

스프링 부트 웹 프로젝트

chapter06

컨트롤러

제공된 자료는 훈련생의 수업을 돕기 위한 것으로, 타인과 공유하시면 안됩니다.

Contents

part.1

HTTP 메시지

part.2

스프링MVC의 구조

part.3

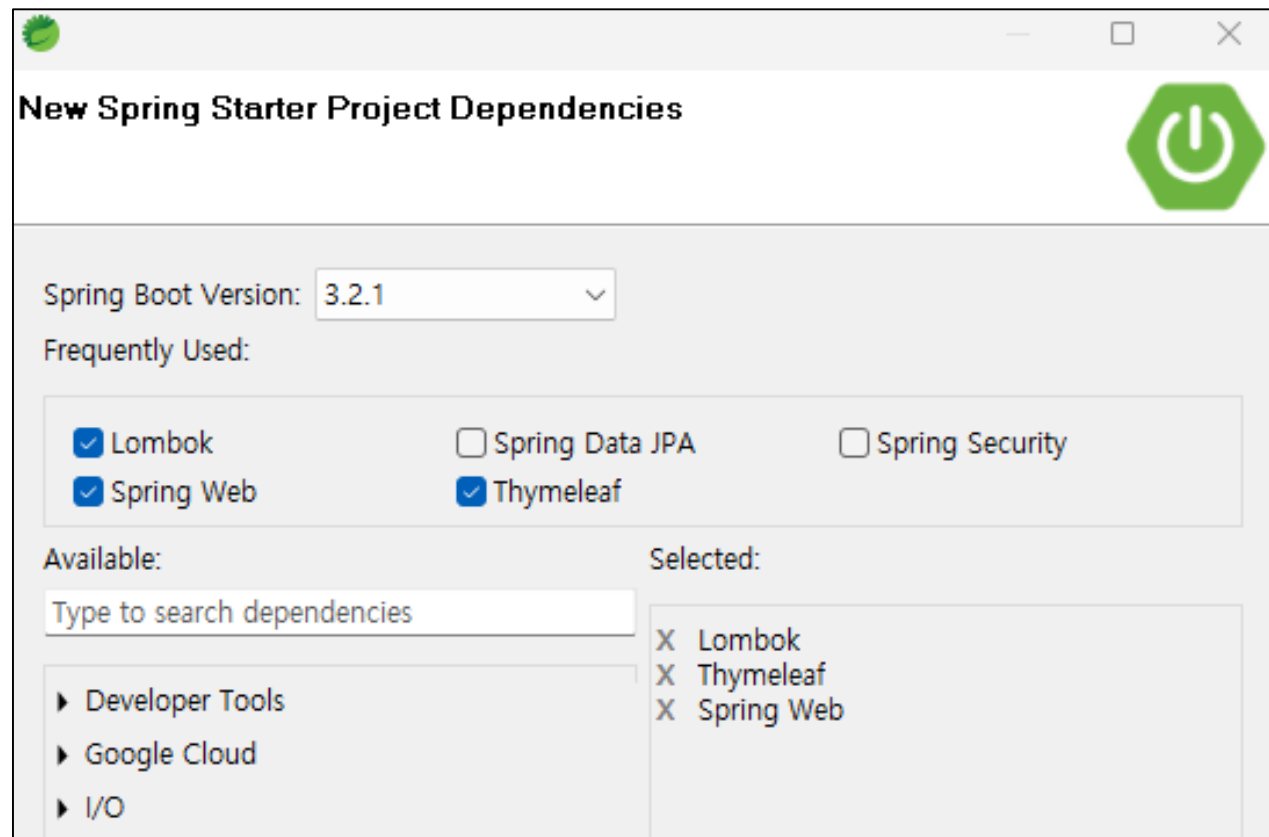
스프링MVC의 Controller

part.4

테스트 도구

실습을 위해 프로젝트를 생성한다.

라이브러리는 'Lombok, Thymeleaf, Spring Web'를 선택한다.



요청메세지는 헤더와 바디 두 부분으로 구성된다.

Header

HTTP 메소드: 요청을 보내는 목적을 나타내는 부분 (예: GET, POST, PUT, DELETE)

URL: 요청을 보낼 서버의 주소

파라미터: URL에 포함된 데이터로, ?key=value 형식으로 전달

데이터형식: 전송할 데이터의 형식 (예: application/json, text/html 등)

Body

Form 데이터: HTML 폼을 통해서 데이터 전송할 때 사용

바디 데이터: 요청 바디에 포함된 데이터로,

JSON, XML 등 다양한 형식의 데이터를 전송할 때 사용

```
POST /board/register HTTP
```

```
Host: localhost:8080
```

```
Content-Type: application/x-www-form-urlencoded
```

```
name=둘리&age=10
```

요청 메시지 예시

요청 URL

서비스 요청 주소는 다음과 같이 구성된다.

사용자가 서비스를 쉽게 이용할 수 있도록 URL은 직관적으로 설계해야 한다.

예를 들어 게시물 목록을 보여주는 경로는 /aaa처럼 애매하게 작성하는 것보다는, /board/list 같이 명확하게 작성하는 것이 좋다.

요청 메소드

요청 메소드는 수행하려는 작업에 따라 적절하게 선택해야 한다.

- GET: 조회를 위해 특정 데이터 요청
- POST: 새로운 데이터 등록
- PUT: 기존 데이터를 수정
- DELETE: 특정 데이터를 삭제

/aaa X

POST /board/list X

/board/list O

GET /board/list O

Header

상태 코드: 서버가 클라이언트 요청을 처리한 결과를 나타내는 코드

응답 데이터 형식: 서버가 반환하는 데이터의 형식 (예: application/json, text/html)

Body

결과 데이터: 요청이 성공적으로 처리되었을 때 서버가 반환하는 실제 데이터 (예: HTML, JSON, XML)

상태 코드

- 2xx: 요청이 성공적으로 처리되었음
- 3xx: 리다이렉션. 새로운 URL로 요청을 다시 보냄
- 4xx: 클라이언트 요청에 문제가 있음
- 5xx: 서버가 요청을 처리하는 도중 문제가 발생함

HTTP 응답 상태 코드, mdn web docs, <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

```
HTTP 200 OK
Content-Type: application/json
{
  "name" : "둘리",
  "age" : 20
}
```

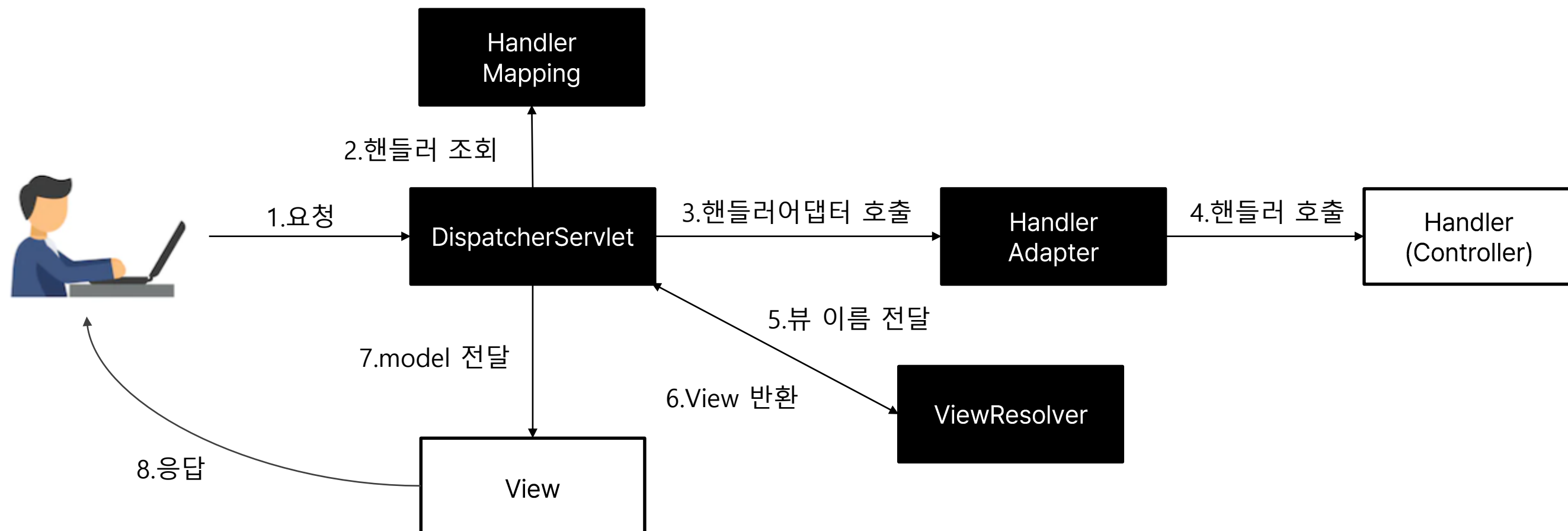
응답 메시지 예시

Spring MVC의 실제 구조는 다음과 같다.

Spring MVC는 크게 DispatcherServlet, HandlerAdapter, ViewResolver로 구성된다.

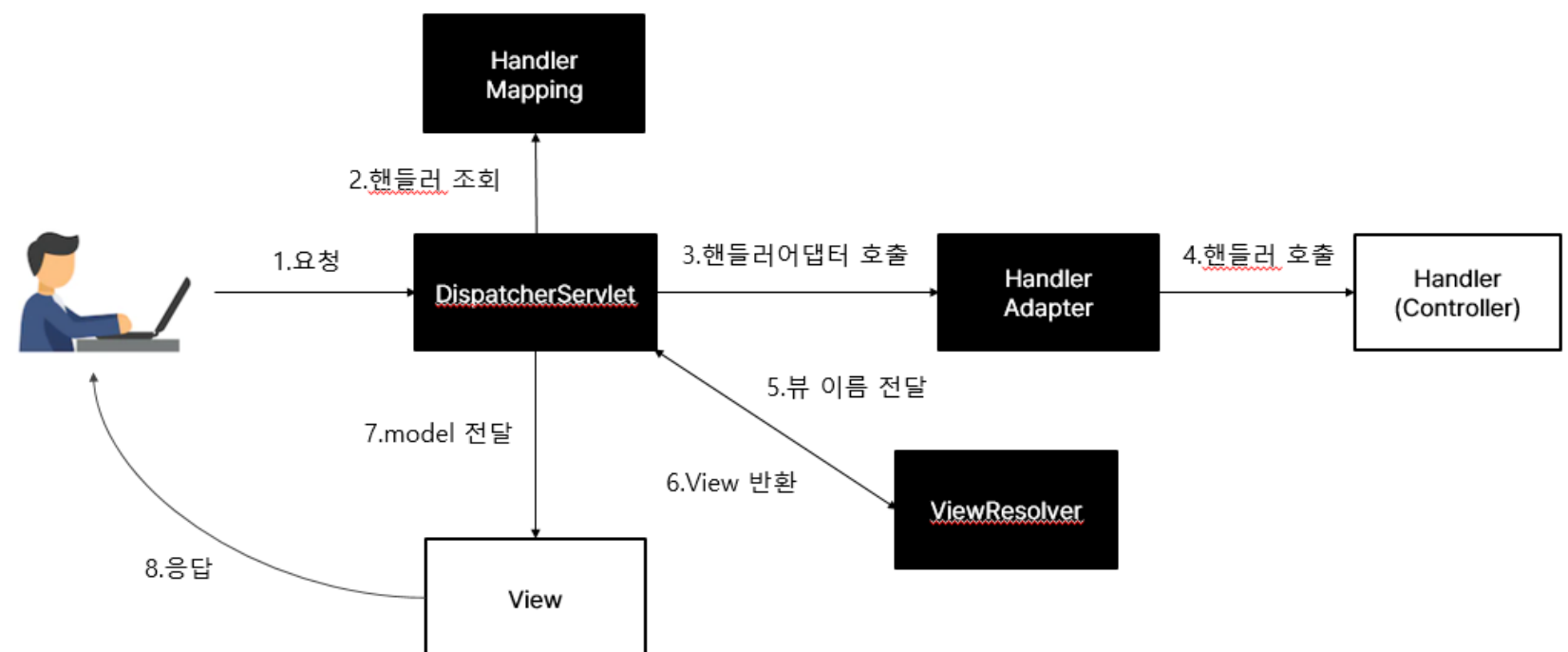
이것들은 스프링 프레임워크에서 자동으로 생성하고 관리하므로, 개발자가 직접 처리할 필요는 없다.

개발자는 Model, View, Controller만 구현하면 된다.



MVC 처리 과정

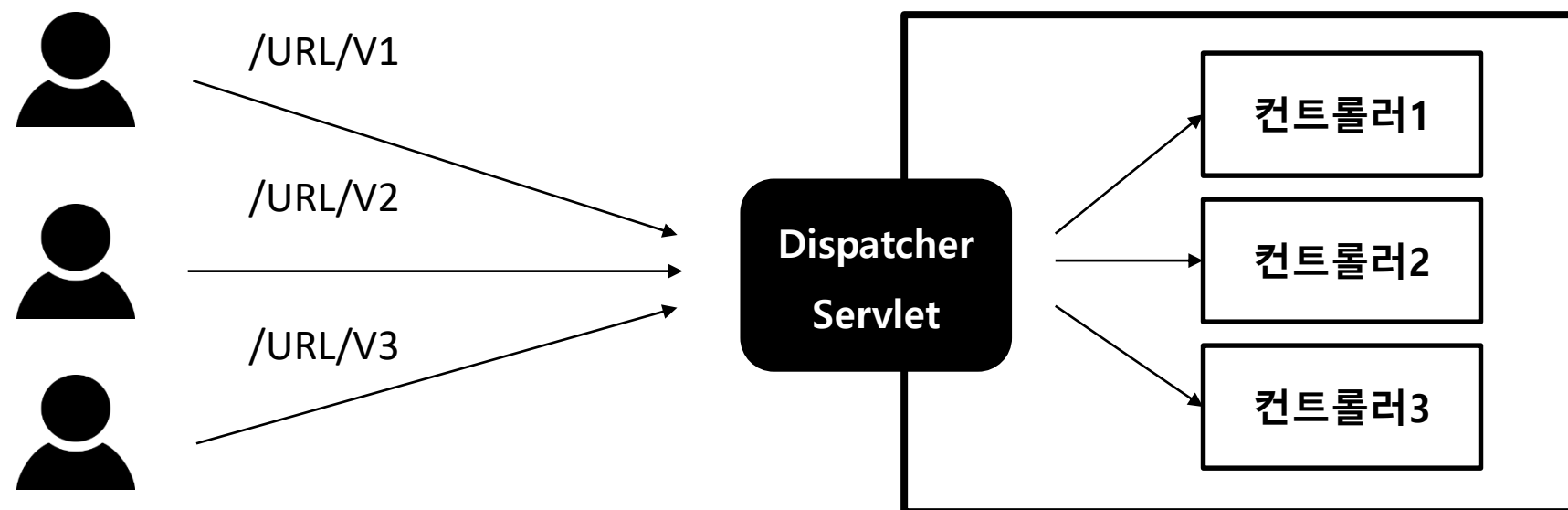
1. 사용자가 요청을 보내면, DispatcherServlet가 요청을 받는다.
2. DispatcherServlet은 요청된 URL과 매핑된 핸들러(컨트롤러) 정보를 찾는다.
3. HandlerAdapter가 핸들러(컨트롤러)를 호출하여 요청을 처리한다.
4. 핸들러(컨트롤러)는 요청을 처리한 후, 응답할 뷰 이름을 반환한다.
5. ViewResolver는 실제 뷰 파일을 찾는다.
6. 모델 데이터를 뷰에 전달한다.
7. 최종적으로 생성된 화면을 사용자에게 전송한다.



DispatcherServlet는 Spring MVC의 핵심 요소로, 사용자 요청을 처리하고 컨트롤러와 뷰를 연결하는 역할이다.

DispatcherServlet의 역할

1. 하나의 서블릿으로 모든 사용자 요청을 중앙에서 처리한다.
2. 적절한 컨트롤러에게 요청을 전달한다.
3. 요청을 처리한 후, 적절한 뷰를 선택하여 사용자에게 응답을 전송한다.

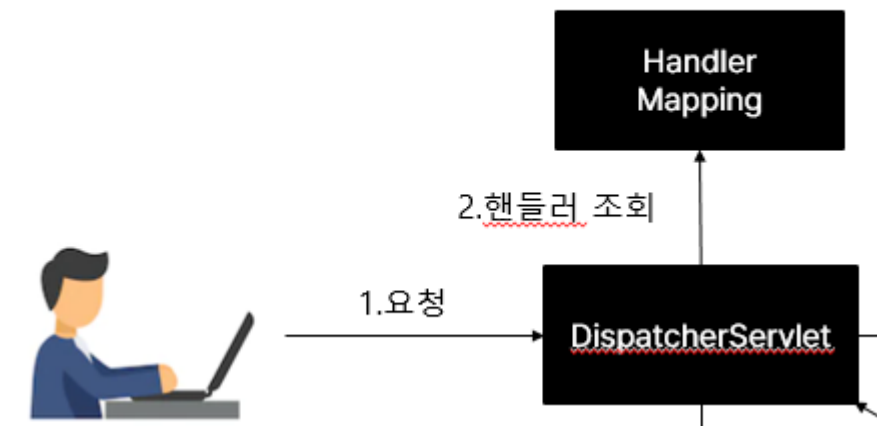


HandlerMapping은 사용자 요청을 분석하여, 해당 요청을 처리할 핸들러(컨트롤러)를 찾는 역할이다.
요청된 URL과 매핑된 컨트롤러를 찾아서 연결한다.

컨트롤러 목록

```

v controller
  > BoardController.java
  > HomeController.java
  > MemberController.java
  
```



```

@Controller
@RequestMapping("/board")
public class BoardController {

    @GetMapping("/list")
    public void list(@RequestParam(defaultValue = "1") int page) {

    }

    @GetMapping("/register")
    public void register() {

    }
}
  
```

```

@Controller
public class HomeController {

    @GetMapping("/")
    public void home() {

    }
}
  
```

```

@Controller
@RequestMapping("/member")
public class MemberController {

    @GetMapping("/list")
    public void list(@RequestParam(name = "page", defaultValue = "1") int page) {

    }

    @GetMapping("/register")
    public void register() {

    }
}
  
```

HandlerAdapter은 선택한 핸들러(컨트롤러)를 실행하는 역할이다.

컨트롤러가 반환한 결과를 적절한 뷰로 변환하여 사용자에게 응답을 전송한다.

그런데 모든 컨트롤러가 동일한 방식으로 처리되지는 않는다. 컨트롤러는 상황에 따라 다양한 반환 타입을 사용할 수 있다. 예를 들어 ModelAndView는 뷰 이름과 데이터를 함께 담고 반환하고, Model은 데이터만 담아 전달한다. HandlerAdapter는 이러한 다양한 반환타입을 처리할 수 있다.



ModelAndView 사용한 컨트롤러

```
@Controller
public class ExampleController {

    @RequestMapping("/example")
    public ModelAndView handleRequest() {

        ModelAndView modelAndView = new ModelAndView();

        modelAndView.setViewName("exampleView"); // 뷰 이름 설정
        modelAndView.addObject("message", "Hello, ModelAndView!"); // 모델 데이터 설정
        return modelAndView;
    }
}
```

Model 사용한 컨트롤러

```
@Controller
public class ExampleController {

    @RequestMapping("/example")
    public String handleRequest(Model model) {

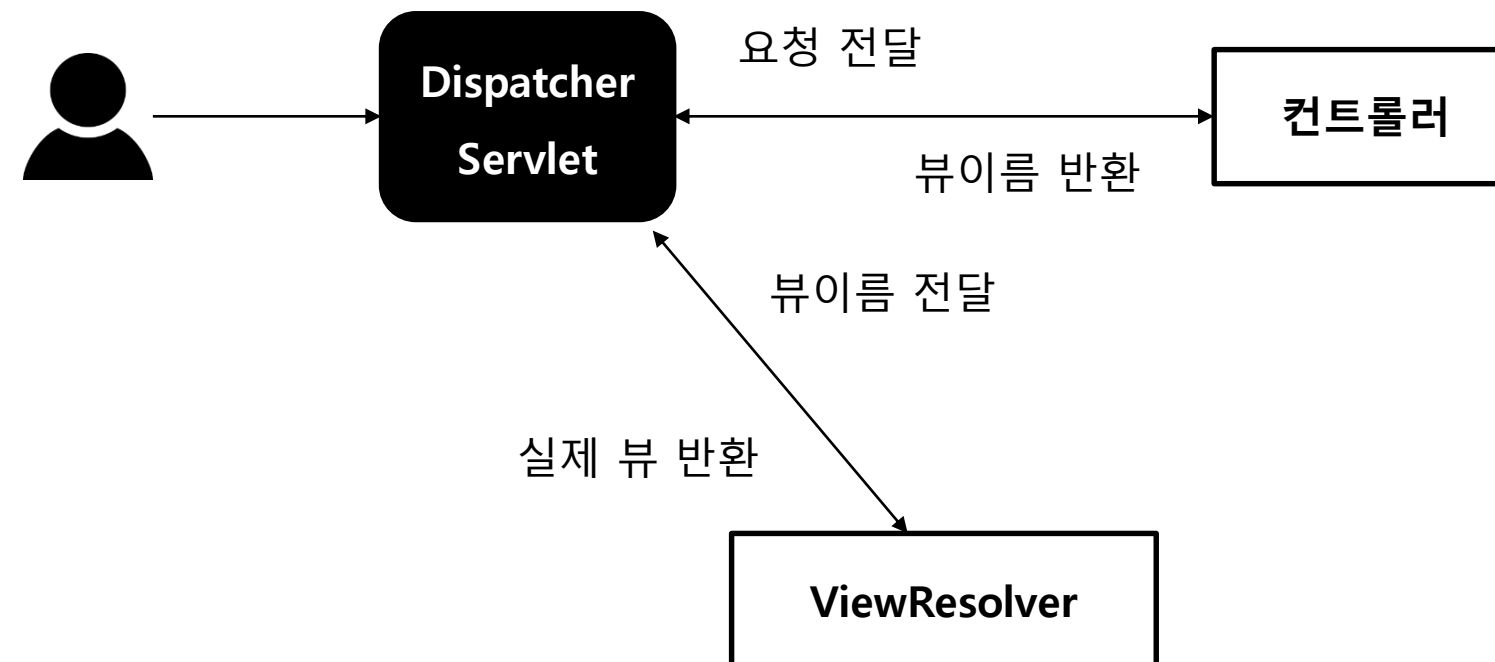
        model.addAttribute("message", "Hello, Model!"); // 모델 데이터 설정
        return "exampleView"; // 뷰 이름 반환
    }
}
```

ViewResolver은 뷰의 이름을 실제 뷰 파일 경로로 변경하는 역할이다.

예: home → /src/main/resources/template/home.html

ViewResolver의 역할

요청에 따라 뷰 파일을 동적으로 선택하여 반환한다.



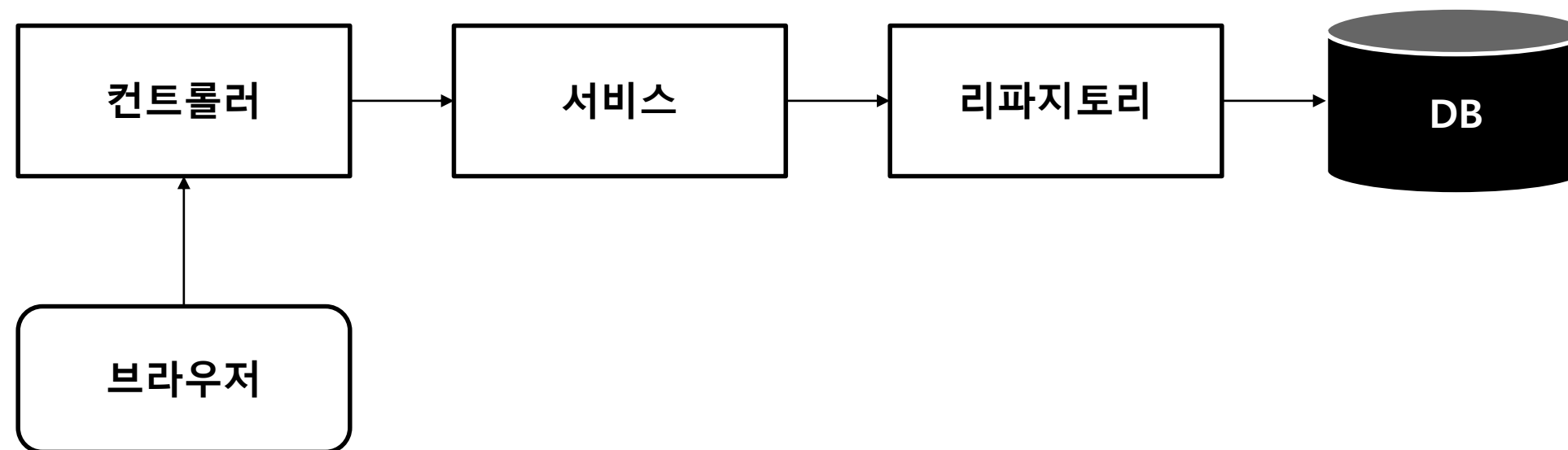
Controller, View, Model

컨트롤러(Controller): 사용자 요청을 처리하고, 모델과 뷰 사이를 연결한다.

서비스(Model): 비즈니스 로직을 처리하고, 데이터를 가공한다.

리포지토리(Model): 데이터베이스에서 데이터를 가져오거나 저장한다.

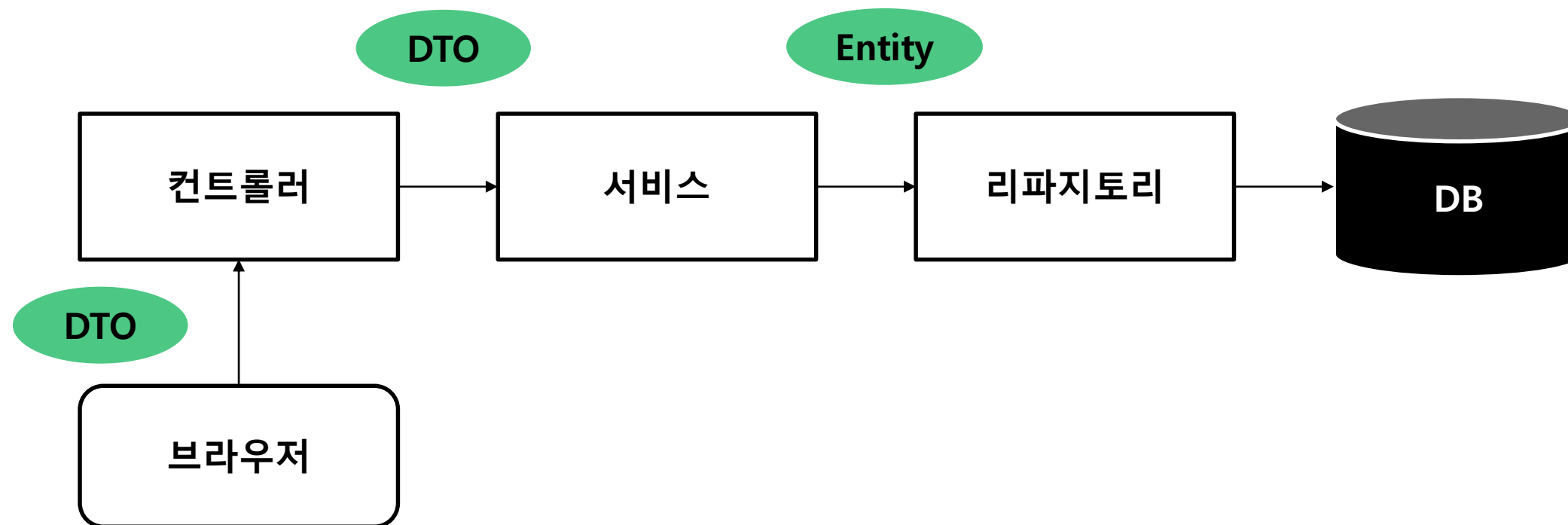
뷰(View): 사용자에게 보여줄 화면을 담당한다.



DTO와 Entity는 데이터를 담아 전달하기 위한 클래스이다.

DTO : 컨트롤러와 뷰가 데이터를 주고받을 때 사용한다.

Entity : 컨트롤러와 데이터베이스가 데이터를 주고받을 때 사용한다.



컨트롤러란?

사용자 요청을 받아들이고, 사용자가 원하는 데이터를 준비하여 응답을 보낸다.

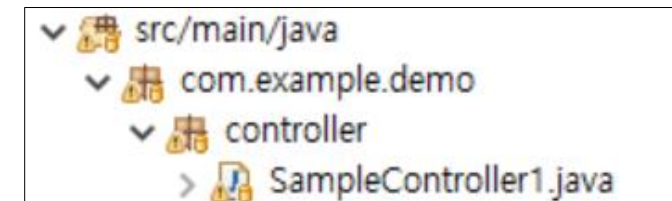
컨트롤러의 특징

- 다양한 타입의 파라미터 처리
- 다양한 타입의 리턴타입 사용
- GET, POST 등 전송 방식을 어노테이션으로 처리

@Controller 어노테이션

- 스프링이 자동으로 빈으로 등록한다.
- 해당 클래스를 컨트롤러로 인식한다.

컨트롤러 위치



컨트롤러 예시

```
@Controller
@RequestMapping("/sample")
public class SampleController {
    @GetMapping("/ex")
    public void ex1() {}
}
```

@RequestMapping

- 사용자 요청을 매핑하는데 사용되는 어노테이션이다.
- 특정 URL을 호출하면 특정 어노테이션이 적용된 메소드가 호출된다.
- 클래스 또는 메소드에 적용할 수 있다.

```
@RequestMapping(value = "/list", method = RequestMethod.GET)
```

@RequestMapping -> @GetMapping @PostMapping

- @RequestMapping을 더 간결하게 표현하기 위해 축약된 어노테이션이 등장했다.
- 이 축약된 어노테이션을 사용하면, 코드를 더 간단하게 작성할 수 있다.

```
@GetMapping("/list")
```


축약 어노테이션의 종류

어노테이션은 요청 목적에 맞게 사용해야 한다.

@GetMapping : 데이터 조회 요청

@PostMapping : 데이터 등록 요청

@PutMapping : 데이터 수정 요청

@DeleteMapping : 데이터 삭제 요청

예시

```
@GetMapping("/list")  
public ResponseEntity list() { }
```

서버에 목록 데이터 요청

```
@PostMapping("/save")  
public ResponseEntity save() { }
```

서버에 데이터 등록 요청

Q1. 다음 URL 주소로 연결되는 메소드를 만들고 호출하세요.

- get방식 + /method/q
- post방식 + /method/q
- put방식 + /method/q
- delete방식 + /method/q

요청메세지의 Header

요청메세지의 헤더에는 다양한 메타데이터가 포함되어 있다.

- Host: 요청이 전송되는 호스트의 주소를 지정
- User-Agent: 클라이언트 소프트웨어의 식별자를 제공
- Content-Type: 요청 본문의 데이터 형식을 지정
- Authorization: 요청을 인증하기 위한 인증 정보를 포함
- Accept: 클라이언트가 받아들일 수 있는 응답의 미디어 유형을 지정

Header에서 메타데이터를 조회하는 방법

- 스프링은 요청의 헤더 메타데이터를 조회할 수 있는 다양한 객체를 제공한다.
- 사용 가능한 객체는 스프링 공식문서에서 확인할 수 있다

요청메세지 예시

```
GET /api/resource?key1=value1&key2=value2 HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Accept: application/json
```

Method Arguments

[See equivalent in the Reactive stack](#)

The next table describes the supported controller method arguments. Reactive types are not supported.

JDK 8's `java.util.Optional` is supported as a method argument in combination with annotations that indicate the required attribute (for example, `@RequestParam`, `@RequestHeader`, and others) and is equivalent to `required=true`.

Controller method argument	Description
<code>WebRequest</code> , <code>NativeWebRequest</code>	Generic access to request parameters and request and response. It delegates to the underlying request object for direct use of the Servlet API.
<code>jakarta.servlet.ServletRequest</code> , <code>jakarta.servlet.ServletResponse</code>	Choose any specific request or response type — for example, <code>HttpServletRequest</code> , <code>HttpServletResponse</code> , or Spring's <code>MultipartRequest</code> , <code>MultipartResponse</code> .
<code>jakarta.servlet.http.HttpSession</code>	Enforces the presence of a session. As a consequence, such as <code>null</code> . Note that session access is not thread-safe. Consider <code>HttpSessionWrapper</code> for thread-safe access.

Spring Docs, Method Arguments,

<https://docs.spring.io/spring->

[framework/reference/web/webmvc/mvc-controller/ann-methods/arguments.html](https://docs.spring.io/spring-framework/reference/web/webmvc/mvc-controller/ann-methods/arguments.html)

파라미터란?

- 파라미터는 사용자가 HTTP 요청을 보낼 때, 함께 전달하는 값이다.
- 파라미터는 사용자 요청을 처리하는 데 사용된다.
- 컨트롤러는 파라미터를 받고, 타입캐스팅도 자동으로 처리한다.

파라미터 종류

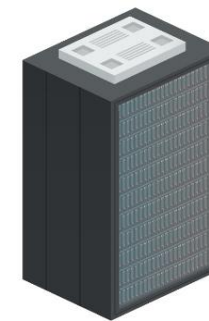
- 쿼리 파라미터: URL 주소에 파라미터를 담아서 전달한다.
- HTML Form: HTML 입력 폼에 데이터를 입력하여 전달한다.
- Request Body : 메시지 본문에 데이터를 직접 담아서 전달한다.

1번 게시물을 보내줘!



클라이언트

http:localhost:8080/get? no=1



서버

Request Param

- 사용자는 URL 주소에 쿼리 문자열형태로 파라미터를 담아서 전달한다.
- 이때 파라미터는 "변수=값" 형태로 작성한다.
- ? 기호로 시작하고 각 파라미터는 &로 구분한다.
- 서버는 @RequestParam 어노테이션을 사용하여 파라미터를 받는다.

보내는 쪽

URL: localhost:8080/ex? i=100



파라미터

i라는 변수에 값100을 담아서 전달

받는 쪽

```
@GetMapping("/ex")  
public ResponseEntity ex(@RequestParam int i) { }
```

Path Variable

- 사용자는 URL 주소 중간에 파라미터를 담아서 전달한다.
- 이때 {변수이름} 형태로 파라미터 위치를 지정한다.
- 서버는 @PathVariable 어노테이션을 사용하여 파라미터를 받는다.

보내는 쪽

localhost:8080/ex/100

└─┬─┘
파라미터

값 100을 전달

받는 쪽

```
@GetMapping("/ex/{i}")  
public ResponseEntity ex(@PathVariable int i) { }
```

ModelAttribute

- String, int 같은 단순한 타입은 RequestParam을 사용하고, 객체는 ModelAttribute를 사용한다.
- 복합적인 데이터를 전달할 때 주로 사용한다. (회원정보, 주문정보)
- ModelAttribute는 생략할 수 있다.

보내는 쪽

localhost:8080/ex?title=자바프로그래밍입문&publisher=한빛컴퍼니&price=1000

받는 쪽

```
@PostMapping("/ex")  
public ResponseEntity ex(@ModelAttribute BookDTO dto) { }
```

HTML Form

- 사용자는 HTML 폼에 데이터를 입력하여 전달한다.
- 데이터는 메시지 바디에 쿼리 파라미터 형식으로 전달된다. (username=둘리&age=15)

보내는 쪽

username: age:

받는 쪽

```
@PostMapping("/register")
public void ex2( @NotNull HttpServletRequest request, @NotNull HttpServlet
    String username = request.getParameter( name: "username");
    int age = Integer.parseInt( s: request.getParameter( name: "age"));
```

×	Headers	Payload	Preview
▼Form Data view source			
username: 둘리			
age: 15			

Request Body

- 사용자는 메시지 바디에 직접 데이터를 담아서 전달한다.
- 주로 POST, PUT 요청시 사용한다.
- 데이터형식으로 주로 JSON, XML, TEXT를 사용한다.
- 서버는 @RequestBody 어노테이션을 메시지에서 데이터를 꺼낸다.
- 이때 메시지 컨버터가 JSON 데이터를 객체로 변환해준다.

보내는 쪽

localhost:8080/ex +

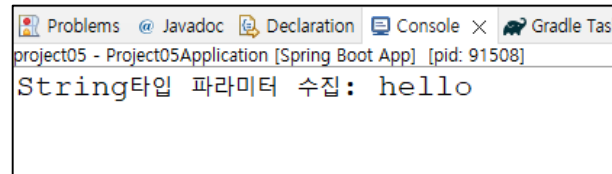
```
... "title": "자바프로그래밍입문",  
... "publisher": "한빛컴퍼니",  
... "price": 20000
```

도서 정보를 전달

받는 쪽

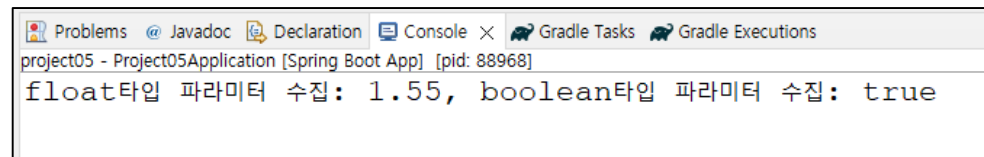
```
@PostMapping("/ex")  
public ResponseEntity ex(@RequestBody BookDTO dto) { }
```

Q1. get방식 + /param/q1 주소로 연결되는 메소드를 추가하세요.
문자열 파라미터를 수집하세요.



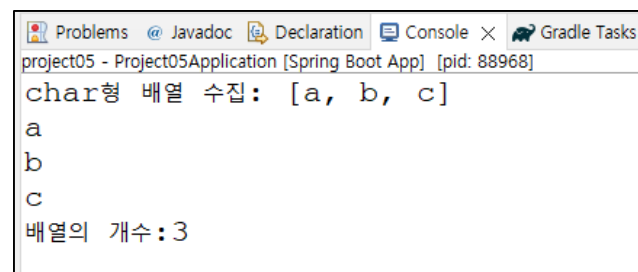
```
project05 - Project05Application [Spring Boot App] [pid: 91508]
String타입 파라미터 수집: hello
```

Q2. get방식 + /param/q2 주소로 연결되는 메소드를 추가하세요.
실수 파라미터와 논리형 파라미터를 수집하세요.



```
project05 - Project05Application [Spring Boot App] [pid: 88968]
float타입 파라미터 수집: 1.55, boolean타입 파라미터 수집: true
```

Q3. get방식 + /param/q3 주소로 연결되는 메소드를 추가하세요.
문자형 배열 파라미터를 수집하세요. 그리고 배열의 요소를 하나씩 출력하고, 배열의 크기를 출력하세요.



```
project05 - Project05Application [Spring Boot App] [pid: 88968]
char형 배열 수집: [a, b, c]
a
b
c
배열의 개수: 3
```

Q4. post방식 + /param/q4 주소로 연결되는 메소드를 추가하세요.

학생정보 파라미터를 수집하세요.

```
Problems @ Javadoc Declaration Console x Gradle Tasks Gradle Executions
project05 - Project05Application [Spring Boot App] [pid: 88968]
객체 수집: StudentDTO (no=1, name=둘리, grade=3)
```

Q5. post방식 + /param/q5 주소로 연결되는 메소드를 추가하세요.

학생 배열 파라미터를 수집하세요.그리고 배열의 요소를 하나씩 출력하고, 배열의 크기를 출력하세요.

```
Problems @ Javadoc Declaration Console x Gradle Tasks Gradle Executions
project05 - Project05Application [Spring Boot App] [pid: 88968]
객체타입 리스트 수집: [StudentDTO (no=1, name=둘리, grade=3), StudentDTO (no=2,
StudentDTO (no=1, name=둘리, grade=3)
StudentDTO (no=2, name=또치, grade=1)
StudentDTO (no=3, name=도우너, grade=2)
리스트의 개수: 3
```

Q6. post방식 + /param/q6 주소로 연결되는 메소드를 추가하세요.

자동차 리스트 파라미터를 수집하세요. 그리고 리스트의 마지막 요소를 출력하세요.

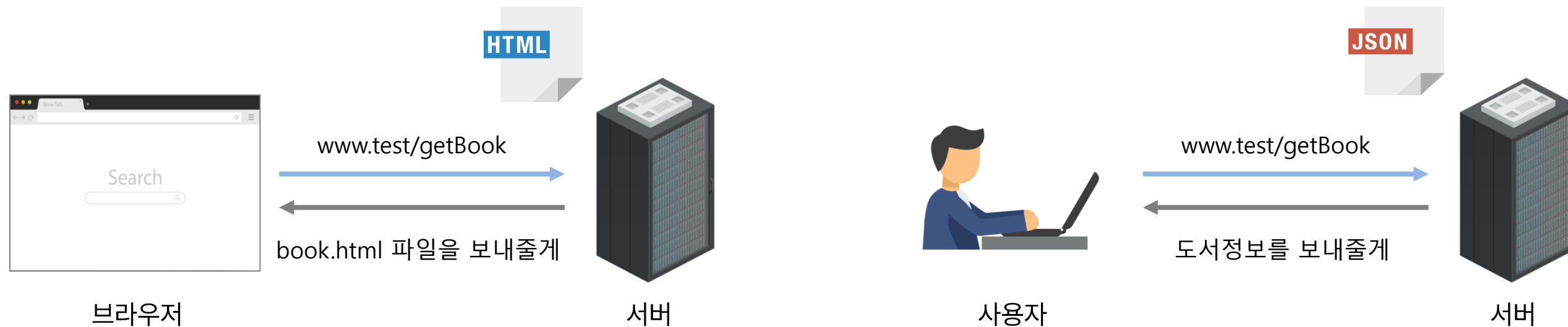
```
Problems @ Javadoc Declaration Console x Gradle Tasks Gradle Executions
project05 - Project05Application [Spring Boot App] D:\0.tool\spring-tool-suite-4-4.19.0.RELEASE-e4.28.0-win32.win32.x86_64.self-extractin
객체타입 리스트 수집: [CarDTO (company=현대, model=코나, color=블랙), Ca
CarDTO (company=현대, model=코나, color=블랙)
CarDTO (company=기아, model=k3, color=블루)
CarDTO (company=쉐보레, model=스파크, color=화이트)
리스트 마지막 요소: CarDTO (company=쉐보레, model=스파크, color=화이트)
```

서버의 응답

- 서버는 사용자에게 응답 데이터를 HTML이나 순수한데이터 형태로 보낸다.
- 컨트롤러 메소드의 반환타입을 설정해서 응답 데이터의 형태를 지정한다.

클라이언트의 처리

- 사용자는 응답 받은 데이터는 브라우저 또는 별도의 프로그램을 통해서 사용한다.

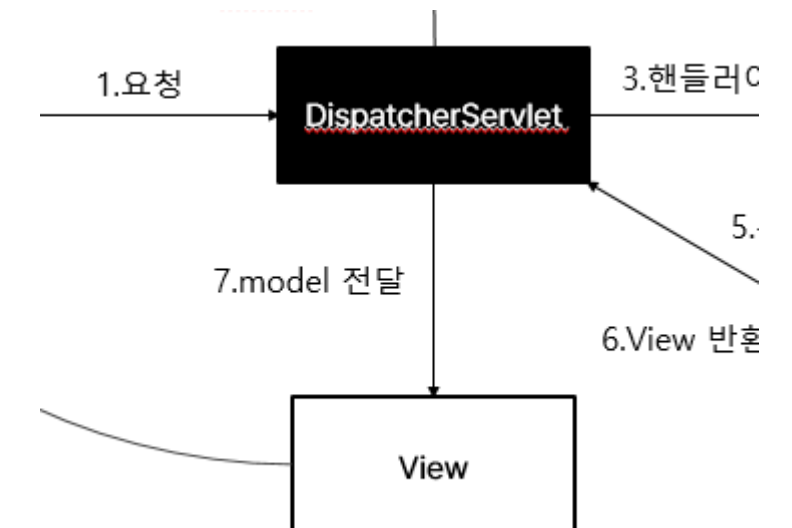


Model과 ModelAndView

- 둘 다 컨트롤러에서 뷰로 데이터를 전달하기 위해 사용된다.
- Model은 데이터만 저장하고, ModelAndView는 데이터와 뷰 이름을 함께 저장한다.

Model과 ModelAndView의 차이점

- Spring 초기에는 ModelAndView를 사용했지만, 이후 버전이 올라가면서 Model이 도입되었다.
- Model이 ModelAndView 보다 직관적이어서 더 많이 사용된다.



ModelAndView 사용

```

@RequestMapping("/example")
public String handleRequest(Model model) {

    model.addAttribute("message", "Hello, Model!"); // 모델 데이터 설정
    return "exampleView"; // 뷰 이름 반환
}
  
```

Model 사용

```

@RequestMapping("/example")
public ModelAndView handleRequest() {

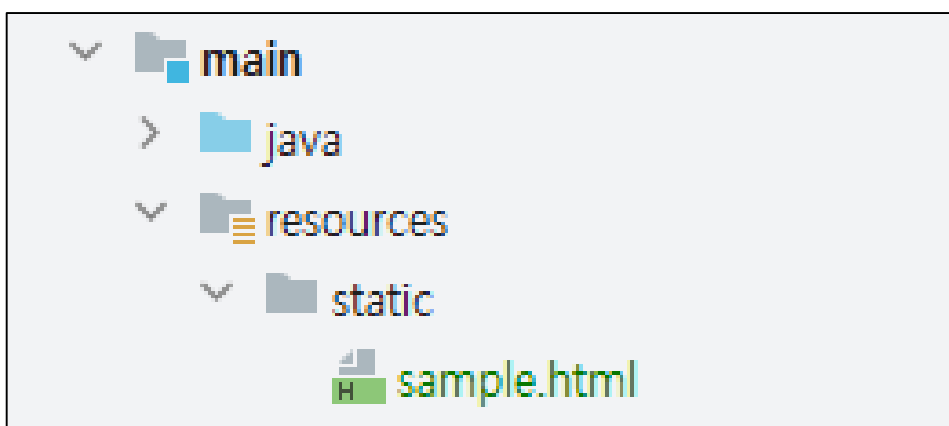
    ModelAndView modelAndView = new ModelAndView();

    modelAndView.setViewName("exampleView"); // 뷰 이름 설정
    modelAndView.addObject("message", "Hello, ModelAndView!"); // 모델 데이터 설정
    return modelAndView;
}
  
```

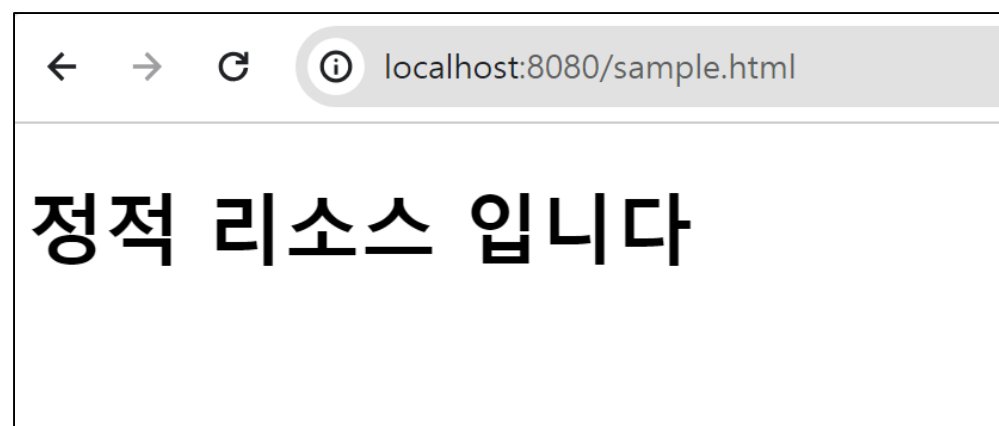
정적 리소스

- 정적인 HTML, CSS, JS을 보낼 때는 정적 리소스를 사용한다.
- 스프링 부트는 static 폴더에 정적 리소스를 저장한다.
- 폴더에 HTML파일을 저장한 후에, 바로 화면에서 호출 할 수 있다.

정적 리소스 경로



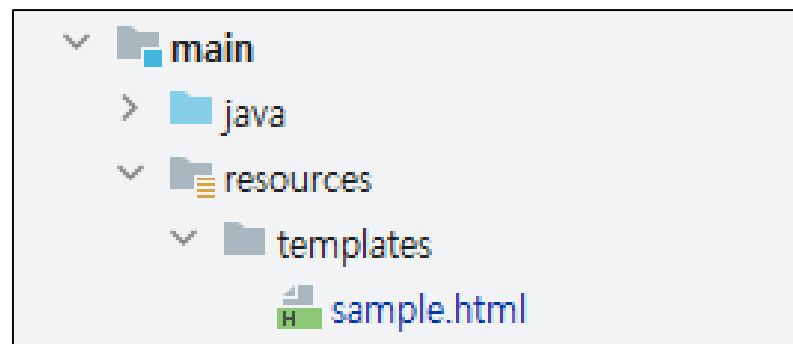
http://localhost:8080/sample.html



뷰 템플릿

- 동적인 HTML을 보낼 때는 뷰 템플릿을 사용한다.
- 타임리프와 같은 뷰 템플릿을 사용할 수 있다.
- 스프링 부트는 template 폴더에 뷰 템플릿을 저장한다.

뷰 템플릿 경로



application.properties

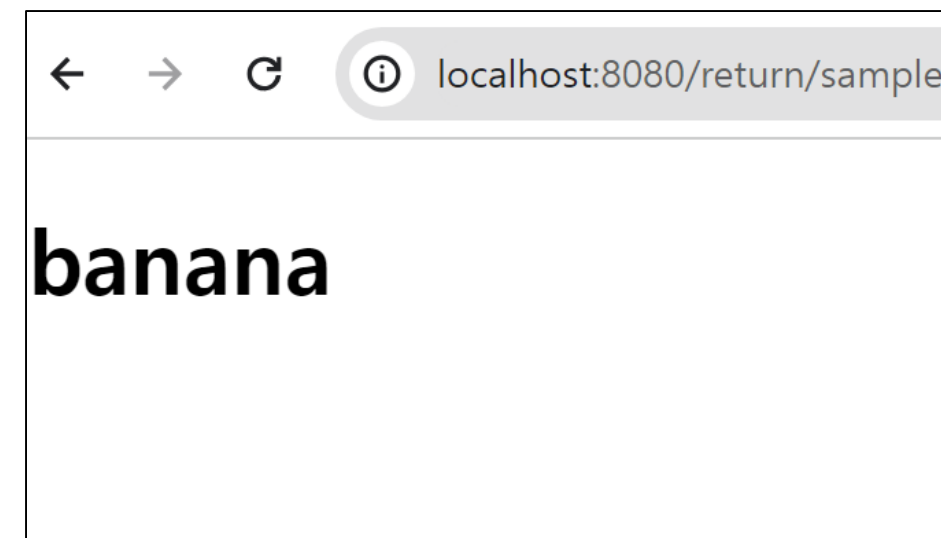
```
spring.thymeleaf.prefix=classpath:/templates/  
spring.thymeleaf.suffix=.html
```

스프링부트의 기본값이므로 생략 가능

void, String

- void: URL 경로를 그대로 HTML 파일 경로가 된다
- String: HTML 파일 경로를 직접 지정한다

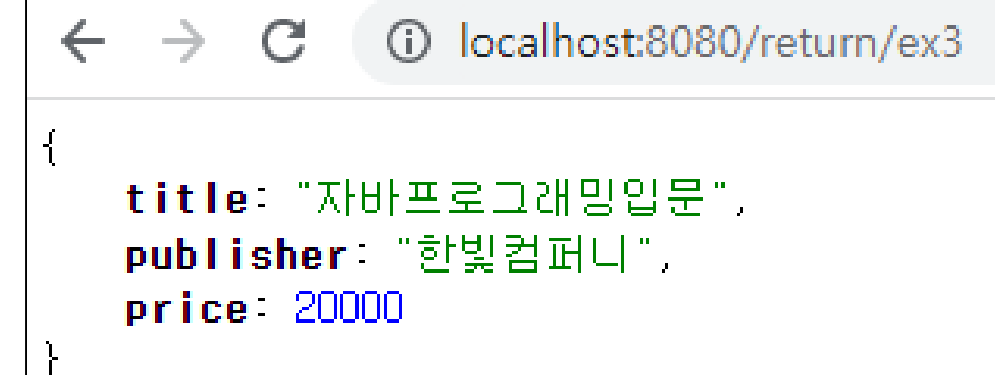
```
public ModelAndView ex1() {  
    ModelAndView modelAndView = new ModelAndView("return/sample")  
                                .addObject("data", "banana");  
    return modelAndView;  
}  
  
public String ex2(Model model) {  
    model.addAttribute("data", "banana");  
    return "return/sample";  
}  
  
@GetMapping("/return/sample")  
public void ex3(Model model) {  
    model.addAttribute("data", "banana");  
}
```



ResponseBody

- @ResponseBody 어노테이션은 순수한 데이터를 보낼 때 사용한다.
- 메시지 바디에 데이터를 직접 담을 수 있다.
- 객체를 JSON타입으로 변환하여 보낸다.

```
@ResponseBody
@GetMapping("/ex3")
public BookDTO ex3() {
    BookDTO bookDto = new BookDTO("자바프로그래밍입문", "한빛컴퍼니", 20000);
    return bookDto;
}
```



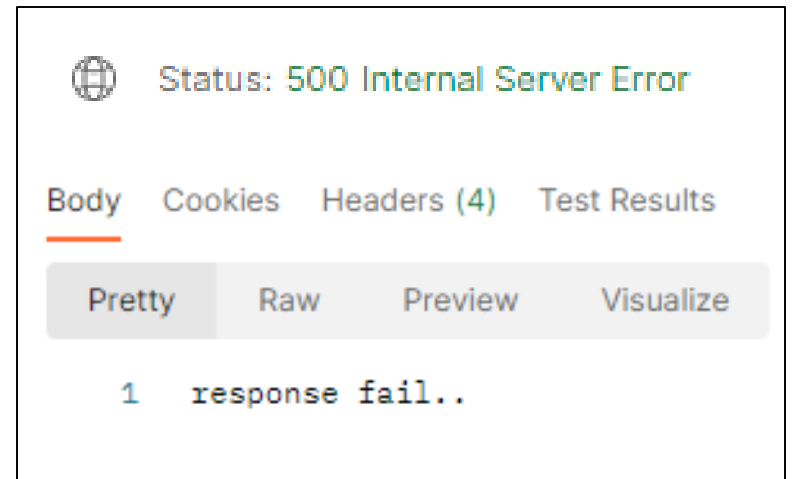
```
{
  title: "자바프로그래밍입문",
  publisher: "한빛컴퍼니",
  price: 20000
}
```

ResponseEntity

- ResponseEntity를 사용하면 HTTP 상태코드와 데이터를 직접 설정할 수 있다.
- 요청 처리에 실패했을 때, 에러코드와 실패메세지를 함께 보낼 수 있다.

500 에러코드와 실패메세지 전달

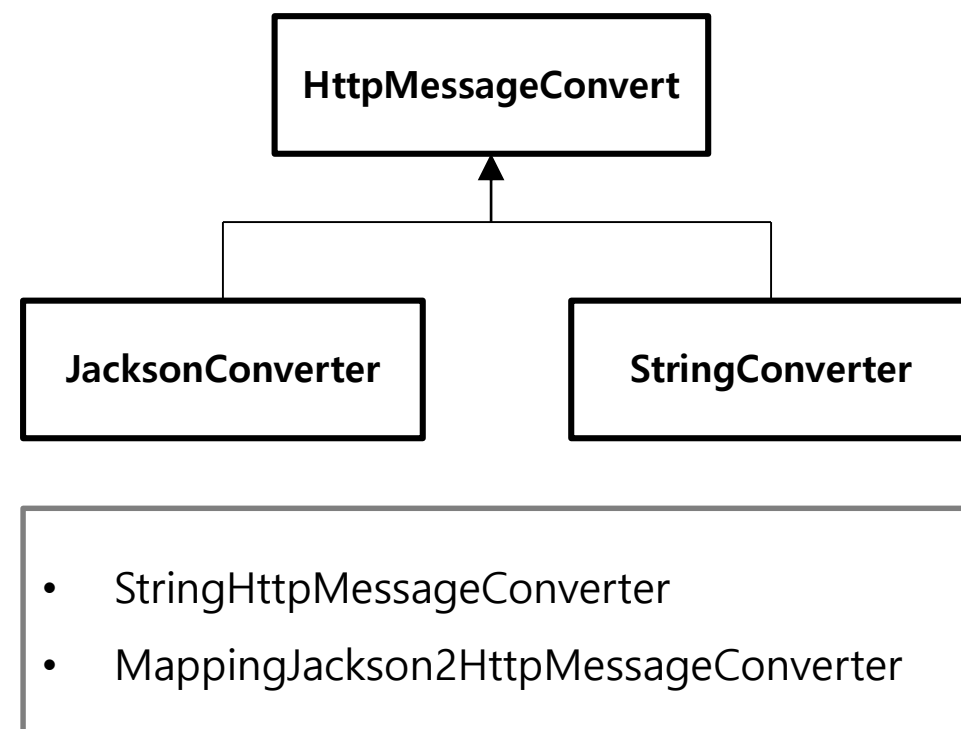
```
@GetMapping("/q6")
public ResponseEntity<String> quiz6() {
    return new ResponseEntity<>("response fail..", HttpStatus.INTERNAL_SERVER_ERROR);
}
```



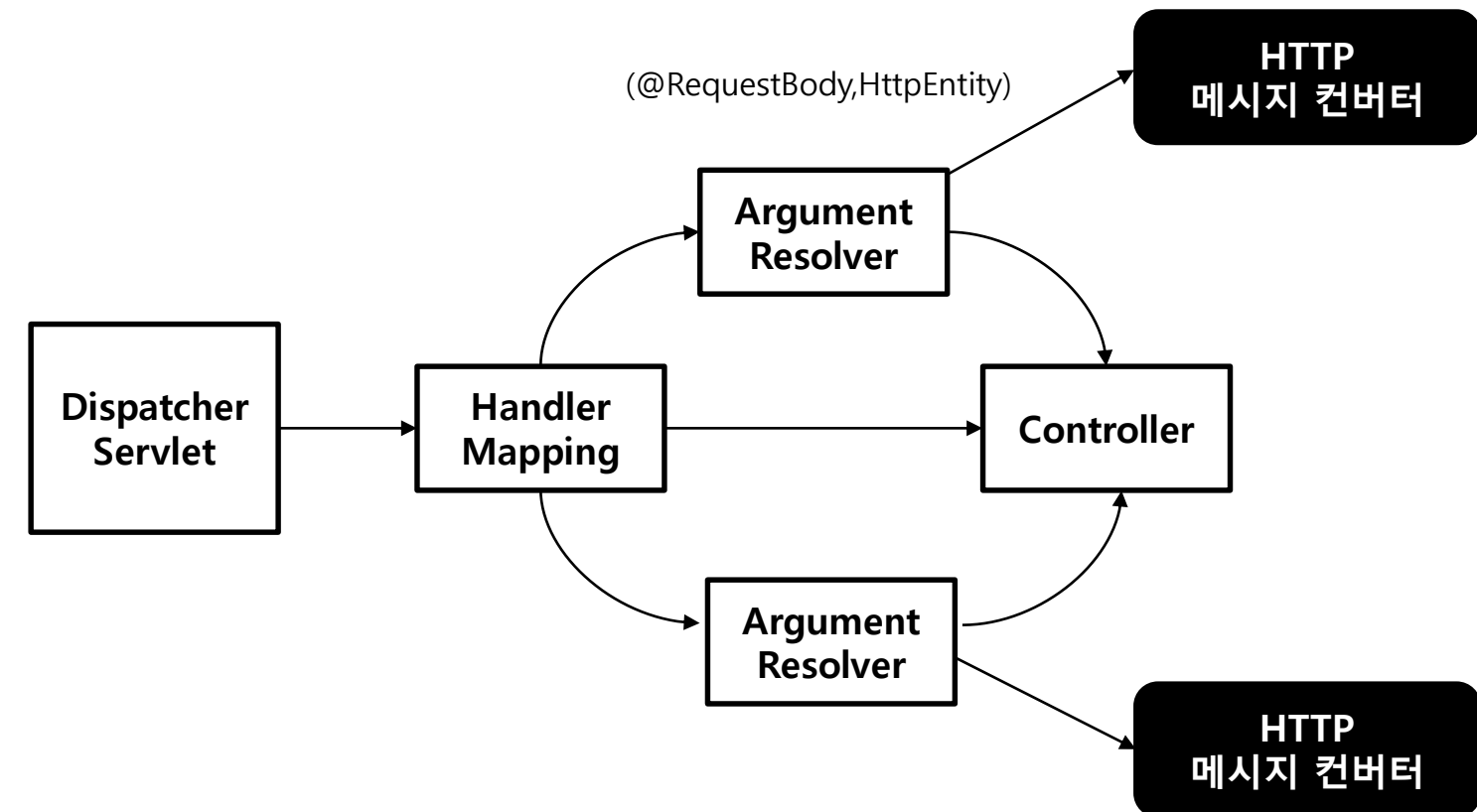
HTTP 메시지 컨버터

- 컨트롤러에서 `@ResponseBody` 또는 `HttpEntity`를 사용하면 메시지 컨버터가 적용된다
- 응답 데이터가 `String`, `int` 같은 기본문자이면 `StringConverter`를 사용한다.
- 응답 데이터가 클래스, `HashMap` 같은 객체타입이면 `JacksonConverter`를 사용한다.
- 메시지 컨버터는 데이터를 변환하고, 메시지 바디에 담아준다.

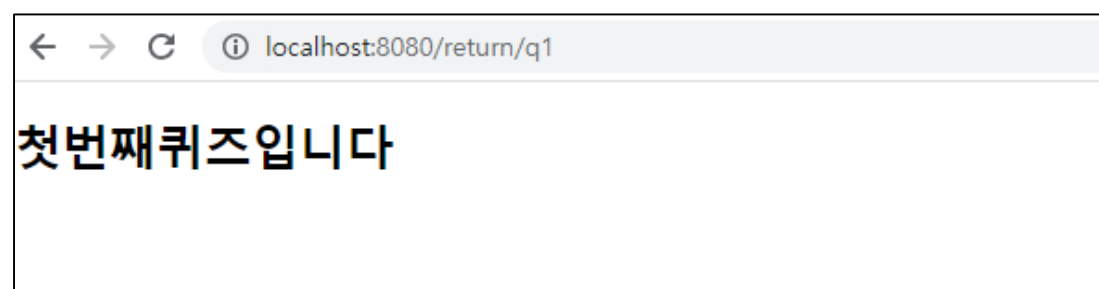
메시지 컨버터 종류



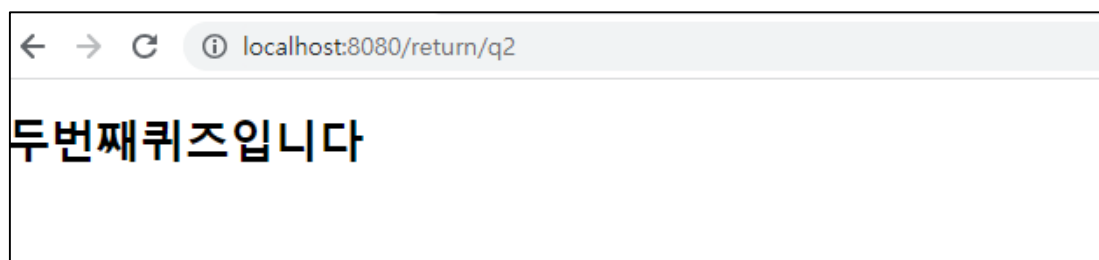
메시지 컨버터 위치



Q1. localhost:8080/return/q1 주소로 요청이 들어오면 q1.html 파일을 반환하세요.



Q2. localhost:8080/return/q2 주소로 요청이 들어오면 test.html 파일을 반환하세요.



Q3. localhost:8080/return/q3 주소로 요청이 들어오면 학생정보를 반환하세요.

먼저 학생클래스를 만드세요. (속성: 번호, 이름, 학년)

```
localhost:8080/return/q3
{
  "no": 1,
  "name": "둘리",
  "grade": 3
}
```

Q4. localhost:8080/return/q4 주소로 요청이 들어오면 자동차정보를 반환하세요.

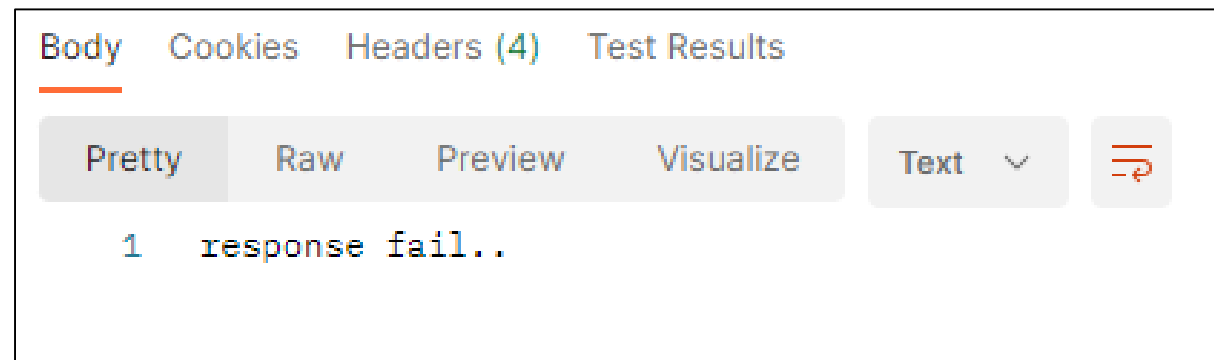
먼저 자동차클래스를 만드세요. (속성: 제조사, 모델, 색)

```
localhost:8080/return/q4
{
  "company": "현대",
  "model": "코나",
  "color": "블랙"
}
```

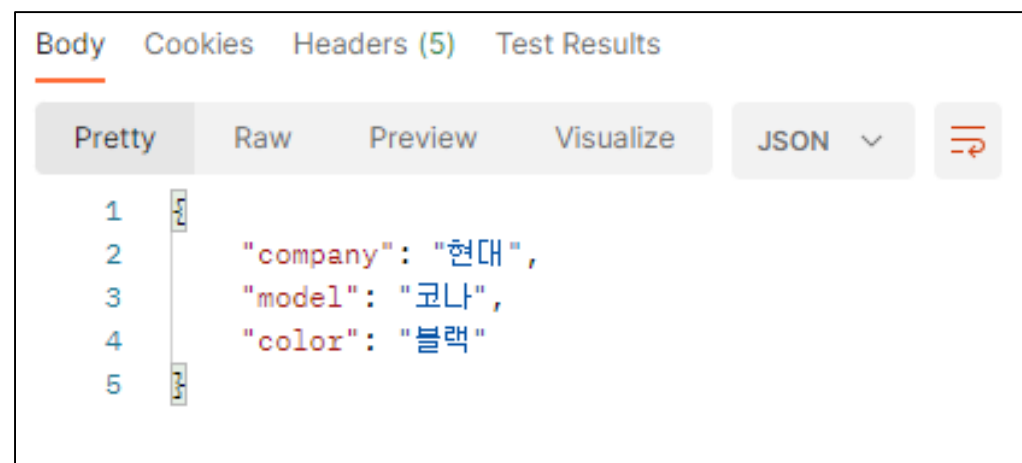
Q5. localhost:8080/return/q5 주소로 요청이 들어오면 학생 목록을 반환하세요.

```
localhost:8080/return/q5
[
  {
    "no": 1,
    "name": "둘리",
    "grade": 3
  },
  {
    "no": 2,
    "name": "또치",
    "grade": 1
  },
  {
    "no": 3,
    "name": "도우너",
    "grade": 2
  }
]
```

Q6. localhost:8080/return/q6 주소로 요청이 들어오면 ResponseEntity를 이용하여 500 에러코드와 "response fail.." 메시지를 반환하세요.



Q7. localhost:8080/return/q7 주소로 요청이 들어오면 ResponseEntity를 이용하여 200 응답코드와 자동차정보를 반환하세요.



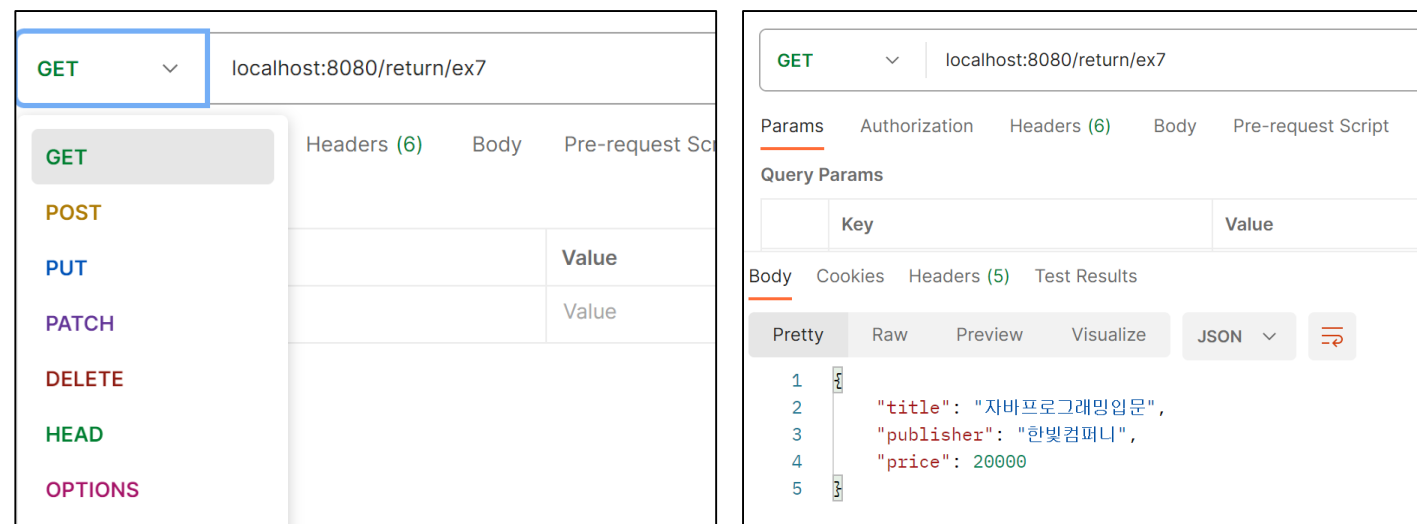
Postman이란?

- Postman은 컨트롤러를 테스트 하기 위한 도구이다.
- Postman은 브라우저의 개발자도구와 비교했을 때 몇가지 장점을 가지고 있다.



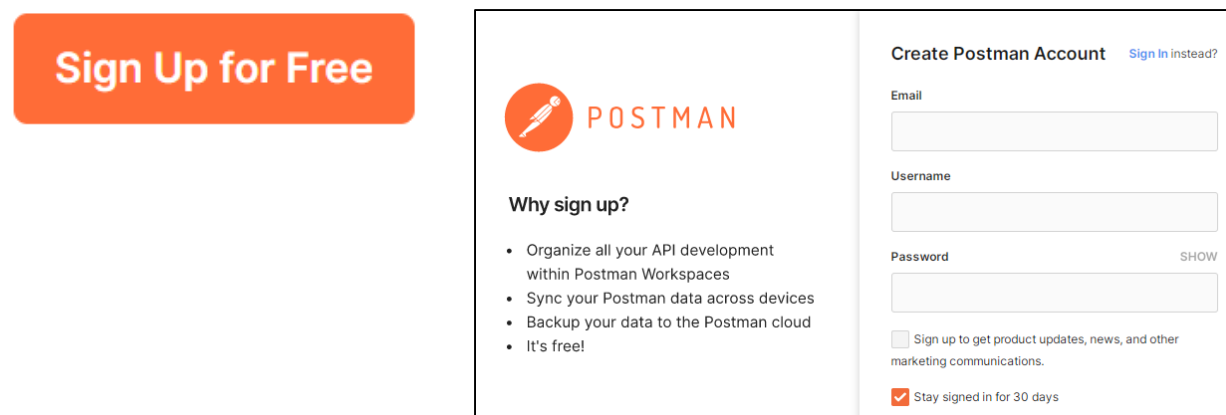
Postman의 기능

- HTTP 요청을 쉽게 구성할 수 있다.
(요청방식, 파라미터, 헤더)
- 요청메시지와 응답메시지를 한번에 볼 수 있다.

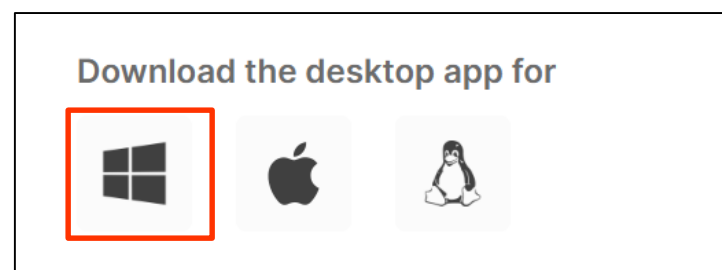


포스트맨 설치 방법

1. 포스트맨 사이트에 접속해서 회원가입을 진행한다.



2. 윈도우 버전 설치파일을 다운로드하고, 설치를 진행한다.

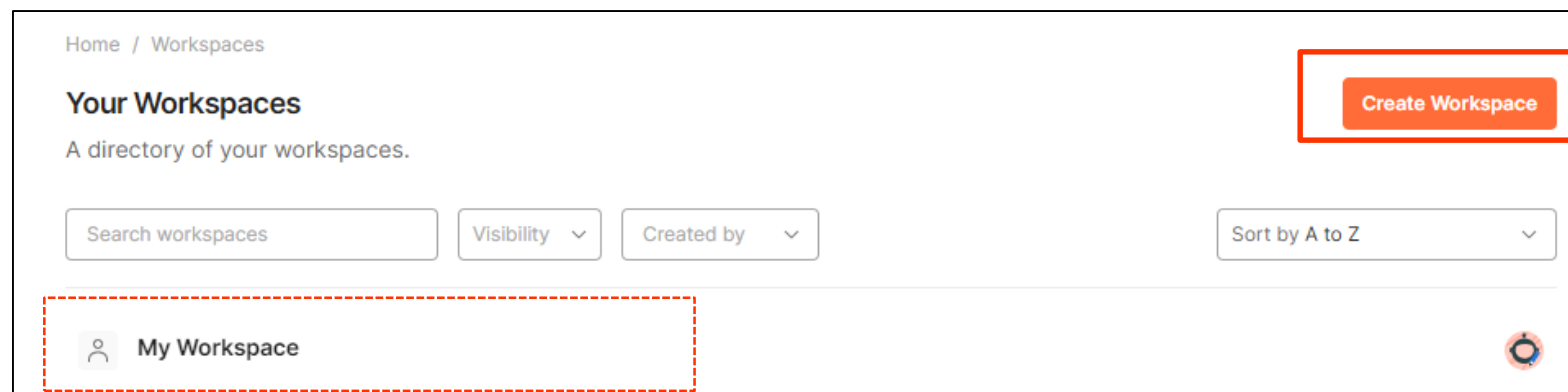


포스트맨 사용 방법

1. 왼쪽 사이드바에서 [workspaces] 메뉴를 클릭한다.



2. [create workspace] 버튼을 클릭하여 작업공간을 생성한다.



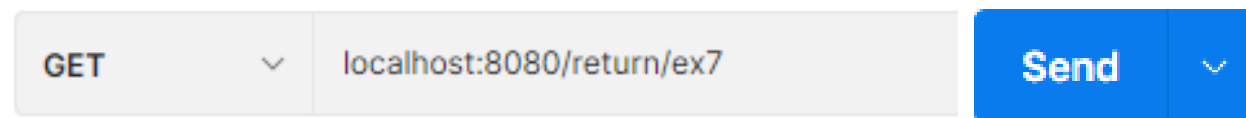
3. [+] 버튼을 스프링6장 폴더를 생성한다.



4. [Add a request] 버튼을 클릭한다.

This collection is empty
[Add a request](#) to start working.

5. URL을 입력하고 [Send] 버튼 클릭한다.



6. 호출 결과를 확인한다.



테스트가 끝나면 제목을 작성하고 [Ctrl + S] 키를 눌러 테스트 내용을 저장한다.

다시 테스트를 진행할 때 재사용할 수 있다.