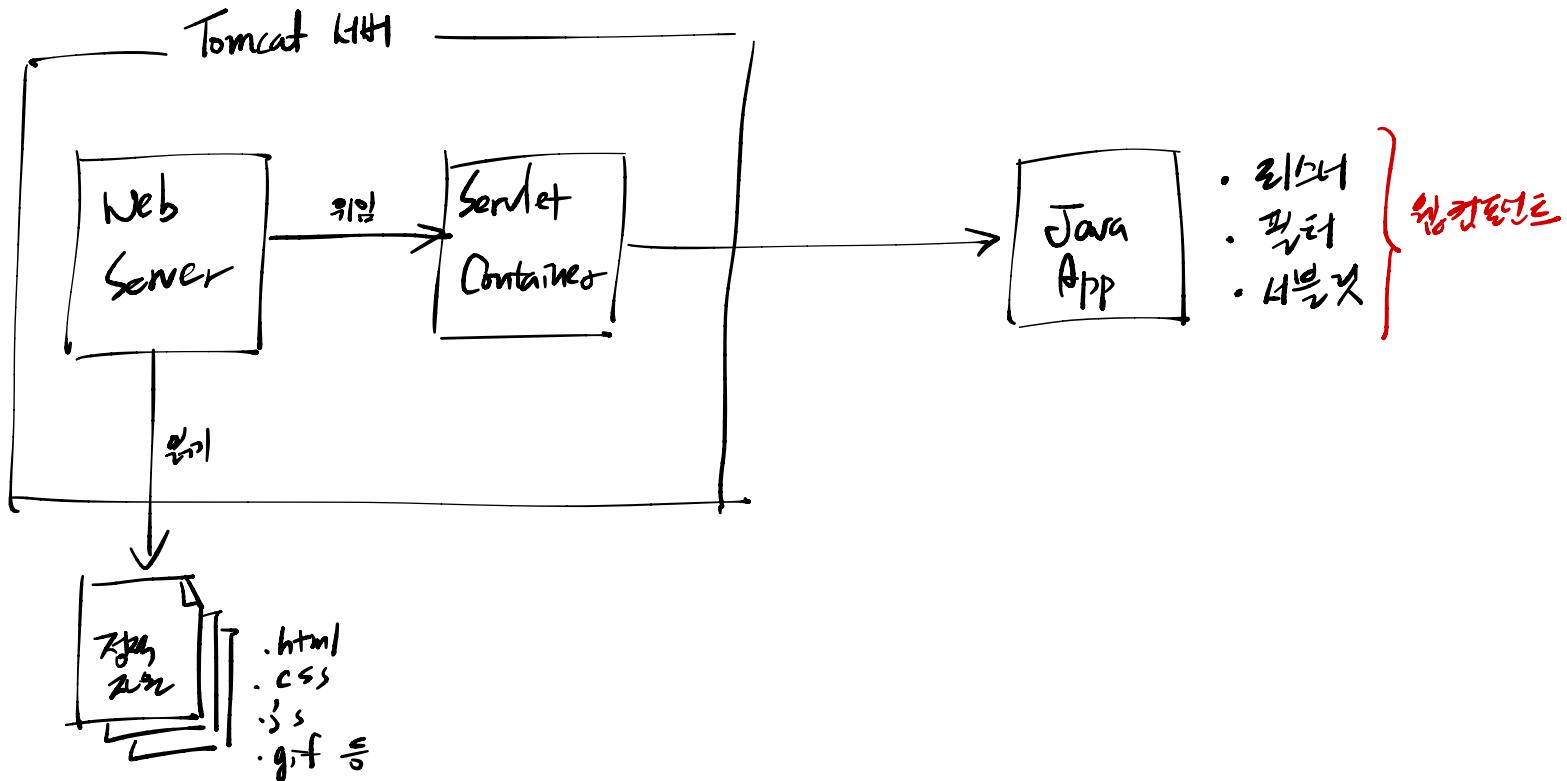


\* 웹 컨테이너(부록)  
↳ 한 개 이상의 기능으로 구성되어 특별한 권한을 부여받을 것. = 특별한 역할은 부여받는 기능



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

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

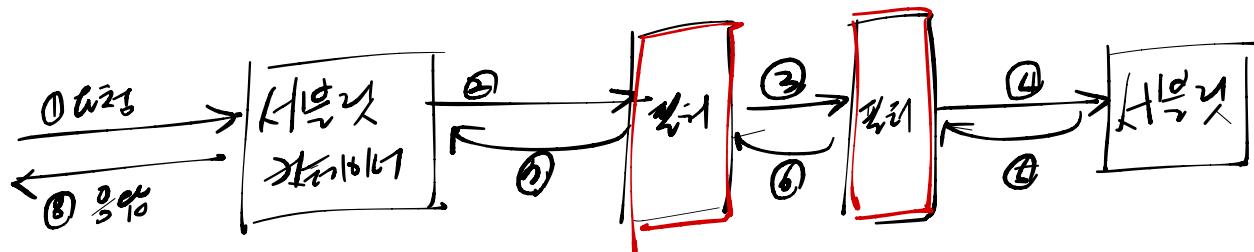
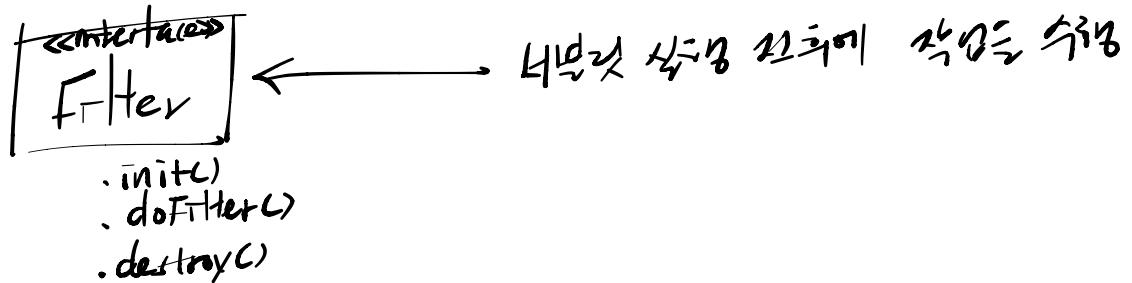
ServletContextListener ← 서버가 컨텍스트가 초기화되거나 종료될 때 호출되는  
contextInitialized() contextDestroyed()

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

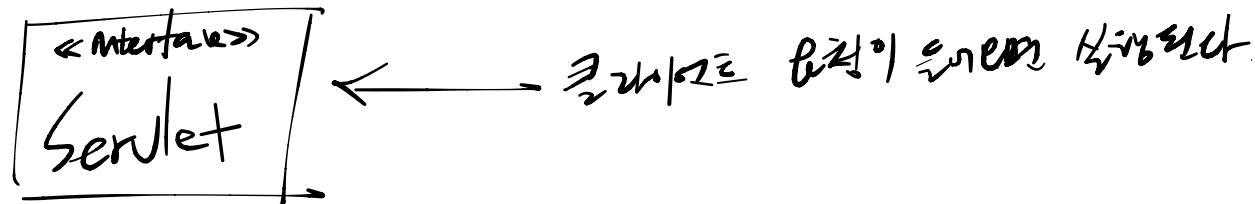
HttpSessionListener ← 클라이언트 세션이 생성되었거나 종료되었을 때  
sessionCreated() sessionDestroyed()

:

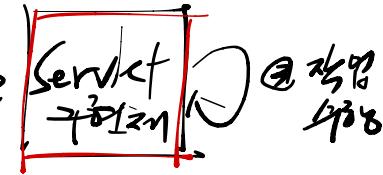
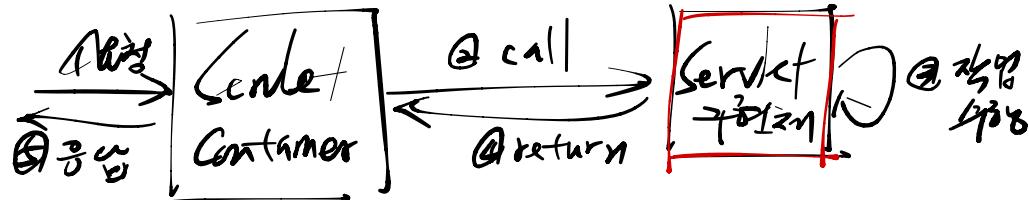
\* 필터링 책임 - 책임 = chain of Responsibility  
 GOF의 책임 체인



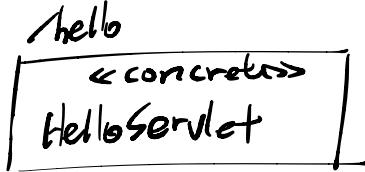
\* 웹 개발언어 - HTML = Command  
 GOF의 디자인 패턴



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



\* Servlet API

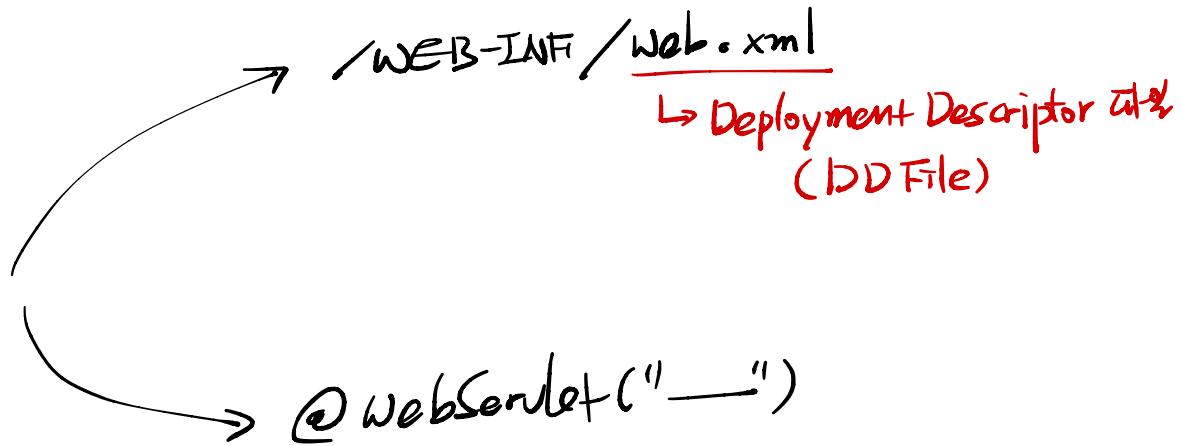


- `init()`
- `service()`
- `destroy()`
- `getServletInfo()`
- `getServletConfig()`

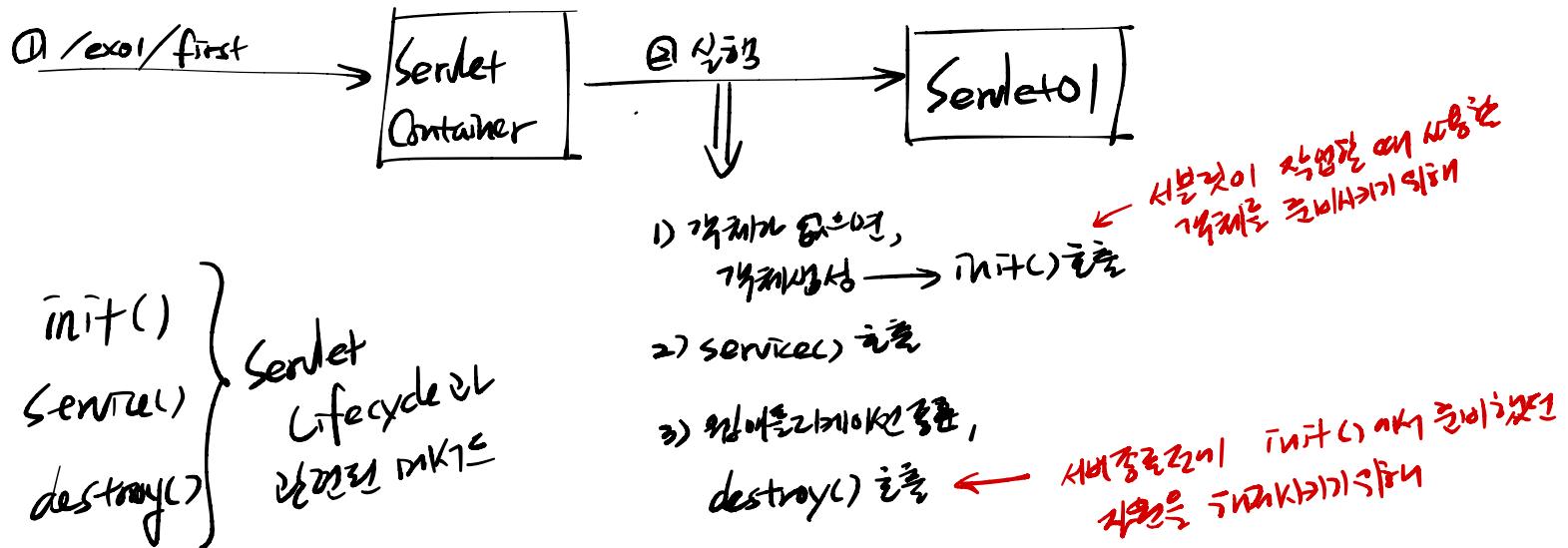
- `init() {} ->`
- `service() {} ->`
- `destroy() {} ->`
- `getServletInfo() {} ->`
- `getServletConfig() {} ->`

\* 웹 서비스  
|> JSP

Servlet 파일



## \* 서블릿의 생명주기



\* service() 데일리

작성

Tomcat 키트 : HttpServletRequest 향상 | HttpServletResponse 향상

↳ 흐름

↓

← 실제 애플리케이션 → ↓

service(ServletRequest req, ServletResponse resp) {}

↑

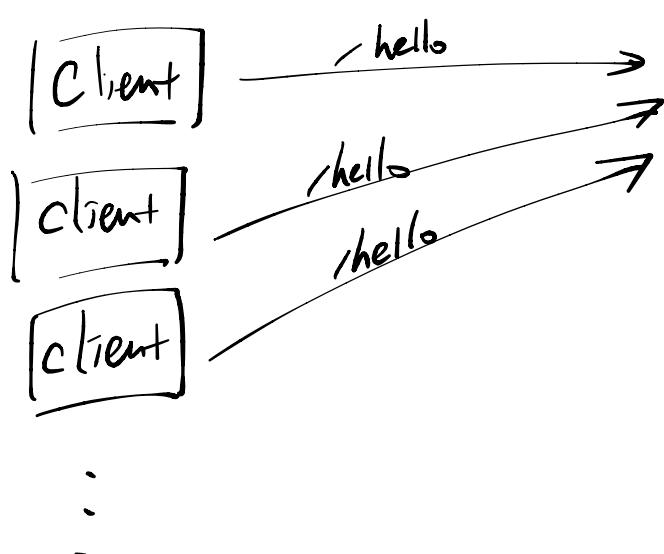
↑

✓ 헤더와 퍼티 설정

✓ 응답을 위한 HttpServletResponse

응답을 위한 HttpServletRequest

\* 클라이언트 요청과 서버의

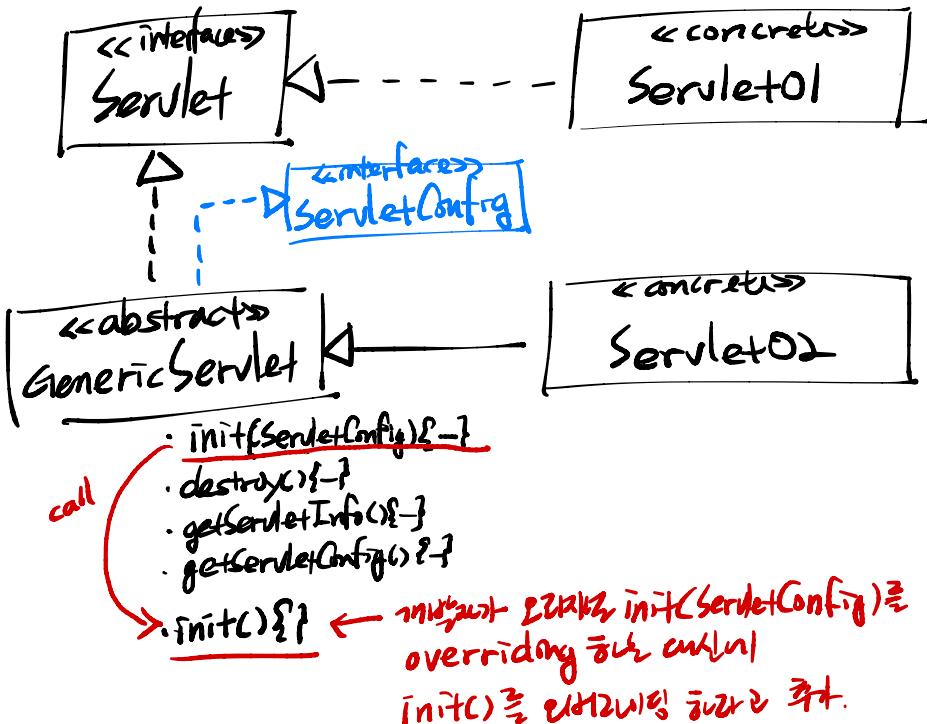


HelloServlet  
인스턴스

↑  
한 번의 가로는 여러 클라이언트가 동시에  
접속!

특정 클라이언트의 작업 결과는  
서버의 인스턴스 필드에 보관하고  
보내!

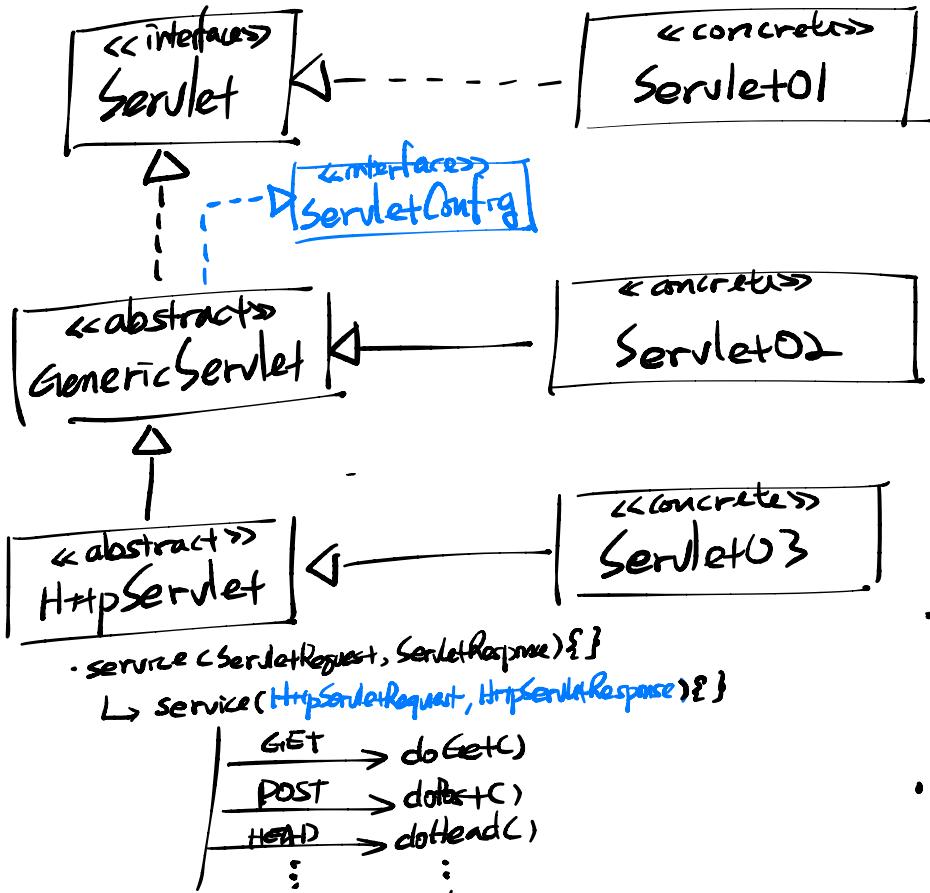
\* 亂世のアーキテクチャ II



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

service() { - }

\* 서블릿의 마법 | III



. init() {} — } \*

~~. service() {} — } \*~~

. destroy() {}

. getServletInfo() {} — }

. getServletConfig() {} — }

. service() {} — }

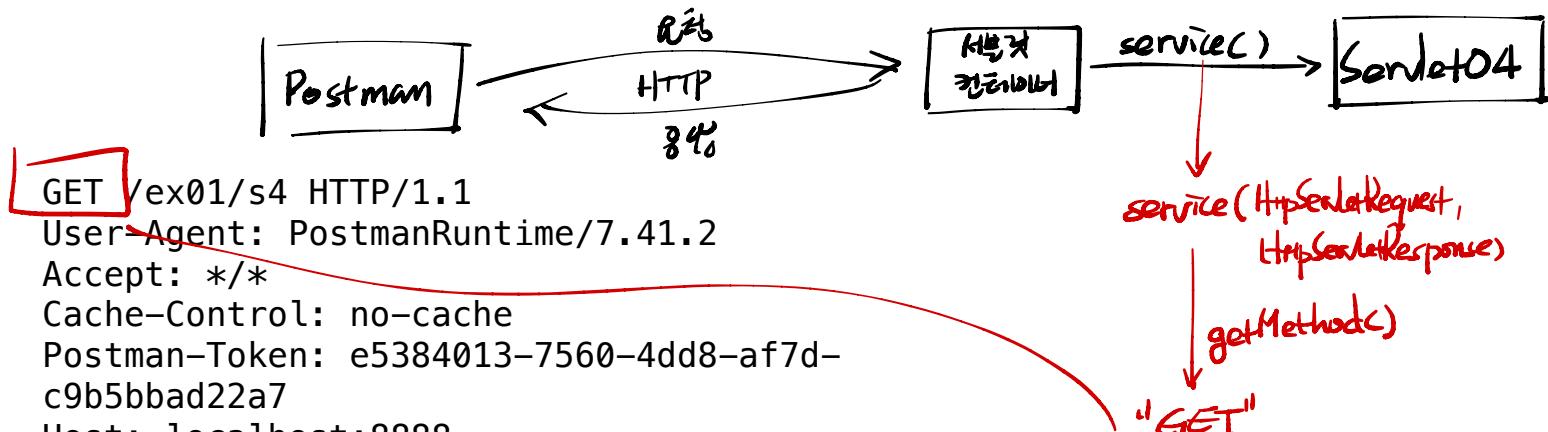
. GET 요청을 처리하는 경우,  
doGet() {} — }

. POST 요청을 처리하는 경우,  
doPost() {} — }

:  
⋮

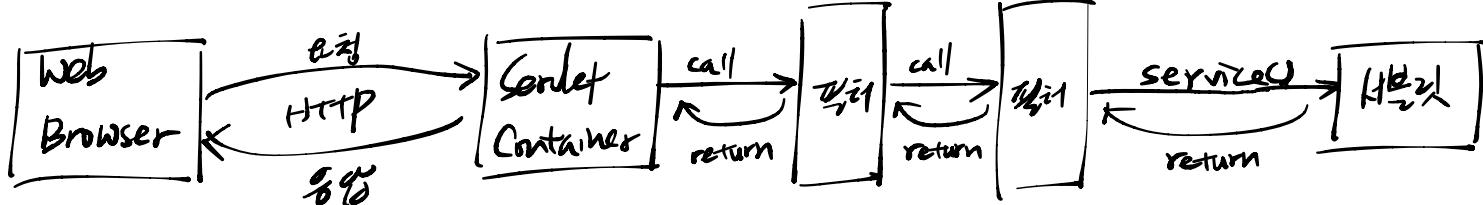
. 모든 요청을 처리하는 경우,  
service(HttpServletRequest, HttpServletResponse)

## \* Backend App init

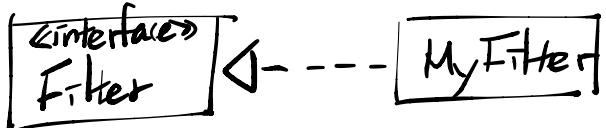


\* 필터 만들기

① 구조



② 구현



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

doFilter()

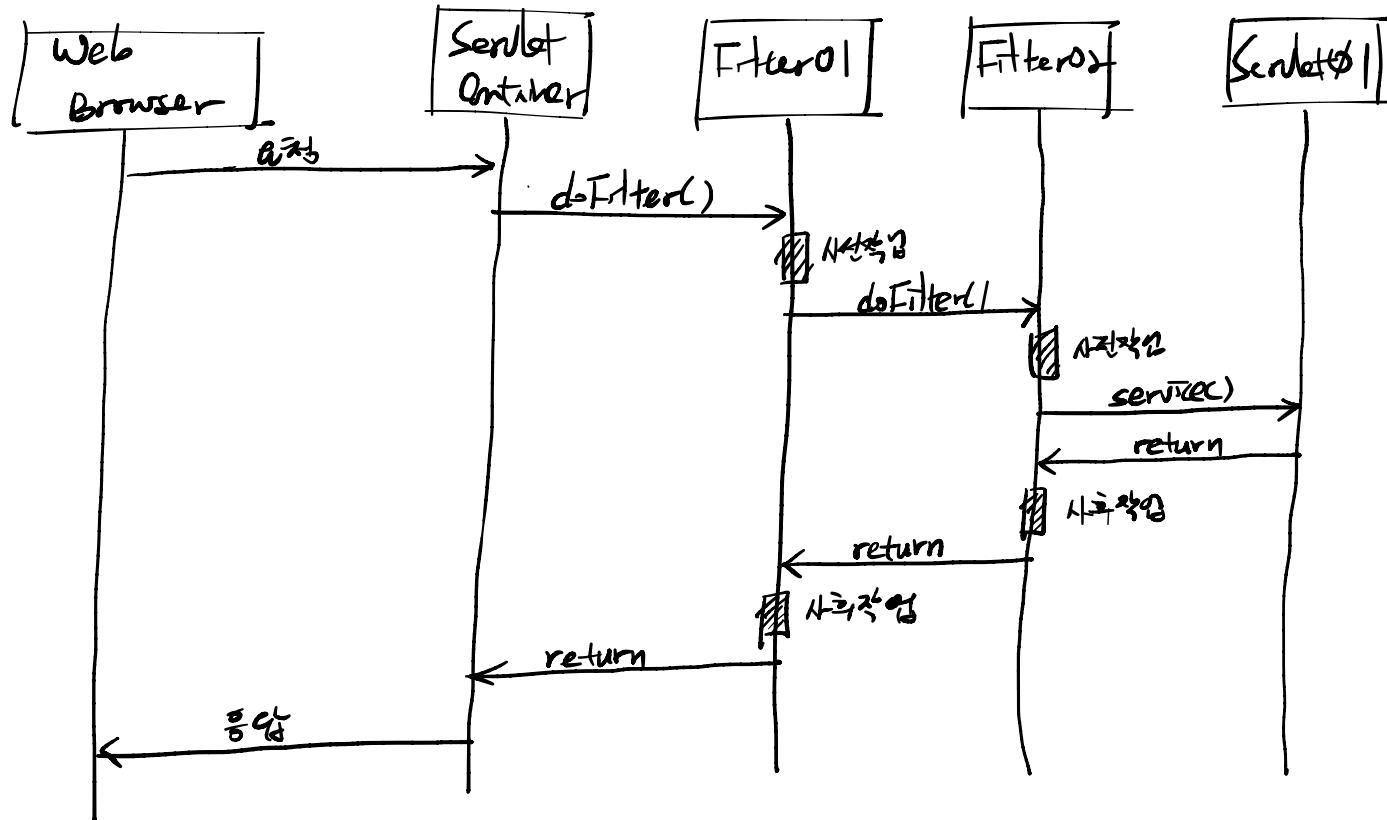
사전작업

다음 처리 또는 서블릿

사후작업

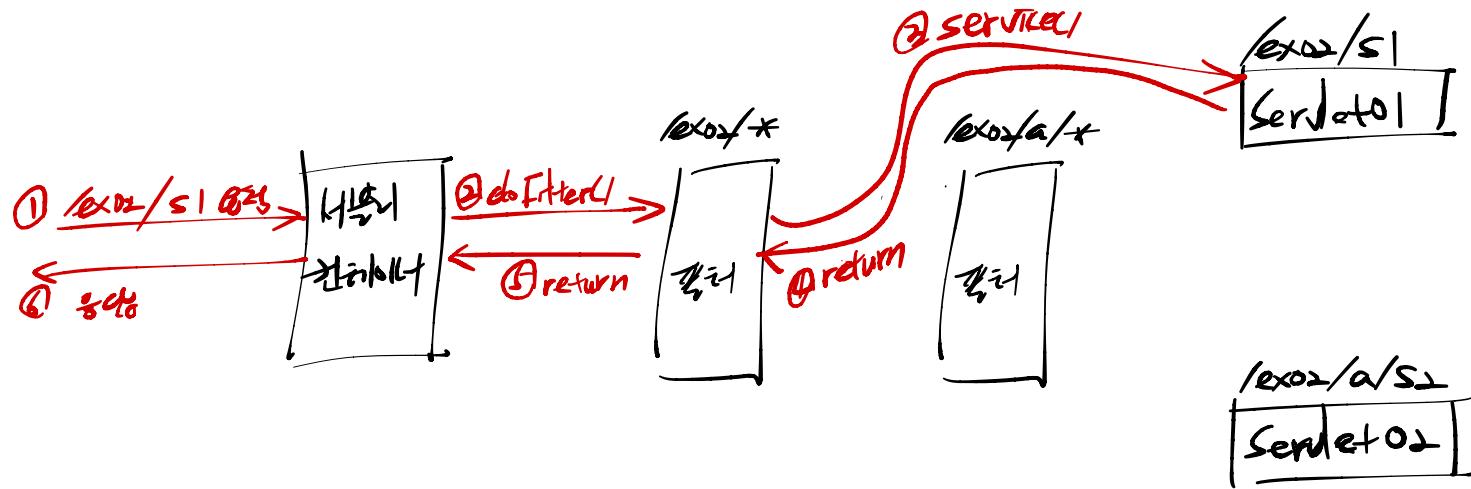
}

## \* 필터링 흐름

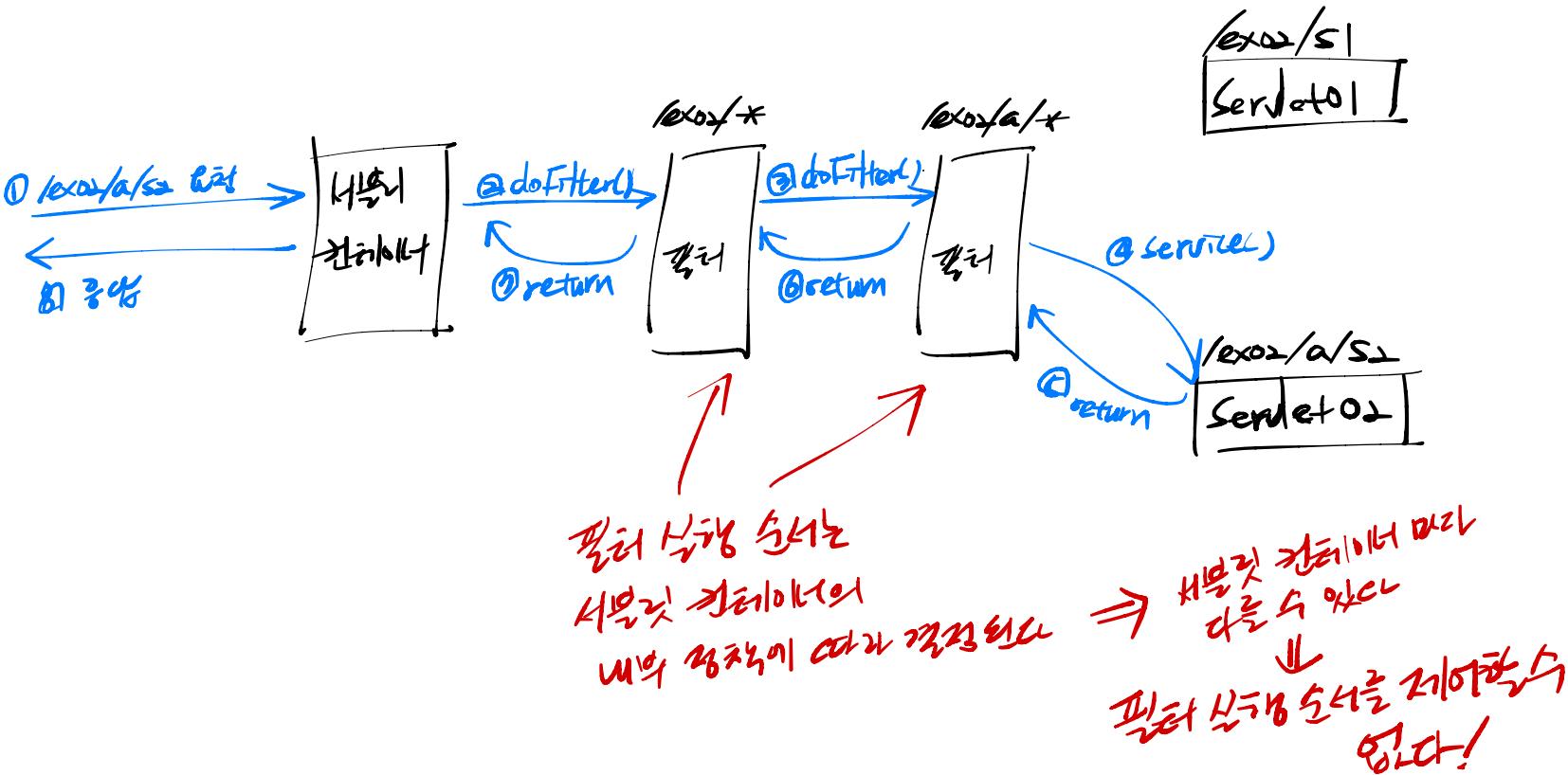


\* 페리 헬즈 a)

lex02/s1 헬즈

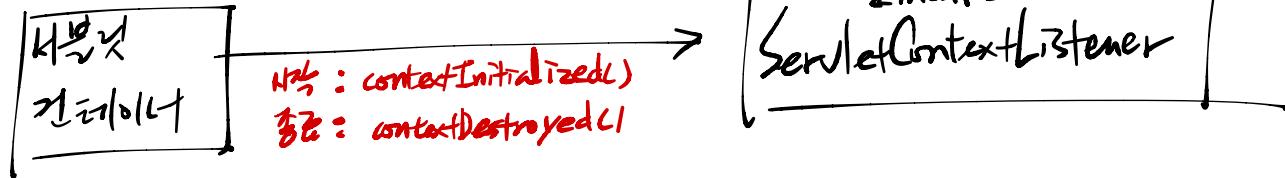


\* 필터링 ①) /ex02/a/s2 페징



\* 3가지 종류

① 컨테이너

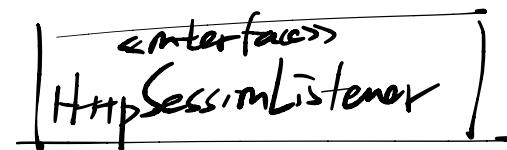


컨테이너 → `requestInitialized()`

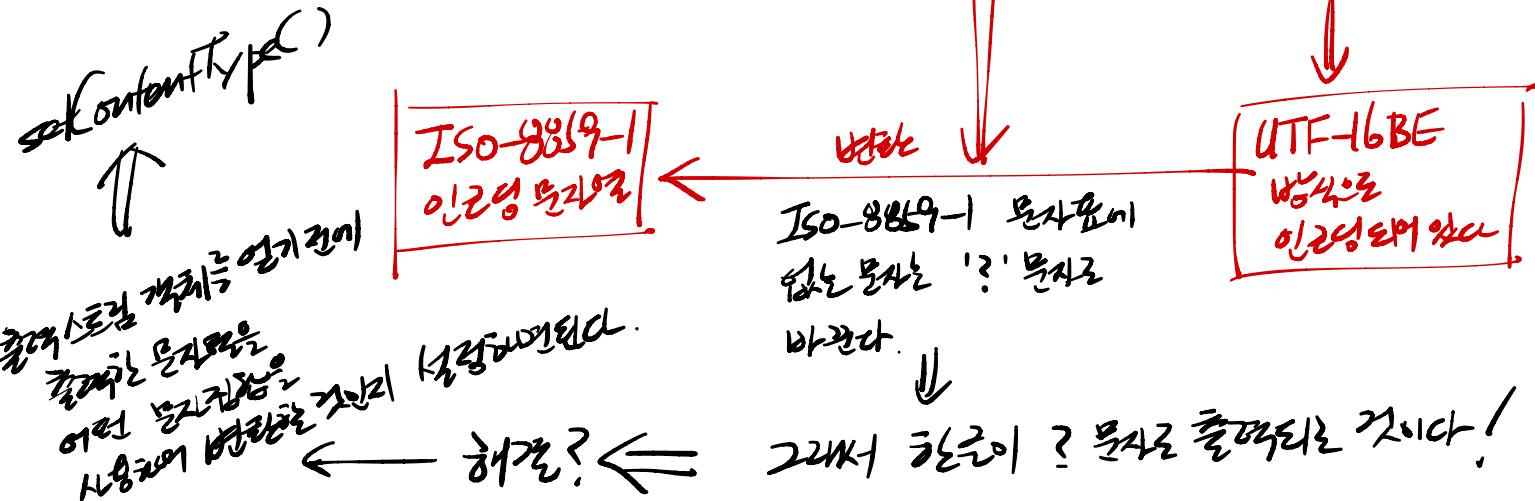
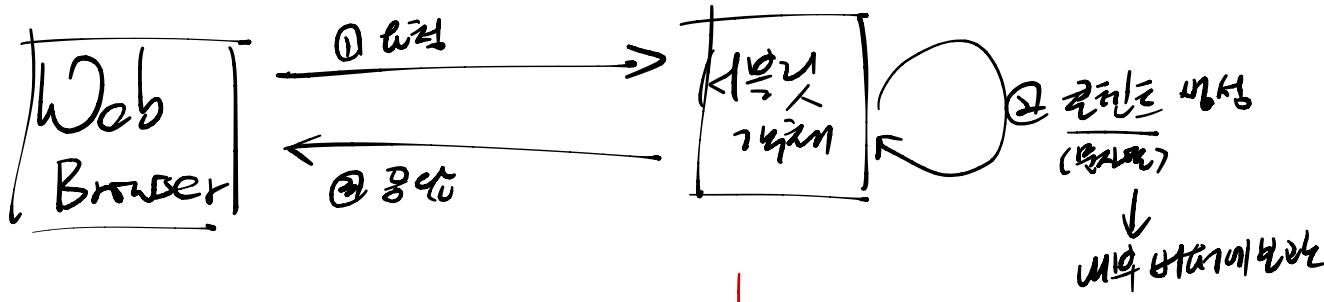
컨테이너 → `requestDestroyed()`

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 AC00AC01 (UTF-16BE)

ISO-8859-1 문자셋 (256자)

A → 41  
B → 42

a → 61  
b → 62

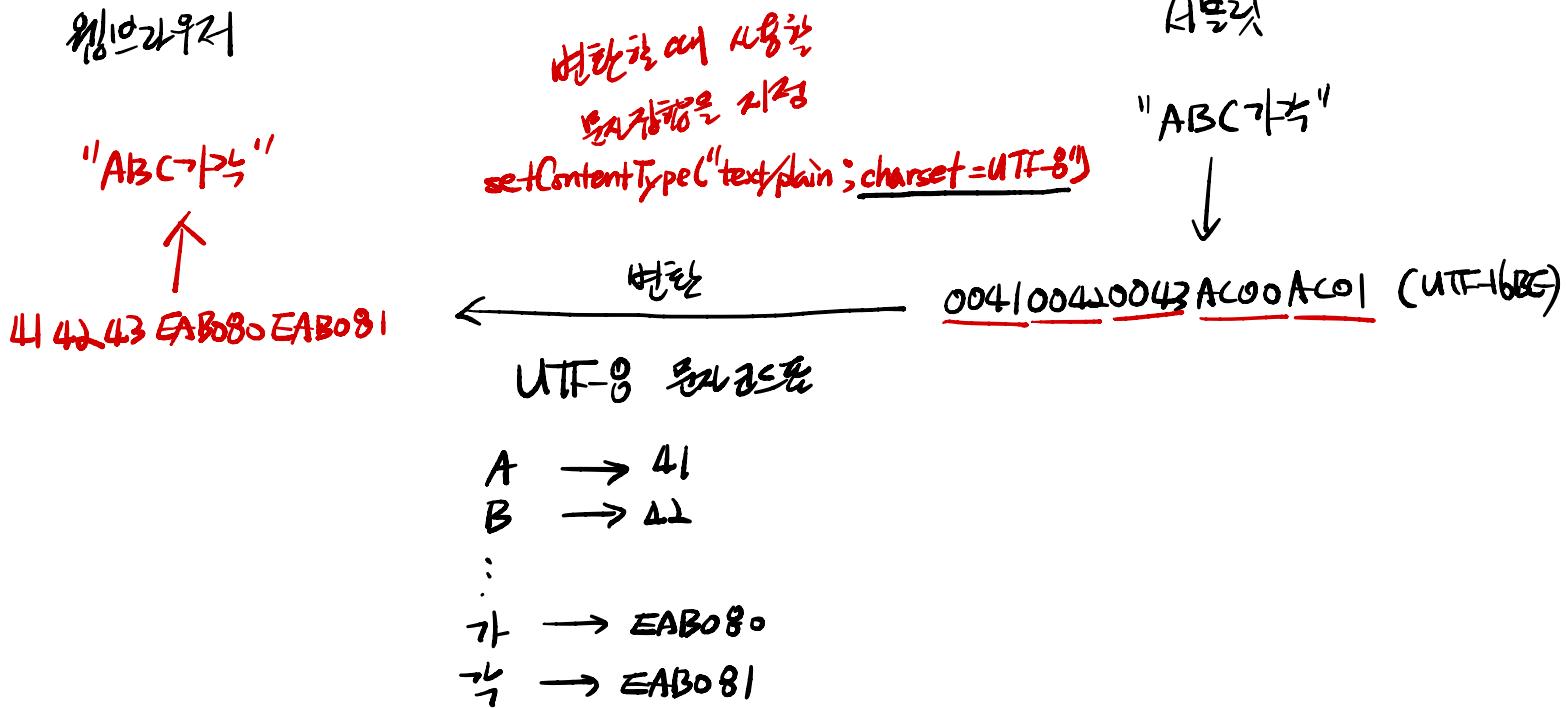
o → 30  
i → 31

:

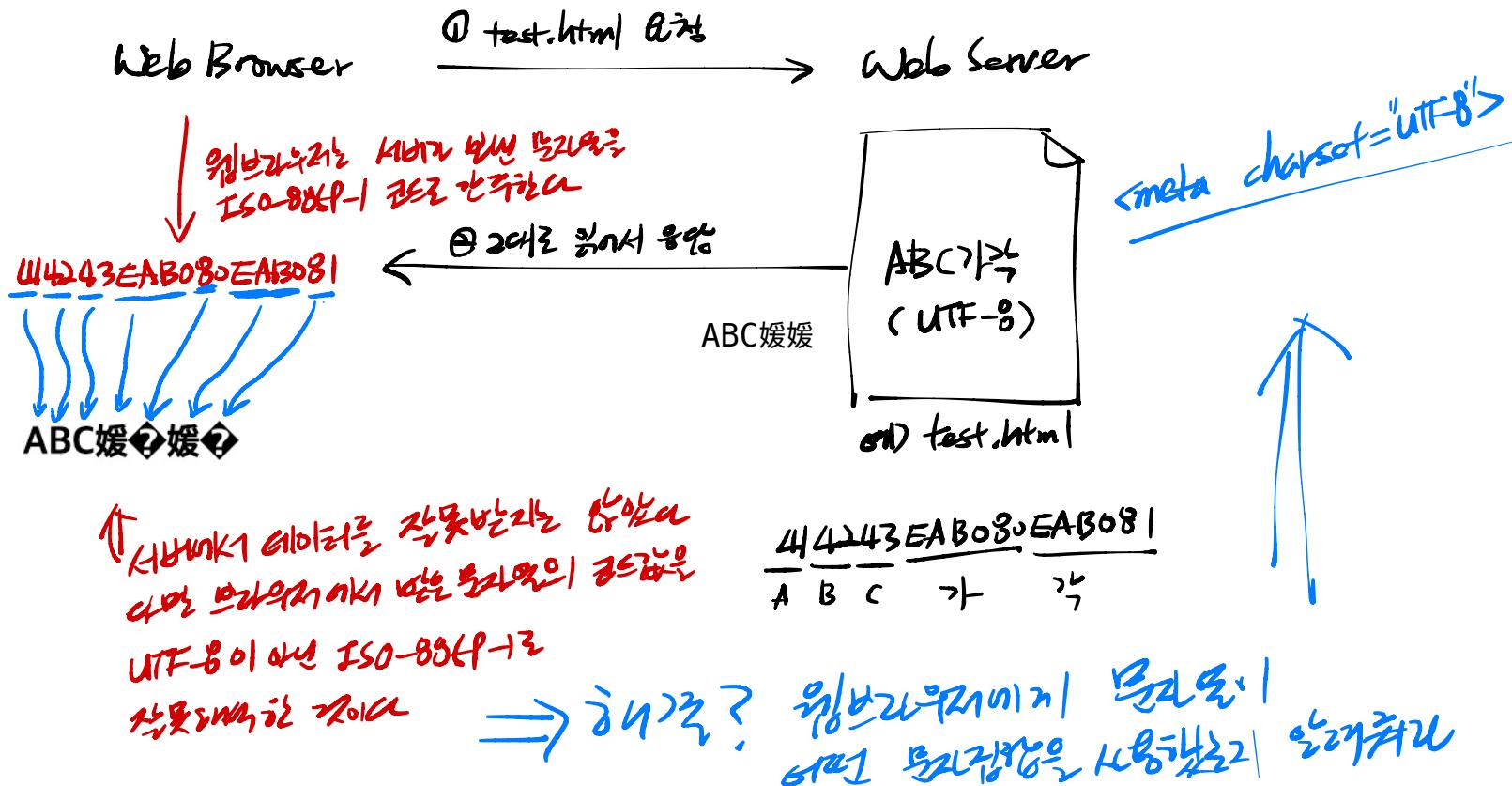
\* UTF-16BE vs UTF-16LE  
" (UTF-16)

'f' 0xAC00 0x00AC  
'A' 0x0041 0x4100

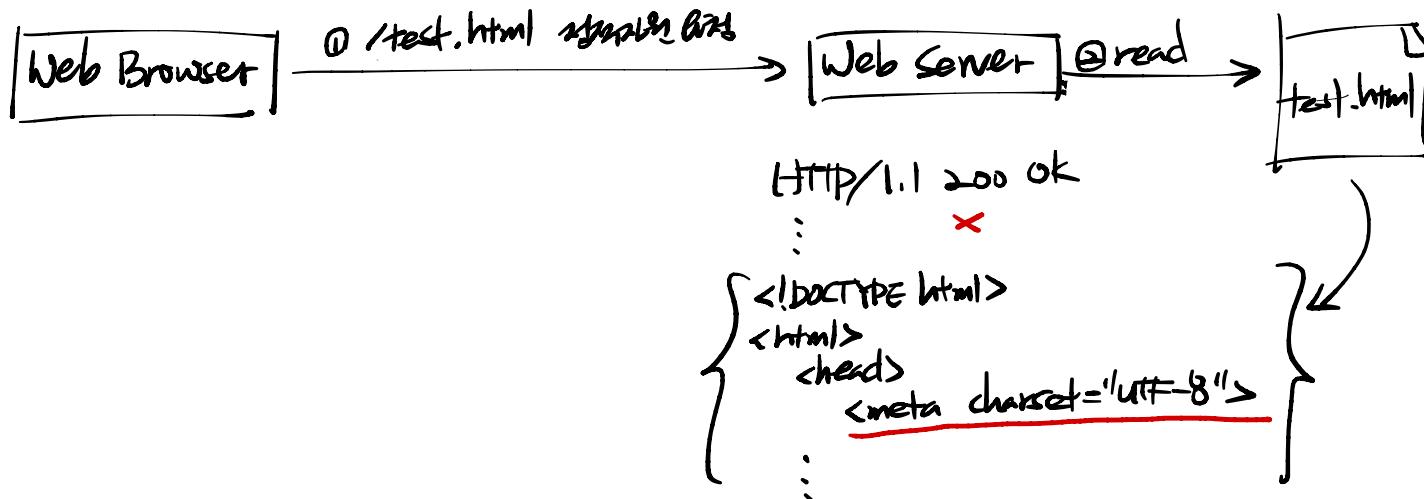
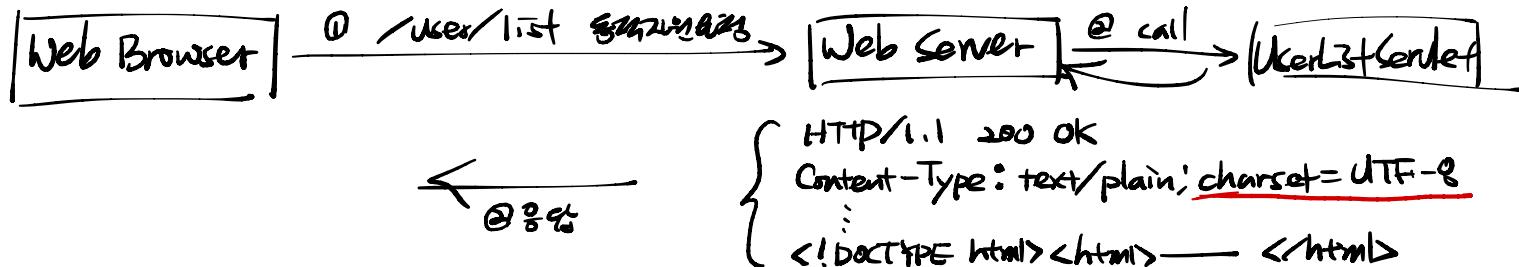
\* 한글기호가 출력되는 경우, HTML의 encoding이 UTF-8이어야 한다는 원칙에 맞는다.



## \* HTML 문자의 흐름 과정



\* សេវាទូរសព្ទ



\* setContenttype()

Multi-purpose  
Internet  
Mail  
Extension

MIME Type  
설정방법

이메일로 전송되는 경우로  
문서의 확장자와 일치하지 않아  
문서형식  
→ 파일은 미리워크로  
작은 파일로  
파일을 사용하는 경우  
문서형식을 사용하는 경우로  
선택.

setContenttype("MIME 타입; charset");

① MIME의 인터넷 기본

MIME Type 형식

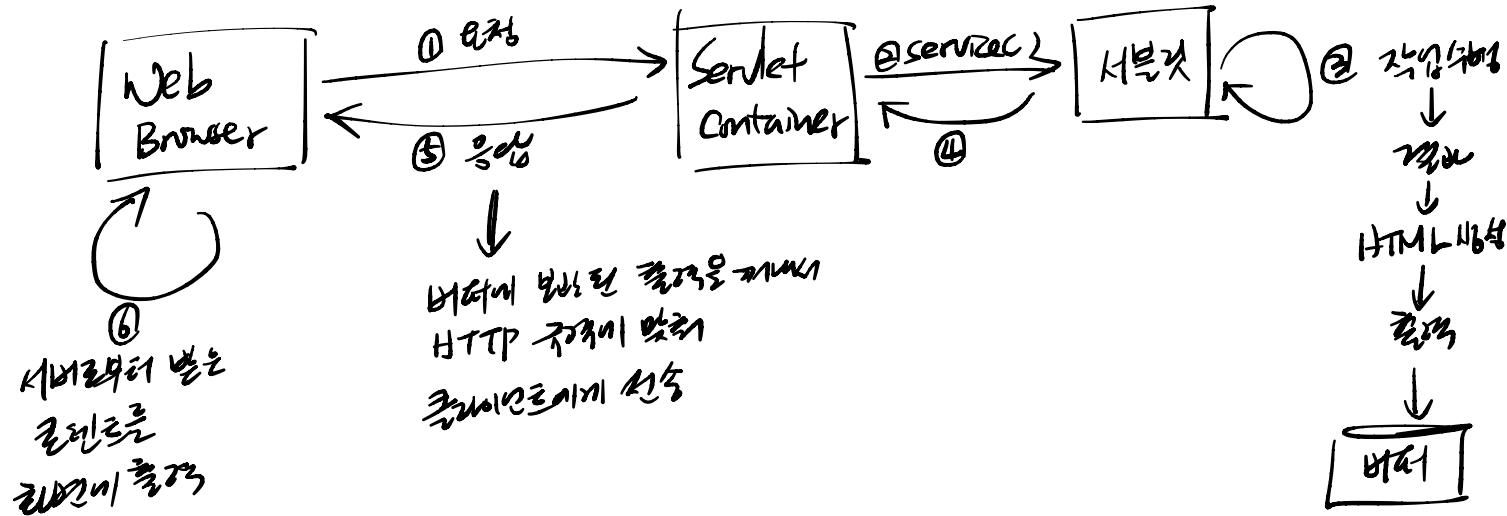
↓  
type/sub-type ; charset = 값

ex) "text/plain; charset=UTF-8"

↓

UTF-16BE → ~~ISO-8859-1~~  
UTF-8

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



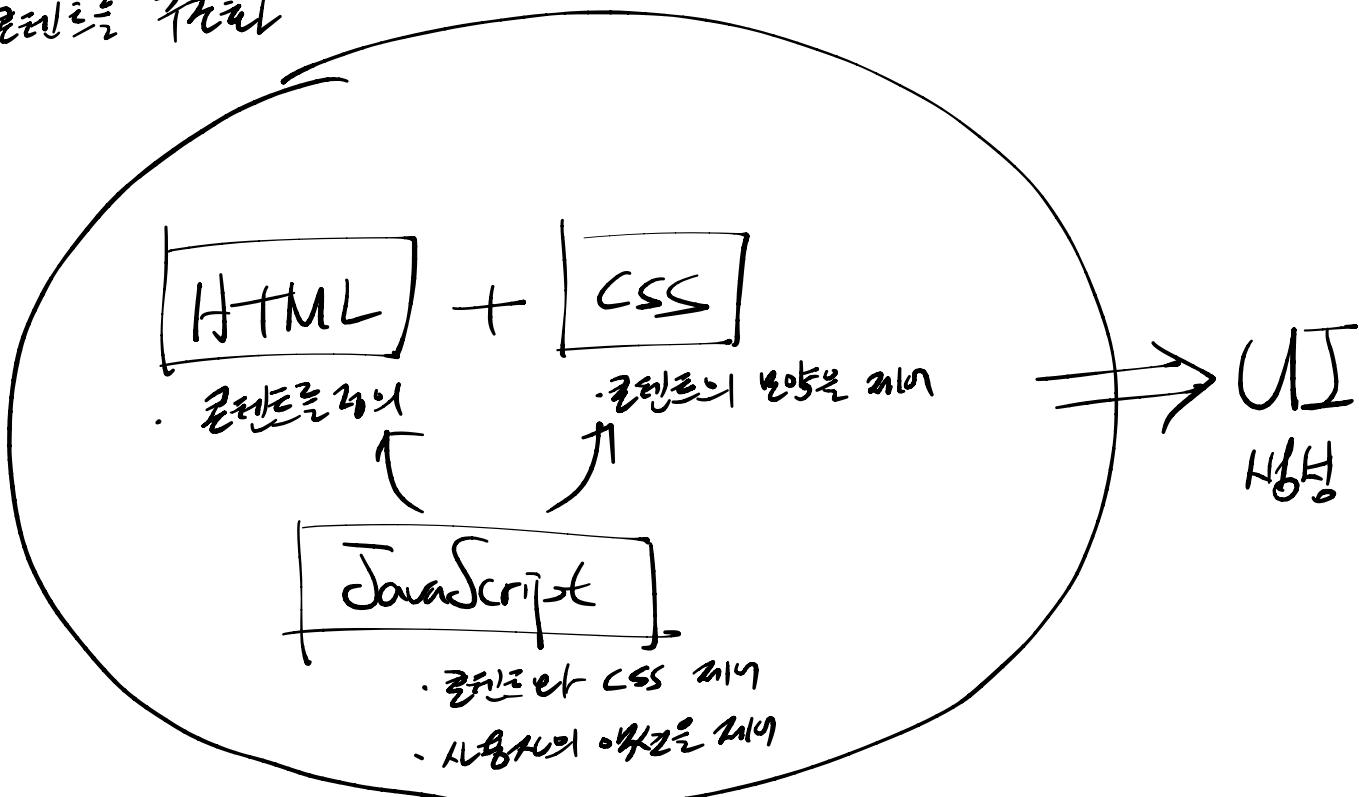
text/plain → 2013 323

text/html → HTML 렌더링

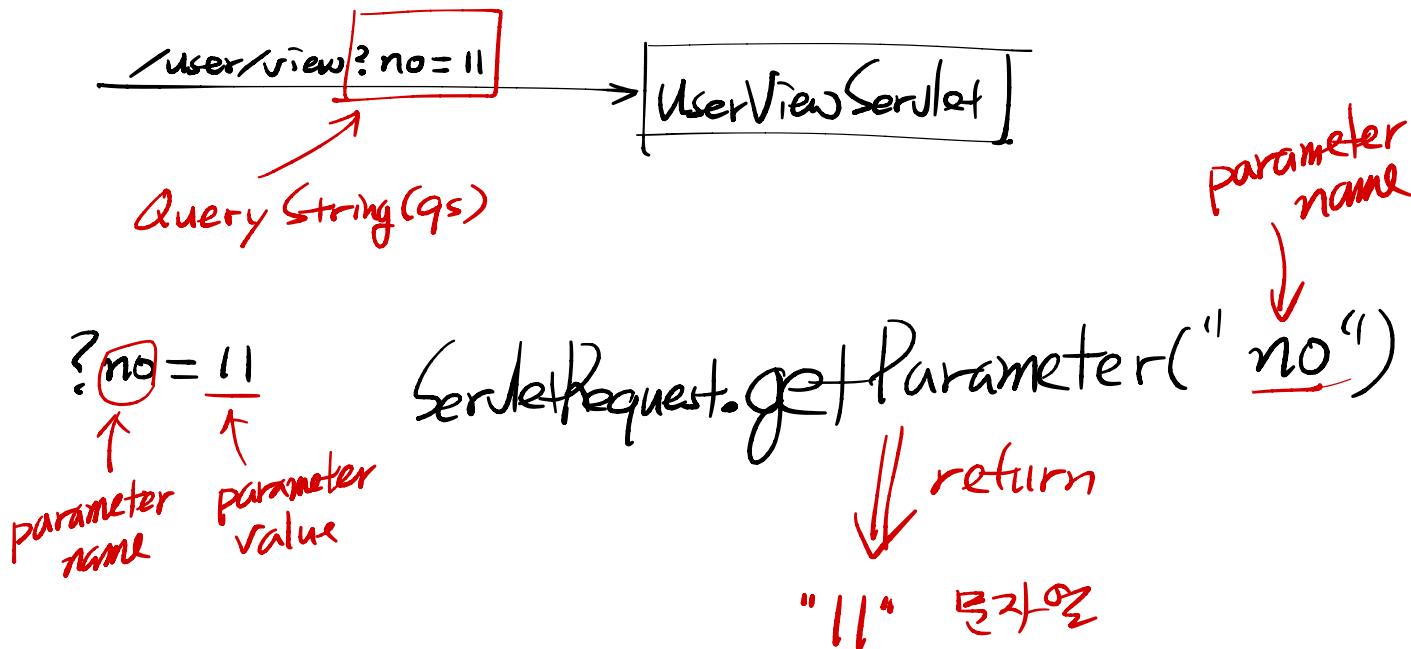
기타 → 파일

## \* HTML (Hyper-Text Markup Language)

↳ 원래는 문자



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



\* HTTP 프로토콜에서HTTP의 주요한 2가지 유형 : GET 이정 2가지 POST 이정

① GET 이정

method  
Request-URI

GET /ex04/s1?name=ABC%EA%B0%80%EA%B0%81&age=20 HTTP/1.1 ↵ Request-Line

User-Agent: PostmanRuntime/7.41.2 ↵

Accept: \*/\* ↵

Cache-Control: no-cache ↵

Postman-Token: eb57aac2-d30f-42a9-a123-bdf29a942df4 ↵

Host: localhost:8888 ↵

Accept-Encoding: gzip, deflate, br ↵

Connection: keep-alive ↵

↖ CRLF (줄바꿈)

HTTP Version

header  
[]  
보조  
클라이언트

## \* Method

- GET
  - POST
  - HEAD
  - PUT
  - DELETE
- OPTIONS
  - TRACE
  - CONNECT

## \* Header

- (general)
- Date
  - Connection
  - Cache-Control
  - Pragma
  - :
- (entity)
- Content-Length
  - Content-Type
  - Last-Modified
  - Expires
  - Content-Encoding
  - :

## (request)

- Accept
- Host
- Referer
- User-Agent
- Authorization
- Accept-Enc

## (response)

## ② POST 메시지

URL이 글자들은 ISO 121063d

POST /ex04/s2 HTTP/1.1 ← request-line

User-Agent: PostmanRuntime/7.41.2 ←

Accept: \*/\* ←

Cache-Control: no-cache ←

Postman-Token: 4c779857-6dc1-4d87-9f4a-3631b712c257 ←

Host: localhost:8888 ←

Accept-Encoding: gzip, deflate, br ←

Connection: keep-alive ←

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

(Content-Length: 32 ←

(name=AB%EA%B0%80%EA%B0%81&age=20 ←

} header

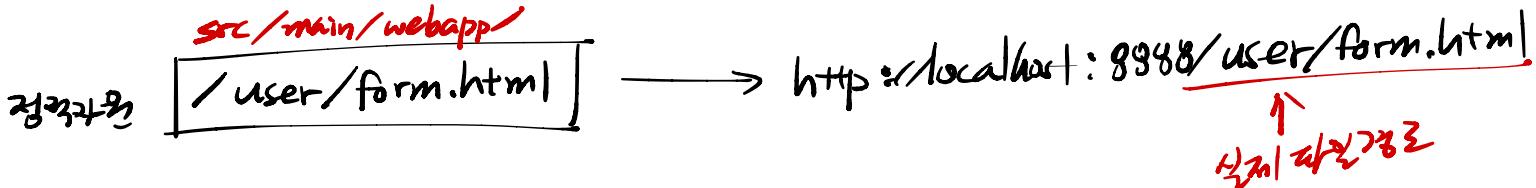
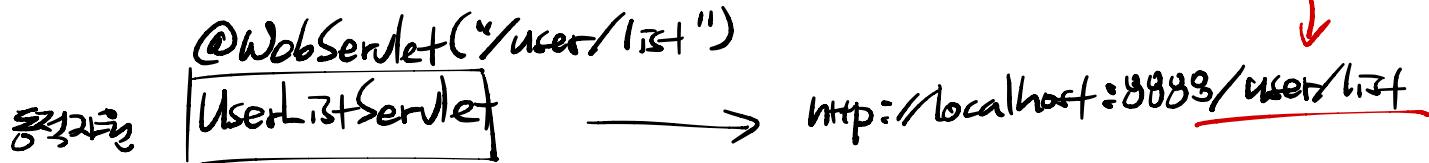
message-body = payload = entity

## \* 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



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  $\Rightarrow$  URL에서 특별한 의미로  
사용된다.

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

URL에서  
문자열은 표현할 때  
%로 표기

Unreserved characters

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

한글 ?  
한국어 → %xx / %xx

"ABC가?" → ABC%EA%B0%80%GA  
EAB080 EAB081  
%B0%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

\* POST 요청으로 보낸 한글 데이터를 읽을 때 깨지는 이유

클라이언트 "ABC가각" ↗ URL 인코딩 (%-인코딩)

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

41 → 0041 A

42 → 0042 B

43 → 0043 C

EA B0 80 → ACOO 가

EA B0 81 → AC01 𠂊

↑  
클라이언트 보낸 문자로  
UTF-8 이라고 간주하고  
UTF-16로 변환

String  
(UTF-16)

POST

HTTP

ISO-8859-1 (default) → UTF-16

41 → 0041 A

42 → 0042 B

43 → 0043 C

EA → 00 EA ē

B0 → 00 B0 o

80 → 00 80 .

EA → 00 EA ē

B0 → 00 B0 o

81 → 00 81 .

\* POST 요청으로 보낸 한글 데이터를 읽을 때 깨지는 이유  $\Rightarrow$  한글 아님!

클라인은 "ABC가각"  $\rightarrow$  URL 인코딩 ("%인코딩")

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

HTTP  
↓  
41 42 43 EA B0 80 EA B0 81

%xx 문자는 xx 바이트 문자 = URL 디코딩(decoding)  
HTTP://www.kt.com/abc  
한글이 자동으로 수용된다  
한글 데이터는 그대로

\* request.setCharacterEncoding("UTF-8")

getParameter("name")  $\Rightarrow$

\* 웹서버 getParameter()는 한글 지원하지 않음.  
한글은 java.util.String 으로 변환된다.

41	$\rightarrow$	00 41
42	$\rightarrow$	00 42
43	$\rightarrow$	00 43
EA	$\rightarrow$	A C 00
B0	$\rightarrow$	A C 01

한글은 한글을  
한글로 변환된다.  
UTF-8 3  
한글은 한글,  
UTF-16 으로  
한글은 한글.

## \* 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 <b>① 전송의 범위, 제한</b>	제한된 규칙 → 데이터 전송 <b>② 제한, 이동</b>
데이터 전송 데이터 형식	URL은 텍스트로 → 전송 불가 즉, 데이터를 데이터를 텍스트로 변환 한 경우 가능 <b>③ Base64</b>	application/x-www-form-urlencoded ⇒ 불가! multipart/form-data ⇒ 가능
보안	URL은 일반적으로 cache된다 다른 사용자에게 노출될 수 있다 <b>✓ 보호되어야 하는 정보는 제외.</b> <b>✓ URL에 데이터를 포함하는 경우는 절대</b>	<ul style="list-style-type: none"> <li>- 암호화되어야 함</li> <li>- 비밀번호 데이터 제외</li> <li>- <b>(보호되어야 하는)</b> 중요한 데이터 제외 (<b>문서가 있는</b>) <b>문서내부</b></li> </ul>

\* 파일 업로드 ( multipart/form-data )  $\Rightarrow$  HTML 와 HTTP 조합

```
<form action="s3" enctype="multipart/form-data" method="post">  
이름: <input type="text" name="name"><br>  
나이: <input type="number" name="age"><br>  
사진: <input type="file" name="photo"><br>  
<input type="submit" value="POST 전송">  
</form>
```

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"  
내이너리 파일 데이터....

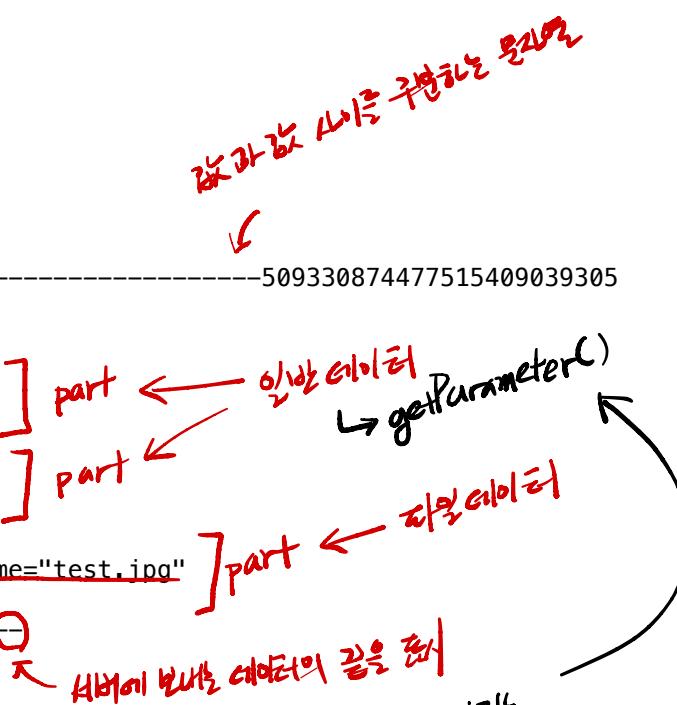
-----509330874477515409039305--

## \* 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....
```

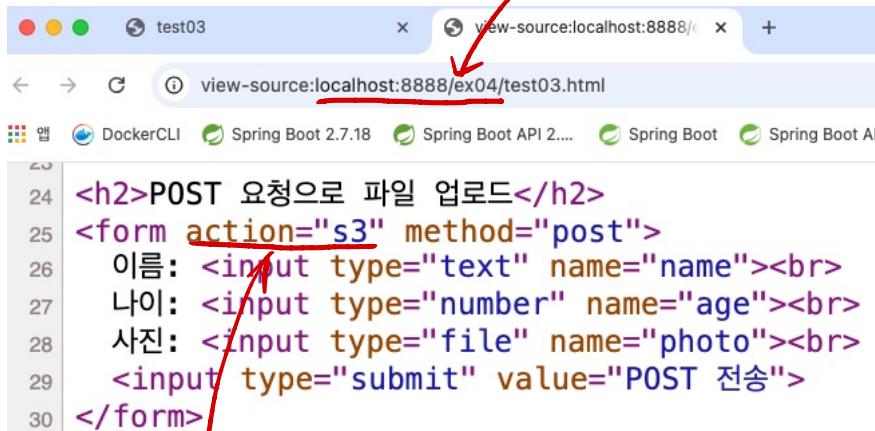
↓  
getPart()  
↓  
Part  
↓write()



multipart/form-data 형식으로 전송된 데이터는  
여기서 파일은 not MultipartFile이다.  
" @MultipartFile "

## \* 철자 URL vs 숫자 URL

현재 페이지 링크



```
24 <h2>POST 요청으로 파일 업로드</h2>
25 <form action="s3" method="post">
26   이름: <input type="text" name="name"><br>
27   나이: <input type="number" name="age"><br>
28   사진: <input type="file" name="photo"><br>
29   <input type="submit" value="POST 전송">
30 </form>
```

/로 시작하는 링크는 파일을 업로드하는 URL입니다.

현재 페이지를 기준으로 경로가 결정됩니다.

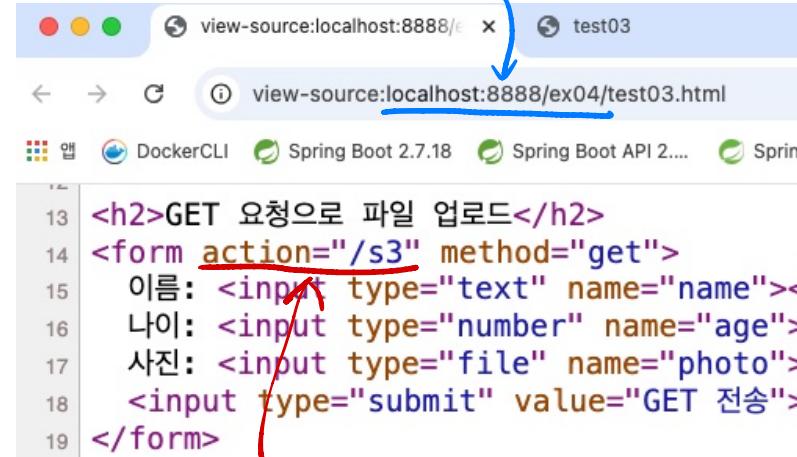
결과 링크:

http://localhost:8888/ex04/s3

현재페이지 링크

결과 링크

현재페이지 링크



```
13 <h2>GET 요청으로 파일 업로드</h2>
14 <form action="/s3" method="get">
15   이름: <input type="text" name="name">
16   나이: <input type="number" name="age">
17   사진: <input type="file" name="photo">
18   <input type="submit" value="GET 전송">
19 </form>
```

/로 시작하는 링크는 파일을 업로드하는 URL입니다.

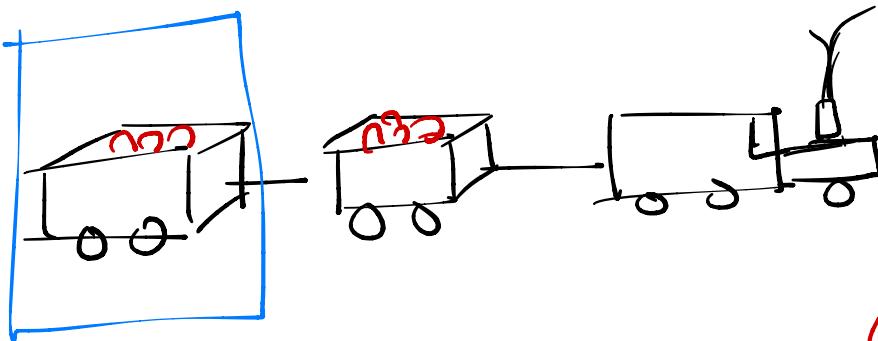
결과 링크:

http://localhost:8888/s3

host 주소

결과 링크

\* payload (유도하중 = 옮기는 집에서 실제 수송료를 부과하는 것)



화물 + 수송장치  
(0.5점) (0.5점)

↑  
수송하는 물건의  
중량이 차량에  
전달되는 것

⇒  
수송하는 물건의  
"무게"라는 것을  
"payload" 라고 하.

1점 2점 3점

POST /ex04/s2 HTTP/1.1  
User-Agent: PostmanRuntime/7.41.2  
Accept: \*/\*  
Cache-Control: no-cache  
Postman-Token: 9f4c5b0b-d685-4d6a-9b86-b700a084b42c  
Host: localhost:8888  
Accept-Encoding: gzip, deflate, br  
Connection: keep-alive  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 33

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

↑  
수송하는 물건의  
"무게"는  
"payload"  
"payload"

## \* 파일첨부 및 첨부파일 처리

```
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"
<test.jpg>
-----509330874477515409039305-----
```

① Servlet 3.0 를 위한 API API  
@MultipartConfig 헤더 → getPart() → Part  
→ 파일

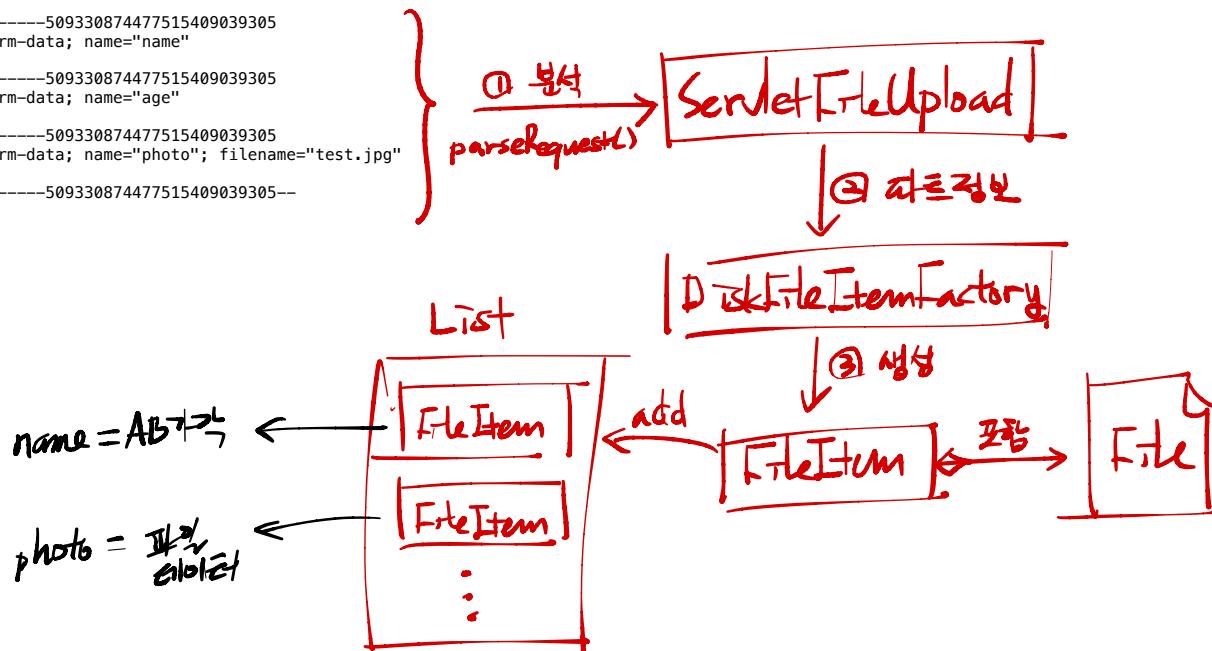
② Third-party Library (third-party library)  
a) Apache Group의 FileUpload  
"FileUpload" 파일로 파일 처리

a) Spring WebMVC 파일처리

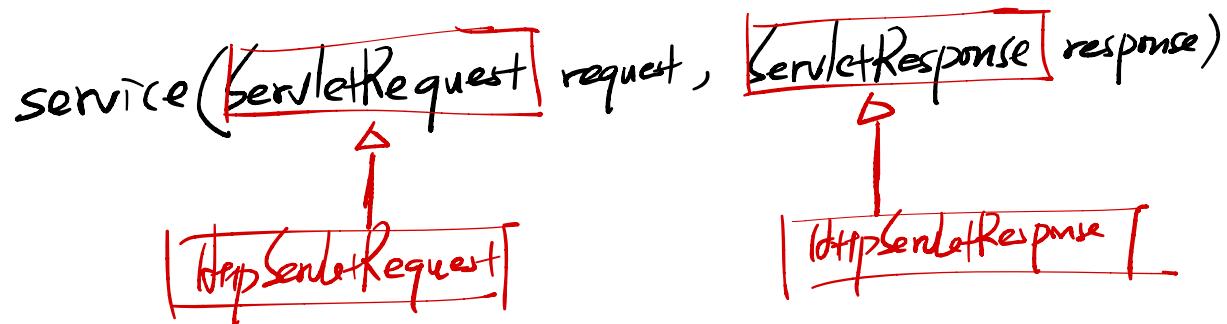
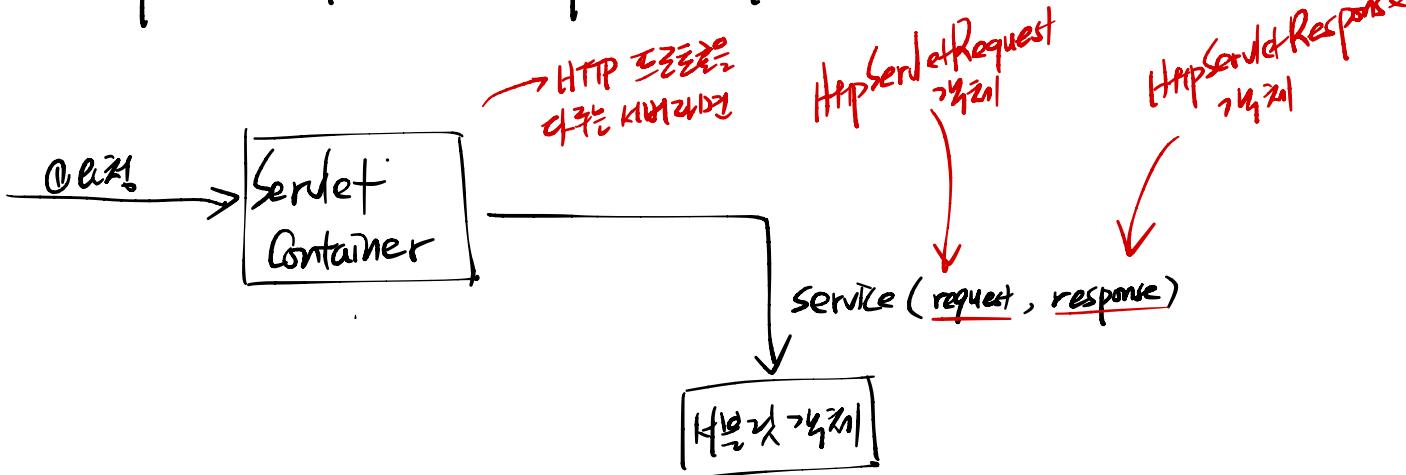
## \* Apache commons-fileupload 2로 1로

```
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
```

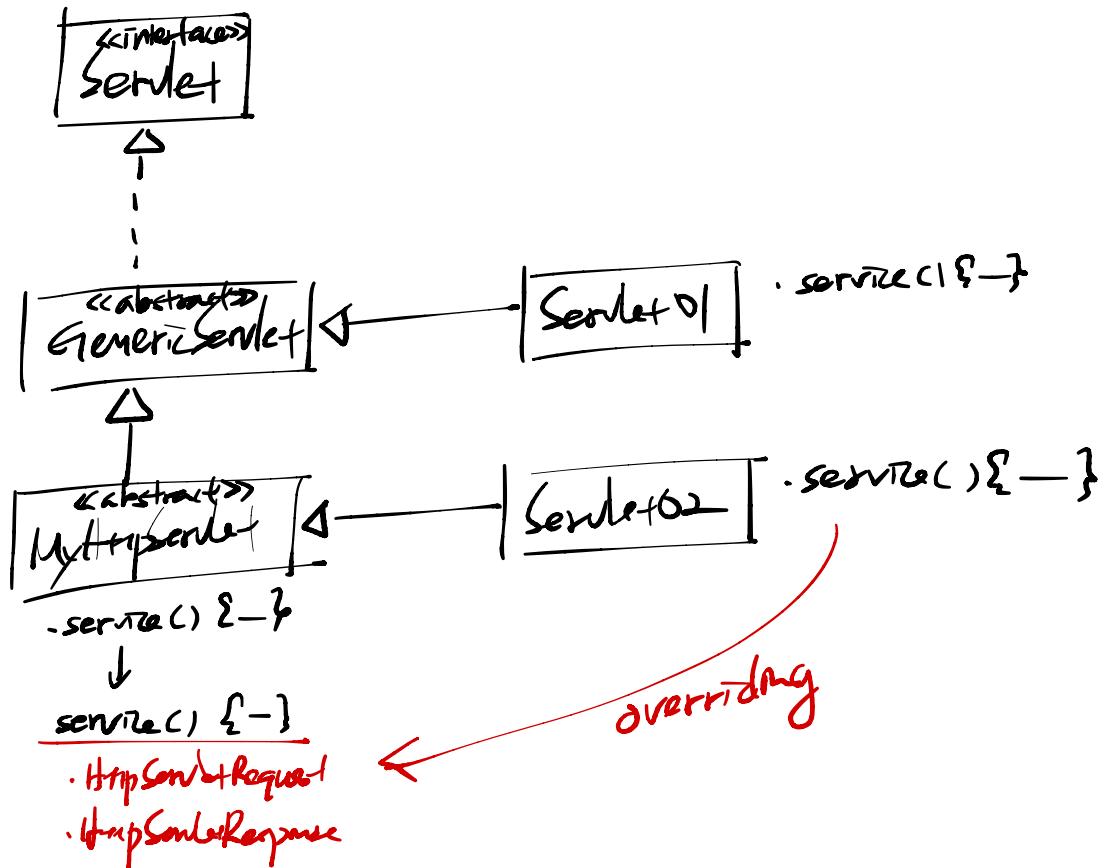
```
-----509330874477515409039305
Content-Disposition: form-data; name="name"
ABC가각
-----509330874477515409039305
Content-Disposition: form-data; name="age"
20
-----509330874477515409039305
Content-Disposition: form-data; name="photo"; filename="test.jpg"
<test.jpg>
-----509330874477515409039305--
```



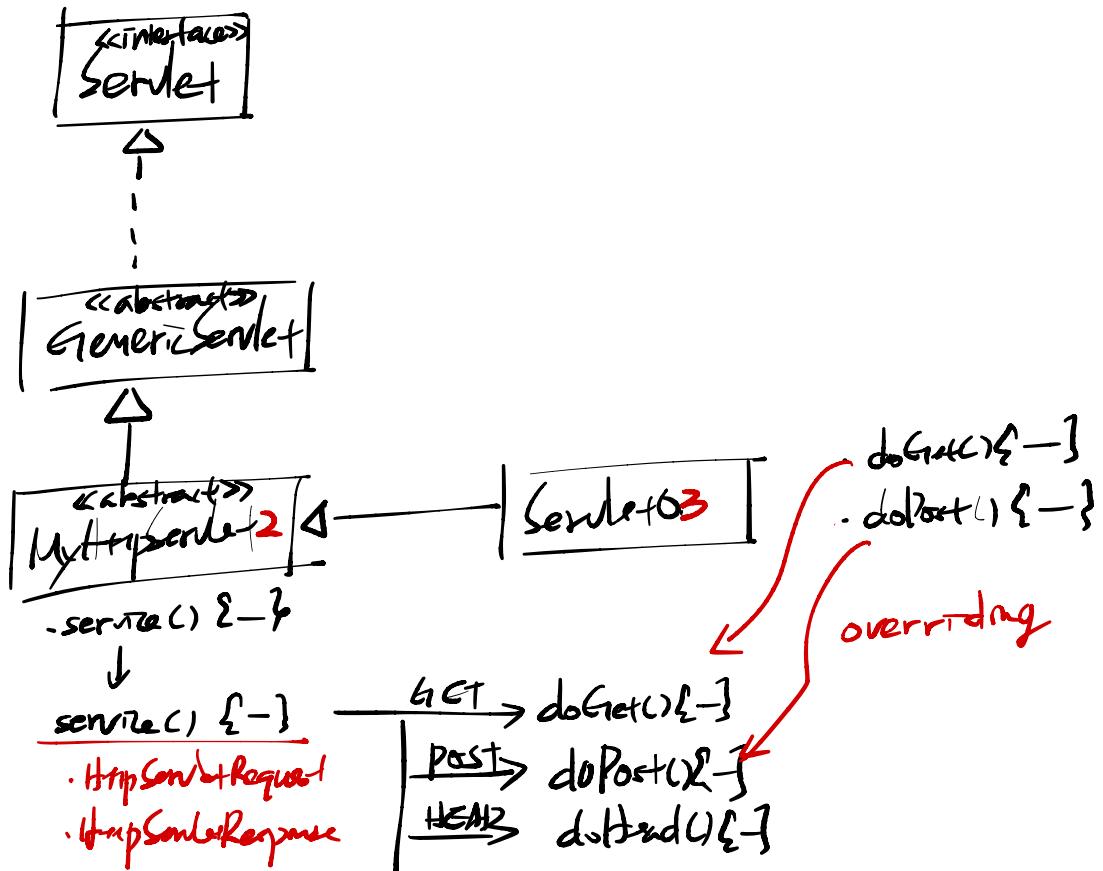
## \* HttpServletRequest et HttpServletResponse



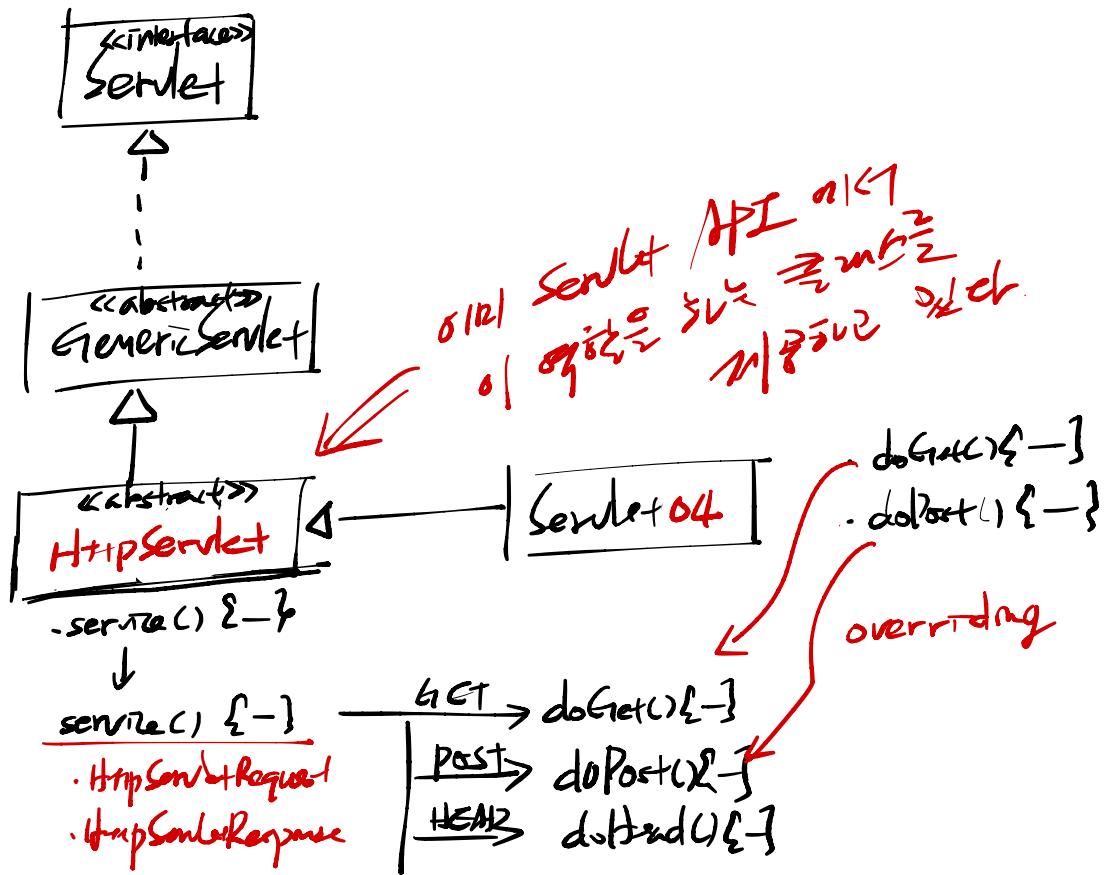
\* HTTP 프로토콜 처리를 위한 기본 구조



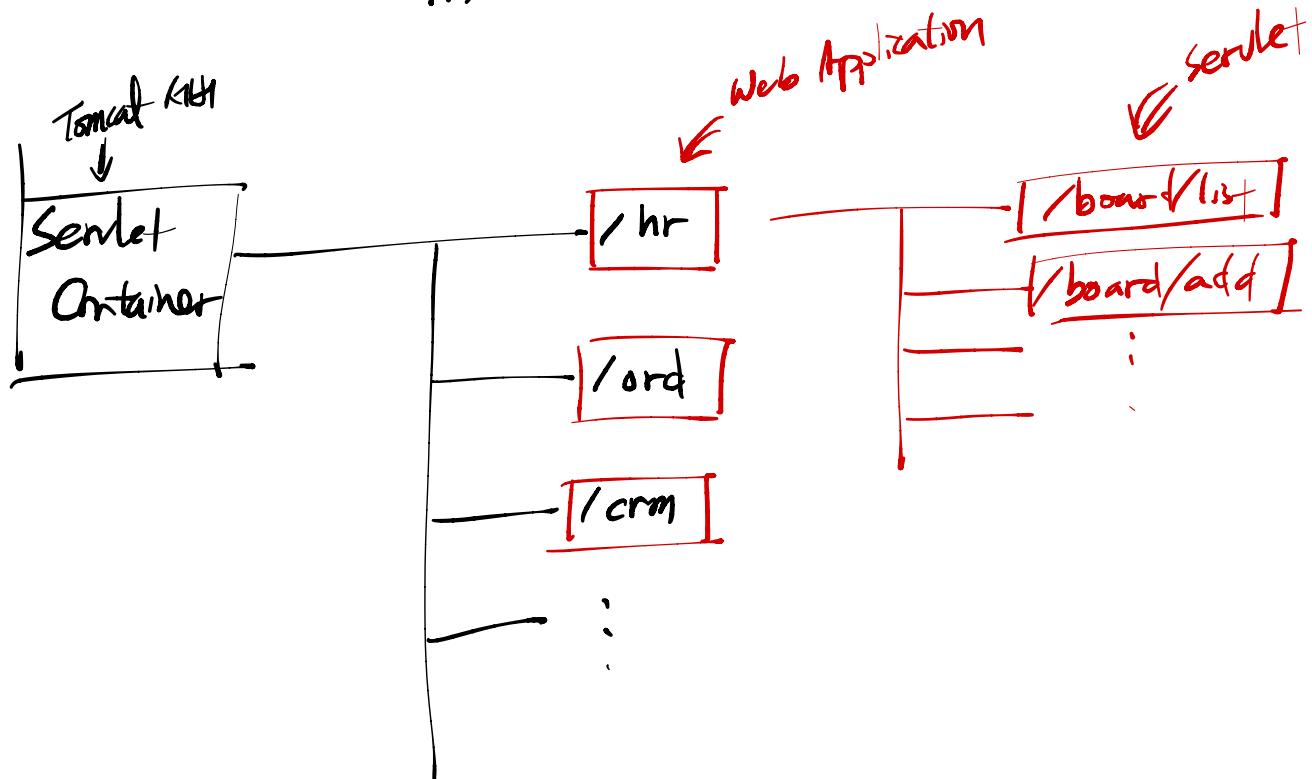
\* HTTP 프로토콜 처리는 서블릿을 활용하는 방식 II



\* HTTP 프로토콜 처리는 서블릿을 활용한다 | III



\* ServletContainer, Web App, Servlet obj (view)



## \* 사용자 설정

① 기본 → 하위 사용자에 대한 경로 설정이 블록체인을 때 사용자 개체 사용

② 사용자 설정이 목적지 이름 개체는 init() 초기화 후 사용할 때 \*

↓  
원정 사용자 개체 사용 → init() 호출 → 이름 개체 초기화 후 사용

원정 사용자 개체 초기화  
설정값을 대

Load On Startup  
⇒ 초기화 설정 값을  
사용자 개체를  
제공 사용

⇒ 초기화 설정 값을  
제공 사용

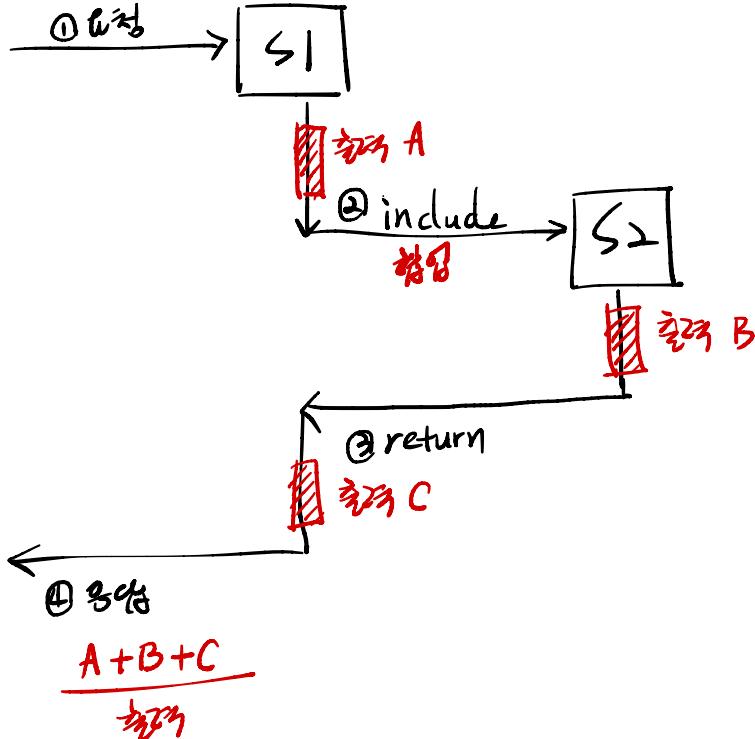
↓  
사용자 설정 초기화!

↓  
API 사용 설정 값을 제공하세요  
API 사용 설정 값을 제공하세요!

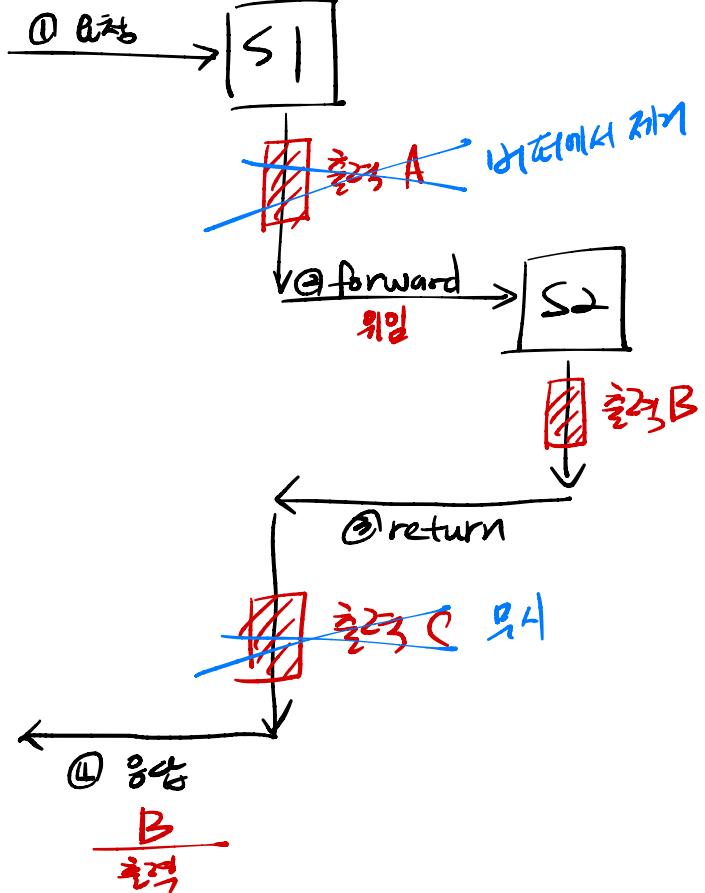
↓  
원정 사용자를 찾고!

\* include et forward

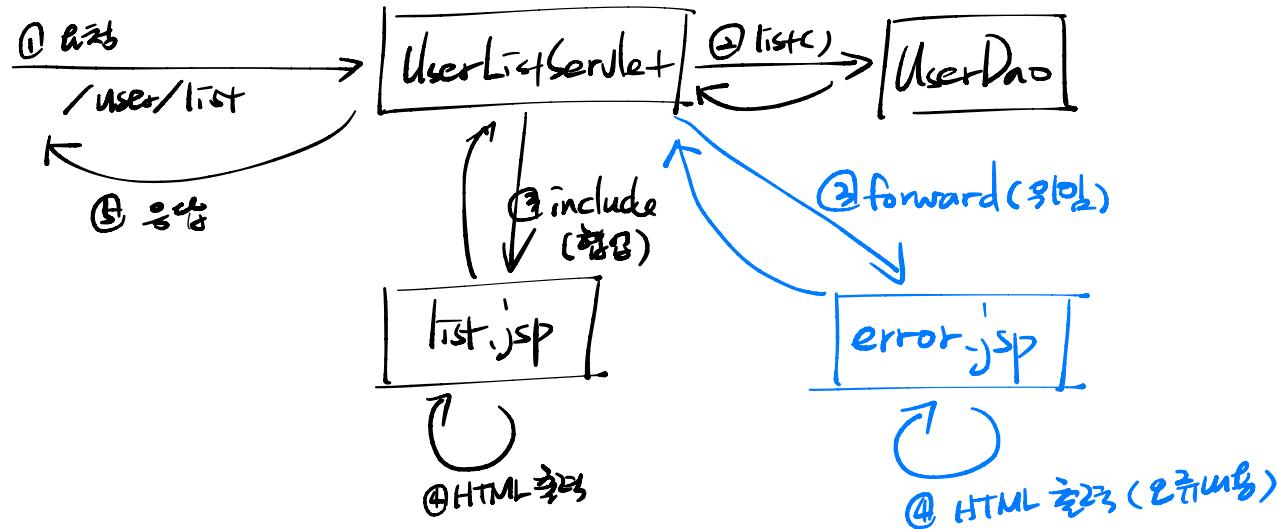
#include



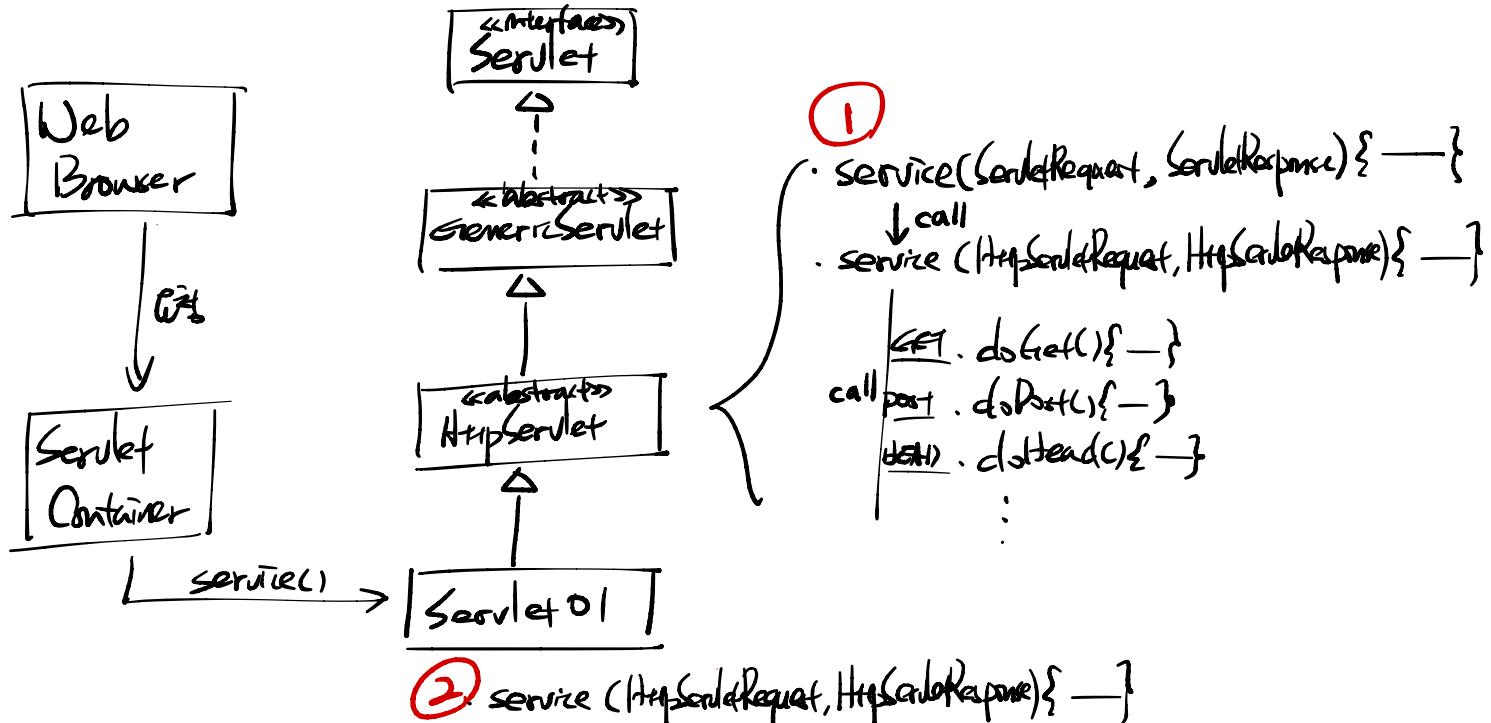
# forward



## \* include / forward ①)



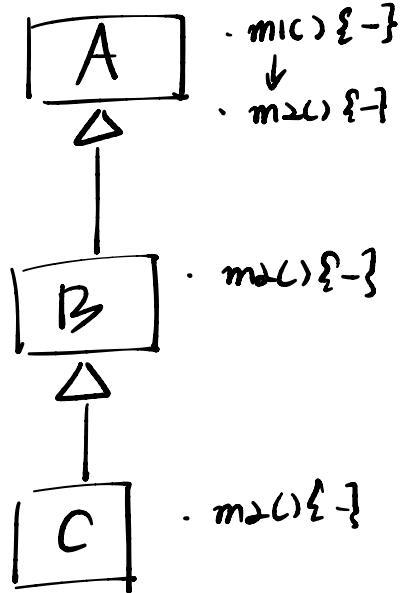
\* service() → service() → doGet() / doPost() / ...



DIKICI ⇒ 가상화된 DIKICI()

↑ 실제 디렉터리 구조의 URL이 실제 파일 경로를 찾는다.

\* D/LCL ဆိုလာသူများ

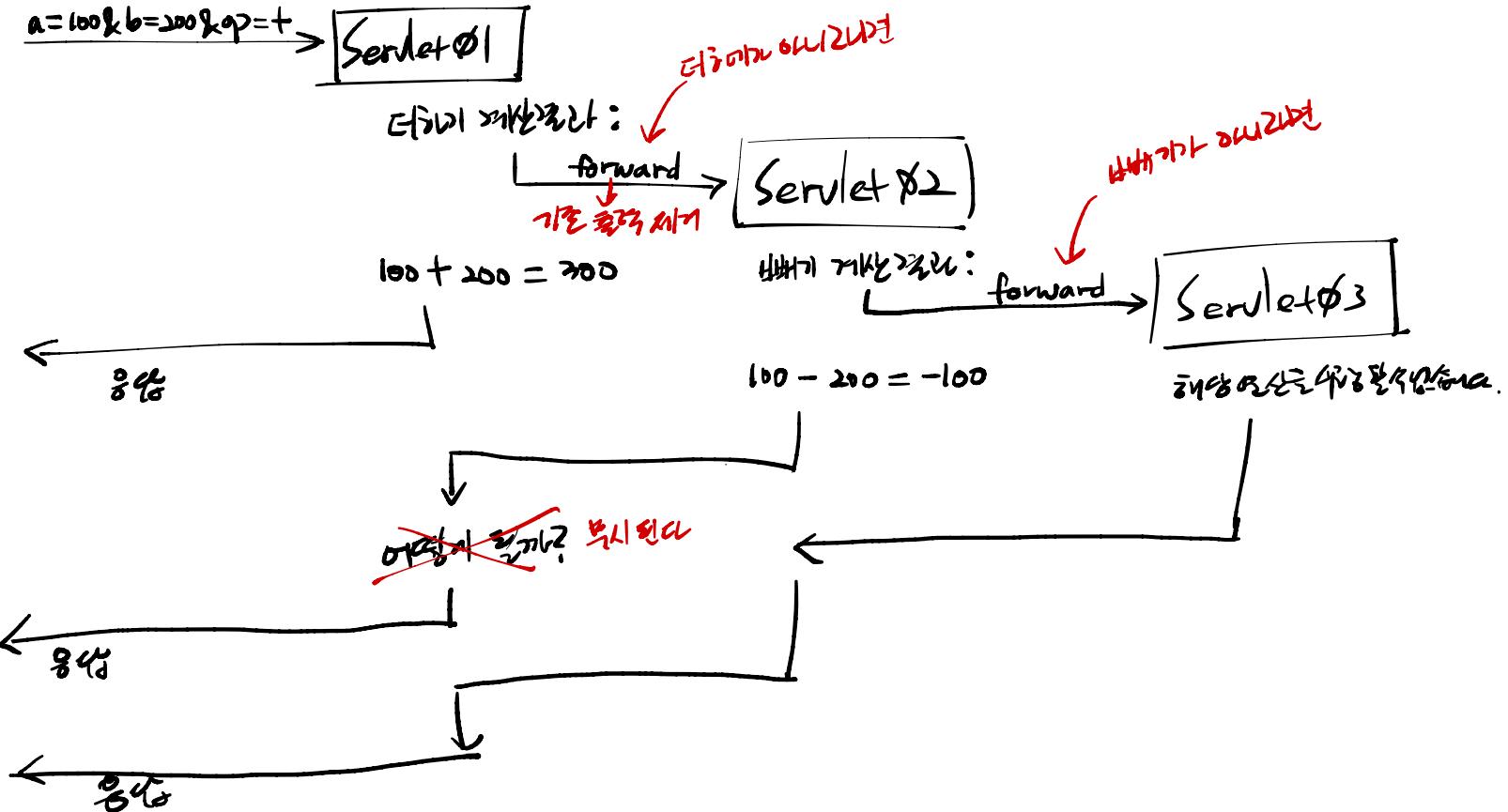


$A \ obj' = \text{new } A(c);$   
 $\ obj'.m1c;$   
 $A.m1c \rightarrow A.m2c$

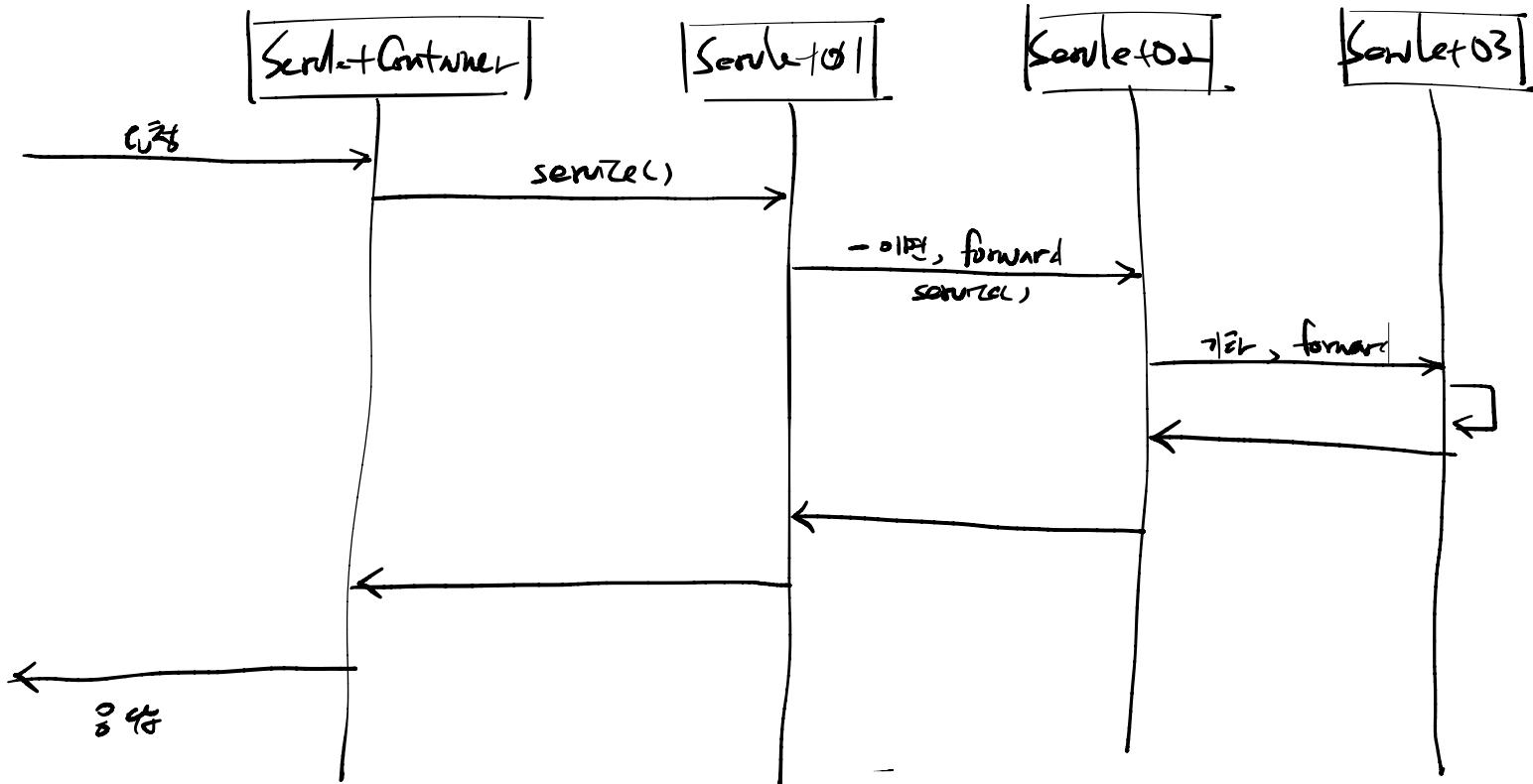
$B \ obj' = \text{new } B(c);$   
 $\ obj'.m1c;$   
 $B.m1c \rightarrow A.m1c \rightarrow B.m2c$

$C \ obj' = \text{new } C(c);$   
 $\ obj'.m1c;$   
 $C.m1c \rightarrow B.m1c \rightarrow A.m1c$   
 $\downarrow$   
 $C.m2c$

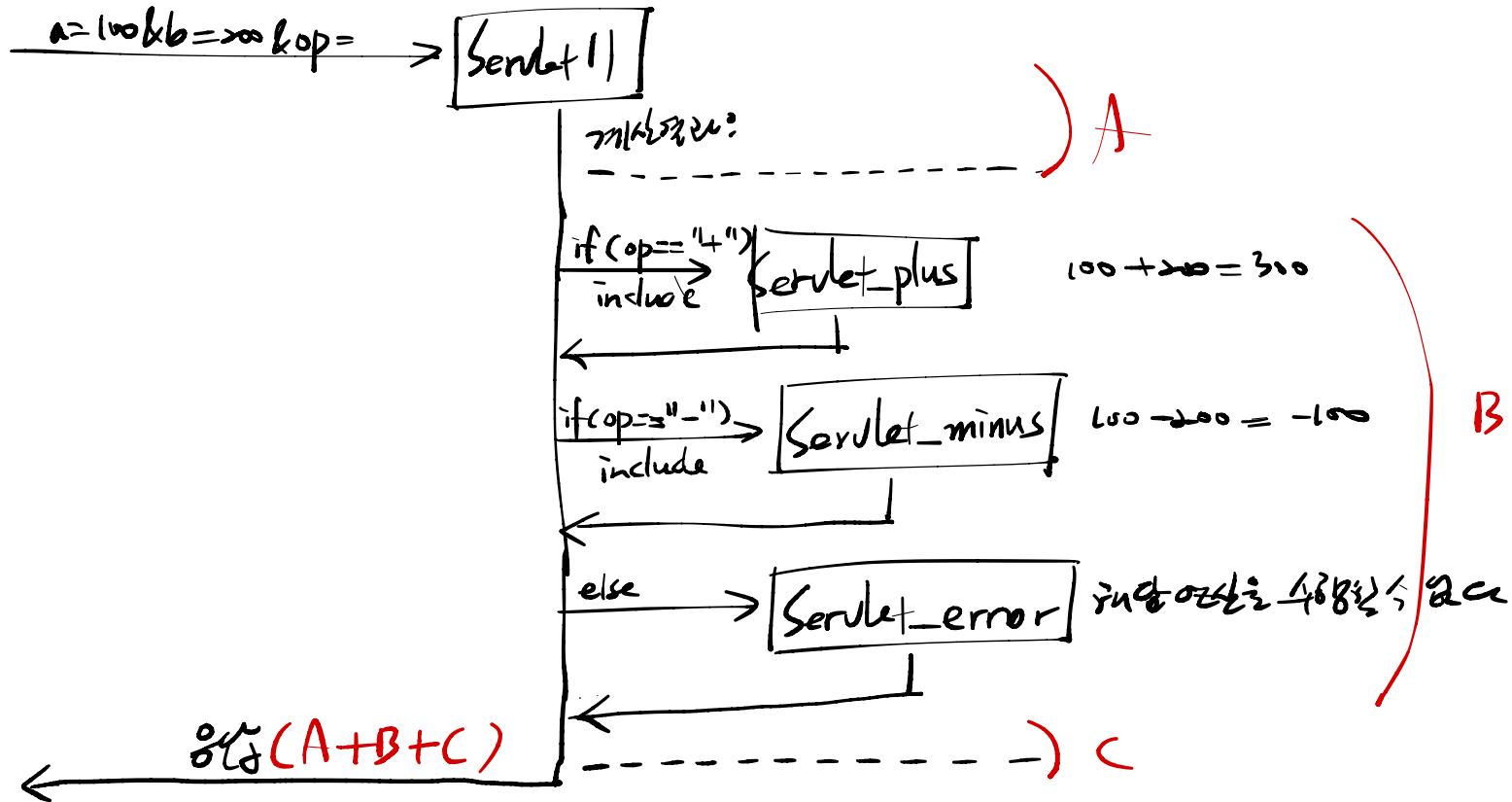
\* forward 활용 예)



\* forward (기본적) - Sequence Drag.

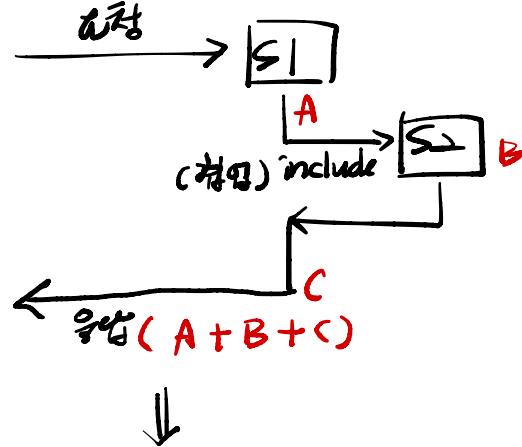


\* include 例題 a)



## \* include vs forward

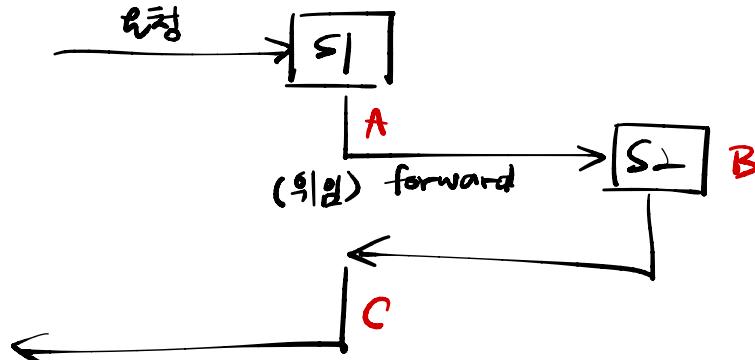
include



S1 이 내용은

setContentType()이  
다른 S2의 내용과  
다를지 모른다.

forward



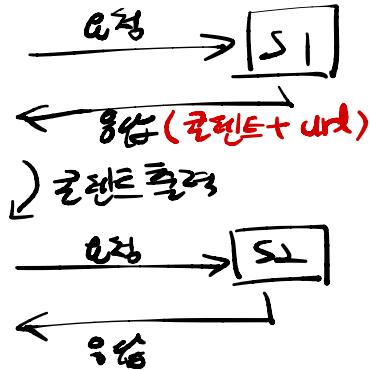
위임 (B) ← forward 전에 수행한 초기화는 차이  
return 값이 // 무시

S1이 내용은

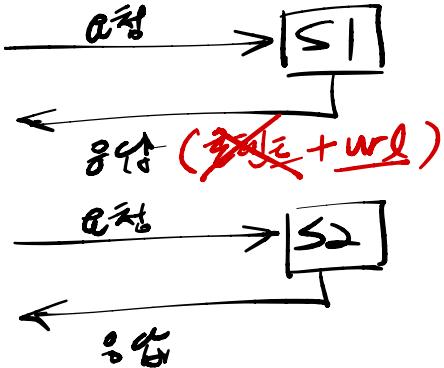
setContentType()은 초기화.  
S2의 setContentType()은 초기화.

## \* refresh vs redirect

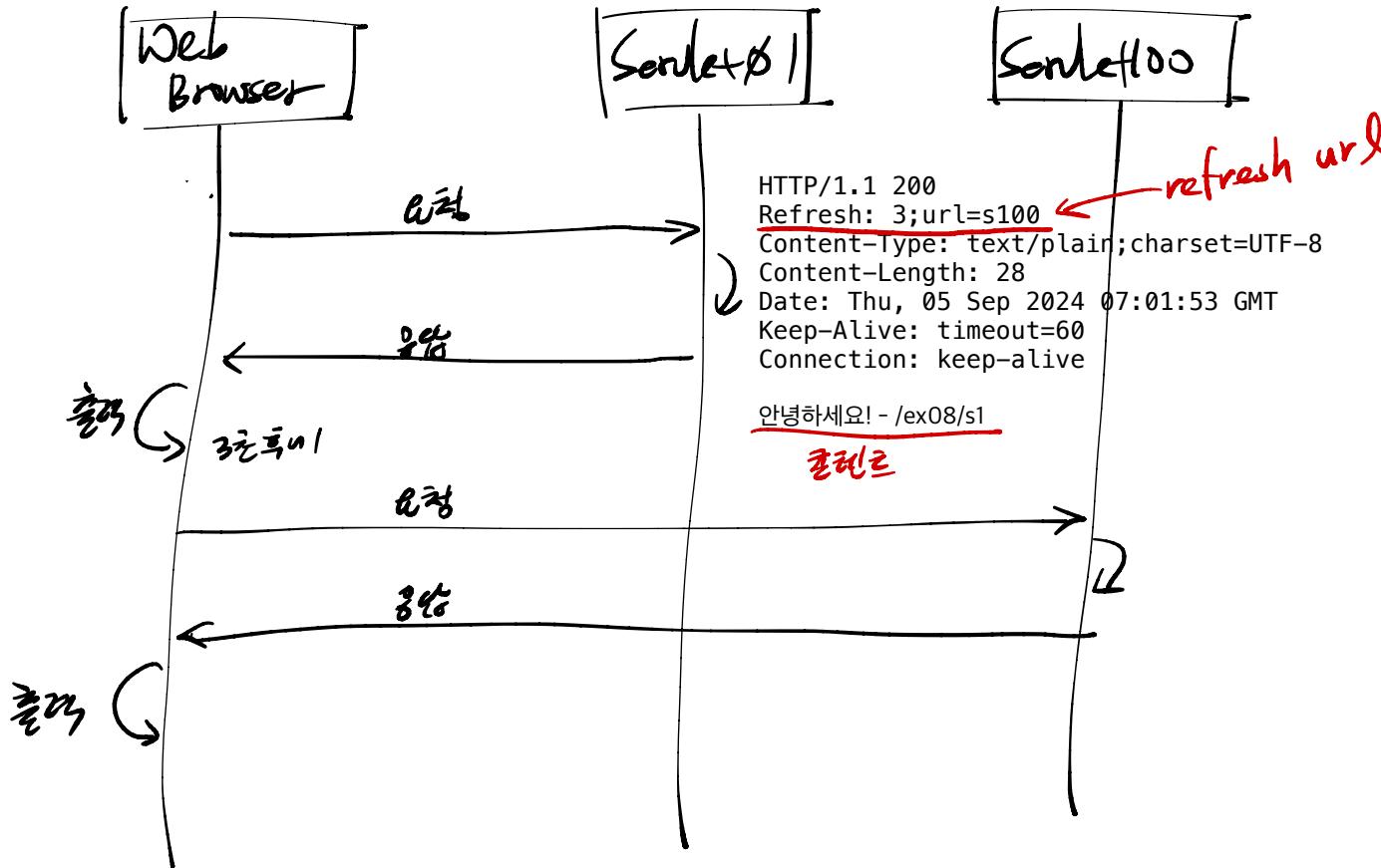
### refresh



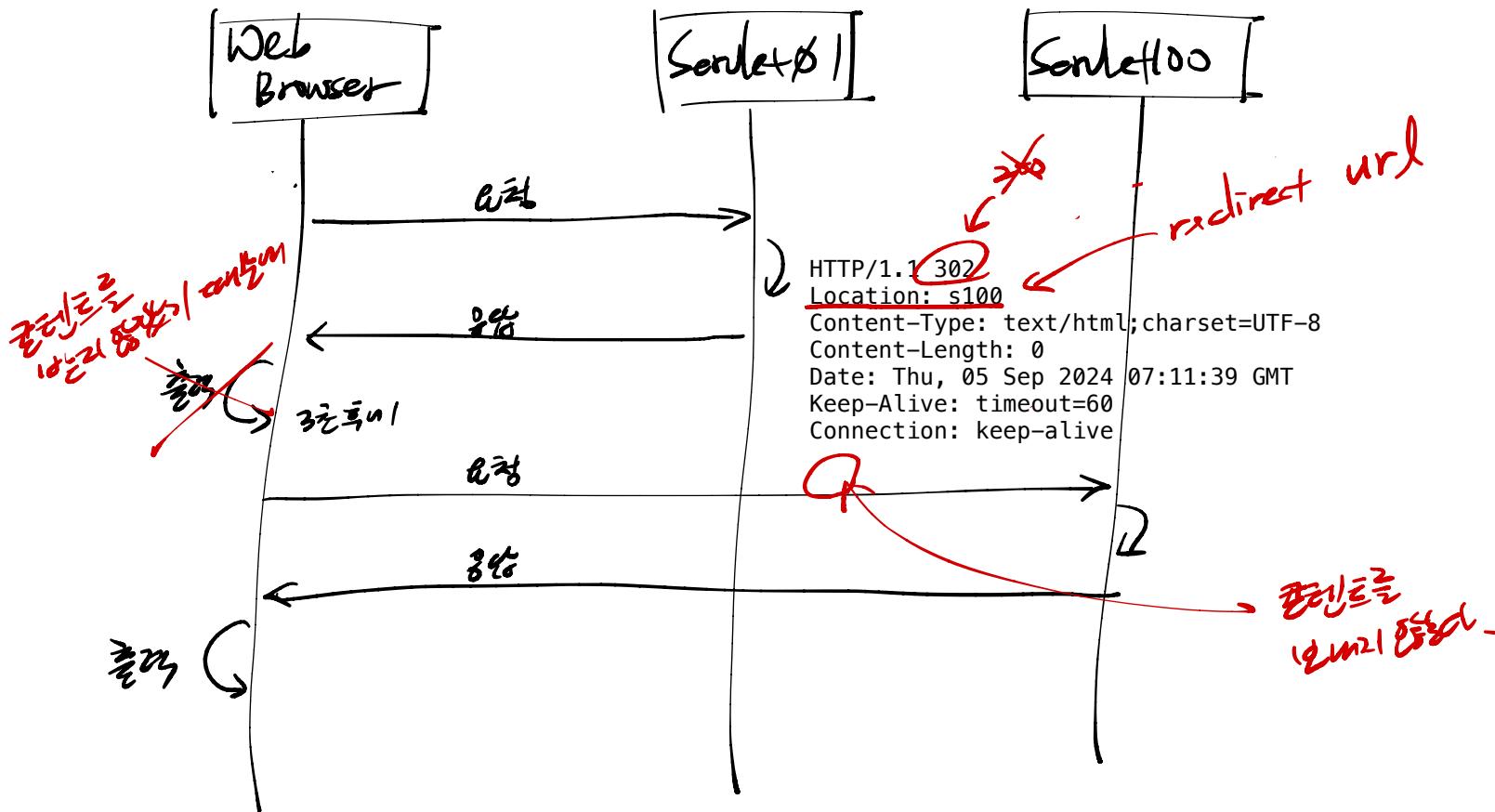
### redirect



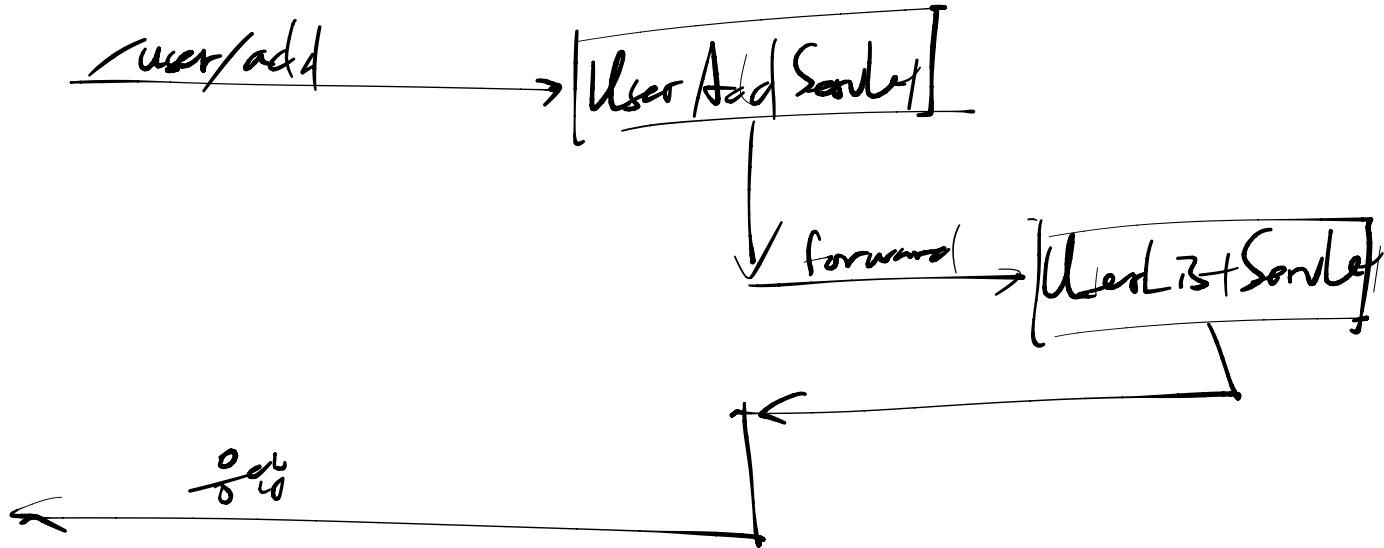
\* refresh = 키보드의 F5



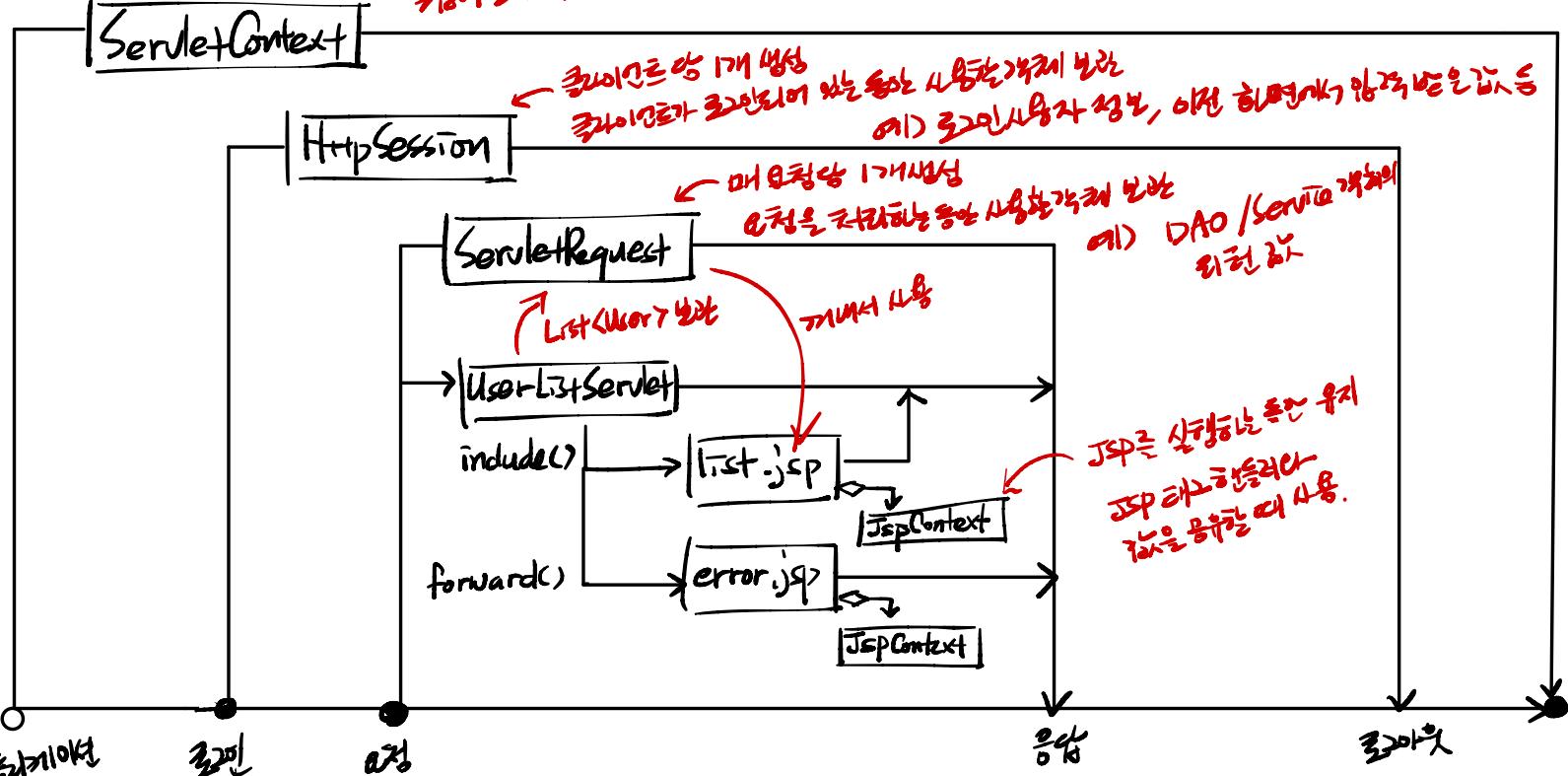
\* redirect (HTTP 302)



\* forward/include 2가지 방법 (on)



\* ① 뷰  
    → 웹 애플리케이션은 기본적으로  
        유저에게 보여줄 내용을 담고 있는  
        모든 것을 가리킨다. ex) DAO, Service, 전처리된 결과물 등.



애플리케이션  
상태

조회

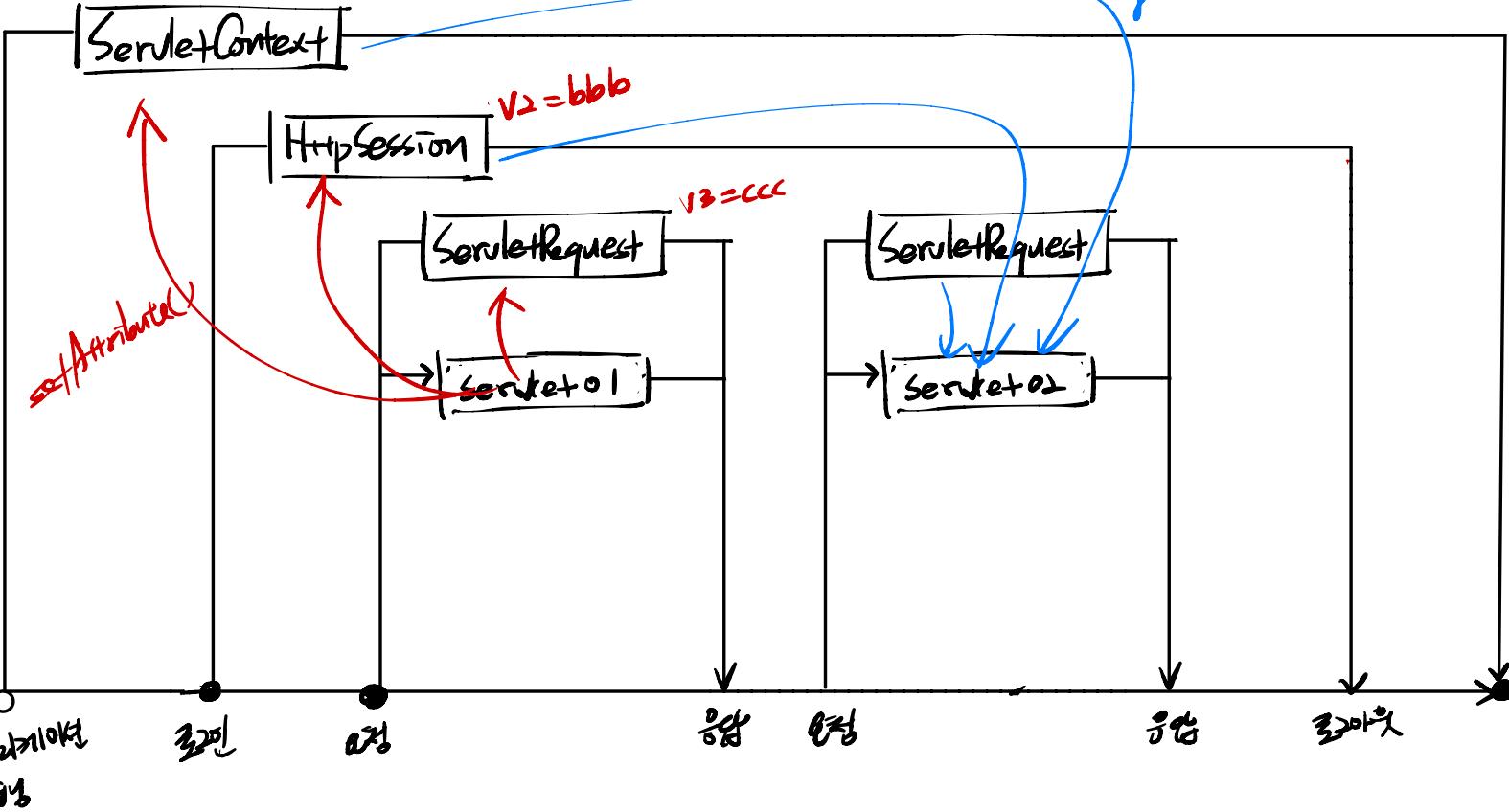
작성

등록

조회수

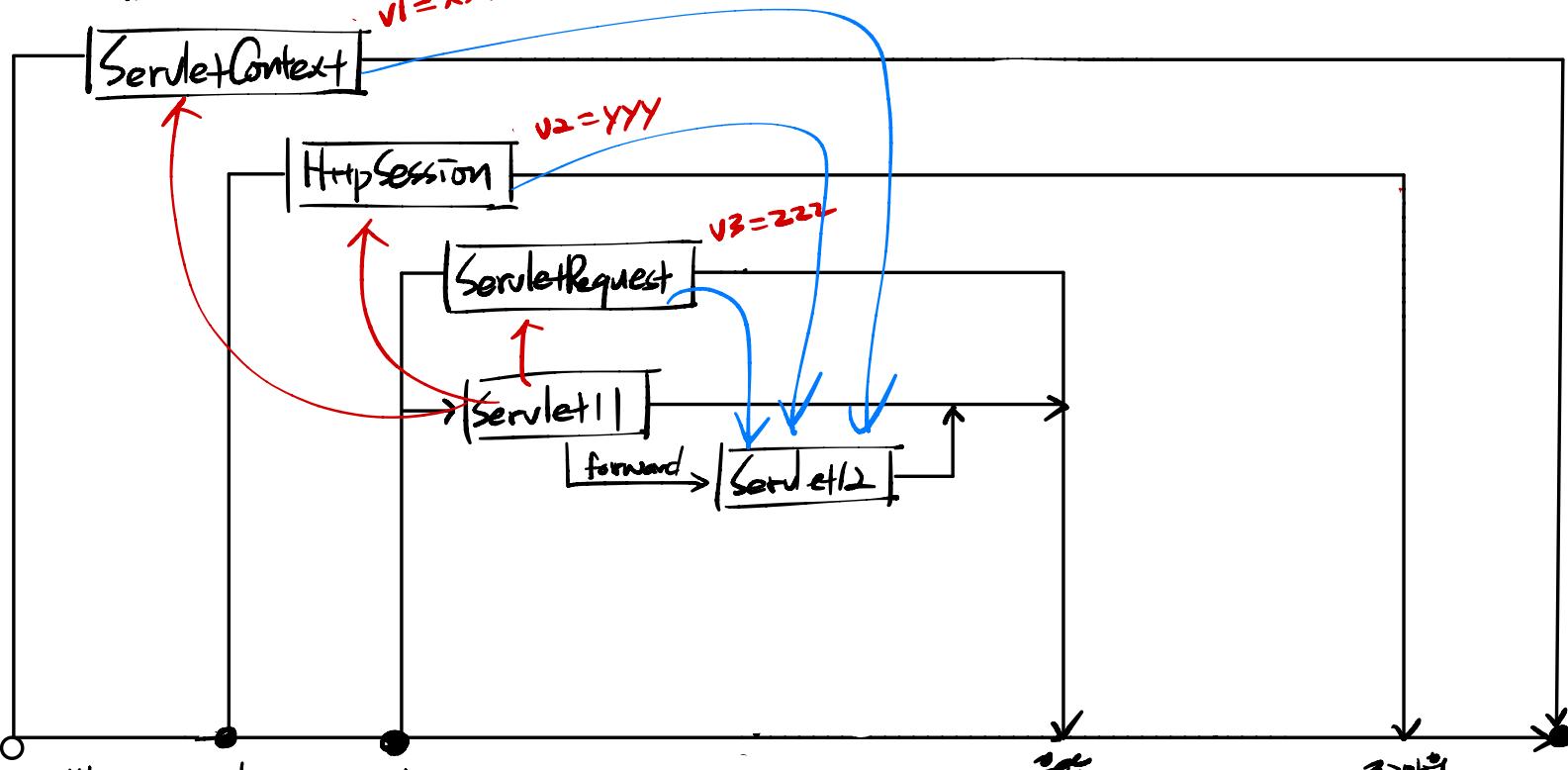
\* 9월 월례회

v1=aaa



\* 918 웹서버

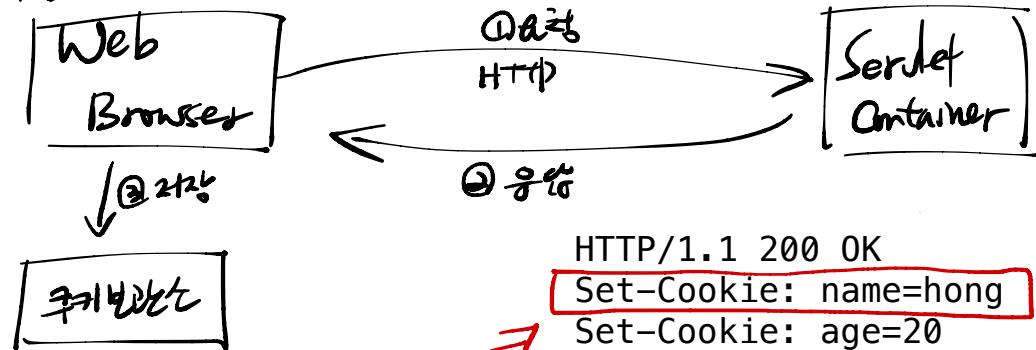
v1=xxx



0초  
320ms  
828ms  
255ms  
320ms

\* Cookie → HTTP Client와 Server가 서로적으로 주고 받는 데이터, 클라이언트와 서버

1) 웹 흐름 : HTTP → 클라이언트



HTTP/1.1 200 OK

Set-Cookie: name=hong  
Set-Cookie: age=20  
Set-Cookie: working=true  
Set-Cookie: name2=íê, è  
Set-Cookie: name3=%ED%99%8D%EA%B8%B8%EB%8F%99  
Content-Type: text/plain; charset=UTF-8  
Content-Length: 35  
Date: Fri, 06 Sep 2024 01:51:03 GMT  
Keep-Alive: timeout=60  
Connection: keep-alive

Cookie c1 = new Cookie("name", "hong");  
response.addCookie(c1);

/ex10/s1 - 쿠키 보냈습니다.

\* Cookie → HTTP Client et Server가 내보내는 주고 받는 데이터, 클라이언트와 서버

2) ⇒ 10분기 : 클라이언트 → 서버

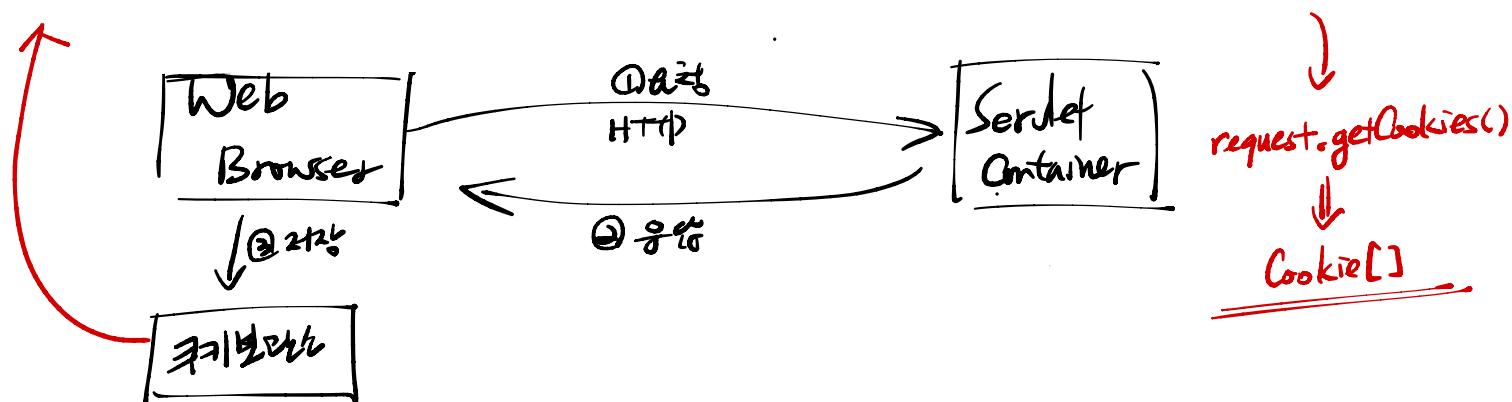
↳ 클라이언트는 서버에 요청할 때, 서버에서 받은 cookie가 있는지  
요청 정보에 포함되어 보내라.

단 cookie에 설정된 사용범위, 유통기간 등により 막을 때

사용범위가 설정 안된 경우, cookie를 보면 내용대로  
통과해버려 쓰는 게 좋다.

유통시간이 설정 안된 경우, 유통기간까지 충족이 되면 가능.

GET /ex10/s2 HTTP/1.1  
User-Agent: PostmanRuntime/7.41.2  
Accept: \*/\*  
Cache-Control: no-cache  
Postman-Token: cbce978a-a2df-4263-b24c-0399b6953829  
Host: localhost:8888  
Accept-Encoding: gzip, deflate, br  
Connection: keep-alive  
Cookie: age=20; name=hong; name2=ié,,e; name3=%ED%99%8D%EA%B8%B8%EB%8F%99; working=true



## \* Cookie 유통 기한 설정하기

```
Cookie c = new Cookie("v2", "bbb");
```

c.setMaxAge(30);  
    ↑  
    这儿

```
response.addCookie(c);
```

```
HTTP/1.1 200 OK
Set-Cookie: v1=aaa
Set-Cookie: v2=bbb; Max-Age=30; Expires=Fri, 06 Sep 2024 02:37:19 GMT
Set-Cookie: v3=ccc; Max-Age=60; Expires=Fri, 06 Sep 2024 02:37:49 GMT
Content-Type: text/plain; charset=UTF-8
Content-Length: 36
Date: Fri, 06 Sep 2024 02:36:49 GMT
Keep-Alive: timeout=60
Connection: keep-alive
```

/ex10/s11 - 쿠키 보냈습니다.

## \* Cookie 네트워크 | 쿠키 전송

```
Cookie c1 = new Cookie("v1", "aaa");
```

```
Cookie c2 = new Cookie("v2", "bbb");
```

```
c2.setPath("/ex10/a");
```

```
Cookie c3 = new Cookie("v3", "ccc");
```

```
c3.setPath("/");
```

HTTP 응답은 지정한 쿠키를 통해 서버의 URL이  
HTTP 응답에 포함되어 쿠키를 통해 서버의 URL이  
HTTP 응답에 포함되어 쿠키를 통해 서버의 URL이  
HTTP 응답에 포함되어 쿠키를 통해 서버의 URL이

```
@WebServlet("/ex10/s21")
```

→ 경로가 /ex10 으로 시작할 때

쿠키 /ex10/a 를 사용할 때

쿠키 / 를 사용할 때

/ex10/s22 요청 → c1, c3

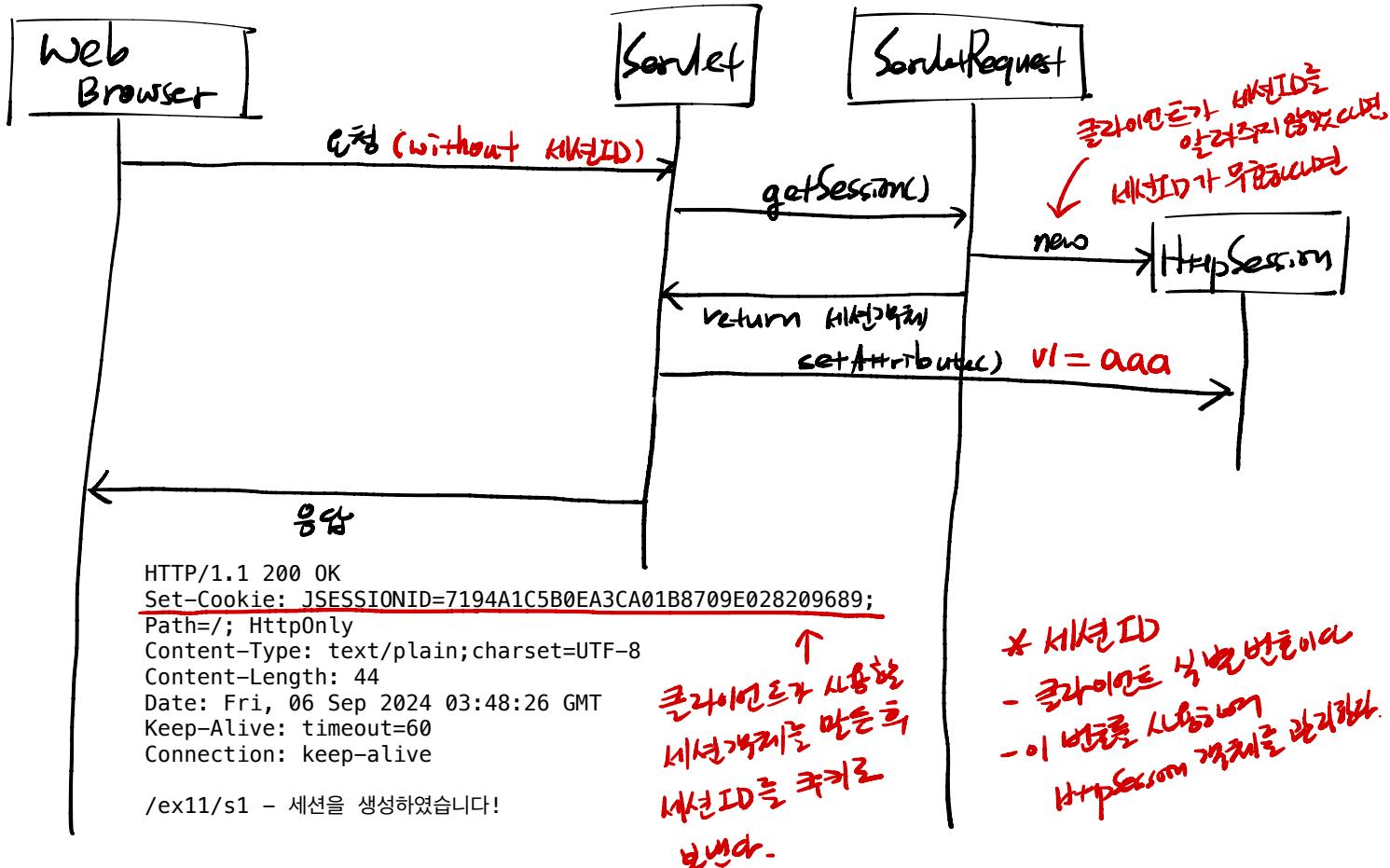
/ex10/a/b/c/s23 요청 → c1, c2, c3

/ex10\_1/s24 요청 → ✗, ✗, c3

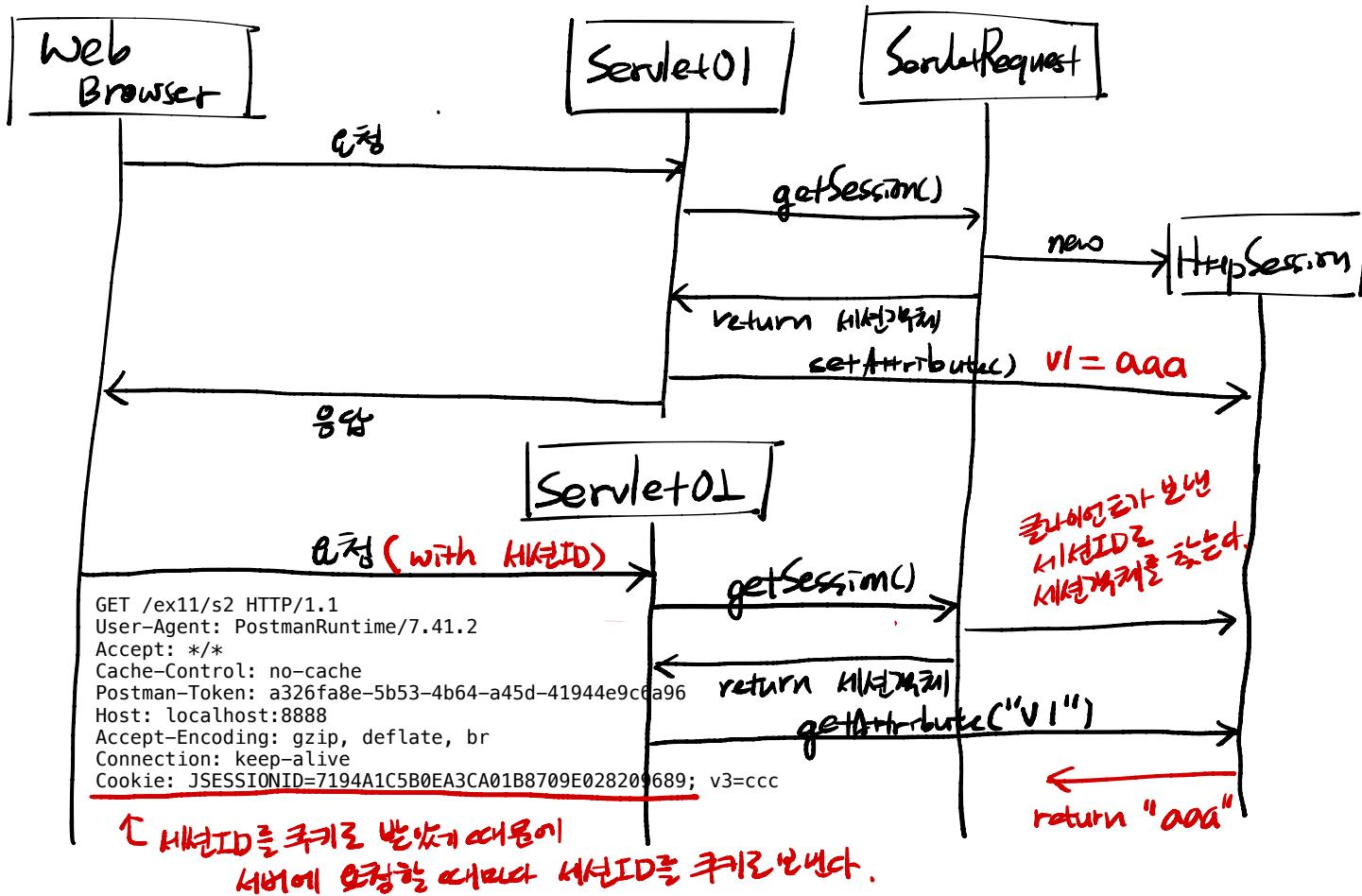
```
HTTP/1.1 200 OK
Set-Cookie: v1=aaa
Set-Cookie: v2=bbb; Path=/ex10/a
Set-Cookie: v3=ccc; Path=/
Content-Type: text/plain; charset=UTF-8
Content-Length: 36
Date: Fri, 06 Sep 2024 02:51:22 GMT
Keep-Alive: timeout=60
Connection: keep-alive
```

/ex10/s21 - 쿠키 보냈습니다.

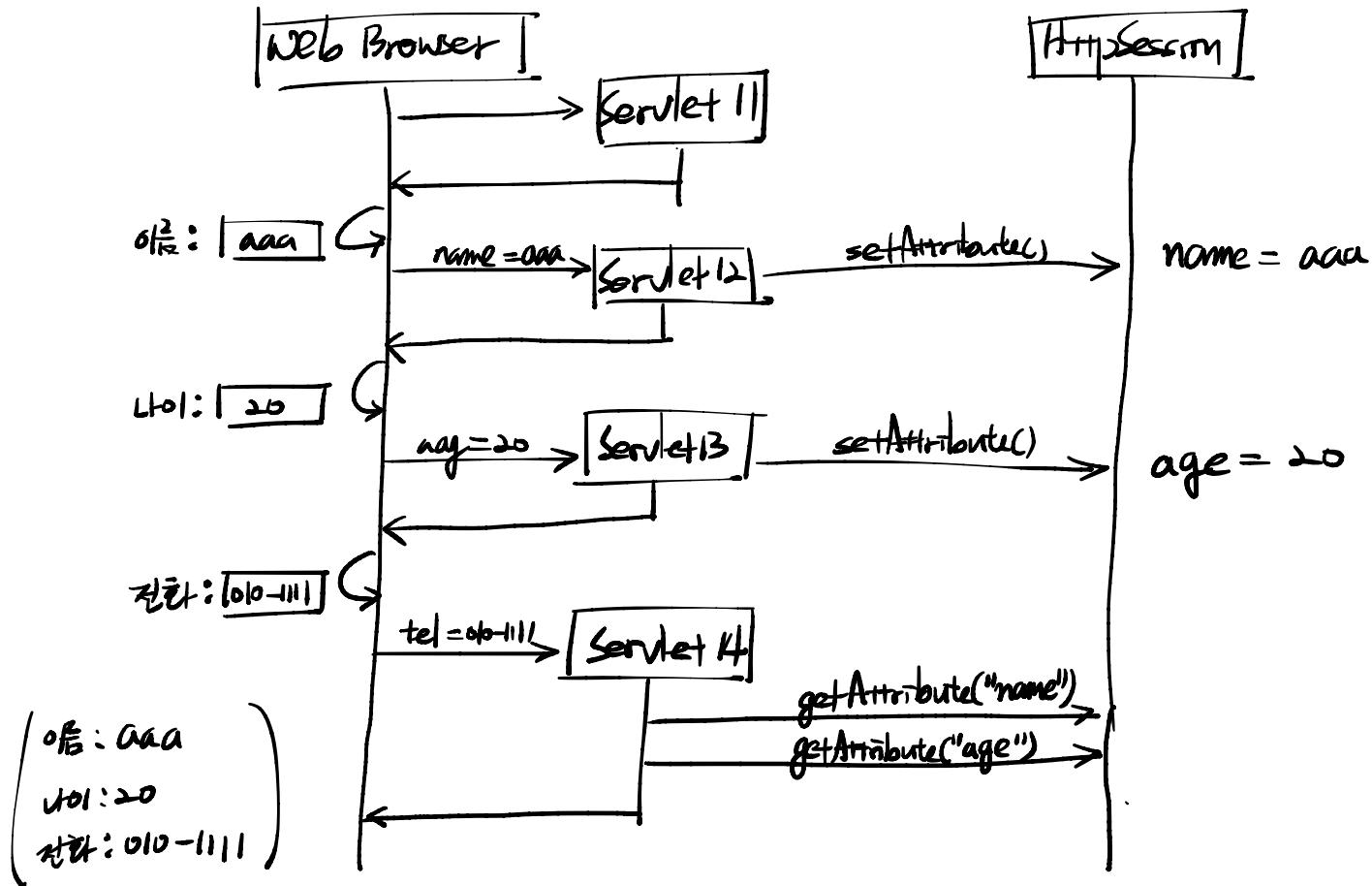
\* 키션 - 클라이언트를 식별하는 고유 ID



\* HttpSession - 클라이언트를 식별하는 객체



\* 서버에 접속한 후) 어떤 데이터가 어떤 서버에서 값을 받는지



\* 웹 허브 (With Spring WebMVC 구조도)

