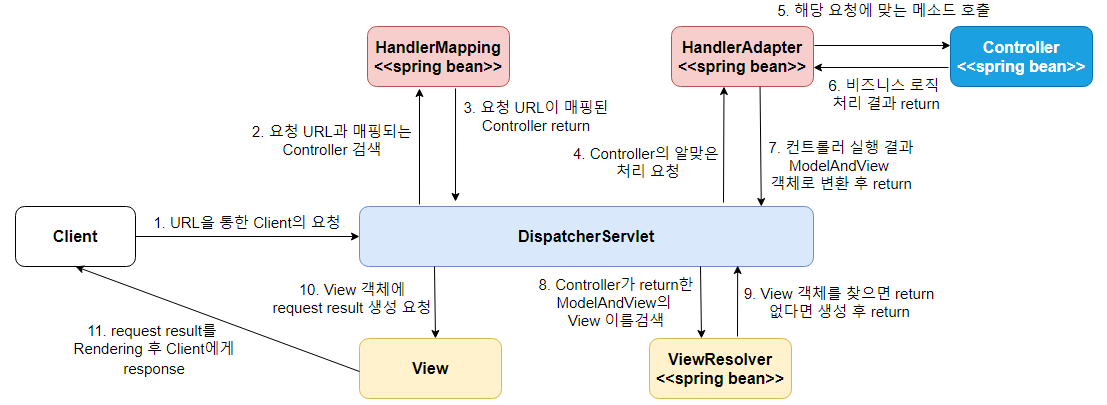
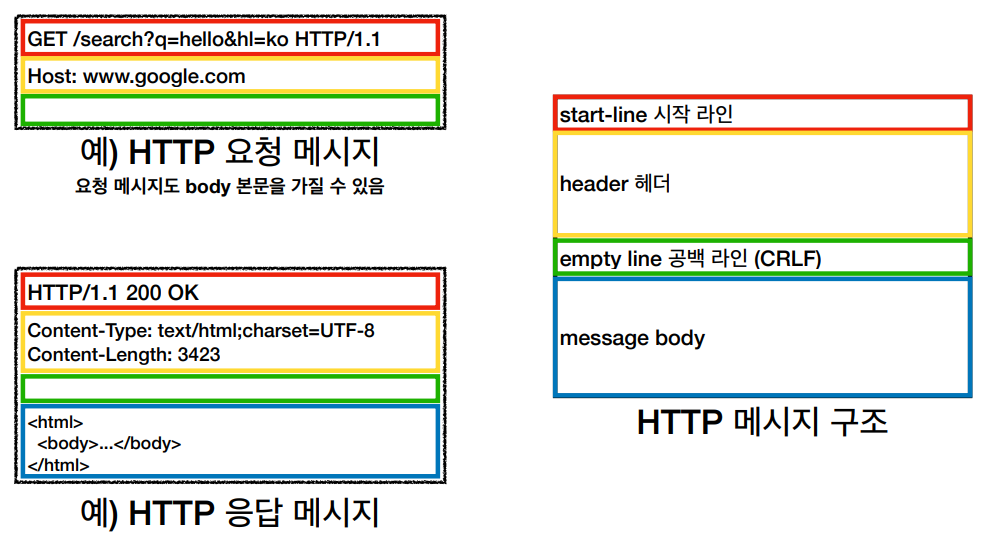
**Spring 복습: 컨트롤러 요청/데이터 수집, 응답 관련 어노테이션 정리**





**HTTP 웹 기본 지식**

|  |
| --- |
| **URL: Uniform Resource Location (위치)** |
| * 프로토콜://호스트명(IP주소):포트번호/Path/…?query파라미터들#fragment * 예시) <http://localhost:8080/hello/world> (전체 URL) |
|  |
| **URI: Uniform Resource Identifier (식별자)** |
| * 전체 URL에서, 포트번호 뒷부분. * 예시) /hello/world |

1. **클라이언트(브라우저)에서 서버로 요청 시, 주로 사용하는 데이터 전달방법 3가지**

|  |
| --- |
| **GET** – 쿼리 파라미터(쿼리 스트링, 파라미터) |
| * /mypage?name=피카츄&age=10 * HTTP Message Body 없이, URL의 쿼리 파라미터에 데이터 포함해서 전달 - 넘어가는 데이터가 URL에 노출됨 * 검색, 필터, 페이징처리 등에서 많이 사용 |
|  |
| **POST** – HTML form 태그 |
| <form action=”/asdf” method=”post” />   * Content-Type: application/x-www-form-unlencoded (default) * MessageBody에 쿼리 파라미터 형식으로 전달 (데이터가 URL 노출 X) |
|  |
| **HTTP Message Body**에 직접 데이터를 담아서 요청 |
| * REST(HTTP) API에서 주로 사용 * JSON, XML, TEXT 등 여러 형태의 데이터를 보낼 수 있음 (주로 JSON 사용) * 전송 방식: POST, PUT, PATCH, … |

* 1. **요청 시 넘어온 데이터를 서버 측에서 꺼내기**
* 기본: String *변수* = request.getParameter(“파라미터명”); 문자열로 리턴
  + 매번 하나씩 꺼내기 번거로움, 스프링엥서 제공하는 다양한 수집 방법 활용 필요…
    - 컨트롤러 메서드 매개변수에 일반 변수 나열 / DTO, VO 나열…

1. **컨트롤러에서 많이 사용되는 annotation, Class**

|  |  |
| --- | --- |
| **Annotation** | **Class** |
| @RequestParam  @RequestBody  @Pathvariable  @ResponseBody  @ModelAttribute | HttpEntity  ResponseEntity  RedirectAttribute |
|  |  |
| **@RequestParam** | |
| * 일반 파라미터 바인딩 (String, int 같은 타입…) * 쿼리 파라미터, HTML form으로 넘어오는 데이터를 받아주는 매개변수 앞에 부착 * **html form**에서, **name=”속성”, value=”속성”** 값으로 넘어오는 파라미터를 매개변수에 바인딩 * Required=true(default)/false 속성으로 파라미터 필수 여부 지정 가능 * 파라미터명과 메서드의 매개변수명이 같으면 annotation value생략 가능 * **기본형**(int 등)은 **파라미터가 아예 안 넘어올 시 null을 취급할 수 없어 500 에러** 발생 - Integer 같은 **wrapper** 클래스로 변경하거나, **defaultValue** 속성으로 기본값 채워야 함 * 파라미터를 Map, MultiValueMap으로 받을 수 있음 - @RequestParam Map<String, Object> paramMap  - <파라미터명, 파라미터값>  - Map: 파라미터 값이 1개가 확실할 때 사용 (key=value)  - MultiValueMap: 파라미터가 2개 이상일 경우 사용 (key=value, key=[value1, value2]   \* 파라미터 수집 \*  @RequestParam("id") String id (스프링 기능)  -> String id = request.getParameter("id"); (서블릿 기능)  @RequestParam("hobby") ArrayList<String> hobby  @RequestParam ArrayList<String> hobby  @RequestParam MultiValueMap<String, Object> map ---> Map X  String[ ] checked = request.getParameterValues("hobby"); | |
| (예시 1) | |
|  | |
| **@RequestBody** | |
| * 데이터를 HTTP Message Body에 직접 담아서 요청 시 사용 * HTTP Message Body의 데이터를 자바 객체에 바인딩해주는 어노테이션   \* @RequestParam, @ModelAttribute 사용 불가  \* **@RequestBody (필요 시) 생략 불가능**   * 일반 데이터: String 타입으로 조회 - @RequestBody String data → StringHttpMessageConverter 사용 * JSON 형태 데이터: String 타입으로 조회 - @RequestBody String data → StringHttpMessageConverter 사용 - com.fasterxml.jackson.databind.ObjectMapper를 통해 자바 객체로 변환 가능  Hello data = ObjectMapper.readValue(messageBody, Hello.java); * **JSON 형태 데이터: 객체로 변환하여 받기** (content-type: application/json) - @RequestBody Hello data → MappingJackson2HttpMessageConverter 사용 - HttpEntity, @RequestBody를 사용하면, HttpMessageConverter가 HttpMessageBody의 내용을 우리가 원하는 문자나 객체 등으로 변환해줌 * JSON 형태의 데이터를 사용할 때는 json 라이브러리 추가 필수 - Jackson-databind, gson 등 (Boot의 경우 dependencies에 추가) | |
|  | |
| **@ModelAttribute** | |
| * 객체 형태로 파라미터 받기 * 쿼리 파라미터, HTML form * 넘어오는 파라미터를, **‘model’ 객체**를 생성하여 그 안에 자동으로 바인딩 (setter 호출) * 파라미터와 객체 내 변수 타입 불일치 시 BindException 발생 * @ModelAttribute 어노테이션 생략 가능 - @RequestParam은 생략 가능 (String, int 등 단순 타입의 경우) - @ModelAttribute은 생략 가능 (나머지…) * 추가 기능: Model에 객체를 자동으로 추가 → view에 전달 - model에 담을 때, 이름은 @Annotation(”*속성*”)으로 지정 가능 - 이름 생략 시 클래스명 첫 글자에 소문자로 붙여서 전달 * 추가 기능: 메서드 레벨에 어노테이션을 붙이면, 해당 컨트롤러 내 매핑 메서드가  될 때마다 매번 자동으로 호출되어 리턴되는 데이터를 Model에 추가  → 모든 view에 데이터 전달 가능 | |
| Ex02, 그 바로 밑에거 | |
|  | |
| **@Pathvariable** | |
| * URL 경로의 값을 꺼낼 때 사용 * 경로 변수명 { *data* }와 파라미터 이름 String *data* 이름이 동일하면, 어노테이션에서 (“*data*”) 는 생략 가능 * RestAPI 방식 개발 시 많이 사용 * @PathVariable 값은 해당 메서드 redirect 시 리턴되는 경로 변수로 바인딩 가능 - return “redirect:/ex03/{data}”; 문자열 ‘data’는 @Pathvariable의 값으로 바인딩 | |
| Ex03 | |
|  | |
| **HttpEntity** | |
| * Http header, body 편리하게 조회 가능 클래스 * Http Message Body 정보를 직접 조회 * @RequestBody처럼 JSON 요청 데이터 바인딩 가능 - HttpEntity<TestDTO> entity → TestDTO data = entity.getBody( ); - HttpMessageConverter가 body의 내용을 변환해줌 | |
|  | |
| **@ResponseBody** | |
| * 응답 데이터를 HttpMessageBody에 직접 담아서 전달 * RestAPI에서 많이 사용 * **View 사용 X** * 한글 깨짐 발생 가능 → ResponseEntity로 header에 인코딩 정보 포함하여 응답 | |
|  | |
| **ReponseEntity**<**E**> | |
| * 응답 데이터 + 헤더 정보를 함께 응답으로 보내줄 수 있는 타입 * REST API에서 많이 사용 * **View 사용 X** | |
| Ex04~07 | |
|  | |
| **RedirectAttribute** | |
| * 리다이렉트 시, 데이터를 보내는 방법 * 컨트롤러 메서드에 매개변수 선언 후 사용 redirectAttributes.addAttribute(”data1", "qwer"); redirectAttributes.addFlashAttribute("data2", "asdf"); - addAttribute(“이름”, “값”);  - URL에 노출, refresh 해도 데이터 유지 (…8080/ex09?data1=qwer)  - th:text=${param.이름} → ${param.data1} - addFlashAttribute(“이름”, 값); *(단발성으로 사용하는 경우)*  - URL 노출 X, 리다이렉트 이후 소멸 (…8080/ex09)  - th:text=${이름} → ${data2} | |
| Ex08, ex09 | |