**[SpringBoot] 스프링부트 Annotation 정리✔**

# Annotation 이란?

Annotation(@)은 사전적 의미로는 주석이라는 뜻이다.  
자바에서 Annotation은 코드 사이에 주석처럼 쓰이며 특별한 의미, 기능을 수행하도록 하는 기술이다.  
즉, 프로그램에게 추가적인 정보를 제공해주는 메타데이터라고 볼 수 있다.(meta data : 데이터를 위한 데이터)

 Annotation을 사용하면 **코드량이 감소하고 유지보수하기 쉬우며, 생산성이 증가한다.**

**어노테이션의 용도**

* **컴파일러에게 코드 작성 문법 에러를 체크하도록 정보를 제공한다.**
* **소프트웨어 개발 툴이 빌드나 배치시 코드를 자동으로 생성할 수 있도록 정보를 제공한다.**
* **실행시(런타임시)특정 기능을 실행하도록 정보를 제공한다.**

**어노테이션을 사용하는 순서**

* **어노테이션을 정의한다.**
* **클래스에 어노테이션을 배치한다.**
* **코드가 실행되는 중에 Reflection을 이용하여 추가 정보를 획득하여 기능을 실시한다.**

*\*Reflection이란 프로그램이 실행 중에 자신의 구조와 동작을 검사하고, 조사하고, 수정하는 것이다.*

## [**Spring의 대표적인 Annotation과 역할**](https://rebornbb.tistory.com/entry/SpringBoot-%EC%8A%A4%ED%94%84%EB%A7%81%EB%B6%80%ED%8A%B8-Annotation-%EA%B0%9C%EB%85%90-%EB%B0%8F-%EC%A0%95%EB%A6%AC#Spring%EC%9D%98%20%EB%8C%80%ED%91%9C%EC%A0%81%EC%9D%B8%20Annotation%EA%B3%BC%20%EC%97%AD%ED%95%A0-1)

### 

## [**@Component**](https://rebornbb.tistory.com/entry/SpringBoot-%EC%8A%A4%ED%94%84%EB%A7%81%EB%B6%80%ED%8A%B8-Annotation-%EA%B0%9C%EB%85%90-%EB%B0%8F-%EC%A0%95%EB%A6%AC#%40Component-1)

**개발자가 생성한 Class를 Spring의 Bean으로 등록할 때 사용하는 Annotation입니다.**

Spring은 해당 Annotation을 보고 Spring의 Bean으로 등록합니다.

@Component(value="myman")

public class Man {

public Man() {

System.out.println("hi");

}

}

### 

## [**@ComponentScan**](https://rebornbb.tistory.com/entry/SpringBoot-%EC%8A%A4%ED%94%84%EB%A7%81%EB%B6%80%ED%8A%B8-Annotation-%EA%B0%9C%EB%85%90-%EB%B0%8F-%EC%A0%95%EB%A6%AC#%40ComponentScan-1)

Spring Framework는**@Component, @Service, @Repository, @Controller, @Configuration** 중 1개라도 등록된 클래스를 찾으면,**Context에 bean으로 등록**합니다. **@ComponentScan Annotation이 있는 클래스의 하위 Bean**을 등록 될 클래스들을 스캔하여 Bean으로 등록해줍니다.

### 

## [**@Bean**](https://rebornbb.tistory.com/entry/SpringBoot-%EC%8A%A4%ED%94%84%EB%A7%81%EB%B6%80%ED%8A%B8-Annotation-%EA%B0%9C%EB%85%90-%EB%B0%8F-%EC%A0%95%EB%A6%AC#%40Bean-1)

**@Bean** Annotation은 **개발자가 제어가 불가능한 외부 라이브러리와 같은 것들을 Bean으로 만들 때** 사용합니다.

## [**@Controller**](https://rebornbb.tistory.com/entry/SpringBoot-%EC%8A%A4%ED%94%84%EB%A7%81%EB%B6%80%ED%8A%B8-Annotation-%EA%B0%9C%EB%85%90-%EB%B0%8F-%EC%A0%95%EB%A6%AC#%40Controller-1)

Spring MVC에서 Controller클래스에 쓰인다.

Spring에게 해당 Class가 Controller의 역할을 한다고 명시하기 위해 사용하는 Annotation입니다.

@Controller // 이 Class는 Controller 역할을 합니다

@RequestMapping("/user") // 이 Class는 /user로 들어오는 요청을 모두 처리합니다.

public class UserController {

@RequestMapping(method = RequestMethod.GET)

public String getUser(Model model) {

// GET method, /user 요청을 처리

}

}

### 

## [**@RequestHeader**](https://rebornbb.tistory.com/entry/SpringBoot-%EC%8A%A4%ED%94%84%EB%A7%81%EB%B6%80%ED%8A%B8-Annotation-%EA%B0%9C%EB%85%90-%EB%B0%8F-%EC%A0%95%EB%A6%AC#%40RequestHeader-1)

Request의 header값을 가져올 수 있으며, 해당 Annotation을 쓴 메소드의 파라미터에 사용합니다.

@Controller // 이 Class는 Controller 역할을 합니다

@RequestMapping("/user") // 이 Class는 /user로 들어오는 요청을 모두 처리합니다.

public class UserController {

@RequestMapping(method = RequestMethod.GET)

public String getUser(@RequestHeader(value="Accept-Language") String acceptLanguage) {

// GET method, /user 요청을 처리

}

}

## [**@RequestMapping**](https://rebornbb.tistory.com/entry/SpringBoot-%EC%8A%A4%ED%94%84%EB%A7%81%EB%B6%80%ED%8A%B8-Annotation-%EA%B0%9C%EB%85%90-%EB%B0%8F-%EC%A0%95%EB%A6%AC#%20%40RequestMapping-1)

**호출하는 클라이언트의 정보를 가져다가 서버(controller)에 전달해주는 매핑**

@RequestMapping은 [서버]에서 디스페처서블릿을 통해

[클라이언트]html의 action태그의 주소와 동일한 문자열을 찾는 매핑기능(연결)이 실행되고 하단에 메서드가 실행

**쉽게 말하자면 요청이 왔을 때 어떤 컨트롤러가 호출이 되어야 하는지 알려주는 지표 같은 것이다.**

@RequestMapping(value = "/hello-basic")

이렇게 매핑을 하면 localhost:8080/hello-basic으로 url을 입력했을 경우에 이것에 해당하는 메서드가 실행된다.

**@RequestMapping은 다중요청도 가능하다** 다중 요청을 하려면 배열로 묶어야 한다.

@RequestMapping(value = {"/hello", "/hello-basic"})

이것처럼 다중 요청을 할 경우에는 둘 중 아무 url이나 입력해도 된다.

**@RequestMapping에서 가장 많이사용하는 부분은 value와 method이다. (더 많지만 여기서는 여기까지만)**

**value는 요청받을 url을 설정하게 된다.**

**method는 어떤 요청으로 받을지 정의하게 된다.(GET, POST, PUT, DELETE 등)**

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

예시를 간단하게 들어보면 이런식으로 가능하다.

그런데 만약 /hello라는 내용으로 GET, POST, PUT, DELETE를 만들려면 어떻게 해야할까?

RestController

public class HelloController {

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

public String helloGet(...) {

...

}

@RequestMapping(value = "/hello", method = RequestMethod.POST)

public String helloPost(...) {

...

}

@RequestMapping(value = "/hello", method = RequestMethod.PUT)

public String helloPut(...) {

...

}

@RequestMapping(value = "/hello", method = RequestMethod.DELETE)

public String helloDelete(...) {

...

}

이런식으로도 만들 수 있다.

그렇지만 불편하다는 생각이 든다.

**이러한 내용을 해결하기 위해 아래와 같은 해결방식이 있다.**

@RestController

@RequestMapping(value = "/hello")

public class HelloController {

@GetMapping()

public String helloGet(...) {

...

}

@PostMapping()

public String helloPost(...) {

...

}

@PutMapping()

public String helloPut(...) {

...

}

@DeleteMapping()

public String helloDelete(...) {

...

}

}

공통적인 url은 class에 @RequestMapping으로 설정을 해주었다.

그리고 @GetMapping, @PostMapping, @PutMapping, @DeleteMapping으로 간단하게 생략이 가능해졌다.

뒤에 추가적으로 url을 붙이고 싶다면 @GetMapping, @PostMapping, @PutMapping, @DeleteMapping 에 추가적인 url을 작성하면 된다.

@RestController

@RequestMapping(value = "/hello")

public class HelloController {

@GetMapping("/hi")

public String helloGetHi(...) {

...

}

}

위에있는 helloGetHi에 들어가기 위해서는 /hello/hi로 들어가야 한다.

**추가적인 내용으로 @RequestMapping은 Class와 Method에 붙일 수 있고**

**@GetMapping, @PostMapping, @PutMapping, @DeleteMapping들은 Method에만 붙일 수 있다.**

*\*GetMapping 과 PostMapping 은 @RequestMapping의 자식클래스이다*

## [**@RequestParam**](https://rebornbb.tistory.com/entry/SpringBoot-%EC%8A%A4%ED%94%84%EB%A7%81%EB%B6%80%ED%8A%B8-Annotation-%EA%B0%9C%EB%85%90-%EB%B0%8F-%EC%A0%95%EB%A6%AC#%40RequestParam-1)

@RequestParam은 사용자가 전달하는 값을 1:1로 매핑해주는 어노테이션이다.

보통 파라미터를 통해 값을 전달할 때 자주 사용한다.

**(1) 파라미터 이름을 지정하고 받기**

@Slf4j

@Controller

public class request\_mapping {

@RequestMapping("/request\_mapping")

public String visit(@RequestParam(name = "name") String name,

@RequestParam(name = "age") Integer age,

Model model) {

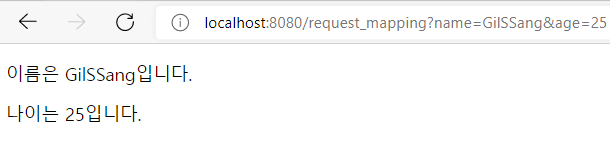
model.addAttribute("name", name);

model.addAttribute("age", age);

return "home";

}

}



* name이라는 파라미터의 값을 name에 저장한다.
* age라는 파라미터의 값을 age에 저장한다.

**(2) 파라미터 이름을 생략하고 받기**

@Slf4j

@Controller

public class request\_mapping {

@RequestMapping("/request\_mapping")

public String visit(@RequestParam String name,

@RequestParam Integer age,

Model model) {

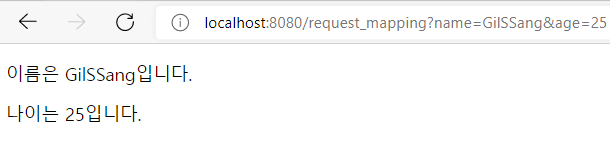
model.addAttribute("name", name);

model.addAttribute("age", age);

return "home";

}

}



* **파라미터의 변수명과 저장 변수명이 동일해야함**

**@RequestParam 파라미터**

**(1) name**

@Slf4j

@Controller

public class request\_mapping {

@RequestMapping("/request\_mapping")

public String visit(@RequestParam(name = "name") String name,

@RequestParam(name = "age") Integer age,

Model model) {

model.addAttribute("name", name);

model.addAttribute("age", age);

return "home";

}

}

* 위에서 알아본 것과 같이 파라미터의 이름을 지정해주는 것이다.

**(2) required**

@Slf4j

@Controller

public class request\_mapping {

@RequestMapping("/request\_mapping")

public String visit(@RequestParam(required = true) String name,

@RequestParam(required = false) Integer age,

Model model) {

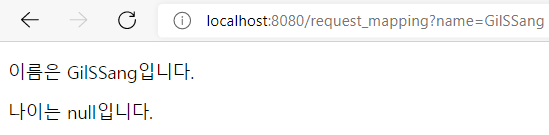
model.addAttribute("name", name);

model.addAttribute("age", age);

return "home";

}

}



* **해당 파라미터가 필수요소인지를 지정해준다.**
* required=true가 default 값이다.
* required=false인데 값이 들어오지 않았다면 null이 들어온다.

**(3) defaultValue**

@Slf4j

@Controller

public class request\_mapping {

@RequestMapping("/request\_mapping")

public String visit(@RequestParam(required = true) String name,

@RequestParam(required = false, defaultValue = "0") Integer age,

Model model) {

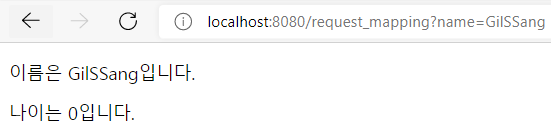
model.addAttribute("name", name);

model.addAttribute("age", age);

return "home";

}

}



* 해당 파라미터가 비어있을 때, default 값을 지정해준다.

**@RequestParam Map으로 조회**

@Slf4j

@Controller

public class request\_mapping {

@RequestMapping("/request\_mapping")

public String visit(@RequestParam Map<String, Object> param,

Model model) {

for (String p : param.keySet()) {

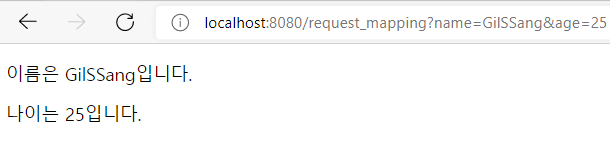
model.addAttribute(p, param.get(p));

}

return "home";

}

}



* 파라미터 값들을 Map으로 받는다.

### 

## [**@RequestBody**](https://rebornbb.tistory.com/entry/SpringBoot-%EC%8A%A4%ED%94%84%EB%A7%81%EB%B6%80%ED%8A%B8-Annotation-%EA%B0%9C%EB%85%90-%EB%B0%8F-%EC%A0%95%EB%A6%AC#%40RequestBody-1)

**Body에 전달되는 데이터를 메소드의 인자와 매칭시켜, 데이터를 받아서 처리할 수 있는 Annotation으로 아래와 같이 사용합니다.** 클라이언트가 보내는 HTTP 요청 본문(JSON 및 XML 등)을 Java 오브젝트로 변환합니다. 아래와 같이 사용합니다.

클라이언트가 body에 **json or xml** 과 같은 형태로 형태로 값(주로 객체)를 전송하면, 해당 내용을 Java Object로 변환합니다.

@Controller // 이 Class는 Controller 역할을 합니다

@RequestMapping("/user") // 이 Class는 /user로 들어오는 요청을 모두 처리합니다.

public class UserController {

@RequestMapping(method = RequestMethod.POST)

public String addUser(@RequestBody User user) {

// POST method, /user 요청을 처리

String sub\_name = user.name;

String sub\_old = user.old;

}

}

### 

## [**@GetMapping**](https://rebornbb.tistory.com/entry/SpringBoot-%EC%8A%A4%ED%94%84%EB%A7%81%EB%B6%80%ED%8A%B8-Annotation-%EA%B0%9C%EB%85%90-%EB%B0%8F-%EC%A0%95%EB%A6%AC#%40GetMapping-1)

RequestMapping(Method=RequestMethod.GET)과 똑같은 역할을 하며, 아래와 같이 사용합니다.

@Controller // 이 Class는 Controller 역할을 합니다

@RequestMapping("/user") // 이 Class는 /user로 들어오는 요청을 모두 처리합니다.

public class UserController {

@GetMapping("/")

public String getUser(Model model) {

// GET method, /user 요청을 처리

}

////////////////////////////////////

// 위와 아래 메소드는 동일하게 동작합니다. //

////////////////////////////////////

@RequestMapping(method = RequestMethod.GET)

public String getUser(Model model) {

// GET method, /user 요청을 처리

}

}

### 

## [**@PostMapping**](https://rebornbb.tistory.com/entry/SpringBoot-%EC%8A%A4%ED%94%84%EB%A7%81%EB%B6%80%ED%8A%B8-Annotation-%EA%B0%9C%EB%85%90-%EB%B0%8F-%EC%A0%95%EB%A6%AC#%40PostMapping-1)

RequestMapping(Method=RequestMethod.POST)과 똑같은 역할을 하며, 아래와 같이 사용합니다.

@Controller // 이 Class는 Controller 역할을 합니다

@RequestMapping("/user") // 이 Class는 /user로 들어오는 요청을 모두 처리합니다.

public class UserController {

@RequestMapping(method = RequestMethod.POST)

public String addUser(Model model) {

// POST method, /user 요청을 처리

}

////////////////////////////////////

// 위와 아래 메소드는 동일하게 동작합니다. //

////////////////////////////////////

@PostMapping('/')

public String addUser(Model model) {

// POST method, /user 요청을 처리

}

}

## [**@ModelAttribute**](https://rebornbb.tistory.com/entry/SpringBoot-%EC%8A%A4%ED%94%84%EB%A7%81%EB%B6%80%ED%8A%B8-Annotation-%EA%B0%9C%EB%85%90-%EB%B0%8F-%EC%A0%95%EB%A6%AC#%40ModelAttribute-1)

클라이언트가 전송하는 HTTP parameter, Body 내용을 Setter 함수를 통해 1:1로 객체에 데이터를 연결(바인딩)합니다. RequestBody와 다르게 HTTP Body 내용은 **multipart/form-data** 형태를 요구합니다. @RequestBody가 json을 받는 것과 달리 @ModenAttribute 의 경우에는 json을 받아 처리할 수 없습니다.

## [**@ResponseBody**](https://rebornbb.tistory.com/entry/SpringBoot-%EC%8A%A4%ED%94%84%EB%A7%81%EB%B6%80%ED%8A%B8-Annotation-%EA%B0%9C%EB%85%90-%EB%B0%8F-%EC%A0%95%EB%A6%AC#%40ResponseBody-1)

@ResponseBody은 메소드에서 리턴되는 값이 View 로 출력되지 않고 HTTP Response Body에 직접 쓰여지게 됩니다. return 시에 json, xml과 같은 데이터를 return 합니다.

@Controller // 이 Class는 Controller 역할을 합니다

@RequestMapping("/user") // 이 Class는 /user로 들어오는 요청을 모두 처리합니다.

public class UserController {

@RequestMapping(method = RequestMethod.GET)

@ResponseBody

public String getUser(@RequestParam String nickname, @RequestParam(name="old") String age {

// GET method, /user 요청을 처리

// https://naver.com?nickname=dog&old=10

User user = new User();

user.setName(nickname);

user.setAge(age);

return user;

}

}

### 

## [**@Autowired**](https://rebornbb.tistory.com/entry/SpringBoot-%EC%8A%A4%ED%94%84%EB%A7%81%EB%B6%80%ED%8A%B8-Annotation-%EA%B0%9C%EB%85%90-%EB%B0%8F-%EC%A0%95%EB%A6%AC#%40Autowired-1)

Spring Framework에서 Bean 객체를 주입받기 위한 방법은 크게 아래의 3가지가 있습니다.**Bean을 주입받기 위하여 @Autowired 를 사용합니다.** Spring Framework가 Class를 보고 Type에 맞게(Type을 먼저 확인 후, 없으면 Name 확인) Bean을 주입합니다.

* **@Autowired**
* **생성자 (@AllArgsConstructor 사용)**
* **setter**

### 

## [**@SpringBootTest**](https://rebornbb.tistory.com/entry/SpringBoot-%EC%8A%A4%ED%94%84%EB%A7%81%EB%B6%80%ED%8A%B8-Annotation-%EA%B0%9C%EB%85%90-%EB%B0%8F-%EC%A0%95%EB%A6%AC#%40SpringBootTest-1)

Spring Boot Test에 필요한 의존성을 제공해줍니다.

// DemoApplicationTests.java

package com.example.demo;

import org.junit.jupiter.api.Test;

import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest

class DemoApplicationTests {

@Test

void contextLoads() {

}

}

## [**@Test**](https://rebornbb.tistory.com/entry/SpringBoot-%EC%8A%A4%ED%94%84%EB%A7%81%EB%B6%80%ED%8A%B8-Annotation-%EA%B0%9C%EB%85%90-%EB%B0%8F-%EC%A0%95%EB%A6%AC#%40Test-1)

JUnit에서 테스트 할 대상을 표시합니다.

### [**GET과 POST의 차이**](https://rebornbb.tistory.com/entry/SpringBoot-%EC%8A%A4%ED%94%84%EB%A7%81%EB%B6%80%ED%8A%B8-Annotation-%EA%B0%9C%EB%85%90-%EB%B0%8F-%EC%A0%95%EB%A6%AC#GET%EA%B3%BC%20POST%EC%9D%98%20%EC%B0%A8%EC%9D%B4-1)

**HTTP**

GET과 POST를 설명하기 전에 HTTP부터 알고 넘어가자.  
네이버 지식백과에서는 HTTP를 다음과 같이 설명하고 있다.

http는 1989년 팀 버너스 리(Tim Berners Lee)에 의하여 처음 설계되어 인터넷을 통한 월드 와이드 웹(World-Wide Web) 기반에서 전 세계적인 정보공유를 이루는데 큰 역할을 하였다. http의 첫번째 버전은 인터넷을 통하여 가공되지 않은 데이터를 전송하기 위한 단순한 프로토콜이었으나, 데이터에 대한 전송과 요구·응답에 대한 수정 등 가공된 정보를 포함하는 프로토콜로 개선되었다.  
[네이버 지식백과] HTTP [hypertext transfer protocol](https://velog.io/@youngblue/%EB%91%90%EC%82%B0%EB%B0%B1%EA%B3%BC)

즉, 인**터넷에서 웹 서버와 사용자의 인터넷 브라우저 사이에 문서를 전송하기 위해 사용되는 통신 규약으로 데이터의 전송과 요구·응답 등에 대한 수행**등이 일어난다.  
이때**HTTP의 메소드를 이용**해 수행이 일어나게 되는데 , **GET과 POST가 HTTP 메소드에 속한다.**

#### 

**GET 이란?**

**GET 은 클라이언트에서 서버로 어떠한 리소스로 부터 정보를 요청하기 위해 사용되는 메서드이다.**

예를들면 게시판의 게시물을 조회할 때 쓸 수 있다.

**GET을 통한 요청은 URL 주소 끝에 파라미터로 포함되어 전송되며, 이 부분을 쿼리 스트링 (query string) 이라고 부른다.**

**방식은 URL 끝에 ? 를 붙이고 그다음 변수명1=값1&변수명2=값2... 형식으로 이어 붙이면 된다.**

예를들어 다음과 같은 방식이다.

www.example.com/show?name1=value1&name2=value2

서버에서는**name1 과 name2 라는 파라미터 명**으로 각각 **value1 과 value2 의 파라미터 값**을 전달 받을 수 있다.

**GET의 특징**

* **GET 요청은 캐시가 가능하다.**
* : GET을 통해 서버에 리소스를 요청할 때 웹 캐시가 요청을 가로채 서버로부터 리소스를 다시 다운로드하는 대신 리소스의 복사본을 반환한다. HTTP 헤더에서 cache-control 헤더를 통해 캐시 옵션을 지정할 수 있다.
* **GET 요청은 브라우저 히스토리에 남는다.**
* **GET 요청은 북마크 될 수 있다.**
* **GET 요청은 길이 제한이 있다.**
* : GET 요청의 길이 제한은 표준이 따로 있는건 아니고 브라우저마다 제한이 다르다고 한다.
* **GET 요청은 중요한 정보를 다루면 안된다. ( 보안 )**
* : GET 요청은 파라미터에 다 노출되어 버리기 때문에 최소한의 보안 의식이라 생각하자.
* **GET은 데이터를 요청할때만 사용 된다.**

**POST 란?**

**POST는 클라이언트에서 서버로 리소스를 생성하거나 업데이트하기 위해 데이터를 보낼 때 사용 되는 메서드다.**

예를들면 게시판에 게시글을 작성하는 작업 등을 할 때 사용할 된다.

**POST는 전송할 데이터를 HTTP 메시지 body 부분에 담아서 서버로 보낸다.**

**( body 의 타입은 Content-Type 헤더에 따라 결정 된다.)**

GET에서 URL 의 파라미터로 보냈던 name1=value1&name2=value2 가 body에 담겨 보내진다 생각하면 된다.

**POST 로 데이터를 전송할 때 길이 제한이 따로 없어 용량이 큰 데이터를 보낼 때 사용하거나**

**GET처럼 데이터가 외부적으로 드러나는건 아니라서 보안이 필요한 부분에 많이 사용된다.**

( 하지만 데이터를 암호화하지 않으면 body의 데이터도 결국 볼 수 있는건 똑같다. )

POST를 통한 **데이터 전송은 보통 HTML form 을 통해 서버로 전송**된다.

**POST의 특징**

* **POST 요청은 캐시되지 않는다.**
* **POST 요청은 브라우저 히스토리에 남지 않는다.**
* **POST 요청은 북마크 되지 않는다.**
* **POST 요청은 데이터 길이에 제한이 없다.**

**GET 과 POST 의 차이점**

* **사용목적** : GET은 서버의 리소스에서 데이터를 요청할 때, POST는 서버의 리소스를 새로 생성하거나 업데이트할 때 사용한다.
* DB로 따지면 GET은 SELECT 에 가깝고, POST는 Create 에 가깝다고 보면 된다.
* **요청에 body 유무** : GET 은 URL 파라미터에 요청하는 데이터를 담아 보내기 때문에 HTTP 메시지에 body가 없다. POST 는 body 에 데이터를 담아 보내기 때문에 당연히 HTTP 메시지에 body가 존재한다.
* **멱등성 (idempotent)** : GET 요청은 멱등이며, POST는 멱등이 아니다.

**\*멱등이란?**

**멱등의 사전적 정의는 연산을 여러 번 적용하더라도 결과가 달라지지 않는 성질을 의미**한다.

**GET은 리소스를 조회한다는 점에서 여러 번 요청하더라도 응답이 똑같을 것 이다.**

**반대로 POST는 리소스를 새로 생성하거나 업데이트할 때 사용되기 때문에 멱등이 아니라고 볼 수 있다.**

**(POST 요청이 발생하면 서버가 변경될 수 있다.)**

GET과 POST는 이런 차이들이 있기 때문에 사용하려는 목적에 맞는 확인한 후에 사용해야한다.

GET과 POST 이외에도 PUT , DELETE 등을 적절히 사용하는게 좋은데 예를들어 봇의 경우에 사이트를 돌아다니면서 GET 요청을 날린다. 이럴 때 DELETE 등을 GET으로 처리하면 봇에 의해 서버에 있는 리소스들이 삭제 되는 상황이 일어 날수 있다!

정리 출처 : [hongsii github](https://hongsii.github.io/2017/08/02/what-is-the-difference-get-and-post)

[정리 출처 :https://noahlogs.tistory.com/35](https://noahlogs.tistory.com/35)

## [**Lombok의 대표적인 Annotation과 역할**](https://rebornbb.tistory.com/entry/SpringBoot-%EC%8A%A4%ED%94%84%EB%A7%81%EB%B6%80%ED%8A%B8-Annotation-%EA%B0%9C%EB%85%90-%EB%B0%8F-%EC%A0%95%EB%A6%AC#Lombok%EC%9D%98%20%EB%8C%80%ED%91%9C%EC%A0%81%EC%9D%B8%20Annotation%EA%B3%BC%20%EC%97%AD%ED%95%A0-1)

Lombok은 코드를 크게 줄여주어 가독성을 크게 높힐 수 있는 라이브러리입니다. 대표적인 Annotation은 아래와 같습니다.

**@Setter**

Class 모든 필드의 Setter method를 생성해줍니다.

**@Getter**

Class 모든 필드의 Getter method를 생성해줍니다.

**@AllArgsConstructor**

Class 모든 필드 값을 파라미터로 받는 생성자를 추가합니다.

**@NoArgsConstructor**

Class 기본 생성자를 자동으로 추가해줍니다.

**@ToString**

Class 모든 필드의 toString method를 생성한다.