2. 서블릿

1)프로젝트 생성

스프링 부트 스타터

Project : Gradle Project

Packaging : War(JSP를 실행하기 위해서 필요함)

Dependencies : Spring Web, Lombok

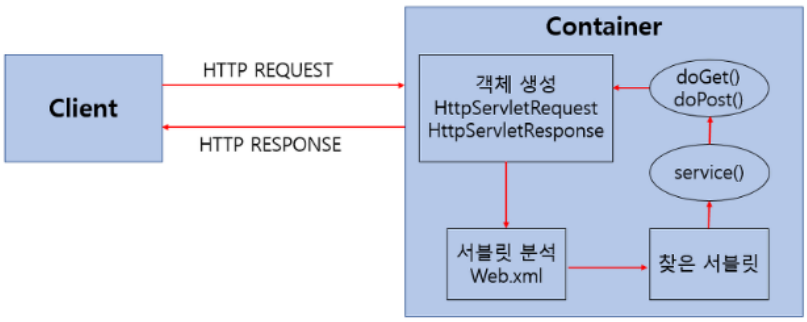
- 동작 확인
  - 기본 메인 클래스 실행(`ServletApplication.main()`)
  - http://localhost:8080 호출해서 Whitelabel Error Page가 나오면 정상 동작

2)Hello 서블릿

`서블릿` : 클라이언트의 요청을 처리하고, 그 결과를 반환하는 Servlet 클래스의 구현 규칙을 지켜 동적 웹 페이지를 만들어주는 자바 웹 프로그래밍 기술, HTTP 스펙을 편리하게 사용 가능함

`서블릿 컨테이너` : 서블릿을 관리해주는 컨테이너, 클라이언트의 요청(Request)을 받아주고 응답(Response)할 수 있게 웹 서버와 소켓 통신하여 JSP와 Servlet이 장동하는 환경을 제공함 Ex) 톰캣

[Servlet 동작 방식]



1. 사용자(클라이언트)가 URL을 입력하면 HTTP Request가 Servlet Container로 전송합니다.
2. 요청을 전송받은 Servlet Container는 HttpServletRequest, HttpServletResponse 객체를 생성합니다.
3. web.xml을 기반으로 사용자가 요청한 URL이 어느 서블릿에 대한 요청인지 찾습니다.
4. 해당 서블릿에서 service메소드를 호출한 후 클리아언트의 GET, POST여부에 따라 doGet() 또는 doPost()를 호출합니다.
5. doGet() or doPost() 메소드는 동적 페이지를 생성한 후 HttpServletResponse객체에 응답을 보냅니다.
6. 응답이 끝나면 HttpServletRequest, HttpServletResponse 두 객체를 소멸시킵니다.

HTTP 요청, HTTP 응답 메시지

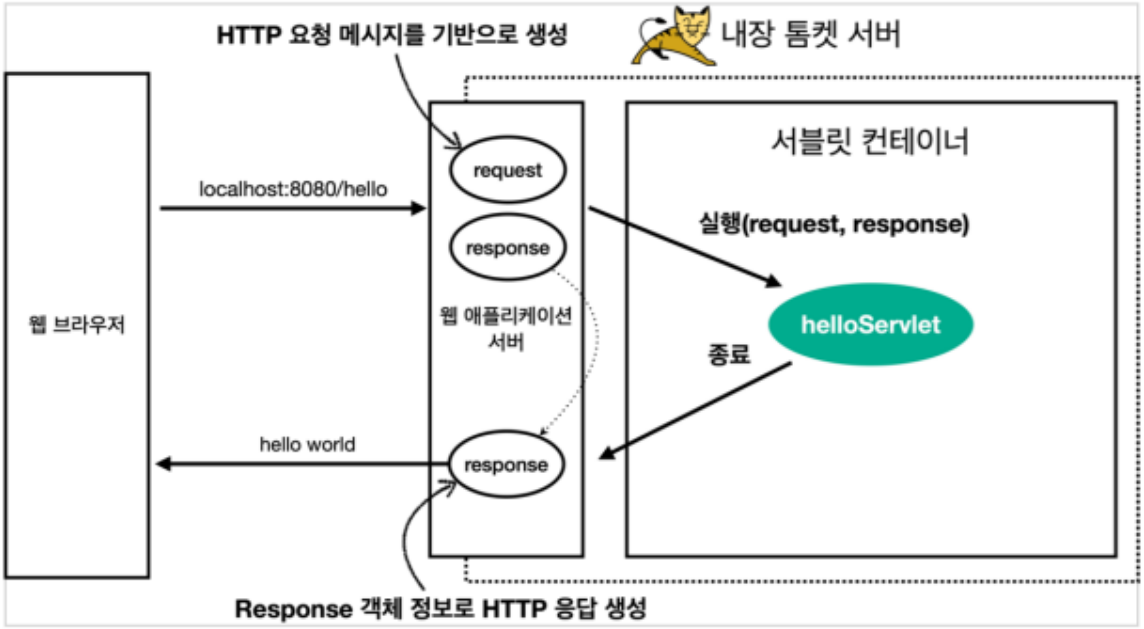
[HTTP 요청]

```
GET /hello?username=world HTTP/1.1
Host: localhost:8080
```

[HTTP 응답]

```
HTTP/1.1 200 OK
Content-Type: text/plain;charset=utf-8
Content-Length: 11

hello world
```

웹 애플리케이션 서버의 요청 응답 구조



스프링 부트 환경에서 서블릿 등록하고 사용해보자.

서블릿 코드->클래스 파일 + 톰캣 = 스프링 부트

-> 스프링 부트는 톰캣 서버를 내장하고있어 서블릿 코드 실행 시 별도로 WAS 설치 불필요

스프링 부트 서블릿 환경 구성

1. 서블릿 자동 등록 설정

```java
@ServletComponentScan //서블릿 자동 등록
@SpringBootApplication
public class ServletApplication {

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

2. 서블릿 등록

```
@WebServlet(name = "helloServlet", urlPatterns = "/hello")
public class HelloServlet extends HttpServlet {

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

```
@WebServlet // 서블릿 애노테이션
name: 서블릿 이름
urlPatterns: URL 매핑
<= 중복 불가능
```

-> HTTP 요청을 통해 매핑된 URL이 호출되면 서블릿 컨테이너는 해당 서블릿의 service() 메서드를 실행함

```
String username = request.getParameter("username"); // request 객체를 통해서 url 상의 쿼리 파라미터 추출, http 스펙(요청 메시지) 파싱 불필요

response.setContentType("text/plain");
response.setCharacterEncoding("utf-8"); // http 헤더에 들어감
response.getWriter().write("hello " + username); // http 메시지 바디에 들어감
```

HTTP 요청 메시지 로그로 확인하기

다음 설정을 추가하자.

application.properties

```
logging.level.org.apache.coyote.http11=debug
```

Request

3)HttpServletRequest - 개요

HttpServletRequest 객체의 역할 :

- HTTP 요청 메시지 파싱

HTTP 요청 메시지

```
POST /save HTTP/1.1
Host: localhost:8080
Content-Type: application/x-www-form-urlencoded

username=kim&age=20
```

리퀘스트 라인(스타트 라인) / 헤더 / 바디로 구성됨

서블릿은 개발자가 HTTP 요청 메시지를 편리하게 사용할 수 있도록 개발자 대신에 HTTP 요청 메시지를 파싱해 HttpServletRequest 객체에 담아서 제공함

- 임시 저장소 기능

해당 HTTP 요청이 시작부터 끝날 때 까지 유지되는 임시 저장소 기능
저장: request.setAttribute(name, value)
조회: request.getAttribute(name)

- 세션 관리 기능

request.getSession(create: true)

4)HttpServletRequest - 기본 사용법

HttpServletRequest가 제공하는 기본 기능들

-start-line 정보-

```
private void printStartLine(HttpServletRequest request) {

 System.out.println("--- REQUEST-LINE - start ---");
 System.out.println("request.getMethod() = " + request.getMethod()); //GET
 System.out.println("request.getProtocol() = " + request.getProtocol()); //
HTTP/1.1
 System.out.println("request.getScheme() = " + request.getScheme()); //http
 // http://localhost:8080/request-header
 System.out.println("request.getRequestURL() = " + request.getRequestURL());
 // /request-header
 System.out.println("request.getRequestURI() = " + request.getRequestURI());
//username=hi
 System.out.println("request.getQueryString() = " + request.getQueryString());
 System.out.println("request.isSecure() = " + request.isSecure()); //https 사용
유무
 System.out.println("--- REQUEST-LINE - end ---");
 System.out.println();
```

```
}
```

```
--- REQUEST-LINE - start ---
request.getMethod() = GET
request.getProtocal() = HTTP/1.1
request.getScheme() = http
request.getRequestURL() = http://localhost:8080/request-header
request.getRequestURI() = /request-header
request.getQueryString() = null
request.isSecure() = false
--- REQUEST-LINE - end ---
```

-헤더 정보-

```
//Header 모든 정보
private void printHeaders(HttpServletRequest request) {
 System.out.println("--- Headers - start ---");
/*
 Enumeration<String> headerNames = request.getHeaderNames();
 while (headerNames.hasMoreElements()) {
 String headerName = headerNames.nextElement();
 System.out.println(headerName + ": " + request.getHeader(headerName));
 }
*/
 request.getHeaderNames().asIterator()
 .forEachRemaining(headerName -> System.out.println(headerName + ":
" + request.getHeader(headerName)));
 System.out.println("--- Headers - end ---");
 System.out.println();
}
```

```
--- Headers - start ---
host: host
connection: connection
cache-control: cache-control
sec-ch-ua: sec-ch-ua
sec-ch-ua-mobile: sec-ch-ua-mobile
sec-ch-ua-platform: sec-ch-ua-platform
upgrade-insecure-requests: upgrade-insecure-requests
user-agent: user-agent
accept: accept
sec-fetch-site: sec-fetch-site
sec-fetch-mode: sec-fetch-mode
sec-fetch-user: sec-fetch-user
sec-fetch-dest: sec-fetch-dest
accept-encoding: accept-encoding
accept-language: accept-language
--- Headers - end ---
```

-Header 편리한 조회-

```
private void printHeaderUtils(HttpServletRequest request) {
        System.out.println("--- Header 편의 조회 start ---");
        System.out.println("[Host 편의 조회]");
        System.out.println("request.getServerName() = " +
request.getServerName()); //Host 헤더
        System.out.println("request.getServerPort() = " +
request.getServerPort()); //Host 헤더
        System.out.println();

        System.out.println("[Accept-Language 편의 조회]");
        request.getLocales().asIterator()
                .forEachRemaining(locale -> System.out.println("locale = " +
locale));
        System.out.println("request.getLocale() = " + request.getLocale());
        System.out.println();

        System.out.println("[cookie 편의 조회]");
        if (request.getCookies() != null) {
            for (Cookie cookie : request.getCookies()) {
                System.out.println(cookie.getName() + ": " + cookie.getValue());
            }
        }
        System.out.println();

        System.out.println("[Content 편의 조회]");
        System.out.println("request.getContentType() = " +
request.getContentType());
        System.out.println("request.getContentLength() = " +
request.getContentLength());
        System.out.println("request.getCharacterEncoding() = " +
request.getCharacterEncoding());

        System.out.println("--- Header 편의 조회 end ---");
        System.out.println();
    }
```

```
--- Header 편의 조회 start ---
[Host 편의 조회]
request.getServerName() = localhost
request.getServerPort() = 8080

[Accept-Language 편의 조회]
locale = ko_KR
locale = ko
locale = en_US
locale = en
request.getLocale() = ko_KR

[cookie 편의 조회]

[Content 편의 조회]
request.getContentType() = null
request.getContentLength() = -1
request.getCharacterEncoding() = UTF-8
--- Header 편의 조회 end ---
```

-기타정보-

```
    //기타 정보
    private void printEtc(HttpServletRequest request) {
        System.out.println("--- 기타 조회 start ---");

        System.out.println("[Remote 정보]");
        System.out.println("request.getRemoteHost() = " +
request.getRemoteHost());
        System.out.println("request.getRemoteAddr() = " +
request.getRemoteAddr());
        System.out.println("request.getRemotePort() = " +
request.getRemotePort());
        System.out.println();

        System.out.println("[Local 정보]");
        System.out.println("request.getLocalName() = " +
request.getLocalName());
        System.out.println("request.getLocalAddr() = " +
request.getLocalAddr());
        System.out.println("request.getLocalPort() = " +
request.getLocalPort());

        System.out.println("--- 기타 조회 end ---");
        System.out.println();
    }
```

```
--- 기타 조회 start ---
[Remote 정보]
request.getRemoteHost() = 0:0:0:0:0:0:0:1
request.getRemoteAddr() = 0:0:0:0:0:0:0:1
request.getRemotePort() = 60212

[Local 정보]
request.getLocalName() = 0:0:0:0:0:0:0:1
request.getLocalAddr() = 0:0:0:0:0:0:0:1
request.getLocalPort() = 8080
--- 기타 조회 end ---
```

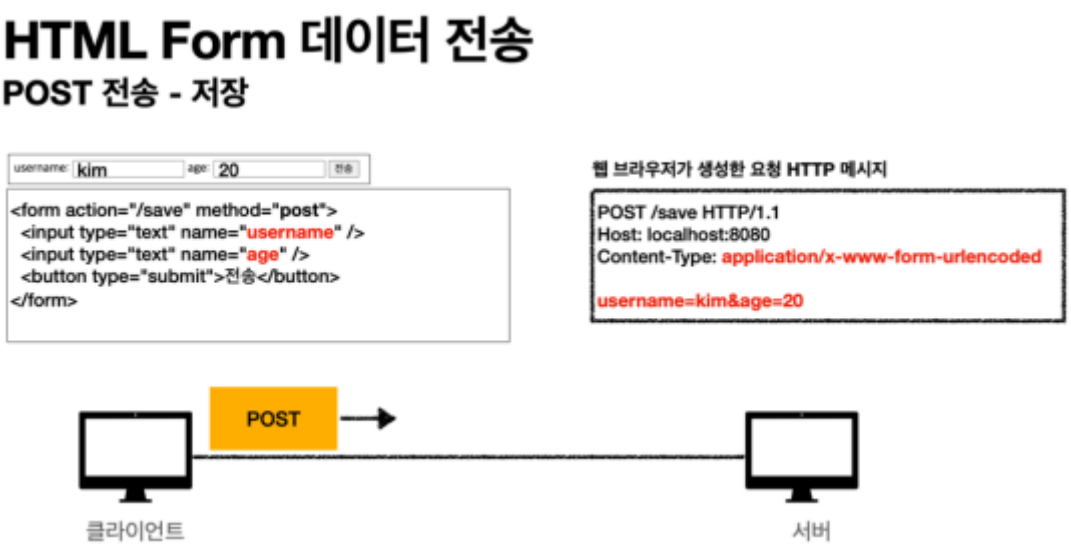
5)HTTP 요청 데이터 - 개요

: `HTTP 요청 메시지`를 통해 `클라이언트` -> `서버` 데이터를 전달하는 방법

1. GET 방식 - 쿼리 파라미터

```
- /url?username=hello&age=20
- 메시지 바디 없이, URL의 쿼리 파라미터에 데이터를 포함해서 전달
- 예) 검색, 필터, 페이징등에서 많이 사용하는 방식
```

2. POST 방식 - HTML Form




```
- content-type: application/x-www-form-urlencoded // form에 담아서 보냈다는 뜻
- 메시지 바디에 쿼리 파리미터 형식으로 전달 username=hello&age=20
- 예) 회원 가입, 상품 주문, HTML Form 사용
```

3. HTTP message body 방식

```
- HTTP API에서 주로 사용, JSON, XML, TEXT
- 데이터 형식은 주로 JSON 사용
- POST, PUT, PATCH
```

6)HTTP 요청 데이터 - GET 쿼리 파라미터

// http://localhost:8080/request-param?username=hello&username=hello2&age=20

전체 파라미터 조회 - request.getParameterNames();

```
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");
```

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

단일 파라미터 조회 - request.getParameter();

```
System.out.println("[단일 파라미터 조회]");
        String username = request.getParameter("username");
        String age = request.getParameter("age");
        System.out.println("username= "+username);
        System.out.println("age= "+age);
        System.out.println();
```

```
[단일 파라미터 조회]
username= hello
age= 20
```

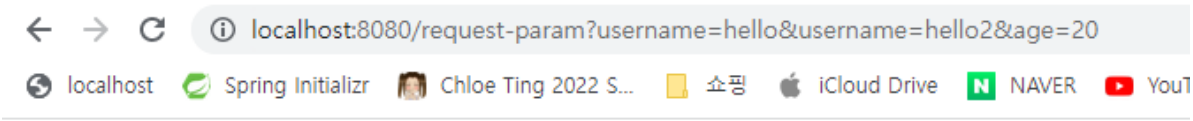
이름이 같은 복수 파라미터 조회 - request.getParameterValues();

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

// 이름이 같은 복수 파라미터를 request.getParamter()로 조회하는 경우, 제일 처음에 나온 파라미터 꺼내옴
```

```
[이름이 같은 복수 파라미터 조회]
username = hello
username = hello2
```

(response객체로 html 화면에 내용 출력하기)

```
response.getWriter().write( s: "ok");
```



ok

7)HTTP 요청 데이터 - POST HTML Form(Feat. Postman)

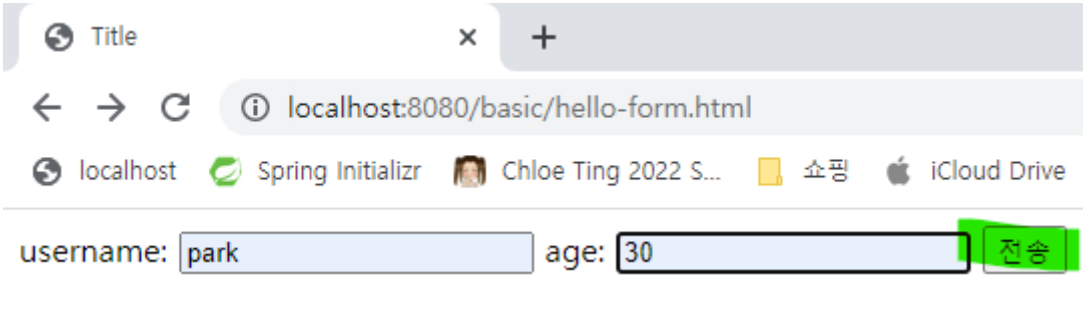
`HTML의 Form`을 사용해서 `클라이언트` -> `서버`로 데이터를 전달하는 방법

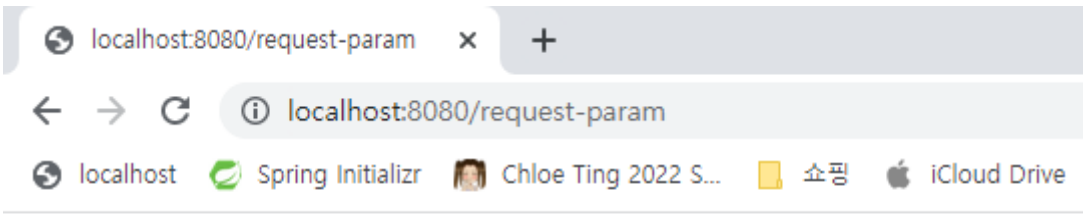
: form태그 action속성에다가 해당 서블릿 urlPatterns 매핑해주기

전송 버튼 클릭 -> http 요청 메시지 바디에 폼에 입력된 값 담아서 넘겨줌+url 해당 페이지로 변경됨

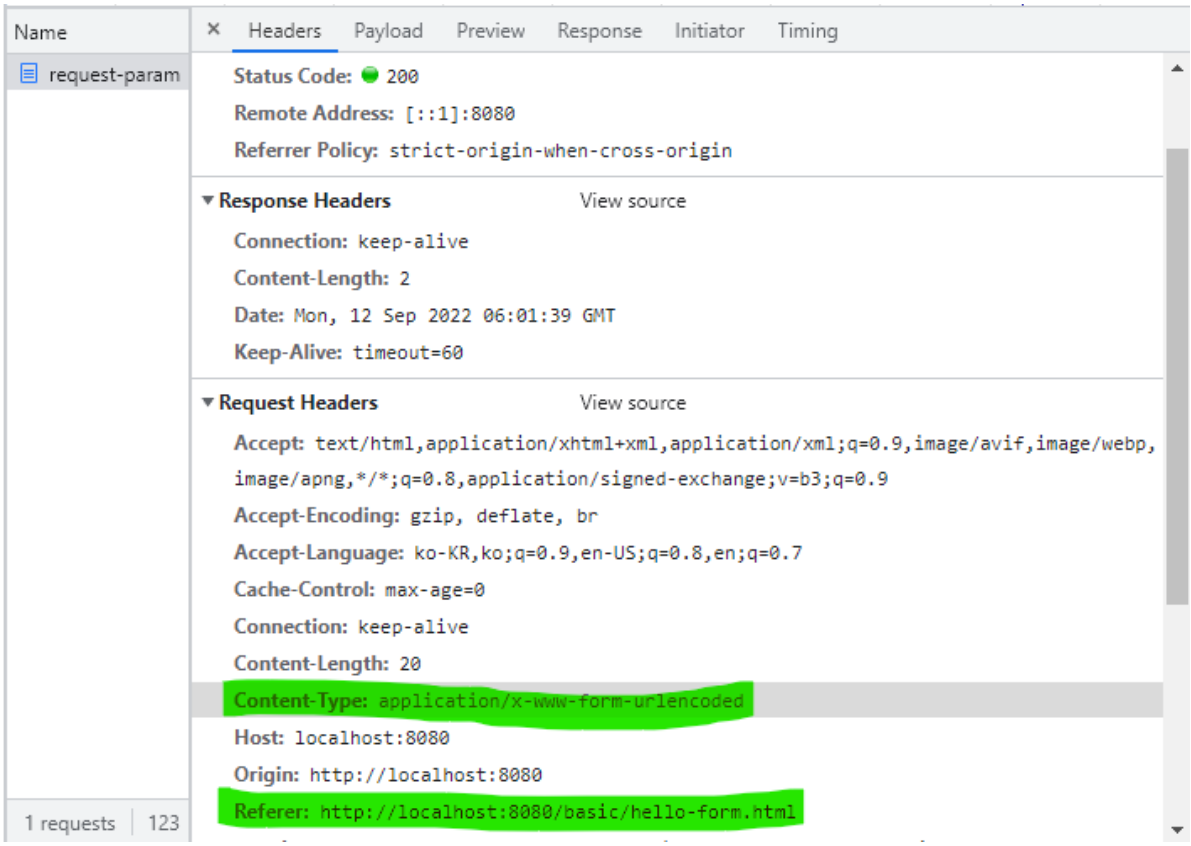
hello-form.html | RequestParamServlet.java

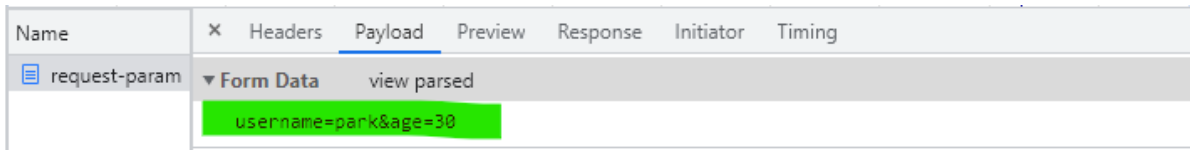
```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<form action="/request-param" method="post">
  username: <input type="text" name="username" />
  age:      <input type="text" name="age" />
  <button type="submit">전송</button>
</form>
</body>
</html>
<!--/basic/hello-form.html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





ok





=> 같은 서블릿을 get방식으로 호출/ 같은 서블릿을 post방식으로 호출 가능함

=> get방식이나 post방식이나 모두 `쿼리 파라미터 형식`으로 입력값을 전달하기때문에(다만, 입력값 위치가 get방식은 요청 메시지 헤더에, post 방식은 요청 메시지 메시지 바디에 담아서 넘겨줌) request.getParamter()매서드는 get/post 방식에서 모두 사용 가능함

Ex) request.getParamter()는 GET URL 쿼리 파라미터 형식도 지원하고, POST HTML Form 형식도 둘 다 지원함

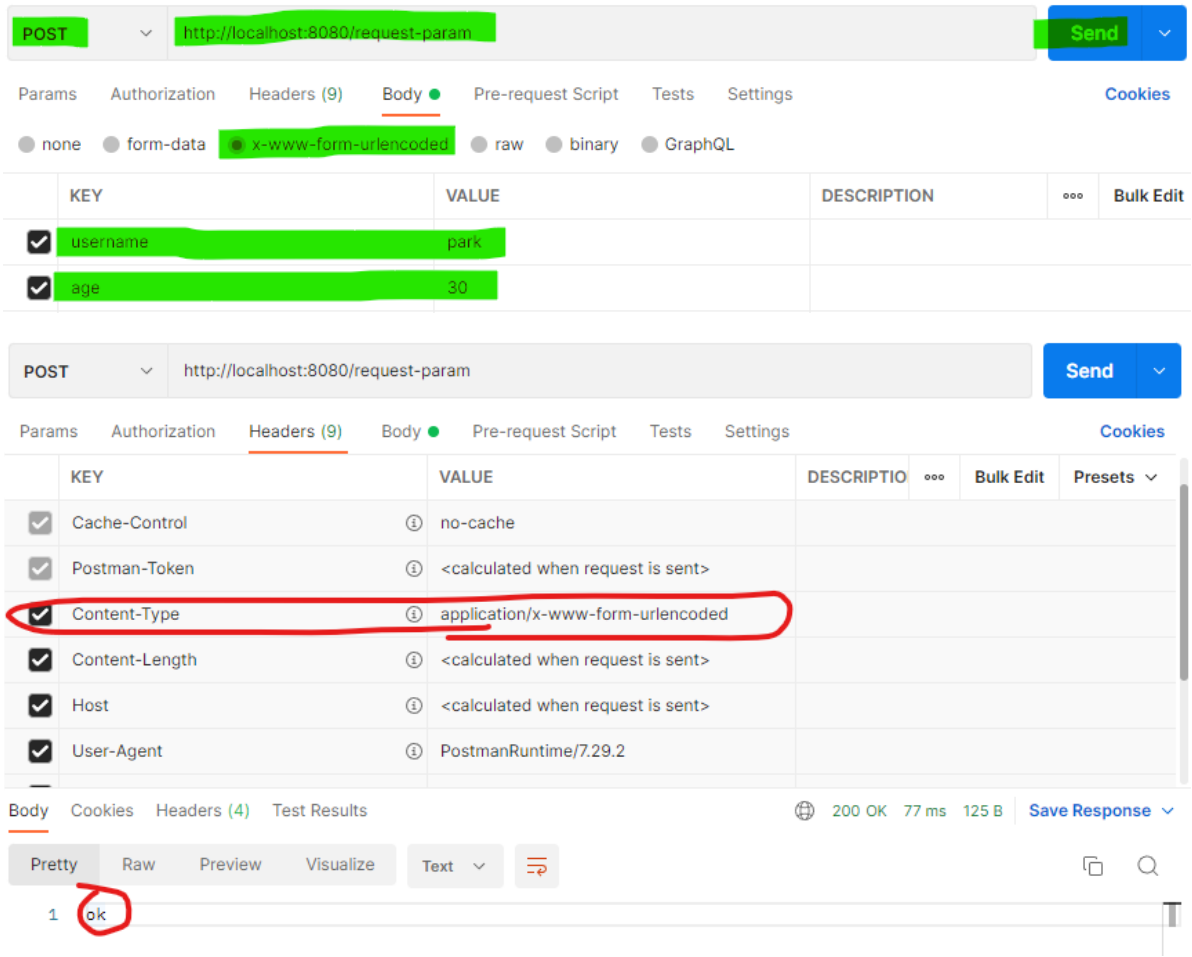
=>클라이언트 입장에선 get 방식, post 두 방식에 차이가 있지만(데이터를 어디에 담아서 보내느냐가 다름) 서버 입장에서는 둘의 형식이 동일하므로, request.getParamter()로 편리하게 구분 없이 조회 가능함

(참고)

content-type : HTTP 메시지 바디에 들어있는 데이터 형식 지정

GET방식인 경우 HTTP 메시지 바디가 비어있기때문에 지정X, POST 방식인 경우 지정 필수 폼으로 데이터를 전송하는 형식은 application/x-www-form-urlencoded

Postman을 사용한 테스트 : POST방식으로 데이터를 전송하고 싶을때마다 HTML form을 만들기 귀찮아서, Postman 사용시 form 안만들고도 post방식 데이터 전송 테스트 가능함



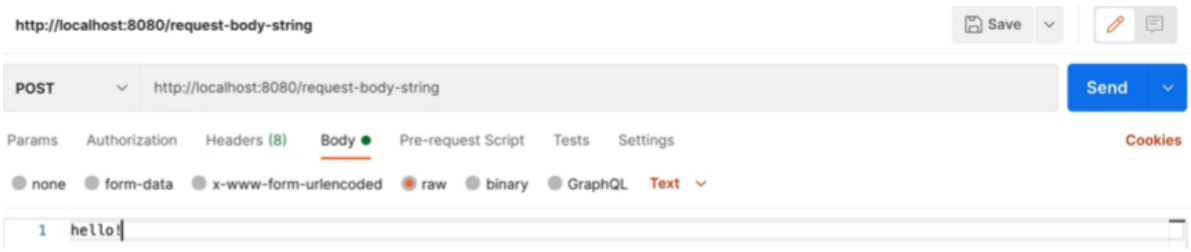
HTML Form에서 보낸 것과 Postman에서 POST방식 선택 -> Body에서 application/x-www-form-urlencoded 선택 후 key(폼 태그에서 name 속성)-value(인풋 태그에 입력되는 값) 넣어서 보낸 것과 동일한 결과를 가져옴 <= *HTML Form 안만들고 POST 방식 테스트 가능함*

8)HTTP 요청 데이터 - API 메시지 바디 - 단순 텍스트

HTTP message body에 데이터를 직접 담아서 요청

- HTTP API에서 주로 사용, JSON, XML, TEXT / http 메시지 바디에 원하는 데이터 직접 담아서 전달
- 데이터 형식은 주로 JSON 사용
- POST, PUT, PATCH

```java
@WebServlet(name = "requestBodyStringServlet", urlPatterns = "/request-body-string")
public class RequestBodyStringServlet extends HttpServlet {

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletExcepti
        ServletInputStream inputStream = request.getInputStream();
        String messageBody = StreamUtils.copyToString(inputStream, StandardCharsets.UTF_8);

        System.out.println("messageBody = " + messageBody);

        response.getWriter().write( s: "ok");
    }
}
```

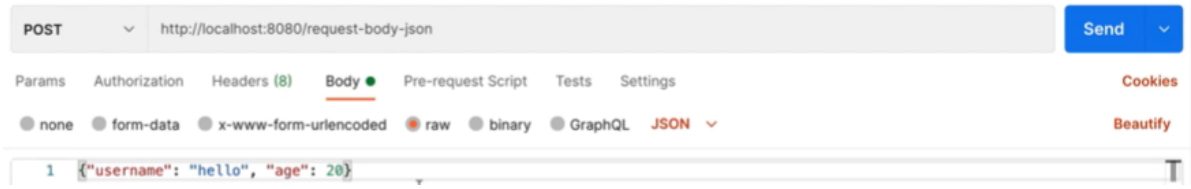



```
messageBody = hello!
```

> 참고
>
> inputStream은 byte 코드를 반환한다. byte 코드를 우리가 읽을 수 있는 문자(String)로 보려면 문자표 (Charset)를 지정해주어야 한다. 여기서는 UTF_8 Charset을 지정해주었다.

9)HTTP 요청 데이터 - API 메시지 바디 - JSON

```
@WebServlet(name = "requestBodyJsonServlet", urlPatterns = "/request-body-json")
public class RequestBodyJsonServlet extends HttpServlet {

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletE
        ServletInputStream inputStream = request.getInputStream();
        String messageBody = StreamUtils.copyToString(inputStream, StandardCharsets.UTF_8);

        System.out.println("messageBody = " + messageBody);
    }
}
```



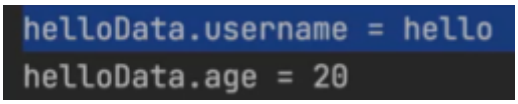
Headers의 Content-Type : application/json

```
messageBody = {"username": "hello", "age": 20}
```

JSON형식 -> 파싱 가능

> 참고
>
> JSON 결과(문자열 형식)를 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.

```java
public class RequestBodyJsonServlet extends HttpServlet {

    private ObjectMapper objectMapper = new ObjectMapper();

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletExcept
        ServletInputStream inputStream = request.getInputStream();
        String messageBody = StreamUtils.copyToString(inputStream, StandardCharsets.UTF_8);

        System.out.println("messageBody = " + messageBody);

        HelloData helloData = objectMapper.readValue(messageBody, HelloData.class);

        System.out.println("helloData.username = " + helloData.getUsername());
        System.out.println("helloData.age = " + helloData.getAge());

        response.getWriter().write( s: "ok");
    }
}
```



Response

10)HttpServletResponse - 기본 사용법

- **HTTP 응답 메시지 생성** : HTTP 응답코드 지정, 헤더 생성, 바디 생성

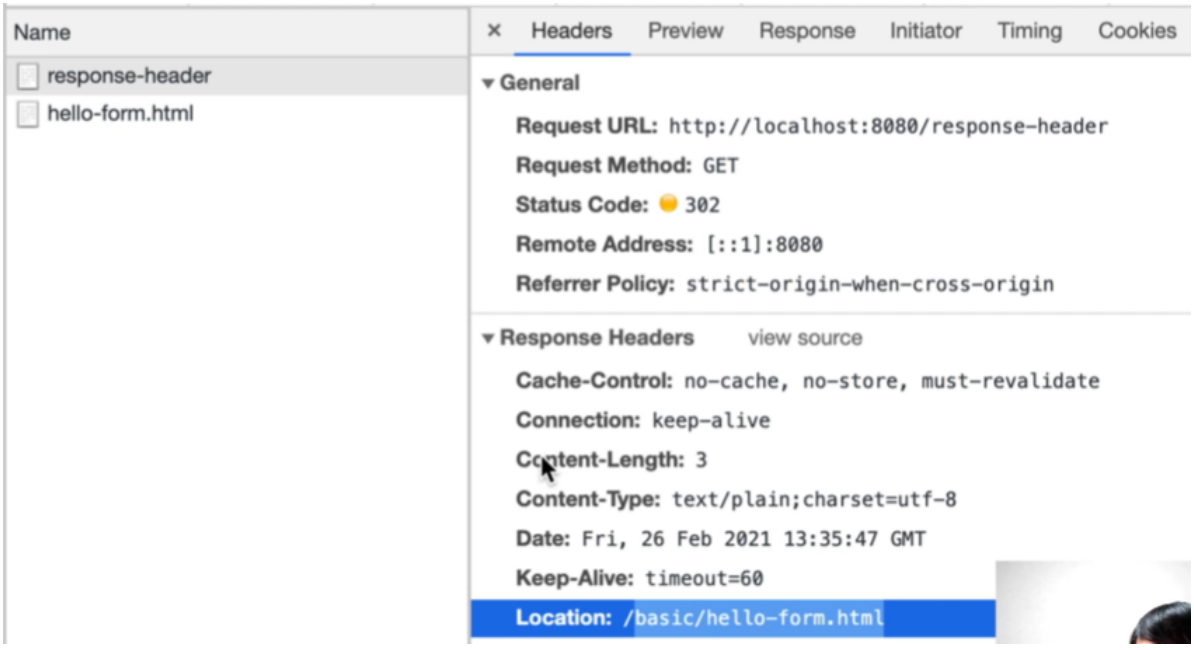
```java
@WebServlet(name = "responseHeaderServlet", urlPatterns = "/response-header")
public class ResponseHeaderServlet extends HttpServlet {

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletExceptio
        //[status-line]
        response.setStatus(HttpServletResponse.SC_OK);

        //[response-headers]
        response.setHeader( name: "Content-Type",  value: "text/plain");
        response.setHeader( name: "Cache-Control",  value: "no-cache, no-store, must-revalidate");
        response.setHeader( name: "Pragma",  value: "no-cache");
        response.setHeader( name: "my-header",  value: "hello");

        PrintWriter writer = response.getWriter();
        writer.println("ok");
    }
}
```

- **편의 기능 제공** : Content-Type, 쿠키, Redirect

```java
//[Header 편의 메서드]
//  content(response);
//  cookie(response);
redirect(response);
```

```java
private void content(HttpServletResponse response) {
    //Content-Type: text/plain;charset=utf-8
    //Content-Length: 2
      response.setHeader("Content-Type", "text/plain;charset=utf-8");
    response.setContentType("text/plain");
    response.setCharacterEncoding("utf-8");
      response.setContentLength(2); //(생략시 자동 생성)
}

private void cookie(HttpServletResponse response) {
    //Set-Cookie: myCookie=good; Max-Age=600;
    //response.setHeader("Set-Cookie", "myCookie=good; Max-Age=600");
    Cookie cookie = new Cookie( name: "myCookie",  value: "good");
    cookie.setMaxAge(600); //600초
    response.addCookie(cookie);
}
```

```java
private void redirect(HttpServletResponse response) throws IOException {
    //Status Code 302
    //Location: /basic/hello-form.html

//      response.setStatus(HttpServletResponse.SC_FOUND); //302
//      response.setHeader("Location", "/basic/hello-form.html");
    response.sendRedirect( location: "/basic/hello-form.html");
}
```



11)HTTP 응답 데이터 - 단순 텍스트, HTML

- **단순 텍스트 응답** Ex) 앞에서 살펴봄 (writer.println("ok");)

```java
@WebServlet(name = "responseHeaderServlet", urlPatterns = "/response-header")
public class ResponseHeaderServlet extends HttpServlet {

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

        //[response-headers]
        response.setHeader( name: "Content-Type",  value: "text/plain"); charset=utf-8;
        response.setHeader( name: "Cache-Control",  value: "no-cache, no-store, must-revalidate");
        response.setHeader( name: "Pragma",  value: "no-cache");
        response.setHeader( name: "my-header",  value: "hello");

        PrintWriter writer = response.getWriter();
        writer.println("ok");
    }
}
```

- **HTML 응답**

```java
@WebServlet(name = "responseHtmlServlet", urlPatterns = "/response-html")
public class ResponseHtmlServlet extends HttpServlet {
    @Override
    protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletExcept
        //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>");
    }
```

- **HTTP API** - MessageBody JSON 응답

12)HTTP 응답 데이터 - API JSON

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

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletExcep
        //Content-Type: application/json
        response.setContentType("application/json");
        response.setCharacterEncoding("utf-8");

        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** - MessageBody JSON 응답

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

13)정리

servlet

`HttpServletRequest` :

- HTTP 요청 메시지 파싱 <- GET, POST, messageBody(단순 텍스트, JSON)
- 임시 저장소 기능
- 세션 관리 기능

`HttpServletResponse` :

- HTTP 응답 메시지 생성 : HTTP 응답코드 지정, 헤더 생성, 바디 생성 <- 단순 텍스트 응답, HTML 응답, HTTP API 응답
- 편의 기능 제공 : Content-Type, 쿠키, Redirect