**1스프링과 mysqldb 연동. 디비와 연결하여 Jdbc와 디비커넥션풀 사용하기.**

**알아야 할 키워드 : jdbc, DriverManager, Connection, 디비커넥션 풀, 히카리, .properties파일, @PropertySource, @Value, RootConfig, HikariConfig, HikariDataSource, DataSource**

Build.gradle  
//Mysqldb와 연동하여 jdbc하기 위한 dependency.  
implementation 'com.mysql:mysql-connector-j:8.1.0'  
//sql연동시 접속하는게 제일 큰 오버헤드 매번 필요할 때 접속하는건 낭비 커넥션 풀을 이용하자  
//디비커넥션풀 운용을 위한 hikari connection pool 설치  
implementation 'com.zaxxer:HikariCP:2.7.4'

jdbc기억나는가? 먼저 db와 연결부터!

mysqldb와의 연결을 위한 설정을 코드와 분리하자. properties파일 형식을 이용하자.  
src>main>resource>application.properties  
jdbc.driver=com.mysql.cj.jdbc.Driver ##연동할 DB 드라이버 지정  
jdbc.url=jdbc:mysql://127.0.0.1:3306/scoula\_db ##연동할 접속 아이피와 포트 그리고 db를 지정  
jdbc.username=scoula //접속할사용자  
jdbc.password=1234 //접속 비밀번호  
연결을 위한 설정을 코드로 가져오자. .properties파일에서 값 읽어오기  
.properties파일을 특정 클래스에서 사용하려면 @PropertySource( { 원하는.properties파일경로 } ) 클래스레벨 어노테이션을 해당 클래스에 적용해라.  
해당 클래스에서 가져온 .properties에서 값을 추출하고 싶다면 필드 멤버 앞에 @Value(“${ 키값 }”)를 사용하여 필드 멤버에 값을 넣을 수 있다.

DB 관련 연결 설정은 rootconfig에서 진행한다. 범용 데이터를 관리하는 rootconfig.  
@Configuraion  
@PropertySource({ “classpath:/application.properties }) //클래스 서치 경로라고 명시.  
public class RootConfig{  
 @Value(“${ jdbc.driver }”) String driver;   
 @Value(“${ jdbc.url }”) String url;   
 @Value(“${ jdbc.username }”) String username;   
 @Value(“${ jdbc.password }”) String password;  
   
 @Bean //dataSource 컨텍스트에 등록  
 public DataSource dataSource( ){  
 HikariConfig config = new HikariConfig( ); //히카리 커넥트풀에 등록 및 사용하기 위한 설정 객체.  
   
 config.setDriverClassName(drive);   
 config.setJdbcUrl(url);   
 config.setUsername(username);   
 config.setPassword(password);  
   
 HikariDataSource dataSource = new HikariDataSource(config);  
 return dataSource; // 커넥트풀에서 꺼낸 db연결 객체라고 생각하면 됨. 컨텍스트에 등록된. DI 후보  
 } }

얻은 dataSource객체로부터 Jdbc의 시작점인 Connection을 얻을 수 있다!

**정리 :**   
디비와 연동 == application.properties설정에 맞는 디비와 연결  
원래 JDBC는 driver manager로부터 바로 connection객체를 연결해 사용하겠지만 디비연결 오버헤드를 줄이기위한 디비커넥션풀을 활용. HikariConnectionPool 라이브러리 사용.  
디비에 대한 작업 시작시 connection객체를 풀에서 반환해주는 DataSource객체를 먼저 얻어야 함.  
원할한 DataSource활용을 위해서 rootconfig에 해당 객체를 컨텍스트에 등록하고 반화하는 메소드 정의. 메소드 안에서 DataSource생성시 hikari 객체를 통해서 properties설정을 이용하여 생성.  
DataSource를 통해서 디비커넥션풀에서 connection객체를 생성 및 반환하여 jdbc 시작!.  
jdbc는 사용하기 번거롭고 복잡. sql문도 코드에 종속되어있음. 여러모로 불편 🡺 sql 매퍼인 MYBATIS활용해보자.

**2mybatis활용 – jdbc는 복잡 불편.**

**알아야 할 키워드 : sqlmapper, mybatis, mybatis-spring, spring-tx, spring-jdbc, SqlSession, DataSource, DBCP, mapper, mapper.xml, mybatisx, mybatis-config.xml, @MapperScan, @Select(sql 어노테이션),  
<mapper namepspace=…>, <select id=…>(sql 태그)**

Mybatis(sql mapper)를 사용하는 이유 : jdbc코드 캡슐화하여 코드 간소화, sql쿼리와 코드 분리, 동적sql 지원(xml 태그로), 객체와 데이터베이스 매핑.

//mybatis설치와 mybatis와 spring 연동을 위한 mybatis-spring설치. Build.gradle  
implementation 'org.mybatis:mybatis:3.4.6'  
implementation 'org.mybatis:mybatis-spring:1.3.2'  
//트랜잭션 관리와 jdbc(데이터베이스 처리)를 스프링이 하겠다는 라이브러리 설치.  
implementation "org.springframework:spring-tx:${springVersion}"  
implementation "org.springframework:spring-jdbc:${springVersion}"

SQL Mapper 원리 :  
받은 커넥션과 mybatis 관련 정보를 SqlSession이라는 객체가 래핑하여 관리. 물론 받을 때 디비커넥션풀을 통해서 받는 datasource객체를 이용.  
SqlSession과 connection은 1대1 대응함. 하지만 멀티 쓰레드 안전성이 없음.  
SqlSession의 생성과 싱글톤 관리 및 멀티 쓰레드 안전성 확보를 위한 SqlSessionFactory를 사용하여, SqlSession객체를 받아 사용. SqlSessionFactory의 생성시 mybatis-config.xml설정을 적용하며 생성.

SQL Mapper 사용시 구성 :  
1)src>main>resources>mybatis-conifg.xml //SQLMAPPER설정파일,  
2)rootconfig //마이바티스 내부 객체인 sqlsession을 생성 및 싱글톤 관리하는 sqlsessionfactory를 컨텍스트에 빈등록하고 반환하는 메소드 & jdbc의 시작점인 connection객체를 관리하는 datasource의 트랜잭션을 관리하는 DatasourceManager를 컨텍스트에 등록하고 반환하는 메소드 정의. 그리고 등록된 매퍼를 스캔.  
3)src>main>java>org.socula>mapper>매퍼인터페이스 : sql과 그에 대한 처리를 지정. 해당 인터페이스에 sql문을 지정할 수 있거나, 혹은 mapper.xml로 sql문을 분리한 파일과 연동하여 사용. 디비와 연동할 메소드 선언.  
4)src>main>resources>org.scoula.mapper>매퍼.xml : 동일한 경로(resources가 아닌 java인)를 가지는 매퍼인터페이스에 사용할 sql문과 관련 설정하는 파일. 타겟 매퍼인터페이스 경로 등록, sql처리할 매퍼인터페이스에 선언된 메소드 지정. Sql문 분리. Sql문 동적 생성 등을 수행.  
5)매퍼인터페이스로부터 쉽게 매퍼.xml을 작성할 수 있도록 돕는 mybatisx 플러그인. : 필수는 아니지만 매퍼인터페이스에 맞는 매퍼.xml 작성이 엄청 수월해진다.

mybatis적용된 코드에 대하여 : 결론적으로 sqlsession객체를 직접 다루는 일이 매우 적다. 매퍼(매퍼 인터페이스)(메소드 정의와 어노테이션 사용)와 매퍼.xml(sql문 관리 및 sql문<->매퍼인터페이스 메소드 연결)만을 사용하여 쉽게 JDBC가 하는 일을 한다.

1)src>main>resources>mybatis-conifg.xml //SQLMAPPER설정파일  
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  
"http://mybatis.org/dtd/mybatis-3-config.dtd">  
<configuration> </configuration>

2)rootconfig //마이바티스 내부 객체인 sqlsession을 생성 및 싱글톤 관리하는 sqlsessionfactory를 컨텍스트에 등록하고 반환메소드 & jdbc의 시작점인 connection객체를 관리하는 datasource의 트랜잭션을 관리하는 Datasource Manager를 컨텍스트에 등록하고 반환하는 메소드 정의. 그리고 등록된 매퍼를 스캔.  
@Configuration @PropertySource({ “classpath:/application.properties })  
@MapperScan(basePackages={ “org.scoula.mapper” }) //매퍼 스캔할 매퍼 지정. 해당 인터페이스 컨텍스트에 빈.  
public class Rootconfig{  
@Autowired ApplicationContext ApplicationContext; //해당 객체는 스프링에서 자동으로 컨텍스트에 빈등록됐음.  
  
@Bean public SqlSessionFactory sqlSessionFactory ( ) throws Exception{//sqlSessionFactory 생성  
 SqlSessionFactoryBean sqlSessionFactory = new SqlSessionFactoryBean( );  
 sqlSessionFactory.setConfigLocation(applicationContext.getResource(“classpath:/mybatis-config.xml”);  
 //sqlsessionfactory이에 mybatis-config설정 설정.  
 sqlSessionFactory.setDataSource(dataSource( ));//sqlsessionfactory에서 사용할 datasource 지정  
   
 return (SqlSessionFactory)sqlSessionFactory.getObject( );  
}  
  
@Bean public DataSourceTransactionManager transactionManager( ){//datasource 트랜잭션 매니저 생성  
 DataSourceTransactionManager manager=new DataSourceTransactionManager(dataSource( ));  
 return manager;  
}  
…  
)

3)src>main>java>org.socula>mapper>매퍼인터페이스  
public interface TimeMapper{  
 //매퍼.xml 없이 인터페이스에서 어노테이션을 사용한 방법.  
 @Select(“select sysdate( )”)//사용할 sql  
 public String getTime( );//mybatis SQL어노테이션과 연동할 메서드 선언  
 //마이바티스가 적용된 어노테이션과 인터페이스(메소드의 시그니처)를 봐서 내부적으로 sqlsessionfactory에서  
 //얻은 sqlsession의 알맞은 메서드(selectOne, selectList, updateOne … )(sql어노테이션 값이 해당 메서드에  
 //인자로 넘어감)를 구현해내고 실행 가능하게함.  
  
 //매퍼.xml로 분리했을 경우 사용법  
 public string getTime2( ); //연동할 메서드만 선언하면 돼. SQL관련은 매퍼.xml에서.  
   
 public List<String> getTimes( );  
 public ROWVO getRow( );  
 public List<ROWVO> getRows( );  
 public ROWVO getRow(Long no);//인자가 필요한 경우1  
 public void createRow(ROWVO row); //인자가 필요한 경우1  
}

4)src>main>resources>org.scoula.mapper>매퍼와같은이름.xml : 동일한 경로(resources가 아닌 java인)를 가지는 매퍼인터페이스에 사용할 sql문과 관련 설정하는 파일.  
//경로 입력시 ‘.’이 아닌 ‘/’ 입력할 것.  
<?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=”org.scoula.mapper.TimeMapper”>//매퍼 인터페이스 경로 알려줘서 타겟 위치 지정.  
 //타겟 매퍼 인터페이스에 적용할 sql관련 설정 하는 곳.  
  
 //@Select어노테이션과 대응하는 select태그. Insert, update태그 등 다양한 동작 태그도 있다.  
 <select id=”getTime2” resultType=”string”>//sql태그의 id이름이 연동할 타겟 매퍼 인터페이스의 메서드  
 //반환 타입과 쿼리 내용을 봐서 알맞은 sqlsession메서드 호출되게끔.  
 select sysdate( )//sql함수도 활용.  
 </select>  
  
 <select id=”getTimes” resultType=”string”>//반환형이 list인 매퍼 인터페이스의 메서드와 연동할 sql설정.  
 select time from tbl\_world;  
 </select>  
 //mybatis-config.xml에서 resultType에 쓰이는 긴 이름을 타입이름만 쓰게하는 typeAliases 설정을 할 수 있다.  
 <select id=”getRow” resultType=”org.scoula.world.domain.ROWVO”>  
 //반환형이 VO인 매퍼 인터페이스의 메서드와 연동할 sql설정.  
 select \* from tbl\_world where id\_num=1;  
 </select>  
  
 <select id=”getRows” resultType=”org.scoula.world.domain.ROWVO”>  
 //반환형이 VO인 매퍼 인터페이스의 메서드와 연동할 sql설정.  
 select \* from tbl\_world;  
 </select>  
  
 <select id=”getRow” resultType=”org.scoula.world.domain.ROWVO”>  
 select \* from tbl\_world where id=#{no}//인자로 단일 값이 넘어왔으니 변수처럼 사용  
 </select>  
  
 <insert id=”create”> //void 반환 값이니까 resultType지정 안함.  
 insert into tbl\_board(title,content,write)  
 values (#{title}, #{content}, #{write}) //인자로 vo가 들어오면 vo의 각 필드 값을 변수처럼 사용가능  
 //내부에서 getter를 불러서 값을 꺼냄  
 //데이터를 넣고 해당 데이터의 fk(auto\_increment된 칼럼)의 값을 알아야할 수 있다!  
 //인자로 들어온게 맵타인 경우는 키이름으로 인지되어 get(키이름)이 동작하여 값을 꺼낸다  
   
 //🡺디비 연산하며 인자로 들어온 참조에 값을 넣어주는 selectkey태그 사용!  
 <selectKey resultType=”Long” keyProperty=”no” keyColumn=”no” order=”AFTER”>  
 //keyProperty==받을 vo의 필드명(setter동작함). keyColumn==원하는 디비상 칼럼값. Order==언제수행하냐  
 select last\_insert\_id( )//mysql 함수를 사용하여 마지막 insert된 id값을 출력.  
 </selectKey>  
 </insert>  
</mapper>  
  
5)매퍼인터페이스로부터 쉽게 매퍼.xml을 작성할 수 있도록 돕는 mybatisx 플러그인. : 3번, 4번 과정을 작업하기 아주 쉽게 해준다. 인텔리제이 플러그인에 검색해서 설치하는 방법 밖에 없다.

사용 : 매퍼인터페이스.메소드를 호출하여 사용하면 된다. Mybatis가 구현한 디비 동작이 실행할 것이다.

2-1)JDBC 처리 관련 로그 세부 사항 출력. – mybatis내부에서 preparedstatement로 sql처리시 ?에 실제 치환값이 적용된 sql를 알아야한다. 🡺 log4jdbc-log4j2 라이브러리를 이용하면 된다.

라이브러리 설치  
Build.gradle  
implementation 'org.bgee.log4jdbc-log4j2:log4jdbc-log4j2-jdbc4:1.16'  
implementation 'org.apache.logging.log4j:log4j-api:2.0.1'  
implementation 'org.apache.logging.log4j:log4j-core:2.0.1'

Log4jdbc-log4j2설정 .properties파일로  
src>main>resources>log4jdbc.log4j2.properties  
log4jdbc.spylogdelegator.name=net.sf.log4jdbc.log.slf4j.Slf4jSpyLogDelegator  
log4jdbc.auto.load.popular.drivers=false  
log4jdbc.drivers=com.mysql.cj.jdbc.Drive  
로그 메시지를 출력하기전에 중간에 가로채서 조작해서 넘기겠다는 설정.

Log4jdbc-log4j2 라이브러리를 사용하기 위한 jdbc설정을 바꾸기  
src>main>resources>application.properties  
jdbc.driver=net.sf.log4jdbc.sql.jdbcapi.DriverSpy ##com.mysql.cj.jdbc.Driver  
jdbc.url=jdbc:log4jdbc:mysql://localhost:3306/scoula\_db ##mysql://127.0.0.1:3306/scoula\_db  
jdbc.username=scoula  
jdbc.password=1234

Sql처리 핵심 로그만 출력하기 위한 로그 레벨 및 타켓 지정  
src>test>resources>log4j2  
<Loggers> 태그 안에 추가  
<Logger name=”jdbc” level=”WARN” additivity=”false”>  
 <AppenderRef ref=”console”/>  
</Logger>  
<Logger name=”jdbc.sqlonly” level=”INFO” additivity=”false”>  
 <AppenderRef ref=”console”/>  
</Logger>