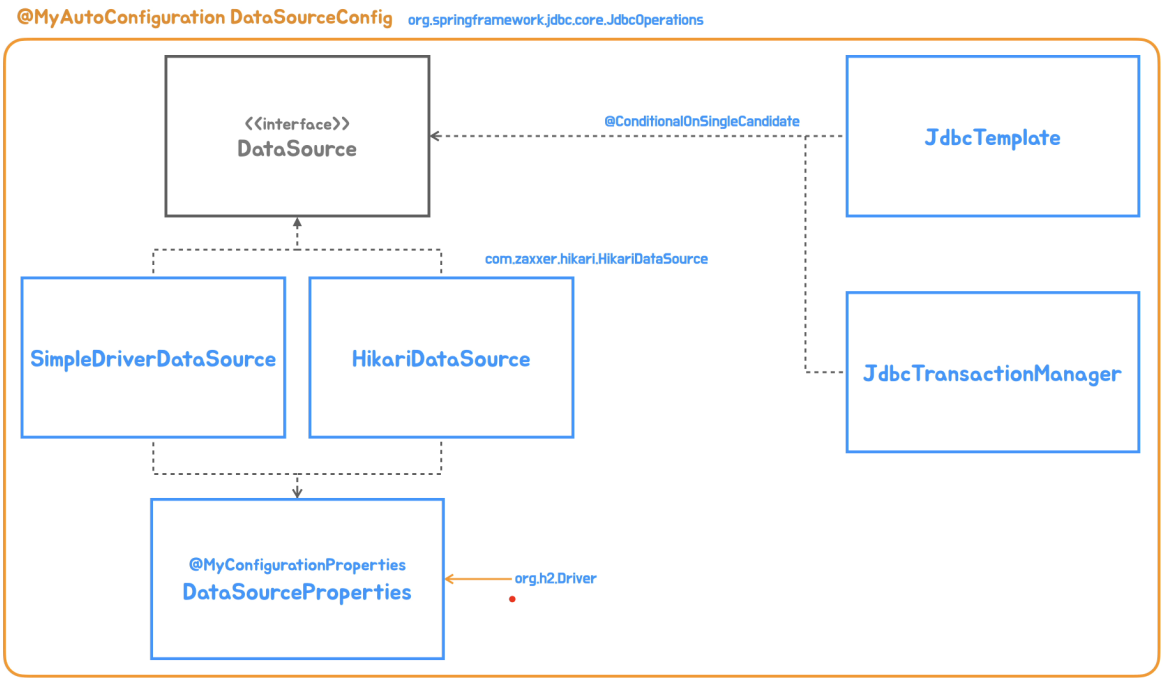
09 스프링 JDBC 자동 구성 개발

자동 구성 클래스와 빈 설계

새로운 기술의 자동 구성 클래스를 작성할 때는 자동 구성 클래스에 적용할 조건과 만들어지는 빈 오브젝트의 종류 등을 먼저 설계한다.



두 개의 DataSource 구현 클래스를 조건에 따라 등록되도록 한다. 이 두 개의 빈은 DataSourceProperties라는 프로퍼티 클래스를 이용한다.

스프링 JDBC

DataSource 자동 구성 클래스

DataSourceConfig은 JdbcOperations 클래스의 존재를 확인해서 등록되도록 한다.

DataSource 빈 메소드에서 프로퍼티로 사용할 프로퍼티를 정의한다.

@EnableTransactionManagement는 애노테이션을 활용한 트랜잭션 기능을 가능하게 해주는 구성용 애노테이션이다.

```
@MyAutoConfiguration
@ConditionalMyOnClass("org.springframework.jdbc.core.JdbcOperations")
@EnableMyConfigurationProperties(MyDataSourceProperties.class)
@EnableTransactionManagement
public class DataSourceConfig {
```

SimpleDriverDataSource는 간단한 테스트에서만 사용해야 한다. 운영 환경에서 사용하면 안 된다.

@ConditionalOnMissingBean을 이용해서 앞에서 DataSource가 등록되면 빈을 만들지 않도록 한다.

```
@Bean
@ConditionalOnMissingBean
DataSource dataSource(MyDataSourceProperties properties) throws ClassNotFoundException {
    SimpleDriverDataSource dataSource = new SimpleDriverDataSource();

    dataSource.setDriverClass((Class<? extends Driver>) Class.forName(properties.getDriverClassName()));
    dataSource.setUrl(properties.getUrl());
    dataSource.setUsername(properties.getUsername());
    dataSource.setPassword(properties.getPassword());

    return dataSource;
}
```

Hikari Data Source는 Hikari 클래스가 존재하는 경우에만 만들어지도록 조건을 걸어준다.

```
@Bean
@ConditionalMyOnClass("com.zaxxer.hikari.HikariDataSource")
@ConditionalOnMissingBean
DataSource hikariDataSource(MyDataSourceProperties properties) {
    HikariDataSource dataSource = new HikariDataSource();

    dataSource.setDriverClassName(properties.getDriverClassName());
    dataSource.setJdbcUrl(properties.getUrl());
    dataSource.setUsername(properties.getUsername());
    dataSource.setPassword(properties.getPassword());

    return dataSource;
}
```

JdbcTemplate과 트랜잭션 매니저 구성

@ConditionalOnSingleCandidate는 빈 구성정보에 해당 타입의 빈이 한 개만 등록되어있는 경우에 조건이 매칭된다.

JdbcTemplate은 DataSource를 주입 받아서 생성한다.

```
@Bean
@ConditionalOnSingleCandidate(DataSource.class)
@ConditionalOnMissingBean
JdbcTemplate jdbcTemplate(DataSource dataSource) {
    return new JdbcTemplate(dataSource);
}
```

스프링의 트랜잭션 추상화 기능을 활용하기 위해서 트랜잭션 매니저 빈도 만들어준다.

```
@Bean
@ConditionalOnSingleCandidate(DataSource.class)
@ConditionalOnMissingBean
JdbcTransactionManager jdbcTransactionManager(DataSource dataSource) {
    return new JdbcTransactionManager(dataSource);
}
```

애노테이션을 이용하는 트랜잭션 기능을 이용하기 위해 @EnableTransactionManagement을 클래스 레벨에 넣는 것도 잊지 말아야 한다.

Hello Repository

> 💡 인터페이스에 default 메소드, static 메소드를 넣어서 활용하는 방법은 자바의 Comparator<T> 인터페이스를 참고하면 도움이 된다.

JdbcTemplate의 queryForObject를 이용해서 하나의 조회 결과를 가져올 수 있다. 오브젝트에 컬럼 값을 매핑하는 경우 RowMapper 인터페이스를 구현해서 ResultSet을 데이터 오브젝트로 전환하는 코드를 넣어 콜백으로 전달해 사용한다.

```
<T> T queryForObject(String sql, RowMapper<T> rowMapper) throws DataAccessException;
```

```
@FunctionalInterface
public interface RowMapper<T> {

  @Nullable
  T mapRow(ResultSet rs, int rowNum) throws SQLException;
}
```

데이터를 변경할 때는 JdbcTemplate의 update()를 이용한다.

리포지토리를 사용하는 HelloService

테스트 코드에 사용되는 애노테이션도 합성 애노테이션으로 만들어 사용하면 편리하다.

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
@ExtendWith(SpringExtension.class)
@ContextConfiguration(classes = HellobootApplication.class)
@TestPropertySource("classpath:/application.properties")
@Transactional
```

```
public @interface HellobootTest {
}
```