웹 애플리케이션 만들기 - 헬로샵

스프링 프레임워크와 JPA를 사용해서 실제 웹 애플리케이션을 만들어보자. 진행 순서는 다음과 같다.

웹 애플리케이션 만들기 진행 순서

1. 프로젝트 환경 설정
2. 도메인 모델과 테이블 설계
3. 애플리케이션 기능 구현

먼저 스프링 프레임워크와 JPA를 사용해서 웹 애플리케이션을 개발할 수 있도록 개발 환경을 설정하겠다. 그 다음으로 요구사항을 분석해서 도메인 모델과 테이블을 설계하고 실제 애플리케이션 기능을 구현하는 순서로 진행하겠다.

예제에 사용하는 기술은 다음과 같다.

- 뷰 : JSP, JSTL
- 웹 계층 : 스프링 MVC
- 데이터 저장 계층 : JPA, 하이버네이트
- 기반 프레임워크 : 스프링 프레임워크
- 빌드 : 메이븐(Maven)

독자분들이 JSP, 스프링 프레임워크, 메이븐에 대해서는 약간씩 지식이 있다고 가정하고 중간마다 간단히 언급하는 정도로만 진행하겠다.

예제에서는 메이븐이 설치되어 있다고 가정한다. (이클립스 LUNA 버전에는 메이븐이 내장되어 있다.)

> **스프링 프레임워크[1]**
>
> 사실상 표준으로 불릴 정도로 최근 자바 애플리케이션들은 스프링 프레임워크를 많이 사용한다.
>
> 지금까지는 순수한 자바 환경(J2SE)에서 JPA를 사용했다. 따라서 데이터베이스 커넥션과 트랜잭션 처리도 JPA가 제공하는 기능을 직접 사용했다. 스프링 프레임워크와 JPA를 사용한다는 것은 스프링 컨테이너 위에서 JPA를 사용한다는 의미다. 따라서 스프링 컨테이너가 제공하는 데이터베이스 커넥션과 트랜잭션 처리 기능을 사용할 수 있다. 물론 직접 사용할 때보다 편리하다.
>
> 스프링 프레임워크만 설명해도 이미 이 책의 분량을 훌쩍 뛰어 넘는다. (스프링 프레임워크만 다루는 전문 서적중에는 1400 페이지가 넘는 것도 있다.) 여기서는 스프링 프레임워크의 기능 중에 JPA와 관련 있는 부분에 초점을 맞추겠다.

프로젝트 실행하기

> **예제 코드 : `ch17-jpa-shop`**

예제 프로젝트는 웹 애플리케이션이다. 따라서 웹 서버를 실행해야 하는데 여기서는 메이븐의 톰캣 플러그인을 사용하겠다. (참고로 `pom.xml` 에 `tomcat7-maven-plugin` 으로 톰캣 플러그인을 이미 설정해 두었다.)

이클립스에서 `ch17-jpa-shop` 프로젝트를 선택한 상태에서 메뉴의 `Run` -> `Run As` -> `Maven Build...` 을 선택하자. 그리고 `Goals` 에 `tomcat7:run` 이라고 입력하고 `Run` 버튼을 선택하자.

기다리면 콘솔에 다음과 같이 나오면서 `http://localhost:8080/` 로 서버가 실행된다.

```
...
[INFO] --- tomcat7-maven-plugin:2.2:run (default-cli) @ jpa-shop ---
[INFO] Running war on http://localhost:8080/
[INFO] Using existing Tomcat server configuration at /Users/hello/jpabook-source
[INFO] create webapp with contextPath:
```

웹 브라우저를 열고 `http://localhost:8080/` 를 입력하면 예제 화면이 나타난다.

프로젝트 환경설정

최근 자바 프로젝트는 규모도 커지고 여러 라이브러리를 함께 사용하므로 프로젝트 환경 설정부터 간단하지 않다. 지금부터 웹 애플리케이션 예제에서 사용할 프로젝트 환경을 어떻게 구성하는지 다음 순서로 알아보자.

프로젝트 환경설정 진행 순서

1. 프로젝트 구조 분석
2. 메이븐과 라이브러리 설정
3. 스프링 프레임워크 설정

먼저 프로젝트 폴더 구조를 분석하고, 메이븐에 사용 라이브러리를 지정한 다음 스프링 프레임워크 환경을 설정하는 순서로 진행하겠다.

- 프로젝트 구조

예제는 메이븐이 제공하는 표준 프로젝트 구조를 사용한다. 다음 프로젝트 구조를 보자.

====== 프로젝트 구조 =====

```
jpashop (프로젝트 루트)
├── src (소스 폴더)
|   ├── main (실행 코드)
|   |   ├── java      (자바 소스코드)
|   |   ├── resources (리소스)
|   |   └── webapp    (웹 폴더)
|   └── test (테스트 코드)
├── target  (빌드 결과)
└── pom.xml (메이븐 설정 파일)
```

- 메이븐과 사용 라이브러리 관리

프로젝트 루트에 있는 메이븐 설정 파일인 `pom.xml`을 열어서 현재 프로젝트 정보와 사용할 라이브러리를 지정하자.

====== `pom.xml` ======

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
    <modelVersion>4.0.0</modelVersion>

    <groupId>jpabook</groupId>
```

```
    <artifactId>jpashop</artifactId>
    <version>1.0-SNAPSHOT</version>
    <name>jpashop</name>
    <packaging>war</packaging>

    <dependencies>
        ... 생략
    </dependencies>

    <build>
        ... 생략
    </build>
</project>
```

`pom.xml`을 간단히 분석해보자.

- `<modelVersion>` : POM 모델 버전, 그대로 사용한다.
- `<groupId>` : 프로젝트 그룹명, 예) `org.springframework`
- `<artifactId>` : 프로젝트를 식별하는 아이디, 예) `spring-core`, `spring-mvc`
- `<version>` : 프로젝트 버전
- `<name>` : 프로젝트 이름
- `<packaging>` : 빌드 패키징 방법을 지정한다. 웹 애플리케이션은 `war`, 자바 라이브러리는 `jar`로 설정한다.
- `<dependencies>` : 사용할 라이브러리를 지정한다.
- `<build>` : 빌드 관련 정보를 설정한다.

전세계에는 수 많은 라이브러리가 있다. 라이브러리 간에 충돌을 피하려면 `groupId` + `artifactId`는 유일해야 한다.

다음으로 사용할 라이브러리를 지정하는 `<dependencies>` 부분을 보자.

====== `<dependencies>` ======

```
<dependencies>

    <!-- 스프링 MVC -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>4.1.6.RELEASE</version>
    </dependency>

    <!-- 스프링 ORM -->
    <dependency>
```

```
        <groupId>org.springframework</groupId>
        <artifactId>spring-orm</artifactId>
        <version>4.1.6.RELEASE</version>
    </dependency>

    <!-- JPA, 하이버네이트 -->
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-entitymanager</artifactId>
        <version>4.3.8.Final</version>
    </dependency>

    <!-- H2 데이터베이스 -->
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <version>1.4.186</version>
        <scope>runtime</scope>
    </dependency>

    <!-- 커넥션 풀 -->
    <dependency>
        <groupId>org.apache.tomcat</groupId>
        <artifactId>tomcat-jdbc</artifactId>
        <version>7.0.57</version>
        <scope>compile</scope>
    </dependency>

    <!-- WEB -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>3.1.0</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>javax.servlet.jsp</groupId>
        <artifactId>javax.servlet.jsp-api</artifactId>
        <version>2.3.1</version>
        <scope>provided</scope>
    </dependency>

    <!-- 로깅 SLF4J & LogBack -->
    <dependency>
        <groupId>org.slf4j</groupId>
```

```
            <artifactId>slf4j-api</artifactId>
            <version>1.7.6</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-classic</artifactId>
            <version>1.1.1</version>
            <scope>runtime</scope>
        </dependency>

        <!-- 테스트 -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
            <version>4.1.6.RELEASE</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>

    </dependencies>
```

`<dependencies>`에는 사용할 라이브러리를 지정한다. `groupId` + `artifactId` + `version`만 적어주면 라이브러리(jar 파일)를 메이븐 공식 저장소에서 자동으로 내려받아 라이브러리에 추가해준다.

프로젝트를 위해 많은 라이브러리가 필요하지만 핵심 라이브러리는 다음 3가지다.

핵심 라이브러리

- 스프링 MVC(`spring-webmvc`) : 스프링 MVC 라이브러리
- 스프링 ORM(`spring-orm`) : 스프링 프레임워크와 JPA를 연동하기 위한 라이브러리
- JPA, 하이버네이트(`hibernate-entitymanager`) : JPA 표준과 하이버네이트를 포함하는 라이브러리, `hibernate-entitymanager`를 라이브러리로 지정하면 다음 중요 라이브러리도 함께 내려받는다.
    - hibernate-core-{버전정보}.jar (하이버네이트 라이브러리)
    - hibernate-jpa–2.1-api-{버전정보}.jar (JPA 2.1 표준 인터페이스가 있는 라이브러리)

기타 라이브러리

- H2 데이터베이스 : H2 데이터베이스는 아주 작은 데이터베이스다. 별도의 설치 없이 JVM 메모리 안에서 동작하는 기능도 있어서 실습용으로 적절하다.
- 커넥션 풀 : `tomcat-jdbc` 커넥션 풀을 사용한다.
- WEB : 서블릿, JSP와 관련된 라이브러리
- 로깅 SLF4J & LogBack : 과거에는 Log4j가 많이 사용되었지만 최근에는 SLF4J + LogBack이 많이 사용된다. 기존에 비해 성능과 기능이 많이 향상되었다.
- 테스트 : 테스트용 라이브러리, `spring-test`는 스프링 프레임워크와 통합 테스트를 지원한다.

> **참고:** 하이버네이트 4.3부터 JPA 2.1을 지원한다.

정상 동작하는지 확인하기 위해 이클립스 메뉴에서 `Run` -> `Run As` -> `Maven Build...`을 선택하자.

그리고 `Goals`에 `package`라고 입력하고 `Run` 버튼을 선택하자.

콘솔에 다음과 같이 나왔으면 성공이다.

```
[INFO] Building war: .../jpashop/target/jpashop.war
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2.550s
[INFO] Finished at: Fri Dec 13 01:35:30 KST 2013
[INFO] Final Memory: 8M/20M
[INFO] ------------------------------------------------------------------------
```

`target` 폴더에 가보면 빌드 결과로 `jpashop.war` 파일이 생성되어 있다.

메이븐을 사용해서 필요한 라이브러리도 설정하고 빌드도 성공했다. 다음으로 스프링 프레임워크와 JPA를 실행하기 위한 설정을 진행해보자.

> **메이븐 `<dependency>`의 `<scope>` 설정**
>
> - compile (기본값) : 컴파일시에 라이브러리를 사용한다.
> - runtime : 실행시에 라이브러리를 사용한다.
> - provided : 외부에서 라이브러리가 제공된다. 컴파일시에 사용하지만 빌드에 포함하지 않는다. 보통 JSP, Servlet 라이브러리들에 사용한다.
> - test : 테스트 코드에만 사용한다.

> **의존성 전이:** `spring-mvc` 라이브러리를 사용하려면 `spring-core` 라이브러리가 필요하다. 이것을 `spring-mvc` 는 `spring-core` 에 의존한다고 표현한다. 메이븐은 의존관계가 있는 라이브러리도 함께 내려받아 라이브러리에 자동으로 추가하는데 이것을 의존성 전이(Transitive dependency)라 한다.

> **참고:** 메이븐 공식 저장소에 등록된 라이브러리는 `http://search.maven.org/` 에서 검색할 수 있다.

- 스프링 프레임워크 설정

지금부터 웹 애플리케이션을 실행하기 위한 환경설정을 진행해보자. 다음 프로젝트 구조를 보자.

```
jpashop
└── src
    └── main (실행 코드)
        ├── java      (자바 소스코드)
        |   └── jpabook
        |       └── jpashop
        |           ├── domain       //도메인 계층
        |           ├── repository   //데이터 저장 계층
        |           ├── service      //서비스 계층
        |           └── web          //웹 계층
        ├── resources (리소스)
        |   └── appConfig.xml     // 3. 스프링 애플리케이션 관련 설정 //**
        |   └── webAppConfig.xml // 2. 스프링 웹 관련 설정 //**
        └── webapp    (웹 폴더)
            └── WEB-INF
                └── web.xml // 1. 웹 애플리케이션 환경 설정 파일 //**
```

프로젝트 환경 설정파일은 다음과 같다.

1. `web.xml` : 웹 애플리케이션 환경 설정 파일
2. `webAppConfig.xml` : 스프링 웹 관련 환경 설정 파일
3. `appConfig.xml` : 스프링 애플리케이션 관련 환경 설정 파일

순서대로 하나씩 알아보자.

====== `src/main/webapp/web.xml` ======

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.c
         version="3.0" metadata-complete="true">

    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</l
    </listener>

    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:appConfig.xml</param-value> //1. appConfig //**
    </context-param>

    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servle
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:webAppConfig.xml</param-value> //2. webAppCon
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

</web-app>
```

`web.xml`을 보면 `appConfig.xml`을 설정하는 부분과 `webAppConfig.xml`을 설정하는 부분으로 나뉘어 있는데 스프링 프레임워크를 설정할 때 보통 웹 계층과 비즈니스 도메인 계층을 나누어 관리한다.

- `webAppConfig.xml` : 스프링 MVC 설정을 포함해서 웹 계층을 담당한다.
- `appConfig.xml` : 비즈니스 로직, 도메인 계층, 서비스 계층, 데이터 저장 계층을 담당한다.

====== `src/main/resources/webAppConfig.xml` ======

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:mvc="http://w
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://ww
```

```
    <mvc:annotation-driven/>

    <context:component-scan base-package="jpabook.jpashop.web"/>

    <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalR
        <property name="viewClass" value="org.springframework.web.servlet.view.J
        <property name="prefix" value="/WEB-INF/jsp/"/>
        <property name="suffix" value=".jsp"/>
    </bean>

</beans>
```

`webAppConfig.xml` 코드를 분석해보자.

- `<mvc:annotation-driven>` : 스프링 MVC 기능을 활성화 한다.
- `<context:component-scan>` : `basePackages` 를 포함한 하위 패키지를 검색해서 `@Component` , `@Service` , `@Repository` , `@Controller` 어노테이션이 붙어 있는 클래스들을 스프링 빈으로 자동 등록한다. 여기서는 웹과 관련된 `jpabook.jpashop.web` 패키지를 검색해서 스프링 빈으로 등록한다. 이 패키지에는 컨트롤러가 있다.
- `<bean>` : 스프링 빈을 등록한다.

JSP, JSTL을 사용하도록 뷰 리졸버(`viewResolver`)를 스프링 빈을 등록했다.

다음으로 가장 중요한 `appConfig.xml` 설정을 살펴보자.

====== `src/main/resources/appConfig.xml` ======

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context" xmlns:tx="h
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://ww

    <tx:annotation-driven/>

    <context:component-scan base-package="jpabook.jpashop.service, jpabook.jpash

    <bean id="dataSource" class="org.apache.tomcat.jdbc.pool.DataSource">
        <property name="driverClassName" value="org.h2.Driver"/>
        <property name="url" value="jdbc:h2:mem:jpashop"/>
        <property name="username" value="sa"/>
        <property name="password" value=""/>
    </bean>

    <bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransact
```

```
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <bean class="org.springframework.dao.annotation.PersistenceExceptionTranslat

    <bean id="entityManagerFactory" class="org.springframework.orm.jpa.LocalCont
        <property name="dataSource" ref="dataSource"/>
        <!-- @Entity 탐색 시작 위치 -->
        <property name="packagesToScan" value="jpabook.jpashop.domain"/>
        <property name="jpaVendorAdapter">
            <!-- 하이버네이트 구현체 사용 -->
            <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAd
        </property>
        <property name="jpaProperties"> <!-- 하이버네이트 상세 설정 -->
            <props>
                <prop key="hibernate.dialect">org.hibernate.dialect.H2Dialect</p
                <prop key="hibernate.show_sql">true</prop>                  <!-
                <prop key="hibernate.format_sql">true</prop>                <!-
                <prop key="hibernate.use_sql_comments">true</prop>          <!-
                <prop key="hibernate.id.new_generator_mappings">true</prop> <!-
                <prop key="hibernate.hbm2ddl.auto">create</prop>            <!-
            </props>
        </property>
    </bean>

</beans>
```

순서대로 분석해보자.

`<tx:annotation-driven/>` : 스프링 프레임워크가 제공하는 어노테이션 기반의 트랜잭션 관리자를 활성화한다. 이 기능은 `@Transactional` 이 붙은 곳에 트랜잭션을 적용한다.

```
<bean id="dataSource" class="org.apache.tomcat.jdbc.pool.DataSource">
    <property name="driverClassName" value="org.h2.Driver"/>
    <property name="url" value="jdbc:h2:mem:jpashop"/>
    <property name="username" value="sa"/>
    <property name="password" value=""/>
</bean>
```

데이터베이스에 접근할 데이터소스를 등록한다. 여기서는 H2 데이터베이스의 접속 URL을 `jdbc:h2:mem:...` 으로 설정해서 JVM 안에서 동작하는 인 메모리 데이터베이스로 사용한다. 인 메모리 데이터베이스를 사용하면 별도의 데이터베이스 서버를 실행하지 않아도 된다. 이제 애플리케이션을 시작할 때 데이터베이스도 애플리케이션안에서 함께 실행되고 애플리케이션을 종료할 때 데이터베이스도 함께 사라진다.

> **참고**: 서버 모드 설정
>
> H2 데이터베이스를 서버 모드로 접근하려면 url 속성을 다음처럼 변경하면 된다.
> `jdbc:h2:tcp://localhost/~/jpashop;MVCC=TRUE`

```
<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionM
    <property name="dataSource" ref="dataSource"/>
</bean>
```

트랜잭션 관리자를 등록한다. 일반적으로 `org.springframework.jdbc.datasource.DataSourceTransactionManager`를 트랜잭션 관리자로 사용하지만 JPA를 사용하려면 `org.springframework.orm.jpa.JpaTransactionManager`를 트랜잭션 관리자로 등록해야 한다. 이 트랜잭션 관리자는 `DataSourceTransactionManager`가 하던 역할도 수행하므로 JPA뿐만 아니라 JdbcTemplate, MyBatis와 함께 사용할 수 있다.

```
<bean class="org.springframework.dao.annotation.PersistenceExceptionTranslationP
```

`PersistenceExceptionTranslationPostProcessor` : `@Repository` 어노테이션이 붙어 있는 스프링 빈에 예외 변환 AOP를 적용한다. 이 AOP는 JPA 예외를 스프링 프레임워크가 추상화한 예외로 변환해준다.

====== JPA 설정 - 엔티티 매니저 팩토리 등록 ======

```
<bean id="entityManagerFactory" class="org.springframework.orm.jpa.LocalContaine
    <property name="dataSource" ref="dataSource"/>
    <!-- @Entity 탐색 시작 위치 -->
    <property name="packagesToScan" value="jpabook.jpashop.domain"/>
    <property name="jpaVendorAdapter">
        <!-- 하이버네이트 구현체 사용 -->
        <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapte
    </property>
    <property name="jpaProperties"> <!-- 하이버네이트 상세 설정 -->
        <props>
            <prop key="hibernate.dialect">org.hibernate.dialect.H2Dialect</prop>
            <prop key="hibernate.show_sql">true</prop>                    <!-- SQ
            <prop key="hibernate.format_sql">true</prop>                  <!-- SQ
            <prop key="hibernate.use_sql_comments">true</prop>            <!-- SQ
            <prop key="hibernate.id.new_generator_mappings">true</prop>   <!-- 새
            <prop key="hibernate.hbm2ddl.auto">create</prop>              <!-- DD
```

```
        </props>
      </property>
  </bean>
```

스프링 프레임워크에서 JPA를 사용하려면 스프링 프레임워크가 제공하는 `LocalContainerEntityManagerFactoryBean`을 스프링 빈으로 등록해야 한다. 순수하게 자바만 사용하는 J2SE 환경에서는 `persistence.xml`에 엔티티 매니저 팩토리 정보를 설정하지만 스프링 프레임워크에서 JPA를 사용하려면 예제처럼 스프링 프레임워크가 제공하는 방식으로 엔티티 매니저 팩토리를 등록해야 한다. 여기서 필요한 설정을 다 할 수 있기 때문에 `persistence.xml`이 없어도 동작한다. 참고로 `persistenceUnitName` 속성을 사용해서 영속성 유닛 이름을 지정하면 `persistence.xml`에 설정한 정보도 사용할 수 있다.

- `LocalContainerEntityManagerFactoryBean` : JPA를 스프링 컨테이너에서 사용할 수 있도록 스프링 프레임워크가 제공하는 기능이다. 이 클래스는 `spring-orm` 라이브러리가 제공한다.
- `dataSource` : 사용할 데이터소스를 등록한다.
- `packagesToScan` : `@Entity`가 붙은 클래스를 자동으로 검색하기 위한 시작점을 지정한다.
- `persistenceUnitName` : 영속성 유닛 이름을 지정한다. 예제처럼 `persistenceUnitName` 이름을 설정하지 않으면 `default`라는 이름의 영속성 유닛을 생성한다.
- `jpaVendorAdapter` : 사용할 JPA 벤더를 지정한다. 여기서는 하이버네이트를 구현체로 사용하므로 `HibernateJpaVendorAdapter`를 등록했다.

하이버네이트 상세 설정

`jpaProperties`를 사용해서 상세 프로퍼티를 설정할 수 있다.

- `hibernate.dialect` : 사용할 데이터베이스 방언을 설정한다. 여기서는 H2 데이터베이스 방언을 지정했다.
- `hibernate.show_sql` : 실행하는 SQL을 콘솔에 출력한다.
- `hibernate.format_sql` : SQL을 보기 좋게 정리해서 출력한다.
- `hibernate.use_sql_comments` : SQL을 출력할 때 어떻게 실행된 SQL인지 또는 사용자가 설정한 코멘트를 남긴다.
- `hibernate.id.new_generator_mappings` : JPA에 맞춘 새로운 ID 생성 방법을 사용한다. 하이버네이트 레거시를 운영하는 것이 아니면 항상 `true`로 설정해야 한다.
- `hibernate.hbm2ddl.auto` : 애플리케이션이 시작될 때 테이블과 기타 DDL을 자동으로 생성한다. 여기에는 4가지 옵션이 있다.
  - `create` : 기존 DDL을 제거하고 새로 생성한다.
  - `create-drop` : `create`와 같은데 애플리케이션을 종료할 때 생성한 DDL을 제거한다.
  - `update` : 현재 데이터베이스 DDL과 비교해서 변경사항만 수정한다.

- `validate` : 현재 엔티티 매핑 정보와 데이터베이스 스키마가 같은지 비교한다. 만약 다르면 경고를 남기고 애플리케이션을 실행하지 않는다. 이 설정은 DDL을 변경하지 않는다.

> **hibernate.id.new_generator_mappings 주의사항:** 이 옵션을 설정하지 않으면 하이버네이트는 과거 버전에서 사용했던 키 생성 전략을 사용하게 된다. 하이버네이트 공식 문서는 `true`로 설정해서 JPA 표준에 맞춘 새로운 키 생성 전략을 권장 한다. 하이버네이트는 과거 버전과 호환을 위해 신규 개발자에게 이 옵션을 설정하도록 했다.

> **참고:** JPA의 동작 환경은 순수 자바인 J2SE와 J2EE 표준 컨테이너 위에서 동작하는 것으로 나눌 수 있다. 스프링 프레임워크는 `LocalContainerEntityManagerFactoryBean`를 사용해서 J2SE 환경의 JPA를 마치 표준 컨테이너 위에서 동작하는 것처럼 에뮬레이션한다.

> **참고:** 하이버네이트 SQL 로그를 콘솔이 아닌 로거를 통해서 남기려면 `logback.xml`에 다음처럼 설정하면 된다. 이렇게 로거를 설정하면 `hibernate.show_sql` 옵션을 꺼야 콘솔에 로그가 중복 출력되지 않는다.
>
> - org.hibernate.SQL: `hibernate.show_sql` 속성과 거의 같은 로그를 남긴다.
> - org.hibernate.type: 실행된 SQL에 바인딩된 파리미터 정보를 로그로 남긴다.
>
> `logback.xml` 설정
>
> ```
> <logger name="org.hibernate.SQL" level="DEBUG">...</logger>
> <logger name="org.hibernate.type" level="TRACE">...</logger>
> ```

정리

드디어 스프링 프레임워크와 JPA를 사용해서 웹 애플리케이션을 개발하기 위한 환경 설정이 끝났다. 스프링 프레임워크에 관한 내용이 많아서 스프링 프레임워크를 잘 알고 있다면 내용을 쉽게 이해했겠지만 그렇지 않으면 쉽지 않았을 것이다.

---

1. http://projects.spring.io/spring-framework ↩