Interceptor

#01. 인터셉터의 이해

- 웹 브라우저가 보내는 컨트롤러의 실행 요청을 중간에 가로채서 어떠한 일을 수행하게 하는 기능
- 모든 웹 페이지가 공통적으로 동일하게 실행해야 하는 기능들을 구현할 수 있다.

1) Spring Interceptor의 동작 시점

1. 컨트롤러 메서드가 실행되기 직전
2. View로 forward 되기 직전
3. 컨트롤러 메서드가 실행 완료된 직후

2) 활용예시

컨트롤러가 실행되기 직전

- 모든 웹 페이지의 실행 정보와 접속한 클라이언트의 정보를 로그로 기록한다.
- 로그인 여부를 판별하여 회원 전용 페이지에 접근할 수 있는가를 판단한다.

컨트롤러가 실행된 직후

- 페이지 실행 시간을 측정하여 이 페이지에 얼마나 머물렀는가를 로그로 남겨 컨텐츠의 인기도를 측정할 수 있는 데이터를 남긴다.

#02. 프로젝트 구성

1. 패키지 정보

`kr.hossam.interceptor`

항목	설정 내용
GroupId	`kr.hossam`
ArtifactId	`interceptor`

2. 의존성 설정

프로젝트 생성 과정에서 `dependencies` 항목에 대해 아래의 항목을 선택

항목 이름	구분
Spring Boot DevTools	기존 사용
Spring Boot Acturator Ops	기존 사용
Spring Web	기존 사용
Thymeleaf Template Engines	기존 사용

항목 이름	구분
Lombok	기존 사용

3. 추가 설정

1) logback 설정파일 추가

`/src/main/resources/logback-spring.xml` 파일을 추가

(이전 수업에서 생성한 파일을 재사용)

2) UserAgent 라이브러리 설정

`build.gradle`에 의존성 설정 추가

```
implementation 'com.github.ua-parser:uap-java:1.6.1'
```

> 여기까지의 과정이 앞으로 모든 예제의 기본 설정 입니다.

4. 프로젝트 구조

```
└main
   ├─java
   │  └─kr
   │      └─hossam
   │          └─interceptor
   │              │  InterceptorApplication.java    (기본파일, 수정안함)
   │              │  MyWebConfig.java               (신규)
   │              │
   │              ├─controllers
   │              │      MyController.java          (신규)
   │              │
   │              ├─helpers
   │              │      UtilHelper.java            (기존파일 재사용 + 내용추가)
   │              │
   │              └─interceptors
   │                     MyInterceptor.java        (신규)
   │
   └─resources
       │  application.properties                   (기본파일, 수정안함)
       │  logback-spring.xml                       (기존파일 재사용)
       │
       ├─static
       │      favicon.ico                          (신규)
       │      robots.txt                           (신규)
       │
       └─templates
              myview.html                          (신규)
```

#03. Interceptor 구성하기

1) Interceptor를 위한 패키지 추가

`interceptor`를 위한 패키지를 추가한다.

> `kr.hossam.imterceptor.interceptors`

일반적으로 controller, models, helpers 등 각 기능별로 패키지를 생성한다.

패키지는 프로그램 단위 안에 포함한다.

패키지가 `kr.hossam.helloworld`인 경우

패키지	기능
`kr.hossam.helloworld.controllers`	컨트롤러 클래스
`kr.hossam.helloworld.helpers`	헬퍼 클래스
`kr.hossam.helloworld.models`	데이터 구조를 정의하는 DTO 클래스
`kr.hossam.helloworld.services`	비즈니스로직을 담는 인터페이스
`kr.hossam.helloworld.services.impl`	비즈니스로직에 대한 구현체
`kr.hossam.helloworld.interceptors`	인터셉터 클래스
`kr.hossam.helloworld.schedulers`	스케쥴러 클래스

> 스케쥴러는 이후 단원에서 다룹니다.

2) Interceptor 클래스 추가

`HandlerInterceptor` 인터페이스를 상속받는 임의의 클래스를 정의하고 interface가 선언해 놓은 메서드를 재정의 한다.

클래스 상단에 `@Component`와 `@Slf4j`를 추가해야 한다.

MyInterceptor.java의 원형 + 기능 설명

```
package study.spring.interceptor;

import org.springframework.lang.Nullable;
import org.springframework.stereotype.Component;
import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;

import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.extern.slf4j.Slf4j;
```

```
@Slf4j
@Component
public class MyInterceptor implements HandlerInterceptor {
    /**
     * Controller 실행 전에 수행되는 메서드
     * 클라이언트(웹브라우저)의 요청을 컨트롤러에 전달 하기 전에 호출된다.
     * return 값으로 boolean 값을 전달하는데 false 인 경우
     * controller를 실행 시키지 않고 요청을 종료한다.
     * 보통 이곳에서 각종 체크작업과 로그를 기록하는 작업을 진행한다.
     */
    @Override
    public boolean preHandle(HttpServletRequest request,
HttpServletResponse response, Object handler)
            throws Exception {
        log.debug("MyInterceptor.preHandle 실행됨");
        return HandlerInterceptor.super.preHandle(request, response,
handler);
    }

    /**
     * view 단으로 forward 되기 전에 수행.
     * 컨트롤러 로직이 실행된 이후 호출된다.
     * 컨트롤러 단에서 에러 발생 시 해당 메서드는 수행되지 않는다.
     * request로 넘어온 데이터 가공 시 많이 사용된다.
     */
    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse
response, Object handler,
            @Nullable ModelAndView modelAndView) throws Exception {
        log.debug("MyInterceptor.postHandle 실행됨");
        HandlerInterceptor.super.postHandle(request, response, handler,
modelAndView);
    }

    /**
     * 컨트롤러 종료 후 view가 정상적으로 랜더링 된 후 제일 마지막에 실행이 되는 메서드.
     * (잘 사용 안함)
     */
    @Override
    public void afterCompletion(HttpServletRequest request,
HttpServletResponse response, Object handler,
            @Nullable Exception ex) throws Exception {
        log.debug("MyInterceptor.afterCompletion 실행됨");
        HandlerInterceptor.super.afterCompletion(request, response,
handler, ex);
    }
}
```

2) Spring Boot의 Web 설정 클래스 추가

> SpringBoot 이전 SpringMVC에서는 설정 사항을 XML포멧으로 사용했지만 SpringBoot부터는 Java 클래스 형태로 설정내용을 구현해야 함

프로젝트 패키지 최상단에 `WebMvcConfigurer` 인터페이스를 상속받는 임의의 클래스를 추가한다.

클래스 상단에 `@Configuration`을 추가해야 한다.

MyWebConfig.java

```
package study.spring.interceptor;

import org.springframework.context.annotation.Configuration;
import
org.springframework.web.servlet.config.annotation.InterceptorRegistration;
import
org.springframework.web.servlet.config.annotation.InterceptorRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class MyWebConfig implements WebMvcConfigurer {
    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        // 직접 정의한 MyInterceptor를 Spring에 등록
        InterceptorRegistration ir = registry.addInterceptor(new
MyInterceptor());
        // 해당 경로는 인터셉터가 가로채지 않는다.
        ir.excludePathPatterns("/hello", "/world");
    }
}
```