Spring Framework
- 스프링 MVC를 이용한 웹 요청 처리_3

CONTENTS

- HandlerInterceptor
- 예외처리

HandlerInterceptor를 통한 요청 가로채기

- **핸들러 인터셉터는 DispatcherServlet 이 컨트롤러를 호출하기 전과 후에 요청과 응답을 참조하거나 가공할 수 있는 일종의 필터**
- 스프링이 기본적으로 제공하는 HandlerMapping은 HandlerInterceptor를 이용해서 컨트롤러가 요청을 처리하기 전과 처리한 후에 알맞은 기능을 수행 할 수 있도록 하고 있다.
- 조건에 따라 컨트롤러에 요청을 전달하지 않고 싶거나 컨트롤러가 요청을 처리한 후에 ModelAndView 객체를 조작하고 싶은 경우에 HandlerInterceptor를 사용하면 됨.
- EX) 로그인 여부 체크

HandlerInterceptor를 통한 요청 가로채기

- **HandlerInterceptor 인터페이스의 메서드.**
  - boolean **preHandle**(**HttpServletRequest** request, **HttpServletResponse** response, **Object** handler)
  - void **postHandle**(**HttpServletRequest** request, **HttpServletResponse** response, **Object** handler, **ModelAndView** modelAndView)
  - void **afterCompletion**(**HttpServletRequest** request, **HttpServletResponse** response, **Object** handler, **Exception** ex)

HandlerInterceptor를 통한 요청 가로채기

- **preHandle()**

  컨트롤러가 호출되기 전에 실행된다. handler 파라메터는 핸들러 매핑이 찾아 준 컨트롤러 빈 오브젝트다. 컨트롤러 실행 이전에 처리해야 할 작업이 있다거나, 요청정보를 가공하거나 추가하는 경우에 사용할 수 있다. 또는 요청에 요청에 대한 로그를 남기기 위해 사용하기도 한다.

  리턴 값이 true 이면 핸들러 실행 체인의 다음 단계로 진행되지만, false 라면 작업을 중단하고 리턴하므로 컨트롤러와 남은 인터셉터들은 실행되지 않는다.

- **postHandle()**

  컨트롤러를 실행하고 난 후에 호출된다. 이 메소드에는 컨트롤러가 돌려준 ModelAndView 타입의 정보가 제공 되서 컨트롤러 작업 결과를 참조하거나 조작할 수 있다.

- **afterCompletion()**

  이름 그대로 모든 뷰에서 최종 결과를 생성하는 일을 포함한 모든 작업이 모두 완료된 후에 실행된다. 요청처리 중에 사용한 리소스를 반환해주기에 적당한 메소드다.

- 핸들러 인터셉터는 하나 이상을 등록할 수 있다. preHandle() 은 인터셉터가 등록된 순서대로 실행된다. 반면에 postHandle() 과 afterCompletion() 은 preHandle() 이 실행된 순서와 반대로 실행된다.

HandlerInterceptor를 통한 요청 가로채기

- **HandlerInterceptor 인터페이스의 구현**
  - **핸들러 인터셉터는 HandlerInterceptor 인터페이스를 구현해서 만든다.** 이 인터페이스를 구현 할 경우 사용하지 않는 메서드도 구현 해주어야 한다.
  - 이러한 불편함을 줄여주기 위해 **HandlerInterceptorAdaptor** 클래스를 제공.
  - HandlerInterceptor 인터페이스를 구현해야 하는 클래스는 **HandlerInterceptorAdaptor** 클래스를 상속 받은 뒤 필요한 메서드만 오버라이딩 하여 사용.

HandlerInterceptor를 통한 요청 가로채기

- **HandlerMapping에 HandlerInterceptor 설정 하기**
  - HandlerInterceptor를 구현 한 뒤에는 **<mvc:interceptors>** 의 interceptors 프로퍼티를 사용해서 **HandlerInterceptorAdaptor** 를 등록해 주면 된다.

```
<!-- 인터셉터 이용한 로그인 체크 -->

<mvc:interceptors>
        <mvc:interceptor>
                <mvc:mapping path="/mypage/**" />
                <beans:bean
class="com.bitcamp.memberboard.interceptor.AuthCheckInterceptor" />
        </mvc:interceptor>
</mvc:interceptors>
```

HandlerInterceptor를 통한 요청 가로채기

- **HandlerInterceptor의 실행 순서**
  - HandlerMapping에는 한 개 이상의 **HandlerInterceptor를 등록 할 수 있음.**
    1. 컨트롤러 실행 전 : 등록된 순서대로 preHandle() 실행
    2. 컨트롤러 실행 후 : 등록된 순서 반대로 postHand() 실행
    3. 처리 완료 후(뷰 생성 후) : 등록된 순서의 반대로 afterCompletion() 실행

```
<mvc:interceptors>
    <mvc:interceptor>
        <mvc:mapping path="/mypage1/**" />
        <beans:bean
        class="com.bitcamp.open.interceptor.AuthCheckInterceptor1" />
    </mvc:interceptor>
    <mvc:interceptor>
        <mvc:mapping path="/mypage2/**" />
        <beans:bean
        class="com.bitcamp.open.interceptor.AuthCheckInterceptor2" />
    </mvc:interceptor>
    <mvc:interceptor>
        <mvc:mapping path="/mypage3/**" />
        <beans:bean
        class="com.bitcamp.open.interceptor.AuthCheckInterceptor3" />
    </mvc:interceptor>
</mvc:interceptors>
```

HandlerInterceptor를 통한 요청 가로채기

4. preHandle() 실행:
    a. Interceptor1.preHandle()
    b. Interceptor2.preHandle()
    c. Interceptor3.preHandle()
5. 컨트롤러 handleRequest() 메서드 실행
6. postHand() 실행
    a. Interceptor3.postHand()
    b. Interceptor2.postHand()
    c. Interceptor1.postHand()
7. 뷰 객체의 render() 메서드 실행에서 응답 결과 생성
8. afterCompletion() 실행
    a. Interceptor3.afterCompletion()
    b. Interceptor2.afterCompletion()
    c. Interceptor1.afterCompletion()

com.bitcamp.open.interceptor.AuthCheckInterceptor

```
package com.bitcamp.open.interceptor;

public class AuthCheckInterceptor extends HandlerInterceptorAdapter {

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

        HttpSession session = request.getSession(false);
        if (session != null) {
                Object authInfo = session.getAttribute("loginInfo");
                if (authInfo != null) {
                        return true;
                }
        }
        response.sendRedirect(request.getContextPath()+"member/login");
        return false;
    }
}
```

스프링 설정

…

```
<!-- 인터셉터 이용한 로그인 체크 -->
    <!-- 여러 핸들러인터셉터를 생정 -->
    <mvc:interceptors>
        <!-- 한개의 핸들러인터셉터를 생성 -->
        <mvc:interceptor>
        <!-- 핸들러인터셉터를 적용할 경로 설정 -->
        <mvc:mapping path="/mypage/**" />
        <!-- 경로중 일부 경로를 제외하고 싶을 때  -->
        <!-- <mvc:exclude-mapping path="/mypage/help/**"/> -->
        <beans:bean
        class="com.bitcamp.open.interceptor.AuthCheckInterceptor" />
    </mvc:interceptor>
</mvc:interceptors>
```

…

예외처리

- **컨트롤러 클래스의 @RequestMapping이 선언된 메소드는 모든 타입의 예외를 발생 시킬 수 있음.**
- 서블릿 컨테이너가 생성한 에러 페이지가 아니라 예외 타입에 따라 스프링 MVC와 연동된 뷰를 이용해서 에러 페이지를 출력하고자 한다면, 스프링이 제공하는 HandlerExceptionResolver 인터페이스를 사용.

  - **AnnotationMethodHandlerExceptionResolver**
    @ExceptionHandler 어노테이션이 적용된 메소드를 찾아 예외처리
  - **DefaultHandlerExceptionResolver**
    스프링에서 내부적으로 발생하는 주요 예외를 처리해주는 표준 예외처리 로직을 담고 있다.
  - **SimpleMappingExceptionResolver**
    예외 타입 별로 뷰 이름을 지정할 때 사용

예외처리

- **예외처리 방법**
  - **HandlerExceptionResolver**

    HandlerExceptionResolver 인터페이스의 resolveException() 메서드는 발생된 예외 객체를 파라미터로 전달받음
  - **@ExceptionHandler**

    @Controller 어노테이션이 적용된 클래스에 @ExceptionHandler 어노테이션이 적용된 메서드 구현
  - **설정 파일을 이용한 선언적 예외 처리**

    SimpleMappingExceptionResolver클래스를 이용하기 위해 설정 파일에 설정

예외처리

- **@ExceptionHandler 어노테이션을 이용한 처리**
  - AnnotationMethodHandlerExceptionResolver 클래스는 **@Controller 어노테이션이 적용된 클래스에서 @ExceptionHandler 어노테이션이 적용된 메서드를 이용해서 예외 처리한다.**

    @ModelAttribute 에서 @ExceptionHandler 에서 지정한 예외가 발생하면 @ExceptionHandler가 적용된 메서드를 이용해서 뷰를 지정.

```
import org.springframework.web.bind.annotation.ExceptionHandler;
…
        @ExceptionHandler(NullPointerException.class)
        public String handleNullPointerException(NullPointerException ex) {
                return "error/nullException";
        }
…
```

예외처리

views/error/nullException.jsp

```
<%@ page language="java" contentType="text/html; charset=utf-8"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>예외 발생</title>
</head>
<body>
요청을 처리하는 과정에서 예외(null)가 발생하였습니다.
</body>
</html>
```