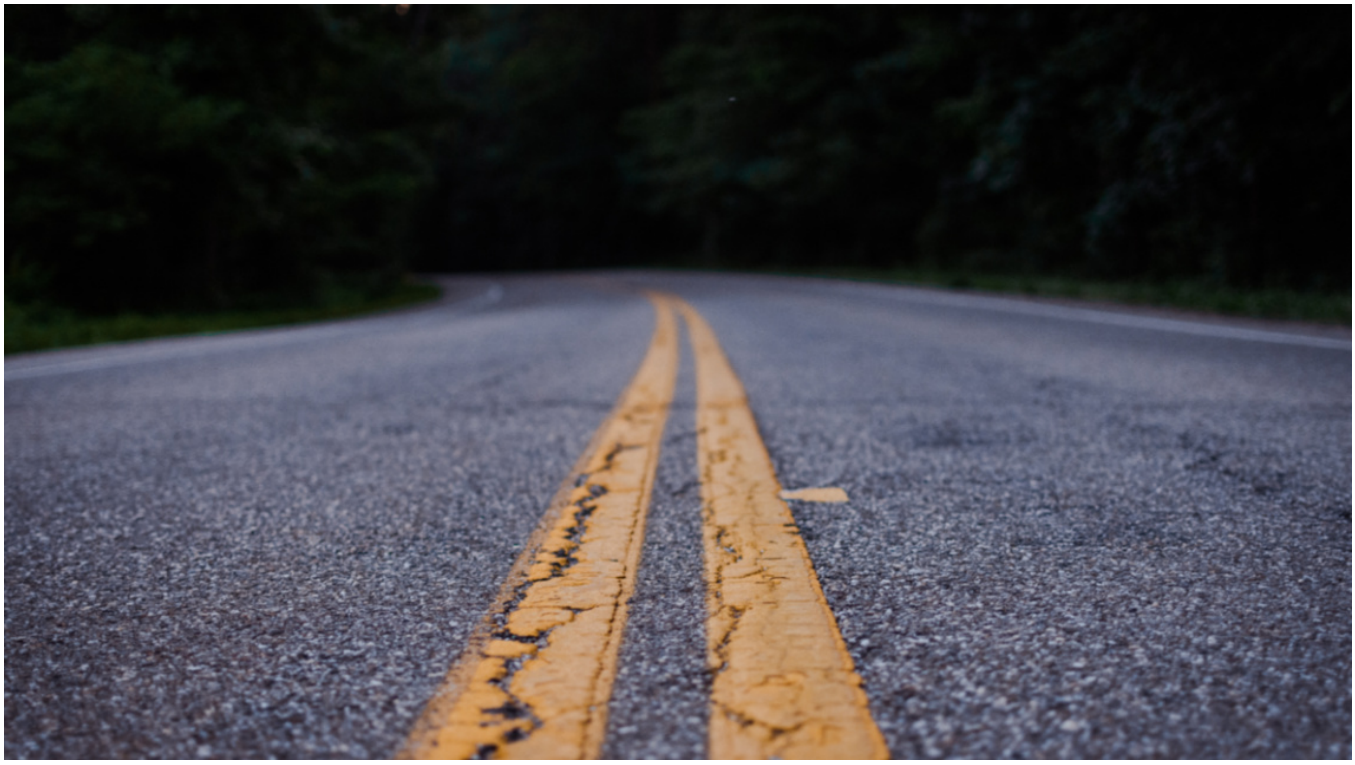
19 | 如何使用服务路由？

2018-10-04 胡忠想



19 | 如何使用服务路由？
朗读人：胡忠想　09′52″ | 4.53M

专栏上一期，我给你讲解了常用的客户端负载均衡算法，它帮我们解决了服务消费者如何从众多可用的服务节点中选取一个最合适的节点发起调用的问题。但在业务中经常还会遇到这样的场景，比如服务 A 部署在北京、上海、广州三个数据中心，所有的服务节点按照所在的数据中心被分成了三组，那么服务 A 的消费者在发起调用时，该如何选择呢？这就是今天我要给你讲解的服务路由的问题。

那么什么是服务路由呢？我的理解是服务路由就是服务消费者在发起服务调用时，必须根据特定的规则来选择服务节点，从而满足某些特定的需求。

那么服务路由都有哪些应用场景？具体都有哪些规则呢？

服务路由的应用场景

根据我的实践经验，服务路由主要有以下几种应用场景：

- 分组调用。一般来讲，为了保证服务的高可用性，实现异地多活的需求，一个服务往往不止部署在一个数据中心，而且出于节省成本等考虑，有些业务可能不仅在私有机房部署，还会采用公有云部署，甚至采用多家公有云部署。服务节点也会按照不同的数据中心分成不同的分组，这时对于服务消费者来说，选择哪一个分组调用，就必须有相应的路由规则。

- 灰度发布。在服务上线发布的过程中，一般需要先在一小部分规模的服务节点上先发布服务，然后验证功能是否正常。如果正常的话就继续扩大发布范围；如果不正常的话，就需要排查问题，解决问题后继续发布。这个过程就叫作灰度发布，也叫金丝雀部署。

- 流量切换。在业务线上运行过程中，经常会遇到一些不可抗力因素导致业务故障，比如某个机房的光缆被挖断，或者发生着火等事故导致整个机房的服务都不可用。这个时候就需要按照某个指令，能够把原来调用这个机房服务的流量切换到其他正常的机房。

- 读写分离。对于大多数互联网业务来说都是读多写少，所以在进行服务部署的时候，可以把读写分开部署，所有写接口可以部署在一起，而读接口部署在另外的节点上。

上面四种应用场景是实际业务中很常见的，服务路由可以通过各种规则来实现，那么服务路由都有哪些规则呢？

服务路由的规则

根据我的实践经验，服务路由主要有两种规则：一种是条件路由，一种是脚本路由。

1. 条件路由

条件路由是基于条件表达式的路由规则，以下面的条件路由为例，我来给你详细讲解下它的用法。

复制代码

```
condition://0.0.0.0/dubbo.test.interfaces.TestService?category=routers&dynamic=true&priority=
```

这里面“condition://”代表了这是一段用条件表达式编写的路由规则，具体的规则是

复制代码

```
host = 10.20.153.10 => host = 10.20.153.11
```

分隔符“=>”前面是服务消费者的匹配条件，后面是服务提供者的过滤条件。当服务消费者节点满足匹配条件时，就对该服务消费者执行后面的过滤规则。那么上面这段表达式表达的意义就是 IP 为“10.20.153.10”的服务消费者都调用 IP 为“10.20.153.11”的服务提供者节点。

如果服务消费者的匹配条件为空，就表示对所有的服务消费者应用，就像下面的表达式一样。

```
=> host ! = 10.20.153.11
```

复制代码

如果服务提供者的过滤条件为空，就表示禁止服务消费者访问，就像下面的表达式一样。

```
host = 10.20.153.10=>
```

复制代码

下面我举一些 Dubbo 框架中的条件路由，来给你讲解下条件路由的具体应用场景。

- 排除某个服务节点

```
=> host != 172.22.3.91
```

复制代码

一旦这条路由规则被应用到线上，所有的服务消费者都不会访问 IP 为 172.22.3.91 的服务节点，这种路由规则一般应用在线上流量排除预发布机以及摘除某个故障节点的场景。

- 白名单和黑名单功能

```
host != 10.20.153.10,10.20.153.11 =>
```

复制代码

这条路由规则意思是除了 IP 为 10.20.153.10 和 10.20.153.11 的服务消费者可以发起服务调用以外，其他服务消费者都不可以，主要用于白名单访问逻辑，比如某个后台服务只允许特定的几台机器才可以访问，这样的话可以机器控制访问权限。

```
host = 10.20.153.10,10.20.153.11 =>
```

复制代码

同理，这条路由规则意思是除了 IP 为 10.20.153.10 和 10.20.153.11 的服务消费者不能发起服务调用以外，其他服务消费者都可以，也就是实现了黑名单功能，比如线上经常会遇到某些调用方不管是出于有意还是无意的不合理调用，影响了服务的稳定性，这时候可以通过黑名单功能暂时予以封杀。

- 机房隔离

```
host = 172.22.3.* => host = 172.22.3.*
```

复制代码

这条路由规则意思是 IP 网段为 172.22.3.* 的服务消费者，才可以访问同网段的服务节点，这种规则一般应用于服务部署在多个 IDC，理论上同一个 IDC 内的调用性能要比跨 IDC 调用性能要

好，应用这个规则是为了实现同 IDC 就近访问。

- 读写分离

```
method = find*,list*,get*,is* => host =172.22.3.94,172.22.3.95
method != find*,list*,get*,is* => host = 172.22.3.97,172.22.3.98
```

这条路由规则意思是 find*、get*、is* 等读方法调用 IP 为 172.22.3.94 和 172.22.3.95 的节点，除此以外的写方法调用 IP 为 172.22.3.97 和 172.22.3.98 的节点。对于大部分互联网业务来说，往往读请求要远远大于写请求，而写请求的重要性往往要远远高于读请求，所以需要把读写请求进行分离，以避免读请求异常影响到写请求，这时候就可以应用这种规则。

2. 脚本路由

脚本路由是基于脚本语言的路由规则，常用的脚本语言比如 JavaScript、Groovy、JRuby 等。以下面的脚本路由规则为例，我来给你详细讲解它的用法。

```
"script://0.0.0.0/com.foo.BarService?category=routers&dynamic=false&rule=" + URL.encode("（fu
```

这里面“script://”就代表了这是一段脚本语言编写的路由规则，具体规则定义在脚本语言的 route 方法实现里，比如下面这段用 JavaScript 编写的 route() 方法表达的意思是，只有 IP 为 10.20.153.10 的服务消费者可以发起服务调用。

```
function route(invokers){
  var result = new java.util.ArrayList(invokers.size());
  for(i =0; i < invokers.size(); i ++){
    if("10.20.153.10".equals(invokers.get(i).getUrl().getHost())){
       result.add(invokers.get(i));
    }
  }
  return result;
 } (invokers)）;
```

既然服务路由是通过路由规则来实现的，那么服务消费者该如何获取路由规则呢？

服务路由的获取方式

根据我的实践经验，服务路由的获取方式主要有三种：

- 本地配置

顾名思义就是路由规则存储在服务消费者本地上。服务消费者发起调用时，从本地固定位置读取路由规则，然后按照路由规则选取一个服务节点发起调用。

- 配置中心管理

这种方式下，所有的服务消费者都从配置中心获取路由规则，由配置中心来统一管理。

- 动态下发

这种方式下，一般是运维人员或者开发人员，通过服务治理平台修改路由规则，服务治理平台调用配置中心接口，把修改后的路由规则持久化到配置中心。因为服务消费者订阅了路由规则的变更，于是就会从配置中心获取最新的路由规则，按照最新的路由规则来执行。

根据我的实践经验，上面三种方式实际使用时，还是有一定区别的。

一般来讲，服务路由最好是存储在配置中心中，由配置中心来统一管理。这样的话，所有的服务消费者就不需要在本地管理服务路由，因为大部分的服务消费者并不关心服务路由的问题，或者说也不需要去了解其中的细节。通过配置中心，统一给各个服务消费者下发统一的服务路由，节省了沟通和管理成本。

但也不排除某些服务消费者有特定的需求，需要定制自己的路由规则，这个时候就适合通过本地配置来定制。

而动态下发可以理解为一种高级功能，它能够动态地修改路由规则，在某些业务场景下十分有用。比如某个数据中心存在问题，需要把调用这个数据中心的服务消费者都切换到其他数据中心，这时就可以通过动态下发的方式，向配置中心下发一条路由规则，将所有调用这个数据中心的请求都迁移到别的地方。

当然，这三种方式也可以一起使用，这个时候服务消费者的判断优先级是本地配置 > 动态下发 > 配置中心管理。

总结

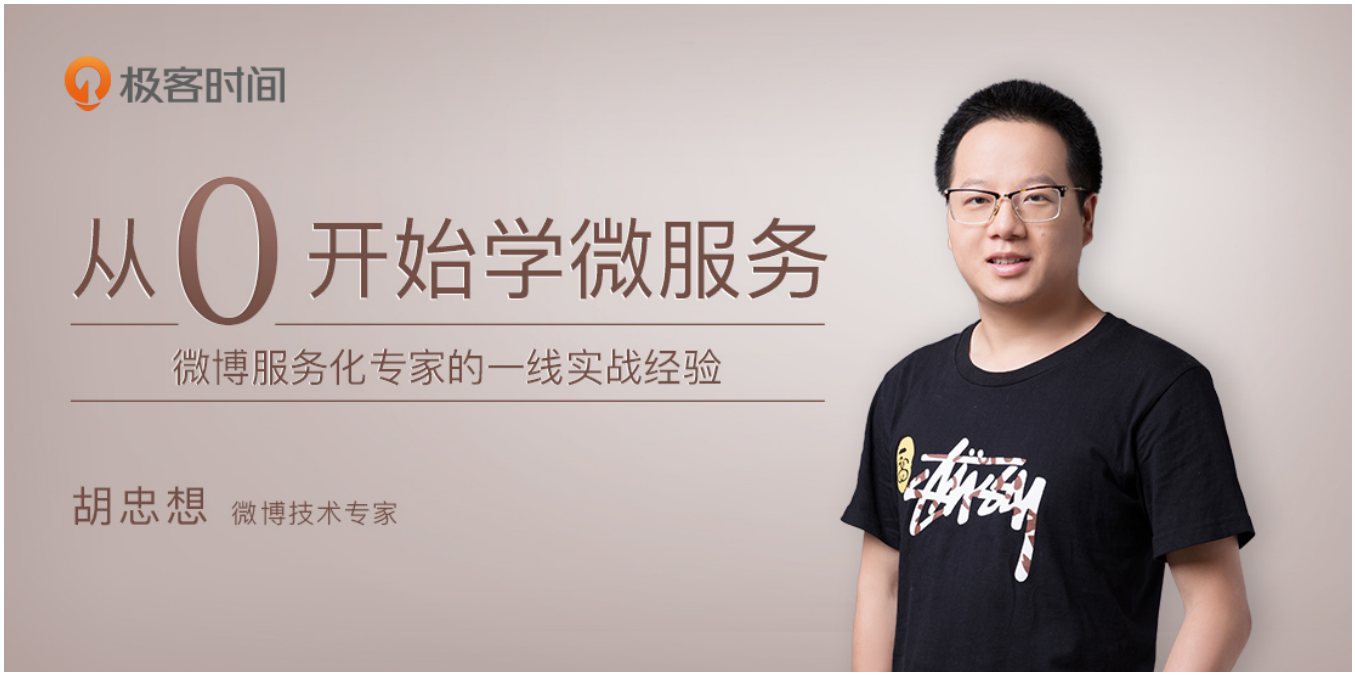
今天我给你讲解了服务路由的作用，简单来讲就是为了实现某些调用的特殊需求，比如分组调用、灰度发布、流量切换、读写分离等。在业务规模比较小的时候，可能所有的服务节点都部署在一起，也就不需要服务路由。但随着业务规模的扩大、服务节点增多，尤其是涉及多数据中心部署的情况，把服务节点按照数据中心进行分组，或者按照业务的核心程度进行分组，对提高服务的可用性是十分有用的。以微博业务为例，有的服务不仅进行了核心服务和非核心服务分组，还针对私有云和公有云所处的不同数据中心也进行了分组，这样的话就可以将服务之间的调用尽量都限定在同一个数据中心内部，最大限度避免跨数据中心的网络延迟、抖动等影响。

而服务路由具体是在本地配置，还是在配置中心统一管理，也是视具体业务需求而定的。如果没有定制化的需求，建议把路由规则都放到配置中心中统一存储管理。而动态下发路由规则对于服务治理十分有帮助，当数据中心出现故障的时候，可以实现动态切换流量，还可以摘除一些有故障的服务节点。

思考题

在实际业务场景中，经常有一类需求就是一个新功能在全量上线前，会圈一批用户优先适用，如果使用服务路由功能的话，你觉得可以怎么做？

欢迎你在留言区写下自己的思考，与我一起讨论。



写留言

精选留言

宝爷 0

用一个简单的路由规则，如果希望百分之一的用户体验新版本，将用户id取模100等于1的用户请求转发给新的服务器。验证后逐步扩大用户比例，最终达到全量。
这里有个问题想请教一下老师，用docker和k8s部署一个mysql的服务，如果这个服务可以弹性伸缩，且我们通过一个虚拟ip来连接这个服务，那么我的请求会按照k8s的负载均衡规则进行路由，我希望操作一个用户相关的请求，都到同一个数据库里面，不知道这个应该怎么操作。如果有多个服务，而且可以动态的增减，这里应该会碰到一堆的问题，如果老师有碰到类似的问题，这里想知道老师是怎么解决的。

2018-10-04

作者回复

我理解这里主要是访问路由的问题，又想用k8s做到mysql的动态扩缩容，又想按照用户访问hash到固定数据库、我不知道k8s能否自定义数据访问路由，我们的做法没有使用k8s，自定义数据访问路由

2018-10-04

_CountingStars 0

配置中心配置路由之后也是需要下发的 那这和动态下发方式不是重复了吗 有什么主要区别呢

2018-10-04

作者回复

配置中心存储可能只是做一个集中存储的地方，客户端启动的时候才会从配置中心拉取配置。动态下发是客户端订阅了配置的更新，如果有变化，客户端就会拉取

2018-10-04