<정적 컨텐츠>
웹을 개발한다는 것은 사실 크게 세가지 방법이 있다.

1.정적 컨텐츠
2.MVC와 템플릿 엔진
3.API

1.정적 컨텐츠
정적 컨텐츠가 뭐냐면 이전에 스프링에서 웰컴 페이지 했던 것처럼 서버에서 처리하는 로직 없이 그대로 웹브라우저에 그대로 보내주는 거다 이게 정적 컨텐츠다

2.MVC와 템플릿 엔진
가장 많이 사용하는 방식이다. 예를들어 JSP, PHP 같은 템플릿 엔진이다.
html을 그냥 주는 게 아니라 서버에서 프로그래밍해서 html을 동적으로 바꿔서 웹으로 보내는거다.
이런걸 템플릿 엔진이라고 하고 그걸 하기 위해서 컨트롤러,
모델, 템플릿 엔진 화면 이 세가지를 모델뷰 컽르롤러, MVC 라고 한다.

3.API
API는 만약 안드로이나 아이폰 클라이언트랑 개발을 해야한다면 json 이라는 데이터 포멧으로 내려준다. html 이런걸 내리는 게 아니라 json 이라는 어떤 데이터 구조 포멧으로 클라이언트 한테 데이터를 전달하는 게 요즘 API 방식이다.

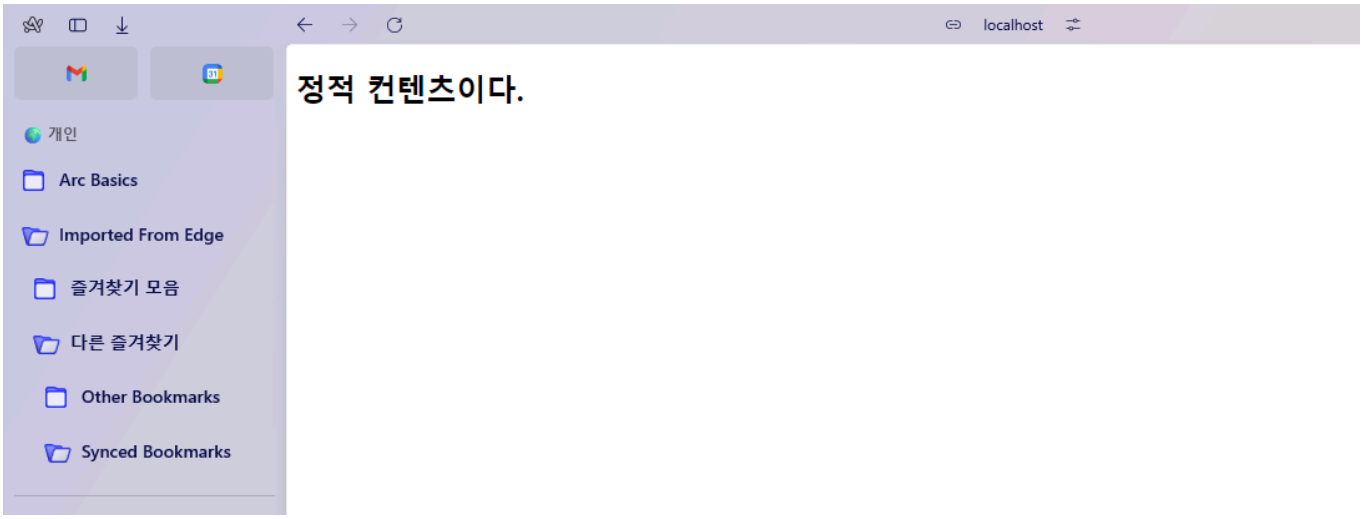
[정적 컨텐츠]
스프링 부트는 정적 컨텐츠 기능을 자동으로 제공한다. 스프링 기본 설정으로 스프링 부트는 정적 컨텐트를 /static 이 폴더에서 찾아서 제동한다고 되어있다.

<실습>
인텔리제이 에서 resources/static 에 hello-static.html이라고 만든다.
아무 html을 작성해 보겠다.

[resources/static/hello-static.html]
<!DOCTYPE html>

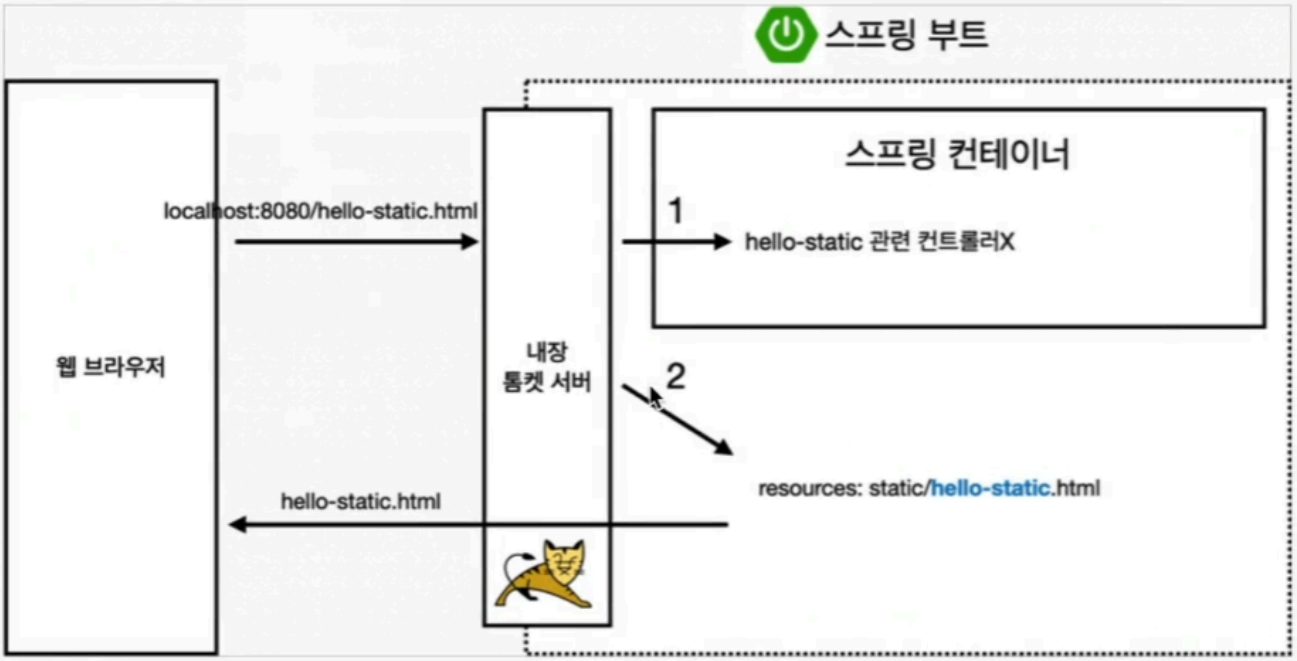
```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <h2>
            정적 컨텐츠이다.
    </h2>
</body>
</html>

이제 서버를 실행시켜 보자.
 서버를 실행시키고 ulr로 바로 접근이 가능하다
 http://localhost:8080/hello-static.html
```



정적파일을 resources/static 에 넣으면 그대로 반환이 된다.
대신 여기에는 어떤 프로그밍을 할수는 없다, html 파일 그대로 반환이 된다.
이걸 그림으로 설명해 보겠다.

정적 컨텐츠 이미지



간단하게 원리를 설명해 보자면
웹 브라우저에서 **localhost:8080/hello-static.html**을 스프링 부트에 내장되어 있는
톰캣 서버가 받는데 톰캣은 그걸 다시 스프링 컨테이너로 넘긴다.
근데 스프링은 요청이 오게되면 컨트롤러 쪽에서 **hell-static**이 있는지 먼저 찾아본다.
컨트롤러가 우선순위를 먼저 가진다는 소리다.

현재 컨트롤러에는 **hello**라는 컨트롤러는 있지만 **hello-static**이라는
매핑이된 컨트롤러는 없다.

그러면 스프링 부트는 어떻게 하냐면 여기 내부에 있는 **resources/static** 에서
hello-statc.html을 찾는다. 찾고 바로 웹으로 반환한다.

[MVC와 템플릿 엔진]
MVC란 **Model-View-Controller** 이다. 과거에는 컨트롤러랑 뷰 라는게 따로 분리되어 있지
않았다. 뷰에 모든 작업을 전부 처리했었다. **JSP**로 예전엔 이런식으로 개발을 많이 했다.
그걸 모델**1** 방식이라고 한다. 지금은 **MVC**스타일로 많이 한다 왜냐면 뷰는 화면을 그리는데에
모든 역량을 집중해야 한다.

그리고 컨트롤러나 모델에 관련된 부분들은 비즈니스 로직과 관련이 있거나 내부적인걸 처리하는
데에 집중을 해야한다. 그래서 **Model-View-Controller** 로 쪼갠거다.
이번엔 존더 내용이 있는 컨트롤러를 만들어 보겠다.

<실습>

저번에 실습한 controller/HelloController 에 하나의 함수를 더 만들어 줄거다.

[controller/HelloController]

```
@GetMapping("hello-mvc")
public String hellomvc(@RequestParam("name") String name, Model model)
{
    model.addAttribute("name",name);
    return "hello-template";
}
```

이번엔 파라미터를 받을거다 hellomvc의 매개값인 String name, 에 어노테이션으로 RequestParam("name") 으로 해준다 이러면 외부 웹에서 쿼리스트링으로 직접 값을 받을수 있다. 그리고 model.addAttribute("name",name);로 외부에서 넘어온 값을 담아 줄거다.

그리고 마지막으로 retuen 으로 hello-templete로 으로 넘겨줄거다.

[resources/templates/hello-templete.html]

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <p th:text="'hello' + ${name}">hello! empty</p>
</body>
</html>
```

내용은 이전 처음에 실습했을때와 같다.

타임리프의 장점 resources/templates/hello-templete.html 이파일을 우클릭해보면
Absolute Path라도 있다. 이걸 웹 URL에 붙이면 서버를 거치지 않고 그대로 html파일을
열어볼수 있다. hello! empty 이 값은 없어도 된다.

<p th:text="'hello' + ${name}">hello! empty> 여기서 "'hello' + ${name}" 이
내용으로 hello! empty 이 내용이 바뀐다. 컨트롤러의

model.addAttribute("name",name); name 키가 ${name} 이거다.

![[Pasted image 20240810182510.png]]

1. **웹 브라우저**: 사용자가 웹 브라우저에서 `localhost:8080/hello-mvc`라는 URL을 요청 한다.
2. **내장 톰캣 서버**: 이 요청이 내장 톰캣 서버에 도달한다.
3. **스프링 컨테이너의 Controller 처리**:
   - 요청이 스프링 컨테이너로 전달되면, `helloController`라는 이름의 컨트롤러에서 해당 요청을 처리한다.
   - 이 컨트롤러는 "hello-template"이라는 이름의 뷰(View)를 반환(return)하고, "name"이라는 키로 "spring"이라는 값을 가진 모델 데이터를 함께 전달한다.
4. **뷰 리졸버(ViewResolver)**:
   - 스프링의 뷰 리졸버는 반환된 뷰 이름("hello-template")을 바탕으로 실제 렌더링할 HTML 템플릿 파일(`templates/hello-template.html`)을 찾는다.
   - 여기서 Thymeleaf와 같은 템플릿 엔진이 해당 HTML 파일을 처리해, 모델 데이터를 HTML에 반영하여 최종적으로 완성된 HTML 문서를 생성한다.
5. **HTML 반환**: 최종적으로 생성된 HTML 문서는 웹 브라우저로 반환되어, 사용자에게 표시된다.

요약

이 이미지는 Spring Boot에서 MVC 패턴을 사용해 클라이언트의 요청을 처리하고, 템플릿 엔진(예: Thymeleaf)을 사용해 동적인 HTML 페이지를 생성하여 응답하는 과정을 설명하고 있다. 컨트롤러는 클라이언트의 요청을 처리하고, 필요한 데이터를 모델로 전달하며, 템플릿 엔진은 이 데이터를 바탕으로 최종 HTML 페이지를 렌더링한다.