14. MVC 4

1) 날짜 값 변환
2) @Path Variable
3) 익셉션 처리

1) 날짜 값 변환

- 날짜를 이용해 회원검색 기능 구현하기

1) 날짜 값 변환

MemberDao.java

from, to 두 개의 LocalDateTime 타입의 파라미터를 받음

```
public List<Member> selectByRegdate(LocalDateTime from, LocalDateTime to){
    List<Member> results = jdbcTemplate.query( sql: "select * from MEMBER where REGDATE between ? and ? " + "order by REGDATE desc",
            new RowMapper<Member>() {
        @Override
        public Member mapRow(ResultSet rs, int rowNum) throws SQLException {
            Member member = new Member(rs.getString( columnLabel: "EMIAL"),
                    rs.getString( columnLabel: "PASSWORD"),
                    rs.getString( columnLabel: "NAME"),
                    rs.getTimestamp( columnLabel: "REGDATE").toLocalDateTime());
            member.setId(rs.getLong( columnLabel: "ID"));
            return member;
        }
    }, from, to);
    return results;
}
```

From, to 사이에 등록된 회원을 검색하는 쿼리문

1) 날짜 값 변환

ListCommand.java

```java
public class ListCommand {

    private LocalDateTime from;
    private LocalDateTime to;

    public LocalDateTime getFrom() { return from; }

    public void setFrom(LocalDateTime from) { this.from = from; }

    public LocalDateTime getTo() { return to; }

    public void setTo(LocalDateTime to) { this.to = to; }
}
```

<input type=”text” name=“from” />
<input type=“text” name=“to” />

사용자로부터 from과 to를
전달받기 위해 빈 클래스를 설정

Jsp 파일의 input태그의
name속성과 일치하는
프로퍼티에 setter를 통해
값을 저장

1) 날짜 값 변환

ListCommand.java

```java
public class ListCommand {

    private LocalDateTime from;
    private LocalDateTime to;

    public LocalDateTime getFrom() { return from; }

    public void setFrom(LocalDateTime from) { this.from = from; }

    public LocalDateTime getTo() { return to; }

    public void setTo(LocalDateTime to) { this.to = to; }
}
```

<input type="text" name="from" />
<input type="text" name="to" />

Input 태그의 value값은 문자열로 전달됨

이전 예제에서는 int, long, string, Boolean 등의 데이터타입은 스프링 MVC가 자동으로 형변환해 프로퍼티에 전달되었음

그렇다면 LocalDateTime은..?

1) 날짜 값 변환

ListCommand.java

```java
public class ListCommand {

    private LocalDateTime from;
    private LocalDateTime to;

    public LocalDateTime getFrom() { return from; }

    public void setFrom(LocalDateTime from) { this.from = from; }

    public LocalDateTime getTo() { return to; }

    public void setTo(LocalDateTime to) { this.to = to; }
}
```

Int OK
Long OK
String OK
Boolean OK
LocalDateTime …?

<input type=”text” name=“from” />
<input type=“text” name=“to” />

LocalDateTime의 경우 스프링 MVC가
형변환을 하기위해 설정을 추가해야함

1) 날짜 값 변환

ListCommand.java

```java
public class ListCommand {

    @DateTimeFormat(pattern = "yyyyMMddHH")
    private LocalDateTime from;
    @DateTimeFormat(pattern = "yyyyMMddHH")
    private LocalDateTime to;

    public LocalDateTime getFrom() { return from; }

    public void setFrom(LocalDateTime from) { this.from = from; }

    public LocalDateTime getTo() { return to; }

    public void setTo(LocalDateTime to) { this.to = to; }
}
```

Int OK
Long OK
String OK
Boolean OK
LocalDateTime OK!!!!

<input type=”text” name=“from” />
<input type=“text” name=“to” />

빈 클래스에서 @DateTimeFormat 어노테이션을 사용해 스프링 MVC에 해당 프로퍼티를 DateTimeFormat 타입으로 형변환을 해야함을 알려주어야함

Patten은 jsp파일에서 문자열로 넘어온 값의 패턴을 명시하고, 명시한 형태를 보고 스프링MVC는 LocalDateTime 타입으로 형변환

1) 날짜 값 변환

MemberListController.java

```java
@Controller
public class MemberListController {
    private MemberDao memberDao;

    public void setMemberDao(MemberDao memberDao) { this.memberDao = memberDao; }

    @RequestMapping("/members")
    public String list(@ModelAttribute("cmd") ListCommand listCommand, Errors errors, Model model){
        if(errors.hasErrors()) return "member/memberList";

        if(listCommand.getFrom() != null && listCommand.getTo() != null){
            List<Member> members = memberDao.selectByRegdate(listCommand.getFrom(), listCommand.getTo());
            model.addAttribute( attributeName: "members", members);
        }

        return "member/memberList";
    }
}
```

컨트롤러를 통해 요청에 대한 매핑처리

1) 날짜 값 변환

ControllerConfig.java

```java
@Autowired
private MemberDao memberDao;

@Bean
public MemberListController memberListController(){
    MemberListController controller = new MemberListController();
    controller.setMemberDao(memberDao);
    return controller;
}
```

컨트롤러를 스프링 컨테이너가 관리할 수 있도록 빈 객체로 등록

1) 날짜 값 변환

formatDateTime.tag

```
<%@ tag body-content="empty" pageEncoding="utf-8" %>
<%@ tag import="java.time.format.DateTimeFormatter" %>
<%@ tag trimDirectiveWhitespaces="true" %>
<%@ attribute name="value" required="true" type="java.time.temporal.TemporalAccessor" %>
<%@ attribute name="pattern" type="java.lang.String" %>
<%
    if (pattern == null) pattern = "yyyy-MM-dd";
%>
<%= DateTimeFormatter.ofPattern(pattern).format(value)%>
```

JSP에서 LocalDateTime 타입의 데이터를 지정한 형식으로 출력하기 위해
커스텀 태그 파일을 만들어줌

해당 커스텀 태그를 이용하면 위의 코드에서 설정한 패턴(yyyy-MM-dd) 형식으로
JSP파일에서 데이터를 출력가능

1) 날짜 값 변환

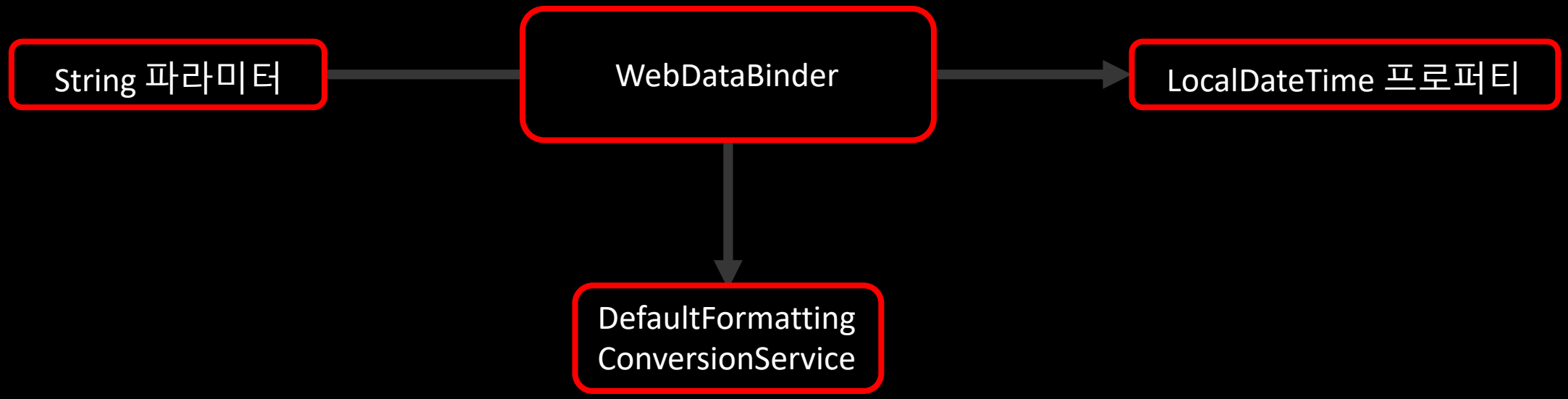
memberList.jsp

```jsp
<form:form modelAttribute="cmd">
    <p>
        <label>from : <form:input path="from" /></label>
        <form:errors path="from" />
        ~
        <label>to : <form:input path="to" /></label>
        <form:errors path="to" />
        <input type="submit" value="조회">
    </p>
</form:form>

<c:if test="${! empty members}">
    <table>
        <tr>
            <th>아이디</th><th>이메일</th>
            <th>이름</th><th>가입일</th>
        </tr>
        <c:forEach var="mem" items="${members}">
            <tr>
                <td>${mem.id}</td>
                <td>
                    <a href="<c:url value="/members/${mem.id}"/> ">
                        ${mem.email}
                    </a>
                </td>
                <td>${mem.name}</td>
                <td><tf:formatDateTime value="${mem.registerDateTime}" pattern="yyyy-MM-dd"/></td>
            </tr>
        </c:forEach>
    </table>
</c:if>
```

커스텀 태그를 이용해 LocalDateTime 타입의 데이터가 yyyy-MM-dd 형식으로 출력됨을 확인할 수 있음

1) 날짜 값 변환

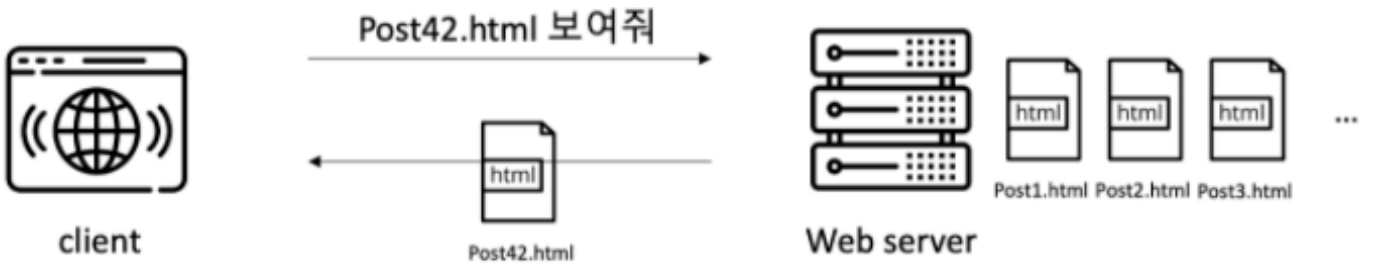


- String 타입으로 전달된 파라미터는 WebDateBinder를 통해 LocalDateTime 타입으로 변환(직접 변환하는 것은 아니고 Conversion Service에 해당 작업을 위임)
- WebDateBinder는 프로퍼티 바인딩에 대한 작업들에 관여(Validator 등)
- 프로퍼티 바인딩이란 오브젝트의 프로퍼티에 값을 넣는 행위
- @EnableWebMVC 어노테이션을 사용하면, 특별한 설정없이 DefaultFormattingConversionService를 사용해 데이터 타입을 변환

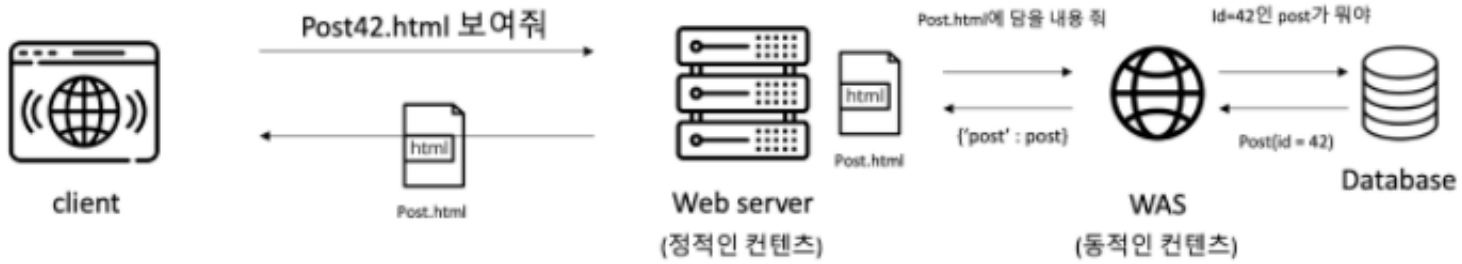
2) @Path Variable

- 가변 경로 처리하기

2) @Path Variable

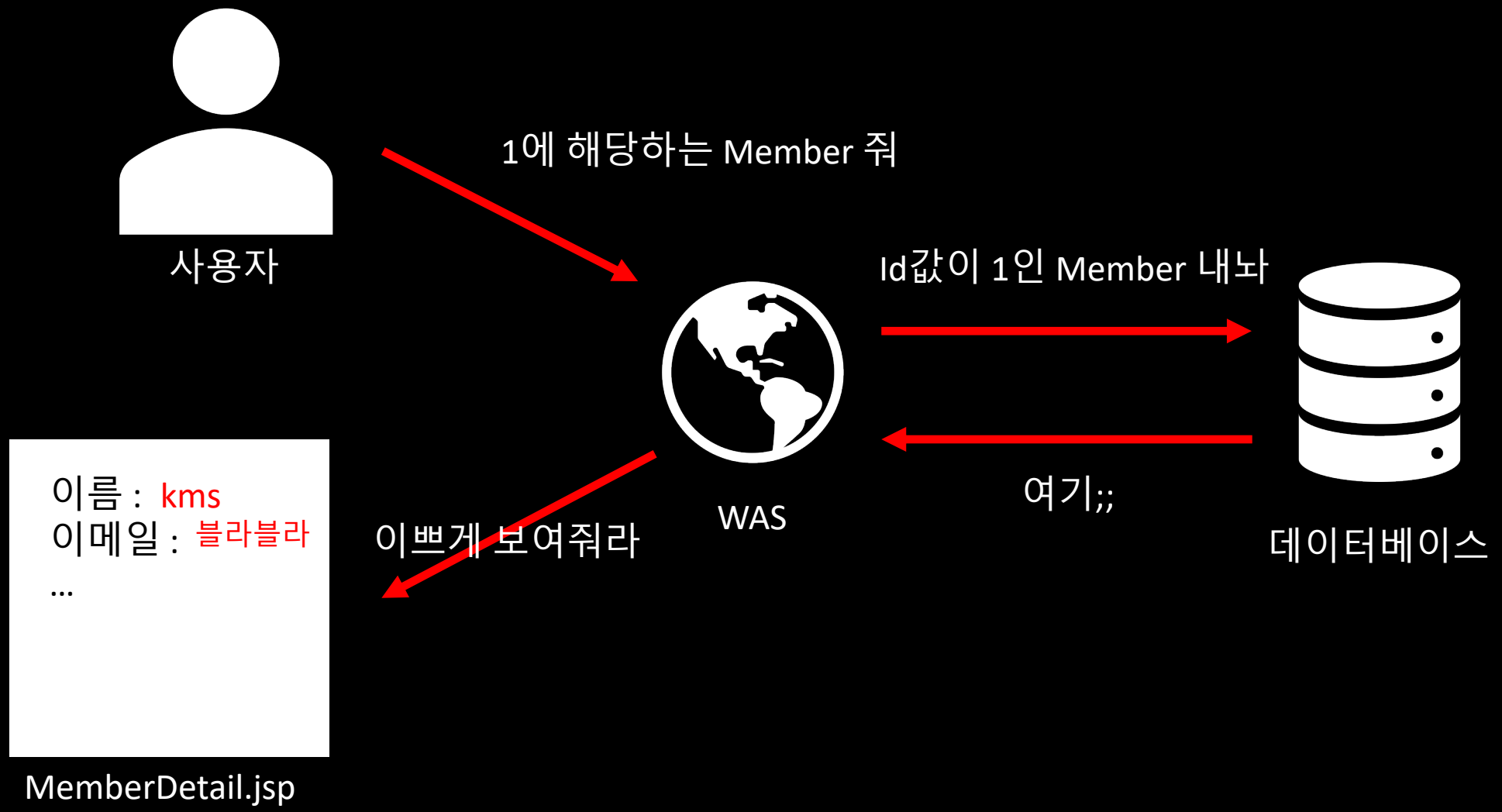


[정적 웹 애플리케이션]

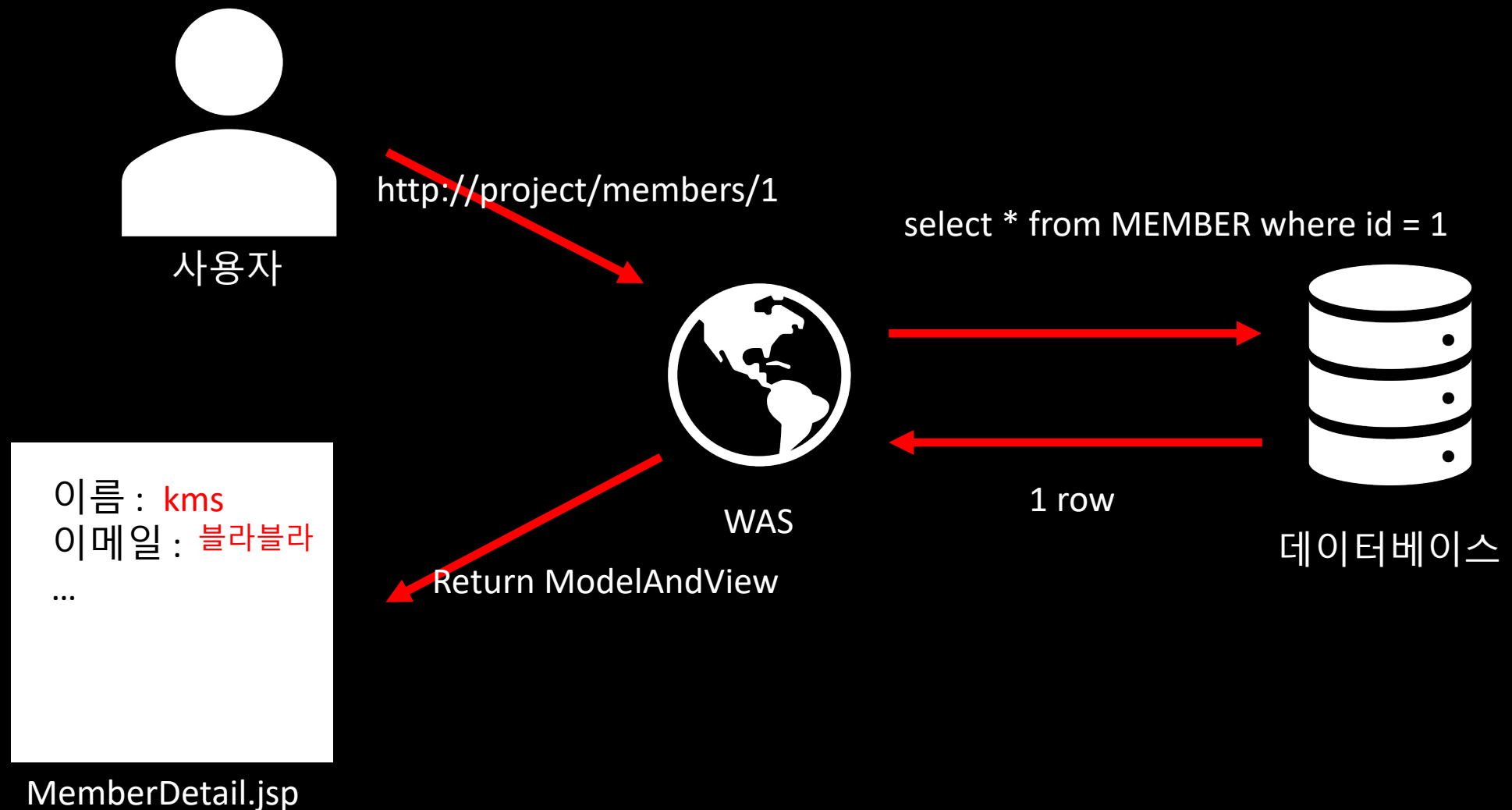


[동적 웹 애플리케이션]

2) @Path Variable



2) @Path Variable



2) @Path Variable

MemberDeailController.java

```java
@Controller
public class MemberDetailController {

    private MemberDao memberDao;

    public void setMemberDao(MemberDao memberDao) { this.memberDao = memberDao; }

    @GetMapping("/members/{id}")
    public String detail(@PathVariable("id") Long memId, Model model){
        Member member = memberDao.selectById(memId);
        if(member == null) throw new MemberNotFoundException();

        model.addAttribute( attributeName: "member", member);
        return "member/memberDetail";
    }
```

매핑 경로에서 중괄호로 감싸져 있는 부분이 경로 변수

{경로변수}에 담겨져 넘어오는 값을 @PathVariable 어노테이션으로 지정한 파라미터에 전달

파라미터에 넘어온 id값을 통해 DB에서 해당하는 데이터를 찾을 수 있음

2) @Path Variable

ControllerConfig.java

```java
@Bean
public MemberDetailController memberDetailController(){
    MemberDetailController controller = new MemberDetailController();
    controller.setMemberDao(memberDao);
    return controller;
}
```

컨트롤러를 스프링 컨테이너가 관리할 수 있도록 빈 객체로 등록

2) 익셉션 처리

3) 익셉션 처리

- 컨트롤러 단위 익셉션 처리
- 공통 익셉션 처리

3) 익셉션 처리

- 컨트롤러 단위 익셉션 처리

MemberDetailController.java

```java
@ExceptionHandler(TypeMismatchException.class)
public String handleTypeMismatchException() { return "member/invalidId"; }

@ExceptionHandler(MemberNotFoundException.class)
public String handleNotFoundException() { return "member/noMember"; }
```

- 컨트롤러 단위의 익셉션 처리는 해당 컨트롤러에서 @ExceptionHandler 어노테이션을 사용해 처리
- @ExceptionHandler어노테이션에 명시한 익셉션과 일치하는 익셉션이 발생하면 해당 메소드가 익셉션을 처리

3) 익셉션 처리

- 컨트롤러 단위 익셉션 처리

invalidId.jsp

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>에러</title>
</head>
<body>
    잘못된 요청입니다.
</body>
</html>
```

MemberDetailController.java

```java
@ExceptionHandler(TypeMismatchException.class)
public String handleTypeMismatchException() { return "member/invalidId"; }

@ExceptionHandler(MemberNotFoundException.class)
public String handleNotFoundException() { return "member/noMember"; }
```

noMember.jsp

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>에러</title>
</head>
<body>
    존재하지 않는 회원입니다.
</body>
</html>
```

3) 익셉션 처리

- 공통 익셉션 처리

CommonExceptionHandler.java

```java
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;

@ControllerAdvice("spring")
public class CommonExceptionHandler {

    @ExceptionHandler(RuntimeException.class)
    public String handleRuntimeException(){
        return "error/commonException";
    }
}
```

- 공통 익셉션처리는 전체 컨트롤러에서 작동할 수 있는 익셉션 처리 방식
- 익셉션 처리를 위한 별도의 클래스를 생성하고 @ControllerAdvice 어노테이션을 사용
- 하나의 익셉션에 대해 컨트롤러 단위와 공통 단위가 동시에 존재한다면, 우선순위는 컨트롤러 단위가 더 높음