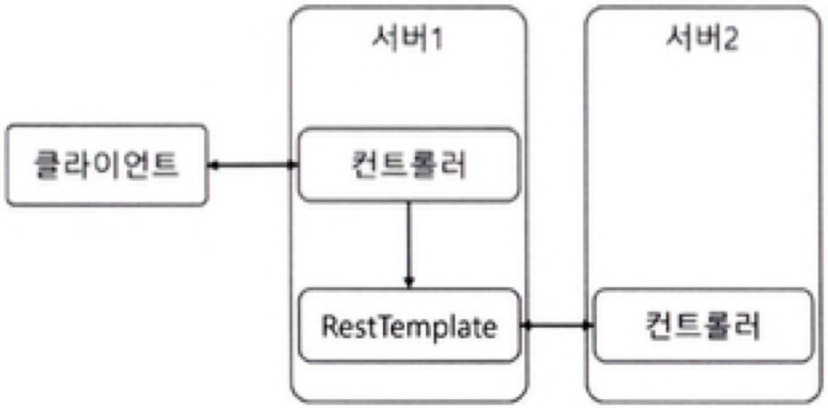
Server Communication

Server Communication

- Server Communication
  - ✓ 최근에 개발되는 서비스들은 마이크로서비스 아키텍처(MSA)를 주로 채택
  - ✓ MSA는 애플리케이션이 가지고 있는 기능(서비스)이 하나의 비즈니스 범위만 가지는 형태
  - ✓ MSA 환경에서는 기능에 따라 분리된 여러 컴포넌트가 존재하고 컴포넌트 성격에 따라 적합한 데이터 저장소를 선택해야 하며 물리적, 논리적으로 분리되어야 하는데 애플리케이션이 제공하는 서비스가 복잡하다면 하나 이상의 컴포넌트를 통합하여 서비스를 제공하며 서비스가 확장되고 기능이 복잡해지면 컴포넌트 종류는 많아지고 자연히 분산 저장된 데이터를 통합해야 할 상황이 발생
  - ✓ 모놀리식 아키텍쳐에서는 데이터 저장소가 하나이므로 조인 쿼리를 사용하거나 클래스들끼리 의존하면 되지만 MSA 아키텍쳐에서는 데이터 저장소가 여러 개 이므로 조인이나 클래스들끼리 의존 관계를 만들 수 없음
  - ✓ 데이터를 통합하기 위해서 가장 많이 사용하는 방법은 Remote API Call
  - ✓ API를 사용하여 다른 시스템의 데이터를 조회하거나 수정, 생성, 삭제 할 수 있음
  - ✓ REST-API, gRPC, XML-RPC, SOAP 같은 프로토콜을 사용할 수 있지만 REST-API를 가장 많이 사용

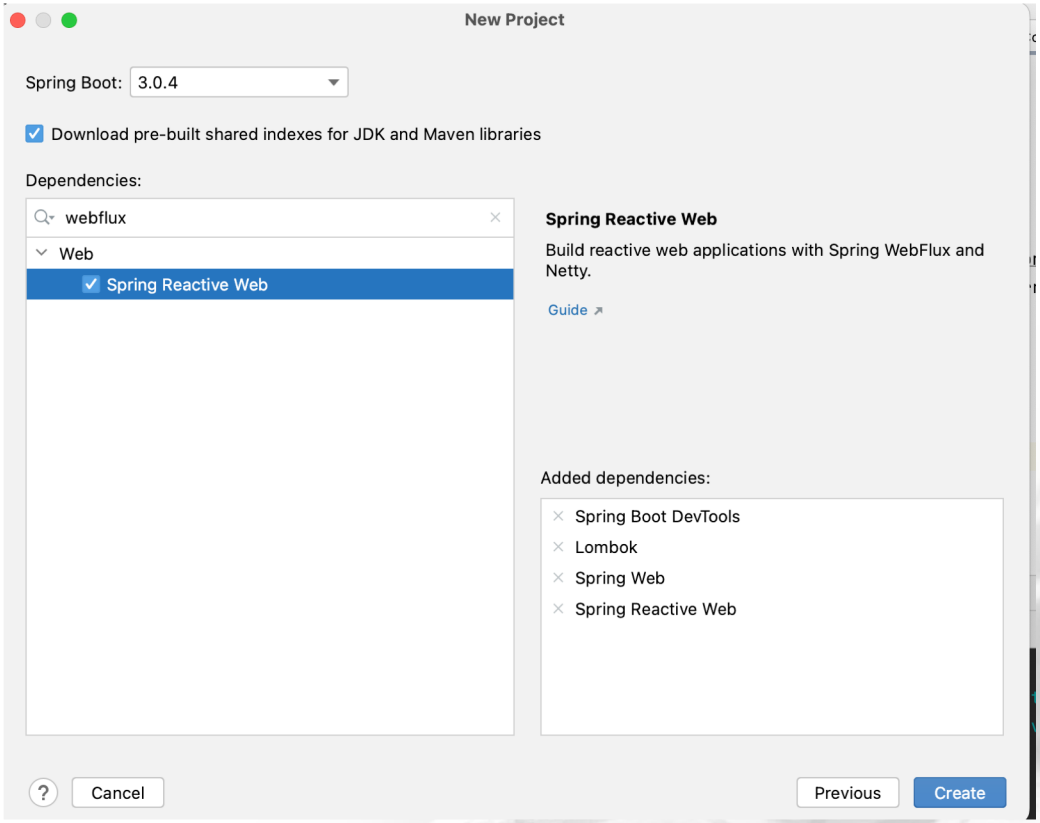
Server Communication

- Server Communication
  - ✓ 각 애플리케이션은 자신이 가진 기능을 API로 외부에 노출하고 다른 서버가 그러한 API를 호출해서 사용할 수 있게 구성되므로 각 서버가 다른 서버의 클라이언트가 되는 경우도 많음



Server Communication

- API Client 프로젝트 생성
  - ✓ Lombok, Spring DevTools, Spring Web, reactive-web 의존성을 가진 프로젝트 생성



Server Communication

- API Client 프로젝트 생성
  - ✓ application.yml 파일을 생성하고 포트를 설정

```
server:
  port: 80
```

Server Communication

- HttpURLConnection
  - ✓ java.net.HttpURLConnection 클래스는 URLConnection을 구현한 클래스(java.net 클래스에서 제공하는 URL 요청을 위한 클래스)
  - ✓ URLConnection은 웹을 통해 데이터를 주고 받는데 사용(RFC 2616을 따름)
  - ✓ 데이터의 타입이나 길이는 거의 제한이 없음
  - ✓ 미리 길이를 알지 못하는 스트리밍 데이터를 주고 받는데 사용
  - ✓ 요청 방식을 확인 or 설정, redirect 여부 결정, 응답 코드와 메시지를 Read , 프록시 서버가 사용되었는지 여부 확인 메서드 등을 가지고 있음
  - ✓ 다양한 HTTP 응답 코드에 해당하는 상수 값들이 정의되어 있음
  - ✓ URLConnection 클래스와 마찬가지로 생성자가 protected로 선언되어있기 때문에 기본적으로는 개발자가 직접 HttpURLConnection 객체를 생성할 수 없음
  - ✓ URL을 사용하는 URL 객체의 openConnection() 메서드가 리턴하는 URLConnection 객체는 HttpURLConnection의 인스턴스가 될 수 있기 때문에 리턴된 URLConnection을 HttpURLConnection으로 캐스팅해서 사용

Server Communication

- HttpURLConnection
  - ✓ 요청 방식 설정
    - HttpURLConnection은 기본적으로 GET 메서드 사용
    - setRequestMethod() 메서드를 사용해서 메서드 변경 가능
    - 요청 방식은 대문자로 전달
    - 지정된 요청 방식 이외의 파라미터 전달시 java.net.ProtocolException이 발생.
    - Method: public void setRequestMethod(String method)
  - ✓ disconnect() 메서드는 특정 호스트와의 대화가 끝난 시점에 클라이언트가 서버와의 연결을 끊을 수 있도록 함: public abstract void disconnect()
  - ✓ 서버 응답 처리: 응답 코드 와 응답 메시지 사용
    - int getResponseCode()
    - public String getResponseMessage()

Server Communication

- ❖ HttpURLConnection
  - ✓ Kakao Open API 데이터 가져오기
    - ● build.gradle 파일의 JSON 파싱을 위한 의존성을 설정

      implementation group: 'org.json', name: 'json', version: '20090211'

Server Communication

- HttpURLConnection
  - ✓ Kakao Open API 데이터 가져오기
    - Controller.JDKCoreController 클래스를 만들고 작성

```
@RestController
public class JDKCoreController {
    @GetMapping("/urlconnection")
    public ResponseEntity<List> onlyJavaHttp(){

        String json = null;
        List list = null;
        try {
            String urlAddr =
 "https://dapi.kakao.com/v3/search/book?target=title&query=";
            //파라미터 인코딩
            String title = "삼국지";
            urlAddr += URLEncoder.encode(title, "utf-8");
            URL url = new URL(urlAddr);
```

Server Communication

- ❖ HttpURLConnection
  - ✓ Kakao Open API 데이터 가져오기
    - ● Controller.JDKCoreController 클래스를 만들고 작성

```
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
//요청 헤더 설정
conn.setRequestProperty("Authorization", "KakaoAK
df1ba560c0cb8327f3f24166c0fd193a");
StringBuilder sb = new StringBuilder();
if (conn != null) {
    conn.setConnectTimeout(20000);
    conn.setUseCaches(false);
    if (conn.getResponseCode() == HttpURLConnection.HTTP_OK) {
        InputStreamReader isr = new
InputStreamReader(conn.getInputStream());
        BufferedReader br = new BufferedReader(isr);
```

Server Communication

- HttpURLConnection
  - ✓ Kakao Open API 데이터 가져오기
    - Controller.JDKCoreController 클래스를 만들고 작성

```
            while (true) {
                String line = br.readLine();
                if (line == null) {
                    break;
                }
                sb.append(line);
            }
            br.close();
            conn.disconnect();
        }
    }
    json = sb.toString();
    System.out.printf("%s₩n", json);
```

Server Communication

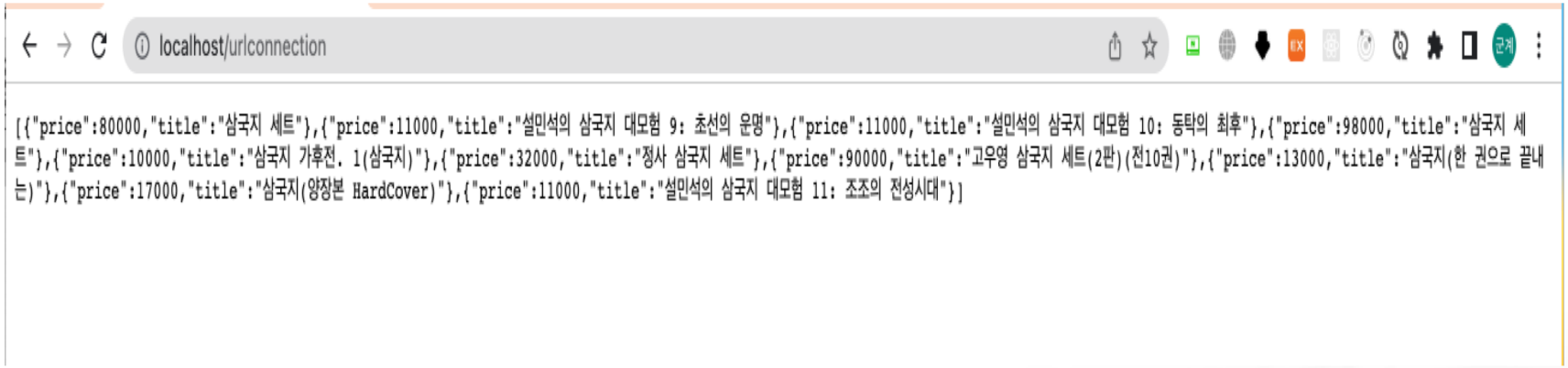
- ❖ HttpURLConnection
  - ✓ Kakao Open API 데이터 가져오기
    - ● Controller.JDKCoreController 클래스를 만들고 작성

```
                JSONObject jsonObject = new JSONObject(json);
                JSONArray documents = jsonObject.getJSONArray("documents");

                list = new ArrayList();
                for(int i=0; i<documents.length(); i++){
                    JSONObject object = documents.getJSONObject(i);
                    Map<String, Object> map = new HashMap<>();
                    map.put("title", object.getString("title"));
                    map.put("price", object.getLong("price"));
                    list.add(map);
                }
            } catch (Exception e) {
                System.out.printf("에러 발생:%s₩n", e.getMessage());
            }
            return new ResponseEntity<>(list, HttpStatus.OK);
        }
    }
```

Server Communication

- HttpURLConnection
  - ✓ Kakao Open API 데이터 가져오기
    - 확인



Server Communication

- HttpClient
  - ✓ HttpURLConnection을 사용할 때보다 좀 더 편하게 개발
  - ✓ JDK 11 버전에서 추가된 API
  - ✓ HttpClient를 이용한 요청 처리

Server Communication

- HttpClient
  - ✓ HttpURLConnection을 사용할 때보다 좀 더 편하게 개발
  - ✓ JDK 11 버전에서 추가된 API

Server Communication

- HttpClient
  - ✓ JDK 11 버전에서 추가된 API
    - Controller 클래스에 메서드 추가

```
@GetMapping("/httpclient")
  public ResponseEntity<List>  apacheHttpClient() {
      String json = null;
      List list = null;

      try {
          //헤더값은 이런식으로 사용합니다(키, 값)headers => "Content-
type","plain/text"
          HttpClient client =
HttpClient.newBuilder().version(HttpClient.Version.HTTP_1_1).build();
          String urlAddr =
"https://dapi.kakao.com/v3/search/book?target=title&query=";
          //파라미터 인코딩
          String title = "삼국지";
          urlAddr += URLEncoder.encode(title, "utf-8");
```

Server Communication

- HttpClient
  - ✓ JDK 11 버전에서 추가된 API
    - Controller 클래스에 메서드 추가

```
String result = client.sendAsync(
        HttpRequest.newBuilder(
                new URI(urlAddr)).GET().headers("Authorization",
                "KakaoAK
df1ba560c0cb8327f3f24166c0fd193a").build(),  //GET방식 요청
                HttpResponse.BodyHandlers.ofString()  //응답은 문자형태
            ).thenApply(HttpResponse::body)  //thenApply메소드로
응답body값만 받기
            .get();  //get메소드로 body값의 문자를 확인
        System.out.println(result);

        JSONObject jsonObject = new JSONObject(result);
        JSONArray documents = jsonObject.getJSONArray("documents");
```

Server Communication

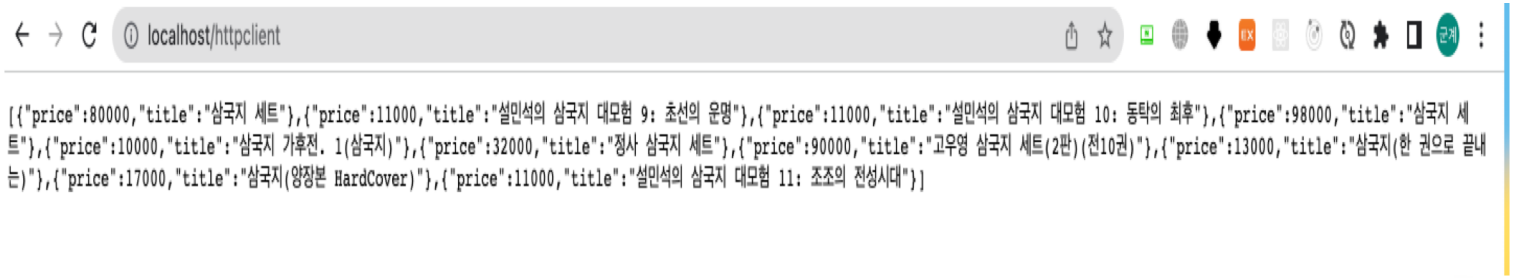
- HttpClient
  - ✓ JDK 11 버전에서 추가된 API
    - Controller 클래스에 메서드 추가

```
            list = new ArrayList();
            for(int i=0; i<documents.length(); i++){
                JSONObject object = documents.getJSONObject(i);
                Map<String, Object> map = new HashMap<>();
                map.put("title", object.getString("title"));
                map.put("price", object.getLong("price"));
                list.add(map);
            }

        }catch (Exception e){
            System.out.printf("에러 발생:%s₩n", e.getMessage());
        }
        return new ResponseEntity<>(list, HttpStatus.OK);
    }
```

Server Communication

- HttpClient
  - ✓ JDK 11 버전에서 추가된 API
    - 확인



Server Communication

- API Server Project
  - ✓ spring-boot-starter-web 과 lombok 의존성을 설정한 프로젝트 생성
  - ✓ application.yml 파일에 포트 설정

```
server:
  port: 9000
```

Server Communication

- API Server Project
  - ✓ 회원 정보 와 관련된 MemberDTO 클래스 생성 -

```
@Getter
@Setter
@ToString
public class MemberDTO {
    private String name;
    private String email;
    private String organization;
}
```

Server Communication

- API Server Project
  - ✓ 회원 정보 와 관련된 RestController 클래스 생성 – controller.CrudController

```
@RestController
@RequestMapping("/api/v1/crud-api")
public class CrudController {
    @GetMapping
    public String getName(){
        return "adam";
    }

    @GetMapping(value="/{variable}")
    public String getVariable(@PathVariable String variable){
        return variable;
    }

    @GetMapping("/param")
    public String getNameWithVariable(@RequestParam String name){
        return "Hello " + name + "!";
    }
```

Server Communication

- API Server Project
  - ✓ 회원 정보 와 관련된 RestController 클래스 생성 – controller.CrudController

```
@PostMapping
public ResponseEntity<MemberDTO> getMember(
        @RequestBody MemberDTO request,
        @RequestParam String name,
        @RequestParam String email,
        @RequestParam String organization){

    System.out.println(request.getName());
    System.out.println(request.getEmail());
    System.out.println(request.getOrganization());

    MemberDTO memberDTO = new MemberDTO();
    memberDTO.setName(name);
    memberDTO.setEmail(email);
    memberDTO.setOrganization(organization);

    return ResponseEntity.status(HttpStatus.OK).body(memberDTO);
}
```

Server Communication

- API Server Project
  - ✓ 회원 정보 와 관련된 RestController 클래스 생성 – controller.CrudController

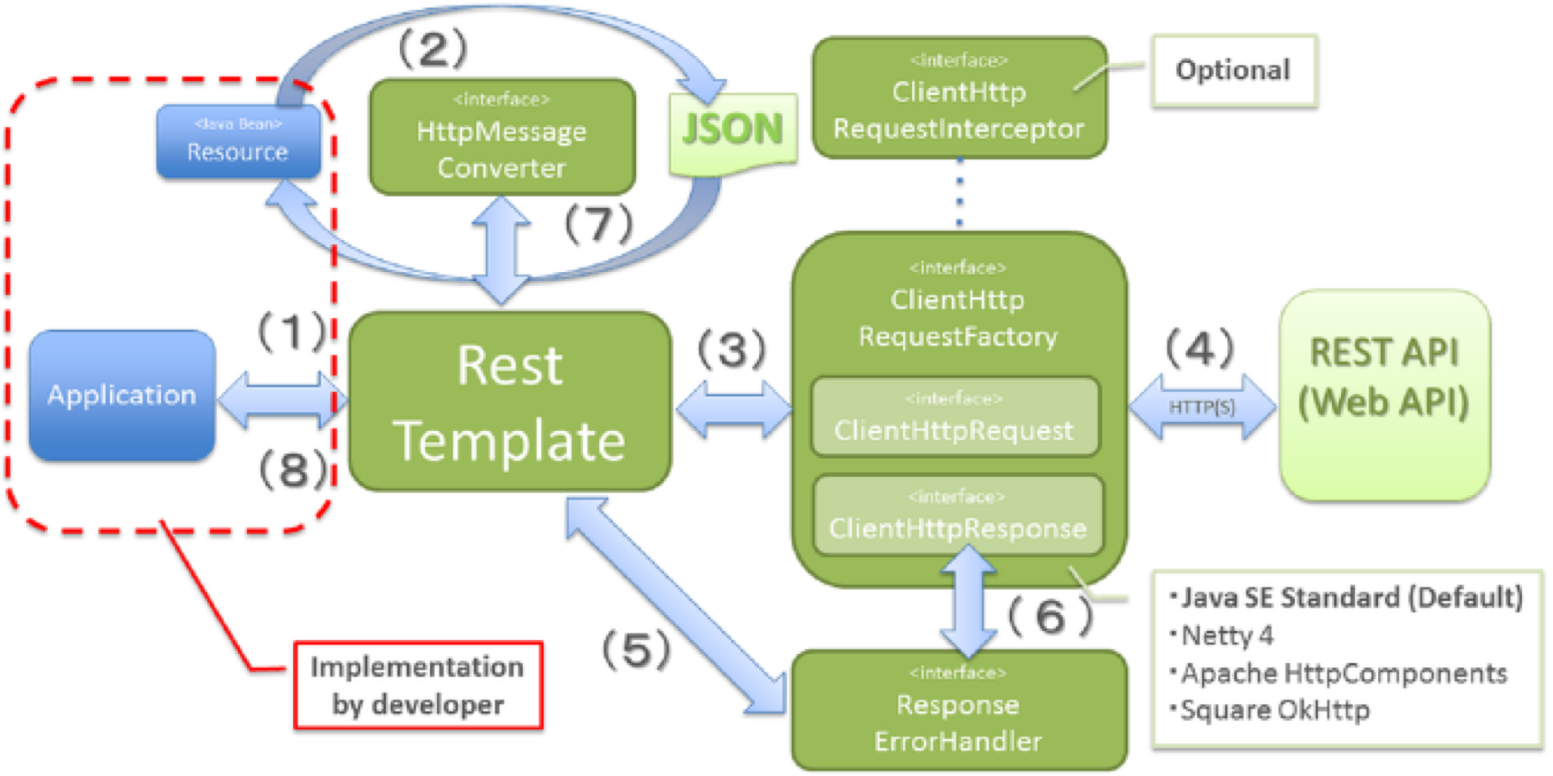
```
@PostMapping(value = "/add-header")
public ResponseEntity<MemberDTO> addHeader(
        @RequestHeader("my-header") String header,
        @RequestBody MemberDTO memberDTO) {
    System.out.println(header);
    return ResponseEntity.status(HttpStatus.OK).body(memberDTO);
    }
}
```

Server Communication

- RestTemplate
  - ✓ org.springframework.http.client 패키지에 있는 다른 서버의 REST-API를 실행할 수 있는 메서드를 제공하는 클래스
  - ✓ httpclient는 HTTP를 사용하여 통신하는 범용 라이브러리이고 RestTemplate 클래스는 HttpClient를 추상화(HttpEntity의 json, xml 등)해서 제공
  - ✓ 다른 서버의 REST-API를 호출할 수 있도록 URI를 설정하고 GET, POST 같은 HTTP 요청 메서드를 사용할 수 있고 요청 메시지의 헤더, 바디를 구성할 수 있는 클래스를 사용할 수 있으며 응답 메시지의 헤더, 상태 코드, 바디를 조회할 수 있는 클래스를 사용할 수 있음
  - ✓ 특징
    - Java 객체를 HTTP 요청의 Body 로 변환하고 HTTP 응답 메시지 Body를 자바 객체로 변환하는 기능을 제공하는데 이 때 메시지의 Content-Type에 따라 적절한 Converter를 찾아 동작하는데 RestTemplate은 이 Converter를 쉽게 확장할 수 있는 구조를 가지고 있어서 JSON 뿐 만 아니라 XML 같은 메시지를 자바 객체로 쉽게 변환할 수 있음
    - HTTP 프로토콜을 직접 사용하여 서버의 REST-API를 호출하는 일련의 네트워크 작업을 추상화한 메서드를 제공
    - 네트워크 기능을 별도의 클래스에 위임해서 커넥션 관리나 메시지를 주고받는 저수준 네트워크 작업은 구현체에 따라 다르게 동작
    - 인터셉터 기능을 제공하며 멀티 스레드 환경에 안전한 클래스이므로 스프링 빈으로 객체를 생성하고 필요한 곳에 주입해서 사용하는 것이 가능

Server Communication

- RestTemplate
  - ✓ 동작 원리



Server Communication

- RestTemplate
  - ✓ 동작 원리
    - 애플리케이션이 RestTemplate를 생성 -> URI, HTTP 메소드 등의 헤더를 담아 요청
    - RestTemplate는 HttpMessageConverter를 사용하여 requestEntity를 요청 메시지로 변환
    - RestTemplate는 ClientHttpRequestFactory로 부터 ClientHttpRequest를 가져와서 요청을 전송
    - ClientHttpRequest는 요청 메시지를 만들어 HTTP 프로토콜을 통해 서버와 통신
    - RestTemplate는 ResponseErrorHandler로 오류를 확인하고 있다면 처리 로직을 태움
    - ResponseErrorHandler는 오류가 있다면 ClientHttpResponse에서 응답 데이터를 가져와서 처리
    - RestTemplate는 HttpMessageConverter를 이용해서 응답 메시지를 java object(class responseType)으로 변환
    - 애플리케이션에 반환

Server Communication

- RestTemplate
  - ✓ 핵심이 되는 클래스
    - HttpMessageConverter
      - 스프링 MVC 프레임워크에서 Content 타입에 따라 메시지를 변환하는 역할을 수행
      - 서버에 요청하는 메시지와 응답 받은 메시지를 자바 객체로 변환할 때 동작
    - ClientHttpRequestFactory
      - HTTP 프로토콜을 사용하여 클라이언트의 요청 과 서버 응답을 처리
      - 요청은 ClientHttpRequest 클래스로 응답은 ClientHttpResponse 클래스로 추상화되어 처리
      - ClientHttpRequest 클래스의 execute() 메서드를 호출하면 ClientHttpResponse 객체를 받을 수 있음
      - 구현체
        - SimpleClientRequestFactory: 동기식으로 동작하면 JDK에서 제공하는 라이브러리를 사용하여 구현

Server Communication

- RestTemplate
  - ✓ 주요 메서드

RestTemplate Method	HTTP Method	설명
executeAny		
exchange	Any	헤더 세팅해서 HTTP Method로 요청을 전송하고 ResponseEntity로 리턴
getForObject	GET	get 요청을 보내고 java object로 매핑받아서 리턴
getForEntity	GET	get 요청을 보내고 ResponseEntity로 리턴
postForLocation	POST	post 요청을 보내고 java.net.URI 로 리턴
postForObject	POST	post 요청을 보내고 ResponseEntity로 리턴
put	PUT	
delete	DELETE	
headForHeaders	HEAD	
optionsForAllow	OPTIONS	

Server Communication

- RestTemplate
  - ✓ RestTemplate를 이용한 ServerBox 활용
    - Service 클래스를 생성하고 메서드 작성 – service.RestTemplateService

```
@Service
public class RestTemplateService {

    public String getName() {
        URI uri = UriComponentsBuilder
            .fromUriString("http://localhost:9000")
            .path("/api/v1/crud-api")
            .encode()
            .build()
            .toUri();

        RestTemplate restTemplate = new RestTemplate();
        ResponseEntity<String> responseEntity = restTemplate.getForEntity(uri,
     String.class);

        return responseEntity.getBody();
    }
```

Server Communication

- RestTemplate
  - ✓ RestTemplate를 이용한 ServerBox 활용
    - Service 클래스를 생성하고 메서드 작성 – service.RestTemplateService

```
public String getNameWithPathVariable() {
    URI uri = UriComponentsBuilder
        .fromUriString("http://localhost:9000")
        .path("/api/v1/crud-api/{name}")
        .encode()
        .build()
        .expand("adam") // 복수의 값을 넣어야할 경우 , 를 추가하여 구분
        .toUri();

    RestTemplate restTemplate = new RestTemplate();
    ResponseEntity<String> responseEntity = restTemplate.getForEntity(uri,
 String.class);

    return responseEntity.getBody();
}
```

Server Communication

- RestTemplate
  - ✓ RestTemplate를 이용한 ServerBox 활용
    - Service 클래스를 생성하고 메서드 작성 – service.RestTemplateService

```
public String getNameWithParameter() {
    URI uri = UriComponentsBuilder
        .fromUriString("http://localhost:9000")
        .path("/api/v1/crud-api/param")
        .queryParam("name", "adam")
        .encode()
        .build()
        .toUri();

    RestTemplate restTemplate = new RestTemplate();
    ResponseEntity<String> responseEntity = restTemplate.getForEntity(uri,
 String.class);

    return responseEntity.getBody();
}
```

Server Communication

- RestTemplate
  - ✓ RestTemplate를 이용한 ServerBox 활용
    - Service 클래스를 생성하고 메서드 작성 – service.RestTemplateService

```
public ResponseEntity<MemberDTO> postWithParamAndBody() {
    URI uri = UriComponentsBuilder
        .fromUriString("http://localhost:9000")
        .path("/api/v1/crud-api")
        .queryParam("name", "adam")
        .queryParam("email", "itstudy@kakao.com")
        .queryParam("organization", "adamsoft")
        .encode()
        .build()
        .toUri();
    MemberDTO memberDTO = new MemberDTO();
    memberDTO.setName("flature!!");
    memberDTO.setEmail("flature@gmail.com");
    memberDTO.setOrganization("Around Hub Studio");

    RestTemplate restTemplate = new RestTemplate();
    ResponseEntity<MemberDTO> responseEntity =
 restTemplate.postForEntity(uri, memberDTO,
            MemberDTO.class);
    return responseEntity;
}
```

Server Communication

- ❖ RestTemplate
  - ✓ RestTemplate를 이용한 ServerBox 활용
    - ● Service 클래스를 생성하고 메서드 작성 – service.RestTemplateService

```
public ResponseEntity<MemberDTO> postWithHeader() {
    URI uri = UriComponentsBuilder
        .fromUriString("http://localhost:9000")
        .path("/api/v1/crud-api/add-header")
        .encode()
        .build()
        .toUri();

    MemberDTO memberDTO = new MemberDTO();
    memberDTO.setName("adam");
    memberDTO.setEmail("itstudy@kakao.com");
    memberDTO.setOrganization("adamsoft");

    RequestEntity<MemberDTO> requestEntity = RequestEntity
        .post(uri)
        .header("my-header", "adamsoft API")
        .body(memberDTO);
```

Server Communication

- RestTemplate
  - ✓ RestTemplate를 이용한 ServerBox 활용
    - Service 클래스를 생성하고 메서드 작성 – service.RestTemplateService

```
        RestTemplate restTemplate = new RestTemplate();
        ResponseEntity<MemberDTO> responseEntity =
    restTemplate.exchange(requestEntity,
            MemberDTO.class);

        return responseEntity;
    }
}
```

Server Communication

- RestTemplate
  - ✓ RestTemplate를 이용한 ServerBox 활용
    - Controller클래스를 생성하고 메서드 작성 – controller.RestTemplateController

```
@RestController
@RequestMapping("/rest-template")
public class RestTemplateController {

    private final RestTemplateService restTemplateService;

    public RestTemplateController(RestTemplateService restTemplateService) {
        this.restTemplateService = restTemplateService;
    }

    @GetMapping
    public String getName() {
        return restTemplateService.getName();
    }

    @GetMapping("/path-variable")
    public String getNameWithPathVariable(){
        return restTemplateService.getNameWithPathVariable();
    }
```

Server Communication

- RestTemplate
  - ✓ RestTemplate를 이용한 ServerBox 활용
    - Controller클래스를 생성하고 메서드 작성 – controller.RestTemplateController

```
    @GetMapping("/parameter")
    public String getNameWithParameter(){
        return restTemplateService.getNameWithParameter();
    }

    @PostMapping
    public ResponseEntity<MemberDTO> postDto(){
        return restTemplateService.postWithParamAndBody();
    }

    @PostMapping("/header")
    public ResponseEntity<MemberDTO> postWithHeader(){
        return restTemplateService.postWithHeader();
    }

}
```

Server Communication

- ❖ RestTemplate 빈 설정
  - ✓ RestTemplate Bean 설정
    - ● RestTemplate 의 Interceptor 클래스 생성 – interceptor.IdentityHeaderInterceptor

```
public class IdentityHeaderInterceptor implements ClientHttpRequestInterceptor {

    private static final String COMPONENT_HEADER_NAME = "X-COMPONENT-ID";
    private static final String COMPONENT_HEADER_VALUE = "ADAM-API";

    @Override
    public ClientHttpResponse intercept(HttpRequest request, byte[] body,
     ClientHttpRequestExecution execution) throws IOException {

        //X-COMPONENT-ID 헤더를 설정하지 않았으면 ADAM-API를 설정
        request.getHeaders().addIfAbsent(COMPONENT_HEADER_NAME,
     COMPONENT_HEADER_VALUE);
        return execution.execute(request, body);
    }
}
```

Server Communication

- RestTemplate 빈 설정
  - ✓ RestTemplate Bean 설정
    - RestTemplate 의 Config 클래스 생성 – config.RestTemplateConfig

```
@Configuration
public class RestTemplateConfig {

    @Bean
    public ClientHttpRequestFactory clientHttpRequestFactory() {
        SimpleClientHttpRequestFactory factory = new
     SimpleClientHttpRequestFactory();

        //대기 시간 과 읽는데 걸리는 시간의 최대값을 설정
        factory.setConnectTimeout(3000);
        factory.setReadTimeout(1000);
        factory.setBufferRequestBody(false);

        return factory;
    }
```

Server Communication

- ❖ RestTemplate 빈 설정
  - ✓ RestTemplate Bean 설정
    - ● RestTemplate 의 Config 클래스 생성 – config.RestTemplateConfig

```
@Bean
public RestTemplate restTemplate(ClientHttpRequestFactory
 clientHttpRequestFactory) {

    RestTemplate restTemplate = new RestTemplate(clientHttpRequestFactory);

    //인터셉터 설정
    restTemplate.getInterceptors().add(new IdentityHeaderInterceptor());
    //에러가 발생했을 때 호출될 핸들러 설정
    restTemplate.setErrorHandler(new DefaultResponseErrorHandler());

    return restTemplate;
  }
}
```

Server Communication

- RestTemplate: keep-alive 와 Connection Pool 설정
  - ✓ 의존성 설정

```
implementation 'org.apache.httpcomponents.client5:httpclient5:5.1.3'
```

Server Communication

- RestTemplate: keep-alive 와 Connection Pool 설정
  - ✓ 환경 설정 파일 추가 – config.PoollingRestTemplateConfig

```
@Configuration
public class PoolingRestTemplateConfig {
    @Bean
    public PoolingHttpClientConnectionManager poolingHttpClientConnectionManager()
    {
        PoolingHttpClientConnectionManager manager = new
     PoolingHttpClientConnectionManager();
        manager.setMaxTotal(100);
        manager.setDefaultMaxPerRoute(5);

        HttpHost httpHost = new HttpHost("http://localhost", 9000);
        manager.setMaxPerRoute(new HttpRoute(httpHost), 10);

        return manager;
    }
```

Server Communication

- RestTemplate: keep-alive 와 Connection Pool 설정
  - ✓ 환경 설정 파일 추가 – config.PoollingRestTemplateConfig

```
@Bean
public RequestConfig requestConfig() {
    return RequestConfig.custom()
            .setConnectionRequestTimeout(3000, TimeUnit.MILLISECONDS)
            .setConnectTimeout(3000, TimeUnit.MILLISECONDS)
            .build();
}

@Bean
public CloseableHttpClient httpClient() {
    return HttpClientBuilder.create()
            .setConnectionManager(poolingHttpClientConnectionManager())
            .setDefaultRequestConfig(requestConfig())
            .build();
}
```

Server Communication

- RestTemplate: keep-alive 와 Connection Pool 설정
  - ✓ 환경 설정 파일 추가 – config.PoollingRestTemplateConfig

```
@Bean
public RestTemplate poolingRestTemplate() {
    HttpComponentsClientHttpRequestFactory requestFactory = new
 HttpComponentsClientHttpRequestFactory();
    requestFactory.setHttpClient(httpClient());
    return new RestTemplate(requestFactory);
}
}
```

Server Communication

- RestTemplate keep-alive 와 Connection Pool 설정
  - ✓ 의존성 설정
    - RestTemplate 의 Config 클래스 생성 – config.RestTemplateConfig 의 @Configuration 주석

Server Communication

- WebClient
  - ✓ Spring WebFlux는 HTTP 요청을 수행하는 클라이언트로 WebClient를 제공
  - ✓ WebClient는 리액터(Reactor) 기반으로 동작하는 API – Netty Client 의 Worker Thread로 수행
  - ✓ 리액터 기반이므로 스레드와 동시성 문제를 벗어나 비동기 형식으로 사용할 수 있음
  - ✓ WebClient의 특징
    - 논블로킹(Non-Blocking) I/O를 지원
    - 리액티브 스트림(ReactiveStreams)의 백프레셔(BackPressure - 데이터 생산과 소비가 불균형적일 때 일어나는 현상)를 지원
    - 적은 하드웨어 리소스로 동시성을 지원
    - 함수형 API를 지원
    - 동기, 비동기 상호 작용을 지원
    - 스트리밍을 지원
  - ✓ WebClient의 생성
    - create() 메서드 이용
    - builder() 메서드 이용

Server Communication

- WebClient
  - ✓ WebClient를 이용한 구현
    - WebClient 서비스 클래스 생성 – service.WebClientService

```
@Service
public class WebClientService {

    public String getName() {
        WebClient webClient = WebClient.builder()
            .baseUrl("http://localhost:9000")
            .defaultHeader(HttpHeaders.CONTENT_TYPE,
MediaType.APPLICATION_JSON_VALUE)
            .build();

        return webClient.get()
            .uri("/api/v1/crud-api")
            .retrieve()
            .bodyToMono(String.class)
            .block();
    }
```

Server Communication

- WebClient
  - ✓ WebClient를 이용한 구현
    - WebClient 서비스 클래스 생성 – service.WebClientService

```
public String getNameWithPathVariable() {
    WebClient webClient = WebClient.create("http://localhost:9000");

    ResponseEntity<String> responseEntity = webClient.get()
        .uri(uriBuilder -> uriBuilder.path("/api/v1/crud-api/{name}")
            .build("adam"))
        .retrieve().toEntity(String.class).block();

    ResponseEntity<String> responseEntity1 = webClient.get()
        .uri("/api/v1/crud-api/{name}", "adam")
        .retrieve()
        .toEntity(String.class)
        .block();

    return responseEntity.getBody();
}
```

Server Communication

- ❖ WebClient
  - ✓ WebClient를 이용한 구현
    - ● WebClient 서비스 클래스 생성 – service.WebClientService

```
public String getNameWithParameter() {
    WebClient webClient = WebClient.create("http://localhost:9000");

    return webClient.get().uri(uriBuilder -> uriBuilder.path("/api/v1/crud-api")
            .queryParam("name", "adam")
            .build())
        .exchangeToMono(clientResponse -> {
            if (clientResponse.statusCode().equals(HttpStatus.OK)) {
                return clientResponse.bodyToMono(String.class);
            } else {
                return clientResponse.createException().flatMap(Mono::error);
            }
        })
        .block();
}
```

Server Communication

- WebClient
  - ✓ WebClient를 이용한 구현
    - WebClient 서비스 클래스 생성 – service.WebClientService

```
public ResponseEntity<MemberDTO> postWithParamAndBody() {
        WebClient webClient = WebClient.builder()
            .baseUrl("http://localhost:9000")
            .defaultHeader(HttpHeaders.CONTENT_TYPE,
MediaType.APPLICATION_JSON_VALUE)
            .build();
        MemberDTO memberDTO = new MemberDTO();
        memberDTO.setName("adam!!");
        memberDTO.setEmail("itstudy@kakao.com");
        memberDTO.setOrganization("adamsoft");
        return webClient.post().uri(uriBuilder -> uriBuilder.path("/api/v1/crud-api")
                .queryParam("name", "adam")
                .queryParam("email", "itstudy@kakao.com")
                .queryParam("organization", "adamsoft")
                .build())
            .bodyValue(memberDTO)
            .retrieve()
            .toEntity(MemberDTO.class)
            .block();
    }
```

Server Communication

- WebClient
  - ✓ WebClient를 이용한 구현
    - WebClient 서비스 클래스 생성 – service.WebClientService

```
public ResponseEntity<MemberDTO> postWithHeader() {
    WebClient webClient = WebClient.builder()
        .baseUrl("http://localhost:9000")
        .defaultHeader(HttpHeaders.CONTENT_TYPE,
MediaType.APPLICATION_JSON_VALUE)
        .build();
    MemberDTO memberDTO = new MemberDTO();
    memberDTO.setName("flature!!");
    memberDTO.setEmail("flature@gmail.com");
    memberDTO.setOrganization("Around Hub Studio");
    return webClient
        .post()
        .uri(uriBuilder -> uriBuilder.path("/api/v1/crud-api/add-header")
            .build())
        .bodyValue(memberDTO)
        .header("my-header", "Wikibooks API")
        .retrieve()
        .toEntity(MemberDTO.class)
        .block();
}
```

Server Communication

- WebClient
  - ✓ WebClient를 이용한 구현
    - WebClient 서비스 클래스 생성 – service.WebClientService

```
public void cloneWebClient() {
    WebClient webClient = WebClient.create("http://localhost:9000");

    WebClient clone = webClient.mutate().build();
}
}
```

Server Communication

- WebClient
  - ✓ WebClient를 이용한 구현
    - WebClient 컨트롤러 클래스 생성 – controller.WebClientController

```
@RestController
@RequestMapping("/web-client")
public class WebClientController {

    private WebClientService webClientService;

    @Autowired
    public WebClientController(WebClientService webClientService) {
        this.webClientService = webClientService;
    }

    @GetMapping
    public String getName() {
        return webClientService.getName();
    }

    @GetMapping("/path-variable")
    public String getNameWithPathVariable() {
        return webClientService.getNameWithPathVariable();
    }
```

Server Communication

- WebClient
  - ✓ WebClient를 이용한 구현
    - WebClient 컨트롤러 클래스 생성 – controller.WebClientController

```
    @GetMapping("/parameter")
    public String getNameWithParameter() {
        return webClientService.getNameWithParameter();
    }

    @PostMapping
    public ResponseEntity<MemberDTO> postDto() {
        return webClientService.postWithParamAndBody();
    }

    @PostMapping("/header")
    public ResponseEntity<MemberDTO> postWithHeader(){
        return webClientService.postWithHeader();
    }
}
```

Server Communication

- WebClient 빈 설정
  - ✓ config.WebClientConfig

```
@Slf4j
@Component
public class WebClientConfig {

    @Bean
    public WebClient webClient() {

        HttpClient httpClient = HttpClient.create()
                .tcpConfiguration(tcpClient -> tcpClient
                        .option(ChannelOption.CONNECT_TIMEOUT_MILLIS, 10000)
                        .doOnConnected(conn -> conn
                                .addHandlerLast(new ReadTimeoutHandler(10))
                                .addHandlerLast(new WriteTimeoutHandler(10))
                        )
                );

        ClientHttpConnector connector = new
    ReactorClientHttpConnector(httpClient.wiretap(true));
```

Server Communication

- WebClient 빈 설정
  - ✓ config.WebClientConfig

```
        return WebClient.builder()
                .clientConnector(connector)
                .baseUrl("http://localhost:9000")
                .defaultHeaders(httpHeaders -> {
                    httpHeaders.add(HttpHeaders.CONTENT_TYPE,
    MediaType.APPLICATION_JSON_VALUE);
                    httpHeaders.add(HttpHeaders.ACCEPT,
    MediaType.APPLICATION_JSON_VALUE);
                })
                .build();
    }
}
```

Server Communication

- HttpInterface
  - ✓ Spring6에서 지원하는 @Httpexchange 어노테이션이 달린 메서드를 가진 자바 인터페이스를 HTTP 서비스로 만들어주는 기능
  - ✓ 인터페이스를 구현하는 프록시를 통해 HTTP 요청을 수행

Server Communication

- HttpInterface
  - ✓ HttpInterface를 사용하기 위한 인터페이스 생성 – service.HttpInterfaceAPIService

```
public interface HttpInterfaceAPIService {
    @GetExchange("/api/v1/crud-api")
    String getName();

    @GetExchange("/api/v1/crud-api/{variable}")
    String getNameWithPathVariable(@PathVariable String variable);

    @GetExchange("/api/v1/crud-api/param")
    String getNameWithParameter(@RequestParam("name") String name);

    @PostExchange("/api/v1/crud-api")
    ResponseEntity<MemberDTO> getMember(@RequestBody MemberDTO
     memberDTO,
                                        @RequestParam("email") String email,
                                        @RequestParam("name") String name,
                                        @RequestParam("organization") String organization);
    @PostExchange("/api/v1/crud-api/add-header")
    ResponseEntity<MemberDTO> postWithHeaderAndBody(@RequestBody
     MemberDTO memberDTO, @RequestHeader("my-header")String header);
}
```

Server Communication

- HttpInterface
  - ✓ HttpInterface Bean을 생성하기 위한 클래스 생성 – config.HttpConfiguration

```
@Configuration
public class HttpConfiguration {
    @Bean
    public WebClient client() {
        return WebClient.builder().baseUrl("http://localhost:9000").build();
    }

    @Bean
    public HttpServiceProxyFactory httpServiceProxyFactory(WebClient client) {
        return HttpServiceProxyFactory.builder(WebClientAdapter.forClient(client)).build();
    }

    @Bean
    public HttpInterfaceAPIService httpInterfaceAPIService(HttpServiceProxyFactory
     httpServiceProxyFactory) {
        return httpServiceProxyFactory.createClient(HttpInterfaceAPIService.class);
    }

}
```

Server Communication

- HttpInterface
  - ✓ Service 클래스 생성 – service.HttpInterfaceService

```
@Service
@RequiredArgsConstructor
public class HttpInterfaceService {
    private final HttpInterfaceAPIService httpInterfaceAPIService;

    public String getName() {
       return httpInterfaceAPIService.getName();
    }

    public String getNameWithPathVariable() {
        return httpInterfaceAPIService.getNameWithPathVariable("아담");
    }

    public String getNameWithParameter() {
        return httpInterfaceAPIService.getNameWithParameter("군계");
    }
```

Server Communication

- HttpInterface
  - ✓ Service 클래스 생성 – service.HttpInterfaceService

```
public ResponseEntity<MemberDTO> postWithParamAndBody() {

    MemberDTO memberDTO = new MemberDTO();
    memberDTO.setName("아담");
    memberDTO.setEmail("ggangpae1@gmail.com");
    memberDTO.setOrganization("adamsoft");

    //RestTemplate restTemplate = new RestTemplate();
    ResponseEntity<MemberDTO> responseEntity =
            httpInterfaceAPIService.getMember(
                    memberDTO, "itstudy@kakao.com", "군계", "free");

    return responseEntity;
}
```

Server Communication

- HttpInterface
  - ✓ Service 클래스 생성 – service.HttpInterfaceService

```
public ResponseEntity<MemberDTO> postWithHeaderAndBody() {

    MemberDTO memberDTO = new MemberDTO();
    memberDTO.setName("아담");
    memberDTO.setEmail("ggangpae1@gmail.com");
    memberDTO.setOrganization("adamsoft");

    //RestTemplate restTemplate = new RestTemplate();
    ResponseEntity<MemberDTO> responseEntity =
            httpInterfaceAPIService.postWithHeaderAndBody(
                    memberDTO, "header");

    return responseEntity;
  }
}
```

Server Communication

- HttpInterface
  - ✓ Controller 클래스 생성 – controller.HttpInterfaceController

```
@RestController
@RequestMapping("/httpinterface")
@RequiredArgsConstructor
public class HttpInterfaceController {
    private final HttpInterfaceService httpInterfaceService;

    @GetMapping
    public String getName() {
        return httpInterfaceService.getName();
    }

    @GetMapping("/pathvariable")
    public String getNameWithPathVariable() {
        return httpInterfaceService.getNameWithPathVariable();
    }

    @GetMapping("/param")
    public String getNameWithParameter() {
        return httpInterfaceService.getNameWithParameter();
    }
```

Server Communication

- HttpInterface
  - ✓ Controller 클래스 생성 – controller.HttpInterfaceController

```
@GetMapping("/postwithparamandbody")
public ResponseEntity<MemberDTO> getMember() {

    return httpInterfaceService.postWithParamAndBody();
}
@GetMapping("/postwithbodyandheader")
public ResponseEntity<MemberDTO> addHeader() {

    return httpInterfaceService.postWithHeaderAndBody();
}

}
```