La couche DAL
Présentation du JPA de J2EE

Pr. HAJJI TARIK

ENSAM-UMI

IATD-4

2025-2026

J2EE

- J2EE, ou **Java 2 Platform, Enterprise Edition**, est une **plateforme de développement Java** destinée à la création d'**applications d'entreprise distribuées, sécurisées et multi-tiers**.

- Elle fournit un ensemble d'**API, de services et de composants** permettant de développer des applications robustes et évolutives côté serveur.
- Aujourd'hui, **J2EE** a évolué vers **Jakarta EE**, qui en est la version modernisée et standardisée par la fondation Eclipse.

Caractéristiques principales :

- Architecture multi-tiers (présentation, logique métier, données)
- Portabilité entre différents serveurs d’applications (Tomcat, GlassFish, WildFly…)
- Gestion intégrée des transactions et de la sécurité
- Support des technologies web (JSP, Servlets, XML, JSON, REST, SOAP…)

Les principales API de J2EE

Catégorie	Principales API / Technologies	Rôle principal	Exemples d'utilisation
Côté Web (Présentation)	Servlets, JSP, JSF	Gestion des requêtes HTTP, génération dynamique et interfaces web	Développement d'interfaces web interactives et contrôleurs serveur
Côté Métier (Logique d'application)	EJB (Enterprise JavaBeans)	Gestion de la logique métier, sécurité et transactions	Composants métier déployés côté serveur d'application
Côté Données (Persistance)	JDBC, JPA	Accès et gestion des données	Connexion aux bases de données, mapping objet-relationnel
Communication et Intégration	JMS, JAX-RS, JAX-WS, JTA	Communication asynchrone, services web et transactions distribuées	Échange de messages, services REST/SOAP, coordination de transactions
Sécurité et Services Complémentaires	JAAS, JNDI, JavaMail	Sécurité, gestion des ressources et messagerie	Authentification, accès aux ressources serveur, envoi de mails

JPA – *Java Persistence API*

- JPA est une spécification Java qui définit un modèle standard pour la gestion de la persistance des données dans les applications Java.
- Elle permet de mapper les objets Java aux tables d'une base de données relationnelle, évitant ainsi d'écrire du code SQL complexe.

Objectifs de JPA

- Simplifier l’accès et la manipulation des données dans les applications Java.
- Offrir une couche d’abstraction au-dessus de JDBC.
- Gérer automatiquement les opérations CRUD (Create, Read, Update, Delete).
- Assurer la portabilité du code entre différents fournisseurs de bases de données.

Concepts clés

Concept	Description
Entity	Classe Java représentant une table de la base de données.
EntityManager	Interface principale qui gère le cycle de vie des entités (création, mise à jour, suppression, recherche).
Persistence Unit	Regroupe la configuration de connexion et les entités à gérer (définie dans persistence.xml).
JPQL (Java Persistence Query Language)	Langage de requêtes orienté objets, similaire au SQL mais basé sur les entités.
Annotations JPA	Permettent de décrire le mapping entre les classes Java et les tables SQL (@Entity, @Table, @Id, @Column…).

Implémentations courantes

Implémentation	Éditeur / Organisation	Points forts	Limitations	Cas d'utilisation typiques
Hibernate	Red Hat	- La plus populaire et mature- Large communauté- Compatible avec la majorité des SGBD- Supporte des fonctionnalités avancées (cache, lazy loading, HQL)	- Configuration parfois complexe- Performances variables selon le mapping	Applications d'entreprise complexes, frameworks comme Spring Boot
EclipseLink	Eclipse Foundation	- Implémentation officielle de la spécification JPA- Haute stabilité- Bonne intégration avec Jakarta EE- Support de JPA 3.1+	- Communauté plus restreinte que Hibernate- Documentation moins abondante	Applications Jakarta EE, serveurs GlassFish et Payara
OpenJPA	Apache Software Foundation	- Léger et modulaire- Intégration facile avec serveurs Apache Geronimo et TomEE	- Moins maintenu récemment- Communauté réduite	Projets académiques ou serveurs Apache
DataNucleus	Open Source (Community Driven)	- Supporte plusieurs types de stockage (RDBMS, NoSQL, fichiers, etc.)- Très flexible	- Moins performant pour les grandes bases- Peu utilisé en production	Projets hybrides ou expérimentaux nécessitant du multi-stockage

Annotations de **Mapping (Structurelles)**

Annotation	Rôle / Description	Exemple
@Entity	Indique qu'une classe Java est une entité persistante (table dans la BD).	@Entity public class Etudiant { ... }
@Table	Spécifie le nom de la table associée à l'entité.	@Table(name = "ETUDIANTS")
@Id	Définit la clé primaire de l'entité.	@Id private Long id;
@GeneratedValue	Génère automatiquement la clé primaire (AUTO, IDENTITY, SEQUENCE...).	@GeneratedValue(strategy = GenerationType.AUTO)
@Column	Définit le mapping entre un attribut et une colonne.	@Column(name = "NOM", nullable = false)
@Transient	Attribut non persistant (non stocké dans la BD).	@Transient private int ageTemp;
@Enumerated	Définit comment stocker une énumération (STRING ou ORDINAL).	@Enumerated(EnumType.STRING)
@Temporal	Indique le type temporel (DATE, TIME, TIMESTAMP).	@Temporal(TemporalType.DATE)

Annotations de **Relation (Association entre Entités)**

Annotation	Rôle / Description	Exemple
@OneToOne	Relation un-à-un entre deux entités.	@OneToOne private Adresse adresse;
@OneToMany	Relation un-à-plusieurs.	@OneToMany(mappedBy = "etudiant") private List<Cours> cours;
@ManyToOne	Relation plusieurs-à-un.	@ManyToOne private Departement departement;
@ManyToMany	Relation plusieurs-à-plusieurs.	@ManyToMany private List<Projet> projets;
@JoinColumn	Définit la colonne de jointure pour une relation.	@JoinColumn(name = "id_dept")
@JoinTable	Spécifie une table d’association (relation ManyToMany).	@JoinTable(name = "ETUDIANT_PROJET")

Annotations de **Configuration (Comportementales / Techniques)**

Annotation	**Rôle / Description**	**Exemple**
@PersistenceContext	Injection automatique d'un EntityManager dans une classe.	@PersistenceContext EntityManager em;
@PersistenceUnit	Injection d'une unité de persistance entière.	@PersistenceUnit EntityManagerFactory emf;
@Version	Gère le versionnement (optimistic locking) d'une entité.	@Version private int version;
@NamedQuery	Déclare une requête nommée au niveau de la classe.	@NamedQuery(name = "Etudiant.findAll", query = "SELECT e FROM Etudiant e")
@NamedQueries	Regroupe plusieurs requêtes nommées.	@NamedQueries({...})