

**aop : 여러 클래스에 공통 구현 코드**

**구현 코드 반복 여러군데 사용**

**1군데 정의 – 재사용**

**공통관심=횡단관심=별도 1개 클래스 정의**

**+ (spring aop , xml, @)**

**핵심관심=종단관심=실제 로직 구현 정의**

**pointcut**

**a.Test**

**a.b.c.Test3**

**execution (public \* a..클래스명.메소드명(..))**

**bean("bean id")**

**within(mvc.first.second.\*)**

**@Aspect--> 공통관심클래스 위 선언**

**@PointCut --> 메소드 내부 (내용 구현 x) 위 선언**

**@Around--> 메소드 위 선언(핵심관심메소드 전후)**

**@Before메소드 위 선언(핵심관심메소드 전**

**@A... 메소드 위 선언(핵심관심메소드 후**

**===> mvnrepository 사이트**

**aspectjwever.jar-->,dependency ->pom.xml**

**자바 디자인 패턴 23가지**

**1. singleton**

**2. factory**

**3. command – 인터페이스(메소드 다른 모든 클래스 공통 정의 규칙) / 상속 하위클래스 (오버라이딩)**

**4. proxy**

**5. front controller + mvc**

===================================================

[**http://localhost:9090/mvc2/hello**](http://localhost:9090/mvc2/hello) **요청--> hello 매핑 서블릿**

[**http://localhost:9090/mvc2/boardlist**](http://localhost:9090/mvc2/boardlist) **요청**

[**http://localhost:9090/mvc2/**](http://localhost:9090/mvc2/hello)**login?id=spring&pw=spring 요청**

**===>** [**http로 localhost:9090 서버의 mvc2 컨텍스트 url의 모든 요청은 FrontControllerServlet이**](http://localhost:9090/mvc2/hello) **전달받도록 한다.**

**.../hello**

**HelloController**

**FrontControllerServlet**

**Controller**

**HandlerMapping**

**("hello", new Helloxxxx())**

**View**

**Model**

**"model", Object**

[**http://localhost:9090/mvc/hello**](http://localhost:9090/mvc/hello) **요청**

**1. FrontControllerServlet 호출 ('/');web.xml**

**2. HelloController 호출 (Controller)**

**xxxxController : handleRequest 메소드 약속**

**인터페이스**

**3. "hello spring" 문자열값 생성(model)**

**4. hello.jsp 가 출력(view)**

**---> hello.jsp**

[**http://localhost:9090/mvc/boardlist**](http://localhost:9090/mvc/boardlist) **요청**

**1. BoardlistController 요청 전달**

**/boardlist ----> BoardlistController 객체생성**

**2. BoardlistController**

**Model : BoardDAO 객체 getList() 메소드 결과 리턴받아서 모델**

**("boardlist", BoardDAO 객체 getList() 메소드 리턴결과)**

**View : /WEB-INF/views/boardlist.jsp**

**3. boardlist.jsp**

**boardlist 라는 모델 출력**

**4. BoardDAO.JAVA**

**public ArrayList<BoardVO> getList( ){**

**오라클 SCOTT BOARD 테이블 3개 게시물 모두 리턴**

**}**

**BoardVO.JAVA : 게시물 1개 표현 자바 객체**

**ArrayList<BoardVO>: 게시물 리스트 자바 객체**

**===> spring mvc xml 태그 설정**

**+ spring mvc annotation 설정**

**<context:component-scan**

**get / post 방식**

**GET**

**1. http:....login?name1=value1&name2=value2**

**2. url 뒤 요청파라미터값 전송 방식**

**3. url?변수명1=값1&**

**4. 영문자 그대로 전송**

**5. 특수문자나 한글 인코딩 전송**

**6. url 공백 포함하면 안된다.( +, %xxx)**

**7. 웹서버 url 제한**

**post**

**1. (post 전송 값 보이지 않는다)**

**name1=value1&name2=value2**

**2. url 별도 분리**

**3. 영문자 그대로 전송**

**4. 특수문자나 한글 인코딩 전송**

**5. url 공백 포함하면 안된다.( +, %xxx)**

**7. 전달 길이 무제한**

**8. 파일 업로드, 암호 전송**

**9. 코드**

**url?id=키보드입력값**

**<a href="/hello?id=spring" > 이동 </A>**

**브라우저 주소표시줄 URL 입력**

**<form action="url" method="get(기본)"**

**<input type=text name=id >**

**<input type=submit value="로그인" >**

**<form>**

**url**

**(보이지 않는다) id=키보드입력값**

**<form action="url" method="post"**

**<input type=text name=id >**

**<input type=submit value="로그인" >**

**<form>**

[**https://search.daum.net/search?w=tot&DA=YZR&t\_\_nil\_searchbox=btn&sug=&sugo=&q=spring**](https://search.daum.net/search?w=tot&DA=YZR&t__nil_searchbox=btn&sug=&sugo=&q=spring)

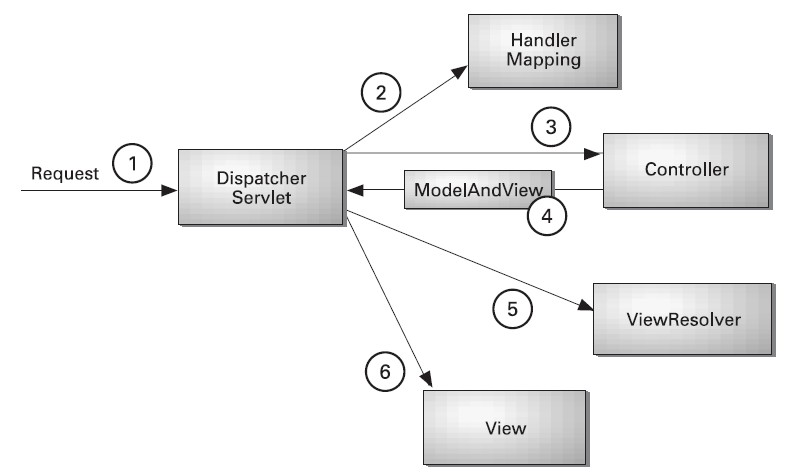
**@Controller**

**@RequestMapping("/hello")**

**@RequestMapping(name="/hello" , method=RequestMethod.GET)**

**@RequestMapping(name="/hello" , method=RequestMethod.POST)**

**spring MVC에서 하나의 요청에 대한 Life Cycle**

****

**Spring MVC내에서 처리되는 하나의 요청에 대한 Life Cycle과정은 위 그림과 같다.**

**클라이언트의 요청에 대한 최초 진입 지점은 DispatcherServlet이 담당하게 된다. 대부분의 MVC를 지원하는 프레임워크가 메인 Servlet을 가지는 것처럼 Spring MVC 또한 메인 Servlet이 최초 진입지점으로 다음의 작업을 처리하게 된다.**

**DispatcherServlet은 Spring Bean Definition에 설정되어 있는 Handler Mapping 정보를 참조하여 해당 요청을 처리하기 위한 Controller를 찾는다.**

**DispatcherServlet은 선택된 Controller를 호출하여 클라이언트가 요청한 작업을 처리한다.**

**Controller는 Business Layer와의 통신을 통하여 원하는 작업을 처리한 다음 요청에 대한 성공유무에 따라 ModelAndView 인스턴스를 반환한다. ModelAndView 클래스에는 UI Layer에서 사용할 Model데이터와 UI Layer로 사용할 View에 대한 정보가 포함되어 있다.**

**DispatcherServlet은 ModelAndView의 View의 이름이 논리적인 View 정보이면 ViewResolver를 참조하여 이 논리적인 View 정보를 실질적으로 처리해야할 View를 생성하게 된다.**

**DispatcherServlet은 ViewResolver를 통하여 전달된 View에게 ModelAndView를 전달하여 마지막으로 클라이언트에게 원하는 UI를 제공할 수 있도록 한다. 마지막으로 클라이언트에게 UI를 제공할 책임은 View 클래스가 담당하게 된다.**