendance. L'annotation permet au conteneur (client, web ou EJB) d'automatiquement injecter une référence sur l'EJB en question

Anatomie d'un EJB

```
public class Main {
    @EJB
    private static BookEJBRemote bookEJB;

    public static void main(String[] args) {
        Book book = new Book();
        book.setTitle("The Hitchhiker's Guide to the Galaxy");
        book.setPrice(12.5F);
        book.setDescription("Scifi book created by Douglas Adams");
        book.setIsbn("1-84023-742-2");
        book.setNbOfPage(354);

        bookEJB.createBook(book);
    }
}
```

Anatomie d'un EJB

- Comme la plupart des composants Java EE 6, les EJB utilisent des annotations pour informer le conteneur de choix ou demander l'injection de dépendances
- Le principe de configuration par l'exception permet de réduire le nombre d'annotations
- Les métadonnées peuvent pendre la forme d'annotations ou de fichier xml, via le fichier de déploiement ejb-jar.xml

Conteneur d'EJB

- Les EJBs sont des composants serveur devant s'exécuter dans un conteneur
- Cet environnement d'exécution fournit les fonctionnalités communes à la plupart des applications d'entreprise :
  - Communication distante : un EJB peut invoquer une méthode distante sans aucun code complexe
  - Injection de dépendance : le conteneur peut injecter plusieurs ressources dans un EJB (destinations et factories JMS, datasources, d'autres EJBs, ...)

Conteneur d'EJB

- Gestion de l'état : pour les session beans stateful, le conteneur s'occupe de gérer de façon transparente l'état
- Gestion du Pooling : un pool d'EJB de type session ou MDB permet de les affecter à une requête puis de les remettre dans le pool
- Gestion du cycle de vie : le conteneur instancie et gère le cycle de vie des instances
- Messages : le conteneur permet aux MDBs d'écouter des destinations et de consommer des messages

Conteneur d'EJB

- Gestion des transactions : un EJB peut définir le mode de transactions qu'il souhaite utiliser, le conteneur s'occupe du commit et rollback
- Gestion de la sécurité : les EJBs peuvent définir les droits associés à leur utilisation, le conteneur les fait respecter
- Gestion des accès concurrents : à part pour les singletons où une déclaration est nécessaire, les autre EJBs sont ”thread safe” par nature

Conteneur d'EJB

- Gestion des intercepteurs : on peut utiliser les mécanismes de l'AOP et le conteneur gère les appels
- Appels de méthodes asynchrones : avec les EJBs 3.1 il est maintenant possible de faire des appels de méthodes asynchrones sans utiliser de messages

Conteneur d'EJB

- En fait, on n'intéragit pas directement avec une instance d'un EJB, mais avec un proxy via le conteneur pour proposer tous les services. Ceci se fait de façon totalement transparente pour le client

Conteneur embarqué

- Les EJBs ont besoin d'un conteneur pour s'exécuter. Les serveurs d'applications contiennent ce conteneur (JBoss, GlassFish, ...), mais ils s'exécutent dans une JVM dédiée
- En mode de développement ou de test il peut être trop long ou trop compliqué d'interagir avec un processus séparé

Conteneur embarqué

- Pour résoudre ces problèmes, des implémentations de serveurs d'applications proposaient leur conteneur embarqué spécifique
- La spécification EJB 3.1 a défini un conteneur embarqué portable entre serveurs

Conteneur embarqué

- L'idée du conteneur embarqué est donc de permettre d'exécuter des applications à base d'EJB dans un environnement Java SE, utilisant ainsi la même JVM et le même class loader
- Ceci facilite les tests, les traitements de type batch et l'utilisation des EJBs dans des applications de type desktop
- L'API javax.ejb.embeddable fournit le même environnement d'exécution que le conteneur Java EE

Conteneur embarqué

- Voici comment créer un conteneur embarqué :

```
EJBContainer ec = EJBContainer.createEJBContainer();
Context ctx = ec.getContext();
BookEJB bookEJB = (BookEJB) ctx.lookup("java:global/BookEJB");
bookEJB.createBook(book);
```

Injection de dépendance et JNDI

- Dans le conteneur, les EJB peuvent utiliser l'injection de dépendance pour recevoir différentes ressources (EJBs, datasources, destinations JMS, …)

```
@EJB
private static BookEJB bookEJB;
```

- On peut aussi utiliser JNDI pour recherche une référence

```
Context ctx = new InitialContext();
BookEJB bookEJB = (BookEJB) ctx.lookup("java:global/chapter06/BookEJB");

java:global[/<app-name>]/<module-name>/<bean-name>[!<fully-qualified-➥
                                                    interface-name>]
```

Méthodes de Callbacks et Interceptors

- De la même façon qu'avec les entités, le conteneur peut appeler des méthodes annotées (@PostConstruct, @PreDestroy, …) quand l'état d'un EJB change. Grâce à ces méthodes, on peut initiliser des attributs, chercher des ressources avec JNDI, relâcher une connexion à une base, …
- Pour les problématiques transversales, on peut aussi utiliser des intercepteurs basés sur le modèle AOP

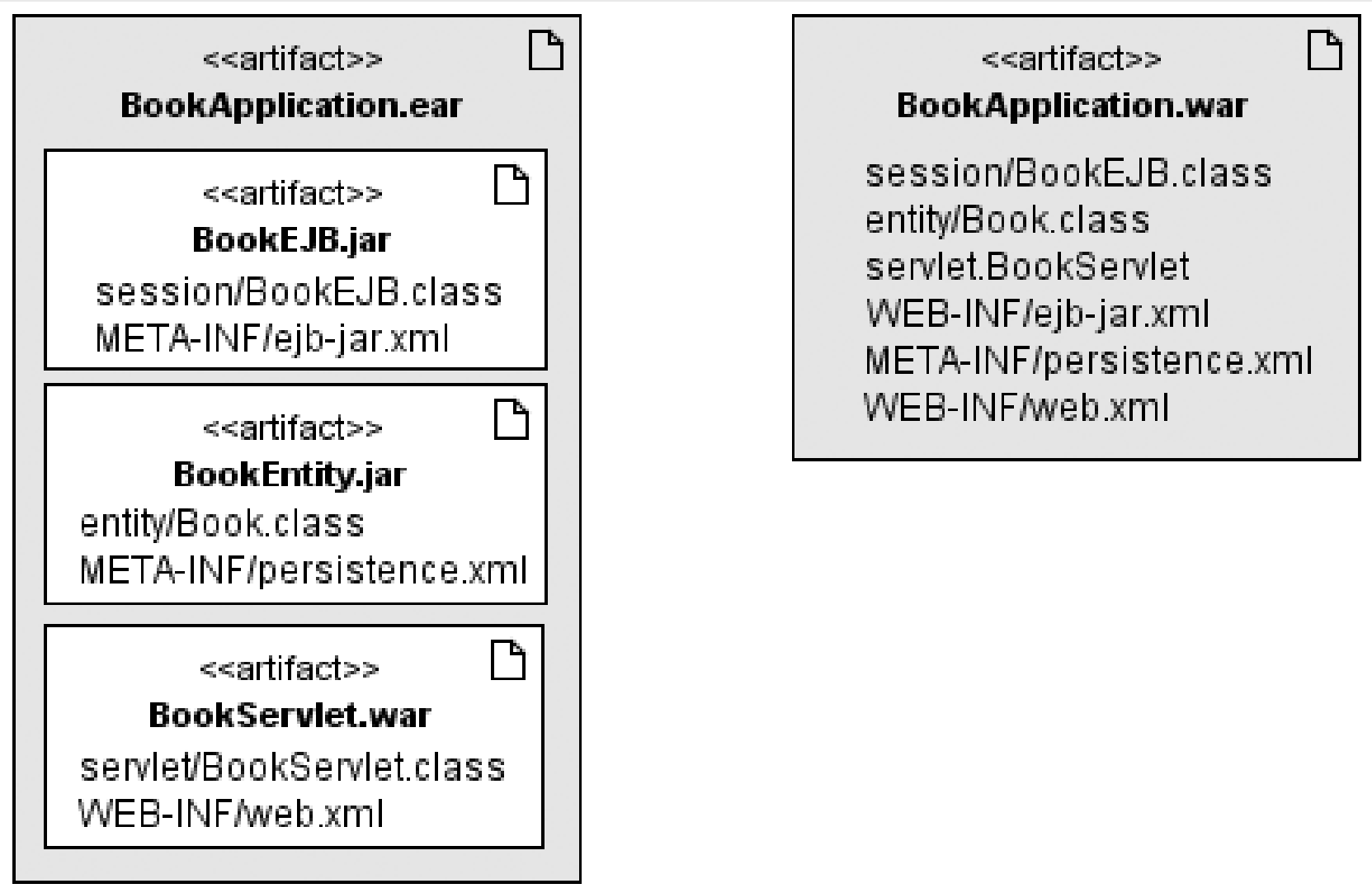
Packaging

- Comme la plupart des composants serveurs (servlets, pages JSF, web services, ...), les EJBs ont besoin d'être packagés avant de pouvoir être déployés dans un conteneur
- Dans la même archive, vous aurez les classes des beans, les interfaces, intercepteurs, super-classes et super-interfaces, exceptions, classes utilitaires et un fichier de description optionnel ejb-jar.xml

Packaging

- Une fois l'archive jar créée, les EJBs peuvent être déployés dans un conteneur
- Une autre option consiste à embarquer le jar dans un ear (archive d'entreprise) et de déployer l'ear
- Depuis la version 3.1, les EJBs peuvent aussi être déployés directement dans un module web (une archive war). Le fichier ejb-jar.xml n'est alors plus dans META-INF/ejb-jar.xml mais dans WEB-INF/ebj-jar.xml

Packaging


<<artifact>>
BookApplication.ear
<<artifact>>
BookEJB.jar
session/BookEJB.class
META-INF/ejb-jar.xml
<<artifact>>
BookEntity.jar
entity/Book.class
META-INF/persistence.xml
<<artifact>>
BookServlet.war
servlet/BookServlet.class
WEB-INF/web.xml
<<artifact>>
BookApplication.war
session/BookEJB.class
entity/Book.class
servlet.BookServlet
WEB-INF/ejb-jar.xml
META-INF/persistence.xml
WEB-INF/web.xml

Les nouveautés des EJBs 3.1

- Pas d'interface : les sessions beans accédés en local n'ont plus besoin d'interface locale
- Déploiement war : il est possible de déployer directement les ejbs dans des war
- Conteneur embarqué : une nouvelle API définit comment exécuter des EJBs avec Java SE
- Singleton : une nouvelle façon de partager des données ou traitements
- Asynchrone : il est maintenant possible de faire des appels asynchrones sans utiliser les MDBs

Les nouveautés des EJBs 3.1

- EJB Lite : un ensemble limité des fonctionnalités pour les profils (Web Profile)
- Nom JNDI portable : la syntaxe de recherche des EJBs est maintenant spécifiée

EJB Lite

- Pour des raisons de compatibilité, la spécification des EJBs 3.1 continue d'inclure les vieux entity beans, interfaces home, EJB QL et autres fonctionnalités remplacées par d'autres spécifications. Du coup, les serveurs d'applications compatibles doivent aussi implémenter ces anciennes technologies, alourdissant la compréhension pour les développeurs
- Pour cela, la spécification définit un ensemble minimal des API appelé EJB Lite

EJB Lite

- EJB Lite inclut l'ensemble des fonctionnalités nécessaires pour construire une application portable, transactionnelle et sécurisée
- Une application EJB Lite peut être déployée dans n'importe quel serveur Java EE qui implémente EJB 3.1

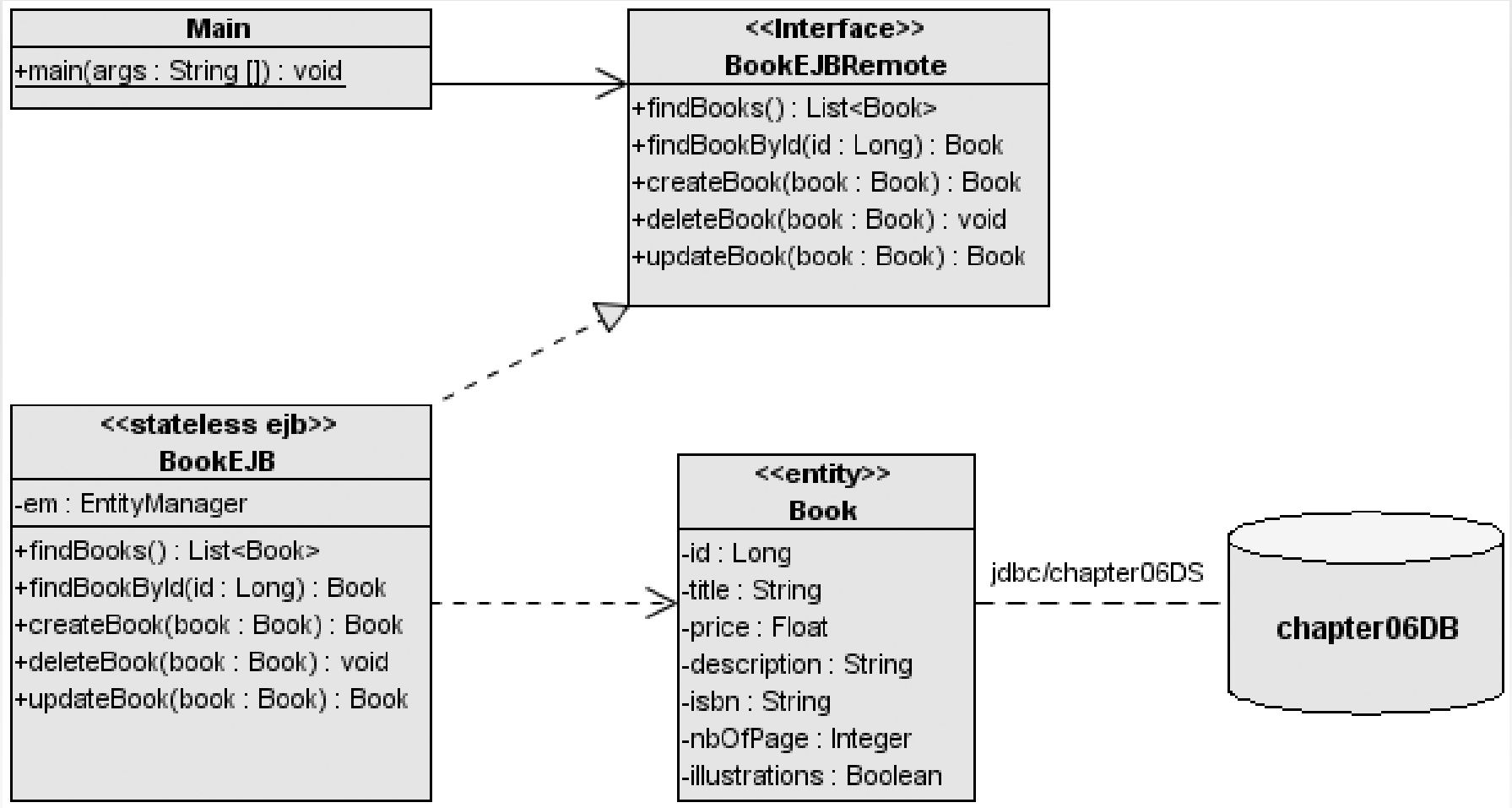
EJB Lite

Feature	EJB Lite	Full EJB 3.1
Session beans (stateless, stateful, singleton)	Yes	Yes
MDBs	No	Yes
Entity beans 1.*x*/2.*x*	No	Yes (proposed for pruning)
No-interface view	Yes	Yes
Local interface	Yes	Yes
Remote interface	No	Yes
2.*x* interfaces	No	Yes (proposed for pruning)
JAX-WS web services	No	Yes
JAX-RS web services	No	Yes
JAX-RPC web services	No	Yes (proposed for pruning)
Timer service	No	Yes
Asynchronous calls	No	Yes
Interceptors	Yes	Yes
RMI/IIOP interoperability	No	Yes
Transaction support	Yes	Yes
Security	Yes	Yes
Embeddable API	Yes	Yes

Implémentation de référence

- GlassFish est devenu l'implémentation de référence de Java EE 5 en 2006
- Aujourd'hui, GlassFish v3 est l'implémentation de référence pour les EJBs 3.1

Exemple complet



Exemple complet

- Nous devrons créer une datasource (jdbc/chapter06DS) qui sera utilisée par les entités
- Les fichiers seront rangés ainsi :
  - src/main/java : Book entité, BookEJB, BookEJBRemote interface, Main class
  - src/main/resources : persistence.xml
  - src/test/java : BookTest
  - src/test/resources : persistence.xml
  - pom.xml

Exemple complet

- Book entité

```
@Entity
@NamedQuery(name = "findAllBooks", query = "SELECT b FROM Book b")
public class Book {
    @Id @GeneratedValue
    private Long id;
    @Column(nullable = false)
    private String title;
    private Float price;
    @Column(length = 2000)
    private String description;
    private String isbn;
    private Integer nbOfPage;
    private Boolean illustrations;
    // Constructors, getters, setters
}
```

Exemple complet

- BookEJB : stateless session bean

```
@Stateless
public class BookEJB implements BookEJBRemote {
    @PersistenceContext(unitName = "chapter06PU")
    private EntityManager em;
    public List<Book> findBooks() {
        Query query = em.createNamedQuery("findAllBooks");
        return query.getResultList();
    }
    public Book findBookById(Long id) {
        return em.find(Book.class, id);
    }
    public Book createBook(Book book) {
        em.persist(book);
        return book;
    }
    public void deleteBook(Book book) {
        em.remove(em.merge(book));
    }
    public Book updateBook(Book book) {
        return em.merge(book);
    }
}
```

Exemple complet

- Remote interface : permet un accès distant à l'EJB

```
@Remote
public interface BookEJBRemote {
    public List<Book> findBooks();
    public Book findBookById(Long id);
    public Book createBook(Book book);
    public void deleteBook(Book book);
    public Book updateBook(Book book);
}
```

Exemple complet

- Persistence unit du BookEJB

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence">
  <persistence-unit name="chapter06PU" transaction-type="JTA">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <jta-data-source>jdbc/chapter06DS</jta-data-source>
    <class>com.apress.javaee6.chapter06.Book</class>
    <properties>
      <property name="eclipselink.ddl-generation" ➥
              value="drop-and-create-tables"/>
      <property name="eclipselink.logging.level" value="INFO"/>
    </properties>
  </persistence-unit>
</persistence>
```

Exemple complet

- Main class

```
public class Main {
    @EJB
    private static BookEJBRemote bookEJB;
    public static void main(String[] args) {
            Book book = new Book();
            book.setTitle("The Hitchhiker's Guide to the Galaxy");
            book.setPrice(12.5F);
            book.setDescription("Scifi book created by Douglas Adams");
            book.setIsbn("1-84023-742-2");
            book.setNbOfPage(354);
            book.setIllustrations(false);
            bookEJB.createBook(book);
            book.setTitle("H2G2");
            bookEJB.updateBook(book);
            bookEJB.deleteBook(book);
    }
}
```

Exemple complet

- Compilation et packaging avec Maven

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" ➥
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"➥
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 ➥
        http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId> com.apress.javaee6</groupId>
    <artifactId>chapter06</artifactId>
    <packaging>jar</packaging>
    <version>1.0</version>
    <name>chapter06</name>
```

Exemple complet

- Compilation et packaging avec Maven

```
<repositories>
   <repository>
      <id>maven-repository.dev.java.net</id>
      <name>Java.net Repository for Maven 1</name>
      <url>http://download.java.net/maven/1/</url>
      <layout>legacy</layout>
   </repository>
   <repository>
      <id>maven2-repository.dev.java.net</id>
      <name>Java.net Repository for Maven 2</name>
      <url>http://download.java.net/maven/2/</url>
      <layout>default</layout>
   </repository>
   <repository>
      <id>glassfish-maven-repository</id>
      <name>GlassFish Maven Repository</name>
      <url>http://maven.glassfish.org/content/groups/glassfish/</url>
   </repository>
   <repository>
      <id>EclipseLink Repo</id>
      <name>Eclipse maven repository http://eclipse.ialto.org/rt/eclipselink/maven.repo/</name>
      <url>http://www.eclipse.org/downloads/download.php?r=1&amp;amp;nf=1&amp;amp;file=/rt/eclipselink/maven.repo</url>
   </repository>
</repositories>
<pluginRepositories>
   <pluginRepository>
      <id>maven2-repository.dev.java.net</id>
      <url>http://download.java.net/maven/2/</url>
   </pluginRepository>
</pluginRepositories>
```

Exemple complet

- Compilation et packaging avec Maven

```
<dependencies>
    <dependency>
        <groupId>org.glassfish.extras</groupId>
        <artifactId>glassfish-embedded-all</artifactId>
        <version>3.0</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>javax</groupId>
        <artifactId>javaee-api</artifactId>
        <version>6.0</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>org.apache.derby</groupId>
        <artifactId>derbyclient</artifactId>
        <version>10.5.3.0_1</version>
    </dependency>
    <dependency>
        <groupId>org.apache.derby</groupId>
        <artifactId>derby</artifactId>
        <version>10.5.3.0_1</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.5</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Exemple complet

- Compilation et packaging avec Maven

```
<build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>2.0.2</version>
        <configuration>
          <source>1.6</source>
          <target>1.6</target>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jar-plugin</artifactId>
        <version>2.2</version>
        <configuration>
          <archive>
            <manifest>
              <mainClass>fr.univmed.jeecourse.jeeexample.Main</mainClass>
            </manifest>
          </archive>
        </configuration>
      </plugin>
    </plugins>
</build>
```

mvn package → chapter06-1.0.jar

Exemple complet

- Déploiement sur GlassFish
  - Démarrer GlassFish et Derby
  - Création de la datasource jdbc/chapter06DS

```
asadmin create-jdbc-connection-pool ➥
--datasourceclassname=org.apache.derby.jdbc.ClientDataSource ➥
--restype=javax.sql.DataSource ➥
--property portNumber=1527:password=APP:user=APP:serverName=localhost:➥
databaseName=chapter06DB:connectionAttributes=;create\\=true Chapter06Pool

asadmin ping-connection-pool Chapter06Pool

asadmin create-jdbc-resource --connectionpoolid Chapter06Pool jdbc/chapter06DS
(asadmin list-jdbc-resources pour lister les datasources)
```

  - Déploiement

```
asadmin deploy --force=true target\chapter06-1.0.jar
```

Exemple complet

- Lancement du Main dans l'ACC (Application Client Container)

```
appclient -client chapter06-1.0.jar
```

Exemple complet

- Classe de test BookEJBTest

```
public class BookEJBTest {
    private static EJBContainer ec;
    private static Context ctx;
    @BeforeClass
    public static void initContainer() throws Exception {
        ec = EJBContainer.createEJBContainer();
        ctx = ec.getContext();
    }
    @AfterClass
    public static void closeContainer() throws Exception {
        ec.close();
    }
    @Test
    public void createBook() throws Exception {
        // Creates an instance of book
        Book book = new Book();
        book.setTitle("The Hitchhiker's Guide to the Galaxy");
        book.setPrice(12.5F);
        book.setDescription("Science fiction comedy book");
        book.setIsbn("1-84023-742-2");
        book.setNbOfPage(354);
        book.setIllustrations(false);
        // Looks up the EJB
        BookEJBRemote bookEJB = (BookEJBRemote) ➥
                ctx.lookup("java:global/ chapter06/BookEJBRemote");
        // Persists the book to the database
        book = bookEJB.createBook(book);
        assertNotNull("ID should not be null", book.getId());
        // Retrieves all the books from the database
        List<Book> books = bookEJB.findBooks();
        assertNotNull(books);
    }
}
```

Exemple complet

- Lancement des tests

```
mvn test

[INFO] ------------------------------------------------------------
[INFO]  BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------
[INFO] Total time: 26 seconds
[INFO] Finished
[INFO] Final Memory: 4M/14M
[INFO] ------------------------------------------------------------
```