Oracle Database 연동 (HikariCP)

생성일	@2022년 8월 16일 오후 4:45
태그	

1. pom.xml 설정

-. dependency 추가 필요

```
<dependency>
      <groupId>com.zaxxer</groupId>
      <artifactId>HikariCP</artifactId>
      <version>5.0.1</version>

      <exclusions>
        <exclusion>
          <groupId>org.slf4j</groupId>
          <artifactId>slf4j-api</artifactId>
        </exclusion>
      </exclusions>

  </dependency>
```

2. root-context.xml 설정

root-context.xml은 스프링이 로딩되면서 읽어들이는 문서이다. 이미 만들어진 클래스를 Bean태그를 사용하여, 스프링의 빈(Bean)으로 등록할 수 있다.

앞서, HikariCP를 pom.xml에 등록하였기 때문에, 해당 클래스를 사용할 수 있다.

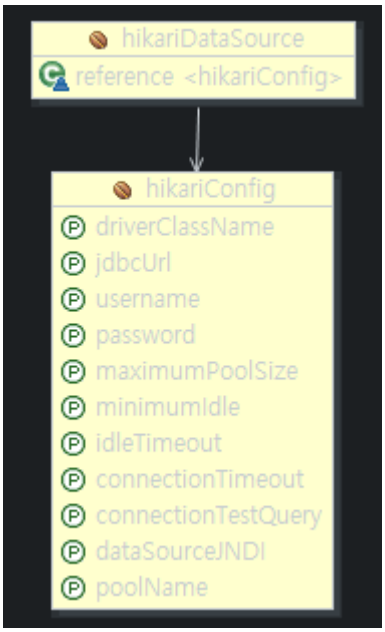
```
<bean id="hikariConfig" class="com.zaxxer.hikari.HikariConfig">
      <description>HikariCP Configuration</description>

      <property name="driverClassName" value="net.sf.log4jdbc.sql.jdbcapi.DriverSpy"/>
```

```
      <property name="jdbcUrl" value="****"/>
      <property name="username" value="****"/>
      <property name="password" value="****"/>

      <property name="maximumPoolSize" value="10"/>
      <property name="minimumIdle" value="2"/>
      <property name="idleTimeout" value="10000"/>
      <property name="connectionTimeout" value="1000"/>
      <property name="connectionTestQuery" value="SELECT 1 FROM dual"/>
      <property name="dataSourceJNDI" value="jdbc/HikariCP"/>
      <property name="poolName" value="*** HikariDataSource ***"/>
  </bean>
```

HikariConfig 객체를 HikariDataSource객체에 주입하기 위해, 아래와 같이 Bean 태그가 추가로 필요하다.

-. constructor-arg 태그의 ref 속성을 통해, 앞서 설정한 Hikariconfig 객체를 주입시킨다.

```
<bean primary="true" id="hikariDataSource" class="com.zaxxer.hikari.HikariDataSource"
 destroy-method="close">
      <description>HikariCP DataSource</description>

      <constructor-arg ref="hikariConfig"/>
</bean>
```



-. 위와 같이, Beans Graph에서 hikariDataSource가 hikariConfig 객체가 주입된 것을 볼 수 있다.

3. JUnit Test코드 작성

1. JUnit Test를 위한 어노테이션 작업

-. ContextConfiguration은 해당 Test에서 사용하는 스프링 설정파일을 등록한다.

-. 스프링 설정파일이 있는 경로를 적어준다.

-. 해당 Test에서는 root-context.xml 설정파일에만 있는 설정만 적용하기 때문에, root-context.xml만 적용하였다.

```
@Log4j2
@NoArgsConstructor


@ExtendWith(SpringExtension.class)
@ContextConfiguration(locations = {
        "file:src/main/webapp/WEB-INF/spring/root-context.xml"
})

@TestInstance(Lifecycle.PER_CLASS)
@TestMethodOrder(OrderAnnotation.class)
public class Ex01_HikariDataSourcesTests {
```

2. JDBC의 Connection 객체를 얻기위한 DataSource 선언

-. DataSource 객체를 필드에 선언하였다.

-. DataSource 객체에 root-context.xml에 설정한 HikariDataSource를 주입하기 위해, @Setter 방식으로 주입하였다.

-. 이 때, 주입가능한 빈 객체가 2가지 이상일 수 있어 @Resource 어노테이션을 사용하여, 주입하려고 하는 타입이 HikaraDataSource타입이란고 명시적으로 적어 주었다.

```java
public class Ex01_HikariDataSourcesTests {



    // -. 주입가능한 빈 객체가 2가지 이상일 때, @Resource 어노테이션을 사용하여
    // 지정이 가능하다.
    // -. 지금 root-context.xml에는 HikariDataSource밖에 없지만,
    // 추후 다양한 DataSource를 사용할 수 있기에 이 방법을 사용하였다.
    @Setter(onMethod_={@Resource(type=HikariDataSource.class)})
    private DataSource dataSource;
```

3. 선처리 작업 (@BeforeAll)과 Test 메소드 (@Test) 실행

-. 앞서 DataSource에 HikariDataSource가 잘 주입되었는 지, 확인을 선처리 작업에서하며,

-. 테스트 메소드에서 DataSource를 통해, Connection 객체를 얻어보자.

```java
@BeforeAll
void beforeAll() {
    log.trace("beforeAll() invoked.");

    // 1. 필드에 의존성 객체가 잘 주입되었는 지 확인하는 코드
    Objects.requireNonNull(this.dataSource);
    log.info("\t+ this.dataSource : {}", dataSource);

} // beforeAll

@Test
@Order(1)
@DisplayName("1. javax.sql.DataSource.getConnection() method test.")
@Timeout(value=3, unit=TimeUnit.SECONDS)
void testGetConnection() throws SQLException {
    log.trace("testGetConnection() invoked.");

    Connection conn = this.dataSource.getConnection();
    Objects.requireNonNull(conn);
```

4. 후처리 작업(@AfterAll)에서 사용한 DatsSource객체를 Close 하기

-. 사용한 자원객체는 해제해주어야한다.

-. DataSource 인터페이스는 Close를 할 수 있는 메소드가 존재하지 않는다. 하지만, HikariDataSource는 존재하기 때문에, Up-Casting을 하여, 형변환을 해주어야 한다.

```java
@AfterAll
void afterAll() {
    log.trace("afterAll() invoked.");

    ((HikariDataSource)this.dataSource).close();
}
```

** 전체 코드

```java
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Objects;
import java.util.concurrent.TimeUnit;

import javax.annotation.Resource;
import javax.sql.DataSource;

import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.MethodOrderer.OrderAnnotation;
import org.junit.jupiter.api.Order;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestInstance;
import org.junit.jupiter.api.TestInstance.Lifecycle;
import org.junit.jupiter.api.TestMethodOrder;
import org.junit.jupiter.api.Timeout;
import org.junit.jupiter.api.extension.ExtendWith;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit.jupiter.SpringExtension;

import com.zaxxer.hikari.HikariDataSource;

import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.extern.log4j.Log4j2;

@Log4j2
@NoArgsConstructor


@ExtendWith(SpringExtension.class)
@ContextConfiguration(locations = {
    "file:src/main/webapp/WEB-INF/spring/root-context.xml"
})

@TestInstance(Lifecycle.PER_CLASS)
@TestMethodOrder(OrderAnnotation.class)
public class Ex01_HikariDataSourcesTests {
```

```
  // -. 주입가능한 빈 객체가 2가지 이상일 때, @Resource 어노테이션을 사용하여
  // 지정이 가능하다.
  // -. 지금 root-context.xml에는 HikariDataSource밖에 없지만,
  // 추후 다양한 DataSource를 사용할 수 있기에 이 방법을 사용하였다.
  @Setter(onMethod_={@Resource(type=HikariDataSource.class)})
  private DataSource dataSource;

  @BeforeAll
  void beforeAll() {
    log.trace("beforeAll() invoked.");

    // 1. 필드에 의존성 객체가 잘 주입되었는 지 확인하는 코드
    Objects.requireNonNull(this.dataSource);
    log.info("\t+ this.dataSource : {}", dataSource);

  } // beforeAll

  @Test
  @Order(1)
  @DisplayName("1. javax.sql.DataSource.getConnection() method test.")
  @Timeout(value=3, unit=TimeUnit.SECONDS)
  void testGetConnection() throws SQLException {
    log.trace("testGetConnection() invoked.");

    Connection conn = this.dataSource.getConnection();
    Objects.requireNonNull(conn);



  } // testGetConnection

  @AfterAll
  void afterAll() {
    log.trace("afterAll() invoked.");

    ((HikariDataSource)this.dataSource).close();
  }


} // end class
```