04: 요청과 응답

학습 목표

- 서블릿 라이프 사이클에 대해 알아본다.
- HttpServletRequest 와 HttpServletResponse 에 대해 살펴본다
- HTTP 메소드 (GET/POST)에 대해 알아본다.
- 리다이렉트와 디스페치의 차이에 대해 알아본다

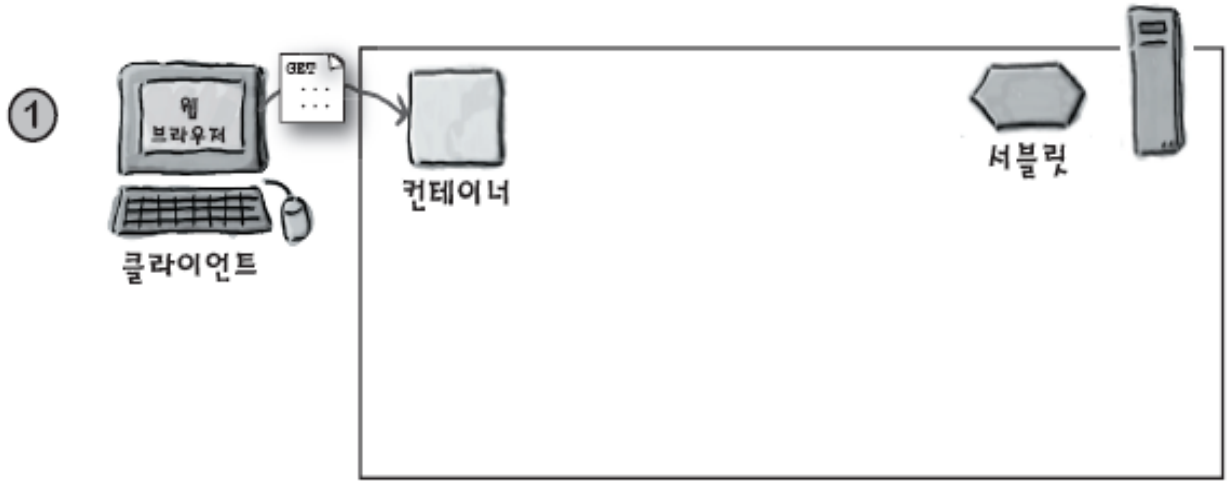
1.서블릿 라이프 사이클

2.HttpServletRequest / HttpServletResponse

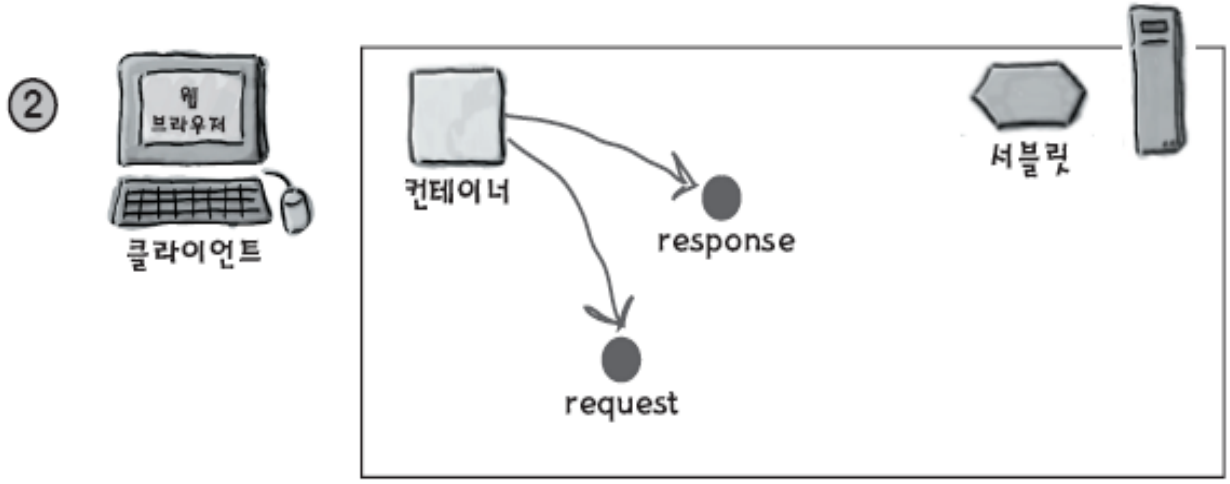
3.HTTP 메소드 : GET / POST

4. 리다이렉트 / 디스패치

서블릿 라이프 사이클 >> 실행 순서



사용자가 서블릿에 대한 링크(URL)를 클릭합니다.



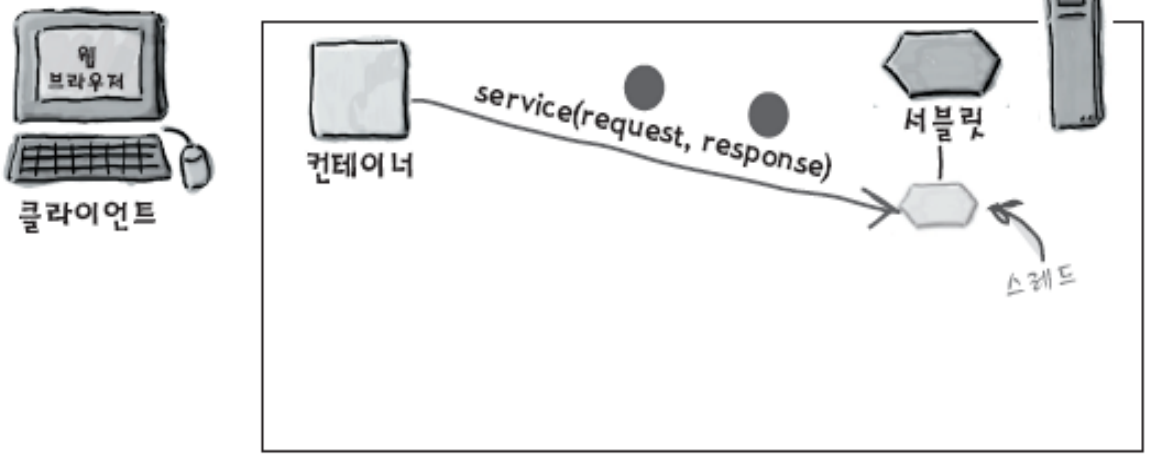
컨테이너는 요청된 Request가 서블릿이라는 것을 간파하고는 다음 두 개의 객체를 생성합니다.

1) HttpServletResponse

2) HttpServletRequest

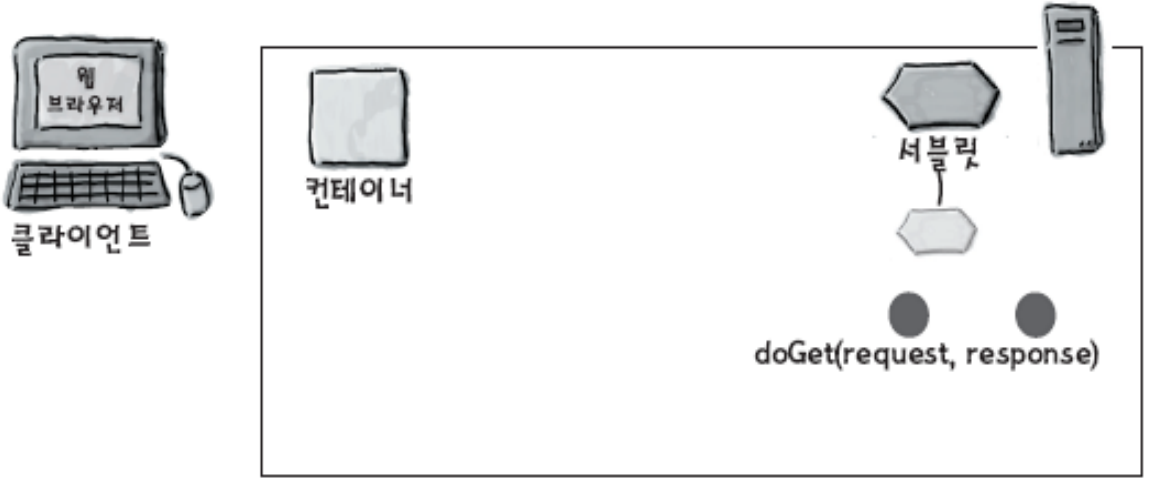
서블릿 라이프 사이클 >> 실행 순서

③



접수한 요청의 URL을 분석하여 어떤 서블릿을 요청했는지 파악합니다(여기서 DD를 참조하지요). 그 다음 해당 서블릿 스레드를 생성하여 Request, Response 객체 참조를 넘깁니다.

④

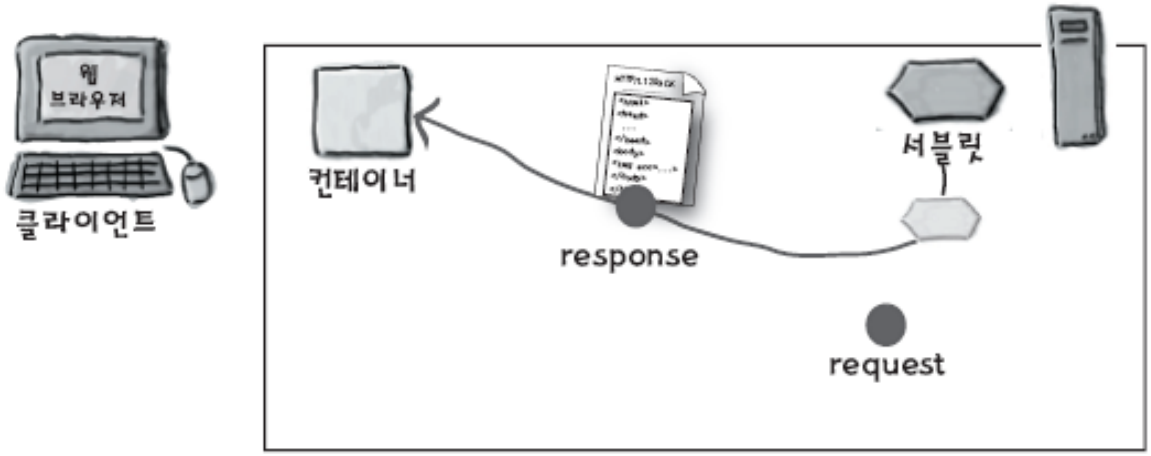


컨테이너는 서블릿 service() 메소드를 호출합니다. 브라우저에서 지정한 방식에 따라 doGet()을 호출할지, doPost()를 호출할지 결정합니다.

클라이언트가 HTTP GET 메소드를 날렸다면, service() 메소드는 서블릿의 doGet() 메소드를 호출합니다. 호출할 때 Request와 Response 객체를 인자로 넘깁니다.

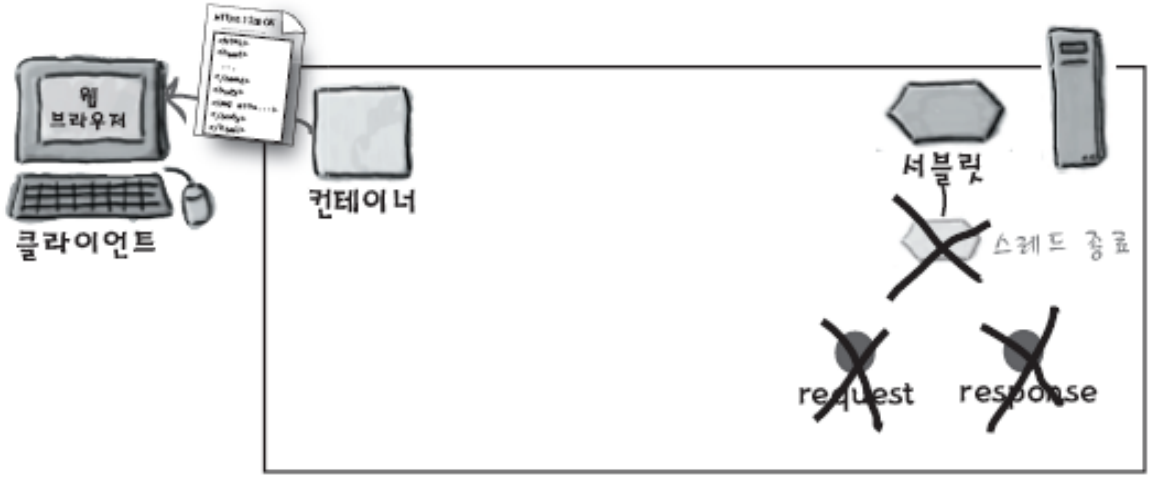
서블릿 라이프 사이클 >> 실행 순서

⑤



서블릿은 클라이언트에게 응답을 작성(write)하기 위하여 Response 객체를 사용합니다. 이 작업을 완료하면, Response에 대한 제어는 컨테이너에게 넘어갑니다.

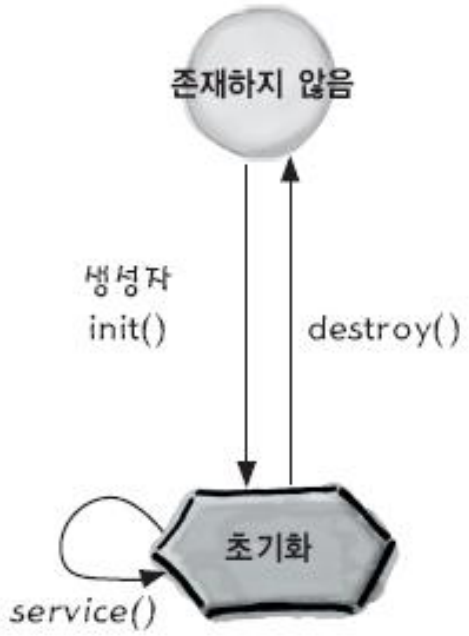
⑥

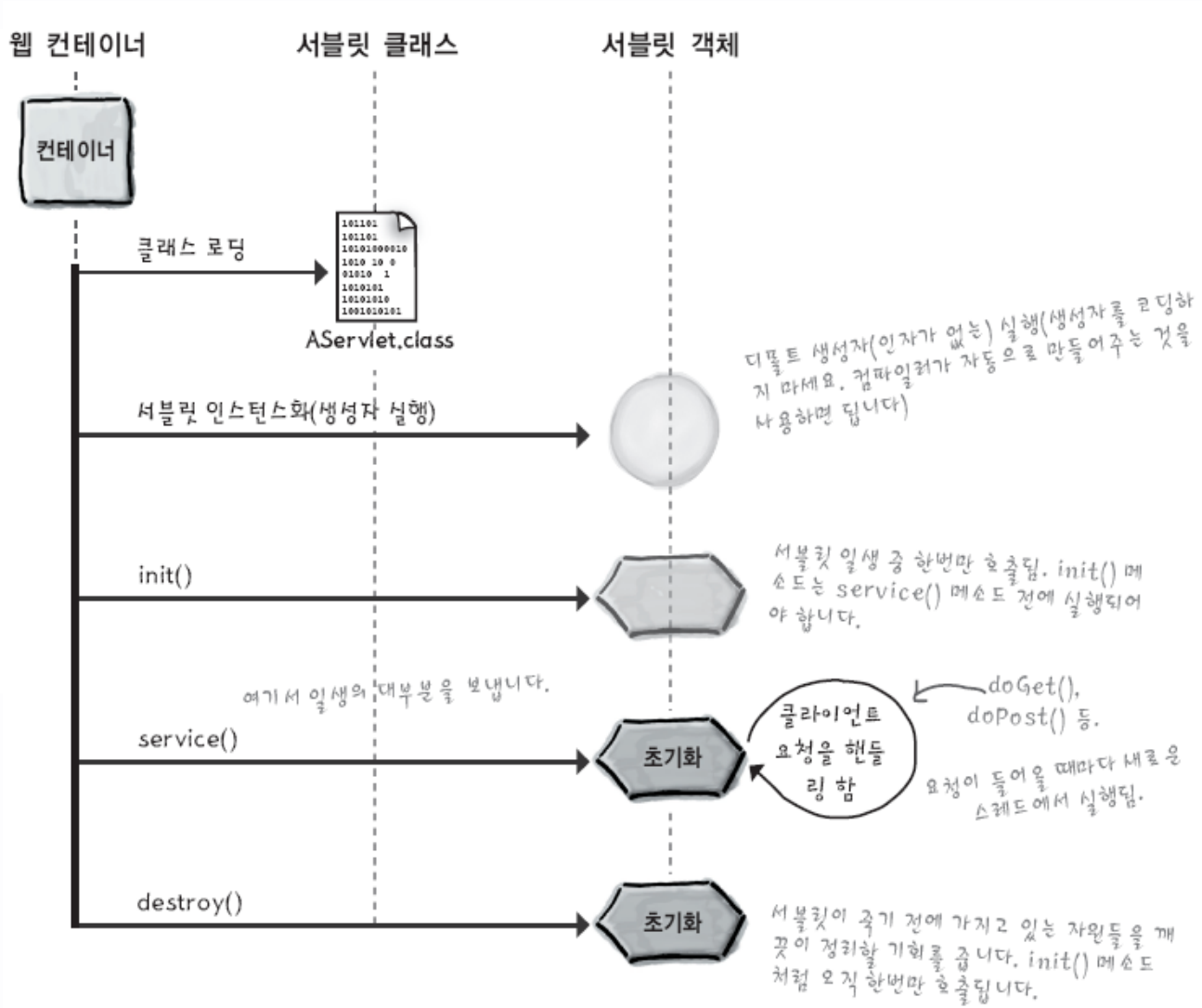


service() 메소드가 끝나면, 스레드를 소멸하거나 아니면 컨테이너가 관리하는 스레드 풀(Pool)로 돌려 보냅니다. 그 다음 Request와 Response 객체는 가비지 컬렉션이 될 준비를 할 것이며, 이 객체에 대한 참조는 이제 범위를 벗어나기에 사라집니다.

마지막으로 클라이언트는 서버로부터 응답을 받게 됩니다.

서블릿 라이프 사이클 >> 초기화





서블릿 라이프 사이클 >> 중요 메소드

init()

- 컨테이너 에서 서블릿 객체를 생성한 다음에 호출한다. **service()** 이전에 실행
- 서블릿을 초기화
- 초기화할 내용(**DB** 접속 등)이 있는 경우 재정의

service()

- 클라이언트의 요청 후 컨테이너에서 쓰레드를 이용하여 호출
- 요청의 **HTTP** 메소드(**GET, POST**등)를 참조하여 해당 메소드(**doGet(), doPost()** 등) 호출 판단
- 거의 재정의 하지 않음

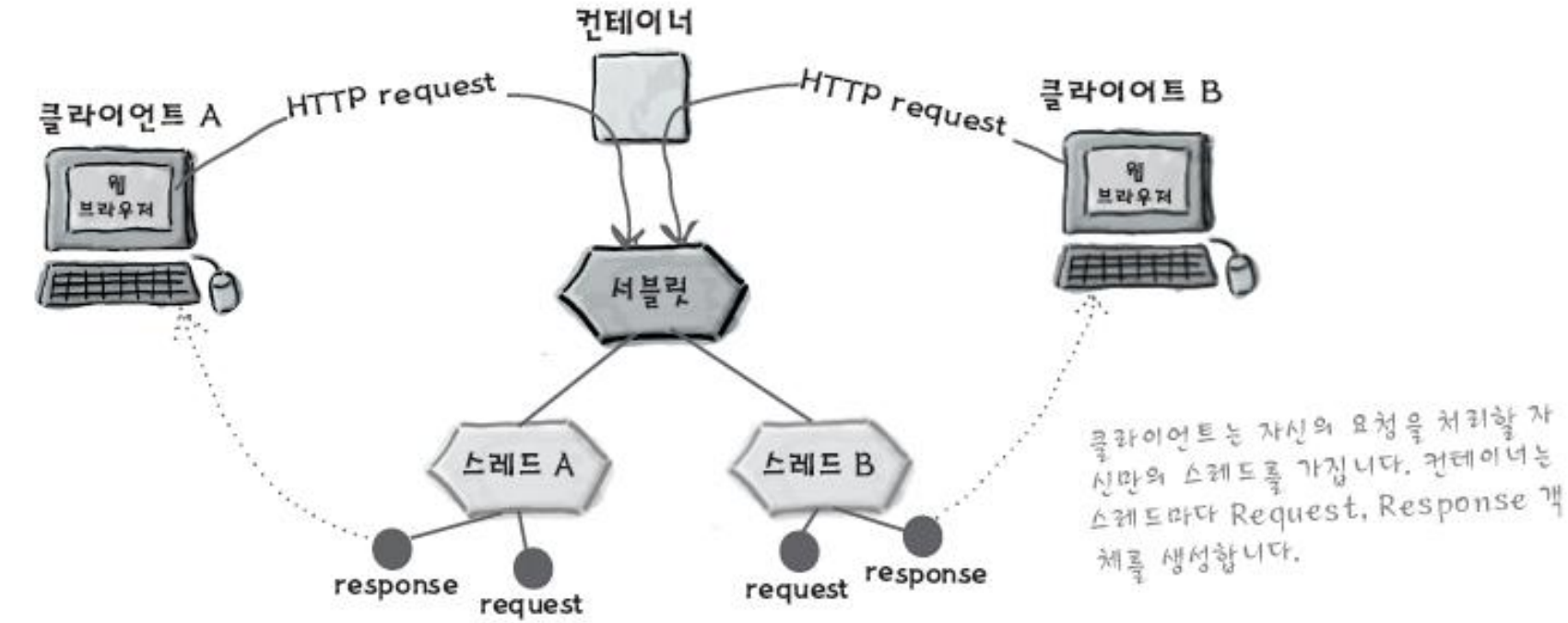
서블릿 라이프 사이클 >> 중요 메소드

doGet() 또는 doPost()

- service() 메소드에서 HTTP 메소드(GET, POST)를 참조하여 호출
- 비즈니스 로직을 구현 또는 호출
- 두 메소드 중 하나는 반드시 재정의하여 구현해야 한다.

서블릿 라이프 사이클 >> 서블릿 쓰레드

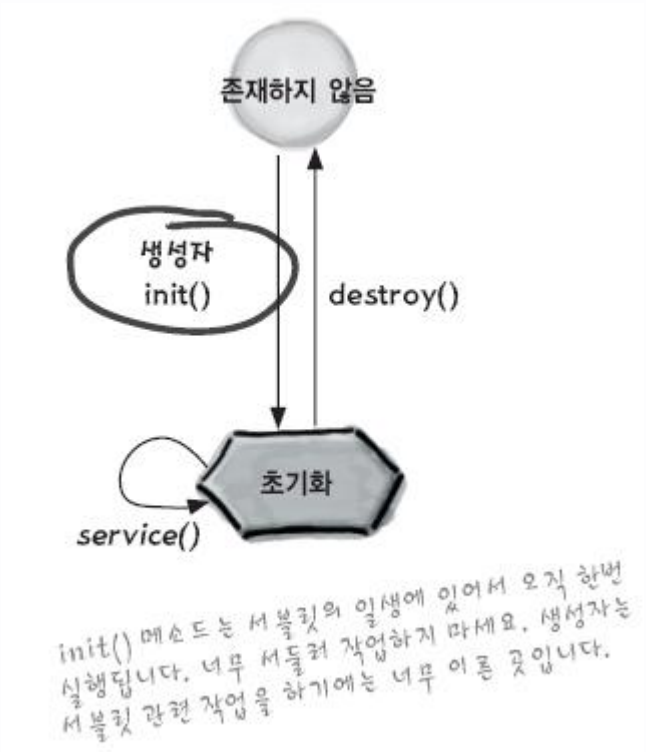
■ 클라이언트의 요청은 서로 다른 쓰레드에서 실행



서블릿 초기화 >> 로딩과 초기화

■ 서블릿 로딩과 초기화

- 컨테이너 시작 시 배포된 웹 애플리케이션 및 서블릿을 확인
- 로딩 : 컨테이너 시작 시 로딩
  또는 최초 클라이언트 요청 시 로딩
- 초기화 : 서블릿 로딩 시 **init()** 를 호출하여
  서블릿을 초기화 함.
- **service()** 메소드는 서블릿 초기화 완료 후에
  실행 가능함.



요청과 응답 >> HttpServletRequest / HttpServletResponse

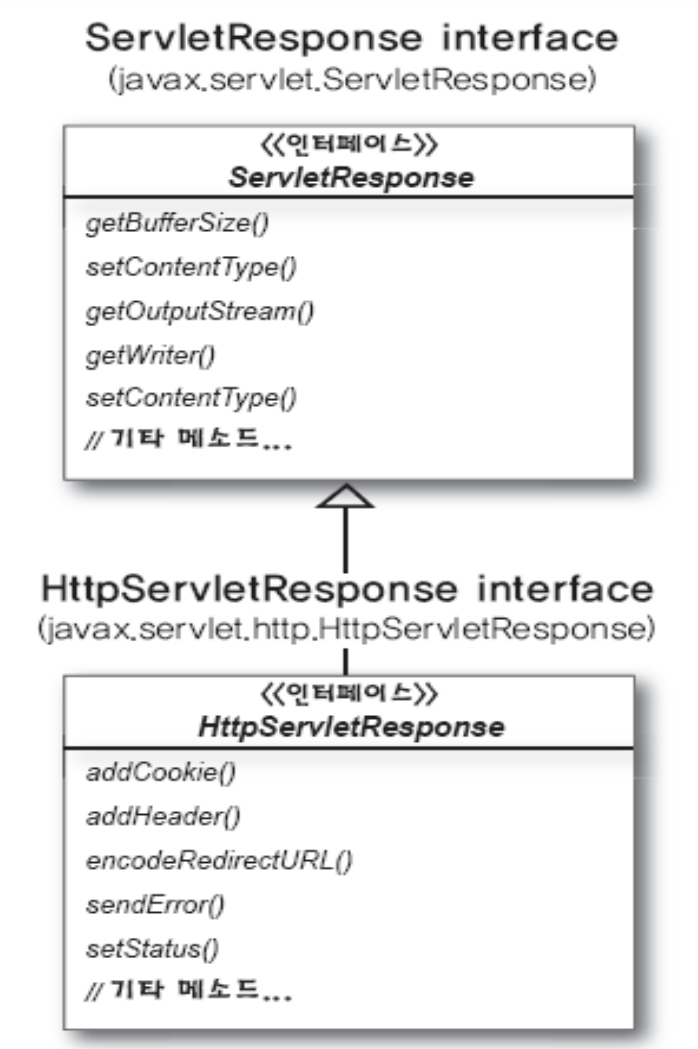
HttpServletRequest

- 쿠키, 헤더, 세션 등 **HTTP** 에 대한 것들을에 대한 처리 관련 메소드 포함
- **HTTP** 프로토콜에 관련된 메소드들이 추가 되어 있음

요청과 응답 >> HttpServletRequest / HttpServletResponse

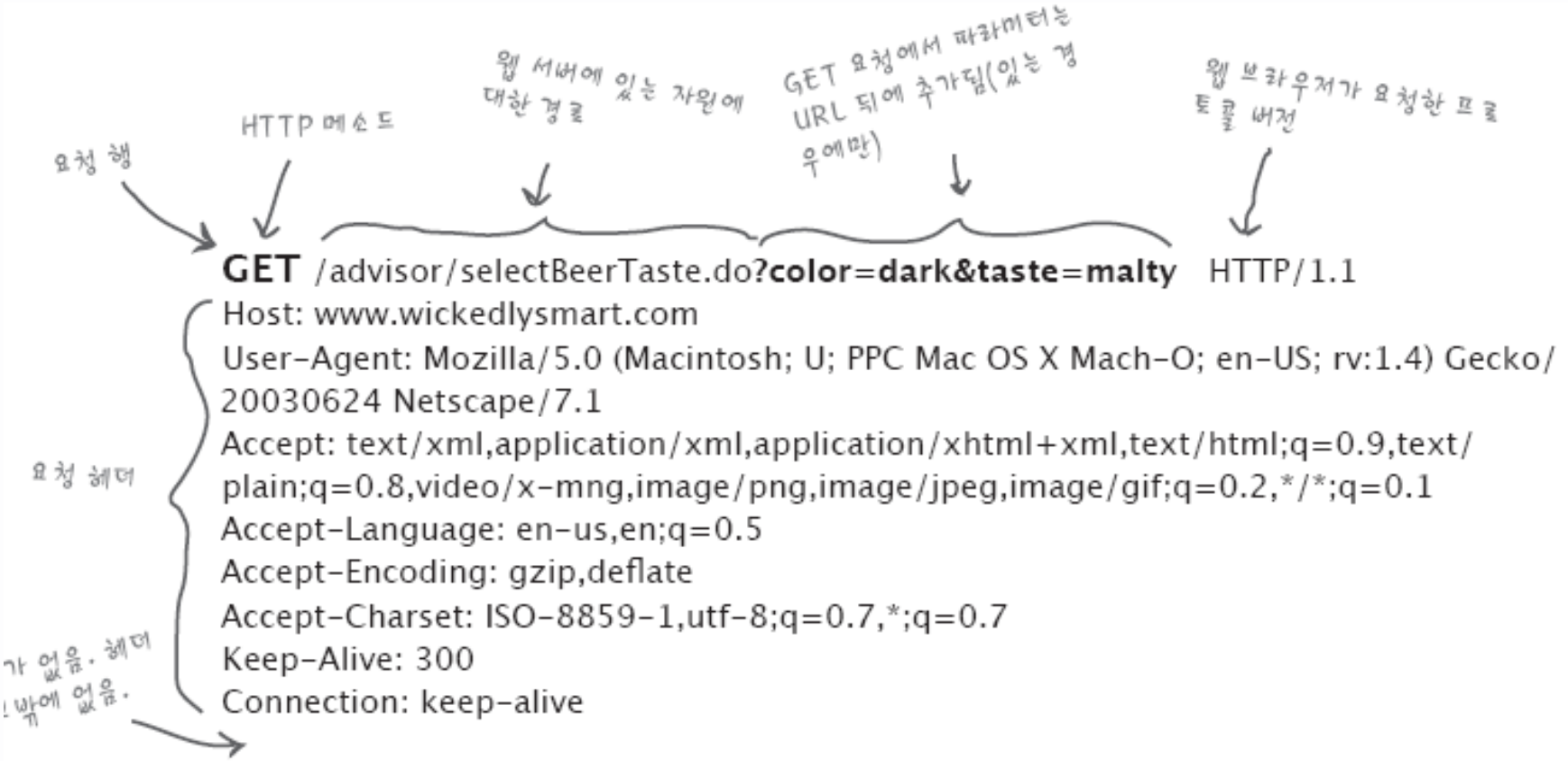
HttpServletResponse

- **HTTP**에 관련된 오류, 쿠키, 헤더 정보에 대한 메소드들이 추가되어 있음



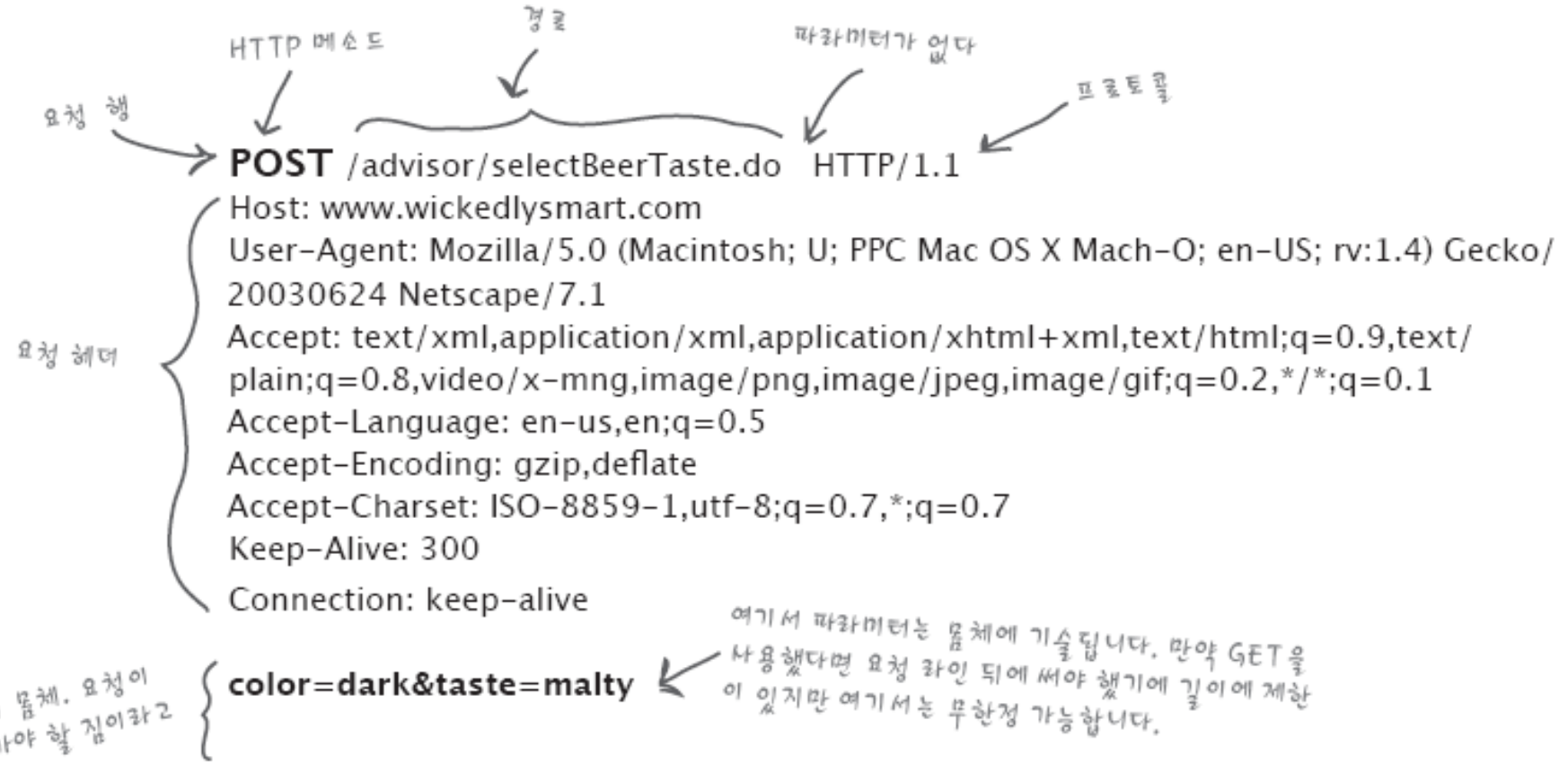
HTTP 메소드 >> GET 과 POST

■ GET메소드



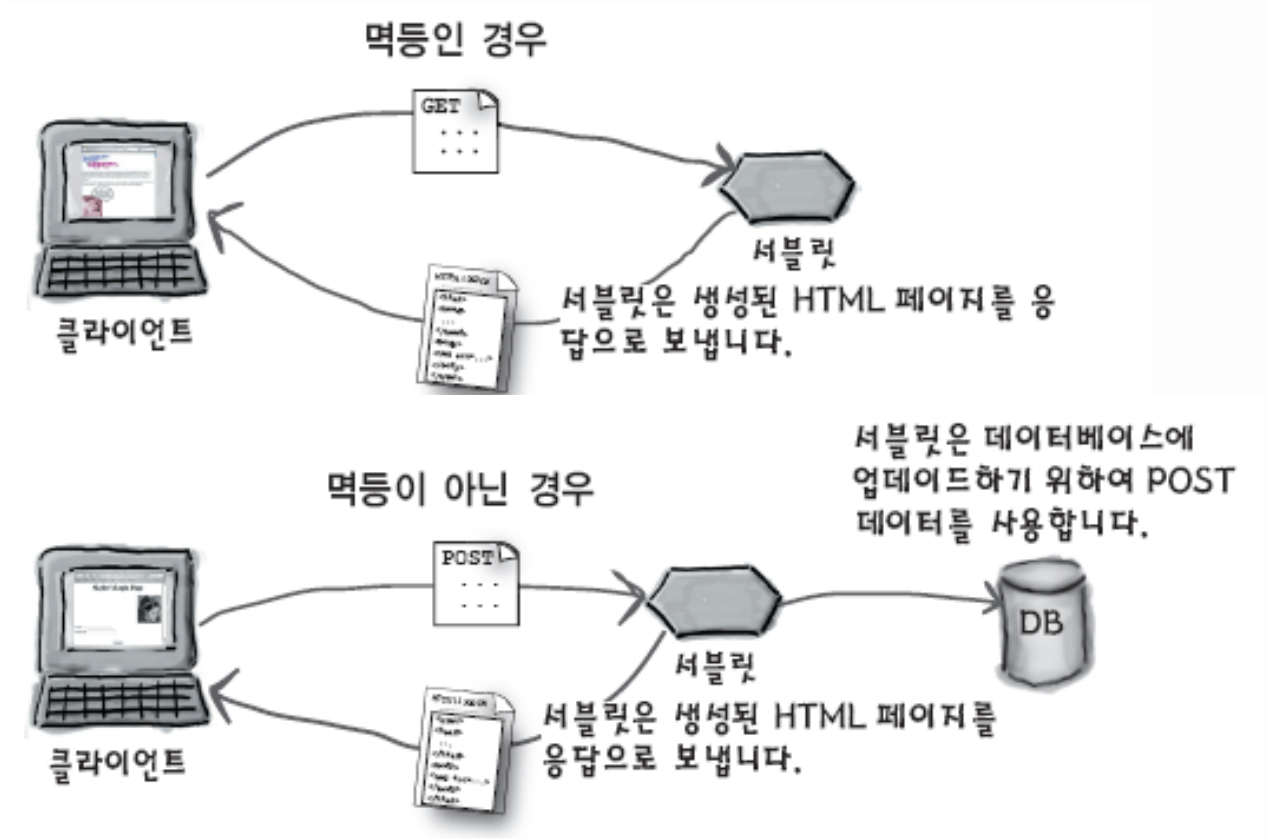
HTTP 메소드 >> GET 과 POST

■ POST메소드



HTTP 메소드 >> GET 과 POST

■ 멱등 (Idempotent) : 아무 부작용 없이 동일한 작업을 한번 이상 계속 할 수 있다는 것을 의미



HTTP 메소드 >> GET 과 POST

■ GET 또는 POST 메소드 요청 결정 방법

- **<form >에 method 정의가 빠진 경우 GET 으로 결정이 된다.**

GET

단순한 하이퍼링크는 항상 GET을 의미합니다.

```
<A HREF="http://www.wickedlysmart.com/index.html/">click here</A>
```

POST

method="POST"라고 못박아 버리면, 당연히 POST죠.

```
<form method="POST"  action="SelectBeer.do">
  Select beer characteristics<p>
  <select name="color" size="1">
    <option>light
    <option>amber
    <option>brown
    <option>dark
  </select>
  <center>
    <input type="SUBMIT">
  </center>
</form>
```

Submit 버튼을 클릭하면 POST 요청의 몸체에 파라미터를 추가하여 전송합니다.

여기서는 color라는 파라미터가 되겠네요. 값은 <option> 항목에 있는 light, amber, brown, dark 중 사용자가 선택한 것이 되겠지요.

HttpServletRequest >> 폼 파라미터

HTML 폼

```
<form method="POST"   action="SelectBeer.do">
  Select beer characteristics<p>
  <select name="color" size="1">
    <option>light
    <option>amber
    <option>brown
    <option>dark
  </select>
  <center>
    <input type="SUBMIT">
  </center>
</form>
```

요청 몸체에 네 개의 옵션 값 중 사용자가 선택한 값 하나를 color라는 이름의 파라미터로 전송합니다. 예를 들면, color=amber와 같이 말입니다.

HttpServletRequest >> 폼 파라미터

HTTP POST 요청

```
POST /advisor/SelectBeer.do HTTP/1.1
Host: www.wickedlysmart.com
User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O; en-US; rv:1.4) Gecko/20030624
Netscape/7.1
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/
plain;q=0.8,video/x-mng,image/png,image/jpeg,image/gif;q=0.2,*/*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive

color=dark
```

옆에 코드는 개발자가 작성하는 부분이 아니라, 브라우저가 자동으로 생성하는 것입니다. 단지 서버로 넘어가는 데이터가 이런 내용이다 정도는 봐 두어야 합니다.

서블릿 클래스

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
                                                    throws IOException, ServletException {
    String colorParam = request.getParameter("color");
      // 여기에는 좀더 화려한 코드를 집어넣으면 되겠죠...
}
```

여기서 colorParam 문자 변수에는 dark라는 값이 들어 가겠죠.

폼에 있는 이름과 짝이 맞아야 합니다.

HttpServletRequest >> 폼 파라미터

■ 하나의 파라미터가 여러 개의 값을 갖는 경우

- **getParameterValues()**를 이용하여 배열을 리턴받아야 한다.

```
<form method=POST
  action="SelectBeer.do">
  Select beer characteristics<p>
  Can Sizes: <p>
  <input type=checkbox name=sizes value="12oz"> 12 oz.<br>
  <input type=checkbox name=sizes value="16oz"> 16 oz.<br>
  <input type=checkbox name=sizes value="22oz"> 22 oz.<br>
  <br><br>

  <center>
    <input type="SUBMIT">
  </center>
</form>
```

```
String [] sizes = request.getParameterValues("sizes");
```

HttpServletRequest >> 자주 사용하는 메소드

- 클라이언트 플랫폼 정보 및 브라우저 정보
  - String client = request.getHeader(“User-Agent”);
- Request 에 관련된 쿠키
  - Cookie[] cookies = request.getCookies();
- 클라이언트 세션 정보
  - HttpSession session = request.getSession();
- Request 의 HTTP 메소드
  - String theMethod = request.getMethod();
- Request 의 입력 스트림
  - InputStream input = request.getInputStream();

HttpServletResponse >> 자주 사용하는 메소드

■ 응답 데이터의 데이터 타입 정의

- response.setContentType("text/html");

■ 출력 스트림 사용 : 가급적 JSP 사용 권장

- Writer writer = response.getWriter();
- OutputStream output = response.getOutputStream();

■ 헤더 정보 설정, 쿠키 추가 등

- response.addCookie();
- response.addHeader();

HttpServletResponse >> 바이너리 데이터 보내기

```
// import 구문 한 무더기

public class CodeReturn extends HttpServlet {

   public void doGet(HttpServletRequest request, HttpServletResponse response)
                                                  throws IOException, ServletException {

      response.setContentType("application/jar");

      ServletContext ctx = getServletContext();
      InputStream is = ctx.getResourceAsStream("/bookCode.jar");

      int read = 0;
      byte[] bytes = new byte[1024];

      OutputStream os = response.getOutputStream();
      while ((read = is.read(bytes)) != -1) {
         os.write(bytes, 0, read);
      }
      os.flush();
      os.close();
   }
}
```

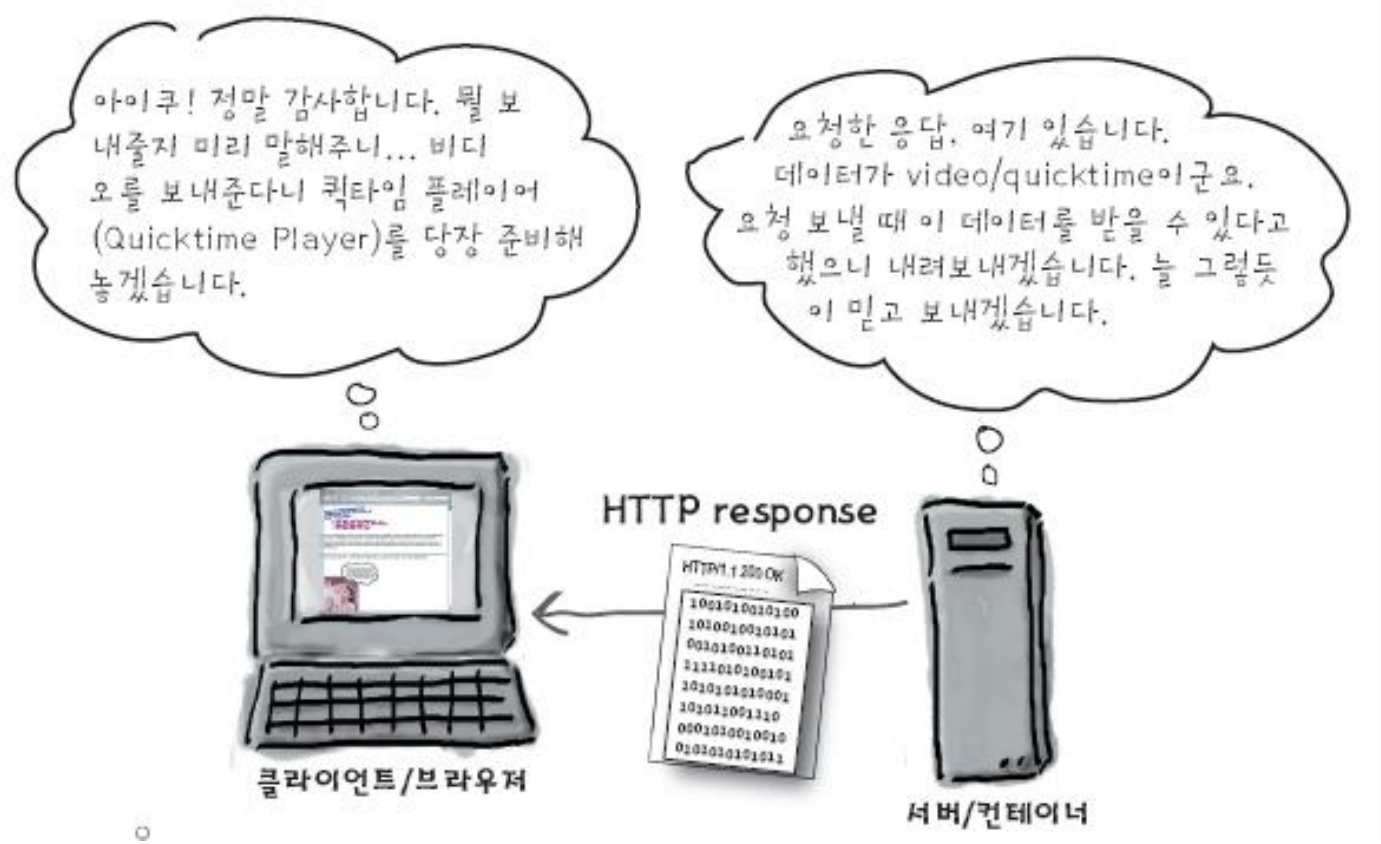
이 코딩의 목적은 브라우저에게 지금 우리가 내려 보내는 것이 JAR라는 사실을 알리려는 것입니다. HTML 페이지가 아니기에 application/jar라고 타입을 설정하는 것입니다.

이 코드를 풀어서 말해보면 "자원 (/bookcode.jar 파일)을 입력 스트림으로 주세요"라는 뜻입니다.

여기가 바로 핵심입니다. 핵심치고는 코딩이 너무 단순한가요? 진짜 특별한 것이 없네요. 늘 보던 전형적인 I/O 코딩이죠. 먼저 JAR 바이트를 읽어, 출력 스트림에 바이트를 기록하는 것뿐입니다.

HttpServletResponse >> 컨텐츠 타입(MIME 타입)

- 응답 데이터의 데이터 타입 정의
- HTTP 헤더 정보 중의 하나

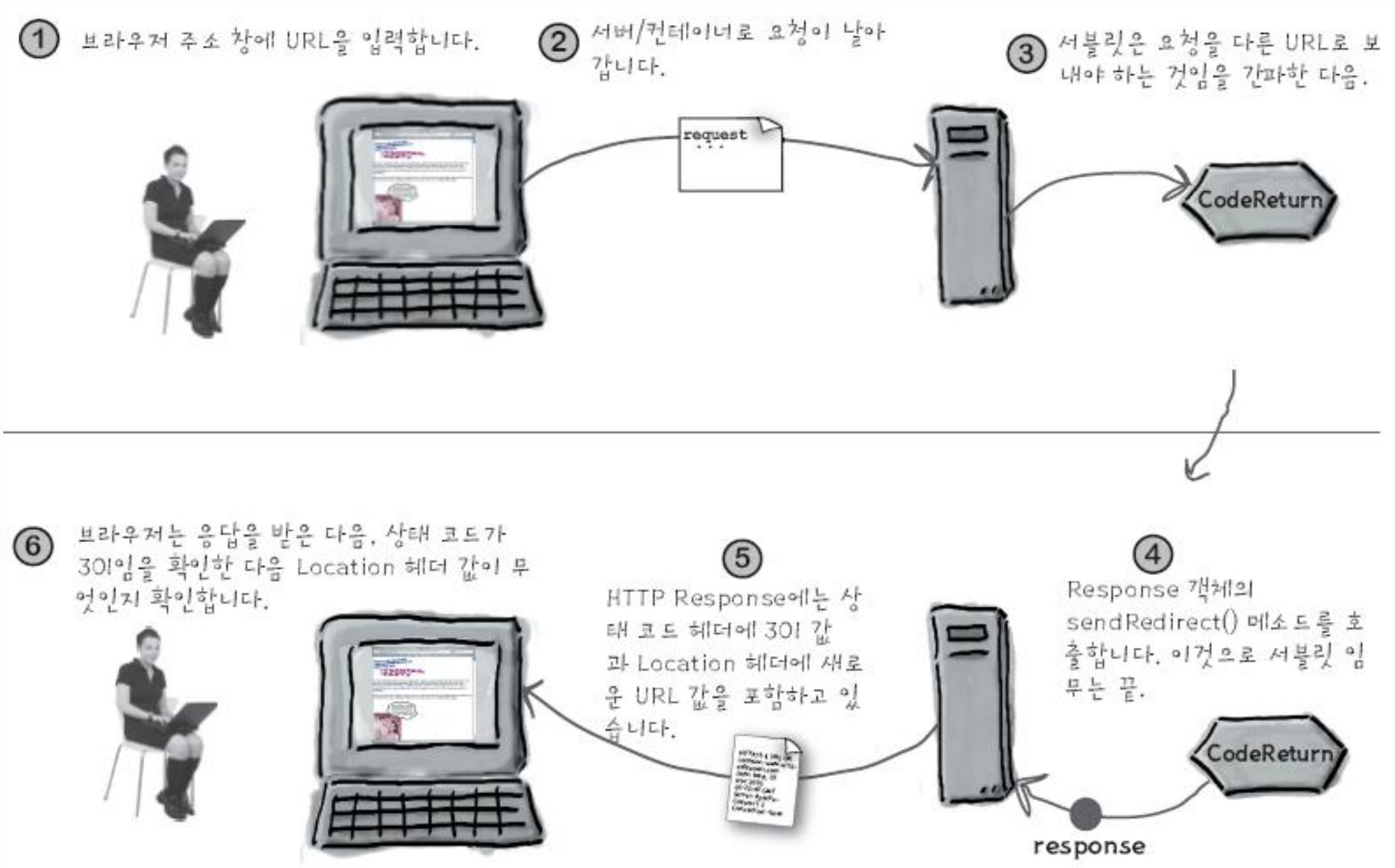


HttpServletResponse >> 컨텐츠 타입(MIME 타입)

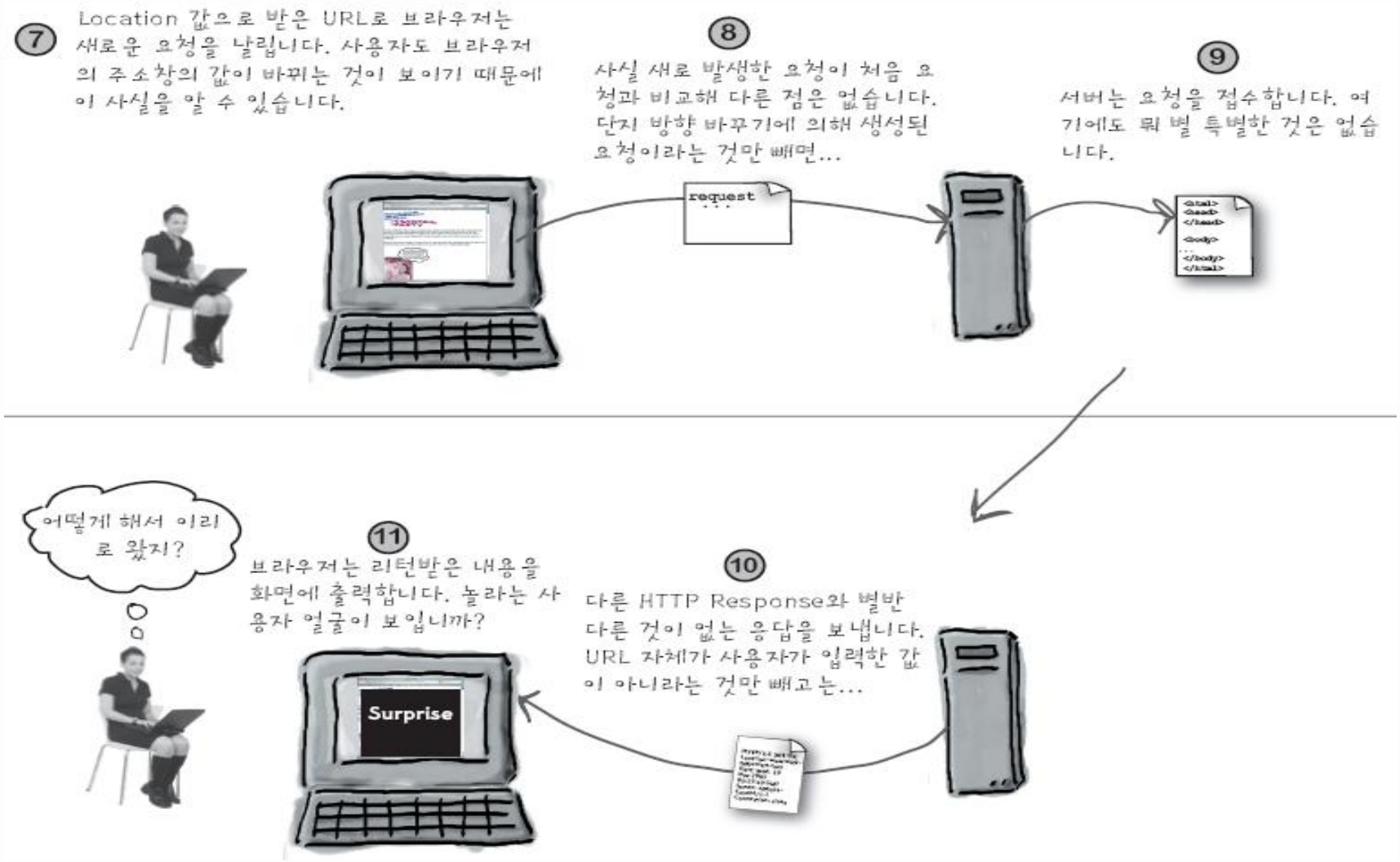
종류

- text/html
- application/pdf
- video/quicktime
- image/jpeg
- application/octet-stream
- application/x-zip

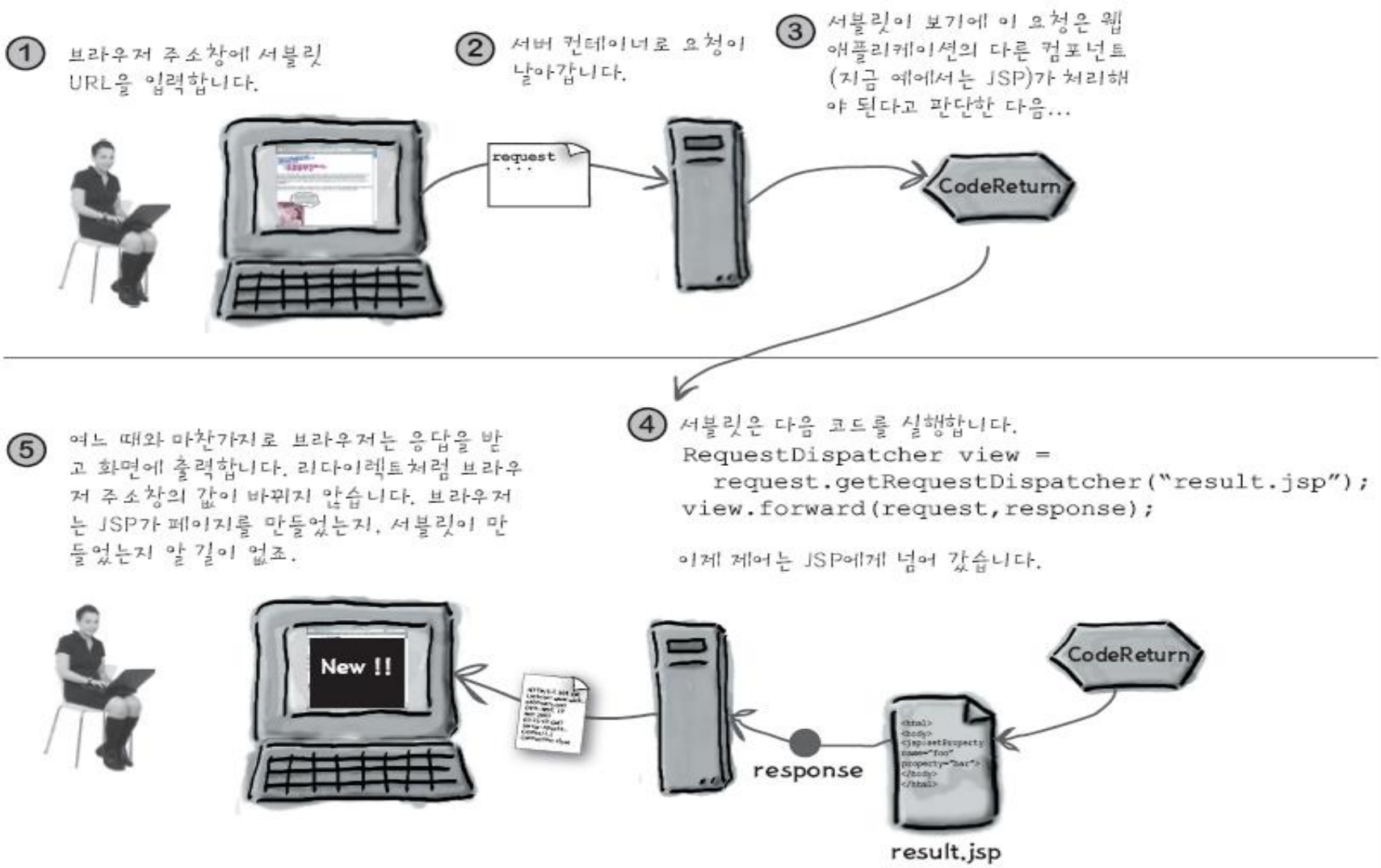
HttpServletResponse >> 리다이렉트 (Redirect)



HttpServletResponse >> 리다이렉트 (Redirect)



HttpServletResponse >> 디스패치 (Dispatch)



HttpServletResponse >> 리다이렉트 와 디시패치

■ 리다이렉트 (Redirect)

- 브라우저에서 작업
- **Response** 객체의 쓰기 작업을 한 후에 **sendRedirect()** 를 할 수 없다.
- **sendRedirect()**의 매개변수로 **URL(String** 객체) 를 사용한다.
- 브라우저의 **URL** 이 변화가 됨

■ 디스패치 (Dispatch)

- 서버에서 작업
- 브라우저의 **URL** 변화가 없음