**스프링 웹 개발 기초**

* **정적 컨텐츠**
* **MVC와 템플릿 엔진**
* **API**

**📒 정적 컨텐츠**

서버에서 하는 거 없이 **파일 그대로를** 웹브라우저로 내려주는 것을 **정적 컨텐츠**라고 한다.

**🖋️ 스프링 부트 정적 컨텐츠 기능**

**정적 컨텐츠**는 src/main/resources/**static** 폴더에 **static.html** 이라는 새로운 파일을 만들어 html 코드를 아래와 같이 작성한다.

<!DOCTYPE HTML>

<html>

<head>

<title>static content</title>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

</head>

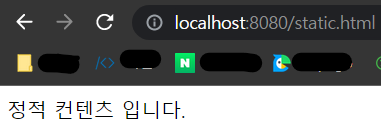
<body>

정적 컨텐츠 입니다.

</body>

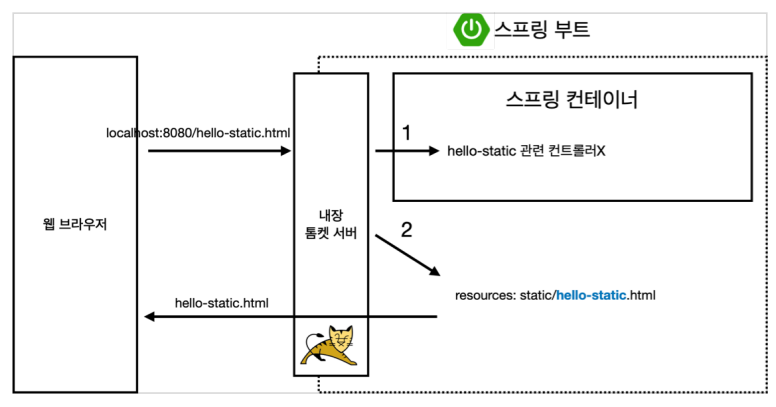
</html>

작성 후 프로젝트를 실행하여 주소 뒤에 새로 만든 파일명을 치고 웹브라우저에 들어가면 <http://localhost:8080/static.html> 아래 화면처럼 정상적으로 **정적 컨텐츠**가 보이는 것을 확인할 수 있다.



**🖋️ 스프링 부트 정적 컨텐츠 원리**

위에 실행한 프로젝트처럼 작동하는 원리는 아래 사진에 간단하게 설명되어 있다.



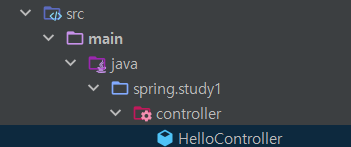
웹 브라우저에서 <http://localhost:8080/static.html> 주소에 들어가면 **내장 톰켓 서버**가 요청을 받고, static.html 을 스프링 부트가 **컨트롤러**에 static 관련 컨트롤러가 있는지 확인을 한다. 이 프로젝트는 컨트롤러 파일이 없기 때문에 넘어가고 그 다음으로 static 관련 **html** 이 있는 지 확인을 하여 찾으면 이 파일을 웹 브라우저에 바로 보내주는 것이 **정적 컨텐츠**의 원리이다.

**📒 MVC와 템플릿 엔진**

html을 서버에서 **동적**으로 프로그래밍 해서 웹브라우저로 내려주는 것을 **템플릿 엔진**이라고 하며, 템플릿 엔진을 사용하기 위한 **컨트롤러, 모델, 템플릿 엔진 화면** 이 3가지 **Model, View, Controller(MVC)** 라고 한다.  
**view**는 화면을 그리며, **model**과 **controller**는 비지니스 로직에 관련이 있거나 내부적인 것을 처리한다.

**🖋️ MVC (Model, View, Controller)**

기존에 만들어 두었던 src/main/java/spring.study1/**controller**에 있는 HelloController 파일에 새로운 **controller** 내용을 추가할 것이다.



웹 어플리케이션에서 localhost:8080/hello-mvc 로 들어오면 해당 메서드 호출하게 된다. 이때 **name이라는 이름으로 param**을 받아, attribute 의 key의 이름이 "name" 이라는 곳에 param로 받은 값을 **model**에 넣어 return을 해준다.

@GetMapping("hello-mvc")

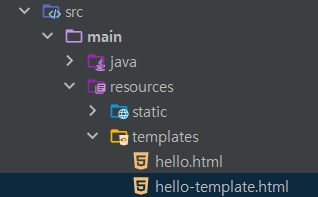
public String helloMvc(@RequestParam("name") String name, Model model) { // 외부에서 param 을 받음

model.addAttribute("name", name);

return "hello-template";

}

**controller**를 만들었으면 이제 그 화면에 대한 **templates**를 만들어야 하기 때문에 src/main/resources/**templates** 폴더에 안에 controller에서 return 받은 이름과 똑같은 **hello-template.html** 파일을 만들어 준다.



이 html에는 HelloController의 name의 param값이 ${name}에 들어간다.

<html xmlns:th="http://www.thymeleaf.org">

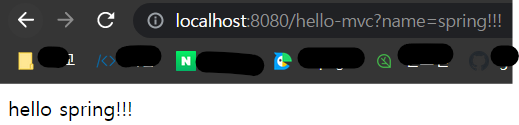
<body>

<p th:text="'hello ' + ${name}">hello! empty</p>

</body>

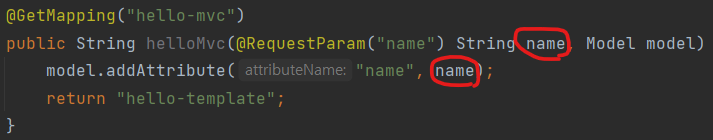
</html>

이 프로젝트를 실행하기 위해 name에 **param**값을 주면서 **hello-mvc**로 들어가야 한다.  
html에서 **GET**방식으로 **param**을 받을 수 있는 방법은 "**?key=value**" 이므로 <http://localhost:8080/hello-mvc?name=spring!!!> 에 들어가면 아래와 같은 화면이 뜬다.

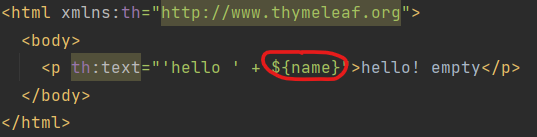


**🖋️ MVC (Model, View, Controller) 원리 설명**

다시 설명을 하면  
**hello-mvc?name=spring!!!** 에서 name의 **param**으로 받은 **spring!!!** 이 아래 코드에 있는 **name**의 자리에 들어가고, 이 값들이 **Model**에 담겨 **hello-template**로 넘어간다.

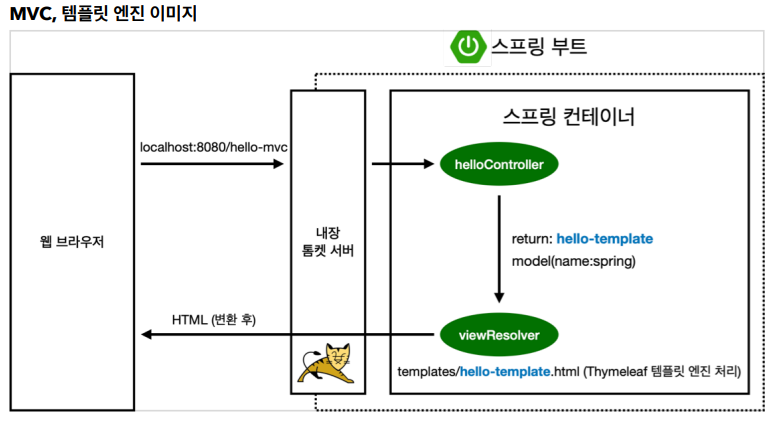


그러면 **template**에서 **Model**의 **Key 값**이 **name**인 것을 찾아, 그 **value**값을 **${name}** 에 치환을 해주면 그 자리에 **spring!!!** 이 들어가게 된다.



**🖋️ MVC (Model, View, Controller) 그림 설명**

위에 실행한 프로젝트처럼 작동하는 원리는 아래 사진에 간단하게 설명되어 있다.



웹 브라우저에 **localhost:8080/hello-mvc**를 넘기면 스프링 부트를 띄우기 전에 **내장 톰켓 서버**를 먼저 거친다. **내장 톰켓 서버**는 hello-mvc가 오면 스프링에게 알려주고, 스프링은 **helloController**에 hello-mvc가 **mapping** 되어 있는 것을 확인해 해당 메서드를 호출해준다. 메서드에서 **hello-template**를 **return** 해주고, **model**에 key는 **name**이고 값은 **spring!!** 이 담긴 것을 spring에게 넘겨주게 된다. 그러면 spring은 **view**를 찾아주고 **템플릿 엔진**을 연결해주는 **viewReslover**을 통해 return 받은 hello-template과 이름이 똑같은 파일을 찾아서 **thymeleaf**라는 템플릿 엔진에게 처리해달라고 넘긴다. 그럼 그 템플릿 엔진이 렌더링을 해 **변환**을 한 html을 웹브라우저에 반환을 한다.

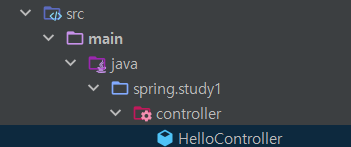
정적일 때에는 변환을 하지 않았지만, **템플릿 엔진**에서는 **변환**을 하고 웹브라우저에 넘겨준다.

**📒 API**

**json** 이라는 데이터 구조 포멧으로 클라이언트로 데이터를 전달하는 방식을 **API** 라고 한다. 또한 서버끼리 데이터를 구조받을 때 **API** 방식을 사용한다.

**🖋️ API 기초**

이번에도 기존에 만들어 두었던 src/main/java/spring.study1/**controller**에 있는 HelloController 파일에 새로운 **controller** 내용을 추가할 것이다.



웹 어플리케이션에서 localhost:8080/hello-string 로 들어오면 해당 메서드 호출하게 된다. 이때 name이라는 이름으로 **param**을 받아, 받은 값을 그대로 **return**을 해준다. 그러면 param으로 받은 name의 값이 그대로 **요청한 서버에 전달**이 되며, 이 값은 **http body**에 name의 값이 직접 들어가게 된다.

@GetMapping("hello-string")

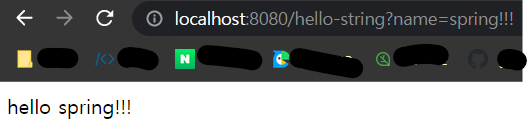
@ResponseBody // http의 body에 param으로 받은 data를 직접 넣어줌

public String helloString(@RequestParam("name") String name) {

return "hello " + name; // param으로 받은 게 그대로 요청한 서버에 바로 전달

}

그래서 바로 <http://localhost:8080/hello-string?name=spring!!!> 에 들어가게 되면 아래 화면과 같이 뜨게 된다.



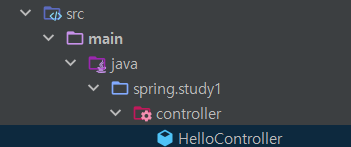
조금 더 자세히 보기 위해 **Ctrl + U** 를 눌러 **페이지 소스**를 보면 값이 그대로 웹 브라우저에 내려간 것을 확인할 수 있다.

https://velog.velcdn.com/images/easyhyun00/post/7749b04e-c12d-4b92-8d91-cdb7a050c39b/image.png

이처럼 API 방식은 템플릿 엔진과 다르게 template이 필요없이 값을 그대로 내보내 준다.

**🖋️ API 활용**

이번에는 값들을 **JSON 형식**으로 key, value 값으로 변환해 주는 방식을 사용할 것이다.  
위에와 똑같이 src/main/java/spring.study1/**controller**에 있는 HelloController 파일에 새로운 **controller** 내용을 추가할 것이다.



**hello-api**가 **mapping** 되어 있어 웹 어플리케이션에서 localhost:8080/hello-api 로 들어오면 해당 메서드 호출하게 되고 **@ResponseBody**가 있어 **http**의 **body**에 data를 직접 넣어준다. 그리고 이번엔 **JSON 형식**으로 data를 넘겨주기 위해 Hello 라는 새로운 **객체**를 만들어 그 **객체**에 값을 넣고 return 해준다.

@GetMapping("hello-api")

@ResponseBody // http의 body에 param으로 받은 data를 직접 넣어줌

public Hello helloApi(@RequestParam("name") String name) {

Hello hello = new Hello();

hello.setName(name);

return hello; // hello 객체를 return

}

static class Hello {

private String name; // private 라서 메서드를 사용해 활용

public String getName() { // getter

return name;

}

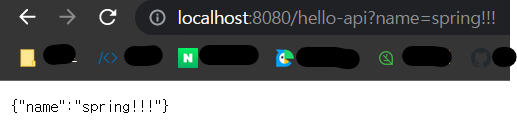
public void setName(String name) { // setter

this.name = name;

}

}

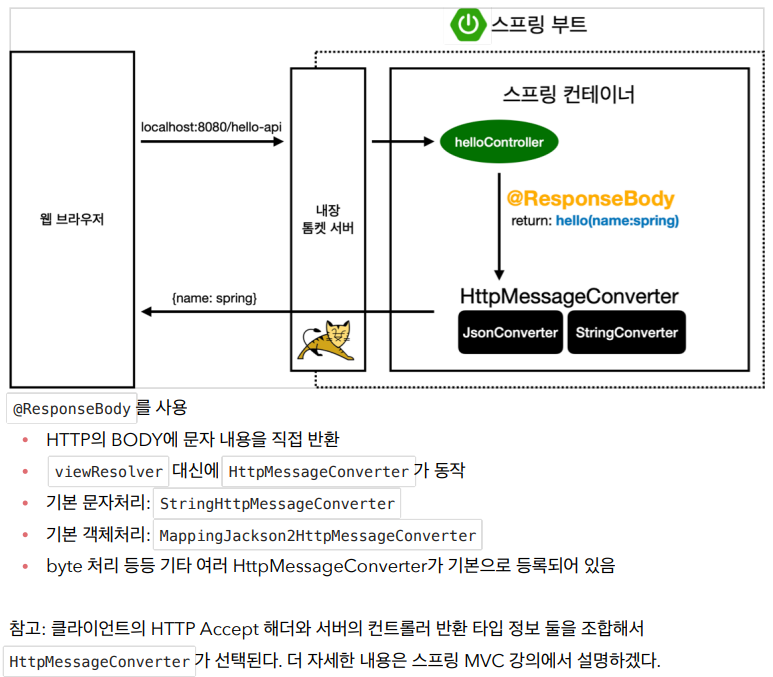
그래서 바로 <http://localhost:8080/hello-api?name=spring!!!> 에 들어가게 되면 아래 화면과 같이 뜨게 된다.



그러면 위에 사진과 같이 **{"key":"value"}** 처럼 **JSON 형식**으로 data가 웹 브라우저에 내려간 것을 확인할 수 있다.

**🖋️ API 원리**

위에 실행한 프로젝트처럼 작동하는 원리는 아래 사진에 간단하게 설명되어 있다.



웹 브라우저에 localhost:8080/hello-api를 타이핑하면 **내장 톰켓 서버**에서 **hello-api**를 spring으로 보내준다. spring은 hello-api가 mapping 되어 있는 것을 확인하고 또한 거기에 **@ResponseBody**가 있어 http 응답에 **data를 그대로** 넘겨주는 동작을 하게 된다.  
그러나 바로 위 프로젝트에서는 **객체**를 넘겨주기 때문에 data 그대로가 아닌 **JSON 방식**으로 만들어서 http 응답에 반환을 해야한다.  
즉, **@ResponseBody**가 있으면 **HttpMessageConverter**가 동작을 하고, **객체**가 return 됐기 때문에 **JsonConverter**가 동작하게 되어 객체를 **JSON 형태**로 바꾸게 되어 **JSON 형태**로 응답을 하게 된다.