

# chpt3

## 1. HTTP는 무엇인가

## 2. HTTP 요청 / 응답의 형식

### 1) 요청

### 2) 응답

url - 자원의 위치 알리는 문자열

## 3. Media Types

JSON (Javascript Object Notation)

## 1. Spring MVC

Model, View, Controller의 약자

## 2. Controller, RequestMapping

### 1. Controller, RequestMapping실습

### 2. 추가 응용 코드

### 3. json 파일 활용 코드

Response Body를 붙이지 않았다면

Response Body를 붙이게 되면

## 에러 해결

### 1) invalid source release: 11

### 2) Logger 선언 시에

### 3) payload 전달 시에

## 3. Controller, RestController

@RestController 를 붙이게 되면

media 파일 경로

## Controller , RestController

### 1) Controller Annotation

## Static (정적)

## Dynamic (동적)

## JSP, Thymeleaf

아이템 전달 - jsp

아이템 전달 - thyme

### (+) JAVA

### 1) List 와 ArrayList 차이

## 에러 발생 & 해결

### 1) Could not find org.apache.tomcat.embed:tomcat.embed:tomcat-embed-jasper.

1차 시도 :

해결법

## postman 활용

## GetPost 방식 활용해보기

# 1. HTTP는 무엇인가

이미지출처

## OSI 7 Layer

L7	응용 계층 (Application Layer)
L6	표현 계층 (Presentation Layer)
L5	세션 계층 (Session Layer)
L4	전송 계층 (Transport Layer)
L3	네트워크 계층 (Network Layer)
L2	데이터 링크 계층 (Data Link Layer)
L1	물리 계층 (Physical Layer)

## TCP/IP 4 Layer

L4	응용 계층 (Application Layer)
L3	전송 계층 (Transport Layer)
L2	인터넷 계층 (Internet Layer)
L1	네트워크 액세스 (Network Access Layer)

- 응용계층의 프로토콜 => HTTP, SMTP, FTP : 주고받을 데이터를 어떤 규칙을 가지고 작성하느냐의 문제
- **HTTP** : HyperText transfer protocol (정보를 담고 있는 텍스트 통신 규약)
  - 응용 계층에 정의된 통신 규약
  - 서버와 클라이언트 간에 메시지를 전달하는 형식을 정의한 규약
- **REST** : REpresentational State Transfer
  - API 를 설계하는 데 있어서 존재하는 규칙

## 2. HTTP 요청 / 응답의 형식

### 1) 요청

이미지출처

```
POST / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh;... )... Firefox/51.0
Accept: text/html,application/xhtml+xml,..., */*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-12656974
Content-Length: 345

-12656974
(more data)
```

Request headers

General headers

Entity headers

1. Request Line : 메소드, Path, Version
2. Request Headers : HTTP 요청에 대한 부수적인 데이터(요청한 형식, 길이, 어디로 요청을 했는지)
3. Request Body : HTTP 요청에 관한 실제 데이터

### 2) 응답

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Connection: Keep-Alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Wed, 10 Aug 2016 13:17:18 GMT
Etag: "d9b3b803e9a0dc6f22e2f20a3e90f69c41f6b71b"
Keep-Alive: timeout=5, max=999
Last-Modified: Wed, 10 Aug 2016 05:38:31 GMT
Server: Apache
Set-Cookie: csrftoken=...
Transfer-Encoding: chunked
Vary: Cookie, Accept-Encoding
X-Frame-Options: DENY

(body)
```

Response headers

Entity headers

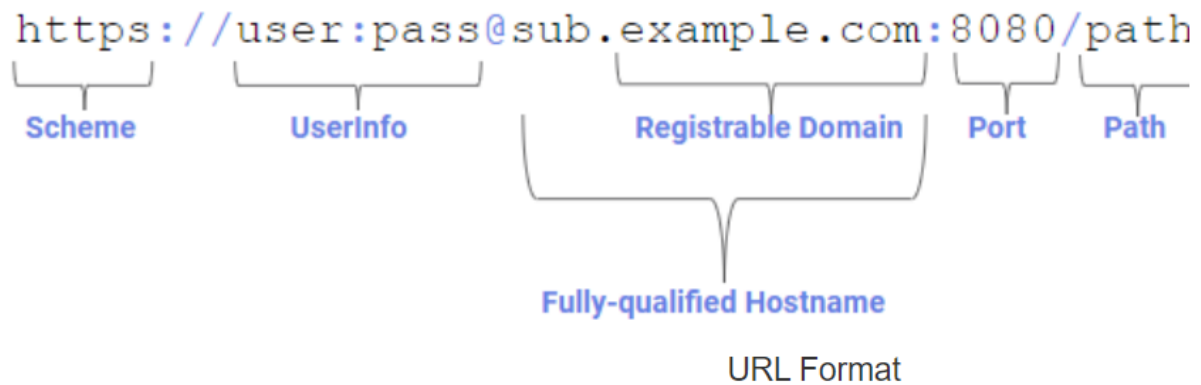
General headers

1. Status Line : 요청 처리에 대한 상태 표시줄
2. Response Headers : HTTP 응답에 대한 부수적인 데이터

3. Request Body : 응답에 대한 데이터

## url - 자원의 위치 알리는 문자열

이미지출처



- Uniform Resource Locator : 인터넷 상에 자원의 위치를 나타내는 문자열
- path 까지가 인터넷 상의 컴퓨터를 나타내는 부분
- path 뒤부터는 컴퓨터 안의 위치(경로)

## 3. Media Types

- 인터넷 상에서 주고받는 데이터의 형식  
(ex)
  - image, jpeg
  - video, mp4
- Content Type : HTTP 의 응답데이터(Body)의 Media Type을 알려주는 헤더

이미지, 설명 출처

=> `application/json` , `multipart/form-data` 가 자주 사용될 것

타입	설명	일반적인 서브타입 예시
text	텍스트를 포함하는 모든 문서를 나타내며 이론상으로는 인간이 읽을 수 있어야 합니다	text/plain, text/html, text/css, text/javascript
image	모든 종류의 이미지를 나타냅니다. (animated gif 처럼) 애니메이션되는 이미지가 이미지 타입에 포함되긴 하지만, 비디오는 포함되지 않습니다.	image/gif, image/png, image/jpeg, image/bmp, image/webp
audio	모든 종류의 오디오 파일들을 나타냅니다.	audio/midi, audio/mpeg, audio/webm, audio/ogg, audio/wav
video	모든 종류의 비디오 파일들을 나타냅니다.	video/webm, video/ogg
application	모든 종류의 이진 데이터를 나타냅니다.	application/octet-stream, application/pkcs12, application/vnd.ms-powerpoint, application/xhtml+xml, application/xml, application/pdf

## 멀티파트 타입

multipart/form-data  
multipart/byteranges

멀티파트 타입은 일반적으로 다른 MIME 타입들을 지닌 개별적인 파트들로 나누어지는 문서의 카테고리를 가리킵니다. 즉 이 타입은 **합성된** 문서를 나타내는 방법입니다. [HTML Forms](#)과 [POST](#) 메서드의 관계 속에서 사용되  
는 multipart/form-data 그리고 전체 문서의 하위 집합만 전송하기 위한 [206 Partial Content](#) 상태 메시지와 함께 사용되는 multipart/byteranges를 제외하고는, HTTP가 멀티파트 문서를 다룰 수 있는 특정한 방법은 존재하지 않습니다: 메시지는 브라우저에 간단히 전달됩니다 (문서를 인라인에 어떻게 디스플레이 할지 모르기에, '다른 이름으로 저장' 윈도우를 제시할 겁니다)

## JSON (Javascript Object Notation)

- 데이터를 주고받을 때 흔히 사용하는 형태
- 속성(Attribute) - 값(Value) 형태와 배열을 활용

<https://developer.mozilla.org/ko/docs/Learn/JavaScript/Objects/JSON>

```
{
  "squadName": "Super hero squad",
  "homeTown": "Metro City",
  "formed": 2016,
  "secretBase": "Super tower",
  "active": true,
  "members": [
    {
      "name": "Molecule Man",
      "age": 29,
      "secretIdentity": "Dan Jukes",
      "powers": [
        "Radiation resistance",
        "Turning tiny",
        "Radiation blast"
      ]
    },
    {
      "name": "Madame Uppercut",
      "age": 39,
      "secretIdentity": "Jane Wilson",
      "powers": [
        "Million tonne punch",
        "Damage resistance",
        "Superhuman reflexes"
      ]
    },
    {
      "name": "Eternal Flame",
      "age": 1000000,
      "secretIdentity": "Unknown",
      "powers": [
        "Immortality",
        "Heat Immunity",
        "Inferno",
        "Teleportation",
        "Interdimensional travel"
      ]
    }
  ]
}
```

위와 같은 형식

=> 일반적 데이터를 표현한 객체는 json 형태로 주고받는다

- 중괄호, 키, 값, 배열 을 조합해서 사용

# 1. Spring MVC

## Model, View, Controller의 약자

1. Model : 서비스 데이터 그 자체
2. View : 사용자가 확인하는 데이터의 표현
3. Controller : 사용자의 입출력 다루는 부분
  - 사용자는 view를 통해서 화면을 보고 controller라는 부분에 하고자하는 명령을 내리게 되면 이 controller은 모델을 변화시키고, 모델을 갱신, 이후 변화된 데이터가 다시 view에 반영된다.

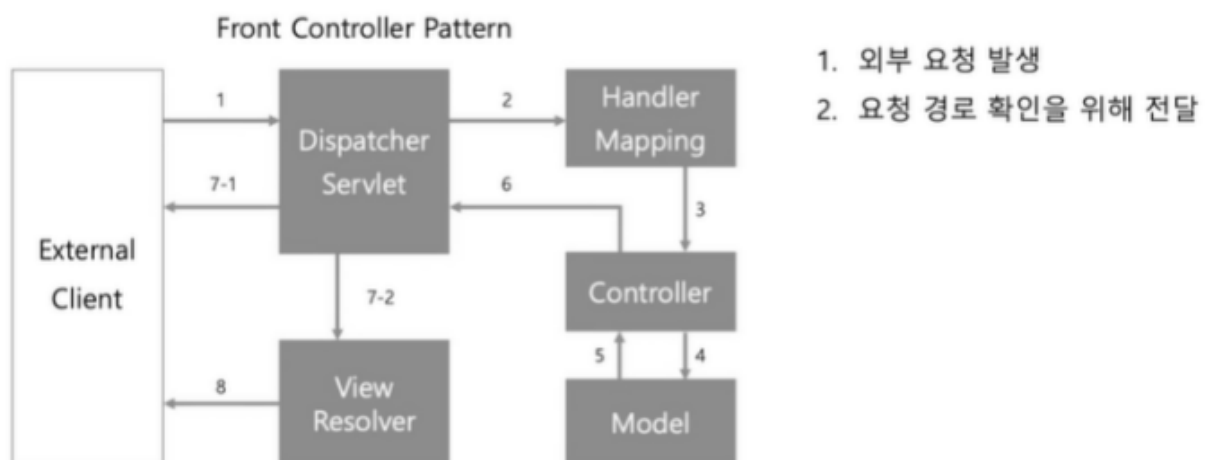
## django mtv & spring mvc 패턴 비교

### - 1) MTV : 모델, 템플릿, 뷰

- 모델 : MVC 모델에 대응, DB에 저장되는 데이터
- 템플릿 : MVC 뷰에 대응, 유저에게 보여지는 화면, 장고는 뷰에서 로직을 처리한 후 html파일을 context와 함께 렌더링, 이 때의 HTML 파일을 템플릿이라 함
- 뷰 : MVC의 컨트롤러에 대응, 요청에 따라 적절한 로직 수행해 결과를 템플릿으로 렌더링  
(+) URLConf : url 패턴을 정의해서 해당 url고 나

### - 2) MVC : 모델, 뷰, 컨트롤러

- 모델 : 데이터를 가지고 있으며, 데이터 처리 로직을 가짐
- 뷰 : 화면의 요청에 대한 결과물을 보여주는 역할, 유저와 어플리케이션 간 인터페이스
- 컨트롤러 : 모델과 뷰 이어주는 역할

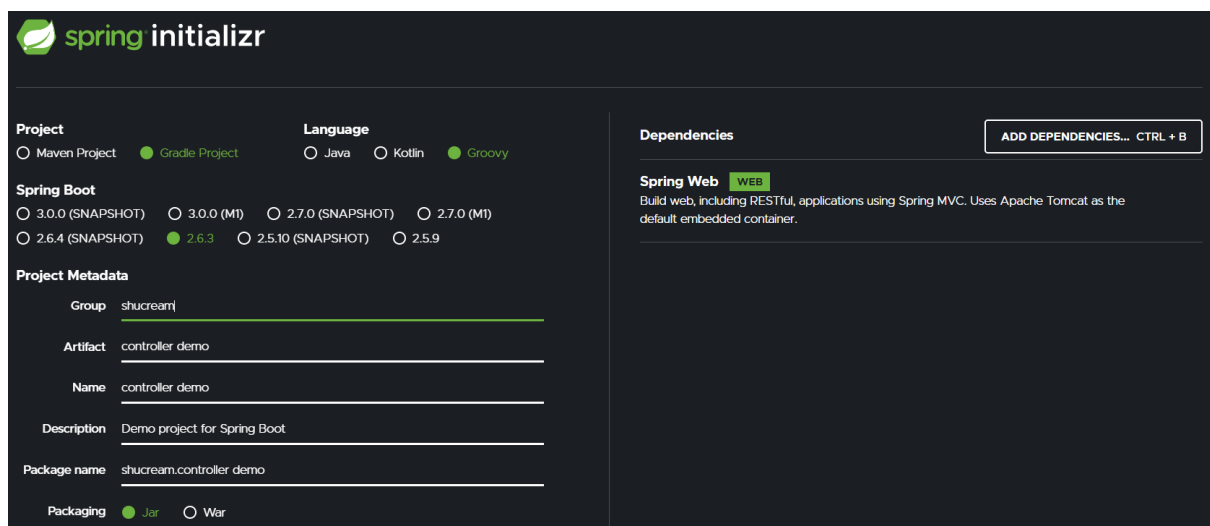


1. 외부 요청 발생
2. 요청 경로 확인을 위해 전달
3. Controller로 전달
4. 모델 조작
5. 갱신된 데이터 전달
6. 응답 전달
7. 응답을 Client로 전송
8. 데이터 전송
9. 데이터를 포함한 view 제작
10. 사용자에게 뷰 제공

## 2. Controller, RequestMapping

### 1. Controller, RequestMapping실습

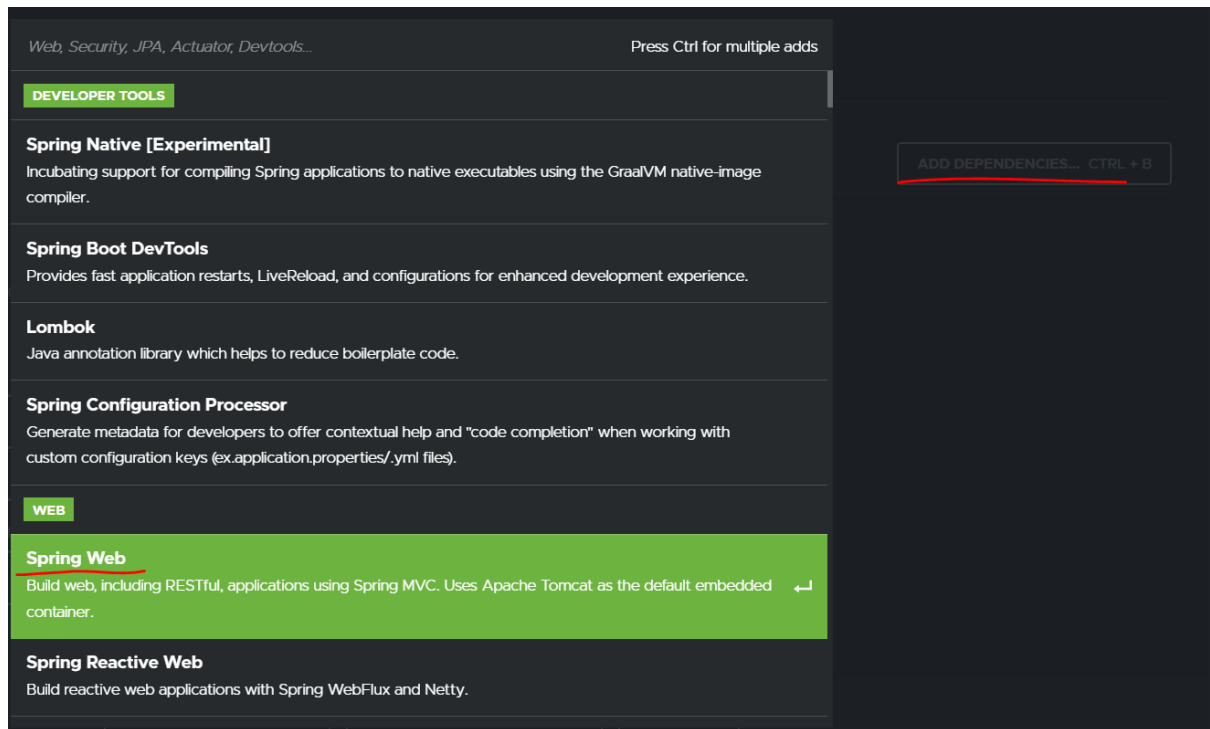
- 스프링 이니셜라이저로 이동



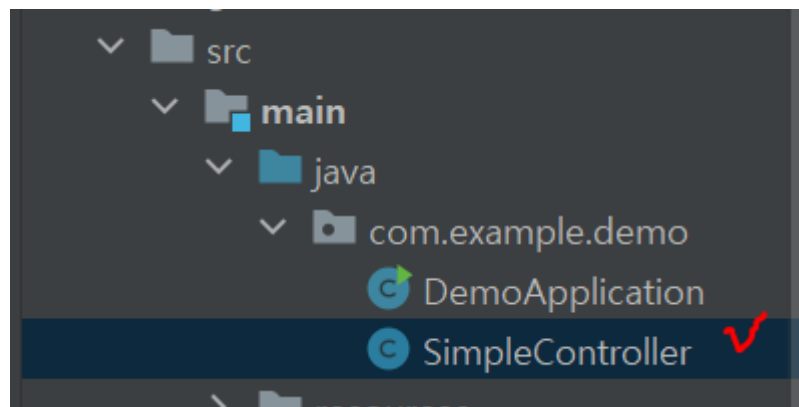
The image shows the Spring Initializr web interface, which is used to generate a new Spring project. The interface is dark-themed and contains several sections for configuring the project.

- Project:** Includes radio buttons for **Maven Project** and **Gradle Project** (selected).
- Language:** Includes radio buttons for **Java**, **Kotlin**, and **Groovy** (selected).
- Spring Boot:** Includes radio buttons for various versions: 3.0.0 (SNAPSHOT), 3.0.0 (M1), 2.7.0 (SNAPSHOT), 2.7.0 (M1), 2.6.4 (SNAPSHOT), 2.6.3 (selected), 2.5.10 (SNAPSHOT), and 2.5.9.
- Project Metadata:** Includes input fields for **Group** (shucream), **Artifact** (controller demo), **Name** (controller demo), **Description** (Demo project for Spring Boot), and **Package name** (shucream.controller demo).
- Packaging:** Includes radio buttons for **Jar** (selected) and **War**.
- Dependencies:** Includes a button **ADD DEPENDENCIES... CTRL + B** and a section for **Spring Web** (WEB) with a description: "Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container."





- 프로젝트 생성 & main-java-폴더명(이니셜라이저에서 했던 거) -controller 용 파일 하나 생성



- import ~ Controller & Annotation 달아주기
- controller 코드

```
package com.example.demo;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
```

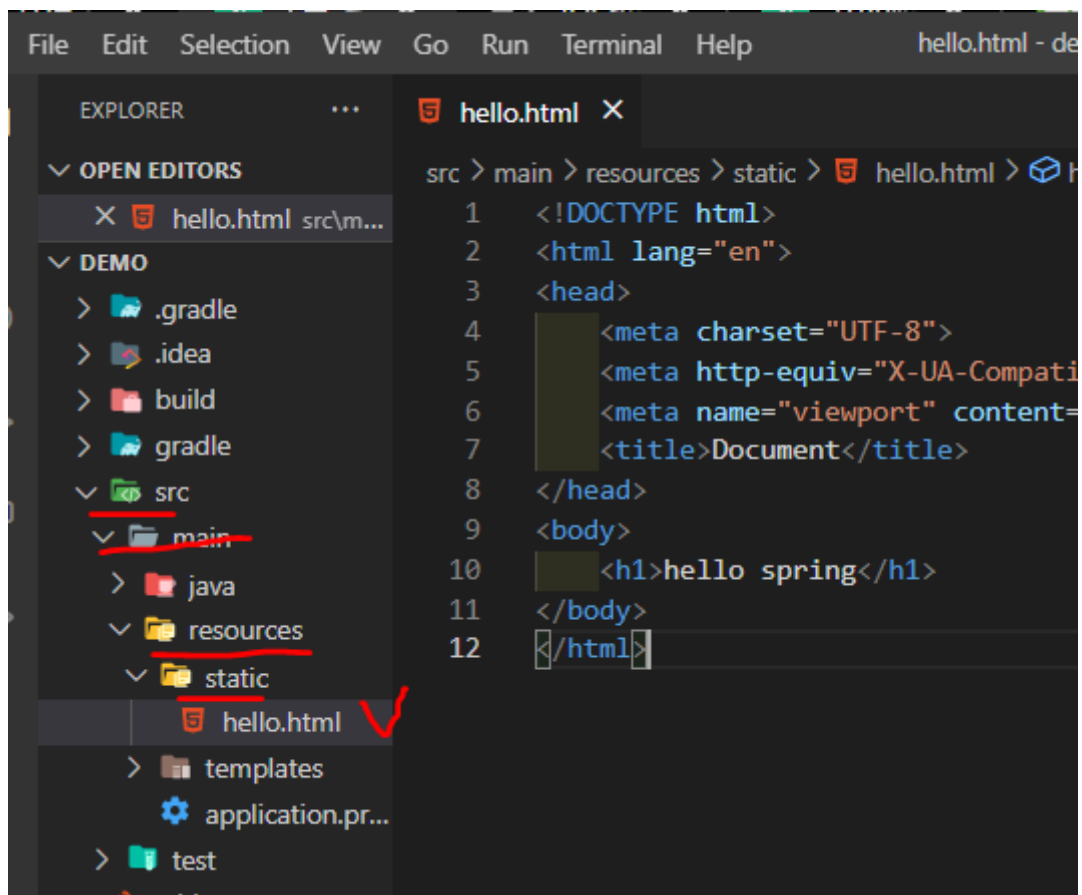
```

public class SimpleController {
    private static final Logger logger
        = LoggerFactory.getLogger(SimpleController.class);

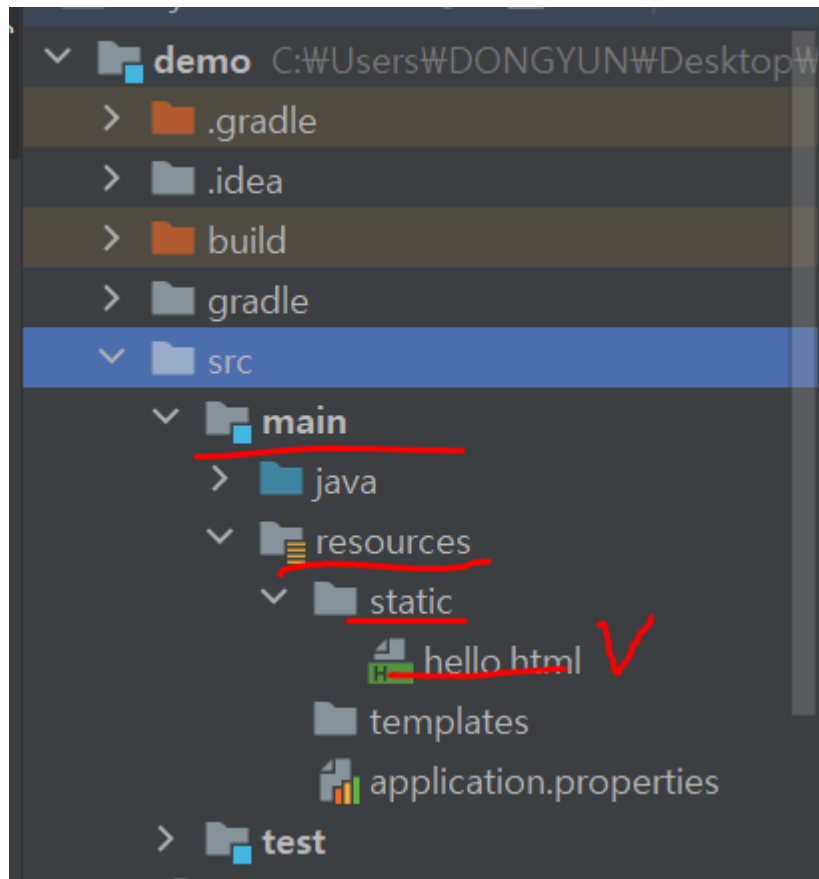
    @RequestMapping(
        value = "hello", //애는 요청의 url로서의 pass를 담당
        method = RequestMethod.GET
    )
    public String hello(){
        return "hello.html";
    }
}

```

- vsc에서 html 만들기(intelliJ에서 열었던 폴더랑 똑같은 폴더를 열게 되면 html이 똑같이 생성됨)



=> intelliJ에서도 생성



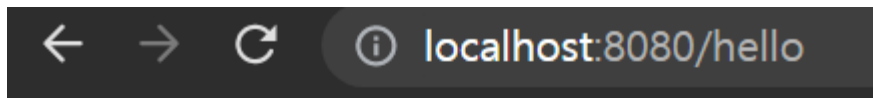
=> 이제 run 하면 결과

```
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.56]
main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1562 ms
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
main] com.example.demo.DemoApplication : Started DemoApplication in 2.521 seconds (JVM running for 2.928)
```

```
: Initializing Spring embedded WebApplicationContext
: Root WebApplicationContext: initialization completed in 1562 ms
: Tomcat started on port(s): 8080 (http) with context path ''
: Started DemoApplication in 2.521 seconds (JVM running for 2.928)
: Initializing Spring DispatcherServlet 'dispatcherServlet'
: Initializing Servlet 'dispatcherServlet'
: Completed initialization in 1 ms
```

포트 8080으로 이동해보자

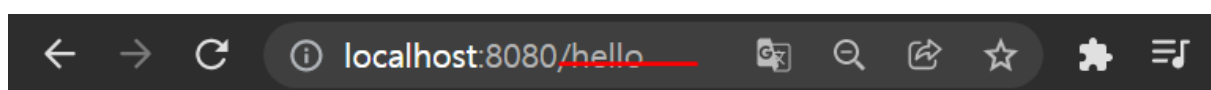
<http://localhost:8080/hello>



# hello spring

(+) 만약 url에 pass 해주는 value를 hungry로 바꾸면

```
SimpleController.java x DemoApplication.java x
8
9  @Controller
10 public class SimpleController {
11     private static final Logger logger
12         = LoggerFactory.getLogger(SimpleController.
13
14     @RequestMapping(
15         value = "hungry", //애는 요청의 url로서의 pass를
16         method = RequestMethod.GET
17     )
18     public String hello(){
19         return "hello.html";
20     }
21 }
```



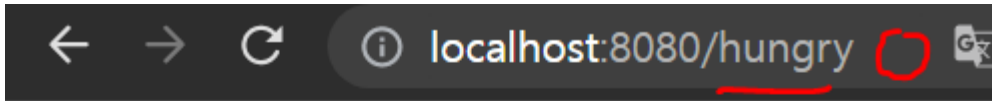
## Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as fallback.

Wed Feb 09 13:56:56 KST 2022

There was an unexpected error (type=Not Found, status=404).

=> url을 hungry로 바꿔줘야 한다



# hello spring

controller, requestmapping 두개를 조합

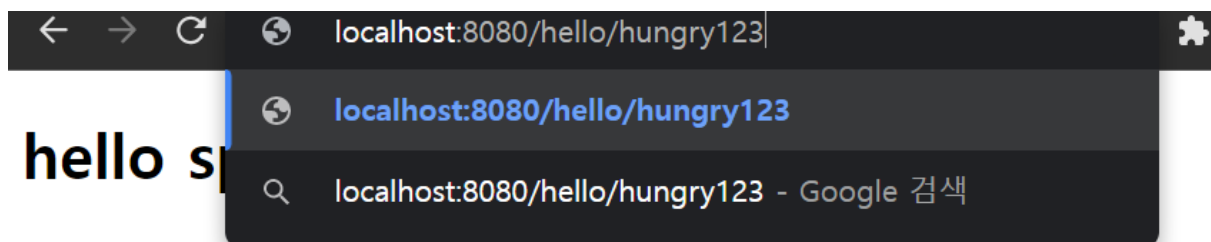
=> 기본적으로 요청을 받는 창구로서의 역할을 수행하게 되는 것

- > requestmapping 말고도 getmapping이 있는데, requestmapping과의 차이점은 request가 get으로 고정된다는 점!
- > 간단한 getMapping 관련 코드

```
1 @GetMapping(  
2     value = "/hello/{id}"  
3 )  
4 public String helloPath(@PathVariable("id") String id){  
5     logger.info("Path variable is" + id);  
6     return "/hello.html";  
7 }
```

## 코드 풀이

1. 만약 url에 `hello/{id}` 값으로 상대경로가 찍혀서 get 되면 helloPath 함수 발동
2. getMapping에 있는 uri에서 "id"라는 아이를 id라는 값으로 받아오라
3. logger.info를 통해서 함수가 발동되면 우리의 로그에 `Path variable is {uri에 찍혔던 id 값}` 을 찍어라
4. hello.html 페이지를 나타나게 하라



이런 식으로 사이트 url 에 id 값을 더하면

```
b.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
b.servlet.DispatcherServlet : Completed initialization in 1 ms
ample.demo.SimpleController  : Path variable is1
ample.demo.SimpleController  : Path variable ishungry123
```

내가 지정한

```
Path variable is {uri에 찍혔던 id값}
```

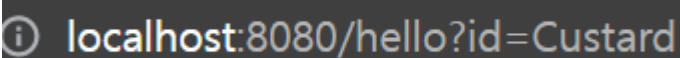
방식으로 로그가남게 된다

- 그리고 보여줄 페이지는 여전히 hello.html로 설정돼있으니깐 url이 달라도 같은 페이지가 보여지는 것

## 2. 추가 응용 코드

```
@RequestMapping(
    value = "hello"
)
public String hello(@RequestParam(name="id", required = false, defaultValue = "")String id){
    logger.info("Path:Hello");
    logger.info("Query Param id : " + id);
    return "hello.html";
}
```

- 만약 hello 라는 url이 발동했을 때
- request parameter 중에서 `id="값"` 으로 구성된 아이가 존재한다면 로그에 `Query Param id : {값}` 으로 찍어라
- 하지만 required=false이므로 필수는 아니다
- 실행 화면  
`http://localhost:8080/hello?id=Custard`  
=> 위처럼 id의 값으로 Custard를 건넸다면 내 로그에



```
: Completed initialization in  
: Path:Hello ✓  
: Query Param id : Custard ✓
```

- 그 아이디의 값이 내가 원하는 형식으로 잘 찍힌다

### 3. json 파일 활용 코드

- main에 SimplePayload 파일 제작

```
package com.example.demo;  
  
public class SimplePayload {  
    private String name;  
    private int age;  
    private String Occupation;  
  
    public SimplePayload(String name, int age, String occupation) {  
        this.name = name;  
        this.age = age;  
        Occupation = occupation;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public int getAge() {  
        return age;  
    }  
  
    public String getOccupation() {  
        return Occupation;  
    }  
  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    public void setAge(int age) {  
        this.age = age;  
    }  
  
    public void setOccupation(String occupation) {  
        Occupation = occupation;  
    }  
  
    @Override  
    public String toString() {  
        return "SimplePayload{" +
```

```

        "name='" + name + '\\\' +
        ", age=" + age +
        ", Occupation='" + Occupation + '\\\' +
        '}'';
    }
}

```

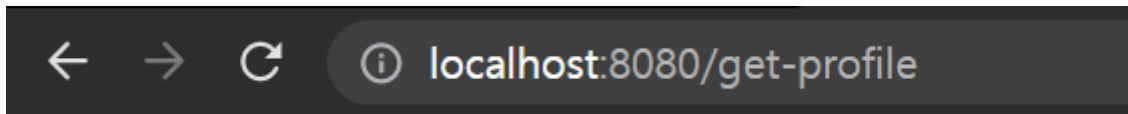
- 위와 같은 추가 파일 작성 (생성자, toString 아이는 모두 generate 기능을 이용해 너무 간편히 만듦)
- `SimpleController`

```

@GetMapping(
    "/get-profile"
)
public @ResponseBody SimplePayload getProfile(){
    return new SimplePayload("custard", 23,"Developer");
}

```

- 위와 같은 코드 작성하면 (데이터를 http의 body에 넣고 운반하겠다는 말)



```

{"name": "custard", "age": 23, "occupation": "Developer"}

```

- 지정된 url에 데이터가 잘 전달된다

## Response Body를 붙이지 않았다면

```

@GetMapping(
    "/get-profile"
)
public SimplePayload getProfile(){
    return new SimplePayload("custard", 23,"Developer");
}

```

- 이런 형태였다면 getProfile() 함수는 getMapping 을 탐색
- getMapping의 path value를 확인 & return 할 html과 같은 뷰를 탐색하게 됨

## Response Body를 붙이게 되면



- 애가 뷰를 탐색하지 않음
- 대신 `getProfile()` 자체로 "나는 데이터다" 라고 인식하게 되는 것

### (+) [Spring PathVariable Annotation 설명]

(<https://www.baeldung.com/spring-pathvariable>)

- `@PathVariable` annotation can be used to handle template variables in the request URI mapping  
=> uri 안에 있는 변수를 가져올 때 사용하는 어노테이션

```
@GetMapping("/api/employees/{id}")
@ResponseBody
public String getEmployeesById(@PathVariable String id) {
    return "ID: " + id;
}
```

### (+) Spring RequestParam Annotation 설명

- `@RequestParam` : 넘어오는 requestParam(쿼리) (쿼리스트링(Query String)) 중에서 지정한 키값을 데리고 오는 역할을 수행
- 다만 해당하는 parameter이 없을 수도 있는 경우가 존재하게 된다면 에러를 방지하기 위해서 `required=False` 를 덧붙여서 필수적으로 요구하지 않게하기

Path variable is {uri에 찍혔던 id값} 형식

쿼리스트링

[쿼리스트링 설명]<https://ysoh.tistory.com/entry/Query-String>)

`http://hostname[:port]/folder/file?변수1=값1&변수2=값2&변수3=값3`

=> "?" 뒤의 굵은 이탤릭체로 표현된 부분이 쿼리스트링 이다. URL 주소와 쿼리스트링은 "?"로 구분되며 변수와 값의 쌍(변수=값)으로 구성된다. 만약 여러 쌍의 변수와 값을 전달할 경우 각각의 쌍을 "&"로 구분

(ex) `http://localhost:8080/JSPLecture/queryStr.jsp?`

`no=200058001&name=Hong`

## (+) endpoint

```
/this-is-an-endpoint  
/another/endpoint  
/some/other/endpoint  
/login  
/accounts  
/cart/items
```

and when put under a domain, it would look like:

```
https://example.com/this-is-an-endpoint  
https://example.com/another/endpoint  
https://example.com/some/other/endpoint  
https://example.com/login  
https://example.com/accounts  
https://example.com/cart/items
```

Can be either http or https, we use https in the example.

Also endpoint can be different for different HTTP methods, for example:

```
GET /item/{id}  
PUT /item/{id}
```

would be two different endpoints - one for **r**etrieving (as in "c**R**ud" abbreviation), and the other for **u**psdating (as in "cr**U**d")

- 같은 URL들에 대해서도 다른 요청을 하게끔 구별하게 해주는 항목

## \*\*|\*(+)JAVA toString 메소드 재정의

- toString :
  - => 객체 생성하고 (ex) `Human me = new Human();`
  - => 생성한 인스턴스를 `System.out.println(me)` 를 통해 찍으면 toString에서 재정의한대로 출력되는 것임
  - => 따로 오버라이드 하지 않으면 클래스 이름 + @ + hash코드 더한 결과로만 나온다

## 에러 해결

### 1) invalid source release: 11

Execution failed for task ':compileTestGroovy'.

invalid source release: 11

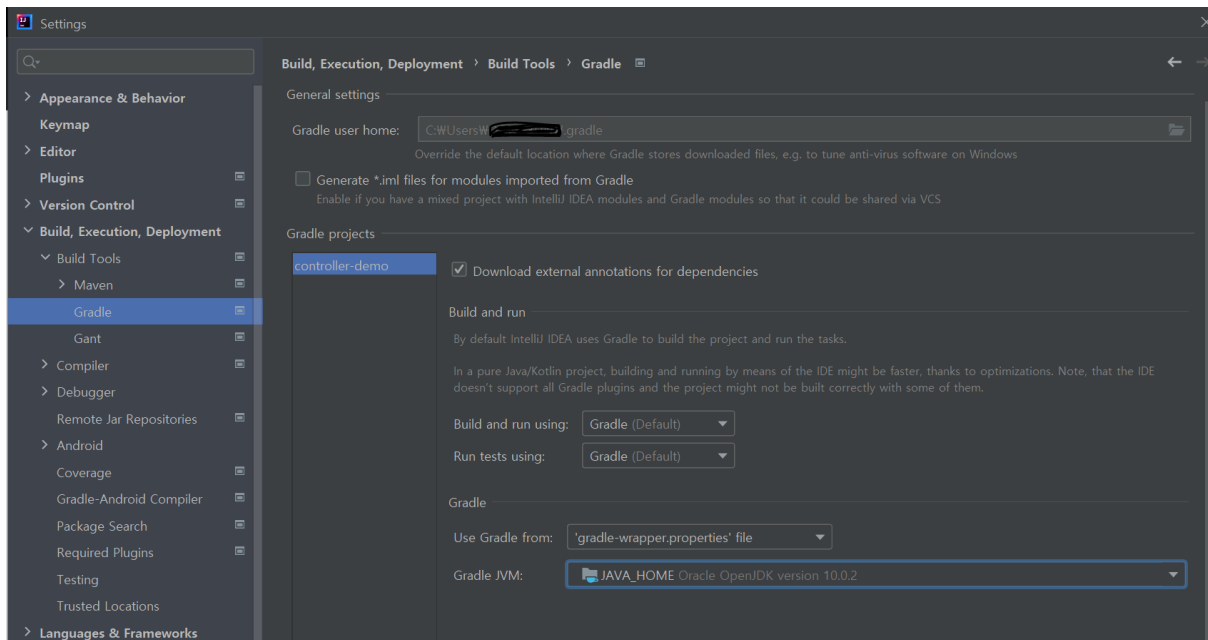
스택오버플로우 링크

=> that error occurs precisely when you are using a source / target level that is not supported by the JVM you are running gradle with.

=> gradle의 jvm설정이 11 이하 버전인데

bulid.gradle에 sourceCompatibility의 설정이 11로 되어 있어서 발생한다.

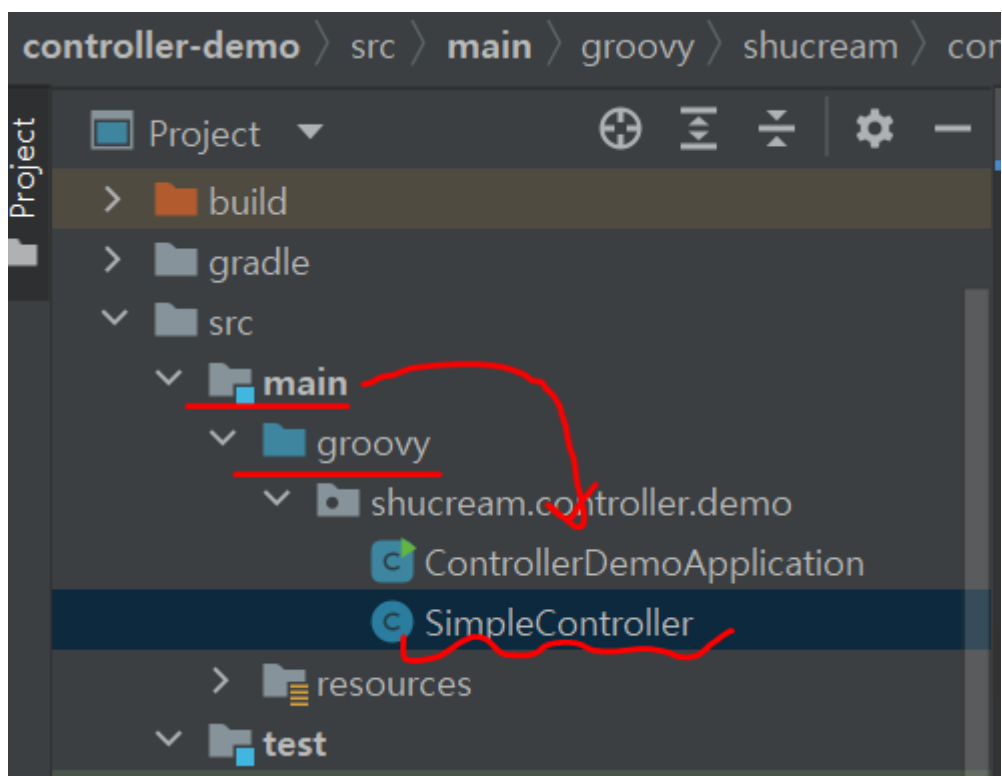
나의 jvm 설정



11이하로 되어 있었다 -> 따라서 이 아이를 11로 고쳐주고 저장함

해결책 참고 블로그 : <https://manystory.tistory.com/87>

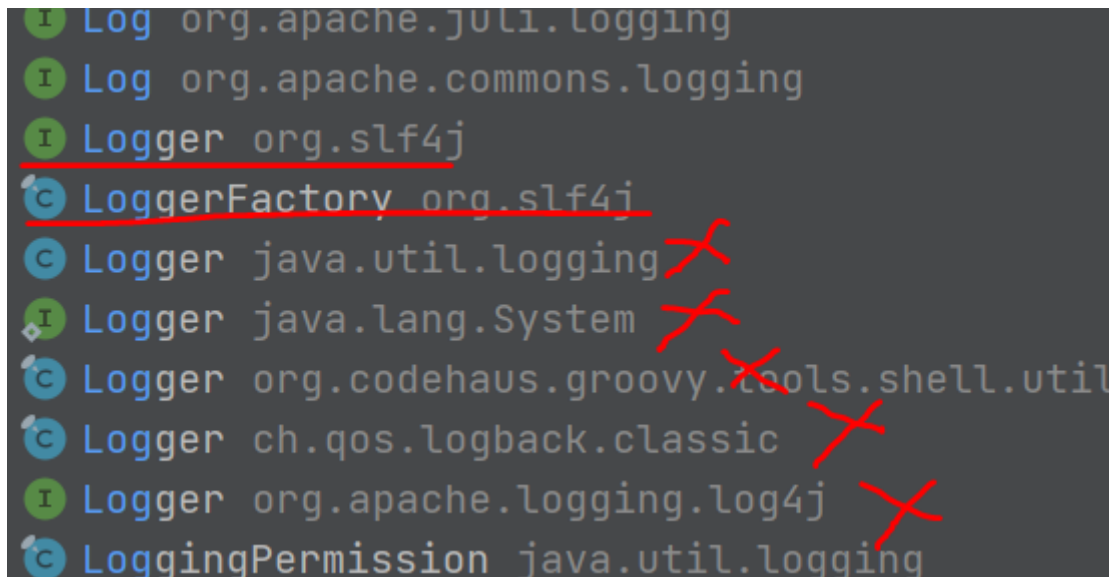
- 그리고 파일 만들 때 main 안에다가 만들어야 함..



## 2) Logger 선언 시에

```
class ch.qos.logback.classic.Logger cannot be cast to class java.util.logging.Logger
(ch.qos.logback.classic.Logger is in unnamed module of loader 'app'; java.util.logging.Logger is in module java.logging of loader 'bootstrap')
at shucream.controller.demo.SimpleController.<clinit>(SimpleController.java:13)
```

이런 에러가 뜬다면 잘못된 Logger를 선택한 것



빨간 밑줄 친 로거를 선택하셔야 합니다

### 3) payload 전달 시에

```
@GetMapping(
    "/get-profile"
)
public SimplePayload getProfile(){
    return new SimplePayload("custard", 23,"Developer");
}
```

- 원래 데이터는 HTTP 응답의 BODY에 작성이 되게 돼있음
- 따라서 annotation을 getProfile에 붙여주길

```
@GetMapping(
    "/get-profile"
)
public @ResponseBody SimplePayload getProfile(){
    return new SimplePayload("custard", 23,"Developer");
}
```

```

@GetMapping(
    "/get-profile"
)
public @ResponseBody SimplePayload getProfile(){
    return new SimplePayload( name: "custard", age: 23, occupation: "Developer");
}

```

### 3. Controller, RestController

```

package com.example.demo;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/rest")
public class SimpleRestController {
    private static final Logger logger = LoggerFactory.getLogger(SimpleRestController.class);

    @GetMapping("/simple-payload")
    public SimplePayload simplePayload(){
        return new SimplePayload("custard", 23, "Developer");
    }
}

```

- Response Body를 붙이지 않아도 잘 작동한다.



localhost:8080/rest/simple-payload

```

{"name": "custard", "age": 23, "occupation": "Developer"}

```

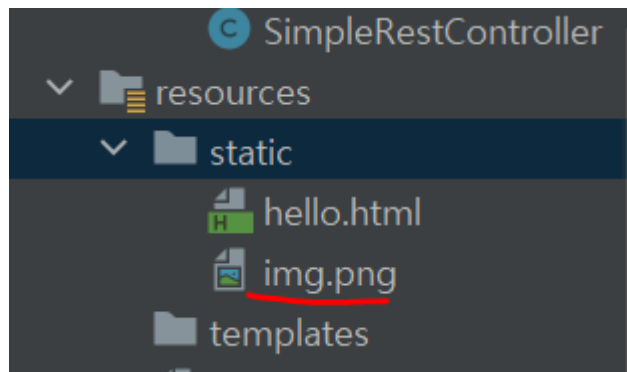
#### @RestController 를 붙이게 되면

- @Controller는 기본적으로 뷰를 제공, 데이터를 제공하는 넓은 범위의 annotation
- @RestController는 기본적으로 데이터를 주고받는 역할을 수행

## media 파일 경로

```
@GetMapping(
    value = "/simple-image",
    produces = MediaType.IMAGE_PNG_VALUE
)
public byte[] simpleImage() throws IOException {
    InputStream inputStream = getClass().getResourceAsStream(""); //name에 해당하는 애
    //inputStream = new FileInputStream(new File("파일위치지정")); //fileinputstream에 지
    return inputStream.readAllBytes();
}
```

- 비디오나 이미지나 모두 바이트들로 이루어져있다!
- 이미지를 추가하고 실제 이 이미지를 불러오도록 코드를 작성

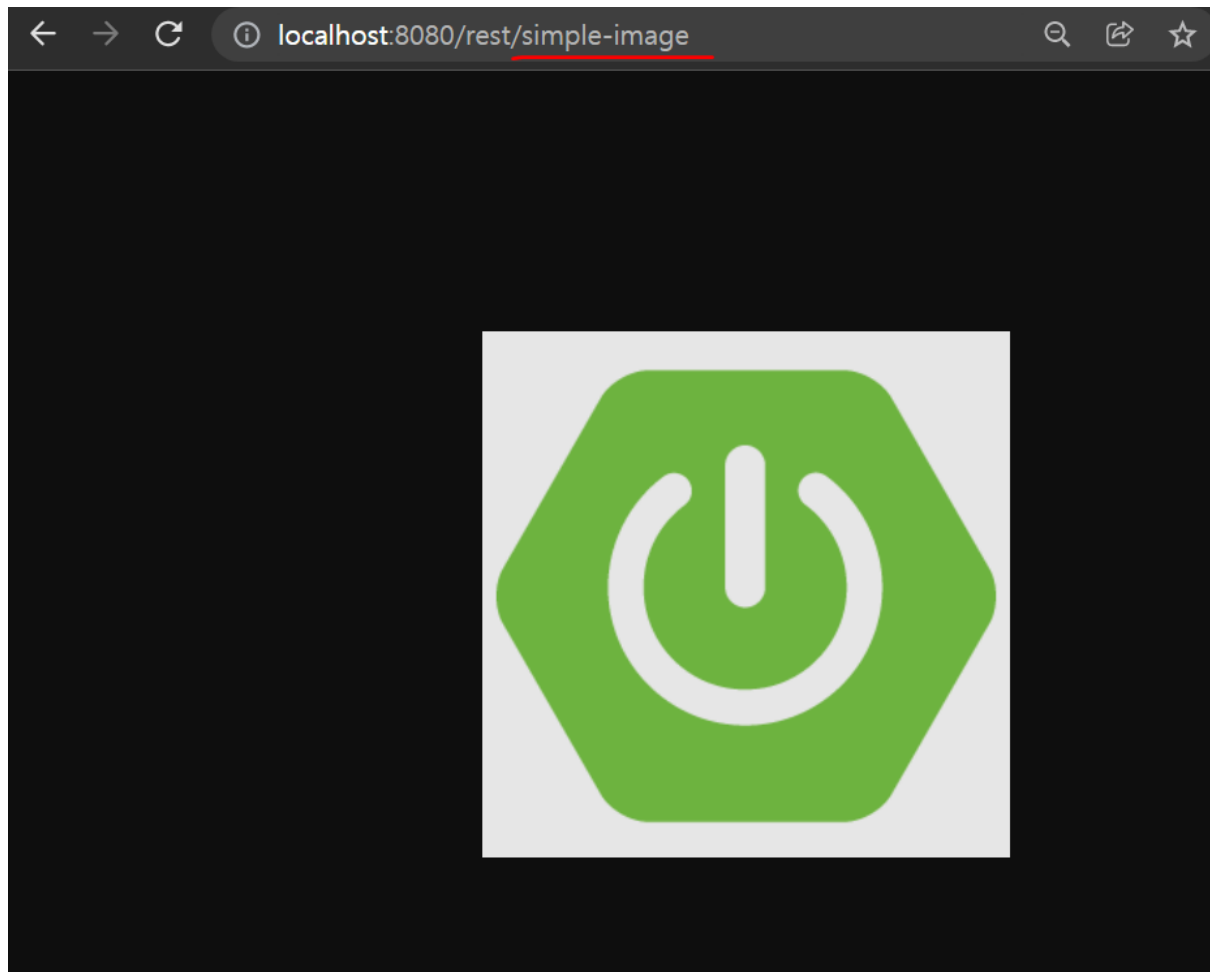


```
@GetMapping(
    value = "/simple-image",
    produces = MediaType.IMAGE_PNG_VALUE
)

public byte[] simpleImage() throws IOException {
    InputStream inputStream = getClass().getResourceAsStream("/static/img.png"); //
    return inputStream.readAllBytes();
}
```

- 항상 경로는 상대 경로로

`getClass().getResourceAsStream("static/img.png");` 이렇게 하면 절대 경로잖아~null로 인식된다.



## Controller , RestController

### 1) Controller Annotation

```
@Controller
public class SampleController{
}
```

- 요청 경로 설정을 위해 Controller Annotation 사용

```
@RequestMapping("/profile") //경로 지정(<http://localhost:8080/profile>)
public String profile(){
    logger.info("in profile"); //위의 경로로 로그에 이거 찍어라
    return "profile.html" // profile.html 보여줘라
}
```



- RequestMapping을 이용해 경로에 따라 실행될 함수 지정 가능

```
@RequestMapping("/profile") //경로 지정 (<http://localhost:8080/profile>)
public String profile(){
    logger.info("in profile"); //위의 경로로 로그에 이거 찍어라
    return "profile.html") // profile.html 보여줘라
}
```

- REQUEST가 GET으로 정해진 GetMapping 어노테이션도 가능
- 메소드 별로 별도의 Annotation이 존재한다

```
@GetMapping(
    value = "/sample-payload",
    produces = MediaType.APPLICATION_JSON_VALUE
)
public @ResponseBody SamplePayload samplePayload(){
    return new SamplePayload(
```

- HTML 외에 데이터 전송을 위해 Body, MediaType 지정 가능

```
@GetMapping(
    value = "sample-image",
    produces = MediaType.IMAGE_PNG_VALUE
)
public byte[] sampleImage() throws IOException{
```

- 어떤 형태의 응답이든 데이터의 일종임

## Static (정적)

- 이미 작성이 완료돼 변하지 않는 파일들  
(ex) html, css, js ,image

## Dynamic (동적)

- webpage : 서버에서 html 문서의 내용을 데이터에 따라 다르게 작성해 제공되는 이미지
- 다양하게 변형되어 나타내는 웹 페이지

# JSP, Thymeleaf

**Project**  
☐ Maven Project ☒ Gradle Project

**Language**  
☒ Java ☐ Kotlin ☐ Groovy

**Spring Boot**  
☐ 3.0.0 (SNAPSHOT) ☐ 3.0.0 (M1) ☐ 2.7.0 (SNAPSHOT) ☐ 2.7.0 (M1)  
☐ 2.6.4 (SNAPSHOT) ☒ 2.6.3 ☐ 2.5.10 (SNAPSHOT) ☐ 2.5.9

**Project Metadata**

Group: com.example

Artifact: demo

Name: demo

Description: Demo project for Spring Boot

Package name: com.example.demo

**Packaging**  
☒ Jar ☐ War

Java: ☐ 17 ☒ 11 ☐ 8

**Dependencies** ADD DEPENDENCIES

✓ **Spring Web** **WEB**  
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

✓ **Thymeleaf** **TEMPLATE ENGINES**  
A modern server-side Java template engine for both web and standalone environments. HTML to be correctly displayed in browsers and as static prototypes.

1. `built.gradle` Spring boot에서 JSP 사용하기 위해..

- [jsp 사용 설명 참고 블로그](#)
- <https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-thymeleaf/2.5.8>

```
dependencies {  
    implementation 'org.springframework.boot:spring-boot-starter-web'  
    implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'//추가  
    //implementation 'org.apache.tomcat.embed:tomcat-embed-jasper'//추가  
    implementation 'javax.servlet:jstl'//추가  
    testImplementation 'org.springframework.boot:spring-boot-starter-test'  
}
```

=>

```
dependencies {  
    implementation 'org.springframework.boot:spring-boot-starter-web'  
    implementation 'org.apache.tomcat.embed:tomcat-embed-jasper:9.0.58'  
    implementation 'javax.servlet:jstl'  
    testImplementation 'org.springframework.boot:spring-boot-starter-test'  
}
```

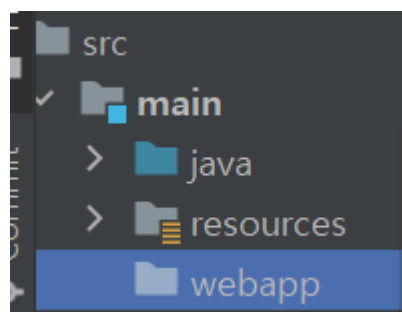
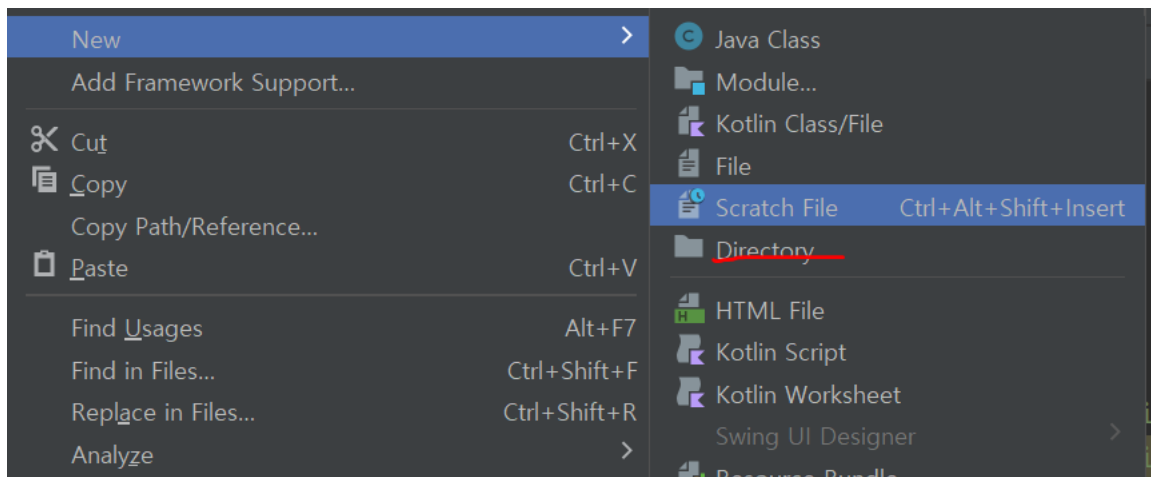
- `implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'`
- `implementation 'org.apache.tomcat.embed:tomcat-embed-jasper'`  
=> 위에 두개 중 하나만 사용한다

```
implementation 'org.apache.tomcat.embed:tomcat-embed-jasper' //JASPER
implementation 'javax.servlet:jstl' // JSTL
```

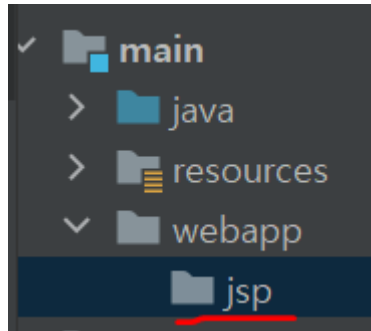
- 일케 두개는 jsp 사용을 위해 추가하는 것

## 디펜던시 추가한 이유? 설명 출처 블로그

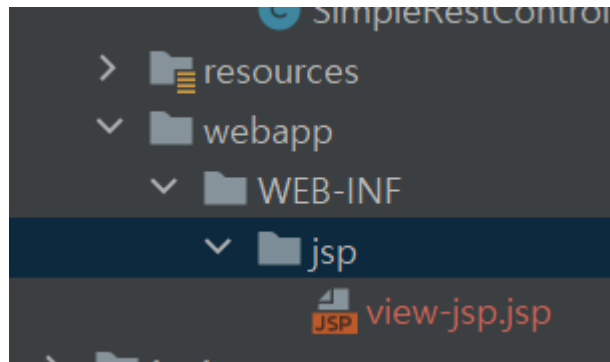
1. 스프링 부트에서는 기본적으로 JSP를 지원하지 않음
  2. 스프링 부트에 내장된 tomcat에는 컴파일하는 jsp 관련 엔진이 포함되지 않음
- dependencies 부분에 JSP 사용 시 필요한 jstl 추가
  - JSP 엔진 역할을 하는 tomcat-embed-jasper library 추가 (보통 thymeleaf나 jasper 중에 하나만 쓴다, 둘다는 안씀)
1. main에 **webapp** 이라는 directory 추가



1. 그 안에 jsp 라는 디렉토리 추가



# 1. vscode - view-jsp.jsp 파일 생성



```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
```

```
1 <%@ page contentType="text/html; charset=UTF-8" language="java" %>
```

- 빨간 부분이 존재하긴 하는데, 저 부분은 지금 우리가 java로 여기 작성하고 있는데 나중에 html로 변환 부탁 이라는 의미를 지닌다

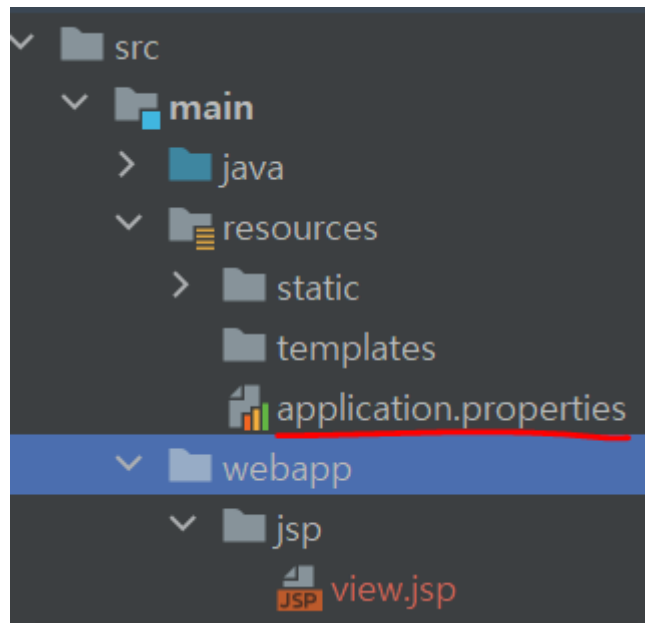
```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<%@ taglib prefix = "c" uri = "http://java.sun.com/jsp/jstl/core" %>
<html>
  <head>
  </head>
  <body>
    <table>
      <thead>
        <tr>
          <th>Name</th>
          <th>Age</th>
          <th>Occupation</th>
        </tr>
      <tbody>
        <c:forEach items="${profiles}" var="profile">
          <tr>
            <td>${profile.name}</td>
```

```

        <td>${profile.age}</td>
        <td>${profile.occupation}</td>
    </tr>
</c:forEach>
</tbody>
</thead>
</table>
</body>
</html>

```

# 1. application properties

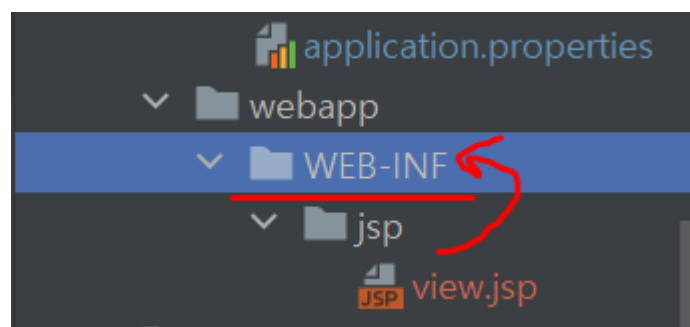


- 스프링에서 더이상 jsp 지원 안해줌, 따라서 지원해달라고 해야함

```

spring.mvc.view.prefix : /WEB-INF/jsp/
spring.mvc.view.suffix : .jsp

```



- WEB-INF라는 디렉토리 추가 생성 후 그 아래 webapp을 위치시키기

- prefix: 경로지정, suffix: 파일 확장자를 찾아줌

## 1. Controller 작성

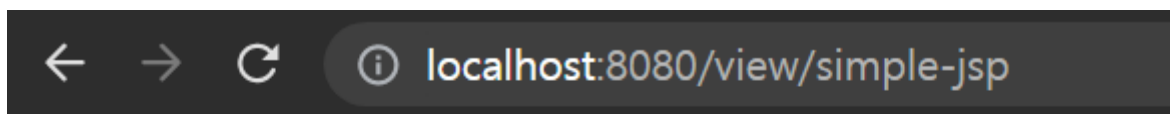
```
package com.example.demo;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;

@Controller
@RequestMapping("view")
public class SimpleController {
    private static final Logger logger
        = LoggerFactory.getLogger(SimpleController.class);

    @GetMapping("/simple-jsp")
    public String simpleJsp(){
        logger.info("in simple jsp");
        return "view-jsp";
    }
}
```

- public String simpleJsp()
  - => 이런 식의 String은 return 해줄 뷰를 찾는다
  - => 만약 이 함수가 serving 해줄 뷰가 존재하지 않게 된다면 해당하는 return 값의 경로로 보내게 된다
  - => 이때 우리가 application.properties에 jsp관련해서 정의해놓은게 있기 때문에 이를 토대로 찾아주세요~ 하는 것



**Name Age Occupation** |

=> run 해보니 잘 뜬다

```

<table>
  <thead>
    <tr>
      <th>Name</th>
      <th>Age</th>
      <th>Occupation</th>
    </tr>
  <tbody>
    <c:forEach items="${profiles}" var="profile">
      <tr>
        <td>${profile.name}</td>
        <td>${profile.age}</td>
        <td>${profile.occupation}</td>
      </tr>
    </c:forEach>
  </tbody>
</table>

```

## 아이템 전달 - jsp

### 1. list import

```
import java.util.List;
```

### 1.

```

@GetMapping("/simple-jsp")
public String simpleJsp(){
    logger.info("in simple jsp");
    List<SimplePayload> profile = new ArrayList<>();
    profile.add(new SimplePayload("custard", 23, "developer"));
    profile.add(new SimplePayload("happy", 22, "teacher"));
    profile.add(new SimplePayload("luckyHappyPretty", 20, "student"));
    return "view-jsp";
}

```

List<SimplePayload> profile = new ArrayList<>();

### 1. Model `model`이라는 데이터 전달용 매개변수 추가

- model에다가 우리가 만든 profile이라는 아이를 "profiles"라는 이름으로 전달할 것

```
model.addAttribute("profiles", profile);
```

```

@GetMapping("/simple-jsp")
public String simpleJsp(Model model){ //model이라는 데이터 추가

```

```

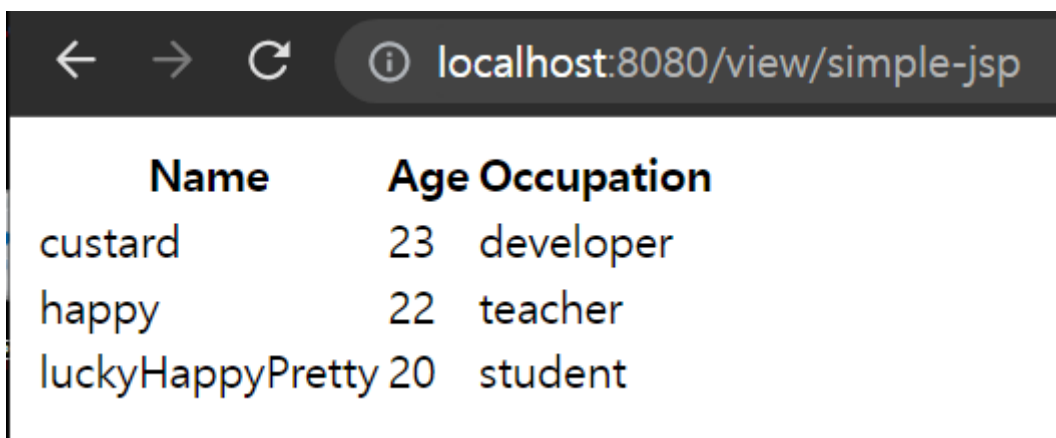
logger.info("in simple jsp");
//profile arraylist 에 요소들 추가
List<SimplePayload> profile = new ArrayList<>();
profile.add(new SimplePayload("custard", 23, "developer"));
profile.add(new SimplePayload("happy", 22, "teacher"));
profile.add(new SimplePayload("luckyHappyPretty", 20, "student"));

model.addAttribute("profiles", profile); //전달 요소를 모델에 추가

return "view-jsp";
}

```

## 1. 결과



Name	Age	Occupation
custard	23	developer
happy	22	teacher
luckyHappyPretty	20	student

- 위 방법말고도 Model and View로도가능

## 아이템 전달 - thyme

### 1. dependency 수정

```

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'//추가
    //implementation 'org.apache.tomcat.embed:tomcat-embed-jasper:9.0.58'
    //implementation 'javax.servlet:jstl'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}

```



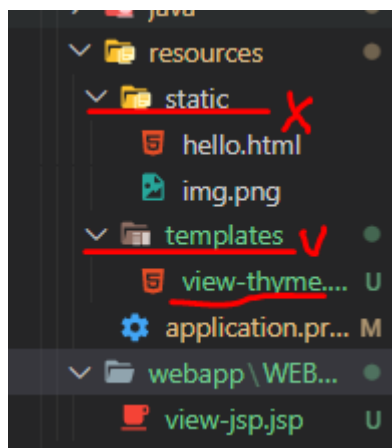
```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
    // implementation 'org.apache.tomcat.embed:tomcat-embed-jasper'
    // implementation 'javax.servlet:jstl'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}
```

## 1. Controller - ModelAndView 방식으로

```
@GetMapping("/simple-thyme")
public ModelAndView simpleThyme(){
    ModelAndView mandv = new ModelAndView();
    logger.info("in simple thyme");
    List<SimplePayload> profile = new ArrayList<>();
    profile.add(new SimplePayload("custard", 23, "developer"));
    profile.add(new SimplePayload("happy", 22, "teacher"));
    profile.add(new SimplePayload("luckyHappyPretty", 20, "student"));

    mandv.addObject("profiles", profile);
    mandv.setViewName("view-thyme");
    return mandv;
}
```

- 함수의 자료형을 ModelAndView로 선언해서 return 값을 이 자료형으로 설정
- ModelAndView로 인스턴스를 만들고, 해당 인스턴스 안에 만들었던 데이터를 addObject로 담아주고, 돌려줄(보여줄) View이름을 setViewName으로 알려주면 => 해당 뷰에 담은 object를 똑같이 돌려주는 것
- 다만 thyme 방식으로 가게 되면 static이 아닌 templates 자리에 뷰 파일(html) 생성해 주기



```

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <table>
    <thead>
      <tr>
        <th>Name</th>
        <th>Age</th>
        <th>Occupation</th>
      </tr>
    <tbody>
      <tr th:each="profile: ${profiles}">
        <td th:text="${profile.name}" >
        <td th:text="${profile.age}" >
        <td th:text="${profile.occupation}" >
      </tr>
    </tbody>
  </thead>
</table>
</body>
</html>

```

- jsp, thyme 따라서 템플릿 작성도 달라진다

```

<!-- <c:forEach items="${profiles}" var="profile">
  <tr>
    <td>${profile.name}</td>
    <td>${profile.age}</td>
    <td>${profile.occupation}</td>
  </tr>
</c:forEach> -->

```

```

> 참고 :
[<https://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html>](<https://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html>)
`/WEB-INF/templates/home.html`
```java
<!DOCTYPE html SYSTEM "http://www.thymeleaf.org/dtd/xhtml1-strict-thymeleaf-4.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:th="http://www.thymeleaf.org">

```

## (+) JAVA

### 1) List 와 ArrayList 차이

#### 설명 출처

- List = 인터페이스
- ArrayList = 클래스

List는 interface다. interface는 공통되는 메소드를 추출해 놓은 클래스로 생각하면 된다.

즉 범위로 생각하면 List 안에 ArrayList, LinkedList...등이 포함

```
ArrayList<Object> list = new ArrayList<>();  
List<Object> list = new ArrayList<>();
```

출처: <<https://yoon-dailylife.tistory.com/7>> [알면 쓸모있는 개발 지식]

- > 위는 그냥 구현체 클래스로 구현  
-> 아래는 ArrayList를 업캐스팅해서 사용  
-> 같은 결과 but List를 사용해 ArrayList를 생성하는 것은 유연성에서 효과d

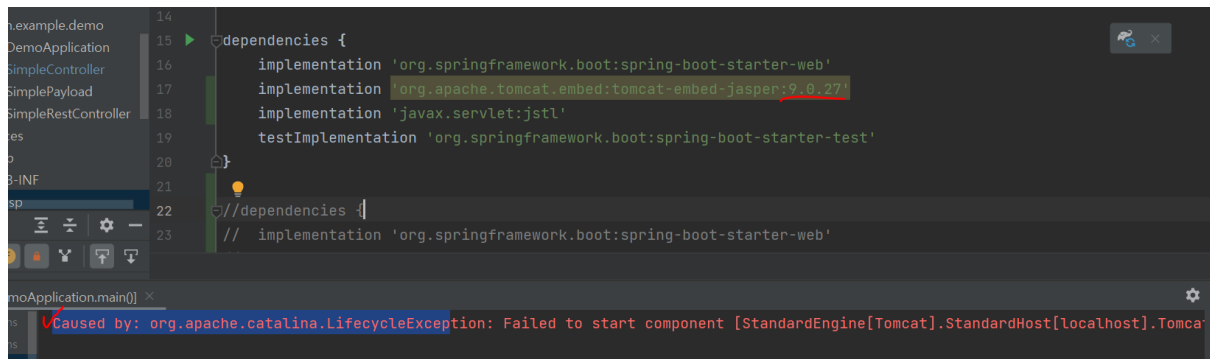
#### 객체는 인터페이스를 사용해라 - 출처

- 매개변수뿐 아니라 반환값, 변수, 필드를 전부 인터페이스 타입으로 선언하라.
- 적합한 인터페이스가 없다면 클래스의 계층구조 중 가장 상위의 클래스를 사용하라.
- <https://bibi6666667.tistory.com/236>
- Arraylist
  - ◆ ArrayList : resizable array리스트에 저장되는 데이터의 수가 늘어나면 자동으로 증가함

## 에러 발생 & 해결

1) `Could not find org.apache.tomcat.embed:tomcat.embed:tomcat-embed-jasper.`

1차시도 :



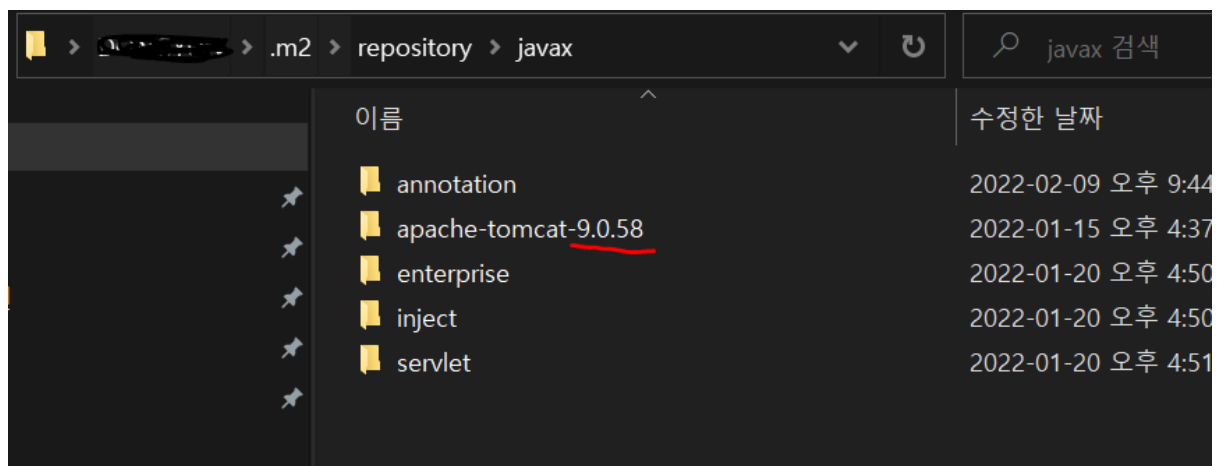
- 버전 명시, 그럼에도 apache를 찾을 수 없다고 뜬 -> tomcat 미설치 문제로 추측

## 해결법

### 해결법 참고 블로그

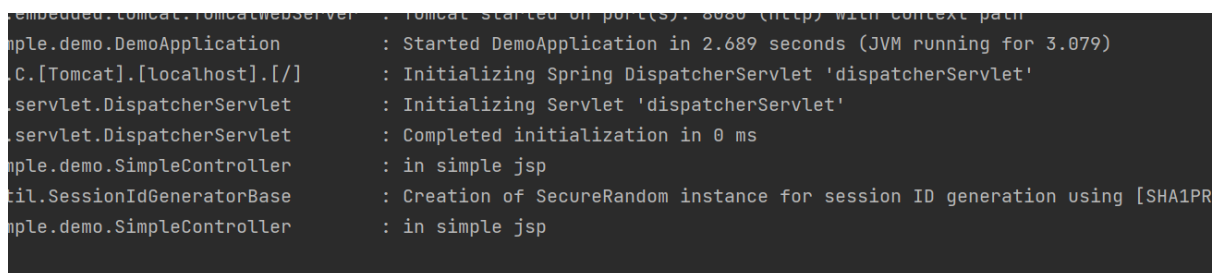
<https://tomcat.apache.org/>

-> 각자 컴퓨터 운영체제 맞게 톰캣 9.0 zip으로 다운로드

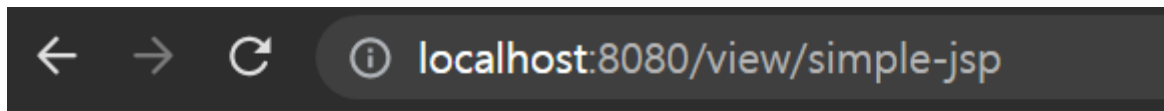


- 해당 파일을 `servlet` 폴더가 있는 곳에 복붙하기

나의 경우 => C:\Users\사용자명\.m2\repository\javax



음 ~ 잘 되네!!!



Name Age Occupation|

## postman 활용

- api만들고 테스트하기 위해서 사용

## GetPost 방식 활용해보기

### post 방식 controller구현

restController.java

```
@PostMapping("/simple-payload")
@ResponseStatus(HttpStatus.NO_CONTENT)
public void simplePayloadPost(@RequestBody SimplePayload simplepayloaddata){
    logger.info(simplepayloaddata.toString());
}
```

postman

으로 post 요청을 아래와 같이 날리게 되면

jsp-payload / http://localhost:8080/rest/simple-payload

POST

http://localhost:8080/rest/simple-payload

Params

Authorization

Headers (9)

Body ●

Pre-reqs

● none

● form-data

● x-www-form-urlencoded

● raw

```
1 {  
2   ... "name" : "custarddd",  
3   ... "age" : 23,  
4   ... "occupation" : "developer"  
5 }
```

- 로그가 잘 찍히는 것을 알 수 있음

```
: Initializing Servlet 'dispatcherServlet'  
: Completed initialization in 0 ms  
: SimplePayload{name='custarddd', age=23, Occupation='developer'}
```

## multipart-post 방식 구현

```
@PostMapping(  
    value = "/simple-multipart",  
    consumes = MediaType.MULTIPART_FORM_DATA_VALUE  
)  
@ResponseStatus(HttpStatus.NO_CONTENT)  
public void simpleMultipartpost(  
    @RequestParam("name") String name,  
    @RequestParam("age") Integer age,  
    @RequestParam("occupation") String occupation,  
    @RequestParam("file") MultipartFile multipartFile  
) {  
    logger.info("name :"+ name);  
    logger.info("age :"+ age);  
    logger.info("o :"+ occupation);  
}
```

```
logger.info("file name :"+ multipartFile.getOriginalFilename());
}
```

- \* **postman** \*에서 form-data 로 파일 형식 보낼 때 아래와 같이 바뀌서 보내주면 된다

	KEY	VALUE
<input checked="" type="checkbox"/>	name	custardpostmulti
<input checked="" type="checkbox"/>	age	23
<input checked="" type="checkbox"/>	occupation	dev
<input checked="" type="checkbox"/>	file	File ▾ <span>Select Files</span>
	Key	Value

response

Text  
 File ✓

POST ▾
 http://localhost:8080/rest/simple-multipart

Params
 Authorization
 Headers (9)
 **Body ●**
 Pre-request Script
 Tests
 Settings

☐ none
☒ form-data
☐ x-www-form-urlencoded
☐ raw
☐ binary
☐ GraphQL

	KEY	VALUE
<input checked="" type="checkbox"/>	name	custard
<input checked="" type="checkbox"/>	age	23
<input checked="" type="checkbox"/>	occupation	dev
<input checked="" type="checkbox"/>	file	sbT2-FyjF/02.PNG ×
	Key	Value

- 내가 원하는 로그대로 잘 찍히게 된다

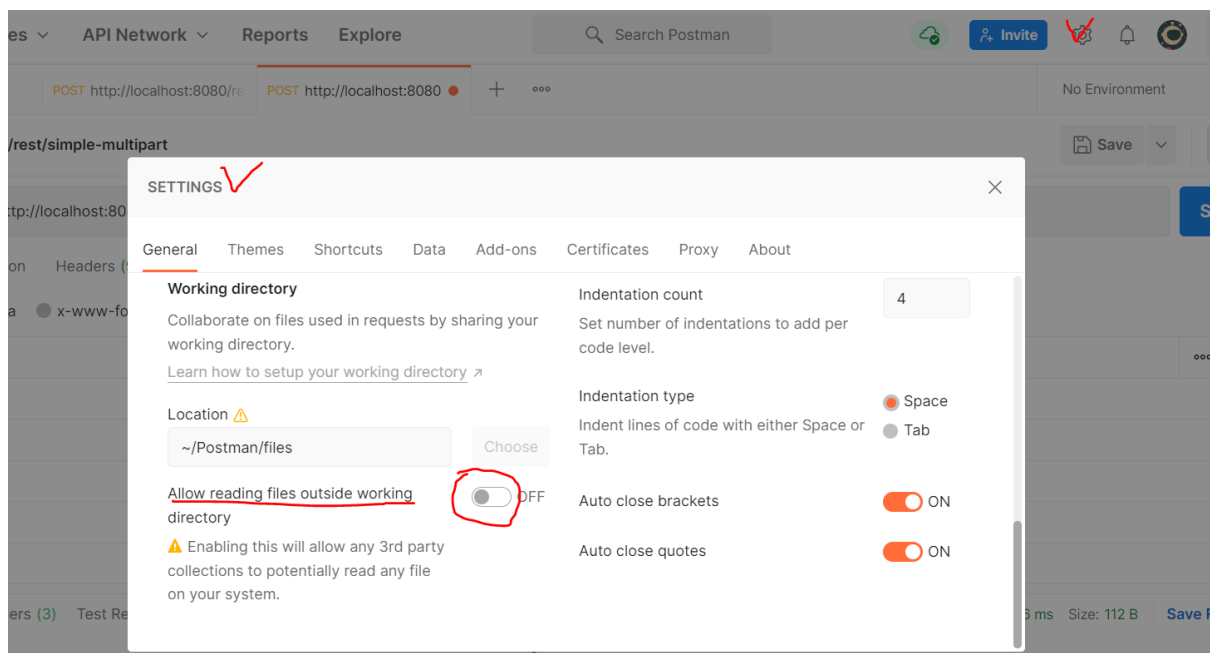
```
curl : file name :1111.PNG
curl : name :custard
curl : age :23
curl : o :dev
curl : file name :02.PNG
```

## 에러

Couldn't upload file Make sure that Postman can read files inside the working directory.

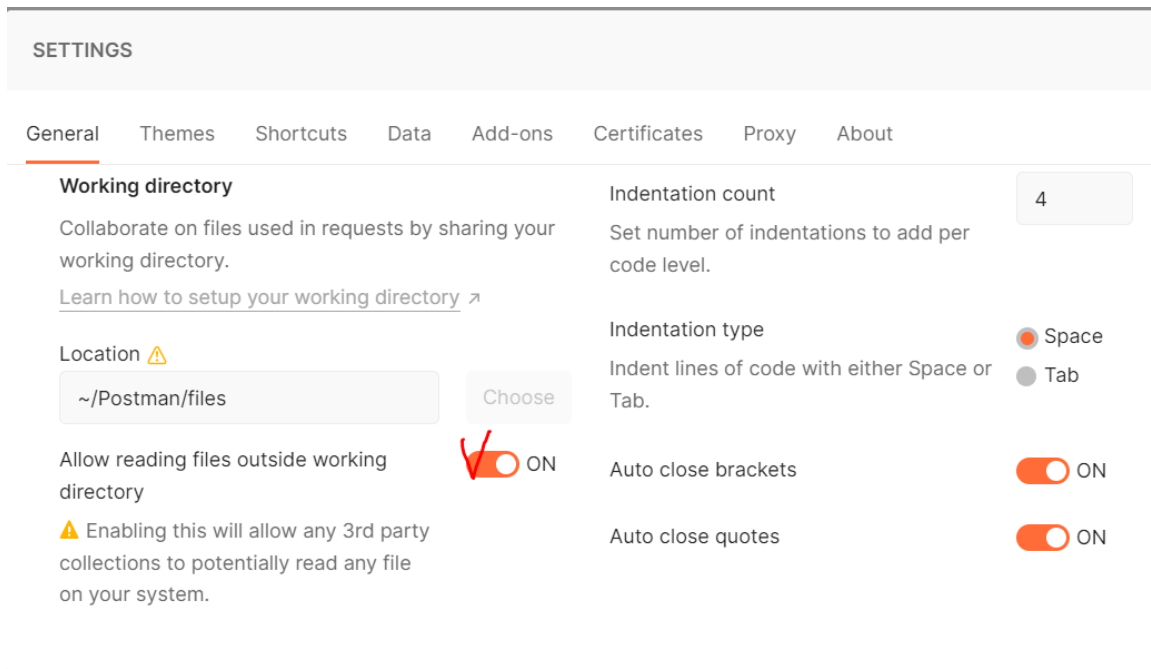
해결법 참조 블로그 : <https://euuuuuz.tistory.com/381>

- 이 부분이 off로 되어있음

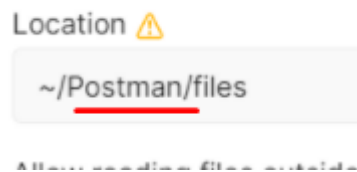


- on으로 변경해주기

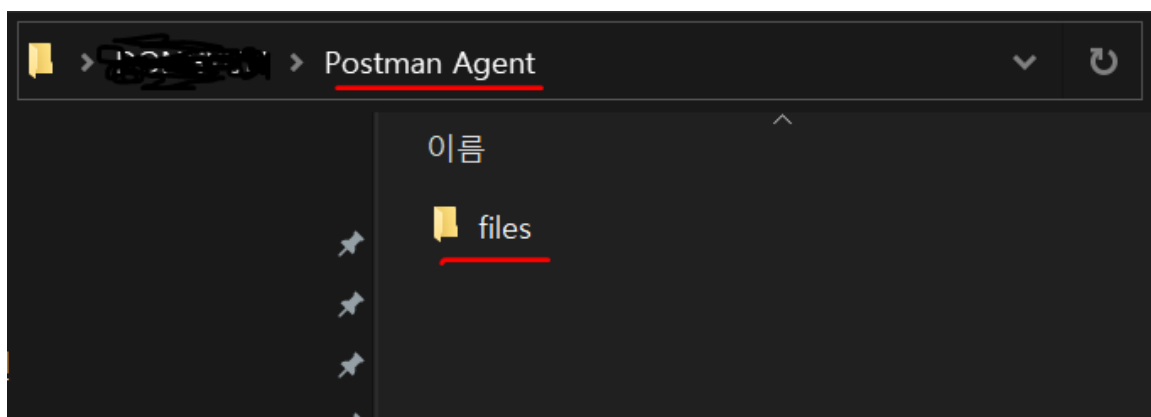




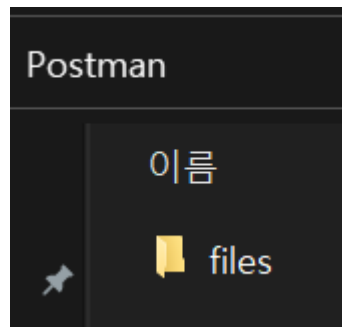
- postman에서 디렉토리를 찾아가야 하는데



- 내 컴퓨터에서는 아래와 같이 `Postman Agent / files` 로 되어 있어서 찾을 수 없다고 에러



- postman으로 이름 변경



- 아래와 같이 잘 올라간다

	KEY	VALUE
<input checked="" type="checkbox"/>	name	custardpostmultiee
<input checked="" type="checkbox"/>	age	23
<input checked="" type="checkbox"/>	occupation	dev
<input checked="" type="checkbox"/>	file	Cx33iubxA/1111.PNG x ✓
	Key	Value