

06. 스프링부트

컴퓨터공학부 201911228 홍지우

01. Post Basic[RestController, PostMapping, RequestBody]

@RequestBody

HTTP 요청의 body 내용을 자바 객체로 매핑하는 역할

@PostMapping

POST 통신을 할 때 사용

- Client에서 특정 String을 "body" 형태로 보내면 그 String을 리턴하는 POST API 생성
- PostMapping에서 API Endpoint는 "/api/v1/string"로 설정
- Controller에서 인자로 받을 변수 앞에 RequestBody 어노테이션을 추가
 - i.e `fun getString(@RequestBody str: String)` / `String getString(@RequestBody String str)`

POST http://localhost:8080/api/v1/string Send

Params Auth Headers (9) **Body** Pre-req. Tests Settings Cookies

raw Text

```
1 Hello, World!
```

```
@PostMapping("/v1/string")
public String postMethod(@RequestBody String str){
    return str;
}
```

Body 200 OK 168 ms 177 B Save Response

Pretty Raw Preview Visualize Text

```
1 Hello, World!
```

02. 본인 프로필 서버에서 출력

- 본인의 프로필을 JSON 형식으로 서버에 넘기면, 서버에서 객체로 변환한 뒤, 해당 객체를 서버에서 출력
- endpoint 정의는 "/api/v1/profile"

POST http://localhost:8080/api/v1/profile

Params Auth Headers (9) **Body** Pre-req. Tests Settings Cookies

raw JSON Beautify

```
1 {  
2   "age": 22,  
3   "name": "jiwoo"  
4 }
```

Body 200 OK 690 ms 189 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {  
2   "age": 22,  
3   "name": "jiwoo"  
4 }
```

```
public class User {  
    int age;  
    String name;  
  
    public User(int age, String name) {  
        this.age = age;  
        this.name = name;  
    }  
}
```

```
@PostMapping("/v1/profile")  
public User postMethod2(@RequestBody User user){  
    return user;  
}
```

03. 아주 간단한 계산기 만들기

- Controller - Service 두 클래스를 이용
- 엔드포인트는 총 4개로 구성. 두 개의 수를 입력으로 받으며, 연산을 하는 수에 해당하는 변수 이름은 "operandFirst", "operandSecond" 로 설정
- 입력 수는 모두 Signed Int로 가정
- 서비스에서 계산을 담당하는 함수 4개 생성. 결과는 "calculationResult" 라는 이름에 담아주고 반환

```
public class OperandRequest {  
    int operandFirst;  
    int operandSecond;  
  
    public OperandRequest(int operandFirst, int operandSecond) {  
        this.operandFirst = operandFirst;  
        this.operandSecond = operandSecond;  
    }  
}
```

```
public class CalculationResult {  
    int calculationResult;  
  
    public CalculationResult(int calculationResult) {  
        this.calculationResult = calculationResult;  
    }  
}
```

```
public class CalculateService {  
    public CalculationResult add(OperandRequest input) {  
        CalculationResult result = new CalculationResult(input.getOperandFirst() + input.getOperandSecond());  
        return result;  
    }  
    public CalculationResult sub(OperandRequest input) {  
        CalculationResult result = new CalculationResult(input.getOperandFirst() - input.getOperandSecond());  
        return result;  
    }  
    public CalculationResult mul(OperandRequest input) {  
        CalculationResult result = new CalculationResult(input.getOperandFirst() * input.getOperandSecond());  
        return result;  
    }  
    public CalculationResult division(OperandRequest input) {  
        CalculationResult result = new CalculationResult(input.getOperandFirst() / input.getOperandSecond());  
        return result;  
    }  
}
```

03. 아주 간단한 계산기 만들기

```
@RestController
@RequestMapping("api")
public class CalculateController {
    CalculateService calculateService = new CalculateService();

    @PostMapping("/v1/calculate/add")
    public CalculationResult getAdd(@RequestBody OperandRequest input){
        return calculateService.add(input);
    }

    @PostMapping("/v1/calculate/sub")
    public CalculationResult getSub(@RequestBody OperandRequest input){
        return calculateService.sub(input);
    }

    @PostMapping("/v1/calculate/mul")
    public CalculationResult getMul(@RequestBody OperandRequest input){
        return calculateService.mul(input);
    }

    @PostMapping("/v1/calculate/div")
    public CalculationResult getDiv(@RequestBody OperandRequest input){
        return calculateService.division(input);
    }
}
```

The screenshot displays two instances of a REST client interface, likely Postman, showing HTTP requests and their corresponding JSON responses.

Top Request:

- Method:** POST
- URL:** http://localhost:8080/api/v1/calculate/add
- Body (JSON):**

```
{  "operandFirst": 22,  "operandSecond": 1}
```
- Response:** 200 OK, 586 ms, 188 B. The response body is:

```
{  "calculationResult": 23}
```


Bottom Request:

- Method:** POST
- URL:** http://localhost:8080/api/v1/calculate/sub
- Body (JSON):**



```
{  "operandFirst": 22,  "operandSecond": 1}
```
- Response:** 200 OK, 11 ms, 188 B. The response body is:

```
{  "calculationResult": 21}
```





03. 아주 간단한 계산기 만들기



POST  http://localhost:8080/api/v1/calculate/mul

Params Auth Headers (9) Body ● Pre-req. Tests Settings


raw  JSON 

```
1 {  
2   ... "operandFirst": 22,  
3   ... "operandSecond": "2"  
4 }
```



Body   200  Body 

Pretty Raw Preview Visualize JSON  





```
1 {  
2   "calculationResult": 44  
3 }
```


POST  http://localhost:8080/api/v1/calculate/div

Params Auth Headers (9) Body ● Pre-req. Tests Sett

raw  JSON 

```
1 {  
2   ... "operandFirst": 22,  
3   ... "operandSecond": "2"  
4 }
```

Body   200  Body 

Pretty Raw Preview Visualize JSON 

```
1 {  
2   "calculationResult": 11  
3 }
```

04. Controller + Service + Repository

ORM

Object-Relational Mapping

- 객체가 테이블이 되도록 매핑 시켜주는 프레임워크 이다.
- ex) 기존쿼리 : `SELECT * FROM MEMBER;`

ORM : Member테이블과 매핑된 객체가 member라고 할 때, `member.findAll()`이라는 메서드 호출로 데이터 조회

JPA

Java Persistence API

- 자바에서 ORM을 사용하기 위한 인터페이스 모음
- Hibernate, EclipseLink, DataNucleus, OpenJPA, TopLink 등
- 인터페이스이기 때문에 JPA를 사용하기 위해서는 JPA를 구현한 Hibernate, EclipseLink, DataNucleus 같은 ORM 프레임워크를 사용해야 한다.

04. Controller + Service + Repository

dependencies 추가

```
implementation "org.springframework.boot:spring-boot-starter-  
data-jpa"  
implementation "com.h2database:h2"  
implementation 'org.projectlombok:lombok:1.18.18'
```

```
annotationProcessor  
'org.projectlombok:lombok'  
testAnnotationProcessor  
'org.projectlombok:lombok'
```

Entity (DB 구조 만들기)

- DB에 쓰일 필드와 여러 엔티티간 연관관계를 정의
- 열 부분이 **Column (필드)** 이고, 행 부분이 엔티티 객체
- 테이블 전체가 **엔티티** 이고, 각 1개의 행들이 **엔티티 객체**

```
package hello.hellospring.user;  
  
import lombok.*;  
  
import javax.persistence.*;  
  
@Data  
@AllArgsConstructor  
@NoArgsConstructor(access = AccessLevel.PROTECTED)  
@Entity(name = "user") //테이블과 링크될 클래스  
  
public class UserEntity {  
    @Id //해당 테이블의 PK필드  
    @GeneratedValue(strategy = GenerationType.IDENTITY) //PK 생성 규칙  
    private Long id;  
  
    @Column(length = 5, nullable = false)  
    private String name;  
  
    @Column(columnDefinition = "TEXT", nullable = false)  
    private int age;  
  
    public String getName() {  
        return name;  
    }  
  
    public int getAge() {  
        return age;  
    }  
  
    @Builder  
    public UserEntity(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
}
```


04. Controller + Service + Repository

Repository

- Entity에 의해 생성된 DB에 접근하는 메서드들을 사용하기 위한 인터페이스
- DB에 어떤 값을 넣거나, 넣어진 값을 조회하는 등의 CRUD(Create, Read, Update, Delete)를 정의하는 곳
- **JpaRepository**를 상속받도록 함으로써 기본적인 동작이 모두 가능
- **JpaRepository<대상으로 지정할 엔티티, 해당 엔티티의 PK의 타입>**

JpaRepository

Java Persistence API

- 자바에서 ORM을 사용하기 위한 인터페이스 모음
- Hibernate, EclipseLink, DataNucleus, OpenJPA, TopLink 등
- 인터페이스이기 때문에 JPA를 사용하기 위해서는 JPA를 구현한 Hibernate, EclipseLink, DataNucleus 같은 ORM 프레임워크를 사용해야 한다.

```
package hello.hellospring.user;

import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<UserEntity, Long> {
}
```

04. Controller + Service + Repository

```
package hello.hellospring.user;

import hello.hellospring.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;

@RestController
@RequestMapping("api")
public class PostController{
    @Autowired
    UserRepository userRepository;

    @PostMapping("/DB/add")
    public UserEntity addDB(@RequestBody UserEntity user){
        //given
        userRepository.save(UserEntity.builder()
            .name(user.getName())
            .age(user.getAge())
            .build());


        List<UserEntity> userList = userRepository.findAll();
        //then
        for(int i = 0; i<userList.size(); i++){
            if(userList.get(i).getName() == user.getName()){
                return userList.get(i);
            }
        }
        return null;
    }
}
```


POST ▼ http://localhost:8080/api/DB/add

Params Auth Headers (9) Body ● Pre-req. Tests Settings

raw ▼ JSON ▼

```
1 {
2   ... "name": "jiwoo",
3   ... "age": "22"
4 }
```

Body ▼  200

Pretty Raw Preview Visualize JSON ▼ 

```
1 {
2   "id": 1,
3   "name": "jiwoo",
4   "age": 22
5 }
```

05. Controller + Service + Repository

- 유저 등록과 더불어 조회 기능이 있는 서버 생성

```
@PostMapping("/DB/find")
public UserEntity findDB(@RequestBody String name){
    List<UserEntity> userList = userRepository.findAll();
    //then
    for(int i = 0; i<userList.size(); i++){
        if(userList.get(i).getName().equals(name)){
            return userList.get(i);
        }
    }
    return null;
}

@PostMapping("/DB/findAll")
public List<UserEntity> findAllDB(){
    List<UserEntity> userList = userRepository.findAll();
    //then
    return userList;
}
```

Body



200 OK 6 ms

Pretty Raw Preview Visualize

```
[{"id":1,"name":"jiwoo","age":20}, {"id":2,"name":"SSG","age":20}]
```

POST



http://localhost:8080/api/DB/find

Params

Auth

Headers (9)

Body

Pre-req.

Tests

Setting

raw



Text



1 jiwoo

Body



Pretty

Raw

Preview

Visualize

```
{"id":1,"name":"jiwoo","age":20}
```