

JPA 안내서

JEUS 9

TMAXSOFT

저작권 공지

Copyright 2025. TmaxSoft Co., Ltd. All Rights Reserved.

회사 정보

(주)티맥스소프트

주소 : 경기도 성남시 분당구 황새울로258번길 29, 티맥스수내타워 8-9층

기술 서비스 센터: 1544-8629

홈페이지: <https://www.tmaxsoft.com>

제한된 권리

이 소프트웨어(JEUS®) 사용설명서와 프로그램은 저작권법과 국제 조약에 의해 보호됩니다. 사용설명서와 프로그램은 TmaxSoft Co., Ltd.와의 사용권 계약 하에서만 사용할 수 있으며, 사용설명서는 사용권 계약의 범위 내에서만 배포 또는 복제할 수 있습니다. 이 사용설명서의 전부 또는 일부를 TmaxSoft의 사전 서면 동의 없이 전자, 기계, 녹음 등의 수단으로 전송, 복제, 배포하거나 2차적 저작물을 작성할 수 없습니다.

이 소프트웨어 사용설명서와 프로그램의 사용권 계약은 어떠한 경우에도 사용설명서 및 프로그램과 관련된 지적 재산권(등록 여부를 불문)을 양도하는 것으로 해석되지 않으며, 브랜드나 로고, 상표 등을 사용할 권한을 부여하지 않습니다. 사용설명서는 오로지 정보 제공만을 목적으로 하며, 이로 인한 계약상의 직접적 또는 간접적 책임을 지지 않습니다. 또한 사용설명서 상의 내용이 법적 또는 상업적인 특정 조건을 만족시킬 것을 보장하지 않습니다. 사용설명서는 제품의 업그레이드나 수정에 따라 예고 없이 변경될 수 있으며, 내용상의 오류가 없음을 보장하지 않습니다.

상표 공지

JEUS®는 TmaxSoft Co., Ltd.의 등록 상표입니다.

Java, Solaris는 Oracle Corporation 및 그 자회사, 관계회사의 등록 상표입니다.

Microsoft, Windows, Windows NT는 Microsoft Corporation의 등록 상표 또는 상표입니다.

HP-UX는 Hewlett Packard Enterprise Company의 등록 상표입니다.

AIX는 International Business Machines Corporation의 등록 상표입니다.

UNIX는 X/Open Company, Ltd.의 등록 상표입니다.

Linux는 Linus Torvalds의 등록 상표입니다.

Noto는 Google Inc.의 상표입니다. Noto 글꼴은 오픈 소스입니다. 모든 Noto 글꼴은 SIL Open Font License, 버전 1.1에 따라 게시됩니다. (<https://www.google.com/get/noto/>)

기타 모든 제품들과 회사 이름은 각각 해당 소유주의 상호, 상표, 또는 등록 상표입니다.

본 사용설명서에 기재된 회사, 시스템, 제품 이름 등에 반드시 상표 표시(™, ®)를 하지는 않습니다.

오픈 소스 소프트웨어 공지

본 제품의 일부 파일 또는 모듈은 다음의 라이선스를 준수합니다.: APACHE2.0, CDDL1.0, EDL1.0, OPEN SYMPHONY SOFTWARE1.1, TRILEAD-SSH2, Bouncy Castle, BSD, MIT, SIL OPEN FONT1.1

관련 상세 정보는 제품의 다음 디렉터리에 기재된 사항을 참고하시기 바랍니다. :

`${INSTALL_PATH}/license/oss_licenses`

유지 보수

| 구분 | 지원항목 | 서비스 내용 |
|----------------|-----------------|--|
| 제품지원 | 패치 & 업그레이드 | 무상 패치 서비스 제공 |
| | | 메이저 버전 업그레이드 시 할인 혜택 |
| | | 웹 지원을 통한 패치 내역 제공 |
| 기술 지원 - 기본 서비스 | 장애 지원 | 장애 발생 시 원인 분석 및 조치 Service Desk팀 → 기술팀 → R&D의 3단계 장애 분석 및 조치 |
| | 일상 지원(온라인 지원) | E-mail, 전화, 원격, 웹 사이트 등 온라인 자원을 통한 질의 응답 서비스 |
| | 고객 맞춤 지원(방문 지원) | 고객의 요청으로 수행하는 방문 지원 서비스 |
| 기술 지원 - 옵션 서비스 | 예방 지원 | 정기 점검을 통한 시스템 운영현황 보고 및 장애 예방 ◦ 관리자 또는 운영자의 요구사항 수렴 ◦ 운영 현황(시스템, 엔진 운영) 보고서 제공 ◦ 필요 시 시스템 개선 권장 사항 보고 |
| 유지 보수 비용 및 기간 | 계약 시 별도 협의 | 계약 시 EOL/EOS 문서 제공 |

안내서 이력

| 제품 버전 | 안내서 버전 | 발행일 | 비고 |
|--------|--------|------------|----|
| JEUS 9 | 3.1.2 | 2025-01-03 | - |
| JEUS 9 | 3.1.1 | 2024-09-27 | - |

목차

| | |
|-------------------------------------|----|
| 1. 소개 | 1 |
| 2. 프로바이더 설정 | 2 |
| 2.1. 데이터베이스 설정 | 2 |
| 2.1.1. 사용 환경별 설정 | 2 |
| 2.1.2. 데이터베이스 타입 설정 | 3 |
| 2.1.3. 스키마 자동 생성 설정 | 4 |
| 2.2. Caching | 6 |
| 2.3. Query 힌트 | 8 |
| 2.4. Logging 설정 | 8 |
| 3. 프로바이더 변경 | 10 |
| 3.1. Persistence 프로바이더 변경 | 10 |
| 3.2. 사용 가능한 Persistence 프로바이더 | 10 |

1. 소개

Jakarta Persistence API(이하 JPA)는 관계형 데이터베이스에 접근하기 위한 표준 ORM 기술을 제공하며, 기존에 EJB에서 제공되던 CMP Entity Bean을 대체하는 기술이다.

JEUS는 JPA 스펙의 기능을 모두 지원한다. JPA는 JSR 338에서 JPA 3.0 스펙으로 정의되어 있으며, EJB 컨테이너에 의존하지 않고 EJB, 웹 모듈 및 Java SE Standalone 클라이언트에서 모두 사용이 가능하다.

또한 JPA는 사용자가 원하는 Persistence 프로바이더(Provider) 구현체를 선택해서 사용할 수 있다. JEUS에서는 기본적으로 Eclipse Persistence Services Project의 **EclipseLink** 구현체를 제공한다.

필요에 따라 다른 구현체를 선택해서 사용할 수 있다. 이에 대한 자세한 사항은 [Persistence 프로바이더 변경](#)을 참고한다.

실제 JPA를 사용할 때는 JPA 스펙에서 제공하는 기본적인 API나 설정과 프로바이더의 특성을 함께 고려해야 한다. 특히, Caching과 같은 것은 중요한 특성으로 이를 모르고 애플리케이션을 개발하면 원하는 결과를 얻지 못할 수도 있다. 따라서 JPA를 환경에 따라 적절하게 구성해서 개발해야 한다. 이러한 것들은 앞으로 다양한 패턴을 통해 소개될 것이므로 EclipseLink JPA 사이트를 계속 참조하기를 권장한다.

- EclipseLink

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

본 안내서는 JEUS의 기본 프로바이더인 EclipseLink를 JEUS에서 사용하는 데 필요한 설정에 대해서만 설명한다. JPA 기술 자체나 프로그래밍 방법에 대해서는 다루고 있지 않기 때문에 이에 대해서는 다음 자료들을 참고한다.

참고 자료

- 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. 프로바이더 설정

본 장에서는 JEUS의 기본 프로바이더인 EclipseLink에 대한 설정을 설명한다. 이러한 설정은 JPA 스펙에서 정의되지 않는 기능을 구현하기 위해 필요한 부분으로, 각 애플리케이션에 맞게 정확하게 설정되어야 한다.

2.1. 데이터베이스 설정

사용 환경, 데이터베이스 타입에 따른 설정과 자동으로 데이터베이스 스키마를 생성하는 설정에 대해 설명한다.

2.1.1. 사용 환경별 설정

사용 환경에 따라 데이터베이스의 설정 방법이 달라진다.

Jakarta EE 환경

Jakarta EE 환경(또는 모드)은 JEUS의 Managed Server(이하 MS) 위에서 웹 컨테이너, EJB 컨테이너, 애플리케이션 클라이언트 컨테이너를 의미한다.



좀 더 정확하게는 각 컨테이너에서 컨트롤하는 Thread를 의미한다. 예를 들어 웹 엔진에 설정하는 Web Thread Pool의 Thread이다. 만약 애플리케이션이 직접 생성한 Thread Pool의 Thread처럼 컨테이너가 관리하지 않는 Thread 상에서는 Java SE 환경과 똑같다.

사용할 데이터베이스 대상에 대한 설정은 persistence.xml Descriptor에 설정한다. 사용할 트랜잭션의 종류에 따라서 <jta-data-source>와 <non-jta-data-source> 값을 설정한다.

- 글로벌 트랜잭션을 사용하는 경우
 - <transaction-type>의 값을 'JTA'로 설정한다.
 - <jta-data-source>에 해당 데이터소스의 JNDI 이름을 설정한다.
- 로컬 트랜잭션을 사용하는 경우
 - <transaction-type>의 값을 'RESOURCE_LOCAL'로 설정한다.
 - <non-jta-data-source>에 해당 데이터소스의 JNDI 이름을 설정한다.

Jakarta EE 모드에서 데이터베이스 설정 예

```
<?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. Jakarta EE 환경에서는 <transaction-type> 값을 지정하지 않을 경우 기본적으로 JTA

트랜잭션으로 설정한다.

2. JEUS에서 DB 데이터소스를 설정하는 방법은 JEUS Server 안내서의 "DB Connection Pool과 JDBC"를 참고한다.

Java SE 환경

Java SE 환경(또는 모드)은 Jakarta EE 컨테이너에서 사용하지 않고 Java Standalone 클라이언트와 같은 환경에서 사용하는 경우를 말한다. 이때는 로컬 트랜잭션만 사용할 수 있으며, 사용되는 데이터베이스의 JDBC 설정을 프로퍼티로 설정해야 한다.

다음의 예제와 같이 **<transaction-type>**의 값을 'RESOURCE_LOCAL'로 설정한다.

Java SE 모드에서 데이터베이스 설정 예

```
<?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>
```

다음은 프로퍼티 값에 대한 설명이다.

| 프로퍼티 값 | 설명 |
|---------------------------|--------------------------------|
| eclipselink.jdbc.driver | 대상 데이터베이스의 JDBC 드라이버 클래스 이름이다. |
| eclipselink.jdbc.url | 대상 데이터베이스의 JDBC URL이다. |
| eclipselink.jdbc.user | 대상 데이터베이스의 사용자명이다. |
| eclipselink.jdbc.password | 대상 데이터베이스의 패스워드이다. |

2.1.2. 데이터베이스 타입 설정

기본적으로 해당 데이터베이스 타입을 JDBC 커넥션 정보를 통해 자동으로 감지한다. 자동 감지 기능이 제대로 동작하지 않거나, 별도의 다른 데이터베이스를 사용하는 경우라면 'eclipselink.target-database' 프로퍼티를 설정할 수 있다.



데이터베이스 타입은 JDBC 드라이버의 DatabaseMetaData.getDatabaseProductName()을 사용하여, 데이터베이스 벤더 이름을 regular expression으로 비교하는 방식을 사용한다.

데이터베이스 타입 설정 예

```
<?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>
```

다음은 지원되는 데이터베이스 타입 값에 대한 설명이다.

| 값 | 설명 |
|-----------------------|--|
| Auto | 자동 감지(기본값) |
| 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 |
| PostgreSQL | PostgreSQL DBMS |
| SQLServer | Microsoft SQLServer DBMS |
| Sybase | Sybase DBMS |
| Customized class name | 기본적으로 지원되지 않는 DBMS를 추가할 때 사용한다. |
| 그 외 | 그 외의 DB에 대해서는 eclipselink의 Persistence Property Extensions Reference 페이지에서 target-database 부분을 참고한다. |

기본적으로 지원되지 않는 DBMS의 경우 별도로 DBMS 지원 기능을 구현하여 사용할 수 있다. 이때는 해당 클래스 이름을 지정한다. 이에 대한 자세한 내용은 본 안내서의 [참고 자료](#)를 참고한다.

2.1.3. 스키마 자동 생성 설정

자동으로 DB 스키마를 생성하는 기능을 사용하는 경우 설정한다. 이를 사용하면 애플리케이션을 deploy할 때 DB 테이블 및 제약 사항(Constraints)을 자동으로 생성한다.

스키마 자동 생성 설정 예

```
<?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>
```



```

</properties>
</persistence-unit>
</persistence>

```

관련된 프로퍼티 설정은 다음과 같다.

| 항목 | 설명 |
|----------------------------|---|
| eclipselink.ddl-generation | <p>스키마 DDL(Data Descriptor Language)을 어떤식으로 생성할지 설정한다.</p> <ul style="list-style-type: none"> ◦ none : 아무것도 하지 않는다. (기본값) ◦ create-tables : 존재하지 않는 테이블을 생성하며, 기존 테이블은 그대로 남겨둔다. ◦ drop-and-create-tables : 기존에 존재하는 테이블을 삭제하고 다시 생성한다. ◦ create-or-extend-tables : 테이블을 생성하되, 기존에 존재하는 테이블이 있으면 없는 column만 추가한다. |

DDL을 생성할 때 기본적으로 다음과 같이 Java 타입을 해당 데이터베이스 SQL 타입에 맞게 생성한다.

| 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.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) |

| Java Type | Derby, JavaDB, Cloudscape | Oracle | DB2 | Sybase | SQLServer | MySQL |
|--|---------------------------------|----------|-------------|----------|-----------|-------------|
| 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

JPA에서는 기본적으로 Persistence Context라고 하는 1st-level Caching을 지원하고 있다. 하지만, 일반적으로 Persistence Context는 트랜잭션별로 새로 생성되기 때문에(Extended Persistence Context는 이에 해당하지 않음) 트랜잭션 간에 Caching을 지원하지 않는다. 이를 보완하기 위해 Eclipse Link에서는 2nd-level Caching 기능을 제공한다.

2nd-level Caching은 EntityManagerFactory 레벨에서 지원되기 때문에, 동일한 EntityManagerFactory에서 생성된 모든 EntityManager의 경우 이 공유 Cache를 사용하게 된다. 즉, Persistence Context에 없는 Entity를 가져올 때 2nd-level Caching을 참고하여 존재하는 경우에 이를 가져온다. 따라서 반복적인 Read 작업의 경우 성능 향상을 가져오게 된다.

Caching 방식 설정 예

```
<?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>
```

다음은 Caching에 관련된 옵션에 대한 설명이다.

| 항목 | 설명 |
|-----------------------------------|--|
| eclipselink.cache.type.default | <p>Caching 방식을 설정한다.</p> <p>아래의 옵션들 이외의 옵션이나 옵션에 대한 자세한 설정은 EclipseLink User Guide의 Cache Type and Size 항목을 참고한다.</p> <ul style="list-style-type: none"> ◦ Full : 객체를 Hard reference로 caching한다. 따라서 Entity가 삭제되기 전까지 항상 Cache에 존재하게 된다. ◦ Weak : 객체를 Weak reference로 caching한다. Garbage Collection(GC)이 되면 사라진다. ◦ Soft : Weak와 유사하지만, 메모리가 부족할 때만 객체를 삭제한다. (기본값) ◦ NONE : 객체를 Cache에 저장하지 않는다. 사용을 권장하지 않으며 Caching 기능을 사용하고 싶지 않다면, eclipselink.cache.shared.default 또는 eclipselink.cache.shared.<ENTITY> 프로퍼티 설정을 권장한다. |
| eclipselink.cache.size.default | Cache에 저장될 최대 객체수를 설정한다. (기본값: 1000) |
| eclipselink.cache.shared.default | <p>공유 Cache를 사용할지 여부를 설정한다.</p> <ul style="list-style-type: none"> ◦ true : 객체가 공유 Cache에 저장된다. 모든 EntityManager는 Caching을 사용하게 된다. (기본값) ◦ false : 객체가 공유 Cache에 저장되지 않는다. 따라서, EntityManager별로 Cache를 공유하지 않는다. 이를 사용하면 2nd-level Caching을 하지 않는 효과가 있다. |
| eclipselink.cache.type.<ENTITY> | <p>Entity별로 Caching을 하는 방식을 지정한다.</p> <p><ENTITY>는 Entity 이름 또는 fully-qualified 클래스 이름이 될 수 있다. 해당 Entity와 관계를 가지는 모든 Entity는 같은 설정을 가져야 한다. eclipselink.cache.type.default의 값 설명과 동일하다.</p> |
| eclipselink.cache.size.<ENTITY> | <p>Entity별로 Cache 사이즈를 지정한다.</p> <p><ENTITY>는 Entity 이름 또는 fully-qualified 클래스 이름이 될 수 있다. eclipselink.cache.size.default의 값 설명과 동일하다.</p> |
| eclipselink.cache.shared.<ENTITY> | <p>Entity별로 공유 Cache를 사용할지를 지정한다.</p> <p><ENTITY>는 Entity 이름 또는 fully-qualified 클래스 이름이 될 수 있다. 해당 Entity와 관계를 가지는 모든 Entity는 동일한 설정을 가져야 한다. eclipselink.cache.shared.default의 값 설명과 동일하다.</p> |



2nd-level Caching이 사용될 때 주의할 것은 외부 애플리케이션에 의해 또는 직접 데이터베이스의 데이터를 변경하는 경우 해당 내용이 Caching에 반영되지 않는다는 점이다. 이런 경우 데이터베이스의 최신 값이 아닌 Caching된 값이 리턴되기 때문에 애플리케이션에 이를 고려해야 한다.

Caching 옵션을 애플리케이션 환경을 고려하여 적절히 설정하거나, EntityManager.refresh(), eclipselink.refresh Query 힌트 또는 locking(pessimistic/optimistic) 등을 사용하여 이런 문제를 피할 수 있다.

2.3. Query 힌트

Query 힌트는 Query 객체를 사용할 때 프로바이더에서 제공하는 기능을 사용하도록 한다.

다음 예제와 같이 Query를 실행할 때 설정할 수도 있고, Named Query를 사용하는 경우 @QueryHint Annotation을 사용하여 지정할 수도 있다.

Query 힌트 사용 예제

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

제공하는 Query 힌트는 다음과 같다.

| 항목 | 설명 |
|------------------------------|--|
| eclipselink.pessimistic-lock | SELECT할 때 Pessimistic Locking을 사용할지 여부를 설정한다. <ul style="list-style-type: none">◦ NoLock : 사용하지 않는다. (기본값)◦ Lock : SELECT ... FOR UPDATE 문을 사용해 Locking을 한다.◦ NoLockWait : SELECT ... FOR UPDATE NO WAIT 문을 사용해 Locking을 한다. |
| eclipselink.refresh | 데이터베이스에서 최신 값을 가져와 Caching을 업데이트할지 여부를 설정한다. <ul style="list-style-type: none">◦ true : 최신 값을 가져오고 Caching을 업데이트한다.◦ false : Cache에 있는 값을 사용한다. (기본값) |

2.4. Logging 설정

좀 더 자세한 로그를 보고 싶다면 Logging 레벨을 설정할 수 있다.

기본적으로 Logging 레벨은 JEUS 서버에 전체적으로 적용되는 레벨(기본적으로 INFO)을 따르게 되지만, 이를 Persistence Unit별로 변경할 경우 eclipselink.logging.level 프로퍼티로 설정할 수 있다.

Logger의 경우 Jakarta EE 모드에서는 기본적으로 JEUS에서 제공되는 Logger(JEUS Logger)를 사용하게 되며, Java SE 모드에서는 standard output으로 출력되는 DefaultLogger를 사용하게 된다. 이를 변경하려면 eclipselink.logging.logger 프로퍼티로 설정할 수 있다.

Logging 설정 예

```
<?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>
```

다음은 설정에 대한 자세한 설명이다.

| 항목 | 설명 |
|----------------------------|--|
| eclipselink.logging.level | Logging 레벨을 지정한다. <ul style="list-style-type: none">◦ OFF : Logging을 하지 않는다.◦ SEVERE◦ WARNING◦ INFO (기본값)◦ CONFIG◦ FINE : SQL 관련 정보를 조회하려면 이 레벨로 설정한다.◦ FINER◦ FINEST |
| eclipselink.logging.logger | 사용할 Logger를 지정한다. <ul style="list-style-type: none">◦ JEUSLogger : JEUS에서 제공되는 Logger이다. (Jakarta EE 모드인 경우 기본값)◦ DefaultLogger : 기본 standard output Logger이다. (Java SE 모드인 경우 기본값)◦ JavaLogger : java.util.logging Logger이다.◦ Custom class name : 별도의 Logger를 구현한 경우 지정한다. |



EclipseLink에 대한 자세한 설정은 [EclipseLink JPA 사용자 안내서](#)를 참고한다.

3. 프로바이더 변경

본 장에서는 JEUS의 기본 프로바이더를 변경하는 방법에 대해서 설명한다.

3.1. Persistence 프로바이더 변경

JEUS에서 기본적으로 제공되는 프로바이더가 아닌 다른 프로바이더를 사용하려면 JPA 스펙에서 제공하는 설정을 사용한다.

먼저 필요한 라이브러리들을 JEUS_HOME/lib/application에 복사하거나 또는 각 애플리케이션별로 패키징한다. 그리고 해당 프로바이더 클래스 이름을 persistence.xml의 <provider> 값에 설정하면, Persistence Unit별로 설정한 프로바이더를 사용한다.

프로바이더 클래스 이름과 관련 프로퍼티 값들도 각 프로바이더별로 다르므로 해당 문서를 참고하여 설정해야 한다. 예를 들어, Hibernate를 사용하는 경우 다음과 같이 설정한다.

Persistence 프로바이더 변경 예

```
<?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>
```

위와 같이 Persistence Unit별로 프로바이더를 변경할 수도 있지만, JEUS의 디폴트 프로바이더를 변경할 수 있다. JEUS를 구동시킬 때 시스템 프로퍼티 jeus.persistence.defaultProvider를 해당 프로바이더 클래스 이름으로 설정하면 프로바이더가 지정되지 않는 모든 Persistence Unit의 경우 해당 디폴트 프로바이더를 사용하게 된다.



JEUS 시스템 프로퍼티 설정은 "JEUS Server 안내서"나 "JEUS Reference 안내서"를 참고한다.

3.2. 사용 가능한 Persistence 프로바이더

다른 Persistence 프로바이더에 대해서는 다음의 각 사이트를 참고한다.

- Hibernate EntityManager

<http://www.hibernate.org>

- OpenJPA

<http://openjpa.apache.org>

- BEA Kodo

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