-Trong java có 2 loại annotation:

+dùng của người khác viết(cụ thể spring framework,jakata…)

+tự mình viết

-@controller : chuyển từ 1 java class thông thường qua 1 restful API web service

+Nghĩa là khi ta chạy file thì nó quét đến “@controller” lập tức nó sẽ hiểu đây là 1 restful API web service

+Là nơi tiếp nhận những dữ liệu từ phía client đổ về có thể là thông qua thanh param hoặc body

-HTTP method():

GET

PUT / POST

DELETE

UPDATE

@ResponseBody cho phép ta chuyển đổi data từ phía BE có dạng key-value (Map,java-Bean…) về dạng Json trả về phí FE

Thằng Controller khác với RestController ở chỗ thằng

Restcontroller = Controller + ResponseBody.

@RequestMapping là một annotation được sử dụng để ánh xạ các yêu cầu HTTP tới các phương thức xử lý tương ứng trong một controller. Annotation này được sử dụng để xác định URL mà phương thức controller sẽ xử lý.

2 đối số:

* value: Chỉ định URL được ánh xạ.

method: Chỉ định phương thức HTTP được hỗ trợ.(vd:method=RequestMethod.***GET)***

=> Chức năng chính:

* Xác định URL: Nó chỉ định URL mà khi người dùng truy cập sẽ kích hoạt phương thức xử lý được gắn annotation này.
* Xác định phương thức HTTP: Nó chỉ định phương thức HTTP nào (GET, POST, PUT, DELETE, v.v.) mà phương thức xử lý sẽ phản hồi.
* Truyền tham số: Nó cho phép truyền các tham số từ URL tới phương thức xử lý.

**@GetMapping, @PostMapping, @PutMapping, @DeleteMapping**

-4 thằng này đều có chức năng giống với requestmapping nhưng khác ở chỗ không cần đến đối số thứ 2 để xác định phương thức HTTP được hỗ trợ. Bởi vì, tên của các annotation này đã xác định rõ phương thức HTTP mà chúng ánh xạ.

@ PathVariable chúng ta có thể sử dụng @PathVariable để hứng id của phía FE và xóa nên nó hay dùng trung với @DeleteMapping để đỡ phải khai báo thêm requestmethod vì mặc định nó là delete

khi một yêu cầu DELETE được gửi đến "/codeptit/home/JOJO/123", phương thức "deleteUser" sẽ được gọi với tham số "id" có giá trị là 123 để xử lý yêu cầu xóa người

ta có thể kết hợp với requestparam nhung th này khá hiếm

vd

@DeleteMapping(value="/codeptit/home/JOJO/{id}")

**public** **void** deleteBuilding(

@PathVariable Integer id,

@RequestParam(value=”Addres,required=false) String s

) {

//xu ly database

System.***out***.println("da xoa toa nha "+id+” tinh “+s);

}

@RequestParam

 là một annotation trong Spring được sử dụng để trích xuất giá trị của một tham số từ yêu cầu HTTP.

Khi một yêu cầu HTTP được gửi đến một phương thức xử lý trong một controller, bạn có thể sử dụng @RequestParam để lấy giá trị của một tham số từ các tham số truy vấn (query parameters) hoặc các tham số form (form parameters) của yêu cầu.

Đối số thứ 2 là required nó nếu không có mặc định là true tức là phí fe bắt buộc phải có thằng value=”name nếu không có thì nó sẽ lỗi vậy nếu ta set cho nó bằng false tức là có hay không đều được

-Trường hợp khi nhận dữ liệu từ param trả về mà kiểu dữ liệu của biến nhận là String

+Trường hợp là null nếu không có key

+Trường hợp là xâu rỗng(“”) nếu có key nhưng không có value

http://localhost:8081/codeptit/home/Tree?question=J003&name=hiep&old=20

@RequestMapping(value="/codeptit/home/Tree",method=RequestMethod.***GET***)

**public** **void** home(@RequestParam(value="question",required=**false**) String e,

@RequestParam(value="name",required=**false**)String g,

@RequestParam(value="old",required=**false**)String o) {

System.***out***.print(e+" "+g+" "+o);

}

@RequestBody Khi một yêu cầu HTTP được gửi đến một phương thức Controller trong Spring, @RequestBody được sử dụng để thông báo rằng dữ liệu gửi từ client sẽ được đọc và ánh xạ vào đối tượng được đánh dấu bằng @RequestBody. Thông thường, dữ liệu được gửi dưới dạng JSON hoặc XML và được chuyển đổi tự động thành đối tượng Java tương ứng trên server.

Vậy chẳng nhẽ nếu có nhiều cặp key-value thì mình phải dùng bấy cái requestparam ?

Để giải quyết ta chỉ cần dùng các cấu trúc có dạng json để lưu nó:

**+Map**

**+JavaBean:** là 1 class chỉ bao gồm các thuộc tính bắt buộc phải là private và các hàm geter,setter,ngoài ra không còn gì khác

@RequestMapping(value="/codeptit/home/array",method=RequestMethod.***GET***)

**public** **void** question2(@RequestParam Map<String,String>mp) {

System.***out***.println(mp);

}

Vd:hứng dữ liệu body thay vì param của fe

@RequestMapping(value="/codeptit/home/array",method=RequestMethod.***GET***)

**public** Map<String,String> question2(@RequestBody Map<String,String>mp) {

**return** mp;

}

Htttp status code:

-Giống như 1 trọng tài phân định lỗi giữa Fe và Be

-2XX: Ok

-4XX:Thường do FE

-5XX:Chắc chắn BE

Custom Exception:

-Trong quá trình tạo Chương trình có một vài lỗi mà java cũng không thể bắt đc cái Exception đó như kiểu dữ liệu từ thành param có key là Name=Null mặc dù nó không ảnh hưởng đến chương trình nhưng nếu Ctr của cta bắt nó phải có giá trị thì ta có thể coi nó là 1 lỗi và custom lỗi đó bởi thằng java nó đâu có biết Name là gì đâu ?

@ControllerAdvice

-Có thể được sử dụng để xử lý ngoại lệ. Khi một ngoại lệ được ném ra trong bất kỳ controller nào, Spring MVC sẽ tìm kiếm phương thức xử lý ngoại lệ phù hợp trong lớp @ControllerAdvice và gọi nó.

Hiểu đơn giản là khi code gặp lỗi thì nó sẽ kèm theo 1 Exception và nó sẽ ưu tiên nhảy vào ControllerAdvice để xử lí Exception đó

Để sử dụng @ControllerAdvice để xử lý ngoại lệ, bạn cần định nghĩa các phương thức xử lý ngoại lệ trong lớp @ControllerAdvice của mình. Phương thức xử lý ngoại lệ phải có annotation @ExceptionHandler. Annotation @ExceptionHandler có thể được sử dụng để chỉ định loại ngoại lệ mà phương thức xử lý sẽ xử lý.

-Thằng này sẽ giải quyết vấn đề những lỗi mà Ctr của chúng ta hay chính là Java không hiểu lỗi mà bản thân chúng ta muốn xử lí.

// xử lí http tatus code

@PostMapping(value="/codeptit/home/List")

**public** Object testError(@RequestBody DanhSachDTO x) {

System.***out***.print(5/0); // khi đến đây là nó chết code và nó kèm theo 1 Exception và nó sẽ ưu tiên nhảy vào luồng của ControllerAdvice và sau đó ta cài đặt theo ý muốn.

**return** x;

}

Cài đặt:

@ControllerAdvice

**public** **class** ControllerAdvisor **extends** ResponseEntityExceptionHandler{

@ExceptionHandler(ArithmeticException.**class**) // tên class ta muốn bắt lỗi

**public** ResponseEntity<Object> handleArithmeticException(

ArithmeticException ex, WebRequest request) { // 2 tham số tham số thứ nhất khởi tạo lỗi chúng ta muốn bắt,tham số thứ 2 trả về http status code

ErrorResonseDTO errorResonseDTO=**new** ErrorResonseDTO();

errorResonseDTO.setError(ex.getMessage());

List<String> details=**new** ArrayList<String>();

details.add("So nguyen khong chia cho 0 duoc");

errorResonseDTO.setDetail(details);

**return** **new** ResponseEntity<>(errorResonseDTO, HttpStatus.***INTERNAL\_SERVER\_ERROR***); // 2 đối số là chi tiết lỗi và http status code chuẩn(ta tự tìm cái tương ứng với lỗi ta gặp phải)

}

}

Restful api design :13 rule hãy code the phong cách của 13 cái rule này

## **1. Learn the basics of HTTP applied to REST**

-rule thứ 1 trong 13 rule: khi gửi yêu cầu thông qua phương thức http method thì ở phía client và server phải chung Phương thức hoặc cùng là GET,cùng Là DELETE mặc tên phương thức không phản ánh chức năng của nó vd ta có thể dùng Delete hoặc update để dữ liệu miễn sao thằng gửi và nhận cùng chung phương thức, nhưng như thế nó sẽ ko đúng rule và ko clean code… , ,vậy nên BE và FE phải quy định trước cái api này quy định là phương thức gì.

<http://localhost:8081/codeptit/home/array?question=J003&name=hiep&old=20>

-ở đây phương thức http sẽ gửi một phương thức lên server cụ thể ở đây là localhost:8081 nếu sau này ta đẩy lên sever thật thì đó chính là tên miền,tiếp theo nó sẽ tìm đến api được yêu cầu ở đây là codeptit/home/array tiếp theo là đến các fill thì ở đây key của bên client trên thanh param phải khớp với bên BE(liên quan đến các anotation request)

## **2. Don't return plain text**

-Rule thứ 2 trong 13 rule: không nên trả dữ liệu dưới dạng text mà nên trả về dạng JSON

## **3. Avoid using verbs in URIs**

-Rule thứ 3 trong 13 rule: không sử dụng động từ đặt tên cho api

+ “/api/Getbuilding/” hoăc “/api/Solvebuilding/” :NO

=>“/api/building/”

Bởi sao lại không nên đặt có động từ bới khi viết api mục đích và công việc của nó đã được định nghĩa bởi http method :get,put,….

## **4. Use plural resource nouns**

Rule thứ 4 trong 13 rule:khi thao tác số nhiều nên sử dụng danh từ số nhiều cho tài nguyên

## Vd: /articles/2/

## 5. Return error details in the response body

-Nếu gặp lỗi hãy trả về chi tiết lỗi đó

-Khi máy chủ API xử lý lỗi, việc trả về chi tiết lỗi trong nội dung JSON sẽ rất thuận tiện (và được khuyến nghị!) để giúp người dùng gỡ lỗi. Lời khen ngợi đặc biệt nếu bạn bao gồm những trường nào bị ảnh hưởng bởi lỗi!

{

"error": "Invalid payoad.", // lỗi gì

"detail": {

"surname": "This field is required." //mô tả tri tiết lỗi đó

}

}

Vd: **public** Object testError(@RequestParam Map<String,String> x) {

**try** {

System.***out***.print(5/0);

}

**catch**(Exception e) {

ErrorResonseDTO errorResonseDTO=**new** ErrorResonseDTO();

errorResonseDTO.setError(e.getMessage());

List<String> details=**new** ArrayList<>();

details.add("So nguyen lam sao chia cho so 0 duoc ?");

errorResonseDTO.setDetail(details);

**return** errorResonseDTO;

}

**return** x;

}

-ta có thể sử dụng Try catch để khác phục tình trạng lỗi bởi nó nhảy vào phần catch nhưng như thế sẽ bị kết thúc Ctr nếu ta làm như trên mặc dù đã đúng rule thứ 5 nhưng nó sẽ ko thực hiện được các future bên dưới và trả về ko đúng Http status code nó vẫn hiện là 200 mặc dù ctr lỗi(chính là rule thứ 6), để khắc phục chúng ta cần tìm hiểu về Controller advice

## 6. Pay attention to status codes

-Điều này là siêu quan trọng. Nếu có một điều bạn cần nhớ từ bài viết này thì có lẽ đó là:

tồi tệ nhất mà API của bạn có thể làm là trả về phản hồi lỗi với mã trạng thái 200 OK (Ctr lỗi ta trả về đầy đủ lỗi và chi tiết lỗi nhưng nó vẫn là 200).

## **8. Don't nest resources**

Rule thứ 8 trong 13 rule:không lồng tài nguyên vào api

Vd: lấy những tác phẩm của tác giả có id là 12

GET: /authors/12/aticles/ (NO) đọc gây khó hiểu

=>GET: /aticles/?author\_id=12/

=>FE và BE cần thống nhất API viết như nào, có chức năng mục đích gì

Và nó liên quan đếm mối quan hệ nhiều nhiều ,một nhiều …

-khi ta thiêt kế thì ta thường để thực thể mạnh ở trước

## 11. Learn the difference between 401 Unauthorized and 403 Forbidden

Lỗi 401: Quá thời hạn truy cập :vd 1 trang web họ con server có 1000 chỗ khi đã quá tải mà một trong số tài khoản đó mà lâu không thao tác mà họ đặt khoảng 30p hay 1t gì đó thì ta sẽ bị out ra

Lỗi 403:Truy cập vào api chưa được phân quyền