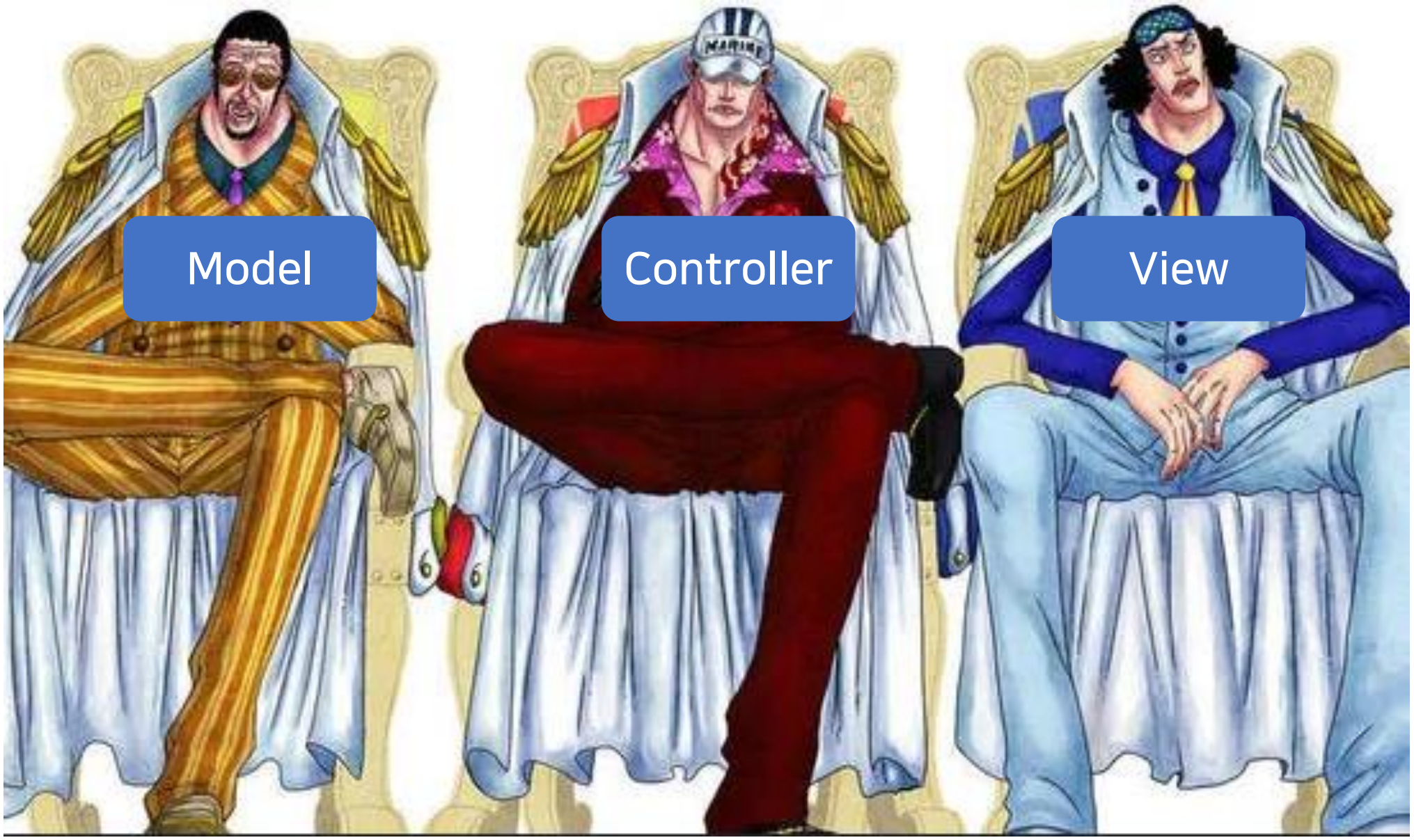
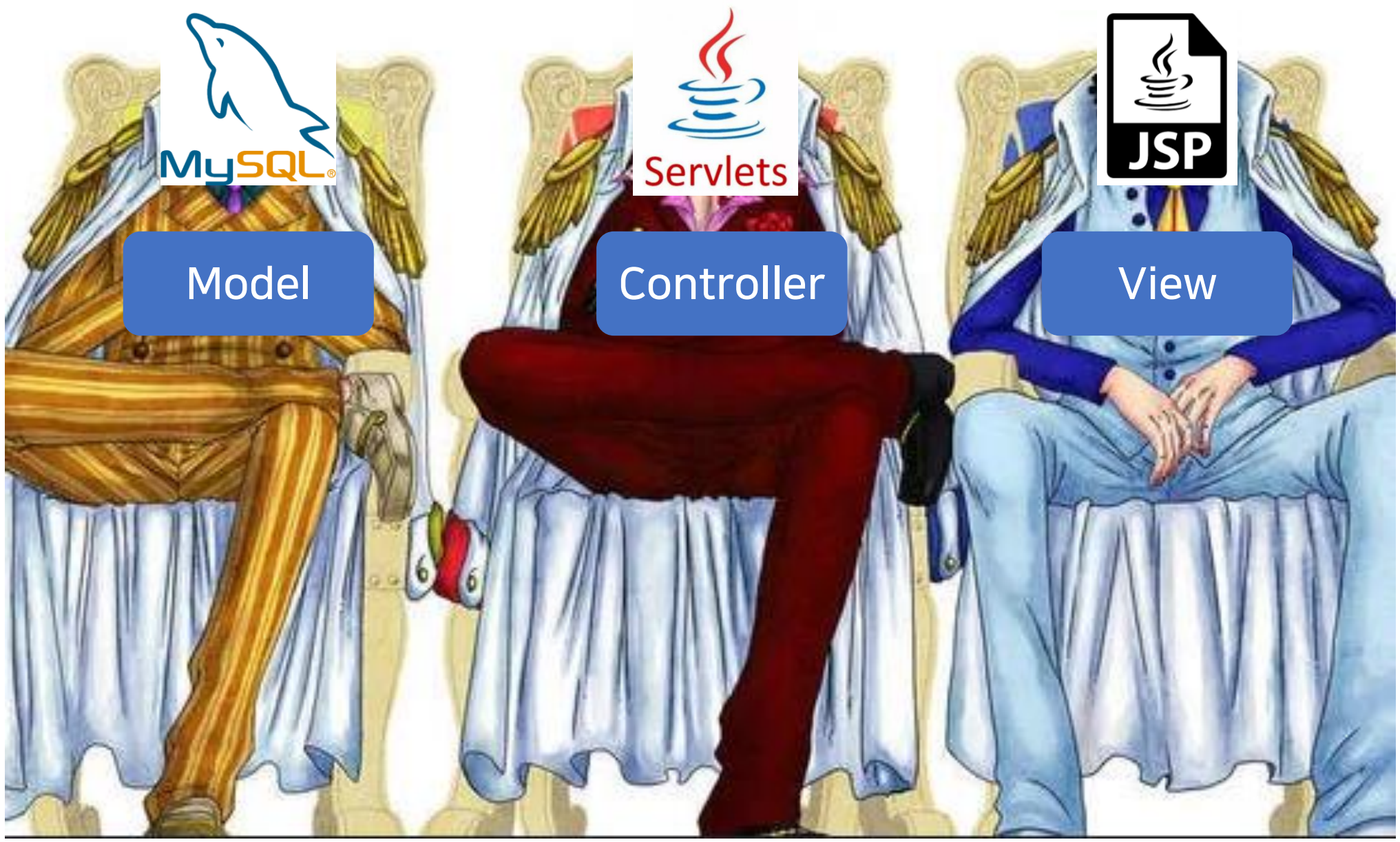
KB

It's Your Life

with

MVC 패턴


Model
Controller
View


MySQL
Servlets
JSP
Model
Controller
View

초록색은 요청(Request)이고,

파란색은 응답(Response)이야!

Client

Get 방식의 Request

/login?username=tetz

Tomcat Server

LoginServlet

@WebServlet(/login)

```
@Override  Tetz +1
protected void doGet(HttpServletRequest request, HttpServletRe
    String username = request.getParameter( s: "username");
    String password = request.getParameter( s: "password");
```

http://localhost:8080/login?username=tetz

로그인 성공!

Web Browser

서버에서 보내는 Response

```
<html>
<h1>로그인 성공!</h1>
</html>
```

Tomcat Server

Web Container

로그인 요청
tetz/1234

동적 처리 요청

Servlets

데이터 확인 요청

데이터 결과 응답

MySQL

Id	pw
tetz	1234
12	12

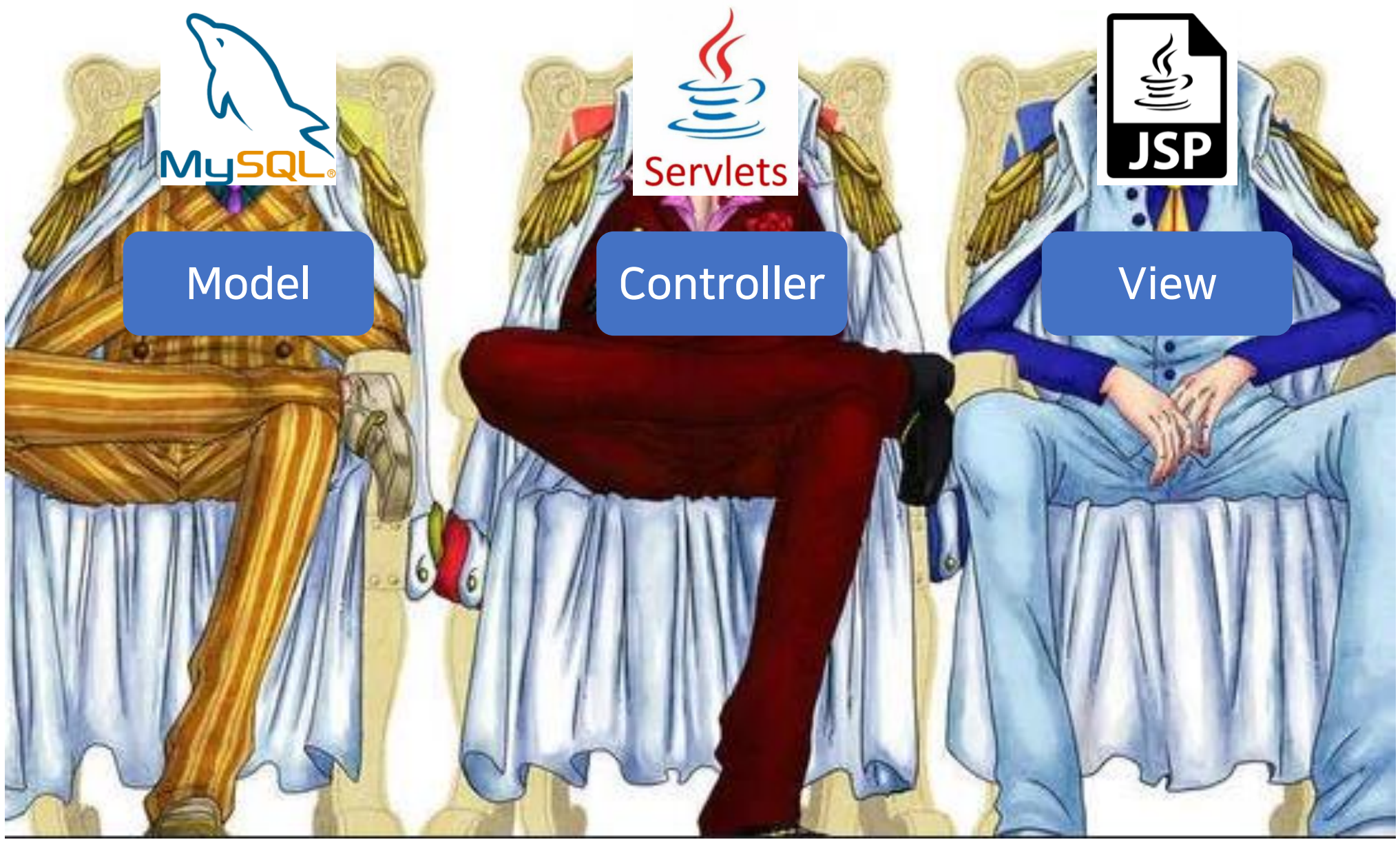
필요 JSP 요청
Scope 에 데이터 전달

JSP

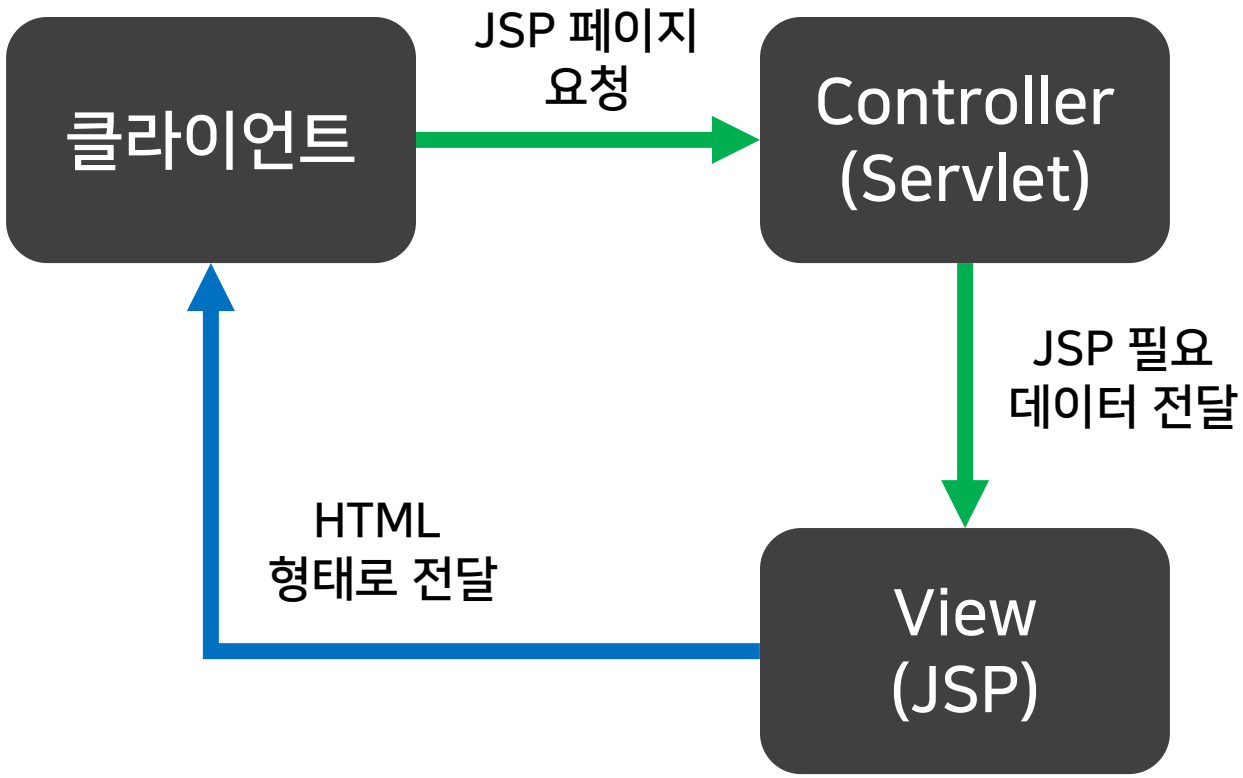
서블릿이 처리한
JSP 페이지 전달

동적인 처리
수행

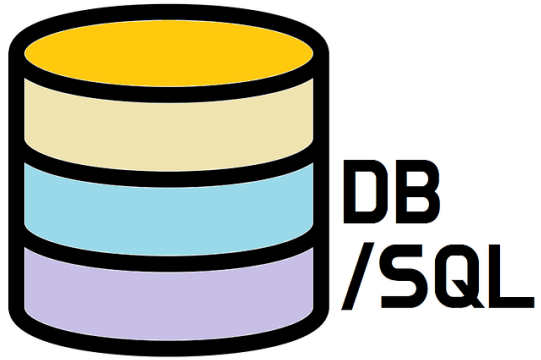
로그인 요청
응답


MySQL
Servlets
JSP
Model
Controller
View

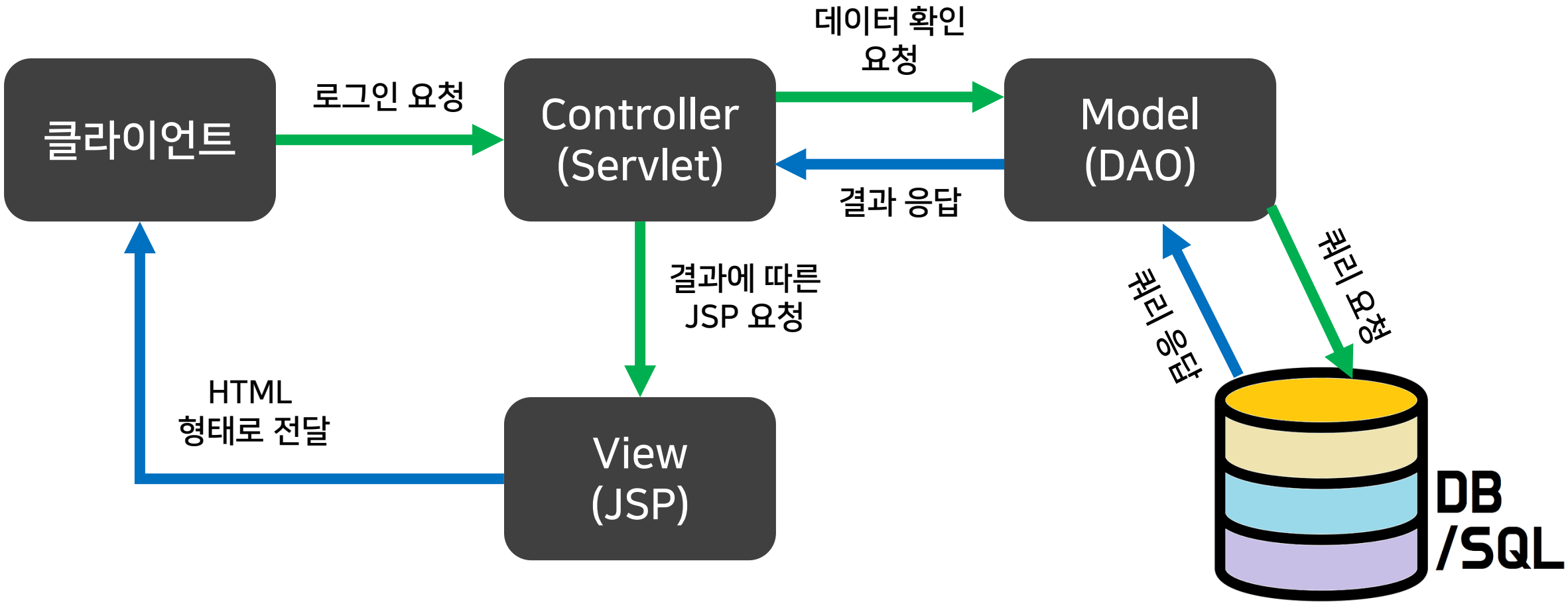
일반 JSP 페이지만 요청한 경우!



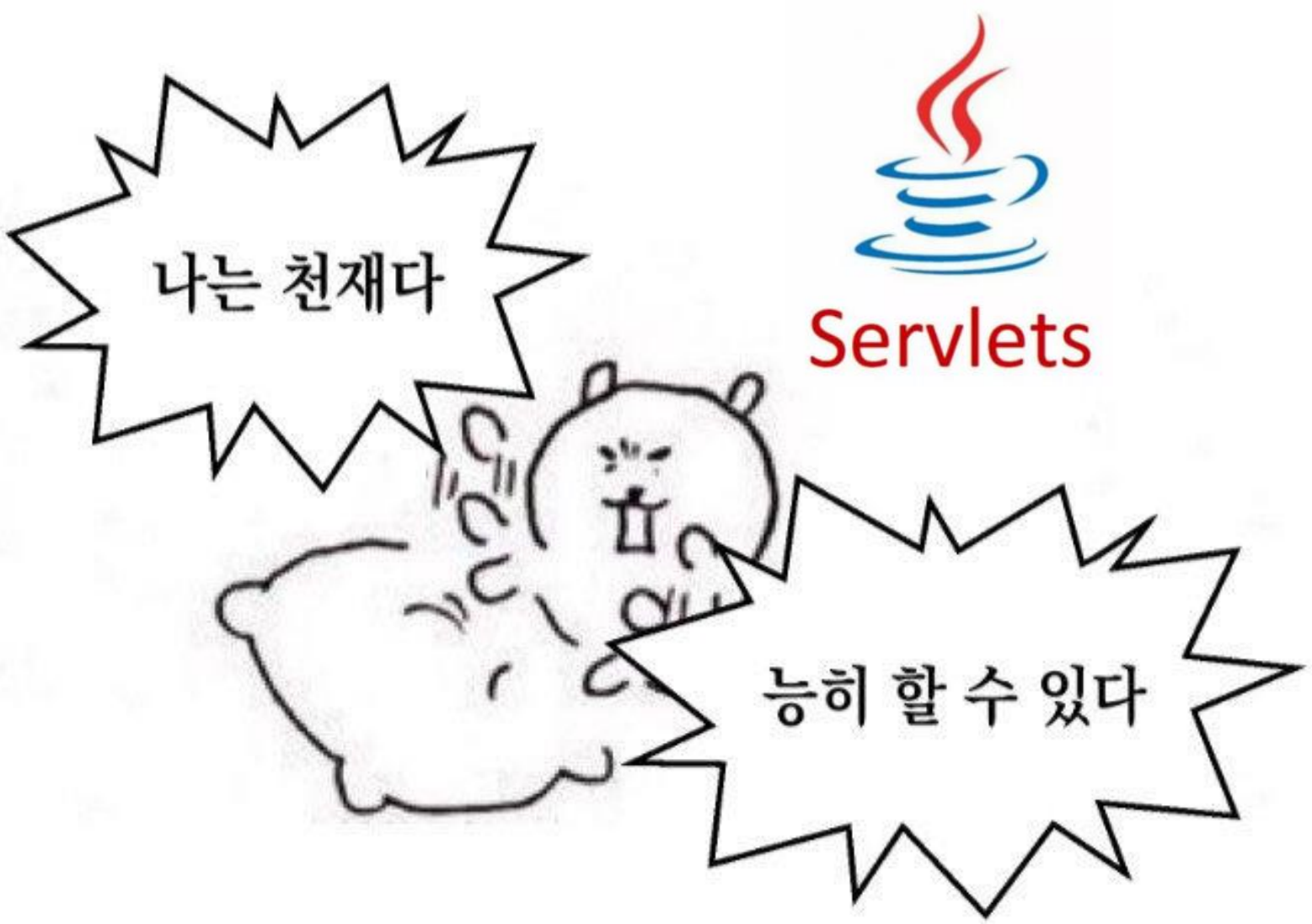
Model
(DAO)

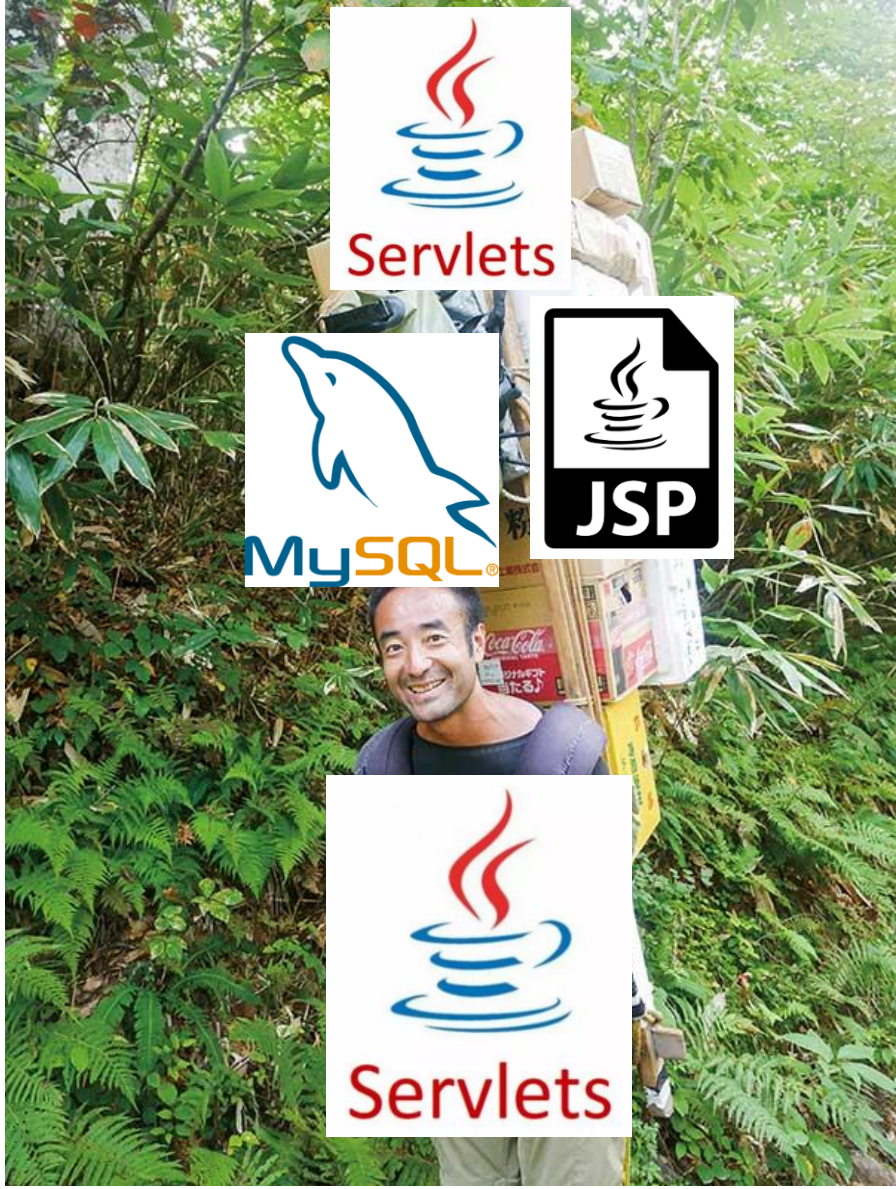


데이터 관련 요청이 같이 들어간 경우

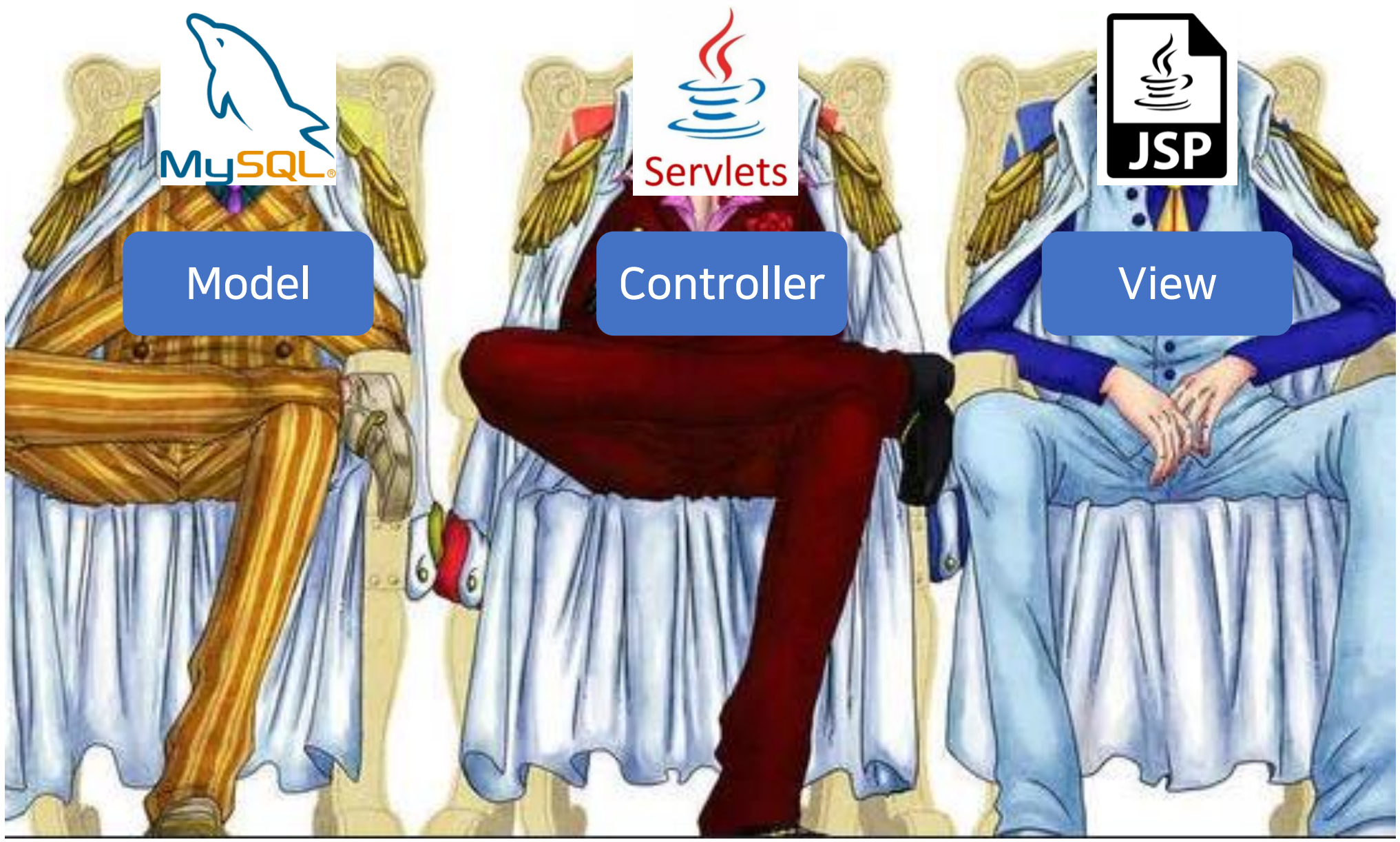


그런데 이거 왜 쓰나요!?


나는 천재다
Servlets
능히 할 수 있다


Servlets
MySQL
JSP
Servlets


유지 보수는?
확장성은?
관심사 분리는?
개발자


MySQL
Servlets
JSP
Model
Controller
View

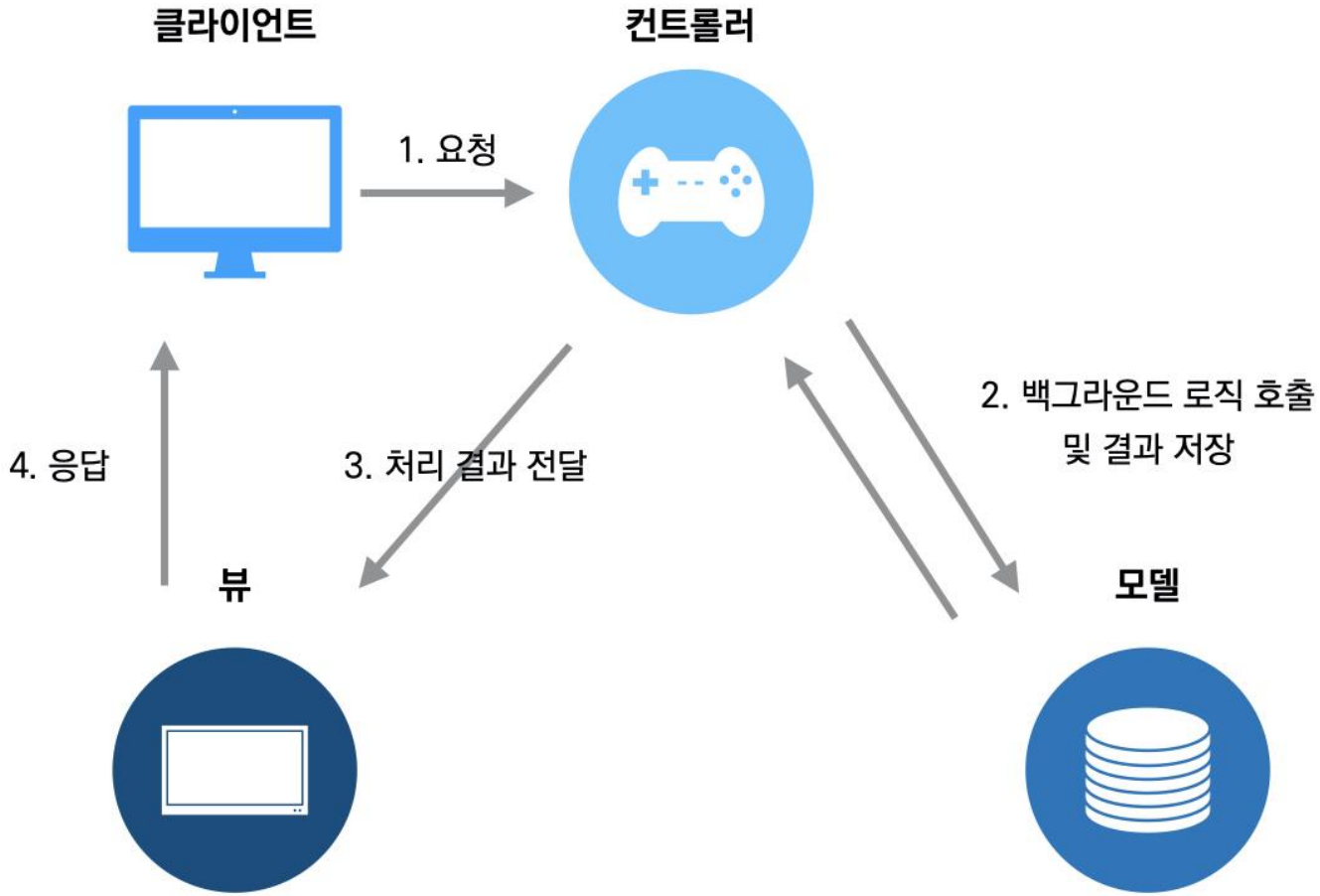
필요한 부분만 신경 쓰면서
개발하면 되는
관심사 분리가 가능!!

각자 개발할 부분만
개발하고 나중에 연결만
시키면 되기 때문에
병렬 개발이 가능합니다!

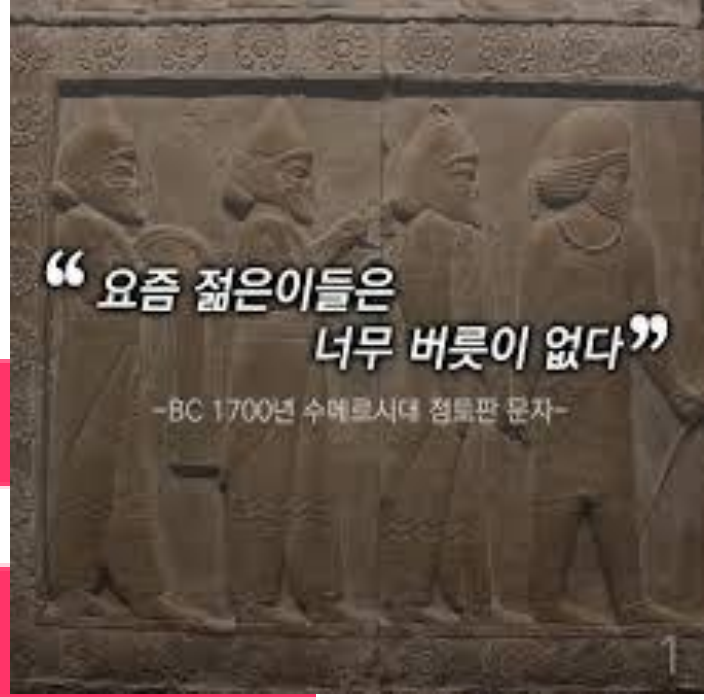
새로운 기능이 필요하면
해당 기능만 만들고
나중에 붙이면 됩니다!

기능별로 분리되어
개발이 되므로
전체를 테스트 할 필요 없이
해당 기능만 테스트를
하면 되기에
테스트에도 유리합니다!

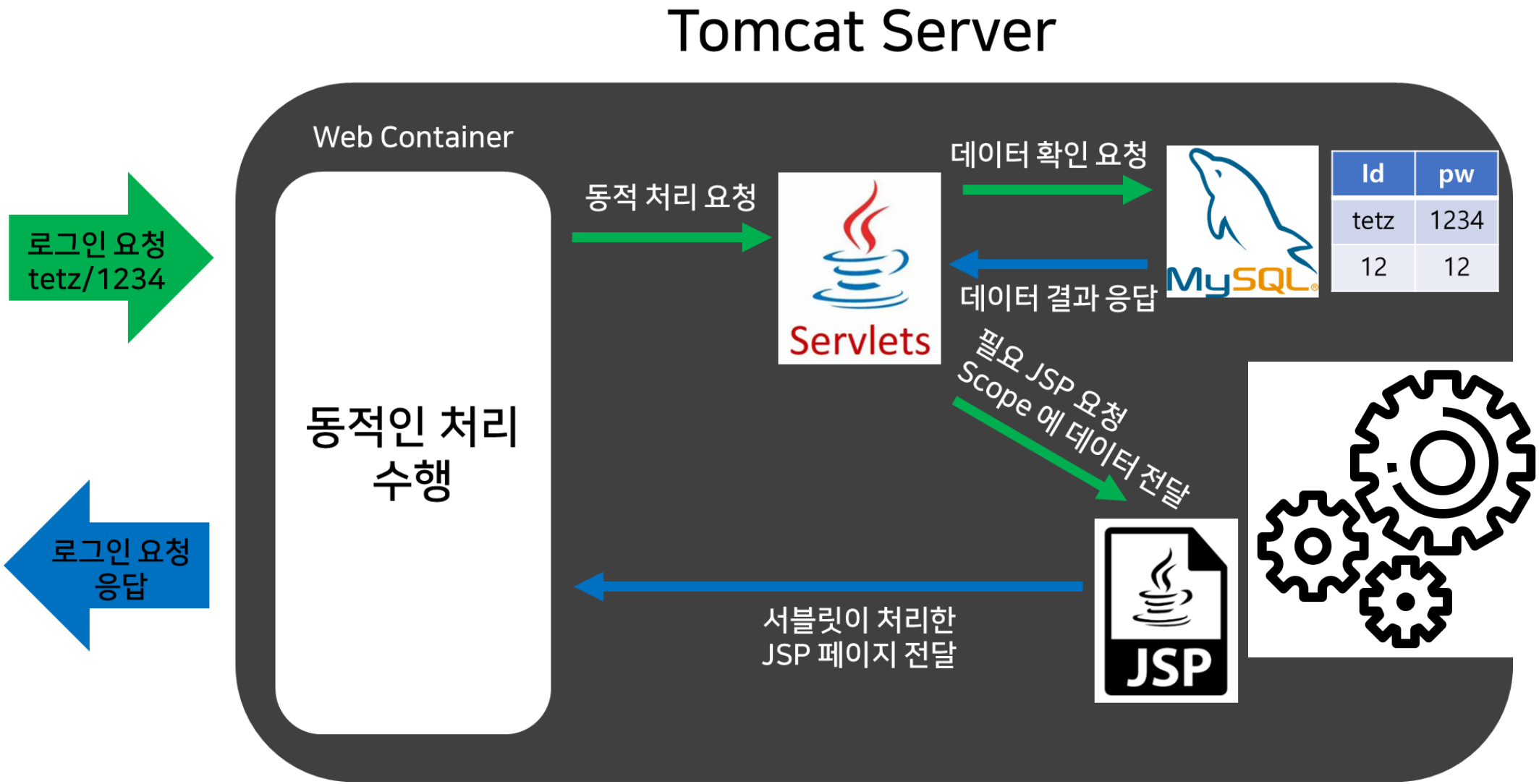
문제가 생겨도
전체에서 문제가 생기지 않아
문제가 생긴 파트만
고치면 되기 때문에
유지 보수에도 유리합니다!


클라이언트
컨트롤러
1. 요청
2. 백그라운드 로직 호출
및 결과 저장
3. 처리 결과 전달
4. 응답
뷰
모델

요즘것들

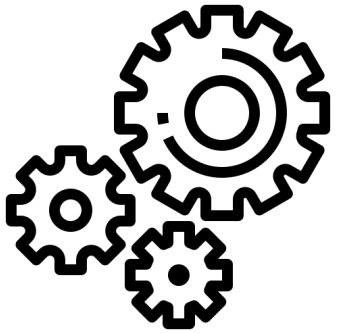


요 구조는 프론트엔드도 누가 처리를 하고 있죠!?




JSP
BACK-END


FRONT-END
희망나눔 성금
060-700-1212
참...좋다!



Client

React

Vue.js

ANGULAR

다양한 요청

요청에 대한 응답

Tomcat Server

spring

Servlets

MySQL


FRONT-END

위와 같은 프론트 프레임워크의 문제를 아시나요?

Servlets

spring

php

결국 HTML 이 남습니다

따라서 검색 엔진이 HTML 을
바탕으로 쉽게 검색할 수 있습니다

React

Vue.js

ANGULAR

HTML 은 <body> 만 제공

결국 모든 내용은
왼쪽의 JS 가 채웁니다!

검색 엔진 입장에서는
검색할 내용이 없습니다!

SERVER-SIDE RENDERING

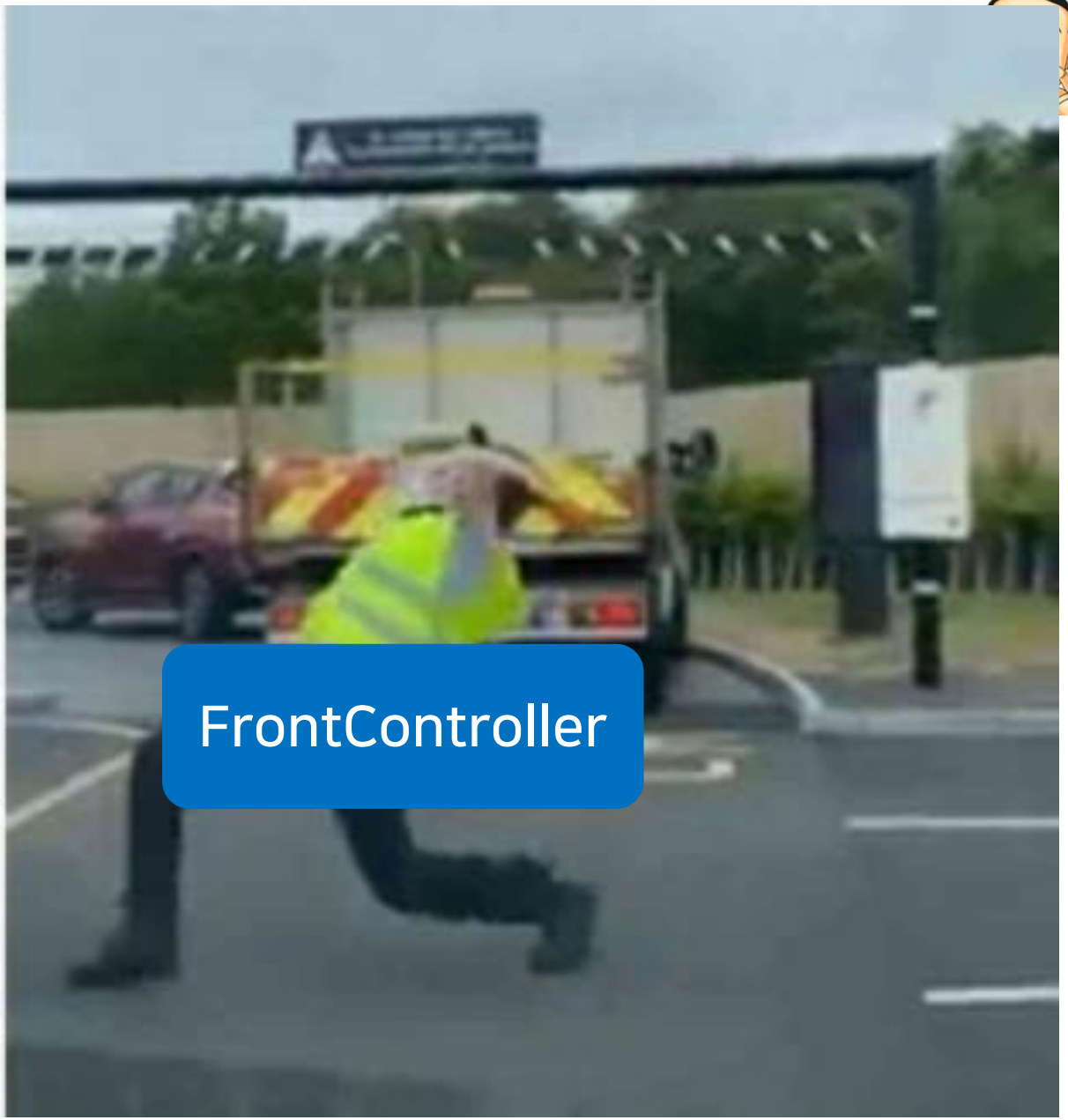
FrontController

view

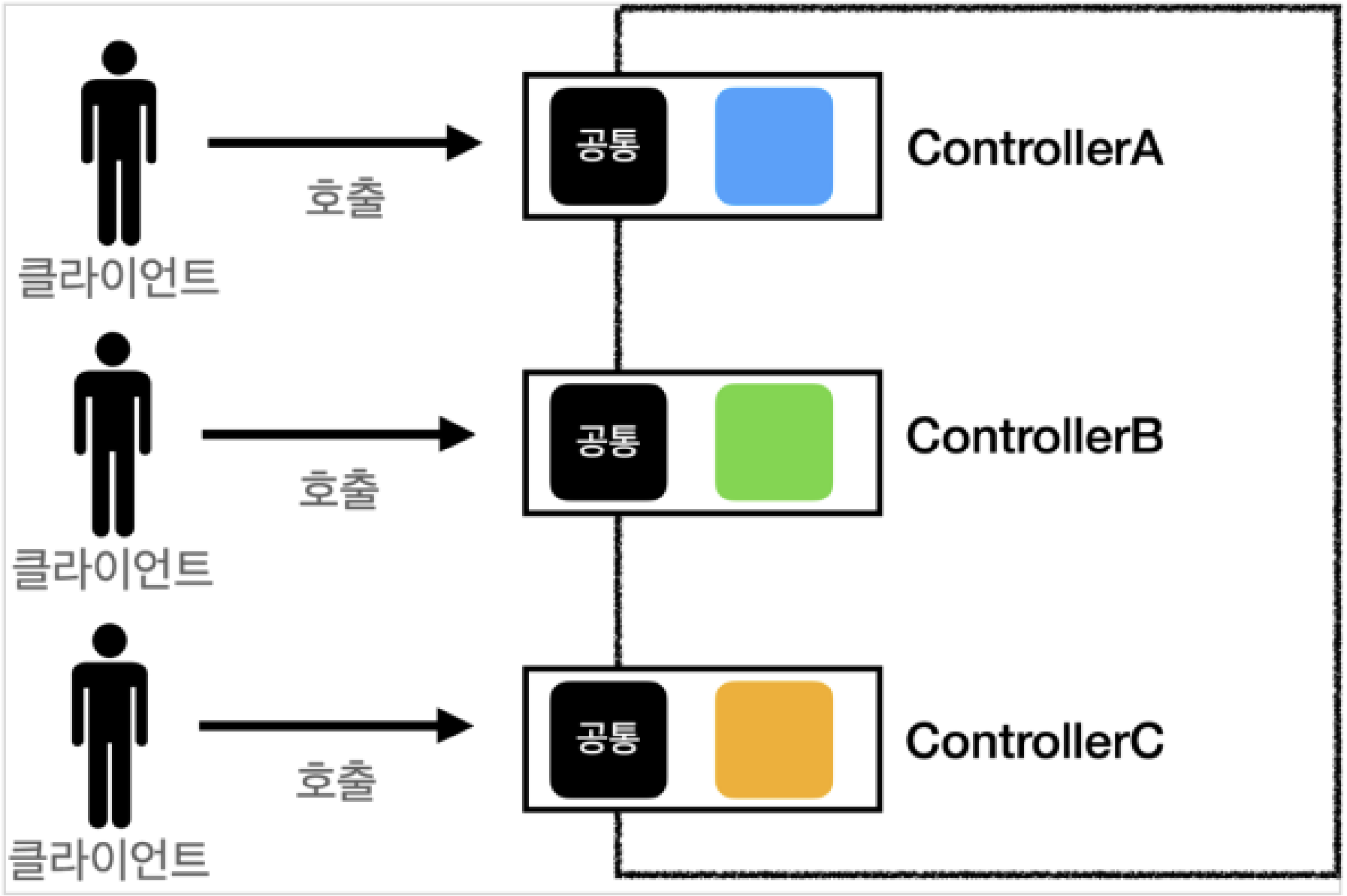
view

view


FrontController


FrontController


SNL
tvN

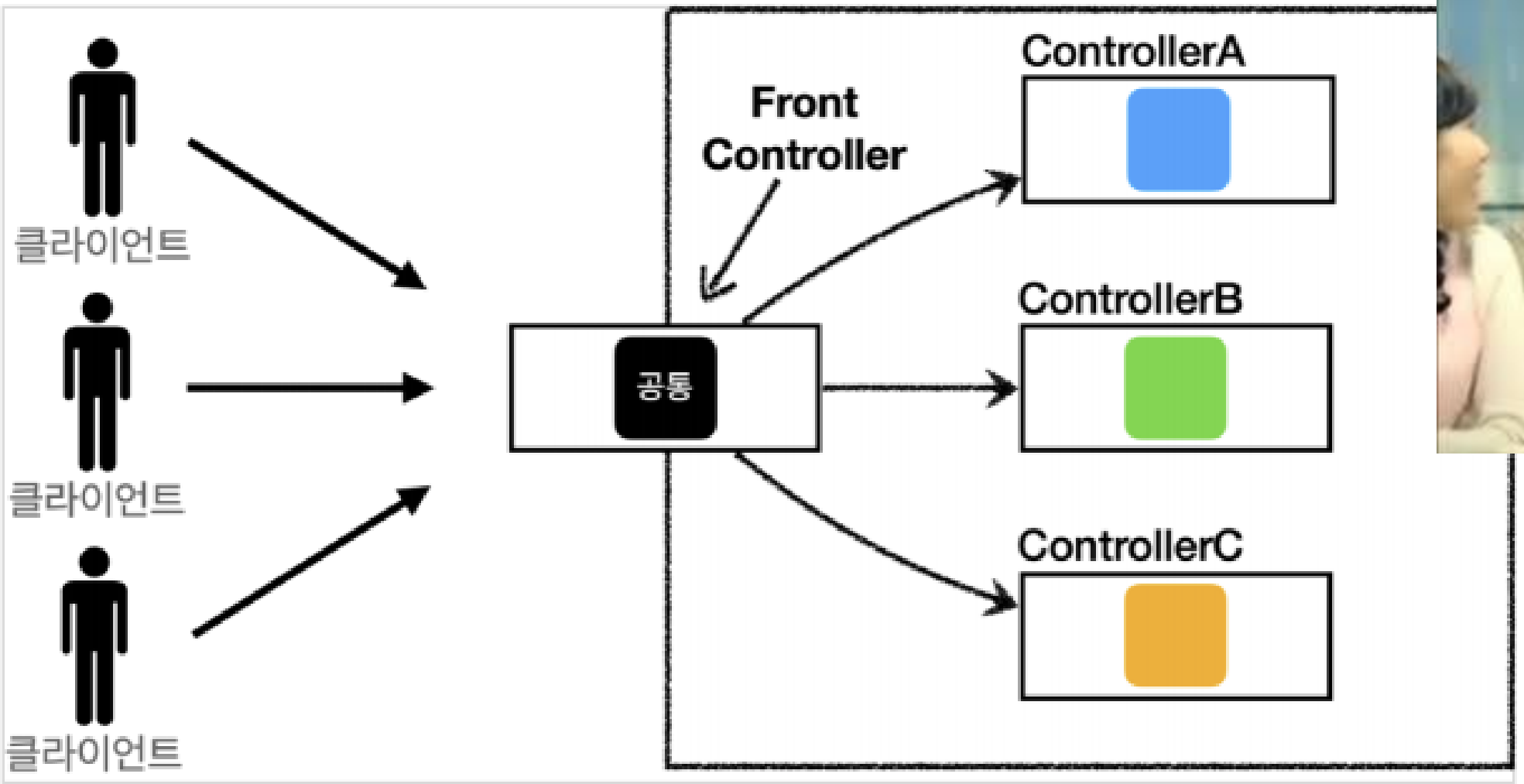

클라이언트
호출
공통
ControllerA
클라이언트
호출
공통
ControllerB
클라이언트
호출
공통
ControllerC

이렇게 되면 어떤 문제가 생길까요!?

공통 작업이 필요한 경우 누락 발생

문제가 생겼을 때, 어디서 문제가 생겼는지
알기 어려움

다양한 예외 발생 시, 각각의 컨트롤러에서
각기 예외를 처리해야함


클라이언트
클라이언트
클라이언트
Front
Controller
공통
ControllerA
ControllerB
ControllerC

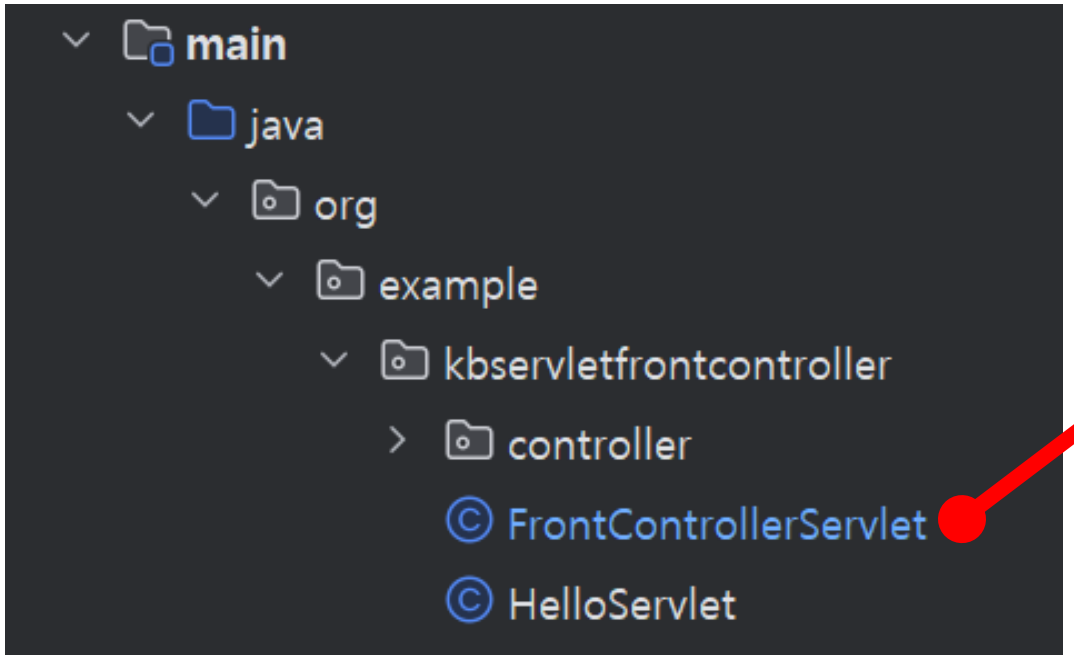

너무 이른가?


쉬운 버전으로
당장 하자

FrontController


내가 좀 성격이 급해
그닥 옳순하지 못해


MBC
NEWS TODAY
"기다리기 싫어"..신호등 파손



FrontControllerServlet 서블릿을 만듭니다!

모든 주소 값이 해당 서블릿을 지나가야 하므로
기본 주소인 "/" 를 매칭

```java
@WebServlet(name= "FrontControllerServlet", value="/") kdtTetz *
public class FrontControllerServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse r
        processRequest(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse
        processRequest(request, response);
    }
```

```java
@WebServlet(name= "FrontControllerServlet", value="/")  kdtTetz *
public class FrontControllerServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse r
        processRequest(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse
        processRequest(request, response);
    }
```

메서드에 상관 없이 모든 작업을 처리해 줄
processRequest 메서드에
request 와 response 를 전달

```java
@WebServlet(name= "FrontControllerServlet", value="/")  kdtTetz *
public class FrontControllerServlet extends HttpServlet {
    private Map<String, Controller> controllerMap = new HashMap<>();

    @Override  kdtTetz *
    public void init() throws ServletException {
        // 생성 시에 필요한 컨트롤러를 추가
    }
```

모든 웹 서비스는 주소에 매핑되는
컨트롤러에 의해 결정이 되므로
Map 컬레션을 사용해서 주소에 매핑되는
컨트롤러 정보를 저장합니다!

```java
@WebServlet(name= "FrontControllerServlet", value="/")  kdtTetz *
public class FrontControllerServlet extends HttpServlet {
    private Map<String, Controller> controllerMap = new HashMap<>();

    @Override  kdtTetz *
    public void init() throws ServletException {
        // 생성 시에 필요한 컨트롤러를 추가
    }
```

각각의 주소와 컨트롤러는 FrontController 가
생성되는 시점에 전부 알고 있어야 하므로
init() 메서드를 사용하여 미리 추가가 되어야 합니다

결국 주소 요청을 구분하는 것이 매우 중요하므로
전체 주소 값인 URI 값에서
기본 주소 값(http://localhost:8080) 을
빼서 주소 요청을 분리 합니다!!

```
private void processRequest(HttpServletRequest request, HttpServletRespo
    String requestURI = request.getRequestURI();
    String contextPath = request.getContextPath();
    String path = requestURI.substring(contextPath.length());
    System.out.println("path = " + path);

    Controller controller = controllerMap.get(path);
```

```java
private void processRequest(HttpServletRequest request, HttpServletRespo
    String requestURI = request.getRequestURI();
    String contextPath = request.getContextPath();
    String path = requestURI.substring(contextPath.length());
    System.out.println("path = " + path);

    Controller controller = controllerMap.get(path);
```

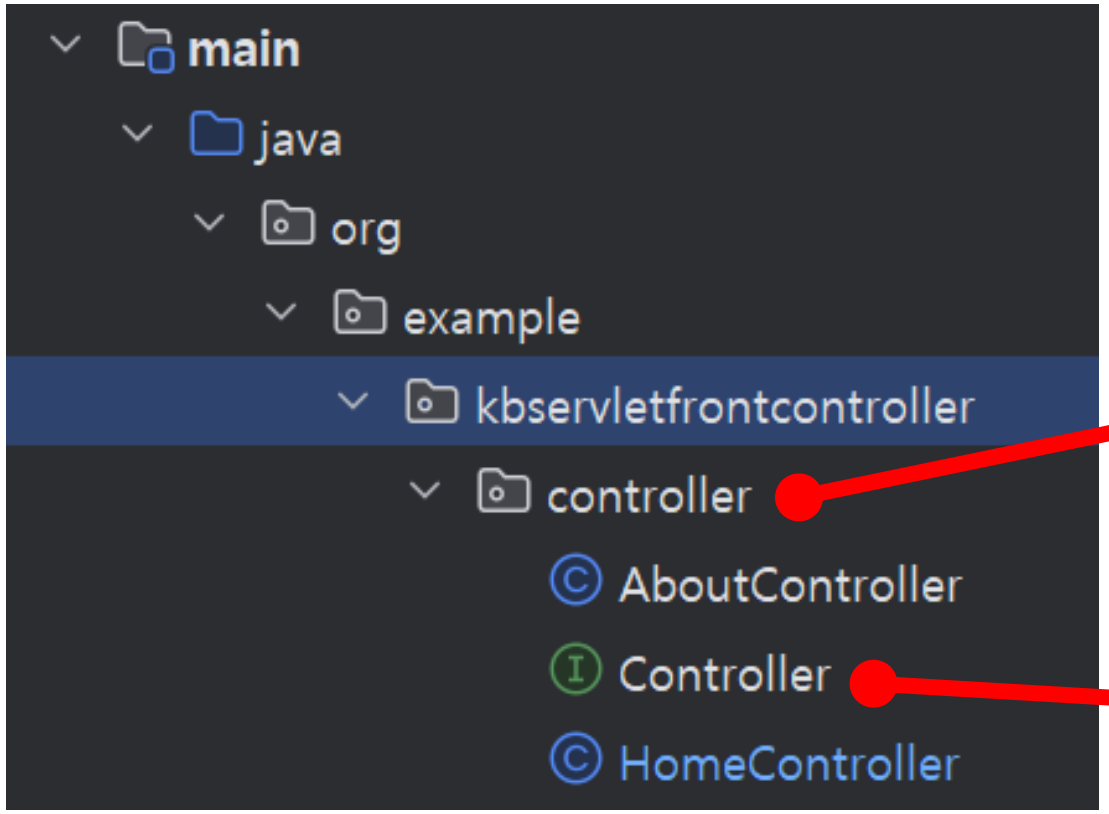
컨트롤러 맵에 저장 된 정보 중에서
지금 들어온 요청에 대한 컨트롤러가 존재하는지
찾아서 리턴하기!

컨트롤러를 찾았으면 해당 컨트롤러에 있는
jsp 파일명을 받아와서 파일을 포워딩 합니다!

```
if (controller != null) {
    String viewName = controller.getView(request, response);
    String targetView = "/WEB-INF/views/" + viewName + ".jsp";
    request.getRequestDispatcher(targetView).forward(request, response);
} else {
    request.getRequestDispatcher( s: "/WEB-INF/views/" + "404" + ".jsp").forward(re
}
```

컨트롤러를 못 찾았다면!?
등록된 적이 없는 서비스 이므로 404.jsp 페이지로 이동 시킵니다!

서비스 컨트롤러 구현



컨트롤러를 모아놓기 위한
controller 패키지를 생성

FrontController 가 처리하는 방식을
서비스 컨트롤러는 지켜야 하기 때문에
강제성을 부여하기 위해 Controller
인터페이스를 사용하여 메서드를 강제합니다!

```java
public interface Controller { 8 usages  2 implementations  kdtTetz
    String getView(HttpServletRequest request, HttpServletResponse response)
}
```

우리는 간단한 서비스 로직을 수행하고
View 역할을 하는 jsp 파일명을 알려 주기만 하면 되므로
getView 라고 하는 간단한 인터페이스를 제공

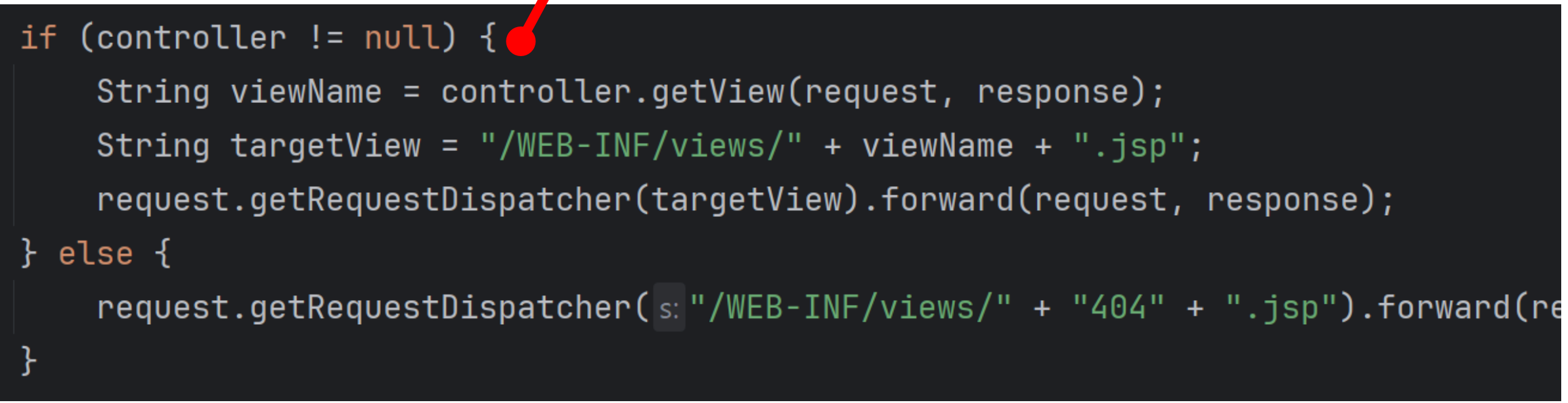
```java
public class HomeController implements Controller { 3 usages  kdtTetz *
    public String getView(HttpServletRequest request, HttpServletResponse response)
        request.setAttribute( s: "imgSrc",  o: "https://www.timeforum.co.kr/files/attach
        return "index";
    }
}
```

HomeController 는 기본 주소 요청이 오면
기본 페이지를 보여주는 기능을 할 예정이므로
이미지 주소를 위한 값을 request 스코프로 전달하고
index.jsp 파일을 서빙할 수 있도록 “index” 문자열 리턴

```java
@Override  kdtTetz *
public void init() throws ServletException {
    // 컨트롤러 맵의 "/" 주소에 방금 만든 HomeController 서비스 추가
    controllerMap.put("/", new HomeController());
}
```

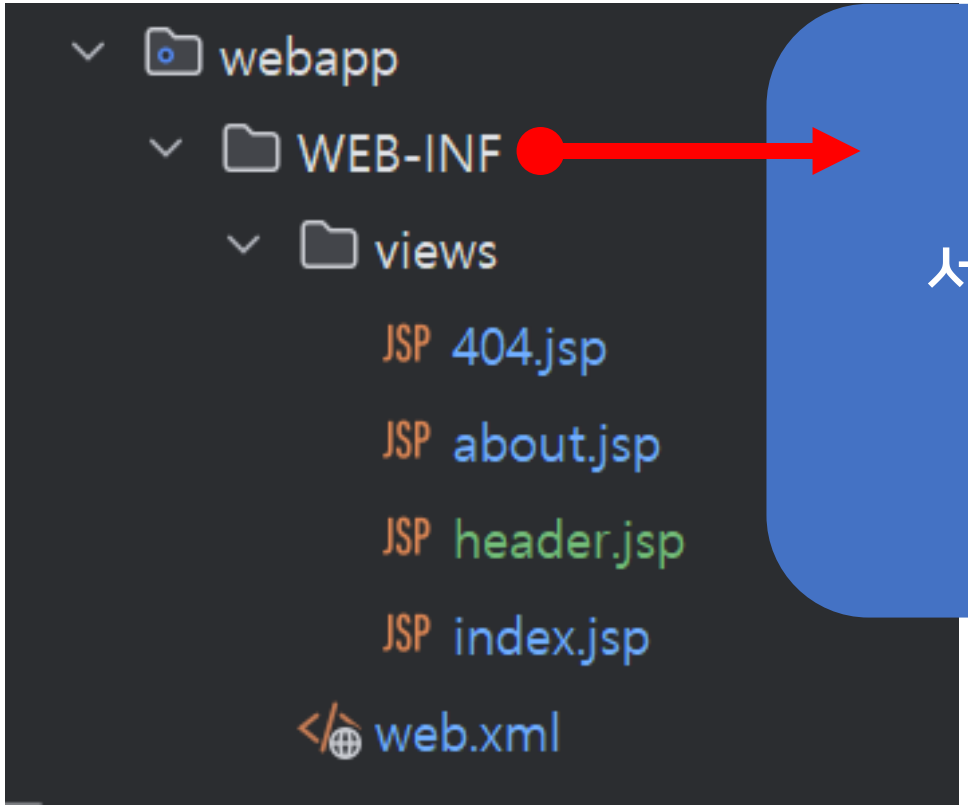
모든 서비스를 관리하는 controllerMap 에
“/” 주소에 대응하는 Homecontroller 클래스를 매핑!

이제는 컨트롤러를 찾을 수 있으므로
/ 요청이 들어오면 /WEB-INF/views 폴더의 index.jsp 파일이 서빙 됩니다!



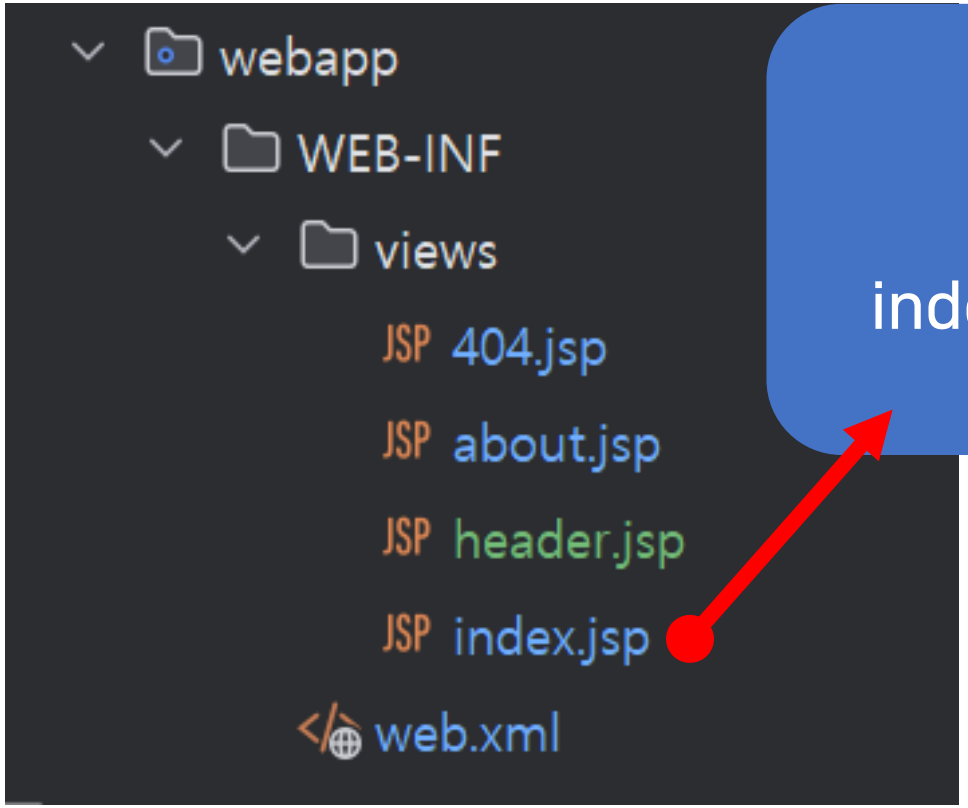
```
if (controller != null) {
    String viewName = controller.getView(request, response);
    String targetView = "/WEB-INF/views/" + viewName + ".jsp";
    request.getRequestDispatcher(targetView).forward(request, response);
} else {
    request.getRequestDispatcher( s: "/WEB-INF/views/" + "404" + ".jsp").forward(re
}
```

View 페이지 구현



WEB-INF 폴더는 브라우저가 파일에 직접 접근을
허용하지 않기 때문에 중요한 파일 혹은
서블릿에 의해 서빙되는 파일을 저장하는 폴더 입니다!

저희는 MVC 의 View 를 운영할 예정이므로
views 폴더를 만들고 필요한 jsp 파일을 만듭니다!



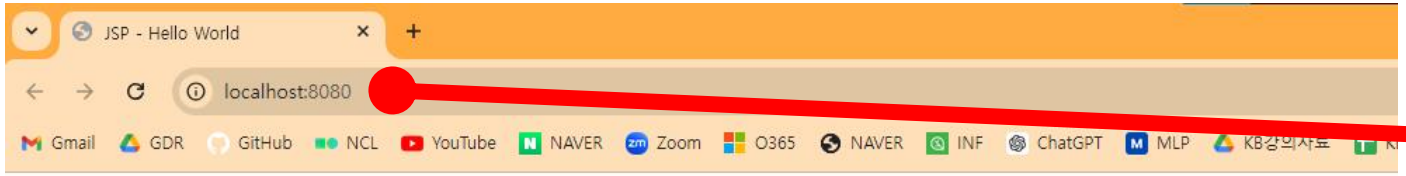
/ → 기본 주소 요청은
HomeController 가 대응을 하며
index.jsp 파일이 서빙되므로 index.jsp 파일을 만듭니다!

```
<header>
    <a href="/">HOME</a>
    <a href="about">ABOUT</a>
    <a href="login">LOGIN</a>
</header>
```

편리한 이동을 위해서
header.jsp 파일 만들기

```jsp
<%@ page contentType="text/html; charset=UTF-8" p
<!DOCTYPE html>
<html>
<head>
    <title>JSP - Hello World</title>
</head>
<body>
<%@ include file="header.jsp" %>
    <h1>어서와. 프론트 컨트롤러는 처음이지!?</h1>
    <img src="${imgSrc}" alt="어서와" />
<br/>
</body>
</html>
```

header.jsp 파일을 포함하고
HomeController 에서
리퀘스트 스코프에 전달한
imgSrc 를 이미지 주소로 사용하는
index.jsp 페이지

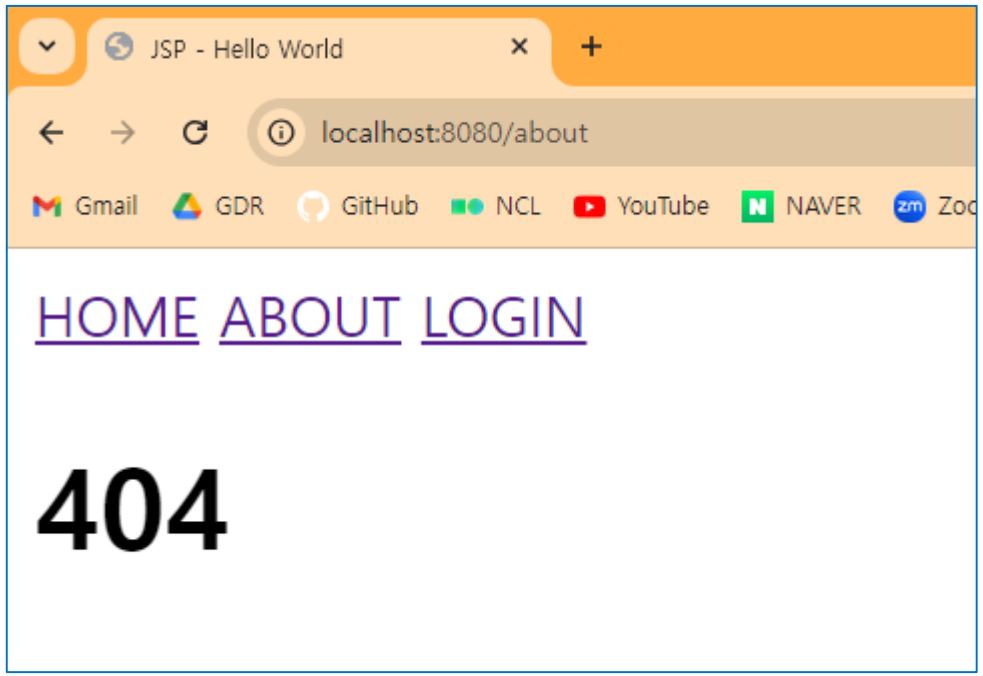


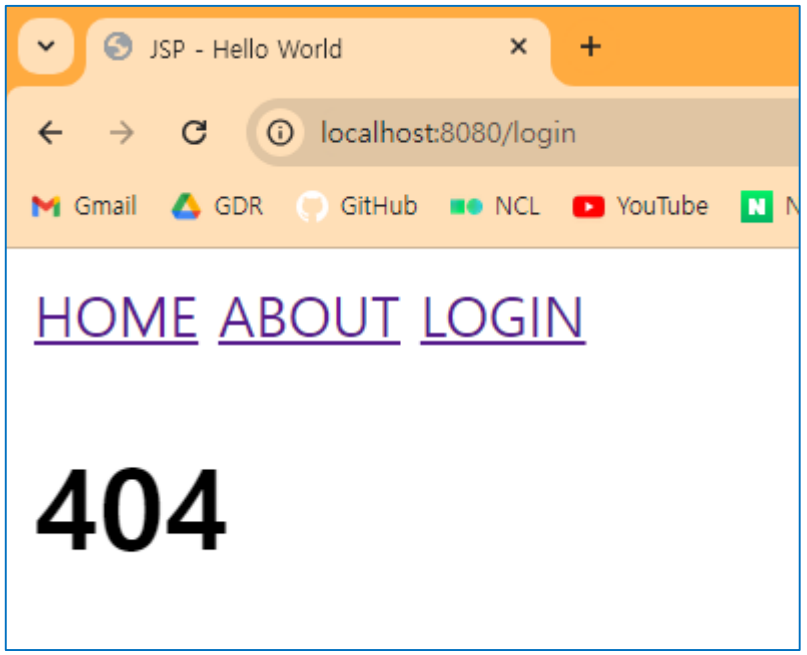
기본 주소(/) 를 요청하면
HomeController 에 의해
index.jsp 파일이 서빙 됩니다!

HOME ABOUT LOGIN

어서와. 프론트 컨트롤러는 처음이지!?

/about, /login 주소는 아직 컨트롤러 연결이 안되어있으므로
404 페이지로 연결 됩니다!





/about, /login 은 컨트롤러맵에 존재하지 않으므로
controller 가 null 이 리턴 됩니다!

```
if (controller != null) {
    String viewName = controller.getView(request, response);
    String targetView = "/WEB-INF/views/" + viewName + ".jsp";
    request.getRequestDispatcher(targetView).forward(request, response);
} else {
    request.getRequestDispatcher( s: "/WEB-INF/views/" + "404" + ".jsp").forward(re
}
```

따라서 404.jsp 페이지가 전달 됩니다!

실습, AboutController 추가하기!

- controller 패키지에 AboutContoller 클래스를 추가해 주세요
- 해당 클래스는 /about 주소에 대응이 되며, 해당 요청이 들어오면 about.jsp 파일을 View 로써 전달 합니다
- 또한, request 스코프를 사용하여 msg 속성에는 “컨트롤러가 전달한 메시지 입니다” 를, imgSrc 속성에는 각자 원하는 이미지의 주소를 전달 합니다
- 아래와 같이 주어진 about.jsp 파일의 코드가 전달 되었을 때 다다음 장의 결과가 출력이 되면 됩니다.

```jsp
<%@ page contentType="text/html; charset=UTF-8" pageEnco
<!DOCTYPE html>
<html>
<head>
    <title>JSP - Hello World</title>
</head>
<body>
    <%@ include file="header.jsp"%>
    <h1>컨트롤러에 의해 배달 된 about.jsp 파일입니다!</h1>

    <h2>${msg}</h2>
    <img src="${imgSrc}" alt="어서와" />
</body>
</html>
```

JSP - Hello World

localhost:8080/about

Gmail GDR GitHub NCL YouTube NAVER Zoom O365 NAVER INF ChatGPT M

HOME ABOUT LOGIN

컨트롤러에 의해 배달 된 about.jsp 파일입니다!

컨트롤러가 전달한 메세지 입니다!

DispatcherServlet