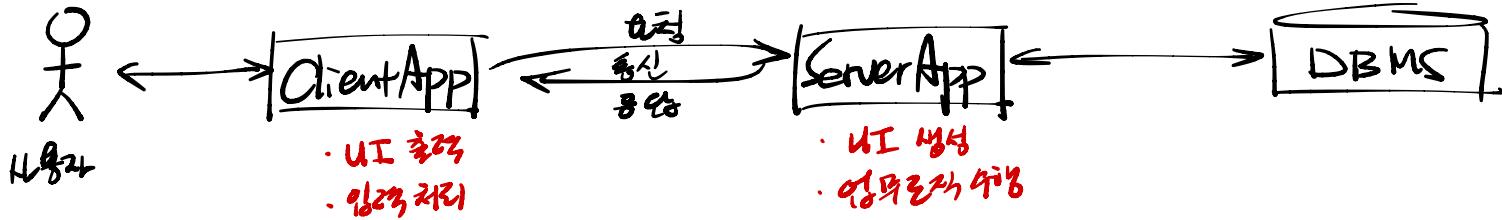
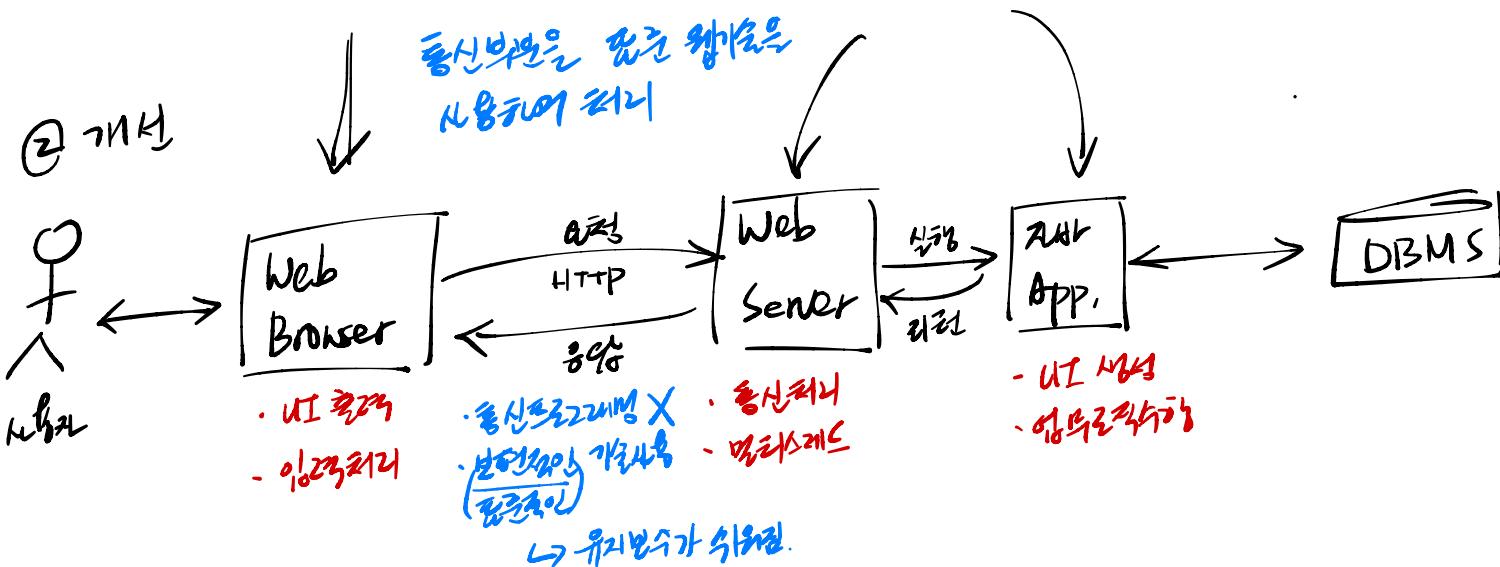


## 49. 웹애플리케이션 구조를 살펴보자

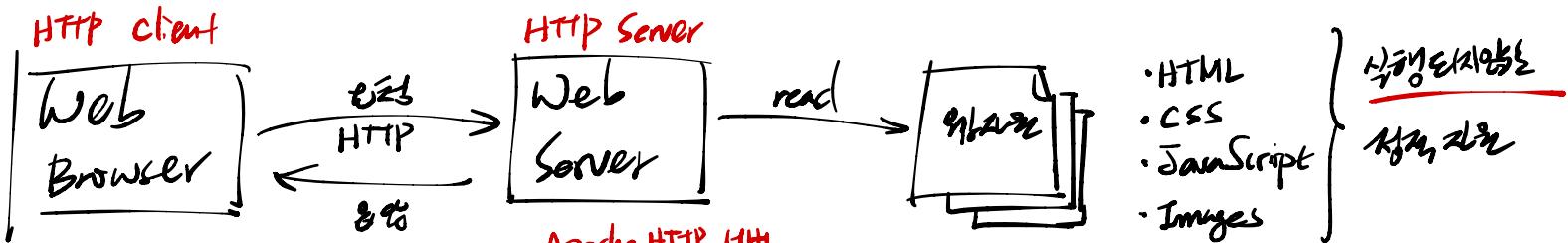
① 단계 → 전용 프로토콜을 사용하여 클라이언트/서버 통신



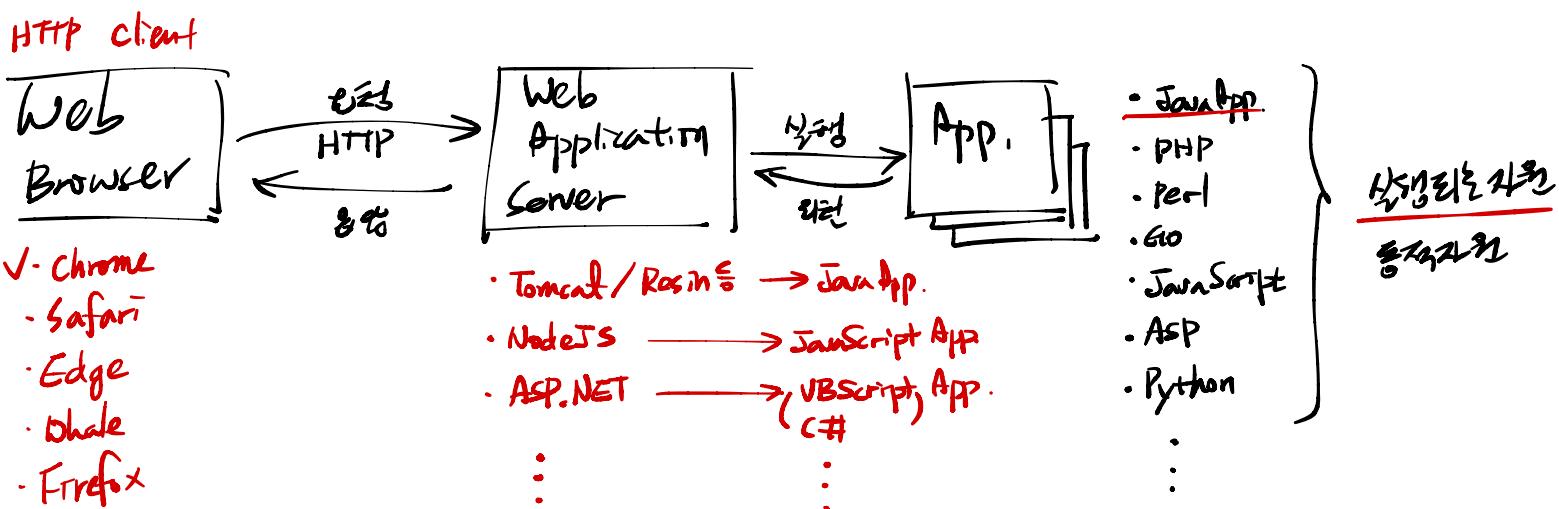
② 개발



\* 웹 사이트의 구조와 작동 원리



- HTML
  - CSS
  - JavaScript
  - Images
- 수정되었습니다  
기록을 위한



- JavaApp.
  - PHP
  - Perl
  - Go
  - JavaScript
  - ASP
  - Python
- 수정되었습니다  
기록을 위한

- ✓ Chrome
- Safari
- Edge
- Opera
- Firefox

\* Web App. or CGI 프로그램

① 험장기 유형 애플리케이션 (제시판, 게시판, 쇼핑몰 등)

HTTP Client

- curl
- wget
- chrome
- Firefox

Web Browser

HTTP Server

Web Server

APP

Common Gateway Interface (CGI)  
↳ 다른 APP. 활용 가능하게 하는 기술

HTTP Client

Web Browser

HTTP Server

Web Server

APP

실행 ← C / C++

↑ CGI 프로그램 (CGI 기능이 포함된 확장자)

② 풍선기 유형 애플리케이션 (쇼핑몰 등) → UI 브라우저 UI → 다양한 HTML 표시로 → 웹사이트 구현이 용이

Web Browser

HTTP Server

CGI

스크립트  
언어

· pl (PERL)  
· php (PHP)  
· asp (ASP)

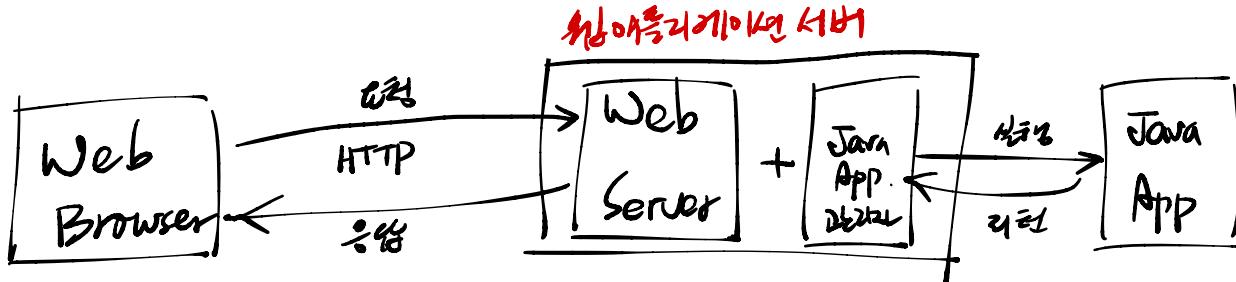
③ 흐름기 유형 애플리케이션 (인수/값 전달, 헤더 처리 등 입력 세팅)

↳ 흐름기 유형 애플리케이션 → OOP 프로그래밍 기법 도입  
구현하기 쉽다

{ Java 언어  
· .NET  
= }

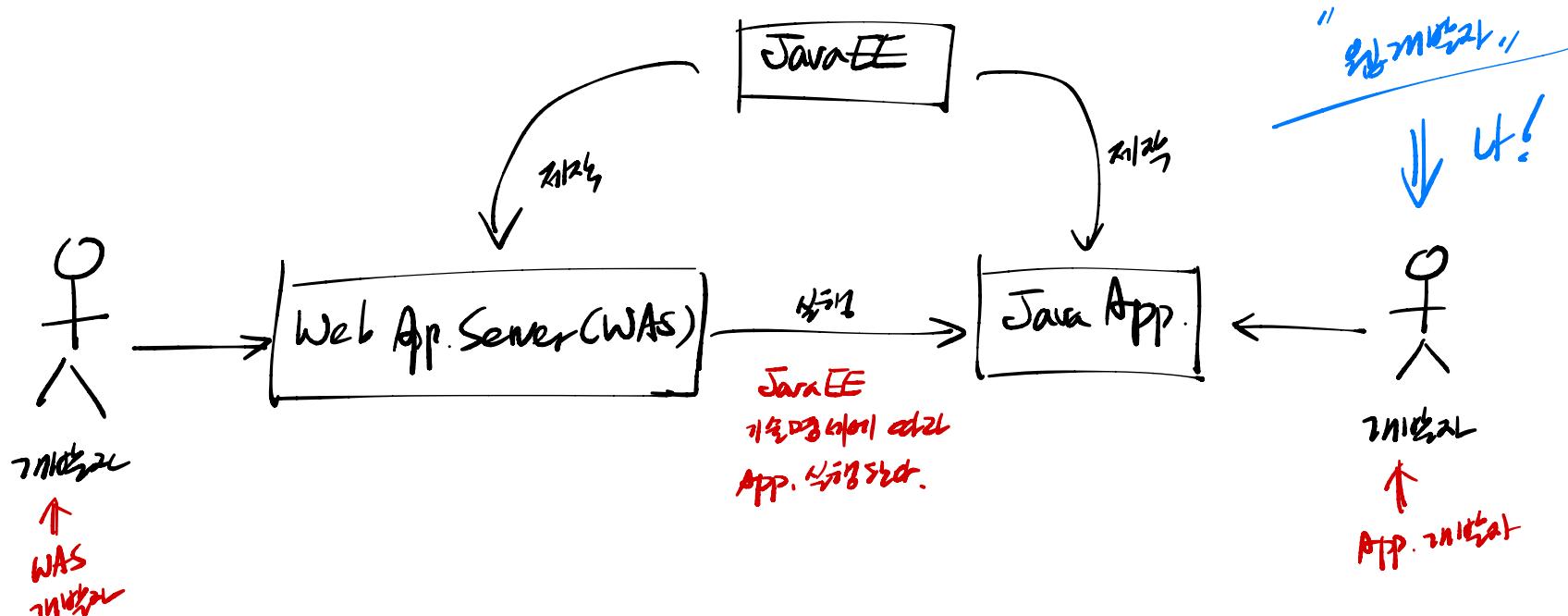
↑ C/C++ 보다 모듈화  
단위별 파일,  
스택별로 즉시  
불용이 연결  
↳ 간접적 X

## \* Java Web Application Architecture



- Tomcat / Resin / Jetty : ~~轻量级~~
  - Weblogic
  - Websphere
  - JBoss
  - BEA
  - IBM
  - JEUS
  - :

## \* Java Web Application ut JavaEE چیぞ咯!



\* Java EE (Enterprise Edition) 기술 영역

↳ 기업용 App. 개발 기술 모음

↳ 데이터베이스, 공유자원 관리, 분산처리, 접근제한 등

애플리케이션

Web



Servlet/JSP

JSTL > EL

ESB

Web Service

Authentication  
&  
Deployment

- 웹 애플리케이션 개발 기술

← 핵심 기술!

⇒ REST API

- 분산 처리 기술

- 서비스 기반으로 서비스 개발

- 인증 및 배포 기술

## \* Java EE 7/8/10/11 버전의 특징

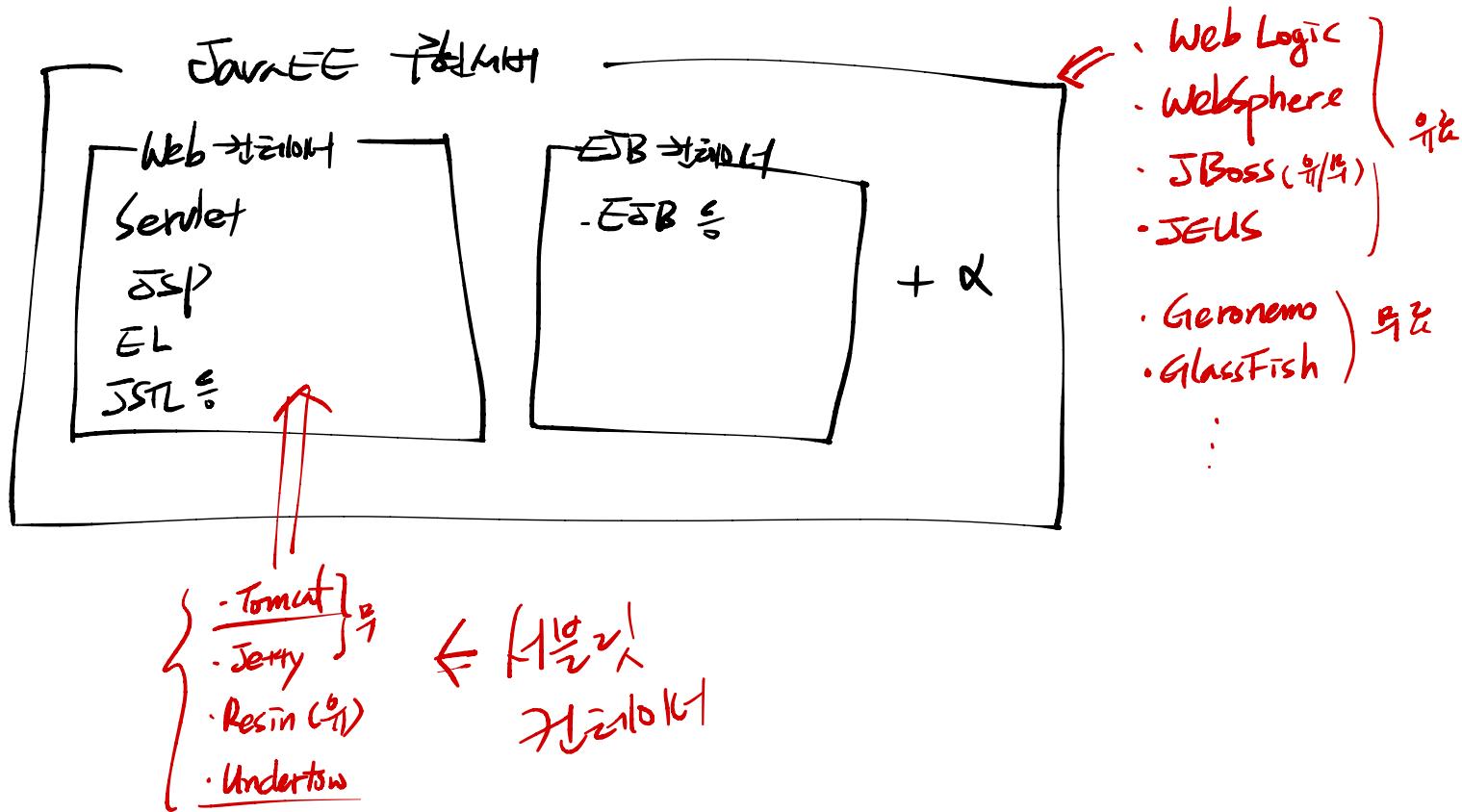
<del>JavaEE 버전</del> <del>제작년도</del>	Servlet	JSP	EL	EB
5	2.5	2.1		3.0
6	3.0	2.2	2.2	3.1
7	3.1	2.3	3.0	3.2
8	4.0	2.3	3.0	3.2

\* Java EE 1/2 버전부터 ESSENTIAL (Implements)

<del>JavaEE 1/2</del>	Web Logic	Web Sphere	JBoss	Tomcat
5	10.3	6.1, 7.0	6.0	6.0
6	11g, 12c(12.1.x)	7.0, 8.0	7.0	7.0
7	12c(12.1.3), 12.2.x	8.5, 9.0	8.0	8.0
8	12.2.1, 14.1.x	9.0.x	9.1, 8.2	8.5, 9.x

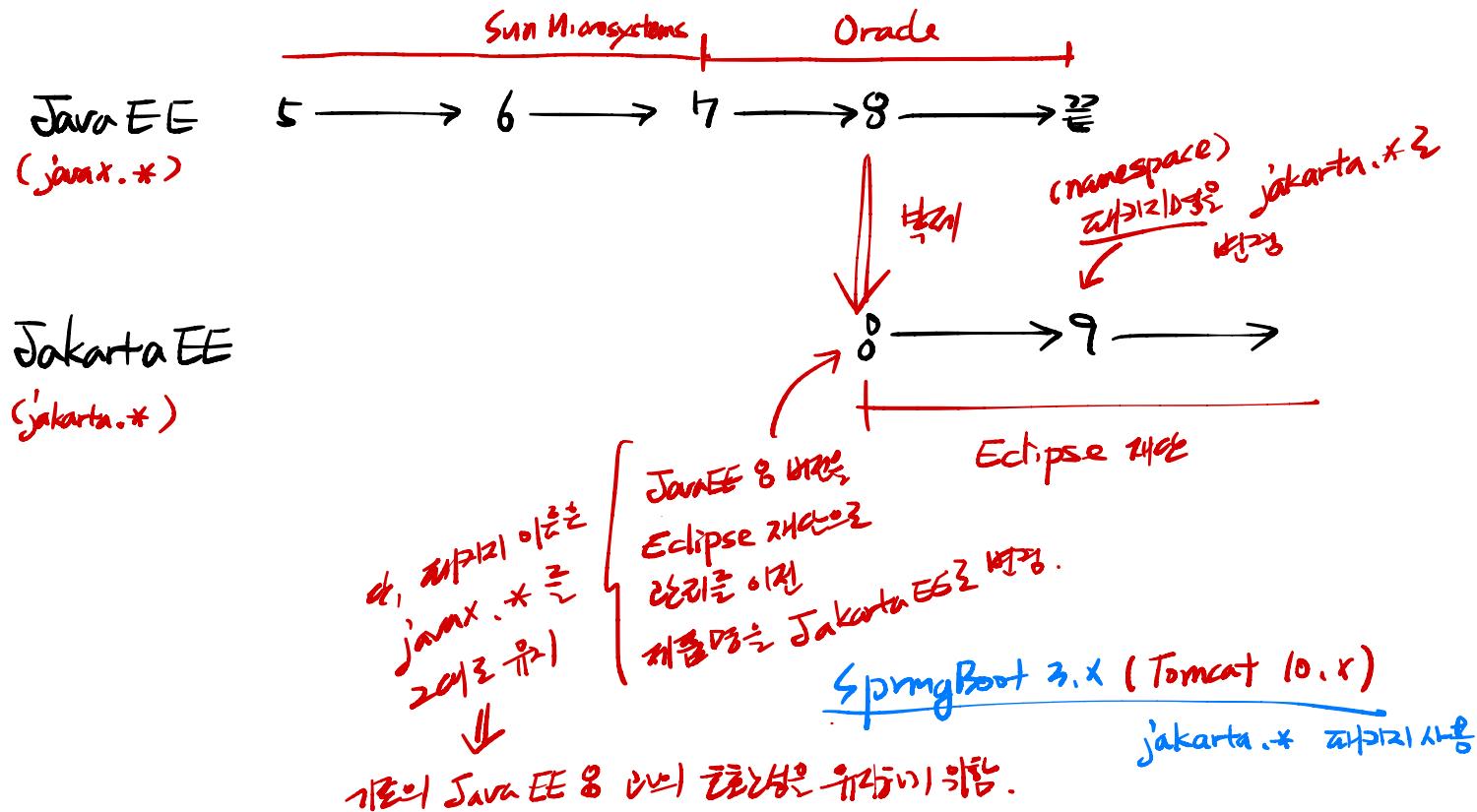
⇒ 2014년 10월 버전이 최신  
2014년 Java EE 버전으로 업그레이드 할 것.

## \* JavaEE 페미터 서브 분류

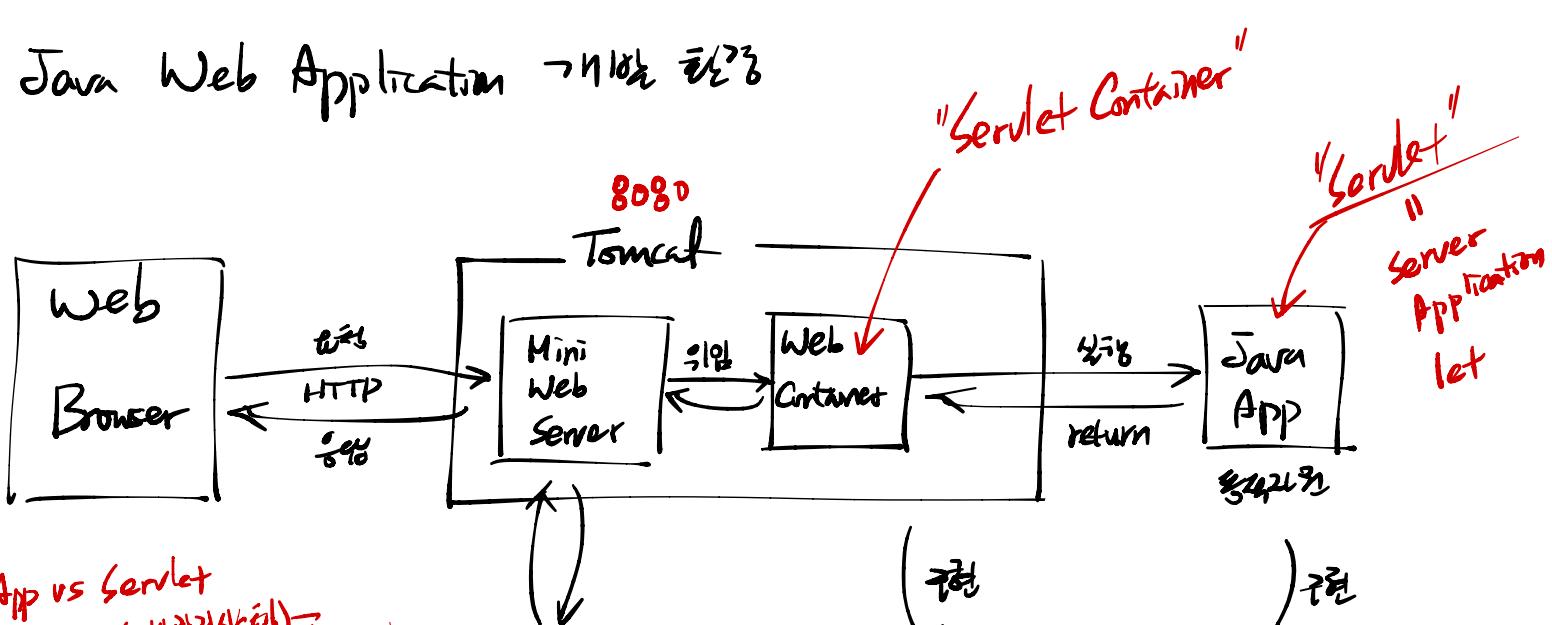


## \* JavaEE & JakartaEE

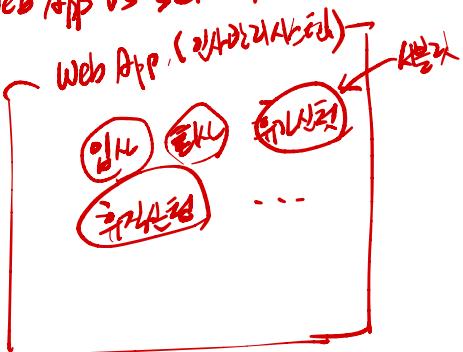
SpringBoot 2.x.x (Tomcat 9.x)



\* Java Web Application → 웹 어플리케이션



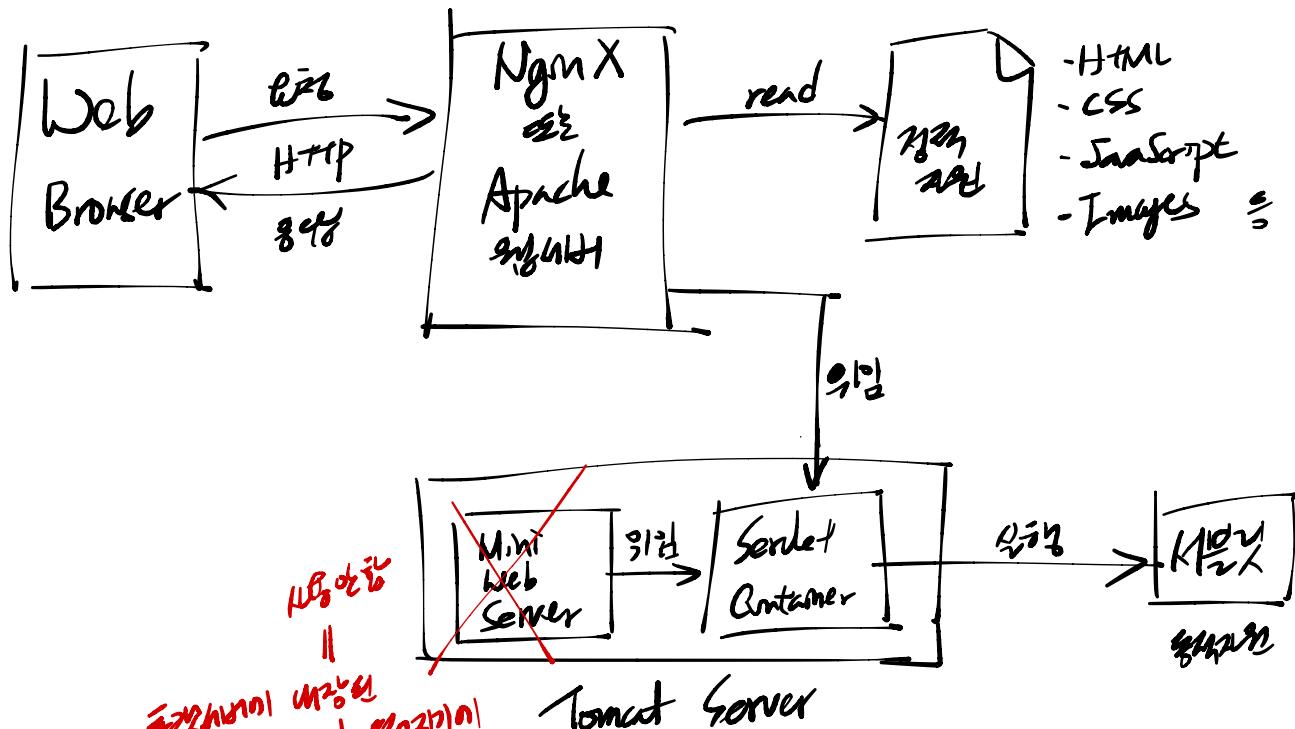
\* Web App vs Servlet



- HTML
- CSS
- JavaScript
- Images

Java EE  
- JSP, Servlet, ...  
- Servlet / JSP -

## \* Java Web Application 운영 원리



Tomcat Server  
External interface  
Frontend application  
middle-tier application  
backend application.  
Network.

## \* Tomcat Server

CATALINA\_HOME/

- bin/ ← 허버 실행과 관련된 파일
- conf/ ← 허버 설정 파일
- lib/ ← 허버 라이브러리 파일
- logs/ ← 허버 실행 로그 파일
- temp/ ← 허버 실행 중에 사용되는 파일을 두는 폴더
- webapps/ ← 웹 어플리케이션 폴더를 두다.
- work/ ← JSP를 Servlet 처리하기로 변환한 파일을 두는 폴더

启动文件 => startup.bat (Windows)  
startup.sh (Mac, Linux)

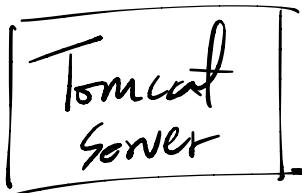
停止文件 => shutdown.bat (Windows)  
shutdown.sh (Mac, Linux)

\* Tomcat 끝내는 것

① 시작하기



startup.bat



Development (개발)

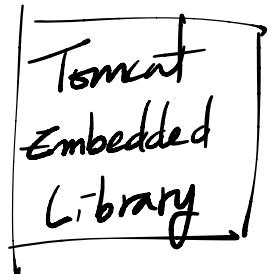
↓ .war

Operation (운영)

② 구현하기



call



운영

Spring Boot



Development + Operation = DevOps

## \* Embedded Tomcat 끝까지 써보기

new Tomcat()

- port = 8888
- baseDir = temp
- URLEncoder = UTF-8

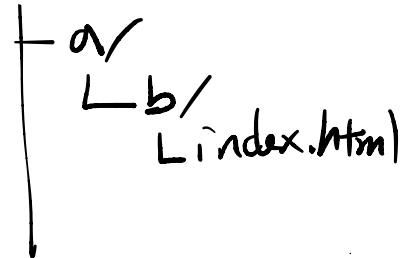
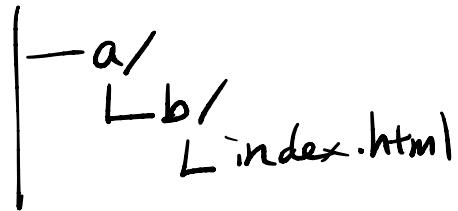
• 실행 파일 경로  $\Rightarrow$  /  
|- 정적 자원 경로  $\rightarrow$  src/main/webapp/  $\leftarrow$  .html, .css, .js, .gif 등  
|- 동적 자원 경로  $\rightarrow$  bin/main  $\leftarrow$  .class, .properties, .xml 등

실행 주소: http://localhost:8888 ↗

- ① 정적 파일 경로인 /index.html 을 찾는다.
- ② 동적 파일 경로인 /index.html 3 번째가 실행된 내용을 찾는다.

\* აბსოლუტური

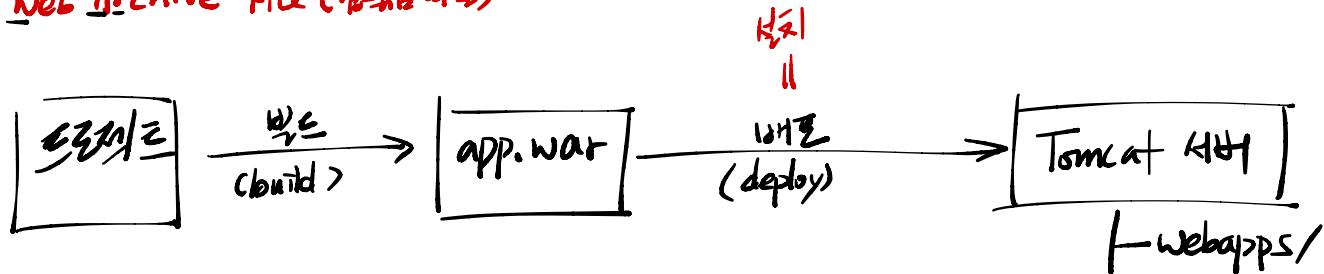
http://localhost:8888/ → src/main/webapp/



\* 웹 어플리케이션 프로젝트 배포

① .war 파일 배포

Web Archive File (웹아카이브)



② 코드 + API 라이브러리 (라이브러리)  
= springboot 앱



\* Maven 파일은 쉽게 편리한 확장자를 가짐

구조도



.html, .css, .js, .gif 등 이미지 파일

WEB-INF

web.xml ← 딜레이션 데스크립터 (Deployment Descriptor File; DD File)

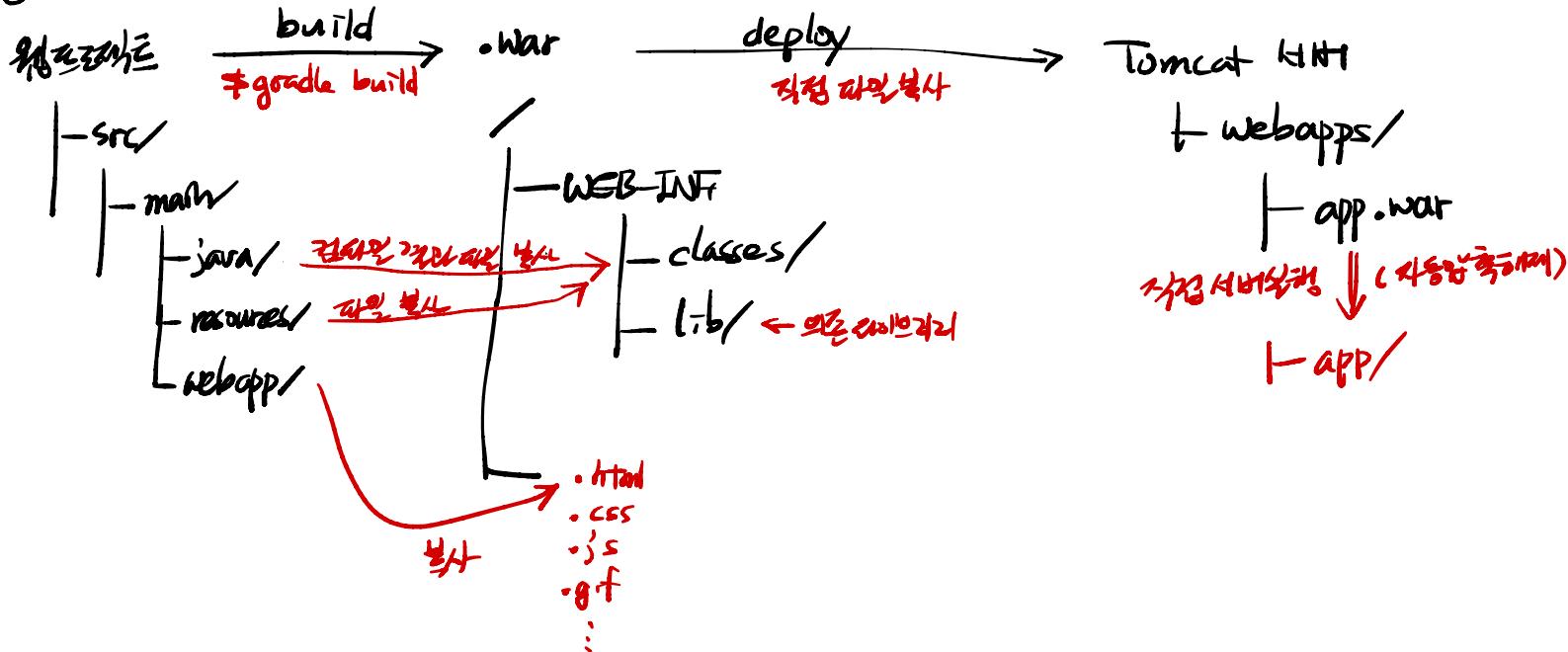
classes/ ← .class, .properties, .xml

lib/ ← .jar

.xml, .properties 등 설정 파일

\* 웹프로젝트 디렉토리

① 디렉토리 구조  $\leftarrow$  웹애플리케이션 기본을 갖춘 후에 버전으로 1842



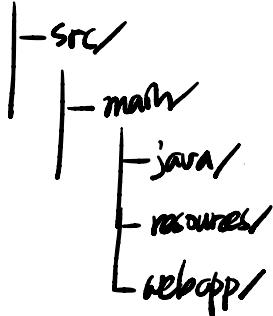
\* 웹프로젝트 파일

② Eclipse 웹 파일 <서버 설정을 적용하는 동안 서버를 파일에

생성되는

Eclipse 임시 파일 폴더

이경스 위치는/.metadata/.plugins/org.eclipse.wst.server.core/



tmp0/

- conf/ ← tomcat 설정 / conf, 폴더는 복사해온다

- logs/ ← 실행 기록 파일

- temp/ ← 일정 주기로 자동으로 파일을 두는 곳

- webapps/ ← 사용 안 함

- work/ ← JSP 변환 파일 두는 곳

- wtpwebapps/ ← 웹프로젝트 바로 폴더

생성되는/

- WEB-INF  
  |  
  + classes/  
  |  
  + lib/

생성되는

\* 웹프로젝트 ID IntelliJ

② 웹프로젝트 + WAS 라이브리 <= SpringBoot 환경



\* 풀不尽 서버로 진화하여 등장

① 웹서버 → ② 애플리케이션 서버

Server App

- 파일통신구현
- 서비스레이팅 구현
- 전송통신망
- 웹서버 구현

Server App

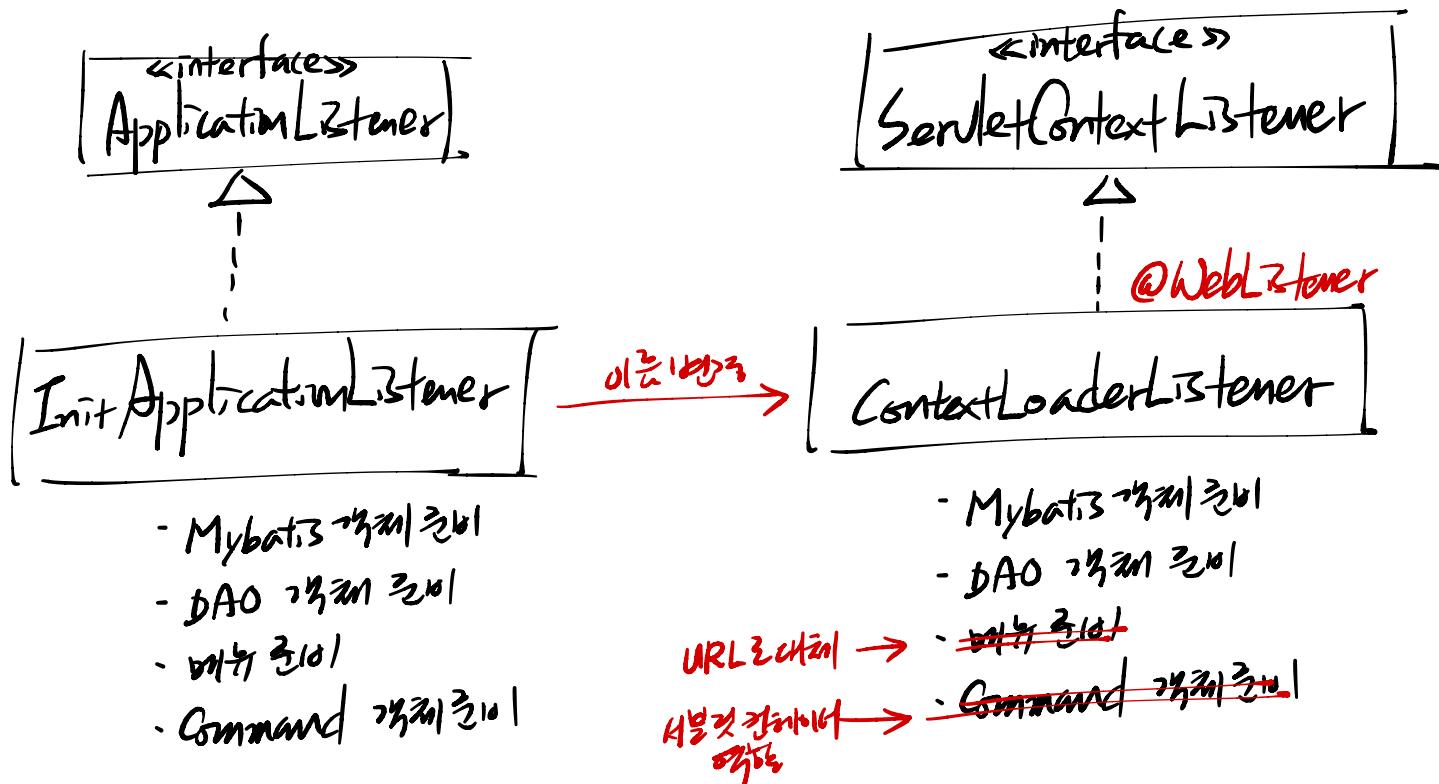
- ~~파일통신구현~~
- ~~서비스레이팅 구현~~
- ~~전송통신망~~
- ~~웹서버 구현~~

Web

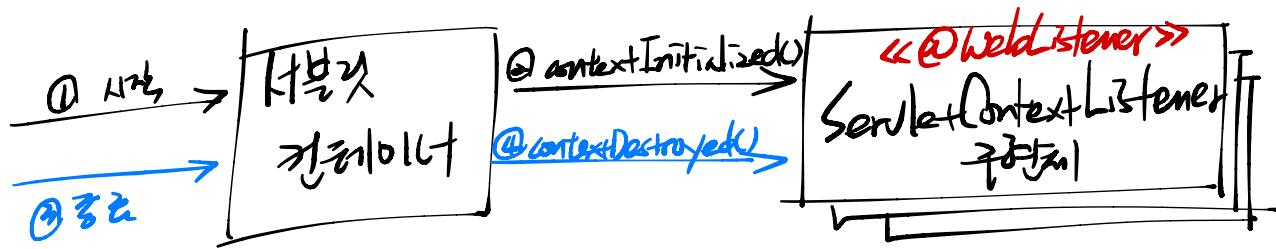
Tomcat

- HTTP 프로토콜을 처리하는 웹브라우저 구현
- 서비스레이팅 구현
- 전송통신망 구현
- 웹서버 기능으로 구현
- 웹사이트의 URL을 기능 구현

\* 리스너 만들기



## \* ServletContext Listener



\* **@WebListener** effect하기

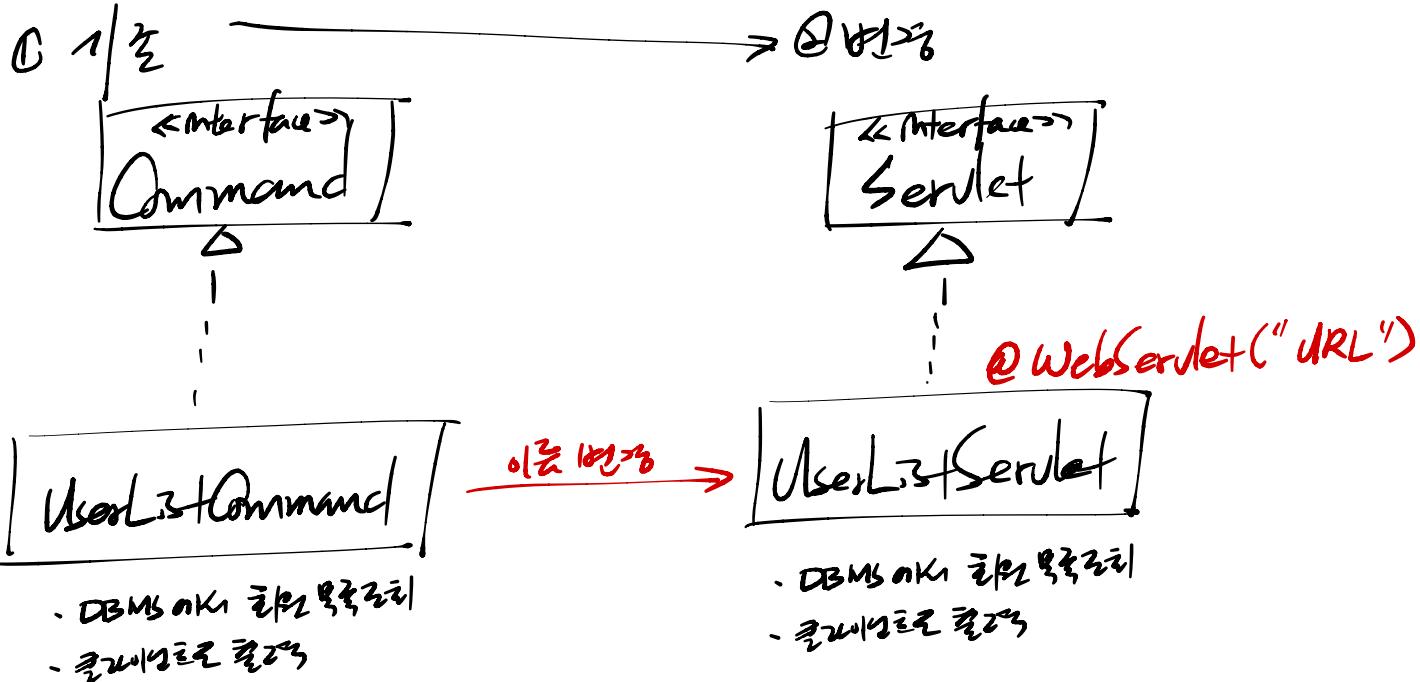
- 라이브러리에 서블릿 컨테이너에 의해 등록되는

수명주기 메서드를

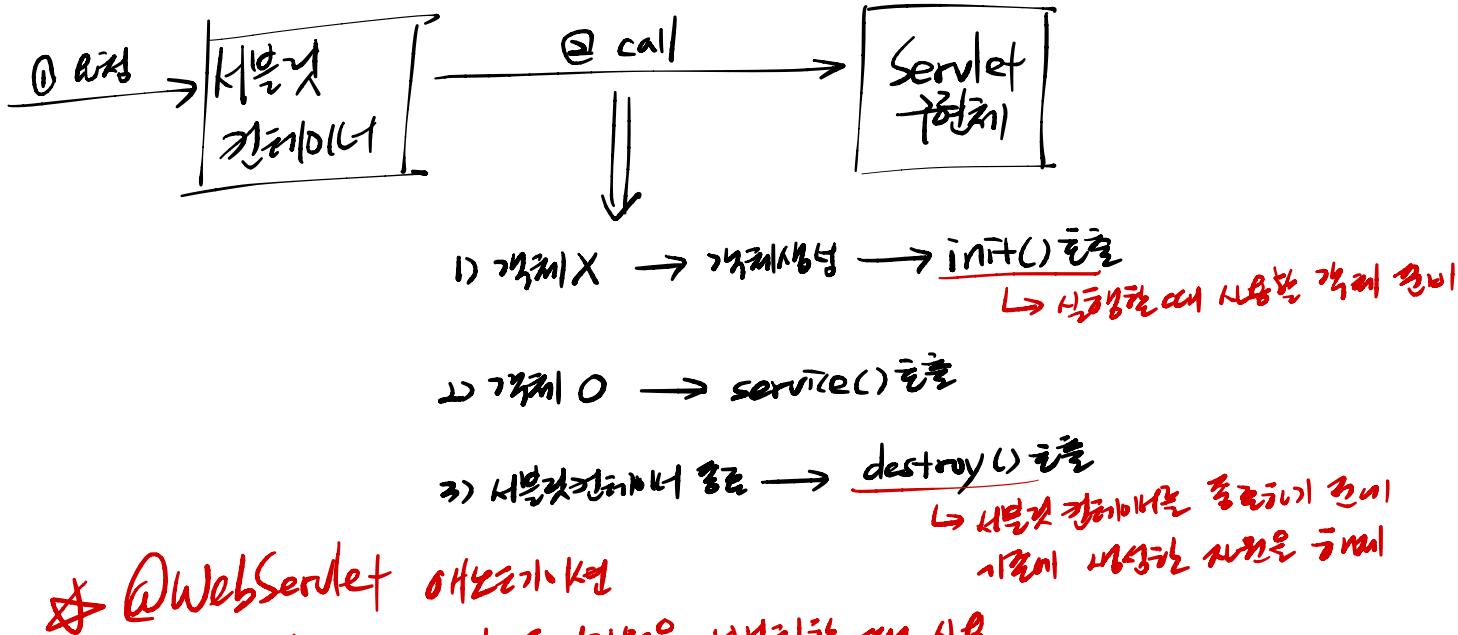
제공하는

클래스를 말합니다.

\* 허용 가능한 방법



## \* Servlet 구조



## \* WebServlet의 특징

- 서블릿작동이 끝나면 종료되는 대체로 특징은 유지된다.
- 클라이언트에게 응답을 때 그 내용을 처리하는 코드를 작성한다.
- URL을 이용하여 코드를 처리하는 처리를 적용한다.

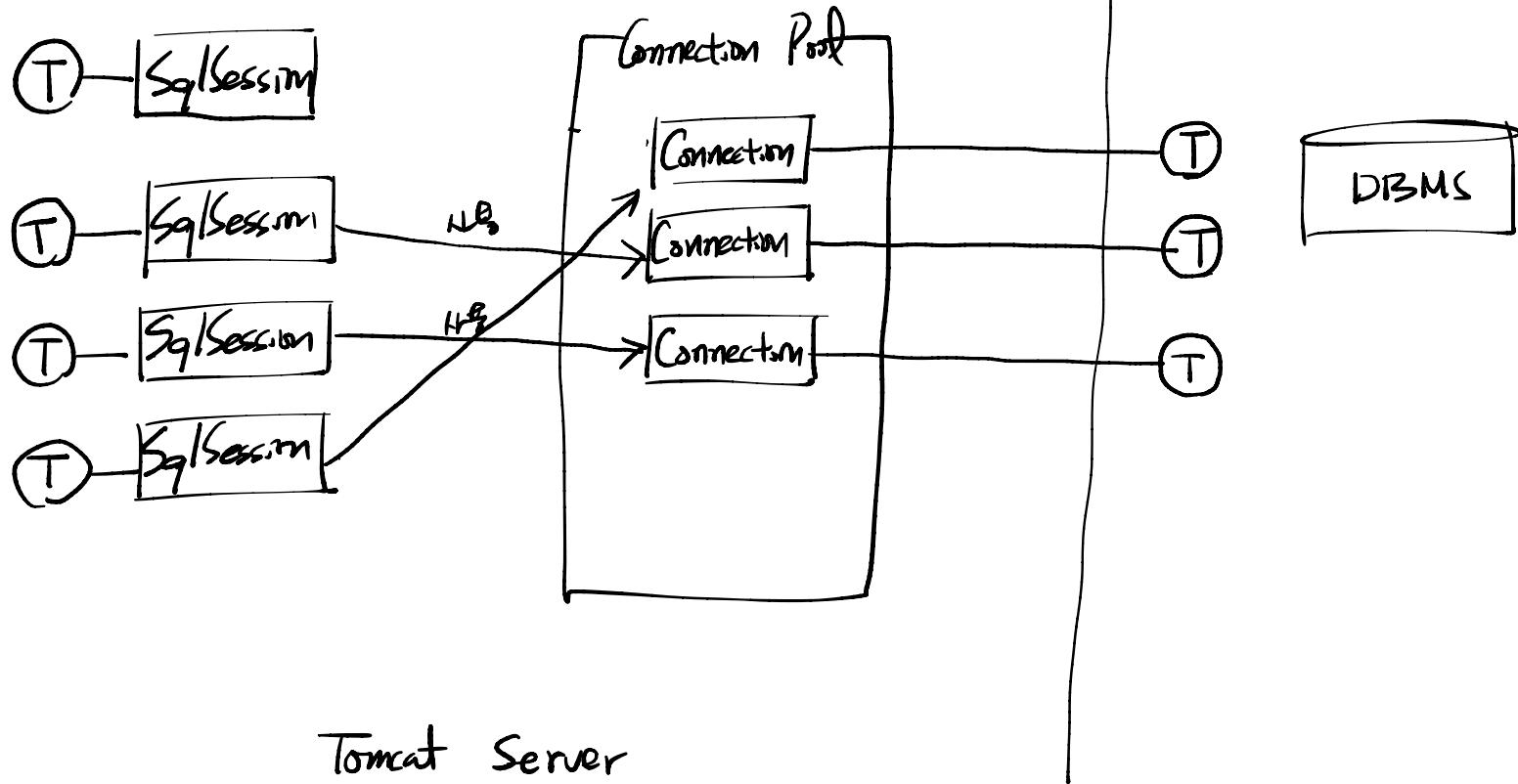
\* HTML 문서에서 다른 자원을 요청하는 경로 주소

<u href="/user/view?no=11"> user11 </u>

HTML reference

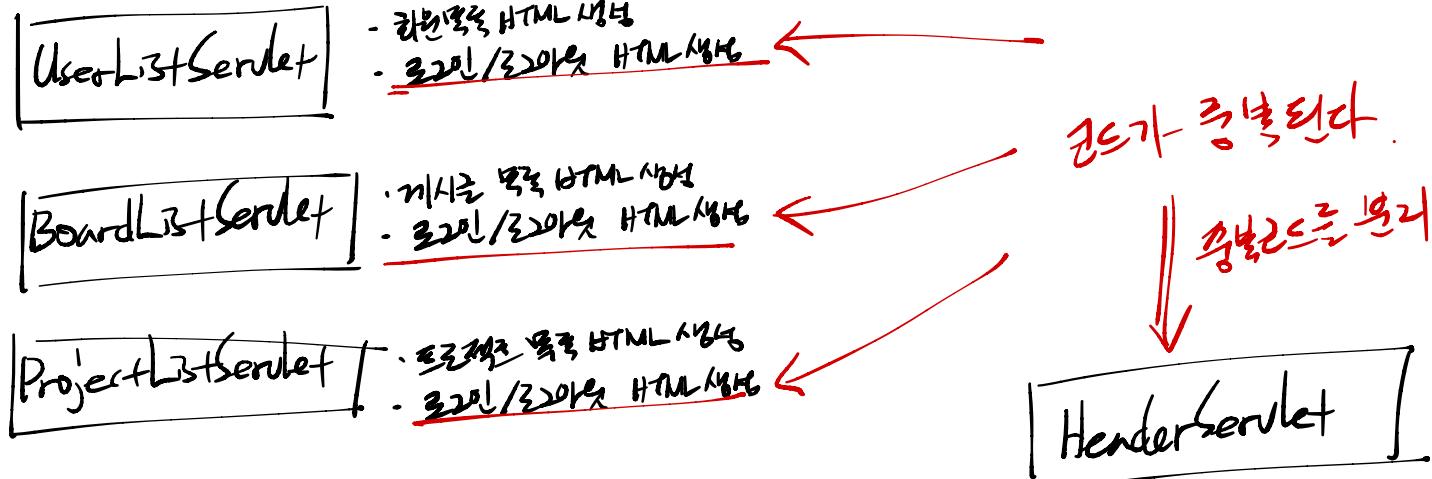
→ 템플릿 엔진  
→ 컨트롤러  
→ 서비스  
→ 뷰  
→ 편집된다.

## \* DBMS et SqlSession



\* 서블릿 include → 다른 서블릿의 실행 결과를 포함시키는 방법

## ① 예제



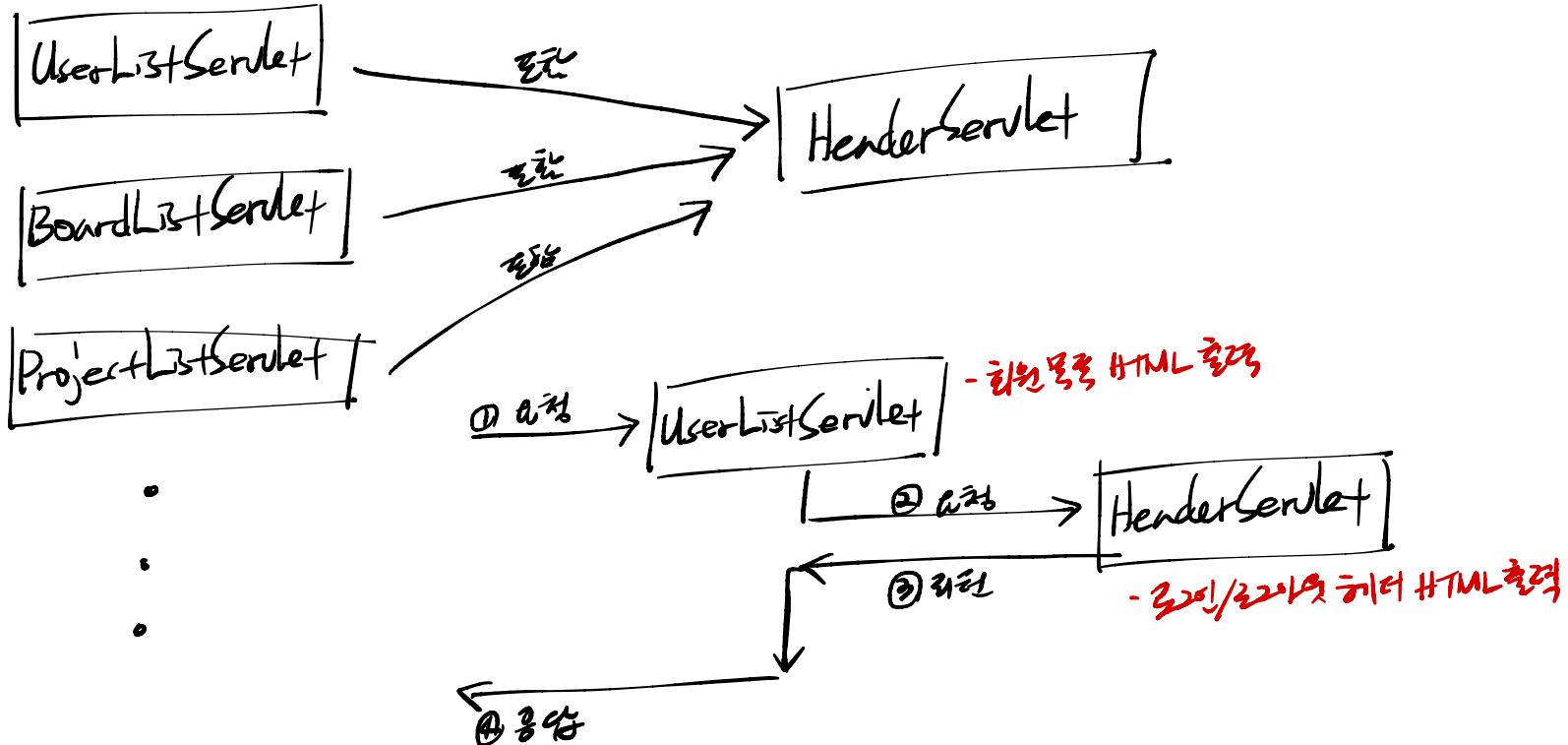
•

•

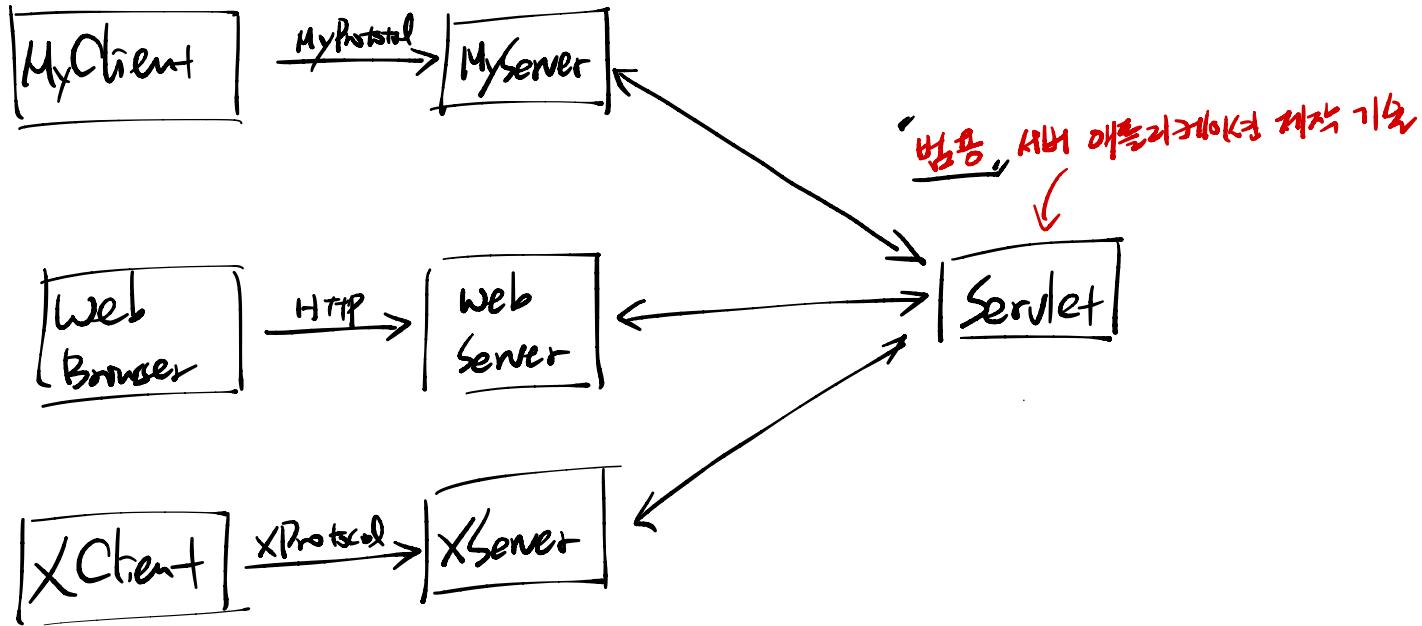
•

\* 서블릿 include → 다른 서블릿의 실행 결과를 포함시키는 방법

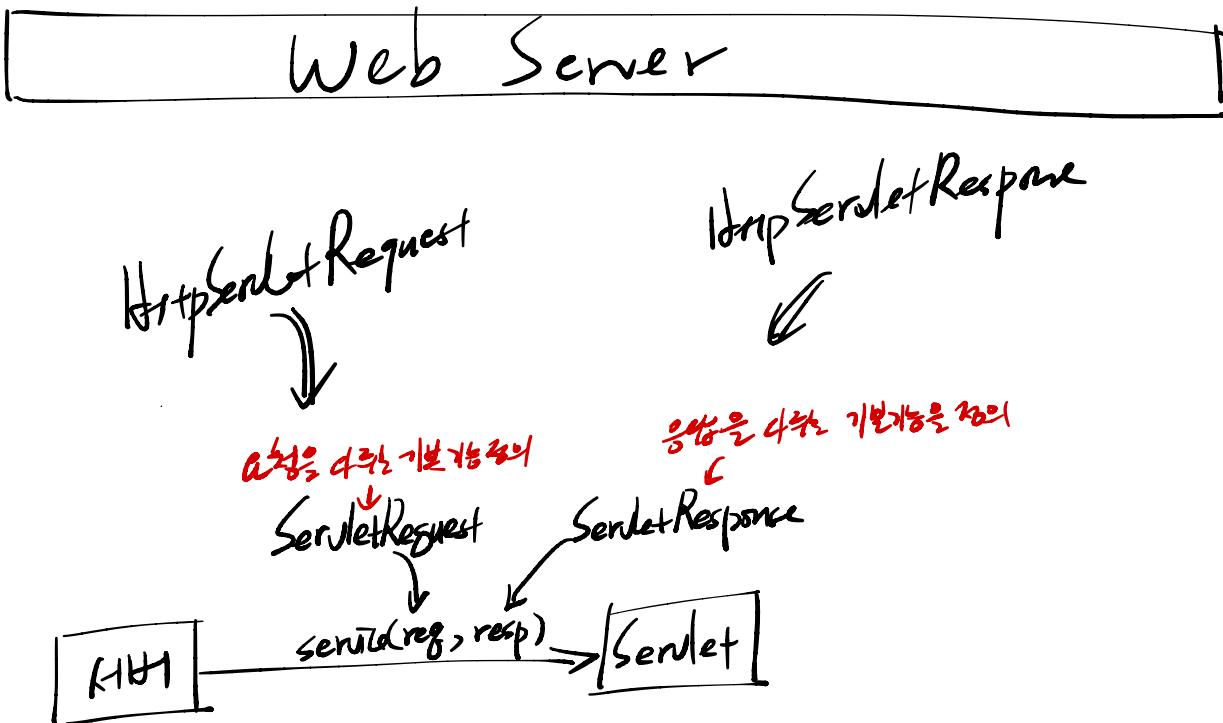
② 구조



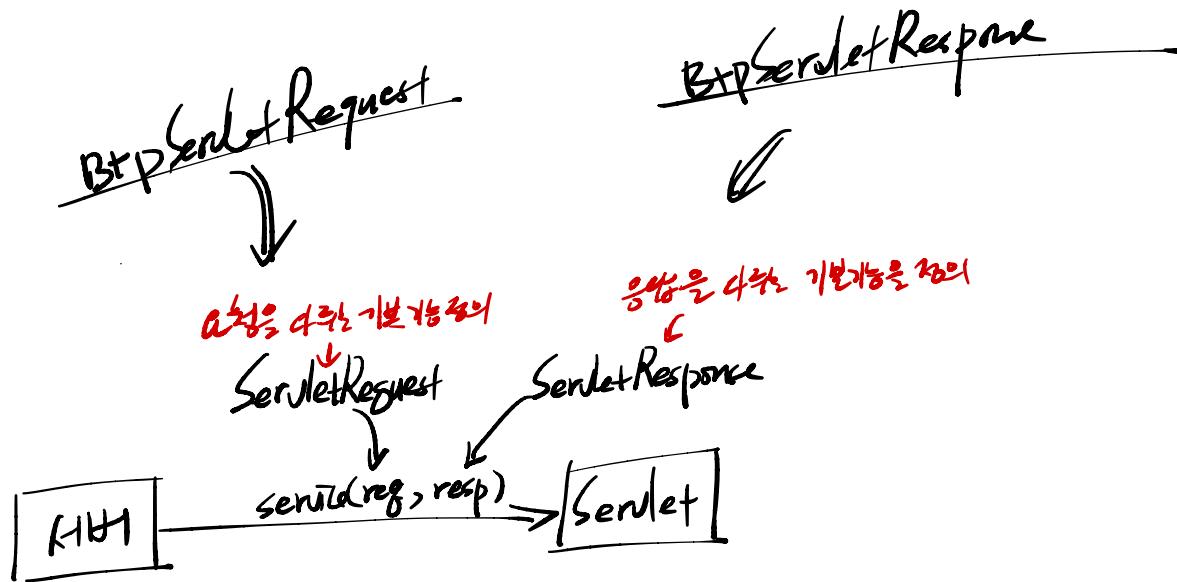
## \* Servlet چیزی چی



## \* ServletRequest et ServletResponse

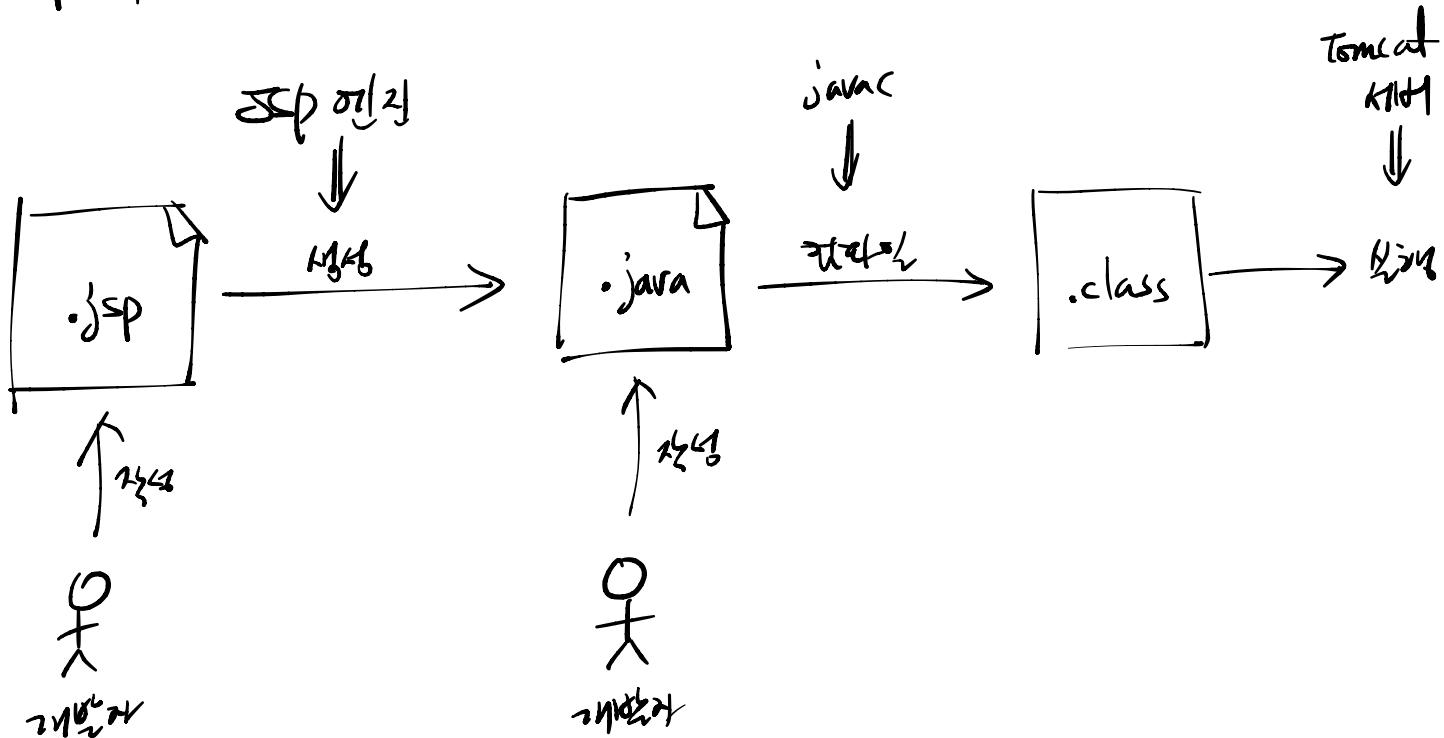


# Bitcamp Server

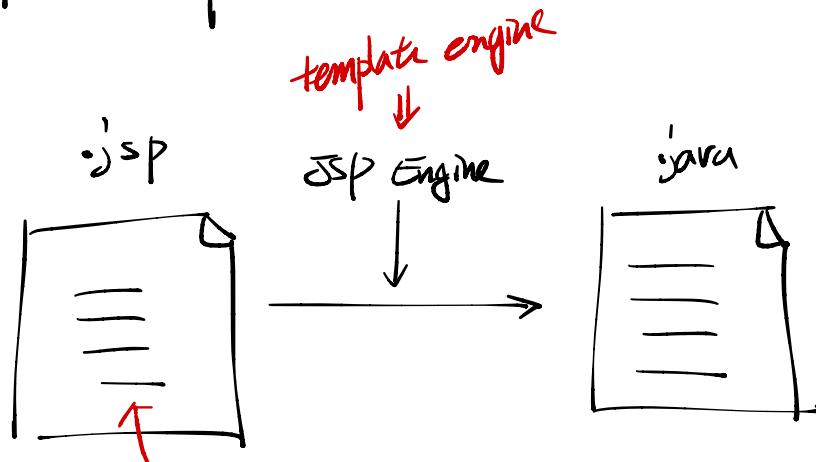


## 50. UI 흐름을 자동 생성하기 — JSP (Java Server Page)

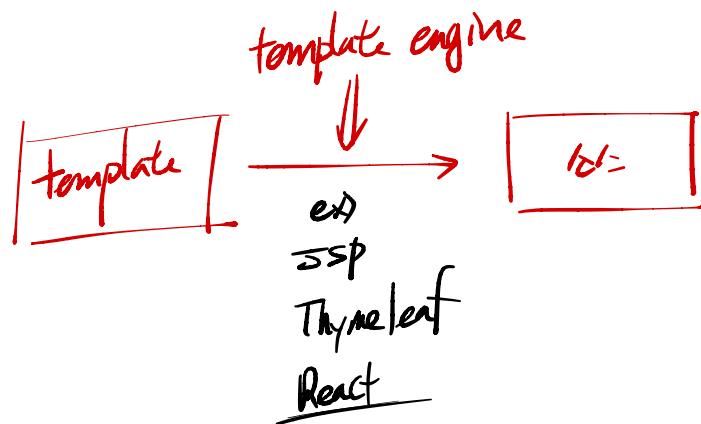
\* JSP 구동 원리



## \* JSP - Template 엔진

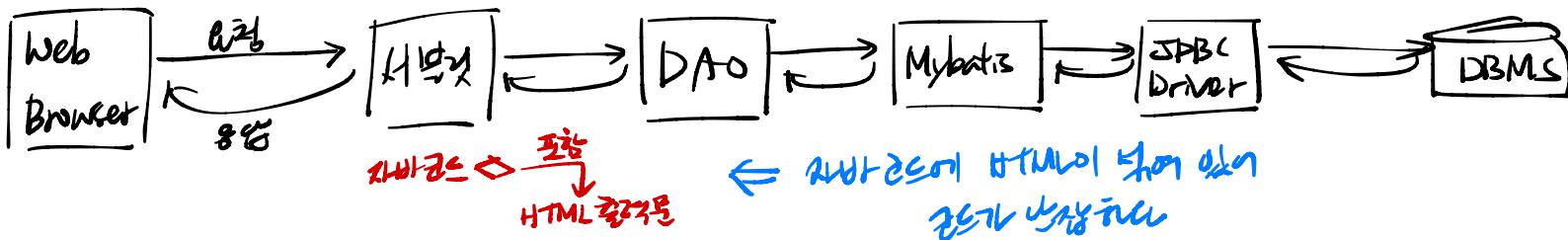


자바 코드를 만들고자 하는 것  
자바 코드를 만들자  
자바 코드를 만들자  
"자바 코드를 만들자"  
"자바 코드를 만들자"

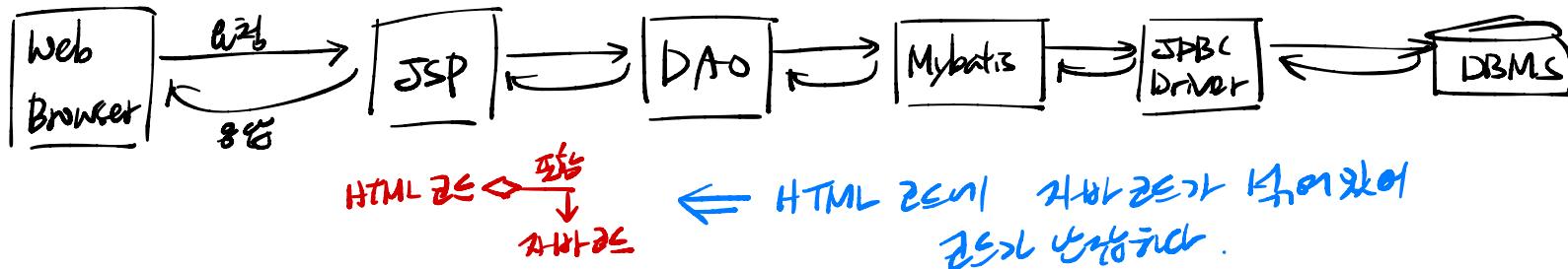


\* Servlet 와 JSP 애플리케이션 아키텍처

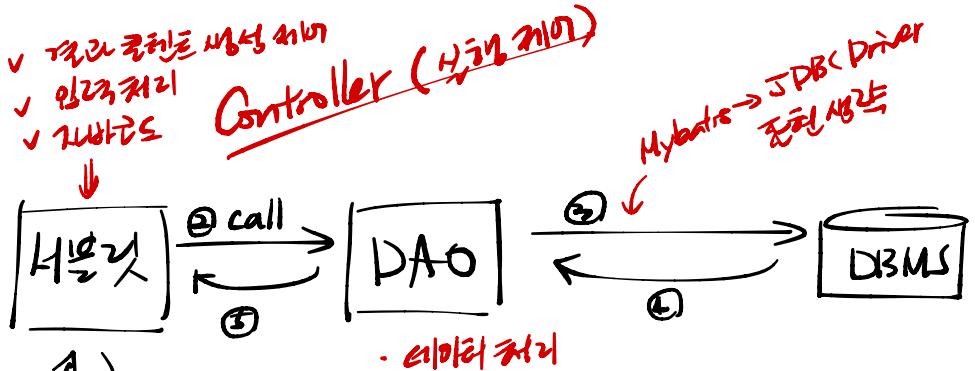
① 서블릿 애플리케이션 아키텍처 - 이전부분



② JSP 애플리케이션 아키텍처 = MVC | 확장



## \* MVC 2 아키텍처

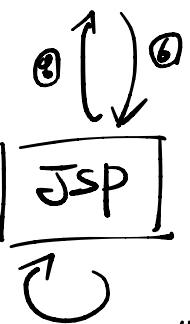


View (UI) (설명)

(다양한 환경에서 쓸 수 있도록 가짐)

UI를 위한 데이터 처리

View(설명)



⑧ HTML 내용 포함

UI 내용

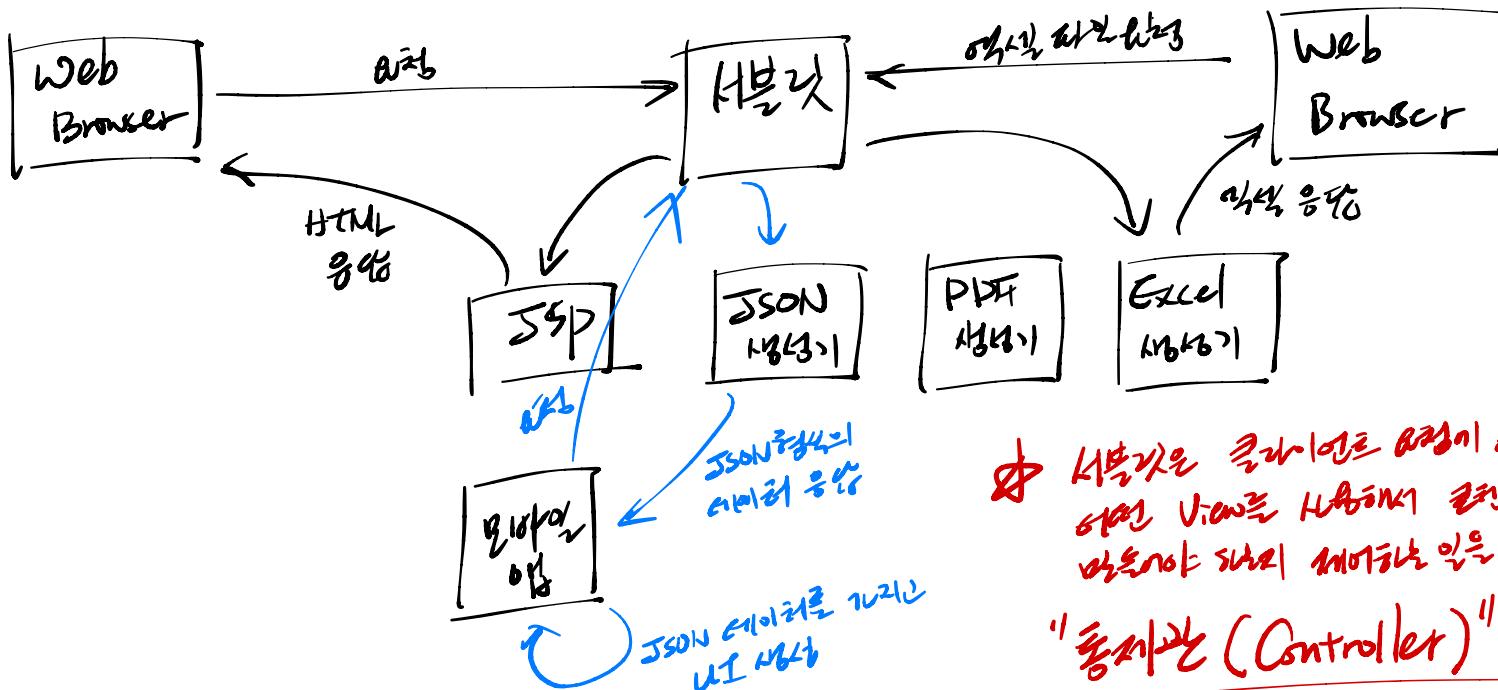
View (설명)

Model (DB 접근 수단)  
Mybatis → JDBC Driver  
정적 접근

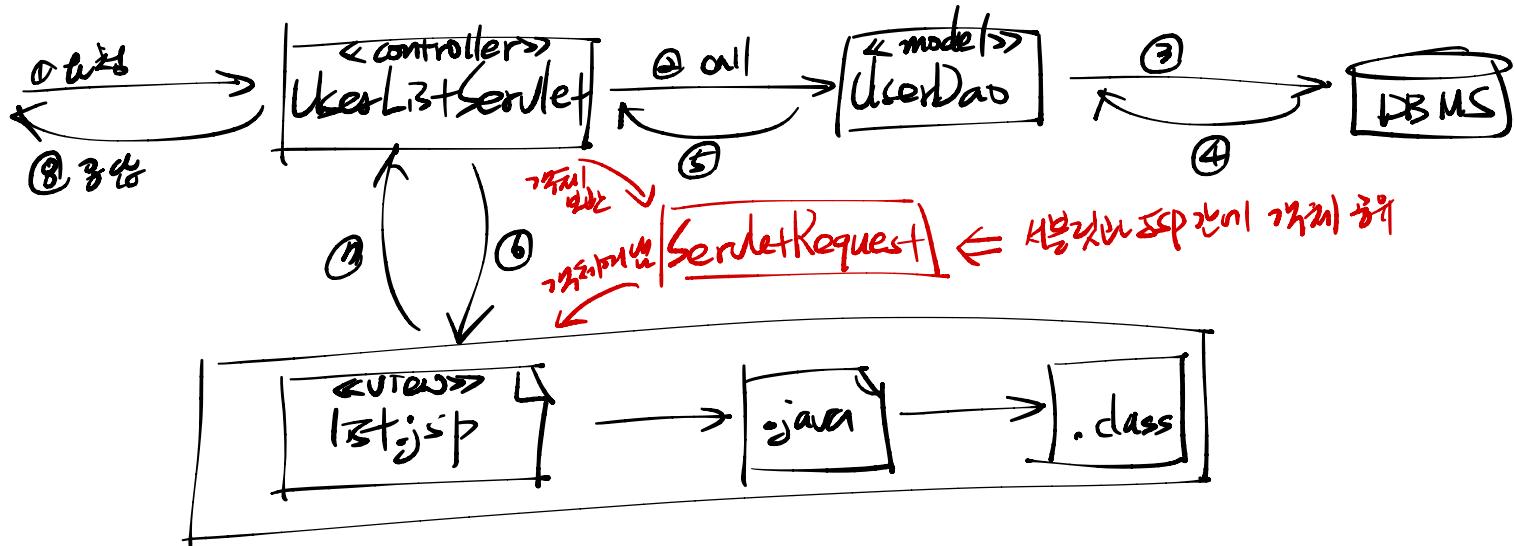
\* 서블릿에서 웹면 흐름을 통제할 때의 예는 아래

- ① 코드가 쓰여있지 않은 유지보수하기 힘들다
- ② 결과 표현코드를 통제하기 힘들다.

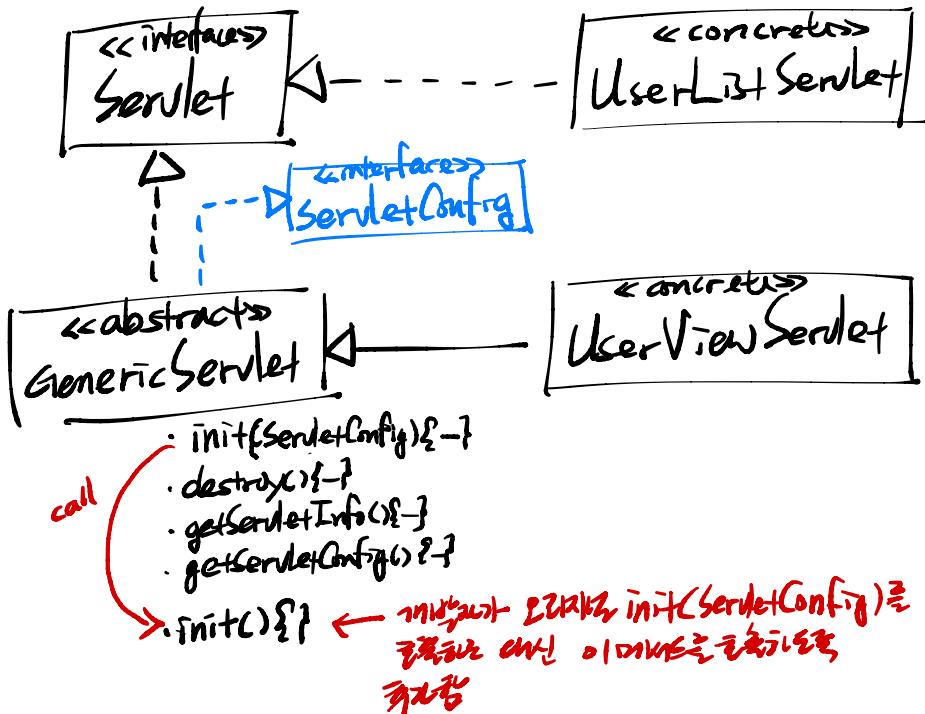
결과적으로 디자이너가 통제해 어렵다. 정부에 있는 경우 관리하기 어렵다.



\* MVC2 ဆို ဘွဲ့တော်များ



\* 亂世のアーキテクチャ



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

service() ↗

## 56. DAO는 로그를 뿐만 아니라

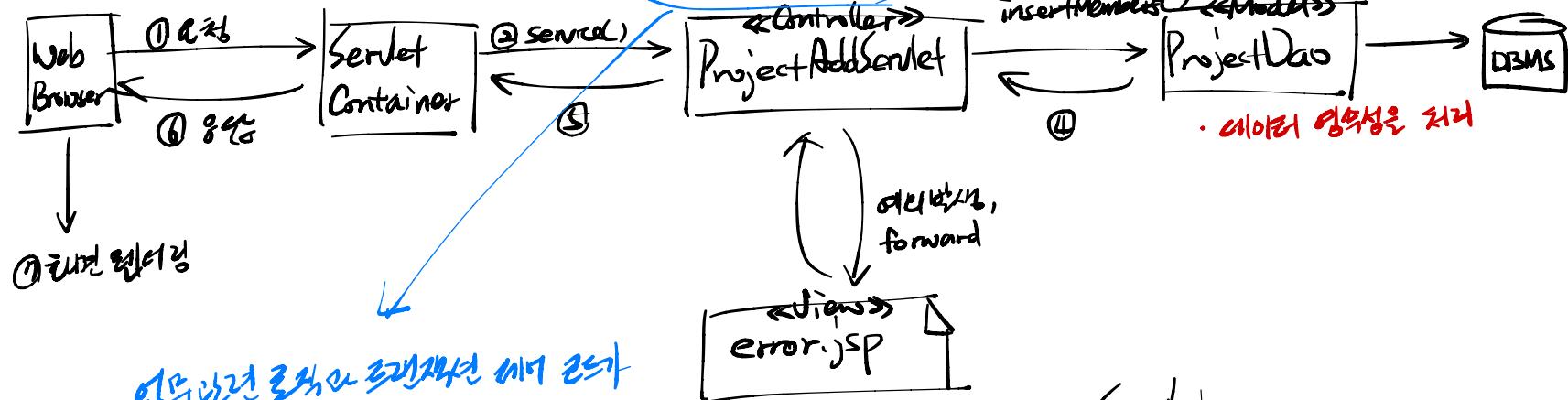
\* 확장

- DAO가 내용을 넣기 위해 데이터가 있는 풀과 데이터 준비
- JSP가 "
- 풀과 관련된 풀과 처리
- 데이터를 넣기

③ insert()  
insertMembers() <--> Model

④ DAO

• 데이터 영역성을 처리

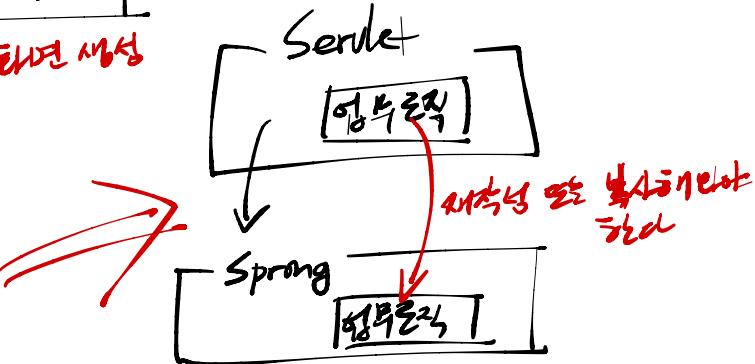


영역별로 관리되는 관리자에게 데이터  
Servlet 가공해야 하는데

↓  
Servlet 가공이 필요하다.

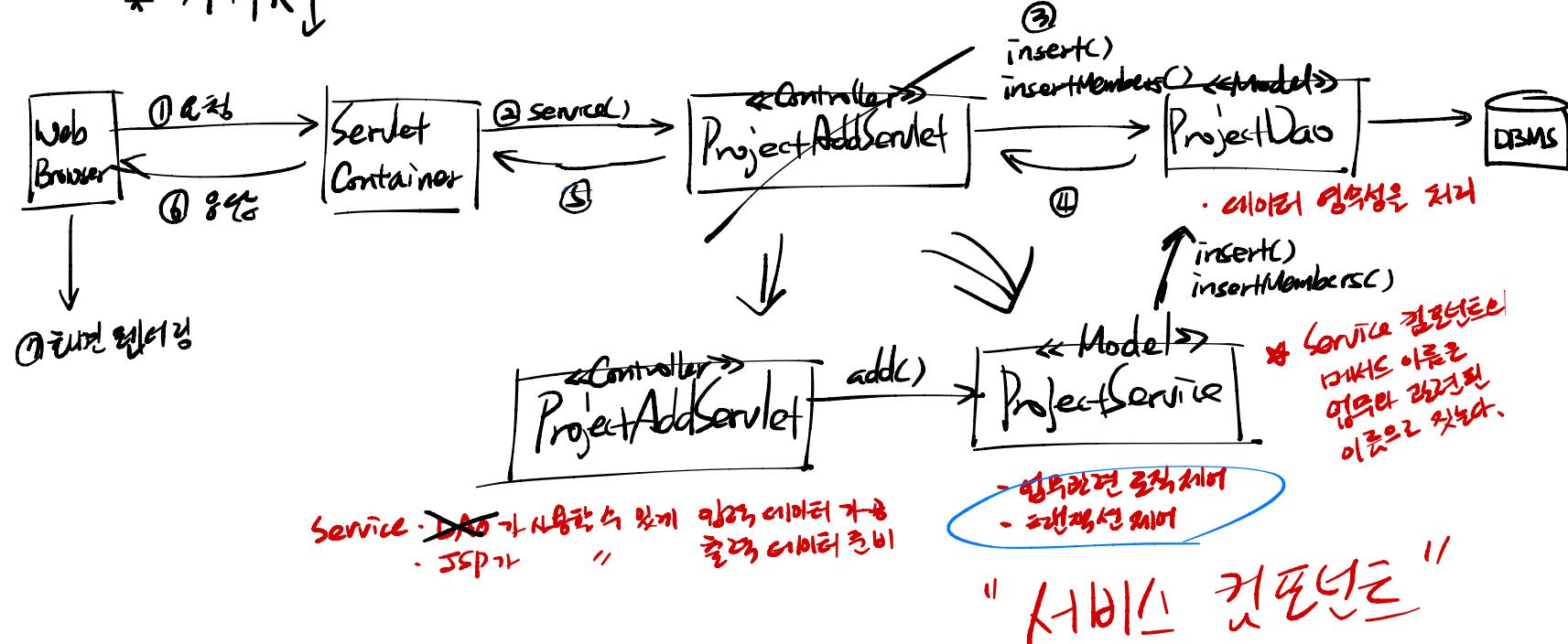
↓  
Servlet은 다른 기술로 고쳐야 한다  
이유로는 오직 데이터만이다

→ 유지보수  
변경하기



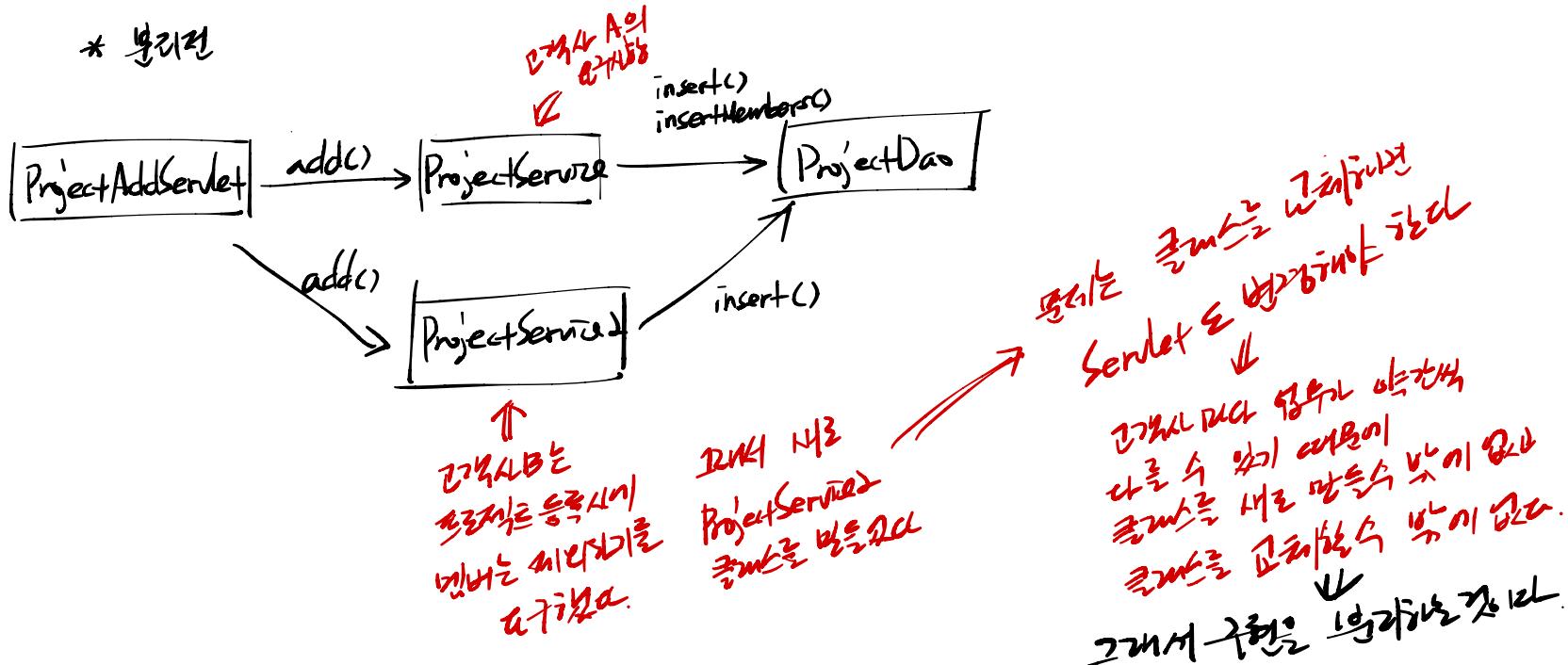
## 56. 웹 서비스로의 분리화하기 - 서비스 컨포넌트 등장

\* 개수↓



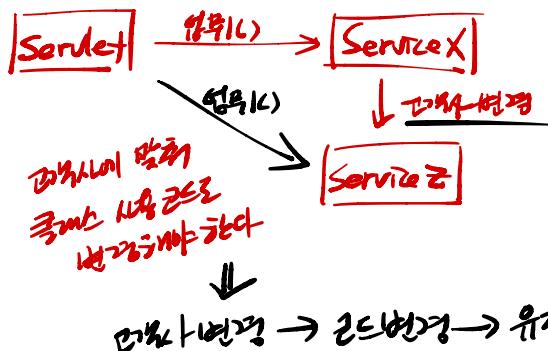
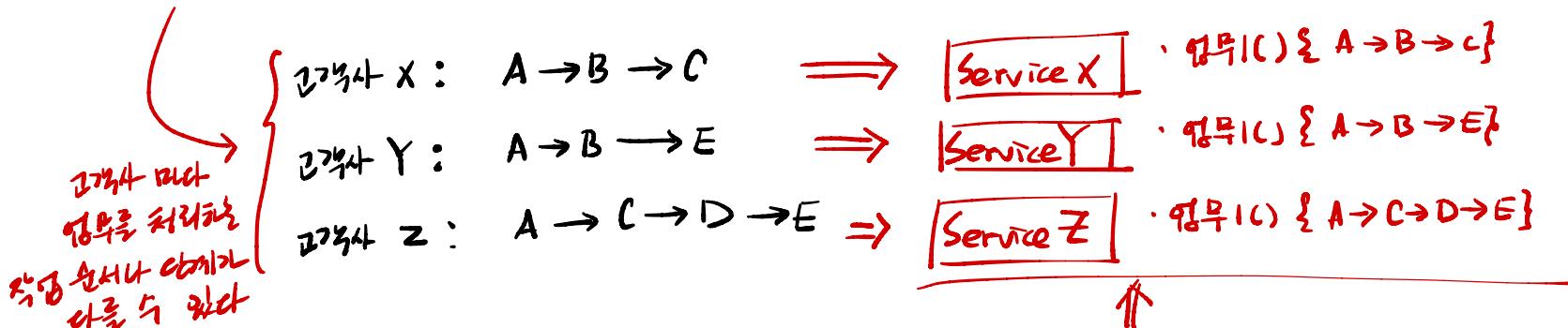
## 56. 웹了很久 프로젝트 - 인터페이스와 구현을 분리하기

\* 분리점

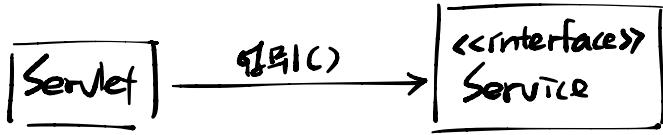


## \* 고객사와 업무

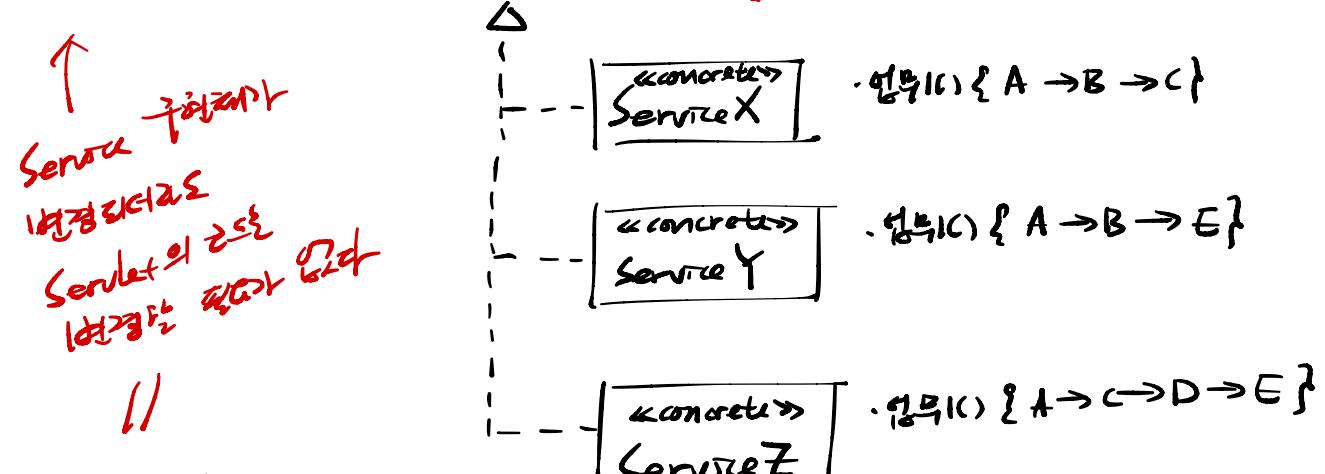
업무1: A작업  $\rightarrow$  B작업  $\rightarrow$  C작업  $\rightarrow$  D작업  $\rightarrow$  E작업 (단순한 업무흐름)



고객사 업무에 맞춰 동작하는  
클래스를 따로 만들거나 하는데



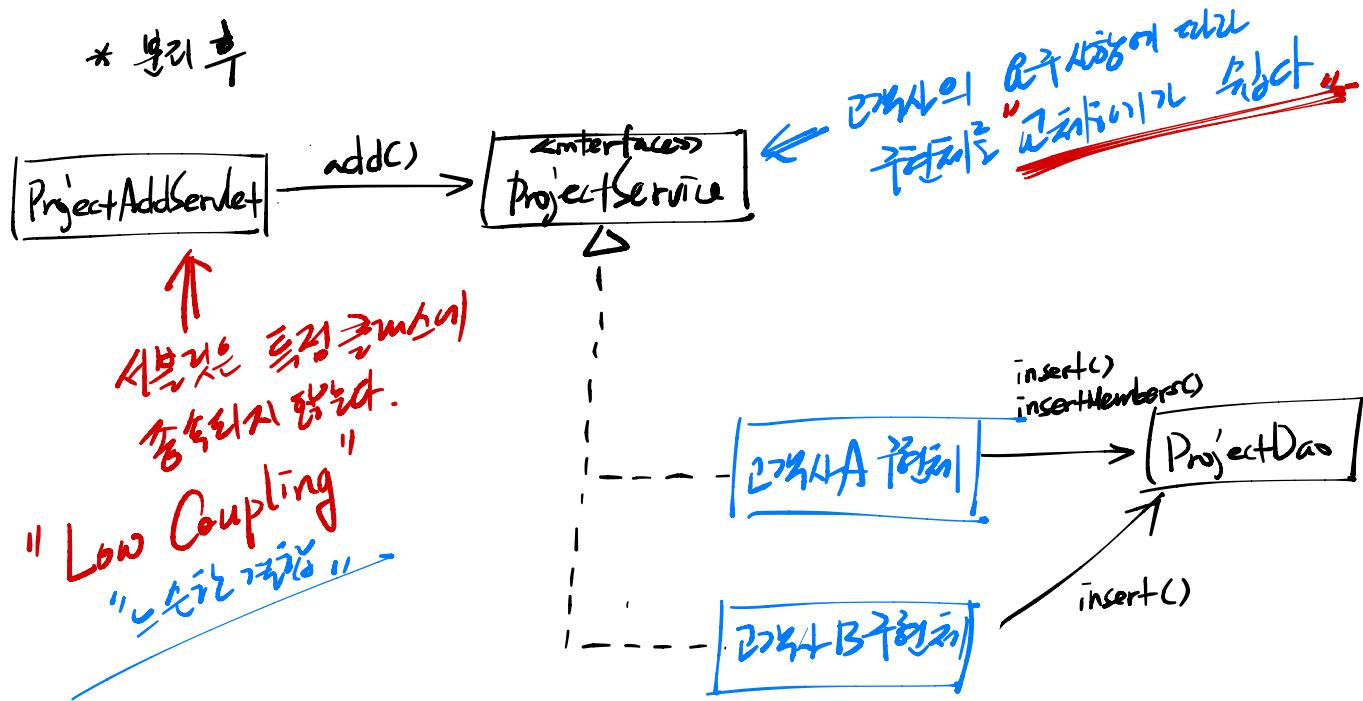
제작자에게  
제공되는  
제작자  
제작자



“제작자에 제작자는  
 → 제작자”는 제작자에 제작자!

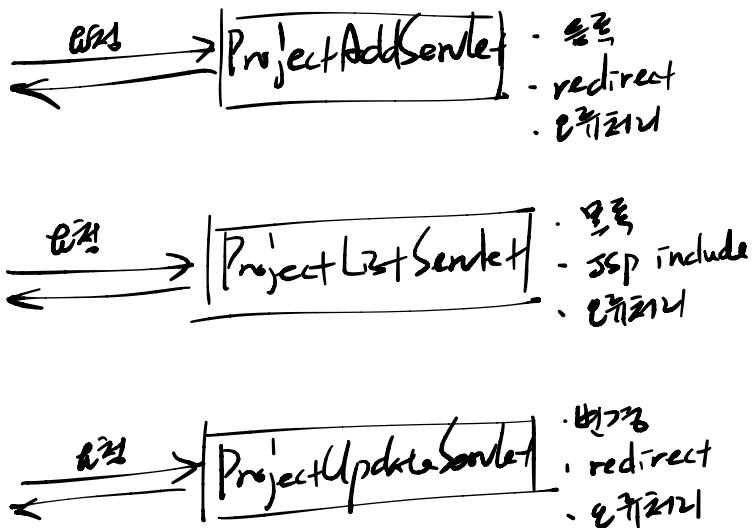
## 56. 헤리티지 풀다운하기 - 인터페이스와 구현을 분리하기

\* 분리 후

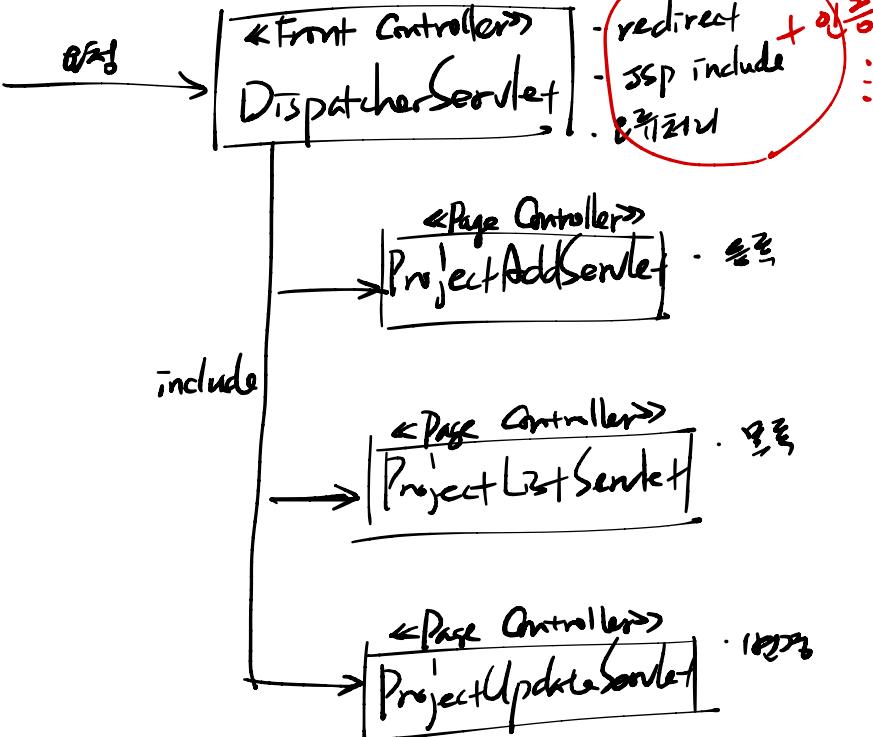


## 57. Front Controller \Controllers သိမ်

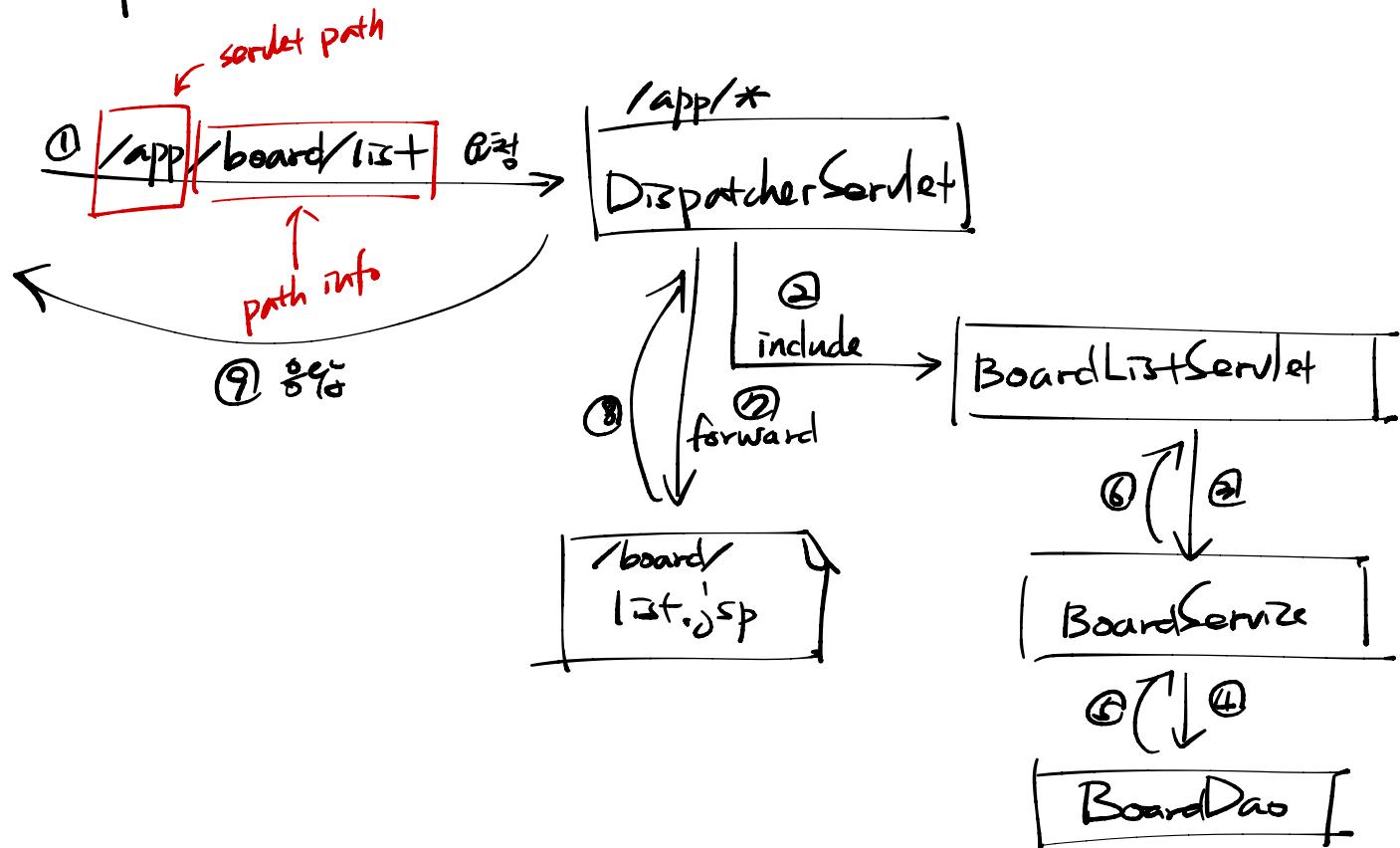
\* အကျဉ်းချုပ်



\* အကျဉ်းချုပ်

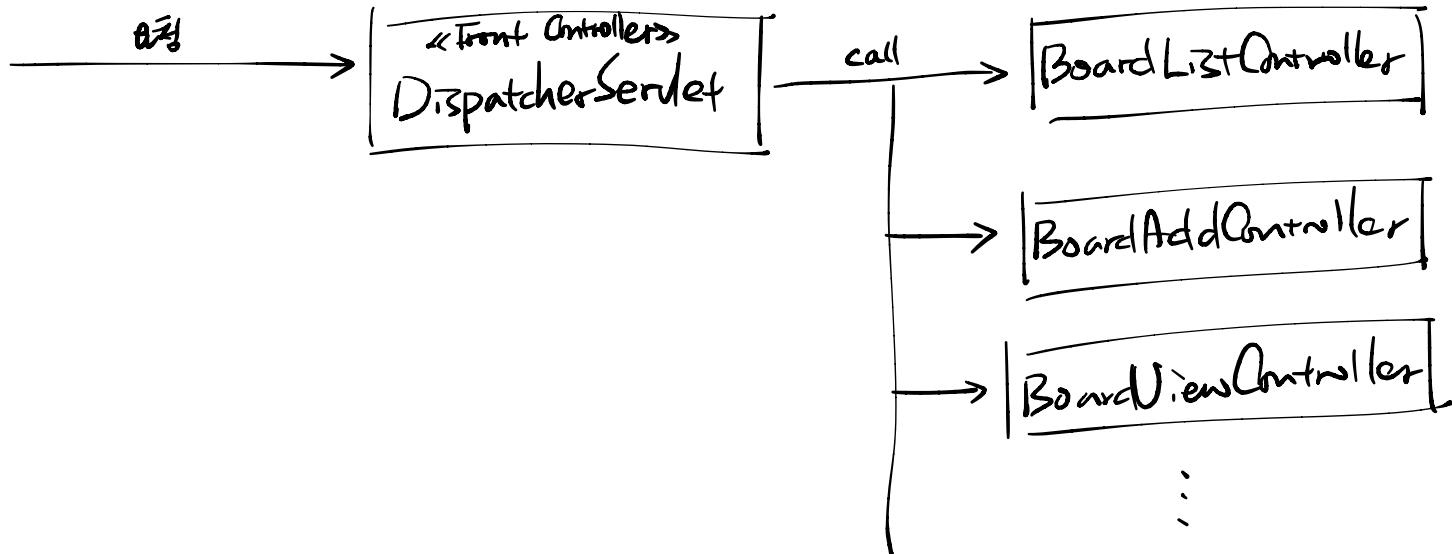


## \* DispatcherServlet



58. ~~제이비디~~ → ~~간단한~~  $\frac{1}{2}$  POJO 를 바꾸기

Plain Old Java Object ← 일부 기능을 쓰지 않고, 새롭게 기능을 추가하는  
수정하기 쉬운 것



58,  $\text{HTTP}(2) \rightarrow \text{HTTP}(\frac{1}{2})$  POJO 3 바꾸기

( $\frac{1}{2}$ )

@WebServlet("/user/list")

```
class UserListServlet  
    extends HttpServlet {
```

UserService userService;

void init() {

}

void doGet(HttpServletRequest request)

=

}

}

⇒

controller map nl  
→  $\text{HTTP}(\frac{1}{2})$  3번 째 짜리

POJO

```
class UserListController {
```

UserService userService;

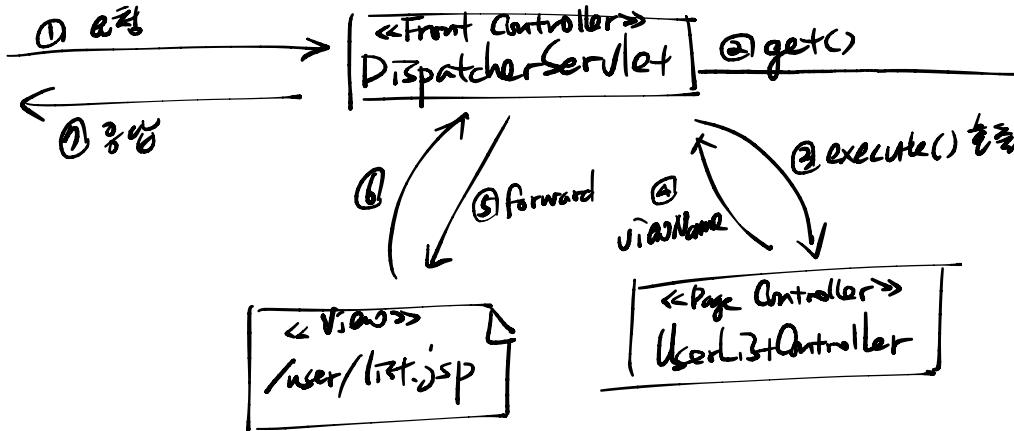
HttpServlet UserListController (UserService us) {  
 userService = us;  
}

void execute(HttpServletRequest request)

=

}

58. ~~제10(2) 칸드풀러~~<sup>2</sup> POJO 3 바꾸기 - 쇠고기 버전



controllerMap : Map<String, Object>

key	value
"/user/list"	UserListController

## 58. 제이지 컨트롤러<sup>2</sup> POJO로 바꾸기 — CRUD 기능 추출하기

\* 현황

\* 개선

UserAddController

UserListController

UserViewController

UserUpdateController

UserDeleteController

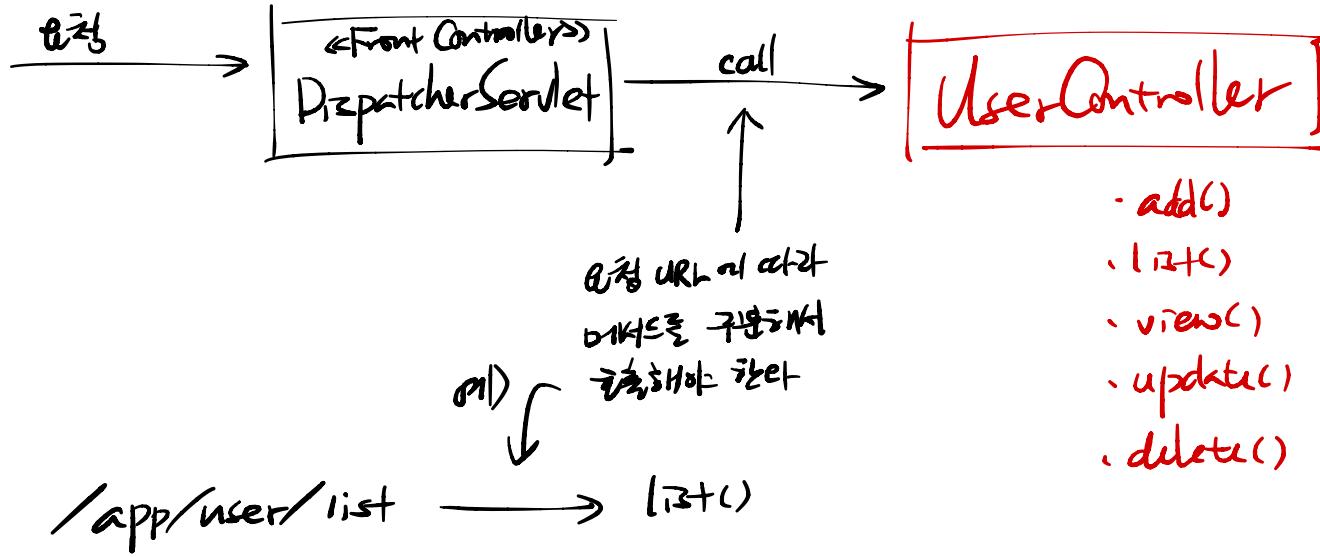


UserController

CRUD 작업  
기능이 분리되었음  
모듈화  
정형화된 코드  
구조화된 코드  
제작 효율화

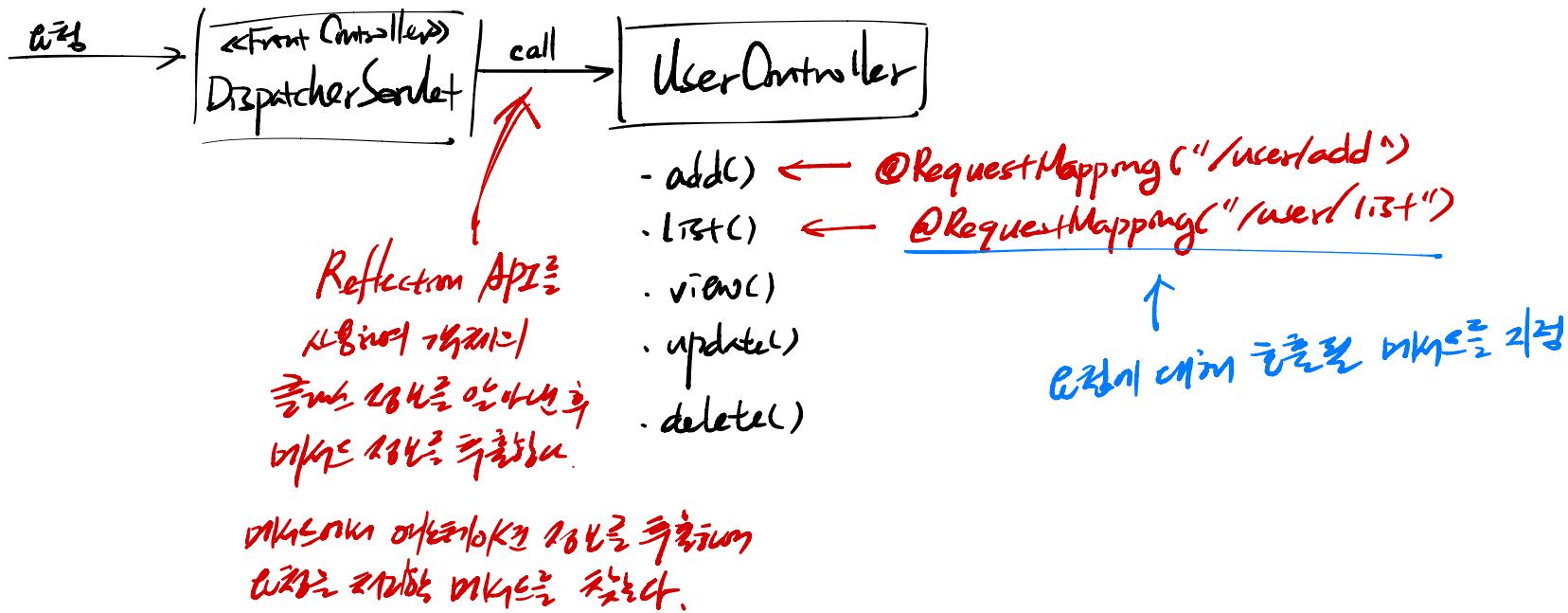
- add()
- list()
- view()
- update()
- delete()

58. 데비지 컨트롤러<sup>2</sup> POJO로 바꾸기 - CRUD 기능 허용하기  $\Rightarrow$  요청에 따라  
서비스 주입은 가능해?

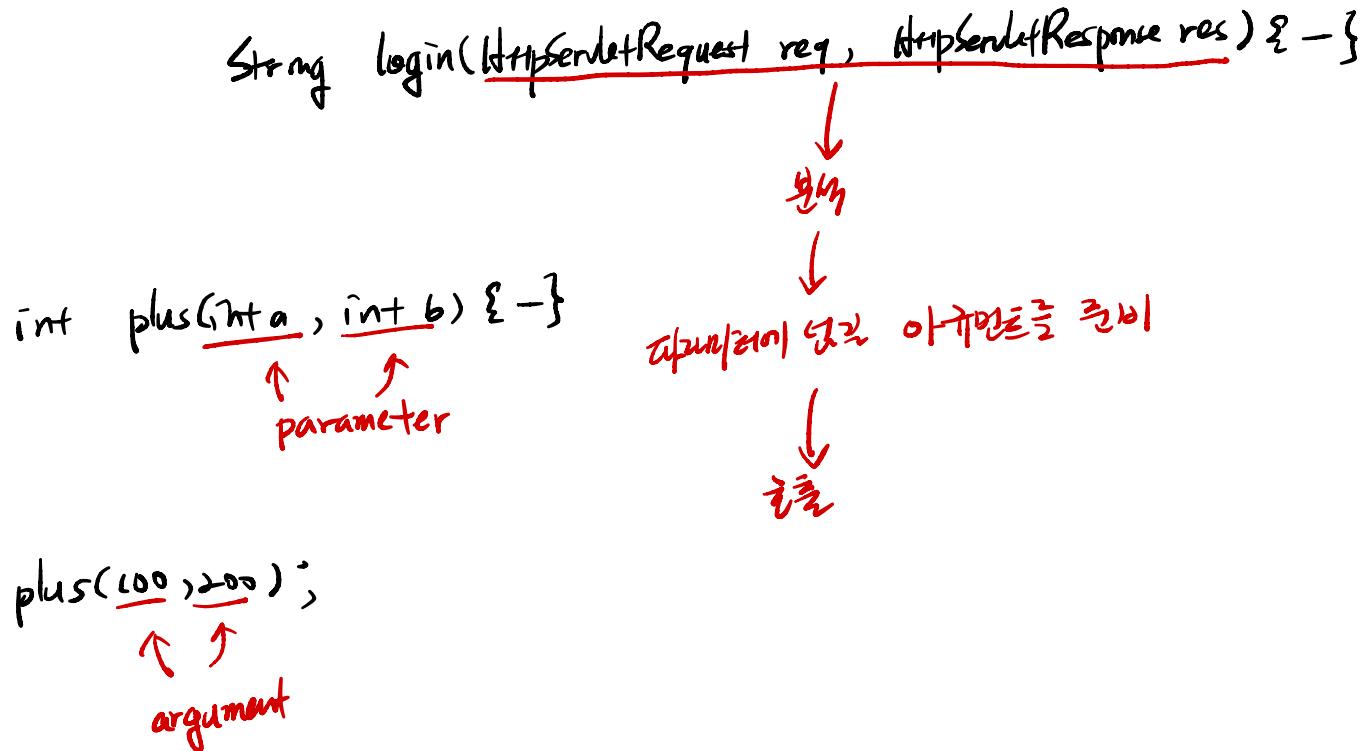


그렇지? 서비스 주입은 가능해!  
서비스 주입은 가능해!

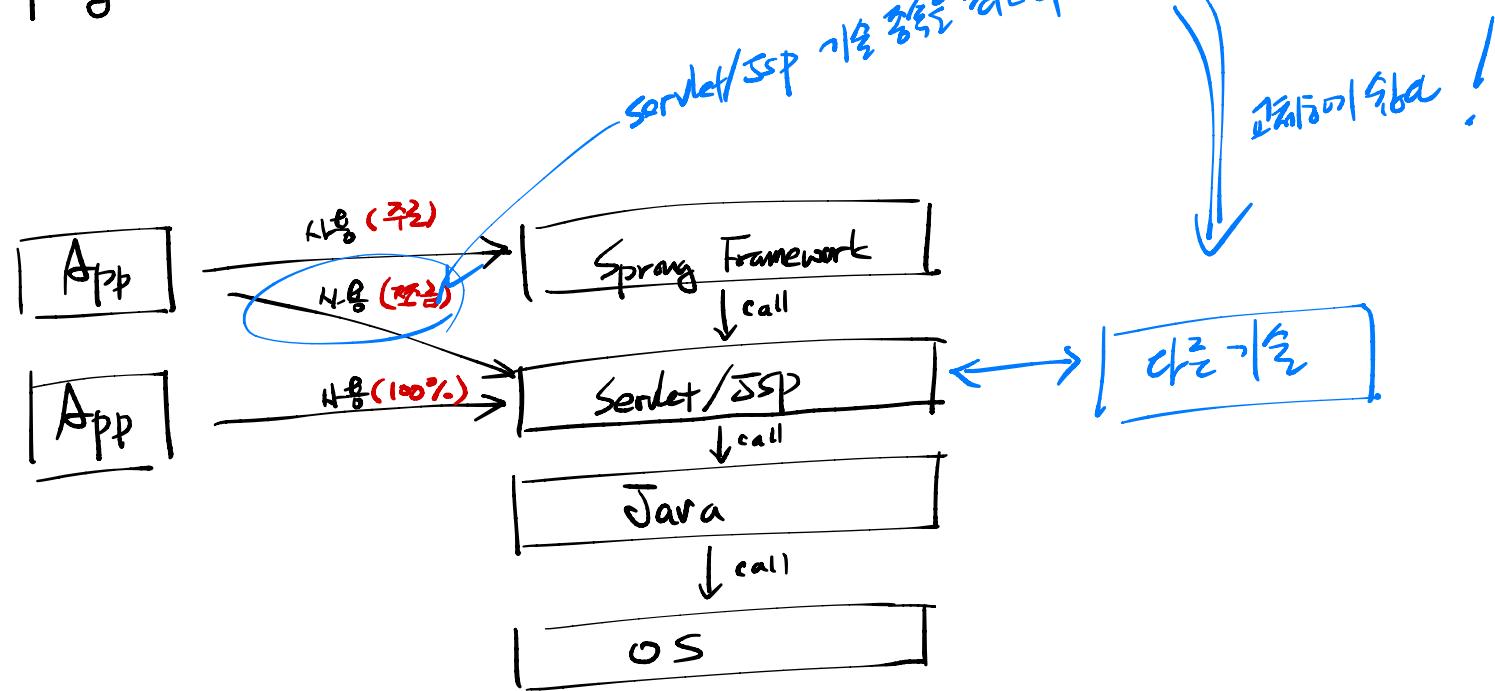
58. 제이비디 컨트롤러 POJO 바꾸기 - CRUD 기능 추가하기 + 어노테이션 활용  
+ Reflection API 활용



59. ⑥ 첨해들려의 파라미터를 HttpServletRequest HttpServletResponse로 변수로 변경하기 = 파라미터 선언이 자주 없다.



\* Spring WebMVC 프레임워크가 처리 기능이 없음



\* Setter ol'unkn ~~uzzet/ai ol'unk o'wruun~~

void setCreatedDate() { - }

0 1 2 3 4

X ↓      ↓

6/22/2012    2012-06-22

"createdDate"

\* 퀴어이언트가 쿼리 문장을 만들고 싶을 때는 이렇게 쓰면 좋다

쿼리문  
name = aaa & email = aaa@test.com & password = 1111 & tel = 010-1111-2222  
값을 넣을



User  
클래스

```
User user = new User();
```

~~user.setNo(10);~~ → "no" 프로퍼티 이름은 있지만 사용하지 않아 초기화

user.setName("aaa"); → "name" " "

user.setEmail("aaa@test.com"); → "email" " "

user.setPassword("1111"); → "password" " "

user.setTel("010-1111-2222"); → "tel" " "

## 60. 헤더에 칸트롤러 사용하는 법

\* 헤더  
→

→ 헤더 칸트롤러가 있는지 없는지 확인

new AuthController(userService)

\* 개선

제이지 칸트롤러 있는지 확인

@Controller

class AuthController {

≡

}

↑

ContextLoaderListener 초기화

@Controller 어노테이션이 있는 클래스를 찾기

이스터스를 찾는 방법

\* 각각의 자동 설정 기능을 사용할 수 있게 Bean의 형태로 설계

### Context Loader Listener

- 스프링 컨테이너를 구현한 추상 클래스
- 부모자 컨테이너 설정 가능

설계

### Application Context

\* IoC (Inversion of Control)  
↳ 생명의 흐름은 예외

- 1) Dependency Injection (의존 관계 주입)
- 2) Event Handler (이벤트 핸들러)

↑  
- 구현을 자동 설정하는 역할을 함. ⇒ "bean container"  
- 구현의 의존 관계를 자동 주입 ⇒ "DI container"  
↓  
"IoC container"

\* ref<sub>2</sub> makes ref<sub>1</sub> final when ref<sub>2</sub> is not final

ref<sub>2</sub> makes ref<sub>1</sub>

@Bean

public DaoFactory createDaoFactory(SqlSessionFactory sf) {

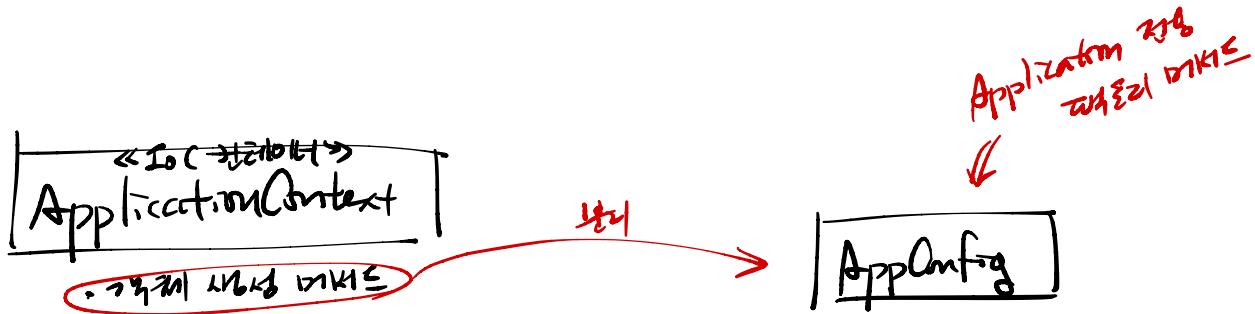
=====  
}



@Bean initializes the value

return a new object.

\* IoC 컨테이너에서 프로젝트용 코드 불러오기



\* 퀼드로 ~~설정~~ 커스텀러를 넣을 때 자동으로 처리해줄 수 있도록  
설정하기

설정하는

방법

@WebServlet("/app/\*")

```
class DispatcherServlet  
extends HttpServlet{}
```

==  
}

DispatcherServlet s =

new DispatcherServlet(...);

serverContext.addServlet("app", s);

ServletContext → 서버

:

\* XML 방식은 등록하는 방식

① XML 설정

< servlet >  
=   
< /servlet >

② Java Config

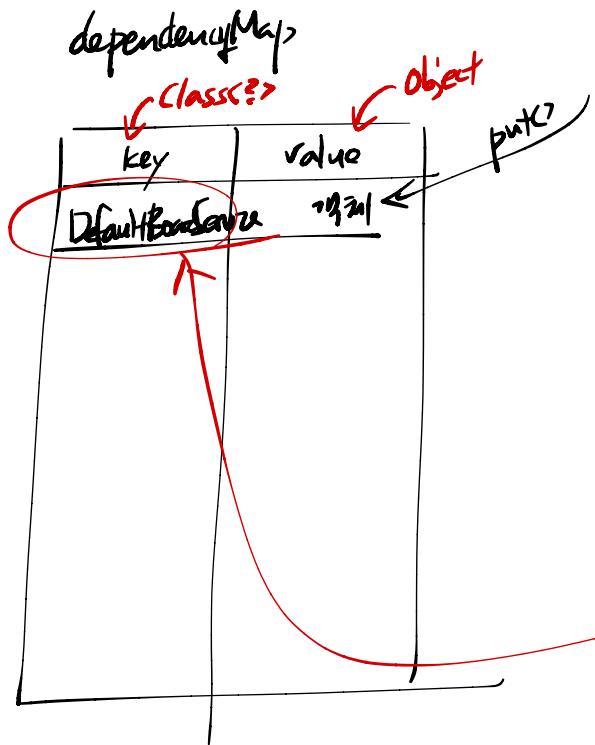
Annotation 사용

@WebServlet("")  
class MyServlet {  
=   
MyServlet s;

제작자 등록

MyServlet s = new MyServlet();  
servletContext.addServlet("my", s);  
:

\*  $\text{DefaultBoardService} \rightarrow \text{BoardController}$



new DefaultBoardService()

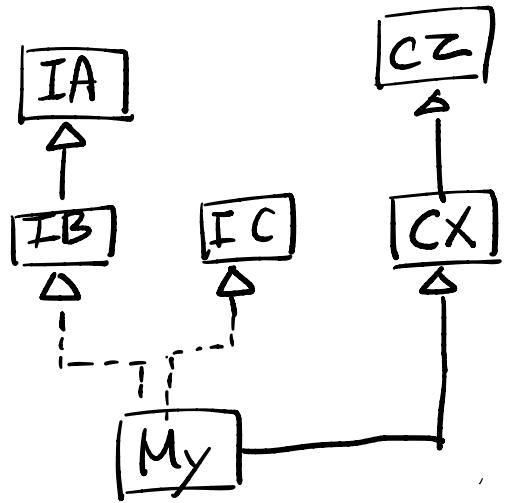
BoardController

new  $\rightarrow$  BoardController(BoardService s, ...)

Is it?

obj. IsInstance(>BoardController)

\* `isInstance()`



`My obj = new My();`

`IA.isInstance(obj)`  $\Rightarrow$  true

`IB.isInstance(obj)`  $\Rightarrow$  true

`IC.isInstance(obj)`  $\Rightarrow$  true

`CZ.isInstance(obj)`  $\Rightarrow$  true

`CX.isInstance(obj)`  $\Rightarrow$  true

`My.isInstance(obj)`  $\Rightarrow$  true