[**Spring Framework/Spring Boot**](/category/Spring%20Framework/Spring%20Boot)

**[Spring Boot] 다중 데이터베이스 사용하기**

장쫄깃 2023. 1. 16. 22:02



**들어가며**

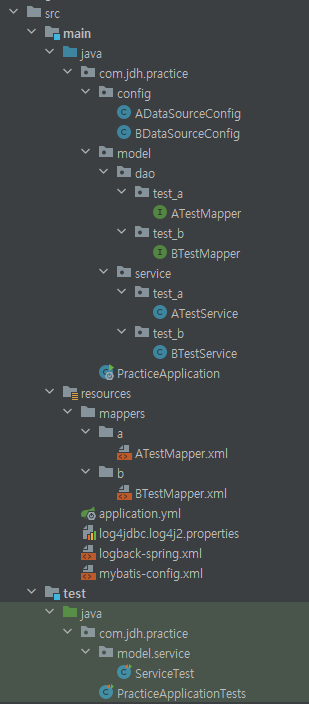
하나의 어플리케이션에서 다중 DB에 접근할 수 있는 설정에 대해 알아보려고 한다.

대략적인 순서로는

1. application.yml 설정파일에 db 연결정보 작성
2. Config 파일 생성
3. service, dao, mapper 파일 생성
4. 테스트

위와 같으며, 패키지 구조도 함께 살펴보겠다.

**0. 프로젝트 구조**



프로젝트 구조는 다음과 같다. 중요하게 봐야할 부분은 Mapper 인터페이스 파일과 쿼리를 작성하는 xml 파일을 DB별로 나눠서 사용할 수 있게끔 나누었다.

**1. application.yml 설정파일에 db 연결정보 작성**



연결하려는 DB의 접속정보들을 작성한다.

**2. Config 파일 생성**

Mybatis를 사용하기 위해 HikariCP의 DataSource를 설정하였다.

HikariCP에 대한 설명은 해당 글을 참고하면 된다.

링크 : <https://jangjjolkit.tistory.com/40>

[[Spring - DB] 1. HikariCP를 이용한 MySQL DB 연결하기](https://jangjjolkit.tistory.com/40" \t "_blank)

[들어가며 기존에 SpringBoot에선 tomcat-jdbc를 기본 DataSource로 제공했다. 하지만 2.0부터 HikariCP가 기본으로 변경되었다.(참고) HikariCP는 이전 버전에서도 많은 사람들이 설정을 변경해서 사용했을 정](https://jangjjolkit.tistory.com/40" \t "_blank)

[jangjjolkit.tistory.com](https://jangjjolkit.tistory.com/40" \t "_blank)

다중 DB를 사용하려면 DB마다 해당 작업을 진행해야 한다. 그런데 Spring의 Bean은 싱글톤을 사용하여 관리하기 때문에 여러 DataSource 관련 Bean을 아무런 설정 없이 사용하면 충돌이 난다.

그렇기 때문에 특정 DB의 config 파일에 **@Primary** 어노테이션을 붙여 특정 빈을 우선적으로 주입하게 하고, 그 외에 DB config 파일에는 **@Qualifier** 어노테이션을 붙여 식별자를 지칭해줘야 한다.

의존성 주입 시 @Primary, @Qualifier에 대한 설명은 해당 글을 참고하면 된다.

링크 : <https://jangjjolkit.tistory.com/37>

[[Spring Boot] 의존성 주입 시 Bean이 여러개라면? (@Primary, @Qualifier)](https://jangjjolkit.tistory.com/37" \t "_blank)

[들어가며 @Autowired는 Component Scan + @Component로 스프링 빈에 등록된 객체를 찾아서 필요한 의존관계를 설정한다. (우선적으로, 타입(Type)으로 해당 빈(Bean)을 찾는다.) 만약 @Autowired를 통한 자동 의존](https://jangjjolkit.tistory.com/37" \t "_blank)

[jangjjolkit.tistory.com](https://jangjjolkit.tistory.com/37" \t "_blank)

**@Primary를 사용한 설정파일**

@Configuration

@MapperScan**(**value **=** "com.jdh.practice.model.dao.test\_a"**)**

public class ADataSourceConfig **{**

private final String A\_DATA\_SOURCE **=** "ADataSource"**;**

// A database DataSource

@Primary

@Bean**(**A\_DATA\_SOURCE**)**

@ConfigurationProperties**(**prefix **=** "spring.test-a.datasource.hikari"**)**

public DataSource ADataSource**()** **{**

**return** DataSourceBuilder**.**create**()**

**.**type**(**HikariDataSource**.**class**)**

**.**build**();**

**}**

// SqlSessionTemplate 에서 사용할 SqlSession 을 생성하는 Factory

@Primary

@Bean

public SqlSessionFactory ASqlSessionFactory**(**DataSource dataSource**)** **throws** Exception **{**

/\*

\* MyBatis 는 JdbcTemplate 대신 Connection 객체를 통한 질의를 위해서 SqlSession 을 사용한다.

\* 내부적으로 SqlSessionTemplate 가 SqlSession 을 구현한다.

\* Thread-Safe 하고 여러 개의 Mapper 에서 공유할 수 있다.

\*/

SqlSessionFactoryBean bean **=** **new** SqlSessionFactoryBean**();**

bean**.**setDataSource**(**dataSource**);**

// MyBatis Mapper Source

// MyBatis 의 SqlSession 에서 불러올 쿼리 정보

Resource**[]** res **=** **new** PathMatchingResourcePatternResolver**().**getResources**(**"classpath:mappers/a/\*Mapper.xml"**);**

bean**.**setMapperLocations**(**res**);**

// MyBatis Config Setting

// MyBatis 설정 파일

Resource myBatisConfig **=** **new** PathMatchingResourcePatternResolver**().**getResource**(**"classpath:mybatis-config.xml"**);**

bean**.**setConfigLocation**(**myBatisConfig**);**

**return** bean**.**getObject**();**

**}**

// DataSource 에서 Transaction 관리를 위한 Manager 클래스 등록

@Primary

@Bean

public DataSourceTransactionManager ATransactionManager**(**DataSource dataSource**)** **{**

**return** **new** DataSourceTransactionManager**(**dataSource**);**

**}**

**}**

**@Qualifier를 사용한 설정파일**

@Configuration

@MapperScan**(**value **=** "com.jdh.practice.model.dao.test\_b"**,** sqlSessionFactoryRef **=** "BSqlSessionFactory"**)**

public class BDataSourceConfig **{**

private final String B\_DATA\_SOURCE **=** "BDataSource"**;**

// A database DataSource

@Bean**(**B\_DATA\_SOURCE**)**

@ConfigurationProperties**(**prefix **=** "spring.test-b.datasource.hikari"**)**

public DataSource BDataSource**()** **{**

**return** DataSourceBuilder**.**create**()**

**.**type**(**HikariDataSource**.**class**)**

**.**build**();**

**}**

// SqlSessionTemplate 에서 사용할 SqlSession 을 생성하는 Factory

@Bean

public SqlSessionFactory BSqlSessionFactory**(**@Qualifier**(**B\_DATA\_SOURCE**)** DataSource dataSource**)** **throws** Exception **{**

/\*

\* MyBatis 는 JdbcTemplate 대신 Connection 객체를 통한 질의를 위해서 SqlSession 을 사용한다.

\* 내부적으로 SqlSessionTemplate 가 SqlSession 을 구현한다.

\* Thread-Safe 하고 여러 개의 Mapper 에서 공유할 수 있다.

\*/

SqlSessionFactoryBean bean **=** **new** SqlSessionFactoryBean**();**

bean**.**setDataSource**(**dataSource**);**

// MyBatis Mapper Source

// MyBatis 의 SqlSession 에서 불러올 쿼리 정보

Resource**[]** res **=** **new** PathMatchingResourcePatternResolver**().**getResources**(**"classpath:mappers/b/\*Mapper.xml"**);**

bean**.**setMapperLocations**(**res**);**

// MyBatis Config Setting

// MyBatis 설정 파일

Resource myBatisConfig **=** **new** PathMatchingResourcePatternResolver**().**getResource**(**"classpath:mybatis-config.xml"**);**

bean**.**setConfigLocation**(**myBatisConfig**);**

**return** bean**.**getObject**();**

**}**

// DataSource 에서 Transaction 관리를 위한 Manager 클래스 등록

@Bean

public DataSourceTransactionManager BTransactionManager**(**@Qualifier**(**B\_DATA\_SOURCE**)** DataSource dataSource**)** **{**

**return** **new** DataSourceTransactionManager**(**dataSource**);**

**}**

**}**

1. **@Configuration** 어노테이션을 사용하여 Spring Framework가 기동 시 설정파일 등록
2. **@MapperScan** 어노테이션을 사용하여 Mapper 인터페이스 경로를 지정하고, **sqlSessionFactoryRef**로 어떤 sqlSessionFactory를 사용할지 지정
3. 특정 DB config 파일에 **@Primary** 어노테이션을 붙여 우선적으로 등록
4. DataSource 메소드 작성  
   * application.yml의 어떤 DataSoruce 설정 정보를 사용하는지 설정 (**@ConfigurationProperties**)
5. SqlSessionFactory 메소드 작성
   * @Primary 어노테이션을 사용하지 않는 config 파일에서 매개변수로 받는 DataSource에 **@Qualifier**를 사용하여 임의의 DataSource를 지정하여 충돌 방지
6. TransactionManager 메소드 작성 (선택)

위와 같이 설정한 후에 Service 단에서 각각의 Mapper를 사용해서 DB에 접근할 수 있다.

**3. service, dao, mapper 파일 생성**

ATestMapper.xml

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

<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace=**"com.jdh.practice.model.dao.test\_a.ATestMapper"**>

<select id=**"selectTest"** resultType=**"String"**>

**select concat("A\_", test\_nm) from test\_table;**

</select>

</mapper>

ATestMapper.java

@Mapper

public interface ATestMapper {

String selectTest() throws Exception;

}

ATestService.java

@RequiredArgsConstructor

@Service

public class ATestService {

private final ATestMapper aTestMapper;

public String getTest() throws Exception {

return aTestMapper.selectTest();

}

}

BTestMapper.xml

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

<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.jdh.practice.model.dao.test\_b.BTestMapper">

<select id="selectTest" resultType="String">

select concat("B\_", test\_nm) from test\_table;

</select>

</mapper>

BTestMapper.java

@Mapper

public interface BTestMapper {

String selectTest() throws Exception;

}

BTestService.java

@RequiredArgsConstructor

@Service

public class BTestService {

private final BTestMapper bTestMapper;

public String getTest() throws Exception {

return bTestMapper.selectTest();

}

}

A DB에서 조회할 때는 조회 데이터 앞에 **"A\_"** 를 붙여 조회하게 했고, B DB에서 조회할 때는 조회 데이터 앞에 **"B\_"** 를 붙여 조회하게 했다.

**4. 테스트**

**Test**

@SpringBootTest

public class ServiceTest {

Logger log = (Logger) LoggerFactory.getLogger(this.getClass());

@Autowired

ATestService aTestService;

@Autowired

BTestService bTestService;

@Test

public void test() throws Exception {

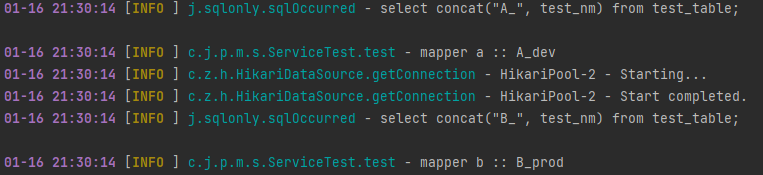
log.info("mapper a :: " + aTestService.getTest());

log.info("mapper b :: " + bTestService.getTest());

}

}

테스트 코드를 작성하여 각각의 DB에서 데이터를 정확하게 조회하는지 확인하였다.



테스트 결과 A, B 각각의 DB에서 데이터를 조회하는 것을 확인하였다.

**정리하며**

관련 소스 코드는 깃허브를 참고하면 된다.

링크 : <https://github.com/JangDaeHyeok/spring_boot_multiple_db_conn_practice>

[GitHub - JangDaeHyeok/spring\_boot\_multiple\_db\_conn\_practice: spring boot 다중 데이터베이스 사용 연습하기](https://github.com/JangDaeHyeok/spring_boot_multiple_db_conn_practice" \t "_blank)

[spring boot 다중 데이터베이스 사용 연습하기. Contribute to JangDaeHyeok/spring\_boot\_multiple\_db\_conn\_practice development by creating an account on GitHub.](https://github.com/JangDaeHyeok/spring_boot_multiple_db_conn_practice" \t "_blank)

[github.com](https://github.com/JangDaeHyeok/spring_boot_multiple_db_conn_practice" \t "_blank)

댓글6

* 아무개

안녕하세요 좋은글 감사합니다. 혹시 이전에올려주신 글 ( https://jangjjolkit.tistory.com/41 )   
이랑 비슷한데 위의 글 같은경우에는 DB가 3개이상일시 사용하는게 좋은가요? @Transaction(readOnly = true)를 사용하는경우에는 스프링에서 읽기전용모드로 설정되어서 성능최적화가 가능하다는데 위글처럼 동적으로 하는경우에는 안되지만 3개이상의 DB를 마음대로 지정할수있는 장점이 있는건가요?

2023. 8. 23. 10:27답글

* [장쫄깃](https://jangjjolkit.tistory.com)

프로필열기

안녕하세요!!  
  
말씀해주신 글은 본 게시물이랑은 성격이 약간 다릅니다.  
  
본 게시물은 단순히 2개 이상의 DB를 사용하는 방법에 대해 설명하고 있구요!  
  
Transaction ReadOnly같은 경우는 보통 성능을 향상시키기 위해 사용합니다. 실제로 DB가 데이터를 직접 쓰거나 수정할 때 select를 동시에 요청하게 되면 병목현상이 발생합니다. 읽기전용으로 성능최적화가 필요한 select 요청은 복제본 DB로 요청을 분산시키는 목적입니다.  
  
2개 이상의 서로 다른 DB를 하나의 프로젝트에서 사용하고 싶은 경우는 본 게시물을 참고하시면 되고, 하나의 DB를 기준으로 Master/Slave DB로 요청을 분산시키고 싶은 경우에는 https://jangjjolkit.tistory.com/41 게시글을 참고하시면 됩니다!  
  
답변이 도움이 되었으면 좋겠습니다ㅎㅎ

* 테스트

A Mapper에서 B데이터베이스의 테이블을 조인할수 있을까요?

* [장쫄깃](https://jangjjolkit.tistory.com)

프로필열기

안녕하세요. 답변이 늦어 죄송합니다.  
  
DB에서의 조인은 힘들 것 같습니다...  
  
각각의 데이터를 조회하고 조인하는 별도 서버 기능을 구현해야 하지 않을까 싶습니다!

* config

안녕하세요  
springboot 에서는 mybatis-spring-starter 의존성을 추가하면 따로 config 설정을 안해줘도 되는것으로 알고있는데  
이렇게 config 파일들은 만드는 이유가 뭔가요 ㅠㅠ  
이제 막 공부하기 시작해서 도움부탁드립니다.

2023. 11. 6. 11:38답글

* [장쫄깃](https://jangjjolkit.tistory.com)

프로필열기

안녕하세요. 답변이 늦어 죄송합니다.  
  
여러 DB에 다중으로 접근하기 위해 Config를 별도로 분리하여 설정하고 있습니다.  
  
위 예시에서는 2개의 DB에 접근했지만, 위 방법으로 2개 이상의 DB 다중 연결이 가능합니다.  
  
만약 1개의 DB만 사용하는 경우는 별도로 설정을 안해주셔도 될 듯 합니다!

2024. 7. 26. 09:43답글

출처: <https://jangjjolkit.tistory.com/55> [장쫄깃 기술블로그:티스토리]