JSON 개요

JSON

- **JSON (JavaSCript Object Notation)**

키-값 쌍으로 이루어져 데이터를 표현하는 포맷 중의 하나로써 비동기 통신 또는 API 통신 등 다양한 분야에서 널리 사용됨. 경량 데이터로써 가볍고 사람이 읽기가 쉬움

* 문자열을 JSON 데이터 포맷으로 사용하는 것

- **JSON 형식**

```
{
 "name": "Jaeseop",
 "age": 21,
 "city": "Incheon",
  "interests" : ["music", "sports", "reading"]
}
```

키 ← "city": "Incheon", → 값

JSON

- **JSON 외의 다른 포맷 예시 - XML**

```
<person>
  <name>Jaeseop</name>
  <age>21</age>
  <city>Incheon</city>
  <interests>
    <interest>music</interest>
    <interest>sports</interest>
    <interest>reading</interest>
  </interests>
</person>
```

JSON

- **JSON 외의 다른 포맷 예시 – YAML**

```
name: Jaeseop
age: 21
city: Incheon
interests:
  - music
  - sports
  - reading
```

JSON

- **JSON 외의 다른 포맷 예시 – CSV**

```
name, age, city, interests
jaeseop, 21, Incheon, music, sports, reading
```

JSON 파싱과 생성
(Java, JavaScript)

JSON 파싱과 생성

- **JavaScript에서의 JSON 파싱 및 생성**

1. JSON.stringify : JavScript 객체를 JSON 문자열로 변환
2. JSON.parse : JSON 문자열을 JavaScript 객체로 변환

```
<script>
  document.addEventListener("DOMContentLoaded", function(){

    // JavaSCript 객체
    let student = {
      name: 'Jaeseop',
      age: 21,
      isAdmin: false,
      courses: ['html', 'css', 'js'],
      wife: null
    };

    // JSON stringify : JavaScript 객체를 JSON 문자열로 변환
    console.log(typeof student);
    let student_json = JSON.stringify(student);
    console.log(student_json);
    console.log(typeof student_json);


    // JSON parse : JSON 문자열을 JavaScript 객체로 변환
    let student_parse = JSON.parse(student_json);
    console.log(student_parse);
    console.log(typeof student_parse);
  });
</script>
```

JSON 파싱과 생성

- **Java에서의 JSON 파싱 및 생성**

gson을 사용하여 JSON 파싱 및 생성하기 위해 pom.xml에 dependency 추가 후
프로젝트 우클릭 → MAVEN → Update Project

```
<dependency>
  <groupId>com.google.code.gson</groupId>
  <artifactId>gson</artifactId>
  <version>2.8.8</version> <!-- Gson 버전 -->
</dependency>
```

JSON 파싱과 생성

- **Java에서의 JSON 파싱 및 생성**

테스트 시 객체 생성하여 확인하기 위한 DTO 생성

```java
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class Data {
	private String name;
	private int age;
}
```

JSON 파싱과 생성

- **Java에서의 JSON 파싱 및 생성**

1. gson.toJson : 객체 → JSON 문자열 변환
2. gson.fromJson : JSON 문자열 → 객체 변환

```java
// Java 객체 생성
Data newData = new Data("Jane Smith", 25);
// Gson 객체 생성
Gson gson = new Gson();
// JSON 생성 (객체 -> JSON 문자열로 변환)
String jsonString = gson.toJson(newData);
// 결과 출력
System.out.println(jsonString);


// JSON 파싱 (JSON 문자열 -> 객체로 변환)
Data data = gson.fromJson(jsonString, Data.class);
// 결과 출력
System.out.println(data);
System.out.println(data.getName());
```

REST API

REST API

- **REST(Repersentational State Transfer)**

웹 서비스와 클라이언트 간의 통신을 위한 아키텍처 스타일

URI를 통해 고유한 식별자를 표현하고 이런 리소스를 조작하여 상호작용하여 필요한 데이터를 조회 하거나 수정, 생성 등을 수행할 수 있음

- **API(Application Programming Interface)**

프로그램들간의 상호작용을 도와주는 매개체

손님(프로그램) ↔ 점원(API) ↔ 요리사(프로그램)

REST API

- **REST 특징**

1. Uniform : URI로 지정한 리소스에 대해 어떠한 조작을 하는지 명시
2. Statelees : 클라이언트와 서버 간의 연결 상태를 유지하지 않음. 세션 또는 쿠기 정보를 별도로 저장하고 관리하지 않으며, REST API는 단순히 들어오는 요청만을 처리
3. Cacheable : HTTP의 캐싱 기능 사용 가능
4. Self-descriptiveness : REST API의 메시지만을 보고 쉽게 이해할 수 있는 구조
5. Client – Server 구조 : 클라이언트와 서버 간의 역할 명확히 분리
   → 서버 : 자원이 있는 곳
   → 클라이언트 : 자원을 요청하는 곳
6. Layered System : REST 서버는 순수 비즈니스 로직을 수행하고, 다중 계층으로 구성될 수 있으며 보안, 로드밸런싱, 암호화 계층 등을 추가하여 구조상의 유연함을 가질 수 있음

REST API

- **CRUD**

REST API에서의 CRUD는 HTTP Method를 통해 정해짐

1. GET : 리소스 조회
2. POST : 리소스 생성
3. PUT : 리소스 수정
4. DELTE : 리소스 삭제

REST API

- **REST API 중심 규칙**

URI는 리소스 자원을 표현해야 함

/member/list.do?idx=12
/member/select/12

잘못된 예시

/member/12

올바른 예시

REST API

- **잘못된 예시인 이유**

REST API는 HTTP Method를 통해 조회, 생성, 수정, 삭제를 표현하기 때문에 URI로 표현할 필요 없음

GET /member/12 → 리소스 조회
POST /member/12 → 리소스 생성
PUT /member/12 → 리소스 수정
DELETE /member/12 → 리소스 삭제

RESTful API

REST API

- **RESTFul API**

위에 설명된 REST API를 활용하여 웹 서비스를 구현하게 되면 'RESTful' 하다고 함

REST 특징과 요구사항을 잘 살려 구현된 서비스 → RESTful

- **RESTFul 하지 못한 경우**

대표적으로는 REST API 중심 규칙을 무시하는 경우가 많음

1. CRUD 기능을 특정 메소드로 모두 처리하는 경우
2. 리소스와 id 외의 정보가 들어가는 경우
3. CRUD 기능 마다 리소스가 모두 다른 경우
4. 기타 등등

간단한 RESTFul API 구현

- 인터셉터 주석

LoginInterceptor 및 AccessInterceptor가 리다이렉트 시키므로 주석처리 후 실습 진행

```
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, O
//          HttpSession session = request.getSession();
//          Integer memberIdx = (Integer) session.getAttribute("memberIdx");
//
//          if (memberIdx == null) {
//              // 로그인되지 않은 사용자이므로 다른 페이지로 리다이렉트
//              response.sendRedirect(request.getContextPath() + "/member/redirect.do");
//
//              return false;
//          }

        return true;
    }
```

```
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse respc
//          String referer = request.getHeader("referer");
//            String requestURI = request.getRequestURI();
//
//            String serverAddress = request.getRequestURL().toString();
//            String localServerAddress = serverAddress.replace(requestURI, "");
//
//            if (requestURI.equals("/board/detail.do") && (referer == null || !refere
//                response.sendRedirect("/common/errorPage");
//                return false;
//            }

        return true;
    }
```

간단한 RESTFul API 구현

- **MemberRestAPI**

컨트롤러 아래와 같이 생성
* @RestController → @Controller + @ResponseBody

```java
@RestController
@RequestMapping("/api/member")
public class MemberRestAPI {

}
```

간단한 RESTFul API 구현

- **GET 구현**

ResponseEntity
→ HTTP 응답에 보낼 데이터들을 담은 객체
→ HttpStatus, HttpHeaders, HttpBody를 포함하여 응답 가능

```java
@GetMapping("/{id}")
public ResponseEntity findMemberByName
(@PathVariable("id") String id) {
    // 원래는 이 부분에 123번 회원에 대한 SELECT 로직이 있어야 함
    // 간단하게 실습하는것이므로 임의 데이터 직접 작성
    Member m = new Member();
    m.setMemberName("김재섭");
    return new ResponseEntity(m, HttpStatus.OK);
}
```

```
C:₩Users₩user>
C:₩Users₩user>curl -X GET http://localhost/api/member/123
{"memberIdx":0,"memberName":"김재섭"}
C:₩Users₩user>
```