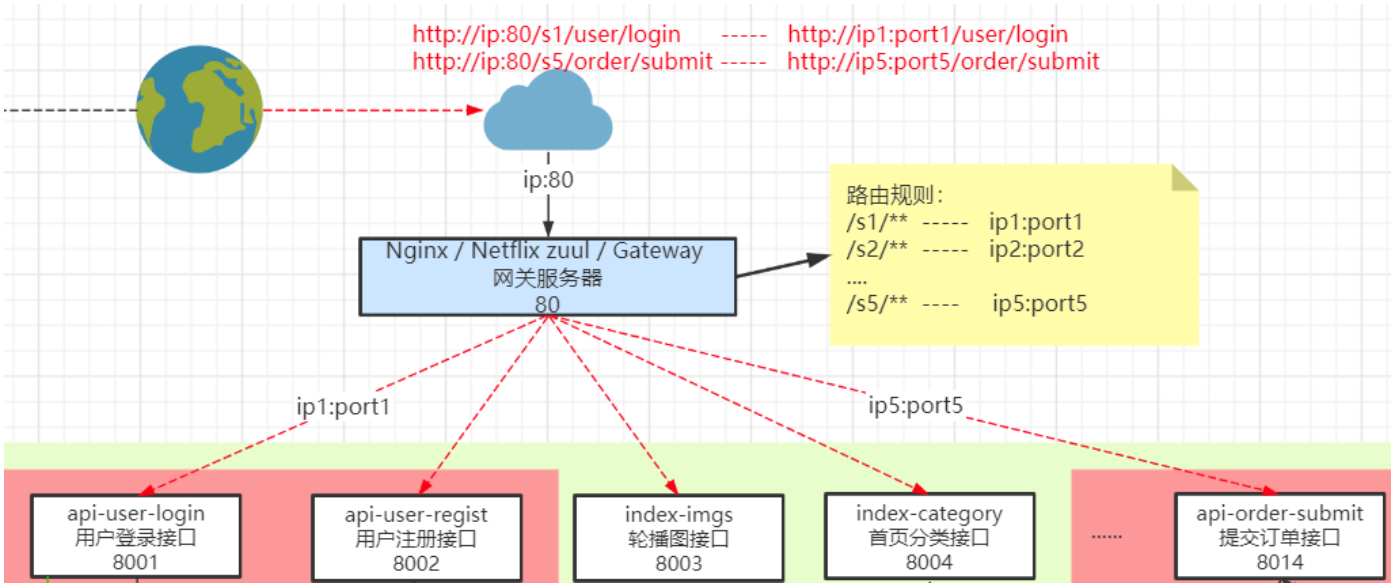
一、网关介绍

1.1 锋迷商城微服务拆分之后遇到的问题?

> 当我们对锋迷商城进行微服务拆分之后，不同的接口是由不同的服务提供的，不同的服务部署在不同的服务器上，因此前端进行接口调用的时候访问不同的接口会请求不同的ip和port，如果将接口服务的访问地址在前端代码中固定写死：
>
> - 前端需要记录很多服务器地址列表
> - 当服务被迁移到不同的服务器上的时候，就必须修改前端代码才能继续访问
> - 当对服务进行集群部署的时候，没有办法实现负载均衡

1.2 什么是API网关?

> 使用服务网关作为接口服务的统一代理，前端通过网关完成服务的统一调用



1.3 网关可以干什么?

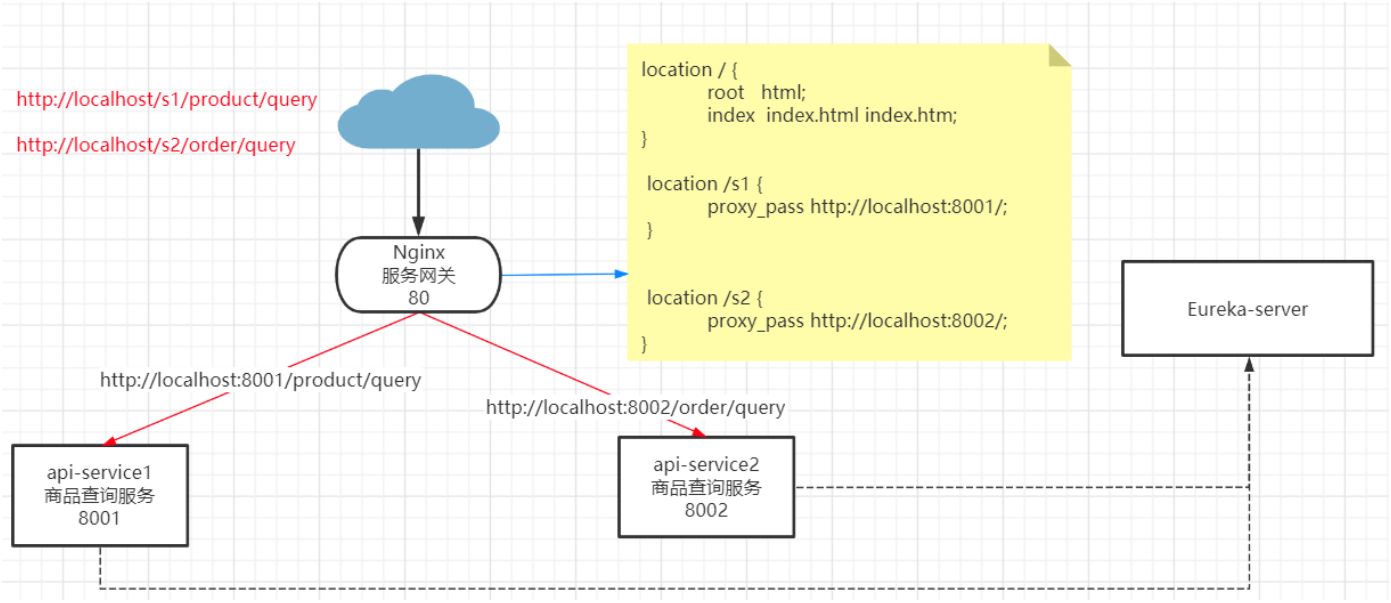
- 路由：接口服务的统一代理，实现前端对接口服务的统一访问
- 过滤：对用户请求进行拦截、过滤（用户鉴权）、监控
- 限流：限制用户的访问流量

1.4 常用的网关

- Nginx
- Spring Cloud Netflix zuul
- Spring Cloud Gateway

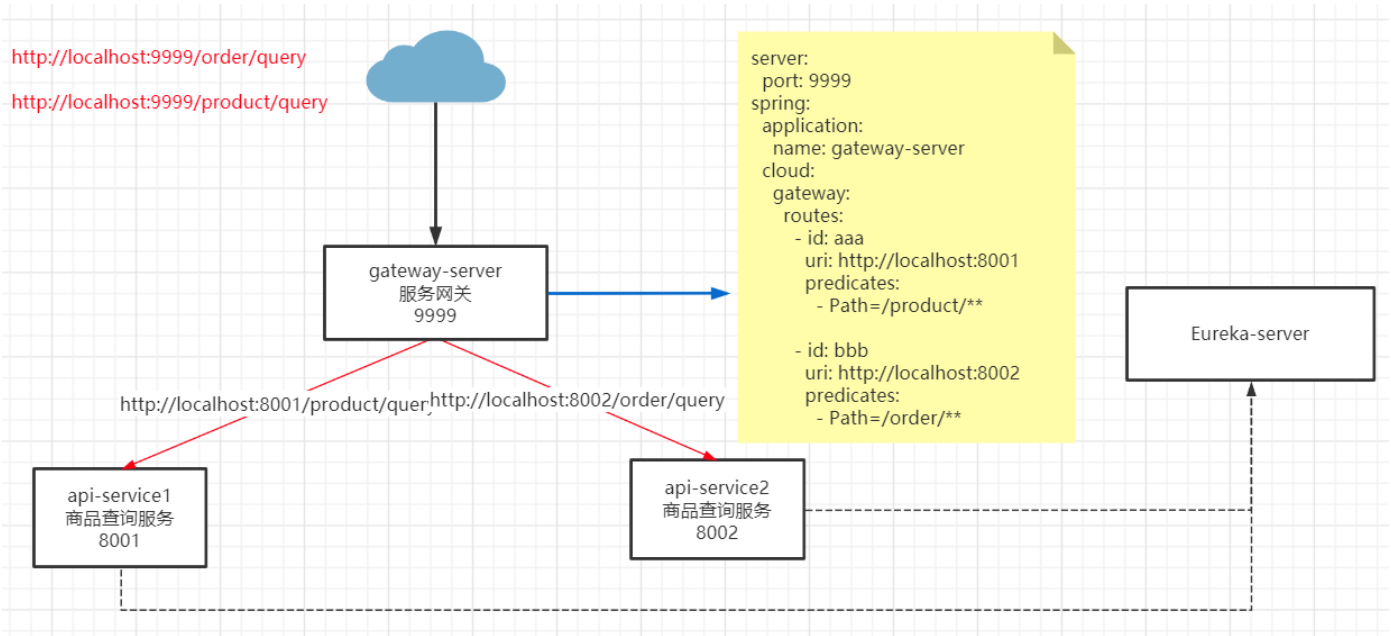
二、使用Nginx实现网关服务

> Nginx通常被用作应用服务器网关，服务网关通常使用zuul或者gateway



三、使用Gateway实现网关服务

> netflix zuul使用文档
> 链接： http://note.youdao.com/noteshare?id=b53758f34f40cb67357ee8ec13d37afd



3.1 搭建gateway服务器

- 创建SpringBoot应用，添加gateway依赖

3.2 配置路由规则

- application.yml

```yaml
server:
  port: 9999
spring:
  application:
    name: gateway-server
  cloud:
    gateway:
      routes:
        # 配置api-service1路由规则
        - id: api-service1
          uri: http://localhost:8001
          predicates:
            - Path=/product/**
    # 配置api-service2路由规则
        - id: api-service2
          uri: http://localhost:8002
          predicates:
            - Path=/order/**
```


四、Gateway工作原理

4.1 名词解释

Route： 路由是网关的基本组成部分，它是由id、目标uri、断言组成，如果断言为true，则匹配该路由，转向到当前路由的URI

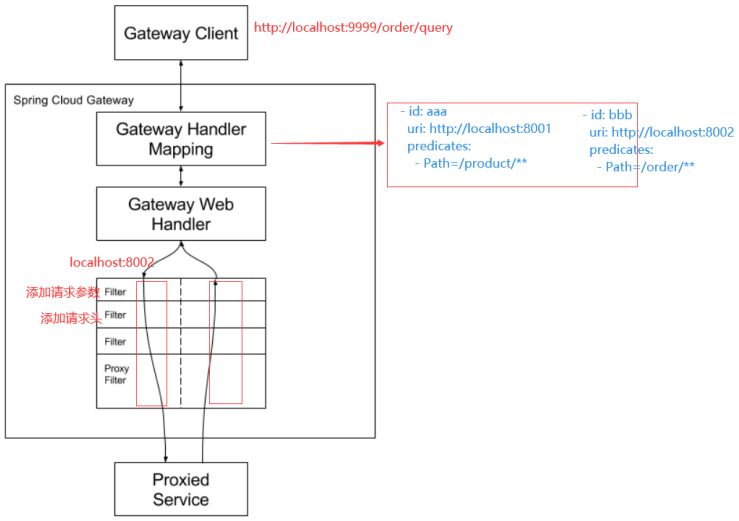
Predicate：断言，用户请求的匹配规则

Filter：过滤器，用于对请求进行前置、后置处理（可以在网关实现对请求或相应的加工处理）

4.2 Gateway工作流程图

3. How It Works

The following diagram provides a high-level overview of how Spring Cloud Gateway works:



五、Predicate断言

SpringCloud Gateway提供了多种断言匹配的方式：

- After
- Before
- Between
- Cookie
- Header
- Host
- Method
- Path
- Query
- RemoteAddr
- Weight

5.1 Path

根据请求路径的正则匹配


```yaml
spring:
  application:
    name: gateway-server
  cloud:
    gateway:
      routes:
        - id: aaa
          uri: http://localhost:8001
          predicates:
            - Path=/product/**
```

```
        - id: bbb
          uri: http://localhost:8002
          predicates:
            - Path=/order/**
```

5.2 Query

> 根据请求携带的参数匹配路由


```
spring:
  application:
    name: gateway-server
  cloud:
    gateway:
      routes:
        - id: aaa
          uri: http://localhost:8001
          predicates:
            # 如果请求url中带有name参数 ---> http://localhost:8001
            - Query=name
        - id: bbb
          uri: http://localhost:8002
          predicates:
            #如果请求url中带有pwd参数 ---> http://localhost:8002
            - Query=pwd
```

5.3 Header

> 根据Header中携带的参数匹配


```
spring:
  application:
    name: gateway-server
  cloud:
    gateway:
      routes:
        - id: aaa
          uri: http://localhost:8001
          predicates:
            - Header=token
        - id: bbb
          uri: http://localhost:8002
          predicates:
            - Header=aa,haha
```

六、过滤器

6.1 Gateway内置网关过滤器

> gateway网关可以对用户的请求和响应进行处理，gateway提供了多个内置的过滤器，不同的过滤器可以完成不同的请求或者响应的处理
>
> 1. AddRequestHeader 在请求头中添加参数
> 2. AddRequestParameter 添加请求参数
> 3. AddResponseHeader
> 4. 6.4. The DedupeResponseHeader GatewayFilter Factory
> 5. 6.5. The Hystrix GatewayFilter Factory
> 6. 6.6. Spring Cloud CircuitBreaker GatewayFilter Factory
> 7. 6.7. The FallbackHeaders GatewayFilter Factory
> 8. 6.8. The MapRequestHeader GatewayFilter Factory
> 9. 6.9. The PrefixPath GatewayFilter Factory
> 10. 6.10. The PreserveHostHeader GatewayFilter Factory
> 11. 6.11. The RequestRateLimiter GatewayFilter Factory
> 12. 6.12. The RedirectTo GatewayFilter Factory
> 13. 6.13. The RemoveRequestHeader GatewayFilter Factory
> 14. 6.14. RemoveResponseHeader GatewayFilter Factory
> 15. 6.15. The RemoveRequestParameter GatewayFilter Factory
> 16. 6.16. The RewritePath GatewayFilter Factory
> 17. 6.17. RewriteLocationResponseHeader GatewayFilter Factory
> 18. 6.18. The RewriteResponseHeader GatewayFilter Factory
> 19. 6.19. The SaveSession GatewayFilter Factory
> 20. 6.20. The SecureHeaders GatewayFilter Factory
> 21. 6.21. The SetPath GatewayFilter Factory
> 22. 6.22. The SetRequestHeader GatewayFilter Factory
> 23. 6.23. The SetResponseHeader GatewayFilter Factory
> 24. 6.24. The SetStatus GatewayFilter Factory
> 25. 6.25. The StripPrefix GatewayFilter Factory
> 26. 6.26. The Retry GatewayFilter Factory
> 27. 6.27. The RequestSize GatewayFilter Factory
> 28. 6.28. The SetRequestHostHeader GatewayFilter Factory
> 29. 6.29. Modify a Request Body GatewayFilter Factory
> 30. 6.30. Modify a Response Body GatewayFilter Factory
> 31. 6.31. Default Filters


```yaml
spring:
  application:
    name: gateway-server
  cloud:
    gateway:
      routes:
        - id: aaa
          uri: http://localhost:8001
          predicates:
            - Path=/red/aaa/product/**
          filters:
```

```
            - AddRequestHeader=token,wahahaawahaha
            - AddRequestParameter=username, ergou
            - SetStatus=404
            # - RewritePath=/red(?<segment>/?.*), $\{segment}
            - StripPrefix=2
```

6.2 自定义服务过滤器

6.2.1 创建网关过滤器 - 实现GatewayFilter

- 实现GatewayFilter, Ordered

```
public class MyFilter01 implements GatewayFilter, Ordered {
    @Override
    public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {
        ServerHttpRequest request = exchange.getRequest();
        ServerHttpResponse response = exchange.getResponse();
        System.out.println("---------------自定义过滤器");

        return chain.filter(exchange);
    }

    @Override
    public int getOrder() {
        return 0;
    }
}
```

- 配置过滤器

```
@Configuration
public class GatewayConfig {

    @Bean
    public RouteLocator routeLocator(RouteLocatorBuilder builder){
        System.out.println("-----------------------init");
        RouteLocator routeLocator = builder.routes().route( r->
                r.path("/product/**")                    // predicates
                 .filters(f->f.filters( new MyFilter01() ))     // filters
                 .uri("http://localhost:8001")                 //uri
        ).build();
        return routeLocator;
    }

}
```

6.2.2 创建网关过滤器 - 继承AbstractNameValueGatewayFilterFactory

> 相当于扩展Gateway内置的网关过滤器

```
/**
```

```
* 创建一个类继承AbstractNameValueGatewayFilterFactory，类名必须以GatewayFilterFactory结尾，类名前面的部分即为当前自定义网关过滤器的 名字
* 添加@Component注解，注册到Spring容器
*/
@Component
public class MyFilterGatewayFilterFactory extends AbstractNameValueGatewayFilterFactory {
    @Override
    public GatewayFilter apply(NameValueConfig config) {

        System.out.println("name:"+config.getName());
        System.out.println("value:"+config.getValue());

        //创建自定义网关过滤器并返回
        GatewayFilter gatewayFilter = new GatewayFilter() {
            @Override
            public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {
                System.out.println("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~自定义网关过滤器");
                return chain.filter(exchange);
            }
        };

        return gatewayFilter;
    }
}
```

配置：

```
spring:
  application:
    name: gateway-server
  cloud:
    gateway:
      routes:
        - id: bbb
          uri: http://localhost:8002
          predicates:
            - Path=/order/**
          filters:
            - MyFilter=aa,bb
```


6.3 全局过滤器

> 上述我们讲到的过滤器，都是配置在某个路由/服务中，我们称之为 网关服务过滤器 ，Gateway提供了内置的全局过滤器，会拦截过滤所有到达网关服务器的请求。
>
> 内置的全局过滤器默认生效，无需开发者干预。
>
> 根据业务的需求我们也可以自定义全局过滤器以实现对所有网关请求的拦截和处理。

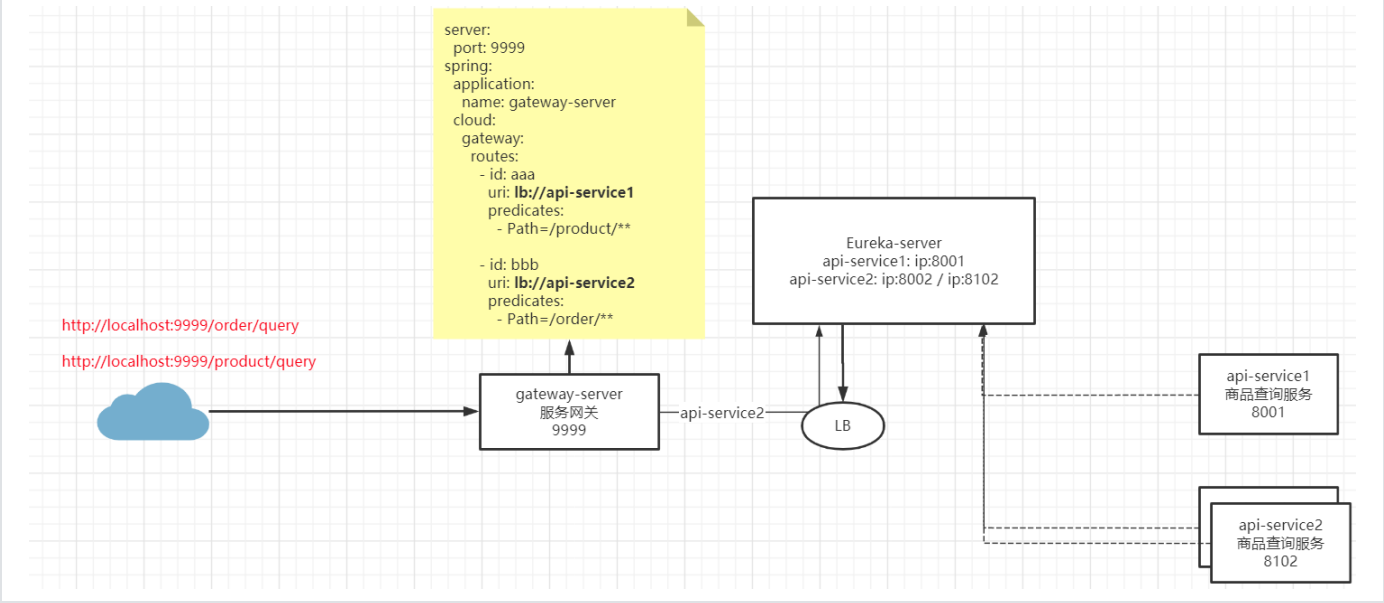
```
@Component
public class MyGlobalFilter implements GlobalFilter, Ordered {
```

```
    @Override
    public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {

        System.out.println("-----------------------------------------------MyGlobalFilter");
        List<String> list = exchange.getRequest().getHeaders().get("token");

        if (list != null && list.size()>0){
            String token = list.get(0);
            System.out.println("token:"+token);
            return chain.filter(exchange);
        }else{
            //如果没有token，或者token过期
            ServerHttpResponse response = exchange.getResponse();
            //设置响应头
            response.getHeaders().add("Content-Type","application/json;charset=utf-8");
            //设置状态码
            response.setStatusCode(HttpStatus.UNAUTHORIZED);
            // 封装响应数据
            String str = "";
            DataBuffer dataBuffer = response.bufferFactory().wrap(str.getBytes());
            return response.writeWith(Mono.just(dataBuffer));
        }
    }

    @Override
    public int getOrder() {
        return 0;
    }
}
```

七、Gateway动态路由配置

> 如果在Gateway网关的路由配置中，直接将服务的ip port配置进去，将导致：
>
> 1.如果服务的地址变更，必须要重新配置gateway的路由规则
>
> 2.如果服务采用集群部署，则不能实现负载均衡

Gateway动态路由



7.1 Gateway服务器添加eureka-server依赖


```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
</dependency>
```

7.2 配置网关路由

- application.yml


```
server:
  port: 9999
spring:
  application:
    name: gateway-server
  main:
    web-application-type: reactive
  cloud:
    gateway:
      routes:
        - id: aaa
          uri: lb://api-service1
          predicates:
            - Path=/product/**

        - id: bbb
          uri: lb://api-service2
          predicates:
            - Path=/order/**
eureka:
  client:
    service-url:
```

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

八、网关限流

8.1 网关限流介绍

> 服务的统一访问
>
> 使用过滤器实现鉴权
>
> 使用网关实现限流：通过限制用户的请求进入到服务中，有效控制应用系统的QPS，达到保护系统的目的

- 在互联网应用的生产环境中，可以由于用户增长超过预期、竞争对手恶意请求等外界的原因都有可能导致服务器无法处理超预期的高并发请求导致服务器宕机。

8.2 网关限流的常见算法

8.2.1 计数器算法

就是通过记录在单位时间请求的数量，在这个时间周期内达到设置定值之后就拒绝后续请求的进入。

- 缺点：如果用户请求频率不均匀，有导致短时间、高并发的风险

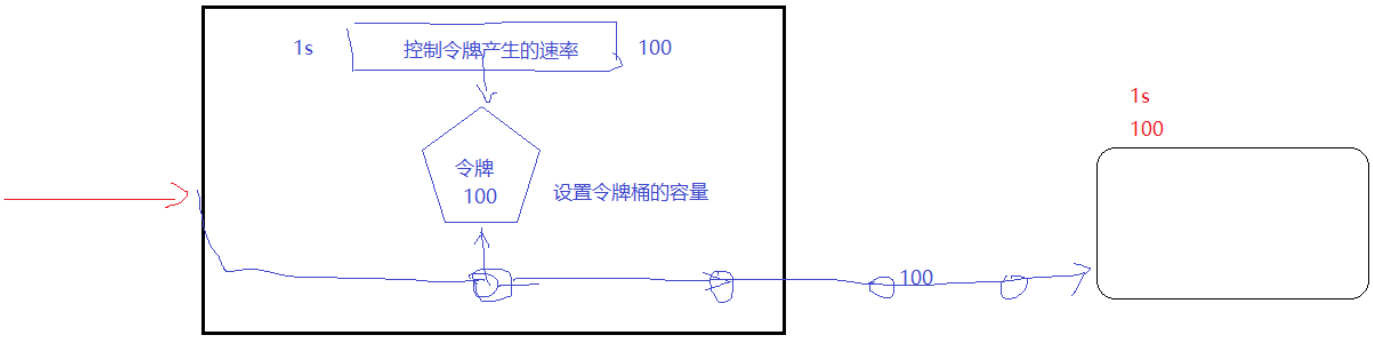
8.2.2 漏桶算法

就是通过控制“漏桶”流出的速率以限制到达服务的用户流量

- 缺点：网关会承载比较大的压力

8.2.3 令牌桶算法

就是所有进入网关的请求必须从令牌桶中得到令牌才可以进行服务调用，我们可以通过控制令牌桶的容量、令牌的产生速率达到控制用户流量的目的。



8.3 Gateway网关限流

> Gateway是基于令牌桶算法，使用redis作为“桶”结合顾虑器实现了网关限流

8.3.1 添加依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis-reactive</artifactId>
</dependency>

<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-pool2</artifactId>
    <version>2.9.0</version>
</dependency>
```

8.3.2 配置keyResolver

```
@Configuration
public class AppConfig {

    @Bean
    public KeyResolver keyResolver() {
        //http://localhost:9999/order/query?user=1
        //使用请求中的user参数的值作为令牌桶的key
       //return exchange ->Mono.just(exchange.getRequest().getQueryParams().getFirst("user"));
        return exchange -> Mono.just(exchange.getRequest().getURI().getPath());
    }

}
```

8.3.3 配置服务的限流规则

```
server:
  port: 9999
spring:
  application:
    name: gateway-server
  main:
    web-application-type: reactive
  cloud:
    gateway:
      routes:
        - id: aaa
          uri: lb://api-service1
          predicates:
            - Path=/product/**

        - id: bbb
          uri: lb://api-service2
          predicates:
            - Path=/order/**
          filters:
            - name: RequestRateLimiter
              args:
```

```yaml
                redis-rate-limiter.replenishRate: 1   #令牌桶每s的填充速度
                redis-rate-limiter.burstCapacity: 2  # 令牌桶容量
                redis-rate-limiter.requestedTokens: 1
                key-resolver: "#{@keyResolver}"
  redis:
    host: 47.96.11.185
    port: 7001
    password: qfedu123
    database: 0
    lettuce:
      pool:
        max-active: 10
        max-wait: 1000
        max-idle: 5
        min-idle: 3
eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka
```