5가. 파일 업로드

* 클라이언트

POST /board/add HTTP/1.1 ← request line

⋮

Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW

↑ <form enctype="multipart/form-data" ...>

⋮

<input type='file' ...>

⋮

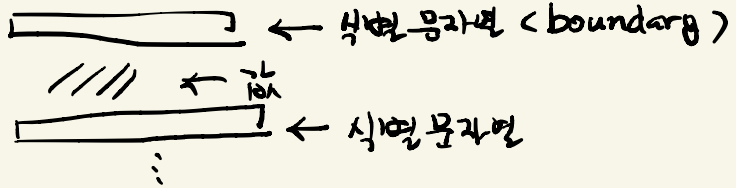
</form>

↑ 파라미터 구별 문자열

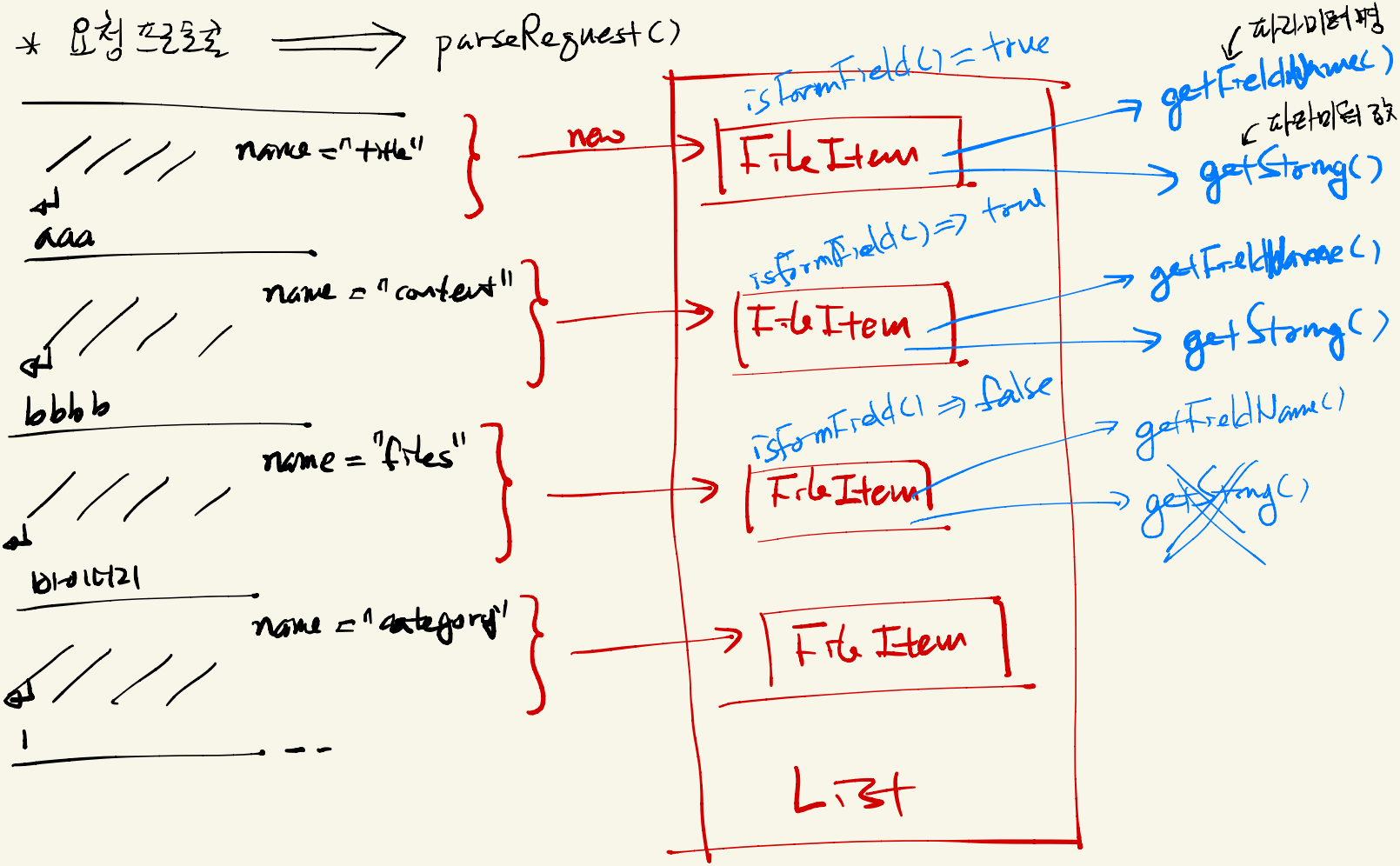
header

⋮

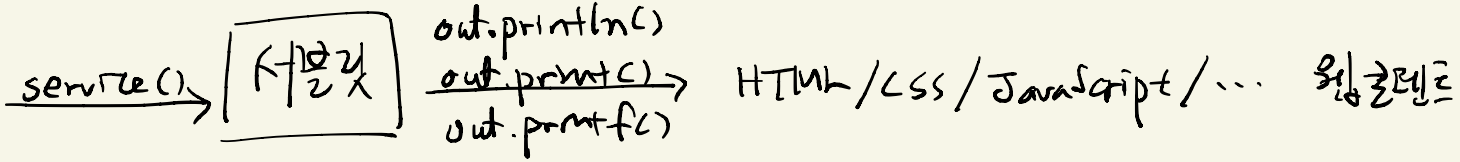
↵

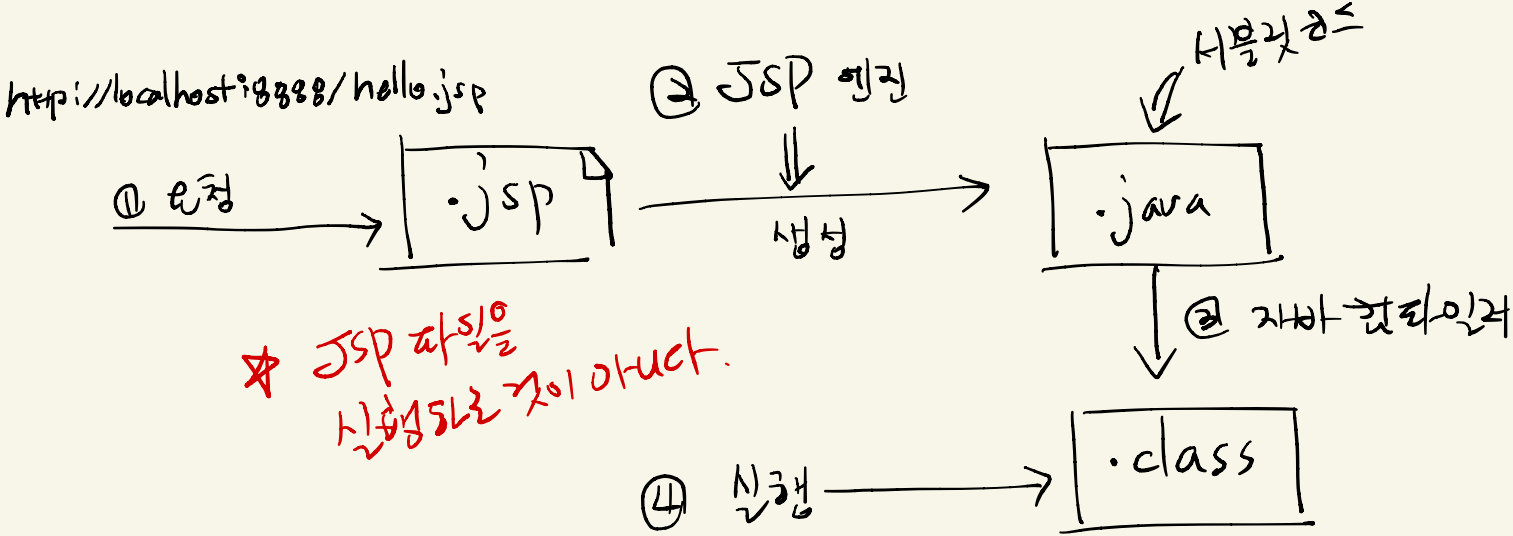


message-body


* 요청 분석 ⟹ parseRequest()
name="title"
aaa
name="content"
bbbb
name="files"
name="category"
new
isFormField() = true
FileItem
isFormField() ⇒ true
FileItem
isFormField() ⇒ false
FileItem
FileItem
List
파라미터명
getFieldName()
파라미터 값
getString()
getFieldName()
getString()
getFieldName()
getString()

* JSP → 서블릿로 자동 생성!





* JSP 파일을 실행하는 것이 아니다.

* JSP 파일 위치

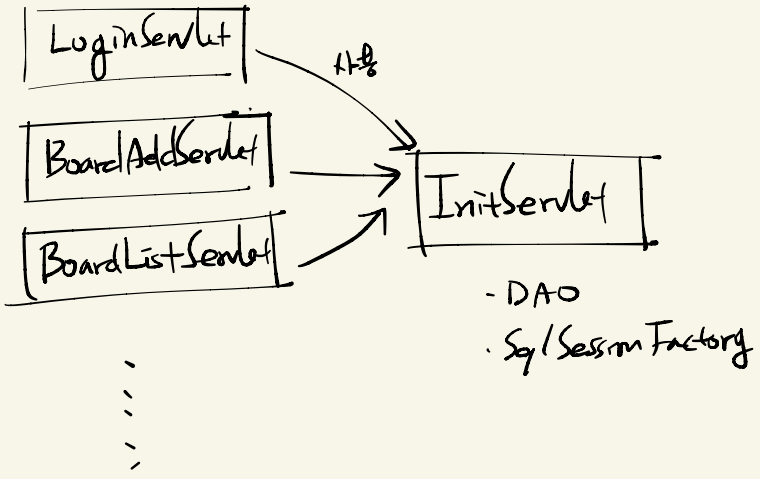
src / main / webapp / *.jsp

↓ 자동생성

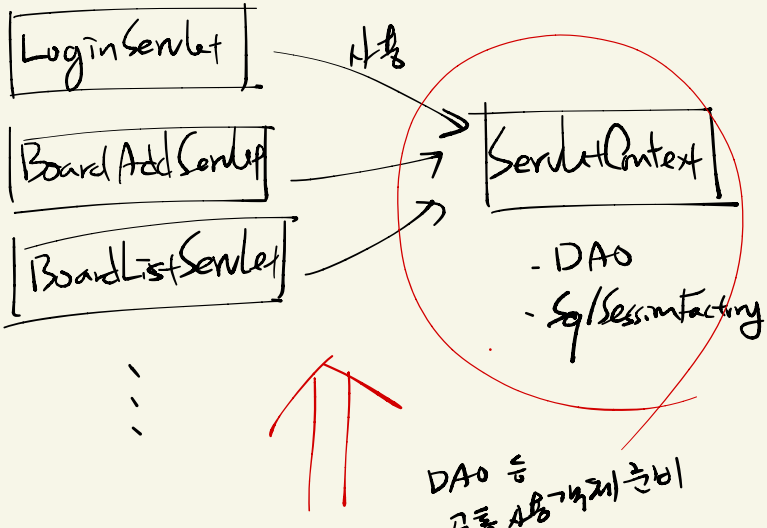
... /temp / *.java

* 63. 커스텀 ServletContext 만들기

① 현황 → ② InitServlet 중복성 제거

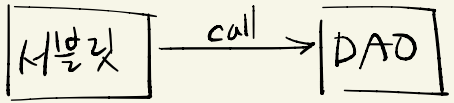


* 문제점
- 여러개의 서블릿이 InitServlet 클래스에 중복된다.

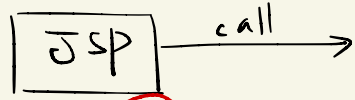


* 64. MVC 모델 1

① 이전방식 ⟶ ② 변경



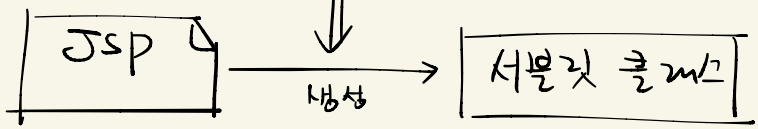
- 입/출력
- 업무 흐름 제어
- 데이터 처리



- 입/출력
- 업무 흐름 제어
- 데이터 처리

* JSP

서블릿 클래스
Generator
=
JSP Engine



- 탬플릿 데이터 ⟶ 출력문
- JSP Action tag ⟶ 자바코드
- scriptlet element ⟶ 자바코드
- ⋮

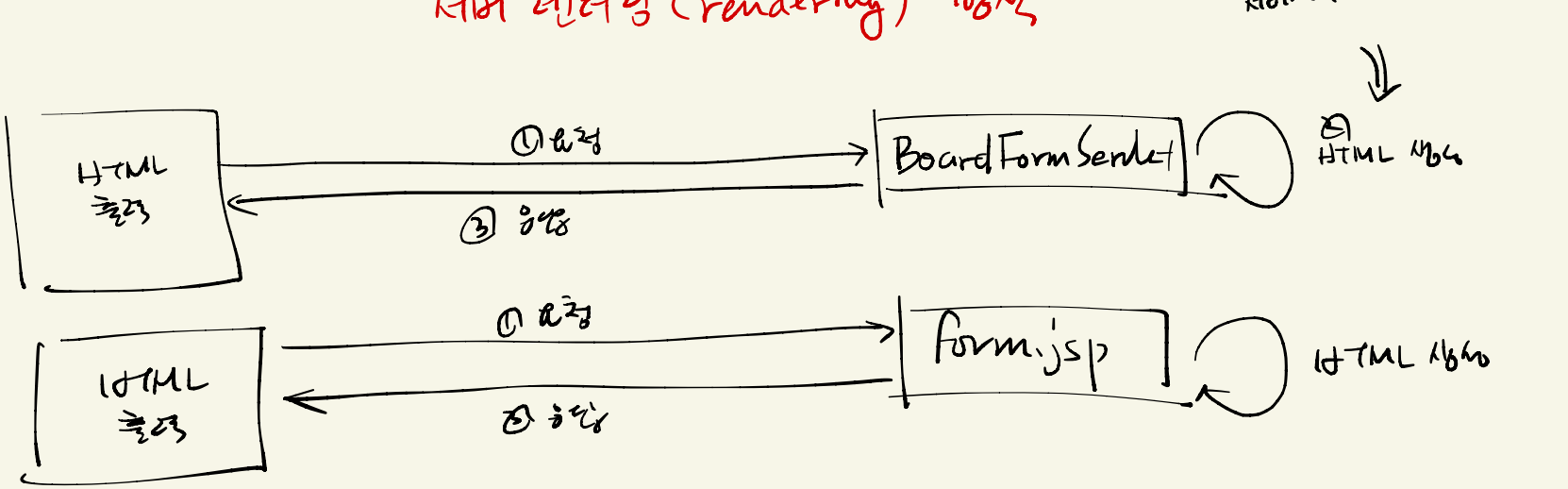
⟹ 워킹이미지를 생성해야 하는 개발 부담을 감소!

* 이전 방식과의 차이점

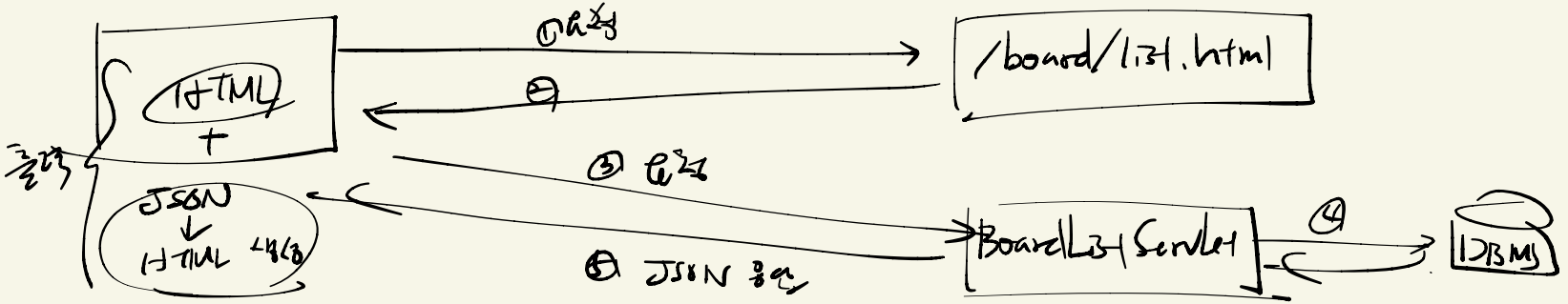
- JSP 기술을 사용해서 클라이언트 출력을 보다 쉽게

클라이언트에서 비중이 높다.

서버 렌더링 (rendering) 방식



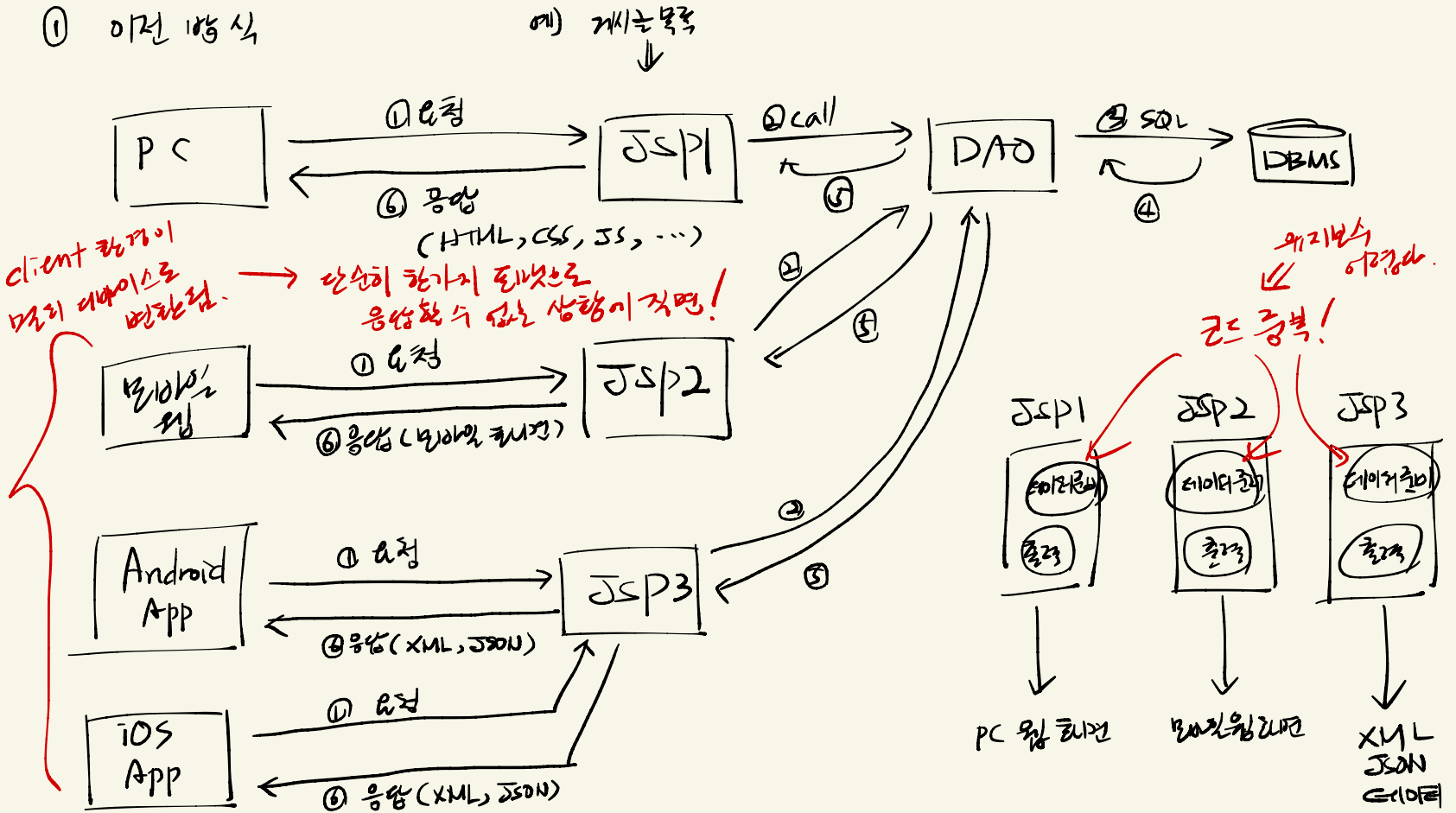
클라이언트 렌더링 방식



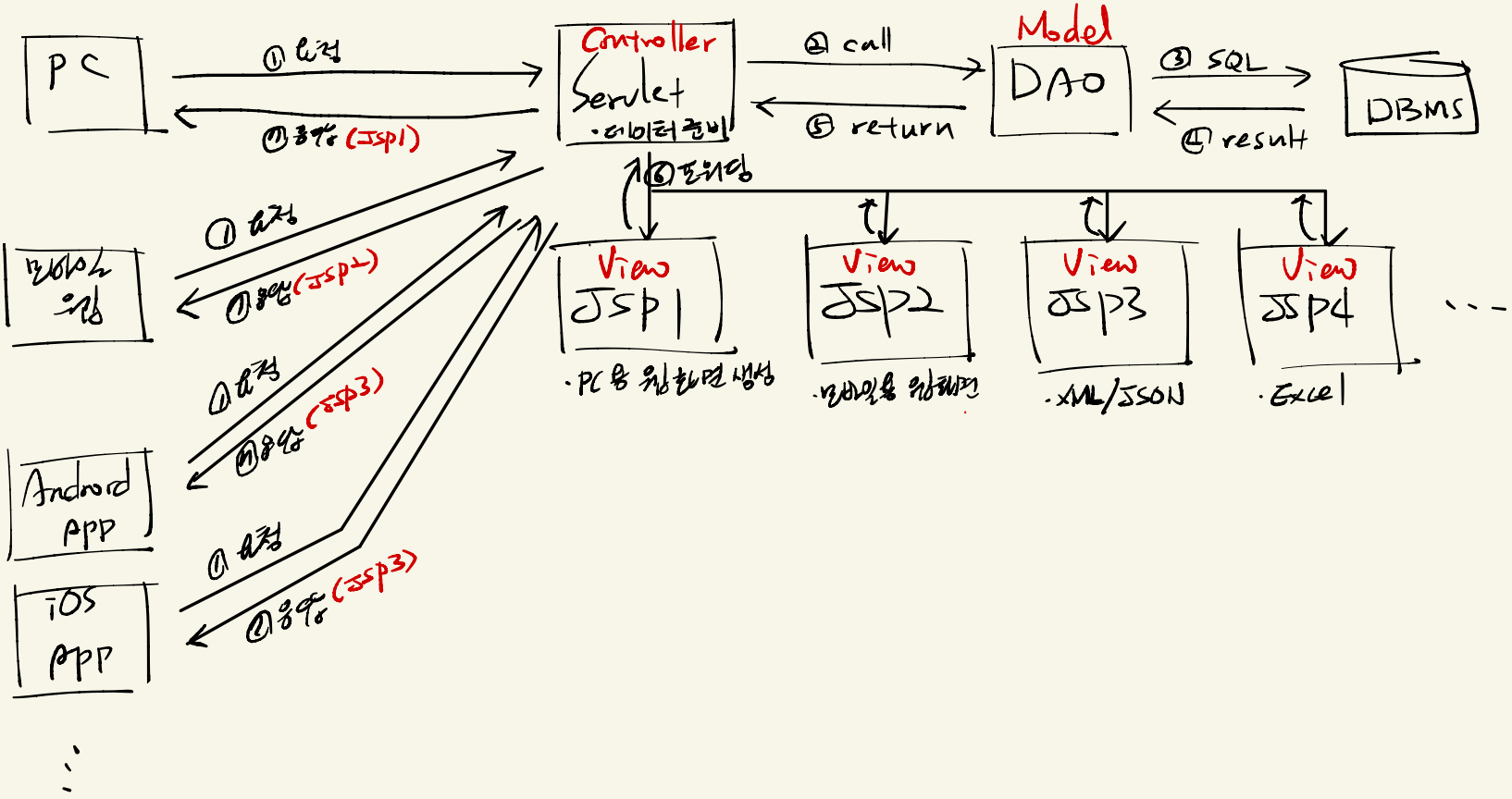
65. MVC 모델 2

① 이전 방식

예) 게시판 목록

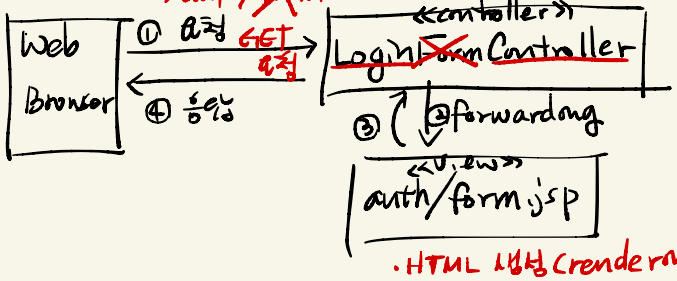


② 개선 내용


PC
① 요청
⑪ 응답 (JSP1)
Controller
Servlet
·데이터 준비
② call
⑤ return
Model
DAO
③ SQL
④ result
DBMS
⑥ 포워딩
① 요청
⑩ 응답 (JSP2)
안드로이드 앱
① 요청
⑩ 응답 (JSP3)
Android App
① 요청
⑩ 응답 (JSP3)
iOS App
View
JSP1
·PC용 웹화면 생성
View
JSP2
·안드로이드용 화면
View
JSP3
·XML/JSON
View
JSP4
·Excel
...

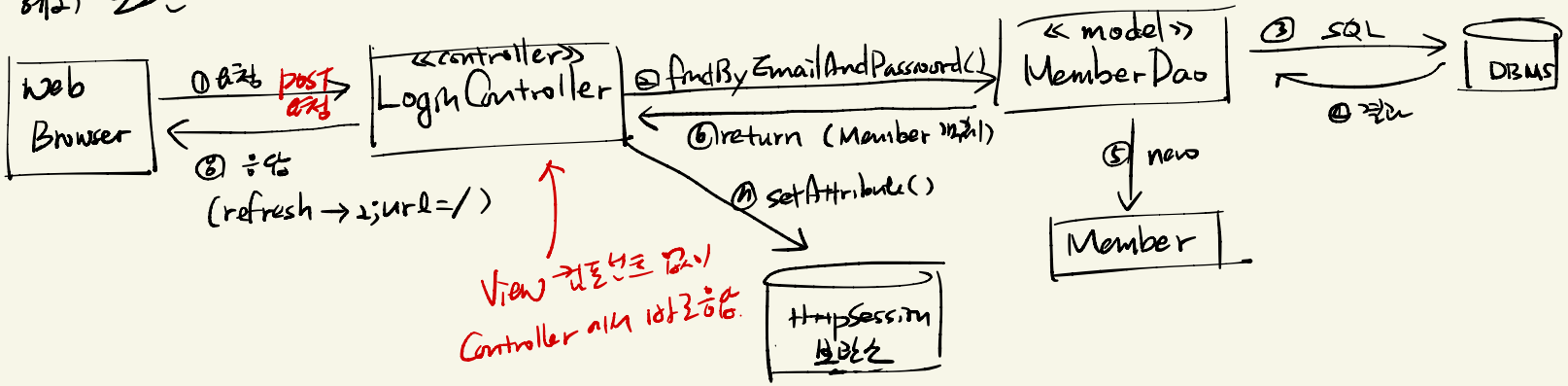
예1) 로그인폼

/auth/~~form~~ login

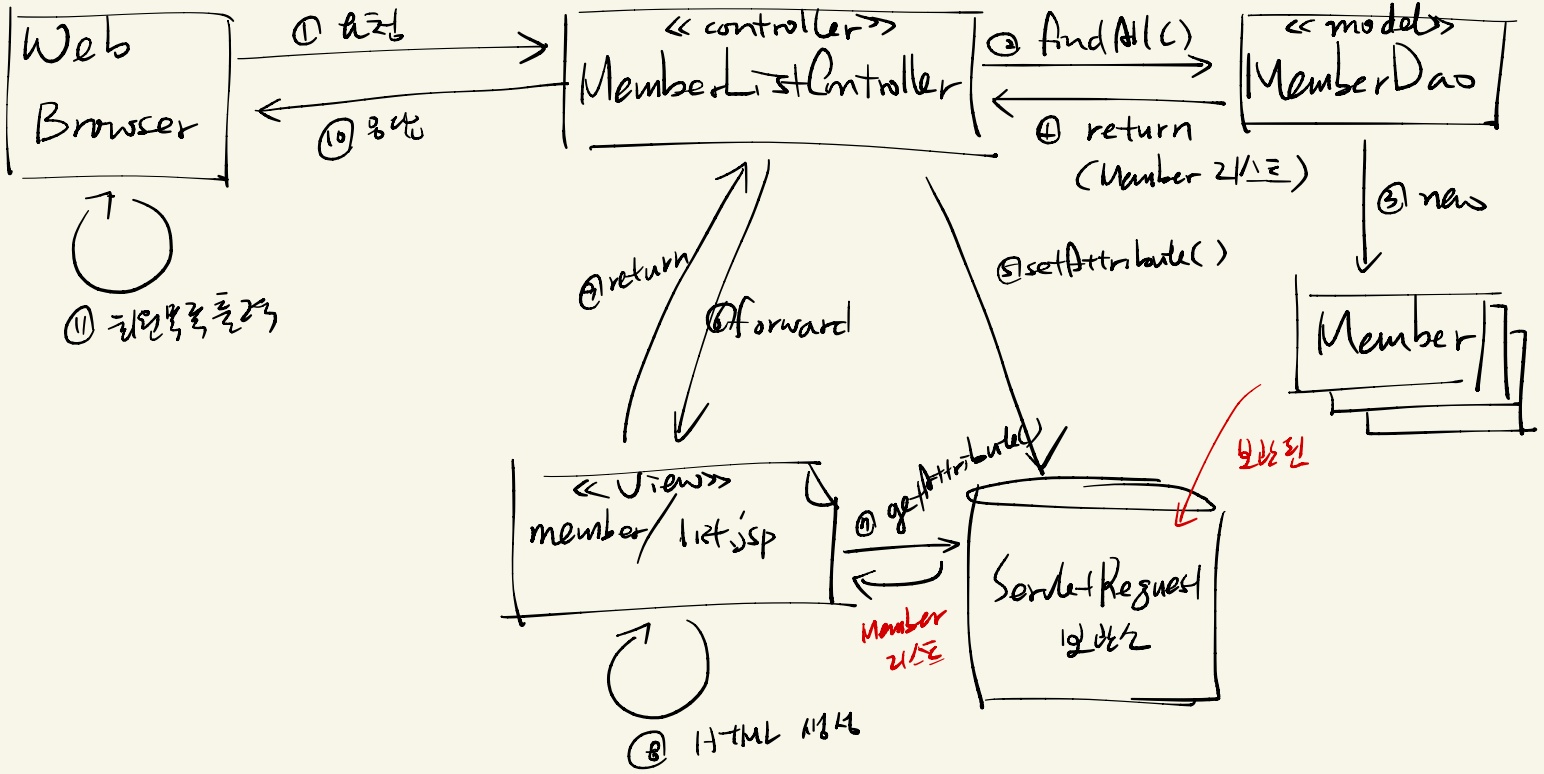


Controller가 하는 일이 많아도 프로그래밍의 일관성을 위해 응답은 servlet이 담당한다!

예2) 로그인



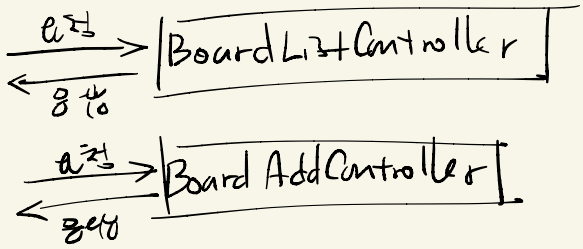
예3) 회원 목록 조회


Web Browser
① 요청
⑩ 응답
⑪ 회원목록클릭
<< controller >>
MemberListController
② findAll()
④ return
(Member 리스트)
<< model >>
MemberDao
③ new
Member
⑤ setAttribute()
⑥ forward
⑦ return
<< View >>
member/list.jsp
⑧ HTML 생성
⑨ getAttribute()
ServletRequest
Member 리스트
보관된

66. Front Controller 만들기 : GoF의 Facade 패턴

= front

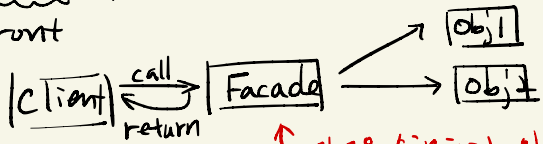
① 이전방식



각 Controller에 대해 개별적으로 a링 ⇒

- 데이터 준비
- View 포워딩
- 데이터 포워딩
- 리다이렉트
- 리프레시

각 Controller 마다 중복코드 작성해야 한다



↑ 작업을 수행하기 위해 어떤 객체를 어떤순서로 어떻게 사용해야 하는지 감춘다 = "encapsulation" (캡슐화)

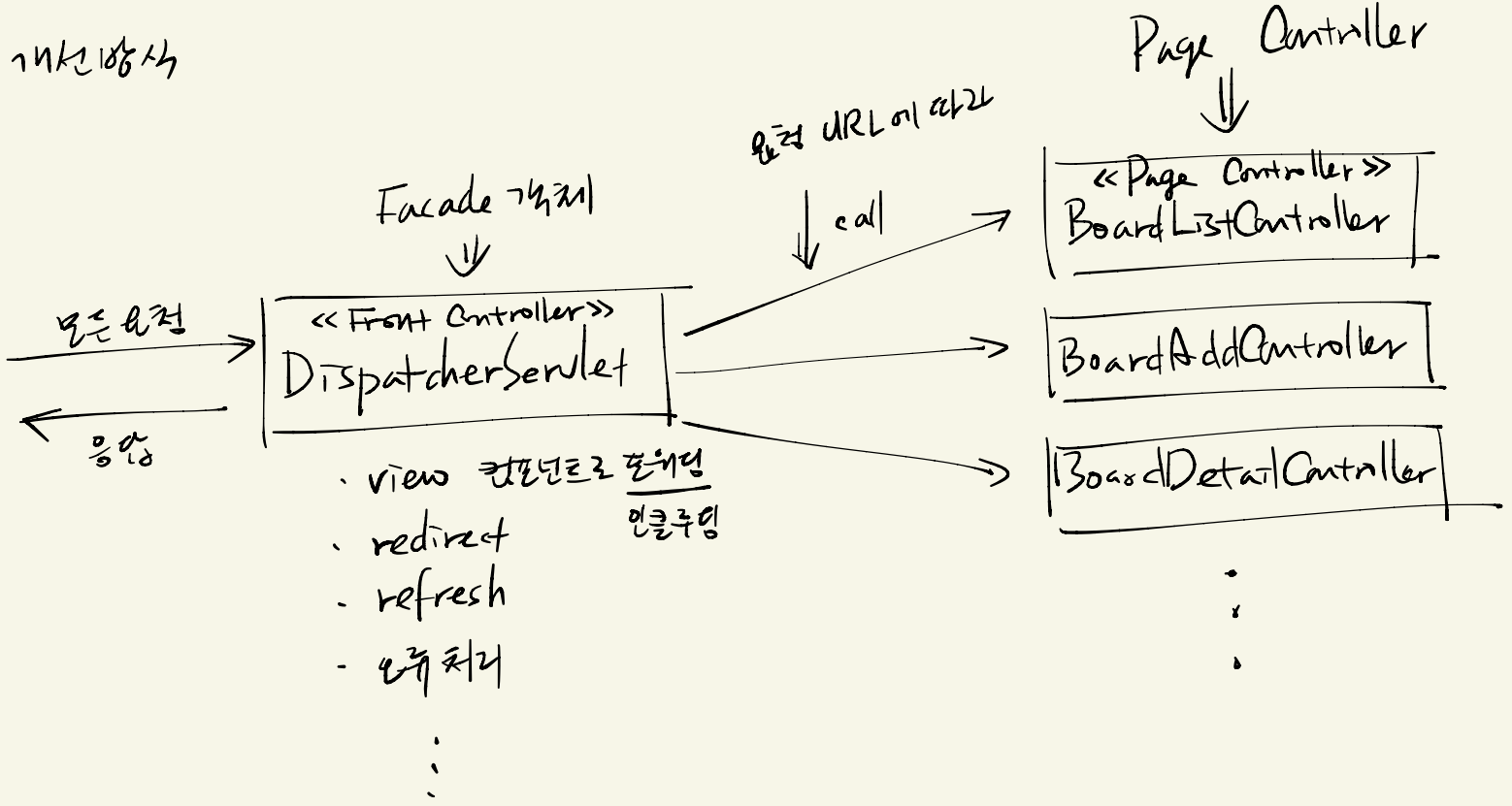
⇓

호출자 입장에서는 복잡한 로직(실행순서)을 알 필요가 없이 편하다

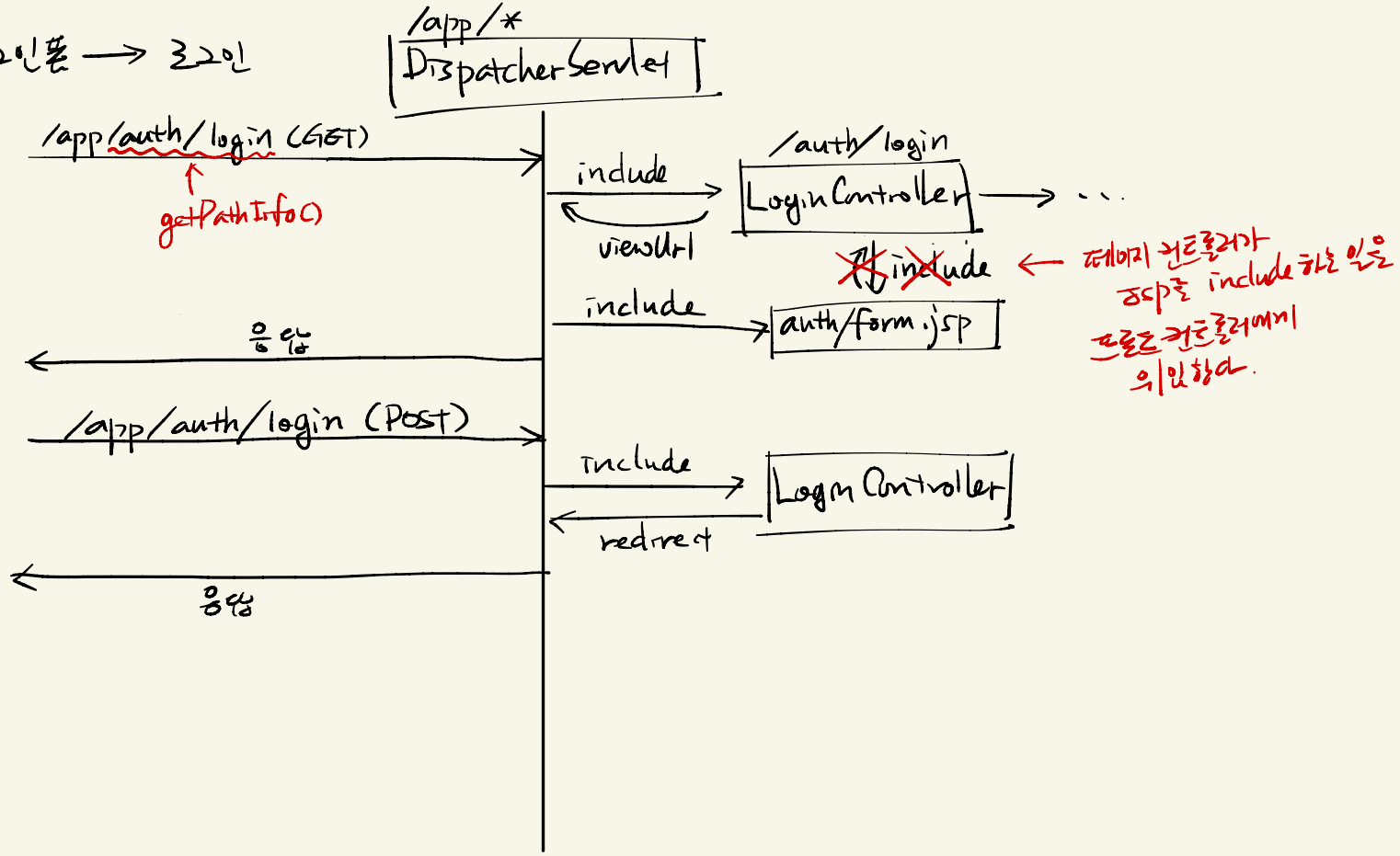
⇓

만약 작업의 흐름이 변경되더라도 호출자에게 영향을 끼치지 않는다.

② 개선방식

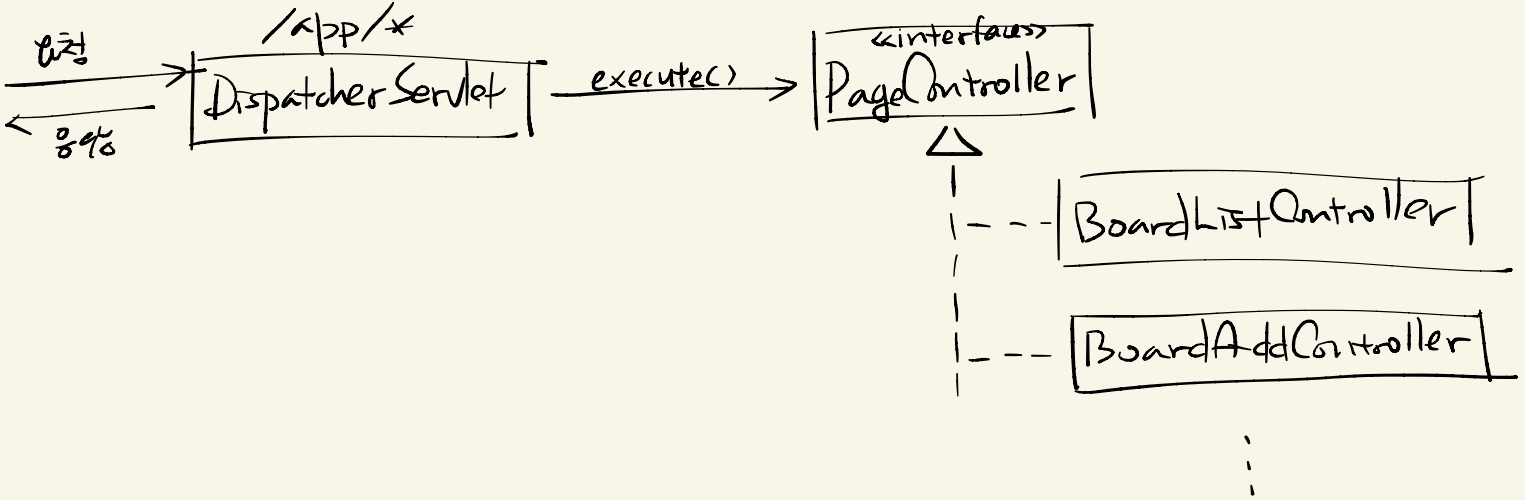

Page Controller
요청 URL에 따라
Facade 객체
call
<< Page Controller >>
BoardListController
모든 요청
<< Front Controller >>
DispatcherServlet
BoardAddController
응답
BoardDetailController
. view 컴포넌트로 포워딩
인클루딩
. redirect
. refresh
. 오류처리

예) 로그인폼 → 로그인

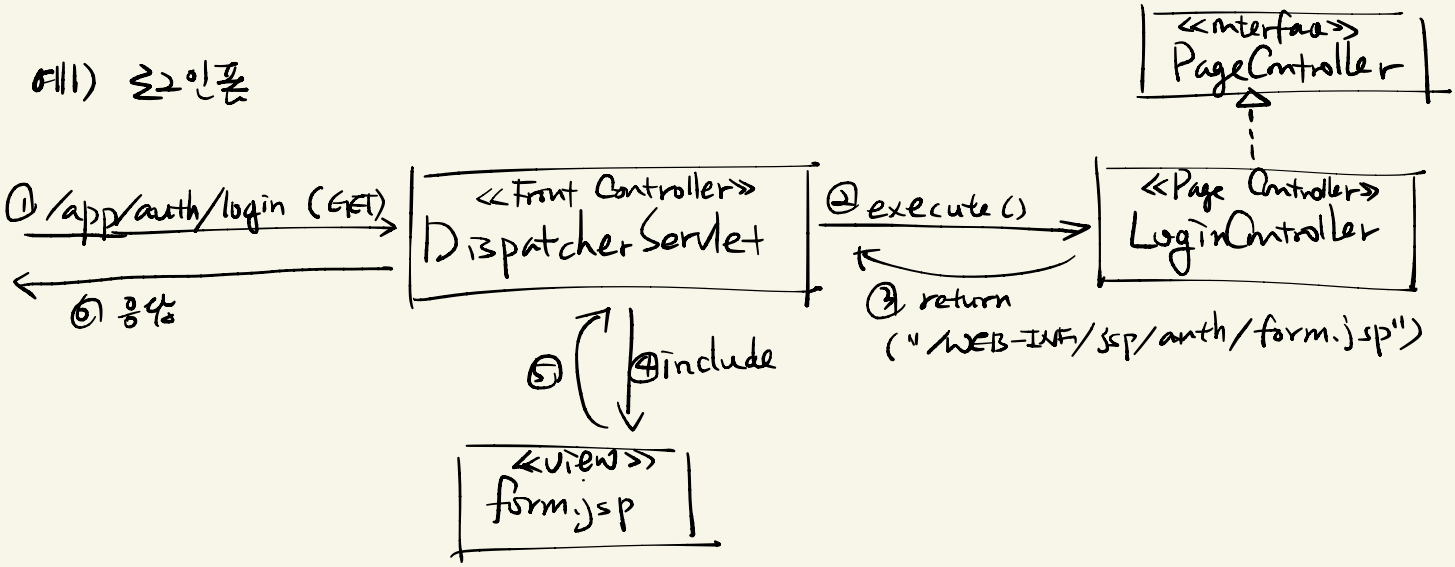


~~include~~ ← 페이지 컨트롤러가 JSP를 include하는 것은 프론트 컨트롤러에게 위임한다.

6가. 페이지 컨트롤러를 POJO로 전환
(Plain Old Java Object)

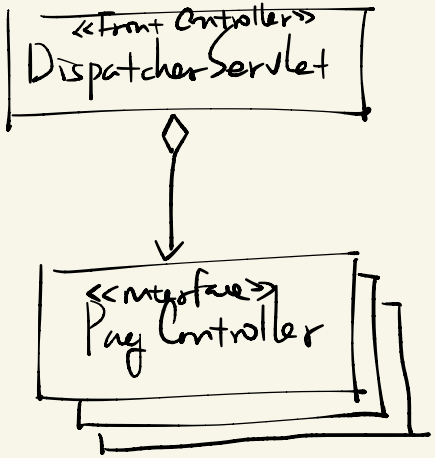


예시) 로그인폼

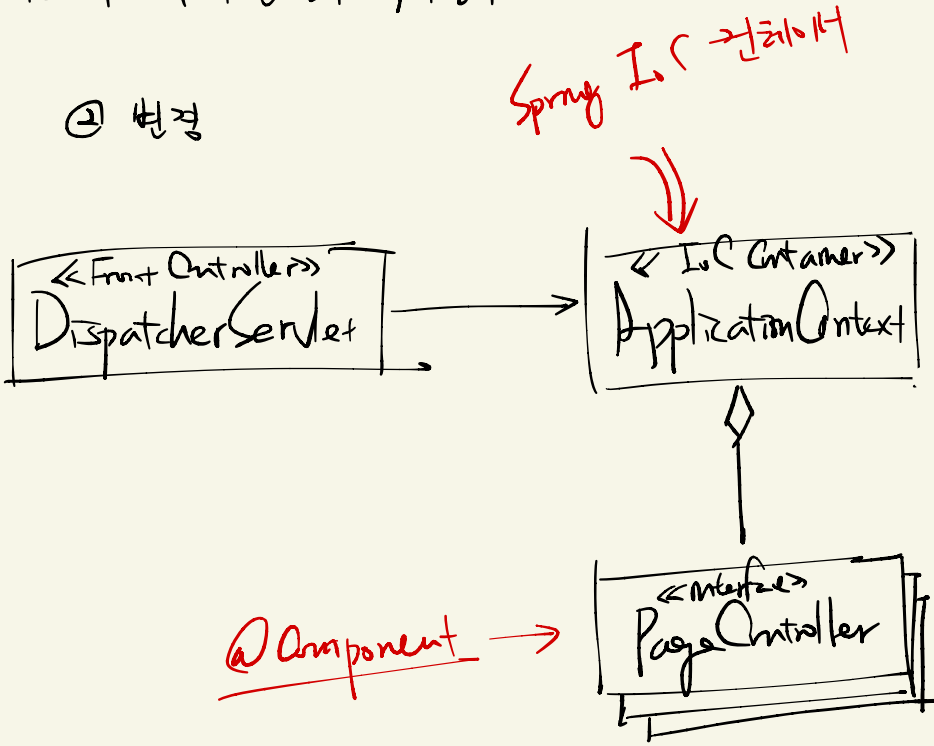

«interface»
PageController
«Front Controller»
DispatcherServlet
«Page Controller»
LoginController
① /app/auth/login (GET)
② execute()
③ return
("/WEB-INF/jsp/auth/form.jsp")
④ include
⑤
⑥ 응답
«view»
form.jsp

68. Spring IoC 컨테이너를 이용해서 페이지 컨트롤러 객체 관리

① 이전 방식

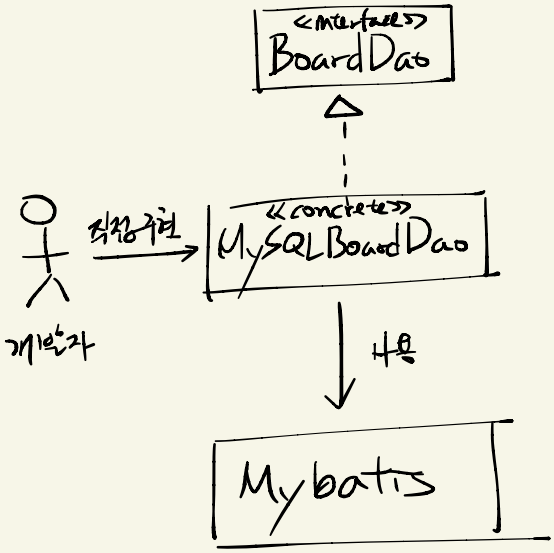


② 변경

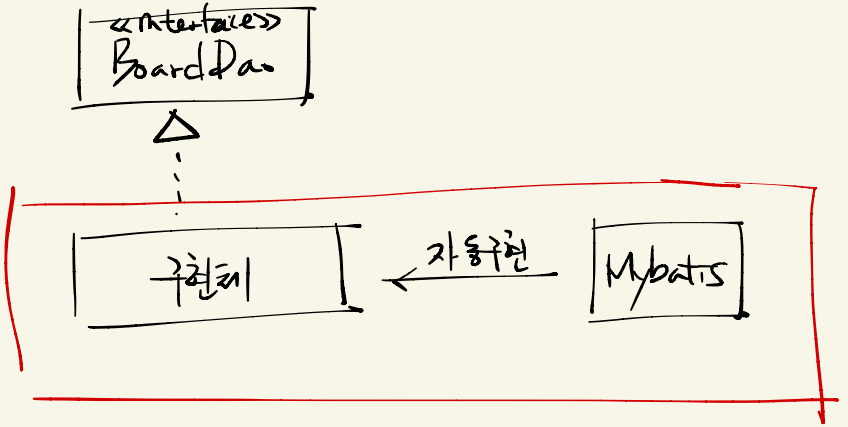


69. Mybatis + Spring IoC 컨테이너

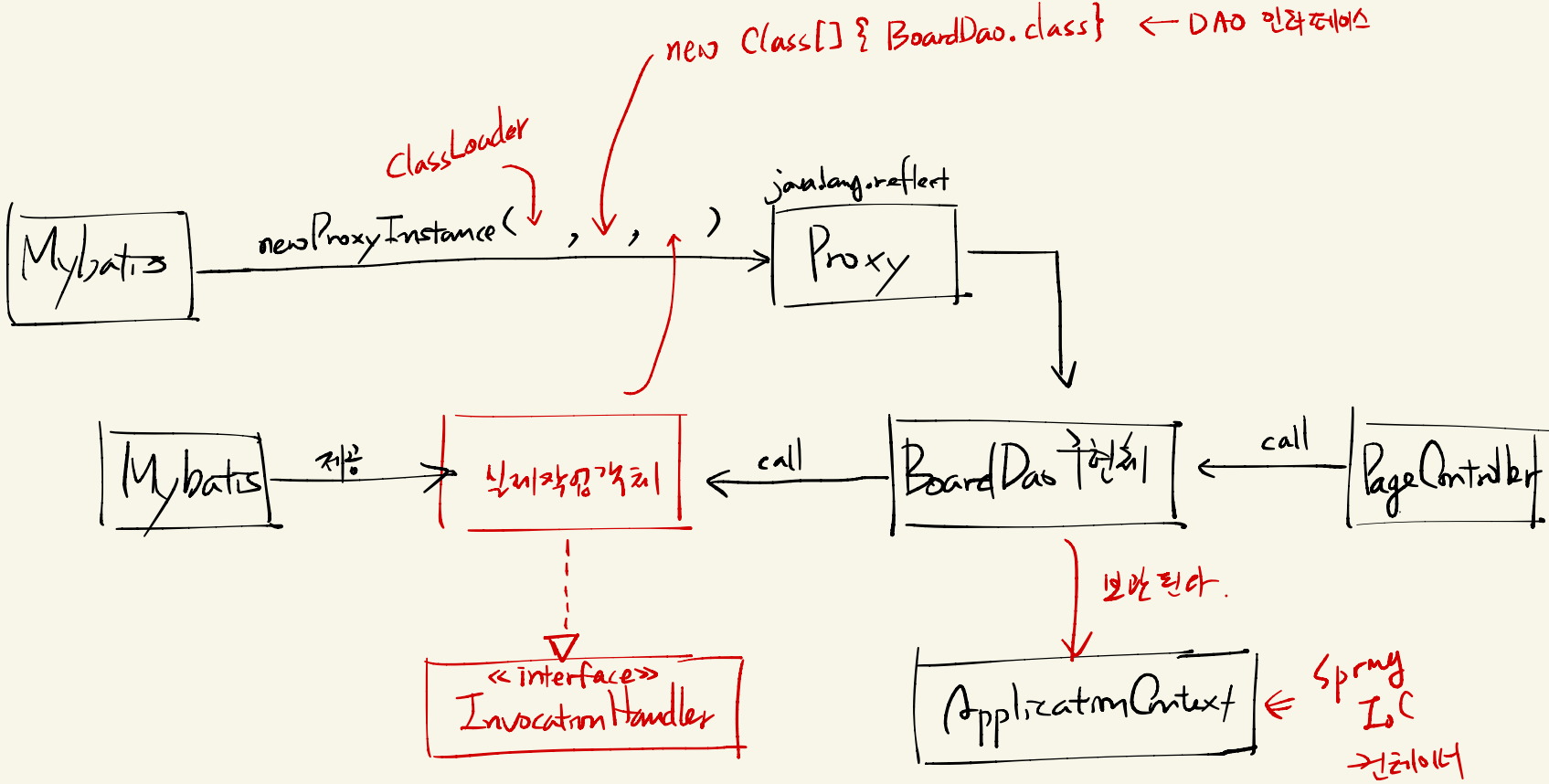
① 이전방식



② 변경

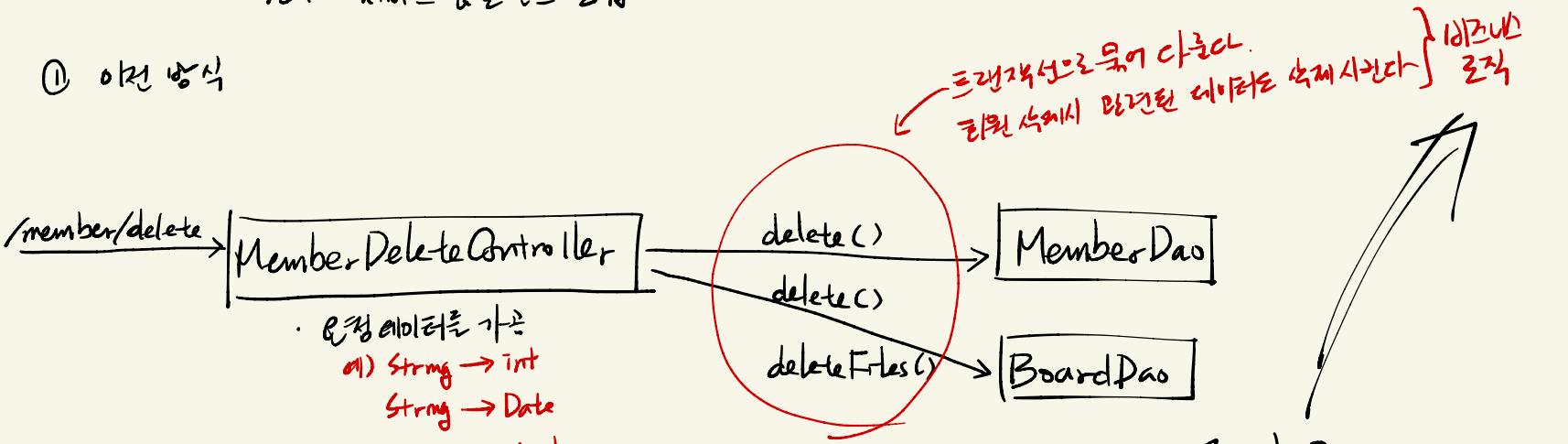


* Mybatis 가 DAO 구현체를 자동생성하는 원리



10. 서비스 컴포넌트 도입

① 이전 방식



- 요청 데이터를 가공
  - 예) String → int
    String → Date
    String → Member
    멀티파트 → 파일로 저장
- 응답 데이터 가공
  - 예) DB 데이터 → HTML
    〃 → JSON
    〃 → XML
    〃 → Excel
- 비즈니스 로직 제어
  - DAO를 사용해 데이터를 준비/처리
  - 트랜잭션 제어

고객사 요구에 따라 Controller를 변경해야 한다 ⇓ 유지보수를 어렵게 한다!

문제?

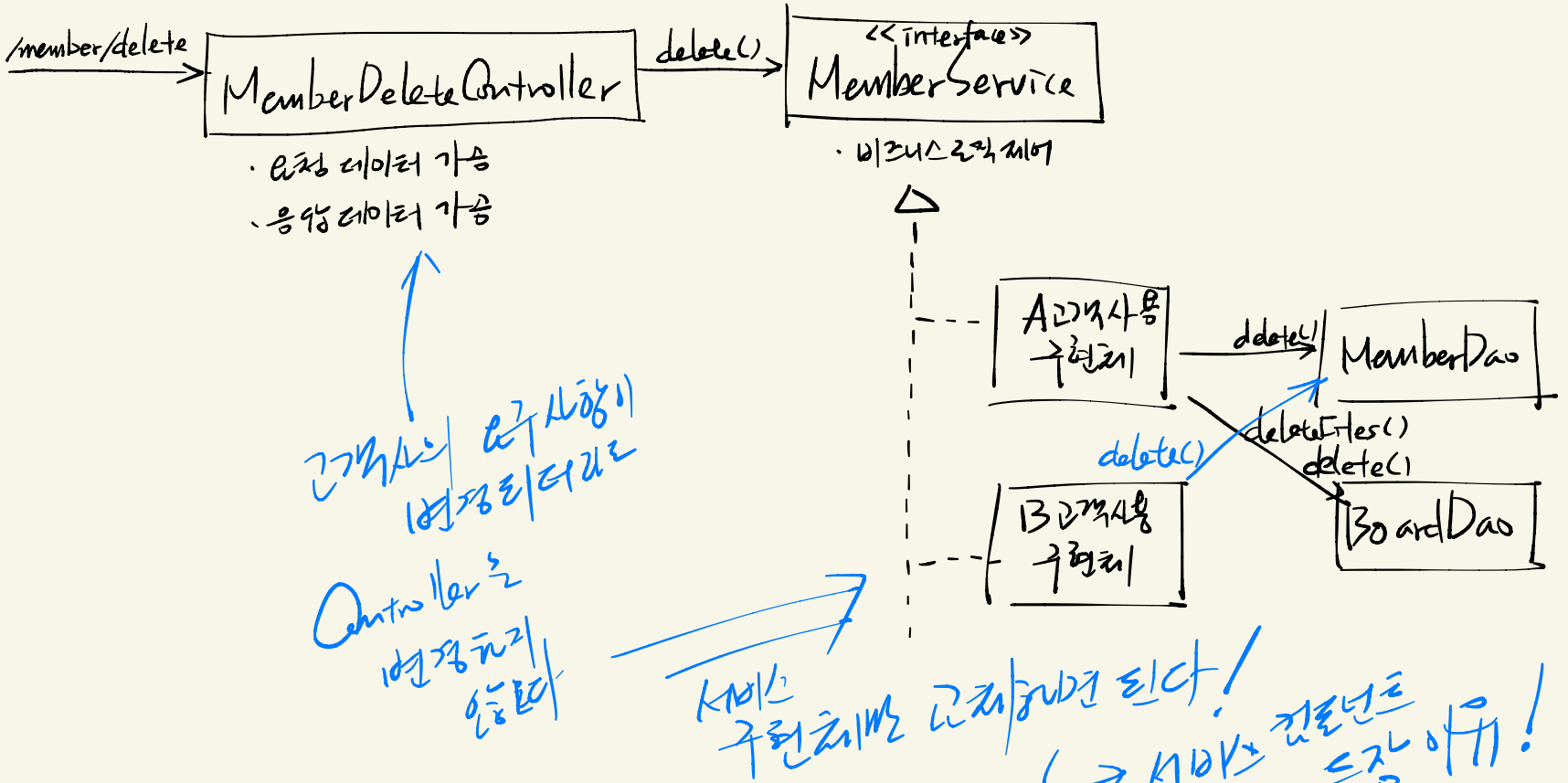
- 고객사 마다 요구사항이 다를 수 있다

A고객사 → 회원과 관련된 모든 데이터 삭제를 원한다

B고객사 → 회원 데이터만 삭제하면 된다

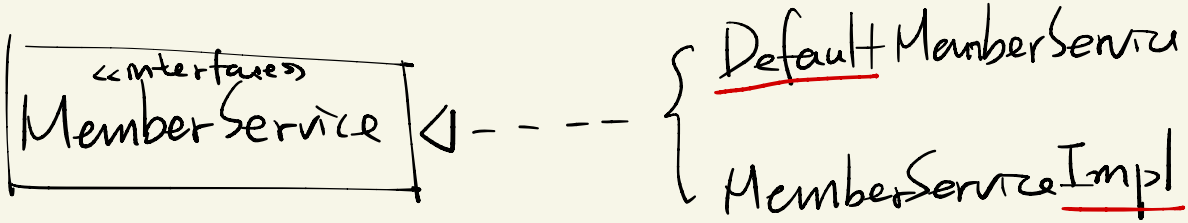
↳ 해결책! 비즈니스 로직 분리

② 변경

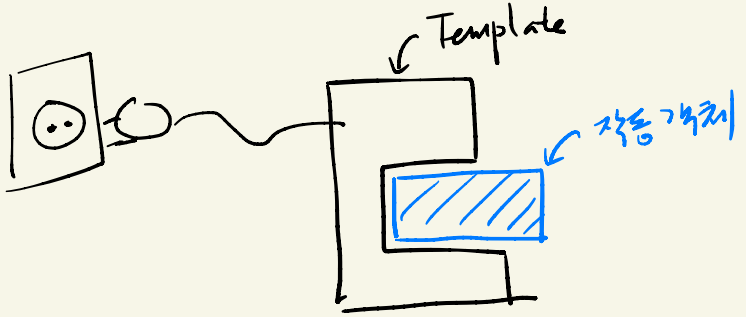

/member/delete
MemberDeleteController
delete()
<<interface>>
MemberService
· 비즈니스로직 제어
· 요청 데이터 가공
· 응답 데이터 가공
A고객사용 구현체
B고객사용 구현체
delete()
MemberDao
deleteFiles()
delete()
BoardDao
고객사는 요구사항이 변경되더라도
Controller는 변경되지 않는다
서비스 구현체만 고치면 된다!
↳ 서비스 컴포넌트 등장 이유!

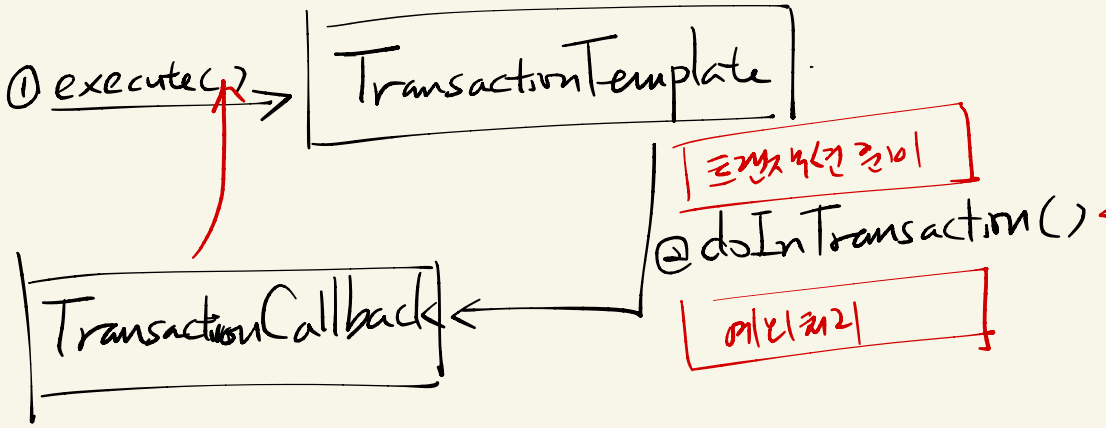
* 서비스 컴포넌트 예

인터페이스 구현체 이름 예



1. TransactionTemplate 클래스의 역할 및 사용법

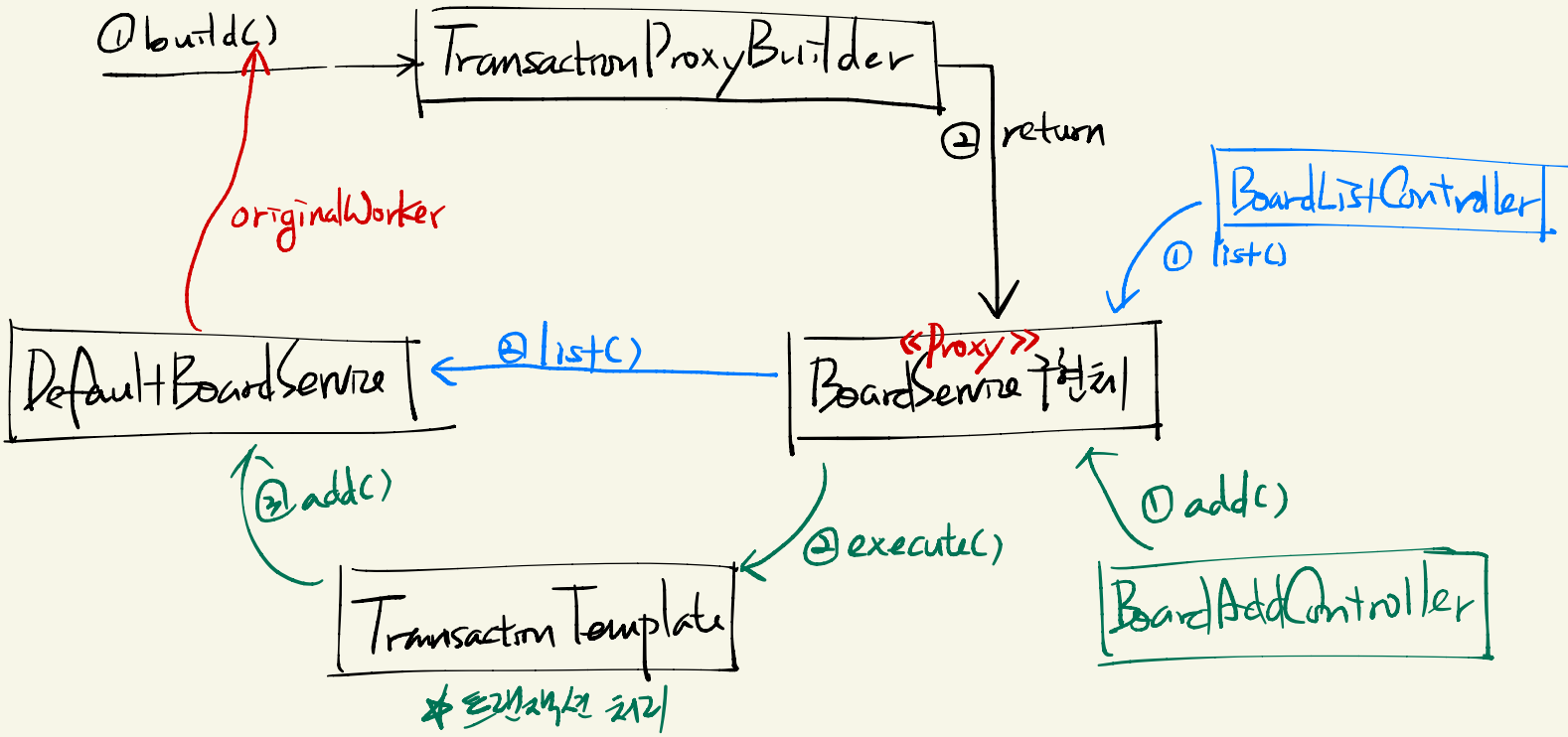




트랜잭션을 처리하는 코드를 중복 작성할 필요가 없다!

개발자는 단지 Transaction 상에서 수행할 작업에만 집중하면 된다.

03. Proxy 객체로 트랜잭션 처리 코드를 캡슐화 하기



14. @Transactional 애노테이션

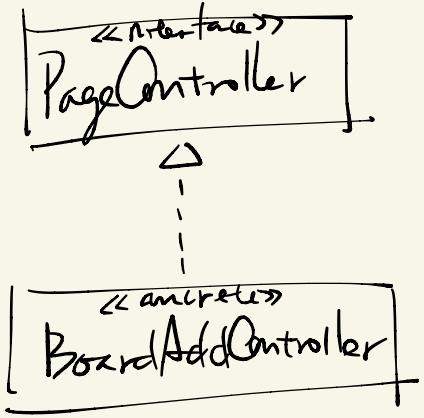
@EnableTransactionManagement

↓ 생성 및 등록

트랜잭션 처리 관련 객체들 ⟶ @Transactional 애노테이션 처리

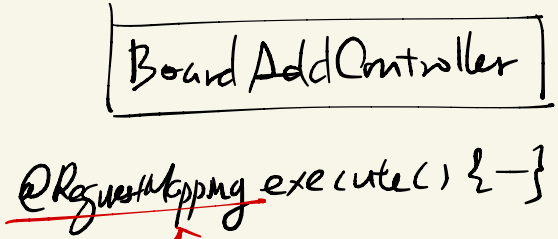
05. @RequestMapping 어노테이션

① 이전 방식



인터페이스를 구현하여
페이지 컨트롤러 생성

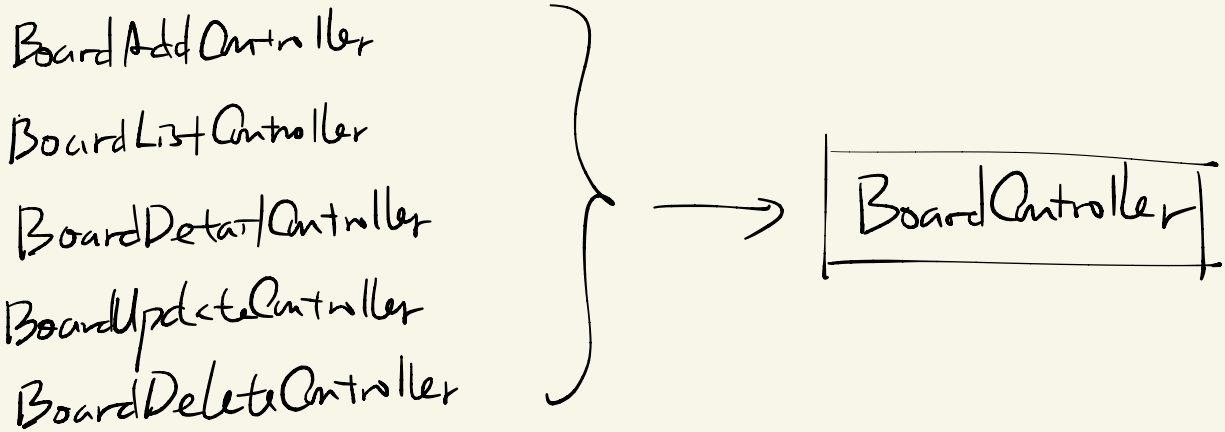
② 변경



어노테이션으로 표시

인터페이스 없이도 가능

16. CRUD 만들기


BoardAddController
BoardListController
BoardDetailController
BoardUpdateController
BoardDeleteController
BoardController

11. Page Controller의 구현방식