Spring Boot Application

스프링 부트 응용

JPA 설정 이해하기

한국기술교육대학교
온라인평생교육원

학습내용

- 영속성 유닛 설정
- 엔티티 매핑 설정
- 식별자 값 자동 증가

학습목표

- 영속성 유닛 설정과 관련한 XML 설정의 의미를 설명할 수 있다.
- 엔티티 매핑과 관련한 다양한 어노테이션의 의미를 설명할 수 있다.
- 식별자 값을 자동으로 증가시키는 전략을 설명하고 적용할 수 있다.

영속성 유닛 설정

1 영속성 유닛 설정하기

〈1〉 영속성 유닛과 EntityManager

- 애플리케이션에서 JPA를 이용하기 위해서는 반드시 EntityManager 객체가 필요
- EntityManager는 EntityManagerFactory로부터 얻어냄
- EntityManagerFactory를 생성할 때 영속성 유닛 정보가 필요함

1 영속성 유닛 설정하기

〈1〉 영속성 유닛과 EntityManager

- 영속성 유닛은 연동하려는 데이터베이스당 하나씩 설정함
- 다른 영속성 유닛과 식별하기 위해 유일한 이름을 name 속성으로 지정

영속성 유닛 설정

1 영속성 유닛 설정하기

⟨1⟩ 영속성 유닛과 EntityManager

persistence.xml	⟨persistence-unit name="JPATest"⟩
JPAClient	// EntityManager 생성 EntityManagerFactory emf = Persistence.createEntityManagerFactory("JPATest"); EntityManager em = emf.createEntityManager();

1 영속성 유닛 설정하기

⟨2⟩ 엔티티 클래스 등록

- JPA는 자신이 영속성 관리할 엔티티 클래스들을 인지해야 함
- 반드시 영속성 유닛에 엔티티 클래스들을 등록해야 함

영속성 유닛 설정

1 영속성 유닛 설정하기

〈2〉 엔티티 클래스 등록

```
<persistence-unit name="JPATest">
    <class>com.mycompany.entity.Board</class>
```

1 영속성 유닛 설정하기

〈3〉 데이터소스 설정

- JPA 구현체는 영속성 유닛에 설정된 데이터소스 설정을 참조하여 데이터베이스를 연결함

```
<properties>
    <!-- 필수 속성 -->
    <property name="javax.persistence.jdbc.driver" value="org.h2.Driver" />
    <property name="javax.persistence.jdbc.url" value="jdbc:h2:tcp://localhost/~/test" />
    <property name="javax.persistence.jdbc.user" value="sa" />
    <property name="javax.persistence.jdbc.password" value="" />
```

영속성 유닛 설정

1 영속성 유닛 설정하기

〈3〉 데이터소스 설정

- 데이터 소스 설정에 사용된 정보는 다음과 같음

프로퍼티	의미
javax.persistence.jdbc.driver	JDBC 드라이버 클래스
javax.persistence.jdbc.url	JDBC URL 정보
javax.persistence.jdbc.user	데이터베이스의 아이디
javax.persistence.jdbc.password	데이터베이스의 비밀번호

1 영속성 유닛 설정하기

〈4〉 Dialect 클래스 설정

ORM의 가장 큰 장점

- 데이터베이스 연동에 필요한 SQL 구문을 ORM에서 생성

영속성 유닛 설정

1 영속성 유닛 설정하기

⟨4⟩ Dialect 클래스 설정

- 데이터베이스에 따라서 **키 생성 방식**도 다르고 **지원되는 함수**도 다름
- 운영 과정에서 데이터베이스가 변경되면 데이터베이스마다 다른 부분들을 모두 수정해야 함

1 영속성 유닛 설정하기

⟨4⟩ Dialect 클래스 설정

- JPA는 특정 데이터베이스에 최적화된 SQL을 생성함
- 이를 위해 Dialect 설정이 필요함

```
<property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect" />
```

영속성 유닛 설정

1 영속성 유닛 설정하기

④ Dialect 클래스 설정

- JPA 구현체인 Hibernate가 제공하는 Dialect 클래스

데이터베이스	Dialect 클래스
DB2	org.hibernate.dialect.DB2Dialect
PostgreSQL	org.hibernate.dialect.PostgreDialect
MySQL	org.hibernate.dialect.MySQLDialect
Oracle(any version)	org.hibernate.dialect.OracleDialect
Oracle 9i/10g	org.hibernate.dialect.Oracle9Dialect

1 영속성 유닛 설정하기

④ Dialect 클래스 설정

- JPA 구현체인 Hibernate가 제공하는 Dialect 클래스

데이터베이스	Dialect 클래스
Sybase	org.hibernate.dialect.SybaseDialect
Microsoft SQL Server	org.hibernate.dialect.SQLServerDialect
SAP DB	org.hibernate.dialect.SAPDBDialect
H2	org.hibernate.dialect.H2Dialect

영속성 유닛 설정

1 영속성 유닛 설정하기

5 JPA 구현체 설정

- JPA를 사용하기 위해서는 JPA 구현체에 대한 설정도 필요함

```
    <!-- 옵션 -->
    <property name="hibernate.show_sql" value="true" />
    <property name="hibernate.format_sql" value="true" />
    <property name="hibernate.use_sql_comments" value="false" />
    <property name="hibernate.id.new_generator_mappings" value="true" />
    <property name="hibernate.hbm2ddl.auto" value="update" />
</properties>
```

1 영속성 유닛 설정하기

5 JPA 구현체 설정

- JPA 구현체와 관련된 설정의 의미

속성	의미
hibernate.show_sql	하이버네이트가 생성한 SQL을 콘솔에 출력함
hibernate.format_sql	하이버네이트가 생성한 SQL을 출력할 때, 보기 좋은 포맷으로 출력함
hibernate.use_sql_comments	SQL에 포함된 주석(Comment)도 같이 출력함
hibernate.id.new_generator_mappings	키 생성 전략을 사용함
hibernate.hbm2ddl.auto	테이블 생성(CREATE)이나 변경(ALTER), 삭제(DROP) 같은 DDL 구문을 자동으로 실행할지 지정함

영속성 유닛 설정

1 영속성 유닛 설정하기

〈5〉 JPA 구현체 설정

- hibernate.hbm2ddl.auto 속성값의 종류와 의미

속성값	의미
create	애플리케이션을 실행할 때, 기존 테이블을 삭제하고 엔티티에 설정된 매핑 정보를 참조하여 새로운 테이블을 생성함(DROP → CREATE)
create-drop	create와 같지만 애플리케이션이 종료되기 직전에 생성된 테이블을 삭제함(DROP → CREATE → DROP)
update	• 기존에 사용 중인 테이블이 있으면 테이블을 생성하지 않고 재사용함 • 없을 때만 새롭게 생성함 • 만약 엔티티 클래스의 매핑 설정이 변경되면 변경된 내용만 반영함(ALTER)

엔티티 매핑 설정

1 엔티티 매핑

〈1〉 @Entity와 @Id

- 클래스 선언부에 @Entity를 설정하면 JPA는 이 클래스로부터 생성된 객체를 엔티티로 사용할 수 있음

1 엔티티 매핑

〈1〉 @Entity와 @Id

- 메모리에 생성된 엔티티 객체는 반드시 다른 엔티티와 식별할 수 있어야 함

엔티티 매핑 설정

1 엔티티 매핑

① @Entity와 @Id

식별자 필드

- 엔티티 객체가 가지고 있는 Primary Key 칼럼과 매핑될 식별자 변수
- JPA는 @Id를 이용해서 식별자 필드를 매핑함

```
@Entity
public class Board {
    @Id
    private Long seq;

    private String title;
    private String writer;
    private String content;
    private Date createDate;
    private Long cnt;
}
```

1 엔티티 매핑

② @Table

- 엔티티 이름과 테이블 이름이 다른 경우 @Table을 이용하여 매핑할 테이블 이름을 지정함

엔티티 매핑 설정

1 엔티티 매핑

2 @Table

- @Table이 지원하는 속성

속성	의미
name	매핑될 테이블 이름을 지정함
catalog	데이터베이스 카탈로그(catalog)를 지정함
schema	데이터베이스 스키마(schema)를 지정함
uniqueConstraints	결합 unique 제약조건을 지정함

1 엔티티 매핑

2 @Table

- @Table이 지원하는 속성

```
@Entity
@Table(name="E_BOARD",
       uniqueConstraints={@UniqueConstraint(columnNames={"SEQ", "WRITER"})})
@Data
public class Board {
    @Id
    @GeneratedValue
    private Long seq;
    private String title;
    private String writer;
```

엔티티 매핑 설정

1 엔티티 매핑

3 @Column

- 엔티티 변수와 테이블의 칼럼을 매핑할 때 사용함
- 생략하면 변수 이름과 동일한 컬럼이 매핑됨

1 엔티티 매핑

3 @Column

- @Column이 지원하는 속성

속성	설명
name	칼럼 이름을 지정함(생략 시 프로퍼티명과 동일하게 매핑)
unique	unique 제약조건을 추가함(기본 값 : false)
nullable	null 상태 허용 여부를 설정함(기본 값 : false)
insertable	INSERT를 생성할 때 이 칼럼을 포함할 것인지 결정함(기본 값 : true)
updatable	UPDATE를 생성할 때 이 칼럼을 포함할 것인지 결정함(기본 값 : true)

엔티티 매핑 설정

1 엔티티 매핑

3 @Column

- @Column이 지원하는 속성

속성	설명
columnDefinition	이 칼럼에 대한 DDL 문을 직접 기술함
length	문자열 타입의 칼럼 길이를 지정함(기본 값 : 255)
precision	숫자 타입의 전체 자릿수를 지정함(기본 값 : 0)
scale	숫자 타입의 소수점 자릿수를 지정함(기본 값 : 0)

1 엔티티 매핑

3 @Column

- @Column 사용 예

사용	생성된 DDL
@Column(nullable = false) private String title;	title varchar(255) not null
@Column(unique = true) private String isbn;	alter table Tablename add constraint UK_Xxx unique (isbn)
@Column(columnDefinition = "varchar(100) default 'No Contents'") private String content;	content varchar(100) default 'No Contents'

엔티티 매핑 설정

1 엔티티 매핑

〈3〉 @Column

- @Column 사용 예

사용	생성된 DDL
@Column(length = 40) private String writer;	writer varchar(40)

@Column(precesion = 10, scale = 2)
private BigDecimal point;
	point numeric(10, 2) // H2, PostgreSQL
	point number(10, 2) // Oracle
	point decimal(10, 2) // MySQL

1 엔티티 매핑

〈4〉 @Temporal

- java.util.Date 타입의 날짜 데이터를 매핑할 때 사용함

엔티티 매핑 설정

1 엔티티 매핑

④ @Temporal

TemporalType을 이용하여 날짜 형식을 지정할 수 있음

TemporalType.DATE	날짜 정보만 출력
TemporalType.TIME	시간 정보만 출력
TemporalType.TIMESTAMP	날짜와 시간 정보를 모두 출력

1 엔티티 매핑

④ @Temporal

```
@Entity
@Table(name="E_BOARD",
       uniqueConstraints={@UniqueConstraint(columnNames={"SEQ", "WRITER"})})
@Data
public class Board {
    @Id
    @GeneratedValue
    private Long seq;
    private String title;
    private String writer;
    private String content;

    @Temporal(TemporalType.DATE)
    private Date createDate;

    private Long cnt;
}
```

엔티티 매핑 설정

1 엔티티 매핑

5 @Transient

- 엔티티 클래스에서 매핑되는 칼럼이 없거나 임시로 사용하는 변수는 매핑에서 제외해야 함

1 엔티티 매핑

5 @Transient

- @Transient는 엔티티 클래스의 특정 변수를 영속 필드에서 제외할 때 사용함

```
@Entity
@Table(name="E_BOARD",
       uniqueConstraints={@UniqueConstraint(columnNames={"SEQ", "WRITER"})})
@Data
public class Board {
    @Id
    @GeneratedValue
    private Long seq;
    private String title;
    private String content;

    @Transient
    private String searchCondition;
    @Transient
    private String searchKeyword;
}
```

식별자 값 자동 증가

1 식별자 값 자동 증가

⟨1⟩ 식별자 필드와 자동 증가

- 엔티티에서 식별자로 지정된 필드에 값이 없으면 예외 발생
- 중복되는 값이 설정되어도 예외 발생

1 식별자 값 자동 증가

⟨1⟩ 식별자 필드와 자동 증가

- 식별자 값은 사용자가 직접 설정할 수도 있지만 일반적으로 서브 쿼리나 시퀀스를 이용하여 자동으로 증가시킴

식별자 값 자동 증가

1 식별자 값 자동 증가

⟨1⟩ 식별자 필드와 자동 증가

@GeneratedValue

- 식별자 필드에 자동으로 증가된 값을 할당할 때 사용하는 어노테이션

1 식별자 값 자동 증가

⟨2⟩ @GeneratedValue의 속성

- @GeneratedValue는 속성 두 개를 사용할 수 있음

속성	의미
strategy	자동 생성 전략을 선택함 (GenerationType 지정)
generator	이미 생성된 키 생성기를 참조함

식별자 값 자동 증가

1 식별자 값 자동 증가

〈2〉 @GeneratedValue의 속성

- 식별자 값 생성 전략

PK 전략	의미
GenerationType.TABLE	테이블을 사용하여 PK 값을 생성함 PK 값 생성을 위한 별도의 테이블이 필요함
GenerationType.SEQUENCE	시퀀스를 이용하여 PK 값을 생성함 시퀀스를 지원하는 데이터베이스에서만 사용할 수 있음
GenerationType.IDENTITY	auto_increment나 IDENTITY를 이용하여 PK 값을 생성함 MySQL이나 H2 같은 데이터베이스를 이용할 때 사용함
GenerationType.AUTO	데이터베이스에 맞는 PK 값 생성 전략을 선택함(기본 설정)

1 식별자 값 자동 증가

〈3〉 테이블 전략

- PK 값만 저장하는 테이블을 생성하기 위해 @TableGenerator를 사용해야 함

```
@Data
@Entity
@TableGenerator(name = "BOARD_SEQ_GENERATOR",
                table = "ALL_SEQUENCES",
                pkColumnValue = "BOARD_SEQ",
                initialValue = 0,
                allocationSize = 1)
public class Board {
    @Id
    @GeneratedValue(strategy = GenerationType.TABLE,
                    generator = "BOARD_SEQ_GENERATOR")
    private Long seq;
```

키 생성 테이블

```
SELECT * FROM ALL_SEQUENCES;
```

SEQUENCE_NAME	NEXT_VAL
BOARD_SEQ	0

(1 row, 3 ms)

식별자 값 자동 증가

1 식별자 값 자동 증가

〈3〉 테이블 전략

- @TableGenerator의 속성과 의미

name	생성기 이름을 지정
table	키 생성 테이블 이름을 지정
pkColumnValue	PK 컬럼에 저장할 이름 지정
initialValue	초기값 지정
allocationSize	자동으로 증가시킬 값 지정

1 식별자 값 자동 증가

〈3〉 테이블 전략

- @GeneratedValue의 속성과 의미

strategy	식별자 생성 전략을 지정
generator	식별자 생성기 지정

식별자 값 자동 증가

1 식별자 값 자동 증가

④ 시퀀스 전략

- 시퀀스(SEQUENCE) 전략은 키를 생성하기 위해 테이블이 아닌 시퀀스를 사용

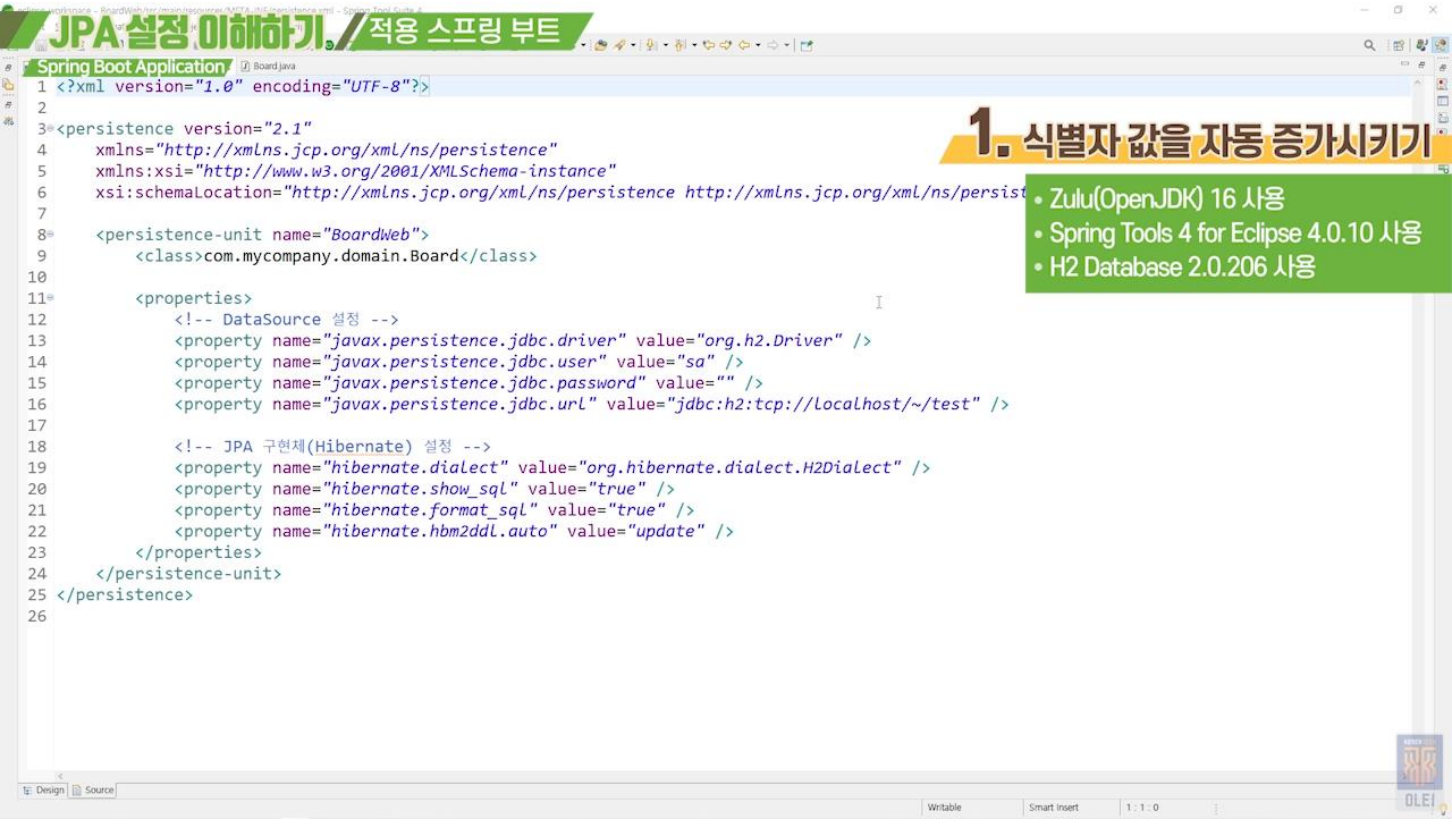
1 식별자 값 자동 증가

④ 시퀀스 전략

- @SequenceGenerator를 이용하여 사용할 시퀀스를 정의함
- @SequenceGenerator와 @GeneratedValue의 속성과 의미는 테이블 전략과 동일함

```
@Data
@Entity
@SequenceGenerator(name = "BOARD_SEQ_GENERATOR",
                   sequenceName = "BOARD_SEQUENCE",
                   initialValue = 1,
                   allocationSize = 1)
public class Board {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE,
                    generator = "BOARD_SEQ_GENERATOR")
```

적용 스프링 부트




```xml
<?xml version="1.0" encoding="UTF-8"?>

<persistence version="2.1"
    xmlns="http://xmlns.jcp.org/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persist

    <persistence-unit name="BoardWeb">
        <class>com.mycompany.domain.Board</class>

        <properties>
            <!-- DataSource 설정 -->
            <property name="javax.persistence.jdbc.driver" value="org.h2.Driver" />
            <property name="javax.persistence.jdbc.user" value="sa" />
            <property name="javax.persistence.jdbc.password" value="" />
            <property name="javax.persistence.jdbc.url" value="jdbc:h2:tcp://localhost/~/test" />

            <!-- JPA 구현체(Hibernate) 설정 -->
            <property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect" />
            <property name="hibernate.show_sql" value="true" />
            <property name="hibernate.format_sql" value="true" />
            <property name="hibernate.hbm2ddl.auto" value="update" />
        </properties>
    </persistence-unit>
</persistence>
```

실습단계
JPA가 무조건 로딩하는 파일
이름이 BoardWeb이라는 객체가 메모리에 뜸
영속성 유닛 이름이 중요함
Entity class가 등록됨
데이터 소스 설정 필요
SQL을 보여줄 것인가?
포맷을 어느 정도 맞춰 줄 것인가?
중요! hbm2ddl.auto 설정 : 없다고 해서 테이블을 만들지 않는 것이 아님
있으면 재사용, 없으면 만들게 됨
필수! 이 클래스가 엔티티 클래스임을 알려 줌. 반드시 필요함
엔티티가 설정된 클래스에는 @ID를 통해 식별자 필드를 매핑할 수 있어야 함
수많은 변수를 식별자 값을 기준으로 세팅
프라이머리 키에 해당하는 변수 위에는 @ID를 설정해야 함
디폴트는 AUTO

적용 스프링 부트

실습단계
H2에 최적화된 전략 사용
시퀀스 전략을 쓰고 싶은 경우
생성할 시퀀스 객체 이름 설정
초기값 1, 1씩 증가시키라는 뜻
시퀀스로 변경
Generator 속성 추가, 이름을 그대로 등록
Insert 기능의 소스로 변경
코드 제거
엔티티 매니저로부터 트랜젝션 객체 획득
트랜젝션 스타트
Insert 처리 후 트랜젝션 종료
중요! 등록, 수정, 삭제 작업은 트랜젝션 안에서 이루어져야 함
중요! 유니크한 값을 뽑아냈다는 것
식별자 값을 자동으로 증가시키는 전략
신경쓰고 싶지 않을 경우 AUTO 전략 사용