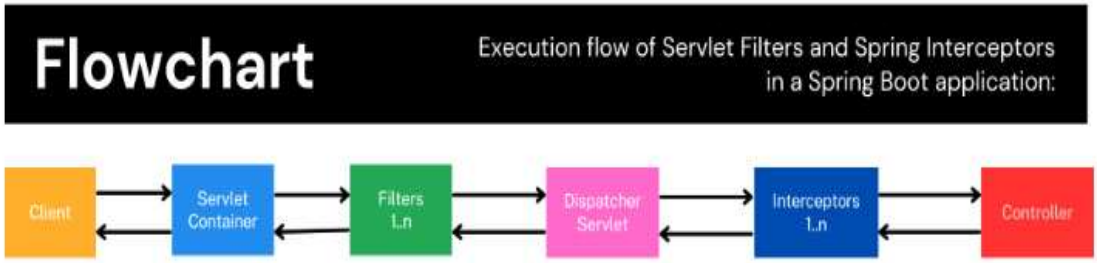
Filter



출처 : https://bootcamptoprod.com/spring-interceptors-vs-filters-ultimate-comparison/

1. 개요

필터(Filter)는 서블릿 스펙에 정의된 기능으로, 웹 애플리케이션 내에서 클라이언트의 요청과 서버의 응답을 가로채어 중간에서 처리할 수 있는 컴포넌트이다. 필터는 요청이 서블릿이나 기타 리소스에 도달하기 전에 특정 작업을 수행하거나, 응답이 클라이언트에 반환되기 전에 작업을 수행할 수 있는 기회를 제공한다. 이로 인해 필터는 다양한 용도로 사용될 수 있으며, 주로 다음과 같은 작업에 사용된다.

1-1) 인코딩 설정

웹 애플리케이션에서 모든 요청과 응답에 대해 일관된 인코딩 방식을 설정하는 것은 매우 중요하다. 필터를 사용하면 모든 요청과 응답의 인코딩을 간편하게 설정할 수 있다. 예를 들어, UTF-8 인코딩을 설정하여 다국어 지원을 강화할 수 있다.

1-2) 보안 관련 처리

필터는 보안 목적으로 많이 사용된다. 예를 들어, 인증(Authentication)과 권한 부여(Authorization)를 처리하는 필터를 구현하여 사용자가 접근할 수 있는 리소스를 제한할 수 있다. 특정 URL 패턴에 대해 사용자의 인증 정보를 확인하거나, 특정 역할(Role)에 따라 접근 권한을 부여하는 등의 작업을 수행할 수 있다.

1-3) 로깅(Logging)

웹 애플리케이션의 모든 요청과 응답을 로깅하는 것은 디버깅과 모니터링에 매우 유용하다. 필터를 통해 요청의 URI, HTTP 메서드, 응답 상태 코드 등의 정보를 로깅하여 후속 분석에 활용할 수 있다. 이를 통해 애플리케이션의 사용 패턴을 파악하고, 문제 발생 시 원인을 추적할 수 있다.

1-4) 요청 및 응답 변환

필터는 요청 또는 응답 객체를 변환하는 데 사용할 수 있다. 예를 들어, 요청 본문을 압축 해제하거나, 응답 본문을 압축하여 전송하는 등의 작업을 수행할 수 있다. 또한, 요청 헤더나 파라미터를 수정하거나, 응답 헤더를 추가하는 등의 작업도 가능하다.

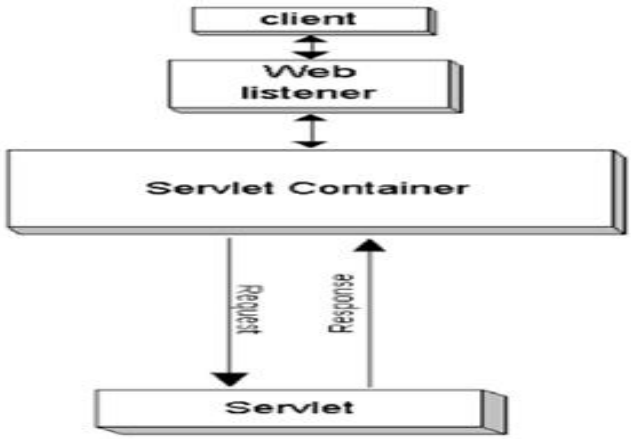
1-5) URL 재작성(URL Rewriting)

필터를 사용하여 클라이언트의 요청 URL을 다른 URL로 재작성할 수 있다. 이는 SEO(Search Engine Optimization)를 개선하거나, 레거시 URL 구조를 유지하면서 새로운 URL 구조를 사용할 때 유용한다. 예를 들어, /old-path로 들어오는 요청을 /new-path로 리다이렉트할 수 있다.

2. Filter의 동작 원리

- Filter는 jakarta.servlet.Filter 인터페이스를 구현하여 사용한다.

[기본 서블릿 처리 흐름]



1) 클라이언트(Client)
- 클라이언트는 웹 브라우저, 모바일 앱 등으로부터 HTTP 요청을 보낸다.

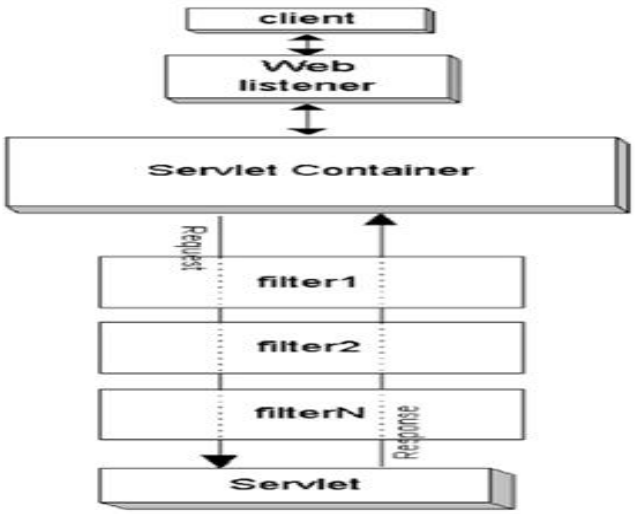
2) 웹 리스너(Web Listener)
- 요청이 서블릿 컨테이너로 전달되기 전에 특정 이벤트를 청취하고 처리하는 역할을 한다.

3) 서블릿 컨테이너(Servlet Container)
- 서블릿 컨테이너는 톰캣(Tomcat)과 같은 웹 서버로, 클라이언트로부터 요청을 받아들이고 요청을 서블릿으로 전달한다,

4) 서블릿(Servlet)
 - 서블릿은 요청을 처리하고 응답을 생성한다.
 - 생성된 응답은 서블릿 컨테이너를 통해 클라이언트로 반환된다.

[요약]
클라이언트 -> 서블릿 컨테이너 -> 서블릿 -> 서블릿 컨테이너 -> 클라이언트

[필터를 포함한 서블릿 처리 흐름]



클라이언트(Client)
 - 클라이언트는 웹 브라우저, 모바일 앱 등으로부터 HTTP 요청을 보낸다.

웹 리스너(Web Listener)
 - 요청이 서블릿 컨테이너로 전달되기 전에 특정 이벤트를 청취하고 처리하는 역할을 한다.

서블릿 컨테이너(Servlet Container)
 - 서블릿 컨테이너는 클라이언트로부터 요청을 받아들인다.
 - 요청을 필터 체인으로 전달한다.

필터(Filter) 1..n:
 - 여러 필터가 체인 형태로 존재하며, 각 필터는 요청을 처리한 후 다음 필터로 전달한다.
 - 필터는 인증, 로깅, 압축, URL 재작성 등의 작업을 수행한다.

서블릿(Servlet):
 - 모든 필터를 통과한 요청은 최종적으로 서블릿에 도달하여 처리된다.
 - 서블릿은 요청을 처리하고 응답을 생성한다.
 - 생성된 응답은 다시 필터 체인을 거쳐 클라이언트로 반환된다.

[요약]
클라이언트 -> 서블릿 컨테이너 -> 필터1 -> 필터2 -> ... -> 필터N -> 서블릿 -> 필터N -> ... -> 필터2 -> 필터1 -> 서블릿 컨테이너 -> 클라이언트

[주요 메서드]

1) init()

- 필터가 초기화될 때 호출되는 메서드로, 필터 설정 정보를 매개변수로 받는다. 필터 초기화 시 필요한 설정 작업을 수행한다.

사용 예시 : 필터 설정 초기화, 리소스 할당 등.

2) doFilter()

- 필터의 핵심 메서드로, 모든 요청과 응답에 대해 호출된다. 이 메서드 내에서 요청을 처리하거나 응답을 수정할 수 있으며, chain.doFilter(request, response)를 호출하여 다음 필터나 서블릿으로 요청을 전달할 수 있다.

사용 예시 : 요청 전처리, 응답 후처리, 보안 검증, 로깅 등.

3) destroy()

- 필터가 종료될 때 호출되는 메서드로, 자원 해제 등의 작업을 수행한다.

사용 예시 : 리소스 해제, 정리 작업 등.

4. Filter 설정 방법

4-1) Filter 클래스를 작성한다. (FilterExSample.java 참조)

4-2) 스프링부트 애플리케이션에 Filter를 등록한다. (FilterConfigSample.java 참조)

5. Filter 사용 예시

5-1) 로깅 Filter - 요청과 응답을 로깅하는 시나리오

```
public class LoggingFilter implements Filter {

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
        long startTime = System.currentTimeMillis();
        System.out.println("Request received at " + startTime);
        chain.doFilter(request, response);
        long endTime = System.currentTimeMillis();
        System.out.println("Response sent at " + endTime);
        System.out.println("Time taken: " + (endTime - startTime) + "ms");
    }
}
```

5-2) 인증 Filter - 사용자가 인증되었는지 확인하는 시나리오

```
public class AuthFilter implements Filter {

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
        HttpServletRequest httpRequest = (HttpServletRequest) request;
        HttpServletResponse httpResponse = (HttpServletResponse) response;

        String authToken = httpRequest.getHeader("Authorization");

        // 인증 검사를 수행합니다 (예: 토큰 검증)
        if (authToken == null || !authToken.equals("VALID_TOKEN")) {
            httpResponse.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
            return; // 인증 실패 시 요청 중단
        }

        chain.doFilter(request, response); // 인증 성공 시 요청 진행
    }

}
```