Spring Framework

스프링 MVC Controller객체 구현

1.@Controller
2.@RequestMapping을 이용한 URL맵핑
2-1.Controller의 화면처리
3.요청 파라미터 처리 3가지
4.Model 데이터 전달자
4-1. @ModelAttribute
4-2. Model & ModelAndView

1. @Controller

아노테이션	설명
@Component[32]	일반적인 컴포넌트로 등록되기 위한 클래스에 사용합니다.
@Controller[33]	컨트롤러 클래스에 사용합니다.
@Service[34]	서비스 클래스에 사용합니다.
@Repository[35]	DAO 클래스 또는 리포지토리 클래스에 사용합니다.

Base-package에 기술된 패키지를 스캔
위 어노테이션이 붙은 클래스를 빈으로 생성

application-config.xml

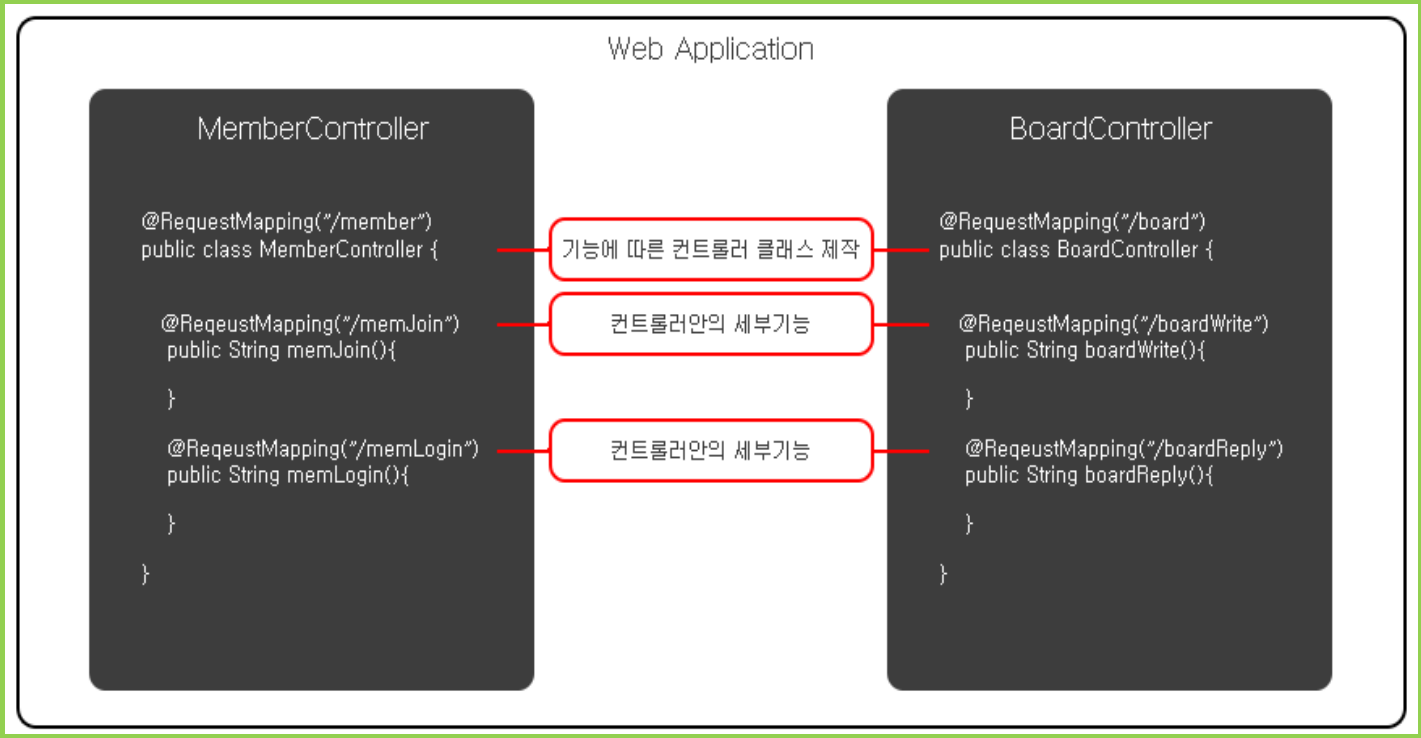
```
 1 <?xml version="1.0" encoding="UTF-8"
 2 <beans xmlns="http://www.springframework.org/schema/beans"
 3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 4     xmlns:p="http://www.springframework.org/schema/p"
 5     xmlns:context="http://www.springframework.org/schema/context"
 6     xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
 7     http://www.springframework.org/schema/context
   http://www.springframework.org/schema/context/spring-context-4.3.xsd">
 8
 9     <context:component-scan base-package="com.coderby.myapp.hello" />
10
11 </beans>
```

2 : @RequestMapping을 이용한 URL맵핑

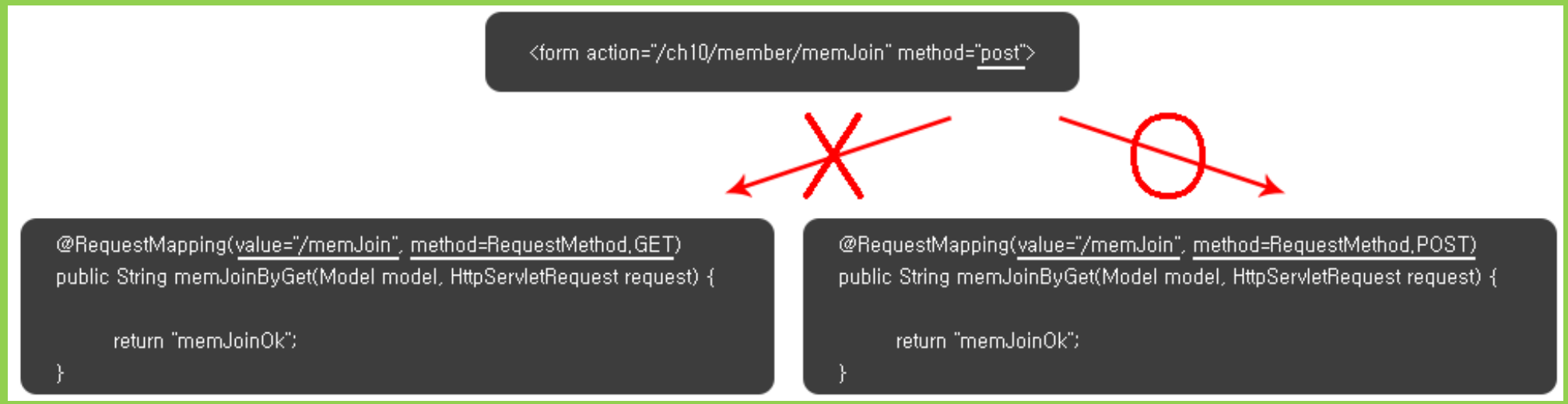
메소드에 @RequestMapping 적용

http://localhost:8181/lec18/memJoin → memJoin() 실행

클래스에 @RequestMapping적용



2 : @RequestMapping을 이용한 URL맵핑



2-1.Controller의 화면처리

```
@RequestMapping(value="/req_ex01")
public void req_ex01() {

}
```

1. void 메서드의 페이지이동

-일반적인 경우 맵핑 URL의 경로를 파일의 이름으로 사용합니다.

들어오는 경로와 나가는 경로가 같으면 void 사용 가능

```
@RequestMapping(value="/req_ex01")
public String req_ex01() {

return "request/req_ex01";
}
```

2. String 메서드의 페이지이동

-String형 메서드의 경우 view폴더 아래의 jsp파일로 이동됩니다.
-이동경로 앞에 redirect: 키워드를 통한 페이지 이동이 가능합니다.

```
@RequestMapping(value="/요청")
public String req_ex01() {

return "redirect:/req_ex01";
}
```

컨트롤러에서 리다이렉트시켜서 화면으로 나갈 수 있음

3. redirect 페이지이동

- redirect:/ 리다이렉트 키워드를 이용하면 다시 브라우저로 요청합니다.
- 즉 다시 Controller로 요청을 보냅니다.

```
@ResponseBody
```

- 메소드에서 리턴되는 값은 view리졸버로 전달되지 않고,
  해당메서드를 호출한 곳으로 결과를 반환합니다.
- 비동기 통신에 이용됨

3 : 요청 파라미터 (request)

메서드에서 파라미터 값의 처리 3가지

-전통적 방식의 처리

메서드(HttpServletRequest request)
String name = request.getParameter("name")

-어노테이션으로 처리 값은 필수로 줘야함 없으면 오류 페이지

메서드(@RequestParam(" name ") String name)
폼태그의 name값을 속성으로 받습니다.

-커맨드객체를 통한 처리 MemberVO 에 게터세터 생성자 to스트링 선언후 받아오기

폼 태그의 값을 받아 처리할 수 있는 Class생성
변수명을 폼 태그의 이름들과 일치하게 생성

스프링에서는 메서드 안에서 사용할 기능을
매개변수로 선언해서 사용합니다.

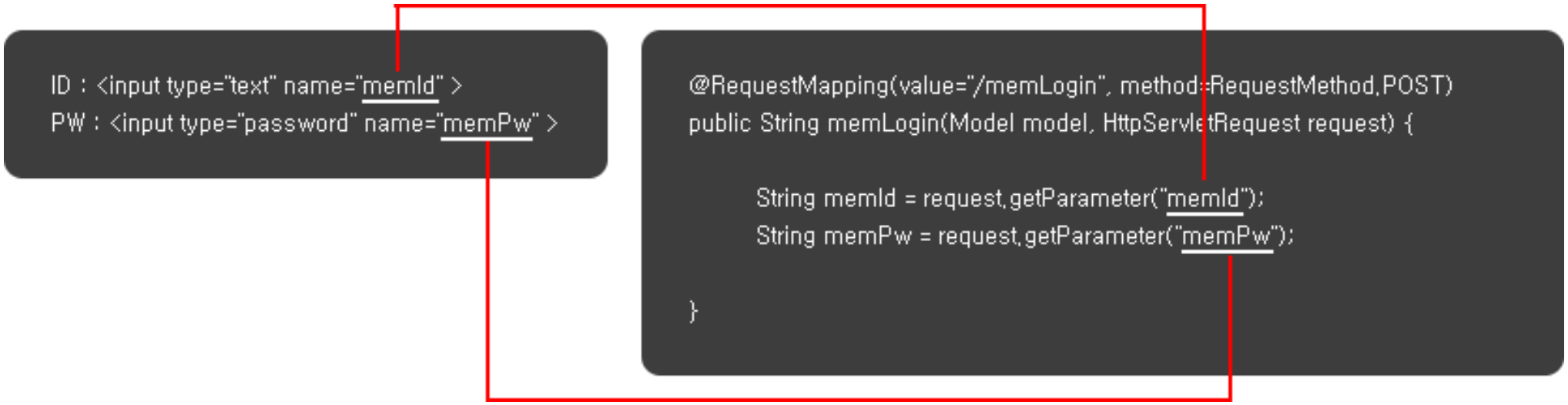
```
@RequestMapping("/param")
public String param(HttpServletRequest request)
```

```
@RequestMapping("/param")
public String param(@RequestParam("id") String id)
```

```
@RequestMapping("/param")
public String param(MemberVO vo)
```

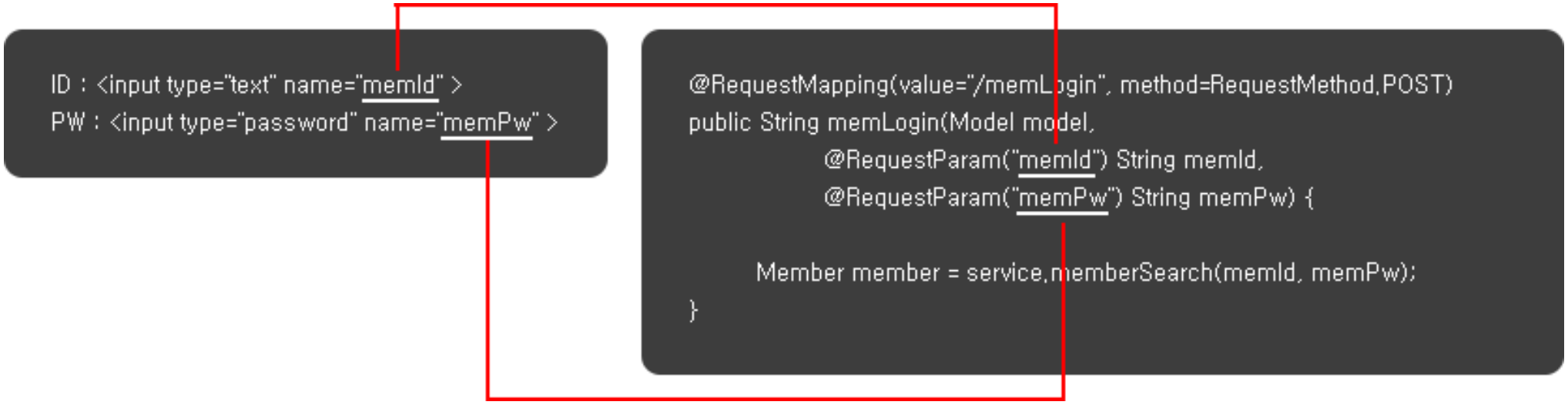
3 : 요청 파라미터 (request)

HttpServletRequest 객체를 이용한 HTTP 전송 정보 얻기 (전통적방식)



3 : 요청 파라미터 (request)

@RequestParam 어노테이션을 이용한 HTTP 전송 정보 얻기 (단일값을 얻을 때)



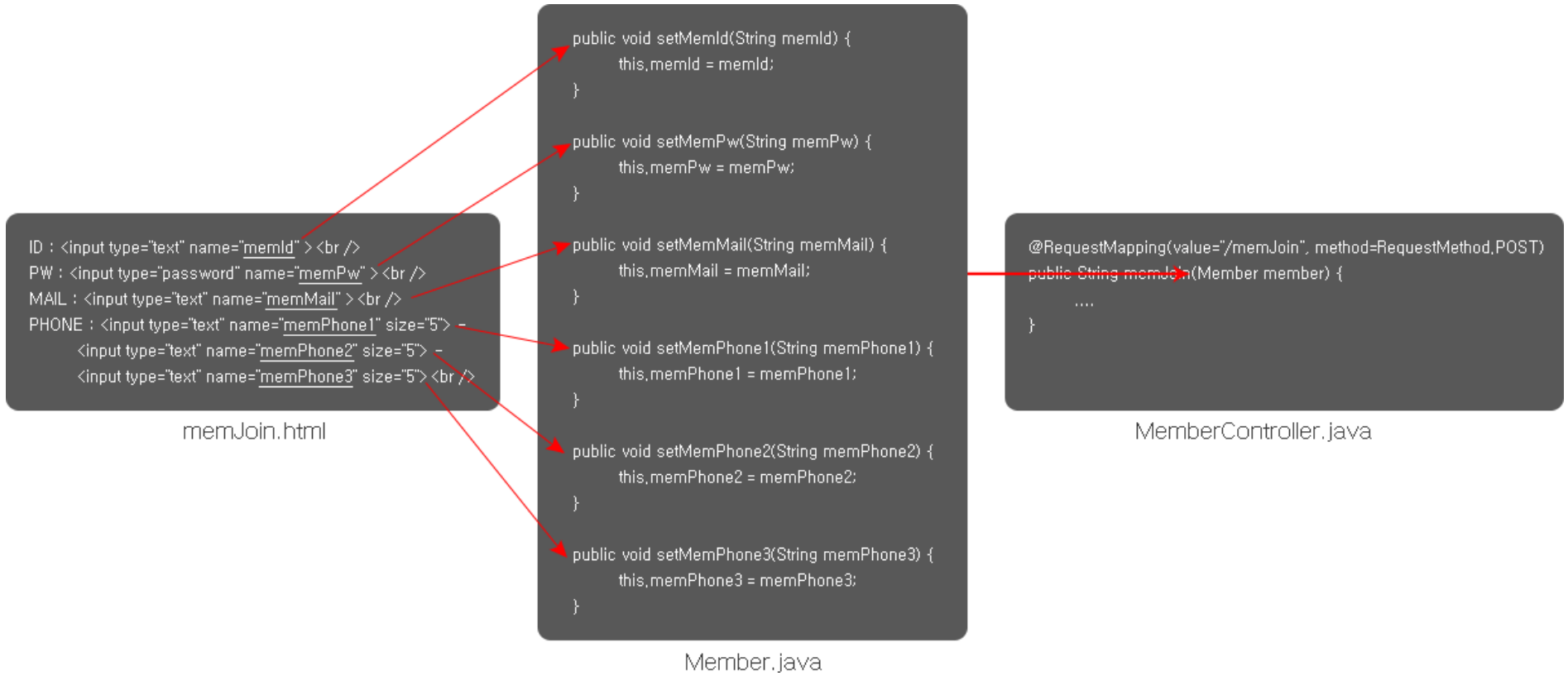
해당 어노테이션의 추가 속성

1. required – 해당 파라미터가 필수가 아닌 경우 지정
2. defaultValue - required지정시 기본값 을 지정

```
@RequestParam(value= " 파라미터값", required = false, defaultValue = “기본값")
```

3 : 요청 파라미터 (request)

커멘드 객체를 이용한 HTTP전송 정보 얻기
– VO객체에 동일한 setter가 있다면 자동으로 저장



4. Model전달자 – 화면에 데이터를 전달하기 위한 객체

4_1.Model 객체 =(ModelMap)

1. Model타입을 메서드의 파라미터로 주입하게 되면 view로 전달하는 데이터를 담아서 보낼 수 있습니다.
2. request.setAttribute()와 유사한 역할을 합니다.

ex) 메서드(Model model)
ex) addAttribute()

4_2.ModelAndView 객체 (페이지와, 데이터를 동시 지정)

ex) addObject("변수명", "값");
ex) setViewName("페이지명");

4_3.@ModelAttribute 어노테이션

1. 전달받은 파라미터를 Model에 담아서 화면까지 전달하려 할 때 사용되는 어노테이션입니다.

ex) @ModelAttribute("받을값") 사용할 변수

4_4.RedirectAttribute 객체

리다이렉트 이동시 파라미터 값을 전달하는 방법
ex) addFlashAttribute()

4_1 : Model & ModelAndView

컨트롤러에서 뷰에 데이터를 전달하기 위해 사용되는 객체로 Model과 ModelAndView가 있다.
두 객체의 차이점은 Model은 뷰에 데이터만을 전달하기 위한 객체이고, ModelAndView는 데이터와 뷰의 이름을 함께 전달하는 객체이다.

Model

```
@RequestMapping(value = "/memModify", method = RequestMethod.POST)
public String memModify(Model model, Member member) {

    Member[] members = service.memberModify(member);

    model.addAttribute("memBef", members[0]);
    model.addAttribute("memAft", members[1]);

    return "memModifyOk";
}
```

memModifyOk.jsp

```
ID : ${memBef.memId}
ID : ${memAft.memId}
```

4_2 : Model & ModelAndView

컨트롤러에서 뷰에 데이터를 전달하기 위해 사용되는 객체로 Model과 ModelAndView가 있다.
두 객체의 차이점은 Model은 뷰에 데이터만을 전달하기 위한 객체이고, ModelAndView는 데이터와 뷰의 이름을 함께 전달하는 객체이다.

ModelAndView

```
@RequestMapping(value = "/memModify", method = RequestMethod.POST)
public String memModify(Model model, Member member) {

    Member[] members = service.memberModify(member);

    model.addAttribute("memBef", members[0]);
    model.addAttribute("memAft", members[1]);

    return "memModifyOk";
}
```

```
    Member[] members = service.memberModify(member);

    ModelAndView mav = new ModelAndView();
    mav.addObject("memBef", members[0]);
    mav.addObject("memAft", members[1]);

    mav.setViewName("memModifyOk");

    return mav;
}
```

4_2 : Model & ModelAndView

Model 이용

```
@RequestMapping(value = "/memModify", method = RequestMethod.POST)
public String memModify(Model model, Member member) {

        Member[] members = service.memberModify(member);

        model.addAttribute("memBef", members[0]);
        model.addAttribute("memAft", members[1]);

        return "memModifyOk";
}
```

데이터 이름 & 데이터

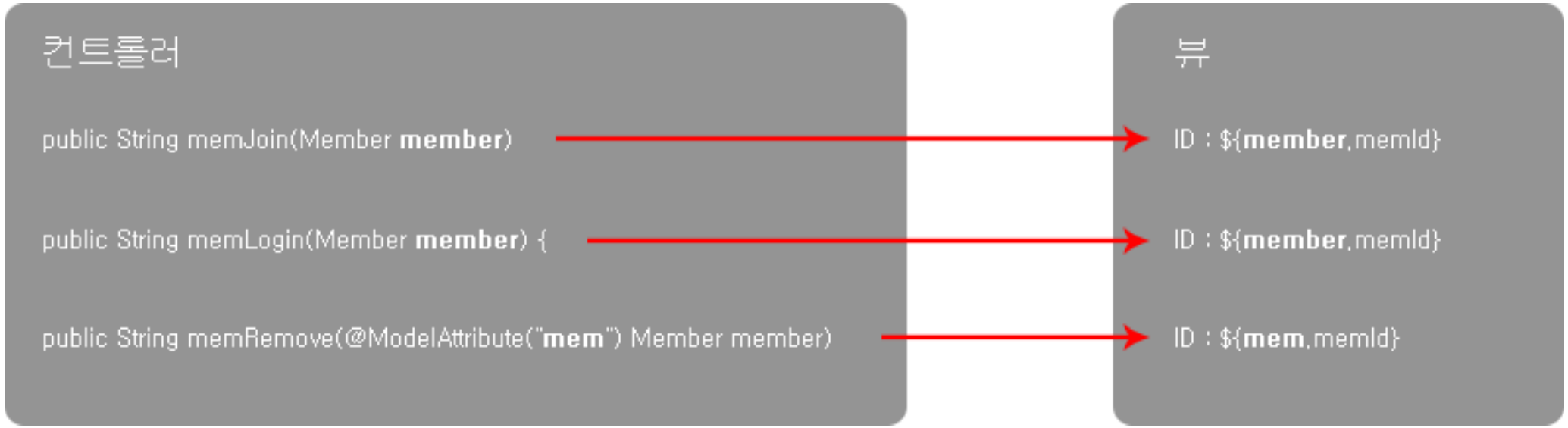
뷰이름

ModelAndView 이용

```
@RequestMapping(value = "/memModify", method = RequestMethod.POST)
public ModelAndView memModify(Member member) {

        Member[] members = service.memberModify(member);

        ModelAndView mav = new ModelAndView();
        mav.addObject("memBef", members[0]);
        mav.addObject("memAft", members[1]);

        mav.setViewName("memModifyOk");

return mav;
}
```

4_3 : @ModelAttribute

@ModelAttribute를 이용하면 커멘드 객체의 이름을 변경할 수 있고, 이렇게 변경된 이름은 뷰에서 커멘드 객체를 참조할 때 사용된다.



5 : EL에 관한 내용

EL은 컨트롤러에서 비즈니스 로직을 실행한 결과를 JSP 문서 내에서 출력하기 위한 용도로 사용

EL의 표현방법은 ${ 와 } 사이에 표현식

표현식에는 Scope 변수를 이용하여 Scope 변수에 바인딩 되어 있는 객체의 메서드를 호출

Scope 변수는 request, session, application을 의미

만일 request 객체에 회원 객체(member)가 바인딩 되어 있을 경우 회원의 이름(name)을 출력하기 위해 ${member.name} 또는 ${member["name"]}을 이용

컨트롤러에서.

```
request.setAttribute("emp", emp);
model.addAttribute("emp", emp);
```

뷰에서

기존 코드는...

```
<%
EmpVO emp = (EmpVO)request.getAttribute("emp");
%>
이름 <input type="text" name="firstName" value="<%=emp.getFirstName()%>">
```

EL을 이용하면

```
이름 <input type="text" name="firstName" value="${emp.firstName}">
```

6. JSTL에 관한 내용

JSTL은 표준화된 태그 라이브러리들을 제공함으로써 보다 편리하게 웹 응용프로그램을 개발할 수 있도록 지원

메이븐 pom.xml에 추가 되어있으므로 별다른 다운 없이 사용가능

종류	URI	prefix
Core	http://java.sun.com/jsp/jstl/core	c
XML processing	http://java.sun.com/jsp/jstl/xml	x
Formatting	http://java.sun.com/jsp/jstl/fmt	fmt
Database access	http://java.sun.com/jsp/jstl/sql	sql
Functions	http://java.sun.com/jsp/jstl/functions	fn

6. JSTL

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

태그	설 명
c:catch	예외처리에 사용
c:out	JspWriter 에 내용 출력
c:set	JSP 에서 사용될 변수 설정 <c:set value="value" target="targetObjectName" property="propertyName" />
c:remove	설정한 변수 제거
c:if	조건처리
c:choose	다중 조건 처리
c:forEach	컬렉션이나 Map 의 각 항목 처리
c:forTokens	구분자로 분리된 각각의 토큰을 처리할 때 사용
c:when	조건에 맞을 때
c:otherwise	맞는 조건이 없을 경우
c:import	URL 을 사용하여 다른 자원의 결과를 삽입
c:redirect	URL 을 재작성
c:param	파라미터 설정