JPA (Java Persistance API)의 소개

JPA는 자바 진영의 ORM 기술 표준

1. ORM(Object Relational Mapping) 이란?

객체와 관계형 데이터베이스를 매핑한다. ibatis나 mybatis는 ORM이 아니다. SQL 구문을 Mapping 하여 실행하는 매퍼이지 ORM이 아니다.

JPA는 ORM 표준 기술로 Hibernate, OpenJPA, EclipseLink, TopLink Essentials과 같은 구현체가 있고 이에 표준 인터페이스가 바로 JPA이다.

ORM 기술 표준을 구현한 프레임워크는 하이버네이트가 대표적이다. 우리는 Spring-DATA-JPA를 이용하여 객체 관점에서 DB에 접근하는 형태로 어플리케이션을 개발할 것이고 그 구현기술로 하이버네이트를 적용할 예정이다.

2. JPA 의 장점

- 생산성 : 반복적인 SQL을 작업과 CRUD 작업을 개발자가 직접안해도 된다.
- 유지보수 : 객체의 수정에 따른 SQL 수정 작업을 개발자가 직접안해도 된다.
- 패러다임 불일치 해결 : 객체와 관계형 데이터베이스를 매핑하는 과정에서 발생하는 문제를 개발자가 직접 안해도 된다.
- 성능 : 캐싱을 지원하여 SQL이 여러번 수행되는것을 최적화 한다.
- 데이터 접근 추상화와 벤더 독립성 : 관계형데이터 베이스를 어떤 벤더를 사용할지에 따라 맵핑 방법도 다르다. 밴더에는 mysql, oracle, h2 등등이 있다. 이 매핑 작업을 개발자가 직접 안해도 된다.
- 표준 : 표준을 알아두면 다른 구현기술로 쉽게 변경할 수 있다.

JPA 시작하기

H2 데이터 베이스 연동하기

h2 데이터베이스의 특징

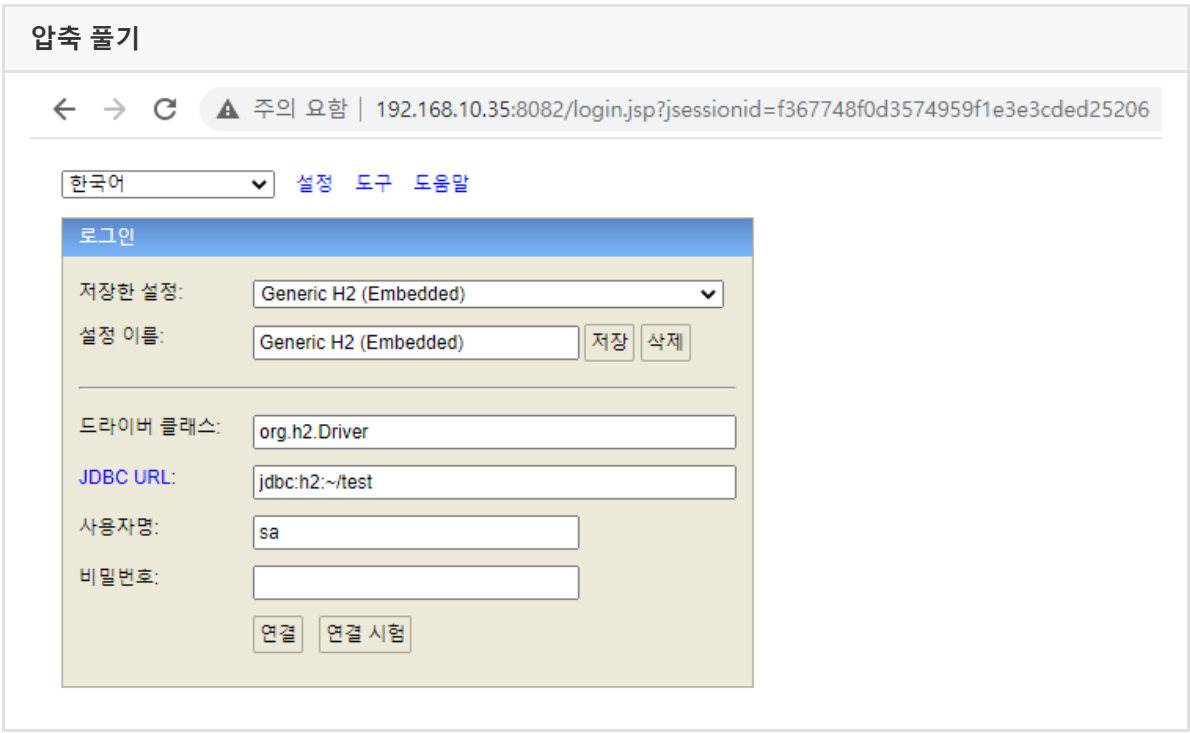
- 개발, 테스트 용도로 가볍고 편리
- 웹화면 제공

설치 URL

- 윈도우 설치 버전: https://h2database.com/h2-setup-2019-10-14.exe
- 윈도우, 맥, 리눅스 실행 버전: https://h2database.com/h2-2019-10-14.zip
- 공식 : https://www.h2database.com

1. H2 설치 및 DB 디렉토리 생성

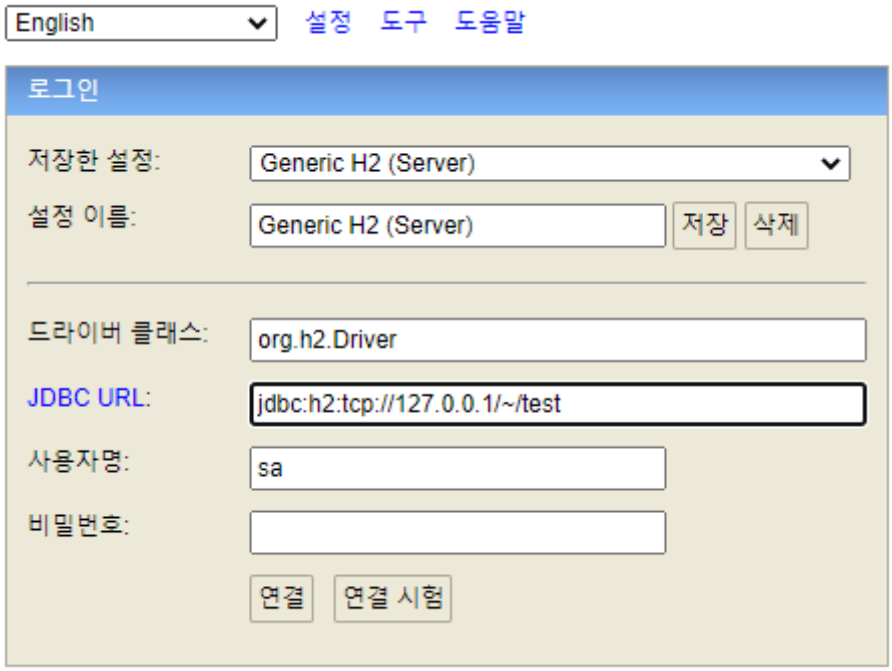
압축 풀기



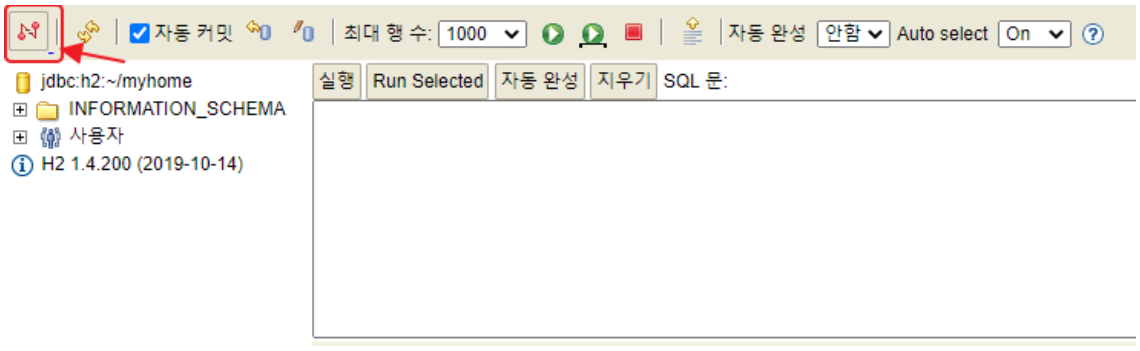
h2.bat 내용을 살펴보면 java 를 실행하는 것을 볼 수 있다. 따라서 java가 깔려있어야 한다.

1. bin/h2.bat 파일을 실행한다.
2. `JDBC URL` 부분에 `jdbc:h2:~/스키마명` (여기서는 test)를 입력하고 `연결`을 클릭한다.
3. `disconnect`를 한 뒤 창을 나간다.
4. bin/h2.bat 을 다시 실행한다.
5. `JDBC URL` 부분에 `jdbc:h2:tcp://127.0.0.1/~/스키마명` 을 입력하고 `연결`을 클릭한다.
6. 앞으로는 (5) 과 같은 JDBC URL 을 사용하여 접속한다.

JDBC URL 에 다음과 같이 작성하고 `연결` 클릭
jdbc:h2:tcp://127.0.0.1/~/myhome



세션을 끊는다.



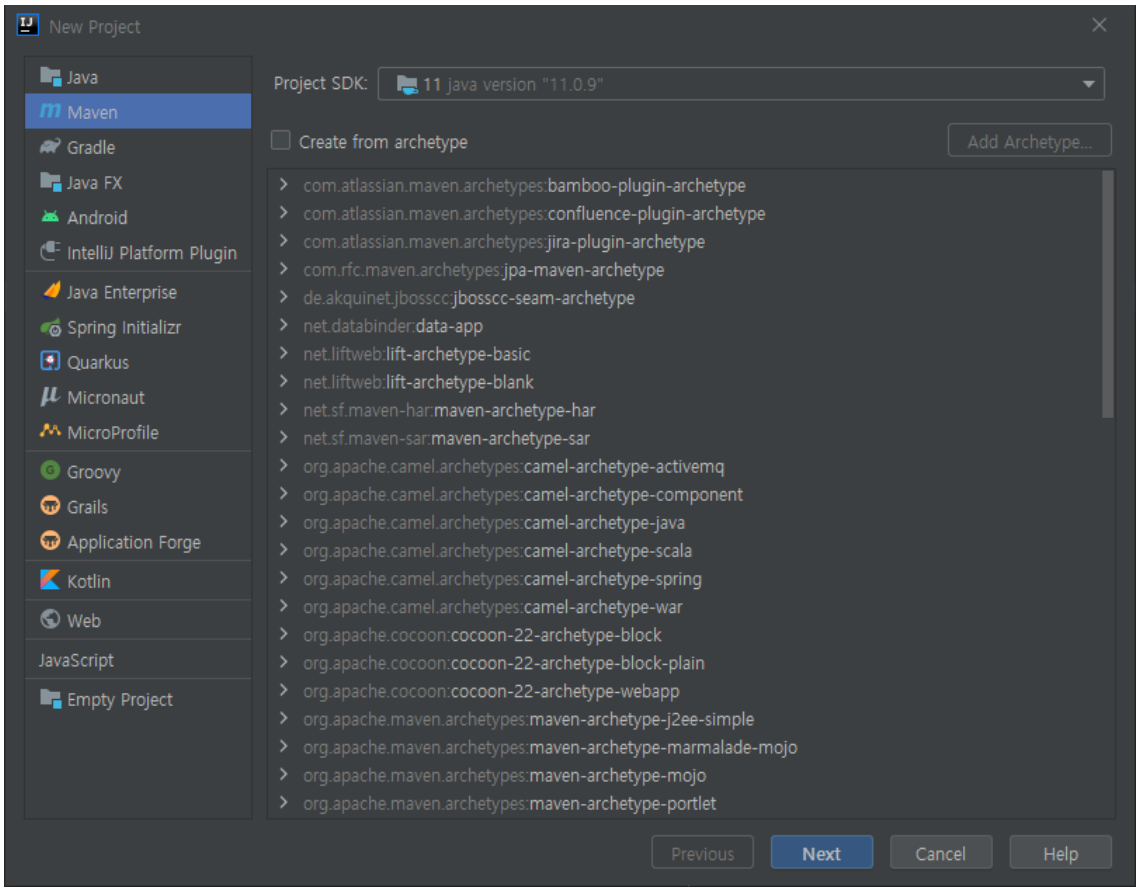
Maven 프로젝트 생성

JPA는 스프링 환경에서만 동작할 수 있는 라이브러리가 아니다.

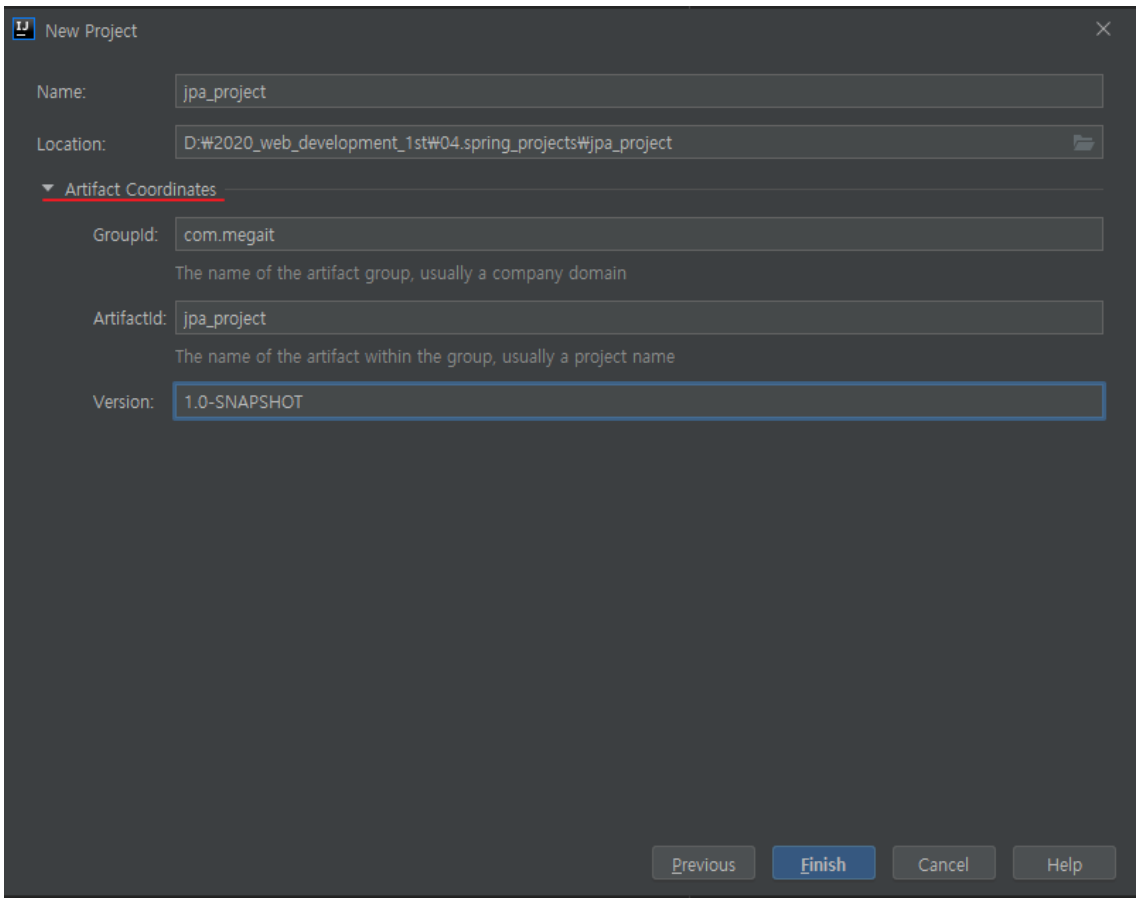
모든 환경에서 사용할 수 있으니 깡통 Maven 프로젝트를 만들어서 웹서비스 없이 JPA를 사용해보자.

Maven이 아니더라도 Gradle을 사용하여 빌드해도 된다.

`File` > `New` > `Project...`



project 및 artifact 이름 짓기

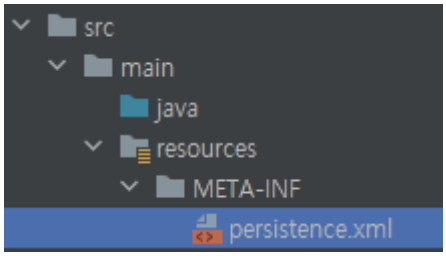


pom.xml

```
    <dependencies>

        <!-- JPA 하이버네이트 -->
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-entitymanager</artifactId>
            <version>5.3.10.Final</version>
        </dependency>

        <!-- H2 데이터베이스 -->
        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
            <version>1.4.200</version>
        </dependency>

        <!-- JAVA9 이상이라면 다음을 추가 (xml 바인딩할 때 JAX을 참조해야 하는데 JAVA9 부터는 JAX가 없어짐. 그래서 추가적인 모듈이 필요함. -->
        <!-- API, java.xml.bind module -->
        <dependency>
            <groupId>javax.xml.bind</groupId>
            <artifactId>jaxb-api</artifactId>
            <version>2.3.0</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.18.16</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
```

src/main/resoureces에 `META-INF` 디렉토리 생성 후

그 안에 `persistence.xml` File 추가



```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.2"
             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/persistence/persistence_2_2.xsd">
    <persistence-unit name="myunit">
        <properties>
            <!-- 필수 속성 -->
            <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"/>
            <property name="hibernate.dialect"
value="org.hibernate.dialect.H2Dialect"/>

            <!-- 옵션 -->
            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.format_sql" value="true"/>
            <property name="hibernate.use_sql_comments" value="true"/>
            <!--<property name="hibernate.hbm2ddl.auto" value="create" />-->

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

참고. Spring-boot 프로젝트에 H2 연동하기

spring-boot 를 maven 으로 생성했다면 위에 설정한 방식을 그대로 따라한다.

gradle 로 생성했다면 다음과 같이 설정한다.

build.gradle 에 다음 의존성 추가

```
implementation 'org.springframework.boot:spring-boot-starter-data-jpa:2.4.1'
implementation 'com.h2database:h2:1.4.200'  // 실제 설치된 H2 버전에 맞게 써야 한다.
```

main/resources/application.yml 에 다음과 같이 작성

(yml 은 **공백 2칸**을 들여쓰기 기준으로 둔다.)

```
spring:
  datasource:
    url: jdbc:h2:tcp://localhost/~/mymall
    username: sa
    password:
    driver-class-name: org.h2.Driver
  jpa:
    hibernate:
      ddl-auto: create
    properties:
```

```
      hibernate:
        format_sql: true
logging.level:
  org.hibernate.SQL: debug
```

```
spring:
  datasource:                                 # 커넥션풀 설정
    url: jdbc:h2:tcp://localhost/~/mymall     # 접근할 JDBC URL
    username: sa                              # 계정명
    password:                                 # 비밀번호
    driver-class-name: org.h2.Driver          # DriverClass 의 FQCN
  jpa:                                        # JPA 설정
    hibernate:                                # JPA의 hibernate 설정
      ddl-auto: create                        # DDL 관련 자동 처리 선택
    properties:                               # JPA의 Properties
      hibernate:                              # JAP.properties.hiberbate 설정
        # show_sql: true  # System.out 에 하이버네이트 실행 SQL을 남긴다.
        format_sql: true  # Log4j 로그를 찍을때 알아보기 편한 상태로 줄바꿈 해서 출력해준다.
logging.level:              # 어느 레벨에서 Logging 할 것인가?
  org.hibernate.SQL: debug  # hibernate.SQL 로그는 디버그 모드로 남길 것이다.
  # org.hibernate.type: trace
```

```
spring.jpa.hibernate.ddl-auto
```

1. create :

   하이버네이트의 SessionFactory가 올라갈 때 테이블을 지우고 새로 만듬.

2. create-drop :

   create와 동일하지만, SessionFactory가 내려가면 해당 테이블을 drop시킨다.

3. update :

   SessionFactory가 올라갈 때 Object를 검사하여 테이블을 alter 시킨다. 데이터는 유지됨.

4. validate :

   update처럼 Object를 검사하지만, 스키마는 아무것도 건드리지 않고, Object와 스키마의 정보가 다르다면 에러를 발생시킨다.

> 하이버네이트 로그 남기기 : https://kwonnam.pe.kr/wiki/java/hibernate/log