

Spring week 7

유채린

chaerin.du.ub@gmail.com



해당 내용은 인프런 강의를 정리한 내용입니다.
잘못된 내용에 대하여 피드백주시면 감사하겠습니다!

(인프런 아이콘 클릭 시, 참고한 인프런 강의 사이트로 이동합니다.)

Index

- 웹 어플리케이션 이해
- 서블릿
- JSP, MVC 패턴

웹 어플리케이션 이해

웹 시스템, WAS, 서블릿 컨테이너, 멀티 쓰레드, CSR, SSR

웹 서버 (Web Server)

- HTTP 기반으로 동작
- 정적 리소스 제공
 - HTML, CSS, JS, 이미지, 영상
- 웹 서버도 프로그램을 실행하는 기능 포함하기도 함

EX) NGINX, APACHE

웹 애플리케이션 서버 (WAS)

- HTTP 기반으로 동작
- 애플리케이션 로직
 - 동적 HTML, HTTP API(REST API), 서블릿, JSP, 스프링 MVC
- 웹 서버 기능 제공 (정적 리소스 제공 가능)
- 애플리케이션 코드를 실행하는데 더 특화

EX) Tomcat, Jetty, Undertow

웹 시스템 구성 – Web Server, WAS, DB

WAS와 DB만으로 시스템 구성이 가능하지만, WAS 서버 과부하 문제가 생기며 WAS 장애 발생 시 오류 화면 노출 불가능 문제가 발생한다.

web server를 시스템 구성에 추가해줌으로써 효율적인 리소스 관리가 가능해지며 WAS나 DB 장애 시, 웹 서버가 오류 화면(오류화면 HTML)을 클라이언트에게 제공할 수 있다.

웹 어플리케이션 사용 시, 서버에서 처리해야 하는 업무

- 서블릿을 지원하는 WAS를 사용함으로써,
개발자는 비즈니스 로직 (초록색 박스)만 개발하면 된다!
- 개발자는 HTTP 스펙을 매우 편리하게 사용할 수 있다.
 - WAS는 request, response 객체를 새로 만들어서 서블릿 객체를 호출
 - 개발자는 request 객체에서 HTTP 요청 정보를 편리하게 꺼내서 사용
 - 개발자는 response 객체에 HTTP 응답 정보를 편리하게 입력
 - WAS는 response 객체에 담겨있는 내용으로 HTTP 응답 정보를 생성

- 서버 TCP/IP 연결 대기, 소켓 연결
- HTTP 요청 메시지를 파싱해서 읽기
- POST 방식, /save URL 인지
- Content-Type 확인
- HTTP 메시지 바디 내용 파싱
 - username, age 데이터를 사용할 수 있게 파싱
- 저장 프로세스 실행
- 비즈니스 로직 실행
 - 데이터베이스에 저장 요청
- HTTP 응답 메시지 생성 시작
 - HTTP 시작 라인 생성
 - Header 생성
 - 메시지 바디에 HTML 생성에서 입력
- TCP/IP에 응답 전달, 소켓 종료

```
@WebServlet(name = "helloServlet", urlPatterns = "/hello") URL이 호출되면 서블릿 코드가 실행된다.
```

```
public class HelloServlet extends HttpServlet {
```

```
@Override
```

HTTP 요청 메시지 기반으로
생성되어 자동 파싱됨

HTTP 응답 정보로 개발자는 response 에
객체를 넣어 넘겨주기만 하면 된다.

```
protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
```

서블릿 컨테이너

- Tomcat처럼 서블릿을 지원하는 WAS를 서블릿 컨테이너라고 한다.
- 서블릿 객체를 생성, 초기화, 호출, 종료하는 생명주기 관리한다.
 - WAS 올라갈 때와 내려갈 때 생성, 종료 해준다.
- 서블릿 객체는 싱글톤으로 관리된다.
 - 최초 로딩시점에 서블릿 객체를 미리 만들어두고 재활용한다.
 - 모든 클라이언트 요청은 동일한 서블릿 객체 인스턴스에 접근
 - 공유 변수(멤버 변수) 사용 -> 주의하자
- JSP도 서블릿으로 변환되어서 사용
- 동시 요청을 위한 멀티 쓰레드 처리 지원 (동시 접속자 관리)
 - 개발자는 멀티 쓰레드 관련 코드를 신경 쓰지 않아도 된다. (비즈니스 로직만 신경쓰자)
 - 싱글톤 객체(서블릿, 스프링 빈) 주의해서 사용하자.

쓰레드

- 프로세스 내에서 실행되는 단위
- 애플리케이션 코드를 순차적으로 실행하는 것
- 쓰레드는 한 번에 하나의 코드 라인만 수행
- 동시 처리 필요하다면 쓰레드 추가 생성 필요
- 생성 비용이 매우 비싸다.
 - 고객의 요청이 올 때마다 생성하게 되면 응답 속도 느려진다. (CPU 많이 씀)
- Context switching 비용 발생
 - 동작 중인 프로세스가 대기 중이 되면서 해당 프로세스의 상태(Context)를 보관하고, 대기하고 있던 다음 순서의 프로세스가 동작하면서 이전에 보관했던 프로세스의 상태를 복구하는 작업
- 쓰레드 생성에 제한이 없다 (CPU, 메모리 임계점 넘는 경우 서버 죽음)
 - WAS에서 쓰레드 풀 크기를 설정해준다.

쓰레드 풀

- 쓰레드 풀에 미리 쓰레드를 만들어 놔서 관리한다.
 - 톰캣은 default로 최대 200개 설정 가능하다.
 - 쓰레드가 필요한 경우, 쓰레드 풀에서 이미 생성되어 있는 쓰레드를 꺼내서 사용한다.
 - 사용 후, 쓰레드 풀에 해당 쓰레드를 반납한다.
 - 최대 쓰레드가 모두 사용 중인 경우 대기 하거나 요청을 거절한다.
-
- 쓰레드를 생성하고 종료하는 비용(CPU)이 절약되고 응답시간이 빠르다.
 - 너무 많은 요청이 들어와도 기존 요청은 안전하게 처리가 가능하다.
-
- 쓰레드 풀 적정 크기는 애플리케이션 로직의 복잡도, CPU, 메모리, IO 리소스 상황에 따라 모두 다르다.

CSR(Client Side Rendering)

- HTML 결과를 자바스크립트를 사용해 웹 브라우저에서 동적으로 생성해서 적용
- 주로 동적인 화면에 사용
- HTTP API(REST API)를 통해서 데이터(주로 JSON)를 전달받아 화면을 그린다.

EX) React.js, Vue.js

SSR(Server Side Rendering)

- HTML 최종 결과를 서버에서 만들어서 웹 브라우저에 전달
- 주로 정적인 화면에 사용
- 이미 생성된 리소스 파일 또는 WAS에서 JSP, 타임리프를 이용한 동적 HTML 파일

EX) JSP, 타임리프

서블릿

스프링 부트 서블릿 환경 구성

- 스프링 부트는 서블릿을 직접 등록해서 사용할 수 있는 애노테이션을 지원한다.
@ServletComponentScan

```
package hello.servlet;

import ...           자동으로 현재 패키지 포함 하위 패키지까지
                     자동으로 서블릿으로 등록할 수 있도록 도와준다.

@WebServletScan // 서블릿 자동 등록
@SpringBootApplication
public class ServletApplication {

    public static void main(String[] args) { SpringApplication.run(ServletApplication.class, args); }

}
```

/WEB-INF

- 웹 서버가 사용하는 파일이 들어있는 중요한 디렉토리
- 외부에서 클라이언트가 곧바로 접근할 수 없도록 막아 두었다.
- 클라이언트에게 서블릿 접근할 수 있게 하기 위해서는 서버에서 서블릿 접근 경로를 지정해야 한다.
→ @WebServlet 또는 web.xml

서블릿 접근 경로 지정 방법: 애노테이션

@WebServlet

- @WebServlet(name = "서블릿 이름", urlPatterns="URL 매핑")
- HTTP 요청을 통해 매핑된 URL이 호출되면 서블릿 컨테이너는 service 메서드를 실행한다.

```
@WebServlet(name = "helloServlet", urlPatterns = "/hello")  
public class HelloServlet extends HttpServlet {  
  
    @Override  
    protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
  
        System.out.println("HelloServlet.service");  
    }  
}
```

name과 urlPatterns는 다른 servlet과 겹치면 안된다.

@WebServlet이 없었을 때에는 web.xml을 통해 서블릿 설정을 했었다.

web.xml에서 서블릿 지정할 때, 서블릿 name과 class를 명시해줬으며, 해당 서블릿에 할당될 url pattern을 추가할 때 url 패턴이 어떤 서블릿에 속하는지 식별하기 위해 서블릿 name을 사용했었다.

@WebServlet은 서블릿 class에 붙여서 사용하기 때문에 굳이 서블릿 name을 명시해줄 필요가 없어서 선택사항이 되었으며, name 속성 따로 명시하지 않을 경우 해당 클래스의 이름을 name속성으로 사용한다.

@WebServlet은 WebServletHandler에 의해 처리되는데 이때 handler가 @WebServlet의 name 속성을 사용하여 BeanDefinition을 만들게 된다.

서블릿 접근 경로 지정하는 또 다른 방법: web.xml

web.xml

- 브라우저가 JAVA Servlet에 접근하기 위해 WAS에 필요한 정보를 알려주는 XML형식의 파일
 - 웹 어플리케이션 환경 설정 파일
- 웹 어플리케이션은 하나의 web.xml 을 가져야 하고, 위치는 WEB-INF 폴더 아래에 있다.
- web.xml 에서 설정된 정보들은 웹 어플리케이션 시작 시 메모리에 로딩된다.

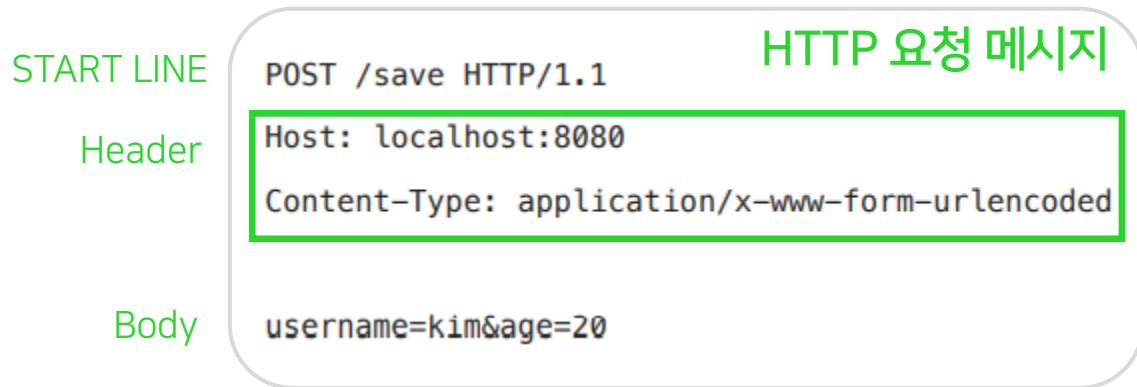
- 작성되는 내용

- servletContext의 초기 변수
- 서블릿 및 jsp에 대한 정의 및 매핑
- Mimetype 매핑
- Session 유효시간
- Welcome file list

```
<web-app>
  <servlet>
    <servlet-name>HelloWorld</servlet-name>
    <servlet-class>servlet.example.createHelloWorldExample
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>HelloWorld</servlet-name>
    <url-pattern>/HelloWorldExample/*</url-pattern>
  </servlet-mapping>
</web-app>
```

HttpServletRequest

- 서버릿은 개발자가 HTTP 요청 메시지를 편리하게 사용할 수 있도록 개발자 대신에 HTTP 요청 메시지를 파싱한다. 그리고 그 결과를 HttpServletRequest 객체에 담아서 제공한다.



- START LINE
HTTP 메소드, URL, 쿼리 스트링, 스키마, 프로토콜
- 헤더
host, content-type 등
- 바디
form 파라미터 형식 조회 또는 message body 데이터 직접 조회(JSON)

- 여러 부가 기능
 - 임시 저장소 기능: 해당 HTTP 요청이 끝날 때 까지 유지되는 임시 저장소 기능
request.setAttribute(name, value), request.getAttribute(name)
 - 세션 관리 기능: 세션 관리 기능 request.getSession(create: true)

HTTP 요청 메시지를 통한 데이터 전달 방법

1. GET - 쿼리 파라미터

- /URL?username=hello&age=20 URL에 ?를 시작으로 파라미터 설정해서 보낸다. 추가 파라미터는 &로 구분한다.
- 메시지 바디 없이, URL의 쿼리 파라미터에 데이터를 포함해서 전달
- Ex) 검색, 페이징 등

2. POST - HTML Form

- content-type: application/x-www-form-urlencoded
- 메시지 바디에 쿼리 파라미터 형식으로 전달 username=hello&age=20
- Ex) 회원 가입, 상품 주문, HTML Form 이용

content-type은 HTTP 메시지 바디의 데이터 형식을 지정한다.
HTTP 메시지 바디에 해당 데이터를 포함해서 보내기 때문에
Body에 포함된 데이터가 어떤 형식인지 content-type을 꼭 지정해야 한다.

```
<form action="/request-param" method="post">
  username: <input type="text" name="username" />
  age: <input type="text" name="age" />
  <button type="submit">전송</button>
</form>
```

클라이언트(웹 브라우저) 입장에서는 두 방식에 차이가 있지만,
서버 입장에서는 둘의 형식이 동일하므로, request.getParameter() 로 편리하게 구분없이 조회할 수 있다.
정리하면 request.getParameter() 는 GET URL 쿼리 파라미터 형식도 지원하고, POST HTML Form 형식도 둘 다 지원한다.

3. HTTP message body에 데이터를 직접 담아 요청

- HTTP API에서 주로 사용, JSON, XML, TEXT
- 데이터 형식은 주로 JSON 사용

Content-type: application/json
JSON 결과를 파싱해서 사용할 수 있는 자바 객체로 변환하려면
Jackson, Gson 같은 JSON 변환 라이브러리를 추가해서 사용해야 한다.
스프링 부트로 Spring MVC를 선택하면
기본으로 Jackson 라이브러리(ObjectMapper)를 함께 제공한다.

HttpServletResponse

- HTTP 응답코드 (200, 400, 404, 500, ..), 헤더, 바디
- 편의 기능 (Content-type, Cookie, redirect)

```
@Override
protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // [status-line]
    response.setStatus(HttpServletResponse.SC_OK); setStatus 응답코드

    // [response-headers] setHeader 헤더 - Content-Type, Cache-Control
    response.setHeader( name: "Content-Type", value: "text/plain;charset=utf-8");
    response.setHeader( name: "Cache-Control", value: "no-cache, no-store, must-revalidate"); cache 완전히 무효화 함
    response.setHeader( name: "Pragma", value: "no-cache"); 과거 버전까지 cache 없앴
    response.setHeader( name: "my-header", value: "hello"); 내가 원하는 헤더도 가능 (커스터마이징 가능)

    // [header 편의 메서드]
    content(response);
    cookie(response);
    redirect(response);

    // [message body]
    PrintWriter writer = response.getWriter(); body
    writer.println("OK");
}
```

HttpServletResponse - 편의 기능

1. Content

```
//Content-Type: text/plain;charset=utf-8
//Content-Length: 2

response.setHeader( name: "Content-Type", value: "text/plain;charset=utf-8");
response.setContentType("text/plain");
response.setCharacterEncoding("utf-8");
```

2. Cookie

```
//Set-Cookie: myCookie=good; Max-Age=600;

response.setHeader( name: "Set-Cookie", value: "myCookie=good; Max-Age=600");

Cookie cookie = new Cookie( name: "myCookie", value: "good");
cookie.setMaxAge(600); //600초
response.addCookie(cookie);
```

3. redirect

```
//Status Code 302
//Location: /basic/hello-form.html

response.setStatus(HttpServletResponse.SC_FOUND); //302
response.setHeader( name: "Location", value: "/basic/hello-form.html");

response.sendRedirect( location: "/basic/hello-form.html");
```

HTTP 응답 데이터

1. 단순 텍스트 응답

- `PrintWriter writer.println("ok")`

2. HTML 응답

- `Content-type: text/html` 지정해야 함

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

HTML

3. HTTP API – MessageBody JSON 응답

- HTTP 응답으로 JSON을 반환할 때는 `content-type`을 `application/json` 로 지정해야 한다.
Jackson 라이브러리가 제공하는 `objectMapper.writeValueAsString()` 를 사용하면 객체를 JSON 문자로 변경할 수 있다.

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

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

HTTP API
(REST API)

JSP, MVC 패턴

템플릿 엔진

- HTML 문서에 동적으로 변경해야 하는 부분만 자바 코드를 적용해서 동적으로 변경할 수 있다.
- JSP, Thymeleaf, Freemarker, Velocity 등
- JSP 라이브러리

```
dependencies {  
    implementation 'org.springframework.boot:spring-boot-starter-web'  
    //JSP 추가 시작  
    implementation 'org.apache.tomcat.embed:tomcat-embed-jasper'  
    implementation 'javax.servlet:jstl'  
    //JSP 추가 끝  
}
```

build.gradle

JSP

- JSP는 성능과 기능면에서 다른 템플릿 엔진과의 경쟁에서 밀리면서, 점점 사장되어 가는 추세이다.

```
*.jsp files are supported by IntelliJ IDEA Ultimate
import
1  <%@ page import="hello.servlet.domain.member.MemberRepository" %>
2  <%@ page import="hello.servlet.domain.member.Member" %>
3  <%@ page contentType="text/html; charset=UTF-8" language="java" %>
4  <%
5      // request, response 사용 가능      JSP 문서 상징
6      MemberRepository memberRepository = MemberRepository.getInstance();
7
8      System.out.println("save.jsp");
9      String username = request.getParameter("username");
10     int age = Integer.parseInt(request.getParameter("age"));
11
12     Member member = new Member(username, age);
13     System.out.println("member = " + member);
14     memberRepository.save(member);
15 %>      자바 코드 입력
16 <html>
17 <head>
18     <meta charset="UTF-8">
19 </head>
20 <body>
21 성공
22 <ul>
23     <li>id=<%=member.getId()%></li>      자바 코드 출력
24     <li>username=<%=member.getUsername()%></li>
25     <li>age=<%=member.getAge()%></li>
26 </ul>
27 <a href="/index.html">메인</a>
28 </body>
29 </html>
```

<%@ page contentType="text/html; charset=UTF-8" language="java" %>

: JSP문서임을 의미

<%@ page import="hello.servlet.domain.member.Member" %>

: 자바의 import문과 동일

<% ~ %>

: 자바 코드 입력

<%= ~ %>



: 자바 코드 출력

JSP를 사용한 덕분에 뷰를 생성하는 HTML 작업을 깔끔하게 가져가고, 중간중간 동적으로 변경이 필요한 부분에만 자바 코드를 적용했다.

이렇게 해도 해결되지 않는 몇가지 고민이 남는다.

JAVA 코드, 데이터를 조회하는 리포지토리 등 다양한 코드가 모두 JSP에 노출되어 있다.

JSP가 너무 많은 역할을 한다.

→  지옥의 유지보수 

→ MVC 패턴의 등장

MVC 패턴 (Model View Controller)

- 비즈니스 로직은 서블릿처럼 다른 곳에서 처리하고, JSP는 목적에 맞게 HTML로 화면(View)을 그리는 일에 집중하도록 하자
- 컨트롤러
HTTP 요청을 받아서 파라미터를 검증하고, 비즈니스 로직을 실행한다.
그리고 뷰에 전달할 결과 데이터를 조회해서 모델에 담는다.
- 모델
뷰에 출력할 데이터를 담아둔다. 뷰가 필요한 데이터를 모두 모델에 담아서 전달해주는 덕분에 뷰는 비즈니스 로직이나 데이터 접근을 몰라도 되고, 화면을 렌더링 하는 일에 집중할 수 있다.
- 뷰
모델에 담겨있는 데이터를 사용해서 화면을 그리는 일에 집중한다.
여기서는 HTML을 생성하는 부분을 말한다.

MVC 패턴 (Model View Controller) - Controller

```
@WebServlet(name = "mvcMemberSaveServlet", urlPatterns = "/servlet-mvc/members/save")
public class MvcMemberSaveServlet extends HttpServlet {

    private MemberRepository memberRepository = MemberRepository.getInstance();

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

        String username = request.getParameter(name: "username");
        int age = Integer.parseInt(request.getParameter(name: "age"));

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

        // Model에 데이터를 보관한다.
        request.setAttribute(name: "member", member);

        View(JSP)로 던지기
        String viewPath = "/WEB-INF/views/save-result.jsp";
        RequestDispatcher dispatcher = request.getRequestDispatcher(viewPath); Controller에서 view 이동할 때 사용됨
        dispatcher.forward(request, response); Servlet에서 JSP 호출 (서버 내부에서 호출-클라이언트가 인지하지 못함)
    }
}
```

MVC 패턴 (Model View Controller) - View

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<html>
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
  <a href="/index.html">메인</a>
  <table>
    <thead>
      <th>id</th>
      <th>username</th>
      <th>age</th>
    </thead>
    <tbody>
      <%= request.getAttribute("member") %>
      <c:forEach var="item" items="${members}">
        <tr>
          <td>${item.id}</td>
          <td>${item.username}</td>
          <td>${item.age}</td>
        </tr>
      </c:forEach>
    </tbody>
  </table>
```

request의 attribute에 담긴 데이터를 편리하게 조회할 수 있다.

jstl 안 쓰는 방식도 가능하지만 지저분하다.

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

MVC 패턴 한계

- forward 중복
- viewPath 중복
 - prefix: /WEB-INF/views/
 - Suffix: .jsp
- 사용하지 않는 코드
 - HttpServletRequest, HttpServletResponse
 - 테스트 케이스 작성 어려움
- 공통 처리의 어려움

→ **프론트 컨트롤러(Front Controller)** 패턴 도입 (입구를 하나로)

```
@WebServlet(name = "mvcMemberSaveServlet", urlPatterns = "/servlet-mvc/members/save")
public class MvcMemberSaveServlet extends HttpServlet {

    private MemberRepository memberRepository = MemberRepository.getInstance();

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

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

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

        // Model에 데이터를 보관한다.
        request.setAttribute("member", member);

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

사용하지 않는 코드

viewPath 중복

forward 중복

감사합니다.