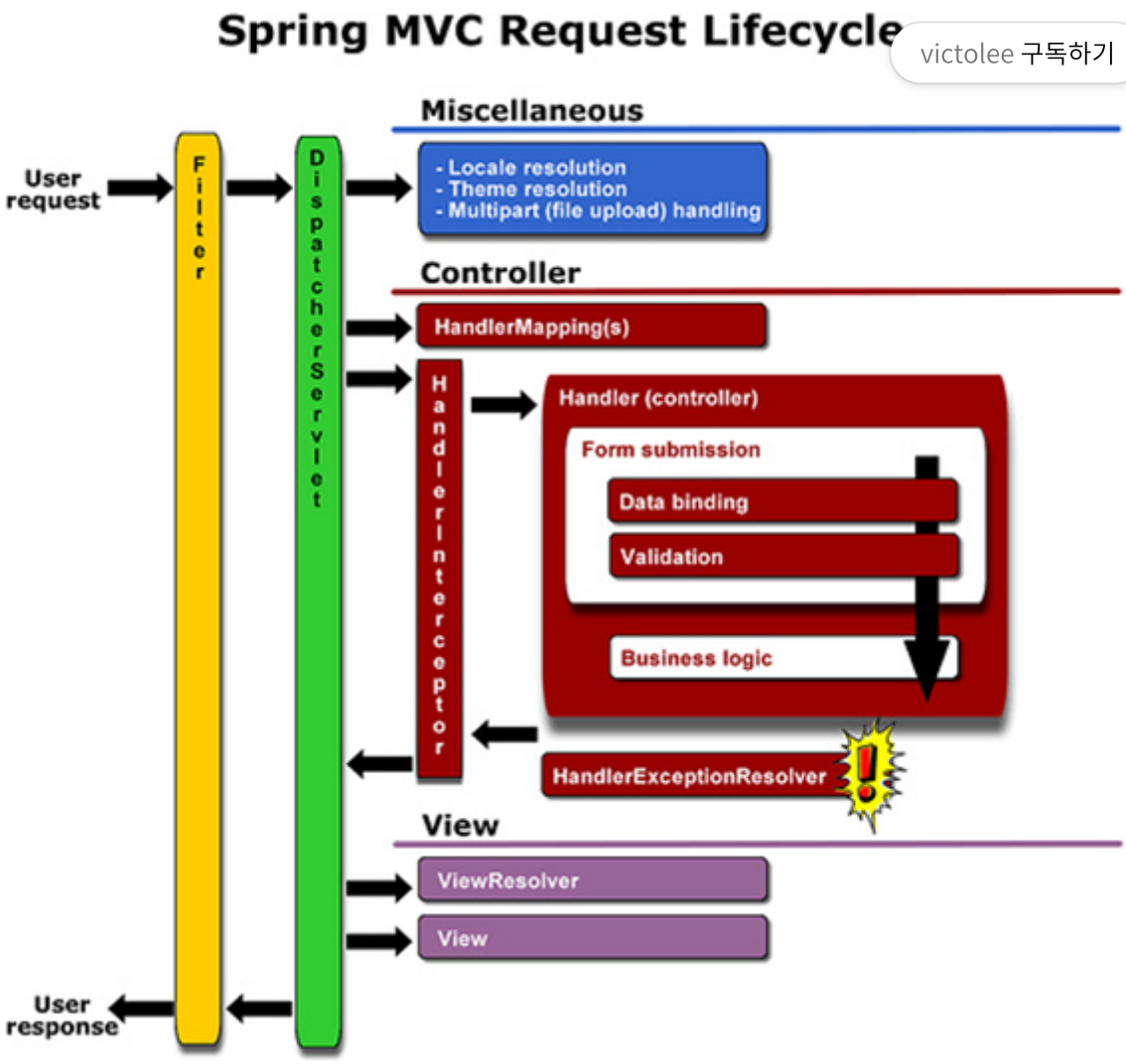
[Spring] Interceptor (1) - 개념 및 예제

2018. 3. 31. 17:55

인터셉터 (Interceptor)

Interceptor란 컨트롤러에 들어오는 요청 **HttpRequest**와 컨트롤러가 응답하는 **HttpResponse**를 가로채는 역할을 합니다.
인터셉터는 관리자만 접근할 수 있는 관리자 페이지에 접근하기 전에 관리자 인증을 하는 용도로 활용될 수 있습니다.
이번 주제에서는 컨트롤러에서 인터셉터를 활용하여 접근 권한을 해보고, 기술 침투적인 HttpSession을 제거하여 세션을 처리하려 합니다.

인터셉터는 Servlet의 앞, 뒤에서 HttpRequest, HttpResponse을 가로채는 Filter와 그 역할이 유사한데, Filter와 Interceptor는 분명히 다릅니다.



Filter와 Interceptor의 차이

1. 호출 시점

Filter는 DispatcherServlet이 실행되기 전 , Interceptor는 DispatcherServlet이 실행된 후

2. 설정 위치

Filter는 web.xml , Interceptor는 spring-servlet.xml

3. 구현 방식

Filter는 web.xml에서 설정을 하면 구현이 가능하지만, Interceptor는 설정은 물론 메서드 구현이 필요합니다.

이 글에서는 인터셉터 환경 설정 및 기본적인 사용 방법과 인터셉터를 활용하여 로그인을 처리하는 예제를 구현하도록 하겠습니다.

환경 설정

spring-servlet.xml


```
<!-- Interceptors -->
<mvc:interceptors>
	<mvc:interceptor>
		<mvc:mapping path="/**" />
		<bean class="com.victolee.interceptor.MyInterceptor" />
	</mvc:interceptor>
</mvc:interceptors>
```

모든 요청을 가로채서(Interceptor) com.victolee.interceptor 패키지에 있는 MyInterceptor 객체를 실행하겠다는 의미입니다.

테스트

Interceptor를 구현하는 방법은 2가지가 있는데, **HandlerInterceptor 인터페이스를 구현**하는 방법과 **HandlerInterceptorAdapter 클래스를 상속** 받는 방법이 있습니다.
상속을 받으면 메서드가 구현되어 있으므로 사용하기 편하기 때문에 상속 받는 방법을 주로 사용하지만, 먼저 인터페이스를 구현하는 방법을 알아보겠습니다.
HandlerInterceptor 인터페이스를 구현하기 위해서는 preHandle() , postHandle() , afterCompletion() 메서드를 구현해야 합니다.

HandlerInterceptor 인터페이스 구현

```
public class MyInterceptor implements HandlerInterceptor{
	// controller로 보내기 전에 처리하는 인터셉터
	// 반환이 false라면 controller로 요청을 안함
	// 매개변수 Object는 핸들러 정보를 의미한다. ( RequestMapping , DefaultServletHandler )
	@Override
	public boolean preHandle(
			HttpServletRequest request, HttpServletResponse response,
			Object obj) throws Exception {

		System.out.println("MyInterCeptor - preHandle");
		return false;
	}

	// controller의 handler가 끝나면 처리됨
	@Override
	public void postHandle(
			HttpServletRequest request, HttpServletResponse response,
```

```
                        Object obj, ModelAndView mav)
                        throws Exception {
        }

        // view까지 처리가 끝난 후에 처리됨
        @Override
        public void afterCompletion(
                        HttpServletRequest request, HttpServletResponse response,
                        Object obj, Exception e)
                        throws Exception {
        }
}
```

1. **preHandle()** 메서드는 컨트롤러가 호출되기 전에 실행됩니다.
   매개변수 obj는 Dispatcher의 HandlerMapping 이 찾아준 컨트롤러의 메서드를 참조할 수 있는 HandlerMethod 객체입니다.

2. postHandle() 메서드는 컨트롤러가 실행된 후에 호출됩니다.

3. afterComplete() 은 뷰에서 최종 결과가 생성하는 일을 포함한 모든 일이 완료 되었을 때 실행됩니다.

각 **메서드의 반환값이 true이면 핸들러의 다음 체인이 실행되지만 false이면 중단하고 남은 인터셉터와 컨트롤러가 실행되지 않습니다.**

테스트를 위해서 적당한 컨트롤러를 생성한 후에, @RequestMapping을 한 메서드를 작성하여, 올바른 URL로 요청해보세요.
그러면 콘솔 창에서 메시지를 확인할 수 있습니다.

HandlerInterceptorAdapter 클래스를 상속 받아 구현

```
public class MyInterceptor extends HandlerInterceptorAdapter {
        @Override
        public boolean preHandle(
                        HttpServletRequest request, HttpServletResponse response,
                        Object obj) throws Exception {

                System.out.println("MyInterCeptor - preHandle");
                return true;
        }
}
```

HandlerInterceptorAdapter 클래스를 상속 받으면, postHandle() , afterCompletion() 메서드가 구현되어 있으므로 필요한 메서드만 오버라이딩 하면 됩니다.

예제 - Interceptor를 이용하여 로그인 처리

Interceptor를 이용하여 로그인을 처리할 수도 있습니다.

```
public class AuthLoginInterceptor extends HandlerInterceptorAdapter{
        @Autowired
        private UserService userService;

        @Override
        public boolean preHandle(HttpServletRequest request, HttpServletResponse response,
                                                                Object handler) throws Exc

                String email = request.getParameter("email");
                String pwd = request.getParameter("pwd");

                UserVO vo = new UserVO();
                vo.setEmail(email);
                vo.setPwd(pwd);
                userService.getUser(vo);

                return false;
        }
}
```

HandlerInterceptorAdapter 클래스를 상속받아 preHandle() 메서드에 로그인을 수행하는 로직을 작성하여 클래스로 만들어 둡니다. (예외 처리는 생략)
클라이언트가 로그인 버튼을 눌렀을 때 email과 pwd를 전달한다고 가정하면, getParameter() 메서드로 데이터를 가져올 수 있고 UserVO를 Service 계층에 전달하여 해당 유저가 존재하는지 확인합니다.

로그인 버튼을 누르면 AuthLoginInterceptor 객체의 preHandle() 메서드가 실행될 것입니다.
preHandler() 메서드는 컨트롤러가 호출되기 전에 실행되는 메서드이기 때문이죠.

로그인을 처리하기 위한 경로를 /user/login라고 할 때, 이 경로에 대해서 위의 인터셉터 로직이 수행되려면 설정이 필요합니다.

spring-servlet.xml

```
<mvc:interceptor>
        <mvc:mapping path="/user/login" />
        <bean class="com.cafe24.security.AuthLoginInterceptor" />
</mvc:interceptor>
```

처음에 작성했던 spring-servlet.xml의 mapping은 모든 경로 범위였으므로 기존 코드를 삭제하고 위의 코드로 수정합니다.

즉 /user/login으로 요청이 오면 AuthLoginInterceptor 객체의 preHandle() 메서드가 실행될 것입니다.
그리고 이렇게 설정을 하면 **컨트롤러에서 /user/login 요청을 처리하는 메서드가 있을 필요가 없습니다.**

뷰 페이지에서는 로그인 버튼을 눌렀을 때 요청이 /user/login으로 요청이 갈 수 있도록 form을 다음과 같이 작성합니다.

```
<form method="post" action="${pageContext.servletContext.contextPath }/user/login">
```

그러면 이제 로그인 버튼을 눌렀을 때 <form> 태그에서 작성한 action 경로에 따라 /user/login으로 요청이 갈 것이고,
이 요청을 처리하기 위해 Controller가 실행되기 전 interceptor가 가로채서 미리 작성해두었던 AuthLoginInterceptor 객체의 preHandle() 메서드가 실행되어 로그인 과정을 처리할 것입니다.

이상으로 Intercpetor가 무엇이고, Filter와의 차이점 그리고 예제에 대해 살펴보았고, 로그인 부분을 Interceptor로 처리할 수도 있습니다.
그러면 컨트롤러에서 /user/login을 처리하기 위한 메서드가 필요 없게 될 것입니다.

이렇게 로그인을 처리하는 것도 좋은 방식이지만 다음 글에서 말씀드릴 세션에서 Interceptor를 사용하면 기술 비침투적으로 세션을 구현할 수 있습니다.

'웹 프로그래밍 > Spring' 카테고리의 다른 글

[Spring] form Validation (0) 2018.04.01
[Spring] AOP(Aspect Oriented Programming) - DAO 실행 시간 측정 예제 (0) 2018.04.01
[Spring] Interceptor (2) - 어노테이션 작성 및 접근 권한, 세션 처리 (2) 2018.03.31
[Spring] Interceptor (1) - 개념 및 예제 (0) 2018.03.31
[Spring] (멀티) 파일 업로드 (2) 2018.03.31
[Spring] 로그 남기기 (Logback) (2) 2018.03.31

[Spring] Ajax - JSON 응답하기 (MessageConverter) (6) 2018.03.31

[Spring] 방명록 애플리케이션 (10) - Mybatis 적용 (0)

NAME

PASSWORD

HOMEPAGE

http://

SECRE WRITE

PREV 1 ··· 121 122 123 124 125 126 127 128 129 ··· 265 NEXT

+ Recent posts

[SpringBoot] Spring Security..

[SpringBoot] 게시판 CRUD (3)..

[SpringBoot] 게시판 CRUD (2)..

[SpringBoot] 게시판 CRUD (1)..