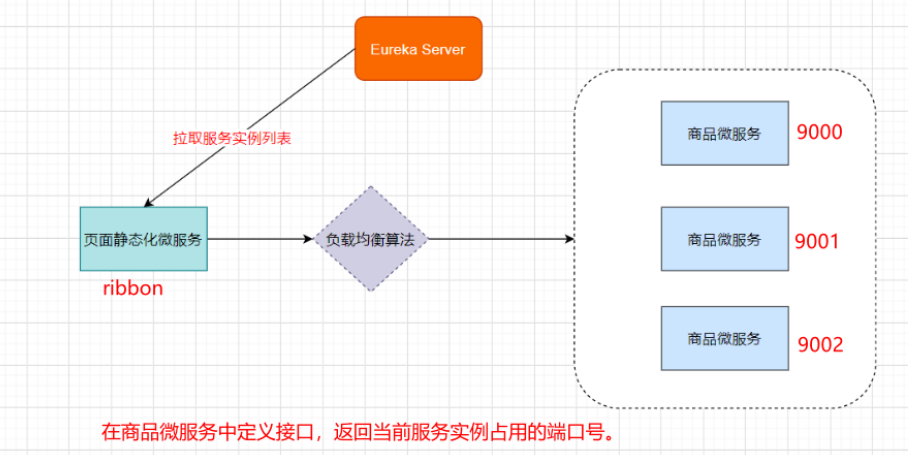
# 案例1：Ribbon高级应用

需求：

​复制商品微服务9001，在9000和9001编写Controller，返回服务实例端口。

​Page微服务中通过负载均衡策略调用demo-service-product的controller



在微服务中使用Ribbon不需要额外导入依赖坐标，微服务中引入过eureka-client相关依赖，会自动引入Ribbon相关依赖坐标。

代码中使用如下，在RestTemplate上添加对应注解即可：

@SpringBootApplication  
@EnableDiscoveryClient //@EnableEurekaClient  
public class PageApplication {  
​  
   public static void main(String[] args) {  
       SpringApplication.run(PageApplication.class,args);  
  }  
​  
   @Bean  
   //Ribbon负载均衡  
   @LoadBalanced  
   public RestTemplate restTemplate(){  
       return new RestTemplate();  
  }  
​  
}  
​

创建demo-serivce-product-9001微服务，创建ServerConfigController，定义方法返回当前微服务所使用的容器端口号

修改服务提供者api返回值，返回当前实例的端口号，便于观察负载情况

@RestController  
@RequestMapping("/server")  
public class ServerConfigController {  
​  
   @Value("${server.port}")  
   private String serverPort;  
​  
   @requestMapping("query")  
   public String findServerPort(){  
       return serverPort;  
  }  
​  
}  
​

在页面静态化微服务中调用demo-server-product下的资源路径：

[http://demo-server-product/server/query](http://lagou-server-product/server/query)

   @RequestMapping("/getPort")  
   public String getProductServerPort(){  
       String url = "http://demo-service-product/server/query";  
       return restTemplate.getForObject(url,String.class);  
  }

# 案例2：在微服务工程中加入Hystrix进行服务熔断和服务降级

目的：商品微服务长时间没有响应，服务消费者—>页面静态化微服务快速失败给用户提示

引入依赖：服务消费者工程（静态化微服务）中引入Hystrix依赖坐标（也可以添加在父工程中）

       <!--熔断器Hystrix-->  
       <dependency>  
           <groupId>org.springframework.cloud</groupId>  
           <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>  
       </dependency>

开启熔断：服务消费者工程（静态化微服务）的启动类中添加熔断器开启注解@EnableCircuitBreaker

/\*\*  
\* 注解简化写法  
\* @SpringCloudApplication = @SpringBootApplication+@EnableDiscoveryClient+@EnableCircuitBreaker  
\*/  
@SpringBootApplication  
@EnableDiscoveryClient //@EnableEurekaClient  
@EnableCircuitBreaker   // 开启熔断  
public class PageApplication {  
​  
   public static void main(String[] args) {  
       SpringApplication.run(PageApplication.class,args);  
  }  
   @Bean  
   @LoadBalanced//Ribbon负载均衡  
   public RestTemplate restTemplate(){  
       return new RestTemplate();  
  }  
}

定义服务降级处理方法：业务方法上使用@HystrixCommand的fallbackMethod属性关联到服务降级处理方法

​  
   /\*\*  
    \* 提供者模拟处理超时，调用方法添加Hystrix控制  
    \*/  
   // 使用@HystrixCommand注解进行熔断控制  
   @HystrixCommand(  
           // 线程池标识，要保持唯一，不唯一的话就共用了  
           threadPoolKey = "getProductServerPort2",  
           // 线程池细节属性配置  
           threadPoolProperties = {  
                   @HystrixProperty(name="coreSize",value = "1"), // 线程数  
                   @HystrixProperty(name="maxQueueSize",value="20") // 等待队列长度  
          },  
           // commandProperties熔断的一些细节属性配置  
           commandProperties = {  
                   // 每一个属性都是一个HystrixProperty  
                   @HystrixProperty(name="execution.isolation.thread.timeoutInMilliseconds",value="2000")  
          }  
  )  
   @RequestMapping("/getPort2")  
   public String getProductServerPort2(){  
       String url = "http://demo-service-product/server/query";  
       return restTemplate.getForObject(url,String.class);  
  }

商品微服务模拟超时操作

@RestController  
@RequestMapping("/server")  
public class ServerConfigController {  
​  
   @Value("${server.port}")  
   private String serverPort;  
​  
   @RequestMapping("/query")  
   public String findServerPort(){  
       try {  
           Thread.sleep(10000);  
      } catch (InterruptedException e) {  
           e.printStackTrace();  
      }  
       return serverPort;  
  }  
}

降级处理：

配置@HystrixCommand注解，定义降级处理方法

   @HystrixCommand(  
           // 线程池标识，要保持唯一，不唯一的话就共用了  
           threadPoolKey = "getProductServerPort3TimeoutFallback",  
           // 线程池细节属性配置  
           threadPoolProperties = {  
                   @HystrixProperty(name = "coreSize", value = "2"), // 线程数  
                   @HystrixProperty(name = "maxQueueSize", value = "20") // 等待队列长度  
          },  
           // commandProperties熔断的一些细节属性配置  
           commandProperties = {  
                   // 每一个属性都是一个HystrixProperty  
                   @HystrixProperty(name = "execution.isolation.thread.timeoutInMilliseconds", value = "2000"),  
                   // hystrix高级配置，定制工作过程细节  
                   // 统计时间窗口定义  
                   @HystrixProperty(name = "metrics.rollingStats.timeInMilliseconds", value = "8000"),  
                   // 统计时间窗口内的最小请求数  
                   @HystrixProperty(name = "circuitBreaker.requestVolumeThreshold", value = "2"),  
                   // 统计时间窗口内的错误数量百分比阈值  
                   @HystrixProperty(name = "circuitBreaker.errorThresholdPercentage", value = "50"),  
                   // 自我修复时的活动窗口长度  
                   @HystrixProperty(name = "circuitBreaker.sleepWindowInMilliseconds", value = "3000")  
          },  
           fallbackMethod = "myFallBack"  // 回退方法  
  )  
   @RequestMapping("/getPort3")  
   public String getProductServerPort3() {  
       String url = "http://demo-service-product/server/query";  
       return restTemplate.getForObject(url, String.class);  
  }  
​  
​  
   /\*\*  
    \* 定义回退方法，返回预设默认值  
    \* 注意：该方法形参和返回值与原始方法保持一致  
    \*/  
   public String myFallBack() {  
       return "-1"; // 兜底数据  
  }