Spring MVC2 13 [스프링 인터셉터]

스프링 MVC 2편 - 백엔드 웹 개발 활용 기술

김영한

2022.10.25

스프링 인터셉터

스프링이 MVC 가 지원하는 제공하는 기술

```
HTTP 요청 → WAS → 필터 → 서블릿 → 스프링 인터셉터 → 컨트롤러
```

디스패처 서블릿과 컨트롤러 호출 직전에 호출 된다.

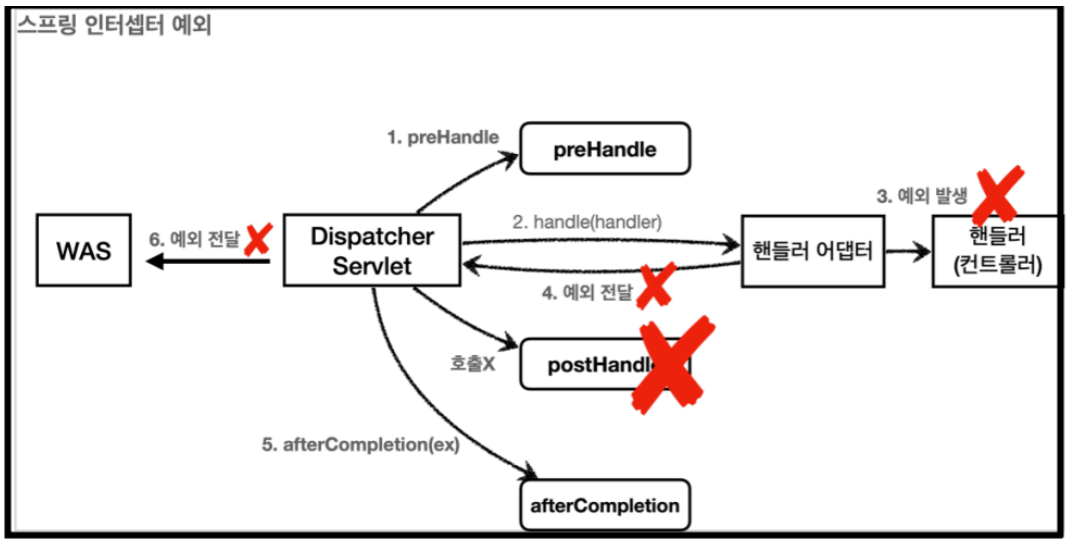
필터와 마찬가지로 체인을 적용 시킬 수 있다.

서블릿 필터보다 편리하고 다양한 기능을 제공한다.

서블릿 → doFIlter

인터셉터 → preHandle , postHandle, afterCompletion 3개로 나눠져있다.

```
디스패처 서블릿 → preHandle → 핸들러어댑터 → 컨트롤러 → postHandle →
디스패처 서블릿 → render(model) 호출 → afterCompletion → View
```



postHandle : 컨트롤러 호출 후 호출 된다.

afterCompletion : 뷰 렌더링 된 이후 호출된다.

예외가 발생해도 afterCompletion은 호출이 된다.

스프링 인터셉터로 요청 로그 남기기

로그 남기는 필터 클래스

```
@Slf4j
public class LogInterceptor implements HandlerInterceptor {

   public static final String LOG_ID = "logId";

   @Override
   public boolean preHandle(HttpServletRequest request, HttpServletResponse
response, Object handler) throws Exception {
   // 진행 -> true 로직 끝 -> false
        String requestURI = request.getRequestURI();
        String uuid = UUID.randomUUID().toString();

        request.setAttribute(LOG_ID, uuid);

        //@RequestMapping : HandlerMethod
        //정적 리소스 : ResourceHttpRequestHandler

        if(handler instanceof HandlerMethod){
            HandlerMethod hm = (HandlerMethod) handler;
        }

        log.info("REQUEST [{}][{}][{}]", uuid, requestURI, handler);

        return true;
   }

   @Override
   public void postHandle(HttpServletRequest request, HttpServletResponse response,
Object handler, ModelAndView modelAndView) throws Exception {
        log.info("postHandle [{}]", modelAndView);
   }

   @Override
   public void afterCompletion(HttpServletRequest request, HttpServletResponse
response, Object handler, Exception ex) throws Exception {
```

```
        String uuid = String.valueOf(request.getAttribute(LOG_ID));
        String requestURI = request.getRequestURI();

        log.info("RESPONSE [{}][{}][{}]", uuid, requestURI, handler);
        if(ex == null){
            log.error("afterCompletion error!!", ex);
        }
    }
}
```

Config에 필터 등록

```
@Configuration
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(new LogInterceptor())
                .order(1)
                .addPathPatterns("/**")
                .excludePathPatterns("/css**","/*.ico","/error");
    }
```

로그인 체크 인터셉터 설계

```
@Slf4j
public class LoginCheckInterceptor implements HandlerInterceptor {

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

        String requestURI = request.getRequestURI();

        log.info("인증 체크 인터셉터 실행 {}", requestURI);

        HttpSession session = request.getSession();

        if(session == null || session.getAttribute(SessionConst.LOGIN_MEMBER) ==
null){
```

```
            log.info("미인증 사용자 요청");
            //로그인으로 redirect
            response.sendRedirect("/login?redirectURL=" + requestURI);
            return false;
        }

        return true;
    }
}
```

인터셉터 확인

```
@Configuration
public class WebConfig implements WebMvcConfigurer {

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(new LogInterceptor())
                .order(1)
                .addPathPatterns("/**")
                .excludePathPatterns("/css**","/*.ico","/error");

        registry.addInterceptor(new LoginCheckInterceptor())
                .order(2)
                .addPathPatterns("/**")
                .excludePathPatterns("/", "/members/add", "/login", "/logout",
                        "/css/**", "/*.ico", "/error");
    }
```

→ 잘못된 URI도 전부 로그인으로 이동시키지만

ArgumentResolver

argumentResolver 를 통해서 로그인 구현

```
@GetMapping("/")
public String homeLoginV3ArgumentResolver(
        @Login Member loginMember, Model model){

    //세션에 회원 데이터가 없으면 home
```

```
    if(loginMember==null){
        return "home";
    }

    model.addAttribute("member", loginMember);
    return "loginHome";
}
```

로그인 애노테이션

```
@Target(ElementType.PARAMETER)
@Retention(RetentionPolicy.RUNTIME)
public @interface Login {
}
```

ArgumentResolver 구현하기

```
@Slf4j
public class LoginMemberArgumentResolver implements HandlerMethodArgumentResolver {
    @Override
    public boolean supportsParameter(MethodParameter parameter) {
        log.info("supportsParameter 실행");

        boolean hasLoginAnnotation = parameter.hasParameterAnnotation(Login.class);
        boolean hasMemberType =
Member.class.isAssignableFrom(parameter.getParameterType());

        return hasLoginAnnotation && hasMemberType;
    }

    @Override
    public Object resolveArgument(MethodParameter parameter, ModelAndViewContainer
mavContainer, NativeWebRequest webRequest, WebDataBinderFactory binderFactory)
throws Exception {

        log.info("resolveArgument 실행");

        HttpServletRequest request = (HttpServletRequest)
webRequest.getNativeRequest();
        HttpSession session = request.getSession();

        if(session == null){
            return null;
        }

        return session.getAttribute(SessionConst.LOGIN_MEMBER);
```

```
    }
}
```

→ 해당 타겟이 맞는지 확인후에 동작.

당장은 앞에서 세션 애노테이션을 사용한

```
    @GetMapping("/")
    public String homeLoginV3Spring(
            @SessionAttribute(name = SessionConst.LOGIN_MEMBER, required = false)
Member loginMember
            , Model model){
        //세션에 회원 데이터가 없으면 home
        if(loginMember==null){
            return "home";
        }
        model.addAttribute("member", loginMember);
        return "loginHome";
    }
```

이것과 큰 이점을 느끼기 힘들다.

→ 여러 컨트롤러에서 공통으로 간단하게 사용가능.