02: 웹 애플리케이션 아키텍처

학습 목표

- 서블릿 컨테이너의 정의와 역할 대해 이해한다.
- HTTP 요청과 HTTP 응답에 대해 이해한다.
- 서블릿의 작성, 배포에 대해 이해한다.
- MVC 디자인 패턴에 대해 이해하고 웹 애플리케이션 구조에 대해 이해한다.

1. 서블릿 컨테이너의 정의 및 역할

2. 서블릿 컨테이너의 HTTP 요청 처리 순서

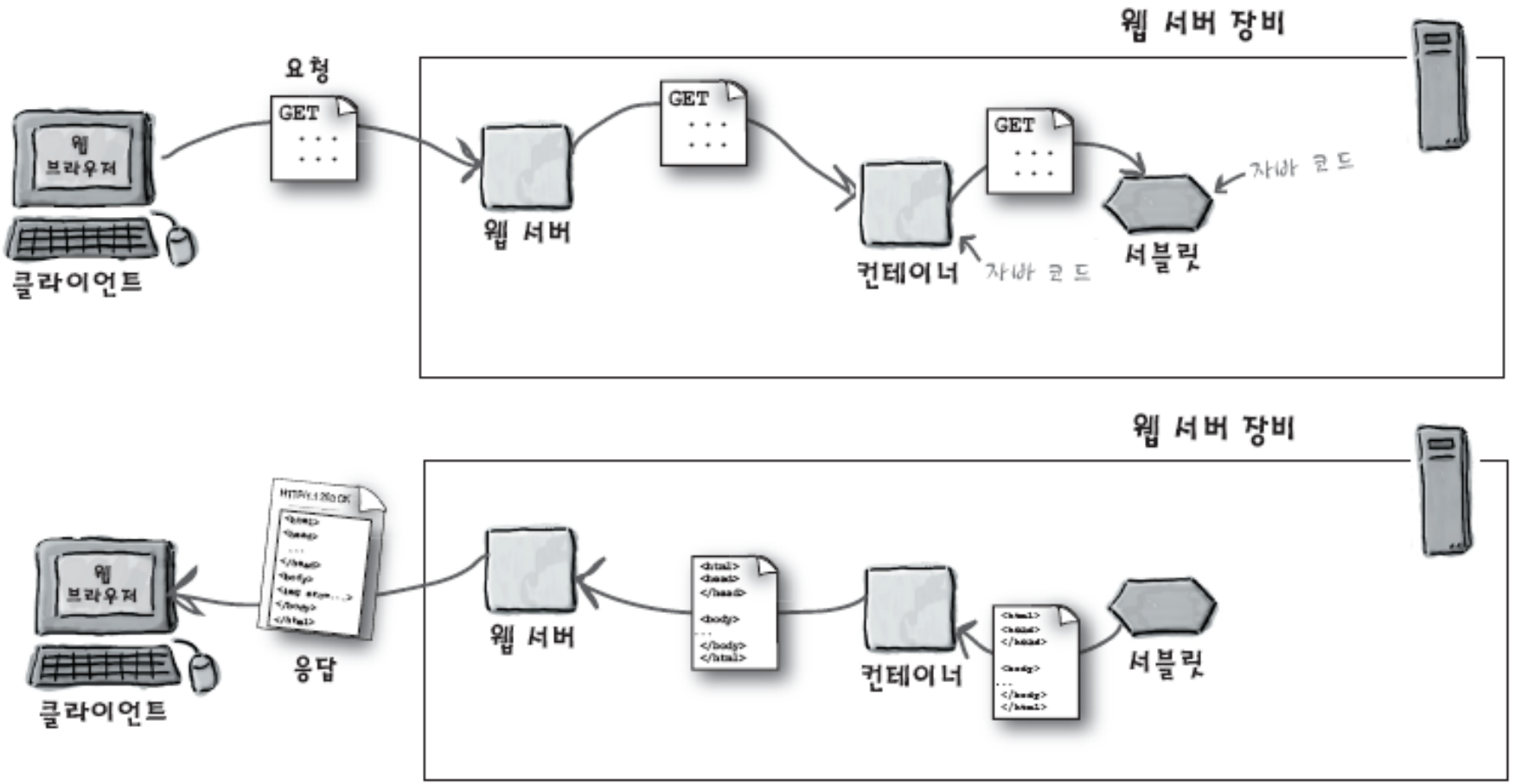
3. 서블릿 작성 및 배포

4. MVC 모델

5. J2EE 애플리케이션 서버

서블릿 컨테이너 >> 정의 및 역할

■ HTTP 요청에 의한 서블릿을 실행시키며 관리하는 프로그램.

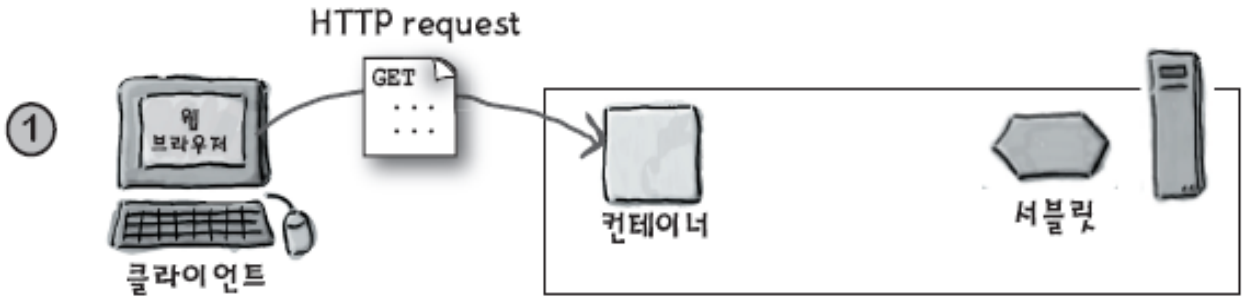


서블릿 컨테이너 >> 정의 및 역할

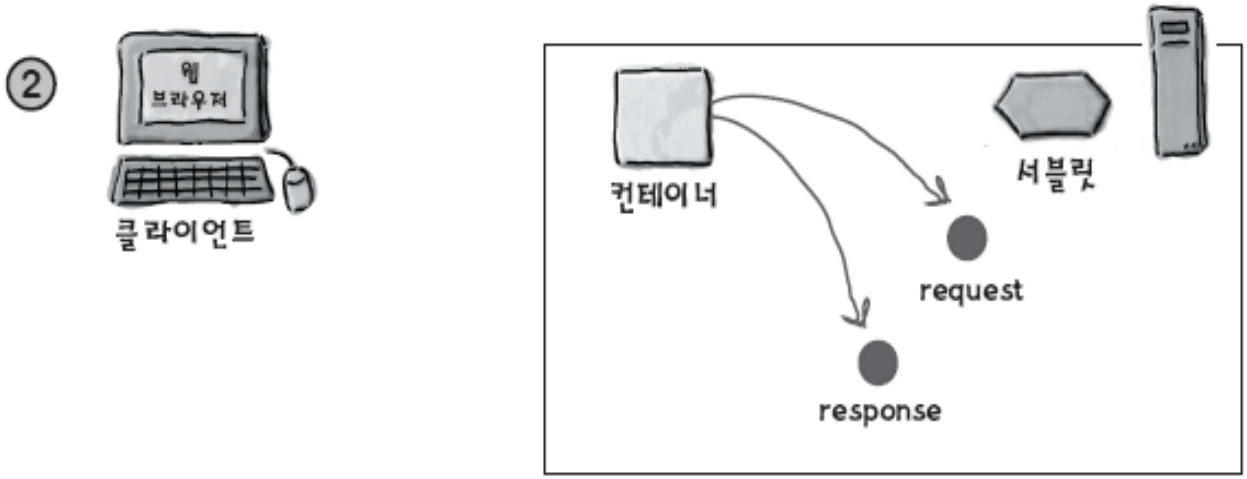
역할

- 통신 지원
- 서블릿 생명주기 관리
- 멀티 스레딩 지원
- 보안 관리
- JSP 지원

서블릿 컨테이너 >> HTTP 요청 처리



사용자가 서블릿에 대한 링크를 클릭합니다.



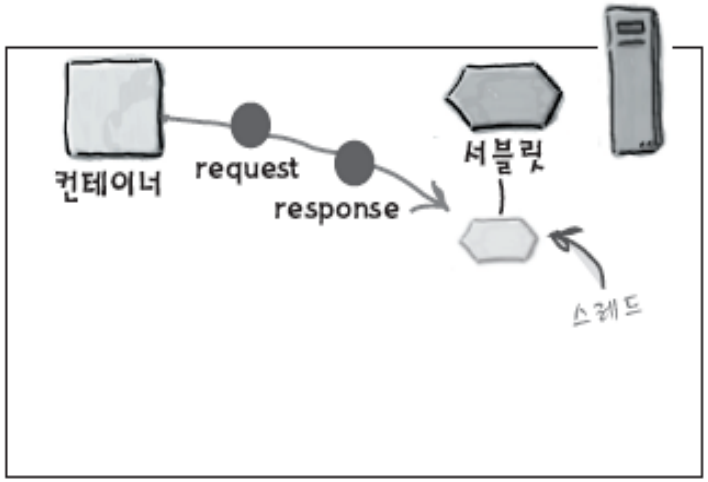
컨테이너는 들어온 요청이 서블릿이라는 것을 간파하곤 다음 두 객체를 생성합니다.

1) HttpServletResponse

2) HttpServletRequest

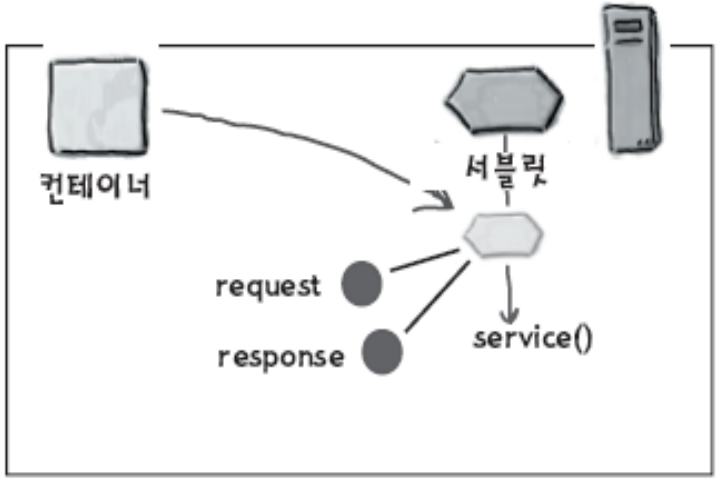
서블릿 컨테이너 >> HTTP 요청 처리

③



사용자가 날린 URL을 분석하여 어떤 서블릿에 대한 요청인지 알아냅니다(여기서 DD를 참조합니다, 이해가 안되어도 그냥 넘어가기 바람). 그 다음 해당 서블릿 스레드를 생성하여 Request/Response 객체를 인자로 넘깁니다.

④

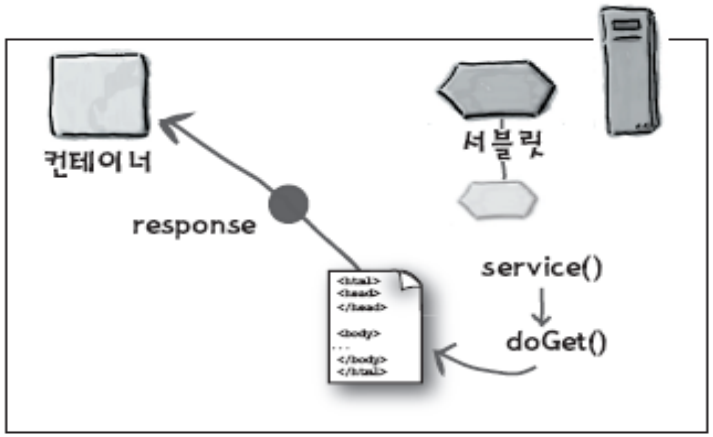


컨테이너는 서블릿 service() 메소드를 호출합니다. 요청에 지정한 방식(Method)에 따라 doGet()을 호출할지, doPost()를 호출할지 결정합니다.

여기서는 일단 HTTP GET 방식이라고 가정합시다.

서블릿 컨테이너 >> HTTP 요청 처리

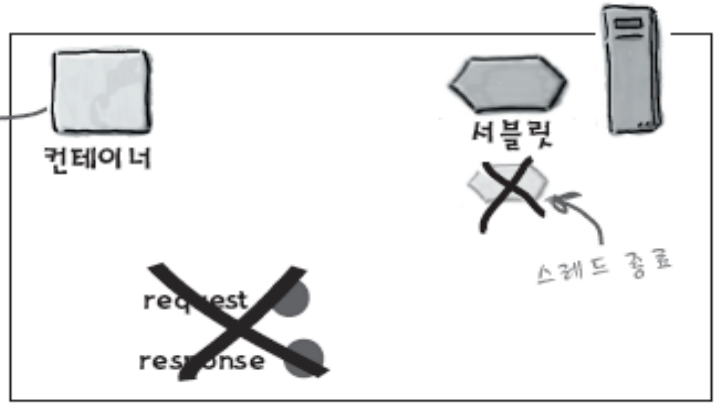
⑤



doGet() 메소드는 동적인 페이지를 생성한 다음, 이를 Response 객체에 실어 보냅니다. 보내고 난 후에도 컨테이너는 Response 객체에 대한 참조(레퍼런스)를 가지고 있다는 것 잊지마세요.

---

⑥



스레드 작업이 끝나면, 컨테이너는 Response 객체를 HTTP Response로 전환하여 클라이언트로 내려 보냅니다. 이제 마지막으로 Request와 Response 객체를 소멸시킵니다.

서블릿 >> 서블릿 작성

실제 프로젝트에서는 서블릿 중 99.99%은 doGet()과 doPost() 메소드만 재정의합니다.

서블릿에는 main() 메소드가 없다는 것을 잊지 마세요. 컨테이너가 서블릿의 생명주기 관련 메소드(예를 들면, doGet())를 호출합니다.

서블릿 중 99.9999%은 HttpServlet을 상속합니다.

여기가 바로 컨테이너가 생성한 Request와 Response 객체 참조를 넘겨받는 곳입니다.

서블릿이 넘겨준 Response 객체 안에는 PrintWriter가 들어 있죠. Response 객체에 HTML 코드를 작성하고 싶다면 이 PrintWriter를 사용하세요. PrintWriter말고도 다양한 출력 옵션이 있는데, HTML이 아닌 이미지 같은 것을 출력하고자 할 때 사용하면 됩니다.

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

public class Ch2Servlet extends HttpServlet {

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

    PrintWriter out = response.getWriter();
    java.util.Date today = new java.util.Date();
    out.println("<html> " +
                "<body>" +
                "<h1 style="text-align:center>" +
                 "HF\'s Chapter2 Servlet</h1>" +
                "<br>" + today +
                "</body>" +
                "</html>");
  }
}
```

서블릿 >> 서블릿 배포

■ 서블릿 이름 : URL명, 배포시 내부명, 실제 클래스명

① register/registerMe 서블릿에 대한 링크를 클릭해야지.

클라이언트가 아는 *URL* 이름

클라이언트는 사실 HTML 안에 존재하는 서블릿 이름만 알고 있으면 됩니다. 서버 상의 실제 어느 디렉토리에 어떤 파일명으로 존재하는지는 관심 없죠. 사실 URL 이름은 클라이언트를 위한 애칭이라고 표현해도 되겠네요.

② 이제부터 이 서블릿의 이름은 EnrollServlet으로 명명하노라.

배포자가 만든 *내부적인* 이름

배포자는 실제 애플리케이션 운영을 위하여 배포명이란 것을 만들어 이를 개발자에게 알려줍니다. 배포명도 사실 URL 이름과 같이 가공의 이름이라고 말할 수 있습니다. 이 이름은 URL 이름과 같을 필요가 없으며, 실제 서블릿 파일 위치 이름과 일치하지 않아도 됩니다.

③ classes / registration / SignUpServlet.class

실제 *파일*명

개발자가 만든 서블릿 클래스 안에는 클래스명과 패키지명이 들어 있습니다. 마찬가지로 서블릿 클래스 파일도 또한 파일 시스템 상의 경로와 파일명이 있습니다. 이는 패키지 디렉토리 구조에 따라 서버마다 다를 수 있습니다.

서블릿 >> 서블릿 배포

■ 배포 서술자 (DD, Deployment Descriptor)

- 서블릿 컨테이너에 서블릿 배포 시 사용하는 **XML** 문서
- **URL**과 서블릿 매핑 정보 포함
- 보안역할 설정, 오류 페이지 설정, 초기화 구성 및 관련 정보 설정 등

■ URL 매핑을 위한 항목

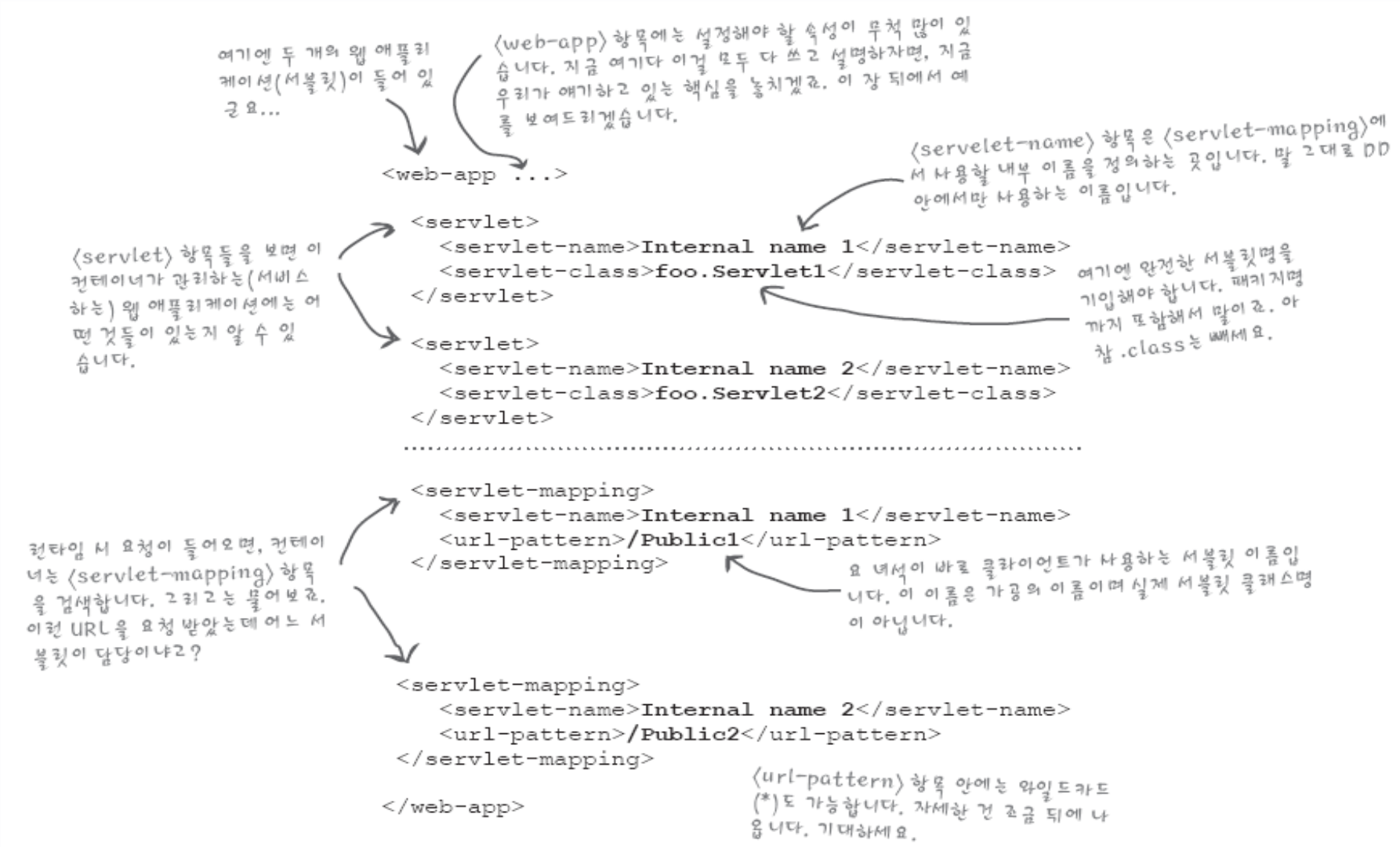
- **<servlet>**

  서블릿 내부명과 완전한 클래스명과의 매핑정보

- **<servlet-mapping>**

  서블릿 내부명과 **URL** 명과의 매핑정보

서블릿 >> 서블릿 배포자 샘플



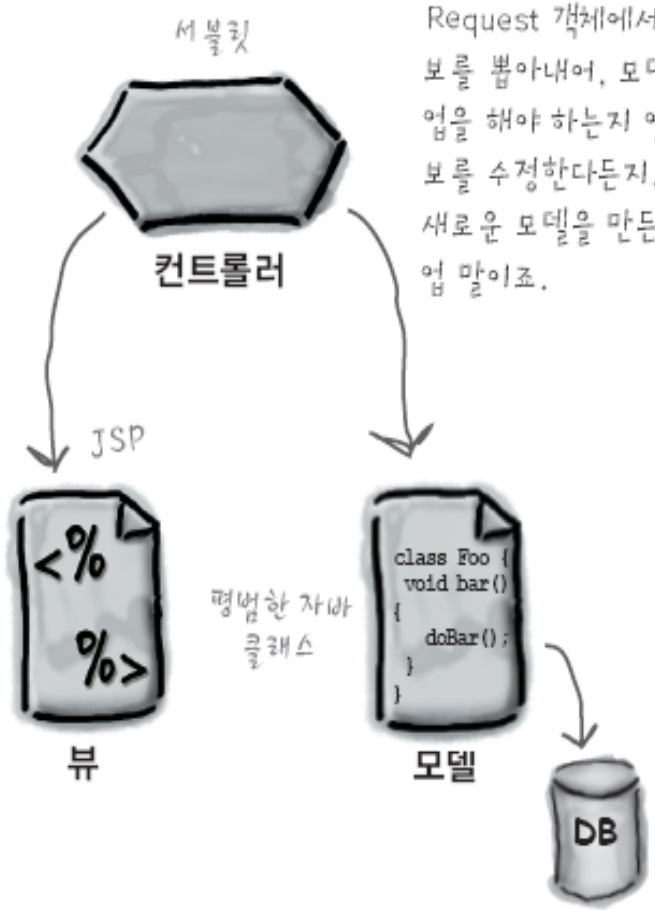
MVC 디자인 패턴 >> 개요

MVC 디자인 패턴

- 모델 (Model) – 컨트롤러 (Controller) – 뷰 (View) 로 구성
- 비즈니스 로직과 프리젠테이션 로직의 분리
- 비즈니스 로직의 재사용 : 프르젠테이션 로직의 변경에 영향을 받지 않음
- 서블릿 과 비즈니스 로직의 분리가 필요

MVC 디자인 패턴 >> 구성요소

서블릿, JSP 환경에서 MVC



컨트롤러(Controller)

Request 객체에서 사용자가 입력한 정보를 뽑아내어, 모델에 대하여 어떤 작업을 해야 하는지 알아냅니다. 모델 정보를 수정한다든지, 뷰(JSP)에게 넘겨줄 새로운 모델을 만든다든지 등과 같은 작업 말이죠.

뷰(View)

프리젠테이션에 대한 책임을 지죠. 뷰는 컨트롤러로부터 모델정보를 읽어옵니다(직, 간접적인 방법 둘 다 가능하며, 뷰가 찾을 수 있는 곳에 컨트롤러가 갖다 두는 방식을 많이 사용합니다). 뷰는 또한 사용자가 입력한 정보를 컨트롤러에게 넘겨주어야 합니다

모델(Model)

비즈니스 로직이 바로 여기에 들어갑니다. 모델 정보(state)를 읽어오거나(getter) 수정하는(setter) 로직도 여기 포함됩니다.

장바구니 예를 들면, 장바구니에 들어 있는 상품이 바로 모델입니다(여기엔 이를 어떻게 처리한다라는 내용도 포함되겠죠).

MVC 패턴에서 모델은 데이터베이스와 통신하는 유일한 곳입니다(물론 DB 통신만을 전담하는 객체를 따로 빼낼 수도 있지만... 이 패턴은 뒤에서 다루겠습니다).

J2EE 애플리케이션 서버 >> 구성

■ **J2EE 스펙 : 서블릿, JSP, EJB 스펙들을 포함**