1. **Các bước tạo project spring boot web :**

**B1**: Tạo project maven

**B2**: Thêm dependency spring-boot-starter-web

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

<version>2.1.1.RELEASE</version>

</dependency>

spring-boot-starter-web đã bao gồm spring, spring-web, spring-webmvc, tomcat nên không cần add server tomcat ngoài cho project nữa.

**B3**: Tạo class main như dưới:

@SpringBootApplication

**public** **class** SpringBootRestApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(SpringBootRestApplication.**class**, args);

}

}

Annotation @SpringBootApplication tương đương với việc dùng kết hợp cả 3 annotation @Configuration, @EnableAutoConfiguration, @ComponentScan. Mặc định nó sẽ scan các component trong cùng package và ở các package con.

1. **Tạo các controller**

* Controller là class dùng để tiếp nhận request và trả về response cho client

@RestController

**public** **class** HelloWorld {

@RequestMapping("/")

**public** String hello() {

**return** "Hello World";

}

}

* Annotation @RestController đã gộp 2 annotation @Controller và @ResponseBody vào làm một. Annotation @ResponseBody tự động chuyển object của java thành dạng json trước khi trả về cho client => Content-Type: application/json. Nếu không có annotation này hệ thống sẽ báo lỗi.
* Khi muốn trả về status code cụ thể cho client, dùng annotation @ResponseStatus(HttpStatus.xxx)
* Annotation @RequestBody: gắn vào 1 tham số của method dùng để map dữ liệu người dùng gửi lên với Object của java.
* Annotation @RequestMapping…. method = RequestMethod.POST
* Annotation @PathVariable: dùng để lấy các tham số trên url

@RequestMapping("/hello/{id}")

**public** String helloPathVariable(@PathVariable("id") String id) {

**return** String.*format*("Hello World, your id is %s", id);

}

* ResponseEntity …

1. **Xử lý lỗi trong spring**
   1. Xử lý exception ở controller bằng annotation @ExceptionHandler

@PostMapping("/post-user2")

**public** String postUser2(@RequestBody User user) **throws** PasswordValidateException {

**if** (user.getPassWord().length() < 8) {

**throw** **new** PasswordValidateException("password must be at least 8 characters");

}

**return** user.toString();

}

@ExceptionHandler(PasswordValidateException.**class**)

**public** ResponseEntity<ErrorResponse> handleException(PasswordValidateException ex) {

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

details.add(ex.getLocalizedMessage());

ErrorResponse error = **new** ErrorResponse("Validation errors", details);

**return** **new** ResponseEntity<ErrorResponse>(error, HttpStatus.***NOT\_FOUND***);

}

* Chỉ có tác dụng trong controller đó
  1. Xử lý exception bằng annotation @ControllerAdvice (cấp độ global)

@ControllerAdvice

**public** **class** GlobalHandleException **extends** ResponseEntityExceptionHandler {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

@ExceptionHandler(ValidateException.**class**)

**public** ResponseEntity<ErrorResponse> resourceNotFoundException(ValidateException ex) {

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

details.add(ex.getLocalizedMessage());

ErrorResponse errorDetails = **new** ErrorResponse("validation errors", details);

**return** **new** ResponseEntity<ErrorResponse>(errorDetails, HttpStatus.***BAD\_REQUEST***);

}

@Override

**protected** ResponseEntity<Object> handleMethodArgumentNotValid(

MethodArgumentNotValidException ex,

HttpHeaders headers,

HttpStatus status,

WebRequest request) {

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

**for** (FieldError error : ex.getBindingResult().getFieldErrors()) {

errors.add(error.getField() + ": " + error.getDefaultMessage());

}

**for** (ObjectError error : ex.getBindingResult().getGlobalErrors()) {

errors.add(error.getObjectName() + ": " + error.getDefaultMessage());

}

ErrorResponse errorDetails = **new** ErrorResponse("validation errors", errors);

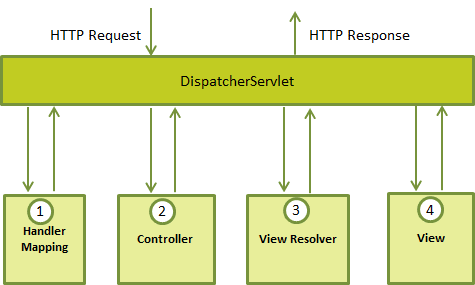
**return** **new** ResponseEntity<Object>(errorDetails, HttpStatus.***BAD\_REQUEST***);

}

}

**4. Debug Spring boot**

logging.level.org.springframework = DEBUG



**5. Validate trong spring**

- Dùng annotation @Valid với request body

- Một số validation annotation thông dụng (javax.validation): @NotBlank, @Size,

6. Filtering Field

- Khi muốn ẩn 1 field của class (bean) khi trả về cho người dùng, ta thêm annotation @JsonIgnore trước field đó.

@JsonIgnore

**private** String field3;

* Khi muốn ẩn field tùy chỉnh, các request khác nhau thì trả về các field khác nhau ta dùng annotation @JsonFilter

@JsonFilter("filterField")

**public** **class** DynamicFilterBean {}

@GetMapping("/filter/dynamic1")

**public** MappingJacksonValue dynamicFilter1() {

DynamicFilterBean bean = **new** DynamicFilterBean("field1", "field2", "field3");

SimpleBeanPropertyFilter filterField = SimpleBeanPropertyFilter.*filterOutAllExcept*("field3");

FilterProvider filters = **new** SimpleFilterProvider().addFilter("filterField", filterField);

MappingJacksonValue mappingJacksonValue = **new** MappingJacksonValue(bean);

mappingJacksonValue.setFilters(filters);

**return** mappingJacksonValue;

}

**6. Lấy các giá trị trong file application.properties**

Cách 1: Dùng class org.springframework.core.env.Environment;

@Autowired

Environment enviroment;

enviroment.getProperty("user-setting.minLengthOfUsername")

Cách 2: Dùng annotation org.springframework.beans.factory.annotation.Value (dùng với thuộc tính của class)

@Value("${user-setting.maxLengthOfUsername}")

String maxLengthOfUsername;

Cách 3: Dùng annotation @ConfigurationProperties (dùng với class)

@Component

@ConfigurationProperties(prefix="user-setting")

**public** **class** UserConfig {

**private** **int** maxAge;

**private** **int** minAge;

**public** **int** getMaxAge() {

**return** maxAge;

}

**public** **void** setMaxAge(**int** maxAge) {

**this**.maxAge = maxAge;

}

**public** **int** getMinAge() {

**return** minAge;

}

**public** **void** setMinAge(**int** minAge) {

**this**.minAge = minAge;

}

}

@Component

@ConfigurationProperties

**public** **class** PaginationConfig {

**private** **int** itemsPerPage;

**public** **int** getItemsPerPage() {

**return** itemsPerPage;

}

**public** **void** setItemsPerPage(**int** itemsPerPage) {

**this**.itemsPerPage = itemsPerPage;

}

}

Sau đó autowired class config này để sử dụng:

@Autowired

UserConfig userConfig;

@Autowired

PaginationConfig paginationConfig;

@GetMapping("/get-config-1")

**public** String getConfig1() {

**return** String.*format*("max age = %s, min age = %s, itemsPerPage = %s",

userConfig.getMaxAge(),

userConfig.getMinAge(),

paginationConfig.getItemsPerPage());

}

================================================

**TẠO SPRING CLOUD CONFIG SERVER**

* Quản lý tập trung cấu hình của các service con
* Config Server lưu trữ cấu hình thông qua git repo nên cần tạo 1 git repo trước

B1: Tạo 1 git repository với các file xxx.properties, trong đó xxx là tên các service muốn quản lý cấu hình tập trung

spring.application.name=user-service

user.username.maxLength=11

user.username.minLength=1

B2: Tạo spring boot starter với dependency là “Config Server”

B3: Thêm annotation @EnableConfigServer trước class main

B4: Cấu hình config server:

spring.application.name=spring-cloud-config-server

server.port=8888

spring.cloud.config.server.git.uri=git-url\*

\*git-url: là đường dẫn đến repo ở B1

B5: Truy cập vào địa chỉ để xem cấu hình của các service con

<http://localhost:8888/xxx/default> , trong đó xxx là tên service (tên file xxx.properties)

Ta có thể link folder git repo vào project spring server config cho tiện quản lý bằng cách click chuột phải vào project, chọn Build Path / Configure build path / Source / Link source…

Ngoài ra mỗi service có thể có các profile (env) khác nhau (dev, qa, stage, production…), để tạo các file cấu hình cho các profile khác nhau ta tạo file .properties với quy tắc xxx-yy.pro

**Cách lấy config server từ các service**

B1: Tạo project service (dùng spring boot với các dependency là Config Client, Web)

B2: Tạo file bootstrap.properties (lưu ý file này sẽ đc load trước file application.properties) với nội dung như sau:

spring.application.name=user-service

spring.cloud.config.uri=http://localhost:8888

B3: Tạo Rest Controller

@RestController

**public** **class** DemoConfig {

@Autowired

Environment enviroment;

@GetMapping("/config")

**public** UsernameConfig getConfigFromServerConfig() {

**return** **new** UsernameConfig(

Integer.*parseInt*(enviroment.getProperty("user.username.minLength")),

Integer.*parseInt*(enviroment.getProperty("user.username.maxLength")));

}

}

***Lưu ý:*** Để thay đổi profile trong spring ta thêm cấu hình sau vào file bootstrap.properties

spring.profiles.active=qa

================================================

**REST CLIENT REQUEST TRONG SPRING BOOT**

**Dùng RestTemplate:**

B1: Tạo class POJO để chứa dữ liệu cần lấy về:

@Data

@JsonIgnoreProperties(ignoreUnknown = **true**)

**public** **class** Post {

**private** **int** id;

@JsonProperty("title")

**private** String titlePost;

**private** String author;

**public** Post() {

}

**public** Post(String titlePost, String author) {

**this**.titlePost = titlePost;

**this**.author = author;

}

}

B2: Tạo bean RestTemplate để sử dụng cho nhiều nơi:

@Configuration

**public** **class** RestClientConfig {

@Bean

**public** RestTemplate restTemplate(RestTemplateBuilder builder) {

**return** builder.build();

}

}

B3: Request url với các phương thức như getForObject, getForEntity, exchange

@RestController

**public** **class** RestTemplateDemo {

@Autowired

**private** RestTemplate restTemplate;

@GetMapping("/rest-client/posts")

**public** List<Post> getAllPost() {

String url = "http://localhost:3000/posts";

ResponseEntity<List> responeEntity = restTemplate.getForEntity(url, List.**class**);

**return** responeEntity.getBody();

}

@GetMapping("/rest-client/posts/{id}")

**public** Post getPost(@PathVariable("id") String id) {

String url = "http://localhost:3000/posts/" + id;

**return** restTemplate.getForObject(url, Post.**class**);

}

@PostMapping("/rest-client/posts")

**public** Post doPost(@RequestBody Post post) {

String url = "http://localhost:3000/posts";

RequestEntity<Post> requestEntity;

**try** {

requestEntity = **new** RequestEntity<>(

post,

HttpMethod.***POST***,

**new** URI(url));

ResponseEntity<Post> exchange = restTemplate.exchange(requestEntity, Post.**class**);

**return** exchange.getBody();

} **catch** (URISyntaxException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**return** **null**;

}

}

**Dùng FeignClient:**

B1: Thêm dependency maven:

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-openfeign</artifactId>

<version>2.1.1.RELEASE</version>

</dependency>

B2: Tạo interface với annotation @FeignClient

@FeignClient(value = "jsonserver", url = "http://localhost:3000/")

**public** **interface** FeignClientService {

@GetMapping("/posts")

List<Post> getPosts();

@GetMapping("/posts/{postId}")

Post getPostById(@PathVariable("postId") Long postId);

}

B3: Thêm annotation @EnableFeignClients ở class main

B4: Autowire interface tạo ở bước 2 và dùng như bt:

@Autowired

**private** FeignClientService feignClient;

@GetMapping("/feign-client/posts")

**public** List<Post> getAllPost() {

**try** {

**return** feignClient.getPosts();

} **catch**(FeignException e) {

System.***out***.println(e.getMessage());

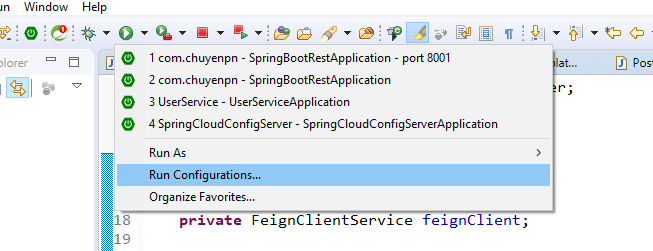
}

**return** **null**;

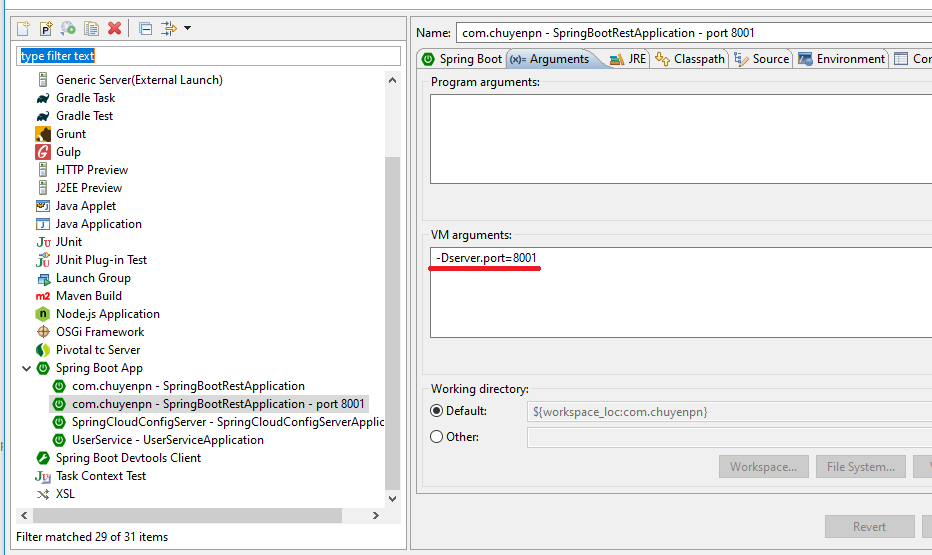
}

**CHẠY APPLICATION SONG SONG Ở CÁC PORT KHÁC NHAU**

B1:



B2:



================================================

**LOAD BALANCING BẰNG RIBBON CLIENT**

B1: Thêm Ribbon Client dependency vào project:

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-ribbon</artifactId>

</dependency>

B2: Thêm annotation @RibbonClient vào interface FeignClient:

@FeignClient(name="feign-client-notification-service")

@RibbonClient(name="feign-client-notification-service")

**public** **interface** FeignClientService {

@GetMapping("/users")

**public** ResponseDTO getAllUsers();

}

B3: Thêm config vào file application.properties

feign-client-notification-service.ribbon.listOfServers=http://localhost:8100, http://localhost:8101

B4: Test lại

@Autowired

FeignClientService feignClient;

@GetMapping("/get-users-from-notification-service")

**public** ResponseDTO getUserFromNotiService() {

**return** feignClient.getAllUsers();

}

Tuy nhiên cách làm trên có phần hard code đó là list các servers ở file properties, vì nếu cần thêm, xóa server thì lại phải thay đổi ở file properties, restart lại service và điều đó làm server bị động khá nhiều. Do đó service registry sinh ra để khắc phục điều này.

================================================

**TẠO SERVICE REGISTRY BẰNG EUREKA NAMING SERVER**

**Tạo Eureka Server**

B1: Thêm dependency vào file pom.xml:

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>

</dependency>

B2: Thêm annotation @EnableEurekaServer vào class main

B3: Thêm config vào file cấu hình applications.properties

server.port=8761

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

B4: Truy cập địa chỉ <http://localhost:8761/>  để xem Eureka dashboard

**Tạo Eureka Client và kết nối đến Eureka Server**

B1: Thêm dependency vào file pom.xml

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

B2: Thêm config vào file application.properties

eureka.client.service-url.default-zone=http://localhost:8761/eureka

B3: Thêm annotation @EnableDiscoveryClient vào class main

B4: Truy cập địa chỉ <http://localhost:8761/> và kiểm tra

***Lưu ý:*** Nếu bị lỗi khi kết nối eureka client đến server thì xóa file application.properties và thay đổi thành file application.yml với nội dung như sau:

eureka:

client:

serviceUrl:

defaultZone: http://localhost:8761/eureka/

================================================

**CÀI ĐẶT API GATEWAY VỚI ZUUL**

B1: Tạo project mới với 2 denpendency sau đây:

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-zuul</artifactId>

</dependency>

B2: Thiết lập cấu hình cho zuul, có kết nối đến eureka naming server

spring.application.name=zuul-server

server.port=8765

eureka.client.service-url.default-zone=http://localhost:8761/eureka

B3: Thêm annotation vào class main

@EnableZuulProxy

@EnableDiscoveryClient

B4: Thay url đến các service cũ bằng url mới theo quy tắc sau

[http://localhost:8765/{ application-name }/{ uri }](http://localhost:8765/%7b%20application-name%20%7d/%7b%20uri%20%7d)

Ví dụ: <http://localhost:8765/notification-service/get-users-from-notification-service>

**CÁCH GỌI MỘT SERVICE TỪ MỘT SERVICE KHÁC THÔNG QUA ZUUL SERVER (KHÔNG THÔNG QUA EUREKA REGISTRY SERVICE)**

Dùng Annotation FeignClient:

@FeignClient(name="{zuul-server-name}")

**public** **interface** FeignClientService {

@GetMapping("/{service-name}/users")

**public** ResponseDTO getAllUsers();

}

**Sửa lỗi dòng đầu tiên file pom.xml:**

<properties>

<java.version>1.8</java.version>

<maven-jar-plugin.version>3.1.1</maven-jar-plugin.version>

</properties>