작성자 : KJY97

스터디 7주차

2021.09.26

섹션 1: 웹 애플리케이션 이해

섹션 2: 서블릿

섹션 3: 서블릿, JSP, MVC 패턴

인프런 김영한 스프링 완전 정복 로드맵 이용

해당 PPT는 강의 내용을 요약, 정리하여 작성하였습니다.

잘못된 내용은 피드백주세요

목 차

Section 01

웹 애플리케이션 이해

요청/응답 메시지

회원 정보 관리 API 설계

http 주요 메서드 종류

http 헤더, 바디

host

해당 강의 내용 보다는 “http 웹 기본 지식” 중 일부 내용을 담았습니다.

요청 메시지

① ② ③

GET /search?q=hello&hl=ko HTTP/1.1

Host: www.google.com

Start-line 시작 라인

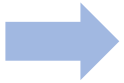
Header 헤더

Empty line 공백 라인(CRLF)

① HTTP 메서드
서버가 수행해야 할 동작 지정

② 요청 대상
절대경로[?쿼리] : /로 시작하는 경로

③ HTTP 버전



request-line

응답 메시지

① ② ③

```
HTTP/1.1 200 OK
Content-Type: text/html;charset=UTF-8
Content-Length: 3423

<html>
  <body> ... </body>
</html>
```

Start-line 시작 라인

Header 헤더 → HTTP 전송에 필요한 모든 부가정보

Empty line 공백 라인(CRLF)

Message body → 실제 전송할 데이터

HTML 문서, 이미지, 영상, JSON 등
Byte로 표현 가능한 모든 데이터 전송 가능

① HTTP 버전

② HTTP 상태 코드 : 요청 성공, 실패 나타냄

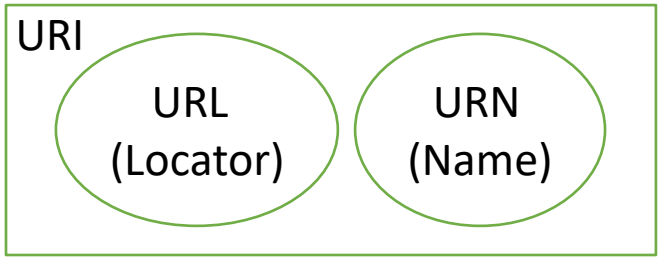
200: 성공 / 400: 클라이언트 요청 오류 / 500: 서버 내부 오류

③ 이유 문구 : 사람이 이해할 수 있는 짧은 상태 코드 설명 글

회원 정보 관리 API 설계

- API URI(Uniform Resource Identifier) 설계에서 가장 중요한 것

  ➔ 리소스 식별!!



- 리소스란?
  - 회원이라는 개념 자체
  - 식별하기 위해서는 리소스와 행위를 분리
    - 리소스: 회원 (명사) → 이것을 URI에 매핑
    - 행위: 조회, 등록, 삭제, 변경 (동사)
- 행위는 어떻게 구분할까?
  - HTTP 메서드 이용

HTTP 주요 메서드 종류

- GET
  - 리소스 조회
  - 쿼리 파라미터를 통해서 서버에 데이터 전달
  - 메시지 바디 사용x
- POST
  - 서버는 요청 데이터를 처리 (어떻게 처리할지 리소스마다 정해야 함)
  - 메시지 바디를 통해 요청 데이터 전달
  - 전달된 데이터는 신규 리소스 등록에 사용
- PUT
  - 리소스가 있으면 대체, 없으면 생성 (덮어쓰기)
  - POST와의 차이점은 클라이언트가 리소스 위치를 알고 URL 지정한다
- PATCH
  - 리소스 부분 변경
- DELETE
  - 리소스 삭제

회원 수정할 때 사용 가능하다

/members/100

PUT /members/100 HTTP/1.1
Content-Type: application/json

{
"username": "old",
"age": 50
}

리소스가 없다면, 신규 리소스 생성

{
"username": "old",
"age": 50
}

PUT /members/100 HTTP/1.1
Content-Type: application/json

{ "age": 50 }

리소스를 완전히 대체하기 때문에
username 필드가 없으면 그대로 대체

{ "age": 50, }

PATCH /members/100 HTTP/1.1
Content-Type: application/json

{ "age": 20 }

Username 필드가 없어도
리소스 부분 변경 가능

{
"username": "old",
"age": **20**
}

DELETE /members/100 HTTP/1.1
Host: localhost:8080

❌ 리소스 제거

클라이언트

서버

HTTP 헤더

- HTTP 전송에 필요한 모든 부가정보
  - 메시지 바디 내용, 크기, 서버 정보, 요청 클라이언트 등
- General 헤더, Request 헤더, Response 헤더

HTTP 바디

- 메시지 본문을 통해 표현 데이터 전달
- 표현이란 요청이나 응답에서 전달할 실제 데이터
- 표현 헤더는 표현 데이터를 해석할 수 있는 정보 제공
- 표현 헤더는 전송, 응답 둘다 사용

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 3423

<html>
   <body> … </body>
</html>
```

표현 헤더

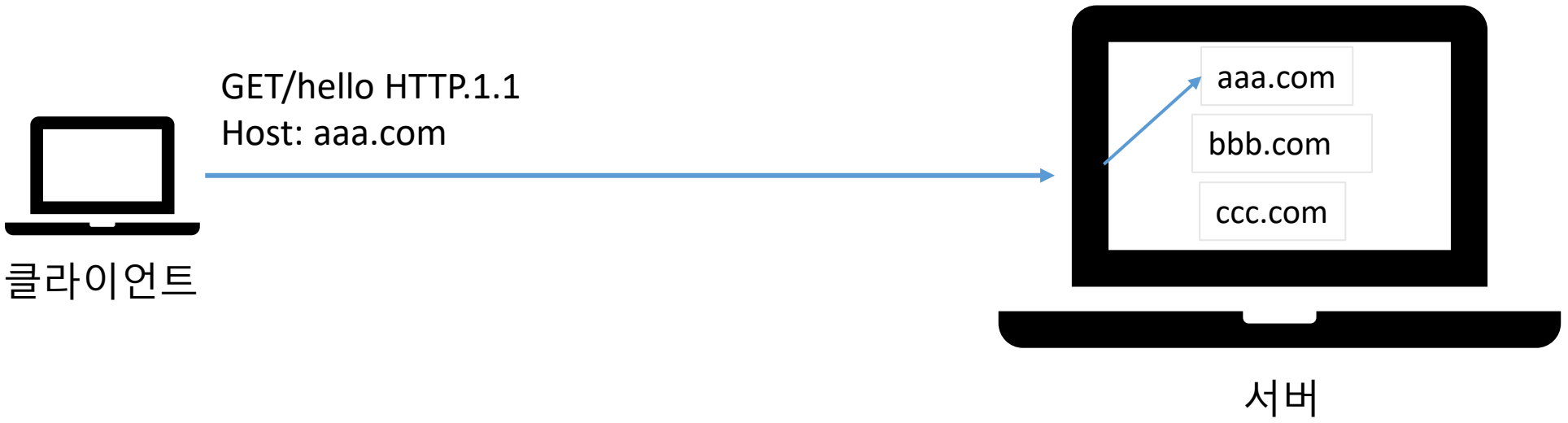
표현 데이터

HTTP 바디 - 협상

- Accept
  - 클라이언트가 선호하는 미디어 타입 전달
- Accept-Charset
  - 클라이언트가 선호하는 문자 인코딩
- Accept-Encoding
  - 클라이언트가 선호하는 압축 인코딩
- Accept-Language
  - 클라이언트가 선호하는 자연 언어
- 협상 헤더는 요청시에만 사용

Host

- 요청한 호스트 정보(도메인)
- 필수!!
- 하나의 서버, ip 주소가 여러 도메인을 처리해야 할 때 사용



Section 02

서블릿

HttpServletRequest

Http 요청 이용한 데이터 전달 방법 – GET, POST, 직접 요청

HttpServletResponse

HttpServletRequest

개발자가 HTTP 요청 메시지를 편리하게 사용할 수 있도록 대신 HTTP 요청 메시지 파싱
그 결과는 HttpServletRequest 객체에 담아서 제공

- Start line
  - HTTP 메소드 → request.getMethod()
  - URL → request.getRequestURL()
  - 쿼리 스트링 → request.getQueryString()
  - 스키마 → request.getScheme()
  - 프로토콜 → request.getProtocol()

- Header
  - 헤더 조회 → request.getHeader()

- Body
  - Form 파라미터 형식 조회
  - Message body 데이터 직접 조회
  - Request.getContentType(), request.getCharacterEncoding() 등

- 그 외 기타 정보

GET 쿼리 파라미터 http 요청 메시지를 통해 클라이언트에서 서버로 데이터 전달 방법1

메시지 바디 없이, URL의 쿼리 파라미터에 데이터 포함해서 전달
메시지 바디를 사용하지 않아서 Content-Type이 없다.
파라미터 이름이 중복되는 경우도 있으나 매우 희박하다.
Ex) 검색, 필터, 페이징

http://localhost:8080/request-param?username=hello&age=20 → request.getParameter() 사용

```
System.out.println("[전체 파라미터 조회] - start");
request.getParameterNames().asIterator()
                .forEachRemaining(paramName -> System.out.println(paramName + ": " + request.getParameter(paramName)));

System.out.println("[전체 파라미터 조회] - end");
```

하나의 파라미터 이름의 단 하나의 값 반환

```
[전체 파라미터 조회] - start
username: hello
age: 20
[전체 파라미터 조회] - end
```

http://localhost:8080/request-param?username=hello&username=kim&age=20 → request.getParameterValues() 사용

```
System.out.println("[이름이 같은 복수 파라미터 조회]");
String[] usernames = request.getParameterValues("username");
for (String name : usernames) {
    System.out.println("usernames = " + name);
}
```

```
[이름이 같은 복수 파라미터 조회]
usernames = hello
usernames = kim
```

HttpServletRequest.java

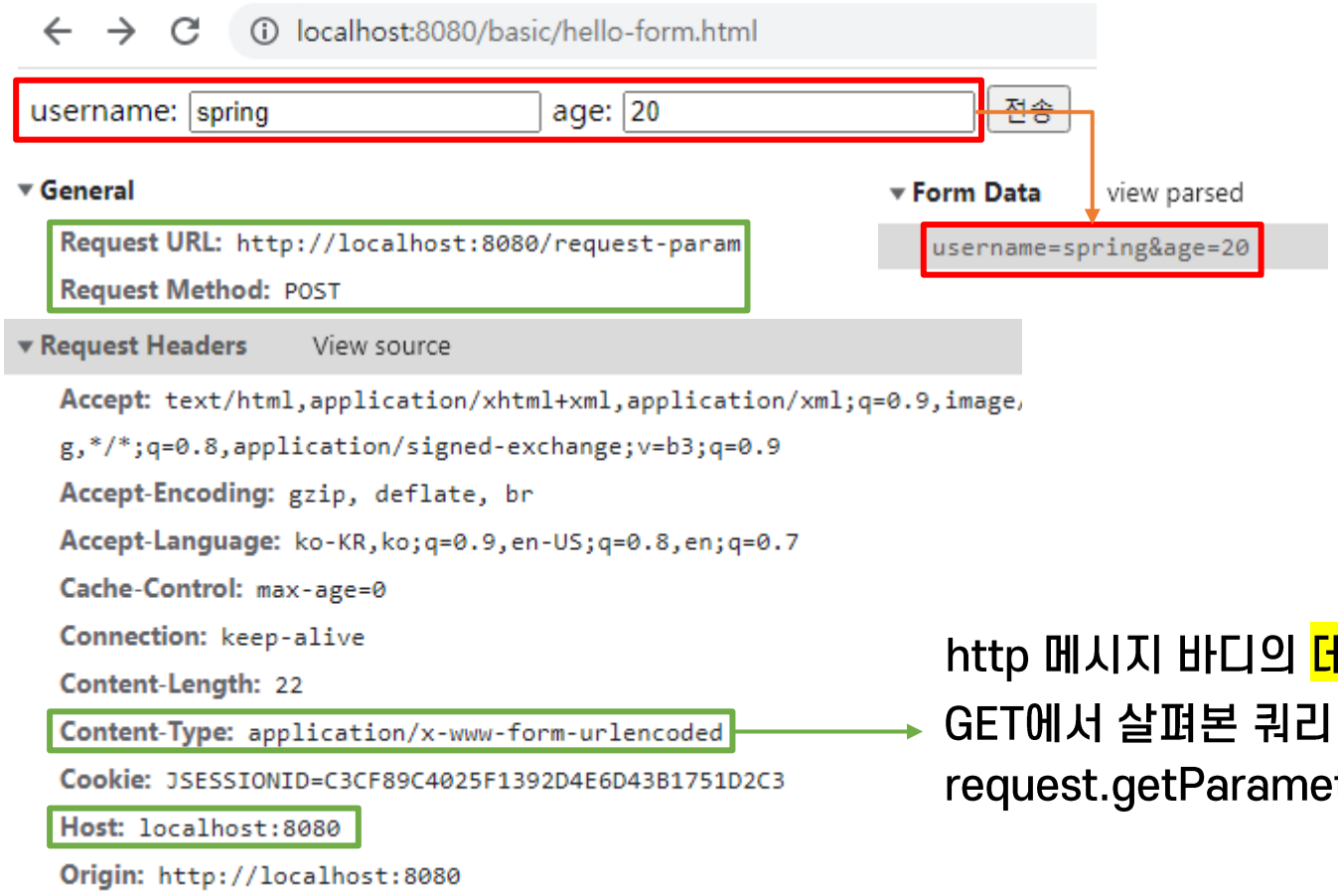
POST HTML Form 방식

http 요청 메시지를 통해 클라이언트에서 서버로 데이터 전달 방법2

메시지 바디에 쿼리 파라미터 형식으로 데이터 전달

Ex) 회원 가입, 상품 주문

클라이언트에서 서버로 데이터 전송



hello-form.html

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form action="/request-param" method="post">
    username: <input type="text" name="username" />
    age: <input type="text" name="age" />
    <button type="submit">전송</button>
</form>
</body>
</html>
```

Form 데이터를 바디로 전송할 때는 get, post만 지원

http 메시지 바디의 데이터 형식 지정하는 부분

GET에서 살펴본 쿼리 파라미터 형식과 동일

request.getParameter()로 조회 가능

Http 메시지 바디에 데이터 직접 담아서 요청

http 요청 메시지를 통해 클라이언트에서 서버로 데이터 전달 방법3

HTTP API에서 주로 사용.

과거에는 XML을 사용했지만, 현재는 주로 JSON 사용

Post, put, path 방식 모두 가능

```java
private ObjectMapper objectMapper = new ObjectMapper();

protected void service(...) {
    ServletInputStream inputStream = request.getInputStream();
    String messageBody = StreamUtils.copyToString(inputStream, StandardCharsets.UTF_8);

    JSON 형식으로 파싱하기 위한 객체
    HelloData helloData = objectMapper.readValue(messageBody, HelloData.class);
    System.out.println("helloData.username = " + helloData.getUsername());
    System.out.println("helloData.age = " + helloData.getAge());
}
```

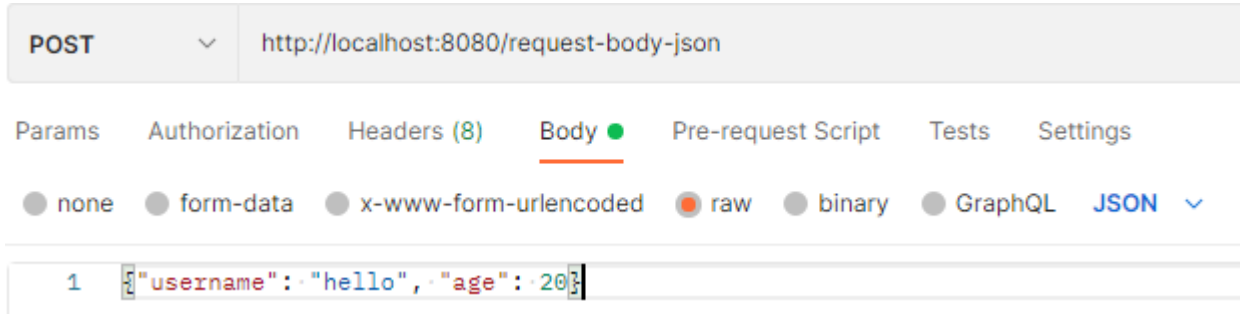
Json 결과를 파싱해서 자바 객체로 변환하기 위해 Jackson 라이브러리 사용

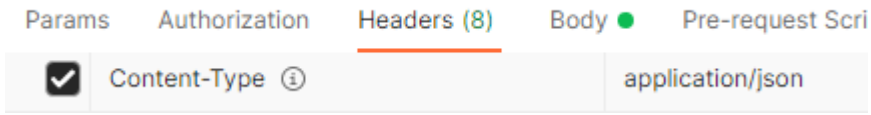
Lombok 사용

```java
@Getter @Setter
public class HelloData {

    private String username;
    private int age;
}
```

```
helloData.username = hello

helloData.age = 20
```

테스트를 위해서 매번 HTML form 만드는 거 귀찮다! ➔POSTMAN 사용





RequestBodyJsonServlet.java

HttpServletResponse

HTTP 응답 메시지를 편리하게 사용하도록 도와주는 객체

- HTTP 응답 메시지 생성
  - HTTP 응답코드 지정 → response.setStatus()
  - 헤더 생성 → response.setHeader()
  - 바디 생성 → PrintWrite write = response.getWriter()

- 편의 기능 제공
  - Content-Type → response.setContentType()
  - 쿠키 → Cookie 객체 사용
  - Redirect → response.sendRedirect()

- 응답 메시지 내용
  - HTTP 응답
    - HTML을 반환할 때는 response.setContentType("text/html")로 지정
  - HTTP API – MessageBody Json 응답
    - Json 반환할 때는 response.setContentType("application/json")로 지정
    - objectMapper.writeValueAsString() 사용하면 Json으로 문자 변경

```
//Content-Type: text/html;charset=utf-8
response.setContentType("text/html");
response.setCharacterEncoding("utf-8");

PrintWriter writer = response.getWriter();
writer.println("<html>");
writer.println("<body>");
writer.println("    <div>안녕?</div>");
writer.println("</body>");
writer.println("</html>");
```

```
HelloData helloData = new HelloData();
helloData.setUsername("kim");
helloData.setAge(20);

//{"username":"kim", "age":20}
String result =
objectMapper.writeValueAsString(helloData);
response.getWriter().write(result);
```

Section 03

서블릿, JSP, MVC 패턴

회원 관리 웹 애플리케이션 만들기

Servlet으로 만들기

```java
response.setContentType("text/html");
response.setCharacterEncoding("utf-8");

PrintWriter w = response.getWriter();
w.write("<html>");
w.write("<head>");
w.write(" <meta charset=\"UTF-8\">");
w.write(" <title>Title</title>");
w.write("</head>");
w.write("<body>");
w.write("<a href=\"/index.html\">메인</a>");
w.write("<table>");     //테이블 만들기
w.write("   <thead>");
w.write("       <th>id</th>");
w.write("       <th>username</th>");
w.write("       <th>age</th>");
w.write("   </thead>");
w.write("   <tbody>");

for (Member member : members) {
    w.write(" <tr>");
    w.write(" <td>" + member.getId() + "</td>");
    w.write(" <td>" + member.getUsername() + "</td>");
    w.write(" <td>" + member.getAge() + "</td>");
    w.write(" </tr>");
}

w.write("   </tbody>");
w.write("</table>");
w.write("</body>");
w.write("</html>");
```

MemberListServlet.java 일부

JAVA 코드에 섞인 view 화면 작업

JAVA 코드에 섞여서 지저분하고 복잡하다!

무엇보다 너무 비효율적

➔ HTML 문서에 동적으로 변경할 부분만 자바 코드로 넣는 템플릿 엔진 사용!

Ex) JSP, Thymeleaf 등

JSP로 만들기

```jsp
<%@ page import="hello.servlet.domain.member.MemberRepository" %>
<%@ page import="hello.servlet.domain.member.Member" %>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%
    // request, response 사용 가능
    MemberRepository memberRepository = MemberRepository.getInstance();

    System.out.println("save.jsp");
    String username = request.getParameter("username");
    int age = Integer.parseInt(request.getParameter("age"));

    Member member = new Member(username, age);
    System.out.println("member = " + member);
    memberRepository.save(member);
%>
```

JAVA 코드 입력
(회원 저장 비즈니스 로직)

```jsp
<html>
<head>
    <meta charset="UTF-8">
</head>
<body>
성공
<ul>
    <li>id=<%=member.getId()%></li>
    <li>username=<%=member.getUsername()%></li>
    <li>age=<%=member.getAge()%></li>
</ul>
<a href="/index.html">메인</a>
</body>
</html>
```

자바 코드 출력

save.jsp

HTML 코드
(뷰 영역)

View 생성하는 HTML 부분 → 깔끔!

동적으로 변경 필요한 부분 → 자바 코드 적용

But..

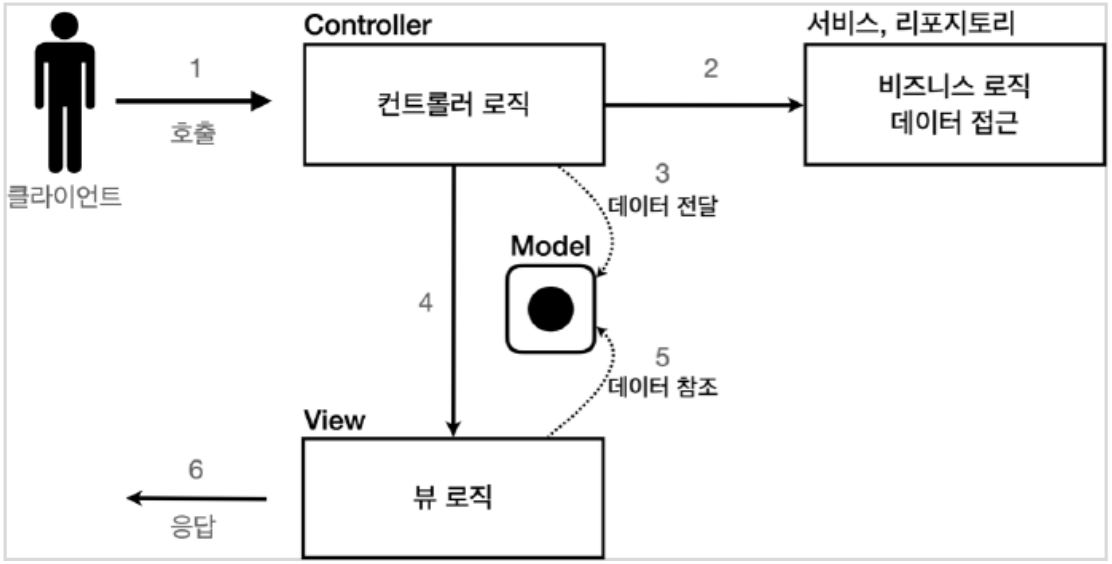
JAVA 코드, 데이터 조회 등 다양한 코드가 JSP에 노출

JSP가 너무 많은 역할을 수행하고 있어 유지보수 어려움

코드가 길어진다면... 끔찍

비즈니스 로직과 UI의 변경 주기가 다르다!!

MVC 패턴으로 만들기



1. http 요청 받아서 파라미터 정보 체크
2. 비즈니스 로직 실행
3. Model에 데이터 보관
4. 비즈니스 실행 결과를 뷰에 던진다.
   뷰는 랜더링에만 집중
5. Model에서 데이터 참조

servlet으로 controller 구현

```
① String username = request.getParameter("username");
  int age = Integer.parseInt(request.getParameter("age"));

② Member member = new Member(username, age);
  memberRepository.save(member);

③ request.setAttribute("member", member);

④ String viewPath = "/WEB-INF/views/save-result.jsp";
  RequestDispatcher dispatcher = request.getRequestDispatcher(viewPath);
  dispatcher.forward(request, response);
```

MvcMemberSaveServlet.java

jsp로 view 구현

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <meta charset="UTF-8">
</head>
<body>
성공
<ul>
    <li>id=${member.id}</li>
⑤   <li>username=${member.username}</li>
    <li>age=${member.age}</li>
</ul>
<a href="/index.html">메인</a>
</body>
</html>
```

save-result.jsp