13 : 필터 & 랩퍼

학습 목표

- 필터를 이해하고 Request 필터와 Response 필터에 대해 알아본다
- 랩퍼에 대해 이해하고 그 사용을 알아본다

1. 필터 정의

2. Request 필터

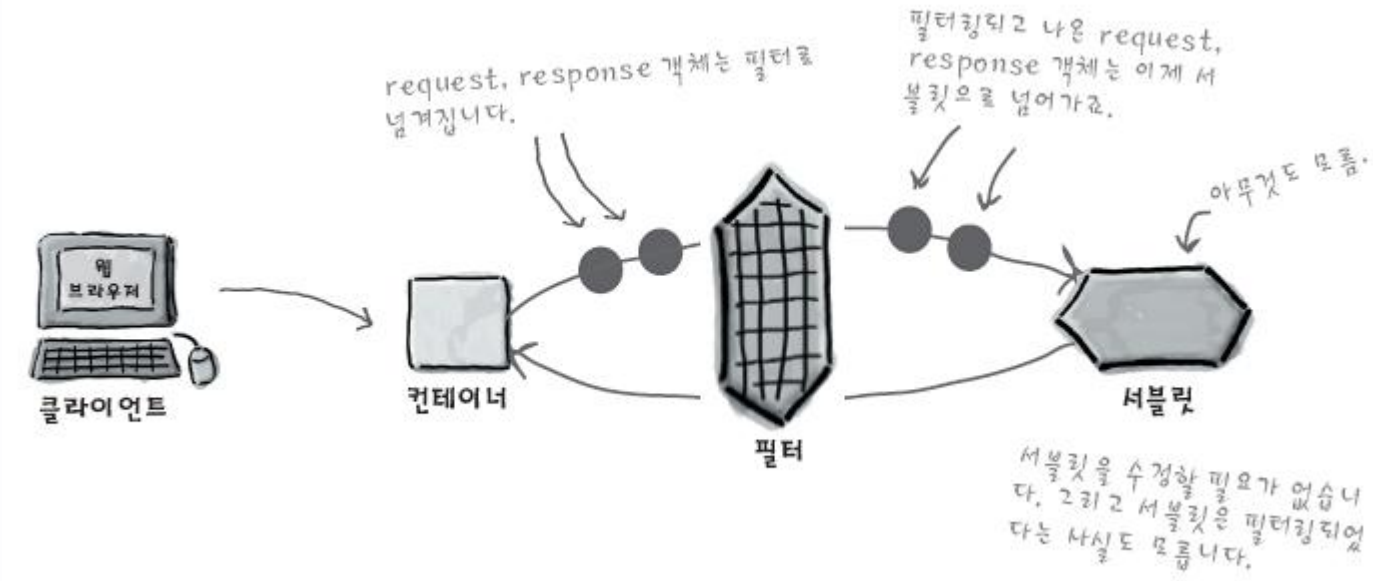
3. Response 필터

4. 랩퍼

필터 >> request 필터와 response 필터

필터

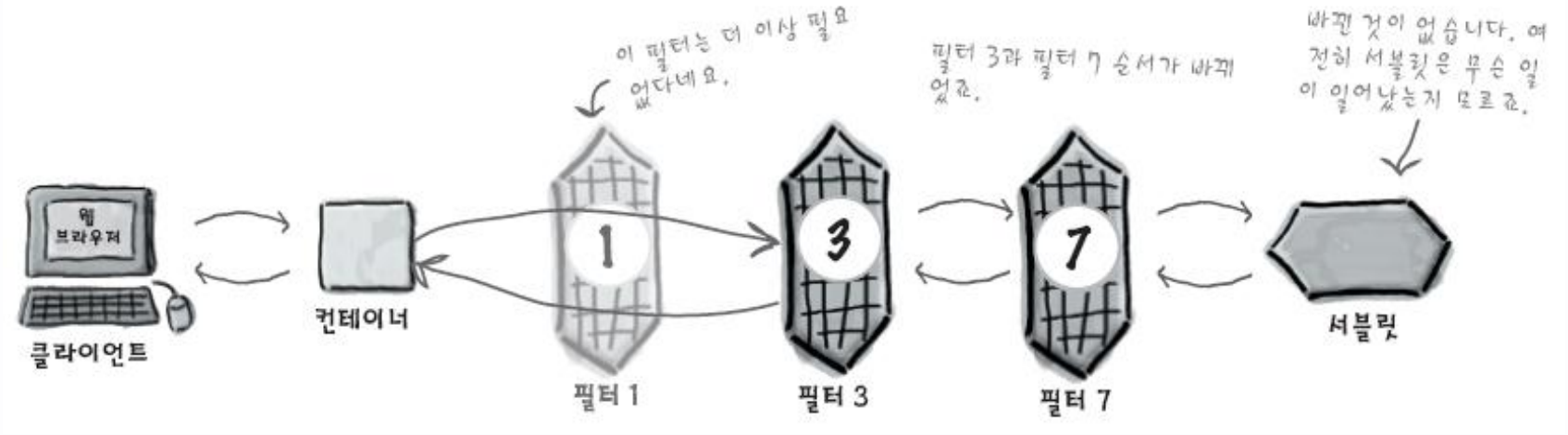
- 자바 컴포넌트
- DD 에 필터를 선언하면 이를 컨테이너에서 사용



Request 필터 >> DD 설정

DD 설정 1

DD 설정 2



Request 필터 >> 라이프 사이클

init()

- 초기화

doFilter()

- 필터의 기능 구현

destroy()

- 필터 인스턴스 제거 시 사용

Request 필터 >> 필터 선언 및 순서 설정

DD 에 설정

필터 정의하기

```
<filter>
  <filter-name>BeerRequest</filter-name>
  <filter-class>com.example.web.BeerRequestFilter
        </filter-class>
  <init-param>
    <param-name>LogFileName</param-name>
    <param-value>UserLog.txt</param-value>
  </init-param>
</filter>
```

URL 패턴과 필터 매핑 선언하기

```
<filter-mapping>
  <filter-name>BeerRequest</filter-name>
  <url-pattern>*.do</url-pattern>
</filter-mapping>
```

서블릿 이름에 필터 매핑 선언하기

```
<filter-mapping>
  <filter-name>BeerRequest</filter-name>
  <servlet-name>AdviceServlet</servlet-name>
</filter-mapping>
```

〈filter〉 선언 규칙

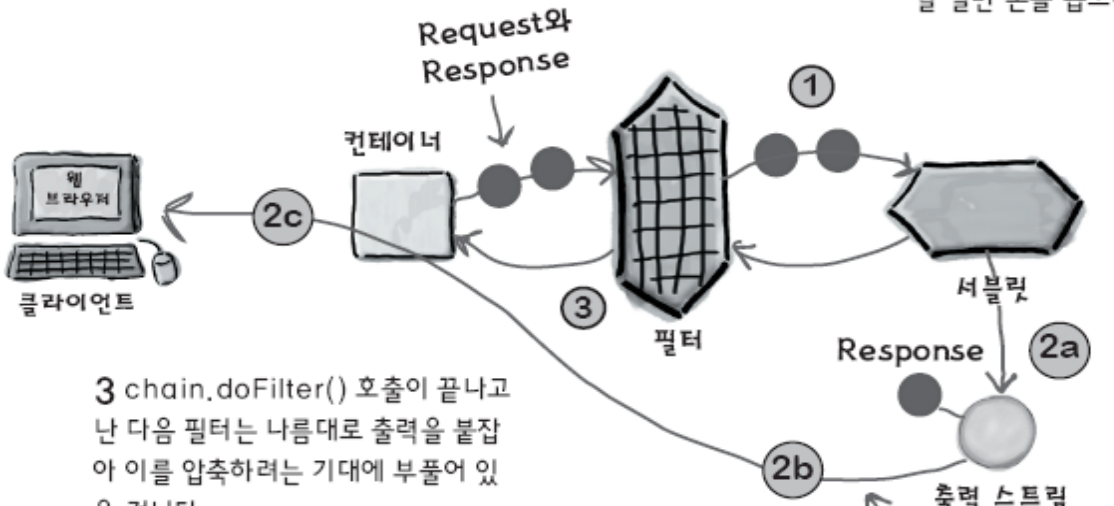
- 〈filter-name〉은 반드시 있어야 하는 항목입니다.
- 〈filter-class〉는 반드시 있어야 하는 항목입니다.
- 〈init-param〉은 옵션 항목이며, 여러 개 있어도 무방합니다.

〈filter-mapping〉 선언 규칙

- 〈filter-name〉은 반드시 있어야 하는 항목이며, 연결된 〈filter〉 선언을 찾기 위해 정의합니다.
- 〈url-pattern〉 또는 〈servlet-name〉, 둘 중 하나는 반드시 있어야 하는 항목입니다.
- 〈url-pattern〉 항목에는 어떤 웹 애플리케이션 리소스들에 필터를 적용할지 정의합니다.
- 〈servlet-name〉 항목은 필터를 적용할 웹 애플리케이션 하나를 정의하는 곳입니다.

Response 필터 >> Response 필터 구조

■ 문제점



1 필터가 서블릿으로 request, response를 넘겨줍니다. 그리고 압축할 날만 손을 꼽으며 기다리고 있죠.

2a 서블릿은 자기 할 일을 한 뒤 출력을 만들겠죠. 하지만 자신이 만든 출력이 압축될 줄은 꿈에도 모를걸요.

2b 출력에 대한 제어는 이제 컨테이너로 넘어가고, 그리고...

2c 이제 클라이언트로 넘어가겠죠... 흠... 이게 문제입니다. 필터는 출력이 클라이언트로 넘어가기 전에 압축해야 되기 때문에 이제나 저제나 하며 기다리고만 있을 겁니다.

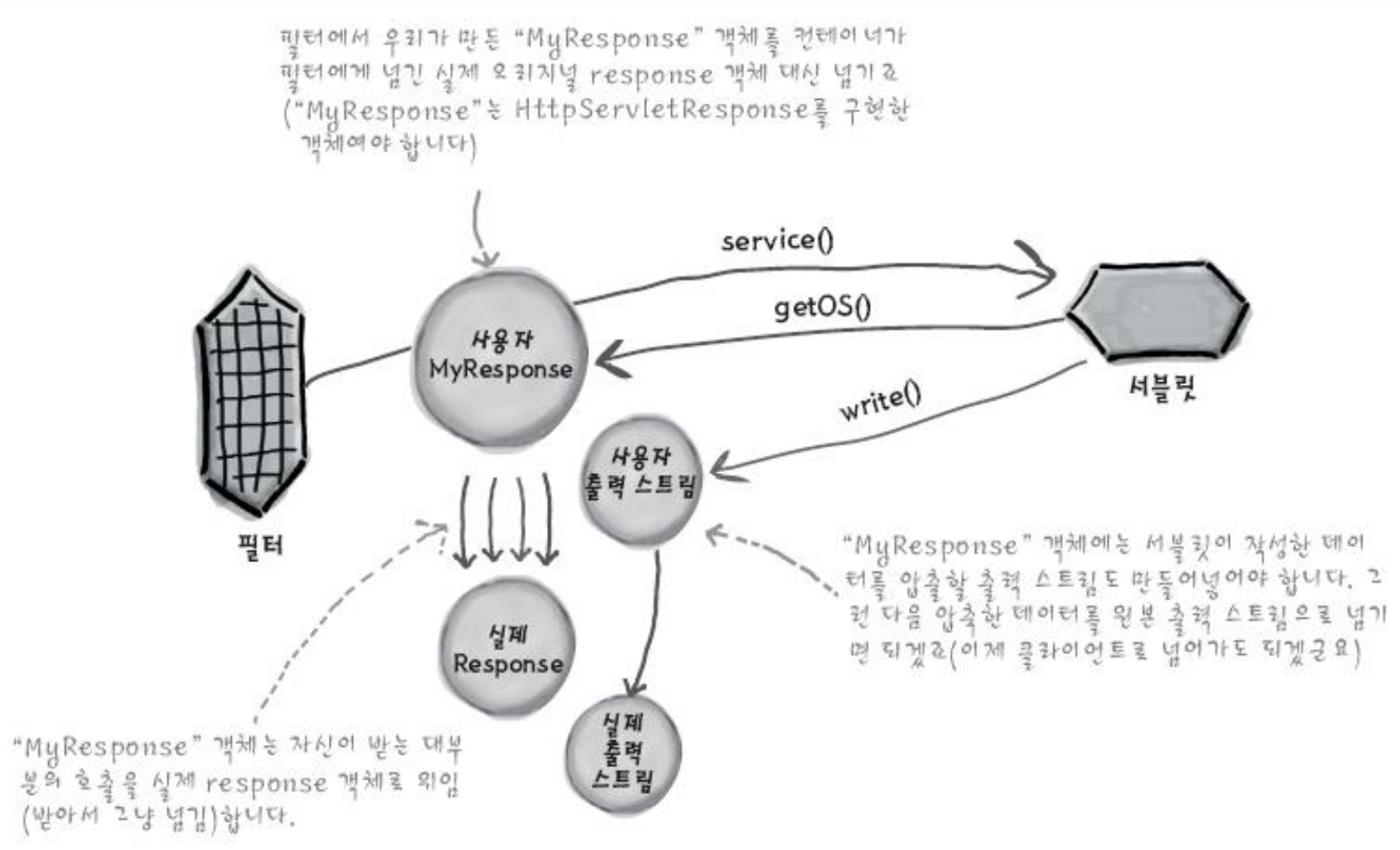
3 chain.doFilter() 호출이 끝나고 난 다음 필터는 나름대로 출력을 붙잡아 이를 압축하려는 기대에 부풀어 있을 겁니다.

하지만 너무 늦었답니다! 출력은 이미 클라이언트로 넘어가 버렸으니까요! 컨테이너가 필터를 위해서 따로 출력 버퍼를 만들지 않기 때문이죠. 개념적으로 아직 필터의 doFilter() 메소드가 스택 제일 위에 있지만서도, 필터가 출력에 어떤 제어를 가하기에는 너무 늦었지요.

이런... 이 부분에서 문제가 발생합니다. 출력은 필터를 위해 기다려 주지 않는다는 사실...

Response 필터 >> Response 필터 작성

■ 새로운 응답 객체를 생성



랩퍼 >> 랩퍼 정의

■ 랩퍼 란?

- 원본 클래스를 둘러 싼 다음, 자신에게 들어오는 호출을 원본 클래스에 위임을 한다.
- 새로운 기능이 필요한 경우 필요한 메소드만 재정의를 한다.
- **request, response 랩퍼**
  - **ServletRequestWrapper**
  - **HttpServletRequestWrapper**
  - **ServletResponseWrapper**
  - **HttpServletResponseWrapper**

랩퍼 >> Response 랩퍼

■ 압축 필터 설계

압축 필터 설계, 버전 2(의사 코드)

이게 의도대로 랩퍼 클래스를 상속받아 보죠.

```
class CompressionResponseWrapper extends HttpServletResponseWrapper {

   // 커스터마이징할 메소드를 여기에 재정의하세요.
}
```

이제 이 안에 필요한 것만 재정의하면 되겠죠. 이건 다음 페이지에서부터 차차 해보도록 하고...

```
class MyCompressionFilter implements Filter {

   public void init(FilterConfig cfg) { }

   public void doFilter( request, response, chain) {

      CompressionResponseWrapper wrappedResp
        = new CompressionResponseWrapper(response);

      chain.doFilter(request, wrappedResp);

      // 압축 로직이 들어갈 자리입니다.
   }
   public void destroy() { }
 }
}
```

이 부분이 바로 랩퍼 클래스로 실제 response 객체를 "감싸는" 부분입니다.

자 이제 필터 체인으로 랩퍼 클래스를 넘겨보죠. 체인에 있는 어떤 클래스도 우리가 response 클래스를 감싼 랩퍼 클래스를 넘겼는지 모르겠죠.

랩퍼 >> Response 랩퍼

■ 압축 필터 설계 : 출력 스트림 추가

압축 필터 설계, 버전 3(의사 코드)

새로 만든 출력 스트림을 취득하기 위해 이 메소드를 재정의합니다.

```
class CompressionResponseWrapper extends HttpServletResponseWrapper {

   public ServletOutputStream getOutputStream() throws... {
     ...
     servletGzipOS = new GzipSOS(resp.getOutputStream());
     return servletGzipOS;
   }

  // 재정의할 메소드가 있다면 여기에 하세요.
}
```

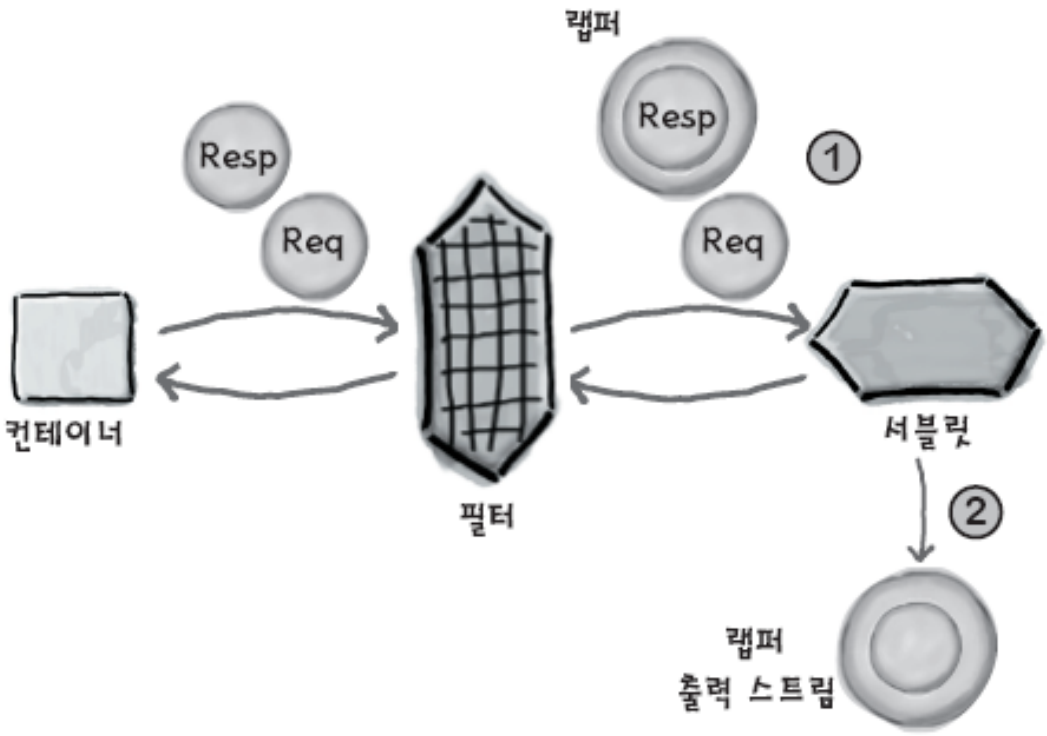
GzipServletOutputStream 클래스가 ServletOutputStream을 상속했다고 합시다. 이곳이 새로 만든 GzipServletOutputStream으로 ServletOutputStream을 감싸고 있는 현장입니다.

누구든 이 메소드를 호출하면 "특별히" 고안한 ServletOutputStream을 넘겨받을 겁니다.

```
class MyCompressionFilter implements Filter {

  public void init(FilterConfig cfg) { }

  public void doFilter( request, response, chain) {

    CompressionResponseWrapper wrappedResp
       = new CompressionResponseWrapper(response);

    chain.doFilter(request, wrappedResp);

    // 압축 로직이 들어갈 자리입니다.
  }
  public void destroy() { }

}
```

랩퍼 >> Response 랩퍼

■ 압축 필터 설계 : 출력 스트림 추가



1 필터는 request 객체와 랩퍼 response 객체를 서블릿으로 넘깁니다. 랩퍼 response 객체는 getOutputStream 메소드를 아주 특별하게 재정의했죠.

2 서블릿이 getOutputStream 메소드를 호출하여 출력 스트림을 받더라도, 그것이 우리가 새로 만든 놈이란 걸 전혀 눈치채지 못할걸요.