JPA Guide

JEUS 9

TMAXSOFT

owners.

The names of companies, systems, and products mentioned in this manual may not necessarily be indicated with a trademark symbol (^{TM}, ®).

Open Source Software Notice

Some modules or files of this product are subject to the terms of the following licenses: APACHE2.0, CDDL1.0, EDL1.0, OPEN SYMPHONY SOFTWARE1.1, TRILEAD-SSH2, Bouncy Castle, BSD, MIT, SIL OPEN FONT1.1

Detailed Information related to the license can be found in the following directory:
${INSTALL_PATH}/license/oss_licenses

Document History

Product Version	Guide Version	Date	Remarks
JEUS 9	3.1.2	2025-03-24	-
JEUS 9	3.1.1	2024-12-24	-

Contents

1. Introduction

Jakarta Persistence API (hereafter JPA) provides standard ORM technology to access a relational database. It replaces CMP entity beans provided in EJB.

JEUS supports all functions of the JPA specifications. JPA is defined as part of the JPA 3.0 specification in JSR 338, does not depend on an EJB container, and can be used by EJBs, web modules, and Java SE Standalone clients.

Moreover, JPA can be used by selecting a persistence provider implementation. JEUS provides the **EclipseLink** implementation classes of Eclipse Persistence Services Project by default.

Other implementations can also be selected for use. For more information, refer to Changing Persistence Provider.

When using JPA, you must consider both the basic attributes of the APIs and the characteristics of the settings and providers, which are supported by the JPA specification. In particular, developing an application without considering an important attribute, such as caching, may not produce the desired result. Therefore, developers should develop an application with proper JPA configuration according to the environment. It is recommended to continue to refer to the EclipseLink JPA website, where such information will continue to be introduced using various patterns.

- EclipseLink

```
http://www.eclipse.org/eclipselink/
```

This document only covers configurations required for using the EclipseLink, the basic provider in JEUS. For further information about the JPA technology or the programming methodology, refer to the following references.

References

- Pro EJB 3 Java Persistence API, Mike Keith and Merrick Schincariol, Apress
- Enterprise JavaBeans 3.0 5th ed., Bill Burke and Richard Monson-Haefel, O'Reilly
- Pro JPA 2 Mastering the Java Persistence API, Mike Keith and Merrick Schincariol, Apress

2. Configuring the Provider

This chapter describes how to configure the basic provider of JEUS, EclipseLink. The configuration is needed to implement features that are not defined in JPA specifications, and needs to be set accurately according to each application.

2.1. Configuring the Database

This section describes how to configure the database for each environment and database type, and how to automatically create the database schema.

2.1.1. Configuring Each Environment

Database configuration is different for each environment.

Jakarta EE Environment

The Jakarta EE environment (or mode) refers to a web container, EJB container, application client container of a JEUS Managed Server.

> More accurately, the environment refers to the thread controlled by each container. An example is the thread of the web thread pool configured in the web engine. If there is a thread which is not controlled by a container, such as a thread pool created by an application, it is also managed in the same way as the container-controlled environment.

Target database to be used is set by the persistence.xml descriptor. The <jta-data-source> and <non-jta-data-source> elements are set according to the transaction type.

- When using global transactions
    - Set <transaction-type> to JTA.
    - Set <jta-data-source> to the JNDI name of the corresponding data source.
- When using local transactions
    - Set <transaction-type> to RESOURCE_LOCAL.
    - Set <non-jta-data-source> to the JNDI name of the corresponding data source.

Configuring Database in the Jakarta EE Mode

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence">
  <persistence-unit name="em" transaction-type="JTA">
    <jta-data-source>jdbc/MyDB</jta-data-source>
  </persistence-unit>
```

```
</persistence>
```

1. If <transaction-type> is not set in the Jakarta EE environment, by default, JTA transaction will be used.

2. For information about how to configure a DB data source in JEUS, refer to "DB Connection Pool and JDBC" in JEUS Server Guide.

Java SE Environment

The Java SE environment (or mode) indicates that JPA is not used in a Jakarta EE container, but in environments like the Java stand-alone client environment. In this environment, only local transactions can be used, and the JDBC properties of the target database must be configured.

Set **<transaction-type>** to 'RESOURCE_LOCAL'.

Configuring Database in the Java SE Mode

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence">
   <persistence-unit name="em" transaction-type="JTA">
     <jta-data-source>jdbc/MyDB</jta-data-source>
   </persistence-unit>
</persistence>
```

The following describes each property:

Value	Description
eclipselink.jdbc.driver	JDBC driver class name of target database.
eclipselink.jdbc.url	JDBC URL of target database.
eclipselink.jdbc.user	Username for target database.
eclipselink.jdbc.password	Password for target database.

2.1.2. Configuring Database Type

In general, database type can be detected through JDBC connection information. However, the 'eclipselink.target-database' property can be set for cases when automatic sensing feature does not work properly or when a separate database is used.

Database type can be detected by using DatabaseMetaData.getDatabaseProductName() of the JDBC driver which searches for a database vendor name using regular expressions.

Configuring Database Type

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

The following describes each database type value:

Value	Description
Auto	Automatic Sensing (Default)
Cloudscape	Cloudscape DBMS
DB2	IBM DB2 DBMS
DB2Mainframe	IBM DB2 Mainframe DBMS
Derby	Apache Derby DBMS
HSQL	HSQL DBMS
JavaDB	JavaDB DBMS
MySQL4	MySQL DBMS
Oracle	Oracle DBMS
PostreSQL	PostreSQL DBMS
SQLServer	Microsoft SQLServer DBMS
Sybase	Sybase DBMS
Customized class name	Used to add DBMS that is not supported by default.
Others	For other DBs, refer to the target-database section in the Persistence Property Extensions Reference page of eclipselink.

If you want to use a DBMS that is not supported by default, you can implement the DBMS support features by specifying the corresponding class name. For more information, refer to References.

2.1.3. Automatic Schema Creation

To use the feature that automatically creates the DB schema, set the following property. The property enables the DB tables and constraints to be automatically created when an application is deployed.

Automatic Schema Creation

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence">
  <persistence-unit name="em">
```

```
        <jta-data-source>jdbc/MyDB</jta-data-source>
        <properties>
          ...
          <property name="eclipselink.ddl-generation" value="create-tables" />
          ...
        </properties>
      </persistence-unit>
    </persistence>
```

The following is a configuration property associated with automatic schema creation.

Item	Description
eclipselink.ddl-generation	Setting on how to create Data Descriptor Language (DDL) for schema. ◦ none: Does nothing (default). ◦ create-tables: Creates new tables while keeping current ones. ◦ drop-and-create-tables: Removes current tables and creates new ones. ◦ create-or-extend-tables: Creates tables, but only adds columns if there are existing tables.

When DDL is created, a Java type is created according to the corresponding the Database SQL type. The following table is a summary of Java types and the Database SQL types:

Java Type	Derby, JavaDB, Cloudscape	Oracle	DB2	Sybase	SQLServer	MySQL
boolean, Boolean	SMALLINT	NUMBER(1)	SMALLINT	BIT	BIT	TINYINT(1)
int, Integer	INTEGER	NUMBER(10)	INTEGER	INTEGER	INTEGER	INTEGER
long, Long	BIGINT	NUMBER(19)	INTEGER	NUMERIC(19)	NUMERIC(19)	BIGINT
float, Float	FLOAT	NUMERIC(19,4)	FLOAT	FLOAT(16)	FLOAT(16)	FLOAT
double, Double	FLOAT	NUMERIC(19,4)	FLOAT	FLOAT(32)	FLOAT(32)	DOUBLE
short, Short	SMALLINT	NUMBER(5)	SMALLINT	SMALLINT	SMALLINT	SMALLINT
byte, Byte	SMALLINT	NUMBER(3)	SMALLINT	SMALLINT	SMALLINT	SMALLINT
java.lang.Number	DECIMAL	NUMBER(38)	DECIMAL(15)	NUMERIC(38)	NUMERIC(28)	DECIMAL(38)
java.math.BigInteger	BIGINT	NUMBER(38)	BIGINT	NUMERIC(38)	NUMERIC(28)	BIGINT
java.math.BigDecimal	DECIMAL	NUMBER(38)	DECIMAL(15)	NUMERIC(38)	NUMERIC(28)	DECIMAL(38)

Java Type	Derby, JavaDB, Cloudscape	Oracle	DB2	Sybase	SQLServer	MySQL
java.lang.String	VARCHAR(255)	VARCHAR(255)	VARCHAR(255)	VARCHAR(255)	VARCHAR(255)	VARCHAR(255)
char, Character	CHAR(1)	CHAR(1)	CHAR(1)	CHAR(1)	CHAR(1)	CHAR(1)
byte[], Byte[], java.sql.Blob	BLOB(64000)	LONG RAW	BLOB(64000)	TEXT	TEXT	TEXT(64000)
char[], Character[], java.sql.Clob	CLOB(64000)	LONG	CLOB(64000)	TEXT	TEXT	TEXT(64000)
java.sql.Date	DATE	DATE	DATE	DATETIME	DATETIME	DATE
java.sql.Time	TIME	DATE	TIME	DATETIME	DATETIME	TIME
java.sql.Timestamp	TIMESTAMP	DATE	TIMESTAMP	DATETIME	DATETIME	DATETIME

2.2. Caching

JPA supports 1st-level caching called persistence context by default. However, since persistence context (except extended persistence context) is newly created for each transaction, caching between transactions is not supported. To compensate for this, Eclipse Link provides 2nd-level caching capability.

Because 2nd-level caching is supported in the EntityManagerFactory level, all EntityManagers created in the same EntityManagerFactory use the shared cache. An entity, that does not exist in persistence context, is obtained from the 2nd-level cache, if the entity exists there. This will help improve performance when repeatedly performing a reading job.

Caching Type Configuration Example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence">
  <persistence-unit name="em">
    <jta-data-source>jdbc/MyDB</jta-data-source>
    <properties>
      ...
      <property name="eclipselink.ddl-generation" value="create-tables" />
      <property name="eclipselink.cache.type.default" value="NONE" />
      <property name="eclipselink.cache.size.default" value="999" />
      <property name="eclipselink.cache.shared.default" value="false" />
      ...
    </properties>
  </persistence-unit>
</persistence>
```

The following table describes caching options:

Item	Description
eclipselink.cache.type.default	Sets the caching type. For options other than the ones listed below, or for advanced settings, refer to the CacheType and Size item of "EclipseLink User Guide." ◦ Full: Caches objects using hard references. They exist in the cache until the entity is removed. ◦ Weak: Caches objects using weak references. They are removed after garbage collection (GC). ◦ Soft: Similar to Weak, but removes objects only when memory is insufficient (default value). ◦ NONE: Does not save objects in a cache. Not Recommended. To disable the caching feature, change the eclipselink.cache.shared.default or eclipselink.cache.shared.*<ENTITY>* property.
eclipselink.cache.size.default	Sets the maximum number of objects that can be saved in the cache (Default value: 1,000)
eclipselink.cache.shared.default	Sets whether a shared cache is used. ◦ true: Objects are saved in the shared cache. All EntityManagers use the cache (Default value). ◦ false: Objects are not saved in the shared cache. EntityManagers do not share the cache. If set to "false," the 2nd-level caching is not used.
eclipselink.cache.type.*<ENTITY>*	Sets the cache type for each entity. *<ENTITY>* may be replaced by an entity name or a fully-qualified class name. All entities related to the entity must have the same setting. Same as the value described for eclipselink.cache.type.default.
eclipselink.cache.size.*<ENTITY>*	Sets the cache size for each Entity. *<ENTITY>* may be replaced by an entity name or a fully-qualified class name. All entities related to the entity must have the same setting. Same as the value described for eclipselink.cache.size.default.
eclipselink.cache.shared.*<ENTITY>*	Sets whether each entity uses the shared cache. *<ENTITY>* may be replaced by an entity name or a fully-qualified class name. All entities related to the entity must have the same setting. Same as the value described for eclipselink.cache.shared.default.

When the 2nd-level caching is used, changes to DB data does not apply to the cache when changes are made directly or by external applications. In this case, the value in the cache, instead of the most recent value, is returned to the applications.

To prevent this, set the caching options accordingly or use EntityManager.refresh() and toplink.refresh query hint or locking (pessimistic/optimistic).

2.3. Query Hint

When using query objects, query hint enables the use of features supported by the provider.

It can be set when a query is performed as shown in the following example. It can also be set by @QueryHintAnnotation when using Named Query.

Using Query Hint

```
List employees = em.createQuery("SELECT e FROM Employee e WHERE e.name = :name")
      .setParameter("name", name)
      .setHint("eclipselink.refresh", true)
      .getResultList();
```

The following are supported query hints.

Item	Description
eclipselink.pessimistic-lock	Sets whether pessimistic locking is used when "SELECT" is performed. ◦ NoLock: Does not use the locking (Default value). ◦ Lock: Uses the locking by executing the "SELECT ... FOR UPDATE" statement. ◦ NoLockWait: Uses the locking by executing the "SELECT ... FOR UPDATE NO WAIT" statement.
eclipselink.refresh	Sets whether to update a cache with the latest values from DB. ◦ true: Gets the latest values and updates the cache. ◦ false: Uses the values in the cache (Default value).

2.4. Configuring Logging

To see more detailed logs, configure the logging level.

By default, the logging level is set to the default level (INFO) of JEUS Server. Use the eclipselink.logging.level property to change the level for each persistence unit.

In the Jakarta EE mode, the JEUS logger provided by JEUS is used by default. In the Java SE mode, the DefaultLogger is used by default. You can change the default logger by using the eclipselink.logging.logger property.

Logging Configuration Example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence">
  <persistence-unit name="em">
    <jta-data-source>jdbc/MyDB</jta-data-source>
    <properties>
      ...
      <property name="eclipselink.logging.level" value="FINE"/>
      <property name="eclipselink.logging.logger" value="DefaultLogger"/>
      ...
    </properties>
  </persistence-unit>
</persistence>
```

The following shows more detailed information:

Item	Description
eclipselink.logging.level	Sets the logging level. ◦ OFF: Log is not recorded. ◦ SEVERE ◦ WARNING ◦ INFO (Default value) ◦ CONFIG ◦ FINE: set to this level for more information related to SQL. ◦ FINER ◦ FINEST
eclipselink.logging.logger	Sets the logger to be used. ◦ JEUSLogger: The logger provided by JEUS. (The default value for the Jakarta EE mode) ◦ DefaultLogger: The default standard output Logger. (The default value for the Java SE mode) ◦ JavaLogger: java.util.logging Logger. ◦ Custom class name: Set to this logger if a separate logger is implemented.

For more information about EclipseLink, refer to EclipseLink JPA User Guide.

3. Changing the Provider

This chapter describes how to change the basic provider of JEUS.

3.1. Changing Persistence Provider

If you want to use providers other than the default provider, change the settings provided by the JPA specifications.

Copy the necessary libraries to the JEUS_HOME/lib/application directory or package each application, and set the <provider> property in the persistence.xml file to the class name of the provider. Corresponding persistence units will use the specified provider.

For configuring each provider, refer to the relevant document because the provider class name and related properties vary depending on the provider. For example, to use Hibernate, the following needs to be configured:

Changing Persistence Provider

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence">
  <persistence-unit name="em">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <jta-data-source>jdbc/MyDB</jta-data-source>
    <properties>
      <!-- add Hibernate properties here -->
    </properties>
  </persistence-unit>
</persistence>
```

The default provider can be used or the provider can be changed by persistence unit as shown in the previous example. If the system property jeus.persistence.defaultProvider is set to a provider class name other than the default name when JEUS starts, all other persistence units that do not have a provider configured will use the default provider.

> For more information about JEUS system property settings, refer to "JEUS Server Guide" or "JEUS Reference Guide".

3.2. Other Persistence Providers

For information about other persistence providers, refer to the following sites:

- Hibernate EntityManager

```
http://www.hibernate.org
```

- OpenJPA

```
http://openjpa.apache.org
```

- BEA Kodo

```
http://docs.oracle.com/cd/E13189_01/kodo/docs40/index.html
```