Spring MVC 08 [스프링 MVC 기본기능 1]

스프링 MVC 1편 - 백엔드 웹 개발 핵심 기술

김영한

2022.08.23

요청 매핑

기본 매핑

```
@RestController
@Slf4j
public class MappingController {


@RequestMapping(value = {"hello-basic","/hello-go2"}, method = RequestMethod.POST)
→배열로 복수 설정 가능, @PostMapping으로 변경가능 HTTP method명+Mapping 애노테이션
   public String helloBasic(){
       log.info("helloBasic");
       return "ok";
   }
}
```

url은 / hello-basic , /hello-go2 , /hello-basic/ 모두 사용가능하다.

PathVariable 매핑

```
@GetMapping("/mapping/{userId}")
public String mappingPath(@PathVariable("userId") String data){
                          = @PathVariable String userId
   log.info("mappingPath userId = {}", data);
   return "ok";
}
```

경로 변수를 가져다가 쓸수 있고 최근 HTTP API는 리소스 경로에 식별자를 넣는 것을 선호한다.

PathVariabble 여러개 매핑

```
@GetMapping("/mapping/{userId}/orders/{orderId}")
public String mappingPath(@PathVariable String userId,@PathVariable String
orderId){
   log.info("mappingPath userId = {}", userId);
   log.info("mappingPath orderId = {}", orderId);
   return "ok";
}
```

매핑 조건으로 params = "mode=debug" 이런 식으로 쿼리파라미터

설정해서 넣어줄 수 있다. 해당 조건이 만족해야 매핑됨.

해더 조건 매핑

```
@GetMapping(value = "/mapping-header", headers = "mode=debug")
public String mappingHeader() {
   log.info("mappingHeader");
   return "ok";
}
```

헤더에 mode 정보가 debug여야 매핑이 동작한다.

미디어 타입 매핑

```
@PostMapping(value = "/mapping-consume", consumes = "application/json")
public String mappingConsumes() {
   log.info("mappingConsumes");
   return "ok";
}
```

content 타입이 미디어 타입이 맞아야 매핑이 동작한다.

```
@PostMapping(value = "/mapping-produce", produces = "text/html")
public String mappingProduces() {
   log.info("mappingProduces");
   return "ok";
```

```
}
```

HTTP 요청의 Accept 헤더를 기반으로 미디어 타입으로 매핑한다. 만약 맞지 않으면 HTTP 406 상태코드(Not Acceptable)을 반환한다

consume : content-type 에 따라 설정

produces : Accept 헤더를 기반으로 미디어 타입 매핑

요청 매핑 **- API** 로 매핑

매핑만 확인해 보면

```
@RestController
@RequestMapping("/mapping/users")
public class MappingClassController {

   @GetMapping → 목록 조회
   public String user(){
       return "get users";
   }

   @PostMapping → 가입
   public String addUser(){
       return "post user";
   }

   @GetMapping("/{userId}") → 조회
   public String findUser(@PathVariable String userId){
       return "get userId = " + userId;
   }

   @PatchMapping("/{userId}") → 수정
   public String updateUser(@PathVariable String userId){
       return "update userId = " + userId;
   }

   @DeleteMapping("/{userId}") → 삭제
   public String deleteUser(@PathVariable String userId){
       return "delete userId = " + userId;
```

```
    }
}
```

요청 HTTP Method에 따라 매핑이 동작한다.

HTTP 요청

기본, 헤더 조회

```
@Slf4j
@RestController
public class RequestHeaderController {

    @RequestMapping("/headers")
    public String headers(HttpServletRequest request,
                          HttpServletResponse response,
                          HttpMethod httpMethod,
                          Locale locale,
                          @RequestHeader MultiValueMap<String, String> headerMap,
                          @RequestHeader("host") String host,
                          @CookieValue(value = "myCookie", required = false) String
cookie
                         ){
        log.info("request={}", request);
        log.info("response={}", response);
        log.info("httpMethod={}", httpMethod);
        log.info("locale={}", locale);
        log.info("headerMap={}", headerMap);
        log.info("header host={}", host);
        log.info("myCookie={}", cookie);
        return "ok";
    }
}
```

여기서 MutliValueMap은 value는 값을 배열로 가져온다.

요청에 대한 파라미터는

https://docs.spring.io/spring-framework/docs/current/reference/html/web.html#mvc-ann-arguments 공식 문서로 확인도 가능하다.

쿼리 파라미터 ,HTML Form

클라이언트에서 서버로 요청 데이터를 전달할 때 주로 3가지 방법을 사용한다.

1. GET - 쿼리 파라미터
2. POST -HTML Form 형식
3. HTTP message body 에 데이터를 직접 담아서 요청한다.

GET 쿼리파라미터 & Form 형식

: 요청 파라미터 조회로 조회 할 수 있다**.**

기본적으로 HttpServletRequest 로 getParameter("파라미터이름") 으로 조회가능하다.

```
@RequestMapping("/request-param-v1")
public void requestParamV1(HttpServletRequest request, HttpServletResponse
response) throws IOException {
   String username = request.getParameter("username");
   int age = Integer.parseInt(request.getParameter("age"));
   log.info("username={}, age={}", username, age);
   response.getWriter().write("ok");
}

@RequestMapping("/request-param-v2")
@ResponseBody
public String requestParamV2(
       @RequestParam("username") String memberName,
       @RequestParam("age") int memberAge){
   log.info("username={}, age={}", memberName, memberAge);
   return "ok";
}
```

같은 방식으로 동작한다. @Responce body를 맞춰준다.

@RequestParam 생략 가능하다 , 파라미터 명과 같을때

```
@RequestMapping("/request-param-v4")
```

```
@ResponseBody
public String requestParamV4(String username,int age){
   log.info("username={}, age={}", username, age);
   return "ok";
}
```

이렇게 까지 생략가능하지만 명시적으로 쓰는것이 좋을 수도..?

필수 파라미터

```
@RequestParam(required = true)
```

필수 파라미터를 required로 설정 가능하다. default값은 true 이다. 필수 값이 없을때는 400 오류를 내려준다.

```
@RequestParam(required = false) int age){
```

만약 이렇게 지정을하고 파라미터를 주지 않으면 500에러가 발생한다. 파라미터 타입을 기본형이 아닌 클래스 형으로 사용하는게 좋고 ,

null, “ “ 빈문자열은 다르다는것을 인지해야 한다.

Map 형식으로 파라미터 받기

```
@RequestMapping("/request-param-map")
@ResponseBody
public String requestParamMap(@RequestParam Map<String, Object> paramMap){
   log.info("username={}, age={}", paramMap.get("username"), paramMap.get("age"));
   return "ok";
}
```

- 만약 ?userId=id1&userId=id2 이런시긍로 두개를 요청하게 되면

  MultiValueMap으로 받을 수 있다.