

Introduction to Web Application Technologies

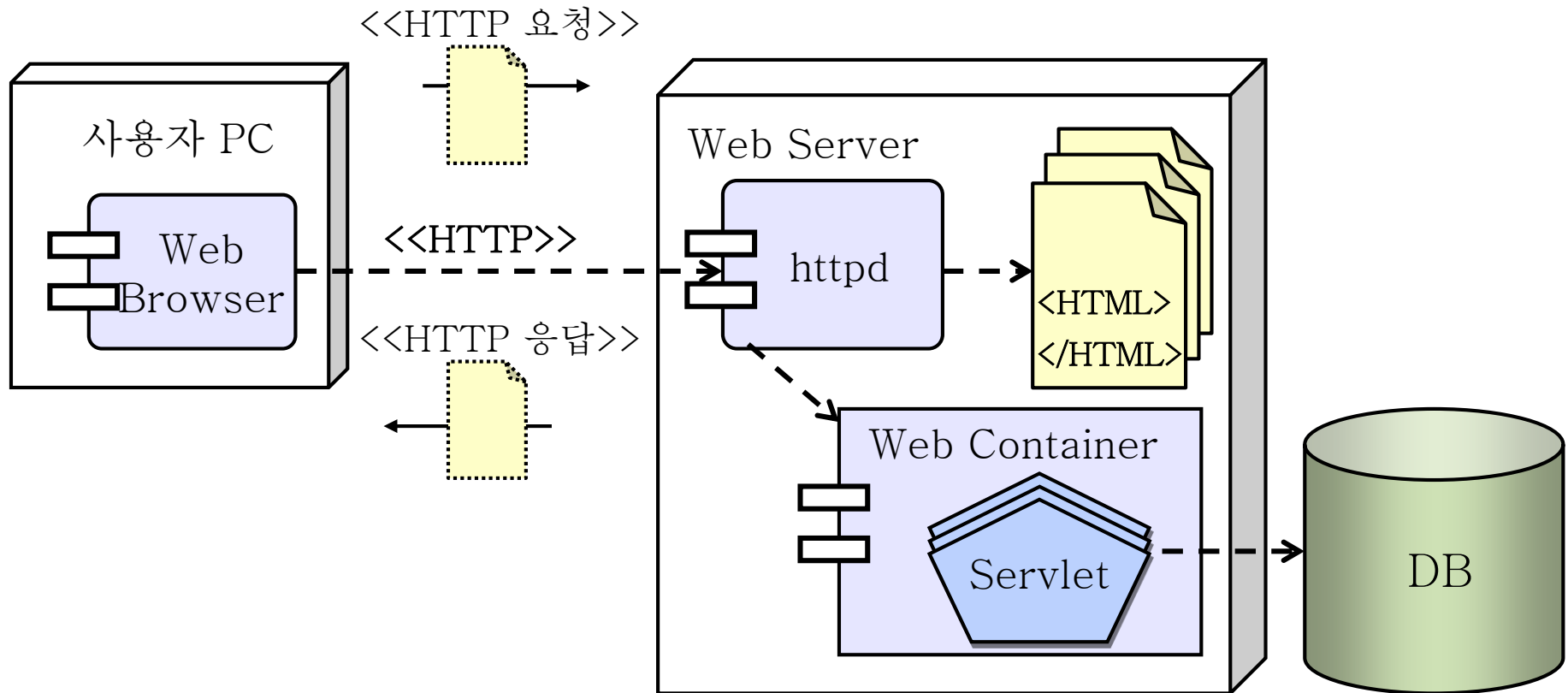


Figure 1/3

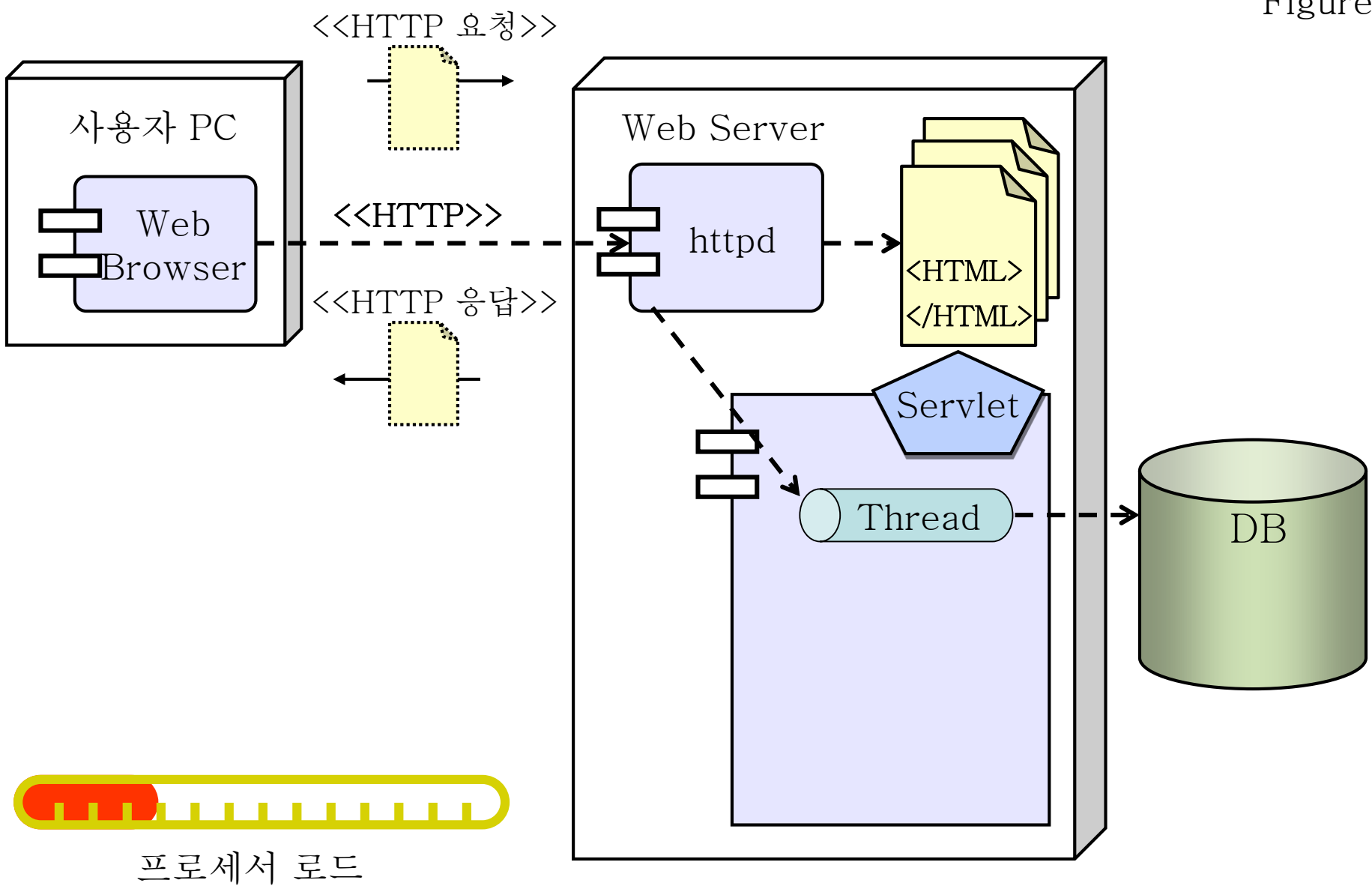


Figure 2/3

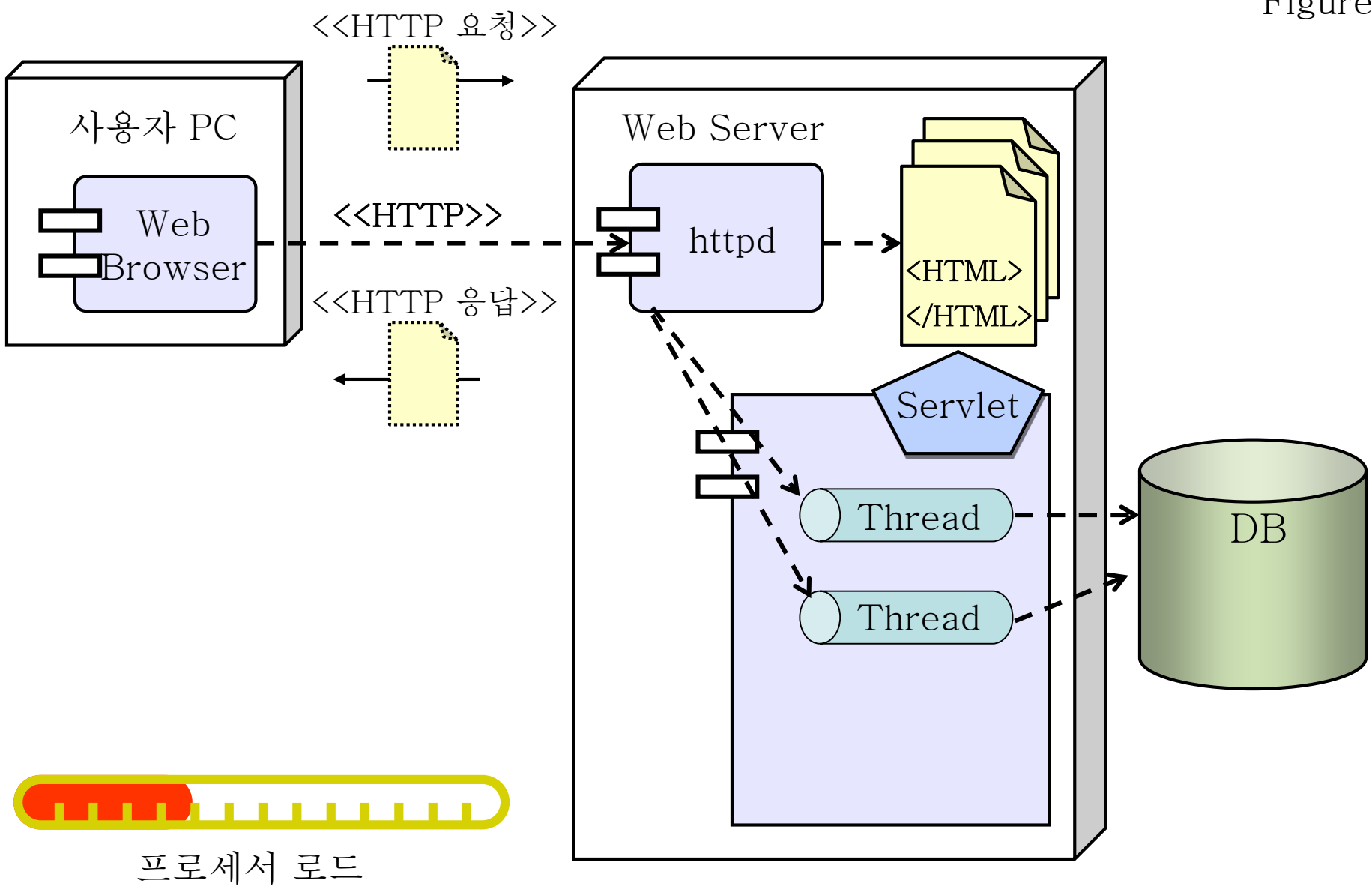
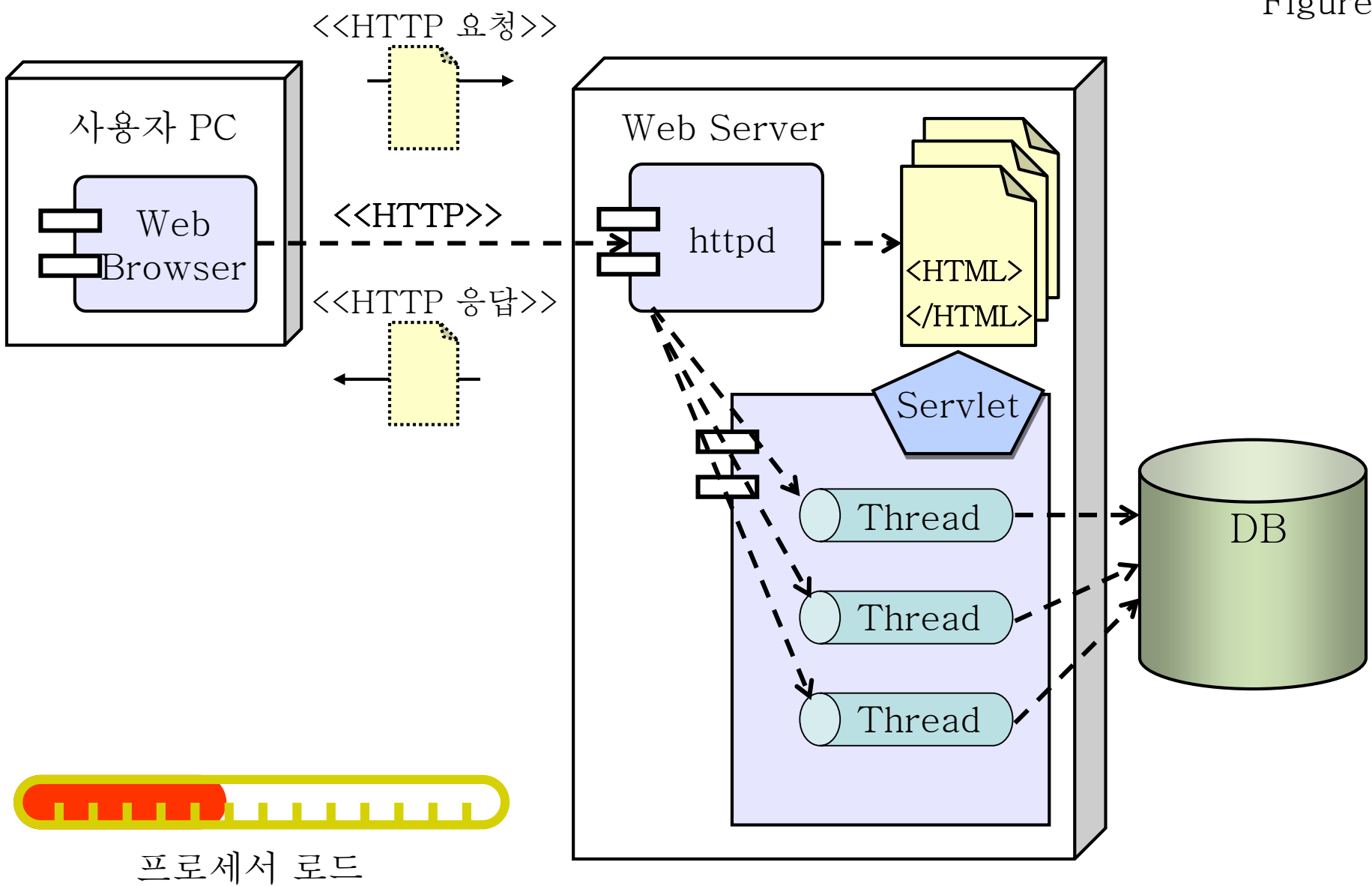


Figure 3/3



□ Java Servlet의 장점

- 각 요청마다 다른 스레드로 실행되기 때문에 서블릿 요청의 처리 속도가 기존 CGI 처리 속도보다 빠르다.
- 확장성이 있다.
- 견고하고, 객체 지향적이다.
- Java 언어로서만 구현된다.
- Platform independent하다.
- Logging 기능에 access할 수 있다.
- 웹 컨테이너가 Servlet에 오류 처리나 보안과 같은 추가 서비스를 제공한다.

□ Java Servlet의 단점

- 비즈니스 로직과 프리젠테이션 로직을 구분하기 힘들기 때문에 웹 응답을 생성하는데 사용하기 어렵다. → JSP (Java Server Page)로 해결
- 동시성 문제를 고려해야 한다.

□ Template Page

- Html내에 코드를 포함하는 방법

□ Html내에 코드를 포함하는 기술

- PHP : Apache → Apache에서만 동작
- ASP (Active Server Page) : Microsoft → IIS에서만 동작
- JSP (Java Server Page) : Sun Microsystems → 웹 서버 독립적

□ JSP

- Java 코드를 포함하는 Html
- JSP page는 Web Container에 의해 서블릿 인스턴스로 변환 되며, 변환된 서블릿은 해당 JSP page에 대한 요청을 처리한다. → 결국 Servlet
- ASP나 PHP page는 HTTP요청이 들어올 때 마다 interpret 되는 반면, JSP는 Java bytecode로 컴파일된다.
- Servlet과 JSP의 주된 차이점은 JSP가 Presentation logic에 포커스를 맞추고 있다는 점이다.

□ JSP의 장점

- OS의 셸이나 프로세스가 아닌 스레드가 사용되기 때문에 웹 응용프로그램의 성능과 확장성이 있다.
- Java를 기반으로 하기 때문에 Platform independent하다.
- Java 언어로 작성되기 때문에 객체지향언어와 모든 API를 사용할 수 있다.

□ JSP의 단점

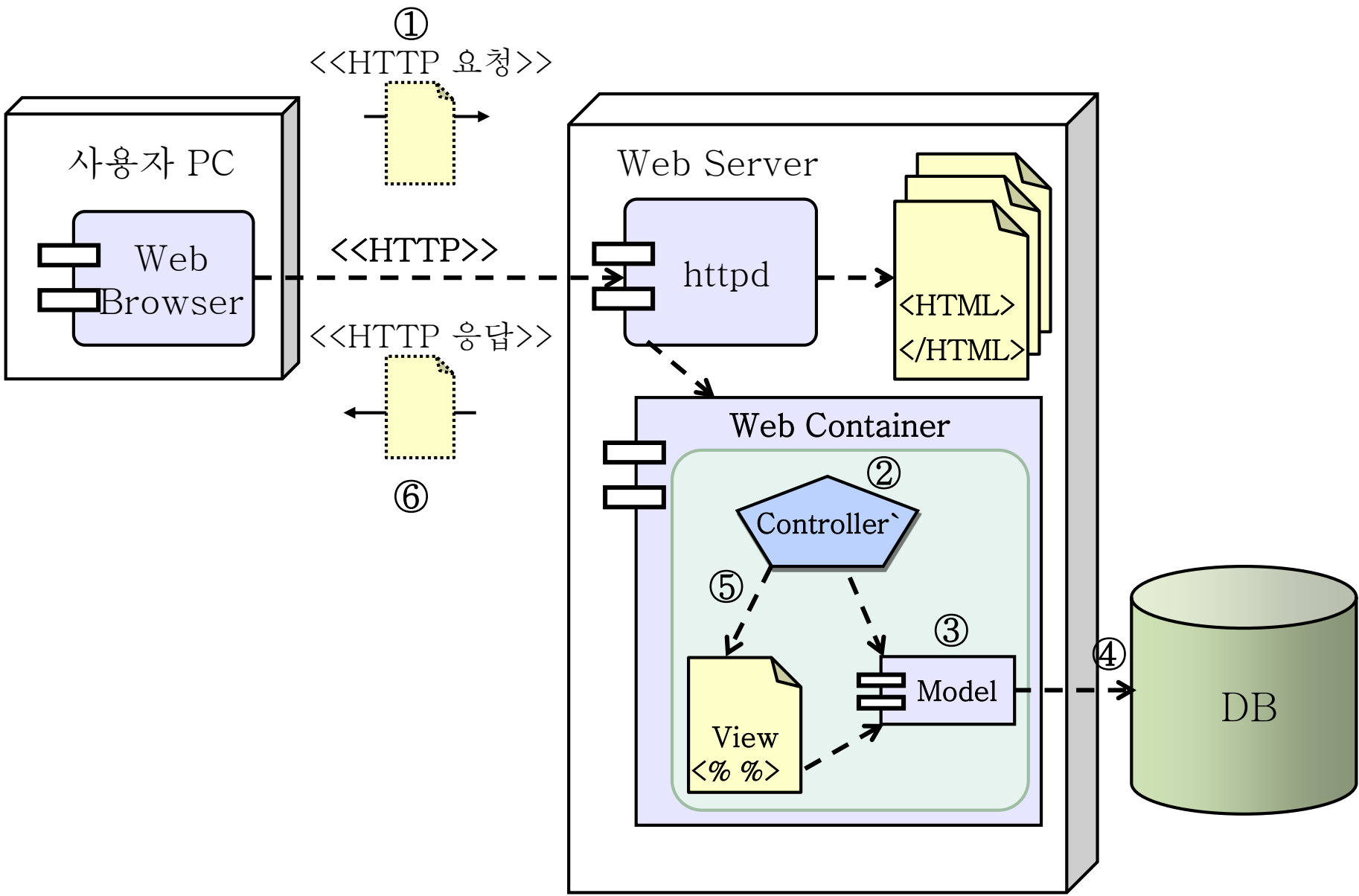
- Presentation logic과 Business logic을 같이 포함하는 경우가 있다.
- 동시성 문제를 고려해야 한다.
- Debugging이 어렵다.

□ Model-View-Controller (MVC) design

- 소프트웨어 시스템을 세 가지 타입의 구조 컴포넌트로 분할하는 소프트웨어 패턴
- Model : 비즈니스 로직 및 도메인 객체
- View : 사용자 인터페이스(UI)
- Controller : Model을 변경하여 UI에 대한 사용자 입력을 해석

□ Using servlets and JSP pages (Model2 Architecture)

- servlet : Controller 역할. 폼 데이터를 검증하고, 폼 데이터를 사용하여 모델을 갱신하고, 응답으로 다음에 보여줄 View를 선택한다.
- JSP page : View 역할. Model로부터 응답을 생성하는데 필요한 데이터를 꺼내서 HTML응답을 만들어내고, 사용자 상호 작용이 가능한 HTML폼을 제공
- Java 클래스 : Model 역할. 웹 응용프로그램의 비즈니스 로직을 구현



□ JSP/servlet을 이용한 Model 2 Architecture의 특징

- 빠른 속도
- 강력한 기능
- 쉬운 개발 (단, 숙련된 자바 개발자의 경우)
- 플랫폼 간 호환성
- 확장성
- 효율적인 유지 보수 가능

□ HTTP Request는 다음의 세 가지 요소로 구성된다.

- 메서드 – URI – Protocol / Version (GET 또는 POST)
- 요청 헤더 (Request Headers)
- 문서 본체 (Entity Body)

```
POST /servlet/SimpleServlet HTTP/1.1
```

```
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
```

```
Referer: http://192.168.142.2/welcome.html
```

```
Accept-Language: ko
```

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

```
Accept-Encoding: gzip, deflate
```

```
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
```

```
Host: 192.168.142.2
```

```
Content-Length: 8
```

```
Connection: Keep-Alive
```

```
name=kim
```

□ GET 메소드

- 요청에 대한 모든 정보를 요청-URI에 담아서 보내는 메소드
- 웹 브라우저의 주소창이나 hyper link를 클릭하는 것은 모두 GET 메소드로 요청하는 것이다.
- 특별히 메소드를 지정하기 않는 경우 모두 GET 메소드 요청이다.

```
GET /welcome.html HTTP/1.1
```

```
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,  
application/x-shockwave-flash, */*
```

```
Accept-Language: ko
```

```
Accept-Encoding: gzip, deflate
```

```
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
```

```
Host: 192.168.0.100:8080
```

```
Connection: Keep-Alive
```

HTTP/1.1 200 OK

ETag: W/"164-1142261479151"

Last-Modified: Mon, 13 Mar 2006 14:51:19 GMT

Content-Type: text/html

Content-Length: 164

Date: Mon, 13 Mar 2006 14:53:05 GMT

Server: Apache-Coyote/1.1

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">

<title>Your First Page</title>

</head>

<body>

<h1>Welcome!!!</h1>

</body>

</html>

□ HTTP Response는 다음의 세 가지 요소로 구성된다.

- Protocol / Version - 상태코드 - 설명
- 응답 헤더 (Response Headers)
- 문서 본체 (Entity Body)

HTTP/1.1 200 OK

Content-Type: text/html; charset=euc-kr

Content-Length: 520

Date: Mon, 27 Feb 2006 14:28:27 GMT

Server: Apache-Coyote/1.1

<html>

<head>

<title>Hello</title>

</head>

<body>

Welcome to this page : kim

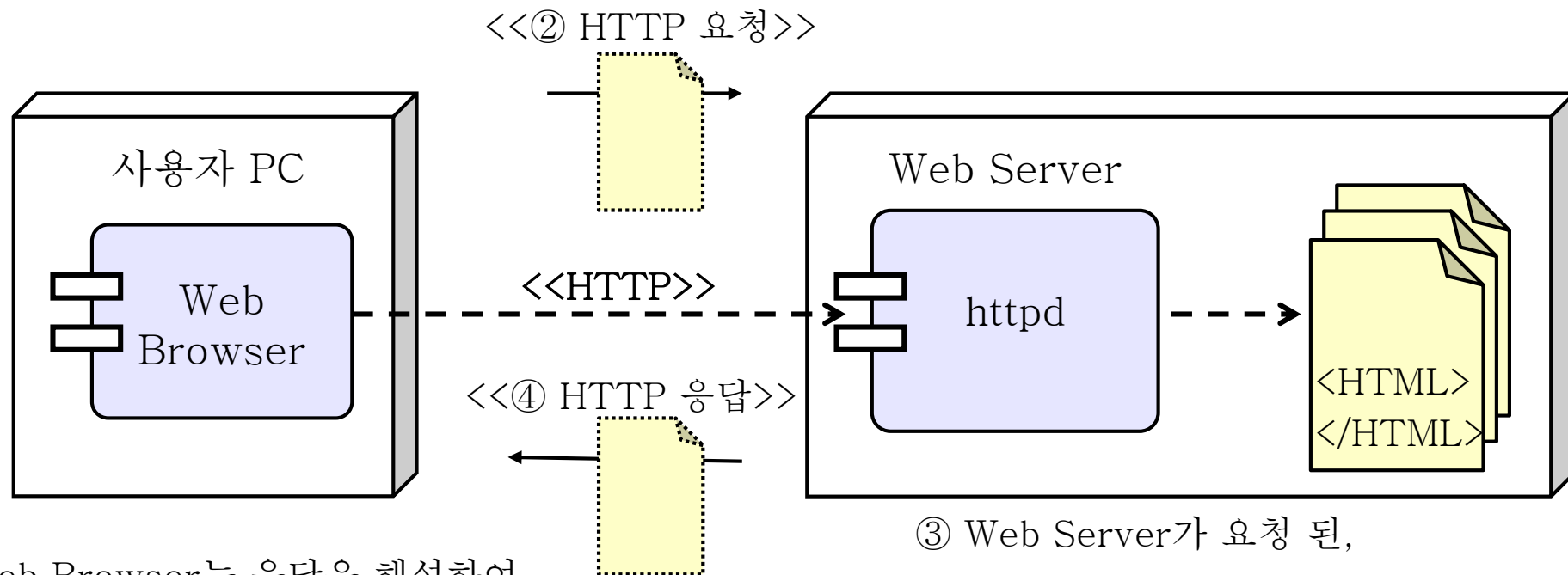
</body>

</html>

- 서버가 처리한 결과를 알려주는 세 자리 정수 코드
- 첫 번째 자리 숫자는 응답 종류에 대한 분류 기호이며, 나머지 두 자리 숫자는 일련 번호
 - 1xx : Informatinal (향후의 사용을 위해 예약. 아직 사용되지 않음)
 - 2xx : Success (성공적으로 수신되고 해독되고 처리된 경우)
 - 3xx : Redirection (완전한 처리를 위해 추가적인 동작이 필요로 하는 경우)
 - 4xx : Client Error
(요구 메시지에 글자상의 문제가 있는 경우거나 메시지를 처리할 수 없을 때)
 - 5xx : Server Error (서버가 요구 메시지를 처리하는 가운데 문제가 발생한 경우)
- 예
 - 200 : OK
 - 304 : Not Modified
 - 404 : Not Found
 - 500 : Internal Server Error

① Web Browser에서 요청을 한다.

`http://192.168.0.100:8080/welcome.html`

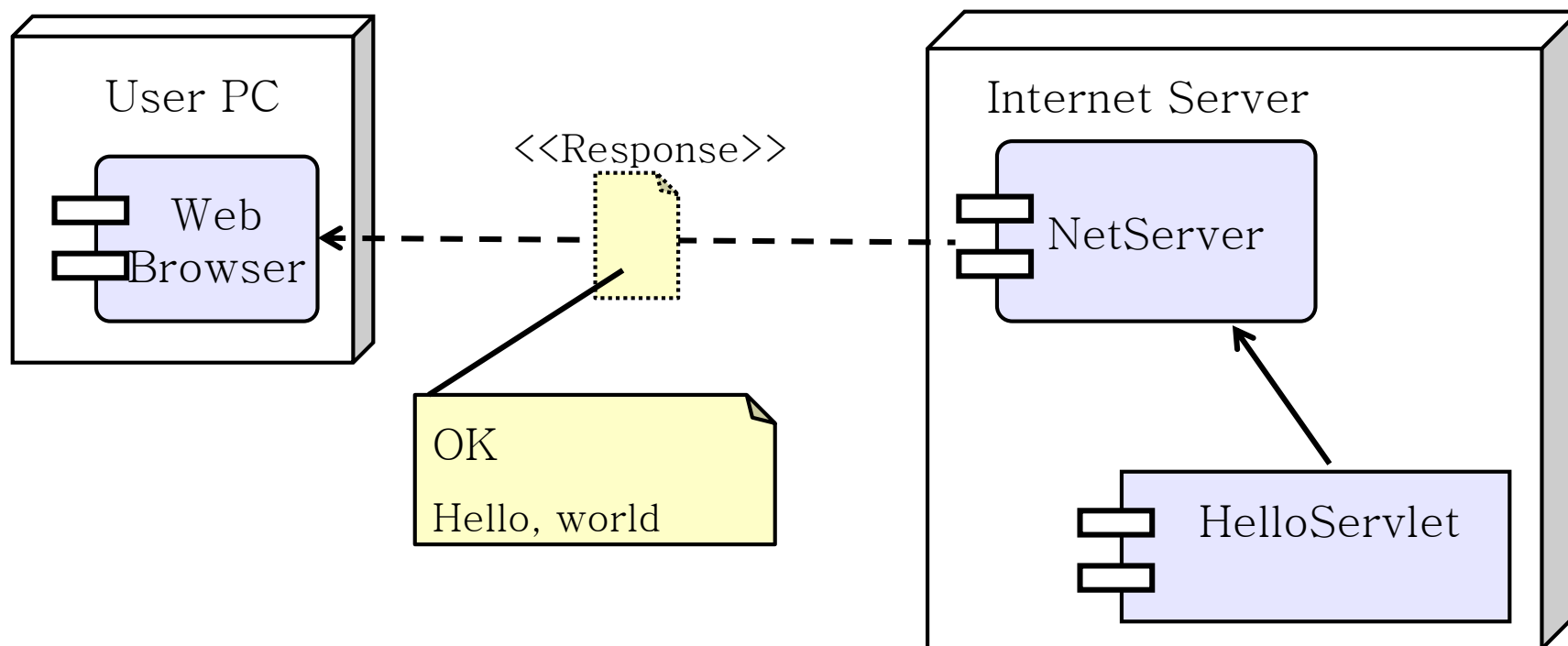


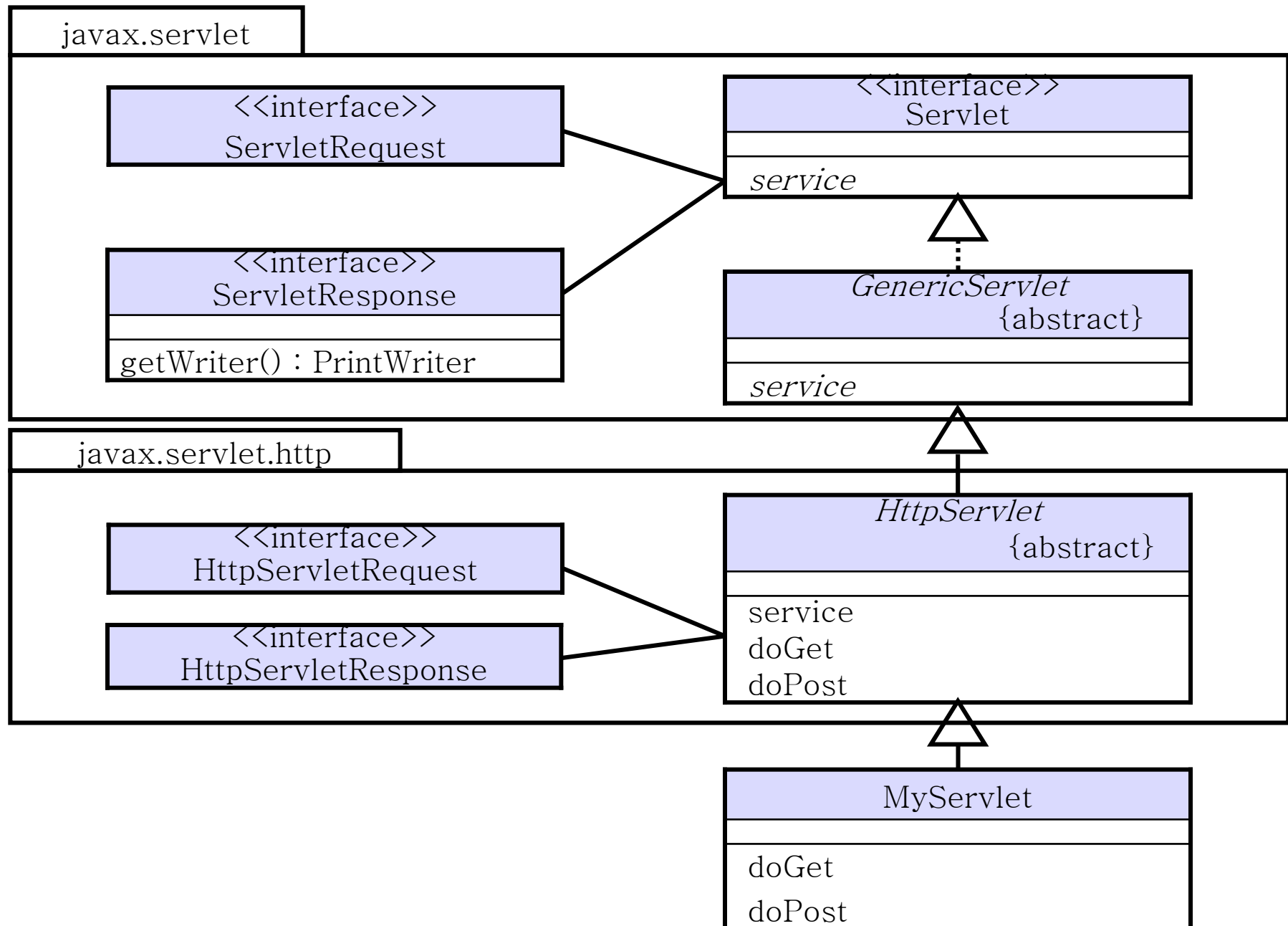
⑤ Web Browser는 응답을 해석하여,
사용자에게 해당 문서를 보여 준다.

③ Web Server가 요청 된,
`welcome.html` 문서를 검색한다.

□ GenericServlet

- 웹(WWW)은 HTTP 프로토콜에서 작동한다.
- HTTP 프로토콜이 모든 웹 응용프로그램의 기반이기는 하지만, 클라이언트-서버 인터넷 서비스를 실행할 수 있는 유일한 것은 아니다.
- 사용자 정의 프로토콜 생성이 가능하다.
- Java servlet 명세는 사용자 정의 프로토콜을 사용하는 servlet을 생성할 수 있는 API를 제공한다.





Servlet 작성하기

1. 필요한 Package import

```
import javax.servlet.*;  
import javax.servlet.http.*;
```

2. HttpServlet 을 상속받는 클래스 작성

```
public class MyServlet extends HttpServlet { }
```

3. doGet() 또는 doPost() Overriding

```
public class MyServlet extends HttpServlet {  
    public void doGet( HttpServletRequest req,  
                      HttpServletResponse res ) {  
    }  
}
```

4. 응답 스트림을 작성하고자 할 때는, HttpServletResponse의 getWriter(); 를 호출하여 스트림을 얻는다.

```
PrintWriter out = res.getWriter();
```

```
package chap02;

import java.io.*;          // PrintWriter
import javax.servlet.*;    // ServletException
import javax.servlet.http.*;

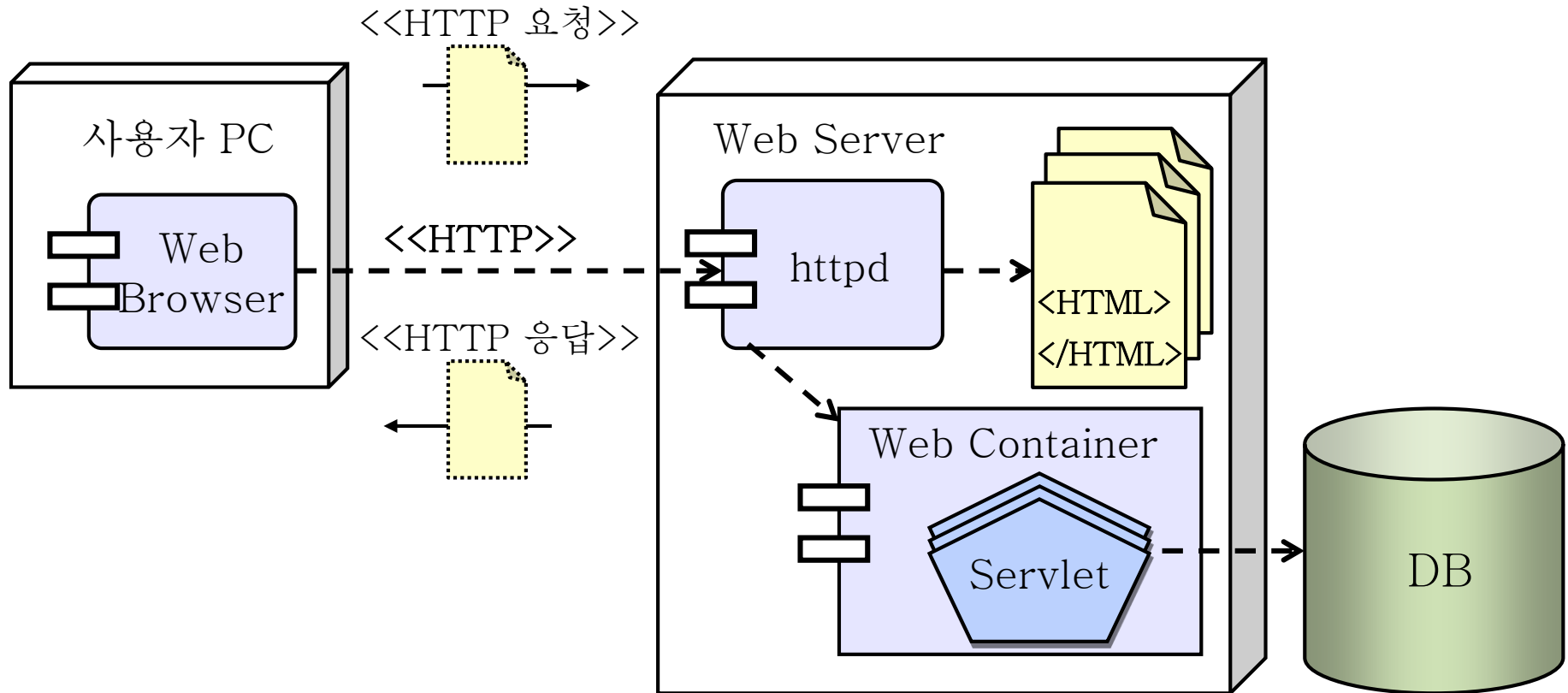
public class WelcomeServlet extends HttpServlet {
    public void doGet( HttpServletRequest req,
                      HttpServletResponse res )
        throws ServletException, IOException {
        res.setContentType("text/html;charset=euc-kr");

        PrintWriter out = res.getWriter();
        out.println( "<html>" );
        out.println( "<head>" );
        out.println( "<title>Your First Page</title>" );
        out.println( "</head>" );
        out.println( "<body>" );
        out.println( "<h1>Welcome, Servlet!!!</h1>" );
        out.println( "</body>" );
        out.println( "</html>" );
        out.close();
    }
}
```

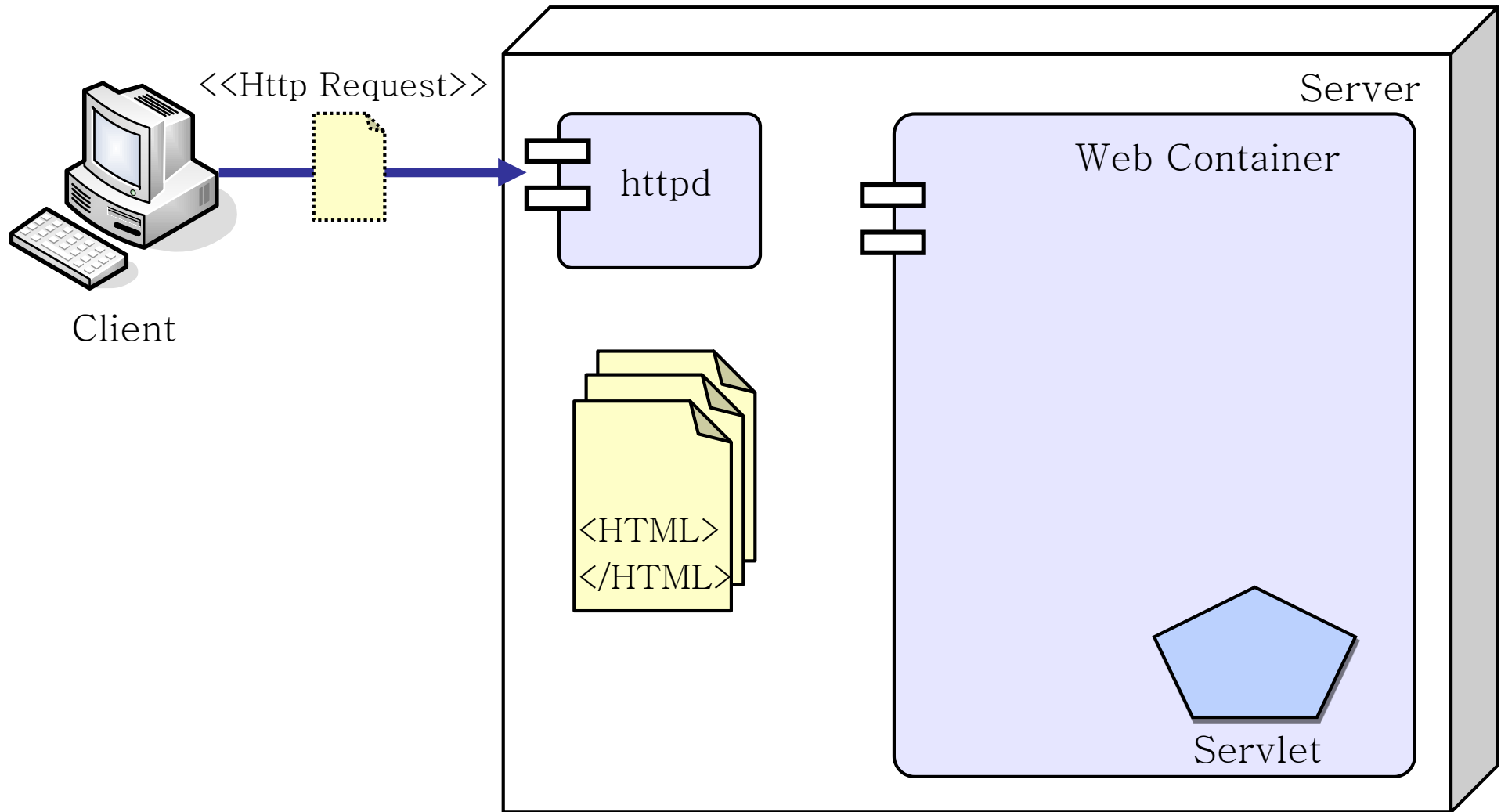
- ❑ HTTP 요청 정보는 HttpServletRequest에 의해 캡슐화 된다.
 - `getHeaderNames()` : 요청 스트림에 있는 각 헤더의 이름으로 이루어진 문자열을 return한다.
 - `getHeader()` : 특정 헤더 값 검색
 - `getIntHeader()`, `getDateHeader()` : 일부 헤더 값은 정수, 날짜를 표현하는 문자열이다. 이러한 정보를 자동으로 변환하여 return하는 메소드들이다.

- ❑ Browser로부터 온 요청 정보는 HttpServletRequest 객체로 생성되고, 객체를 통하여 요청(request)정보를 알 수 있다.

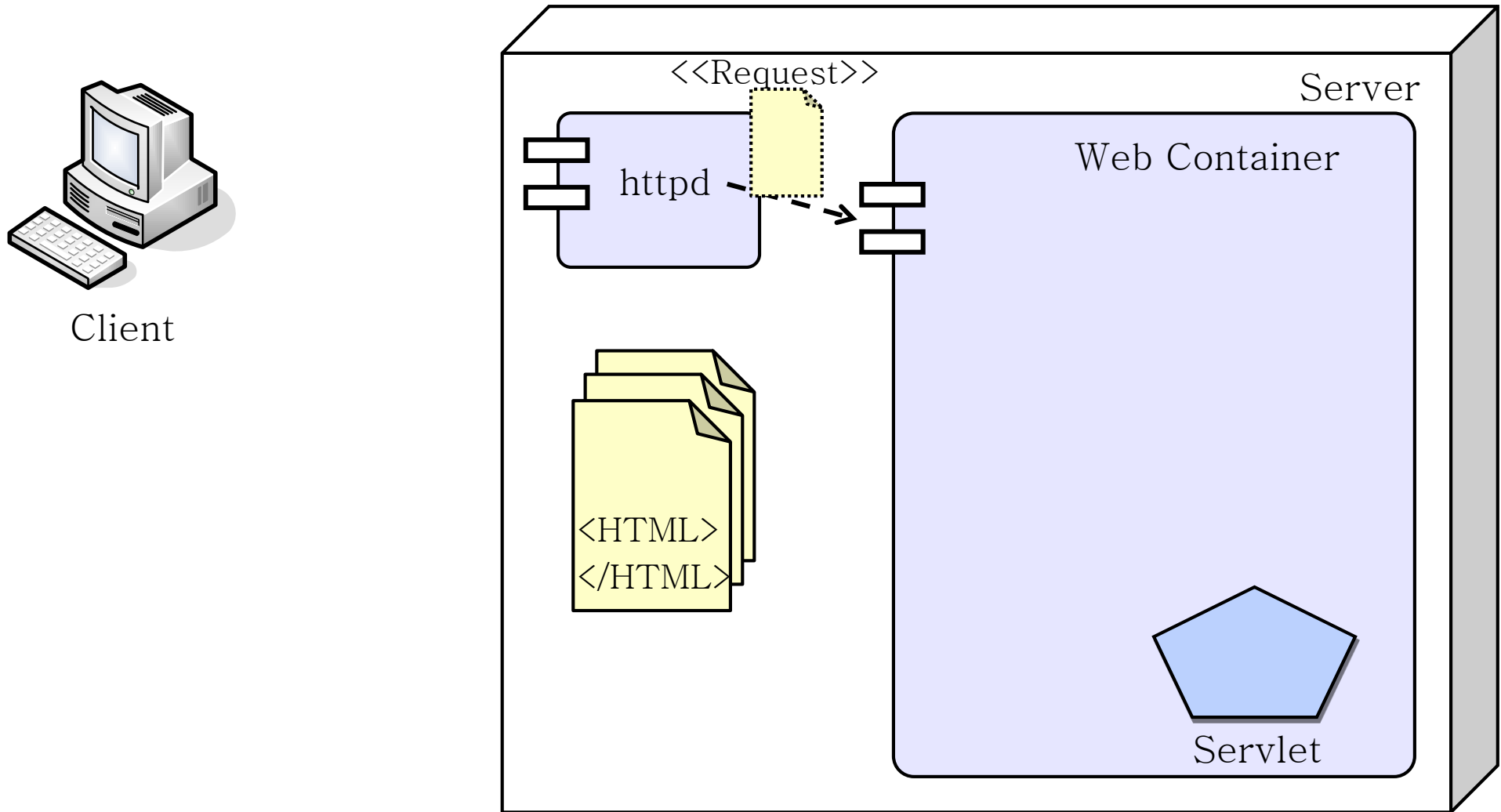
- ❑ HTTP 응답 정보는 HttpServletResponse에 의해 캡슐화 된다.
 - `setContentType()` : MIME 타입 설정
ex) `response.setContentType("text/html;charset=euc-kr");`
 - `getWriter()` : HTML응답 생성 스트림(PrintWriter)객체를 얻는다.
- ❑ Browser로부터 온 요청에 대한 응답을 생성할 때 HttpServletResponse 객체를 사용한다.



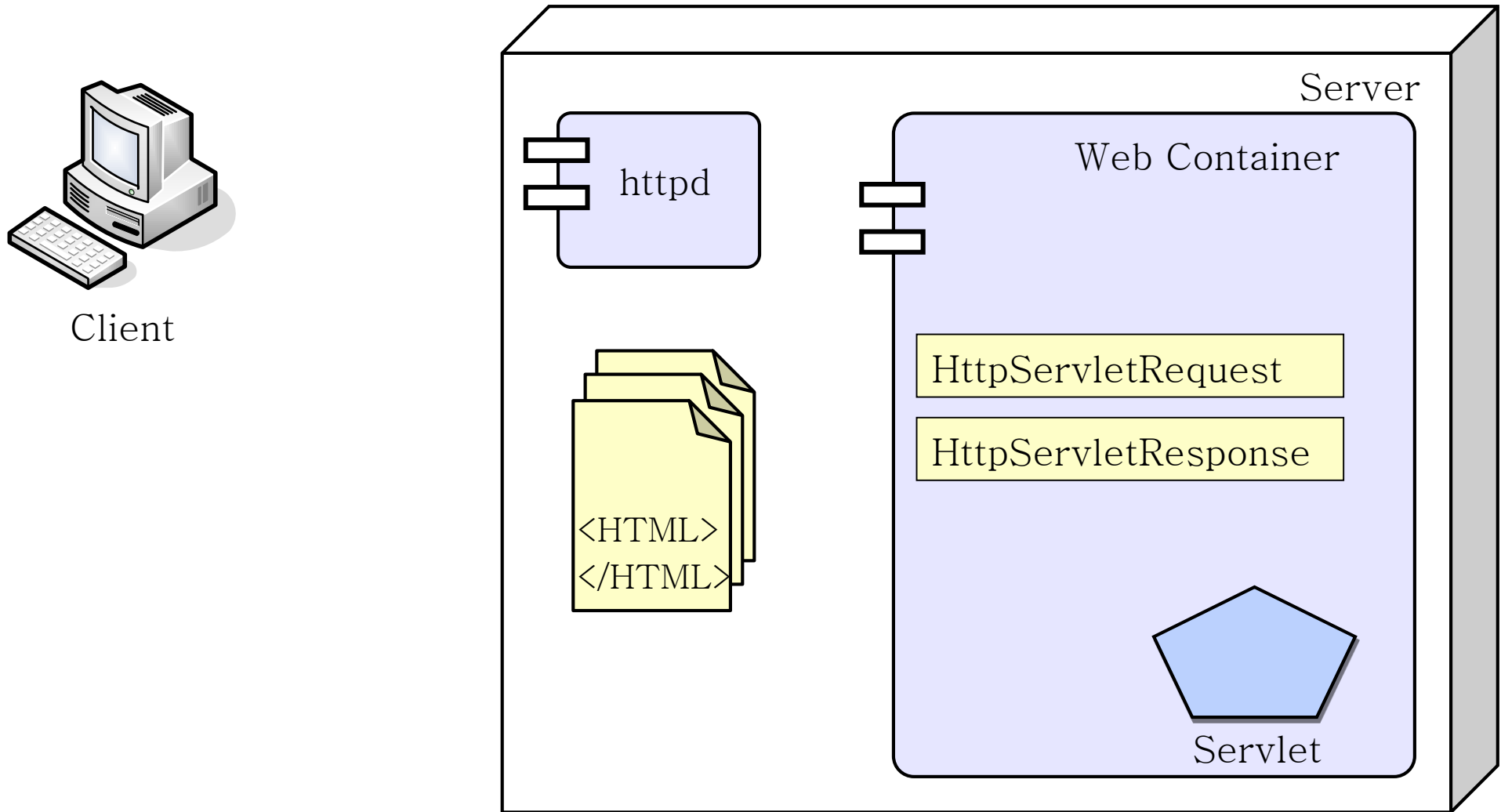
1. Client가 HTTP서비스 (Web Server) 에 HTTP Request 전달



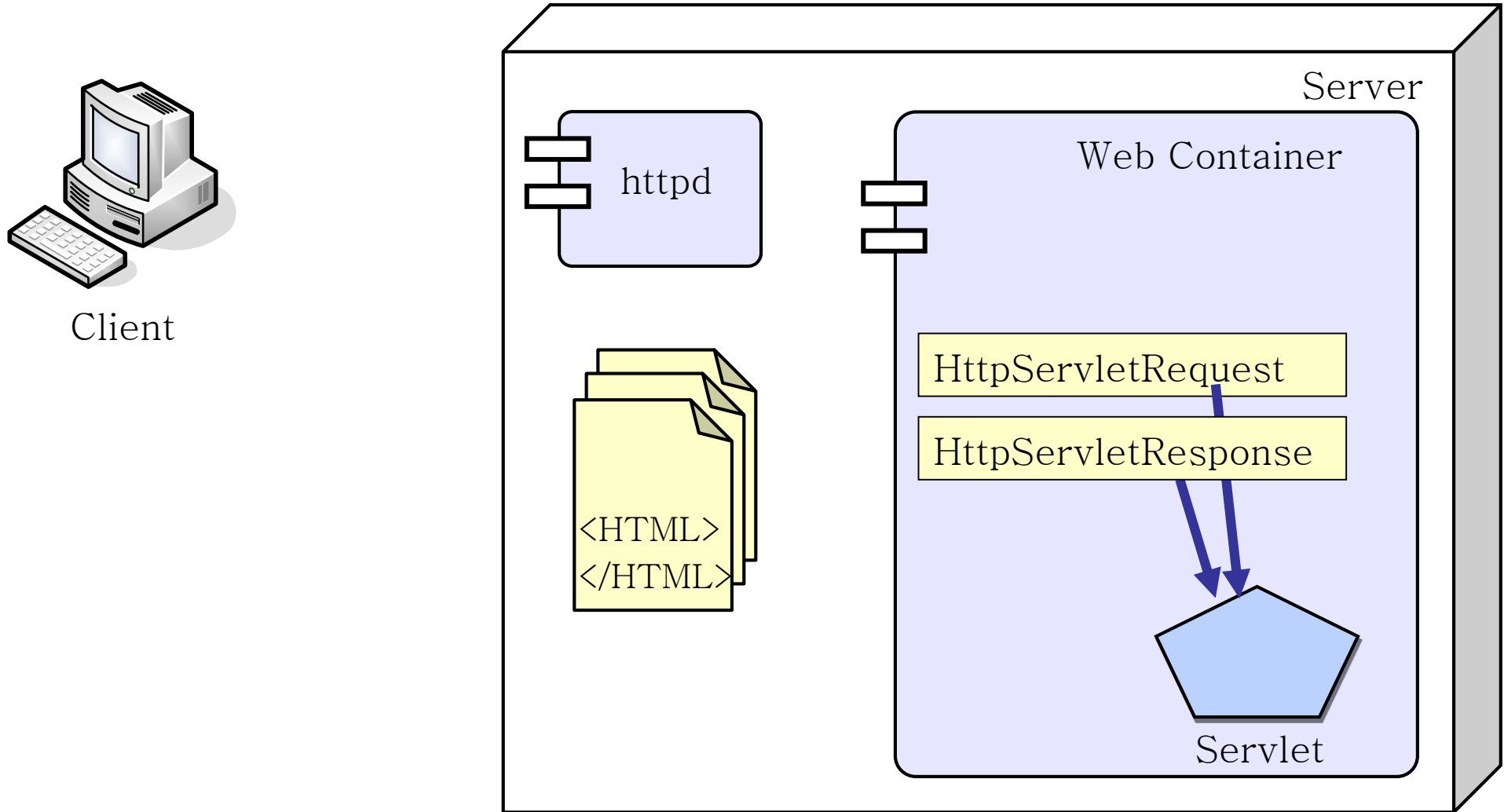
2. HTTP 서비스가 Web Container에 Request 전달



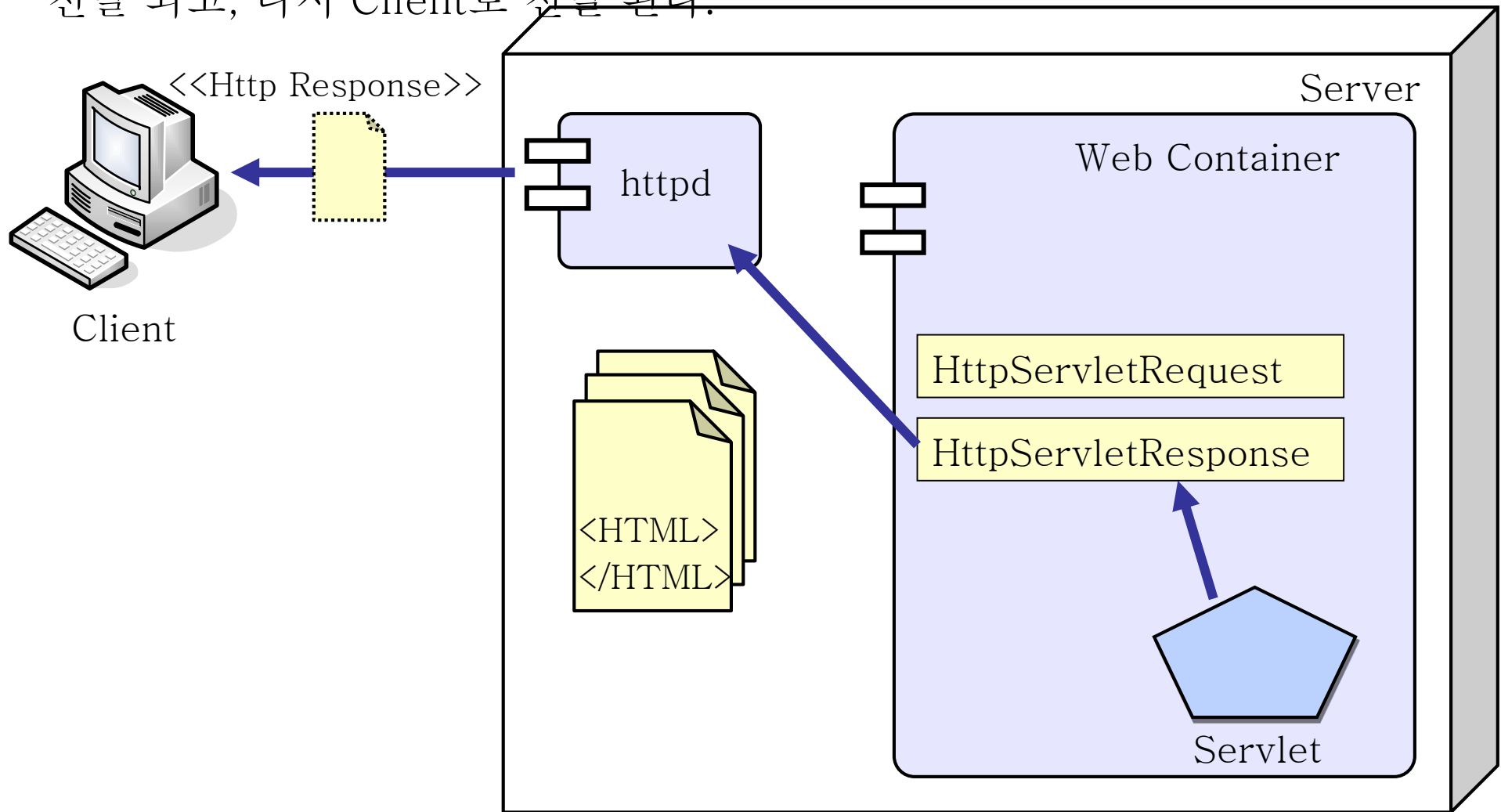
3. Web Container가 HttpServletRequest와 HttpServletResponse 객체 생성

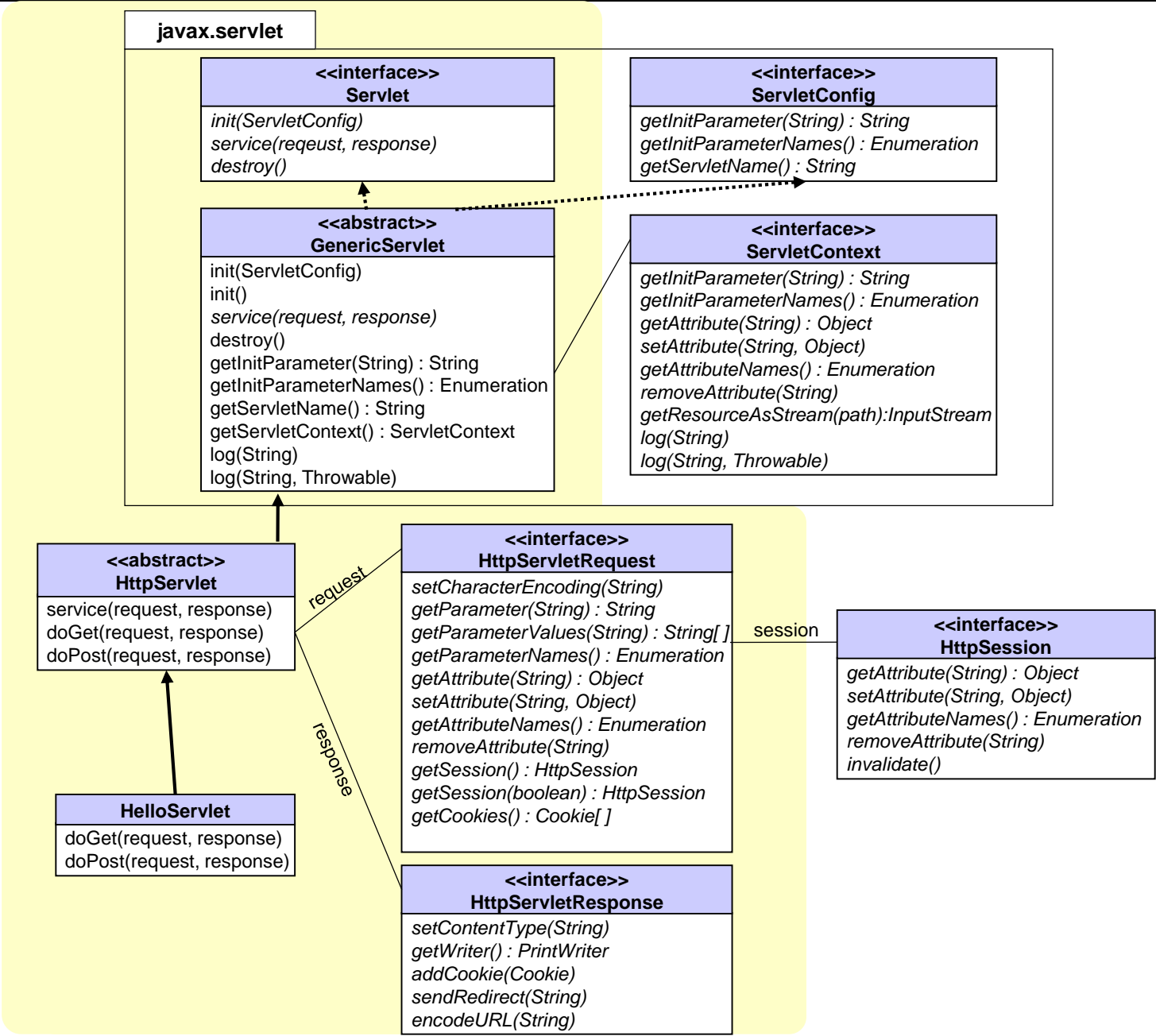


4. Web Container가 요청된 servlet의 service메소드 실행. 이 때, request와 response 객체가 인자로 전달되고 service 메소드는 쓰레드로 실행 된다.



5. Servlet에서 생성한 응답 텍스트가 HTTP 응답 스트림에 실려, HTTP 서비스로 전달 되고, 다시 Client로 전달 된다.





Developing a Simple Servlet The Uses HTML Forms

Component	Tag	Description
Textfield	<INPUT TYPE='text' ...>	한 줄로 된 텍스트 입력
Submit button	<INPUT TYPE='submit'>	클릭시 폼 제출
Reset	<INPUT TYPE='reset'>	폼에 있는 필드를 지움
Checkbox	<INPUT TYPE='checkbox' ...>	하나 이상의 옵션 선택
Radio button	<INPUT TYPE='radio' ...>	하나의 옵션 선택
Password	<INPUT TYPE='password' ...>	한 줄로 된 텍스트 입력 단, 내용 확인 불가
Hidden field	<INPUT TYPE='hidden' ...>	화면에 보이지 않으나, 데이터를 갖고 있음.
Select	<SELECT ...> <OPTION ...> </SELECT>	목록 상자에서 하나 이상의 옵션 선택
Textarea	<TEXTAREA ...>... </TEXTAREA>	여러 라인 텍스트 입력


```
<form name="myFormName"
      method="post"
      action="example.jsp">
```

INPUT TAG 에 사용 가능한
입력 요소들

```
</form>
```

	<code><input type="text" size="20" maxlength="20"></code>	한줄 텍스트
	<code><input type="password" size="20" maxlength="20"></code>	암호입력
	<code><textarea cols="18" rows="3"></textarea></code>	여러줄 텍스트
	<code><select> <option>1학년</option> <option>2학년</option> </select></code>	목록상자
<input type="checkbox"/> 등산 <input type="checkbox"/> 낚시	<code><input type="checkbox"> 등산 <input type="checkbox"> 낚시</code>	선택(체크)
<input type="radio"/> 남자 <input type="radio"/> 여자	<code><input type="radio"> 남자 <input type="radio"> 여자</code>	선택(라디오)
	<code><input type="file" size="10"></code>	파일업로드
	<code><input type="hidden"></code>	숨김 상자
	<code><input type="button" value="버튼"></code>	버튼
	<code><input type="submit" value="전송버튼"></code>	전송버튼
	<code><input type="reset" value="리셋"></code>	리셋버튼
	<code><input type="image" border="0" src="write.gif"></code>	이미지버튼

□ 앞 화면에서 이름에 pyokim 암호에 12345를 누르고 확인(submit)을 클릭하면...

- `<form>` `</form>` tag 사이에 있는 `input` tag 내용이 `name=value` 쌍으로 URL에 조립이 되면서 서버로 전달이 된다.
- 이 때, `<form>`의 `action` 속성에 지정된 값이 url이 된다.
- url과 폼 태그 값은 ? 로 구분되어 진다.
- 폼 태그가 2개 이상 전송 될 때, 각각은 & 로 구분되어 진다.

http://127.0.0.1:8080/servlet/chap03.LoginServlet?name=pyokim&pass=12345

URL

폼 태그 값

□ FORM tag 이용

- Form tag안의 Component 들의 값이 Submit 될 때, name,value쌍으로 전송된다.
- GET,POST 요청 가능 (FORM tag의 METHOD 속성에 지정)
- METHOD 속성을 생략하면, default로 GET 요청이 된다.

□ 주소필드에 직접 입력

- `http://192.168.0.100:8080/servlet/chap03.FormTestServlet?name=kim&pass=abc`
- GET만 가능

□ <A HREF> 태그 이용

- `<a href =`
 `'http://192.168.0.100:8080/servlet/chap03.FormTestServlet?name=kim&pass=abc'`
 이동 ``
- GET만 가능

□ GET : 폼 데이터가 URL에 포함되어 전송

- 폼 데이터 양이 적은 경우 – 256byte 제한
- 요청에 대한 북마크(즐거찾기)를 하고자 하는 경우
- 검색 이외의 작업에 사용해서는 안된다.
- URL과 데이터 구분은 ? 로 되어 진다.
- 이름=값 쌍이 여러 개인 경우, 각각은 & 로 구분되어 진다.

□ POST : 폼 데이터가 요청 body에 포함되어 전송

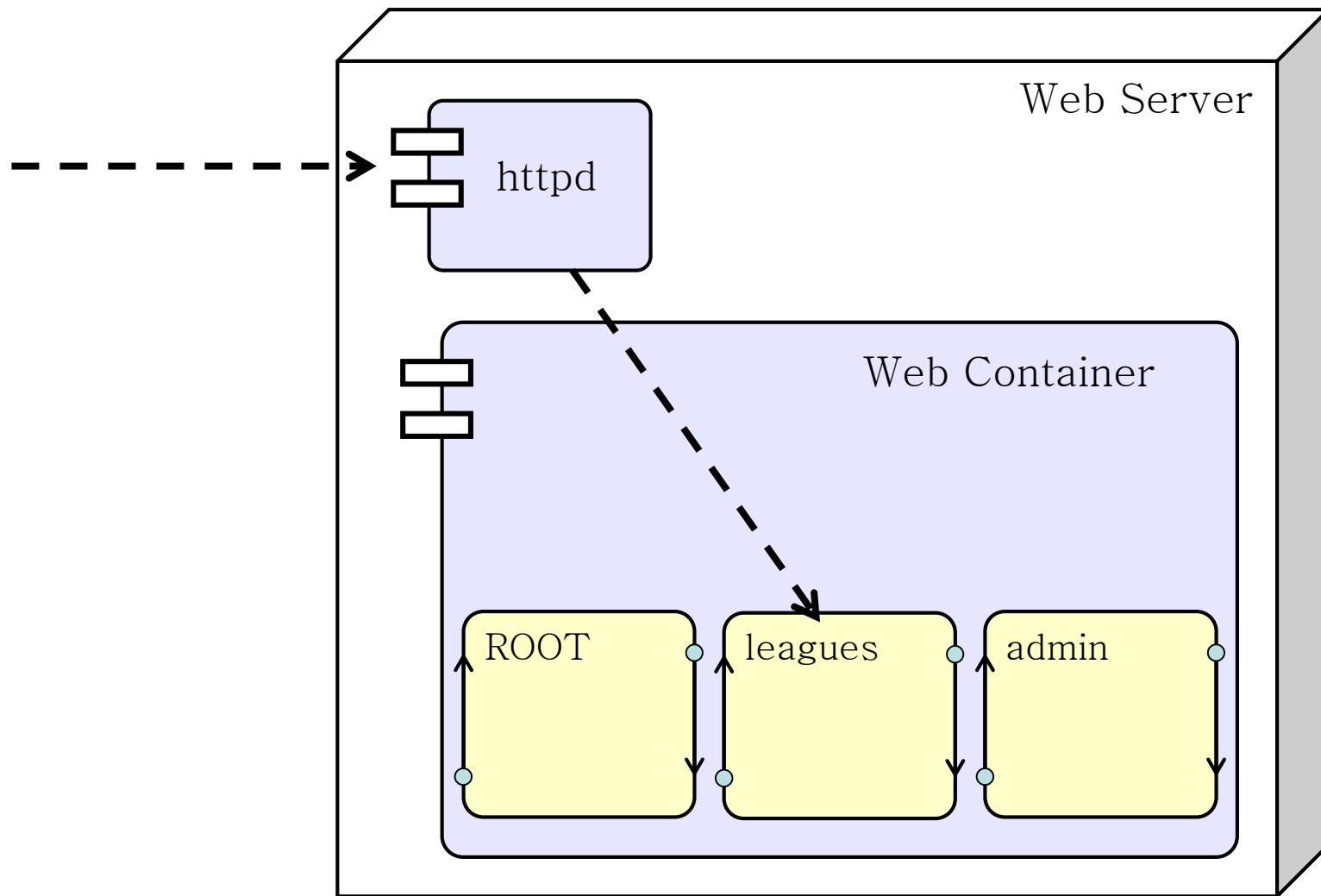
- 폼 데이터 양이 큰 경우 – 데이터 크기에 제한이 없다.
- 패스워드등 폼 데이터의 내용이 URL에 보여지지 않아야 하는 경우
- 이름=값 쌍이 여러 개인 경우, 각각은 & 로 구분되어 진다.

<<interface>> ServletRequest
getParameter(name:String):String getParameterValues(name:String):String[] getParameterNames(): Enumeration setCharacterEncoding(env:String);

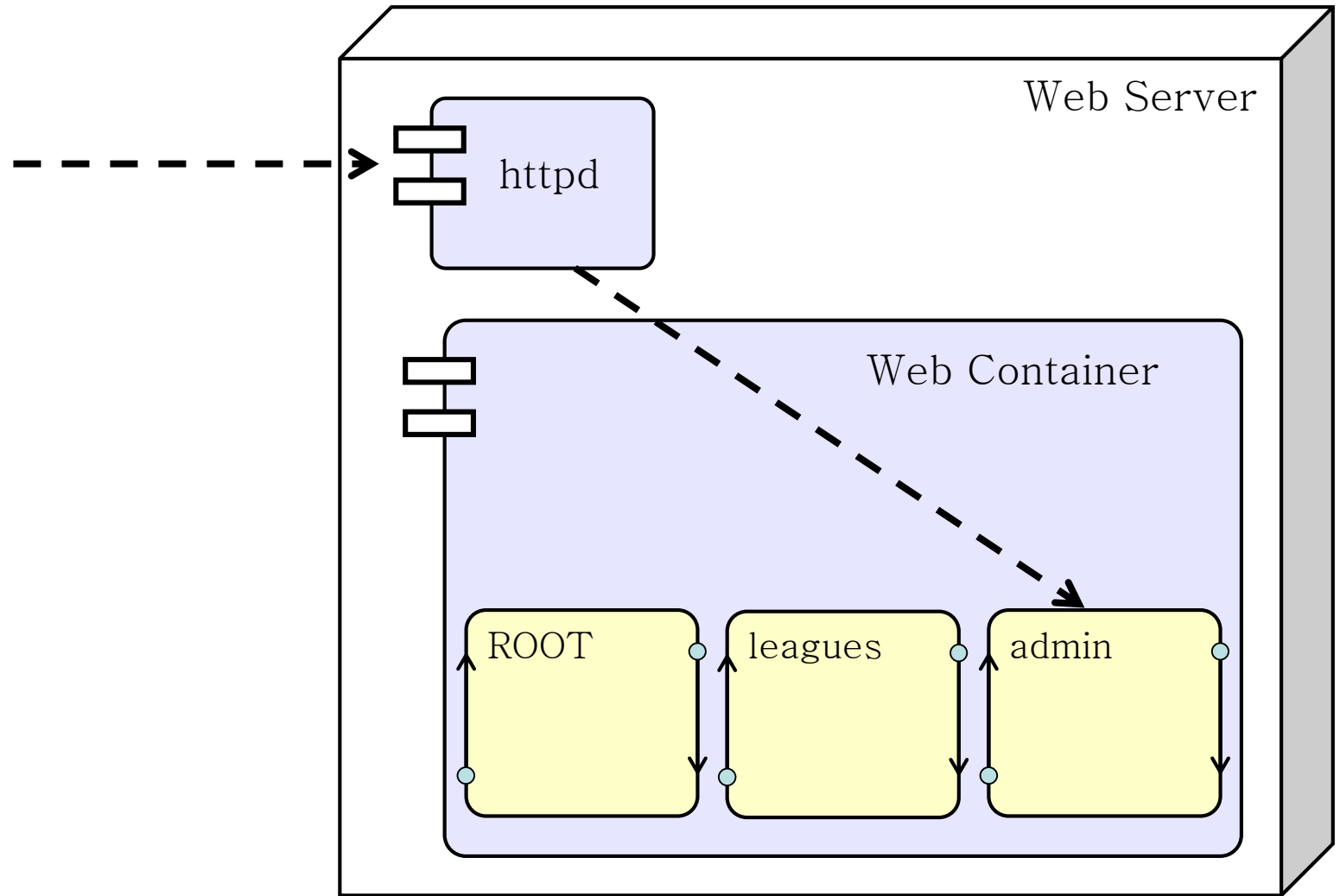
```
request.setCharacterEncoding("euc-kr");  
String name = req.getParameter("name");  
String pass = req.getParameter("pass");  
String[] hobby = req.getParameterValues("hobby");  
  
Enumeration e = request.getParameterNames();  
while (e.hasMoreElements()) {  
    String element = (String)e.nextElement();  
    System.out.println(element);  
    System.out.println(req.getParameter(element));  
}
```

Developing a Web Application Using a Deployment Descriptor

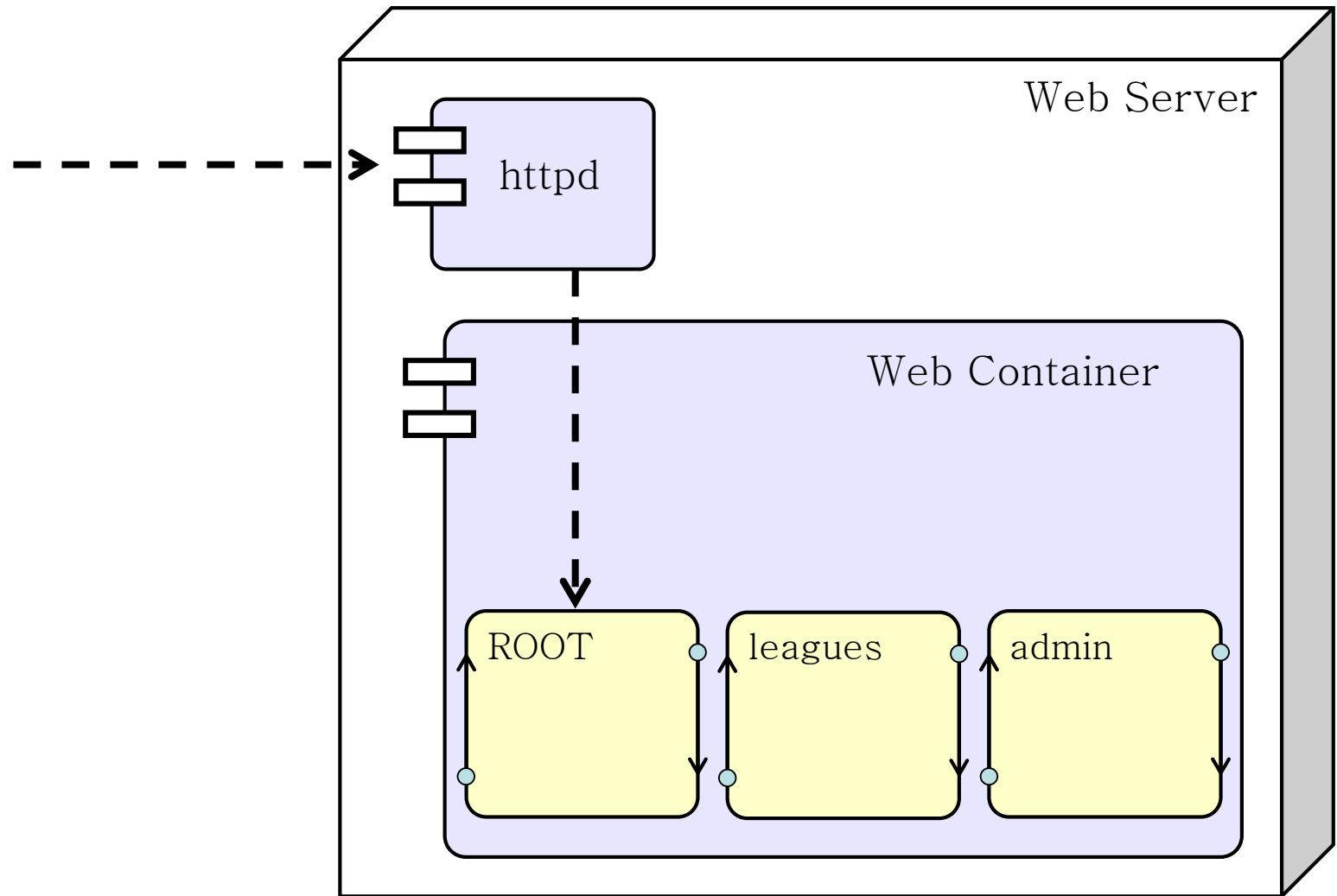
`http://www.soccer.org/leagues/registration.html`



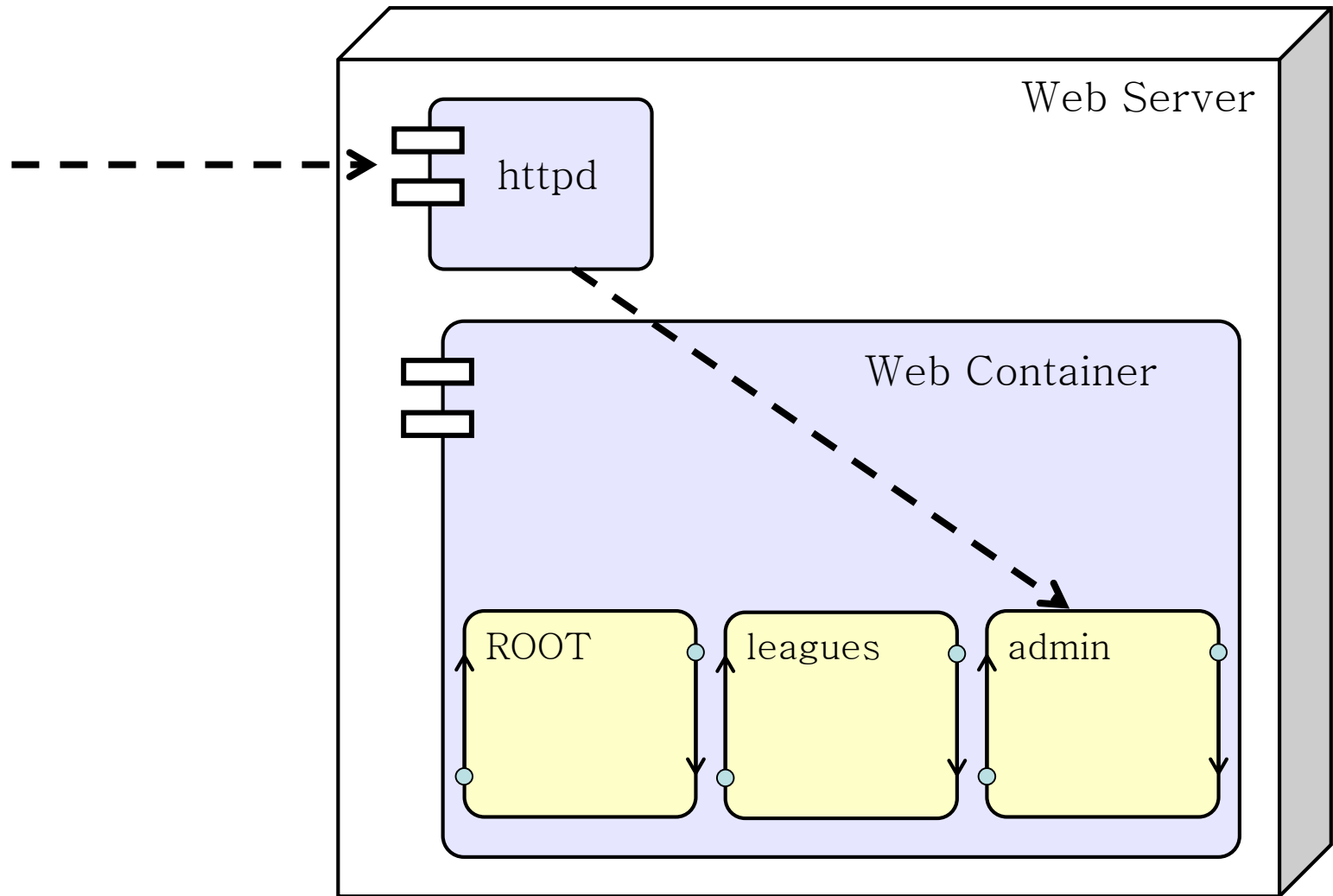
<http://www.soccer.org/admin/registration.html>



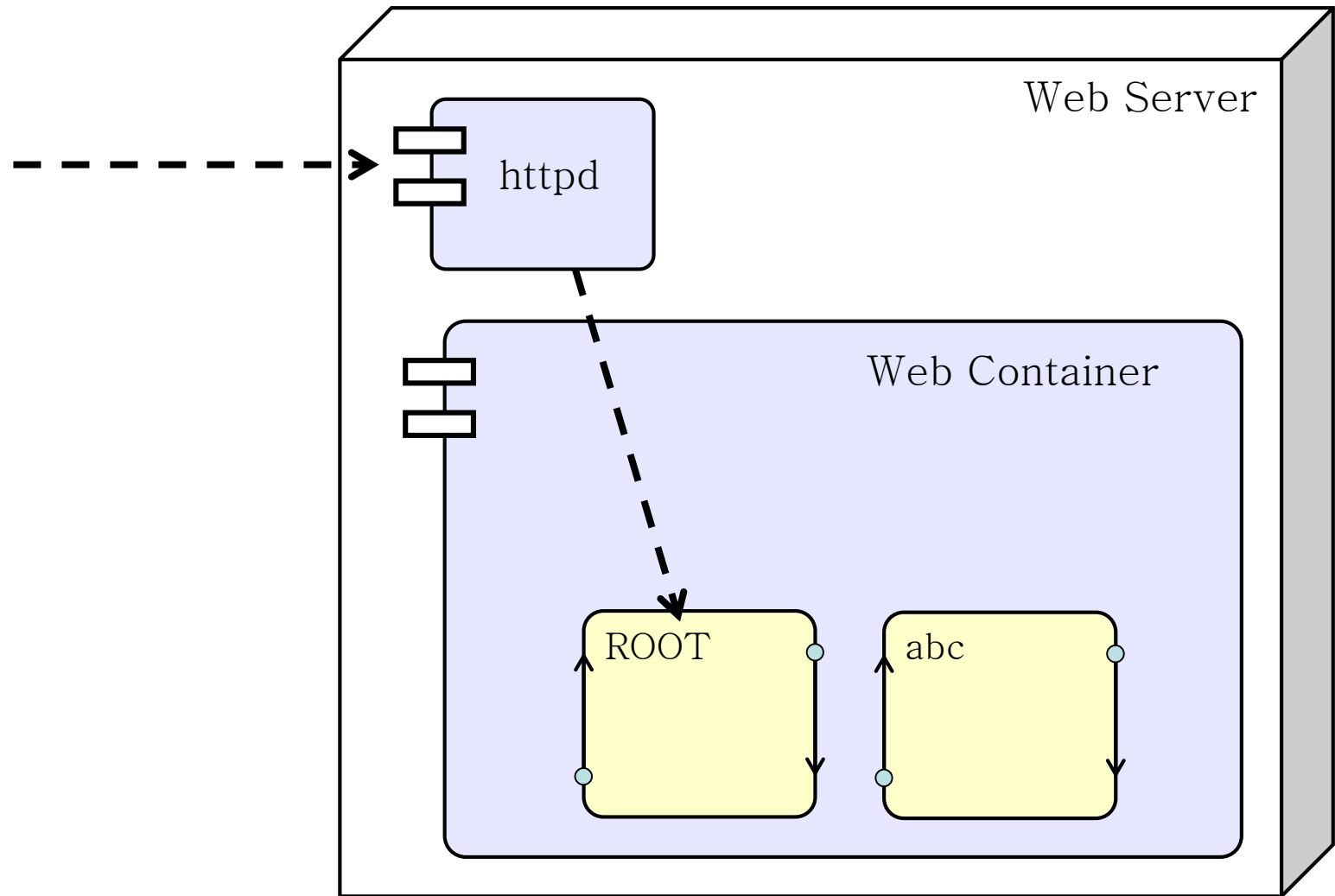
<http://www.soccer.org/registration.html>



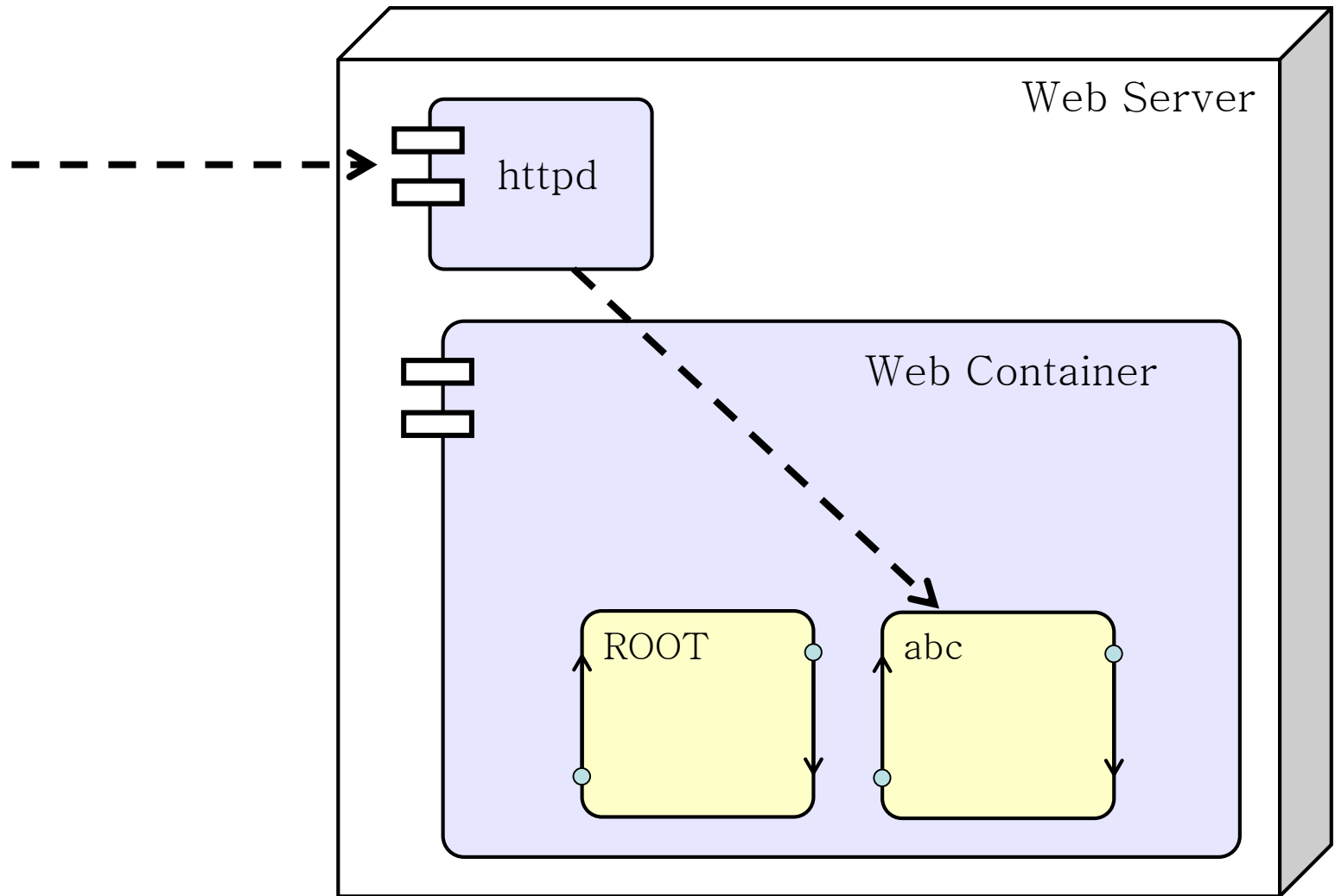
<http://www.soccer.org/admin/servlet/org.soccer.web.LeagueAdminServlet>



<http://127.0.0.1:8080/welcome.html>



`http://127.0.0.1:8080/abc/welcome.html`



❑ Simple Servlet 배치 전략의 문제점

- URL에 package 이름까지 포함된 Servlet Class 이름을 사용하여 access한다.
- package와 class 이름이 너무 길어서 page설계자의 작업이 많아질 수 있다.
- 디렉토리 구조와 클래스 이름을 그대로 노출시키기 때문에 보안에 취약해질 수 있다.

❑ 문제 해결

- Servlet에 대한 URL 매핑 지원
- DD (Deployment Descriptor)에서 설정 한다.

`http://127.0.0.1:8080/servlet/chap03.LoginServlet`



`http://127.0.0.1:8080/login`

□ Deployment Descriptor (배치 설명자) 란?

- 특정 Web Application에 대한 구성/배치 정보를 지정하는 XML 파일 (web.xml)
- 반드시 `WEB_ROOT_DIR\WEB-INF\`에 위치해야 한다.

```
<?xml version="1.0" encoding="euc-kr"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">
```

```
</web-app>
```

□ Servlet Mapping

항상 <servlet> tag와 <servlet-mapping> tag가 쌍으로 있어야 한다.

```
<?xml version="1.0" encoding="euc-kr"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">

  <servlet>
    <servlet-name>Login</servlet-name>
    <servlet-class>chap03.LoginServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>Login</servlet-name>
    <url-pattern>/login</url-pattern>
  </servlet-mapping>

</web-app>
```

일치

/ 로 시작해야 한다.

- ❑ Servlet Mapping을 하면, 이를 url 부분을 url-pattern tag에 선언한 것으로 대체 한다.

```
/chap03/login.html
```

```
<html>
<head>
  <title>로그인 화면</title>
</head>
<body>  로그인<br>
<form action=' /login' method='post' name='formtest'>
  이름 : <input type=text name=name value=''> <br>
  암호 : <input type=password name=pass ><br><br>
  <input type=submit value="확인">
  <input type=reset value="초기화">
</form>
</body>
</html>
```


- ❑ Servlet Mapping 을 여러 개 선언 할 경우, <servlet> tag와 <servlet-mapping> tag의 선언 위치에 주의 하자.

<servlet>

```
<servlet>
  <servlet-name>Login</servlet-name>
  <servlet-class>chap03.LoginServlet</servlet-class>
</servlet>

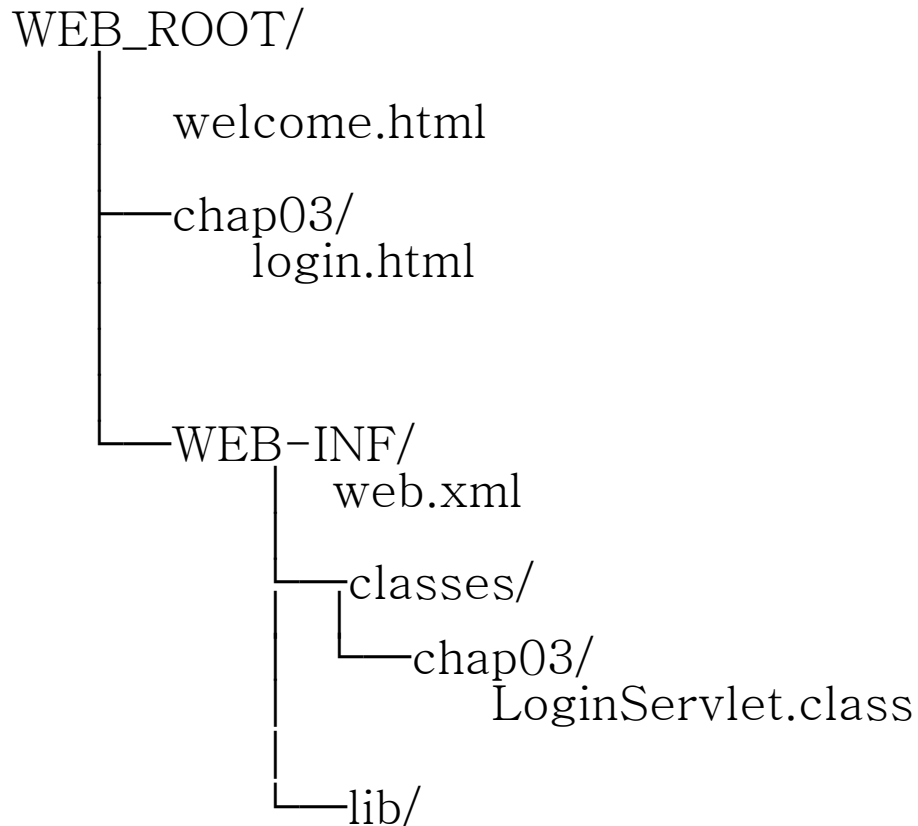
<servlet>
  <servlet-name>Welcome</servlet-name>
  <servlet-class>chap02.WelcomeServlet</servlet-class>
</servlet>
```

<servlet-mapping>

```
<servlet-mapping>
  <servlet-name>Login</servlet-name>
  <url-pattern>/login</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>Welcome</servlet-name>
  <url-pattern>/welcome</url-pattern>
</servlet-mapping>
```

- WEB_ROOT : C:\practice\workspace\SE4_WEB_LAB\web

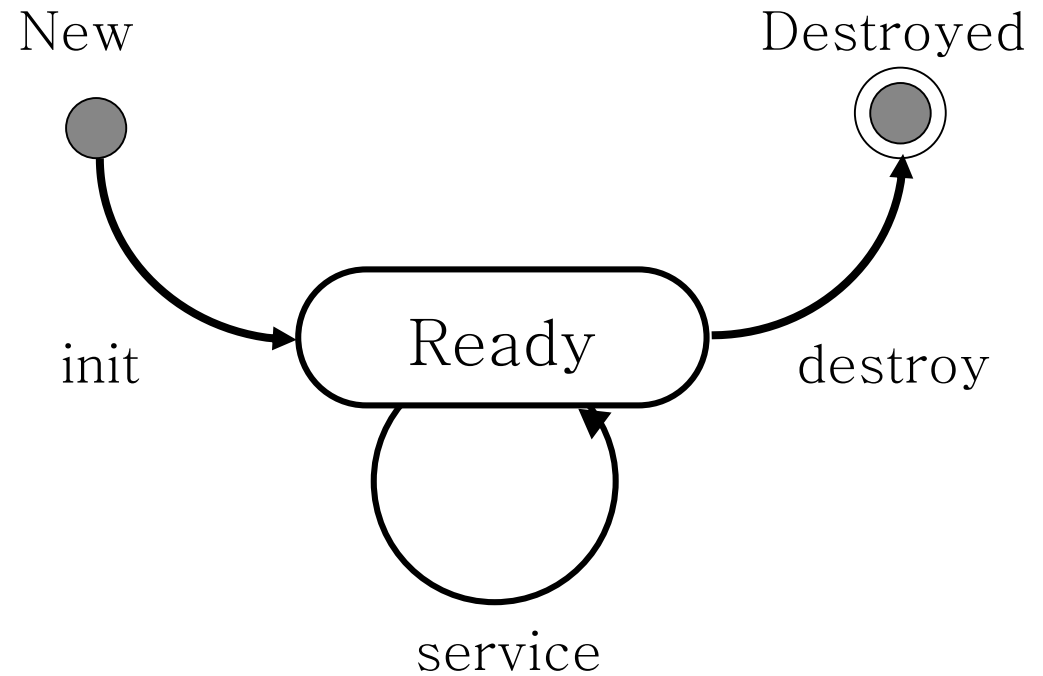
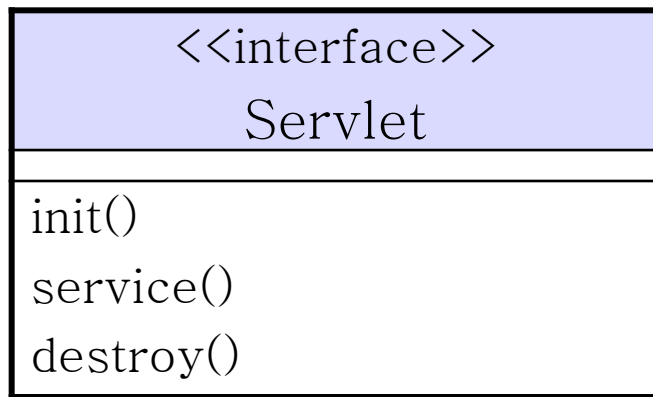


- HTML, JSP
→ *WEB_ROOT /*
- servlet 및 그 외 관련 클래스들
→ *WEB_ROOT / WEB-INF/classes/*
- JAR
→ *WEB_ROOT / WEB-INF/lib*
- DD(web.xml)
→ *WEB_ROOT / WEB-INF/*

Configuring Servlets

1. Servlet Life Cycle

2. ServletConfig API



❑ init()

- Servlet 인스턴스가 처음 생성될 때, 호출 된다.
- 이 메소드를 Overriding 하여 Servlet에 필요한 초기화 작업을 수행할 수 있다.
- 주의 : 여러 인스턴스에서 공유하는 자원은 웹 어플리케이션 수준에서 구성해야 한다.
즉, init 메소드에서 초기화하는 자원은 하나의 인스턴스에서만 공유 된다.

❑ service()

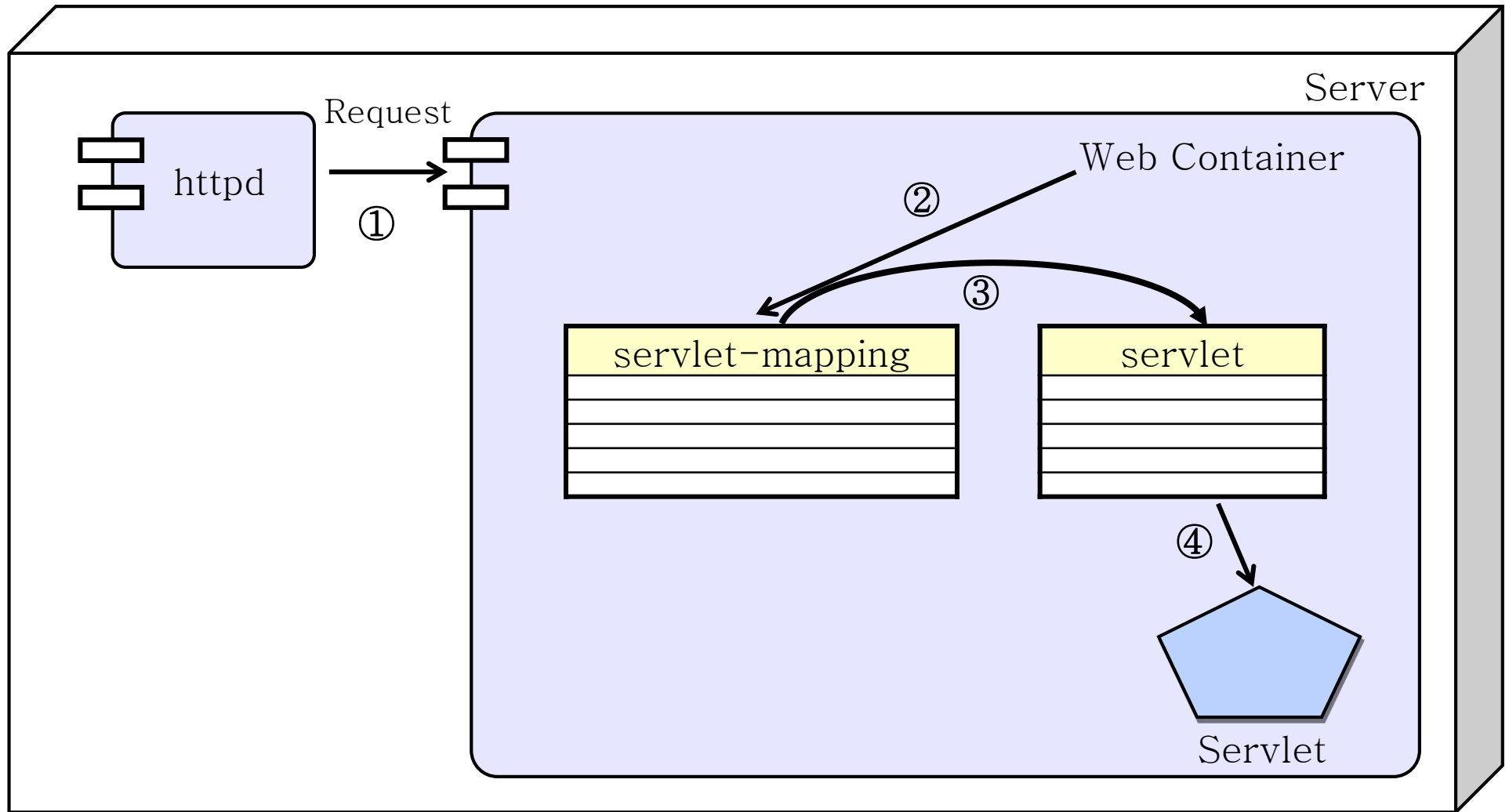
- 사용자 요청을 처리 하는 메소드
- 대부분 직접 구현하지 않고, service메소드에 의해 호출 되는 doGet, doPost 메소드를 Overriding한다.

❑ destroy()

- Servlet 인스턴스가 제거될 때, 호출 된다.

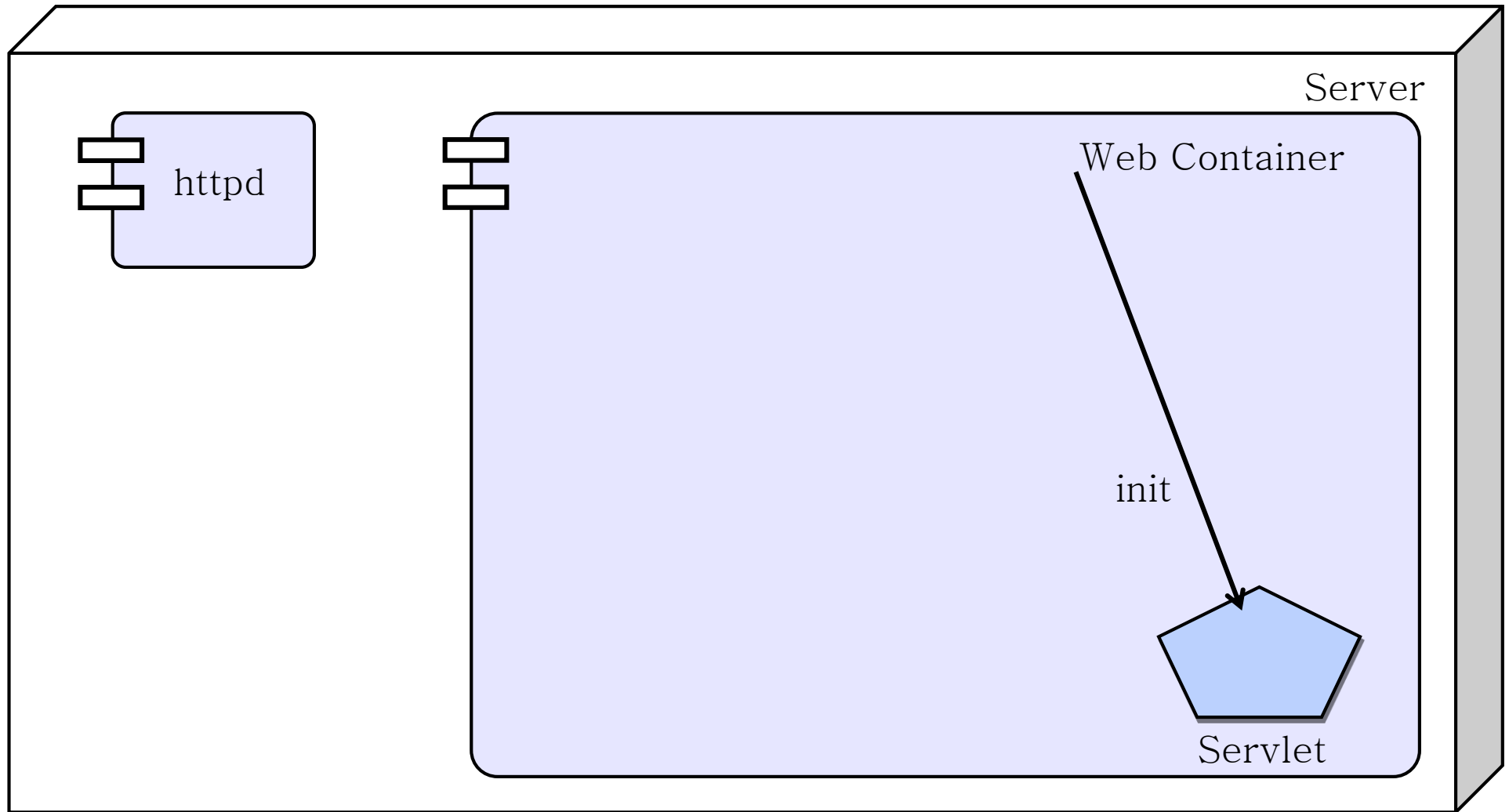
Servlet Life Cycle

- Request에 대해 Web Container가 해당 되는 Servlet을 생성한다.



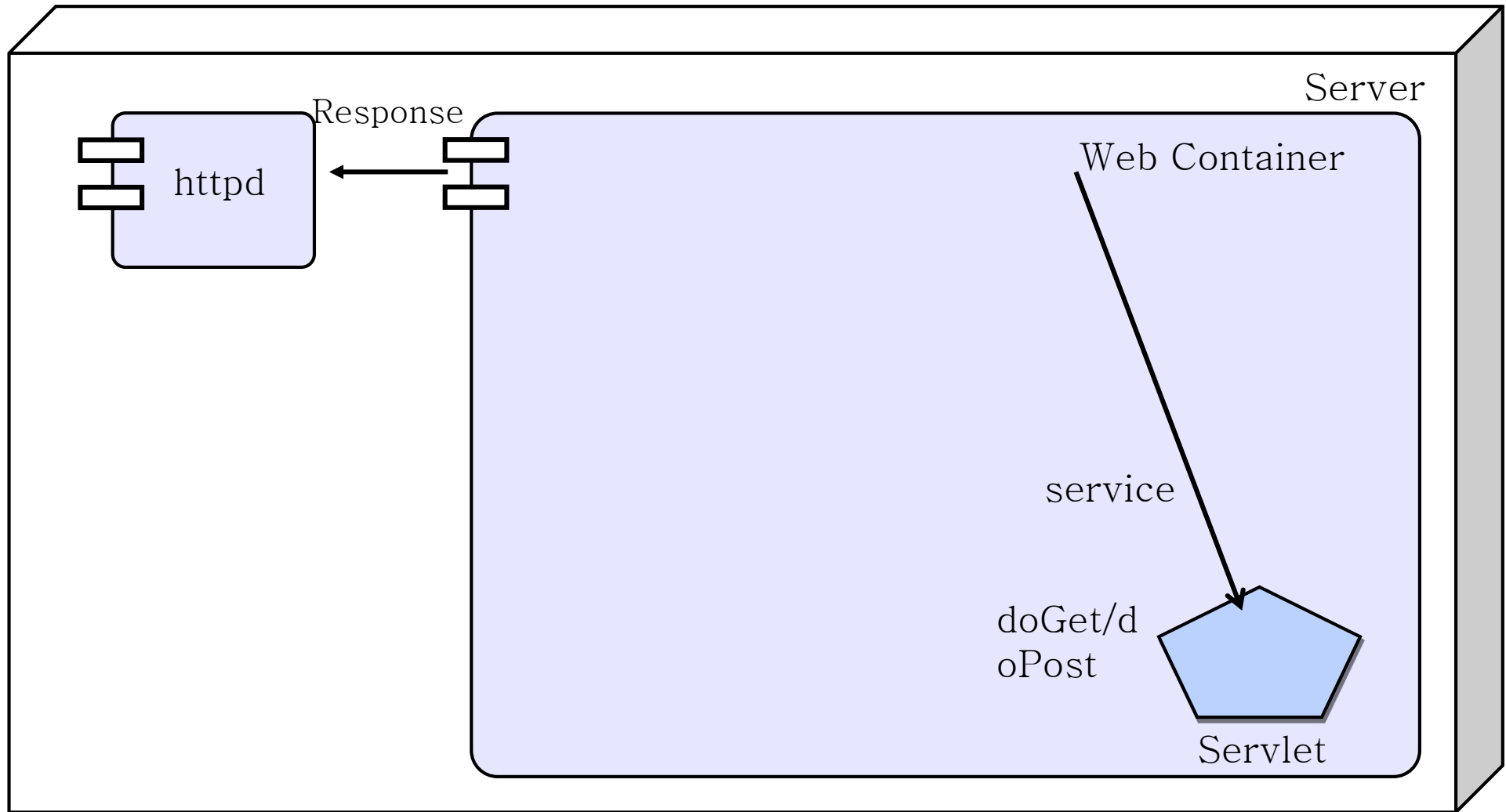
Servlet Life Cycle

- Servlet 인스턴스가 생성 되면, Web Container가 init 메소드를 호출 한다.



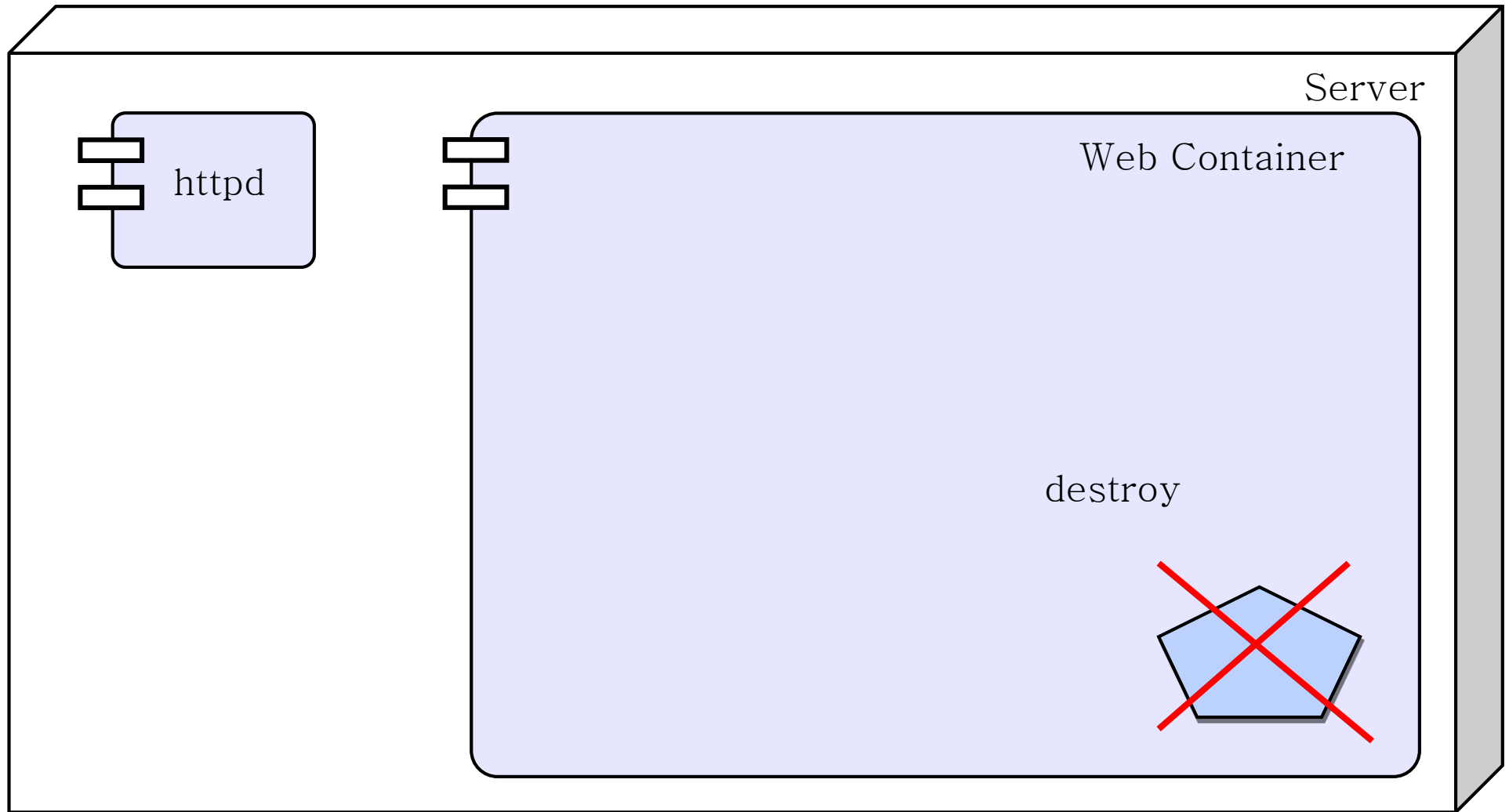
Servlet Life Cycle

- 요청을 처리하기 위해 Web Container는 Servlet의 service 메소드를 호출 하고, service메소드는 doGet 또는 doPost를 호출 한다.



Servlet Life Cycle

- Servlet 인스턴스가 제거될 때, Web Container에 의해 destroy 메소드가 호출 된다.



```
package chap05;

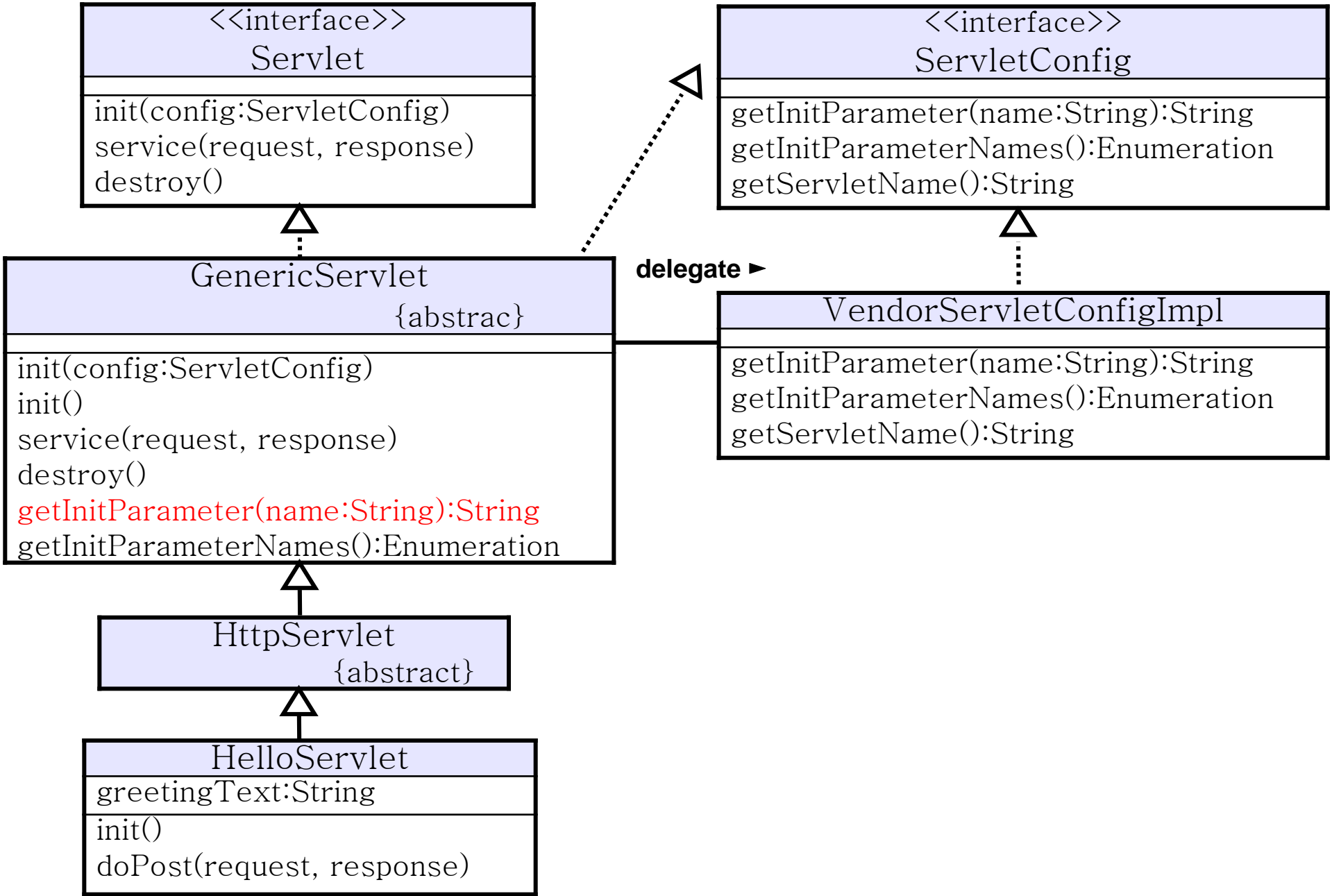
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;

public class LifeCycle extends HttpServlet {

    public void init() {
        System.out.println( "Init Method Called..." );
    }

    public void doGet( HttpServletRequest req,
                      HttpServletResponse res )
                      throws ServletException, IOException {
        System.out.println( "doGet Method Called..." );
        res.setContentType( "text/html;charset=euc-kr" );
        PrintWriter out = res.getWriter();
        out.println( "안녕" );
        out.close();
    }

    public void destroy() {
        System.out.println( "Destroy Method Called..." );
    }
}
```



- ❑ Servlet이 초기화 될 때, 공통적으로 수행되는 작업은 메모리에 모든 외부 자원을 로드하는 것이다.
- ❑ 일반적으로 이런 자원의 디렉토리와 파일 경로 정보는 Servlet 초기화 파라미터로 저장 된다.

```
<servlet>
  <servlet-name>Hello</servlet-name>
  <servlet-class>chap05.HelloServlet</servlet-class>
  <init-param>
    <param-name>greetingText</param-name>
    <param-value>Hello</param-value>
  </init-param>
</servlet>
```

```
public class HelloServlet extends HttpServlet {
    private String greetingText;

    public void init() {
        greetingText = getInitParameter( "greetingText" );
        System.out.println( ">> greetingText = " + greetingText );
    }

    public void doGet( HttpServletRequest request,
                      HttpServletResponse response )
                      throws IOException {
        request.setCharacterEncoding( "euc-kr" );
        response.setContentType( "text/html;charset=euc-kr" );

        PrintWriter out = response.getWriter();

        out.println( "<HTML><HEAD>" );
        out.println( "<TITLE>Hello Servlet</TITLE></HEAD>" );
        out.println( "<BODY BGCOLOR='white'>" );
        out.println( "<B>" + greetingText + ", "
                    + request.getParameter( "name" ) + "</B>" );
        out.println( "</BODY>" );
        out.println( "</HTML>" );
        out.close();
    }
}
```

Sharing Resources Using the Servlet Context

❑ Web Application?

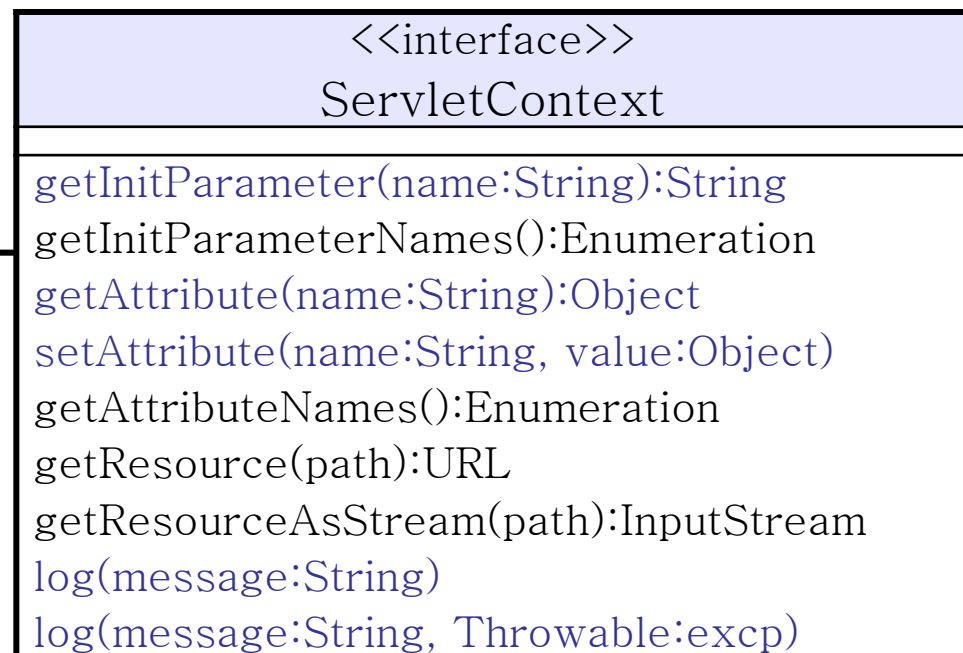
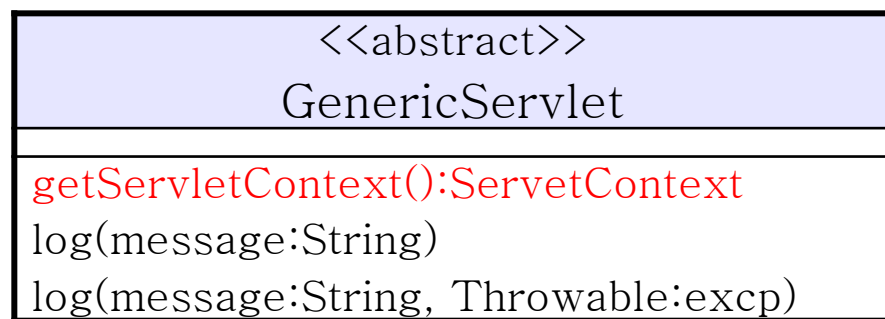
- HTML 페이지, 미디어 파일, 데이터, 자원 파일, Servlet/JSP, 기타 보조 클래스 및 객체와 같이 필요한 정적/동적 자원을 모두 포함하는 Collection
- Deployment Descriptor 는 Web Application에서 사용하는 구조와 서비스를 지정하는데 쓰인다.

❑ ServletContext object

- Web Application의 Runtime 표현
- Web Application == Context

□ ServletContext API가 제공하는 주요 기능

- Application범위 초기화 파라미터에 대한 읽기 전용 Access
- Application수준 파일 자원에 대한 읽기 전용 Access
- Application범위 속성에 대한 읽기/쓰기 Access
- 로깅 기능



- Context내에서 사용될 초기화 매개 변수 선언 (web.xml)

```
<context-param>
```

```
  <param-name>adminPath</param-name>
```

```
  <param-value>C:\practice\workspace\SE4_WEB_LAB\web\admin.txt</param-value>
```

```
</context-param>
```

```
<context-param>
```

```
  <param-name>myName</param-name>
```

```
  <param-value>홍길동</param-value>
```

```
</context-param>
```

```
public class ContextParamServlet extends HttpServlet {
    public void doGet( HttpServletRequest req, HttpServletResponse res )
        throws IOException {
        res.setContentType( "text/html;charset=euc-kr" );

        ServletContext context = getServletContext();
        String adminPath = context.getInitParameter( "adminPath" );

        BufferedReader br = new BufferedReader( new FileReader(adminPath) );
        String admin = br.readLine();

        PrintWriter out = res.getWriter();
        out.println( "<HTML>" );
        out.println( "<BODY BGCOLOR='white'>" );
        out.println( "<B> ContextParam 페이지</B><br>" );

        if ( admin == null || admin.equals("") ) {
            context.log("어드민이 지정되지 않았습니다.");
            out.println( "<B>로그를 확인하세요.</B><br>" );
        } else {
            out.println( "<B> " +admin+ "님에게 문의하세요.</B><br>" );
        }
        out.println( "</BODY>" );
        out.println( "</HTML>" );
        out.close();
    }
}
```

```
ServletContext context = sce.getServletContext();
String catalogFileName =
    context.getInitParameter( "catalogFileName" );
InputStream is = null;
BufferedReader catReader = null;
try {
    is = context.getResourceAsStream( catalogFileName );
    catReader = new BufferedReader( new InputStreamReader( is ) );
}
```

```
public class ContextParamServlet extends HttpServlet {
    public void doGet( HttpServletRequest req, HttpServletResponse res )
        throws IOException {
        res.setContentType( "text/html;charset=euc-kr" );

        ServletContext context = getServletContext();
        String adminPath = context.getInitParameter( "adminPath" );

        BufferedReader br = new BufferedReader( new FileReader(adminPath) );
        String admin = br.readLine();

        PrintWriter out = res.getWriter();
        out.println( "<HTML>" );
        out.println( "<BODY BGCOLOR='white'>" );
        out.println( "<B> ContextParam 페이지</B><br>" );

        if ( admin == null || admin.equals("") ) {
            context.log("어드민이 지정되지 않았습니다.");
            out.println( "<B>로그를 확인하세요.</B><br>" );
        } else {
            out.println( "<B> " +admin+ "님에게 문의하세요.</B><br>" );
        }
        out.println( "</BODY>" );
        out.println( "</HTML>" );
        out.close();
    }
}
```

- ❑ Log는 날짜별로 별도의 텍스트 파일로 기록된다.
- ❑ c:\Wpractice\workspace\SE4_WEB_LAB\web\admin.txt 내용을 지우고 다시 servlet요청
- ❑ C:\TOMCAT_HOME\logs\localhost_log.날짜.txt 에서 확인

TOMCAT_HOME\logs\localhost_log.2006-03-15.txt

2006-03-15 00:41:22 StandardContext[]어드민이 지정되지 않았습니다.

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;

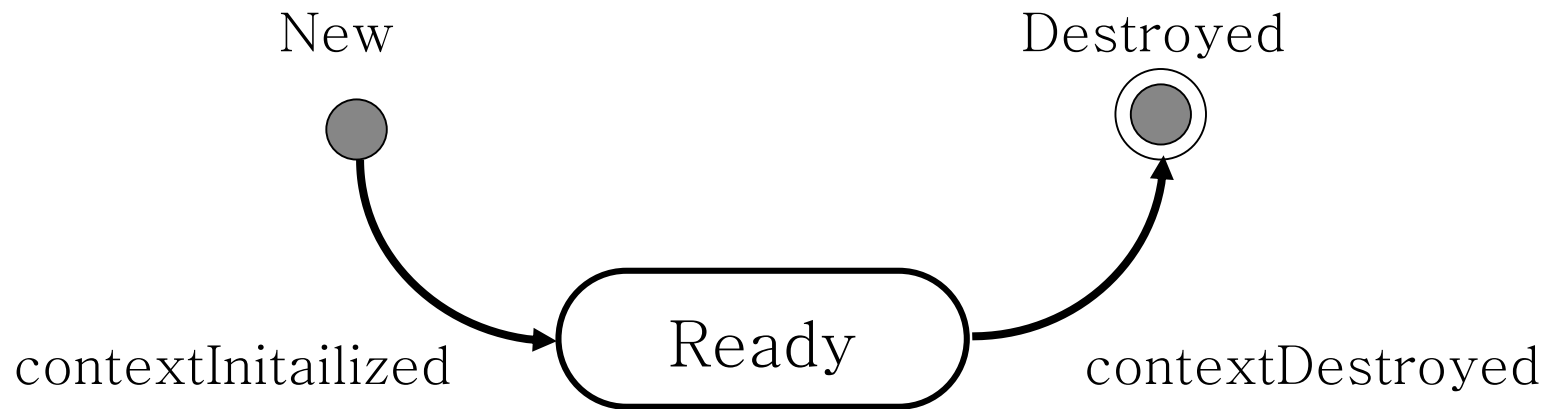
public class ProductServlet extends HttpServlet {
    public void doGet( HttpServletRequest req, HttpServletResponse res )
        throws IOException {

        ServletContext context = getServletContext();
        context.setAttribute( "product1", "냉장고" );
        context.setAttribute( "product2", "TV" );
        context.setAttribute( "product3", "세탁기" );

        res.setContentType( "text/html;charset=euc-kr" );
        PrintWriter out = res.getWriter();
        out.println( "<HTML>" );
        out.println( "<BODY BGCOLOR='white'>" );
        out.println( "<B> product1 : 냉장고</B><br>" );
        out.println( "<B> product2 : TV</B><br>" );
        out.println( "<B> product3 : 세탁기</B><br>" );
        out.println( "를 context 영역에 올려 놓았습니다.^^" );
        out.println( "</BODY>" );
        out.println( "</HTML>" );
        out.close();
    }
}
```

```
public class OrderServlet extends HttpServlet {
    public void doGet( HttpServletRequest req, HttpServletResponse res )
        throws IOException {
        ServletContext context = getServletContext();
        res.setContentType( "text/html;charset=euc-kr" );
        PrintWriter out = res.getWriter();
        out.println( "<HTML>" );
        out.println( "<BODY BGCOLOR='white'>" );
        out.println( "<B> " + (String)context.getAttribute( "myName" )
            + "님</B><br>" );
        out.println( "<B> product1 : "
            + (String)context.getAttribute( "product1" ) + "</B><br>" );
        out.println( "<B> product2 : "
            + (String)context.getAttribute( "product2" ) + "</B><br>" );
        out.println( "<B> product3 : "
            + (String)context.getAttribute( "product3" ) + "</B><br>" );
        out.println( " 위의 제품을 주문하실수 있습니다!!!!" );
        out.println( "</BODY>" );
        out.println( "</HTML>" );
        log( "-----" );
        log( "로그도 찍었당..." );
        log( "-----" );
    }
}
```


- ❑ Servlet에 수명 주기가 있듯이 Web Application에도 수명 주기가 있다.
- ❑ 각각의 Web Application은 Web Container가 시작될 때 초기화되며, Web Container가 종료될 때 제거 된다.
- ❑ Web Application이 시작될 때와 종료될 때에는 event가 발생하며, event에 트리거 되는 것이 ServletContextListener이다.

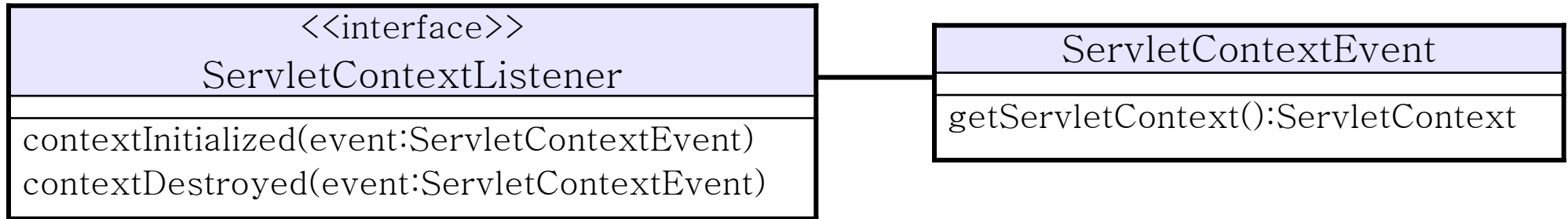


- DD(web.xml)에 listener클래스를 지정한다.

```
<listener>  
  <listener-class>chap06.ProductListListener</listener-class>  
</listener>
```

- ServletContextListener 인터페이스를 implements하는 클래스를 작성한다.
 - contextInitialized() : Web Application 시작시 호출되며,
주로 JDBC Connection Pool이나 Biz.객체(카달로그)
등과 같이 한 번 생성하면 Web Application 전체에 걸쳐
사용하는 공유 자원을 초기화 하는데 쓰인다.
 - contextDestroyed() : Web Application 종료시 호출
Connection 제거, 변경된 공유 자원 Write 등의 작업

ServletContextListener API



- Servlet과는 달리 `ServletContextListener` 에서 `ServletContext`를 이용하기 위해서는 `contextInitialized()` 나 `contextDestroyed()` 의 parameter인 `ServletContextEvent`를 통해 얻어와야 한다.

```
package chap06;
import javax.servlet.*;

public class ProductListListener
    implements ServletContextListener {

    public void contextInitialized( ServletContextEvent sce ) {

        ServletContext context = sce.getServletContext();

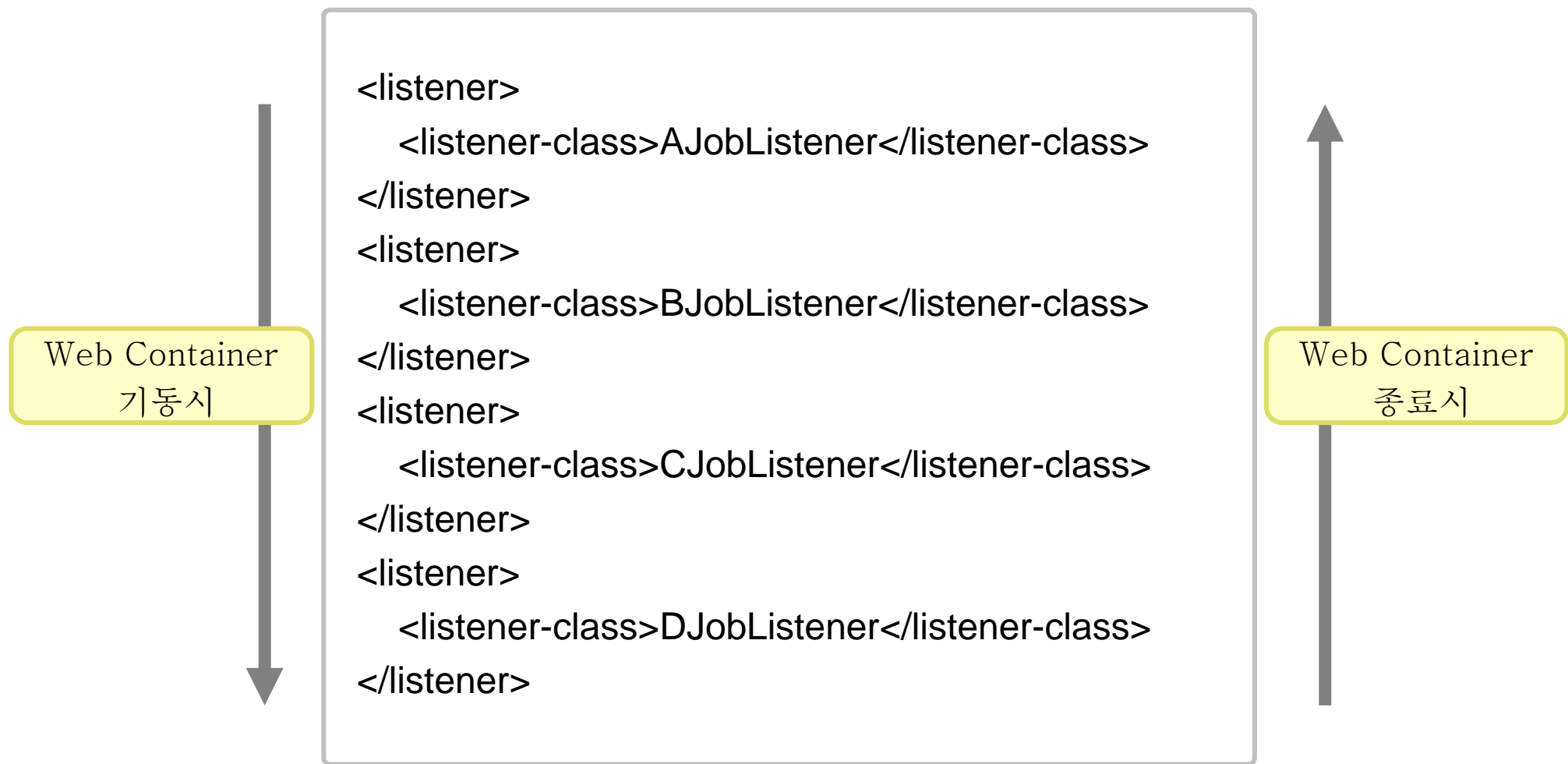
        String str = context.getInitParameter( "myName" );
        context.setAttribute( "myName", str );

        context.setAttribute( "product1", "냉장고" );
        context.setAttribute( "product2", "세탁기" );
        context.setAttribute( "product3", "전자렌지" );

        System.out.println( "contextInit 메소드가 수행되었음" );
    }

    public void contextDestroyed( ServletContextEvent sce ) {
    }
}
```

- DD(web.xml)에 listener클래스를 여러 개 지정할 수 있다.



Developing JSP Pages

- ❑ HTML 페이지에 Java 코드를 추가할 수 있게 해주는 기술
- ❑ Presentation과 Business logic 분리가 목적
- ❑ 페이지 요청시, Web Container에 의해 서블릿으로 변환 되어 서블릿이 수행 된다.
- ❑ 장점
 - Web Designer가 Java를 몰라도, 페이지 디자인이 가능하게 한다.
 - Java 프로그래머가, Web Design을 건드리지 않고 코드를 작성할 수 있다.

```

<%! private static final String DEFAULT_NAME = "World"; %>
<HTML>
<HEAD>
<TITLE>Hello JavaServer Page</TITLE>
</HEAD>
<%-- Determine the specified name (or user default) --%>
<%
    String = name request.getParameter( "name" );
    if ( (name==null) || (name.length()) == 0 ) {
        name = DEFAULT_NAME;
    }
%>
<BODY BGCOLOR='white'>
<B>Hello, <%=name%></B>
</BODY>
</HTML>

```


□ 아래의 코드를 작성하고,

c:\Wpractice\workspace\SE4_WEB_LAB\web\chap08\hello.jsp로 저장 한다.

hello.jsp

```
<%@ page contentType="text/html; charset=euc-kr" %>
<HTML>
<HEAD>
<TITLE>Hello JavaServer Page</TITLE>
</HEAD>
<BODY BGCOLOR='white'>
<%
    request.setCharacterEncoding( "euc-kr" );
    out.print( "Hello, " );
%>
<%= request.getParameter( "name" ) %>
</BODY>
</HTML>
```

- Step 1: 웹 브라우저에서 다음의 url을 호출한다.

```
http://127.0.0.1:8080/chap08/hello.jsp?name=kim
```

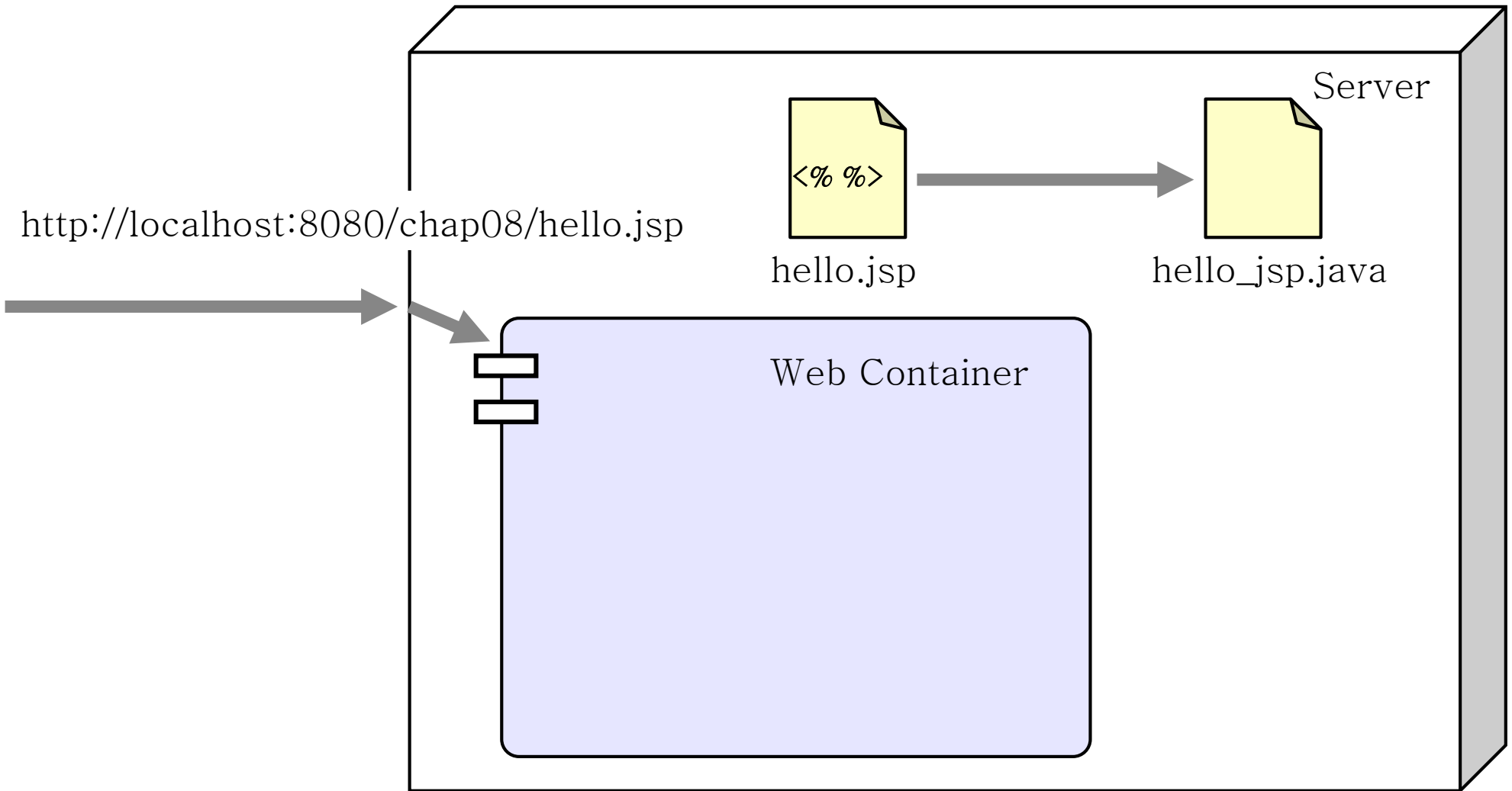
- Step 2: 생성된 java파일을 확인한다.

```
C:\practice\workspace\SE4_WEB_LAB\
web\WEB-INF\jspwork\org\apache\jsp\chap08\hello_jsp.java
```

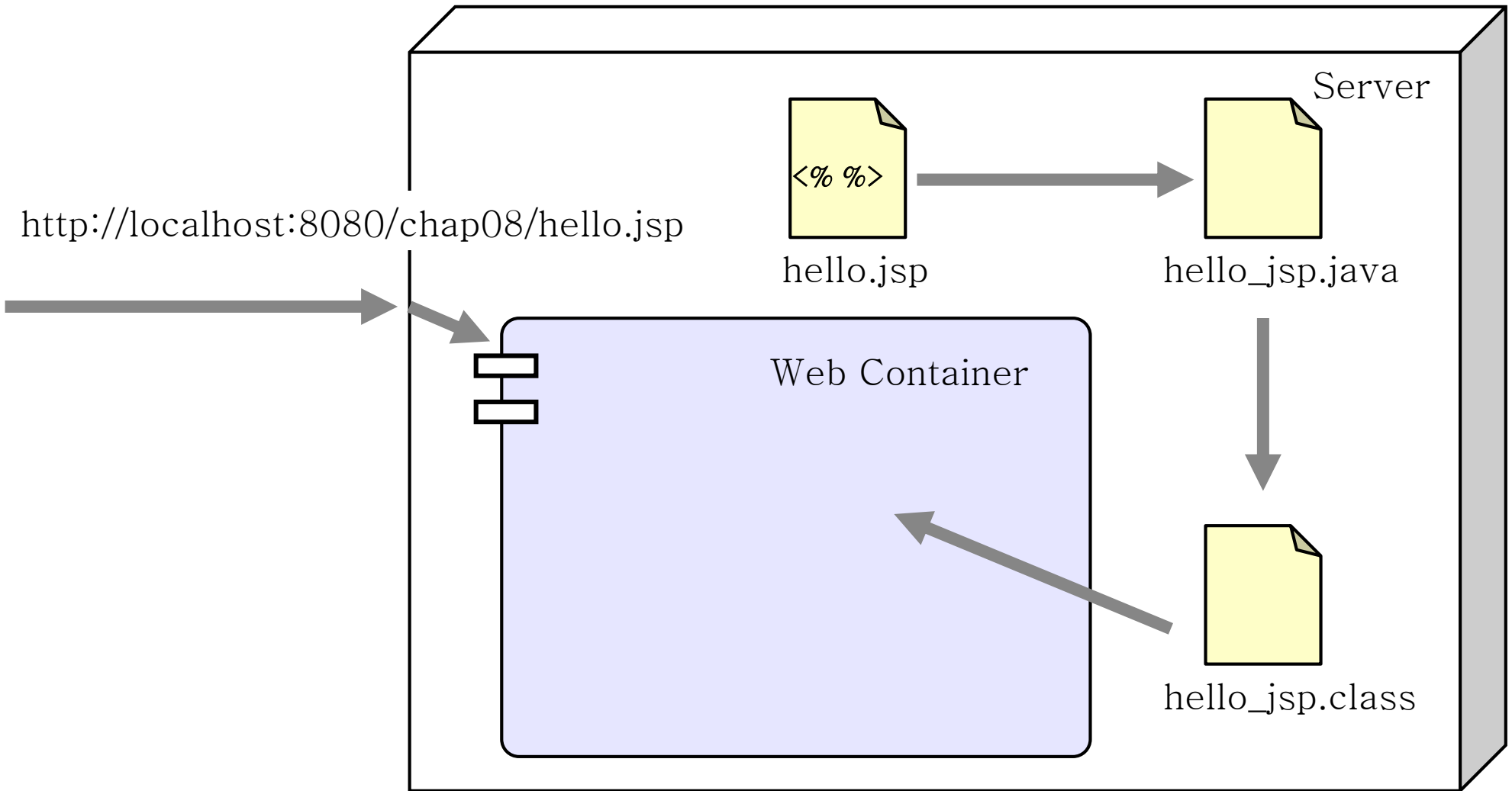
- 참고 : 변환된 java 파일 생성 위치 지정-ROOT.xml

```
<?xml version='1.0' encoding='euc-kr'?>
<Context docBase="C:\practice\workspace\SE4_WEB_LAB\web"
        path=""
        workDir="C:\practice\workspace\SE4_WEB_LAB\web\WEB-INF\jspwork"
        reloadable="true">
</Context>
```

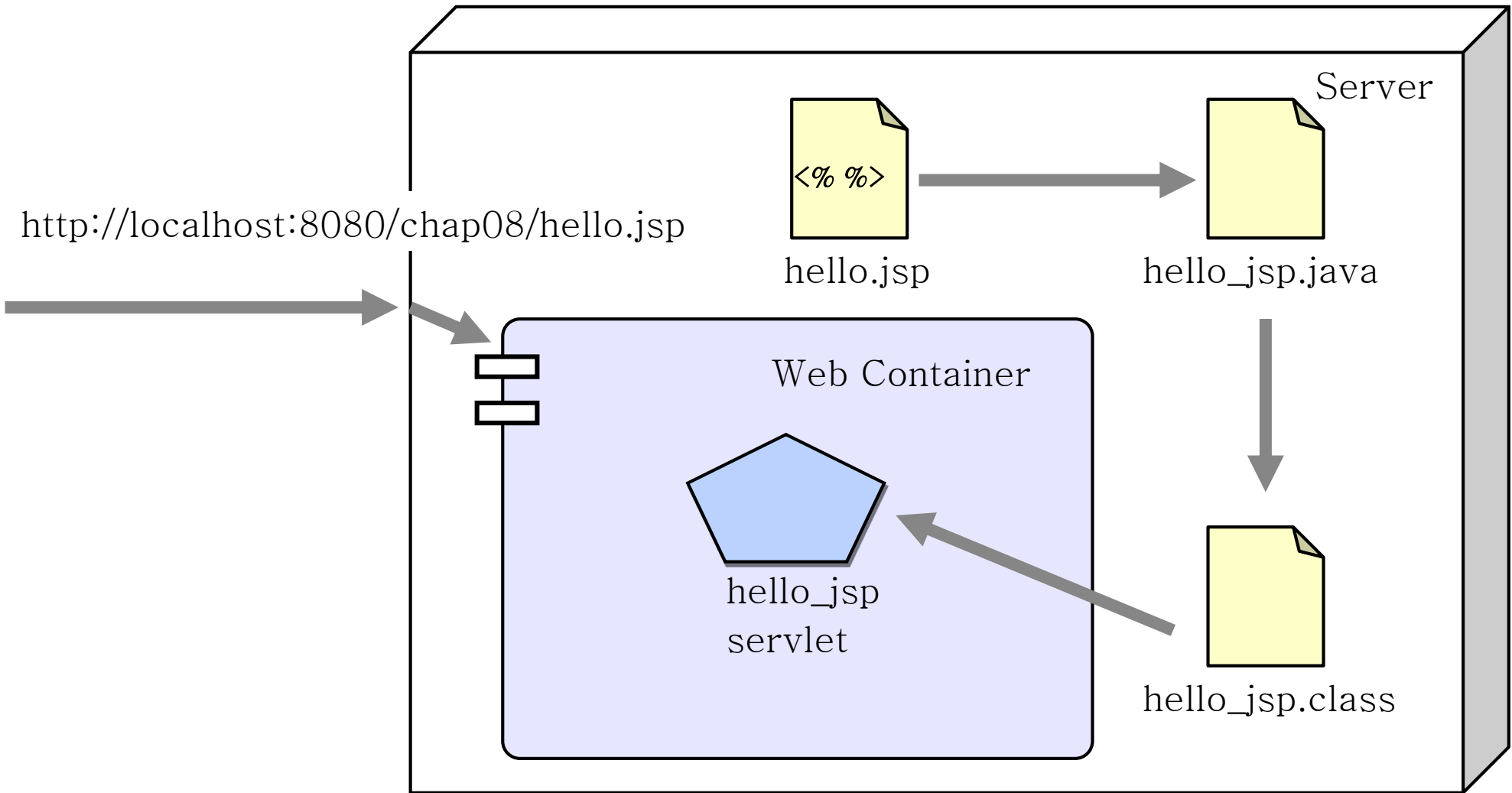
- JSP에 대해 **첫 번째 요청**이 오면, Web Container는 .jsp를 Servlet(.java)으로 변환 한다.



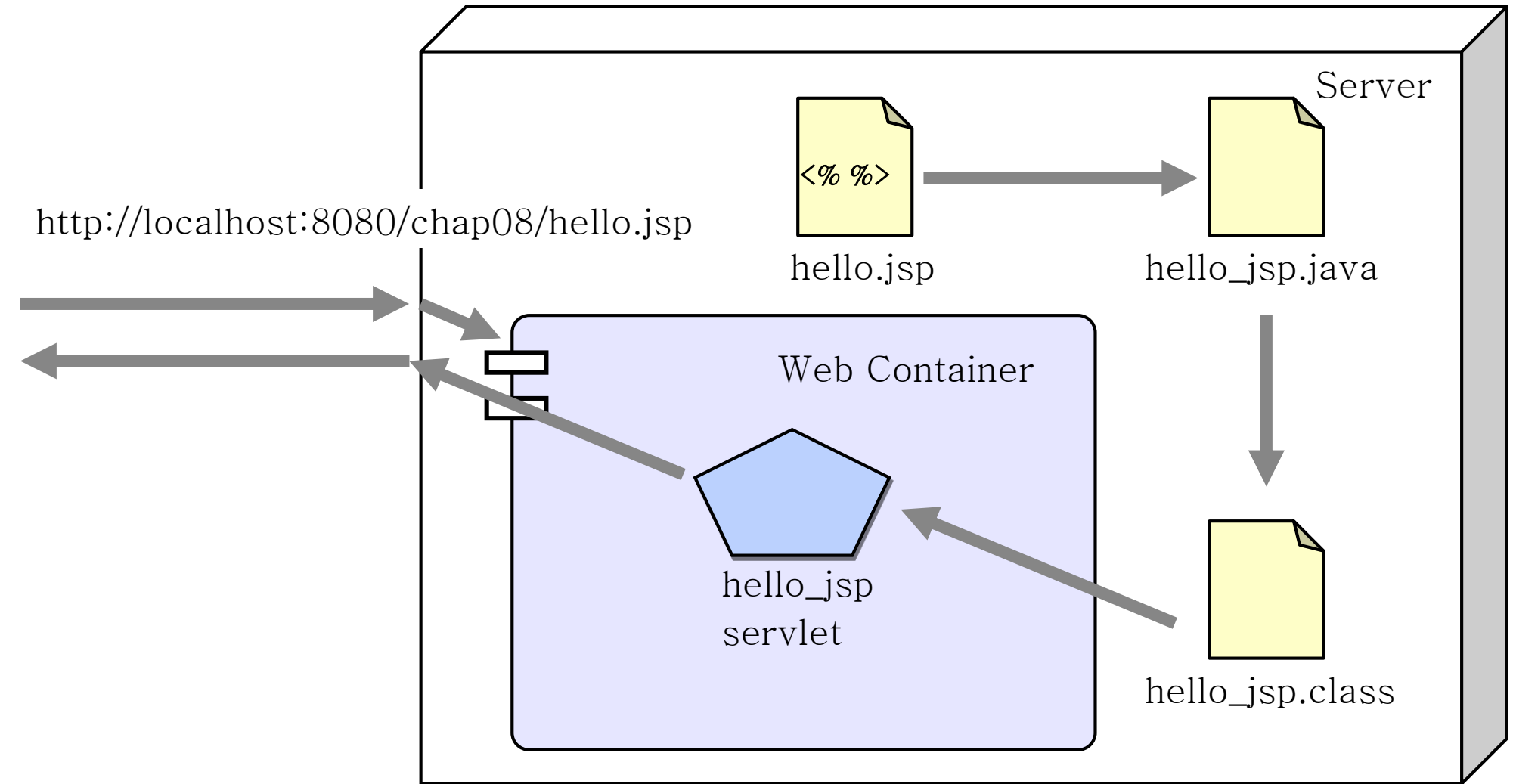
- Web Container는 .java 를 .class로 컴파일 하고, Web Container의 JVM에 의해 컴파일 된 클래스 파일은 load된다.



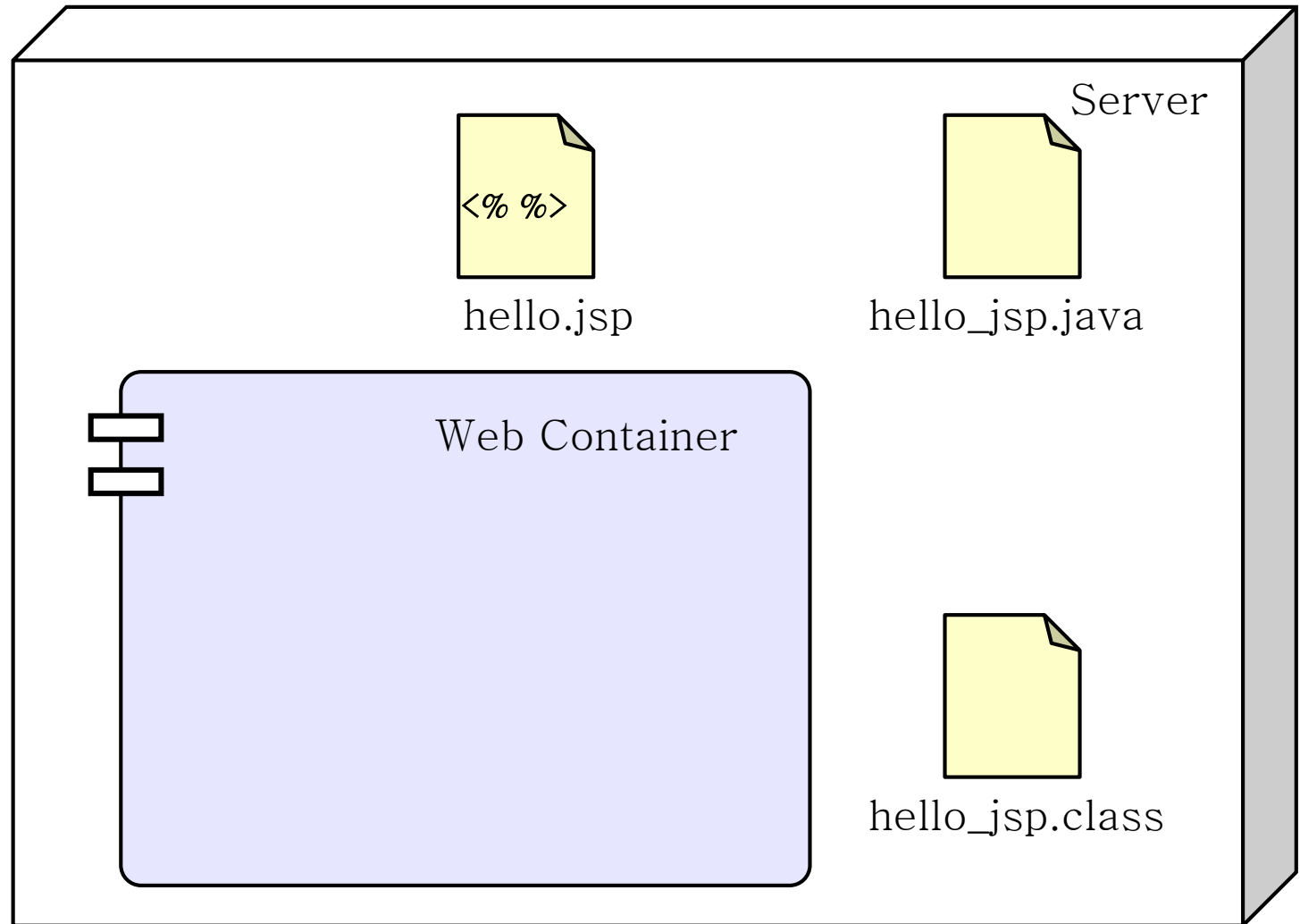
- Web Container는 servlet 인스턴스를 생성하고, jspInit 메소드를 호출한다.



- Web Container는 `_jspService` 메소드를 호출하여 사용자 요청을 처리하고, 응답을 준다.

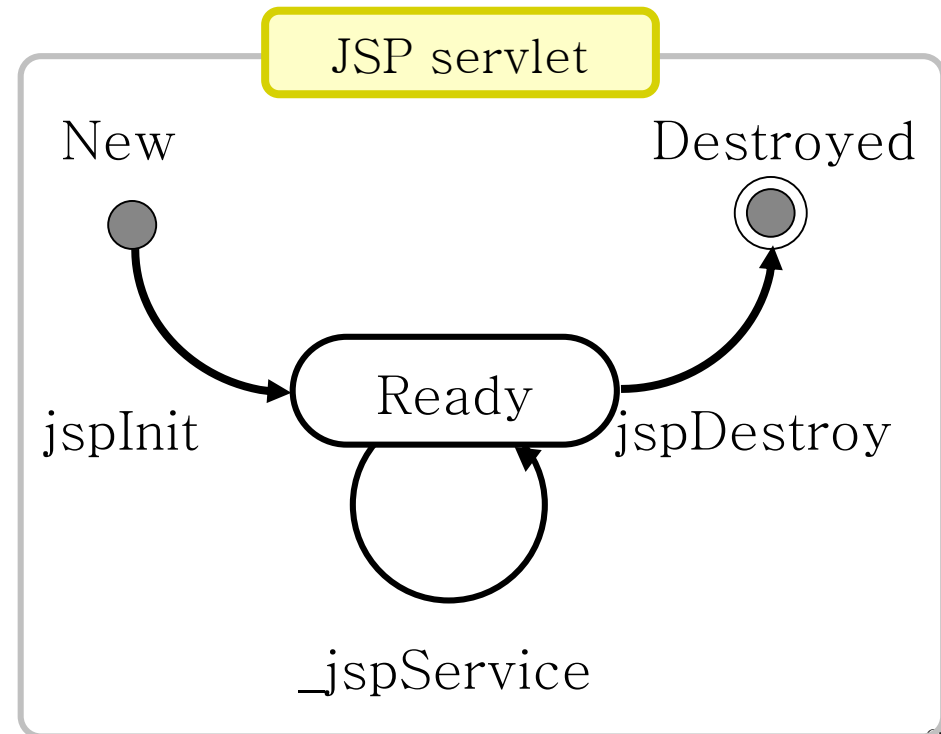
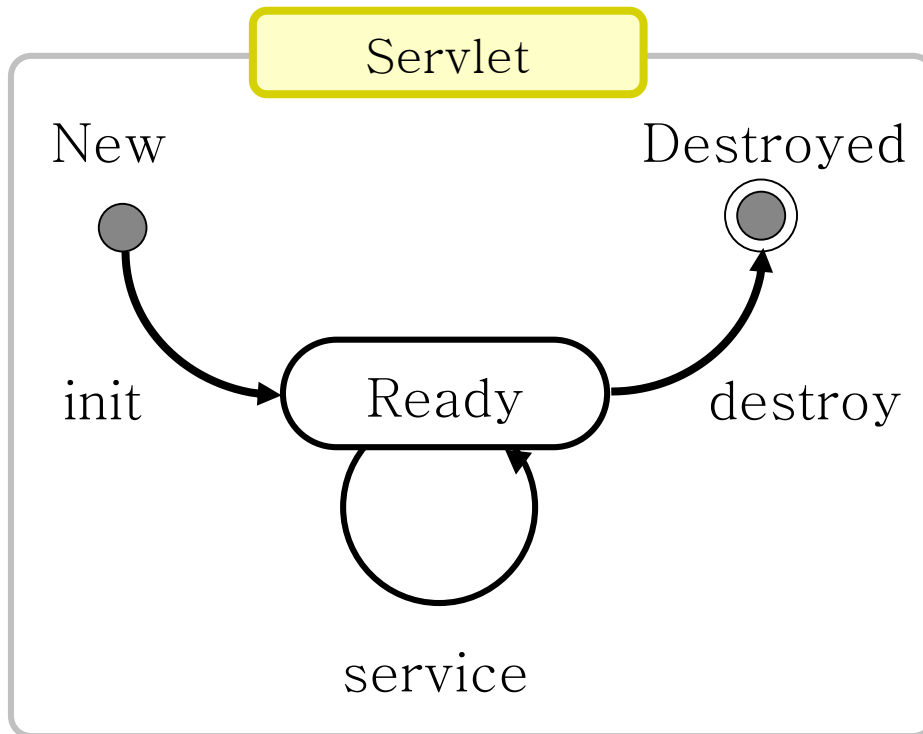


- Web Container가 JSP servlet 인스턴스를 제거할 때는, `jspDestroy`를 호출하여 마무리 작업을 한다.



JSP 특징

- ❑ JSP 파일을 변경하지 않는다면, .jsp파일에 대한 컴파일은 다시 일어나지 않는다.
- ❑ JSP 파일을 변경할 때 마다, Web Container는 compile, load, initialization의 과정을 수행한다.
- ❑ JSP 파일 변경시에는 Tomcat을 shutdown/start과정을 할 필요가 없다.
- ❑ JSP의 배포 환경은 html과 같다.
- ❑ 주의 : 구 버전의 JSP파일을 overwrite할 경우 제대로 반영이 되지 않는 경우가 발생할 수 있다.



Comments	<code><%-- comments --%></code>
Directive tag	<code><%@ directive %></code>
Declaration tag	<code><%! decl %></code>
Scriptlet tag	<code><% code %></code>
Expression tag	<code><%= expr %></code>

❑ HTML comments (XML도 동일)

- HTTP response로 보내지지만, 화면에는 보이지 않는다. (소스보기로 볼 수 있다.)

```
<!-- This is HTML Comments -->
```

❑ JSP page comments

- JSP 파일 내에서만 존재하고, servlet code로 바뀌어 질 때는 포함되지 않는다.

```
<%-- This is JSP Comments --%>
```

❑ Java comments

- servlet code에는 존재하나, HTTP response로 전해지지 않는다.

```
<%  
    /* This is Java Comments */  
%>
```

❑ 전체 JSP page에 영향을 미치는 정보를 기술할 때 쓰인다.

❑ 문법

```
<%@ DirectiveName [attr="value"]* %>
```

❑ Directive Types

- page, include, taglib 등 3종류가 있다.

```
<%@ page session="false" %>
```

```
<%@ page contentType="text/html; charset=euc-kr" %>
```

```
<%@ page import="java.io.*" %>
```

```
<%@ include file="incl/copyright.html" %>
```

- ❑ Servlet 클래스의 멤버변수/멤버메소드에 해당하는 코드를 작성하게 해주는 tag이다.

- ❑ 문법

```
<%! JavaClassDeclaration %>
```

- ❑ Example

- 멤버변수 선언

```
<%! public static final String DEFAULT_NAME="World"; %>  
<%! int counter = 0; %>
```

- 멤버메소드 선언

```
<%! public String getName(HttpServletRequest req) {  
    return req.getParameter( "name" );  
}  
  
%>
```

❑ `_jspServlet` 메소드의 로컬 변수와 코드를 작성하는 tag이다.

❑ 문법

```
<% JavaCode %>
```


❑ Example

▪ 로컬변수 선언

```
<% int i = 0; %>
```

▪ 메소드 내용 코드

```
<%  
    if ( i > 10 ) {        if ( i > 10 ) {  
%>                          out.print("I is a b..");  
I is a big number.        } else {  
%>                          out.print("I is a s..");  
    } else {                }  
%>                          }  
I is a small number.  
%>  
    }  
%>
```



❑ out.print() 의 역할을 한다.

❑ 문법

```
<%= JavaExpression %>
```

❑ Example

```
<B>Ten is <%= ( 2 * 5 ) %></B>
```

```
The current day and time is: <%= new java.util.Date() %>
```

❑ 주의 : ‘;’을 붙이지 않는다.

```
<%= new java.util.Date() ; %>
```



```
<% out.print( new java.util.Date() ; ) ; %> → error
```

❑ Scriptlet tag와 Expression tag에서 사용할 수 있는 암시적으로 선언된 변수

Variable Name	Description
request	HttpServletRequest 객체 참조 변수
response	HttpServletResponse 객체 참조 변수
out	JspWriter 객체 참조 변수
session	HttpSession 객체 참조 변수
application	ServletContext 객체 참조 변수
config	ServletConfig 객체 참조 변수
pageContext	단일 요청의 환경에 대한 변수
page	자바 클래스의 this와 동일
exception	발생 하는 Throwable 객체에 대한 참조 변수

- ❑ 한 JSP page 전체에 대한 지시구문을 선언한다.
 - 여러 개의 page 구문을 사용할 수 있지만, import 속성을 제외하고는 한 페이지에 한 번씩만 선언할 수 있다.
 - page 지시어는 JSP 파일의 어느 위치에 와도 상관 없으나, 가장 첫 부분에 사용하는 것이 좋다.

```
<%@ page import="java.util.Date" %>
```

```
<%@ page contentType="text/html; charset=euc-kr" %>
```


❑ import

- 필요한 package를 import 한다. 여러 package import시 ','기호로 구분한다.

```
<%@ page import="java.io.*, java.util.Date" %>
```

❑ session

- JSP 페이지를 HTTP세션에 참여시킬지 여부를 정의한다.
- 값은 true(default) 또는 false

```
<%@ page session="true" %>
```

❑ errorPage

- 해당 JSP 페이지가 발생시키는 모든 runtime exception을 처리할 다른 JSP페이지를 지정한다.
- 값은 현재 웹 계층에 상대적이거나, 컨텍스트 루트에 상대적인 URL이다.

```
<%@ page errorPage="error.jsp" %>
```

```
<%@ page errorPage="/error/errorForm.jsp" %>
```

❑ isErrorPage

- 해당 JSP 페이지가 다른 JSP 페이지에 있는 `errorPage` 속성의 대상인지 정의한다.
- 값은 `true` 또는 `false`(default)
- `true`인 경우, `exception` 내장 변수를 참조할 수 있다.

```
<%@ page isErrorPage="false" %>
```

❑ contentType

- 출력 스트림의 MIME타입을 정의한다.

```
<%@ page contentType="text/html; charset=euc-kr" %>
```

참고 – chap08.WelcomeServlet.java

```
public class WelcomeServlet extends HttpServlet {

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

        request.setCharacterEncoding("euc-kr");
        response.setContentType( "text/html;charset=euc-kr" );

        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hello JavaServer Page</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<body bgcolor='white'>");

        String name = request.getParameter("name");

        if ( name == null || name.equals("") ) {
            out.println("<b>No Input Information.</b><br>");
            out.println("NAME : " + name + "<br>");
        } else {
            out.println( "<b>" + getWelcomeMessage( name ) + "</b>" );
        }

        out.println("</body>");
        out.println("</html>");
        out.close();
    }

    public String getWelcomeMessage( String name ) {
        return "Welcome :" + name;
    }
}
```

JSP 실습 – WelcomeServlet.java → welcome.jsp

- ❑ chap08.WelcomeServlet.java 를 JSP로 구현하여 보자.
- ❑ `WEB_APP_ROOT\Wchap08\welcome.jsp` 로 작성
- ❑ 요청 url : `http://127.0.0.1:8080/chap08/welcome.jsp?name=홍길동`

<!-- Content Type을 설정하는 Directive Tag를 작성 -->

```
<HTML>
<HEAD>
<TITLE>Hello JavaServer Page</TITLE>
</HEAD>
<BODY BGCOLOR='white'>
<%
    request.setCharacterEncoding("euc-kr");

    String name = request.getParameter("name");
    if ( name == null || name.equals("") ) {
%>
<B>입력 정보가 없습니다.</B><br>
NAME 값 : <%= name %><br>
<%    } else {
%>
<b><!-- getWeclomeMessage호출 결과를 출력하는 Expression Tag를 작성 -->
</b>
<%
    }
%>

<!-- getWeclomeMessage 메소드를 선언하는 Declaration tag 작성 -->`
</BODY>
</HTML>
```

```
<%@ page contentType="text/html; charset=euc-kr" %>
<HTML>
<HEAD>
    <TITLE>Hello JavaServer Page</TITLE>
</HEAD>
<BODY BGCOLOR='white'>
<%
    request.setCharacterEncoding("euc-kr");

    String name = request.getParameter("name");
    if ( name == null || name.equals("") ) {
%>
<B>No Input Information.</B><br>
NAME : <%= name %><br>
<% } else {
%>
<b> <%=getWelcomeMessage( name )%> </b>
<%
    }
%>
<%!
public String getWelcomeMessage( String name ) {
    return "Welcome :" + name;
}
%>
</BODY>
</HTML>
```

- ❑ JSP 페이지는 try-catch 블록 안에 넣을 수 없다.
- ❑ JSP 페이지에서 오류 페이지를 지정해서 exception 처리를 맡긴다.
- ❑ 하나의 JSP 페이지에 대해 오류 페이지를 하나만 지정할 수 있다. 예외마다 각기 다른 오류 페이지를 지정할 수 없다.

throw_error.jsp

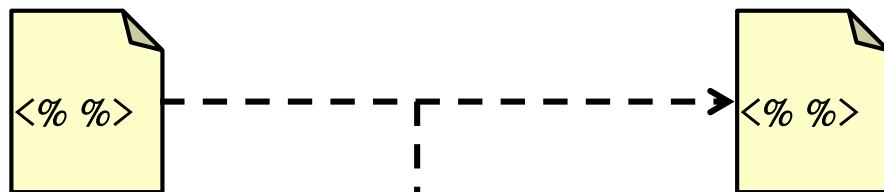
```
<%@page errorPage="/chap08/error/ExceptionPage.jsp"%>
```

exceptionPage.jsp

```
<%@page isErrorPage="true"%>
```

/chap08/throws_error.jsp

/chap08/error/exceptionPage.jsp



Web Container는 오류를
catch하여 error 처리
페이지로 전달한다.

JSP Page Exception Handling 실습

throw_error.jsp

```
<%@ page contentType="text/html; charset=euc-kr" %>
<%@ page session="false" %>

<!-- error page를 /chap08/error/exceptionPage.jsp로 선언 하시오. --%>
<%@ page errorPage="/chap08/error/exceptionPage.jsp" %>

<!-- This page will cause an "null pointer" exeception --%>

<HTML>
<!-- 생략 -->
</HTML>
```

exceptionPage.jsp

```
<%@ page contentType="text/html; charset=euc-kr" %>
<%@ page session="false" %>
<!-- 이 페이지가 error page임을 선언 --%>
<%@ page isErrorPage="true" %>

<HTML>
<!-- 생략 -->

</HTML>
```

Translation time error

Web Container가 JSP 페이지의 Scripting elements의 구문을 분석할 수 없을 때
예) “%>” 없이 “<%!” 만 사용한 경우

Compile time error

자바 문법 오류
예) 자바 문장을 끝낼 때, ‘;’ 사용하지 않음

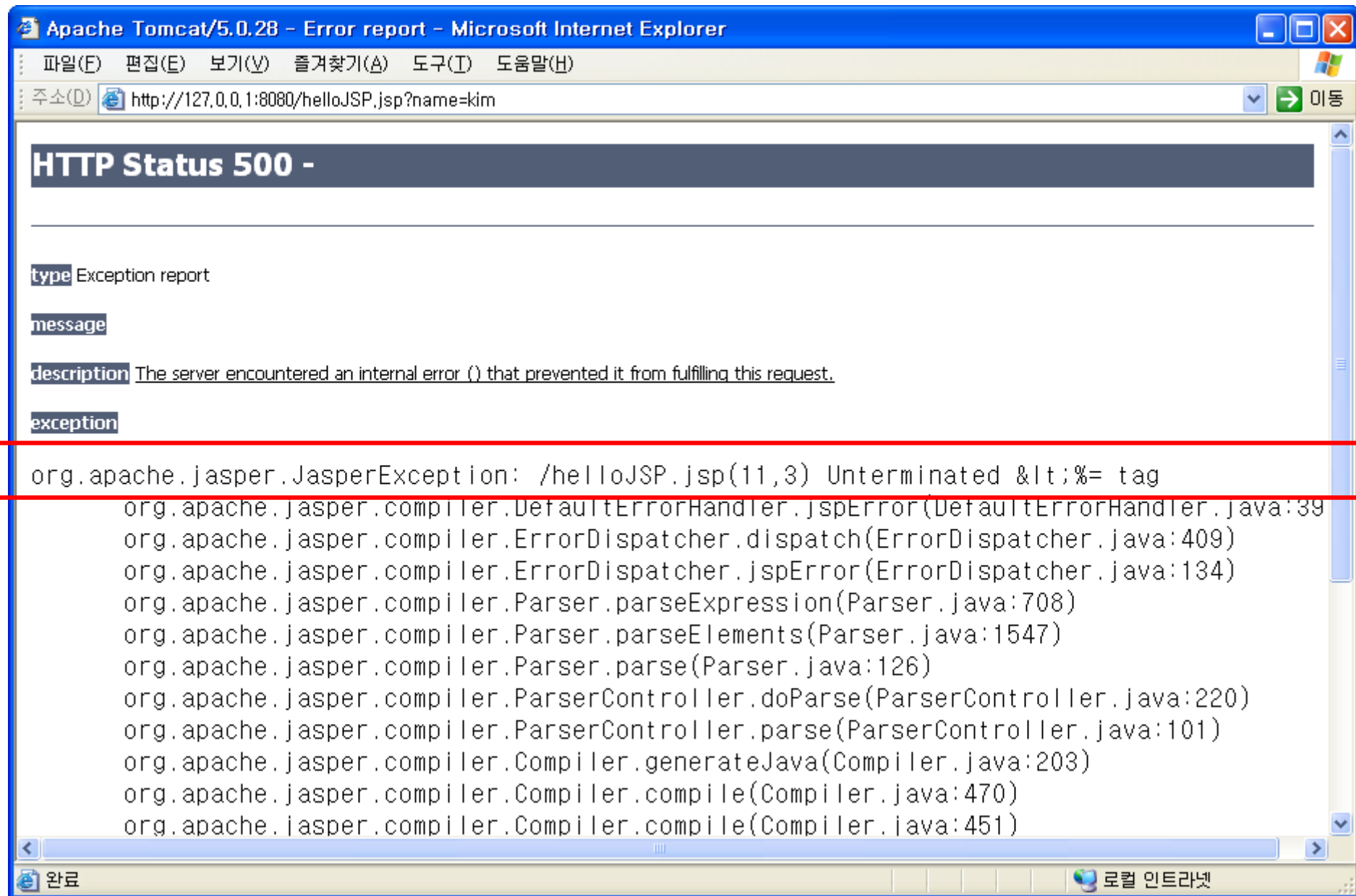
Runtime error

실행 도중, Exception이 발생하는 경우
예) `getParameter()` 인자에 HTML form 태그에 없는 이름을 사용하면, `getParameter()` 는 `null`을 return하는데 이 값을 가지고 조작하는 경우 `NullPointerException` 발생

http://127.0.0.1:8080/hello.jsp?name=kim

hello.jsp

```
<%@ page contentType="text/html; charset=euc-kr" %>
<HTML>
<HEAD>
<TITLE>Hello JavaServer Page</TITLE>
</HEAD>
<BODY BGCOLOR='white'>
<%
    request.setCharacterEncoding("euc-kr");
    out.print("Welcome : " );
%>
<%= request.getParameter("name")
</BODY>
</HTML>
```



http://127.0.0.1:8080/hello.jsp?name=kim

hello.jsp

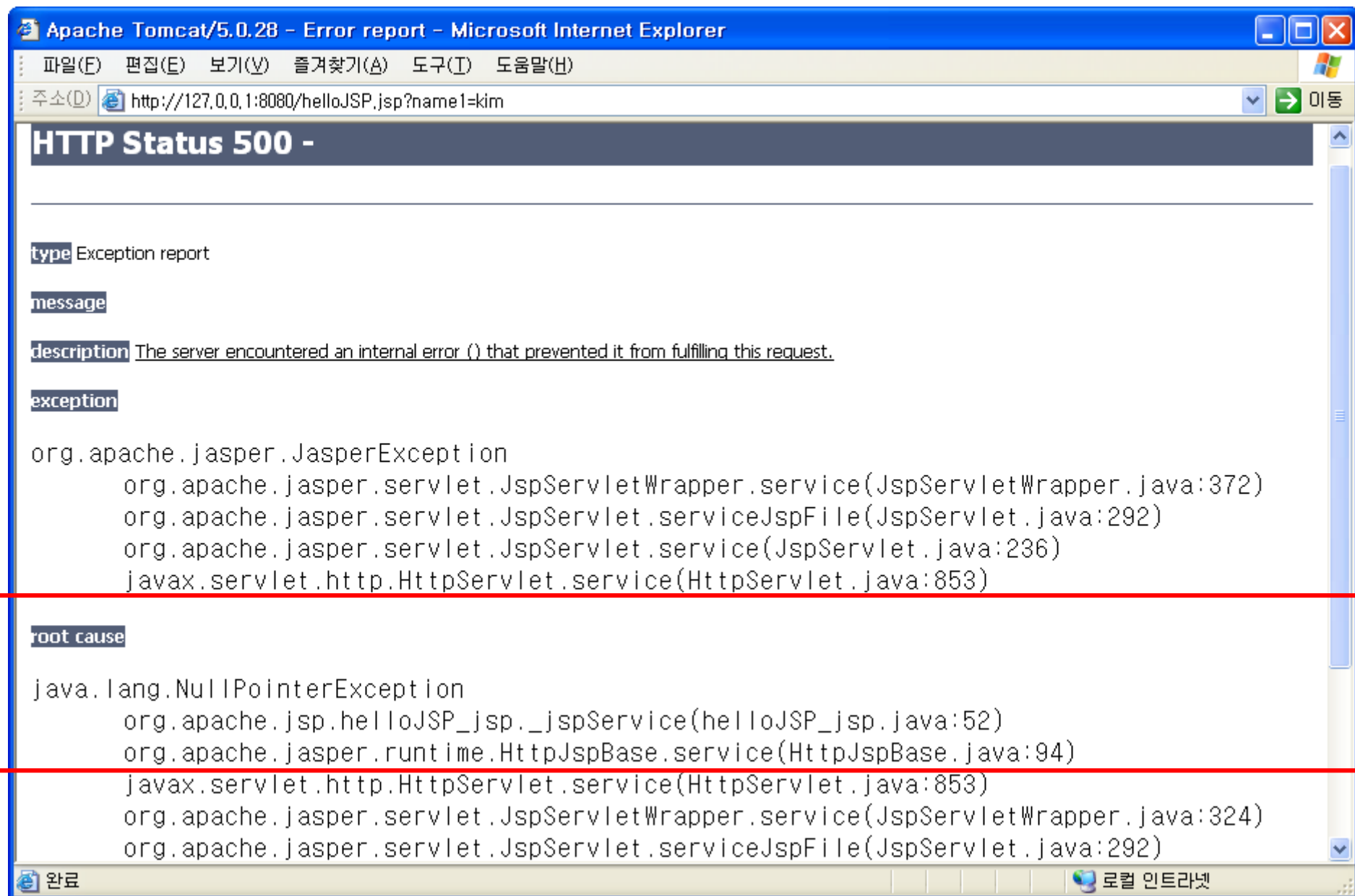
```
<%@ page contentType="text/html; charset=euc-kr" %>
<HTML>
<HEAD>
<TITLE>Hello JavaServer Page</TITLE>
</HEAD>
<BODY BGCOLOR='white'>
<%
    request.setCharacterEncoding("euc-kr");
    out.print("Welcome : " )
%>
<%= request.getParameter("name") %>
</BODY>
</HTML>
```



http://127.0.0.1:8080/hello.jsp?name1=kim

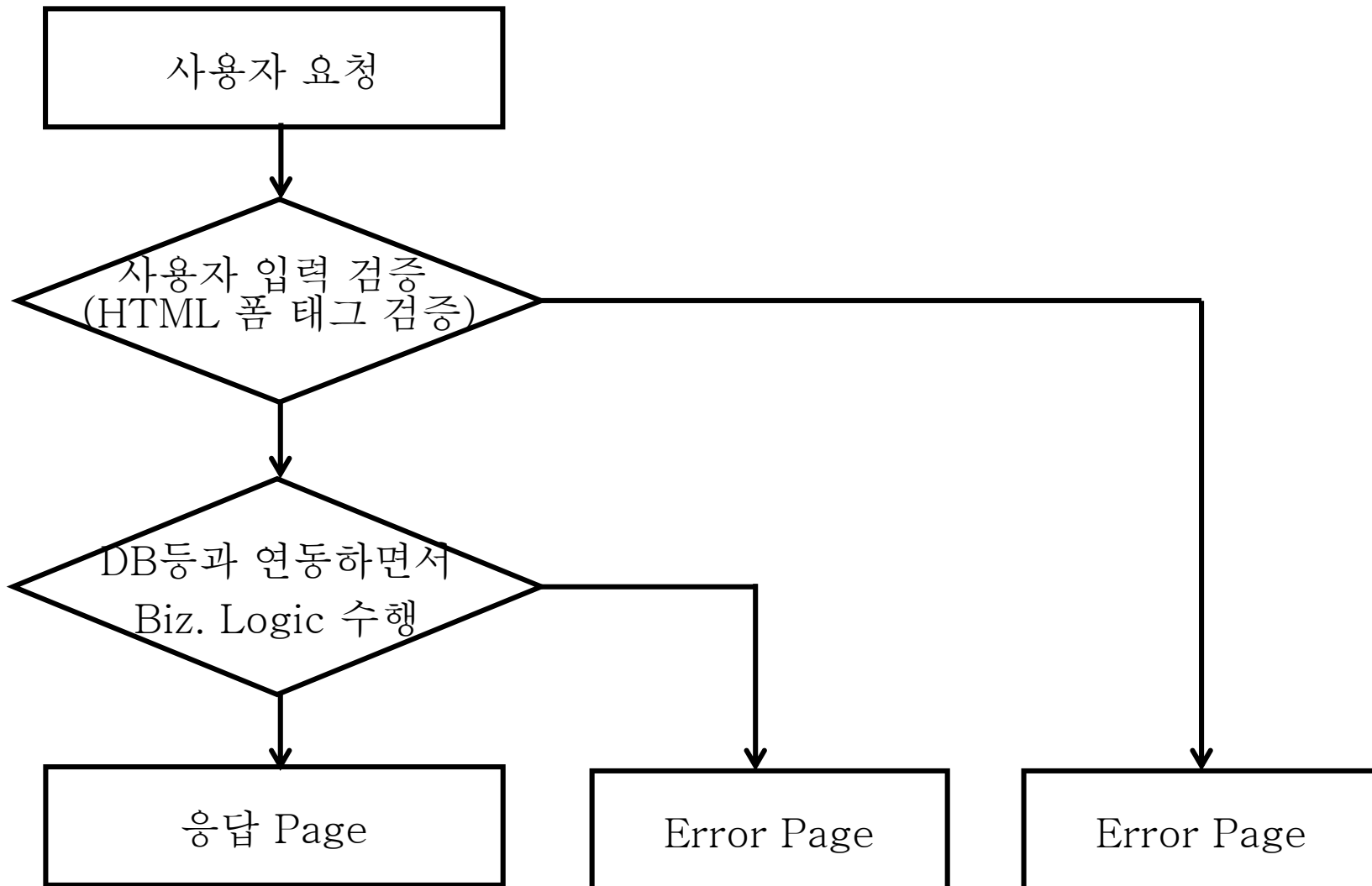
hello.jsp

```
<%@ page contentType="text/html; charset=euc-kr" %>
<HTML>
<HEAD>
<TITLE>Hello JavaServer Page</TITLE>
</HEAD>
<BODY BGCOLOR='white'>
<%
    request.setCharacterEncoding("euc-kr");
    out.print("Welcome :" );
    String name = request.getParameter("name");
    if ( name.equals("kim") ) {
        out.println( name );
    }
%>
</BODY>
</HTML>
```



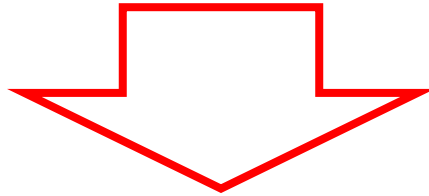
Developing Web Applications Using the MVC Pattern
Developing Web Applications Using the Model2 Architecture

□ 일반적인 Web Application의 실행 흐름



❑ Web Application의 내용을 doGet()/doPost() 안에 작성을 다 한다면?

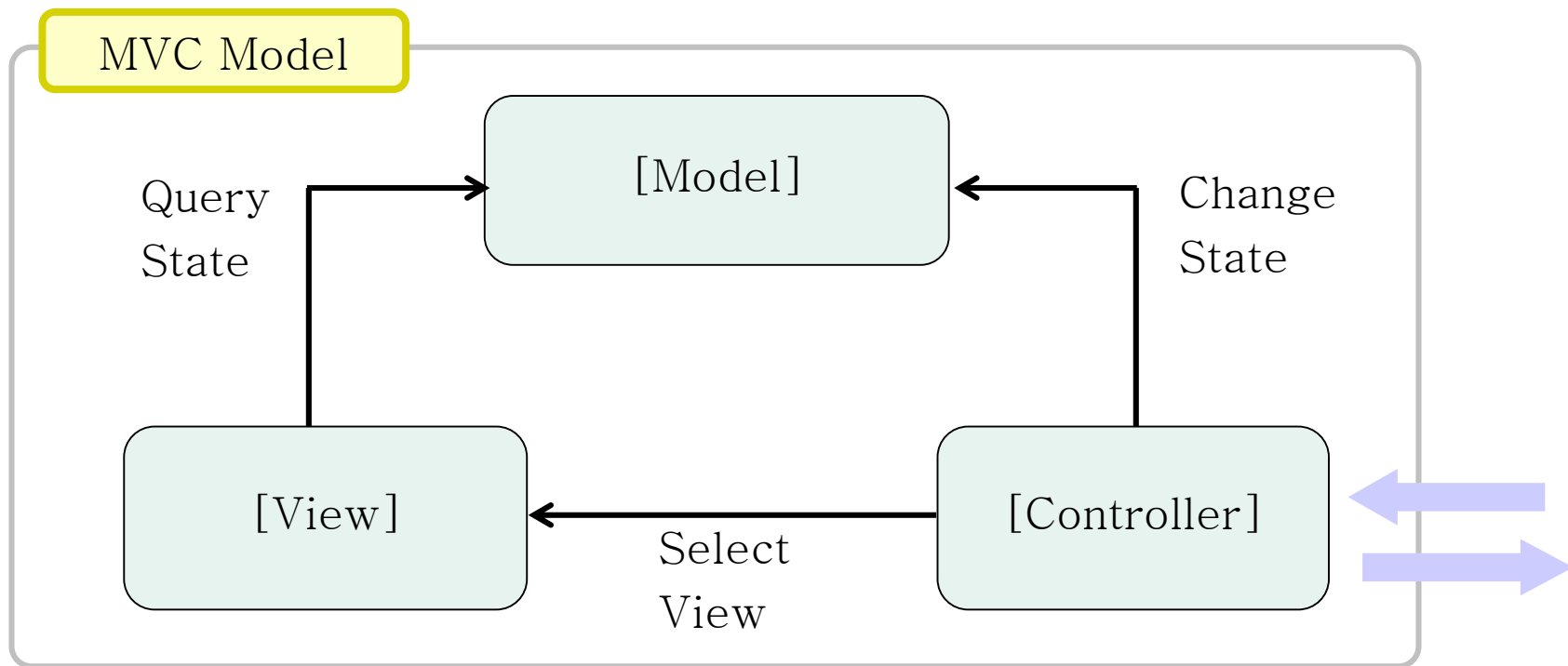
- 모듈화 문제 : 수정 사항이 발생한다면?
- Use Case 추가 : Admin유저가 추가 된다면?
- 기반 데이터를 파일에서 DB로 변경한다면?



- 각각의 역할에 따라 모듈을 분리하면 어떨까?

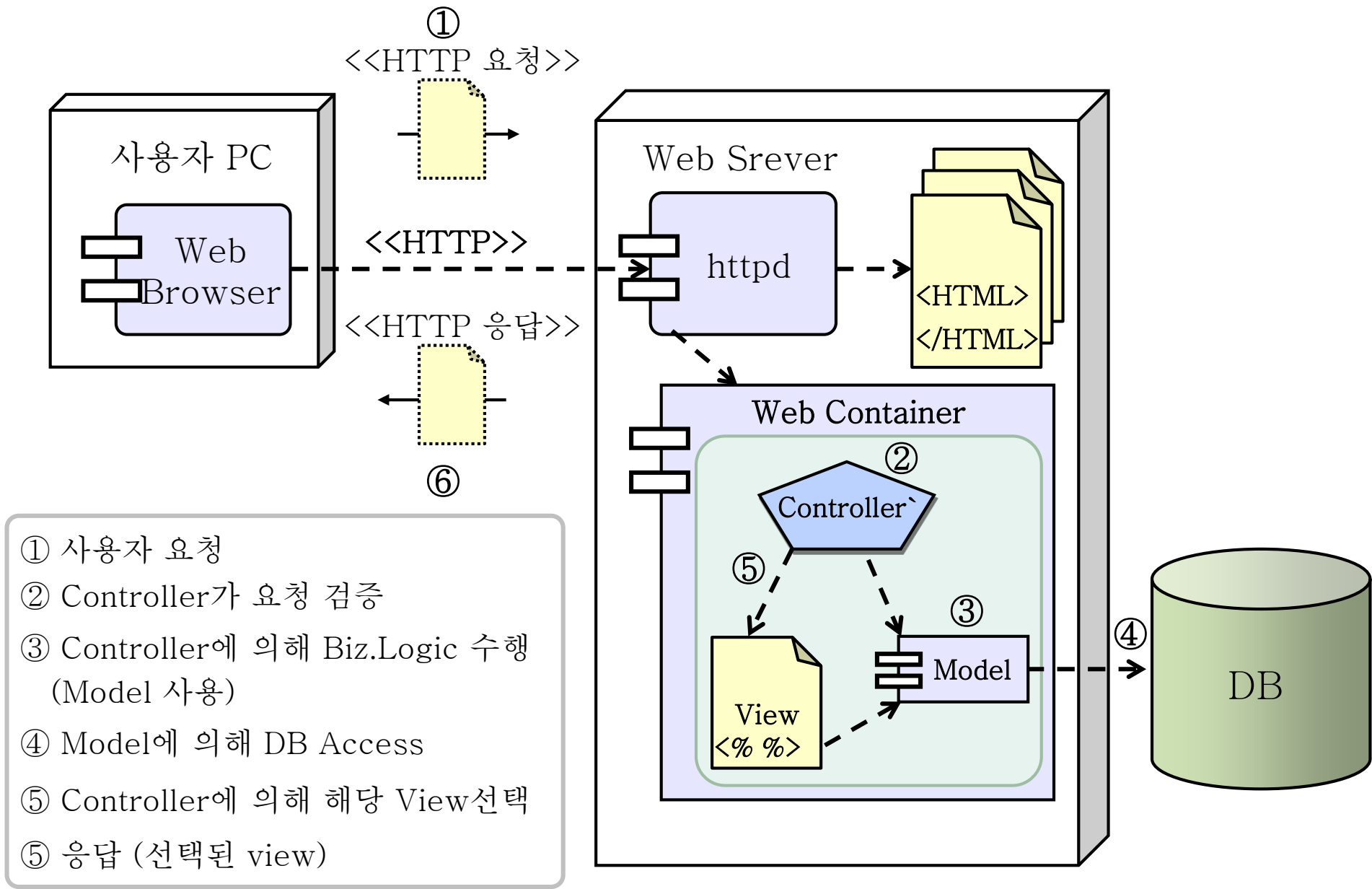
❑ Model-View-Controller (MVC) design

- 소프트웨어 시스템을 세 가지 타입의 구조 컴포넌트로 분할하는 소프트웨어 패턴
- Model : 비즈니스 로직 및 도메인 객체
- View : 사용자 인터페이스(UI)
- Controller : UI에 대한 사용자 입력을 해석, Model 이용 Biz.Logic 수행, 응답 View 선택



□ Using servlets and JSP pages

- servlet : Controller 역할. 폼 데이터를 검증하고, 폼 데이터를 사용하여 모델을 갱신하고, 응답으로 다음에 보여줄 View를 선택한다.
- JSP page : View 역할. Model로부터 응답을 생성하는데 필요한 데이터를 꺼내서 HTML응답을 만들어내고, 사용자 상호 작용이 가능한 HTML폼을 제공
- Java 클래스 : Model 역할. 웹 응용프로그램의 비즈니스 로직을 구현

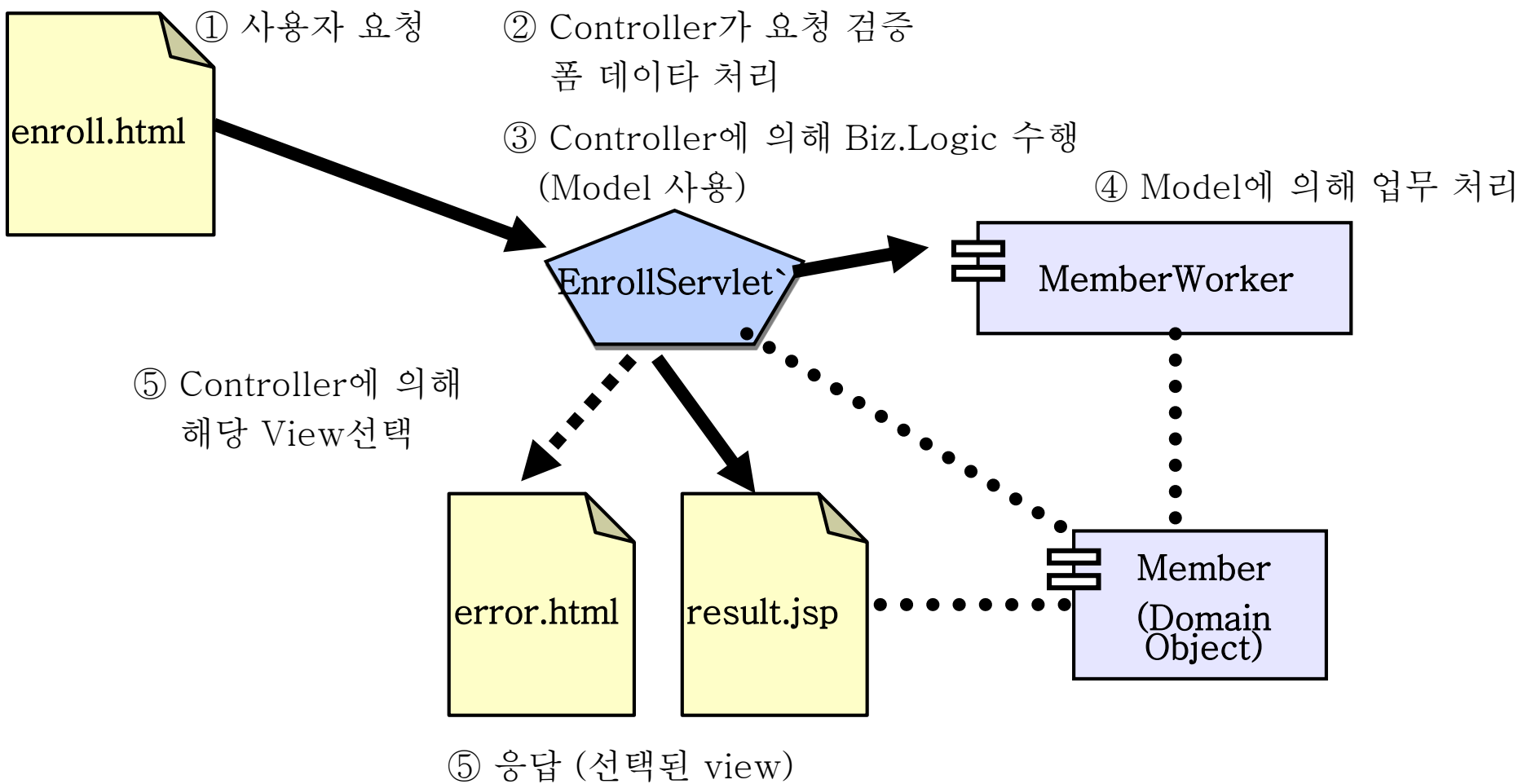


Model 2 Architecture

□ JSP/servlet을 이용한 Model 2 Architecture의 특징

- 빠른 속도
- 강력한 기능
- 쉬운 개발 (단, 숙련된 자바 개발자의 경우)
- 플랫폼 간 호환성
- 확장성
- 효율적인 유지 보수 가능

Model 2 Architecture 실습- Eroll



□ 동적 Page(JSP)인 경우

- 응답 view가 사용자 등록 성공 메시지와 같이 동적으로 변화하는 경우, jsp로 작성
- RequestDispatcher : servlet이 request 정보를 다른 동적 자원인 sevlet이나 jsp에게 forwarding 하게 해준다.

```
RequestDispatcher view =  
    req.getRequestDispatcher( "/success.jsp" );  
view.forward( req, res );
```

- 이 때, jsp페이지에 누구에 대한 등록 성공인지에 대한 객체 정보를 jsp페이지에 전달해야함.
- 이 정보를 전달되는 HttpServletRequest 객체에 실어 보낸다. (setAttribute())

```
req.setAttribute( "cust", cust );
```

- Example : 응답 view → success.jsp, 전송 데이터 → Customer 클래스의 ref 변수 cust

```
RequestDispatcher view =  
    req.getRequestDispatcher( "/success.jsp" );  
req.setAttribute( "cust", cust );  
view.forward( req, res );
```

□ 정적 Page(html)인 경우

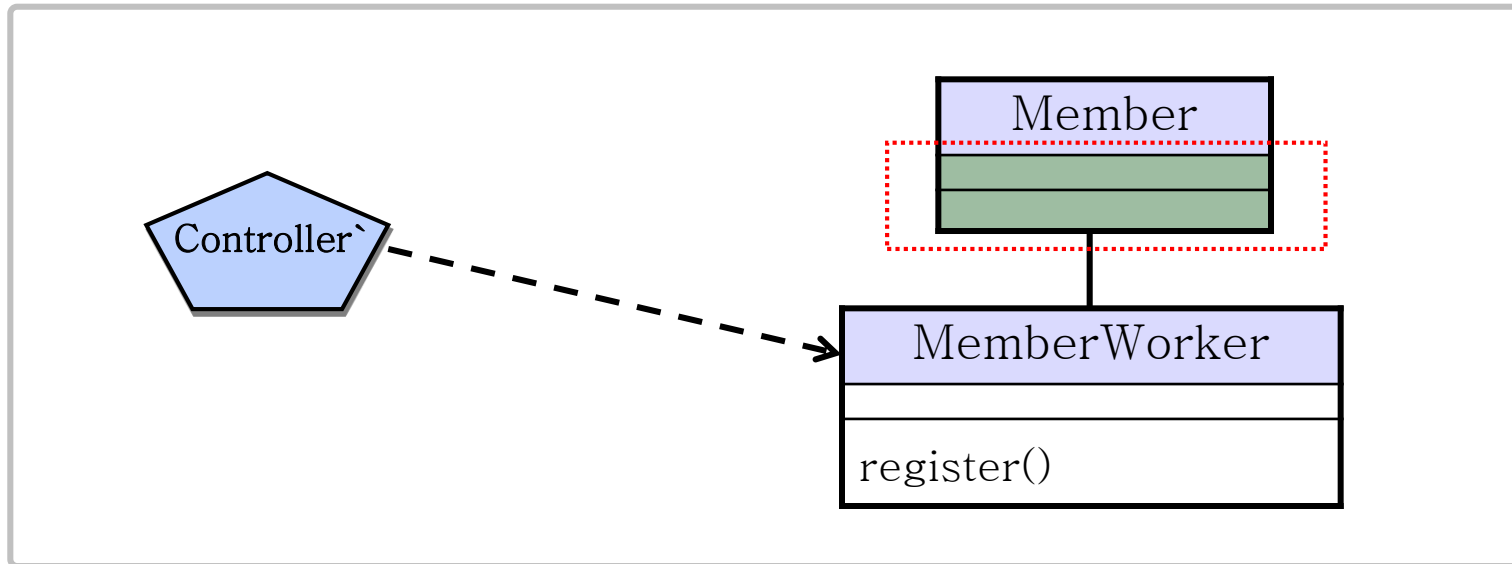
- 응답 view가 단순 메시지로 구성 되어, 정적인 html로 작성된 경우 응답 view에 전달 할 data가 필요 없고, 단순히 클라이언트에게 그 html을 전송만 해주면 된다.
- 이 때는, HttpServletResponse 가 갖고 있는 sendRedirect() 를 이용한다.
- Example : 응답 view가 msg.html인 경우

```
response.sendRedirect( "/msg.html" );
```


Integrating Web Applications With Databases

□ Model tier

- 비즈니스 로직 및 도메인 객체에 대한 내용을 담고 있다.
- 데이터를 Database 혹은 File등을 통해 얻고, 결과를 저장한다.
- 현 Model tier의 이슈
 - ▶ 파일을 통해 데이터를 관리하다가, DB로 변경할 수 있다.
 - ▶ DB 컬럼 이름 변경 등의 Database Schema가 변경될 수 있다.
 - ▶ 이러한 경우, Biz. Logic은 동일하지만 모델 클래스를 수정해야 한다.

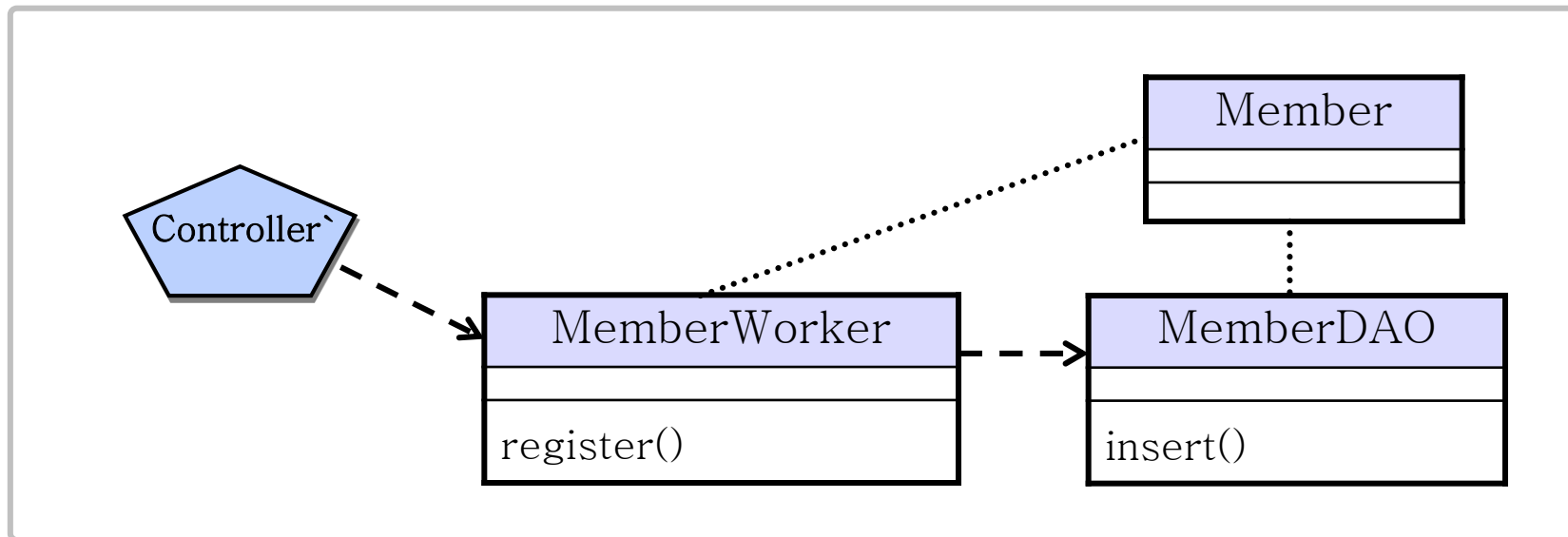


□ DAO (Data Access Object) Pattern

- Database Access 코드를 갖는 클래스를 따로 구현하는 것을 말한다.
- Biz. Logic과 DB Access 코드가 분리되어 진다.
- DB 변경 등에 따른, 코드 수정을 최소화 한다.

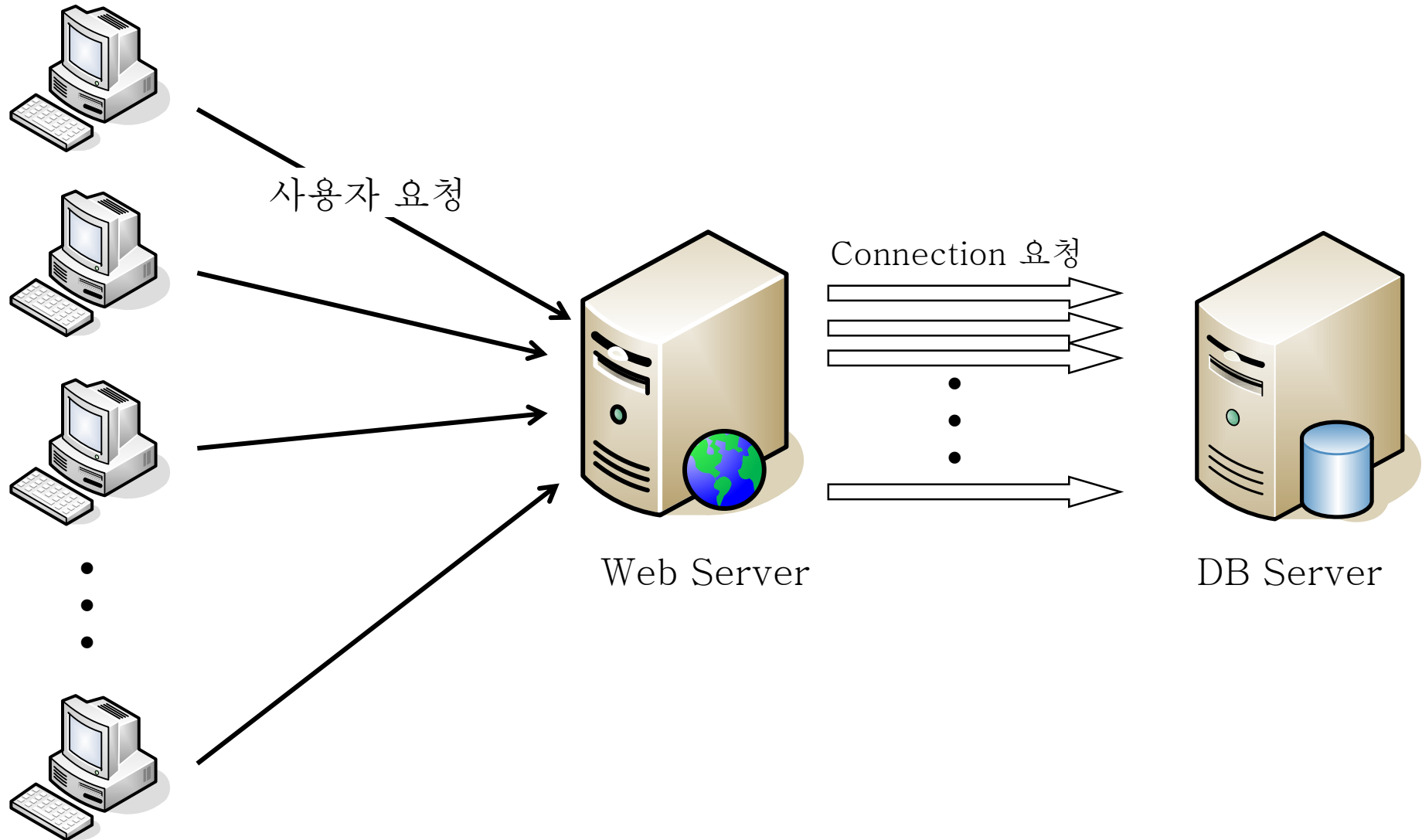
□ Model tier

- Business Service : DAO class를 통해 데이터를 Domain Object에 담고,
Biz. Logic 수행
- Domain Object : 데이터를 담는 클래스
- DAO class : Database Access 코드를 갖는 클래스



Database Access Problem

□ 동시에 100명이 요청한다면?

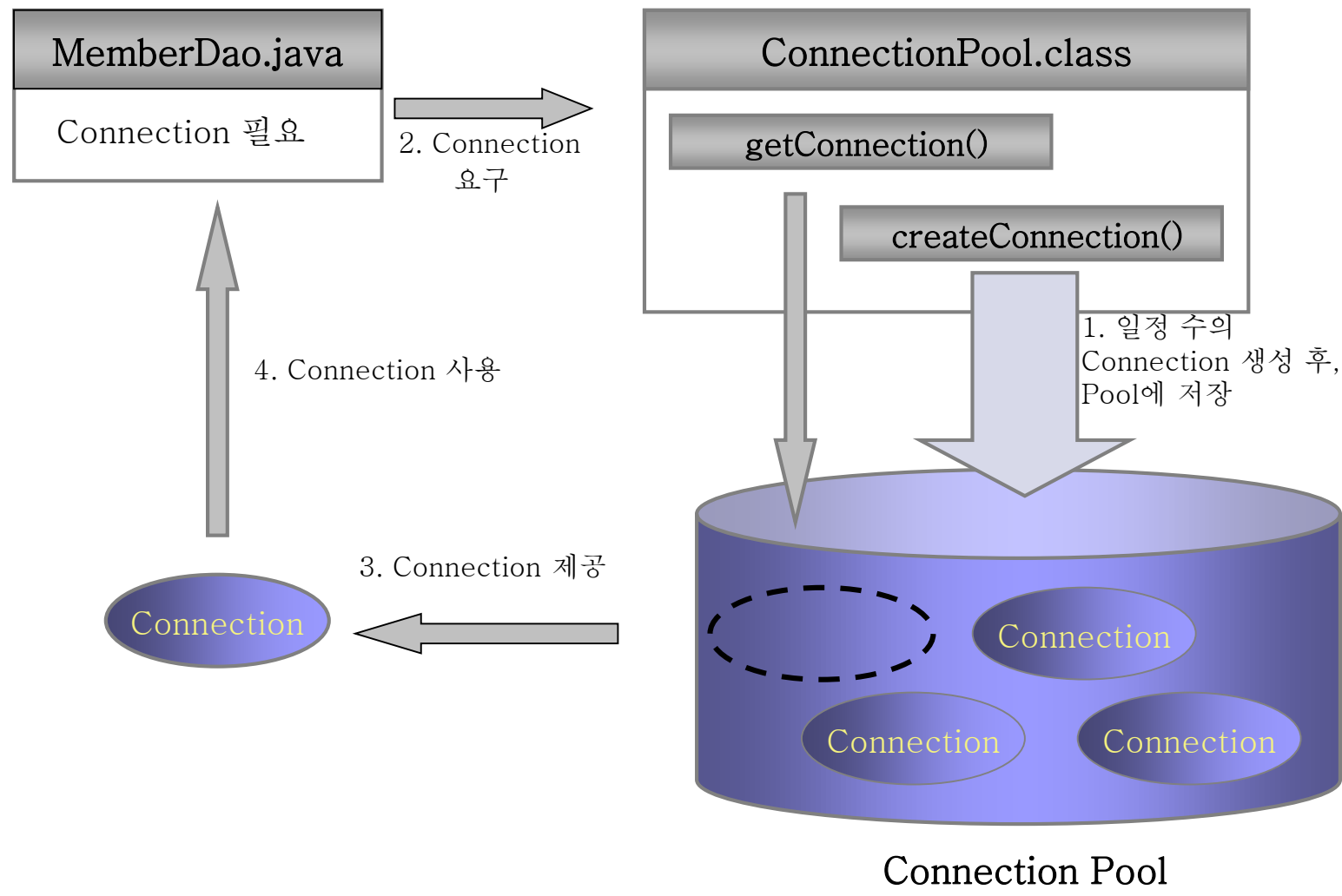


□ DB Connection Pooling

- Pooling 기법이란, 미리 데이터베이스 Connection을 여러 개 만들어서 특정 공간에 저장해 놓고, 여러 사용자가 필요할 때 마다 하나씩 꺼내서 사용하고 다시 집어 넣는 방식을 말한다.
- Pooling 기법
 1. Connection을 생성해서 보관
 2. Connection에 대한 요청이 들어오면, 보관중인 Connection중 하나를 넘겨줌
 3. 사용이 끝난 Connection을 다시 보관
- Connection Pooling의 장점
 1. 속도향상
 2. 자원공유
 3. Connection객체 제어

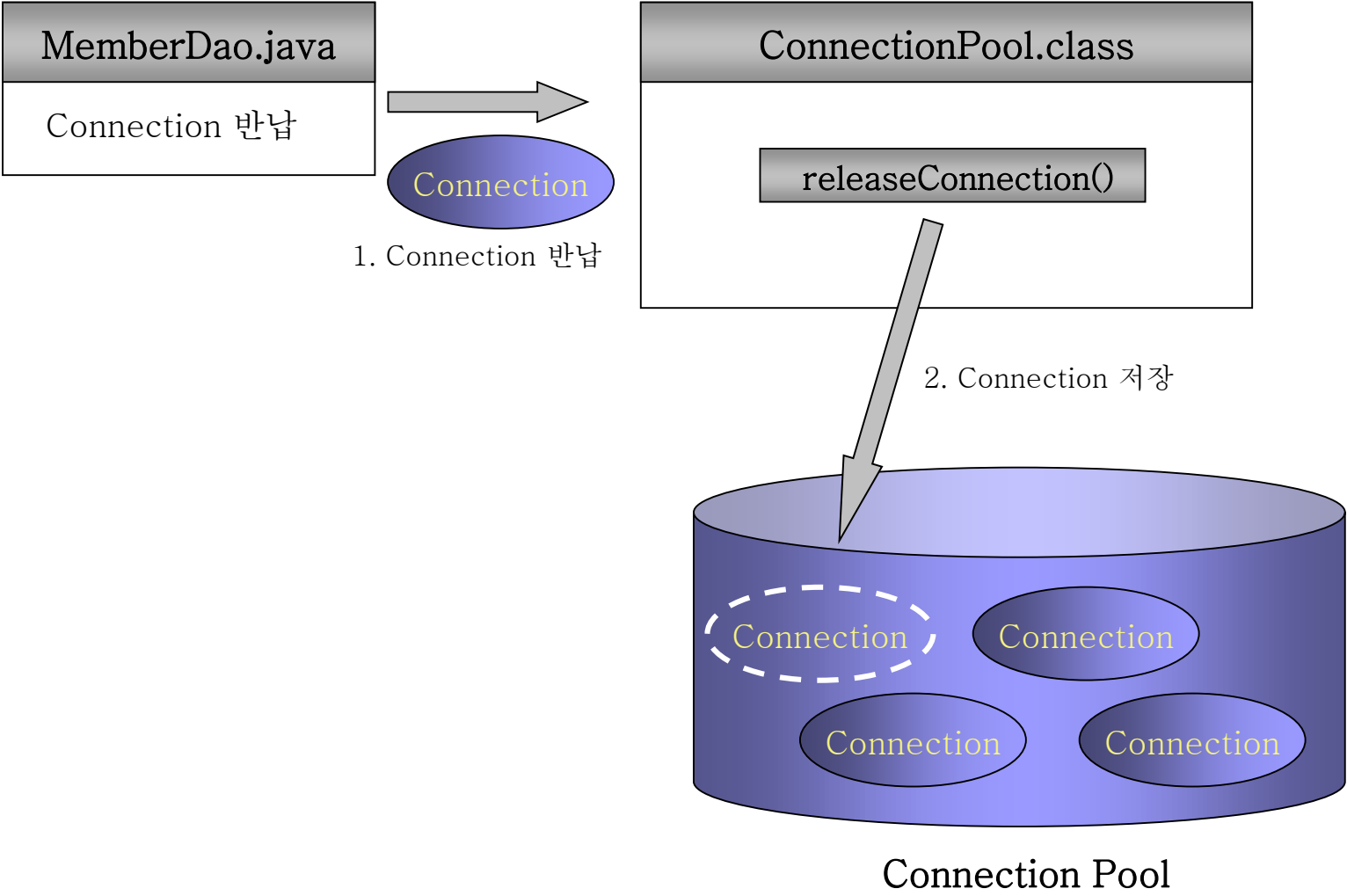
Connection Pool

❑ Connection 객체의 보관과 사용

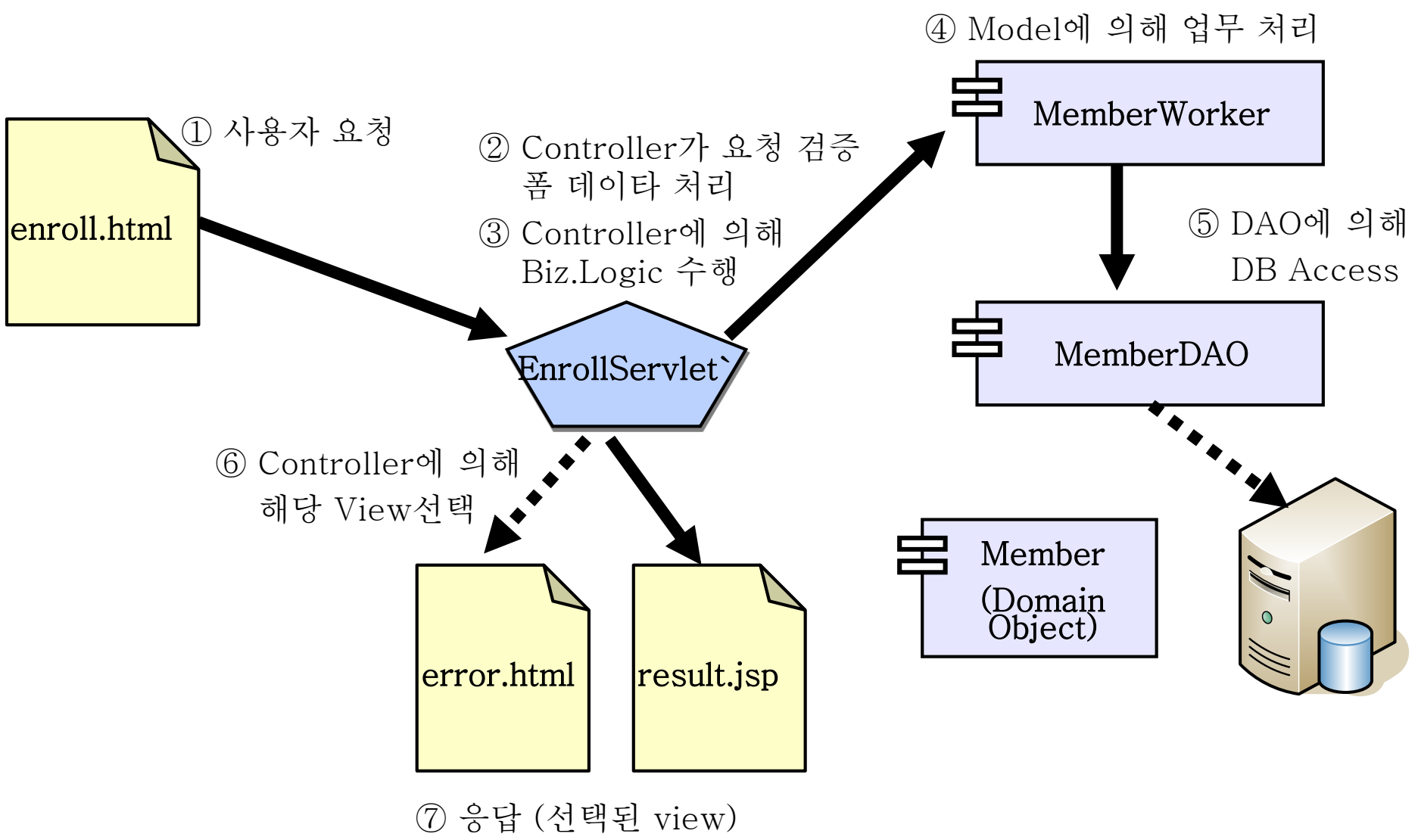


Connection Pool

❑ Connection 객체의 반환 및 저장



실습 – DAO / Connection Pool – Eroll



Building Reusable Web Presentation Components

1. Including JSP Page Fragments

- `include` directive
- `jsp:include` standard action

□ 복잡한 레이아웃을 가지는 Page의 경우, 공통 되는 일부분을 다른 파일로 저장한 후, 그 파일을 포함시킬 수 있다.

- 배너, 로고 등의 공통 부분을 추가한다.
- 각 페이지의 일관된 모양을 유지할 수 있다.

□ 조각(Fragment) 페이지

- HTML, 혹은 JSP 페이지의 일부분
- 주의: <HTML> 또는 <BODY> 태그가 있어서는 안 된다.

□ 조각 페이지를 포함하는 방법

```
<%@ include file="fragmentURL" %>
```

```
<jsp:include page="fragmentURL" />
```

□ 차이점

- include directive : 생성되는 자바 파일은 1개, 조각 페이지가 변경 되어도 자동 반영되지 않는다. → 현재 버전 반영됨.
- jsp:include Standard action tag: 생성되는 자바 파일 2개, 조각 페이지 동적 갱신. 성능 저하를 초래할 수 있다.

❑ 조각 페이지 – incl/test1.jsp

```
<%@ page contentType="text/html; charset=euc-kr" %>
<B>This is Test Page</B>
```

❑ Including 페이지 – helloDirective.jsp

```
<%@ page contentType="text/html; charset=euc-kr" %>
<HTML>
<HEAD>
<TITLE>Hello JavaServer Page</TITLE>
</HEAD>
<BODY BGCOLOR='white'>
안녕하세요
<br><br>
-----<br>
directive tag include를 사용한 결과입니다.<br>
<%@ include file="incl/test1.jsp" %><BR><BR>
-----<br>
</BODY>
</HTML>
```

❑ 조각 페이지 – incl/test2.jsp

```
<%@ page contentType="text/html; charset=euc-kr" %>
<B>This is Test Page</B>
```

❑ Including 페이지 – helloSA.jsp

```
<%@ page contentType="text/html; charset=euc-kr" %>
<HTML>
<HEAD>
<TITLE>Hello JavaServer Page</TITLE>
</HEAD>
<BODY BGCOLOR='white'>
안녕하세요
<br><br>
-----<br>
jsp standard tag include를 사용한 결과입니다.<br>
<jsp:include page="incl/test2.jsp" /><br>
-----
</BODY>
</HTML>
```