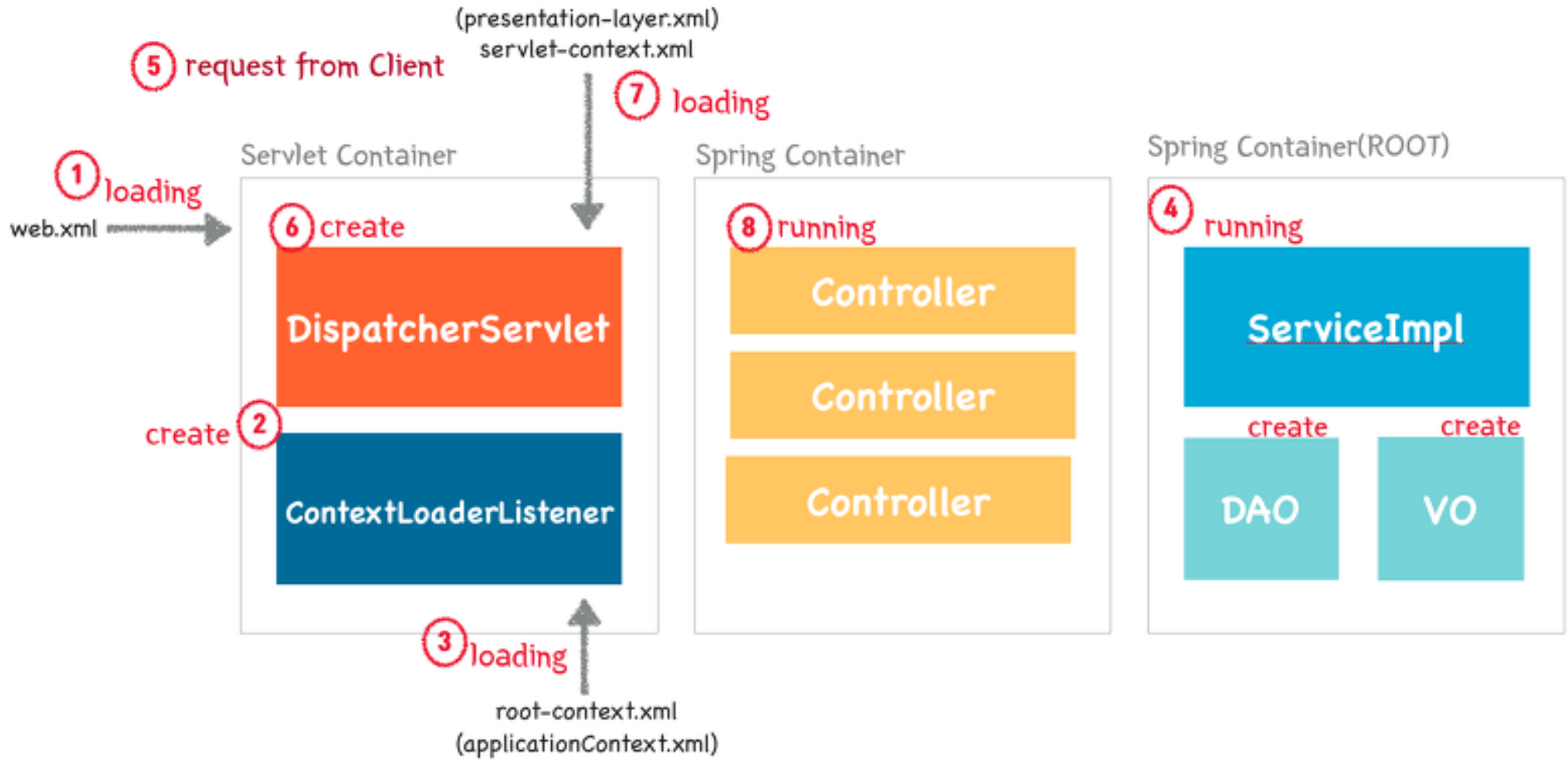
Spring교육자료_3주차

장윤정

1

웹어플리케이션의 컨텍스트 구성

웹 어플리케이션 동작원리(1/2)



Spring Framework

웹 어플리케이션 동작원리(2/2)

① 웹 어플리케이션이 실행되면 Tomcat(WAS)에 의해 web.xml이 로딩된다.

② web.xml에 등록되어 있는 ContextLoaderListener가 생성된다.

③ 생성된 ContextLoaderListener는 root-context.xml(설정파일)을 로딩한다.

④ 이 때 Spring Container(ROOT)가 구동된다.
Container 내부에는 root-context.xml (ContextLoaderListener 설정파일) 에 등록되어있는
공통 Bean들 및 개발자가 작성한 비즈니스 로직에 대한 부분과 DAO, VO 객체들을 포함하고 있다.

⑤ 클라이언트로부터 웹 어플리케이션에 요청이 들어온다.

⑥ DispatcherServlet이 생성된다.

⑦ DispatcherServlet은 servlet-context.xml (DispatcherServlet 설정파일) 을 로딩한다.

⑧ 이 때 두 번째 Spring Container가 구동되며, 요청에 맞는 Controller 들이 동작한다.
Spring Container의 Controller는 첫 번째로 생성된 Spring Container(ROOT) 내부에 있는
클래스들과 협업하여 알맞은 작업을 처리하게 된다.

웹 어플리케이션 컨텍스트 구성(1/3)

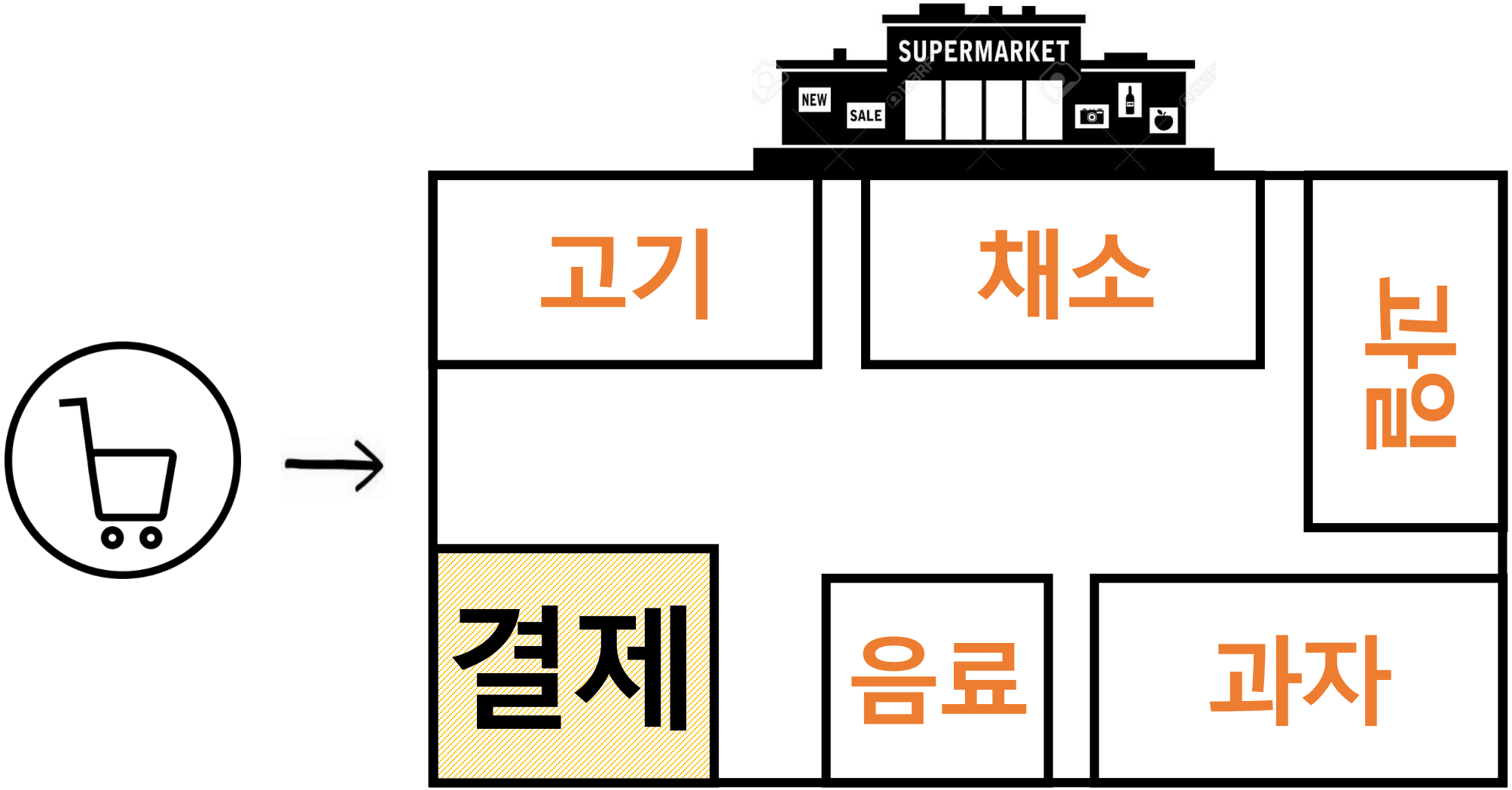
01 서블릿 컨텍스트와 루트 컨텍스트 계층 구조

가장 많이 사용되는 기본적인 구성방법
Spring 웹 기술을 사용하는 경우,
웹 관련 Bean들은 서블릿 컨텍스트에 두고
나머지는 루트 컨텍스트에 등록한다.

루트 컨텍스트는 모든 서블릿 레벨 컨텍스트의 부모 컨텍스트

Spring 웹 외에도 기타 웹 프레임워크나
HTTP 요청을 통해 동작하는 각종 서비스를 함께 사용 가능

- 미리 만들어진 어플리케이션 컨텍스트의 설정을 그대로 가져다 사용하면서 그 중 일부 Bean만 설정을 변경하고 싶을 때
- 어플리케이션 안에 성격이 다른 설정을 분리해서 2개 이상의 컨텍스트를 구성하면서, 각 컨텍스트가 공유하고 싶은 게 있을 때



- 어떤 코너에서 물건을 사더라도 결제(공통부분)는 한 곳에서 가능

웹 어플리케이션 컨텍스트 구성(2/3)

02

루트 컨텍스트 단일 구조

Spring 웹 기술을 사용하지 않고,
서드파티 웹 프레임워크나 서비스 엔진만을 사용해서
프레젠테이션 계층을 만든다면 Spring 서블릿을 둘 이유가 없다.

이때는 루트 애플리케이션 컨텍스트만 등록해주면 된다.

웹 어플리케이션 컨텍스트 구성(3/3)

03 서블릿 컨텍스트 단일 구조

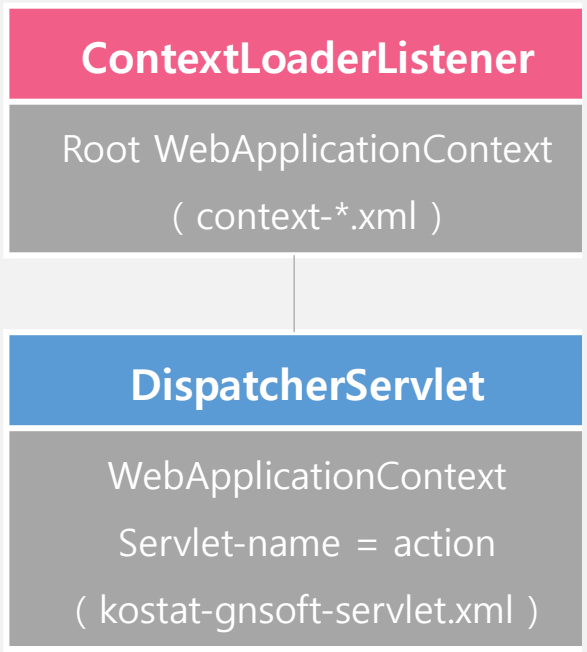
Spring 웹 기술을 사용하면서, Spring 외의 프레임워크나
서비스 엔진에서 Spring의 Bean을 사용할 생각이 아니라면
루트 컨텍스트를 생략할 수도 있다.

대신 서블릿에서 만들어지는 컨텍스트에
모든 Bean을 다 등록하면 된다.
계층 구조를 사용하면서 발생할 수 있는 혼란을 피하고
단순한 설정을 선호한다면 이 방법을 선택할 수 있다.

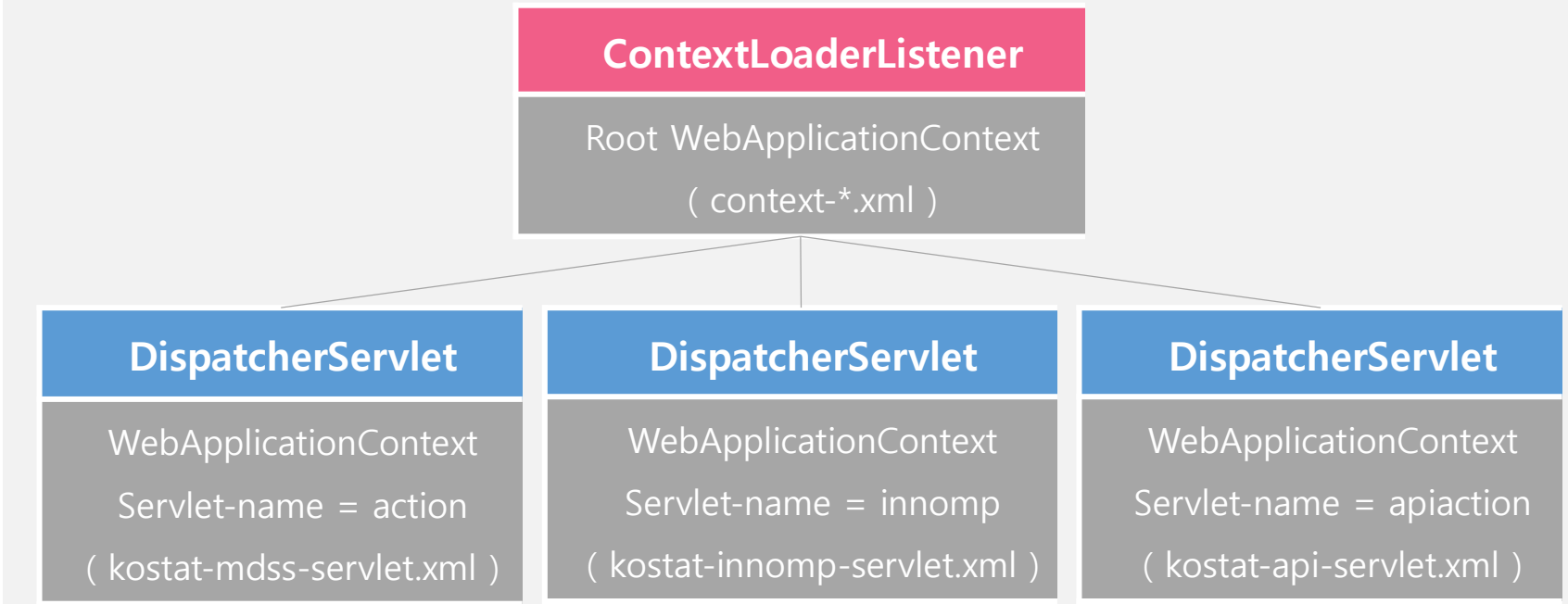
회사 홈페이지와 업무 프로젝트 DispatcherServlet 비교

- 루트 컨텍스트와 서블릿 컨텍스트 계층 구조에서 DispatcherServlet 설정 비교

회사홈페이지



MDIS 외부포탈 기준

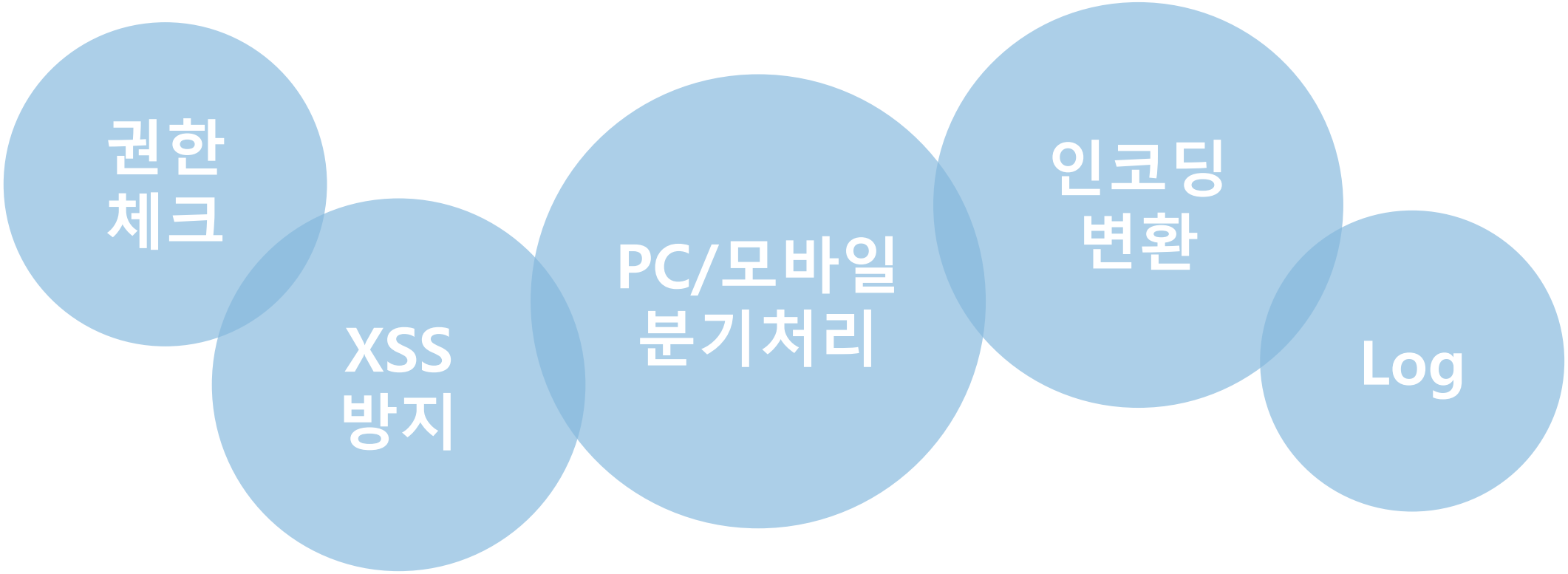


DispatcherServlet을 나눈 이유는?

2

Filter / Interceptor / AOP

이런 공통업무 관련 코드들을
모든 페이지마다 작성해야 한다면..?



실행흐름에 끼어들기

STEP1
- 사용자 로그인

STEP2
- 로그인 한 사용자 정보를 세션에 저장

STEP3
- 사용자가 기능을 요청

STEP4
- 시스템은 사용자가 요청한 기능을 수행하기 전 사용자의 세션을 체크

STEP5
- 올바른 세션일 경우 기능을 처리, 세션이 없거나 올바르지 않을 경우 사용자를 로그인 페이지로 이동

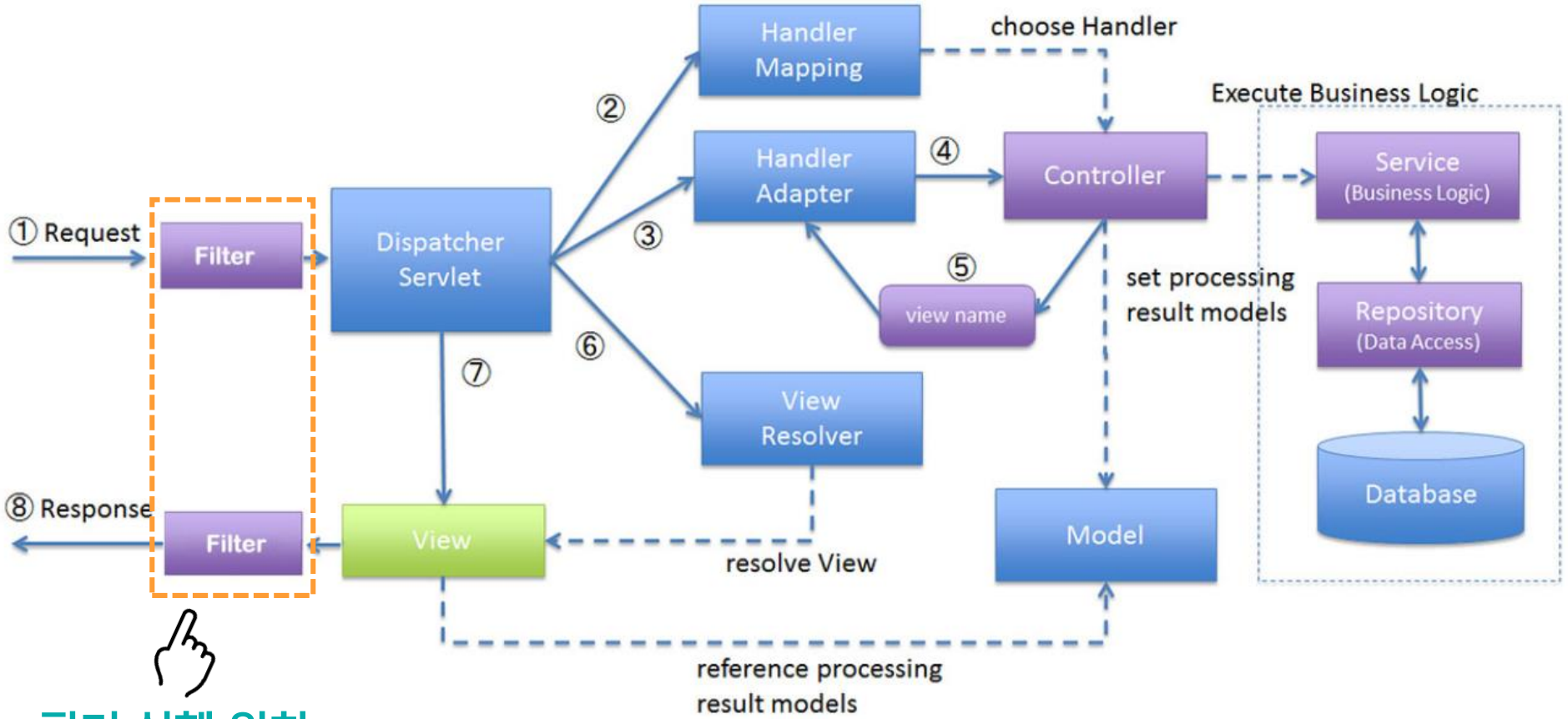
STEP4, STEP5를 처리하는 방법

프로젝트에 따라 Filter나 Interceptor를 사용

Filter

DispatcherServlet에 가기 전에 Filter가 있다!

- 클라이언트와 DispatcherServlet 사이에 위치하여 요청과 응답정보를 변경



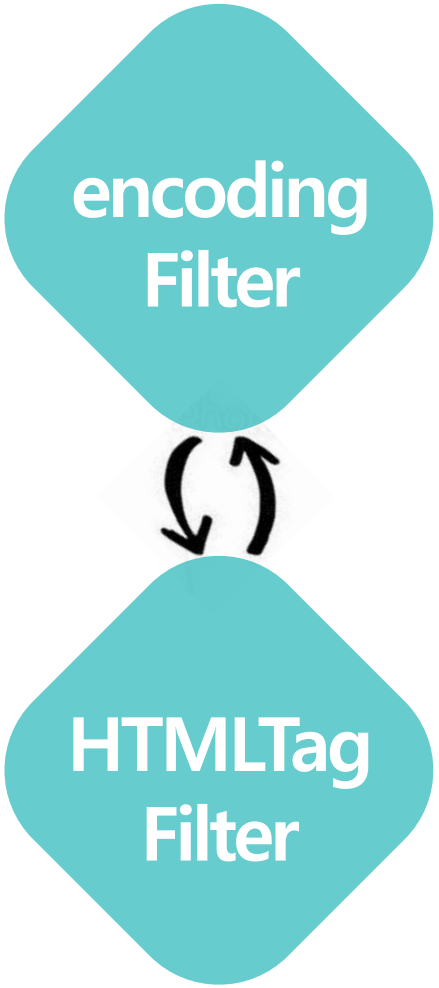
web.xml에서 Filter설정

- 클라이언트 요청 시 설정된 순서대로 적용 (응답이 나갈 때에는 반대 순서)

```
<filter>
    <filter-name>encodingFilter</filter-name>
    <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>utf-8</param-value>
    </init-param>
    <init-param>
        <param-name>forceEncoding</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>encodingFilter</filter-name>
    <url-pattern>*.do</url-pattern>
</filter-mapping>

<filter>
    <filter-name>HTMLTagFilter</filter-name>
    <filter-class>go.kostat.mdss.offer.common.filter.HTMLTagFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>HTMLTagFilter</filter-name>
    <url-pattern>*.do</url-pattern>
</filter-mapping>
```

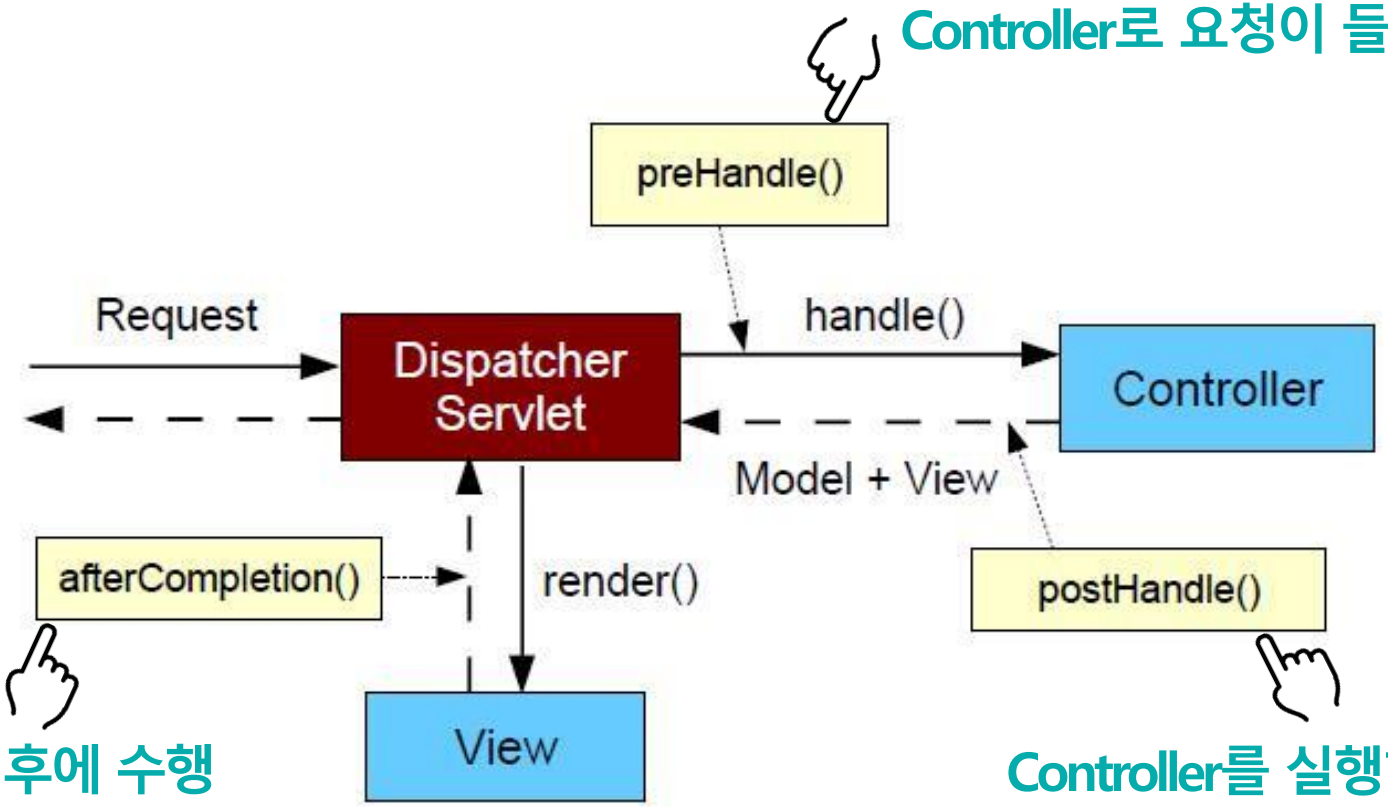
- Filter의 init() 메소드가 호출될 때 전달되는 파라미터 값 설정
- 특정 URL 패턴 요청에 대해 어떤 filter를 사용할 지 지정



Interceptor

요청을 가로채는 Interceptor

- DispatcherServlet이 Handler(Controller)를 호출하기 전, 후로 요청을 가로채서 처리



HandlerMapping은
DispatcherServlet으로부터 매핑작업을 요청받으면,
그 결과로 **핸들러 실행 체인(HandlerExcutionChain)**을 돌려준다.
이 핸들러 실행 체인은 하나 이상의 Interceptor를 거쳐서
Controller가 실행될 수 있도록 구성되어 있다.

Interceptor를 등록하지 않았다면 바로 컨트롤러가 실행되지만,
하나 이상의 Interceptor를 지정했다면
순서에 따라 Interceptor를 거친 후에 Controller가 호출된다.

Interceptor 적용된 소스 확인하기

비슷하지만 다른 Filter와 Interceptor

- 특정 URL에 접근할 때 제어하는 용도로 사용하는 것은 동일

차이점	Filter	Interceptor
실행위치	클라이언트의 요청이 발생되면 DispatcherServlet 진입 전 실행	Filter가 실행되고 난 후 DispatcherServlet에서 Controller 호출할 시점의 전, 후에 실행
실행순서	**Request – Filter – DispatcherServlet – Interceptor - Controller**	
사용되는곳	인코딩 변환, 로그인 체크, 권한 체크, XSS(Cross site script) 방어에 사용되고, 모든 페이지 내용에 공통적으로 특정 내용을 끼워 넣는 작업 등에 많이 사용	
메소드	• **init()** • **doFilter()** • **destroy()**	• **preHandle()** • **postHandle()** • **afterCompletion ()** • **afterConcurrentHandlingStarted()**

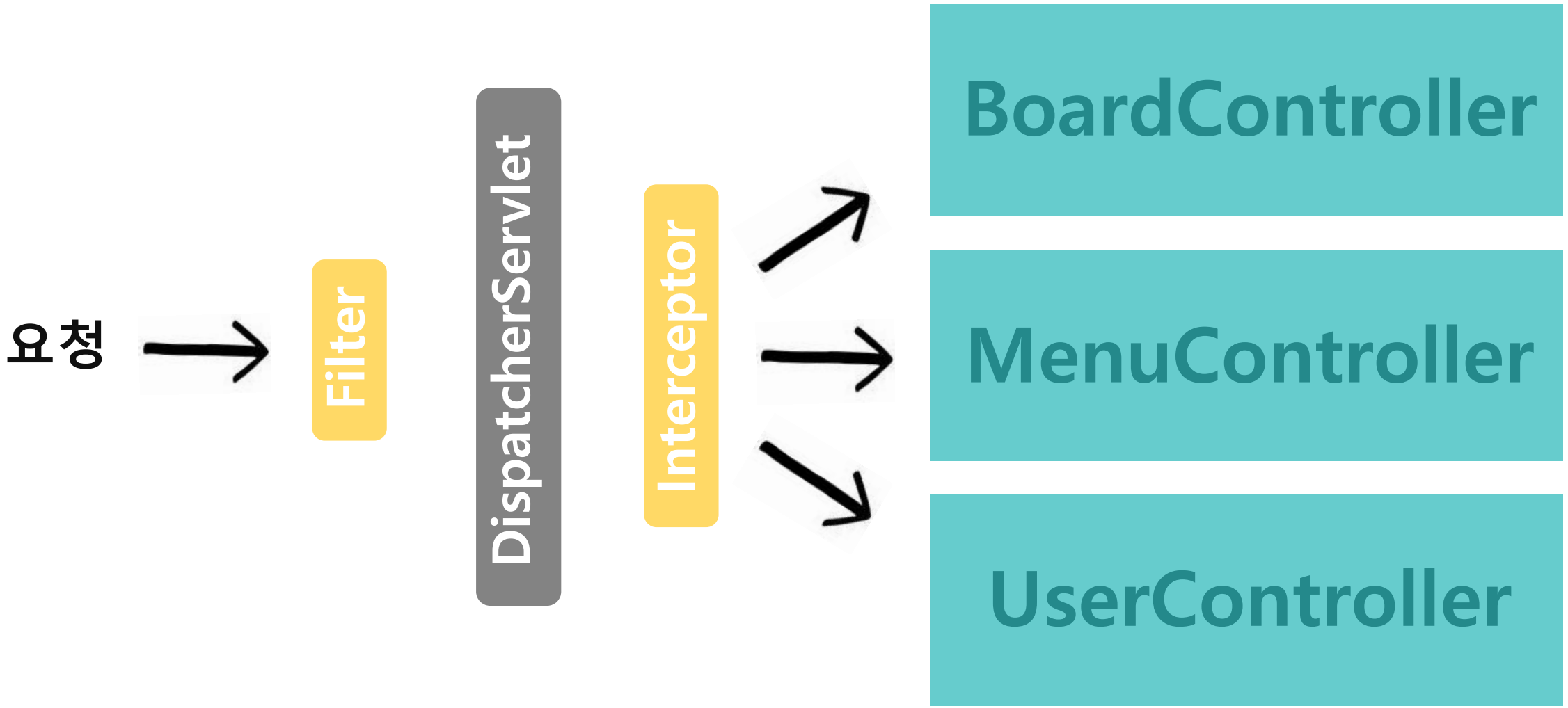
AOP(Aspect Oriented Programming)

- 관점 지향 프로그래밍 : 횡단 관점의 분리

기능을 핵심 비지니스 로직과
공통 모듈로 구분하여,
핵심 로직에 영향을 미치지 않고
비지니스 로직 사이사이에
공통 모듈을 끼워 넣는 개발 방법

AOP 개념잡기(1/3)

- Filter와 Interceptor 기준으로 보는 관점



AOP 개념잡기(2/3)

- 핵심관심(core concerns)

BoardController

insertBoard()

권한을 체크한다.

게시판을 등록한다.

로그를 남긴다.

MenuController

updateMenu()

권한을 체크한다.

메뉴를 수정한다.

로그를 남긴다.

UserController

deleteUser()

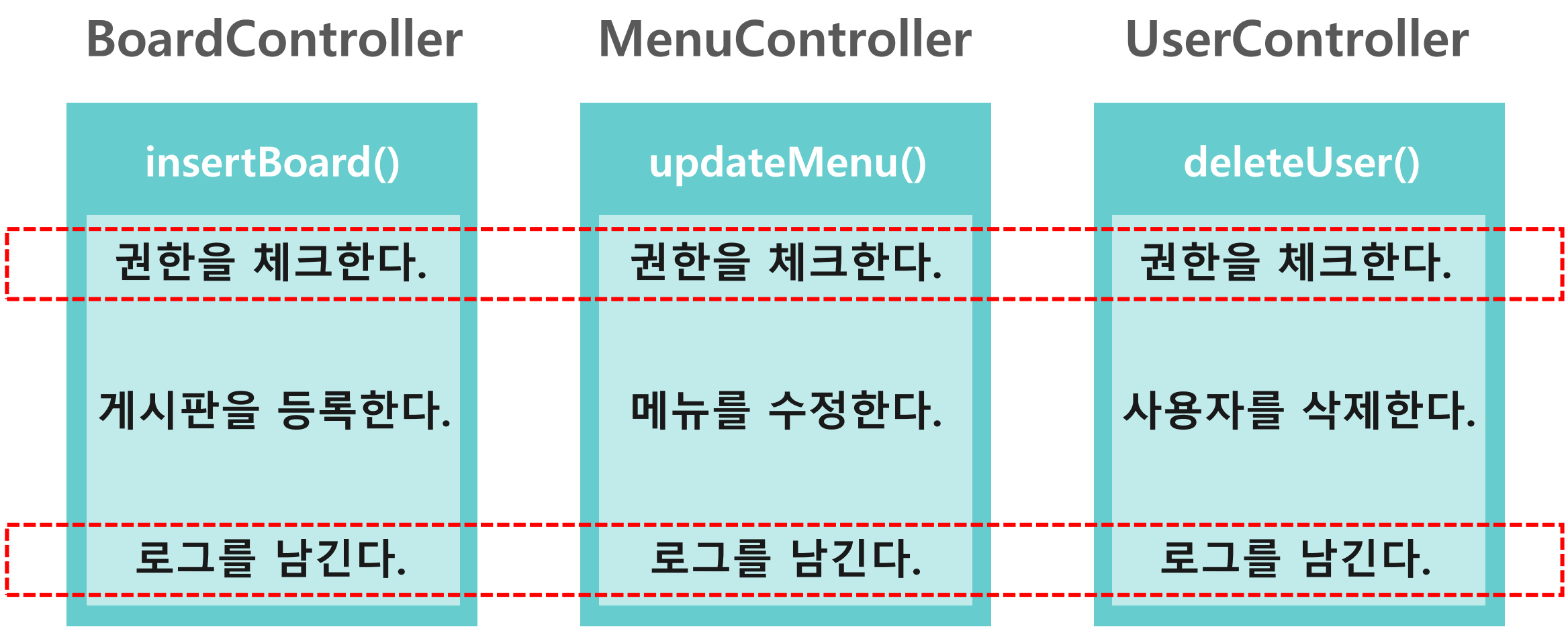
권한을 체크한다.

사용자를 삭제한다.

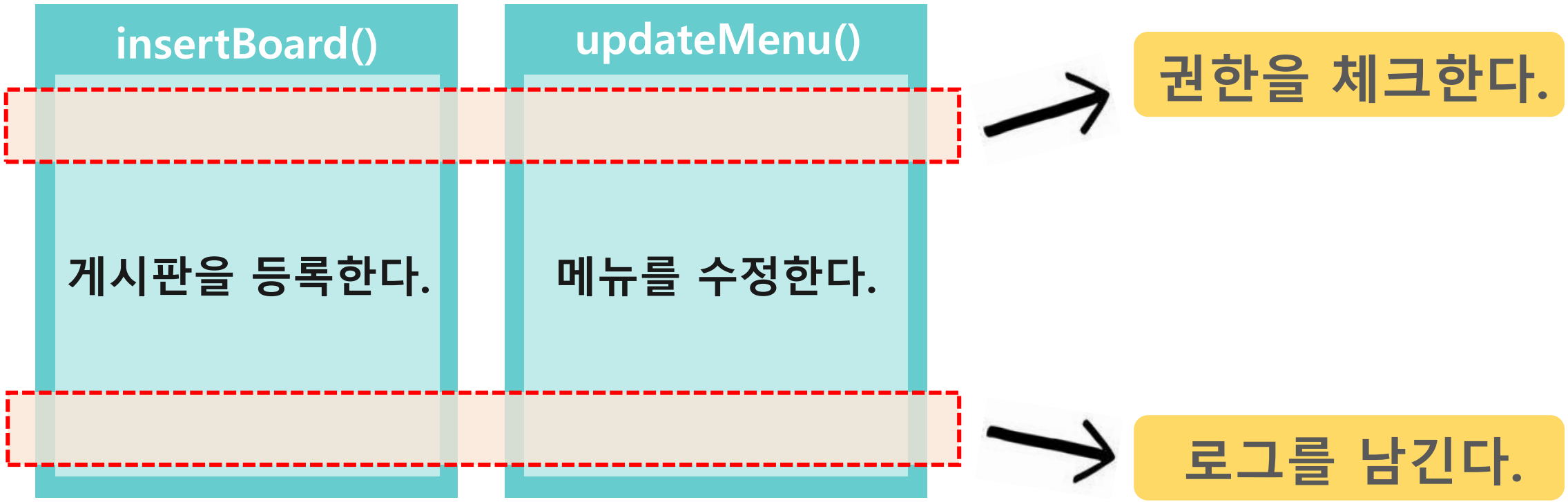
로그를 남긴다.

AOP 개념잡기(3/3)

- 횡단관심(cross-cutting concerns)



AOP – 어드바이스(Advice)



뽑아낸 공통 코드 = 어드바이스(Advice)

AOP – 조인포인트(Joinpoint)

Spring에서는

메소드 조인포인트만 제공

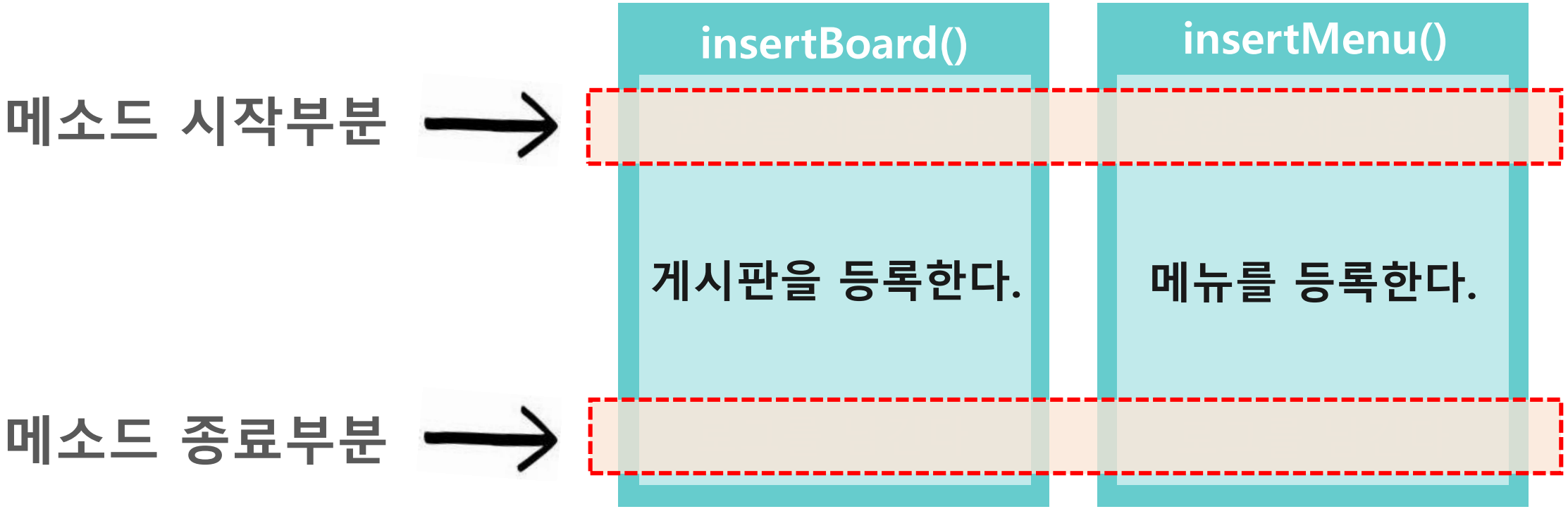
따라서 여기서 말하는

조인포인트는

모든 메소드를 의미

어드바이스가 적용될 수 있는 위치 = 조인포인트(Joinpoint)

AOP – 포인트컷(Pointcut)



특정 조건에 의해 필터링 된 조인포인트 = 포인트컷(Pointcut)

조인포인트 중에 특정 메소드에서만 공통기능을 수행시키기 위해 사용

Advice는 무엇을(what) + 언제(when)

Advice → 뽑아낸 공통코드

(무슨 행동을 할지 정해짐)

↓

그 행동을 '언제' 할지도 정의한 것.

- **@Before**
  - 메소드 호출 전 수행해라.
- **@After**
  - 메소드 결과가 성공인지 에러인지 상관없이 메소드 완료 후 수행해라.
- **@AfterReturning**
  - 메소드가 성공적으로 실행된 후 수행해라.
- **@AfterThrowing**
  - 메소드 수행 중 예외 발생 시 수행해라.
- **@Around**
  - 메소드 호출 전과 후에 수행해라.

Pointcut은 Joinpoint 중에서 어디에 (where)

Advice

언제(when),
무엇을(what)

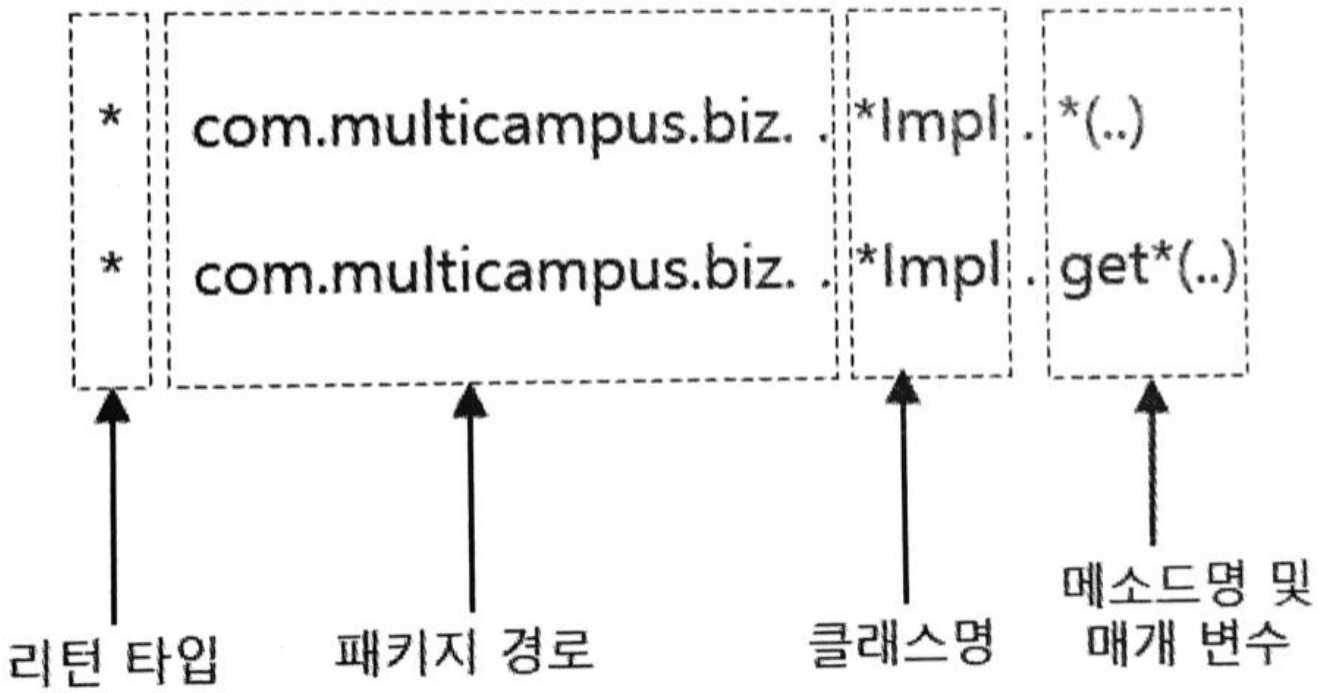
Pointcut

어디에 (where)

EX)

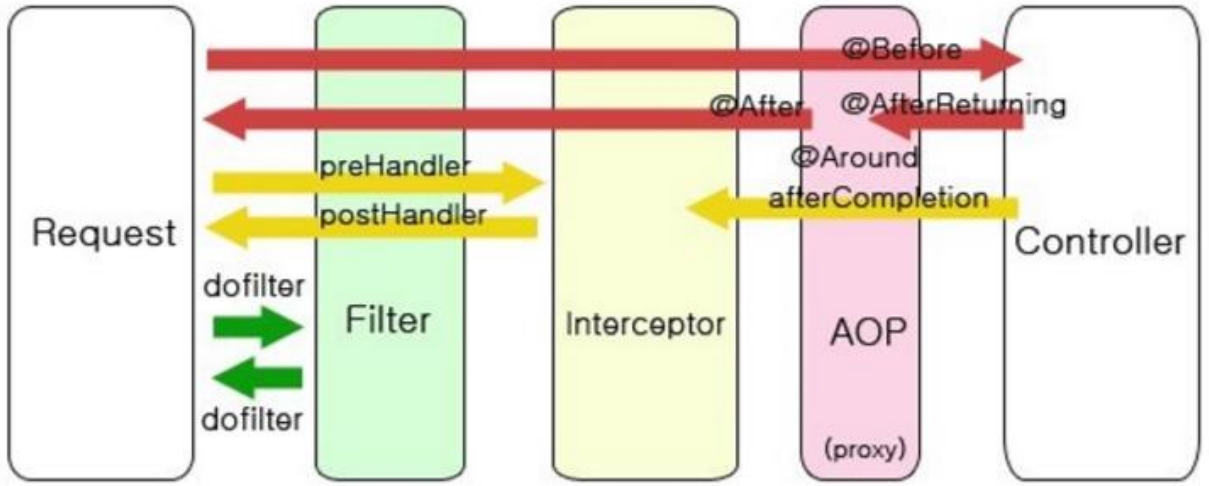
메소드 이름이 insert로 시작하는 메소드들은 메소드가 호출되기 전에 (@Before) 로그를 남긴다.

Pointcut 표현



AOP 적용된 소스 확인하기

Filter . Interceptor . AOP 비교



차이점	Filter	Interceptor	AOP
실행순서	1	2	3
설정위치	web.xml	xml or java	xml or java
실행메소드	init() doFilter() destroy()	preHandle() postHandle() afterCompletion() afterConcurrentHandlingStarted()	Pointcut으로 @before, @after, @around 등 위치를 지정하여 자유롭게 메소드 생성

오타 및 잘못된 내용 발견 시 알려주시면 감사하겠습니다 ☺