JDBC를 이용한 간편 인증/권한 처리

1. 인증과 권한에 대한 처리는 크게 보면 Authentication Manager를 통해서 이루어 지는데 이때 인증이나 권한 정보를 제공하는 존재 Provider가 필요하다.

2. 테이블 설계

```
create table tbl_member( --회원 테이블
        userid varchar2(50)  primary key --회원 아이디
        ,userpw varchar2(100) not null -- 비밀번호
        ,username varchar2(100) not null -- 회원이름
        ,regdate date default sysdate --가입날짜
        ,updatedate date default sysdate --수정날짜
        ,enabled char(1) default '1'
 );

create table tbl_member_auth ( --권한 부여 테이블
      userid varchar2(50) not null --아이디
      ,auth varchar2(50) not null  --권한 부여
      ,constraint fk_member_auth foreign key(userid) references
tbl_member(userid) --외래키 설정
);
```

3. 스프링 시큐리티에서 제공되는 BCryptPasswordEncoder 클래스를 이용해서 패스워드를 암호화 시킨다.
그러므로 security-context.xml을 다음과 같이 수정한다.

```
        중략...

        <bean id="bcryptPasswordEncoder"

class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder" />
        <!-- 1. 스프링 시큐리티에서 제공되는 BCryptPasswordEncoder 클래스를
이용해서 패스워드를 암호화 처리한다.
                    2. bcrypt는 태생 자체가 패스워드를 저장하는 용도로 설계된
것으로 특정 문자열을 암호화 한다.
                        체크하는 쪽에서는 암호화 된 패스워드가 가능한
패스워드인지만 확인하고 다시 원래 원문으로 되돌리지는 못한다.
                    3. BCryptPasswordEncoder는 스프링 시큐리티 API 안에
포함되어 있다. -->
```

```
                        ..중략

        <security:authentication-manager>
                        <security:authentication-provider>
                         <security:jdbc-user-service  data-source-ref="dataSource"
/>
                         <!--시큐리티와 스프링 jdbc 연결 -->
                                <security:password-encoder
                                        ref="bcryptPasswordEncoder" />
                                <!-- BCryptPasswordEncoder 빈아이디
bcryptPasswordEncoder 호출 -->

                                </security:authentication-provider>
        </security:authentication-manager>
```

4.src/test/java 패키지에 net.abc.security 패키지를 생성하고, JUnit 테스트 파일 MemberTests.java 클래스 파일을 만든다.

```
package  net.abc.security;

import  java.sql.Connection;
import  java.sql.PreparedStatement;

import  javax.sql.DataSource;

import  org.junit.Test;
import  org.junit.runner.RunWith;
import  org.springframework.beans.factory.annotation.Autowired;
import  org.springframework.security.crypto.password.PasswordEncoder;
import  org.springframework.test.context.ContextConfiguration;
import  org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import  lombok.Setter;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration({
   "file:src/main/webapp/WEB-INF/spring/root-context.xml",
   "file:src/main/webapp/WEB-INF/spring/security-context.xml"
   })
public  class  MemberTests {
```

```
  @Setter(onMethod_ = @Autowired) //setter() 메서드 DI의존성 주입->자동의존성
//주입
  private PasswordEncoder pwencoder;

  @Setter(onMethod_ = @Autowired)
  private DataSource ds;//커넥션 풀 관리 ds생성

  @Test //JUnit 테스트
  public void testInsertMember() {

    String sql = "insert into tbl_member(userid, userpw, username) values
(?,?,?)";

    for(int i = 0; i < 100; i++) {

      Connection con = null; //오라클 연결
      PreparedStatement pstmt = null; //쿼리문 수행

      try {
        con = ds.getConnection();
        pstmt = con.prepareStatement(sql);

        pstmt.setString(2, pwencoder.encode("pw" + i));//비번 암호화

        if(i <80) {

          pstmt.setString(1, "user"+i);
          pstmt.setString(3,"일반사용자"+i);

        }else if (i <90) {

          pstmt.setString(1, "manager"+i);
          pstmt.setString(3,"운영자"+i);

        }else {

          pstmt.setString(1, "admin"+i);
          pstmt.setString(3,"관리자"+i);

        }
```

```
        pstmt.executeUpdate();

      }catch(Exception e) {
        e.printStackTrace();
      }finally {
        if(pstmt != null) { try { pstmt.close();  } catch(Exception e) {} }
        if(con != null) { try { con.close();  } catch(Exception e) {} }
      }
    }//end for
}


@Test
public void testInsertAuth() {

    String sql = "insert into tbl_member_auth (userid, auth) values (?,?)";

    for(int i = 0; i < 100; i++) {

      Connection con = null;
      PreparedStatement pstmt = null;

      try {
        con = ds.getConnection();
        pstmt = con.prepareStatement(sql);


        if(i <80) {
          pstmt.setString(1, "user"+i);
          pstmt.setString(2,"ROLE_USER");

        }else if (i <90) {

          pstmt.setString(1, "manager"+i);
          pstmt.setString(2,"ROLE_MEMBER");

        }else {
          pstmt.setString(1, "admin"+i);
          pstmt.setString(2,"ROLE_ADMIN");
```

```
          }
          pstmt.executeUpdate();

        }catch(Exception e) {
          e.printStackTrace();
        }finally {
          if(pstmt != null) { try { pstmt.close();  } catch(Exception e) {} }
          if(con != null) { try { con.close();  } catch(Exception e) {} }

        }
      }//end for
    }
}
```

5. 인증을 하는데 필요한 쿼리 users-by-username-query 와 권한을 확인하는데 필요한 쿼리 authorities-by-username-query 를 사용하여 security-context.xml 일부를 수정한다.

```
<security:authentication-manager>
                        <security:authentication-provider>
                    <!--  <security:jdbc-user-service data-source-ref="dataSource"
/> -->

                        <security:password-encoder
ref="bcryptPasswordEncoder" />

                        <security:jdbc-user-service
                                    data-source-ref="dataSource"
                                    users-by-username-query="select
                                    userid , userpw , enabled from
tbl_member where userid = ? "
                                    authorities-by-username-query="select
                                    userid, auth from tbl_member_auth
where userid = ? " />
          </security:authentication-provider>
          </security:authentication-manager>
```

6. 구글 크롬에서 /sample/admin 매핑주소로 접근하여 아이디 admin90, 비번 pw90 으로 로그인 하면 ROLE_ADMIN 즉 ADMIN 페이지로 정상적으로 로그인 된다.

모델 dao와 mybatis매퍼 태그, JUnit테스트 연결하기

1. 데이터 저장빈 클래스를 각각 만든다.

```
package net.abc.vo;

import java.util.Date;
import java.util.List;

import lombok.Data;

@Data
public class MemberVO {

	private String userid;
	private String userpw;
	private String userName;
	private boolean enabled;

	private Date regDate;
	private Date updateDate;
	private List<AuthVO> authList;

}
```

```
package net.abc.vo;



import lombok.Data;

@Data //setter(),getter(),toString(),기본생성자까지 자동생성
public class AuthVO {

  private String userid;
  private String auth;
}
```

2. 모델 dao를 만든다.
   먼저 MemberMapper 인터페이스를 만든다.

```
package net.abc.dao;
```

```
import net.abc.vo.MemberVO;

public interface MemberMapper {

	public MemberVO read(String userid);
}
```

MemberMapper 인터페이스를 구현상속한 자손클래스 DAOImpl을 만든다.

```
package net.abc.dao;

import org.apache.ibatis.session.SqlSession;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import net.abc.vo.MemberVO;

@Repository
public class MemberMapperDAOImpl implements MemberMapper {

	@Autowired
	private SqlSession sqlSession;

	@Override
	public MemberVO read(String userid) {
		return this.sqlSession.selectOne("read",userid);
	}
}
```

3. src/main/resources 밑에 net/abc/mappers/member 하위에 member.xml 매퍼 태그를 만든다.

```
<?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="Member">

	<resultMap type="member" id="memberMap">
<!-- 이건 결과 매핑의 가장 기본적인 형태이다. id와 result 모두 한개의 칼럼을 한개의 프로퍼티(멤버변수)나 간단한 데이터 타입의 필드에 매핑한다.
```

```
둘 사이의 차이점은 id 값은 객체 인스턴스를 비교할 때 사용되는 구분자 프로퍼티로
처리되는 점이다. 이 점은 일반적으로 성능을 향상시키지만 특히 캐시와 내포된(nested)
결과 매핑(조인 매핑)의 경우에 더 그렇다. -->
                <id property="userid" column="userid" />
                <result property="userid" column="userid" />
                <result property="userpw" column="userpw" />
                <result property="userName" column="username" />
                <result property="regDate" column="regdate" />
                <result property="updateDate" column="updatedate" />
                <collection property="authList" resultMap="authMap">
                </collection><!--컬렉션 제네릭 타입으로 복수개 값을 반환할수 있다.
resultMap id authMap을 참조한다.-->
        </resultMap>

        <resultMap type="authvo" id="authMap">
                <result property="userid" column="userid" />
                <result property="auth" column="auth" />
        </resultMap>

        <select id="read" resultMap="memberMap"
                parameterType="java.lang.String">
                SELECT
                mem.userid, userpw, username, enabled, regdate, updatedate,
auth
                FROM
                tbl_member mem LEFT OUTER JOIN tbl_member_auth auth on
mem.userid = auth.userid
                WHERE mem.userid = #{userid}
        </select>
</mapper>
<!--  LEFT Outer Join기법 : From 절다음에 테이블명을
          기술할때 왼쪽,오른쪽에 기술하는데 오른쪽 테이블
           명에 데이터가 없는 경우 사용하는 조인기법이다.-->
```

4. src/test/java 경로에 net.abc.mapper 패키지를 만들고 JUnit 테스트 파일을 생성한다.

```
package net.abc.mapper;



import org.junit.Test;
```

```
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringRunner;

import lombok.Setter;
import lombok.extern.log4j.Log4j;
import net.abc.dao.MemberMapper;
import net.abc.vo.MemberVO;

@RunWith(SpringRunner.class)
@ContextConfiguration({"file:src/main/webapp/WEB-INF/spring/root-context.xml"})
@Log4j //로그 기록
public class MemberMapperTests {

  @Setter(onMethod_ = @Autowired) //setter()메서드를 통한 자동의존성 주입
  private MemberMapper mapper;



  @Test //JUnit 테스트
  public void testRead() {

    MemberVO vo = mapper.read("admin90"); //admin90 아이디에 대한 회원정보
//와 권한을 검색

    log.info(vo);//vo 객체값을 로그로 이클립스 콘솔모드에 출력
    //System.out.println(vo);

    vo.getAuthList().forEach(authVO -> log.info(authVO));//람다식으로 아이디와
//권한을 로그로 이클립스 콘솔모드
    //에 출력
    //vo.getAuthList().forEach(authVO -> System.out.println(authVO));
  }

}
```

자동 로그인(remember-me)

1. 최근 웹페이지들은 '자동로그인' 이나 '로그인 기억하기'라는 이름으로 한 번 로그인하면 일정 시간 동안 다시 로그인을 하지 않아도 되는 기능을 가지고 있다. 영어로는 'remember-me' 라고 표현한다. 이 기능은 거의 대부분 쿠키(Cookie)를 이용해서 구현한다.

2. 스프링 시큐리티에서는 'remember-me' 기능을 메모리 상에서 처리하거나 데이터베이스를 이용하는데 security-context.xml에서 <security:remember-me>태그를 이용해서 구현할 수 있다.

3. 자동 로그인 기능을 처리하는 방식 중에서 가장 많이 사용하는 방식은 로그인이 되었던 정보를 데이터베이스에 저장해 두었다가 사용자의 재방문시 세션에 정보가 없으면 데이터베이스를 조회해서 사용하는 방식이다. 서버의 메모리상에서만 저장하는 방식보다 좋은 점은 데이터베이스가 공유되기 때문에 좀 더 안정적으로 운영이 가능하다는 점이다.

4. 로그인 정보를 유지하는 테이블을 설계한다.

```
--스프링 시큐리티 자동로그인 정보를 유지하는 테이블
create table persistent_logins(
   username varchar2(64) not null --회원아이디
   ,series varchar2(64) primary key --비번
   ,token varchar2(64) not null --토큰 정보
   ,last_used timestamp not null --로그인 한 날짜 시간
);
```

5. 자동 로그인에서 데이터베이스를 이용하는 설정은 data-source-ref 속성만 사용하면 된다. 다음과 같이 security-context.xml을 수정한다.

```
중략...

 <bean id="customAccessDenied"
                class="net.abc.security.CustomAccessDeniedHandler" />
    <!--  CustomAccessDeniedHandler 빈 아이디 객체명 customAccessDenied
생성-->

    <bean id="customLoginSuccess"
                class="net.abc.security.CustomLoginSuccessHandler" />

                <bean id="bcryptPasswordEncoder"

class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder" />
                <!-- 1. 스프링 시큐리티에서 제공되는 BCryptPasswordEncoder
```

```
클래스를 이용해서 패스워드를 암호화 처리한다.
                        2. bcrypt는 태생 자체가 패스워드를 저장하는 용도로 설계된
것으로 특정 문자열을 암호화 한다.
                              체크하는 쪽에서는 암호화 된 패스워드가 가능한
패스워드인지만 확인하고 다시 원래 원문으로 되돌리지는 못한다.
                        3. BCryptPasswordEncoder는 스프링 시큐리티 API 안에
포함되어 있다. -->

          <bean id="customUserDetailsService"
                    class="net.abc.security.CustomUserDetailsService" />
                    <!-- CustomUserDetailsService 빈아이디 customUserDetailsService
등록-->

      <security:http> <!-- 스프링 시큐리티가 http에서 알수 있게 하는 시작 지점 설정
-->

      중략..

      <!--
                              <security:logout logout-url="/customLogout"
                              invalidate-session="true" />  로그아웃시
customLogout매핑주소가 실행됨.
                               invalidate-session="true"는 로그아웃시 세션을 무효화
시키는 설정이다. -->

                               <security:remember-me
                              data-source-ref="dataSource"
token-validity-seconds="604800" />
                              <!-- 자동로그인 기능(remember-me),
                              data-source-ref로 데이터 베이스를 이용,
token-validity-seconds속성은 쿠키 유효시간(초단위) -->

                              <security:logout logout-url="/customLogout"
                              invalidate-session="true" delete-cookies="remember-me"
/>
                        <!--   remember-me는 자동 로그인 에서 사용하는 쿠키이름을
삭제,invalidate-session="true"는 세션무효화 -->
      </security:http>

      <security:authentication-manager>
```

```
                        <security:authentication-provider
                        user-service-ref="customUserDetailsService">
                        <!-- 시큐리티 인증 제공자로 customUserDetailsService
빈아이디 호출 -->



                        <security:password-encoder
ref="bcryptPasswordEncoder" />



            </security:authentication-provider>
          </security:authentication-manager>

</beans>
```