Web Applications Developments

ITWT413

LECTURE 9: JAKARTA EE WEB PROFILE

JAKARTA WEBSOCKET

DR. HALA SHAARI

Jakarta EE Tutorial- Structure

Basic Platform

- Resource Creation
- Injection
- Packaging

Jakarta EE Core Profile

- Jakarta CDI Lite
- Jakarta REST
- Jakarta JSON

Jakarta EE Web Profile

- Jakarta CDI Full
- Jakarta Validation
- Jakarta Security
- Jakarta Servlets
- Jakarta Faces
- Jakarta WebSocket
- Jakarta Persistence
- Jakarta Enterprise Beans Lite

Jakarta EE Platform

- Jakarta Mail
- Jakarta Messaging
- Jakarta Batch

Jakarta EE 10

Lecture Agenda

- Jakarta Persistence
  - Entities
  - Requirements for Entity Classes
  - Persistent Fields and Properties in Entity Classes
  - Primary Keys in Entities
  - Multiplicity in Entity Relationships
  - Direction in Entity Relationships
  - Embeddable Classes in Entities
  - Entity Inheritance
  - Abstract Entities
  - Querying Entities

Introduction to Jakarta Persistence

- Jakarta Persistence provides Java developers with an object/relational mapping facility for managing relational data in Java applications. Jakarta Persistence consists of four areas:
  - Jakarta Persistence
  - The query language
  - The Jakarta Persistence Criteria API
  - Object/relational mapping metadata

Entities

- An entity is a lightweight persistence domain object. Typically, an entity represents a table in a relational database, and each entity instance corresponds to a row in that table.
- The primary programming artifact of an entity is the entity class, although entities can use helper classes.
- The persistent state of an entity is represented through either persistent fields or persistent properties.
- These fields or properties use object/relational mapping annotations to map the entities and entity relationships to the relational data in the underlying data store.

Requirements for Entity Classes

An entity class must follow these requirements.

- The class must be annotated with the jakarta.persistence.Entity annotation.
- The class must have a public or protected, no-argument constructor. The class may have other constructors.
- The class must not be declared final. No methods or persistent instance variables must be declared final.
- If an entity instance is passed by value as a detached object, such as through a session bean's remote business interface, the class must implement the Serializable interface.
- Entities may extend both entity and non-entity classes, and non-entity classes may extend entity classes.
- Persistent instance variables must be declared private, protected, or package-private and can be accessed directly only by the entity class's methods. Clients must access the entity's state through accessor or business methods.

Persistent Fields and Properties in Entity Classes

- The persistent state of an entity can be accessed through either the entity's instance variables or properties. The fields or properties must be of the following Java language types:
  - Java primitive types
  - java.lang.String
  - Other serializable types, including: Wrappers of Java primitive types, java.math.BigInteger, java.math.BigDecimal, java.util.Date, ….
  - Enumerated types
  - Other entities and/or collections of entities
  - Embeddable classes

Primary Keys in Entities

- Each entity has a unique object identifier.
- A customer entity, for example, might be identified by a customer number. The unique identifier, or primary key, enables clients to locate a particular entity instance. Every entity must have a primary key.
- An entity may have either a simple or a composite primary key.
  - Simple primary keys use the jakarta.persistence.Id annotation to denote the primary key property or field.
  - Composite primary keys are used when a primary key consists of more than one attribute, which corresponds to a set of single persistent properties or fields.
  - Composite primary keys must be defined in a primary key class.
  - Composite primary keys are denoted using the jakarta.persistence.EmbeddedId and jakarta.persistence.IdClass annotations.

Primary/ Composite key type

- The primary key, or the property or field of a composite primary key, must be one of the following Java language types:
  - Java primitive types
  - Java primitive wrapper types
  - java.lang.String
  - java.util.Date (the temporal type should be DATE)
  - java.sql.Date
  - java.math.BigDecimal
  - java.math.BigInteger
- Floating-point types should never be used in primary keys. If you use a generated primary key, only integral types will be portable.

primary key requirements

- A primary key class must meet these requirements.
  - The access control modifier of the class must be public.
  - The properties of the primary key class must be public or protected if property-based access is used.
  - The class must have a public default constructor.
  - The class must implement the hashCode() and equals(Object other) methods.
  - The class must be serializable.
  - A composite primary key must be represented and mapped to multiple fields or properties of the entity class or must be represented and mapped as an embeddable class.
  - If the class is mapped to multiple fields or properties of the entity class, the names and types of the primary key fields or properties in the primary key class must match those of the entity class.

Multiplicity in Entity Relationships

- Multiplicities are of the following types.
  - One-to-one: Each entity instance is related to a single instance of another entity.
  - One-to-many: An entity instance can be related to multiple instances of the other entities
  - Many-to-one: Multiple instances of an entity can be related to a single instance of the other entity.
  - Many-to-many: The entity instances can be related to multiple instances of each other.

Direction in Entity Relationships

- The direction of a relationship can be either bidirectional or unidirectional.
- A bidirectional relationship has both an owning side and an inverse side.
- A unidirectional relationship has only an owning side. The owning side of a relationship determines how the Persistence runtime makes updates to the relationship in the database.

Embeddable Classes in Entities

- Embeddable classes are used to represent the state of an entity but don't have a persistent identity of their own, unlike entity classes.
- Instances of an embeddable class share the identity of the entity that owns it.
- Embeddable classes exist only as the state of another entity. An entity may have single-valued or collection-valued embeddable class attributes.
- Embeddable classes have the same rules as entity classes but are annotated with the jakarta.persistence.Embeddable annotation instead of @Entity.

Entity Inheritance

- Entities support class inheritance, polymorphic associations, and polymorphic queries.
- Entity classes can extend non-entity classes, and non-entity classes can extend entity classes.
- Entity classes can be both **abstract and concrete**.

Managing Entities

- Entities are managed by the entity manager, which is represented by jakarta.persistence.EntityManager instances.
- Each EntityManager instance is associated with a persistence context: a set of managed entity instances that exist in a particular data store.
- A persistence context defines the scope under which particular entity instances are created, persisted, and removed.
- The EntityManager interface defines the methods that are used to interact with the persistence context.

Managing an Entity Instance's Lifecycle

- You manage entity instances by invoking operations on the entity by means of an EntityManager instance. Entity instances are in one of four states: new, managed, detached, or removed.
  - New entity instances have no persistent identity and are not yet associated with a persistence context.
  - Managed entity instances have a persistent identity and are associated with a persistence context.
  - Detached entity instances have a persistent identity and are not currently associated with a persistence context.
  - Removed entity instances have a persistent identity, are associated with a persistent context, and are scheduled for removal from the data store.

Querying Entities

- Jakarta Persistence provides the following methods for querying entities.
  - The Jakarta Persistence query language (JPQL) is a simple, string-based language similar to SQL used to query entities and their relationships. See The Jakarta Persistence Query Language for more information.
  - The Criteria API is used to create typesafe queries using Java programming language APIs to query for entities and their relationships. See Using the Criteria API to Create Queries for more information.
- Both JPQL and the Criteria API have advantages and disadvantages.

JPQL and the Criteria API have advantages and disadvantages

- Just a few lines long, JPQL queries are typically more concise and more readable than Criteria queries. Developers familiar with SQL will find it easy to learn the syntax of JPQL. JPQL named queries can be defined in the entity class using a Java programming language annotation or in the application's deployment descriptor. JPQL queries are not typesafe, however, and require a cast when retrieving the query result from the entity manager. This means that type-casting errors may not be caught at compile time. JPQL queries don't support open-ended parameters.
- Criteria queries allow you to define the query in the business tier of the application. Although this is also possible using JPQL dynamic queries, Criteria queries provide better performance because JPQL dynamic queries must be parsed each time they are called. Criteria queries are typesafe and therefore don't require casting, as JPQL queries do. The Criteria API is just another Java programming language API and doesn't require developers to learn the syntax of another query language. Criteria queries are typically more verbose than JPQL queries and require the developer to create several objects and perform operations on those objects before submitting the query to the entity manager.

Questions?