@RestController

@Controller에 @ResponseBody가 결합된 어노테이션

- 컨트롤러 클래스 하위 메소드에 @ResponseBody 어노테이션을 붙이지 않아도 문자열과 JSON 등을 전송할 수 있음
- View를 거치지 않고 HTTP ResponseBody에 직접 Return 값을 담아 보내게 됨
- Spring Framework 4 버전부터 사용가능한 어노테이션

GET API

@RequestMapping

클래스와 메소드의 RequestMapping을 통해 URL을 mapping하여 경로를 설정, 해당 메소드에서 처리

- (예시) `@RequestMapping(value="/hello" method=RequestMethod.GET)`
  - value: url 설정
  - method: GET, POST, DELETE, PUT, PATCH 등
- 스프링 4.3버전 부터 간단하게 사용할 수 있는 어노테이션 사용 가능
  - `@GetMapping("/hello")`
    - @PathVariable: GET 형식의 요청에서 파라미터를 전달하기 위해 URL에 값을 담아 요청하는 방법
      - 방식1: {변수}의 이름과 메소드의 매개변수 이름이 일치해야 함

```
@GetMapping("/hello/{variable}")
public String getVariable (@PathVariable String variable) { }
```

      - 방식2: {변수}의 이름과 메소드의 매개변수 이름이 다를 경우

```
@GetMapping("/hello/{variable}")
public String getVariable (@PathVariable("variable") String var) { }
```

  - `@PostMapping("/hello")`
  - `@DeleteMapping("/hello")`
  - `@PutMapping("/hello")`
  - `@PatchMapping("/hello")`

@RequestParam

GET 형식의 요청에서 쿼리 문자열을 전달하기 위해 사용되는 방법

- ? 를 기준으로 {키}={값} 형태로 전달
  - 복수 형태로 전달할 경우 `&` 사용
  - 방식1

```
/hello?name=jhin&email=jhin@aaa.com
```

```
@GetMapping("/hello")
public String getRequestParam(
    @RequestParam String name,
    @RequestParam String email
) { }
```

  - 방식2: 어떤 요청 값이 들어올지 모를 경우 사용하는 방식

```
@GetMapping("/hello")
public String getRequestParam(@RequestParam Map<String, String> param) {
    StringBuilder sb = new StringBuilder();

    param.entrySet().forEach(map -> {
        sb.append(map.getKey()+":"+map.getValue()+"\n");
    });
```

```
        return sb.toString();
    }
```

API와 DTO의 사용

key와 value가 정해져있지만, 받아야할 파라미터가 많을 경우에 사용한다

- 예시

```
public class MemberDTO {
    private String name;
    private String email;
}
```

```
@GetMapping("/hello")
public String getRequestParam(MemberDTO memberDTO) {
    return memberDTO.toString();
}
```

POST API

resource를 추가하기 위해 사용되는 API

- @RequestMapping + POST method의 조합
- 일반적으로 추가하고자 하는 Resource를 http body에 추가하여 서버에 요청
    - @RequestBody를 이용하여 body에 담겨있는 값을 받아야 함

```
@PostMapping("/member")
public String postMember(@RequestParam Map<String, Object> postData) {
    StringBuilder sb = new StringBuilder();

    postData.entrySet().forEach(map -> {
        sb.append(map.getKey()+":"+map.getValue()+"\n");
    });
    return sb.toString();
```

DELETE API

서버를 통해 resource를 삭제하기 위해 사용되는 API

- 일반적으로 @PathVariable을 통해 resource ID 등을 받아 처리

```
@DeleteMapping("/delete/{variable}")
public String deleteVariable (@PathVariable String variable) { }
```

PUT API

해당 resource가 존재하면 갱신하고, 없을 경우에는 새로 생성해주는 API

- 기본적인 동작 방식은 POST API와 동일

```
@PutMapping("/hello")
public String postMemberDTO (@RequestBody MemberDTO memberDTO) {}
```

```
@PutMapping("/hello")
public ResponseEntity<MemberDTO> postMemberDTO (@RequestBody MemberDTO memberDTO) {
```

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

Response Entity

Spring Framework에서 제공하는 클래스 중 HttpEntity라는 클래스를 상속받아 사용하는 클래스

- 사용자의 HttpRequest에 대한 응답 데이터를 포함
- 포함하는 클래스
    - HttpStatus
    - HttpHeaders
    - HttpBody