관리자 시스템 Page 이동. 검색 정보 기록. (취약점 점검시 발견됨)

Interceptor.

업무 시스템에 Interceptor가 구현되어 있어 Interceptor을 이용해서 이동 Page - 검색 정보을 DataBase에 기록하는 것을 구현했다.

-로그인 정보 (IP, ID) 등

구현하면서 궁금했던 점

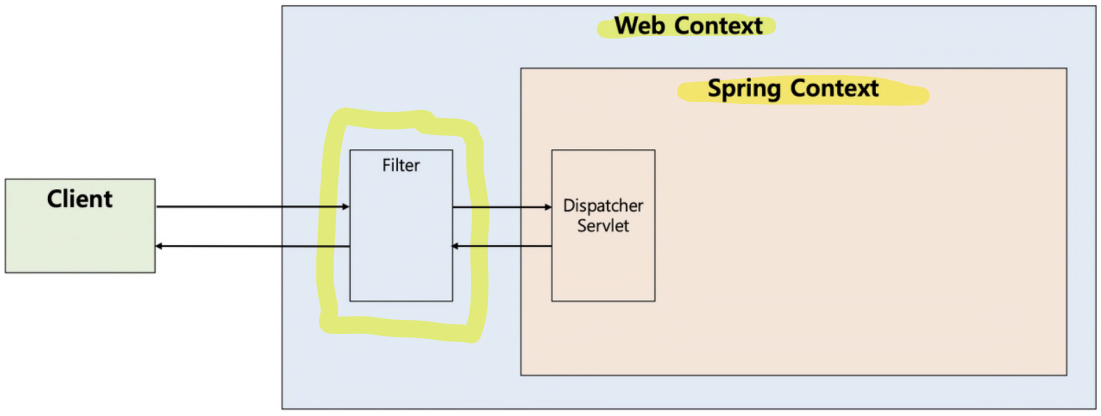
1. Interceptor와 Filter의 차이?
2. WebContentInterceptor란 무엇인가.

Filter 란

1. 필터(Filter)

[필터(Filter)란?]

필터(Filter)는 J2EE표준 스펙 기능으로 디스패처 서블릿(Dispatcher Servlet)에 요청이 전달되기 전/후에 url 패턴에 맞는 모든 요청에 대해 부가작업을 처리할 수 있는 기능을 제공한다. 즉, 스프링 컨테이너가 아닌 톰캣과 같은 웹컨테이너에 의해 관리가 되므로 디스패처 서블릿으로 가기 전에 요청을 처리하는 것이다.
이러한 과정을 그림으로 표현하면 다음과 같다.



[필터(Filter)의 메소드]

필터를 추가하기 위해서는 javax.servlet의 Filter 인터페이스를 구현(implements)해야 하며 이는 다음의 3가지 메소드를 가지고 있다.

- init 메소드
- doFilter 메소드
- destroy 메소드

```
public interface Filter {

    public default void init(FilterConfig filterConfig) throws ServletException {}

    public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException;

    public default void destroy() {}
}
```

init 메소드

init 메소드는 필터 객체를 초기화하고 서비스에 추가하기 위한 메소드이다. 웹 컨테이너가 1회 init 메소드를 호출하여 필터 객체를 초기화하면 이후의 요청들은 doFilter를 통해 전/후에 처리된다.

doFilter 메소드

doFilter 메소드는 url-pattern에 맞는 모든 HTTP 요청이 디스패처 서블릿으로 전달되기 전/후에 웹 컨테이너에 의해 실행되는 메소드이다. doFilter의 파라미터로는 FilterChain이 있는데, FilterChain의 doFilter 통해 다음 대상으로 요청을 전달하게 된다.

destroy 메소드

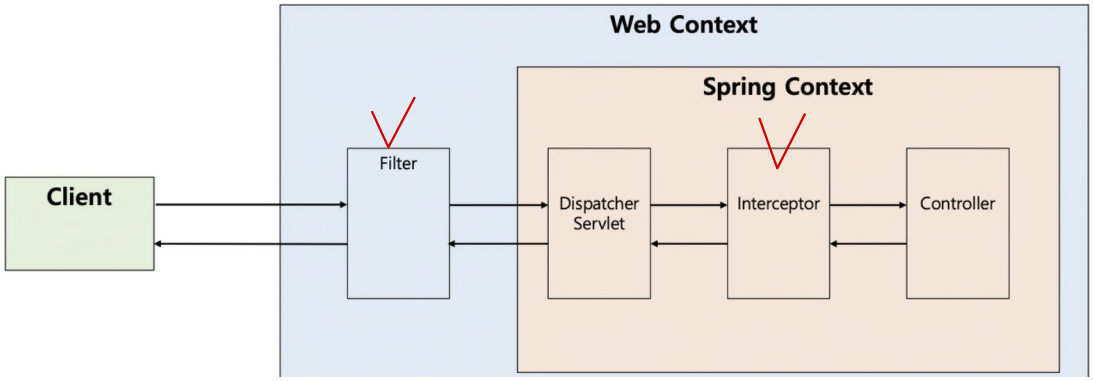
destroy 메소드는 필터 객체를 서비스에서 제거하고 사용하는 자원을 반환하기 위한 메소드이다. 이는 웹 컨테이너에 의해 1번 호출되며 이후에는 이제 doFilter에 의해 처리되지 않는다.

Interceptor란

2. 인터셉터(Interceptor)

[인터셉터(Interceptor)란?]

인터셉터(Interceptor)는 J2EE 표준 스펙인 필터(Filter)와 달리 Spring이 제공하는 기술로써, 디스패처 서블릿(Dispatcher Servlet)이 컨트롤러를 호출하기 전과 후에 요청과 응답을 참조하거나 가공할 수 있는 기능을 제공한다. 즉, 웹 컨테이너에서 동작하는 필터와 달리 인터셉터는 스프링 컨텍스트에서 동작을 하는 것이다.
디스패처 서블릿은 핸들러 매핑을 통해 적절한 컨트롤러를 찾도록 요청하는데, 그 결과로 실행 체인(HandlerExecutionChain)을 돌려준다. 그래서 이 실행 체인은 1개 이상의 인터셉터가 등록되어 있다면 순차적으로 인터셉터들을 거쳐 컨트롤러가 실행되도록 하고, 인터셉터가 없다면 바로 컨트롤러를 실행한다.
인터셉터는 스프링 컨테이너 내에서 동작하므로 필터를 거쳐 프론트 컨트롤러인 디스패처 서블릿이 요청을 받은 이후에 동작하게 되는데, 이러한 과정을 그림으로 표현하면 다음과 같다.



[인터셉터(Interceptor)의 메소드]

인터셉터를 추가하기 위해서는 org.springframework.web.servlet의 HandlerInterceptor 인터페이스를 구현(implements)해야 하며, 이는 다음의 3가지 메소드를 가지고 있다.

- preHandle 메소드
- postHandle 메소드
- afterCompletion 메소드

```
public interface HandlerInterceptor {

    default boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)
        throws Exception {

        return true;
    }

    default void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler,
        @Nullable ModelAndView modelAndView) throws Exception {
    }

    default void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler,
        @Nullable Exception ex) throws Exception {
    }
}
```

preHandle 메소드

preHandle 메소드는 컨트롤러가 호출되기 전에 실행된다. 그렇기 때문에 컨트롤러 이전에 처리해야 하는 전처리 작업이나 요청 정보를 가공하거나 추가하는 경우에 사용할 수 있다.
preHandle의 3번째 파라미터인 handler 파라미터는 핸들러 매핑이 찾아준 컨트롤러 빈에 매핑되는 HandlerMethod라는 새로운 타입의 객체로써, @RequestMapping이 붙은 메소드의 정보를 추상화한 객체이다.
또한 preHandle의 반환 타입은 boolean인데 반환값이 true이면 다음 단계로 진행이 되지만, false라면 작업을 중단하여 이후의 작업(다음 인터셉터 또는 컨트롤러)은 진행되지 않는다.

postHandle 메소드

postHandle 메소드는 컨트롤러를 호출된 후에 실행된다. 그렇기 때문에 컨트롤러 이후에 처리해야 하는 후처리 작업이 있을 때 사용할 수 있다. 이 메소드에는 컨트롤러가 반환하는 ModelAndView 타입의 정보가 제공되는데, 최근에는 Json 형태로 데이터를 제공하는 RestAPI 기반의 컨트롤러(@RestController)를 만들면서 자주 사용되지는 않는다.

[필터(Filter)의 용도 예시]

- 보안 관련 공통 작업
- 모든 요청에 대한 로깅 또는 감사
- 이미지/데이터 압축 및 문자열 인코딩

필터에서는 스프링과 무관하게 전역적으로 처리해야 하는 작업들을 처리할 수 있다.
대표적인 예시로 보안과 관련된 공통 작업이 있다. 필터는 인터셉터보다 앞단에서 동작하기 때문에 전역적으로 해야 하는 보안 검사(XSS 방어 등)를 하여 올바른 요청이 아닐 경우 차단을 할 수 있다. 그러면 스프링 컨테이너까지 요청이 전달되지 못하고 차단되므로 안정성을 더욱 높일 수 있다.
또한 필터는 이미지나 데이터의 압축이나 문자열 인코딩과 같이 웹 애플리케이션에 전반적으로 사용되는 기능을 구현하기에 적당하다. Filter는 다음 체인으로 넘기는 ServletRequest/ServletResponse 객체를 조작할 수 있다는 점에서 Interceptor보다 훨씬 강력한 기술이다.

[인터셉터(Interceptor)의 용도 예시]

- 인증/인가 등과 같은 공통 작업
- API 호출에 대한 로깅 또는 감사
- Controller로 넘겨주는 정보(데이터)의 가공

인터셉터에서는 클라이언트의 요청과 관련되어 전역적으로 처리해야 하는 작업들을 처리할 수 있다.
대표적으로 인증이나 인가와 같이 클라이언트 요청과 관련된 작업 등이 있다. 이러한 작업들은 컨트롤러로 넘어가기 전에 검사해야 하므로 인터셉터가 처리하기에 적합하다.
또한 인터셉터는 필터와 다르게 HttpServletRequest나 HttpServletResponse 등과 같은 객체를 제공받으므로 객체 자체를 조작할 수는 없다. 대신 해당 객체가 내부적으로 갖는 값은 조작할 수 있으므로 컨트롤러로 넘겨주기 위한 정보를 가공하기에 용이하다. 예를 들어 JWT 토큰 정보를 파싱해서 컨트롤러에게 사용자의 정보를 제공하도록 가공할 수 있는 것이다.
그 외에도 우리는 다양한 목적으로 API 호출에 대한 정보들을 기록해야 할 수 있다. 이러한 경우에 HttpServletRequest나 HttpServletResponse를 제공해주는 인터셉터는 클라이언트의 IP나 요청 정보들을 포함해 기록하기에 용이하다.

Reference : https://mangkyu.tistory.com/173

WebContentInterceptor ?

org.springframework.web.servlet.mvc

Class WebContentInterceptor

java.lang.Object
  org.springframework.context.support.ApplicationObjectSupport
    org.springframework.web.context.support.WebApplicationObjectSupport
      org.springframework.web.servlet.support.WebContentGenerator
        org.springframework.web.servlet.mvc.WebContentInterceptor

All Implemented Interfaces:

`Aware`, `ApplicationContextAware`, `ServletContextAware`, `HandlerInterceptor`

```
public class WebContentInterceptor
extends WebContentGenerator
implements HandlerInterceptor
```

Handler interceptor that checks the request for supported methods and a required session and prepares the response by applying the configured cache settings.

Cache settings may be configured for specific URLs via path pattern with `addCacheMapping(CacheControl, String...)` and `setCacheMappings(Properties)`, along with a fallback on default settings for all URLs via `WebContentGenerator.setCacheControl(CacheControl)`.

Pattern matching can be done with `PathMatcher` or with parsed `PathPattern`s. The syntax is largely the same with the latter being more tailored for web usage and more efficient. The choice depends on the presence of a `resolved` `String` lookupPath or a `ServletRequestPathUtils.parseAndCache(javax.servlet.http.HttpServletRequest)` parsed} `RequestPath` which in turn depends on the `HandlerMapping` that matched the current request.

All the settings supported by this interceptor can also be set on `AbstractController`. This interceptor is mainly intended for applying checks and preparations to a set of controllers mapped by a HandlerMapping.

기본적인 browser cache 설정 방안
1. static resource에 대해서는 <mvc:resources> element의 cache-period 값 설정
2. dynamic resource에 대해서는 Spring에서 제공하는 WebContentInterceptor의 cacheSeconds 값 설정

* <mvc:resources> 설정
Spring Servlet Context 파일에 설정

설정 예

```
<mvc:resources mapping="/js/**" location="/js/" cache-period="86400"/>
<mvc:resources mapping="/style/**" location="/style/" cache-period="86400"/>
<mvc:resources mapping="/image/**" location="/image/" cache-period="31556926"/>
```

cache-period

- Specifies the cache period for the resources served by this resource handler, in seconds.
- The default is to not send any cache headers but rather to rely on last-modified timestamps only.
- Set this to 0 in order to send cache headers that prevent caching, or to a positive number of seconds in order to send cache headers with the given max-age value.

* WebContentInterceptor 설정
Spring Servlet Context 파일에 설정

설정 예 1) /api 아래에 대해서 웹 브라우저 cache 방지

```
<mvc:interceptors>
    <mvc:interceptor>
        <mvc:mapping path="/api/**/*"/>
        <bean class="org.springframework.web.servlet.mvc.WebContentInterceptor">
            <property name="cacheSeconds" value="0"/>
        </bean>
    </mvc:interceptor>
</mvc-interceptors>
```

설정 예 2) /api 아래에 대해서 웹 브라우저 cache 방지, 예외적으로 사진은 하루 동안 cache

```
<mvc:interceptors>
    <mvc:interceptor>
        <mvc:mapping path="/api/**/*"/>
        <bean class="org.springframework.web.servlet.mvc.WebContentInterceptor">
            <property name="cacheSeconds" value="0"/>
            <property name="cacheMappings">
               <props>
                  <prop key="/api/photo/**">86400</prop>
               </props>
            </property>
         </bean>
    </mvc:interceptor>
</mvc-interceptors>
```

WebContentInterceptor Cache 관련 속성
1) cacheSeconds

- Cache 기간, 초단위 (기본값: -1)

cacheSeconds < 0	cache 관련 헤더를 설정하지 않음
cacheSeconds == 0	cache 방지 Pragma: no-cache Expires: Thu, 01 Jan 1970 00:00:00 GMT Cache-Control: max-age=0, no-cache, no-store
cacheSeconds > 0	cache Expires: Wed, 03 Jul 2013 05:58:02 GMT Cache-Control: max-age=31536000

2) useExpiresHeader