在16 服务网关（D版 基础）.note一文中，我们通过使用Spring Cloud Zuul构建了一个基础的API网关服务，同时也演示了Spring Cloud Zuul基于服务的自动路由功能。在本文中，我们将进一步详细地介绍关于Spring Cloud Zuul的路由功能，以帮助读者可以更好的理解和使用它，以完成更复杂的路由配置。

传统路由配置

所谓的传统路由配置方式就是在不依赖于服务发现机制的情况下，通过在配置文件中具体指定每个路由表达式与服务实例的映射关系来实现API网关对外部请求的路由。

没有Eureka和Consul的服务治理框架帮助的时候，我们需要根据服务实例的数量采用不同方式的配置来实现路由规则：

- 单实例配置：通过一组zuul.routes.<route>.path与zuul.routes.<route>.url参数对的方式配置，比如：

```
zuul.routes.user-service.path=/user-service/**
zuul.routes.user-service.url=http://localhost:8080/
```

该配置实现了对符合/user-service/**规则的请求路径转发到http://localhost:8080/地址的路由规则，比如，当有一个请求http://localhost:1101/user-service/hello被发送到API网关上，由于/user-service/hello能够被上述配置的path规则匹配，所以API网关会转发请求到http://localhost:8080/hello地址。

- 多实例配置：通过一组zuul.routes.<route>.path与zuul.routes.<route>.serviceId参数对的方式配置，比如：

```
zuul.routes.user-service.path=/user-service/**
zuul.routes.user-service.serviceId=user-service
ribbon.eureka.enabled=false
user-service.ribbon.listOfServers=http://localhost:8080/,http://localhost:8081/
```

该配置实现了对符合/user-service/**规则的请求路径转发到http://localhost:8080/和http://localhost:8081/两个实例地址的路由规则。它的配置方式与服务路由的配置方式一样，都采用了zuul.routes.<route>.path与zuul.routes.<route>.serviceId参数对的映射方式，只是这里的serviceId是由用户手工命名的服务名称，配合<serviceId>.ribbon.listOfServers参数实现服务与实例的维护。由于存在多个实例，API网关在进行路由转发时需要实现负载均衡策略，于是这里还需要Spring Cloud Ribbon的配合。由于在Spring Cloud Zuul中自带了对Ribbon的依赖，所以我们只需要做一些配置即可，比如上面示例中关于Ribbon的各个配置，它们的具体作用如下：

- ribbon.eureka.enabled：由于zuul.routes.<route>.serviceId指定的是服务名称，默认情况下Ribbon会根据服务发现机制来获取配置服务名对应的实例清单。但是，该示例并没有整合类似Eureka之类的服务治理框架，所以需要将该参数设置为false，不然配置的serviceId是获取不到对应实例清单的。

- user-service.ribbon.listOfServers：该参数内容与zuul.routes.<route>.serviceId的配置相对应，开头的user-service对应了serviceId的值，这两个参数的配置相当于在该应用内部手工维护了服务与实例的对应关系。

不论是单实例还是多实例的配置方式，我们都需要为每一对映射关系指定一个名称，也就是上面配置中的<route>，每一个<route>就对应了一条路由规则。每条路由规则都需要通过path属性来定义一个用来匹配客户端请求的路径表达式，并通过url或serviceId属性来指定请求表达式映射具体实例地址或服务名。

服务路由配置

服务路由我们在上一篇中也已经有过基础的介绍和体验，Spring Cloud Zuul通过与Spring Cloud Eureka的整合，实现了对服务实例的自动化维护，所以在使用服务路由配置的时候，我们不需要向传统路由配置方式那样为serviceId去指定具体的服务实例地址，只需要通过一组zuul.routes.<route>.path与zuul.routes.<route>.serviceId参数对的方式配置即可。

比如下面的示例，它实现了对符合/user-service/**规则的请求路径转发到名为user-service的服务实例上去的路由规则。其中<route>可以指定为任意的路由名称。

```
zuul.routes.user-service.path=/user-service/**
zuul.routes.user-service.serviceId=user-service
```

对于面向服务的路由配置，除了使用path与serviceId映射的配置方式之外，还有一种更简洁的配置方式：zuul.routes.<serviceId>=<path>，其中<serviceId>用来指定路由的具体服务名，<path>用来配置匹配的请求表达式。比如下面的例子，它的路由规则等价于上面通过path与serviceId组合使用的配置方式。

```
zuul.routes.user-service=/user-service/**
```

传统路由的映射方式比较直观且容易理解，API网关直接根据请求的URL路径找到最匹配的path表达式，直接转发给该表达式对应的url或对应serviceId下配置的实例地址，以实现外部请求的路由。那么当采用path与serviceId以服务路由方式实现时候，没有配置任何实例地址的情况下，外部请求经过API网关的时候，它是如何被解析并转发到服务具体实例的呢？

在Spring Cloud Netflix中，Zuul巧妙的整合了Eureka来实现面向服务的路由。实际上，我们可以直接将API网关也看做是Eureka服务治理下的一个普通微服务应用。它除了会将自己注册到Eureka服务注册中心上之外，也会从注册中心获取所有服务以及它们的实例清单。所以，在Eureka的帮助下，API网关服务本身就已经维护了系统中所有serviceId与实例地址的映射关系。当有外部请求到达API网关的时候，根据请求的URL路径找到最佳匹配的path规则，API网关就可以知道要将该请求路由到哪个具体的serviceId上去。由于在API网关中已经知道serviceId对应服务实例的地址清单，那么只需要通过Ribbon的负载均衡策略，直接在这些清单中选择一个具体的实例进行转发就能完成路由工作了。