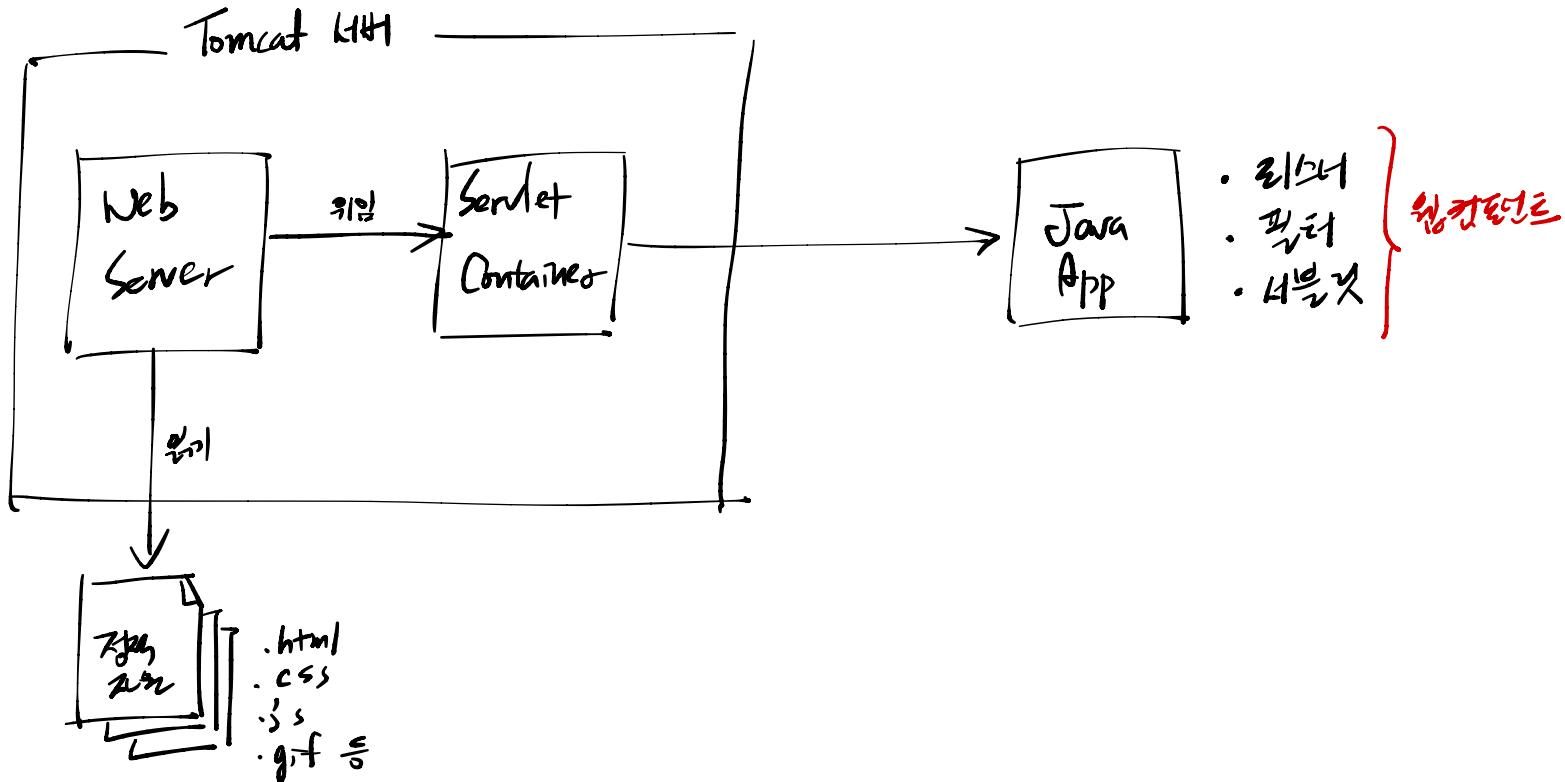


* 웹 컨테이너(부록)
↳ 한 개 이상의 기능으로 구성되어 특별한 권한을 부여받을 것. = 특별한 역할은 부여받는 기능



* 예상할 수 있는 관계 - 관찰자 = Observer (Observer)
Subject

※ 관찰자가 특정 상황에 반응하는
경우에는 관찰자 \Rightarrow "관찰자"

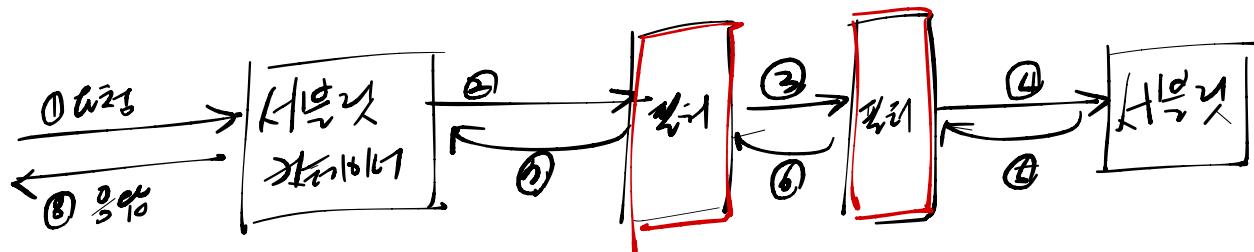
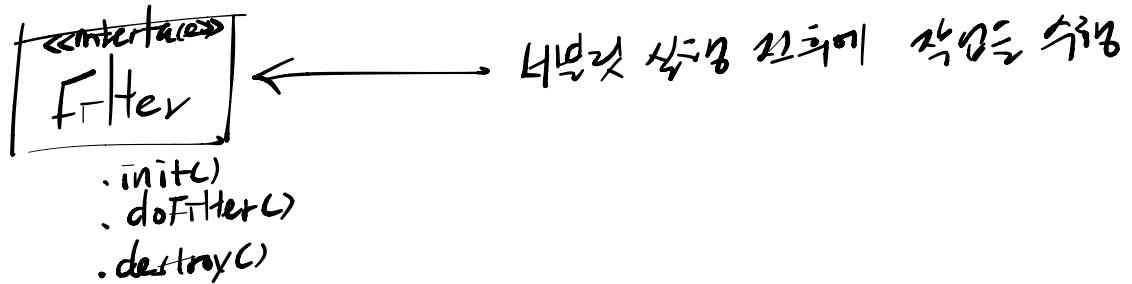
ServletContextListener ← 서버가 컨텍스트가 초기화되거나 종료될 때 호출되는
contextInitialized() contextDestroyed()

ServletRequestListener ← 웹서버의 모든 요청이 제작되었거나 제거될 때 호출되는
requestInitialized() requestDestroyed()

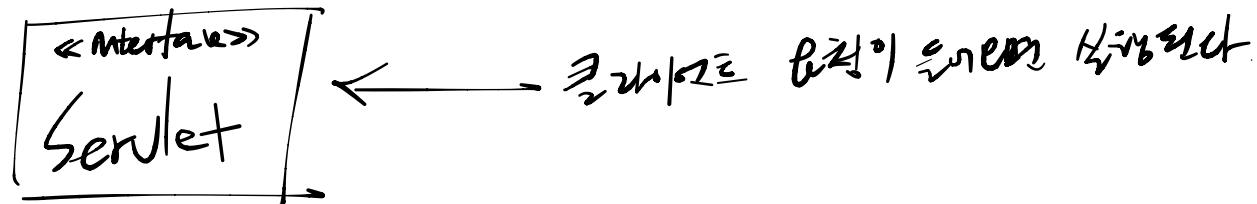
HttpSessionListener ← 클라이언트 세션이 생성되었거나 종료되었을 때
sessionCreated() sessionDestroyed()

:

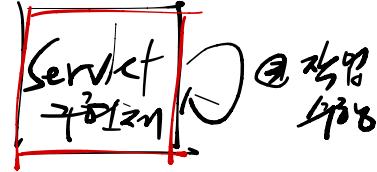
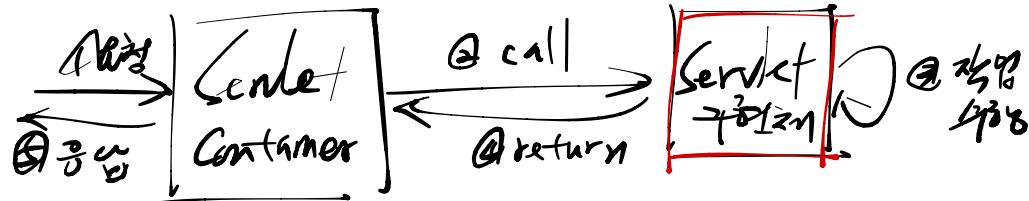
* 필터 구조화 - 책임 = chain of Responsibility
 GOF의 책임 체인



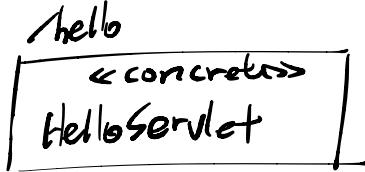
* 웹 개발언어 - HTML = Command
 GOF의 디자인 패턴



- . init()
- service()
- . destroy()
- . getServletInfo()
- . getServletConfig()



* Servlet API

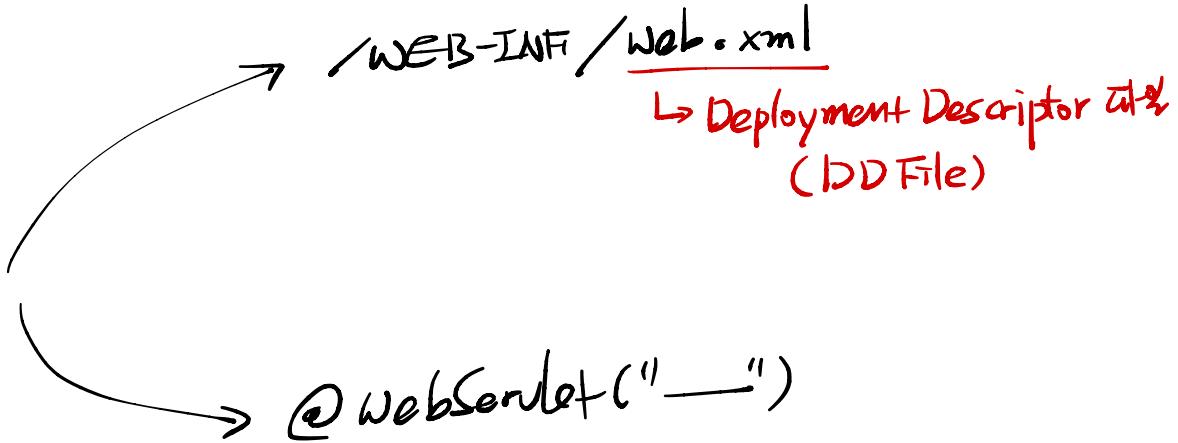


- `init()`
- `service()`
- `destroy()`
- `getServletInfo()`
- `getServletConfig()`

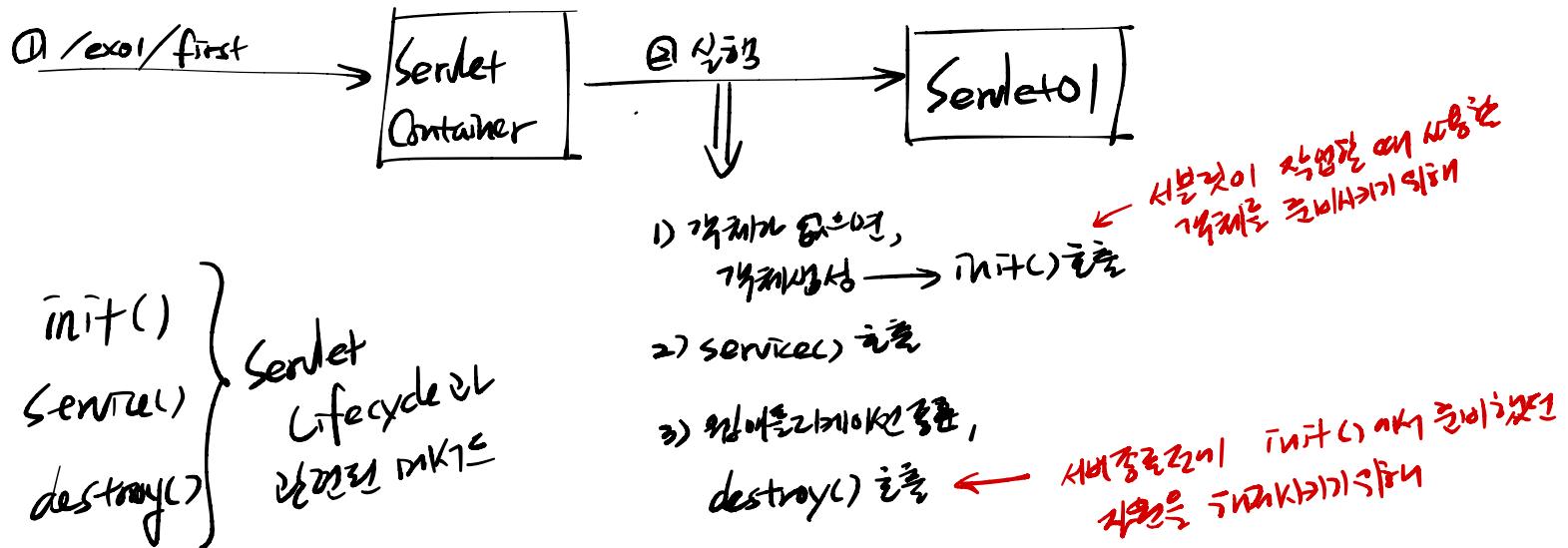
- `init() {} ->`
- `service() {} ->`
- `destroy() {} ->`
- `getServletInfo() {} ->`
- `getServletConfig() {} ->`

* 웹 서비스
|> JSP

Servlet 편집기



* 서블릿의 생명주기



* service() 데일리

작성

Tomcat 키트 : HttpServletRequest 향상 | HttpServletResponse 향상

↳ 흐름

↓

← 실제 애플리케이션 → ↓

service(ServletRequest req, ServletResponse resp) {}

↑

↑

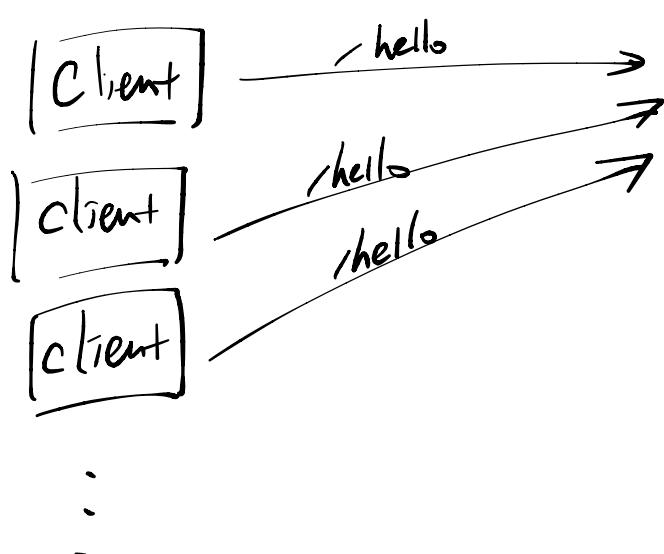
✓ 헤더와 클리에이션

✓ 응답을 할 때 어떤 HTTP를 써야

응답을 할 때 어떤 HTTP를 써야

Tomcat 키트는 HTTP 프로토콜로 통신해야 한다!

* 클라이언트 요청과 서버의

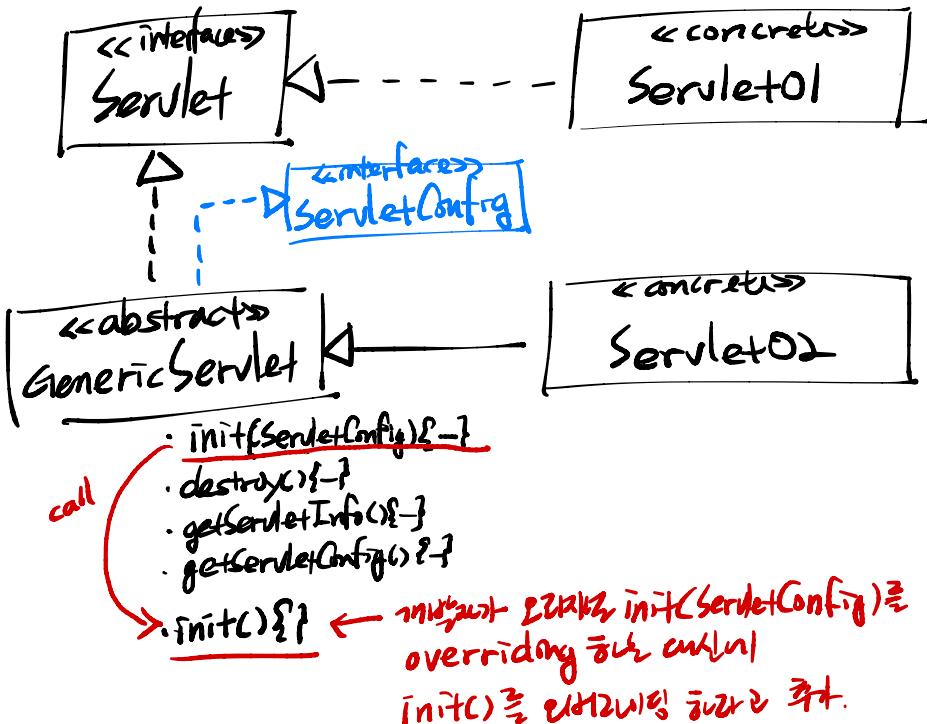


[HelloServlet
인스턴스]

↑
한 개의 객체는 여러 클라이언트가 공유
된다!

특정 클라이언트의 작업 결과는
서버의 인스턴스 필드에 보관하고
返还!

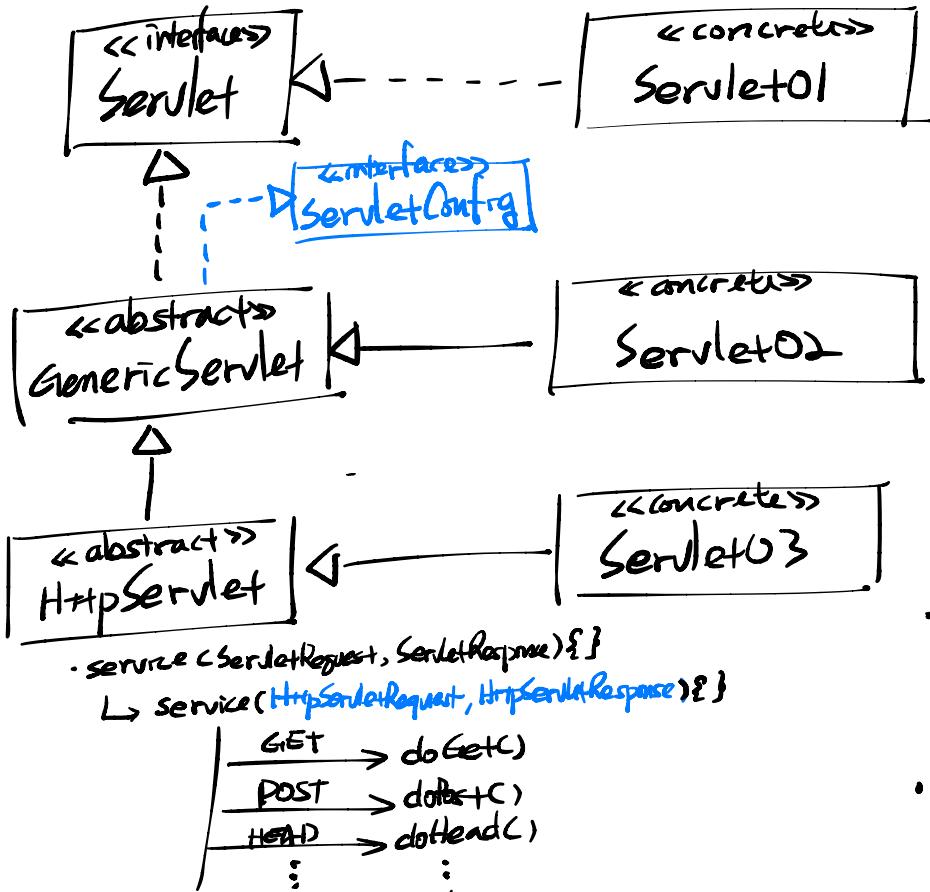
* 亂世のアーキテクチャ II



- init() { - } *
- service() { - } *
- destroy() { - }
- getServletInfo() { - }
- getServletConfig() { - }

service() { - }

* 서블릿의 마법 | III



. init() {} — } *

~~. service() {} — } *~~

. destroy() {}

. getServletInfo() {} — }

. getServletConfig() {} — }

. service() {} — }

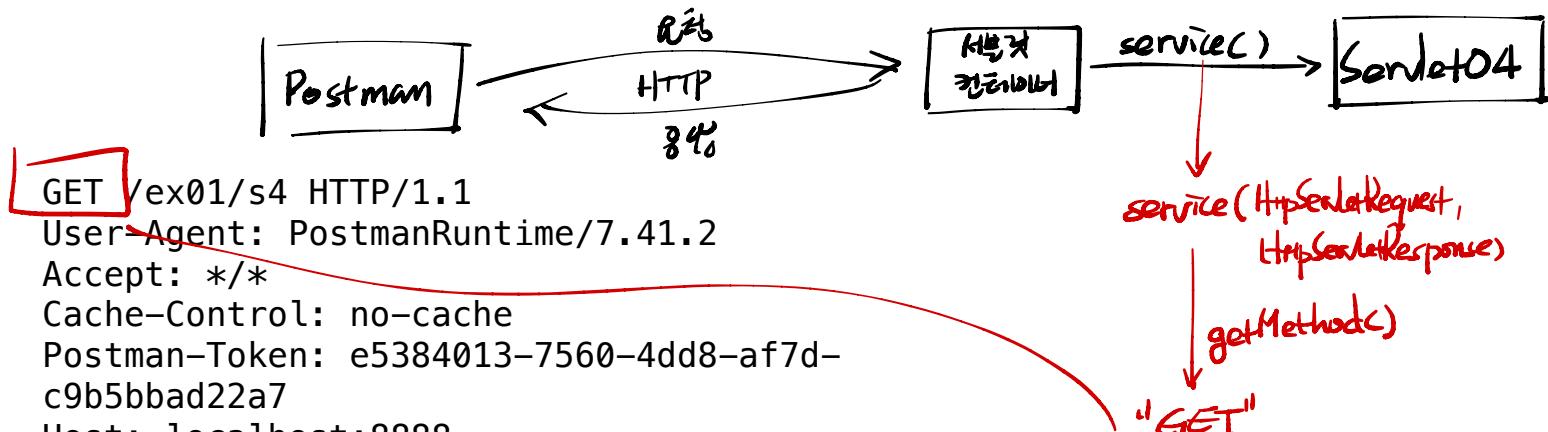
. GET 요청을 처리하는 경우,
doGet() {} — }

. POST 요청을 처리하는 경우,
doPost() {} — }

:
⋮

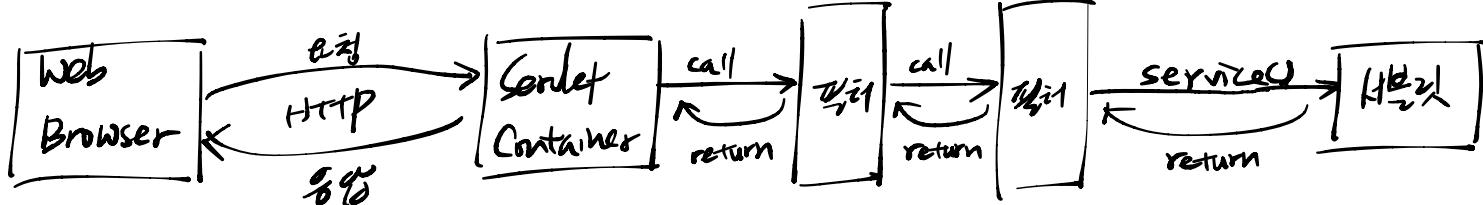
. 모든 요청을 처리하는 경우,
service(HttpServletRequest, HttpServletResponse)

* Backend App init

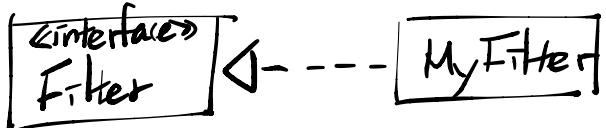


* 필터 만들기

① 구조



② 구현



- init()
- doFilter()
- destroy()

doFilter()

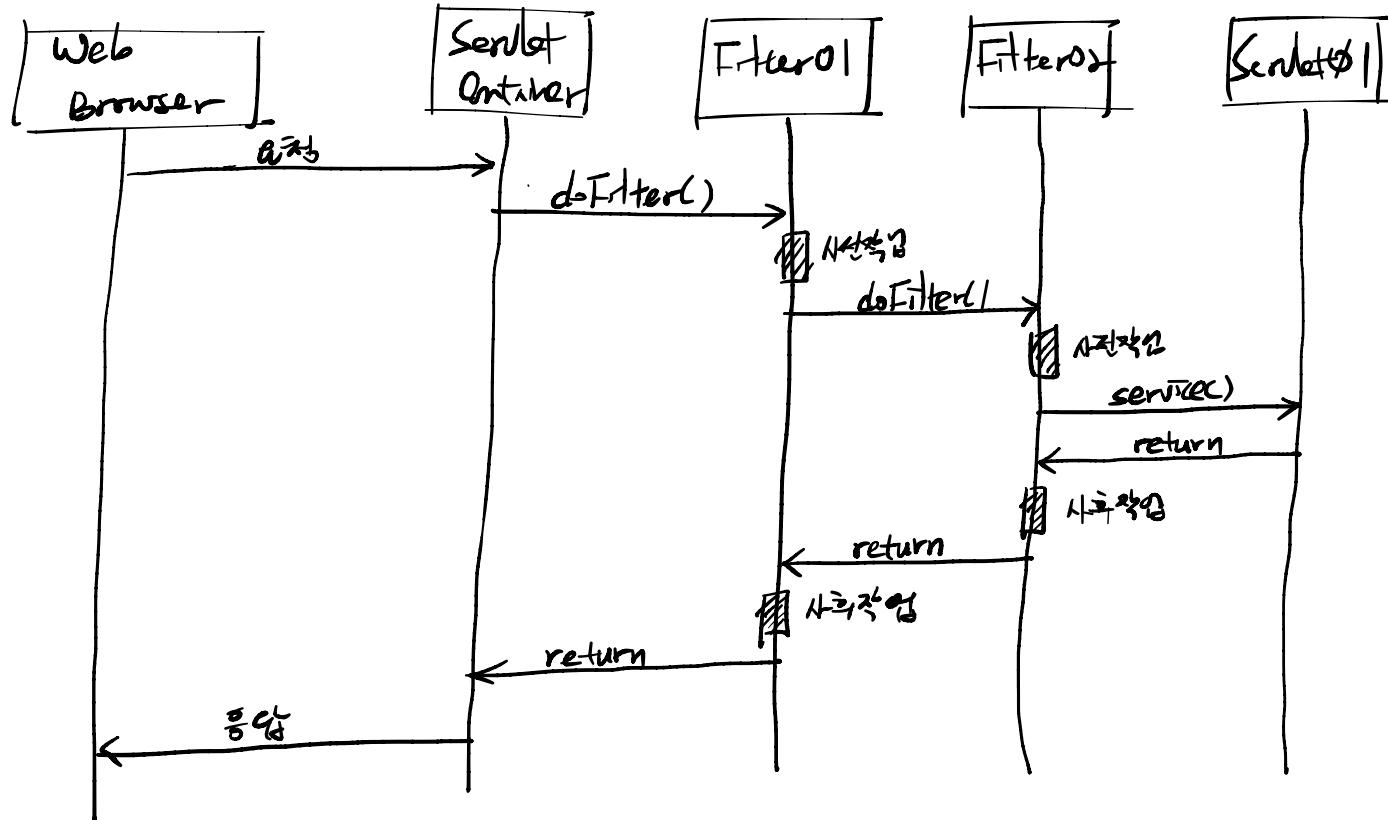
사전작업

다음 처리 또는 서블릿

사후작업

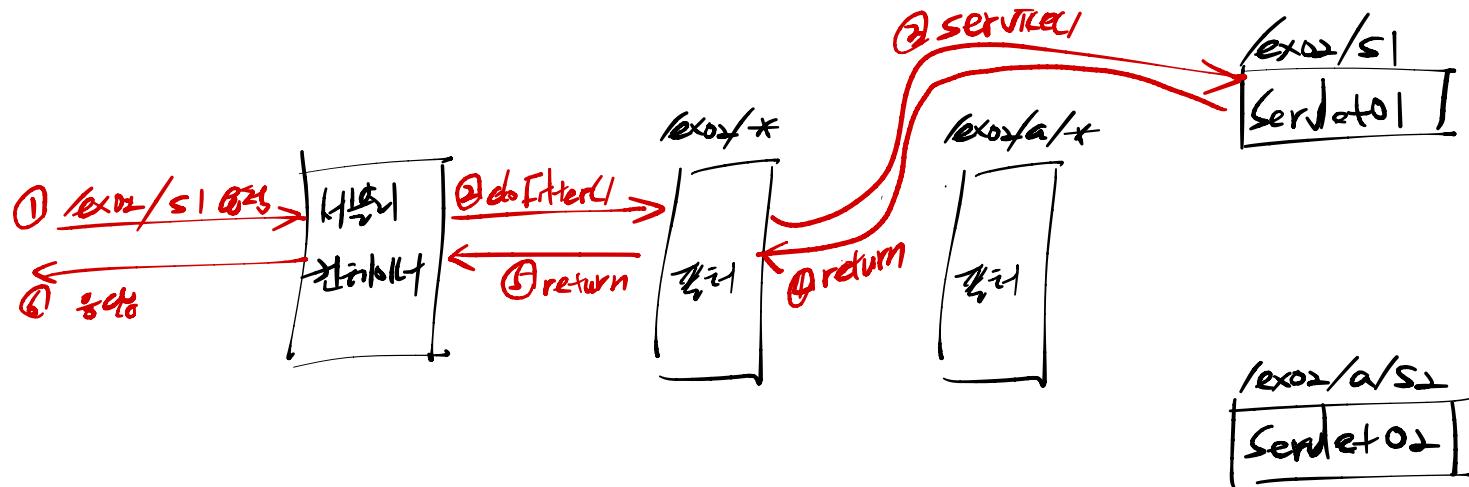
}

* 필터링 흐름

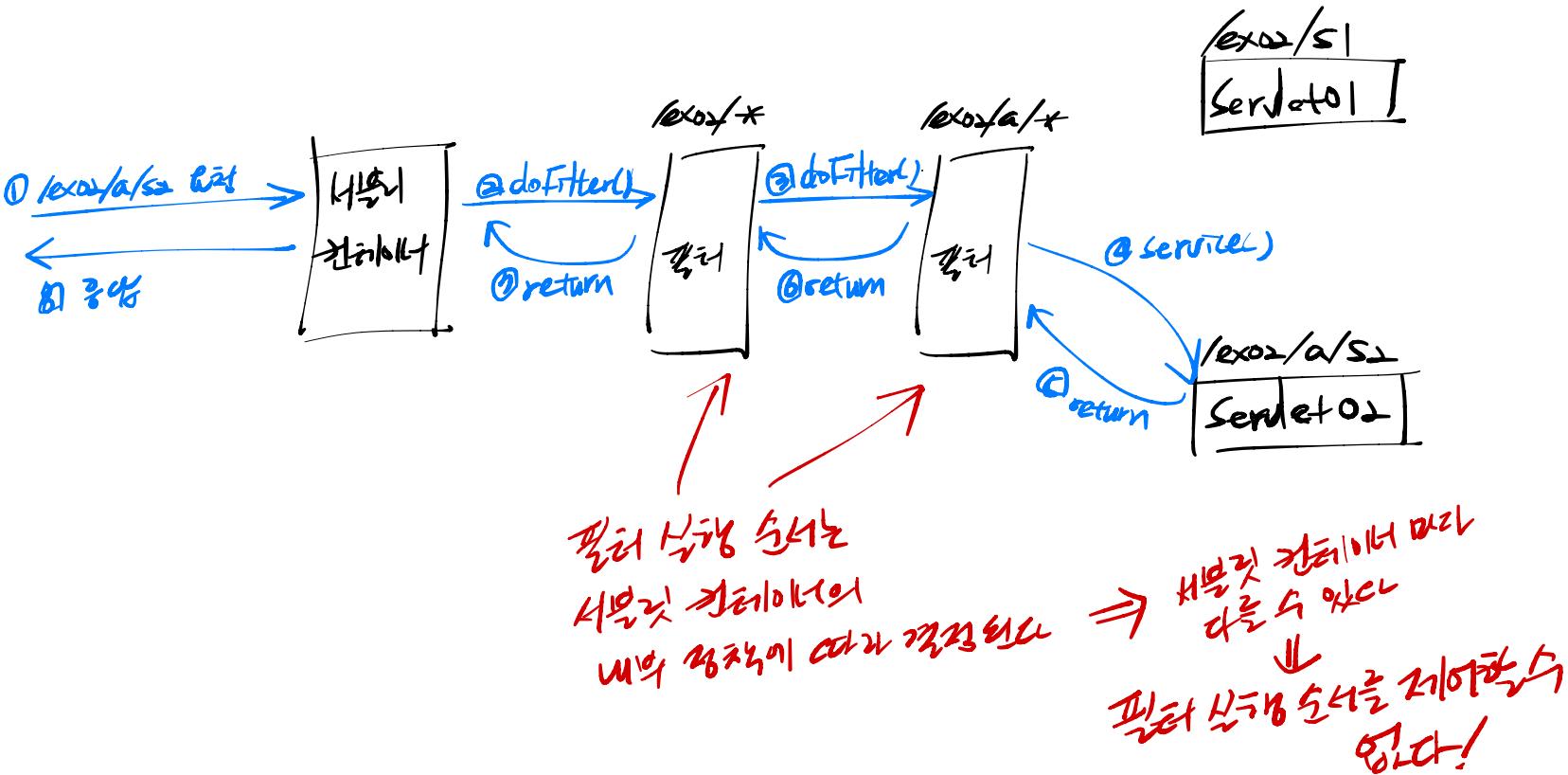


* 페리 헬즈 a)

lex02/s1 헬즈

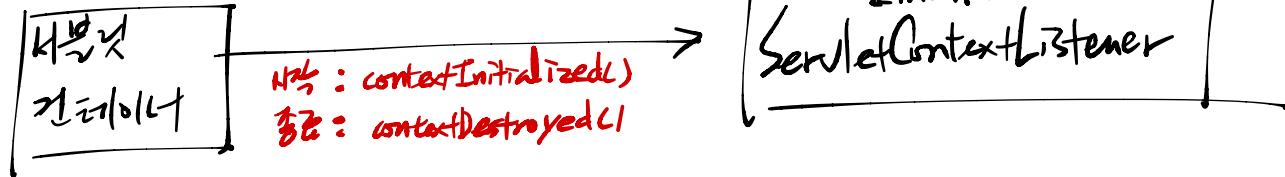


* 필터링 ①) /ex02/a/s2 페징



* 3가지 종류

① 컨테이너

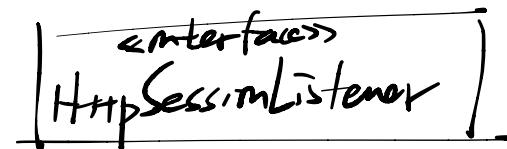


컨테이너 → `requestInitialized()`

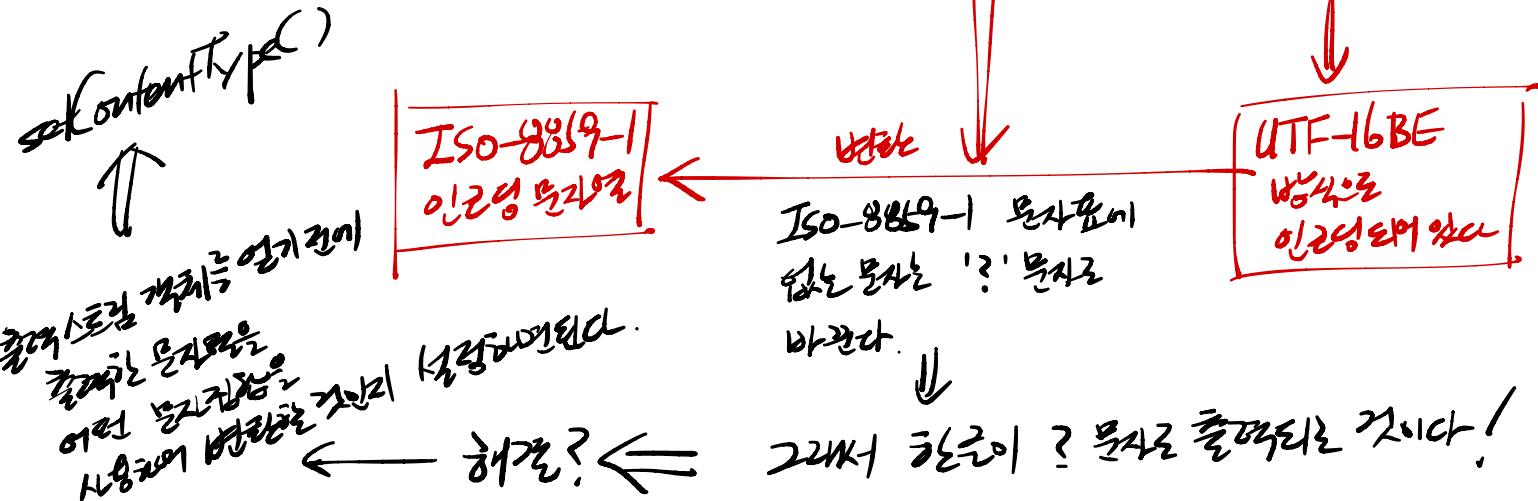
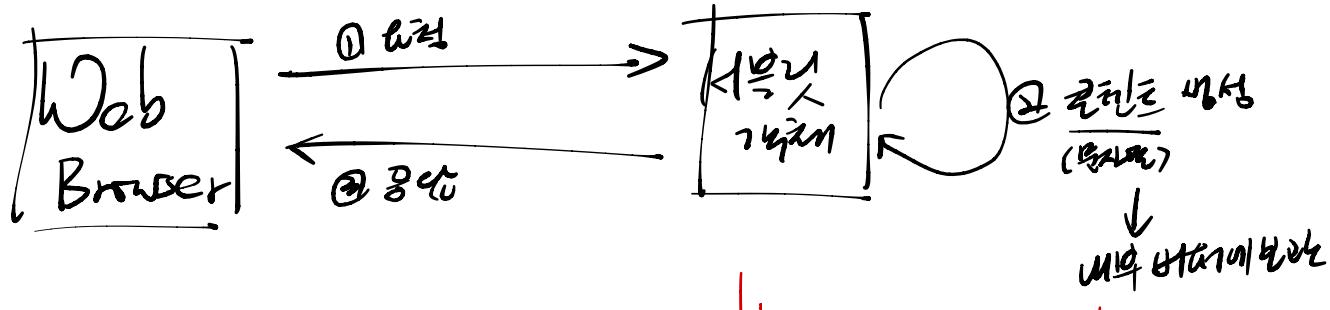
컨테이너 → `requestDestroyed()`

서블릿 컨텍스트 → `sessionCreated()`

서블릿 컨텍스트 → `sessionDestroyed()`



* 문자열 출력하고 읽을 때 깨짐



* 한글이 흐赡한 문자열에서 한글이 깨지는 예

한글

"ABC??"



442433F3F

한글

한글

"ABC가?"



004100420043 A00A01 (UTF-16BE)

ISO-8859-1 문자셋 (256자)

A → 41
B → 42

a → 61
b → 62

⋮
o → 30
i → 31

⋮

* UTF-16BE vs UTF-16LE
" "(UTF-16)

'f' 0xA00 0x0A0
'A' 0x0041 0x4100

* **내부기능**이 **특정한 동작방법**과 **방법**의 **개념**을 예 \Rightarrow **내부기능**

월정으로 놀러온다

"ABC→P_L"

1

4443 EAB080 EAB081

제작자와 내용
 $\frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$

서별기

"ABC가족"

11

004|004|0043 A00 A01 (UTF-16BE)

UTF-8 한글판

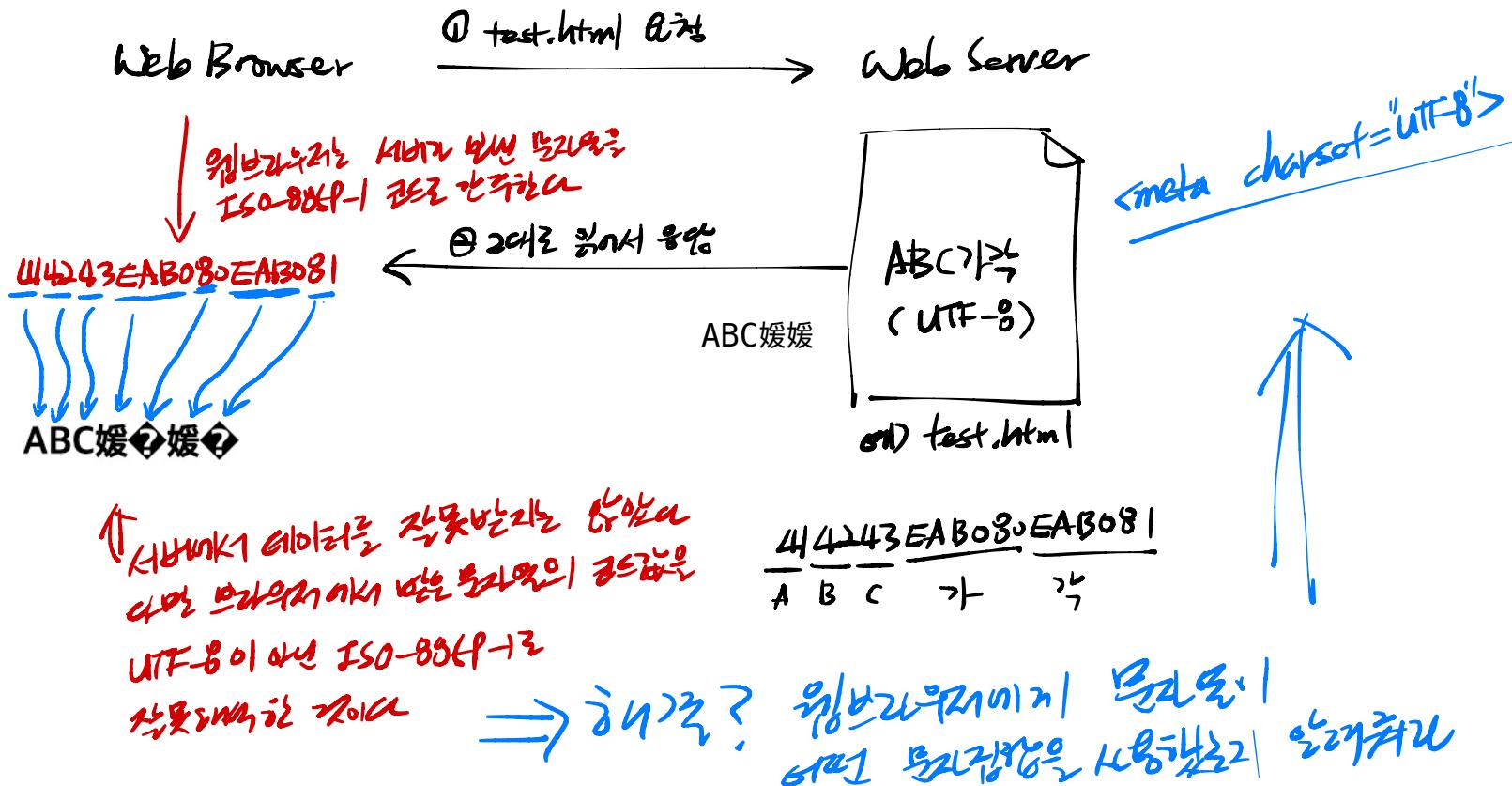
$$\begin{array}{ccc} A & \longrightarrow & 41 \\ B & \longrightarrow & 42 \end{array}$$

1

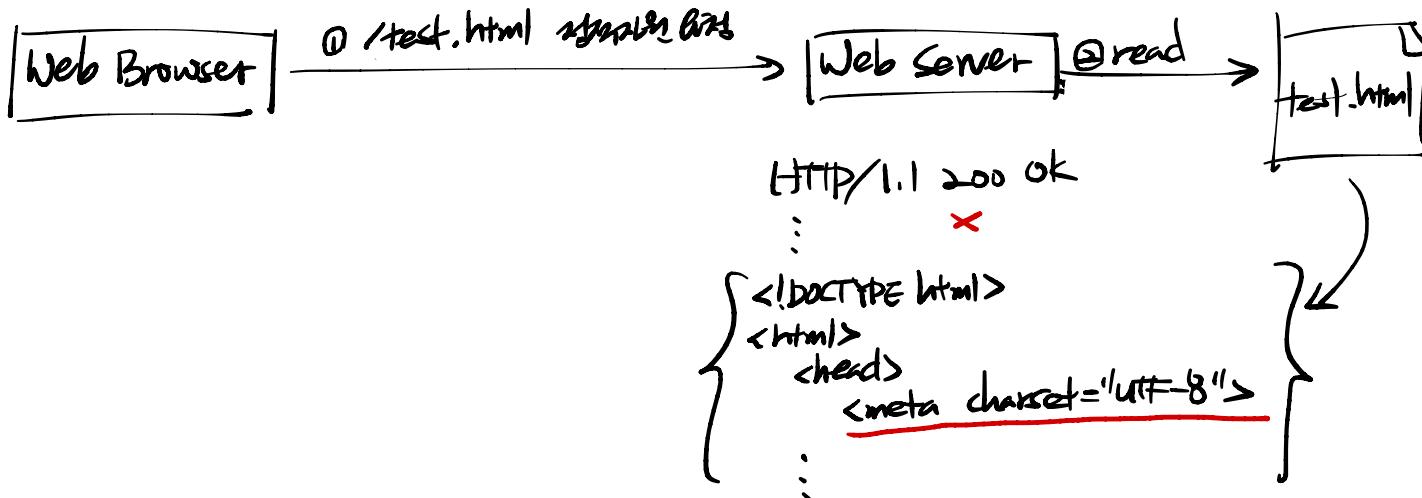
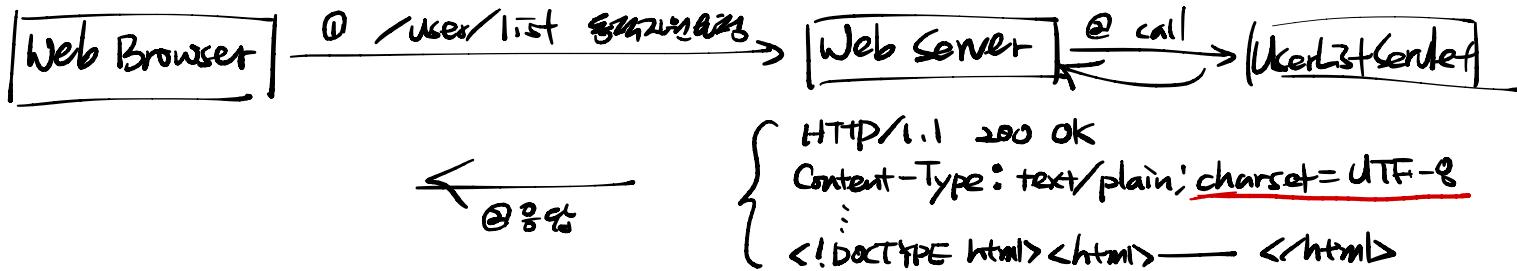
$$\gamma_L \rightarrow EABO8^\circ$$

$\xrightarrow{\text{L}}$ → EAB081

* HTML 문서의 한글 깨짐



* សេវាទូរសព្ទ



* setContentType()

Multi-purpose
Internet
Mail
Extension)) **인터넷의
인증기능** ⇒ 이미지로 전송하는 컨텐츠가
이면 신뢰성 있는지 알기하기 위해
인증 필요 → 자료는 네트워크로
전송되는 컨텐츠가
제작자는 자작권을 보호
하거나 사용자에게 내용을
알려-

`setContentType("MIME 타입"; "문자열");`
[문자열의 인터넷 표현]

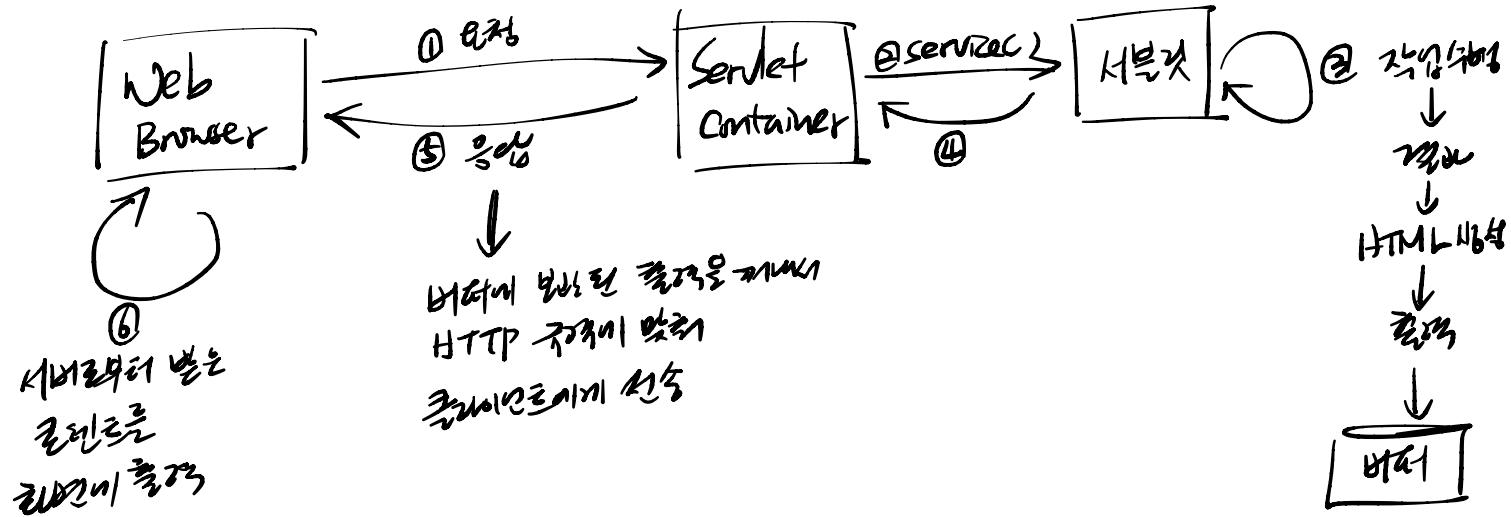
MIME Type 형식

type / sub-type ; zur Zeit = 10

(a) "text/plain; charset=UTF-8"

UTF-16BE $\xrightarrow{\text{映射}}$ ~~ISO-8859-1~~
UTF-8

* Web Browser et HTML
 ↳ 웹 클라우드를 제공하는 웹서버 = 브라우저 + 웹서버 + 서버 + DB



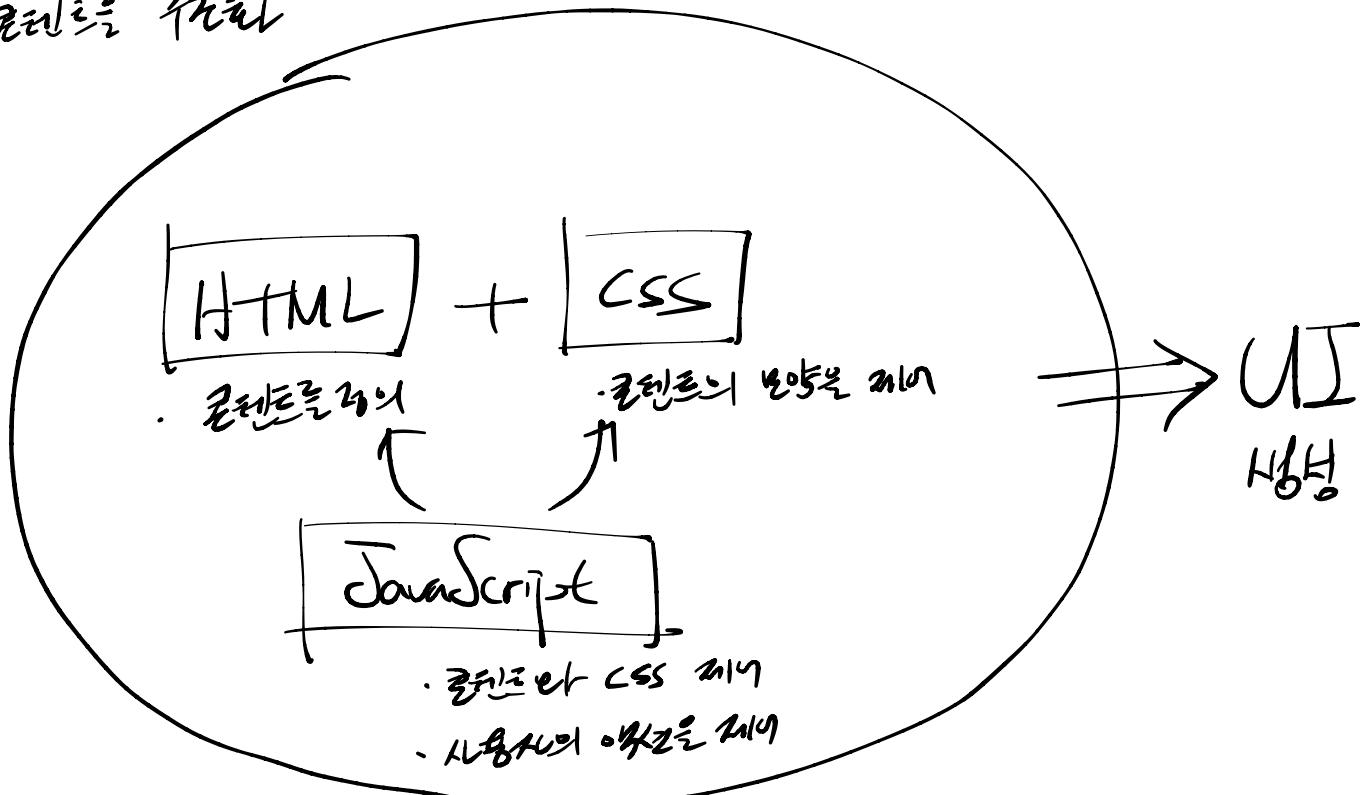
text/plain → 2013 323

text/html → HTML 렌더링

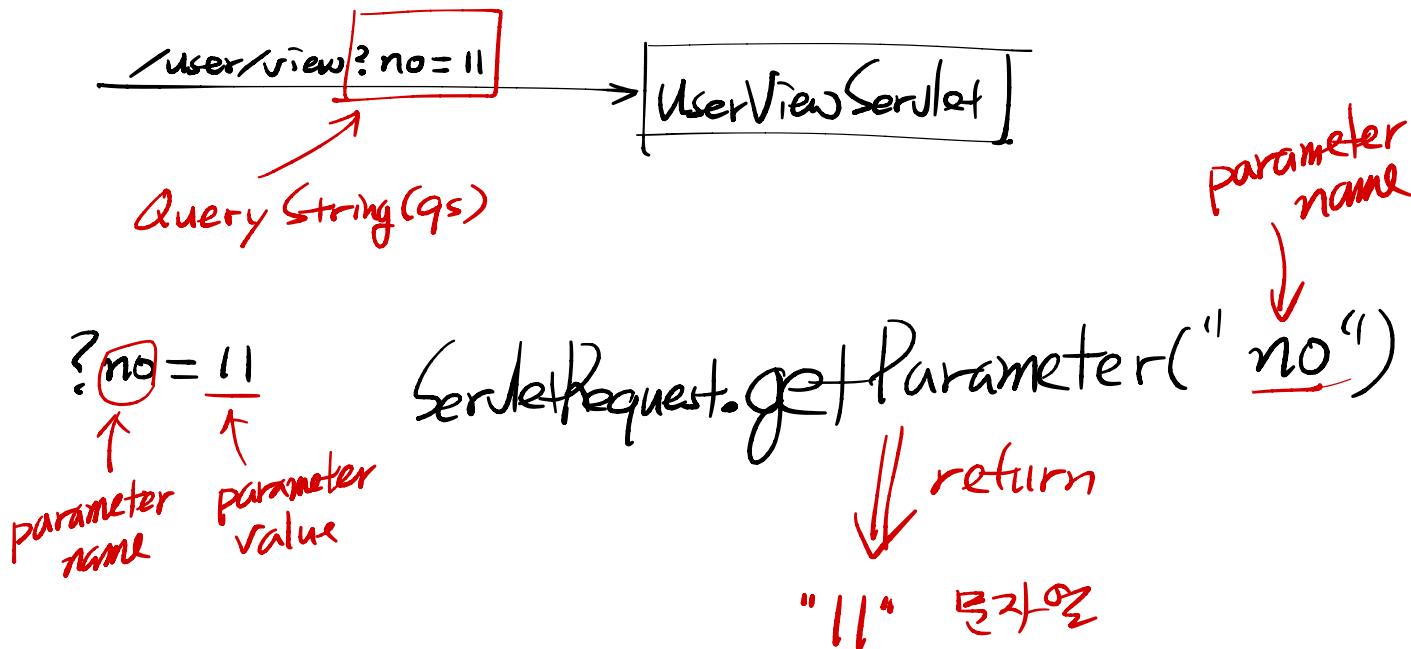
기타 → 다운로드 및 출력

* HTML (Hyper-Text Markup Language)

↳ 원래는 주제



* HTTP 요청 처리 과정 - URL이 포함된 문자열 \Rightarrow GET 요청



* HTTP 프로토콜에서 기본적인 데이터 전송 방식: GET 요청 & POST 요청

② POST 요청

method Request-URI HTTP Version

POST /ex04/s1 HTTP/1.1

User-Agent: PostmanRuntime/7.41.2

Accept: */*

Cache-Control: no-cache

Postman-Token: 658b66bd-9c7a-4b12-8ae9-5a603aa74bbd

Host: localhost:8888

Accept-Encoding: gzip, deflate, br

Connection: keep-alive

Content-Type: application/x-www-form-urlencoded

Content-Length: 33

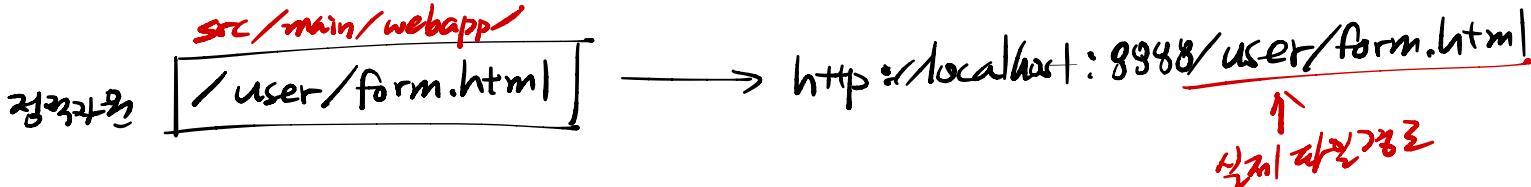
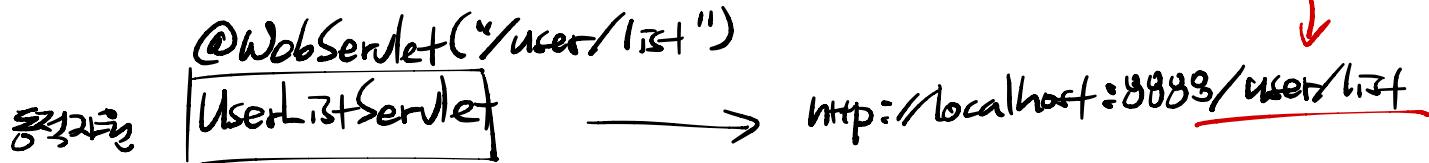
name=%EA%B0%80%EA%B0%81ABC&age=30

* URL (Uniform Resource Locator)

Identifier
URI

- URL ex) `http://www.google.com`
- URN ex) `urn:isbn:123456`

统一资源定位符
全局唯一标识符



* URL Encoding = Percent(%) Encoding

↪ URL 을 작성할 때 URL에서 특별한 의미를 지니는 문자를 URL에서 사용할 수 있게
기호에 따라 변환해야 한다.

RFC-3986에 의해 URL을 작성
(기준)

일부 문자를 특별한
의미로 사용하는
기호

Reserved keyword → URL에서 특별한 의미로
사용된다.

!	%21	;	%3B
#	%23	=	%3D
\$	%24	?	%3F
&	%26	@	%40
,	%2C	[%5B
)	%29]	%5D
*	%2A		
+	%2B		
,	%2C		
/	%2F		
:	%3A		

Unreserved characters

A-Z a-z 0-9 - . ~

한글 ?
한국어 → %xx / %xx

"ABC가?" → ABC%EA%BD%80%CA
EAB080 EAB081
%BD%81

* گزینه هایی که ممکن است

① GET طبقه

http://localhost:8888/user/add?name=ABC&email=bbb%40test.com&password=&tel=ddd

Query String

ServletRequest.getParameter("name")

"ABC"

② POST طبقه

:

Content-Type: application/x-www-form-urlencoded

Content-Length: 33

name=%EA%B0%80%EA%B0%81ABC&age=30

URL decode
کارهای انجام شده

* 클라이언트가 보낸 한글 데이터를 읽을 때 깨지는 이유

클라이언트

name=ABC%EA%BO%80%EA%BO%81

HTTP
↓
41 42 43 EA B0 80 EA B0 81

%XX 문자를 XX 문자로 변환 = URL 디코딩(decoding)
서버에서 자동으로 수행된다

getParameter("name")

UTF-8 → UTF-16

String
(UTF-16)

ISO-8859-1 (default) → UTF-16

GET
요청

41 → 0041 A

42 → 0042 B

43 → 0043 C

EA B0 80 → A C 00 가

EA B0 81 → A C 01 는

↑
클라이언트가 보낸 문자열
UTF-8 이라고 간주하고
UTF-16로 변환

POST

한국

41 → 0041 A

42 → 0042 B

43 → 0043 C

EA → 00 EA è

B0 → 00 B0 o

80 → 00 80 .

EA → 00 EA è

B0 → 00 B0 o

81 → 00 81 .

클라이언트가 보낸 데이터를
ISO-8859-1로 인식한다.

* GET 요청 vs POST 요청

	GET	POST
데이터 전송 방식	URL에 포함 (Query String)	Message Body
전송 데이터 크기	일반적인 범위는 URL 크기를 제한한다. 작은 데이터 전송 ① 전송의 범위, 제한	제한된 규칙 → 데이터 전송 ② 제한, 이동
데이터 전송 데이터 형식	URL은 텍스트로 → 전송 불가 즉, 데이터를 데이터를 텍스트로 변환 한 경우 가능 ③ Base64	application/x-www-form-urlencoded ⇒ 불가! multipart/form-data ⇒ 가능
보안	URL은 일반적으로 cache된다 다른 사용자에게 노출될 수 있다 ✓ 보호되어야 하는 정보는 제외. ✓ URL에 데이터를 포함하는 경우는 절대	<ul style="list-style-type: none"> - 암호화되어야 함 - 비밀번호나 계정번호 제외 - (보호되어야 하는) 중요한 데이터 제외 (문서가 있는) 문서내용

* 파일 업로드 (multipart/form-data)

POST /ex04/s3 HTTP/1.1
User-Agent: PostmanRuntime/7.41.2
Accept: */*
Cache-Control: no-cache
Postman-Token: f4f27045-01b8-49e7-b354-9b644ea9e831
Host: localhost:8888
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Type: multipart/form-data; boundary=-----509330874477515409039305
Content-Length: 26069

-----509330874477515409039305
Content-Disposition: form-data; name="name"
ABC가각
-----509330874477515409039305
Content-Disposition: form-data; name="age"
20
-----509330874477515409039305
Content-Disposition: form-data; name="photo"; filename="test.jpg"
비아니리 파일 데이터....
-----509330874477515409039305

getParameters("name")
null

] part ← 일반 데이터
] part ← 일반 데이터

] part ← 파일 데이터

↓
HTTP://www.daum.net의 글을 봐

multipart/form-data 형식으로 전송된 데이터는
여기서 파일은 not null이 아님.