6. REST

1. REST API 서버
2. REST 관련 스프링 어노테이션
3. REST API 서버 테스트
4. HTTP 서버 통신

1.2 REST

- REST 란
  - Representational Safe Transfer
  - 모바일과 같은 다양한 클라이언트의 등장하면서 Backend 하나로 다양한 Device를 대응하기 위해 REST의 필요성이 증대
  - HTTP URI를 통해 제어할 자원을 명시하고 HTTP Method(GET, POST, PUT, DELETE)를 통해 해당 자원을 제어하는 명령을 내리는 방식의 아키텍처(구현 지침).
  - REST의 원리를 따르는 시스템은 RESTful이란 용어로 지칭

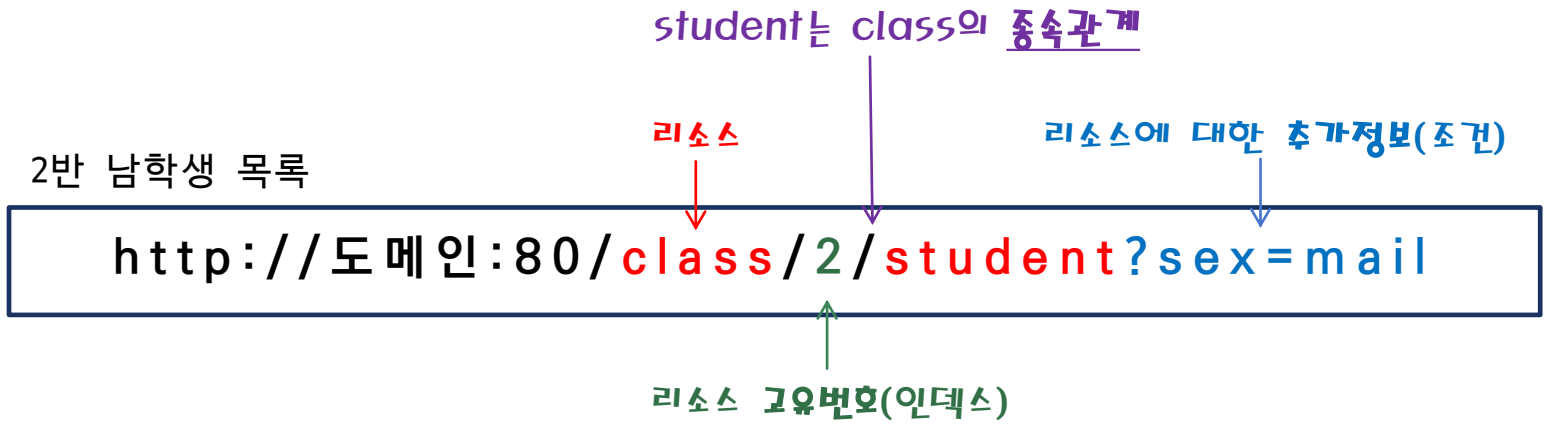
- REST API
  - REST 아키텍쳐 스타일을 따르는 웹 API

- REST API 구성요소

1.3 자원(URI)

- URI(Uniform Resource Identifier)
  - 고유한 리소스(자원)를 나타내는 주소로서 **자원의 식별자** 역할

```
http://도메인/class                  ← 반 목록
http://도메인/class/2                ← 2반의 정보
http://도메인/class/2/student        ← 2반의 학생 목록
http://도메인/class/2/student/10     ← 2반의 10번 학생
```



1.4 행위 (HTTP METHOD)

▶ REST요청

HTTP Method	REST URI
GET	/users
GET	/users/hong
POST	/users
PUT, PATCH	/users
DELETE	/users/hong

REST 요청방식은 4가지 메소드를 사용하여 CRUD를 처리하며 URI는 제어하려는 **자원**을 나타냄

Action
목록(R)
보기(R)
등록(C)
수정(U)
삭제(D)

▶ 기존의 요청

URI	HTTP Method
/getUserList.do	GET
/getUser.do?id=hong	GET
/insertUser.do	POST
/updateUser.do	POST
/deleteUser.do?id=hong	GET

기존의 요청방식은 GET과 POST만으로 CRUD를 처리하며, URI는 액션을 나타냄

1.5 표현(Representations)

- 자원의 현재 상태를 나타내는 데이터 형태(JSON, XML, HTML)

- JSON(JavaScript Object Notation)
  - Javascript에서 객체를 만들 때 사용하는 표현식
  - 경량(lightweight)의 DATA 교환 방식으로 이용
  - http://www.json.org/

  - JSON 형식
    - Object

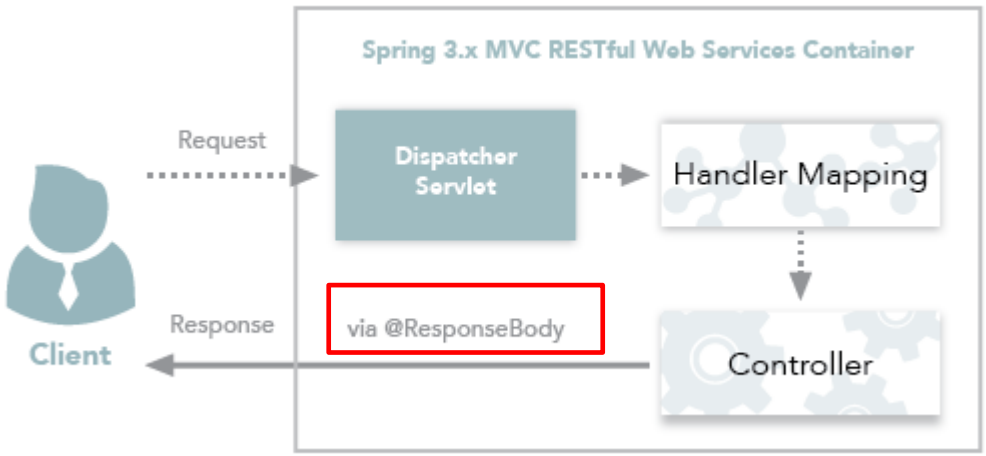
```
{ "firstName":"gildong", "age":20 }
```

    - List

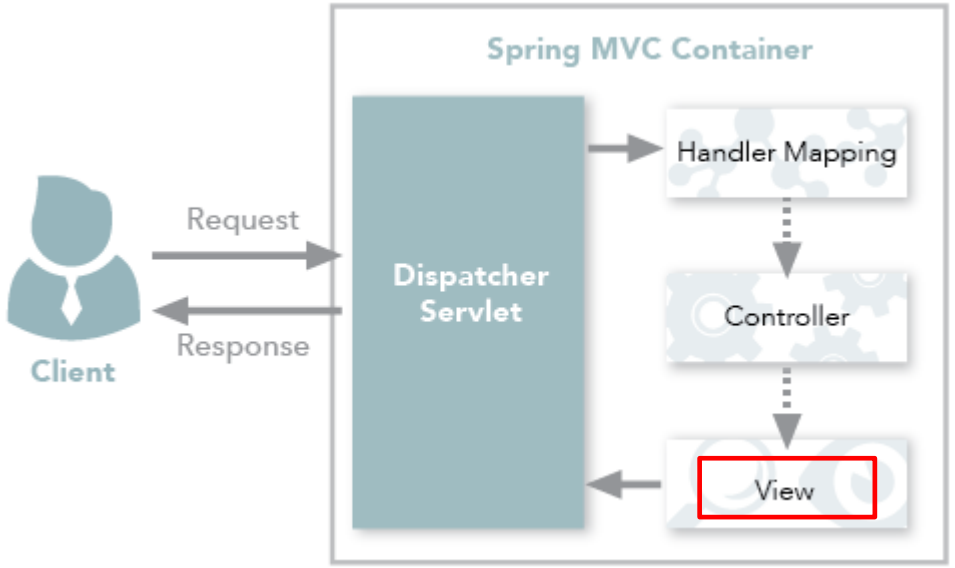
```
[ "scott", "tom", "king" ]
```

1.6 AJAX

AJAX



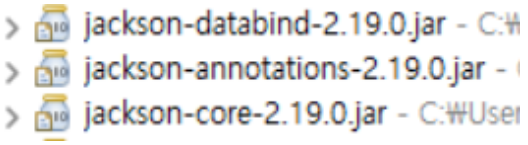
전통적인 페이지 요청 방식



2.1 JSON 라이브러리 의존성 설정

- Pom.xml

```
<dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
     <artifactId>jackson-databind</artifactId>
     <version>2.19.0</version>
</dependency>
```



- Spring 설정
  - 스프링 컨테이너에 MappingJacksonHttpMessageConverter 빈이 등록됨.
  - servlet-context.xml 파일에서 <mvc:Annotation-driven/> 설정이 REST 관련 어노테이션 처리

2.2 스프링 endpoint

@Controller

```
@Controller
public class RestfulUserController {

@RequestMapping("userList")
public @ResponseBody  Map getUserList() {
```

@RestController = @Controller + @ResponseBody

```
@RestController
public class RestfulUserController {

@RequestMapping("userList")
public Map getUserList(UserVO vo) {
```

2.3 JSON 어노테이션

@RequestBody

- HTTP Request Body(요청 몸체)를 Java 객체로 전달받을 수 있다.
- HttpMessageConverter를 사용해서 선언한 메서드 인자 타입으로 변환한다

@ResponseBody

- Java 객체를 HTTP Response Body(응답 몸체)로 전송할 수 있다.
- ModelAndView를 스킵하고 메시지변환기를 사용해서 리소스를 전송

```
@RequestMapping("getUser")
public @ResponseBody UserVO getUserList( @RequestBody UserVO vo) {
    return service.getUser(vo);
}
```

2.4 ObjectMapper

- JSON 스트링을 Java 객체로 , Java 객체를 JSON 스트링으로 변환
  - https://github.com/FasterXML/jackson -> tutorial 참조

▶ @ResponseBody

```
@RequestMapping("/ajaxResult")
public void test(HttpServletResponse response) throws IOException {
    UserVO vo = new UserVO("홍길동",20);
    ObjectMapper om =new ObjectMapper();
    String result = om.writeValueAsString(vo);
    response.setContentType("application/json; charset=UTF-8");
    response.getWriter().append(result);
}
```

▶ @RequestBody

```
String jsonStr = "{\"name\":\"홍길동\",\"age\":20}";
UserVO userVO = om.readValue(jsonStr, UserVO.class);
```

2.5 SERVER 응답

- ResponseEntity
  - Client의 플랫폼에 구애받지 않는 독립적인 Restful API를 개발하기 위해, 상태코드, 응답헤더, 응답메시지, 반환데이터를 모두 지정해서 반환해주기 위해 사용
  - Client요청에 대한 응답을 한번 더 감싸는 역할

```
@RequestMapping(method=RequestMethod.POST, consumes="application/json" )
public ResponseEntity getUserList() {
   URI location = ...;
   HttpHeaders responseHeaders = new HttpHeaders();
   responseHeaders.setLocation(location);
   responseHeaders.set("MyResponseHeader", "MyValue");
   ObjectMapper om = new ObjectMapper();
   String body= om.writeValueAsString(list);
   return new ResponseEntity<String>(body,  responseHeaders, HttpStatus.CREATED);
}
@RequestMapping(consumes="application/json" )
public ResponseEntity getUserList() {
    return ResponseEntity
            .created(location)
            .header("MyResponseHeader", "MyValue")
            .body("Hello World");
}
```

2.6 JACKSON 어노테이션

- @JsonIgnorePropertie
  - Serializer/Deseriallize 시 무시할 속성이나 속성 목록을 표시하는 데 사용
- @JsonIgnore
  - 필드 레벨에서 무시 될 수 있는 개별 속성을 표시
- @JsonIgnoreType
  - 주석이 달린 형식의 모든 속성을 무시하도록 시정
- @JsonInclude
  - 어노테이션 속성을 제외하는데 사용
  - Serialize 시 동작을 지정. 기본적으로 잭슨은 값의 유무와 상관없이 무조건 serialize하게 되지만 not null 이거나 none empty 일 경우에만 serialize 된다.
- @JsonProperty
  - Getter/seetter의 이름을 property와 다른 이름을 사용할 수 있도록 설정
- @JsonFormat

2.6 JACKSON 어노테이션

```
@JsonIncludeProperties({"passwd", "modDate"})
@JsonInclude(JsonInclude.Include.NOT_NULL)
public class UserVO {
    @JsonProperty(name="userid")
    private String id;
    private String passwd;

    @JsonFormat(pattern="yyyy-MM-dd")
    private Date regDate;

    @JsonIgnore
    private Date modDate;
}
```

3.1 CLIENT 개발 및 테스트

- XMLHttpRequest 객체
- fetch()
- $.ajax()
- axios()

- curl

- rest client 도구

3.2 fetch 함수

- HTTP content-type과 data의 관계
  - contentType을 application/json으로 명시해줄 경우 반드시 Stringify를 해야 한다
  - Ajax는 data를 pre-process 과정을 통해 query-string 형태로 변경시킨다.

▶ JSON string

```
fetch({
  url: "serverurl",
  type: "post",
  contentType: "application/json",
  data: JSON.stringify(data)
});
```

▶ query string(질의문자열)

```
const data = {name:"kim", age:20}
fetch({
  url: "severurl",
  type: "post",
  data: data
});
```

data는 queryString으로 인코딩 되어 전송
?name=kim&age=20

3.3 jQuery 함수

- HTTP content-type과 data의 관계
  - contentType을 application/json으로 명시해줄 경우 반드시 Stringify를 해야 한다
  - Ajax는 data 값이 아닐 경우 pre-process 과정을 통해 query-string 형태로 변경시킨다.

▶ JSON string

```
$.ajax({
  url: "serverurl",
  type: "post",
  data: JSON.stringify(data),
  contentType: "application/json"
});
```

▶ query string(질의문자열)

```
$.ajax({
  url: "severurl",
  type: "post",
  data: data
});
```

data의 값이 key-value pair형태로 인코딩 되어 전송

3.4 컨트롤러 테스트

MocMVC

```
@Log4j2
@TestInstance(Lifecycle.PER_CLASS)
@WebAppConfiguration                    // Test for Controller
@ExtendWith(SpringExtension.class)
@ContextConfiguration({"classpath:/spring/*-context.xml",
                       "file:src/main/webapp/WEB-INF/spring/appServlet/servlet-context.xml" })
public class ReplyControllerTests {
    @Setter(onMethod_ = { @Autowired })
    private WebApplicationContext ctx;

    private MockMvc mockMvc;

    @BeforeAll
    public void setup() {
        this.mockMvc = MockMvcBuilders.webAppContextSetup(ctx).build();
   }

}
```

3.4 컨트롤러 테스트 요청

MockMvcRequestBuilders : 요청 보내기

```
@Test
public void testList() throws Exception {
    ReplyVO reply = new ReplyVO();
    reply.setBno(20L);
    reply.setReply("마이콜");
    reply.setReplyer("댓글작성");

    String jsonStr = new ObjectMapper().writeValueAsString(reply);

    mockMvc.perform(MockMvcRequestBuilders
                    .get("/board/20/replies")
                    .param("page", "1")
            .contentType(MediaType.APPLICATION_JSON_VALUE)
            .content(jsonStr))
```

3.4 컨트롤러 테스트 응답처리

andDo() : 응답결과 출력

```
mockMvc.perform(get("/api/replies/1"))
       .andExpect(status().isOk())
       .andDo(print());
```

getContentAsString() : 응답결과 받기

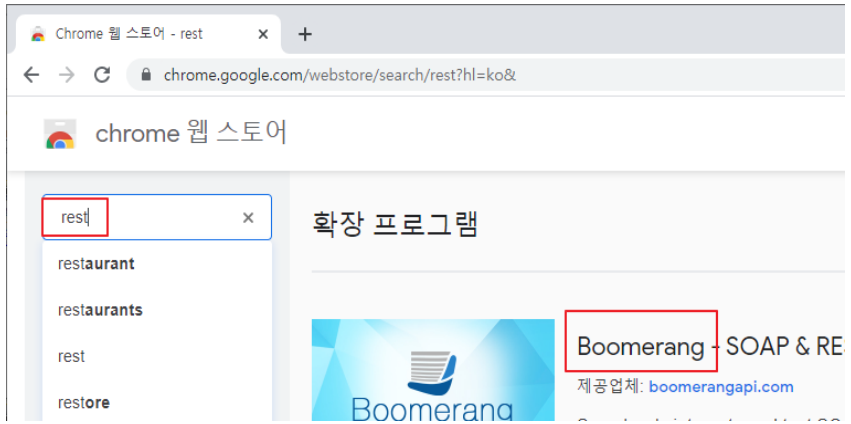
```
 String result = mockMvc
                 .perform( get("/api/replies")
                           .param("page","1"))
                 .andExpect(status().isOk())
                 .andExpect(content().contentType(MediaType.APPLICATION_JSON))
                 .andReturn().getResponse().getContentAsString();
```

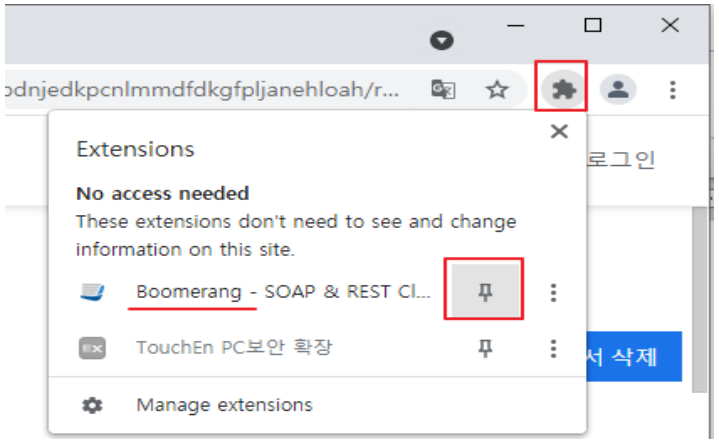
getModelAndView() : 응답페이지 받기

```
mockMvc.perform(get("/board/1"))
       .andReturn().getModelAndView().getModelMap()
```

3.5 확장앱 설치

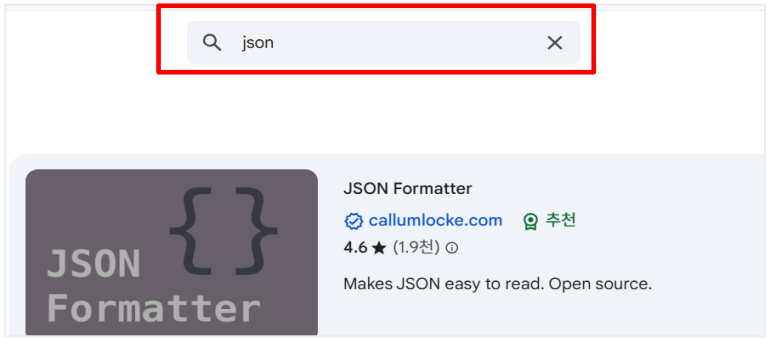
- 크롬웹스토어
- Boomerang : 개발된 API를 테스트할 수 있고, 팀원들간 공유를 할 수 있게 해주는 플랫폼





pin 고정

- json formatter



3.6 cURL

- https://curl.haxx.se/docs/manual.html
- 커맨드라인 환경에서 REST API 요청 보내기
- cURL 옵션

short	Long	설명
-I	--head	응답 헤더 출력(옵션이 없으면 응답 본문만 출력함)
-v	--verbose	중간 처리과정, 오류 메시지, 요청 메시지와 응답 메시지를 헤더와 본문을 포함해 전체 출력
-x		요청 메소드를 지정(옵션이 없으면 기본값은 GET)
-H		
-d	--data	HTTP post data
-L	--location	서버에서 HTTP301이나 HTTP302 응답이 왔을 경우 redirect URL로 따라간다.
-o	--output FILE	Remote에서 받아온 데이터를 FILE에 저장. (download 시 유용)
-T		서버에 파일 업로드

3.6 cURL

- XML

```
$curl –I –H “Content-Type:application/xml” –X get –d ‘<message><name>hong</name></message>’
http://localhost:80/getUserInfo
```

- JSON

```
$curl –I –H “Content-Type:application/json” –X get –d ‘{“name”:””jong”}’ http://localhost:80/getUserInfo
```

- User CRUD

```
조회:
    curl http://localhost/app/users/aa
삭제:
    curl http://localhost/app/users/aa  -X DELETE
등록:
    curl -X POST -H "Content-Type: application/json" -d
"{\"id\":\"test\",\"password\":\"1111\",\"name\":\"aaaa\",\"role\":\"Admin\"}"  http://locahost/app/users
수정:
   curl -X PUT  -H "Content-Type: application/json" -d "{\"id\":\"choi\",
\"name\":\"test\",\"password\":\"1234\",\"role\":\"Admin\"}" http://localhost/app/users
```

```
 curl -X POST -H "Content-Type: application/json" -d @data.json  http://locahost/app/users
```

3.7 CORS

- CORS
  - Cross Orgin Resource Sharing(외부 도메인 연결 요청)
  - 참고사이트: https://brunch.co.kr/@adrenalinee31/1

4.1 HTTP 서버 통신

- 참고사이트
  - https://sjh836.tistory.com/141

- 통신방법
  - URLConnection / HttpURLConnection / HttpClient
  - RestTemplate
  - WebClient

4.2 URLConnection

- JDK 1.2 의 java.net 패키지
- 단점
  - 4xx, 5xx 면 IOException 발생
  - 타임아웃 설정안됨
  - 쿠키제어 불가

```
URL url = new URL("http://");
  openConnection
  getInputStream, getOutputStream
  InputStream, OutputStream 처리
```

4.3 HttpClient

- Apache 프로젝트
- URLConnection와 비교하여 장점
  - 모든 응답코드를 읽을 수 있다. httpResponseStatusLine().getStatusCode()
  - 타임아웃 설정 가능
  - 쿠키제어 가능
- 단점
  - URLConnection 보다는 코드가 간결해졌지만 여전히 반복적인 코드가 존재
  - 스트림 처리 로직을 별도로 짜야한다
  - 응답의 컨텐츠타입에 따라 별도 로직이 필요

```
CloseableHttpClient httpClient = HttpClient.createDefault();
HttpGet  httpGet = new HttpGet("http://")
CloseableHttpResponse response = httpClient.execute(httpGet)
HttpEntity entity = response.getEntity();
Stream으로 entity.getContent() 처리
```

4.4 RestTemplate

- 스프링에서 제공하는 http 통신 템플릿
- http서버와의 통신을 단순화하여 기계적이고 반복적인 코드를 최대한 줄여줌
- HttpClient를 추상화하여 제공해주는 것으로 내부 통신은 Apache HttpComponents를 사용
- RESTful 원칙을 지킴
- json, xml 를 쉽게 응답

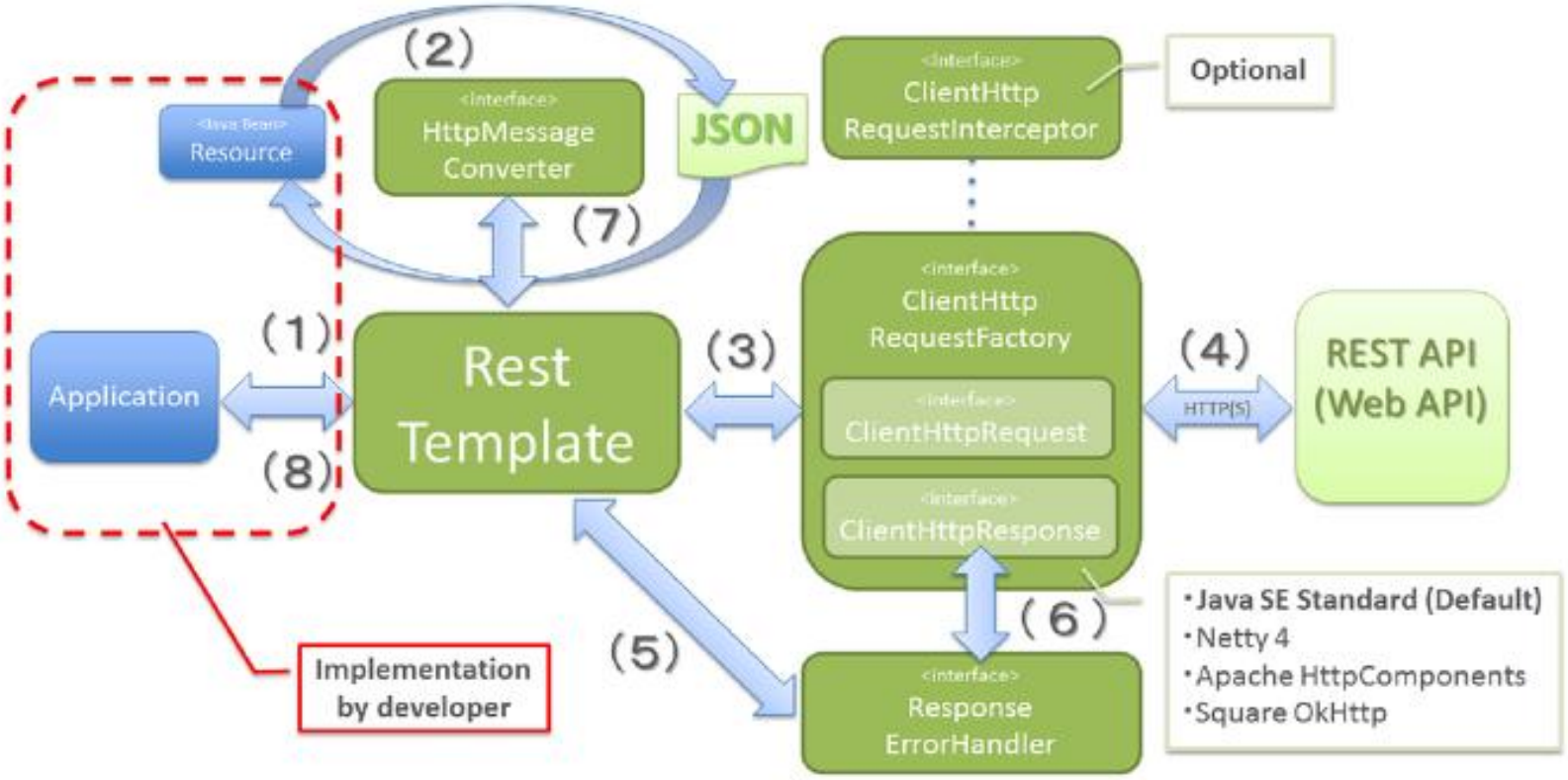
```
Public Pofile fetchFacebookProfile(String id) {
  RestTemplate rest = new RestTemplate();
  Return rest.getForObject("http://graph.facebook.com/{id}, UserDTO.class, id);
}
```

4.4 RestTemplate

- RestTemplate connectionpool 적용

```
HttpComponentsClientHttpRequestFactory factory = new HttpComponentsClientHttpRequestFactory();

factory.setReadTimeout(5000);                       // 읽기시간초과, ms
factory.setConnectTimeout(3000);                    // 연결시간초과, ms

HttpClient httpClient = HttpClientBuilder.create()
    .setMaxConnTotal(100)                           // connection pool 적용
    .setMaxConnPerRoute(5)                          // connection pool 적용
    .build();
factory.setHttpClient(httpClient);                  // 동기실행에 사용될 HttpClient 세팅

RestTemplate restTemplate = new RestTemplate(factory);

String url = "http://testapi.com/search?text=1234";
Object obj = restTemplate.getForObject("요청할 URI 주소", 자바클래스.class);

System.out.println(obj);
```

4.4 RestTemplate



4.4 RestTemplate

- RestTemplate 주요 메서드

RestTemplate Method	HTTP Method	설명
execute	Any	
exchange	Any	헤더세팅해서 HTTP Method로 요청보내고 ResponseEntity로 반환받음
getForObject	GET	get 요청을 보내고 java object로 매핑받아서 반환받음
getForEntity	GET	get 요청을 보내고 ResponseEntity로 반환받음
postForLocation	POST	post 요청을 보내고 java.net.URI 로 반환받음
postForObject	POST	post 요청을 보내고 ResponseEntity로 반환받음
put	PUT	
delete	DELETE	
headForHeaders	HEAD	
optionsForAllow	OPTIONS	

4.5 WebClient

- RestTemplate를 대체하는 HTTP 클라이언트.
- 스프링 5.0 에서 WebClient가 나왔고 현재는 WebClient를 사용하기를 권고.
- WebClient는 Spring WebFlux에서 HTTP Client로 사용되는 `비동기`적으로 작동하는 모듈.
- RestTemplate는 요청에 대한 응답이 수신될 때까지 호출한 스레드를 `Blocking`하는 동기식 클라이언트입니다. 반면에 WebClient는 `Non-Blocking` 방식이기 때문에 메인 스레드를 차단하지 않고 여러 요청을 동시에 수행이 가능.

```
import org.springframework.web.reactive.function.client.WebClient;

Mono<Person> entityMono = client.get()
    .uri("/persons/1")
    .accept(MediaType.APPLICATION_JSON)
    .exchangeToMono(response -> {
        if (response.statusCode().equals(HttpStatus.OK)) {
            return response.bodyToMono(Person.class);
        }
        else {
            return response.createError();
        }
    });
```