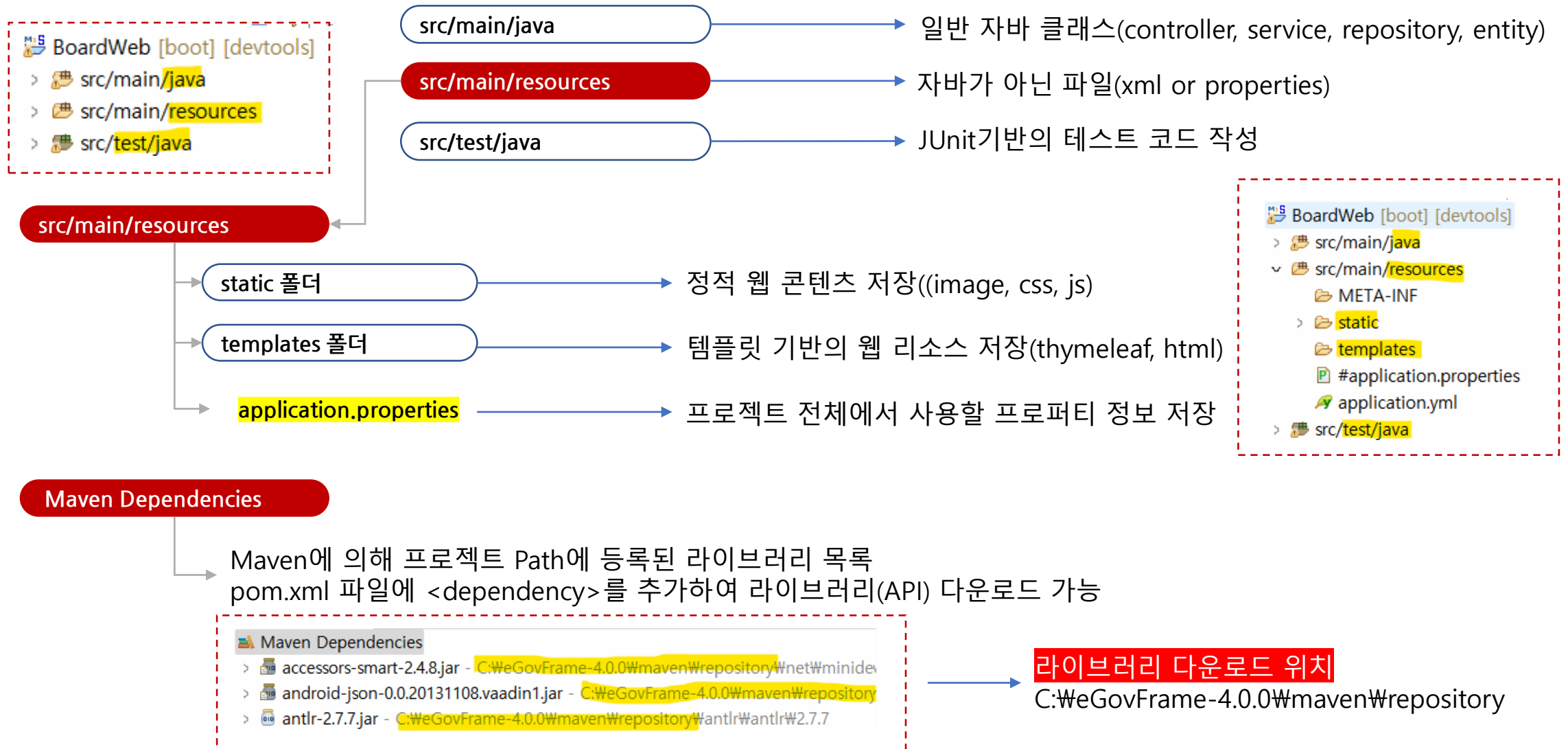


Spring boot 애플리케이션 개발

Spring BOOT, Spring JPA, Thymeleaf

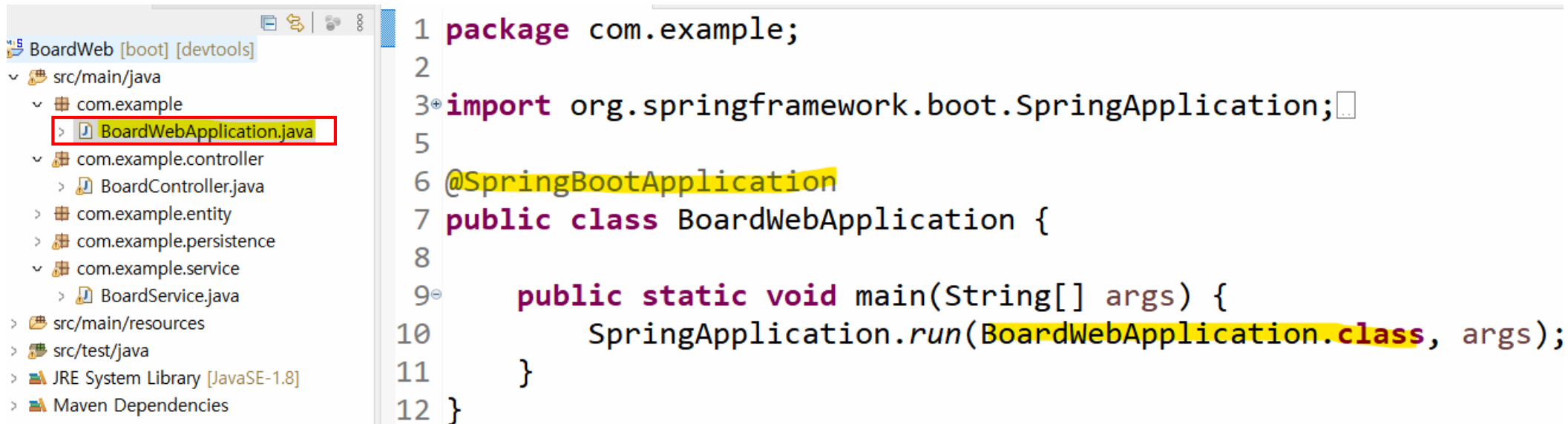
Hello Spring Boot~ 출력하기

1. 스프링 부트 Project 구조



2. 스프링 부트 애플리케이션 실행하기

웹 애플리케이션으로 실행



- > 프로젝트 생성시 지정한 package에 "프로젝트이름+Application" 형태의 자바 애플리케이션이 존재함
- > 스프링 부트 애플리케이션은 기본적으로 웹 애플리케이션으로 실행되며, 내장 Tomcat이 8080포트로 구동됨

Tomcat started on port(s): 8081 (http) with context path '/sp10'
Started BoardWebApplication in 13.84 seconds (JVM running for 21.333)
Initializing Spring DispatcherServlet 'dispatcherServlet'
Initializing Servlet 'dispatcherServlet'
Completed initialization in 1 ms

3. 외부 프로퍼티 사용하기

application.properties

애플리케이션의 환경을 관리하는 설정파일

-> 애플리케이션에서 사용하는 **설정 정보를 외부 프로퍼티로 분리** 하여 설정변경, 애플리케이션 기능 또는 동작 변경 가능

```
#WebApplicationType 설정
spring.main.web-application-type=none or servlet

#Application Banner 설정
spring.main.banner-mode=off or console

#Tomcat Server Port 변경
server.port=8081 or -1(랜덤포트)
```

src/main/resources 소스 폴더에서 application.properties 파일을 제거 후 application.yml 파일로 만들어서 사용 가능

들여쓰기 주의 (두 칸 띄우기)

Yaml 파일(야믈)

- > XML이나 JSON과 마찬가지로 데이터의 의미와 구조를 쉽게 전달하기 위한 파일
- > 기존의 XML이나 JSON보다 쉽게 작성 할 수 있고 가독성이 뛰어나
- > Properties를 이용한 애플리케이션 관리를 **야믈(yml)파일**로도 가능

```
#WebApplicationType 설정
spring:
  main:
    web-application-type: none or servlet

#Application Banner 설정
  banner-mode: off or console

#Tomcat Server Port 변경
server:
  port: 8081 or -1(랜덤포트)
```

4. Spring Boot 프로젝트 만들고 실행하기

application.properties

spring.main.web-application-type=servlet

spring.main.banner-mode=off

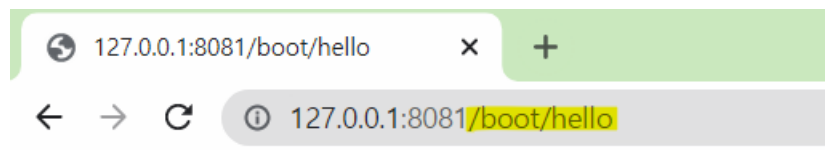
Tomcat Server Port 변경

server.port=8081

#Context path 수정

server.servlet.context-path=/boot

Hello Spring Boot~



Hello Spring Boot~

Spring boot 애플리케이션 개발

Spring BOOT, Spring JPA, Thymeleaf

Spring Boot 동작원리

1. 설정 파일의 단순화 및 Auto Configuration(자동설정)

application.properties

persistence.xml - 사용 불필요

```
server.port=8081
server.servlet.context-path=/sp10

logging.level.org.hibernate=info

#데이터베이스 설정(DataSource)
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/com
spring.datasource.username=com
spring.datasource.password=com01

#MySQL 상세지정
spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect
#스키마 생성(create,update)
spring.jpa.hibernate.ddl-auto=update
#실행되는 SQL문을 보여주기
spring.jpa.show-sql=true
#실제 JPA 구현체인 Hibernate가 동작하면서 발생하는 SQL을 포매팅해서 출력
spring.jpa.properties.hibernate.format_sql=true

#JSP View 경로 설정
spring.mvc.view.prefix=/WEB-INF/board/
spring.mvc.view.suffix=.jsp

<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
</dependency>
```

application.yml

```
server:
  port: 8081
  servlet:
    context-path: /sp10
logging:
  level:
    '[org.hibernate]': info
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/com
    username: com
    password: com01
  jpa:
    database-platform: org.hibernate.dialect.MySQL5InnoDBDialect
    show-sql: true
    hibernate:
      ddl-auto: create
    properties:
      hibernate:
        format_sql: true
  mvc:
    view:
      prefix: /WEB-INF/board/
      suffix: .jsp`
```

1-1. 설정 파일의 단순화 및 Auto Configuration(자동설정)

자동 설정이란?

- > 스프링 부트에서 복잡한 XML 설정 없이 웹 애플리케이션을 작성할 수 있는 이유는 **내부적으로 자동설정이 동작**하기 때문
- > 스프링이 제공하는 객체, 사용자가 작성한 객체를 자동으로 메모리에 로딩 하는 기능

@SpringBootApplication

메인 클래스 위에 기본적으로 설정되는 어노테이션, 아래 3개의 어노테이션으로 구성됨

@SpringBootConfiguration

환경 설정 클래스를 지정할 때 사용하는 어노테이션
@Configuration과 동일한 기능을 제공하며 이름만 변경한 것
@Bean 으로 설정된 클래스들을 생성한다.

@EnableAutoConfiguration

스프링 부트는 스프링컨테이너를 구동할 때, 두 단계로 나누어 객체를 생성함
① **@ComponentScan**이 동작하여 사용자가 작성한 객체들이 생성됨
② **@EnableAutoConfiguration**에 의해 자동설정 클래스에 의해 객체들이 생성됨
-> spring-boot-autoconfigure-x.x.x.jar 파일에 포함되어 있음
-> META-INF/**spring.factories** 파일에 등록된 **자동설정 클래스들을 처리함**

@ComponentScan

@Configuration, @Repository, @Service, @Controller, @RestController가 붙은 클래스의 객체를 메모리에 생성

META-INF

spring

org.springframework.boot.autoconfigure.AutoConfiguration.imports

1-2. 설정 파일의 단순화 및 Auto Configuration(자동설정)

META-INF/**spring.factories** 파일에 등록된 **자동설정 클래스들을 처리함**

웹 애플리케이션 개발과 관련된 자동 설정 클래스

org.springframework.boot.autoconfigure.web.servlet.**WebMvcAutoConfiguration**

```
@AutoConfiguration(after = { DispatcherServletAutoConfiguration.class,  
TaskExecutionAutoConfiguration.class,  
ValidationAutoConfiguration.class })  
@ConditionalOnWebApplication(type = Type.SERVLET)  
@ConditionalOnClass({ Servlet.class, DispatcherServlet.class, WebMvcConfigurer.class })  
@ConditionalOnMissingBean(WebMvcConfigurationSupport.class)  
@AutoConfigureOrder(Ordered.HIGHEST_PRECEDENCE + 10)  
public class WebMvcAutoConfiguration {
```

2. pom.xml의 API 설정 간소화(starter)

Maven을 이용한 라이브러리 의존성관리(pom.xml)

스프링 부트 스타터 (Starter : 라이브러리들의 묶음)

Available:

Type to search dependencies

- ▶ Developer Tools
- ▶ I/O
- ▶ Messaging
- ▶ Microsoft Azure
- ▶ NoSQL
- ▶ Observability
- ▶ Ops
- ▶ SQL
- ▶ Security
- ▶ Spring Cloud
- ▶ Spring Cloud Circuit Breaker

- > 스프링 부트 프로젝트는 Maven을 기반으로 프로젝트에서 필요한 의존성을 관리함
- > 스프링 부트의 스타터는 필요한 라이브러리들을 묶어서 **패키지 형태로 제공함**
- > 일반적으로 스타터는 관련된 라이브러리 의존성이 설정되지만 다른 스타터를 포함 하기도 함
- > 스프링 부트는 다양한 모듈의 스타터를 제공하며 **"spring-boot-starter-모듈명"** 형태의 이름을 가짐
- > 스타터는 스프링 부트 프로젝트를 생성 할 때 추가할 수 있다.
- > 프로젝트가 생성된 이후에는 pom.xml에서 **Ctrl + space**를 누르고 **Add Starters**... 를 이용하여 추가 할 수 있다.
- > Lombok 등 스타터가 아닌 라이브러리도 있음

〈특정 라이브러리 버전 정보 변경〉

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>5.3.7</version>
</dependency>
```

〈자바버전이나 모든 라이브러리 버전 정보 변경〉

```
<properties>
  <java.version>1.8</java.version>
  <spring-framework.version>5.3.7</spring-framework.version>
</properties>
```

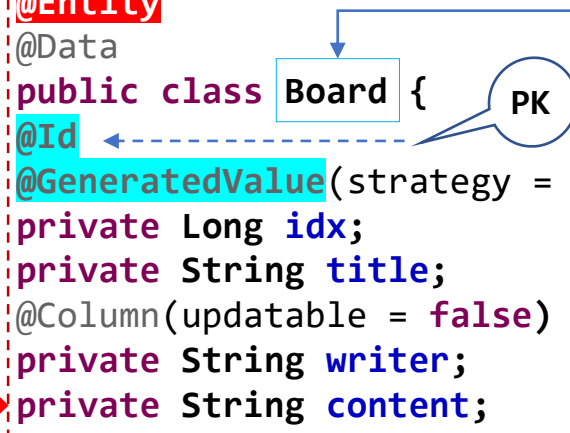
3. ORM 기술을 통한 데이터베이스 테이블 매핑(Entity, ORM)

VO 클래스

```
@Data
public class Board {
    private int seq;
    private String title;
    private String writer;
    private String content;
    private Date createDate;
    private int cnt;
}
```

Entity 클래스

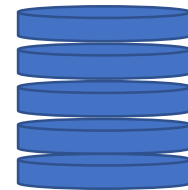
```
@Entity
@Data
public class Board {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long idx;
    private String title;
    @Column(updatable = false)
    private String writer;
    private String content;
    @Column(insertable = false, updatable = false,
        columnDefinition = "datetime default now()")
    private Date indate;
    @Column(insertable = false, updatable = false,
        columnDefinition = "int default 0")
    private Long count;
}
```



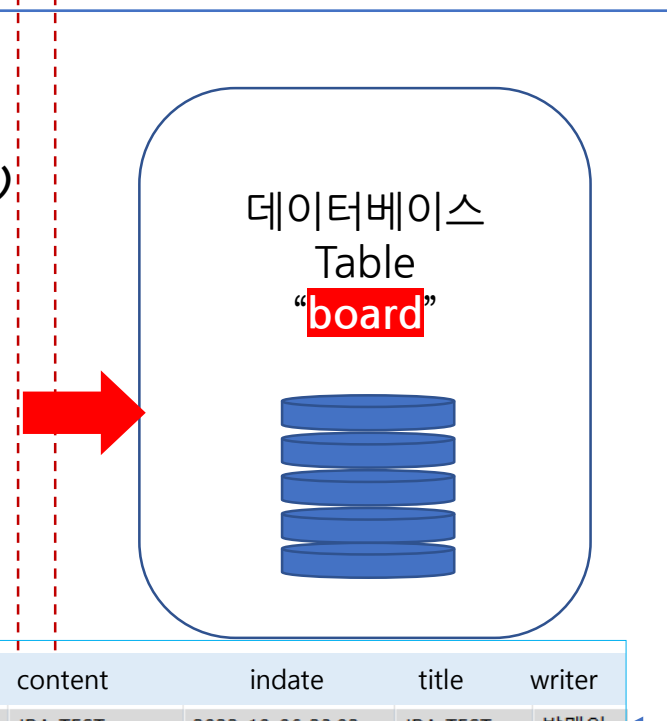
ORM

(Object Relational Mapping)

데이터베이스
Table
"board"



idx	count	content	indate	title	writer
1	0	JPA TEST	2022-10-06 23:02:...	JPA TEST	박매일
2	0	JPA TEST	2022-10-06 23:04:...	JPA TEST	관리자



3-1. ORM 기술을 통한 데이터베이스 테이블 매핑(Entity, ORM)

Hibernate:

```
drop table if exists board
```

Hibernate:

```
create table board (  
  idx bigint not null auto_increment,  
  content varchar(255),  
  count int default 0,  
  indate datetime default now(),  
  title varchar(255),  
  writer varchar(255),  
  primary key (idx)  
) engine=InnoDB
```

Connection profile
Type: MySQL_5.1 Name: com Database: Status: Connected, Auto Commit

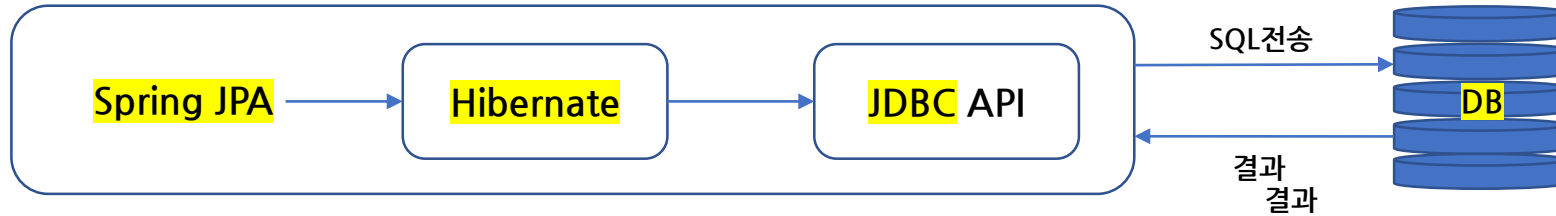
1 select * from board

Problems Tasks Properties Servers Bookmarks Console Javadoc SVN Repositories SQL Results Execution Plan DBIO Search Query

type query expression here Status Result1

idx	content	count	indate	title	writer
✓ Succes select * f...	2022. 9. ...	com			

3-2. Spring JPA를 통한 SQL 쿼리사용 절감(Spring JPA, Hibernate)



Spring MyBatis

Mapper interface

-> 구현체(SqlSessionFactory)

```

public interface BoardMapper {
    public List<Board> getList();
    public void insert(Board vo);
    public Board read(int idx);
    public void update(Board vo);
    public void delete(int idx);
}
  
```

Spring JPA

Repository interface

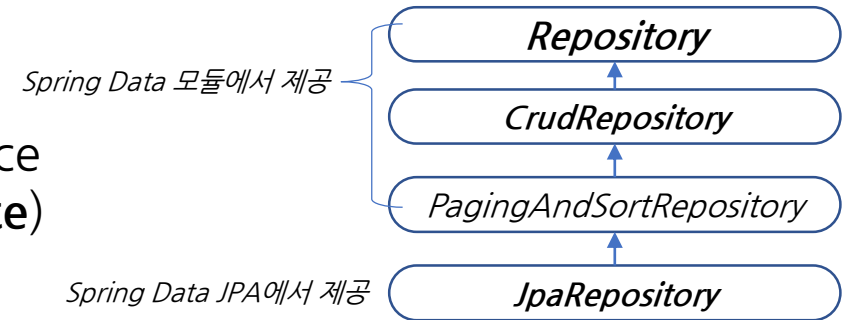
-> 구현체(Hibernate)

@Repository

```

public interface BoardRepository extends JpaRepository<Board, Long> {
}
  
```

SQL쿼리 사용 감소



- JPQL (Java Persistence Query Language)
- @Query
- Querydsl

* Spring Data JPA Reference

3-3. CrudRepository Method

All Methods	Instance Methods	Abstract Methods
Modifier and Type	Method and Description	
long	<code>count()</code>	Returns the number of entities available.
void	<code>delete(T entity)</code>	Deletes a given entity.
void	<code>deleteAll()</code>	Deletes all entities managed by the repository.
void	<code>deleteAll(Iterable<? extends T> entities)</code>	Deletes the given entities.
void	<code>deleteAllById(Iterable<? extends ID> ids)</code>	Deletes all instances of the type T with the given IDs.
void	<code>deleteById(ID id)</code>	Deletes the entity with the given id.
boolean	<code>existsById(ID id)</code>	Returns whether an entity with the given id exists.
<code>Iterable<T></code>	<code>findAll()</code>	Returns all instances of the type.
<code>Iterable<T></code>	<code>findAllById(Iterable<ID> ids)</code>	Returns all instances of the type T with the given IDs.
<code>Optional<T></code>	<code>findById(ID id)</code>	Retrieves an entity by its id.
<code><S extends T></code> <code>S</code>	<code>save(S entity)</code>	Saves a given entity.
<code><S extends T></code> <code>Iterable<S></code>	<code>saveAll(Iterable<S> entities)</code>	Saves all given entities.

쿼리 메서드 Repository 인터페이스에서 메소드 이름을 기반으로 JPQL을 생성하는 기능
검색대상이 Entity

find + 엔티티 이름 + By + 변수 이름

`findBoardByTitle();`

-> Board 엔티티에서 title 변수 값에 해당하는 검색SQL을 생성함

`public interface BoardRepository extends JpaRepository<Board, Long>{`

`findBoardByTitle();`

`findByTitle();`

} 동일한 결과

}

키워드	예	생성되는 JPQL
And	<code>findByTitleAndContent</code>	<code>where b.title=?1 and b.content=?2</code>
Or	<code>findByTitleOrContent</code>	<code>where b.title=?1 or b.content=?2</code>
Between	<code>findByCreateDateBetween</code>	<code>where b.createDate between ?1 and ?2</code>
LessThan	<code>findByCntLessThan</code>	<code>where b.cnt < ?1</code>
LessThanEqual	<code>findByCntLessThanEqual</code>	<code>where b.cnt <= ?1</code>
GreaterThan	<code>findByCntGreaterThan</code>	<code>where b.cnt > ?1</code>
GreaterThanEqual	<code>findByCntGreaterThanEqual</code>	<code>where b.cnt >= ?1</code>
IsNull	<code>findByWriterIsNull</code>	<code>where b.writer is null</code>
NotNull	<code>findByWriterNotNull</code>	<code>where b.writer is not null</code>
Not	<code>findByCntNot</code>	<code>where b.cnt <> ?1</code>
Containing	<code>findByContentContaining</code>	<code>where b.content like '% ?1 ' %'</code>
OrderBy	<code>findByContentOrderBySeqDesc</code>	<code>where b.content like '% ?1 ' %'</code> <code>order by b.seq desc</code>

3-4. CrudRepository Method

@Query 간단한 쿼리는 쿼리 메서드로 처리 가능하지만 조인이나 복잡한 쿼리는 @Query를 사용합니다.

1. 위치기반 파라미터 설정 **?1, ?2** 처럼 1부터 시작하는 파라미터의 순서를 이용하는 방식

```
@Query("select b from Board b where b.title like %?1% order by b.idx desc")  
List<Board> getSearchList(String keyword);
```

2. 이름기반 파라미터 설정 **:XXX** 처럼 :파라미터이름을 이용하는 방식

```
@Query("select b from Board b where b.title like %:keyword% order by b.idx desc")  
List<Board> getSearchList(@Param("keyword") String keyword);
```

4. 다양한 View template 사용(Thymeleaf, FreeMarker, Mustache, Groovy Templates 등)

JSP 사용시

-> pom.xml에 추가

```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
</dependency>

<dependency>
  <groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>
</dependency>
```

```
webapp
├── WEB-INF
│   └── board
│       ├── getList.jsp
│       └── header.jsp
```

#JSP View 경로 설정

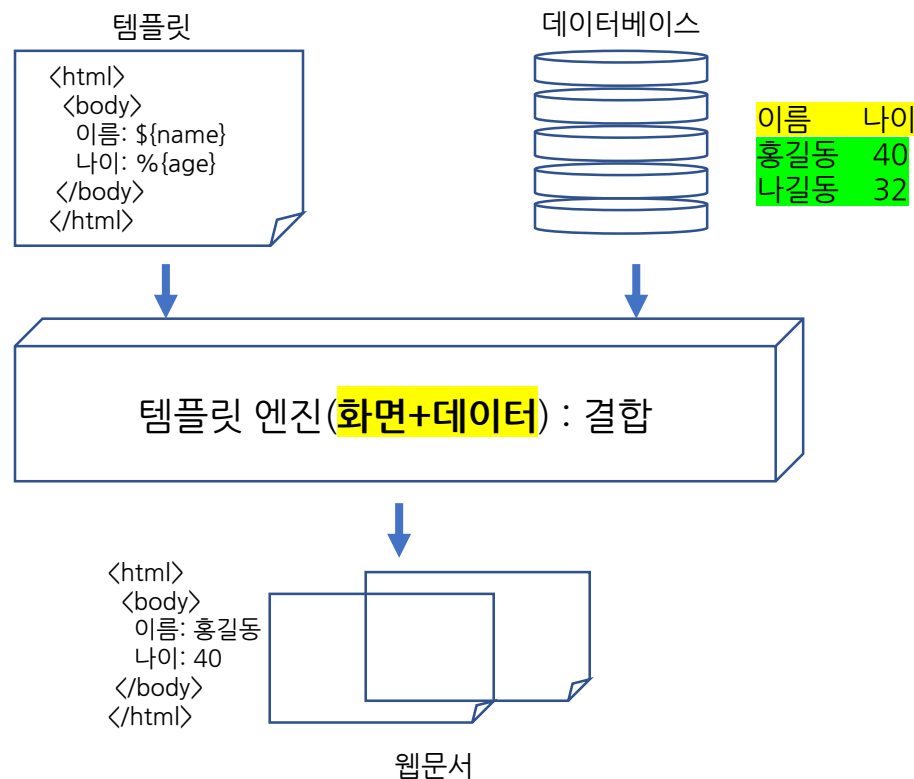
```
spring.mvc.view.prefix=/WEB-INF/board/
spring.mvc.view.suffix=.jsp
```



Thymeleaf

JSP + EL + JSTL => 템플릿 엔진

사용자에게 제공되는 **화면**과 **데이터** 처리 로직을 분리시켜서 개발 및 유지보수의 편의성을 증대 시킴



Spring boot 애플리케이션 개발

Spring BOOT, Spring JPA, Thymeleaf

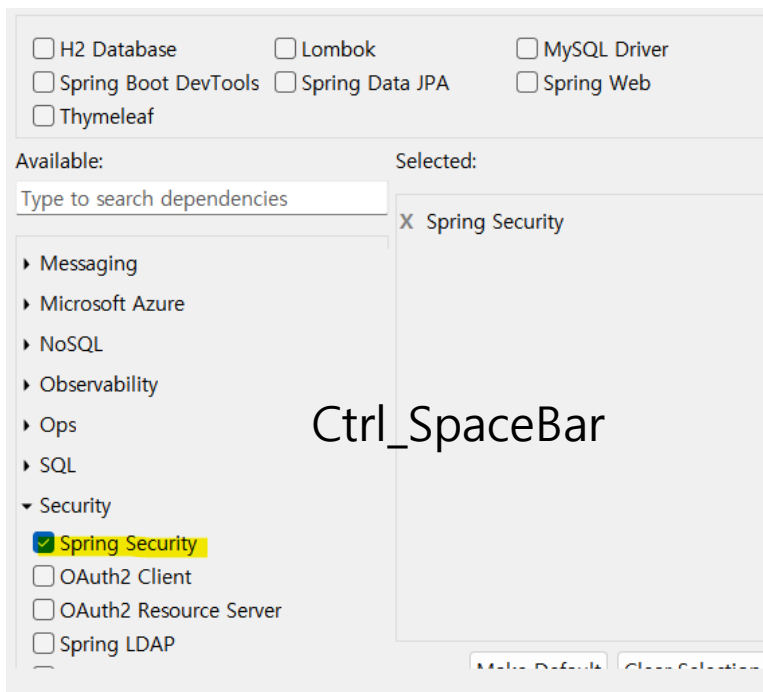
1. 인증과 인가 개념

인증(Authentication)

-> 시스템 사용자를 식별하는 것(로그인 처리)

인가(Authorization)

-> 인증된 사용자가 해당 기능을 실행할(접속할) 권한이 있는지 확인하는 것(권한체크)



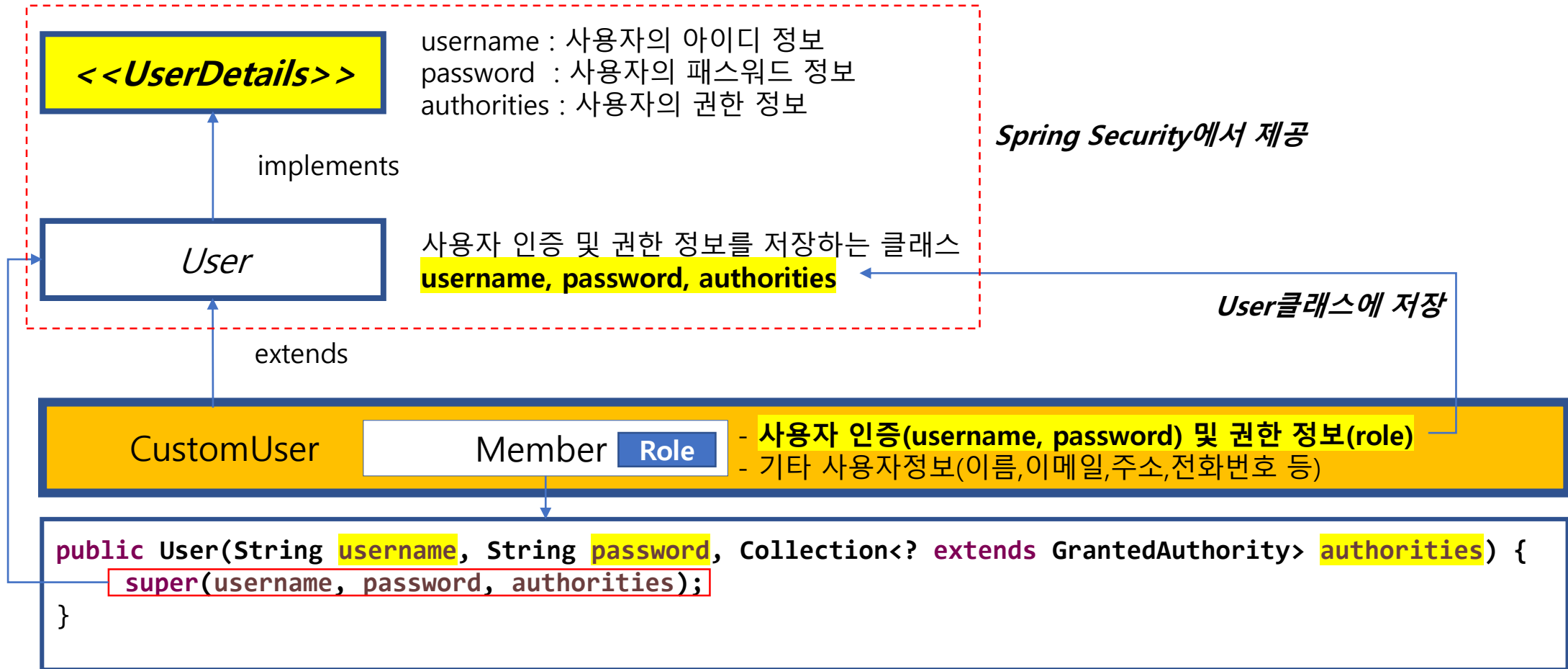
1. 시큐리티 스타터 추가
2. 애플리케이션 다시 실행
3. 시큐리티 관련 자동 설정 클래스 동작
4. 시큐리티 관련된 다양한 객체 생성

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-security</artifactId>
```

```
</dependency>
```

```
<dependency>  
  <groupId>org.springframework.security</groupId>  
  <artifactId>spring-security-taglibs</artifactId>  
</dependency>
```

2. 사용자 계정 정보를 저장할 Entity 클래스 만들기(CustomUser)



2-1. 사용자 계정 정보를 저장할 Entity 만들기(CustomUser)

```

@Entity
@Data
public class Member {
    @Id
    private String username;
    private String password;
    private String name;
    @Enumerated(EnumType.STRING) // EnumType.ORDINAL : 0 1 2
    private Role role;
    private boolean enabled; // 계정의 활성화/비활성화 여부
}

```

```

public enum Role {
    ADMIN, MANAGER, MEMBER;
}

```

```

@Data
public class CustomUser extends User{

    private Member member; // 사용자의 추가정보를 활용하기 위함

    public CustomUser(Member member) {
        super(member.getUsername(),
            member.getPassword(), AuthorityUtils.createAuthorityList("ROLE_" + member.getRole().toString()));
        this.member = member;
    }
}

```

3. UserDetails 인터페이스

UserDetails 란?

Spring Security에서 **사용자의 정보를 담은 인터페이스**이다.

Spring Security에서 사용자의 정보를 불러오기 위해서 구현해야 하는 인터페이스로 기본 오버라이드 메서드들은 아래와 같다.

메소드	리턴 타입	설명
getAuthorities()	Collection<? extends GrantedAuthority>	계정의 권한 목록을 리턴
getPassword()	String	계정의 비밀번호를 리턴
getUsername()	String	계정의 고유한 값을 리턴 (ex : DB PK값, 중복이 없는 이메일 값)
isAccountNonExpired()	boolean	계정의 만료 여부 리턴
isAccountNonLocked()	boolean	계정의 잠김 여부 리턴
isCredentialsNonExpired()	boolean	비밀번호 만료 여부 리턴
isEnabled()	boolean	계정의 활성화 여부 리턴

* UserDetails 인터페이스는 구현해야 될 메서드가 많아서 -> **User 클래스**를 사용하는 것이 편하다

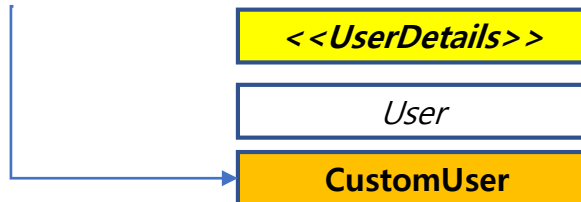
4. 사용자 정보를 가져오는 UserDetailsService 구현하기(Spring JPA 연동)

```

@Service
public class UserDetailsServiceImpl implements UserDetailsService{
    // Spring JPA이용하기
    @Autowired
    private MemberRepository memberRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        // username정보를 이용해서 상세조회 시도(회원이 존재하는지 체크)
        Member member=memberRepository.findById(username).get();
        if(member==null) {
            throw new UsernameNotFoundException(username+" 없음");
        }
        // UserDetails(로그인한 회원의 정보를 저장하는 인터페이스)를->구현한 User객체 생성하여 리턴한다.
        // return new User(member.getUsername(),
        //      "{noop}" + member.getPassword(), // 권한목록세팅
        //      AuthorityUtils.createAuthorityList("ROLE_" + member.getRole().toString()));
        return new CustomUser(member);
    }
}

```



로그인에 성공하면 CustomUser정보가 메모리에 등록 : JSP에서 활용

5. Spring Security 환경설정 클래스 만들기

@Configuration

```
public class SecurityConfiguration{
    @Autowired
    private UserDetailsServiceImpl userDetailsService;

    @Bean
    public PasswordEncoder passwordEncoder() {
        return PasswordEncoderFactories.createDelegatingPasswordEncoder();
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception{

        http.csrf().disable();// CSRF인증 토큰 비활성화
        http.authorizeHttpRequests() // 사용자의 요청을 핸들링 할 수 있다.
        // "/", "/member/**" 파일에 대한 접근은 무조건 허용한다.
            .antMatchers("/", "/member/**").permitAll()
            // board경로 접근은 인증된 사용자만 허용한다.
            .antMatchers("/board/**").authenticated()
            .and()
            // 로그인을 해야되는 경우에 이동하는 페이지
            // 로그인이 성공하면 /board/list로 이동한다.
            .formLogin()
            .loginPage("/member/login")
            .defaultSuccessUrl("/board/list")
            .and()
            .logout()
            .logoutUrl("/member/logout")
            .logoutSuccessUrl("/");

        http.userDetailsService(userDetailsService);
        return http.build();
    }
}
```

로그인 폼을 제공할것다.

/member/login으로 요청이 오면(**get**방식)
사용자 로그인 페이지로 이동하게 한다.

로그인 페이지로 이동한 후에

username :

password :

입력 후 **post**방식으로 **/member/login**을 요청하면
Spring Security인증이 실행된다.
인증이 성공하면 / board/list로 이동한다.

6. 비밀번호 암호화(사용자 입력)

```
@SpringBootTest
```

```
class BoardWebApplicationTests {
```

```
@Autowired
```

```
private MemberRepository memberRepository;
```

```
@Autowired
```

```
private PasswordEncoder encoder;
```

```
@Test
```

```
void memberTest() {
```

```
    Member member=new Member();
```

```
    member.setUsername("parkcell");
```

```
    member.setPassword(encoder.encode("parkcell"));
```

```
    member.setName("박에셀");
```

```
    member.setRole(Role.MEMBER);
```

```
    member.setEnabled(true);
```

```
    memberRepository.save(member);
```

```
}
```

```
}
```

Status	Result1	username	enabled	name	password	role
1		bitcocom	true	박매일	bitcocom	MEMBER
2		parkcell	true	박에셀	{bcrypt}\$2a\$10\$gDYS5sTZ9Ymhb5cwppNk7.LqtnbHKZzme9CCUpnTz3BNSrq.YDSDC	MEMBER
3		test	true	손님	test123	ADMIN



7. JSP에서 로그인한 사용자 정보 보여주기

1. JSP상단에 Spring Security관련 태그 라이브러리 등록

```
<%@taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>
```

2. <sec:authentication> 태그와 *principal* 이라는 이름의 속성을 사용

<sec:authentication property="principal"/> → UserDetailsService에서 반환된 객체(CustomUser)

<sec:authentication property="principal.member"/> CustomUser객체의 getMember()를 호출

<sec:authentication property="principal.member.name"/> 사용자 이름 가져오기

<sec:authentication property="principal.member.role"/> 사용자 권한정보 가져오기

7-1. 표현식을 이용하여 동적 화면 구성

표현식	설명
hasRole('role')	해당 권한이 있을 경우
hasAnyRole('role1','role2')	포함된 권한 중 하나라도 있을 경우
isAuthenticated()	권한에 관계없이 로그인 인증을 받은 경우
isFullyAuthenticated()	권한에 관계없이 인증에 성공했고, 자동 로그인이 비활성인 경우
isAnonymous()	권한이 없는 익명의 사용자일 경우
isRememberMe()	자동 로그인을 사용하는 경우
permitAll	모든 경우 출력함
denyAll	모든 경우 출력하지 않음

```
<!-- 표현식이 지정한 권한에 맞을 때만 출력 -->
```

```
<sec:authorize access="isAnonymous()">
```

**로그인
회원가입**

```
</sec:authorize>
```

```
<sec:authorize access="isAuthenticated()">
```

**로그아웃
회원정보보기**

```
</sec:authorize>
```

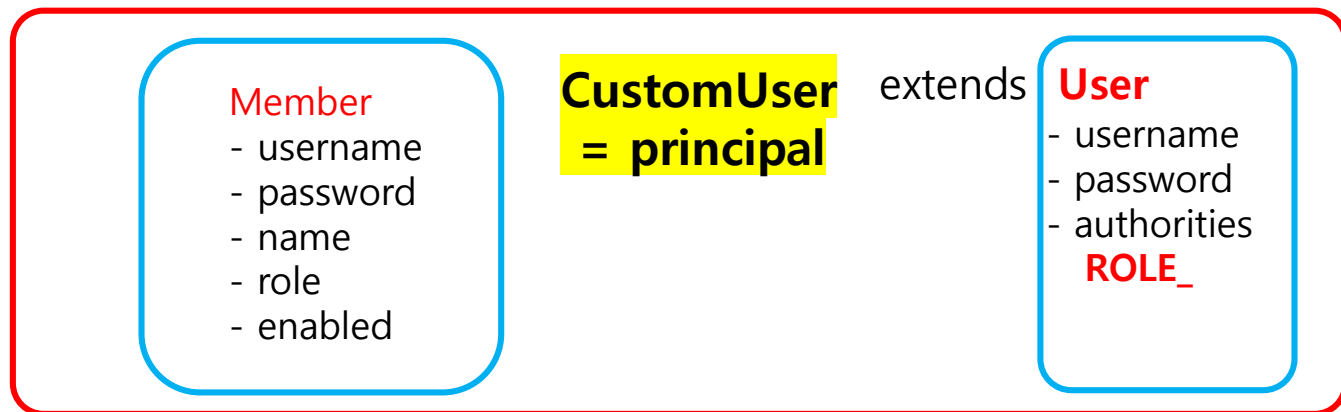
```
<sec:authorize access="hasRole('ROLE_ADMIN')">
```

관리자 페이지

```
</sec:authorize>
```

7-2. EL을 이용하여 사용자 정보 출력

```
<c:set var="user" value="${SPRING_SECURITY_CONTEXT.authentication.principal}"/>
<c:set var="auth" value="${SPRING_SECURITY_CONTEXT.authentication.authorities}"/>
```



```

console.log('${user}');           → CustomUser(member=Member(username=bitcocom, password=
                                   {bcrypt}$2a$10$1TISkmAx66tXZ6bDhAQIH.eONGsTApXct.IkNvspmXkGyIftprZHW, name=박매일, role=ADMIN,
                                   enabled=true))
console.log('${user.member}');    → Member(username=bitcocom, password=
                                   {bcrypt}$2a$10$1TISkmAx66tXZ6bDhAQIH.eONGsTApXct.IkNvspmXkGyIftprZHW, name=박매일, role=ADMIN,
                                   enabled=true)
console.log('${user.username}');  → bitcocom
console.log('${user.member.username}'); → bitcocom
console.log('${auth[0]}');        → [ROLE_ADMIN]
  
```

8. Bootstrap 4 로그인 페이지 만들기

<https://bootsnipp.com/tags/login/4>

