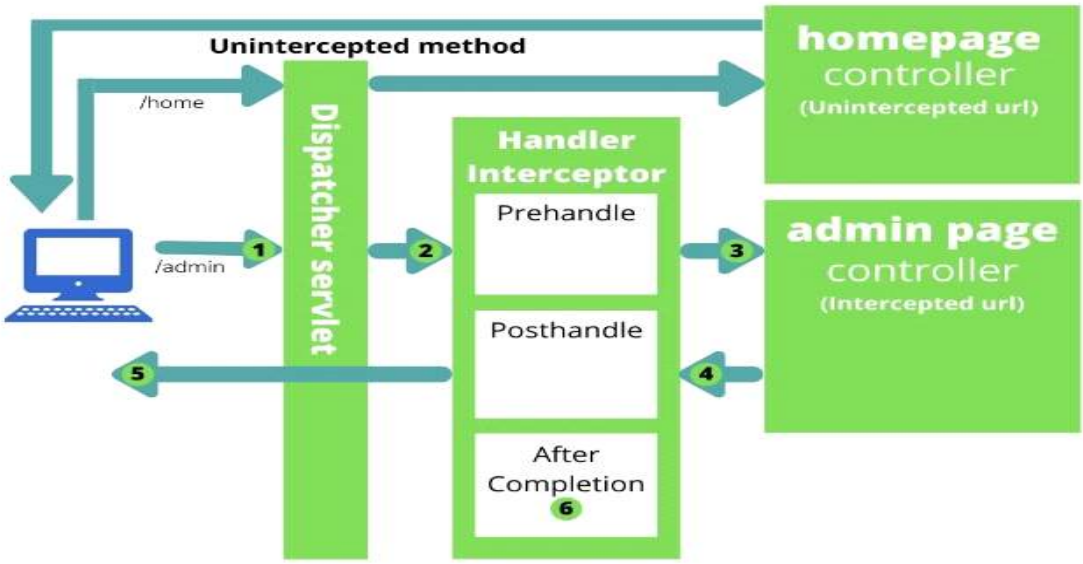
Interceptor



1. 개요

- Interceptor는 스프링 MVC에서 HTTP 요청과 응답을 가로채어 전처리와 후처리를 할 수 있는 메커니즘이다.

- 스프링부트에서 매우 강력한 기능을 제공하며, 이를 통해 요청과 응답의 흐름을 제어하고 다양한 부가 작업을 수행할 수 있다. Interceptor를 잘 활용하면 애플리케이션의 구조와 유지보수성을 크게 향상시킬 수 있다.

- 필터와 유사하지만, 스프링 MVC의 컨트롤러를 직접 호출하기 전에 추가적인 처리를 할 수 있는 점에서 차이가 있다.

- 주로 인증, 권한 검사, 로깅, 성능 측정 등의 작업을 수행할 때 사용된다.

2. Interceptor의 주요기능

2-1) 요청 전처리
- 요청이 컨트롤러에 도달하기 전에 특정 작업을 수행할 수 있다.

2-2) 요청 후처리
- 컨트롤러의 로직이 실행된 후, 뷰가 렌더링되기 전에 특정 작업을 수행할 수 있다.

2-3) 요청 완료 후 작업
- 뷰가 렌더링된 후 또는 요청이 완전히 완료된 후에 특정 작업을 수행할 수 있다.

3. Interceptor 인터페이스 및 주요 메서드

스프링 MVC의 Interceptor는 HandlerInterceptor 인터페이스를 구현하여 사용한다.
이 인터페이스는 세 가지 메소드를 제공한다.

3-1) preHandle()
컨트롤러가 호출되기 전에 실행된다.

[사용법]
boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception

기능: 컨트롤러가 실행되기 전에 호출된다. 이 메소드에서 true를 반환하면 요청이 계속 진행되고, false를 반환하면 요청이 중단된다.
사용 예시: 인증 및 권한 검사, 요청 로깅, 사용자 세션 검증 등.

3-2) postHandle()
컨트롤러가 호출된 후, 뷰가 렌더링되기 전에 실행된다.

[사용법]
void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView) throws Exception

기능: 컨트롤러가 요청을 처리한 후, 뷰가 렌더링되기 전에 호출된다.
사용 예시: 모델 데이터 수정, 추가적인 뷰 설정 등.

3-3) afterCompletion()
모든 요청 처리가 완료된 후 실행된다.

[사용법]
void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) throws Exception

기능: 뷰가 렌더링된 후 또는 요청이 완전히 완료된 후 호출된다.
사용 예시: 리소스 정리, 성능 로깅 등.

4. Spring boot Interceptor 구현방법

4-1) Interceptor클래스를 작성한다. (InterceptorSample.java 참조)
4-2) Interceptor클래스를 스프링 애플리케이션에 등록한다. (InterceptorSampleConfig.java 참조)

5. 다양한 Interceptor 사용 예제

5-1) 로깅 Interceptor - 특정 URL에 대해 로직의 수행시간을 구하는 시나리오

```
@Component
public class LoggingInterceptor implements HandlerInterceptor {

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {
        long startTime = System.currentTimeMillis();
        request.setAttribute("startTime", startTime);
        System.out.println("Request URL: " + request.getRequestURI() + " :: Start Time: " + startTime);
        return true;
    }

    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) throws Exception {
        long startTime = (Long) request.getAttribute("startTime");
        long endTime = System.currentTimeMillis();
        System.out.println("Request URL: " + request.getRequestURI() + " :: End Time: " + endTime);
        System.out.println("Request URL: " + request.getRequestURI() + " :: Time Taken: " + (endTime - startTime));
    }
}
```

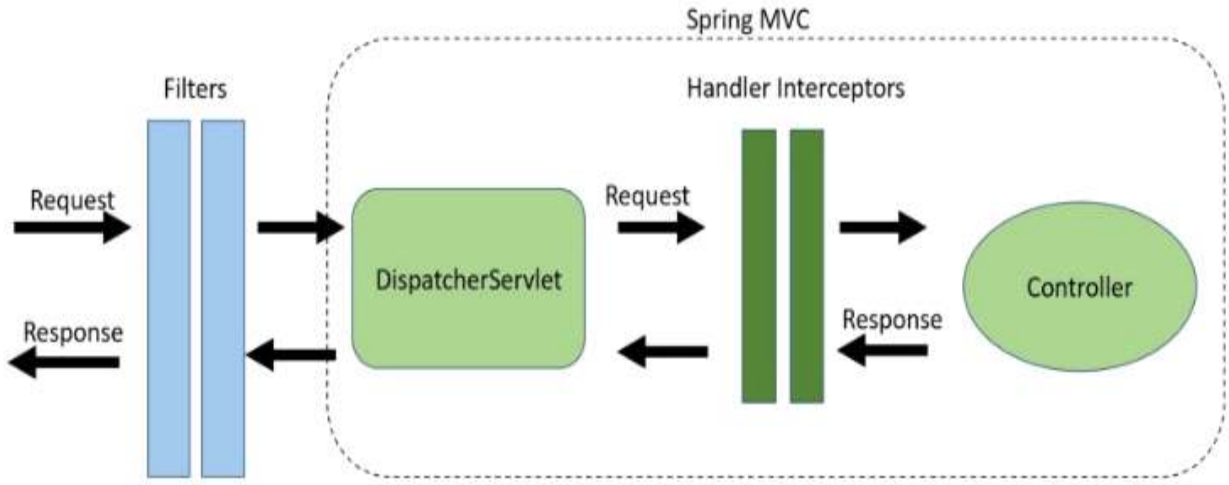
5-2) 권한 검사 Interceptor

특정 URL에 대해 권한을 검사하고, 권한이 없으면 요청을 중단하는 시나리오

```
@Component
public class AuthInterceptor implements HandlerInterceptor {

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {
        String authToken = request.getHeader("Authorization");
        // 권한 검사를 수행합니다 (예: 토큰 검증)
        if (authToken == null || !authToken.equals("VALID_TOKEN")) {
            response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
            return false; // 권한이 없으면 요청 중단
        }
        return true;
    }}
```

6. Interceptor와 Filter의 차이점



6-1) Filter

- 서블릿 스펙에 정의된 기능으로, 요청이 서블릿에 도달하기 전후에 요청과 응답을 가로채고 조작할 수 있다.

- 주로 인코딩 설정, 보안 관련 처리, 로깅 등에 사용된다.

레벨: 서블릿 스펙 레벨
위치: 디스패처 서블릿 앞
목적: 요청과 응답을 가로채어 처리, 주로 인코딩 설정, 보안 관련 처리, 로깅 등에 사용됩니다.
메소드: doFilter() , init() , destory()

6-2) Interceptor

- 스프링 MVC에 정의된 기능으로, 디스패처 서블릿이 컨트롤러를 호출하기 전후에 요청과 응답을 가로채고 조작할 수 있다.

- 주로 인증, 권한 검사, 로깅, 성능 측정 등에 사용된다.

레벨: 스프링 MVC 레벨
위치: 디스패처 서블릿과 컨트롤러 사이
목적: 요청과 응답의 흐름을 제어하며, 주로 인증, 권한 검사, 로깅, 성능 측정 등에 사용된다.
메소드 : preHandle() , postHandle() , afterCompletion()