웹 스터디

2강 – 자바 웹서버 개론

Servlet과 Jdbc, Spring MVC

오늘 다뤄볼 주제

Java 웹 서버를 차근차근 쌓아 올려 봅시다.

```java
import ...

public class WebServer {
    1 usage
    private static final Logger log = LoggerFactory.getLogger(WebServer.class);
    1 usage
    private static final int DEFAULT_PORT = 8080;

    public static void main(String args[]) throws Exception {
        int port = 0;
        if (args == null || args.length == 0) {
            port = DEFAULT_PORT;
        } else {
            port = Integer.parseInt(args[0]);
        }

        // 서버소켓을 생성한다. 웹서버는 기본적으로 8080번 포트를 사용한다.

        try (ServerSocket listenSocket = new ServerSocket(port)) {
            log.info("Web Application Server started {} port.", port);

            // 클라이언트가 연결될때까지 대기한다.
            Socket connection;
            while ((connection = listenSocket.accept()) != null) {
                RequestHandler requestHandler = new RequestHandler(connection);
                requestHandler.start();
            }
        }
    }
}
```

Listen thread

서버로 오는 요청을 받아
connect thread를 생성

connect thread

요청마다 생성되어 처리

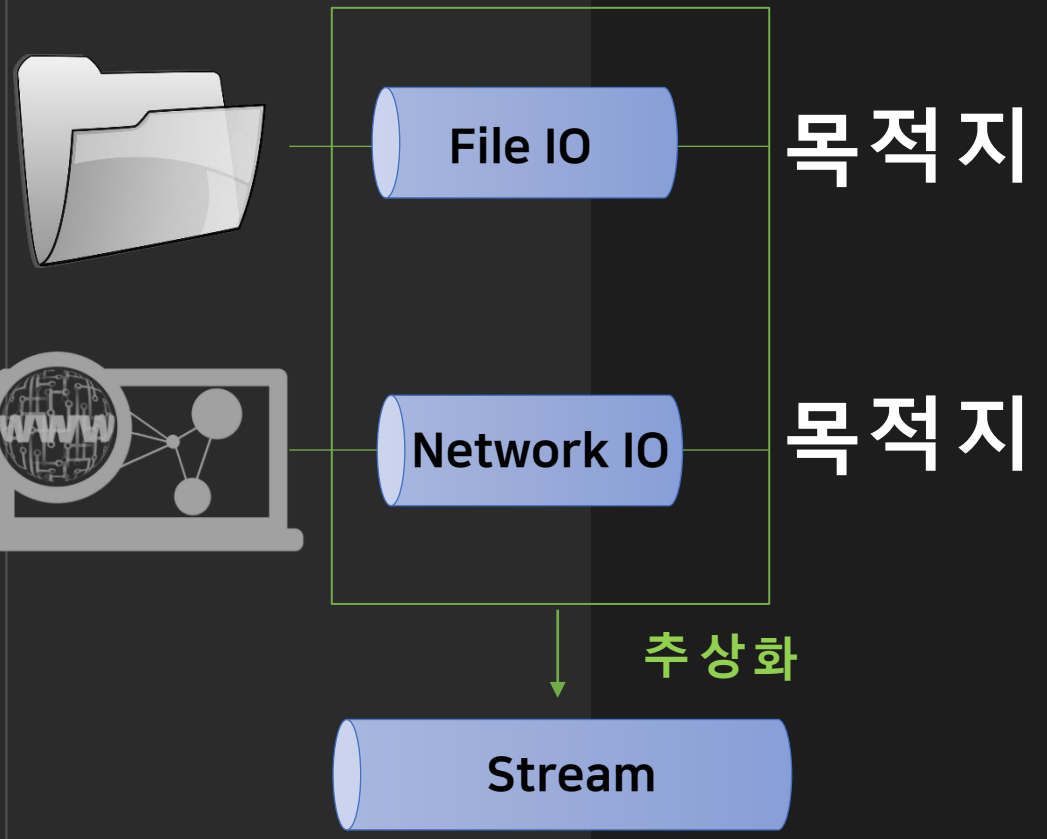
저번에 설명 드린 threadPool

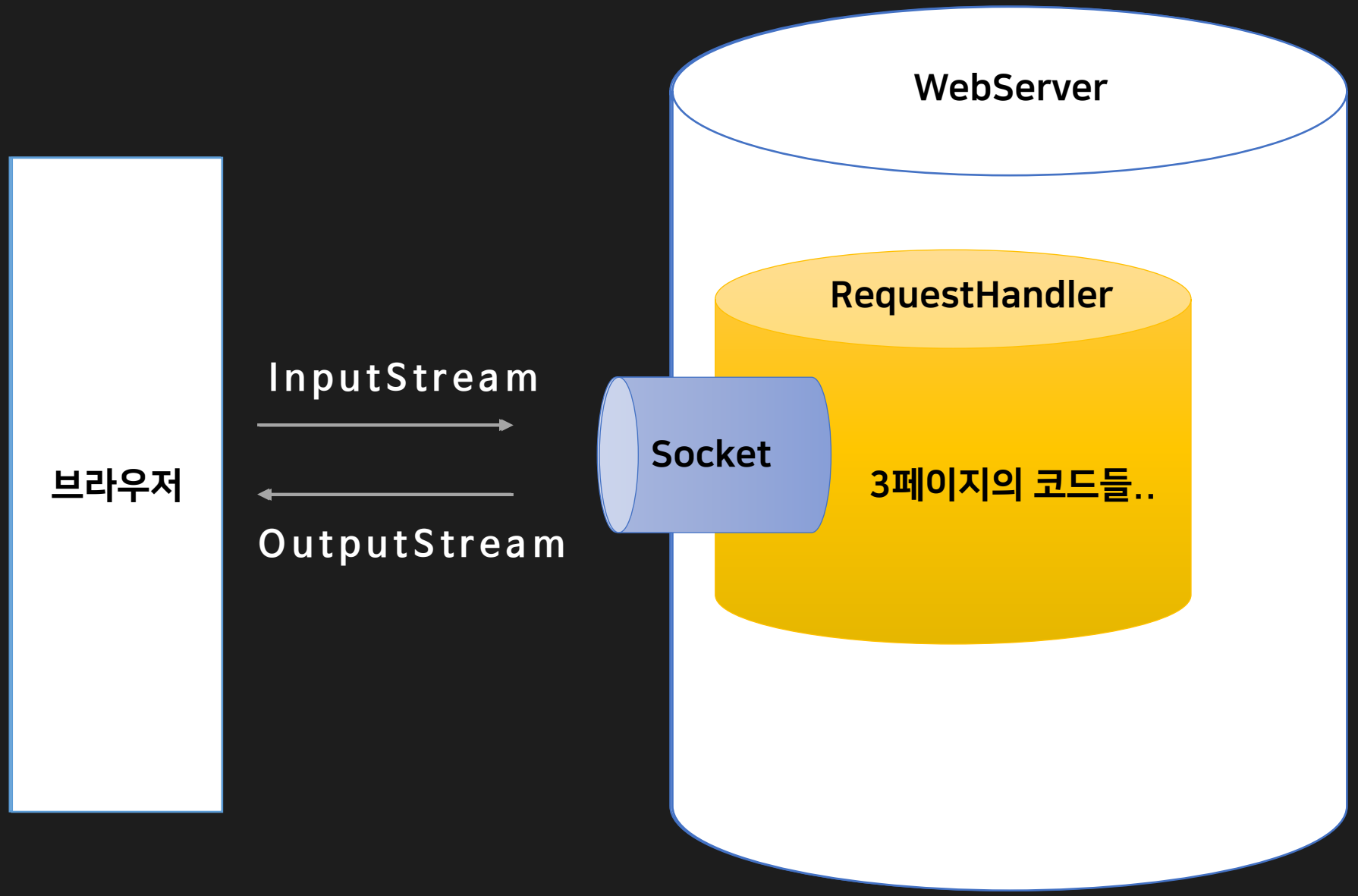
JAVA21부턴 virtual thread도 생김
(아직 실험적)

```java
public class RequestHandler extends Thread {
    4 usages
    private static final Logger log = LoggerFactory.getLogger(RequestHandler.class);

    5 usages
    private Socket connection;

    2 usages
    public RequestHandler(Socket connectionSocket) { this.connection = connectionSocket; }

    public void run() {
        log.debug("New Client Connect! Connected IP : {}, Port : {}", connection.getInetAddress(),
                connection.getPort());

        try (InputStream in = connection.getInputStream(); OutputStream out = connection.getOutputStream()) {
            // TODO 사용자 요청에 대한 처리는 이 곳에 구현하면 된다.
            DataOutputStream dos = new DataOutputStream(out);
            byte[] body = "Hello World".getBytes();
            response200Header(dos, body.length);
            responseBody(dos, body);
        } catch (IOException e) {
            log.error(e.getMessage());
        }
    }

    1 usage
    private void response200Header(DataOutputStream dos, int lengthOfBodyContent) {
        try {
            dos.writeBytes( s: "HTTP/1.1 200 OK \r\n");
            dos.writeBytes( s: "Content-Type: text/html;charset=utf-8\r\n");
            dos.writeBytes( s: "Content-Length: " + lengthOfBodyContent + "\r\n");
            dos.writeBytes( s: "\r\n");
        } catch (IOException e) {
            log.error(e.getMessage());
        }
    }
```

Stream이란?

데이터가 전송되는 통로




WebServer
RequestHandler
3페이지의 코드들..
Socket
InputStream
OutputStream
브라우저

서버를 켜고 localhost:8080
접속해봅시다.

Hello World

반환값으로 html 읽어서 outStream에 쓰면?

-> 웹페이지

```
    try (InputStream in = connection.getInputStream(); OutputStream out = connection.getOutputStream())
        // TODO 사용자 요청에 대한 처리는 이 곳에 구현하면 된다.
        DataOutputStream dos = new DataOutputStream(out);
        byte[] body = "Hello World".getBytes();
        response200Header(dos, body.length);
        responseBody(dos, body);
    } catch (IOException e) {
        log.error(e.getMessage());
    }
}

private void response200Header(DataOutputStream dos, int lengthOfBodyContent) {
    try {
        dos.writeBytes("HTTP/1.1 200 OK \r\n");
        dos.writeBytes("Content-Type: text/html;charset=utf-8\r\n");
        dos.writeBytes("Content-Length: " + lengthOfBodyContent + "\r\n");
        dos.writeBytes("\r\n");
    } catch (IOException e) {
        log.error(e.getMessage());
    }
}

private void responseBody(DataOutputStream dos, byte[] body) {
    try {
        dos.write(body, 0, body.length);
        dos.flush();
    } catch (IOException e) {
        log.error(e.getMessage());
    }
}
```

스켈레톤 코드는
Output만 내보내고 있습니다.
Input도 받아봅시다.

어떤 형식으로 들어올까요?

```java
InputStreamReader inputStreamReader = new InputStreamReader(in);
BufferedReader bufferedReader = new BufferedReader(inputStreamReader);
Map<String, List<String>> headers = new HashMap<>();

while(true){
    String rawHeader = bufferedReader.readLine();
    if(rawHeader.isEmpty())
        break;

    List<String> values = new LinkedList<>();
    String[] parsedHeaders = rawHeader.split( regex: " ");
    String key = parsedHeaders[0];
    if (key.endsWith(":")){
        key = key.substring(0, key.length() - 1);
    }
    values.addAll(Arrays.asList(parsedHeaders).subList(1, parsedHeaders.length));
    headers.put(key, values);
}
System.out.println(headers);
DataOutputStream dos = new DataOutputStream(out);
log.info(in.toString());
byte[] body = "Hello World".getBytes();
response200Header(dos, body.length);
responseBody(dos, body);
```

```
18:21:29.778 [DEBUG] [Thread-1] [webserver.RequestHandler] - New Client Connect! Connected IP : /0:0:0:0:0:0:0:1, Port : 13525
GET /index.html HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Cache-Control: max-age=0
sec-ch-ua: "Not_A Brand";v="8", "Chromium";v="120", "Google Chrome";v="120"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate, br
Accept-Language: ko,en;q=0.9,ko-KR;q=0.8,en-US;q=0.7
Cookie: csrftoken=RCEFEEe5KZsIAFG73Jpf2XI6Oeh0J6q8
18:21:29.779 [INFO ] [Thread-0] [webserver.RequestHandler] - java.net.Socket$SocketInputStream@64d0d8d3
```

Http는 프로토콜, 규약일 뿐입니다.

이 문자열을 이용하기 위해선 소켓 열고, 파싱하고 객체에
매핑하고 소켓 닫고... 아이고.. 귀찮아.. 어차피 정해진 형식인데..

어차피 정해진 양식대로 데이터 주고 받는 일을 일일이 구현 해야 하나요?

- 서버 TCP/IP 연결 대기, 소켓 연결
- HTTP 요청 메시지를 파싱해서 읽기
- POST 방식, /save URL 인지
- Content-Type 확인
- HTTP 메시지 바디 내용 피싱
- 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 종

이것만 하고 싶다!

서블릿

https://mangkyu.tistory.com/14

서블릿만으로 충분한가요?

2. 서블릿

#2.인강/4. 스프링 MVC 1/강의#

목차

프로젝트 생성

사전 준비물

- Java 11 설치
- IDE: IntelliJ 또는 Eclipse 설치

스프링 부트 스타터 사이트로 이동해서 스프링 프로젝트 생성

https://start.spring.io

- 프로젝트 선택
  - Project: **Gradle - Groovy** Project
  - Language: Java
  - Spring Boot: 2.4.x
- Project Metadata
  - Group: hello
  - Artifact: servlet
  - Name: servlet
  - Package name: hello.servlet

- Packaging: **War (주의!)**
- Java: 11

- Dependencies: **Spring Web**, **Lombok**

> **주의!**
> Packaging는 Jar가 아니라 War를 선택해주세요. JSP를 실행하기 위해서 필요합니다.

주의! - 스프링 부트 3.0

스프링 부트 3.0을 선택하게 되면 다음 부분을 꼭 확인해주세요.

- **1. Java 17 이상**을 사용해야 합니다.
- **2. javax 패키지 이름을 jakarta로 변경**해야 합니다.
  - 오라클과 자바 라이센스 문제로 모든 javax 패키지를 jakarta로 변경하기로 했습니다.

패키지 이름 변경 예)

- **JPA 애노테이션**
  - javax.persistence.Entity ➡ jakarta.persistence.Entity
- **스프링에서 자주 사용하는 @PostConstruct 애노테이션**
  - javax.annotation.PostConstruct ➡ jakarta.annotation.PostConstruct
- **스프링에서 자주 사용하는 검증 애노테이션**
  - javax.validation ➡ jakarta.validation

스프링 부트 3.0 관련 자세한 내용은 다음 링크를 확인해주세요: https://bit.ly/springboot3

build.gradle

```
plugins {
  id 'org.springframework.boot' version '2.4.3'
  id 'io.spring.dependency-management' version '1.0.11.RELEASE'
  id 'java'
  id 'war'
}

group = 'hello'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '11'

configurations {
```

```
    compileOnly {
        extendsFrom annotationProcessor
    }
}

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-web'
    compileOnly 'org.projectlombok:lombok'
    annotationProcessor 'org.projectlombok:lombok'
    providedRuntime 'org.springframework.boot:spring-boot-starter-tomcat'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}

test {
    useJUnitPlatform()
}
```

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

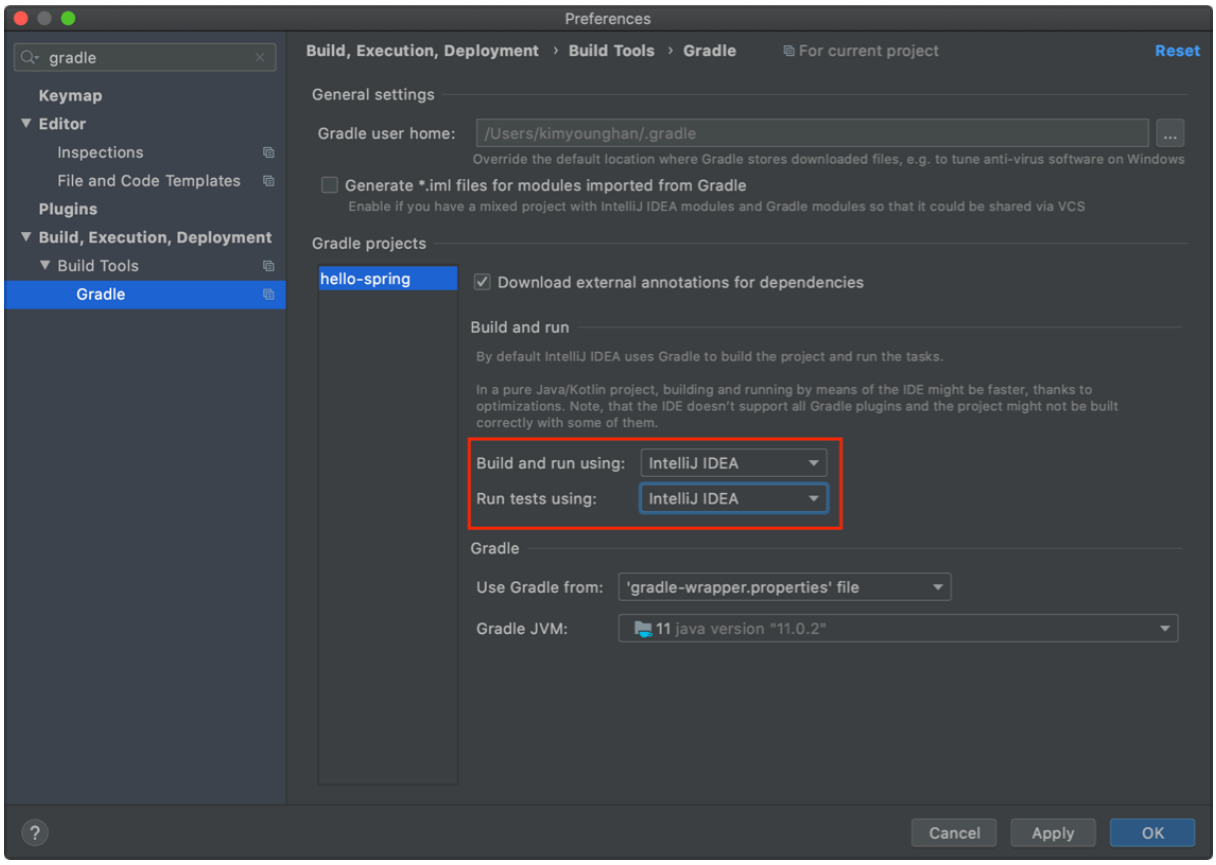
IntelliJ Gradle 대신에 자바 직접 실행

최근 IntelliJ 버전은 Gradle을 통해서 실행 하는 것이 기본 설정이다. 이렇게 하면 실행속도가 느리다. 다음과 같이 변경하면 자바로 바로 실행해서 실행속도가 더 빠르다

- Preferences ➜ Build, Execution, Deployment ➜ Build Tools ➜ Gradle
  - Build and run using: Gradle ➜ IntelliJ IDEA
  - Run tests using: Gradle ➜ IntelliJ IDEA

> **윈도우 사용자**
> File ➜ Setting

설정 이미지



> **주의!**
>
> **IntelliJ 무료 버전의 경우 해당 설정을 IntelliJ IDEA가 아니라 Gradle로 설정해야 한다.**
>
> Jar 파일의 경우는 문제가 없는데, War의 경우 톰캣이 정상 시작되지 않는 문제가 발생한다.
>
> 유료 버전은 모두 정상 동작한다.
>
> 또는 `build.gradle`에 있는 다음 코드를 제거해도 된다.
>
> `providedRuntime 'org.springframework.boot:spring-boot-starter-tomcat'`

롬복 적용

1. Preferences ➜ plugin ➜ lombok 검색 실행 (재시작)
2. Preferences ➜ Annotation Processors 검색 ➜ Enable annotation processing 체크 (재시작)
3. 임의의 테스트 클래스를 만들고 @Getter, @Setter 확인

> **윈도우 사용자**
>
> File ➜ Setting

Postman을 설치하자

다음 사이트에서 Postman을 다운로드 받고 설치해두자

- https://www.postman.com/downloads

Hello 서블릿

스프링 부트 환경에서 서블릿 등록하고 사용해보자.

> **참고**
> 서블릿은 톰캣 같은 웹 애플리케이션 서버를 직접 설치하고,그 위에 서블릿 코드를 클래스 파일로 빌드해서 올린 다음, 톰캣 서버를 실행하면 된다. 하지만 이 과정은 매우 번거롭다.
> 스프링 부트는 톰캣 서버를 내장하고 있으므로, 톰캣 서버 설치 없이 편리하게 서블릿 코드를 실행할 수 있다.

스프링 부트 서블릿 환경 구성

`@ServletComponentScan`
스프링 부트는 서블릿을 직접 등록해서 사용할 수 있도록 `@ServletComponentScan`을 지원한다. 다음과 같이 추가하자.

hello.servlet.ServletApplication

```
package hello.servlet;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.web.servlet.ServletComponentScan;

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

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

```
    }

}
```

서블릿 등록하기

처음으로 실제 동작하는 서블릿 코드를 등록해보자.

hello.servlet.basic.HelloServlet

```
package hello.servlet.basic;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@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");
        System.out.println("request = " + request);
        System.out.println("response = " + response);

        String username = request.getParameter("username");
        System.out.println("username = " + username);

        response.setContentType("text/plain");
        response.setCharacterEncoding("utf-8");
        response.getWriter().write("hello " + username);
    }
```

```
}
```

- `@WebServlet` 서블릿 애노테이션
  - name: 서블릿 이름
  - urlPatterns: URL 매핑

HTTP 요청을 통해 매핑된 URL이 호출되면 서블릿 컨테이너는 다음 메서드를 실행한다.

`protected void service(HttpServletRequest request, HttpServletResponse response)`

- 웹 브라우저 실행
  - http://localhost:8080/hello?username=world
  - 결과: hello world
- 콘솔 실행결과

```
HelloServlet.service
request = org.apache.catalina.connector.RequestFacade@5e4e72
response = org.apache.catalina.connector.ResponseFacade@37d112b6
username = world
```

> **주의**
> IntelliJ 무료 버전을 사용하는데, 서버가 정상 실행되지 않는다면 **프로젝트 생성 → IntelliJ Gradle 대신에 자바 직접 실행**에 있는 주의 사항을 읽어보자.

HTTP 요청 메시지 로그로 확인하기

다음 설정을 추가하자.

`application.properties`

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

서버를 다시 시작하고, 요청해보면 서버가 받은 HTTP 요청 메시지를 출력하는 것을 확인할 수 있다.

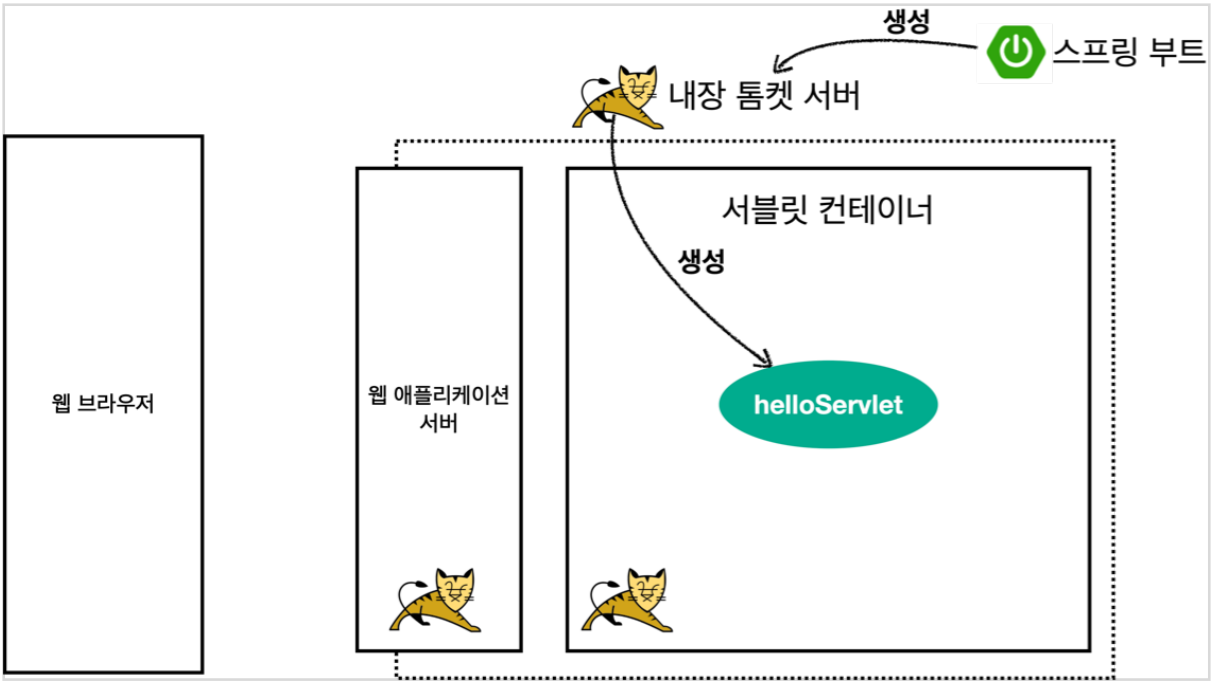
```
...o.a.coyote.http11.Http11InputBuffer: Received [GET /hello?username=servlet
```

```
HTTP/1.1
Host: localhost:8080
Connection: keep-alive
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="88", "Google Chrome";v="88", ";Not A Brand";v="99"
sec-ch-ua-mobile: ?0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 11_2_1) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/88.0.4324.150 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/
webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:8080/basic.html
Accept-Encoding: gzip, deflate, br
Accept-Language: ko,en-US;q=0.9,en;q=0.8,ko-KR;q=0.7

]
```

> **참고**
> 운영서버에 이렇게 모든 요청 정보를 다 남기면 성능저하가 발생할 수 있다. 개발 단계에서만 적용하자.

서블릿 컨테이너 동작 방식 설명

내장 톰캣 서버 생성



HTTP 요청, HTTP 응답 메시지

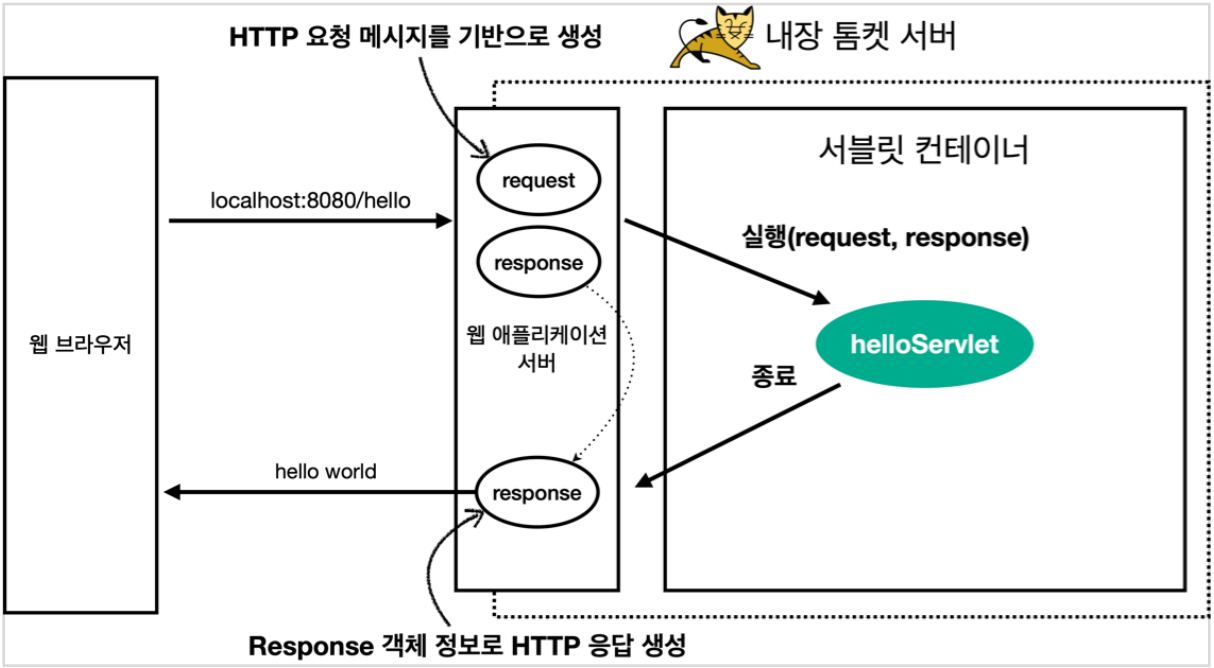
[HTTP 요청]

```
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



> **참고**
> HTTP 응답에서 Content-Length는 웹 애플리케이션 서버가 자동으로 생성해준다.

welcome 페이지 추가

지금부터 개발할 내용을 편리하게 참고할 수 있도록 welcome 페이지를 만들어두자.

`webapp` 경로에 `index.html`을 두면 http://localhost:8080 호출시 `index.html` 페이지가 열린다.

`main/webapp/index.html`

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<ul>
    <li><a href="basic.html">서블릿 basic</a></li>
</ul>
</body>
</html>
```

이번 장에서 학습할 내용은 다음 `basic.html`이다.

`main/webapp/basic.html`

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<ul>
    <li>hello 서블릿
```

```
        <ul>
            <li><a href="/hello?username=servlet">hello 서블릿 호출</a></li>
        </ul>
    </li>
    <li>HttpServletRequest
        <ul>
            <li><a href="/request-header">기본 사용법, Header 조회</a></li>
            <li>HTTP 요청 메시지 바디 조회
                <ul>
                    <li><a href="/request-param?username=hello&age=20">GET -
쿼리 파라미터</a></li>
                    <li><a href="/basic/hello-form.html">POST - HTML Form</a></
li>
                    <li>HTTP API - MessageBody -> Postman 테스트</li>
                </ul>
            </li>
        </ul>
    </li>
    <li>HttpServletResponse
        <ul>
            <li><a href="/response-header">기본 사용법, Header 조회</a></li>
            <li>HTTP 응답 메시지 바디 조회
                <ul>
                    <li><a href="/response-html">HTML 응답</a></li>
                    <li><a href="/response-json">HTTP API JSON 응답</a></li>
                </ul>
            </li>
        </ul>
    </li>
</ul>
</body>
</html>
```

HttpServletRequest - 개요

HttpServletRequest 역할

HTTP 요청 메시지를 개발자가 직접 파싱해서 사용해도 되지만, 매우 불편할 것이다. 서블릿은 개발자가 HTTP 요청 메시지를 편리하게 사용할 수 있도록 개발자 대신에 HTTP 요청 메시지를 파싱한다. 그리고 그 결과를 `HttpServletRequest` 객체에 담아서 제공한다.

HttpServletRequest를 사용하면 다음과 같은 HTTP 요청 메시지를 편리하게 조회할 수 있다.

HTTP 요청 메시지

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

username=kim&age=20
```

- START LINE
  - HTTP 메소드
  - URL
  - 쿼리 스트링
  - 스키마, 프로토콜
- 헤더
  - 헤더 조회
- 바디
  - form 파라미터 형식 조회
  - message body 데이터 직접 조회

HttpServletRequest 객체는 추가로 여러가지 부가기능도 함께 제공한다.

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)`

> **중요**

> HttpServletRequest, HttpServletResponse를 사용할 때 가장 중요한 점은 이 객체들이 HTTP 요청 메시지, HTTP 응답 메시지를 편리하게 사용하도록 도와주는 객체라는 점이다. 따라서 이 기능에 대해서 깊이있는 이해를 하려면 **HTTP 스펙이 제공하는 요청, 응답 메시지 자체를 이해**해야 한다.

HttpServletRequest - 기본 사용법

HttpServletRequest가 제공하는 기본 기능들을 알아보자.

hello.servlet.basic.request.RequestHeaderServlet

```
package hello.servlet.basic.request;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
import java.io.IOException;

//http://localhost:8080/request-header?username=hello
@WebServlet(name = "requestHeaderServlet", urlPatterns = "/request-header")
public class RequestHeaderServlet extends HttpServlet {

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

        printStartLine(request);
        printHeaders(request);
        printHeaderUtils(request);
        printEtc(request);

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

start-line 정보

```
//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();
}
```

결과

```
--- REQUEST-LINE - start ---
request.getMethod() = GET
request.getProtocol() = HTTP/1.1
request.getScheme() = http
request.getRequestURL() = http://localhost:8080/request-header
request.getRequestURI() = /request-header
request.getQueryString() = username=hello
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();
}
```

결과

```
--- Headers - start ---
host: localhost:8080
connection: keep-alive
cache-control: max-age=0
sec-ch-ua: "Chromium";v="88", "Google Chrome";v="88", ";Not A Brand";v="99"
sec-ch-ua-mobile: ?0
upgrade-insecure-requests: 1
user-agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 11_2_0) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/88.0.4324.150 Safari/537.36
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/
webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
sec-fetch-site: none
sec-fetch-mode: navigate
```

```
sec-fetch-user: ?1
sec-fetch-dest: document
accept-encoding: gzip, deflate, br
accept-language: ko,en-US;q=0.9,en;q=0.8,ko-KR;q=0.7
--- Headers - end ---
```

Header 편리한 조회

```
//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();
}
```

결과

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

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

[cookie 편의 조회]

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

기타 정보

기타 정보는 HTTP 메시지의 정보는 아니다.

```
//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();
}
```

결과

```
--- 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() = 54305

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

> **참고**
> 로컬에서 테스트하면 IPv6 정보가 나오는데, IPv4 정보를 보고 싶으면 다음 옵션을 VM options에 넣어주면 된다.
> `-Djava.net.preferIPv4Stack=true`

지금까지 HttpServletRequest를 통해서 HTTP 메시지의 start-line, header 정보 조회 방법을 이해했다. 이제 본격적으로 HTTP 요청 데이터를 어떻게 조회하는지 알아보자.

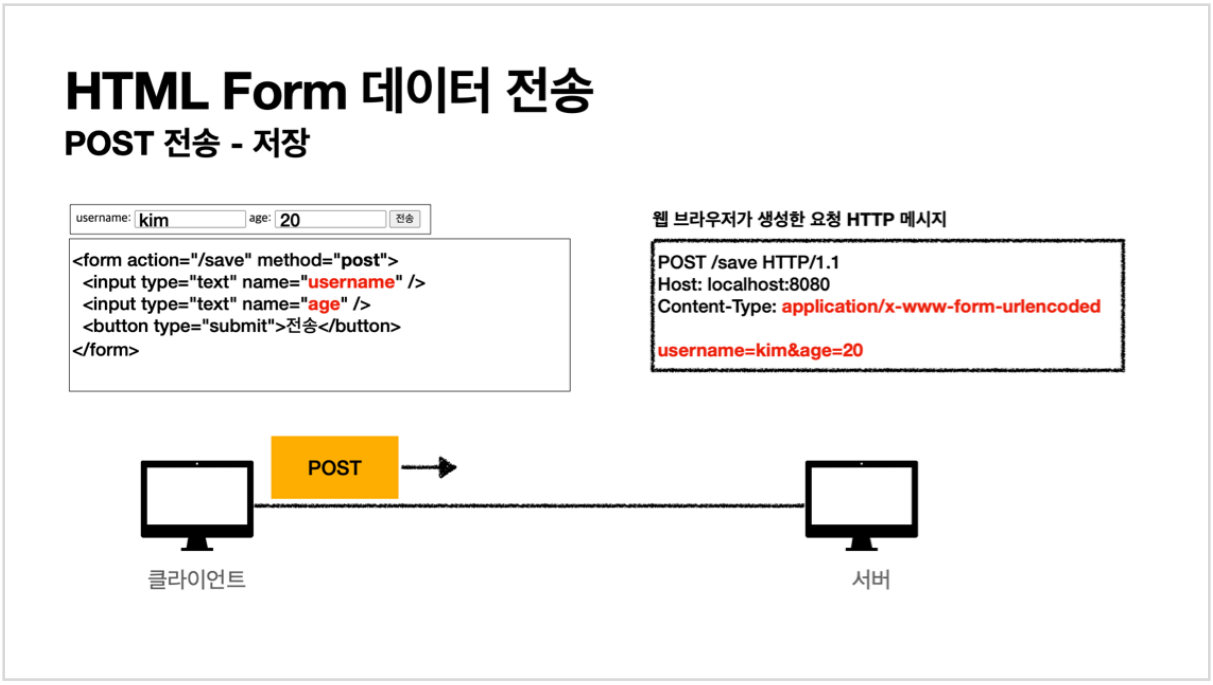
HTTP 요청 데이터 - 개요

HTTP 요청 메시지를 통해 클라이언트에서 서버로 데이터를 전달하는 방법을 알아보자.

주로 다음 3가지 방법을 사용한다.

- **GET - 쿼리 파라미터**
  - /url**?username=hello&age=20**
  - 메시지 바디 없이, URL의 쿼리 파라미터에 데이터를 포함해서 전달
  - 예) 검색, 필터, 페이징등에서 많이 사용하는 방식
- **POST - HTML Form**
  - content-type: application/x-www-form-urlencoded
  - 메시지 바디에 쿼리 파리미터 형식으로 전달 username=hello&age=20
  - 예) 회원 가입, 상품 주문, HTML Form 사용
- **HTTP message body**에 데이터를 직접 담아서 요청
  - HTTP API에서 주로 사용, JSON, XML, TEXT
- 데이터 형식은 주로 JSON 사용
  - POST, PUT, PATCH

POST- HTML Form 예시



하나씩 알아보자.

HTTP 요청 데이터 - GET 쿼리 파라미터

다음 데이터를 클라이언트에서 서버로 전송해보자.

전달 데이터

- username=hello
- age=20

메시지 바디 없이, URL의 **쿼리 파라미터**를 사용해서 데이터를 전달하자.
예) 검색, 필터, 페이징등에서 많이 사용하는 방식

쿼리 파라미터는 URL에 다음과 같이 `?`를 시작으로 보낼 수 있다. 추가 파라미터는 `&`로 구분하면 된다.

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

서버에서는 `HttpServletRequest`가 제공하는 다음 메서드를 통해 쿼리 파라미터를 편리하게 조회할 수 있다.

쿼리 파라미터 조회 메서드

```
String username = request.getParameter("username"); //단일 파라미터 조회
Enumeration<String> parameterNames = request.getParameterNames(); //파라미터 이름들 모두 조회
Map<String, String[]> parameterMap = request.getParameterMap(); //파라미터를 Map 으로 조회
String[] usernames = request.getParameterValues("username"); //복수 파라미터 조회
```

RequestParamServlet

```
package hello.servlet.basic.request;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.Enumeration;

/**
 * 1. 파라미터 전송 기능
 * http://localhost:8080/request-param?username=hello&age=20
 * <p>
 * 2. 동일한 파라미터 전송 가능
 * http://localhost:8080/request-param?username=hello&username=kim&age=20
 */
@WebServlet(name = "requestParamServlet", urlPatterns = "/request-param")
public class RequestParamServlet extends HttpServlet {

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

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

        /*
        Enumeration<String> parameterNames = request.getParameterNames();
        while (parameterNames.hasMoreElements()) {
```

```
            String paramName = parameterNames.nextElement();
            System.out.println(paramName + "=" +
request.getParameter(paramName));
        }
        */

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

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

        String age = request.getParameter("age");
        System.out.println("request.getParameter(age) = " + age);
        System.out.println();

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

        resp.getWriter().write("ok");
    }
}
```

실행 - 파라미터 전송

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

결과

```
[전체 파라미터 조회] - start
```

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

[단일 파라미터 조회]
request.getParameter(username) = hello
request.getParameter(age) = 20

[이름이 같은 복수 파라미터 조회]
request.getParameterValues(username)
username=hello
```

실행 - 동일 파라미터 전송

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

결과

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

[단일 파라미터 조회]
request.getParameter(username) = hello
request.getParameter(age) = 20

[이름이 같은 복수 파라미터 조회]
request.getParameterValues(username)
username=hello
username=kim
```

복수 파라미터에서 단일 파라미터 조회

`username=hello&username=kim`과 같이 파라미터 이름은 하나인데, 값이 중복이면 어떻게 될까?
`request.getParameter()`는 하나의 파라미터 이름에 대해서 단 하나의 값만 있을 때 사용해야 한다.
지금처럼 중복일 때는 `request.getParameterValues()`를 사용해야 한다.
참고로 이렇게 중복일 때 `request.getParameter()`를 사용하면 `request.getParameterValues()`의 첫 번째 값을 반환한다.

HTTP 요청 데이터 - POST HTML Form

이번에는 HTML의 Form을 사용해서 클라이언트에서 서버로 데이터를 전송해보자.
주로 회원 가입, 상품 주문 등에서 사용하는 방식이다.

특징

- content-type: `application/x-www-form-urlencoded`
- 메시지 바디에 쿼리 파리미터 형식으로 데이터를 전달한다. `username=hello&age=20`

`src/main/webapp/basic/hello-form.html` 생성

```
<!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>
```

실행해보자.

- http://localhost:8080/basic/hello-form.html

> **주의**
> 웹 브라우저가 결과를 캐시하고 있어서, 과거에 작성했던 html 결과가 보이는 경우도 있다. 이때는 웹 브라우저의 새로 고침을 직접 선택해주면 된다. 물론 서버를 재시작 하지 않아서 그럴 수도 있다.

POST의 HTML Form을 전송하면 웹 브라우저는 다음 형식으로 HTTP 메시지를 만든다. (웹 브라우저 개발자 모드 확인)

- **요청 URL**: http://localhost:8080/request-param
- **content-type**: `application/x-www-form-urlencoded`
- **message body**: `username=hello&age=20`

`application/x-www-form-urlencoded` 형식은 앞서 GET에서 살펴본 쿼리 파라미터 형식과 같다.
따라서 **쿼리 파라미터 조회 메서드를 그대로 사용**하면 된다.
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.

> **참고**
> content-type은 HTTP 메시지 바디의 데이터 형식을 지정한다.
> **GET URL 쿼리 파라미터 형식**으로 클라이언트에서 서버로 데이터를 전달할 때는 HTTP 메시지 바디를 사용하지 않기 때문에 content-type이 없다.
> **POST HTML Form 형식**으로 데이터를 전달하면 HTTP 메시지 바디에 해당 데이터를 포함해서 보내기 때문에 바디에 포함된 데이터가 어떤 형식인지 content-type을 꼭 지정해야 한다. 이렇게 폼으로 데이터를 전송하는 형식을 `application/x-www-form-urlencoded`라 한다.

Postman을 사용한 테스트

이런 간단한 테스트에 HTML form을 만들기는 귀찮다. 이때는 Postman을 사용하면 된다.

Postman 테스트 주의사항

- POST 전송시
  - Body ➜ `x-www-form-urlencoded` 선택
  - Headers에서 content-type: `application/x-www-form-urlencoded`로 지정된 부분 꼭 확인

HTTP 요청 데이터 - API 메시지 바디 - 단순 텍스트

- **HTTP message body**에 데이터를 직접 담아서 요청

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

- 먼저 가장 단순한 텍스트 메시지를 HTTP 메시지 바디에 담아서 전송하고, 읽어보자.
- HTTP 메시지 바디의 데이터를 InputStream을 사용해서 직접 읽을 수 있다.

RequestBodyStringServlet

```
package hello.servlet.basic.request;

import org.springframework.util.StreamUtils;

import javax.servlet.ServletException;
import javax.servlet.ServletInputStream;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.nio.charset.StandardCharsets;

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

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

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

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

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

```
}
```

Postman을 사용해서 테스트 해보자.

> **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.

문자 전송

- POST http://localhost:8080/request-body-string
- content-type: text/plain
- message body: `hello`
- 결과: `messageBody = hello`

HTTP 요청 데이터 - API 메시지 바디 - JSON

이번에는 HTTP API에서 주로 사용하는 JSON 형식으로 데이터를 전달해보자.

JSON 형식 전송

- POST http://localhost:8080/request-body-json
- content-type: **application/json**
- message body: `{"username": "hello", "age": 20}`
- 결과: `messageBody = {"username": "hello", "age": 20}`

JSON 형식 파싱 추가

JSON 형식으로 파싱할 수 있게 객체를 하나 생성하자

`hello.servlet.basic.HelloData`

```
package hello.servlet.basic;

import lombok.Getter;
import lombok.Setter;
```

```
@Getter @Setter
public class HelloData {

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

lombok이 제공하는 `@Getter`, `@Setter` 덕분에 다음 코드가 자동으로 추가된다.(눈에 보이지는 않는다.)

```
package hello.servlet.basic;

public class HelloData {

    private String username;
    private int age;

    //==== lombok 생성 코드 ====//
    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

> **참고**: 만약 잘 동작하지 않는다면 프로젝트 생성에 롬복 부분을 다시 확인하자.

```
package hello.servlet.basic.request;

import com.fasterxml.jackson.databind.ObjectMapper;
import hello.servlet.basic.HelloData;
import org.springframework.util.StreamUtils;

import javax.servlet.ServletException;
import javax.servlet.ServletInputStream;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.nio.charset.StandardCharsets;

/**
 * http://localhost:8080/request-body-json
 *
 * JSON 형식 전송
 * content-type: application/json
 * message body: {"username": "hello", "age": 20}
 *
 */
@WebServlet(name = "requestBodyJsonServlet", urlPatterns = "/request-body-
json")
public class RequestBodyJsonServlet extends HttpServlet {

    private ObjectMapper objectMapper = new ObjectMapper();

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

        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("ok");
    }
}
```

Postman으로 실행해보자.

- POST http://localhost:8080/request-body-json
- content-type: **application/json** (Body ➜ raw, 가장 오른쪽에서 JSON 선택)
- message body: `{"username": "hello", "age": 20}`

출력결과

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

> **참고**
> 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.

> **참고**
> HTML form 데이터도 메시지 바디를 통해 전송되므로 직접 읽을 수 있다. 하지만 편리한 파리미터 조회 기능(`request.getParameter(...)`)을 이미 제공하기 때문에 파라미터 조회 기능을 사용하면 된다.

HttpServletResponse - 기본 사용법

HttpServletResponse 역할

HTTP 응답 메시지 생성

- HTTP 응답코드 지정
- 헤더 생성
- 바디 생성

편의 기능 제공

- Content-Type, 쿠키, Redirect

HttpServletResponse - 기본 사용법

hello.servlet.basic.response.ResponseHeaderServlet

```
package hello.servlet.basic.response;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

/**
 * http://localhost:8080/response-header
 *
 */
@WebServlet(name = "responseHeaderServlet", urlPatterns = "/response-header")
public class ResponseHeaderServlet extends HttpServlet {

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse
```

```
response)
            throws ServletException, IOException {

        //[status-line]
        response.setStatus(HttpServletResponse.SC_OK); //200

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

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

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

Content 편의 메서드

```
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)
}
```

쿠키 편의 메서드

```
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("myCookie", "good");
    cookie.setMaxAge(600); //600초
    response.addCookie(cookie);
}
```

redirect 편의 메서드

```
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("/basic/hello-form.html");
}
```

HTTP 응답 데이터 - 단순 텍스트, HTML

HTTP 응답 메시지는 주로 다음 내용을 담아서 전달한다.

- 단순 텍스트 응답
  - 앞에서 살펴봄 (`writer.println("ok");`)
- HTML 응답
- HTTP API - MessageBody JSON 응답

HttpServletResponse - HTML 응답

hello.servlet.web.response.ResponseHtmlServlet

```
package hello.servlet.basic.response;
```

```java
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

@WebServlet(name = "responseHtmlServlet", urlPatterns = "/response-html")
public class ResponseHtmlServlet extends HttpServlet {

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

        //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 응답으로 HTML을 반환할 때는 content-type을 `text/html`로 지정해야 한다.

실행

- http://localhost:8080/response-html
- 페이지 소스보기를 사용하면 결과 HTML을 확인할 수 있다.

HTTP 응답 데이터 - API JSON

hello.servlet.web.response. ResponseJsonServlet

```
package hello.servlet.basic.response;

import com.fasterxml.jackson.databind.ObjectMapper;
import hello.servlet.basic.HelloData;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
 * http://localhost:8080/response-json
 *
 */
@WebServlet(name = "responseJsonServlet", urlPatterns = "/response-json")
public class ResponseJsonServlet extends HttpServlet {

    private ObjectMapper objectMapper = new ObjectMapper();

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

        //Content-Type: application/json
        response.setHeader("content-type", "application/json");
        response.setCharacterEncoding("utf-8");

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

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

        response.getWriter().write(result);
    }
}
```

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.

실행

- http://localhost:8080/response-json

> **참고**
> `application/json`은 스펙상 utf-8 형식을 사용하도록 정의되어 있다. 그래서 스펙에서 charset=utf-8과 같은 추가 파라미터를 지원하지 않는다. 따라서 `application/json` 이라고만 사용해야지 `application/json;charset=utf-8`이라고 전달하는 것은 의미 없는 파라미터를 추가한 것이 된다. response.getWriter()를 사용하면 추가 파라미터를 자동으로 추가해버린다. 이때는 response.getOutputStream()으로 출력하면 그런 문제가 없다.

정리

서블릿 컨테이너

https://mangkyu.tistory.com/14

Spring MVC의 등장

프론트 컨트롤러 – 디스패쳐 서블릿
어노테이션 기반의 편리함
그 외 (너무) 방대한 기능

은 다음시간에… 직접 만들어볼 예정입니다 ☺ 시간이 안될듯…

아직 다루지 않은 것 : Database 통신
-〉 JdbcTemplate

이건 직접 구현해봅시다.

사실 뼈대는 이미 드렸고 구현해오는 게 과제였습니다.(라이브러리화)

라이브러리 코드를 짜면서 얻을 점

1. 공통 로직을 분리하는 좋은 방법 익히기
2. 해당 라이브러리의 내부 구조 익히기

```java
public User findByUserId(String userId) throws SQLException {
    // connection 관리
    //statement 관리
    //resultset 관리
    Connection con = null;
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    try {
        con = ConnectionManager.getConnection();
        // sql
        String sql = "SELECT userId, password, name, email FROM USERS WHERE userid=?";
        pstmt = con.prepareStatement(sql);
        // 패러미터 선언 및 세팅 (1이므로 따로 선언 안해도 되긴 함.)
        // 여기서 트랜잭션은 사용 안함.
        pstmt.setString( parameterIndex: 1, userId);

        rs = pstmt.executeQuery();
        // row 데이터 추출
        User user = null;
        if (rs.next()) {
            user = new User(rs.getString( columnLabel: "userId"), rs.getString( columnLabel: "password"), rs.getString( columnLabel: "name"),
                    rs.getString( columnLabel: "email"));
        }

        return user;
    } finally {
        if (rs != null) {
            rs.close();
        }
        if (pstmt != null) {
            pstmt.close();
        }
        if (con != null) {
            con.close();
        }
    }
}
```

요구사항

9. 공통된 부분과 DB 쿼리 요청마다 달라지는 부분은 다음과 같습니다.

작업	공통 라이브러리	개발자가 구현할 부분
Connection 관리	O	X
SQL	X	O
Statement 관리	O	X
ResultSet 관리	O	X
Row 데이터 추출	X	O
패러미터 선언	X	O
패러미터 Setting	O	X
트랜잭션 관리	O	X

```java
public void insert(User user) throws SQLException {

    Connection con = null;

    PreparedStatement pstmt = null;
    try {
        con = ConnectionManager.getConnection();
        //sql
        String sql = createQueryForInsert();
        pstmt = con.prepareStatement(sql);
        setValueForInsert(user, pstmt);

        pstmt.executeUpdate();
    } finally {
        if (pstmt != null) {
            pstmt.close();
        }

        if (con != null) {
            con.close();
        }
    }
}

1 usage
private void setValueForInsert(User user, PreparedStatement pstmt) throws SQLException {
    pstmt.setString( parameterIndex: 1, user.getUserId());
    pstmt.setString( parameterIndex: 2, user.getPassword());
    pstmt.setString( parameterIndex: 3, user.getName());
    pstmt.setString( parameterIndex: 4, user.getEmail());
}

1 usage
private String createQueryForInsert() {
    return "INSERT INTO USERS VALUES (?, ?, ?, ?)";
}
```

1. 메소드 분리

Ctrl + Alt + M

2. 클래스 분리

```java
no usages
public class InsertJdbcTemplate {

    no usages
    public void insert(User user, UserDao userDao) throws SQLException {

        Connection con = null;

        PreparedStatement pstmt = null;
        try {
            con = ConnectionManager.getConnection();
            //sql
            String sql = userDao.createQueryForInsert();
            pstmt = con.prepareStatement(sql);
            userDao.setValueForInsert(user, pstmt);

            pstmt.executeUpdate();
        } finally {
            if (pstmt != null) {
                pstmt.close();
            }

            if (con != null) {
                con.close();
            }
        }
    }
}
```

```java
public class UserDao {
    1 usage
    public void insert(User user) throws SQLException {

        InsertJdbcTemplate jdbcTemplate = new InsertJdbcTemplate();
        jdbcTemplate.insert(user, userDao: this);
    }

    1 usage
    public void setValueForInsert(User user, PreparedStatement pstmt) throws SQLException {
        pstmt.setString( parameterIndex: 1, user.getUserId());
        pstmt.setString( parameterIndex: 2, user.getPassword());
        pstmt.setString( parameterIndex: 3, user.getName());
        pstmt.setString( parameterIndex: 4, user.getEmail());
    }

    1 usage
    public String createQueryForInsert() {
        return "INSERT INTO USERS VALUES (?, ?, ?, ?)";
    }
```

그 다음은 어떻게 하셨나요?

현재 코드의 특징

쿼리, 파라미터, mapping
모두
매번 달라질 확률이 큽니다.
단 나머지 코드는 공통.

```java
public void insert(User user) throws SQLException {

    Connection con = null;

    PreparedStatement pstmt = null;
    try {
        con = ConnectionManager.getConnection();
        //sql
        String sql = createQueryForInsert();
        pstmt = con.prepareStatement(sql);
        setValueForInsert(user, pstmt);

        pstmt.executeUpdate();
    } finally {
        if (pstmt != null) {
            pstmt.close();
        }

        if (con != null) {
            con.close();
        }
    }
}
1 usage
public abstract String createQueryForInsert();

1 usage
public abstract void setValueForInsert(User user, PreparedStatement pstmt) throws SQLException;
}
```

```java
1 usage
public void insert(User user) throws SQLException {

    InsertJdbcTemplate jdbcTemplate = new InsertJdbcTemplate(){

        1 usage
        public void setValueForInsert(User user, PreparedStatement pstmt) throws SQLException {
            pstmt.setString( parameterIndex: 1, user.getUserId());
            pstmt.setString( parameterIndex: 2, user.getPassword());
            pstmt.setString( parameterIndex: 3, user.getName());
            pstmt.setString( parameterIndex: 4, user.getEmail());
        }

        1 usage
        public String createQueryForInsert(){
            return "INSERT INTO USERS VALUES (?, ?, ?, ?)";
        }
    };

    jdbcTemplate.insert(user);
}
```

UserDao 의존성 분리
추상 클래스로 일급 객체 사용

그냥 상속 쓰면 안되나요?

매번 달라질 확률이 크다고 했는데 그럴 때마다 매번 새로운 클래스를 만들고 상속하게 해야 될텐데...

더 개선할 수 없을까요?

```
    }
    no usages
    public void update2(String sql) throws SQLException {

        Connection con = null;

        PreparedStatement pstmt = null;
        try {
            con = ConnectionManager.getConnection();
            pstmt = con.prepareStatement(sql);
            setValue(pstmt);

            pstmt.executeUpdate();
        } finally {
            if (pstmt != null) {
                pstmt.close();
            }

            if (con != null) {
                con.close();
            }
        }
    }
    1 usage   1 implementation
    public abstract String createQuery();

    2 usages   1 implementation
    public abstract void setValue(PreparedStatement pstmt) throws SQLException;
}
```

우선 사소하지만 이름..

JdbcTemplate의 User 의존성 제거

SQL 구문을 굳이 추상 메소드로?

아직 SELECT가 남아있습니다!

지금까지는 너무 쉽나요

우리의 라이브러리가 SELECT도 지원하기 위해

SELECT는 UPDATE와 어떤 차이가 있나요?

1. UPDATE의 경우 성공, 실패만 알면 됩니다. 실패하면 어차피 error를 catch하기 때문에 해당 기능은 있습니다.
2. SELECT는 DB에서 정보를 조회해오는 쿼리입니다.
3. 조회해온 쿼리 정보를 받아와야 합니다.

문제점 : 어떤 정보를 받아오건 객체에 MAPPING 해서 지원하고 싶다.

해당 Mapping은 어떤 정보를 받아올 지 모르니 매번 달라지고 개발자가 만들어야 할 영역입니다.
➔ 똑같이 추상 메소드로 일급 객체 만들게 하면 되지 않을까요?

```java
no usages
public abstract class SelectJdbcTemplate {

    1 usage
    public List query(String sql) throws SQLException {

        Connection con = null;
        PreparedStatement pstmt = null;
        ResultSet rs = null;
        try {
            con = ConnectionManager.getConnection();
            pstmt = con.prepareStatement(sql);
            setValues(pstmt);

            rs = pstmt.executeQuery();
            // row 데이터 추출
            List<Object> result = new ArrayList<>();
            while(rs.next()){
                result.add(mapRow(rs));
            }
            return result;

        } finally {
            if (rs != null) {
                rs.close();
            }
            if (pstmt != null) {
                pstmt.close();
            }
            if (con != null) {
                con.close();
            }
        }
    }

    no usages
    public Object queryForObject(String sql) throws SQLException{
        List result = query(sql);
        if(result.isEmpty()){
            return null;
        }
        return result.get(0);
    }

    1 usage
    abstract void setValues(PreparedStatement pstmt) throws SQLException;

    1 usage
    abstract Object mapRow(ResultSet rs) throws SQLException;
}
```

끝?

```java
no usages
public abstract class SelectJdbcTemplate {

    1 usage
    public List query(String sql) throws SQLException {

        Connection con = null;
        PreparedStatement pstmt = null;
        ResultSet rs = null;
        try {
            con = ConnectionManager.getConnection();
            pstmt = con.prepareStatement(sql);
            setValues(pstmt);

            rs = pstmt.executeQuery();
            // row 데이터 추출
            List<Object> result = new ArrayList<>();
            while(rs.next()){
                result.add(mapRow(rs));
            }
            return result;

        } finally {
            if (rs != null) {
                rs.close();
            }
            if (pstmt != null) {
                pstmt.close();
            }
            if (con != null) {
                con.close();
            }
        }
    }

    no usages
    public Object queryForObject(String sql) throws SQLException{
        List result = query(sql);
        if(result.isEmpty()){
            return null;
        }
        return result.get(0);
    }

    1 usage
    abstract void setValues(PreparedStatement pstmt) throws SQLException;

    1 usage
    abstract Object mapRow(ResultSet rs) throws SQLException;
}
```

불편한 점

1. select 쿼리를 위한 클래스가 따로 있다.
2. Object를 반환하므로 매번 타입 변환을 해서 써야 한다.

```java
3 usages  1 inheritor  1 related problem
public abstract class JdbcTemplate {

    1 usage
    public void update(String sql) throws SQLException {

        Connection con = null;

        PreparedStatement pstmt = null;
        try {
            con = ConnectionManager.getConnection();
            pstmt = con.prepareStatement(sql);
            setValues(pstmt);

            pstmt.executeUpdate();
        } finally {
            if (pstmt != null) {
                pstmt.close();
            }

            if (con != null) {
                con.close();
            }
        }
    }

    1 usage
    public List query(String sql) throws SQLException {

        Connection con = null;
        PreparedStatement pstmt = null;
        ResultSet rs = null;
        try {
            con = ConnectionManager.getConnection();
            pstmt = con.prepareStatement(sql);
            setValues(pstmt);

            rs = pstmt.executeQuery();
            // row 데이터 추출
            List<Object> result = new ArrayList<>();
            while(rs.next()){
                result.add(mapRow(rs));
            }
            return result;

        } finally {
            if (rs != null) {
```

select 쿼리를 위한 클래스가 따로 있다.

-> JdbcTemplate으로 통합

query와 queryForObject를 옮긴 뒤 추상 메소드에 mapRow를 추가?

```java
public void insert(User user) throws SQLException {

    JdbcTemplate jdbcTemplate = new JdbcTemplate(){

        2 usages
        public void setValues(PreparedStatement pstmt) throws SQLException {
            pstmt.setString( parameterIndex: 1, user.getUserId());
            pstmt.setString( parameterIndex: 2, user.getPassword());
            pstmt.setString( parameterIndex: 3, user.getName());
            pstmt.setString( parameterIndex: 4, user.getEmail());
        }

        1 usage
        @Override
        public Object mapRow(ResultSet rs) throws SQLException {
            return null;
        }
    };

    jdbcTemplate.update( sql: "INSERT INTO USERS VALUES (?, ?, ?, ?)");
}
```

필요없는 곳까지 mapRow를 정의.

템플릿 메서드 패턴의 단점

-> 해당 추상 클래스에 강한 의존성

```java
package core.jdbc;

import java.sql.ResultSet;
import java.sql.SQLException;

no usages
public interface RowMapper {
    no usages
    Object mapRow(ResultSet rs) throws SQLException;
}
```

PreparedStatementSetter.java

```java
package core.jdbc;

import java.sql.PreparedStatement;
import java.sql.SQLException;

no usages
public interface PreparedStatementSetter {
    no usages
    void setValues(PreparedStatement pstmt) throws SQLException;
}
```

각각의 추상 메소드를 인터페이스로 분리합시다.

```java
1 usage  1 related problem
public void update(String sql, PreparedStatementSetter pss) throws SQLException {

    Connection con = null;

    PreparedStatement pstmt = null;
    try {
        con = ConnectionManager.getConnection();
        pstmt = con.prepareStatement(sql);
        pss.setValues(pstmt);

        pstmt.executeUpdate();
    } finally {
        if (pstmt != null) {
            pstmt.close();
        }

        if (con != null) {
            con.close();
        }
    }
}

1 usage
public List query(String sql, PreparedStatementSetter pss, RowMapper rowMapper) throws SQLException {

    Connection con = null;
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    try {
        con = ConnectionManager.getConnection();
        pstmt = con.prepareStatement(sql);
        pss.setValues(pstmt);

        rs = pstmt.executeQuery();
        // row 데이터 추출
        List<Object> result = new ArrayList<>();
        while(rs.next()){
            result.add(rowMapper.mapRow(rs));
        }
        return result;

    } finally {
        if (rs != null) {
            rs.close();
        }
        if (pstmt != null) {
            pstmt.close();
        }
```

해당 인터페이스를 파라미터로 받자.
더 이상 추상 클래스일 필요 x

```java
public class UserDao {
    1 usage
    public void insert(User user) throws SQLException {

        JdbcTemplate jdbcTemplate = new JdbcTemplate();

        PreparedStatementSetter pss = new PreparedStatementSetter() {
            2 usages
            @Override
            public void setValues(PreparedStatement pstmt) throws SQLException {
                pstmt.setString( parameterIndex: 1, user.getUserId());
                pstmt.setString( parameterIndex: 2, user.getPassword());
                pstmt.setString( parameterIndex: 3, user.getName());
                pstmt.setString( parameterIndex: 4, user.getEmail());
            }
        };
        String sql = "INSERT INTO USERS VALUES (?, ?, ?, ?)";
        jdbcTemplate.update(sql, pss);
    }

    1 usage
    public User findByUserId(String userId) throws SQLException {
        JdbcTemplate jdbcTemplate = new JdbcTemplate();

        String sql = "SELECT userId, password, name, email FROM USERS WHERE userid=?";

        PreparedStatementSetter pss = new PreparedStatementSetter() {
            2 usages
            @Override
            public void setValues(PreparedStatement pstmt) throws SQLException {
                pstmt.setString( parameterIndex: 1, userId);
            }
        };

        RowMapper rowMapper = new RowMapper() {
            1 usage
            @Override
            public Object mapRow(ResultSet rs) throws SQLException {
                return new User(
                        rs.getString( columnLabel: "userId"),
                        rs.getString( columnLabel: "password"),
                        rs.getString( columnLabel: "name"),
                        rs.getString( columnLabel: "email"))
                        ;
            }
        };
        return (User)jdbcTemplate.queryForObject(sql, pss, rowMapper);
    }
```

드디어 끝..?

1. select 쿼리를 위한 클래스가 따로 있다.
2. Object를 반환하므로 매번 타입 변환을 해서 써야 한다.

사용자의 편의성을 늘려봅시다.
(사랑받는 동료 개발자가 됩시다!)

```java
package core.jdbc;

import java.sql.ResultSet;
import java.sql.SQLException;

5 usages  1 implementation
public interface RowMapper<T> {
    1 usage  1 implementation
    T mapRow(ResultSet rs) throws SQLException;
}
```

제네릭을 사용합시다.

```java
1 usage
public <T> List<T> query(String sql, PreparedStatementSetter pss, RowMapper<T> rowMapper) throws SQLException {

    Connection con = null;
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    try {
        con = ConnectionManager.getConnection();
        pstmt = con.prepareStatement(sql);
        pss.setValues(pstmt);

        rs = pstmt.executeQuery();
        // row 데이터 추출
        List<T> result = new ArrayList<>();
        while(rs.next()){
            result.add(rowMapper.mapRow(rs));
        }
        return result;

    } finally {
        if (rs != null) {
            rs.close();
        }
        if (pstmt != null) {
            pstmt.close();
        }
        if (con != null) {
            con.close();
        }
    }
}

1 usage
public <T> T queryForObject(String sql, PreparedStatementSetter pss, RowMapper<T> rowMapper) throws SQLException{
    List<T> result = query(sql, pss, rowMapper);
    if(result.isEmpty()){
        return null;
    }
    return result.get(0);
}

}
```

```java
1 usage
public User findByUserId(String userId) throws SQLException {
    JdbcTemplate jdbcTemplate = new JdbcTemplate();

    String sql = "SELECT userId, password, name, email FROM USERS WHERE userid=?";

    PreparedStatementSetter pss = new PreparedStatementSetter() {
        2 usages
        @Override
        public void setValues(PreparedStatement pstmt) throws SQLException {
            pstmt.setString( parameterIndex: 1, userId);
        }
    };

    RowMapper<User> rowMapper = new RowMapper<User>() {
        1 usage
        @Override
        public User mapRow(ResultSet rs) throws SQLException {
            return new User(
                    rs.getString( columnLabel: "userId"),
                    rs.getString( columnLabel: "password"),
                    rs.getString( columnLabel: "name"),
                    rs.getString( columnLabel: "email"))
                    ;
        }
    };
    return jdbcTemplate.queryForObject(sql, pss, rowMapper);
}
```

더... 더 없을까...

1. SQLException throws 좀.. 그만...
2. 쿼리에 값을 꼭 콜백 인터페이스로 전달해야할까?
3. 콜백 인터페이스는 람다가 제맛.

SQLException 그만..

```java
no usages
public class DataAccessException extends RuntimeException{

    no usages
    public DataAccessException() {
        super();
    }

    no usages
    public DataAccessException(String message) {
        super(message);
    }

    no usages
    public DataAccessException(String message, Throwable cause) {
        super(message, cause);
    }

    no usages
    public DataAccessException(Throwable cause) {
        super(cause);
    }

    no usages
    public DataAccessException(String message, Throwable cause, boolean enableSuppression, boolean writableStackTrace) {
        super(message, cause, enableSuppression, writableStackTrace);
    }
}
```

```java
1 usage
public void update(String sql, PreparedStatementSetter pss) throws DataAccessException {


    try(Connection con = ConnectionManager.getConnection();
    PreparedStatement pstmt = con.prepareStatement(sql)) {
        pss.setValues(pstmt);
        pstmt.executeUpdate();
    } catch (SQLException e){
        throw new DataAccessException();
    }
}
```

```java
1 usage
public void insert(User user) throws SQLException {

    JdbcTemplate jdbcTemplate = new JdbcTemplate();
```

가독성 최고!

쿼리에 값을 꼭 콜백 인터페이스로 전달해야 할까?

```
no usages
public void update2(String sql, Object... parameters) throws DataAccessException {

    try (Connection con = ConnectionManager.getConnection();
    PreparedStatement pstmt = con.prepareStatement(sql)){

        for (int i = 0; i < parameters.length; i++) {
            pstmt.setObject( parameterIndex: i+1, parameters[i]);
        }
        pstmt.executeUpdate();
    } catch (SQLException e){
        throw new DataAccessException();
    }
}
```

```
1 usage
public void insert(User user) {
    JdbcTemplate jdbcTemplate = new JdbcTemplate();
    String sql = "INSERT INTO USERS VALUES (?, ?, ?, ?)";
    jdbcTemplate.update2(sql, user.getUserId(), user.getPassword(), user.getName(), user.getEmail());
}
```

사용하기 아주 간단하네요!

콜백 인터페이스엔 람다가 제맛.

```
1 usage
public User findByUserId(String userId) {
    JdbcTemplate jdbcTemplate = new JdbcTemplate();

    String sql = "SELECT userId, password, name, email FROM USERS WHERE userid=?";

    PreparedStatementSetter pss = new PreparedStatementSetter() {
        2 usages
        @Override
        public void setValues(PreparedStatement pstmt) throws SQLException {
            pstmt.setString( parameterIndex: 1, userId);
        }
    };

    RowMapper<User> rowMapper = rs -> new User(
            rs.getString( columnLabel: "userId"),
            rs.getString( columnLabel: "password"),
            rs.getString( columnLabel: "name"),
            rs.getString( columnLabel: "email"));

    return jdbcTemplate.queryForObject(sql, pss, rowMapper);
}
```

다시...

라이브러리 코드를 짜면서 얻을 점

1. 공통 로직을 분리하는 좋은 방법 익히기
2. 해당 라이브러리의 내부 구조 익히기

여러분 스스로 라이브러리를 만들고 실력을 늘려봅시다!