스프링 웹 개발 기초

Tags	API MVC 정적 컨텐츠 템플릿 엔진
학습일	@Jan 17, 2021

- 웹 개발의 3가지 기본 방식

1. 정적 컨텐츠

: 서버에서 하는 일 없이, **파일을 그대로** 웹브라우저로 내려주는 방식

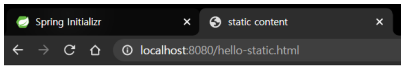
- 스프링 부트에서 기본적인 정적 컨텐츠 기능을 제공

1. 코드 작성(HTML)

```
<!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>
```

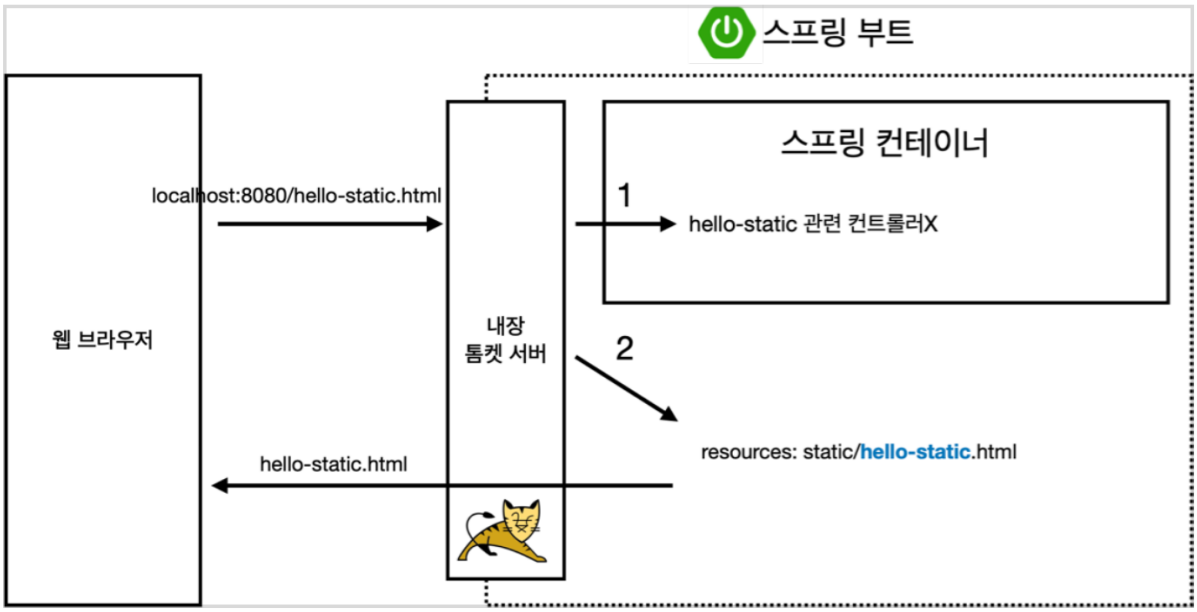
2. 실행

Application을 실행 한 후 `http://localhost:8080/hello-static.html` 에 접속하면 아래와 같은 화면이 출력된다.



정적 컨텐츠 입니다.

3. 원리



Controller가 우선순위를 가지고, 존재하지 않은 경우, static에서 html 파일을 찾는다.

2️⃣MVC와 템플릿 엔진

: **HTML 파일**을 프로그래밍해서, **동적으로 변환**하여 내보내는 방식. Model, View, Controller를 합쳐서 MVC라고 한다. JSP, PHP 등이 소위 말하는 템플릿 엔진. 최근 많이 쓰는 방식 ✔️

- **MVC**: Model, View, Controller
  - Model, Controller: 비즈니스 로직과 관련, 내부적으로 무언갈 처리하는 역할
  - View: 화면을 그리는 역할
  - 과거에는 Controller와 View가 분리되지 않고, View에서 모든 작업을 진행했어야 했다.

1. 코드 작성

```
package hello.hellospring.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class HelloController {
    @GetMapping("hello")
    public String hello(Model model) {
        model.addAttribute("data", "hello!!");
        return "hello";
    }
```

```
    @GetMapping("hello-mvc")
    public String helloMvc(@RequestParam(value = "name", required = false) String name, Model model) {
        model.addAttribute("name", name);
        return "hello-template";
    }
}
```

```
<html xmlns:th="http://www.thymeleaf.org">
<body>
<p th:text="'hello ' + ${name}">hello! empty</p>
</body>
</html>
```

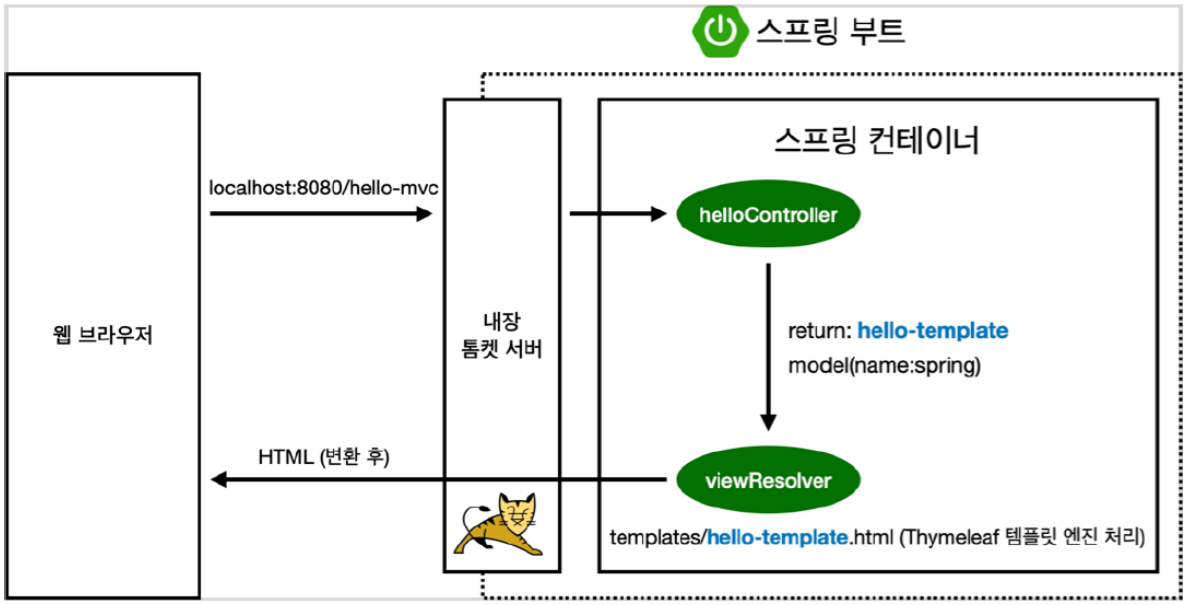
- ${name} : 모델에서 key 값이 name인 것을 찾아와라

2. 실행

- http://localhost:8080/hello-mvc?name=heejin
- HTTP GET 방식에서 파라미터 넘겨주는 방식

3. 원리

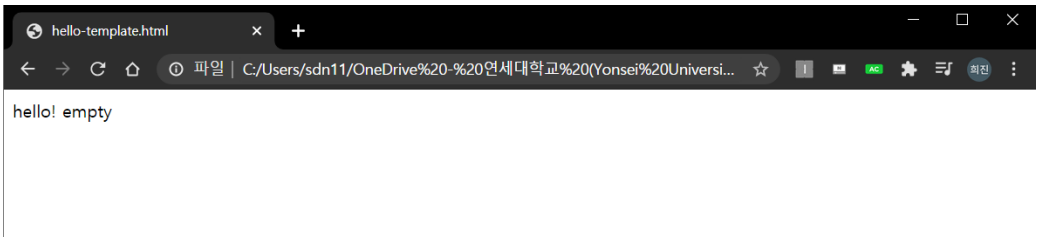
MVC, 템플릿 엔진 이미지



스프링 부트를 띄울 때 같이 띄우는 내장 톰캣 서버를 먼저 거친다. 스프링은 **helloController**라는 method에 mapping이 되어있다는 것을 발견하고, 이는 hello-template을 return, model에는 name(key):spring(value)를 넘겨준다. 이는 화면과 관련있는 **viewResolver**로 연결되는데, 이에 맞는 view를 찾아주고 template engine을 연결시켜주는 역할이다. return 값과 똑같은 이름을 찾아서 thymeleaf 템플릿 엔진에 처리를 넘긴다. 템플릿 엔진은 이를 렌더링 해서 변환을 한 HTML 을 웹 브라우저에 전달. (static은 변환을 하지 않는다!)

> thymeleaf의 장점

: 서버 없이 해당 파일을 그냥 열어봐도, 파일의 껍데기를 볼 수 있다. `hello-template.html` 의 absolute path를 주소창에 호출하면, <p> 태그 안의 값이 보여진다. 서버를 통해 thymeleaf를 실행하면, 태그 내부 값이, 태그에 지정된 값으로 **치환(변환)되어** 보여진다.



hello-template.html의 절대 경로 [IntelliJ상의 파일에서 우클릭 - Copy Path - Absolute Path]

3️⃣API

: **json**이라는 데이터 포맷으로 클라이언트에게 데이터를 전해주는 방식. API가 **데이터를 내려주고**, 화면은 클라이언트가 알아서 정리하는 방식을 주로 사용한다. 서버끼리 통신할 때는 HTML이 필요하지 않고, 어떤 데이터가 오가는지가 중요하기 때문에, API 방식을 사용한다.

1. 공통 특징
   - `@ResponseBody` 를 사용하면 뷰 리졸버(viewResolver)를 사용하지 않음. 대신에 `HttpMessageConverter` 가 동작한다.
   - HttpMessageConverter의 결과를 HTTP의 BODY에 직접 반환~~(HTML BODY TAG를 말하는 것이 아님)~~
2. String_문자를 내려주는 방식
   - HTTP 응답에 **문자가 그대로** 내려가는 방식.
   - 기본 문자 처리 : StringHttpMessageConverter
3. Json_데이터를 내려주는 방식 (⭐이게 바로 진짜 API 방식)
   - **객체**를 전달한다. Spring에선 기본 default가 **json 방식**으로 데이터를 만들어서 HTTP 응답에 반응하도록 한다.
   - 기본 객체 처리 : MappingJackson2HttpMessageConverter
     - Jackson : 객체를 json 타입으로 변환해주는 대표적인 라이브러리 (spring의 default)
     - GSON : 구글에서 만든, 객체 → json 타입 변환 라이브러리

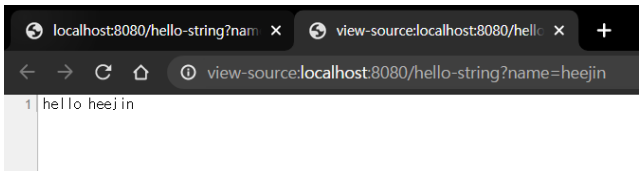
1. 코드 작성_String

```java
package hello.hellospring.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
```

```java
@Controller
public class HelloController {
    @GetMapping("hello")
    public String hello(Model model) {
        model.addAttribute("data", "hello!!");
        return "hello";
    }

    @GetMapping("hello-mvc")
    public String helloMvc(@RequestParam(value = "name", required = false) String name, Model model) {
        model.addAttribute("name", name);
        return "hello-template";
    }

    @GetMapping("hello-string")
    @ResponseBody
    public String helloString(@RequestParam("name") String name) {
        return "hello " + name;
    }
}
```

2. 실행_String

- http://localhost:8080/hello-string?name=heejin

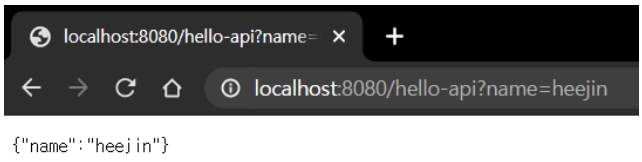


페이지 소스를 봐도, 문자 그대로만 보이는 것이 API의 특징

3. 코드_json

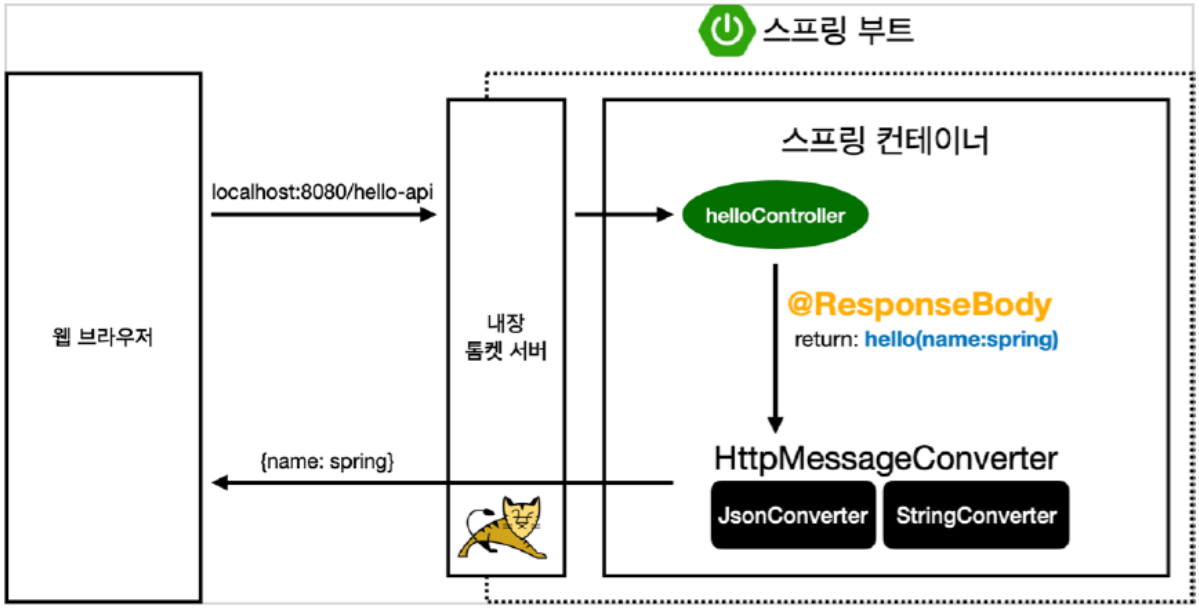
```java
@GetMapping("hello-api")
    @ResponseBody
    public Hello helloApi(@RequestParam("name") String name){
        Hello hello = new Hello();
        hello.setName(name);
        return hello; // 문자가 아니라 객체를 넘김
    }

    static class Hello {
        private String name;
        public String getName() {
            return name;
        }

        public void setName(String name) {
            this.name = name;
        }
    }
```

4. 실행_json



json 타입으로 보여진다.

5. 원리



@ResponseBody가 있으면, HttpMessageConverter가 동작한다. return 값이 문자열이면 StringConverter, 객체이면 JsonConverter를 통해 json 형태로 바꿔서 내보낸다.