




Spring annotation

- **@EqualsAndHashCode**
 - Equals 와 hashCode 자동 생성
 - Equals의 경우 두 객체의 내용이 같은지, 동등성 비교 연산
 - Hashcode의 경우 두 객체가 같은 객체인지, 동일성 비교 연산
 - 기본적으로 @EqualsAndHashCode(callSuper = false)입니다.
 - callSuper를 true로 했을 때는, equals와 hashCode 메소드 자동 생성시 부모 클래스의 필드까지 감안해서 비교 연산을 진행합니다.



@EqualsAndHashCode 예시

```
@Entity
@Getter @Setter @EqualsAndHashCode(of="id")
@Builder @AllArgsConstructor @NoArgsConstructor
public class Account {

    @Id @GeneratedValue
    private Long id;

    @Column(unique= true)
    private String email;

    @Column(unique= true)
    private String nickname;

    private String password;

    private boolean emailVerified;

    private String emailCheckToken;

    private LocalDateTime joinedAt;

    private String bio;


    private String url;

    private String occupation;

    private String location;
```



@Lob

- 컨텐츠의 길이가 너무 길 경우 바이너리 파일로 DB에 저장해야 합니다. 보통 이런 경우에 @Lob를 사용합니다.
 - 속성 타입이 string, char[] 이면 CLOB으로 매핑 되고
 - 그 외에는 BLOB으로 매핑 됩니다.
- 



@Basic


- 테이블의 단순타입 column 매핑에 사용됩니다.
- Argument fetch 는 eager(즉시 로딩), lazy(지연 로딩) 지정 가능하고, eager가 기본 값이면 optional을 true, false 형태로 지정 가능합니다.

수업 예시)

```
@Lob @Basic(fetch=FetchType.EAGER)
private String profileImage;
```



@enablewebsecurity

- 웹 보안을 활성화하기 위해 사용합니다.
 - 이 어노테이션 자체로는 유용하지 않고, 스프링 시큐리티가 `WebSecurityConfigurer`를 구현 혹은
 - 컨텍스트의 `WebSecurityConfiguredAdapter`를 확장한 빈으로 설정되어 있어야 합니다.
- 



@enablewebsecurity

```
@Configuration
@EnableWebSecurity
@RequiredArgsConstructor
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    private final AccountService accountService;
    private final DataSource dataSource;

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .mvcMatchers(...patterns: "/", "/login", "/sign-up", "/check-email-token",
                "/email-login", "/login-by-email", "/search/study").permitAll()
            .mvcMatchers(HttpMethod.GET, ...mvcPatterns: "/profile/*").permitAll()
            .anyRequest().authenticated();

        http.formLogin().loginPage("/login").permitAll();
        http.logout().logoutSuccessUrl("/");

        http.rememberMe()
            .userDetailsService(accountService)
            .tokenRepository(tokenRepository());
    }
}
```

Spring validator를 사용시 @valid annotation으로 검증이 필요한 객체를 가져오기 전에 수행할 method를 지정해주는 annotation입니다.

@InitBinder

```
@InitBinder("signUpForm")
public void initBinder(WebDataBinder webDataBinder){
    webDataBinder.addValidators(signUpFormValidator);
}

@GetMapping("/sign-up")
public String signUpForm(Model model){
    //model.addAttribute("signUpForm", new SignUpForm());
    model.addAttribute( new SignUpForm());
    return "account/sign-up";
}

@PostMapping("/sign-up")
public String signUpSubmit (@Valid SignUpForm signUpForm, Errors errors){
    if(errors.hasErrors()){
        return "account/sign-up";
    }

    Account account = accountService.processNewAccount(signUpForm);
    accountService.login(account);
    return "redirect:/";
}
```


@Transactional

- 클래스, 메소드에 @Transactional이 선언되면 해당 클래스에 트랜잭션이 적용된 프록시 객체 생성
- 프록시 객체는 @Transactional이 포함된 메서드가 호출될 경우, 트랜잭션을 시작하고 Commit or Rollback을 수행

```
@Transactional(readOnly = true)
@Override
public UserDetails loadUserByUsername(String emailOrNickname) throws UsernameNotFoundException {
    Account account = accountRepository.findByEmail(emailOrNickname);
    if(account == null){
        account= accountRepository.findByNickname(emailOrNickname);
    }

    if(account==null){
        throw new UsernameNotFoundException(emailOrNickname);
    }

    return new UserAccount(account);
}
```

@AuthenticationPrincipal

- 스프링 시큐리티가 제공해주는 기능 중에 하나로 스프링 **MVC** 핸들러 파라미터에 @AuthenticationPrincipal를 사용하면 getPrincipal() 로 리턴 받을 수 있는 객체를 바로 주입받을 수가 있음
- 있는 지 없는지만 확인하는 용도로 사용
- 인증 안한 경우에 null
- 인증 한 경우에는 **username**과 **authorities** 참조 가능

@AuthenticationPrincipal

- 강의에서는 CurrentUser라는 어노테이션 만들어서 사용.
- Anonymous user라면 null, 아닌 경우에는 실제 account객체로 받아오도록 변경.

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.PARAMETER)
@AuthenticationPrincipal(expression = "#this == 'anonymousUser' ? null : account" )
public @interface CurrentUser {

}
```

@PathVariable,

- URL 경로에 변수를 넣어주기도 하고,
- 수업 예시처럼
@PathVariable 파라미터를 사용하면
URI의 일부를 변수로 전달할 수 있다.

```
@GetMapping("/profile/{nickname}")
public String viewProfile(@PathVariable String nickname, Model model, @CurrentUser Account account){
    Account byNickname = accountRepository.findByNickname(nickname);
    if(nickname==null){
        throw new IllegalArgumentException(nickname + "에 해당하는 사용자가 없습니다.");
    }
    model.addAttribute(byNickname);
    model.addAttribute(s: "isOwner", byNickname.equals(account));
    return "account/profile";
}
```

Thymeleaf

- 타임리프는 템플릿 엔진 중 하나인데, 템플릿 엔진이란 html과 데이터를 결합한 결과물을 만들어주는 도구입니다.
- CSRF 토큰 또한 인증해줍니다(사이트 간 요청 위조 검증).
- 별도의 설정을 추가해야만 이용할 수 있습니다.
- <dependency>
- <groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-thymeleaf</artifactId>
- </dependency>

Thymeleaf 속성

표현식

- 변수 : `${...}`
- 선택 변수 : `*{...}`
- 메시지 : `#{...}`
- Link URL : `@{...}`

text operation

- 문자열 연결 : `+`
- 문자 대체 : `|The name is ${name}|`

이외에 다른 것들은 자바스크립트와 동일

Thymeleaf 예시

- ```
<!DOCTYPE html>
<html lang="en"
 xmlns:th="http://www.thymeleaf.org">
<head th:replace="fragments.html :: head"></head>
<body class="bg-light">
<div th:replace="fragments.html :: main-nav"></div>
<div class="container">
 <div class="row mt-5 justify-content-center">
 <div class="col-2">
 <!-- Avatar -->
 <svg th:if="{#strings.isEmpty(account.profileImage)}"
class="img-fluid float-left rounded img-thumbnail"
 th:data-jdenticon-value="{account.nickname}"
width="125" height="125"></svg>

 </div>
```

# Thymeleaf fragment

---

- Thymeleaf에서 공통 레이아웃 처리를 위해서 fragment 기능을 제공하고 있습니다.

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org" xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
<head th:fragment="head">
 <meta charset="UTF-8">
 <title>Study</title>
 <link rel="stylesheet" href="/node_modules/bootstrap/dist/css/bootstrap.min.css"/>
 <link rel="stylesheet" href="/node_modules/font-awesome/css/font-awesome.min.css"/>
 <script src="/node_modules/jquery/dist/jquery.min.js"></script>
 <script src="/node_modules/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
 <script src="/node_modules/jdenticon/dist/jdenticon.min.js"></script>
 <style>
 .container {
 max-width: 100%;
 }
 </style>
</head>
```



# Thymeleaf fragment

- 앞에서 본 fragment 이외에도, namespace, title 등이 있습니다.

## namespace 정의

```
1 <!DOCTYPE html>
2 <html xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">
3 ...
4 </html>
```

*Colored by Color Scripter* 

html에 이렇게 정의하면 레이아웃 화면으로 인식됩니다.

## title 처리

```
1 <title layout:title-pattern="$LAYOUT_TITLE : $CONTENT_TITLE">Eblo</title>
```



이렇게 하면 "Layout에 정의된 타이틀+ ' : '+컨텐츠페이지 타이틀" 형태로 생성됩니다.

## 결과 화면

```
1 <title>Eblo : Thymeleaf 예제</title>
```



# Thymeleaf fragment

- Fragment에 있는 것을 가져와서 사용하는 방법은 insert, replace, include가 있습니다.

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
 <head th:replace="fragments.html :: head"></head>
 <body class="bg-light">
 <div th:replace="fragments.html :: main-nav"></div>
 <div class="alert alert-warning" role="alert" th:if="{account!=null && !account.emailVerified}">
 스타디 가입을 완료하려면 계정 인증 이메일을 확인 하세요
 </div>
 <div class="container">
 <div class="py-5 text-center">
 <h2>스타디</h2>
 </div>
 <div th:replace="fragments.html :: footer"></div>
 </div>
 </body>
</html>
```

# Thymeleaf fragment

- Include와 replace는 태그를 제외하고 제외하고 결과를 가져오고 th:insert는 fragment 태그를 포함해서 가져오는 차이가 있습니다.

태그와 fragment를 사용하는 예제

```
1 <footer th:include="fragments/commonFooter :: commonFooter"></footer>
2 <footer th:replace="fragments/commonFooter :: commonFooter"></footer>
3 <footer th:insert="fragments/commonFooter :: commonFooter"></footer>
```

결과 화면

```
1 <footer>
2 Layout Common Footer
3 </footer>
4 <footer>
5 Layout Common Footer
6 </footer>
7 <footer><footer>
8 Layout Common Footer
9 </footer></footer>
```

감사합니다