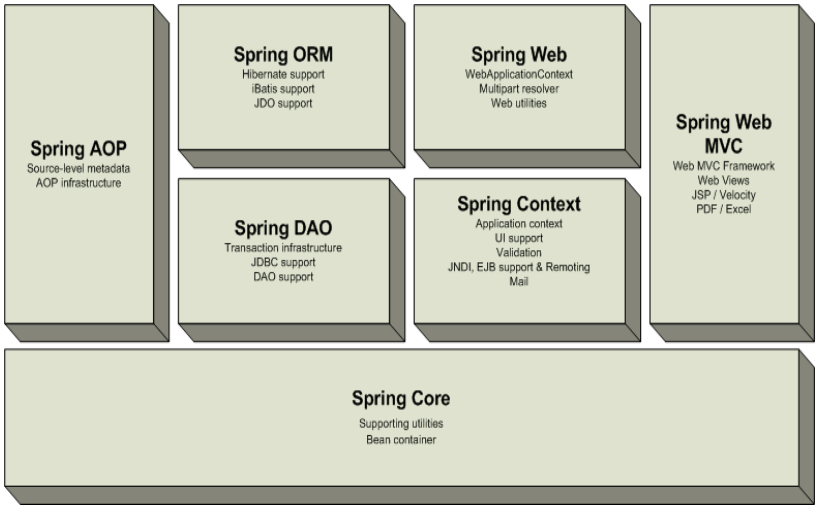
SPRING IOC

SPRING DI-setter / constructor

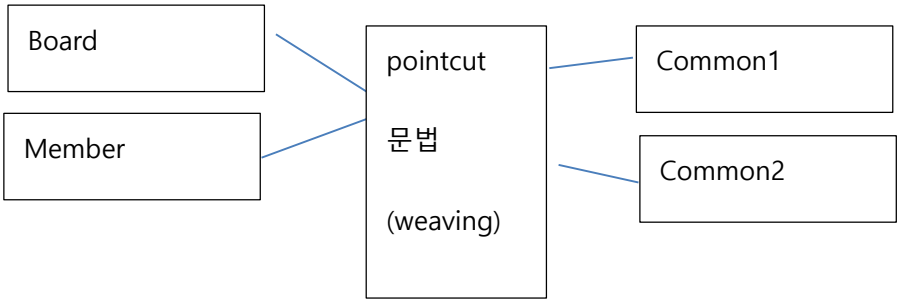
SPRING AOP-aspect oriented program

SPRING MVC

SPRING JDBC



aop – 집중적으로 처리할 클래스의 메소드 집중 핵심 로직 + 모든 클래스의 메소드 공통 로직



execution (public * *.*.*(..)) --> 실행 메소드 설정 + 이전 / 이후 / 전후 모두

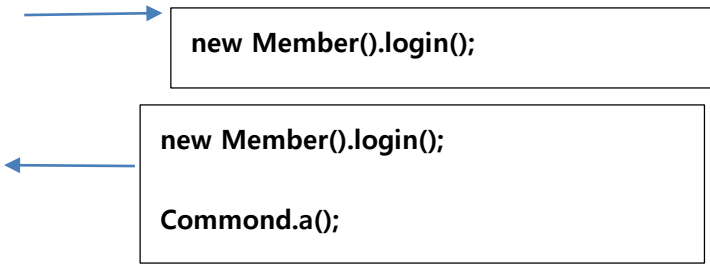
1> 해당 메소드 집중 처리(핵심코드=핵심관심=종단관심) + 모든 메소드 공통 처리(공통관심=횡단관심)

2> 코드 낭비 방지 = 재사용 = 별도 클래스/메소드들만 모으자

3> 공통처리 – 자유롭게 조립(포함/비포함) – 자바 소스 수정 없이

4> new Member().login(); 호출시 Commond.예외처리() 조립되어있다면 실행결과 리턴

===> 대리/위임/추가 구현 코드 같이 실행 결과 리턴 = proxy pattern



집중처리=집중관심=종단관심=메소드단위 --> 집중관심 모은 클래스 = target 클래스
공통관심=횡단관심=메소드 --> aspect 클래스
pointcut 문법
"execution (modifier returntype 패키지명.클래스명.메소드명(매개변수))"

Member- target 클래스
login(){
메소드전/후/전후 결정
방법 – point-cut
로그인처리
}
insert(){
회원가입
}

Board target 클래스

getBoard(){

게시물조회
}
registerBoard(){
게시물등록

}

Product target 클래스

shop(){

상품소개
}
buy(){
상품구입
}

spring aop api

+

xml설정

Common-aspect클래스

소요시간측정메소드,

db연결/해제

예외처리
로그인 in 기록

-pointcut 문법

<aop:config

<aop:pointcut `expression=`

`"execution(public * aop1..*.*(..))"`

`(modifier 리턴타입 패키지명.클래스명.메소드명(..)`

*** : 모든 요소**

(..) : 모든 매개변수

.. : 하위패키지 포함

예>expression="execution (public int *.*.get*(String))"

1. target클래스 – 핵심관심 구현 클래스들

2. aspect클래스 – 공통관심 구현 클래스들

3. pointcut 에 따라서 1+2 ---> target클래스의 어떤 메소드에 aspect의 어떤 메소드를 끼워넣을지 선정

4-1. target클래스 메소드 실행이전 aspect클래스메소드내용 끼워넣는다

4-2. target클래스 메소드 실행이후 aspect클래스메소드내용 끼워넣는다

4-3. target클래스 메소드 실행이전 /이후 모두 aspect클래스메소드내용 끼워넣는다

aop1.Board public void getBoard() register()

aop1.Member login()

xml / @

```
<bean id="common" class="..Common" />
<aop:pointcut expression="execution(public void a.Board.getBoard() ) " id="p"/>

"execution(public * a.Board.gerBoard() ) "
"execution(public * *.*.*(..) ) "
"execution(public * a..Board.get*( .. ) ) "

<aop:aspect ref=" common " id="asp" >
<aop:before point-ref="p" method="test" />
</aop:psect>
```

Common

a()

b()

Common:a()

Board:getBoard()

실습

1> AOP 스프링 라이브러리 추가

1-1.aspectjweaver.jar-mvnrepository.com

1-2. maven 태그 복사

```
<!-- https://mvnrepository.com/artifact/org.aspectj/aspectjweaver -->
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.9.9.1</version>
    <scope>runtime</scope>
</dependency>
```

1-3. 프로젝트₩pom.xml ---> MAVEN (스프링 프로젝트=maven 기능 사용 내장) 라이브러리(종속 라이브러리까지) 자동 다운로드 / 버전 / .....

(gradle = maven 대신 다른 툴)

1-4. MAVEN DEPENDENCIES₩*.jar

2> Aspect 클래스 정의/ target 클래스 정의

target – 핵심관심	aspect
Member	Common
Board	xml 설정 (메소드선정, 실행전 시점)

xml	annotation(자바파일내부)

<bean id="common" class="..Common" />
<bean id="board" class="..Board" />
<bean id="member" class="..Member" />
<aop:config>
<aop:pointcut expression= "execution(public void a.Board.getBoard())" id="p"/>
 	
<aop:aspect ref=" common " id="asp" >
<aop:before point-ref="p" method="a" />
</aop:apsect>
</aop:config>
 	
<!--@Service, Repository, Component, Autowired
<context:component-scan base-packages="aop1" />
<!--@PointCut, Around, Before, After, Apsect
<aop:aspectj-proxy />
	@Component
	@Aspect
	class Common{
	@PointCut("execution(public void a.Board.getBoard())")
	p(){ }
	
	@Before("p()")
	a(){}
	@After("p()")
	b(){}
	@Around("p()")
	c(){}
	}
	@Component
	class Board
	@Component
	class Member

19장 ioc , di xml	26장 @Service, Repository, Component, Autowired, Qualifier
20장 aop xml	@PointCut, Around, Before, After, Apsect
21장 spring mvc xml	annotation

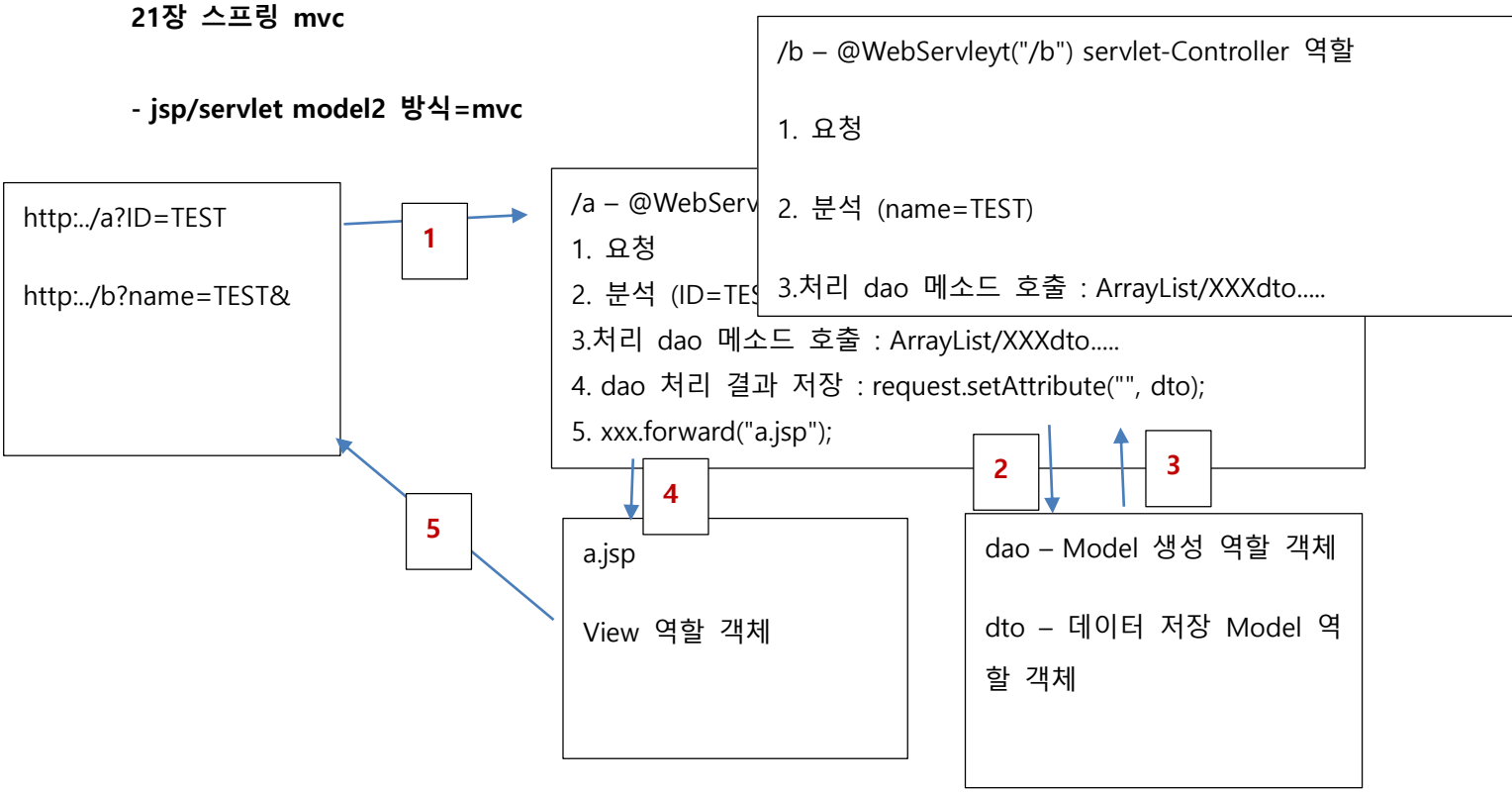
AOP - SERVLET API - ServletFilter - 요청 이후 . 응답 이전 실행 시점 - 서블릿마다 공통 수행 내용 정의

jsp/servlet = MVC 역할 파일 설계 = 1개 완성품

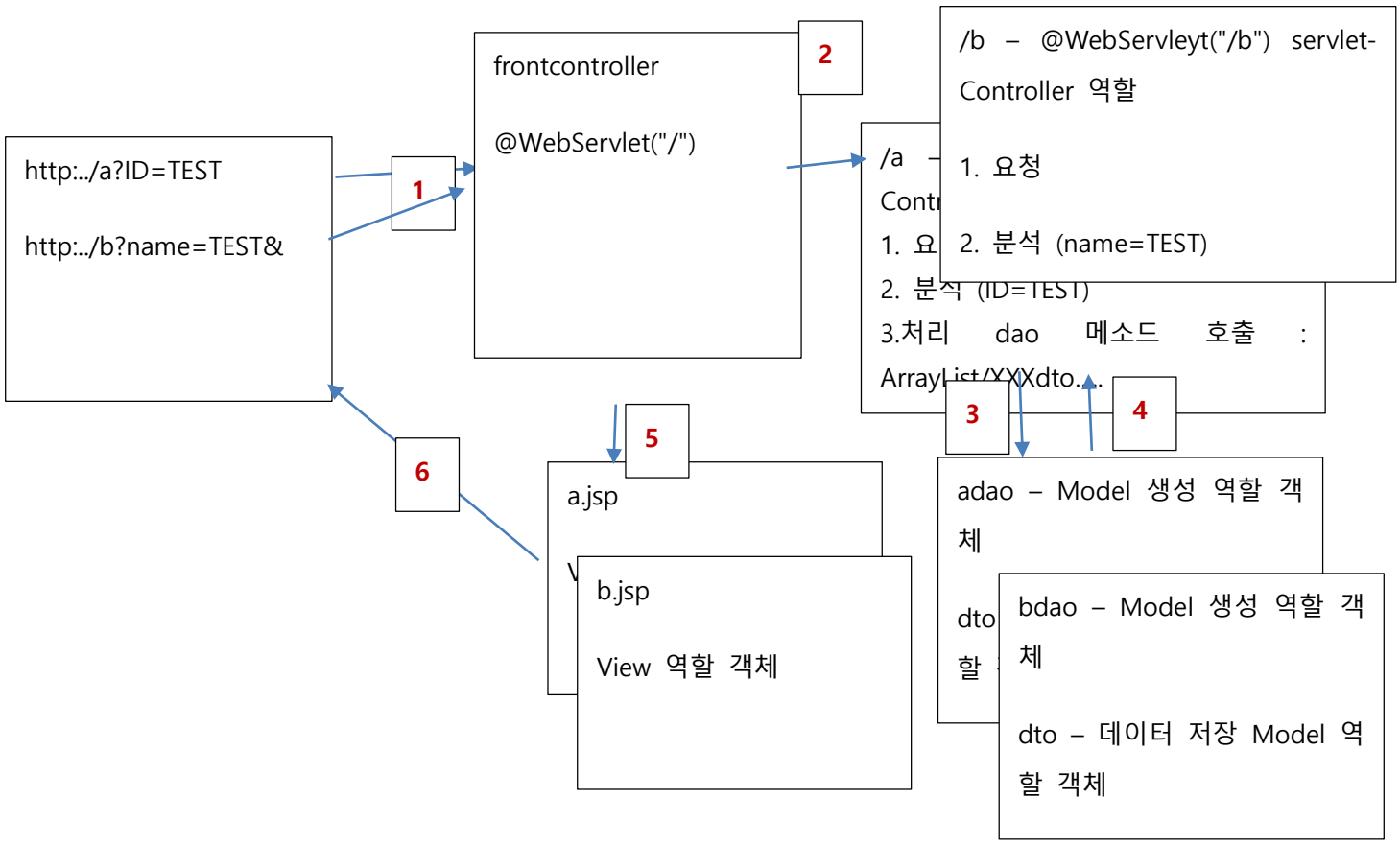
MODEL - VIEW - CONTROLLER

21장 스프링 mvc

- jsp/servlet model2 방식=mvc



- spring mvc 요청 - frontcontroller + mvc



실습

1> 스프링 라이브러리 사용하지 않고

dynamic web project - 스프링 mvc 보이도록 구현

nonspring

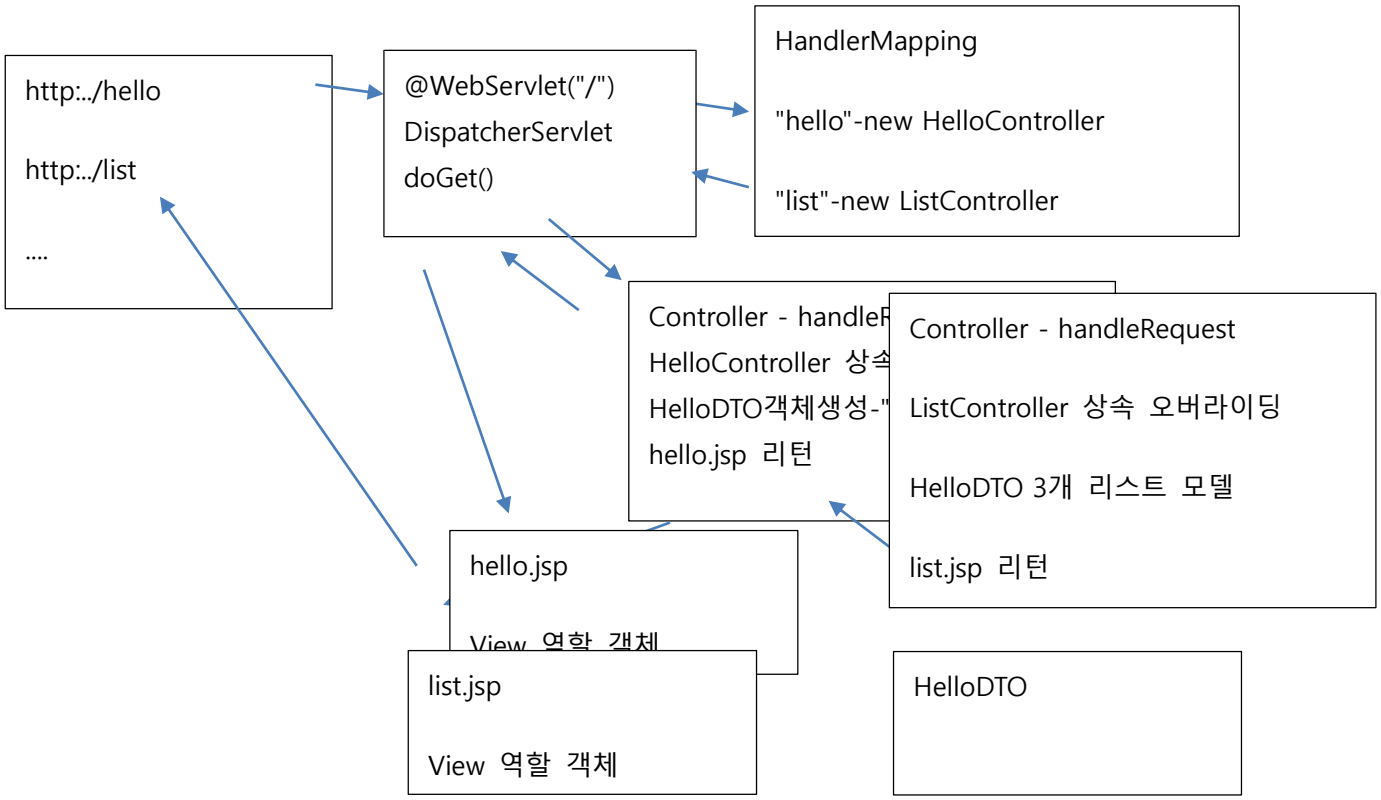
1>DispatcherServlet.java = frontcontroller역할 = 스프링 api 이름

2> Controller 인터페이스 - handleRequest() 메소드 선언

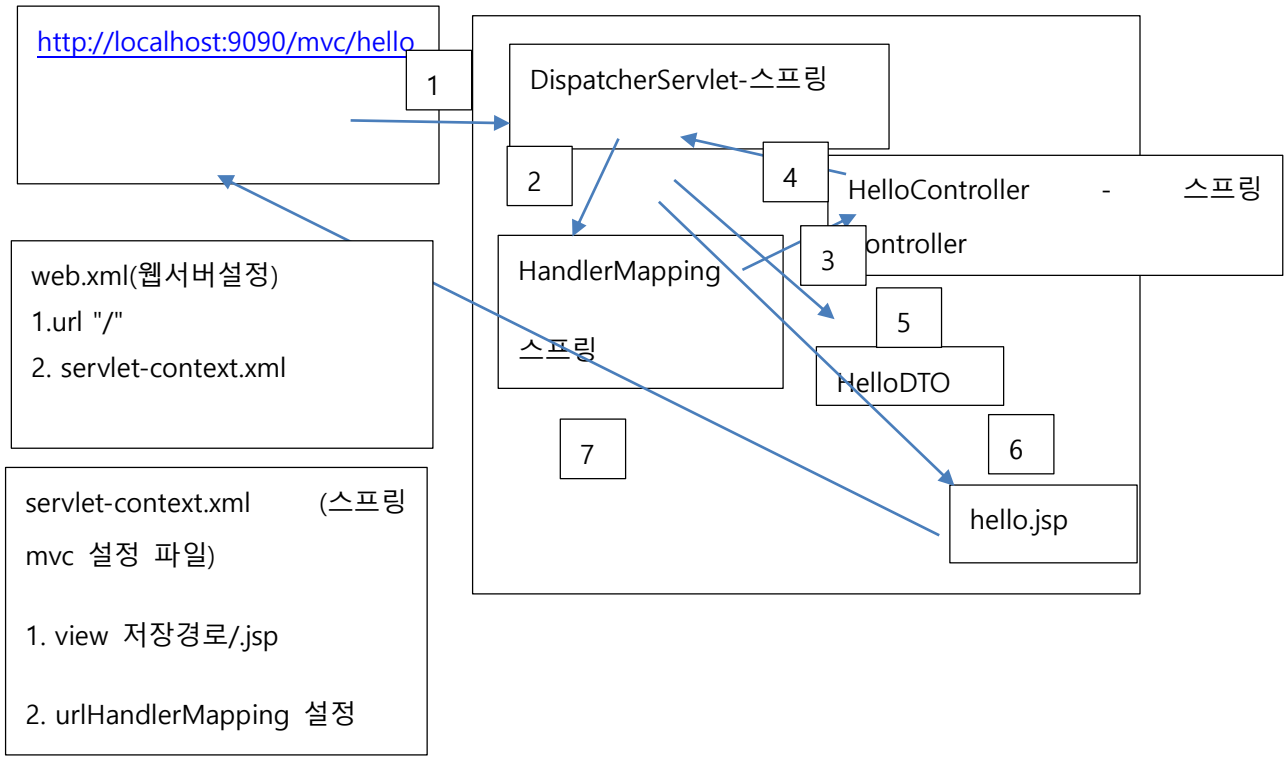
xxxxxController클래스들 - 2> 상속

3> HandlerMapping.java url ---> 매핑 서블릿이름 호출

4> DTO , jsp



2> 스프링 라이브러리 사용하고
spring project - 스프링 mvc 수정 구현



http://localhost:9090/mvc/loginform 1> servlet-context.xml /loginform ->LoginFormController 객체 매핑 추가 2> LoginFormController - 스프링 Controller 상속 ModelAndView handleRequest(.... ,....) 2-1. 모델 "title" String " 로그인양식 " 2-2. 뷰 loginform 3> loginform.jsp 로그인양식:모델 전달받은 값 action->http://localhost:9090/mvc/loginresult 아이디 암호 로그인	http://localhost:9090/mvc/loginresult 1> servlet-context.xml /loginresult ->LoginResultController 객체 매핑 추가 2> LoginResultController - 스프링 Controller 상속 ModelAndView handleRequest(.... ,....) 2-1. 아이디, 암호 ->LoginDTO 객체 생성 2-2. 모델 저장 : "login", LoginDTO 객체 2-2. 뷰 loginresult 3>loginresult.jsp xxx 회원님 암호는 xxx 입니다.

<ioc + di> @Service: new @Repository : new @Component : new @Autowired @Qualifier("")	xml 선언 <context:component-scan ....
<aop> @Aspect @Pointcut @After @Before @Around	<aop:aspectj-autoproxy />
@ComponentScan @Bean : 메소드 실행 리턴결과 객체 new	<ioc+di – 자바클래스 설정파일> @Configuration