Spring Boot Part4

스프링 RESTful API 서버 만들기

REST(Representational State Transfer)

Representational State Transfer의 줄임말로, 직역하면 '상태를 표현하는 전송법' 정도로 해석된다. 무슨말인지 도무지 감이 오질 않는다...

먼저, '전송법'이란 단어가 사용되었는데 도대체 무엇을, 어디로 전송하는 법을 말하는 걸까. 여기서 말하는 전송법은 '클라이언트가 서버에 요청을 전송하는 방법'이라 생각하면 된다. 클라이언트가 서버에 요청을 전송하는 것은 url을 통해 진행된다. 즉, 우리가 url에 입력하는 내용에 따라 서버로 요청이 전달되는 것이다. 여기까지 해석 후 다시 REST의 의미를 살펴보면 '상태를 표현하여 서버로 요청을 보내는 URL 작성법' 정도가 되겠다. 결국, REST란 클라이언트가 서버에 요청을 보낼 때 사용하는 URL의 작성 규칙이며, 이 규칙은 상태를 표현한다는 특징이 있다는 것이다. 이를 정리하면...

- REST의 URL 작성 규칙 : URL에 제어할 자원을 명시한다.
- 상태를 표현하는 특징 : 제어의 상태를 URL에서 표현하지 않고, GET, ~~PUT~~, PUT, DELETE 메서드를 표현한다.
  POST

아직 무슨말인지 감이 잘 안 올 수도 있다. 다음 슬라이드를 보자.

REST란 클라이언트가 서버로 요청을 보내는 URL 작성법이라 하였다. 그럼 REST의 사용 전 후 URL이 어떻게 달라지는지 예를 들어보겠다.

Non RESTful URL	RESTful API
localhost:8080/board/list	(GET) localhost:8080/boards
localhost:8080/board/read-all	(GET) localhost:8080/members
localhost:8080/board/get?no=1	(GET) localhost:8080/boards/5
localhost:8080/board/read?no=1	(POST) localhost:8080/boards
localhost:8080/board/update	(DELETE) localhost:8080/boards/5
localhost:8080/board/change	(~~UPDATE~~) PUT localhost:8080/boards/5

- 위의 표처럼 REST의 URL 작성법을 따르지 않으면 URL만 읽고 어떤 요청을 서버에게 전송하는 것인지 감이 오질 않는다. 모든 개발자가 본인만의 성향대로 URL을 작성하기 때문에 규칙성이 없기 때문이다. 또한 URL에 list, read, get, update 등의 단어를 사용하여 제어의 상태까지 표현해주고 있어, url이 길어진다.
- 반면, REST에서 권장하는 URL 작성법을 사용하면 URL만 봐도 무슨 요청인지 쉽게 유추가 가능하다. 또한, URL에 직접 제어의 상태를 작성하지 않기 때문에 URL이 훨씬 간결해진다. 제어의 상태는 GET(조회), POST(삽입), PUT(수정), DELETE(삭제) 등이 있다.

그럼 REST가 무엇인지 정리하겠다.

REST란 클라이언트가 서버로 요청을 보내는 URL의 작성 방법을 규격화한 것이다. 또한, 제어의 상태를 URL에 표기하지 않고 GET, POST, PUT, DELETE 등의 메서드를 사용하며 별도로 표기하기 때문에 URL이 간결해지며 획일화된 요청 URL을 가질 수 있다.

그리고 서버에 보내는 요청 URL을 모두 REST 형식에 맞추어 개발한 프로젝트를 'RESTful 하다' 라고 표현한다.

추가적으로 REST에서 권장하는 URL 명명법을 알아보겠다. 지키지 않는다고 오류가 나진 않지만 가급적 지키도록 노록하자!

- URL은 제어할 자원을 명사로 표현하며, 복수형을 사용한다.

  ex> localhost:8080/member X, localhost:8080/members O
- URL은 소문자를 작성한다.

  ex> localhost:8080/Boards X, localhost:8080/boards O
- 복합어(합성어)의 경우 언더바(_) 대신 하이픈(-)을 사용한다.

  ex> localhost:8080/shop_members X, localhost:8080/shop-members O
- URL 마지막에 슬래쉬(/)를 포함하지 않는다.

  ex> localhost:8080/members/ X, localhost:8080/members O
- URL에 제어의 상태를 포함하지 않는다.

  ex> localhost:8080/members/insert X, localhost:8080/members O

API(Application Programming Interface)

Application Programming Interface의 줄임말로, 서로 다른 두 소프트웨어가 상호작용하기 위한 프로그래밍 기술을 뜻한다.

서로 다른 두 소프트웨어가 상호작용 한다는 것은, 두 소프트웨어가 데이터를 주고 받는 것이라 생각하면 된다.

결국, API는 서로 다른 두 시스템이 데이터를 주고 받기 위한 프로그래밍 기술이다.

실 생활을 예로 들면, 손님이 식당에 가서 알바생에게 주문을 한다. 알바생은 주문을 셰프에게 전달하고 셰프는 음식을 만든다. 만들어진 음식을 알바생이 손님에게 가져온다. 이때, 손님과 셰프는 서로 다른 시스템이고 알바생이 두 시스템을 상호작용할 수 있게 해주는 API인 것이다.

이러한 개념을 우리의 학습으로 가져와서 생각해보자. 우리 학습에서 서로 다른 두 시스템은 Spring으로 만드는 백엔드 프로젝트와 React로 만드는 프론트엔드 프로젝트이다. 그리고 우리는 이 둘의 데이터 전달을 위해 코드를 구현할 것이다. 이 코드가 API인 것이다.

RESTful API

RESTful API는 말 그래도 RESTful과 API의 개념을 합쳐 놓은 것이다.

API는 두 시스템이 데이터 전달을 할 수 있는 프로그래밍 기술(코드)이며, RESTful은 데이터 요청을 위한 url 작성에 대한 규칙이었다.

결국, RESTful API를 만든다는 것은, REST가 권장하는 URL 작성법을 지키며, 두 시스템의 데이터 전달 기능을 구현하는 것을 의미한다.

그럼 RESTful API 구현을 위해 Spring 프로젝트와 React 프로젝트에서 어떤 작업을 하면 될까.

솔직히 React 프로젝트에서는 할게 없다. Spring프로젝트만 RESTful하게 만들면, React 프로젝트에서는 axios로 데이터 요청만 하면 되기 때문에 신경 쓸 것이 없기 때문이다. 이제부터는 Spring 프로젝트에 RESTful API를 만드는 과정을 알아보자.

RESTful API 서버 만들기

1. 다른 시스템(React 프로젝트)에서 전송되는 요청을 받고, 받은 요청에 대한 응답을 제공하는 클래스를 생성한다.

   다른 시스템(혹은 클라이언트)에서 요청이 오면(요청은 url을 통해 온다고 학습하였다) 그 요청을 받을 클래스를 먼저 생성해야 한다. 이 클래스는 들어오는 요청이 무엇인지 판단하여, 요청에 따라 필요한 기능을 실행시켜준다. 실 생활로 말하면 관제탑과 같은 역할을 맡은 클래스라 보면 되겠다. 관제탑은 항시 운영되며, 비행기 조정사가 어떤 메세지를 보내면 관제탑이 그 메세지를 받아, 적절하게 대응해준다. 이러한 역할을 하는 클래스를 스프링에서는 컨트롤러(Controller)라 부른다. 다른 시스템(혹은 클라이언트)의 요청을 받은 컨트롤러 클래스는 다음과 같이 생성한다.

   1) 반드시 default 패키지 안에 클래스를 생성한다. 클래스명은 자유이나 통상적으로 Controller라는 키워드를 붙여준다.

   2) 생성한 클래스 위에 @Controller 혹은 @RestController 어노테이션을 붙여준다. 끝!

   결국 컨트롤러 클래스를 만드는 다른 방식이 있는 것이 아니다. 우리가 만드는 여느 클래스와 같이 클래스를 만들어주고, 어노테이션만 붙여주면 끝인 것이다. 두 어노테이션이 하는 역할은 다음과 같다.

   - @Controller : 클래스의 객체를 생성해주며, 컨트롤러 역할을 부여. react를 사용하지 않는 웹 개발에 사용.
   - @RestContoller : 클래스의 객체를 생성해주며, 컨트럴러 역할을 부여. 추가적으로 REST 메서드를 활용하여 요청을 받을 수 있음.

     REST 메서드란 제어의 상태를 표현하는 GET, POST, PUT, DELETE 등을 말 함.

지금까지 설명한 요청을 받을 수 있는 컨트롤러 역할의 클래스를 코드로 보면 다음과 같다.

```
@RestController
public class TestController {

}
```

이렇게 클래스 위에 @RestController 어노테이션만 추가하면 해당 클래스는 요청을 받을 수 있는 컨트롤러가 된다. 그리고 이 클래스는 REST 방식으로 요청을 받을 수 있는 어노테이션을 사용할 수 있다. 어노테이션은 다음과 같다.

REST 요청을 받는 어노테이션	설 명
@GetMapping("요청 url")	REST의 GET 메서드로 요청을 받는다.
@PostMapping("요청 url")	REST의 POST 메서드로 요청을 받는다.
@PutMapping("요청 url")	REST의 PUT 메서드로 요청을 받는다.
@DeleteMapping("요청 url")	REST의 DELETE 메서드로 요청을 받는다.

아직 잘 이해가 안갈 수 있다. 용어가 아직 익숙치 않기 때문이다. 잠시 뒤 postman 프로그램을 설치하여 실행하는 것을 보면 감을 잡을 수 있을 것이다.

컨트롤러 클래스를 조금 더 작성해보겠다.

```java
@Slf4j
@RestController
public class TestController {

    @GetMapping("/members")
    public void getMemberList(){
      log.info("getMemberList 메서드 실행");
    }

    @GetMapping("/members/{memId}")
    public void getMember(){
        log.info("getMember 메서드 실행");
    }

    @PostMapping("/members")
    public void insertMember(){
        log.info("insertMember 메서드 실행");
    }
}
```

- 클래스 위의 @slf4j 어노테이션은 롬북 라이브러리가 제공하는 어노테이션이다. 해당 어노테이션을 사용하면 log.info() 메서드를 사용할 수 있으며, println() 출력문 대신 웹 개발에 사용한다.
- 클래스에는 3개의 메서드가 선언되어 있다. 그리고 각 메서드에는 @GetMapping(), @PostMapping() 어노테이션이 사용되었다. 이 어노테이션의 매개변수에는 문자열로 요청 url을 작성한다. @RestController 어노테이션이 붙은 클래스는 요청이 들어오면 이를 감지하여, 요청 url과 동일한 url을 매개변수로 갖는 어노테이션이 붙은 메서드를 실행시켜준다.
- GetMapping어노테이션은 REST의 GET 메서드 요청을 받을 때 사용하며, PostMapping 어노테이션은 REST의 POST 메서드 요청을 받을 때 사용한다.
- 계속 말하지만 요청이란 URL에 작성한 문자열이다. 그렇기 때문에 해당 기능을 실행하기 위해서는 웹 브라우저의 URL을 사용하여 요청을 보내면 된다. Spring 프로젝트의 내장 WAS인 톰캣은 8080 PORT에서 실행중이다. 그렇기 때문에 URL에 localhost:8080/members 라고 입력하면, 해당 요청을 받은 컨트롤러 클래스의 메서드가 실행 될 것이다.
- 웹 브라우저에서 url을 직접 입력하여 요청을 보내면 무조건 REST의 GET 메서드 방식으로 요청을 하게 된다. 그렇기 때문에 PostMapping 어노테이션이 붙은 메서드는 웹 브라우저에서 url을 입력하여 요청을 아무리 보내도 실행되지 않을 것이다. 해결을 위해선 React로 직접 클라이언트를 만들어야 겠지만, 배보다 배꼽이 크므로 postman 프로그램을 설치하여 실습을 이어가겠다.

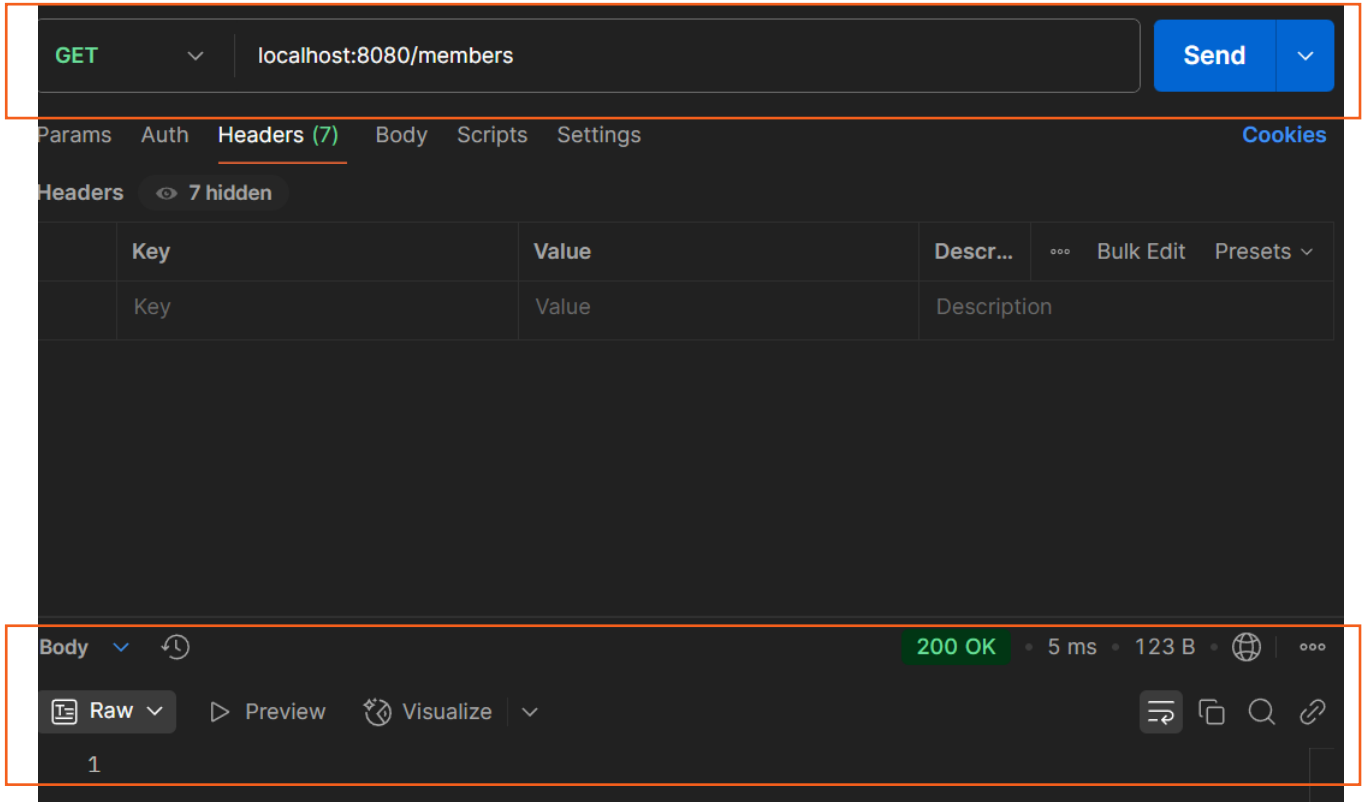
백엔드 서버를 만든 후, 잘 구현되었는지 확인을 위해서는 백엔드 서버를 실행시키는 프론트엔드도 만들어야 한다.

하지만, 백엔드 서버를 잘 만들었는지 테스트를 위해, 프론트엔드를 만드는 것은 비효율적이다.

더욱이 실무에서는 백엔드만 다루는 개발자들도 있는데, 이 개발자들은 테스트 하기가 난감할 것이다.

postman은 이러한 문제점을 해결할 수 있는 소프트웨이다. postman은 가상의 프론트엔드를 제공하여 백엔드와 통신할 수 있는 기능을 제공한다.

다음은 postman의 실행 화면이다. (postman의 내용에 대해서는 강의에서는 사용하겠지만, 본 강의노트에서는 더 이상 기재하지 않는다)



이 곳에서 요청 url과 REST의 메서드를 결정한다.

이 곳은 요청에 대한 응답 결과를 보여준다.

200번은 정상적으로 요청과 응답이 진행된 것이다.

만약 메서드에서 리턴되는 데이터가 있다면 리턴된 데이터를 보여준다

```java
@Slf4j
@RestController
public class TestController {

    @GetMapping("/members")
    public void getMemberList(){
      log.info("getMemberList 메서드 실행");
    }

    @GetMapping("/members/{memId}")
    public void getMember(){
        log.info("getMember 메서드 실행");
    }

    @PostMapping("/members")
    public void insertMember(){
        log.info("insertMember 메서드 실행");
    }
}
```

- postman을 이용하여 PostMapping, GetMapping에 맞게 요청을 보내고 실행 결과를 테스트 해보았을 것이다. 다시 언급하지만 요청 URL은 단순히 글자 맞추기니 너무 어렵게 생각하지도, 여려워 하지도 말길 당부한다!

  (REACT에서도 axios의 글자만 잘 맞춰주면 되요... ㅜ)
- 클래스에서 선언한 getMember() 메서드는 어떤 요청을 보내면 실행이 될지 알아보자. 해당 메서드 위에는 @GetMapping("/members/{memId}")라고 작성되어 있다. 해당 메서드를 실행시킬 수 있는 url은 다음과 같다.

  localhost:8080/members/1

  localhost:8080/members/5

  localhost:8080/members/java

  결국 {memId}에는 어떠한 정보가 들어와도 실행된다. {}는 url 에서 넘어오는 데이터를 받을 때 사용하는 문법이다.

  /members/1 이라는 요청이 들어오면 memId라는 변수에 1을 받겠다는 의미다.

  /members/java 라는 요청이 들어오면 memId라는 변수에 java를 받겠다는 의미다.

  이때 memId는 변수라고 생각하면 된다. 그렇기에 memId라는 글자는 마음대로 변경할 수 있다.

```java
@GetMapping("/members/{memId}")
public void getMember(@PathVariable("memId") String id){
    log.info("getMember 메서드 실행 : id = " + id);
}
```

- 위의 메서드는 url을 통해 전달되는 데이터를 받는 코드를 작성한 것이다.
- url을 통해 전달되는 데이터를 받을 때만 메서드의 매개변수에 @PathVariable 어노테이션을 사용한다.
- @PathVariable 어노테이션의 소괄호 안에는 url로 넘어오는 데이터의 이름을 작성한다.

  @PathVariable(“memId”) 코드는 memId라는 이름으로 url로 넘어오는 데이터를 받겠다는 의미이다.
- @PathVariable 어노테이션 뒤에는 넘어오는 데이터를 저장한 변수를 선언해준다.

  (@PathVariable(“memId”) String id)를 전체적으로 해석하면

  url을 통해 memId라는 이름으로 넘어오는 데이터를 id라는 변수로 받겠다. 라고 해석하면 된다.

  만약, postman에서 localhost:8080/members/java 라고 url을 요청하면

  getMember 메서드가 실행되면서 memId라는 이름으로 넘어온 “java”라는 데이터를 id라는 변수로 받겠다. 라고 해석된다.

@PathVariable 어노테이션을 사용하면 하나의 데이터만 받을 수 있는건 아니다. 다음은 2개의 데이터를 받는 예시이다.

```
@GetMapping("/boards/{name}/{age}")
public void getBoard(@PathVariable("name") String n, @PathVariable("age") int a){
    log.info("getBoard 메서드 실행");
    log.info("name = " + n);
    log.info("age = " + a);
}
```

postman에서 localhost:8080/boards/kim/20 을 입력하여 url을 요청하면

n과 a 변수에 각각 “kim”과 200이라는 데이터가 들어온 것을 확인 할 수 있다.

@PostMapping이 사용된 메서드에는 더 많은 데이터를 전달 받을 수 있다.

REST의 POST 메서드는 데이터를 삽입하는 상태를 표현한 것이며, 데이터를 삽입하기 위해선 실제 추가할 데이터가 많이 전달되어야 하기 때문이다.

다음은 @PostMapping이 사용된 메서드에서 데이터를 받는 방식이다.

우선 postman에서 3개의 데이터를 전달한다. (react에서 객체 형식으로 데이터를 전달 한 것 기억하시죠?)

이렇게 전달되는 3개의 데이터를 자바에서 받기 위해서는 먼저 전달되는 데이터의 키와 동일한 변수를 갖는 클래스를 생성해줘야 한다. 그리고 롬북 라이브러리를 사용해여 getter, setter, toString을 생성하였다.

```
@Setter
@Getter
@ToString
public class BoardDTO {
    private int boardNum;
    private String title;
    private String writer;
}
```

요청에 의해 실행되는 메서드에서 데이터를 받기 위해서는 매개변수 자리에 전달되는 데이터의 key와 동일한 이름의 변수를 갖는 클래스를 매개변수로 지정해주면 된다.

이때 @RequestBody 어노테이션은 요청 url의 body 부분을 가져오겠다는 말이다.

이렇게 매개변수를 작성하면 key와 변수의 값을 매칭하여 값을 받아온다.

```
@PostMapping("/boards")
public void insertBoard(@RequestBody BoardDTO boardDTO){
    log.info("insertBoard 메서드 실행");
    log.info(boardDTO.toString());
}
```

url은 크게 header부분과 body 부분으로 이루어져 있습니다. url을 통해 전달되는 데이터는 body영역에 담겨 전달되는 구조이기 때문에 @RequestBody를 사용합니다.

지금까지 @GetMapping과 @PostMapping의 사용법에 대해 알아보았다.

@DeleteMapping과 @PutMapping은 각각 GetMapping과 PostMapping의 사용법과 동일하게 때문에 강의 시간의 실습으로 대체하겠다.

- Delete 예시)

```
@DeleteMapping("/boards/{boardNum}")
public void deleteBoard(@PathVariable("boardNum") int boardNum){
  log.info("기능 실행")
}
```

- Put 예시)

```
@PutMapping("/boards/{boardNum}")
  public void putBoard(@PathVariable("boardNum") int boardNum, @RequestBody BoardDTO boardDTO){
    log.info("" + boardNum);
    log.info(boardDTO.toString());
  }
```

컨트롤러 클래스에 @RequestMapping 어노테이션을 사용하면 보다 간단하게 url매핑을 사용할 수 있다. (이 방법을 추천함)

```java
@RestController
public class BoardController {
    @GetMapping("/boards")
    public void getList(){
        System.out.println("getList() 메서드 호출");
    }

    @GetMapping("/boards/{boardNum}")
    public void getBoard(@PathVariable("boardNum") int boardNum){
        System.out.println("getBoard() 메서드 호출");
        System.out.println("boardNum = " + boardNum);
    }

    @PostMapping("/boards")
    public void insertBoard(@RequestBody BoardDTO boardDTO){
        System.out.println("insertBoard() 메서드 호출");
        System.out.println(boardDTO);
    }

    @PutMapping("/boards/{boardNum}")
    public void updateBoard(@RequestBody BoardDTO boardDTO, @PathVariable("boardNum") int boardNum){
        System.out.println("updateBoard 메서드 호출");
        System.out.println("boardNum = " + boardNum);
        System.out.println(boardDTO);
    }
}
```

```java
@RestController
@RequestMapping("/boards")
public class BoardController {
    @GetMapping("")
    public void getList(){
        System.out.println("getList() 메서드 호출");
    }

    @GetMapping("/{boardNum}")
    public void getBoard(@PathVariable("boardNum") int boardNum){
        System.out.println("getBoard() 메서드 호출");
        System.out.println("boardNum = " + boardNum);
    }

    @PostMapping("")
    public void insertBoard(@RequestBody BoardDTO boardDTO){
        System.out.println("insertBoard() 메서드 호출");
        System.out.println(boardDTO);
    }

    @PutMapping("/{boardNum}")
    public void updateBoard(@RequestBody BoardDTO boardDTO, @PathVariable("boardNum") int boardNum){
        System.out.println("updateBoard 메서드 호출");
        System.out.println("boardNum = " + boardNum);
        System.out.println(boardDTO);
    }
}
```

여기서 잠깐!

이전 슬라이드에서 리액트에서 자바로 데이터가 전달될 때 자바스크립트 객체 형식으로 데이터가 전달된다고 표현했습니다. 그리고 react에서 우리가 axios를 사용하며 자바에서 리턴되는 데이터를 리액트에서 받을 때도 자바스크립트의 객체 형식으로 받는다고 배웠습니다. 하지만 이 말은 틀린 말입니다.

즉, 리액트에서 자바로 전달되는 데이터의 형태와 자바에서 리액트로 전달되는 데이터의 형태 모두 자바스크립트의 객체 형식이 아니라는 말입니다.

자바와 리액트(자바스크립트)가 데이터를 주고 받을 때는 JSON 형태의 데이터를 서로 주고 받습니다. JSON이 무엇인지 말하기 전에, 자바와 리액트(자바스크립트)가 애초가 자바스크립트의 객체 형태로 데이터를 주고 받을 수 없다는 것을 알아야 합니다. 이유는 간단합니다. 자바와 리액트(자바스크립트)는 다른 언어이기 때문입니다. 우리가 사과를 미국 사람에게 그대로 말하면 미국 사람이 알아 듣나요? 아닙니다. 미국사람은 사과라는 단어를 모르기 때문입니다. 프로그래밍 언어도 마찬가지입니다. 애초에 자바스크립트의 객체라는 데이터를 자바는 알아들을 수 없습니다. 이러한 언어 차이로 발생하는 문제는 실생활에서 번역기를 사용하여 해결하고 있습니다. 더 좋은 해결책은 뭘까요? 바로 전세계 사람들의 공용 언어를 만들고, 그 언어를 사용하는 것입니다. 하지만 여러 문제로 인해 현실 가능성이 매우 낮죠. 그렇지만, 프로그래밍 언어의 세계에서는 공용 언어를 만드는 것이 가능했습니다. 정확히는 공용언어라기 보다는 공용으로 사용 가능한 데이터의 형태를 만들었다가 맞는 표현이겠네요. 서로 다른 프로그래밍 언어 모두가 사용 가능한 데이터의 형태를 바로 JSON이라 합니다.

JSON(Javascript Object Notation)

JSON은 '말 그대로 자바스크립트 객체 표기법'입니다. JSON은 자바스크립트의 객체는 아니지만, 자바스크립트 객체를 베이스로 만들어진 것은 사실입니다. 그럼 자바스크립트 객체와 JSON은 무슨 차이냐면 JSON은 자바스크립트 객체를 문자열화 시킨 데이터입니다. 결국 JSON은 문자열 데이터입니다. 문자열 데이터는 대부분의 언어가 가지고 있기 때문에 자바와 리액트는 문자열 데이터는 주고 받을 수 있습니다. 그렇기에 자바스크립트의 객체를 문자열처럼 바꾸어, 문자열 데이터를 주고 받는 겁니다. 이렇게 설명하면 애초에 문자 형태로 데이터를 주고 받을 거라면, 기존의 문자열 데이터를 사용하면 되지, 왜 JSON이라는 어려운 문자열 데이터를 새로 만드는지 의문이 생길 수도 있습니다. 이에 대해서는 강의 시간에 추가 설명하겠습니다. 결론은, 'JSON은 자바와 자바스크립트가 서로 데이터를 주고 받을 수 있게, 자바스크립트 객체를 베이스로 만든 문자 데이터다' 라는 것만 기억해주세요!