[Spring] Interceptor (2) - 어노테이션 작성 및 접근 권한, 세션 처리

2018. 3. 31. 18:44

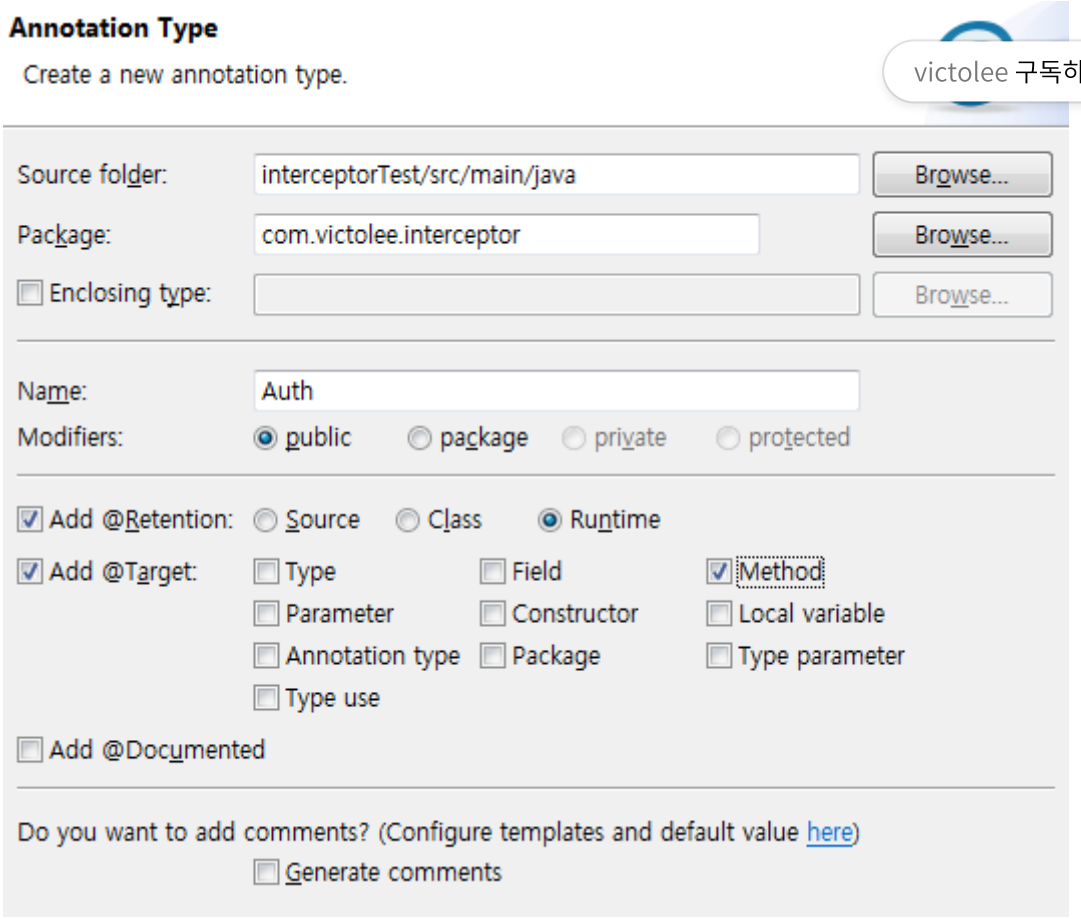
이전 글에서 Interceptor가 무엇인지 알아보았습니다.
이번에는 어노테이션을 직접 작성하여 이 어노테이션이 interceptor 기능을 하여 접근 권한과 세션을 구현할 수 있도록 하려고 합니다.

지금까지 세션을 처리하는 방법으로는 HttpSession 이라는 객체를 사용했는데, 이는 톰캣에서 제공하는 객체입니다.
스프링은 기술 비침투를 추구하므로, HttpSession 객체를 사용하여 세션을 처리하는 것은 스프링 답지 못한 개발입니다.
이제 세션을 처리하기 위한 어노테이션을 만들어서 interceptor를 통해 세션을 구현해보도록 하겠습니다.

1. 접근 권한 - @Auth 어노테이션

@Auth 어노테이션은 클라이언트가 로그인된 회원인지 아닌지, 그리고 일반 회원인지 관리자인지 구분할 용도인 어노테이션입니다.

1) @Auth 어노테이션 생성



어노테이션 생성할 때 위의 창이 팝업될 것입니다.

@Retention을 체크하여, Runtime을 클릭합니다.
세션을 처리하기 위한 어노테이션이므로, 실행 시에 동작을 해야합니다.

@Target도 체크를 하는데, 세션을 처리하기 위해서 메서드에 어노테이션을 작성할 것이므로 메서드에 체크를 합니다.

그리고 아래와 같이 코드를 작성합니다.

com.victolee.security/Auth.java

```
@Retention(RUNTIME)
@Target(METHOD)
public @interface Auth {
        public enum Role {ADMIN, USER}

        // 이와 같이 작성하면 메서드 위에 @Auth(role=Role.ADMIN)과 같이 작성 가능
        public Role role() default Role.USER;
}
```

접근 권한이 필요한 메서드 위에 @Auth 어노테이션을 작성하면 접근 권한이 있는 사용자인지 아닌지 판별할 수 있습니다.
enum의 역할은 관리자만 접근할 수 있는지, 일반 사용자도 접근할 수 있는지 역할을 부여하기 위해서입니다.

예를들어 어떤 경로에 비회원 , 회원 , 관리자가 접근 할 수 있는 범위가 다르다면 @Auth 어노테이션을 컨트롤러의 메서드 위에 명시함으로써 접근 권한을 둘 수 있습니다.

이제 접근 권한 판별 로직을 수행하는 클래스를 작성하겠습니다.

2)

com.victolee.security/AuthInterceptor.java

```
public class AuthInterceptor extends HandlerInterceptorAdapter{

	@Override
	public boolean preHandle(HttpServletRequest request, HttpServletResponse response,
                                                        Object handler) throws Exc
		// 1. handler 종류 확인
		// 우리가 관심 있는 것은 Controller에 있는 메서드이므로 HandlerMethod 타입인지 체크
		if( handler instanceof HandlerMethod == false ) {
			// return true이면  Controller에 있는 메서드가 아니므로, 그대로 컨트롤러로
			return true;
		}

		// 2.형 변환
		HandlerMethod handlerMethod = (HandlerMethod)handler;

		// 3. @Auth 받아오기
		Auth auth = handlerMethod.getMethodAnnotation(Auth.class);

		// 4. method에 @Auth가 없는 경우, 즉 인증이 필요 없는 요청
		if( auth == null ) {
			return true;
		}

		// 5. @Auth가 있는 경우이므로, 세션이 있는지 체크
		HttpSession session = request.getSession();
		if( session == null ) {
			// 로그인 화면으로 이동
			response.sendRedirect(request.getContextPath() + "/user/login");
			return false;
		}

		// 6. 세션이 존재하면 유효한 유저인지 확인
		UserVO authUser = (UserVO)session.getAttribute("authUser");
		if ( authUser == null ) {
			response.sendRedirect(request.getContextPath() + "/user/login");
			return false;
		}
```

```
                // 7. admin일 경우
                String role = auth.role().toString();
                if( "ADMIN".equals(role) ) {
                        // admin임을 알 수 있는 조건을 작성한다.
                        // ex) 서비스의 id가 root이면 admin이다.
                        if( "root".equals(authUser.getId()) == false ){   // admin이 아니므로
                                response.sendRedirect(request.getContextPath());
                                return false;
                        }
                }

                // 8. 접근허가, 즉 메서드를 실행하도록 함
                return true;
        }
}
```

각 코드의 설명은 주석으로 작성했습니다.

핵심은 return인데, **return true이면 컨트롤러로 요청이 진행되고, false이면 컨트롤러로 진행하지 않고 바로 응답하게 됩니다**

3)

이제 설정 파일에서 interceptor을 등록해야 합니다.

spring-servlet.xml

```
<mvc:interceptors>
        <mvc:interceptor>
                <!-- 모든 경로에 대해 @Auth 어노테이션이 실행되도록 한다. -->
                <mvc:mapping path="/**" />
                <!-- /user/login은 또 다른 interceptor가 처리한다.
                        즉 컨트롤러에 매핑되어 있는 것이 아니므로 제외시킨다. -->
                <mvc:exclude-mapping path="/user/login"/>

                <!-- defaultServlet이 처리하는 경로는 제외 -->
                <mvc:exclude-mapping path="/assets/**"/>
                <bean class="com.victolee.security.AuthInterceptor" />
        </mvc:interceptor>
</mvc:interceptors>
```

이전 글에서 작성했던 로그인을 처리하는 interceptor를 제외하고,

또 defaultServlet이 처리해야 하는 /assets 경로도 제외해야 합니다.

4)

이제 마지막으로 컨트롤러에서 접근 권한을 체크해야 합니다.

회원 정보 수정의 경우에는 세션이 존재해야 하고, 그 유저가 실제 존재하는 유저여야 하므로 @Auth로 체크하는 것이 적합합니다.

com.victolee.interceptor/InterceptorController.java


```
@Auth
@RequestMapping(value="/modify", method=RequestMethod.GET)
public String modify(HttpSession session, Model model) {
        UserVO vo = (UserVO)session.getAttribute("authUser");
        vo = userService.getUser(vo.getNo());

        model.addAttribute("vo", vo);
        if( vo == null ) {
                return "redirect:/main";
        }

        return "/user/update";
}
```

포인트는 메서드 위에 작성된 **@Auth** 어노테이션입니다.

전체적인 흐름은 다음과 같습니다.
설정 파일에서 제외 시킨 일부 요청을 제외한 모든 요청에 대해 com.victolee.security/AuthInterceptor 객체의 preHandle() 메서드가 실행될 것이며, 이 클래스 내부에서는 메서드에 @Auth 어노테이션이 있는지 확인하고, 세션 및 유저 정보가 있는지를 확인하여 접근이 가능한 유저인지 확인합니다.
만약 AuthInterceptor 객체의 preHanlder() 메서드에서 return false가 되면 접근 권한이 없는 사용자인 것입니다.

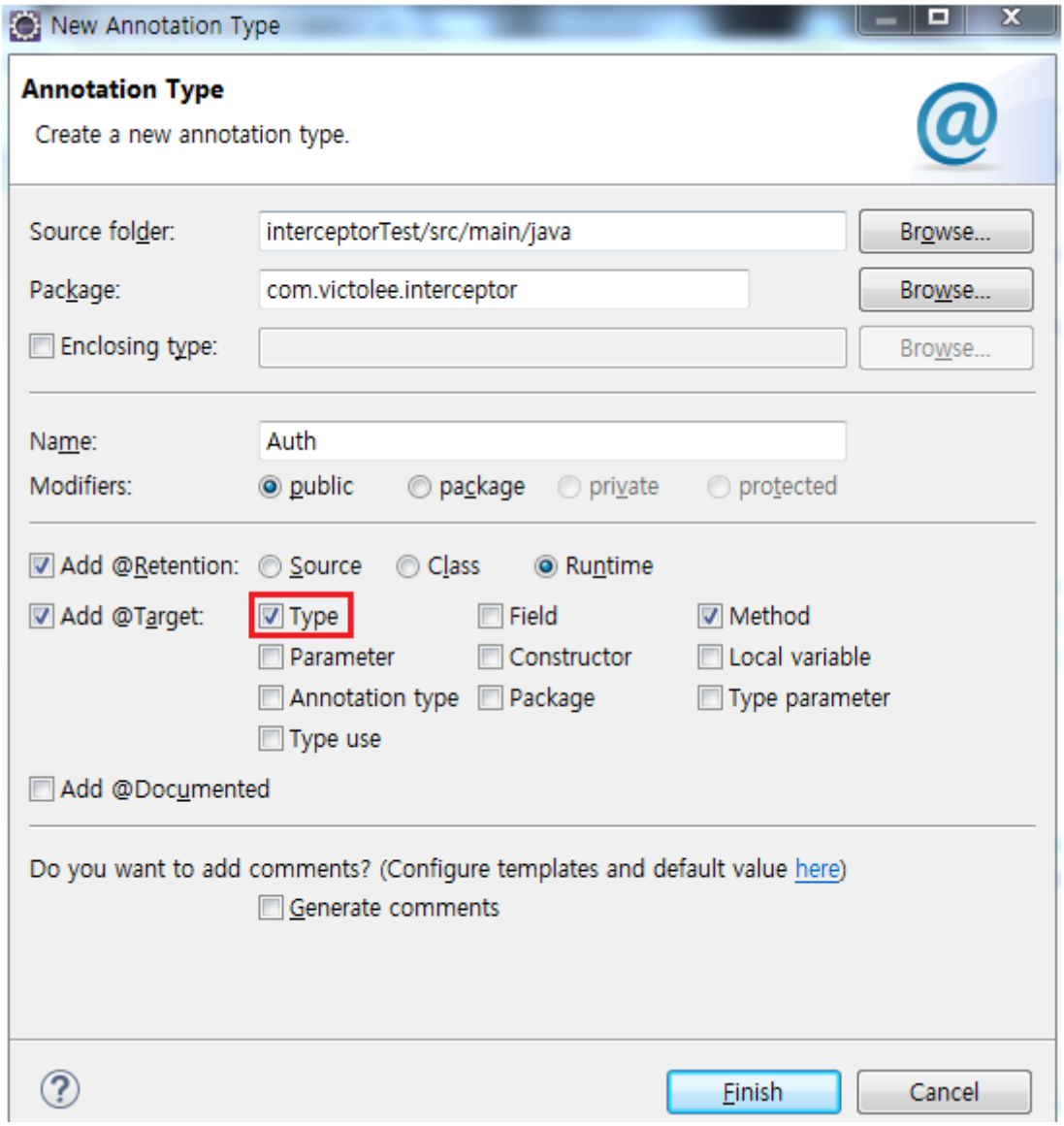
실제로 "회원정보 수정 페이지"에 대한 경로를 기억한 다음 로그인을 하지 않은 상태에서 그 경로로 접근할 경우 로그인하는 페이지로 이동할 것입니다.
로그인 페이지로 이동하는 이유는 AuthInterceptor객체의 preHandle() 메서드에서 redirect 시켰기 때문입니다.

2. 접근 권한 - 관리자 접근

관리자만 접근할 수 있는 경로라면 컨트롤러의 메서드 위에 작성할 @Auth 어노테이션을 다음과 같이 수정하면 됩니다.
@Auth(role=Role.ADMIN)

그런데 컨트롤러를 구성할 때 관리자 페이지 관련 컨트롤러들은 하나의 패키지(com.victolee.interceptor.admin)로 묶는 경우가 많습니다.
이 경우 각각의 메서드마다 @Auth(role=Role.ADMIN) 어노테이션을 추가하는 것은 코드의 중복입니다.
그래서 전체 클래스에 대해 어노테이션을 작성하면 좀 더 깔끔한 코드가 될 것입니다.

1) @Auth 어노테이션 생성



클래스, 메서드 두 범위에 대해서 어노테이션이 필요하므로 @Target에서 Type과 Method에 체크를 합니다.

2)

AuthInterceptor 클래스의 일부를 수정합니다.

com.victolee.security / **AuthInterceptor.java**

```
// 3. @Auth 받아오기
Auth auth = handlerMethod.getMethodAnnotation(Auth.class);
Auth adminRole = handlerMethod.getMethod().getDeclaringClass().getAnnotation(Auth.class);


// 7. admin일 경우
if( adminRole != null ) {
        String role = adminRole.role().toString();
        if( "ADMIN".equals(role) ) {
                if( "root".equals(authUser.getId()) == false ){
```

```
                        response.sendRedirect(request.getContextPath());
                        return false;
                }
        }
}
```

adminRole 변수를 통해 클래스 범위에 @Auth 어노테이션이 있는지 확인할 수 있습니다.

그리고 나서 컨트롤러에서 메서드가 아닌 클래스 범위에 @Auth(role=Role.ADMIN) 어노테이션을 추가하면 클래스 범위 전체에 대해 관리자만 접근 할 수 있도록 권한을 제한할 수 있습니다.

3. 세션 처리 - @AuthUser

이번에는 세션 처리를 하는 방법에 대해 알아보겠습니다.
접근 권한에서 사용했던 컨트롤러를 보시면 HttpSession 객체를 사용하여 세션을 관리하고 있습니다.
이는 기술 침투적인 부분이므로 스프링 답지 못한 코드입니다.
따라서 이 부분을 @AuthUser 어노테이션으로 대체하여 세션을 관리하도록 하겠습니다.

1) @AuthUser 어노테이션 생성

```
@Retention(RUNTIME)
@Target(PARAMETER)
public @interface AuthUser {

}
```

@AuthUser는 메서드 파라미터에 작성 할 어노테이션이므로 @Tartget(PARAMETER)입니다.
내용은 아무것도 작성하지 않아도 됩니다.

2)
다음으로 @AuthUser 어노테이션에 대하여 수행 할 클래스를 작성합니다.
이 클래스는 HandlerMethodArgumentResolver 인터페이스를 구현해야 하며, resolverArgument 메서드를 구현하면 됩니다.

com.victolee.security / AuthUserHandlerMethodArgumentResolver

```
public class AuthUserHandlerMethodArgumentResolver implements HandlerMethodArgumentResolver

        @Override
        public Object resolveArgument(
                        MethodParameter param, ModelAndViewContainer modelAndViewContainer
                        NativeWebRequest webRequest, WebDataBinderFactory binderFactory) th
```

```
                // 1. 파라미터에 @AuthUser가 붙어 있는지 , 타입이 UserVO인
                if( supportsParameter(param) == false ) {
                        // 내가 해석할 수 있는 파라미터가 아니다.
                        return WebArgumentResolver.UNRESOLVED;
                }

                // 5. 여기까지 진행이 되었다면, @AuthUser가 붙어있고 타입이 UserVO인 경우이다.
                HttpServletRequest request = webRequest.getNativeRequest(HttpServletRequest
                HttpSession session = request.getSession();
                if( session == null) {
                        return null;
                }

                return session.getAttribute("authUser");
        }


        @Override
        public boolean supportsParameter(MethodParameter parameter) {
                // 2. @AuthUser가 붙어 있는지 확인
                AuthUser authUser = parameter.getParameterAnnotation(AuthUser.class);

                // 3. @AuthUser가 안붙어 있는 경우
                if( authUser == null ) {
                        return false;
                }

                // 4. UserVO 타입이 아닌 경우
                if( parameter.getParameterType().equals(UserVO.class) == false) {
                        return false;
                }

                return true;
        }

}
```

컨트롤러의 매개변수에 @AuthUser 어노테이션이 있는지 확인하고, 매개변수의 타입이 UserVO 인지 확인한 후,
HttpSession 객체를 이용하여 세션 객체를 반환합니다.

3)
ArgumentResolver가 처리할 수 있도록 **spring-sevlet.xml** 파일에 아래의 코드를 추가합니다.

```
<mvc:annotation-driven>
        <!-- argument resolver -->
        <mvc:argument-resolvers>
```

```
                <bean class="com.victolee.security.AuthUserHandlerMethodArgumentResolve
        </mvc:argument-resolvers>
</mvc:annotation-driven>
```


(에러가 발생한다면 원래 작성되어 있던 `<mvc:annotation-driven>` 태그가 있는지 확인해주세요.)

여기까지 설정을 마치면, 요청한 URL과 매핑된 메서드의 파라미터에 @AuthUser가 있을 경우, AuthuserHandlerMethodArgumentResolver가 실행되어 세션을 반환합니다.

4. 정리

지금까지 살펴봤던 interceptor 개념과 어노테이션의 개념으로 스프링스러운 코드를 작성할 수 있게 되었습니다.
@Auth 어노테이션으로 접근 권한이 가능해졌고, @AuthUser 어노테이션으로 기술 비침투적인 세션 관리가 가능해졌습니다.

마지막으로 어노테이션을 적용했을 때와 안했을 때 비교 코드를 살펴보겠습니다.

```
// 원래 코드
@RequestMapping(value="/modify", method=RequestMethod.GET)
public String modify(HttpSession session, Model model) {
        UserVO authUser = (UserVO)session.getAttribute("authUser");
        authUser = userService.getUser(authUser.getNo());

        model.addAttribute("authUser ", authUser );
        if( authUser == null ) {
                return "redirect:/main";
        }

        return "/user/update";
}


// interceptor 적용
@Auth
@RequestMapping(value="/modify", method=RequestMethod.GET)
public String modify(@AuthUser UserVO vo, Model model) {
        UserVO authUser = userService.getUser(vo.getNo());

        model.addAttribute("authUser ", authUser );
        if( authUser == null ) {
                return "redirect:/main";
        }
```

```
        return "/user/update";
}
```


이상으로 interceptor 주제를 마치도록 하겠습니다.
정리하면 interceptor는 컨트롤러로 들어오는 요청 및 반환하는 응답을 가로채는 것이며,
로그인, 접근 권한, 세션 처리 등 다양하게 활용 수 있습니다.

'웹 프로그래밍 > Spring' 카테고리의 다른 글

victolee 구독하기

SECRE WRITE

개념많음 2019.10.23 14:39

잘읽었어요!

Delete Reply

victolee 우르르응 PROFILE 2019.10.27 18:51 신고

감사합니다^^

Delete

PREV 1 ··· 14 15 16 17 18 19 20 21 22 ··· 36 NEXT

+ Recent posts

[SpringBoot] Spring Security..

[SpringBoot] 게시판 CRUD (3)..

[SpringBoot] 게시판 CRUD (2)..

[SpringBoot] 게시판 CRUD (1)..