

Java EE

Java EE

Présentation

Standard de déploiement

Ressources de serveur d'applications

Composants

- EJB Session

- EJB Message-Driven

- Entités et Unités de persistance

Services

- Transactions

- Timers

Java EE

Java Enterprise Edition (vs Java Standard Edition)

Anciennement J2EE

Nouvellement JakartaEE

Norme définissant une infrastructure pour héberger des applications distribuées

JSR (Java Specification Request) : Spécifications

Java EE – Historique

J2EE 1.2 – 1999

J2EE 1.3 – 2001 (JSR 58)

J2EE 1.4 – 2003 (JSR 151)

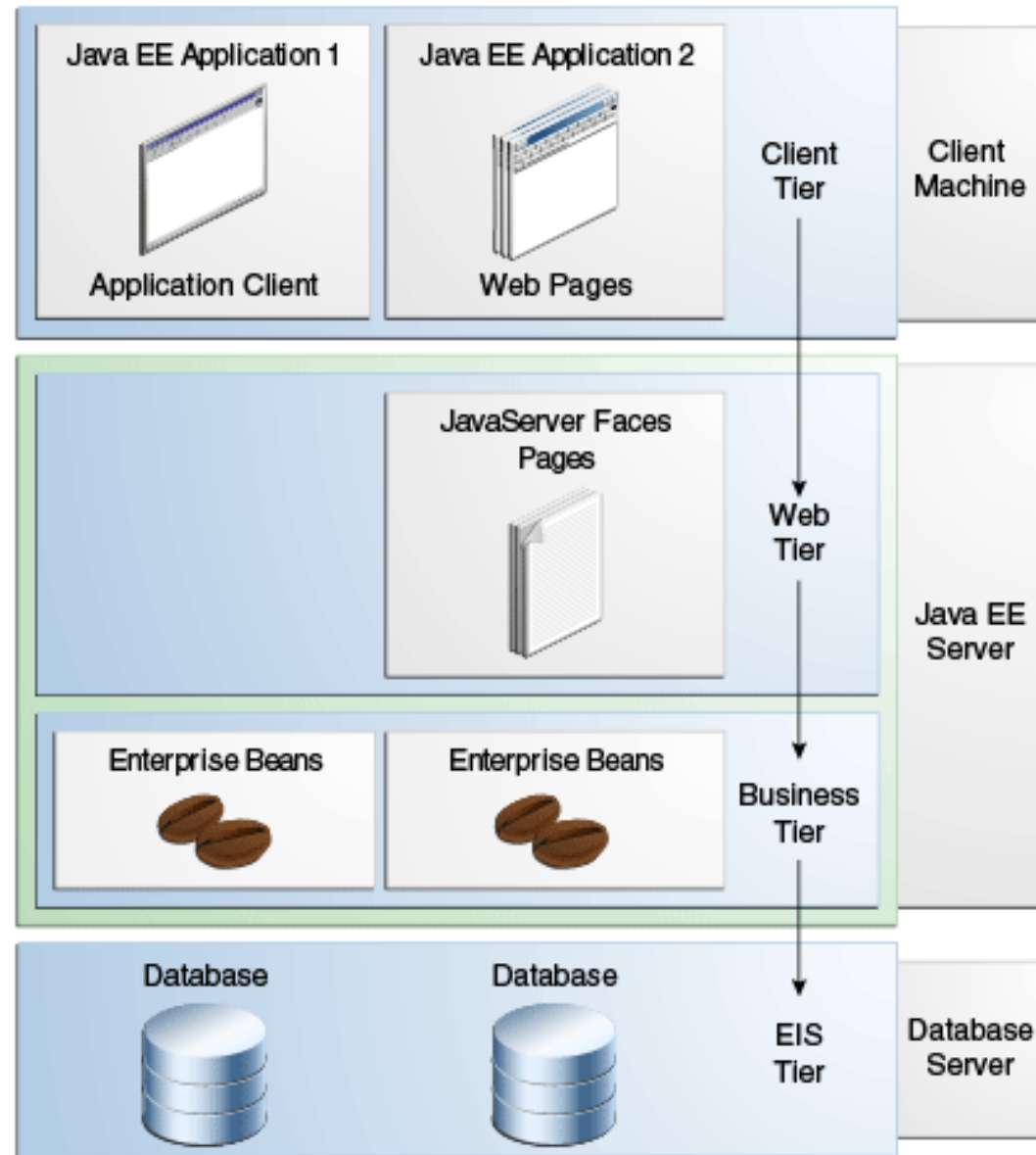
Java EE 5 – 2006 (JSR 244)

Java EE 6 – 2009 (JSR 316)

Java EE 7 – 2013 (JSR 342)

Java EE 8 – 2017 (JSR 366)

Java EE – Applications distribuées multi-tiers



Java EE 7 – Full Profile

WebSocket	JSON-P	JBatch	Concurrency	DI	JAX-RS	JACC	JASPIC
EL	JSF	BV	EJB	CDI	Web Services Metadata	JAX-WS	Debugging
JSP	Servlet	JPA	Interceptors	JTA	JAX-RPC (opt.)	EWS	Application Deployment (opt.)
JSTL	JavaMail	JMS	Common Annotations	JCA	JAXR	JAXM	Management
<div>JAXB JAXP JDBC JMX JAF StAX</div>							

Java EE 7 – Web Profile

WebSocket	JSON-P	JBatch	Concurrency	DI	JAX-RS	JACC	JASPIC
EL	JSF	BV	EJB	CDI	Web Services Metadata	JAX-WS	Debugging
JSP	Servlet	JPA	Interceptors	JTA	JAX-RPC (opt.)	EWS	Application Deployment (opt.)
JSTL	JavaMail	JMS	Common Annotations	JCA	JAXR	JAXM	Management
<div>JAXB JAXP JDBC JMX JAF StAX</div>							

Java EE 8

Java Servlet 4

JavaServer Pages (JSP) 2.3

JavaServer Standard Tag Library (JSTL) 1.2

JavaServer Faces (JSF) 2.3

Enterprise JavaBean (EJB) 3.2

Java Message Service API (JMS) 2.1

Java Persistence API (JPA) 2.2

Context and Dependency Injection (CDI) 2.0

Java API for RESTfull Web Services (JAX-RS) 2.1

Java API for XML-based Web Services (JAX-WS) 2.2

Java API for JSON Processing (JSON-P) 1.1

Java API for Websocket 1.1

Java API for JSON Binding (JSON-B) 1.0

Java EE

Présentation

Standard de déploiement

Ressources de serveur d'applications

Composants

- EJB Session

- EJB Message-Driven

- Entités et Unités de persistance

Services

- Transactions

- Timers

Standard de déploiement

EAR : top archive level

- 1 ou + modules EJB (.jar)

- 1 ou plus modules WEB (.war)

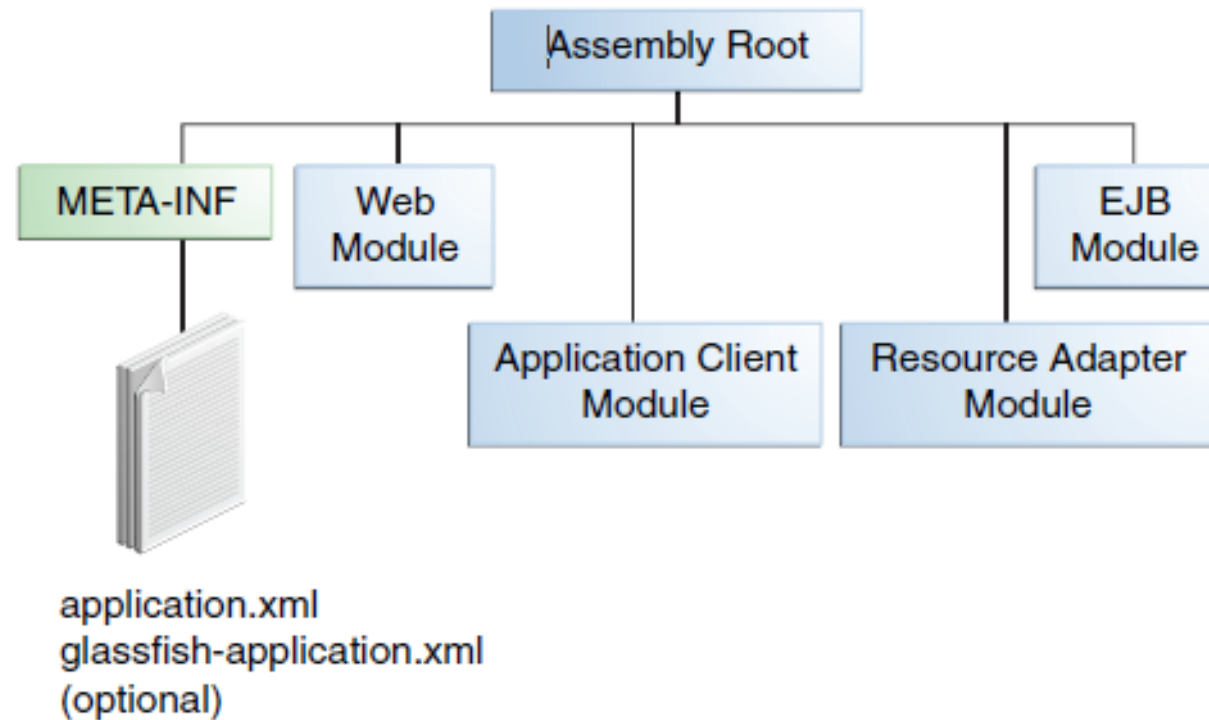
- 1 descripteur de déploiement

 - (application.xml sous META-INF)

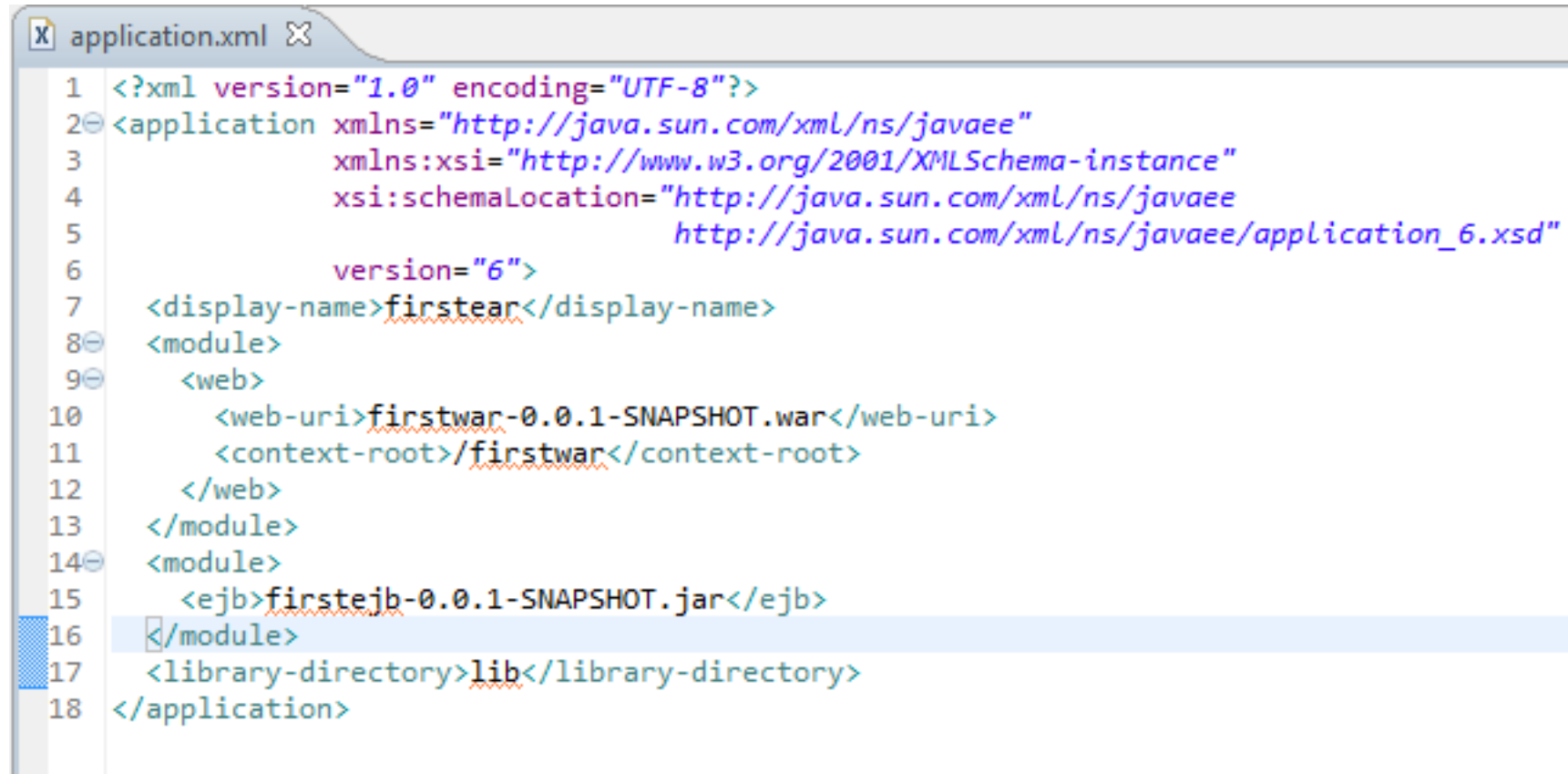
- 1 répertoire pour les lib communes aux ejb et aux war (souvent lib)

Standard de déploiement

FIGURE 1-6 EAR File Structure



Standard de déploiement



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <application xmlns="http://java.sun.com/xml/ns/javaee"
3             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4             xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5                                 http://java.sun.com/xml/ns/javaee/application_6.xsd"
6             version="6">
7   <display-name>firstear</display-name>
8   <module>
9     <web>
10      <web-uri>firstwar-0.0.1-SNAPSHOT.war</web-uri>
11      <context-root>firstwar</context-root>
12    </web>
13  </module>
14  <module>
15    <ejb>firstejb-0.0.1-SNAPSHOT.jar</ejb>
16  </module>
17  <library-directory>lib</library-directory>
18 </application>
```

Standard de déploiement

EJB JAR : archive jar

- 1 ou + EJBs

- 1 descripteur de déploiement

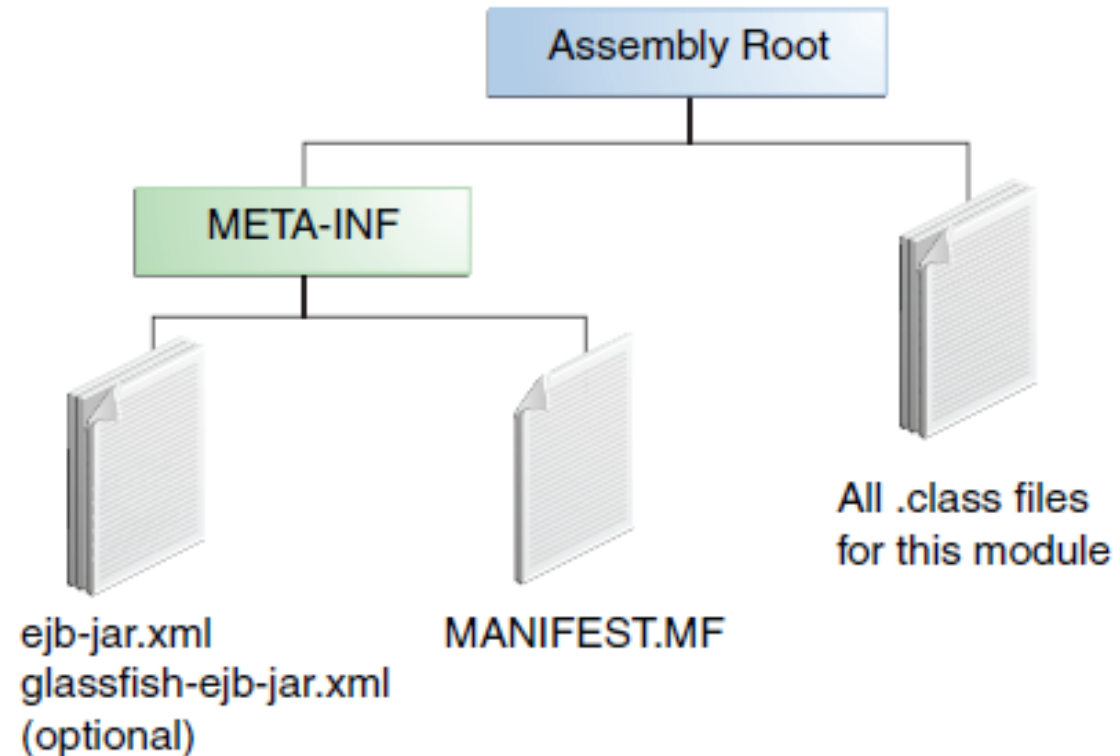
 - (ejb-jar.xml sous META-INF)

- 0 ou 1 définition de persistance unit

 - (persistence.xml sous META-INF)

Standard de déploiement

FIGURE 22-2 Structure of an Enterprise Bean JAR



Standard de déploiement

WAR : archive war

Composants web (Servlets, JSP, JSF...)

1 descripteur de déploiement

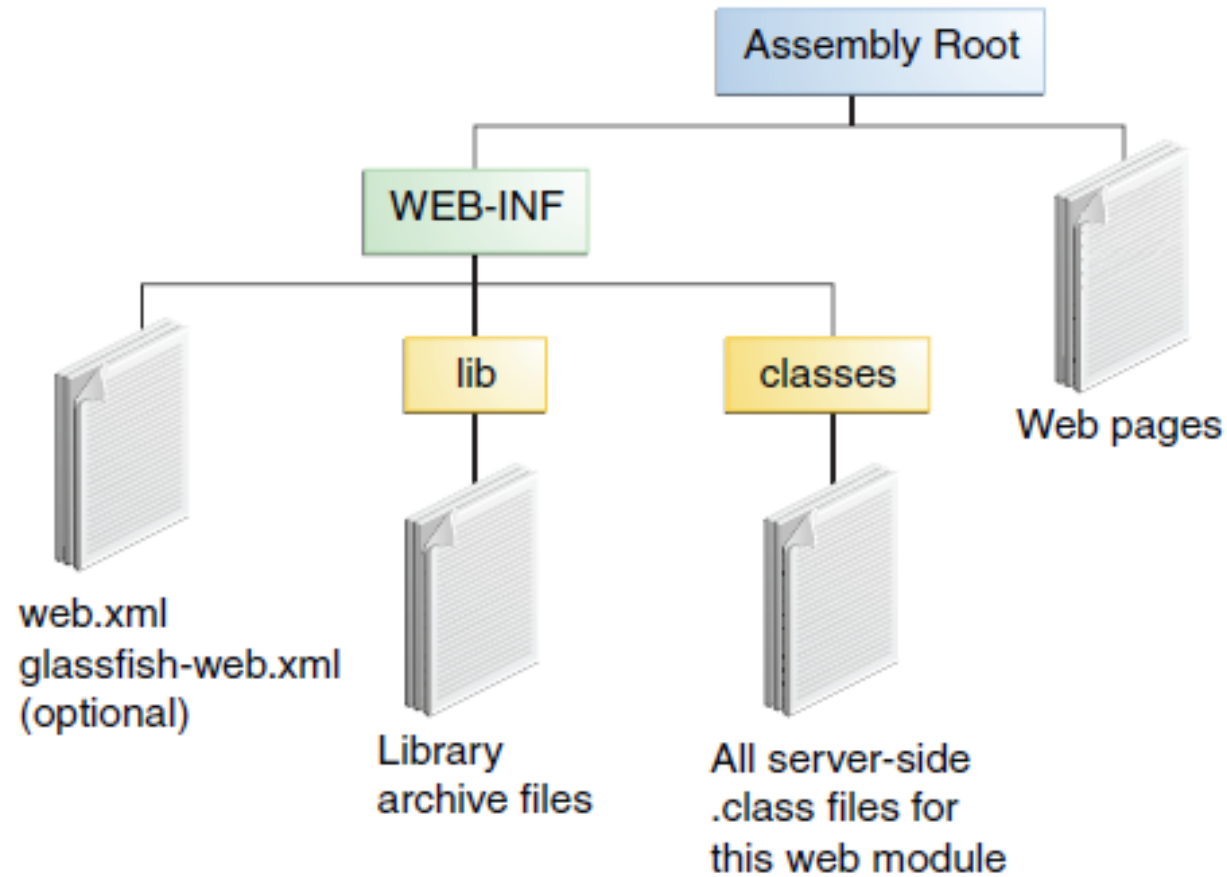
(web.xml sous WEB-INF)

Depuis Java EE 6, les war peuvent aussi contenir des EJBs...

Les Unités de persistance peuvent être packagées dans le war

Standard de déploiement

FIGURE 3-2 Web Module Structure



Standard de déploiement

Impact pour les projets Maven

- 1 projet EAR (packaging = ear)

- 1 projet EJB (packaging = ejb)

- 1 projet WEB (packaging = war)

Standard de déploiement

Le projet EAR :

Référence le projet ejb et le projet web comme module

Les déclare comme dépendances (de type ejb et war)

Standard de déploiement

Le projet EJB :

Déclare une dépendance de scope provided vers l'API JavaEE

Le projet WAR :

Déclare une dépendance de scope provided vers l'API JavaEE
(Web ou Full)

Déclare une dépendance de scope provided vers le projet EJB

Standard de déploiement

TP

Créer un projet Maven à partir de l'archetype 'wildfly-javaee7-webapp-ear-archetype'

```
mvn archetype:generate -Dfilter=wildfly
```

Les importer dans votre IDE

Installer Wildfly

Déployer l'ear sur le Server Wildfly

Tester

Recommencer avec l'archetype 'wildfly-javaee7-webapp-ear-blank-archetype'

Supprimer le dossier 'src' du projet *-ear

Java EE

Présentation

Standard de déploiement

Ressources de serveur d'applications

Composants

- EJB Session

- EJB Message-Driven

- Entités et Unités de persistance

Services

- Transactions

- Timers

Ressources

Déclaration de ressources dans le serveur d'application

Via la console d'administration

DataSourcees

Sessions JavaMail

Ressources JMS

Connection factories

Destination

Ressources

Accès à ces ressources dans le code

Par injection

Via JNDI

Ressources

TP

Se connecter à la console d'administration de Wildfly

Déclarer un pool de connection vers la base de données Mysql

Déclarer une datasource JNDI sur cette pool et l'utiliser dans votre projet

Java EE

Présentation

Standard de déploiement

Ressources de serveur d'applications

Composants

- EJB Session

- EJB Message-Driven

- Entités et Unités de persistance

Services

- Transactions

- Timers

Enterprise Beans

EJB ???

➔ Composants distribués transactionnels !

Enterprise Beans

EJB ???

➔ Composants distribués transactionnels !

Unité logicielle autonome appliquant de la logique business

Enterprise Beans

EJB ???

➔ Composants distribués transactionnels !

Le client de l'EJB et l'EJB sont possiblement sur 2 machines physiques distinctes

Un EJB pourra être retrouvé à distance via son nom JNDI

Enterprise Beans

EJB ???

➔ Composants distribués transactionnels !

L'intégrité des données accédées par l'EJB est garantie par le conteneur

Java EE

Présentation

Standard de déploiement

Ressources de serveur d'applications

Composants

EJB Session

EJB Message-Driven

Entités et Unités de persistance

Services

Transactions

Timers

EJB Session

Composant encapsulant de la logique business

Par ex. l'implémentation d'un processus, d'un traitement

*Ex: EJB StudentManager, proposant une méthode
createStudent(String name)*

EJB Session

Stateless : pas d'état conversationnel avec le client

Statefull : état conversationnel avec le client

Singleton : une seule instance

EJB Session Stateless (No Interface)

```
package converter.ejb;

import java.math.BigDecimal;
import javax.ejb.*;

@Stateless
public class ConverterBean {
    private BigDecimal yenRate = new BigDecimal("83.0602");
    private BigDecimal euroRate = new BigDecimal("0.0093016");

    public BigDecimal dollarToYen(BigDecimal dollars) {
        BigDecimal result = dollars.multiply(yenRate);
        return result.setScale(2, BigDecimal.ROUND_UP);
    }

    public BigDecimal yenToEuro(BigDecimal yen) {
        BigDecimal result = yen.multiply(euroRate);
        return result.setScale(2, BigDecimal.ROUND_UP);
    }
}
```

EJB Session Client (with injection)

```
@WebServlet  
public class ConverterServlet extends HttpServlet {  
    @EJB  
    ConverterBean converterBean;  
    ...  
}
```

EJB Session Statefull (Remote Interface)

```
package cart.ejb;

import cart.util.BookException;
import java.util.List;
import javax.ejb.Remote;

@Remote
public interface Cart {
    public void initialize(String person) throws BookException;
    public void initialize(String person, String id)
        throws BookException;
    public void addBook(String title);
    public void removeBook(String title) throws BookException;
    public List<String> getContents();
    public void remove();
}
```

EJB Session Statefull (Impl class)

```
package cart.ejb;

import cart.util.BookException;
import cart.util.IdVerifier;
import java.util.ArrayList;
import java.util.List;
import javax.ejb.Remove;
import javax.ejb.Stateful;

@Stateful
public class CartBean implements Cart {
    String customerName;
    String customerId;
    List<String> contents;

    public void initialize(String person) throws BookException {
        if (person == null) {
            throw new BookException("Null person not allowed.");
        } else {
            customerName = person;
        }

        customerId = "0";
        contents = new ArrayList<String>();
    }
}
```

EJB Session

TP

Créer un EJB Session Stateless avec Local Interface dans le module EJB

Créer un webservice REST dans le module Web

Injecter l'EJB dans le webservice

Appeler depuis le webservice une méthode déclarée dans l'EJB

Java EE

Présentation

Standard de déploiement

Ressources de serveur d'applications

Composants

EJB Session

EJB Message-Driven

Entités et Unités de persistance

Services

Transactions

Timers

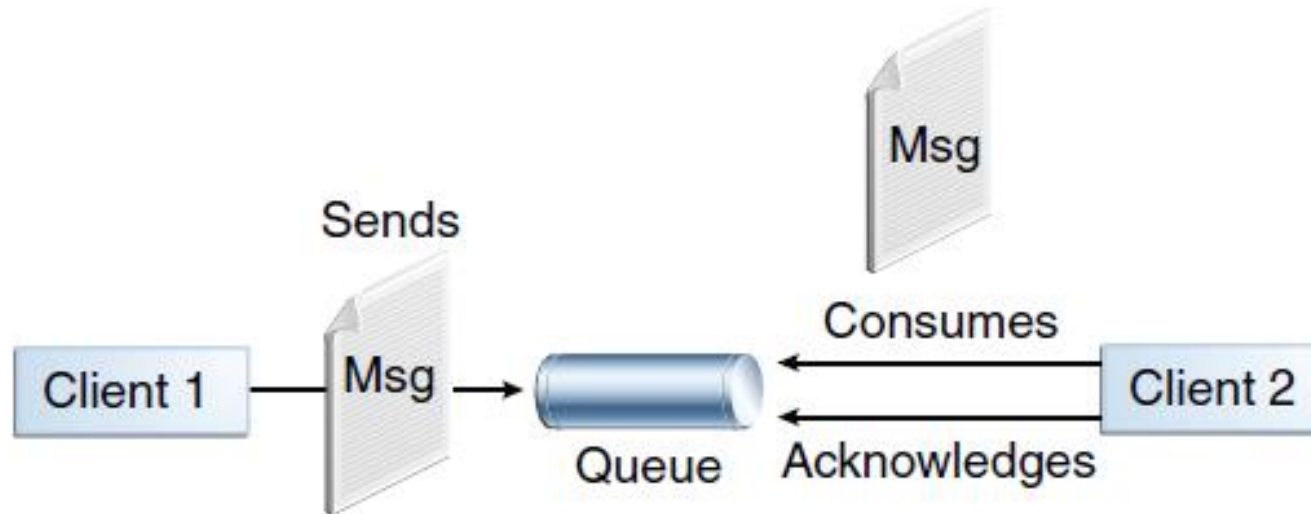
EJB Message Driven

Composant encapsulant de la logique business, et dont le déclenchement sera fait par envoi de message

Par ex. l'implémentation d'un traitement asynchrone long

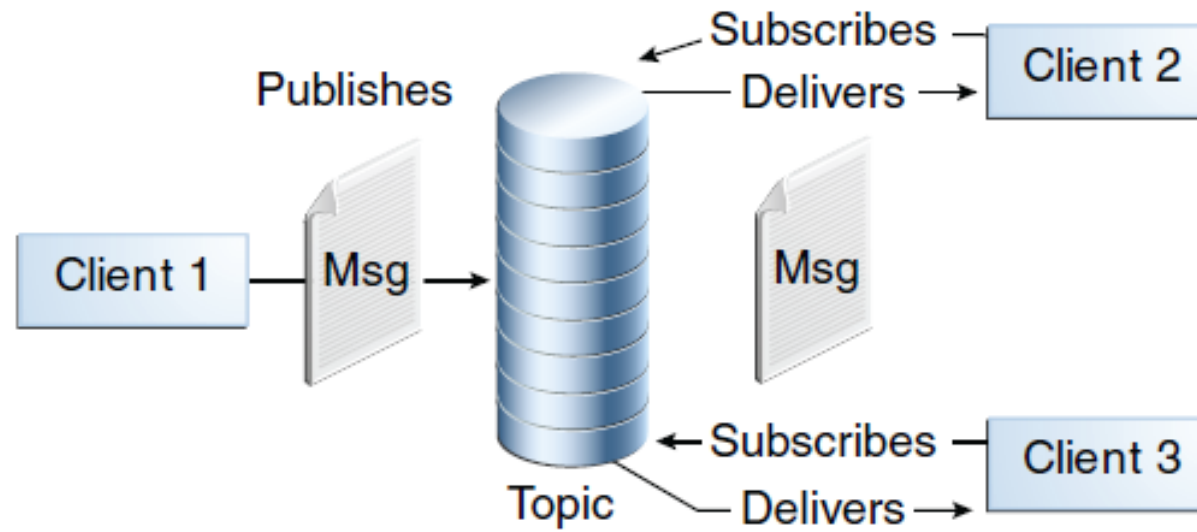
EJB Message Driven (Queue)

FIGURE 47-3 Point-to-Point Messaging



EJB Message Driven (Topic)

FIGURE 47-4 Publish/Subscribe Messaging



EJB Message Driven – JMS Producer

```
public class HelloWorldJmsProducer {  
  
    // On va injecte la queue déclarée sur le serveur via son nom JNDI  
    @Resource(name = "java:/asi2-HelloWorld")  
    private Queue queue;  
  
    @Inject    // On injecte le contexte JMS  
    // On redéfinie la ConnectionFactory du contexte si on n'utilise pas celle par  
    // défaut. Il faut aussi la déclarer sur le serveur.  
    //@JMSConnectionFactory(value = "java:comp/DefaultJMSConnectionFactory")  
    private JMSContext context;  
  
    public void sendHelloWord() {  
        // On crée un producer à partir du contexte et on envoie notre message sur la queue  
        context.createProducer().send(queue, "Hello world !");  
    }  
}
```

EJB Message Driven – JMS Consumer

```
// On utilise l'annotation @MessageDriven pour configurer le listener sur la queue
@MessageDriven(name = "HelloWorldMDB", activationConfig = {
// Queue ou Topic
    @ActivationConfigProperty(propertyName = "destinationType", propertyValue =
"javax.jms.Queue"),
// Nom JNDI
    @ActivationConfigProperty(propertyName = "destination", propertyValue = "asi2-HelloWorld"),
// Configuration pour reconnaître automatiquement les messages recus
    @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue = "Auto-
acknowledge") })
// Le consumer doit implémenter l'interface MessageListener et définir la méthode onMessage(Message
message)
public class HelloWorldJmsConsumer implements MessageListener {
    private static Logger logger = Logger.getLogger(HelloWorldJmsConsumer.class.getName());

    @Override
    public void onMessage(Message var1) {
        logger.info(">>> onMessage() - " + var1.toString());
        try {
            if (var1 instanceof TextMessage) {
                String msg = ((TextMessage) var1).getText();
                logger.info(msg);

                // Do something...
            }
        } catch (JMSEException e) {
            e.printStackTrace();
        }
        logger.info("<<< onMessage()");
    }
}
```

Message Driven Bean

TP

Activer le module JMS dans Wildfly en utilisant la configuration standalone-full.xml à la place du standalone.xml

```
# Dans la commande de lancement du serveur
```

```
$WILDFLY_HOME/bin/standalone.sh -c standalone-full.xml
```

```
# Dans les options JVM de lancement du serveur
```

```
-Djboss.server.default.config=standalone-full.xml
```

Créer une Queue dans Wildfly

Créer un EJB Message Driven dans le module EJB

Créer un webservice dans le module Web

Poster depuis le webservice un message dans la queue

Java EE

Présentation

Standard de déploiement

Ressources de serveur d'applications

Composants

- EJB Session

- EJB Message-Driven

- Entités et Unités de persistance

Services

- Transactions

- Timers

JPA

API permettant de

- définir des entités métier
- créer / requêter / mettre à jour / supprimer ces entités
- configurer l'ORM (emplacement et nature de la base de données)

JPA persistence unit

Fichier persistence.xml sous META-INF

```
<?xml version="1.0" encoding="UTF-8"?>
  <persistence xmlns="http://java.sun.com/xml/ns/persistence">
    <persistence-unit name="em1">
      <jta-data-source>jdbc/MyDB2DB</jta-data-source>
      <properties>
        <property name="eclipselink.target-database"
          value="DB2"/>
      </properties>
    </persistence-unit>
  </persistence>
```

JPA Entité

```
@Entity
@Table(name = "ROOM")
public class Room implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "room_id")
    private Integer id;

    @Column(name = "number")
    private String number; //immutable

    @Column(name = "capacity")
    private Integer capacity;

    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @JoinColumn(name = "building_id")
    private Building building; //immutable
```


JPA EntityManager

```
@PersistenceContext
EntityManager em;
...
public LineItem createLineItem(Order order, Product product,
    int quantity) {
    LineItem li = new LineItem(order, product, quantity);
    order.getLineItems().add(li);
    em.persist(li);
    return li;
}

public void removeOrder(Integer orderId) {
    try {
        Order order = em.find(Order.class, orderId);
        em.remove(order);
    } ...
}
```

JPA Query et Criteria

```
// User.java
@Entity
@Table(name = "users")
@NamedQuery(name = "Users.list", query = "select u from User u")
public class User implements Serializable { ... }
```

```
// UserDao.java
@PersistenceContext
EntityManager em;

// Création d'une Query simple
public User getUserById(int id) {
    User user = (User)em.createQuery("from User u where u.id = :id")
        .setParameter("id", id)
        .getSingleResult();

    return user;
}

// Utilisation d'une NameQuery
public List<User> listUser() {
    List<User> users = em.createNamedQuery("Users.list")
        .getResultList();

    return users;
}
```

JPA Query et Criteria

```
// Requete par criteria
public List<User> getUserByLogin(String login) {
    CriteriaBuilder builder = em.getCriteriaBuilder();

    CriteriaQuery<User> crit = builder.createQuery(User.class);
    Root<User> root = crit.from(User.class);
    crit.select(root)
        .where(builder.like(builder.lower(root.get("login")), login.toLowerCase()));

    List<User> users = em.createQuery(crit).getResultList();

    return users;
}
```

JPA

TP

Déclarer une datasource dans Wildfly

Modifier l'une unité de persistance du module EJB

Créer une entité User(id, login, password, firstName, lastName, birthdate)

Créer une DAO implémentée sous forme d'EJB Session Stateless (UserDao)

Créer un WebService qui appelle l'EJB

Java EE

Présentation

Standard de déploiement

Ressources de serveur d'applications

Composants

- EJB Session

- EJB Message-Driven

- Entités et Unités de persistance

Services

- Transactions

- Timers

Transaction Management

```
@TransactionAttribute(NOT_SUPPORTED)
@Stateful
public class TransactionBean implements Transaction {
    ...
    @TransactionAttribute(REQUIRES_NEW)
    public void firstMethod() {...}

    @TransactionAttribute(REQUIRED)
    public void secondMethod() {...}

    public void thirdMethod() {...}

    public void fourthMethod() {...}
}
```

Java EE

Présentation

Standard de déploiement

Ressources de serveur d'applications

Composants

- EJB Session

- EJB Message-Driven

- Entités et Unités de persistance

Services

- Transactions

- Timers

Timer Service – Explicit timers

```
@Timeout  
public void timeout(Timer timer) {  
    System.out.println("TimerBean: timeout occurred");  
}
```

```
long duration = 6000;  
Timer timer =  
    timerService.createSingleActionTimer(duration, new TimerConfig());
```

```
Timer timer = timerService.createTimer(intervalDuration,  
    "Created new programmatic timer");
```


Timer Service – Automatic timers

```
@Schedule(dayOfWeek="Sun", hour="0")  
public void cleanupWeekData() { ... }
```