1. **Eureka：服务注册**

**启动一个服务注册中心**：

**依赖：**

<dependency>

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

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

</dependency>

**Application类注解：**@EnableEurekaServer

**Yml配置文件：**

server:

port: 8761

eureka:

instance:

hostname: localhost

client:

registerWithEureka: false

fetchRegistry: false

serviceUrl:

defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/

spring:

application:

name: eurka-server

**创建服务提供者：**

**依赖：**

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

**Application类注解：**@EnableEurekaClient

**Yml文件配置：**

server:

port: 8762

spring:

application:

name: service-hi

eureka:

client:

serviceUrl:

defaultZone: <http://localhost:8761/eureka/>

1. **ribbon：负载均衡**

**方式一：restTimeplate**

**依赖：**

<dependency>

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

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

</dependency>

<dependency>

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

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

<dependency>

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

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

</dependency>

**向程序的IOC注入一个bean，并且加上@LoadBalanced注解：**

@SpringBootApplication

@EnableEurekaClient

@EnableDiscoveryClient

public class ServiceRibbonApplication {

public static void main(String[] args) {

SpringApplication.run( ServiceRibbonApplication.class, args );

}

@Bean

@LoadBalanced

RestTemplate restTemplate() {

return new RestTemplate();

}

}

**通过restTimeplate的getForObject()调用服务提供者提供的接口，实现负载均衡**

@RequestMapping("/book2")

public Book book2() {

Book book = restTemplate.getForObject("http://HELLO-SERVICE/getbook1", Book.class);

return book;

}

**方式二：Feign**

**在程序启动类加上注解：@EnableFeignClients**

**依赖：**

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

**定义一个feign接口，通过@ FeignClient（“服务名”），来指定调用哪个服务。**

@FeignClient(value = "service-hi")

public interface SchedualServiceHi {

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

String sayHiFromClientOne(@RequestParam(value = "name") String name);

}

**在Web层的controller层，对外暴露一个”/hi”的API接口，通过上面定义的Feign客户端SchedualServiceHi 来消费服务。**

@RestController

public class HiController {

@Autowired

SchedualServiceHi schedualServiceHi;

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

public String sayHi(@RequestParam String name){

return schedualServiceHi.sayHiFromClientOne(name);

}

}

1. **Hystrix：断路器、请求缓存**

**请求缓存：**

* 1. **方法重载开启缓存：继承HystrixCommand类**
  2. **注解：@CacheResult(开启缓存) @HystrixCommand**

@CacheResult可以用在Service方法上，表示该方法开启缓存，默认情况下方法的所有参数都是缓存的key。

**4.zuul：路由网关**

**依赖：**

<dependency>

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

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

</dependency>

**路由作用：**

**Application类注解：**@EnableZuulProxy

**配置文件：**

zuul:

routes:

api-a:

path: /api-a/\*\*

serviceId: service-ribbon

api-b:

path: /api-b/\*\*

serviceId: service-feign

将以/api-a/开头的请求转发到service-ribbon服务

**服务过滤：**

**新建一个类继承ZuulFilter**

@Component

public class MyFilter extends ZuulFilter{

private static Logger log = LoggerFactory.getLogger(MyFilter.class);

@Override

public String filterType() {

return "pre";

}

@Override

public int filterOrder() {

return 0;

}

@Override

public boolean shouldFilter() {

return true;

}

@Override

public Object run() {

return null;

}

}

filterType：返回一个字符串代表过滤器的类型，在zuul中定义了四种不同生命周期的过滤器类型，具体如下： 

* pre：路由之前
* routing：路由之时
* post： 路由之后
* error：发送错误调用
* filterOrder：过滤的顺序
* shouldFilter：这里可以写逻辑判断，是否要过滤，本文true,永远过滤。
* run：过滤器的具体逻辑。可用很复杂，包括查sql，nosql去判断该请求到底有没有权限访问。