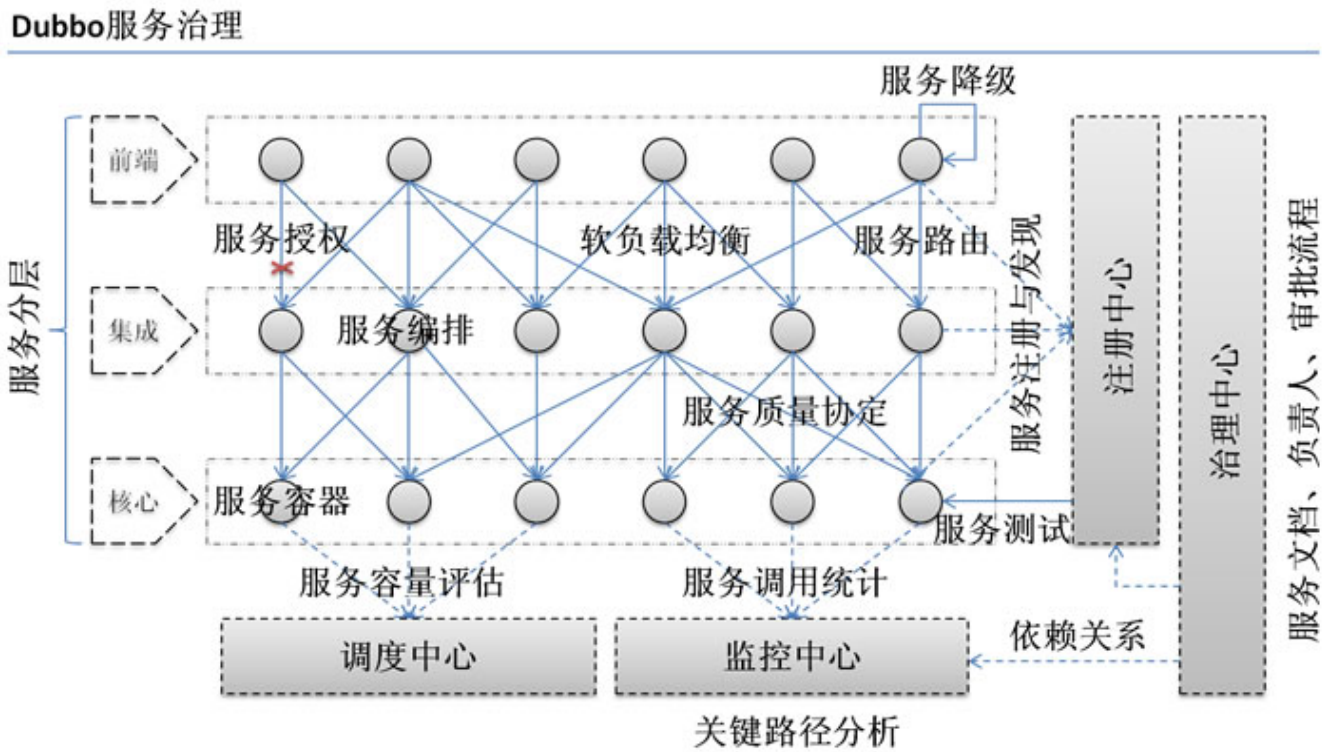
07-Dubbo服务治理

什么是服务治理？

为什么需要服务治理？

1 需求



在大规模服务化之前，应用可能只是通过 RMI 或 Hessian 等工具，简单的暴露和引用远程服务，通过配置服务的URL地址进行调用，通过 F5 等硬件进行负载均衡。

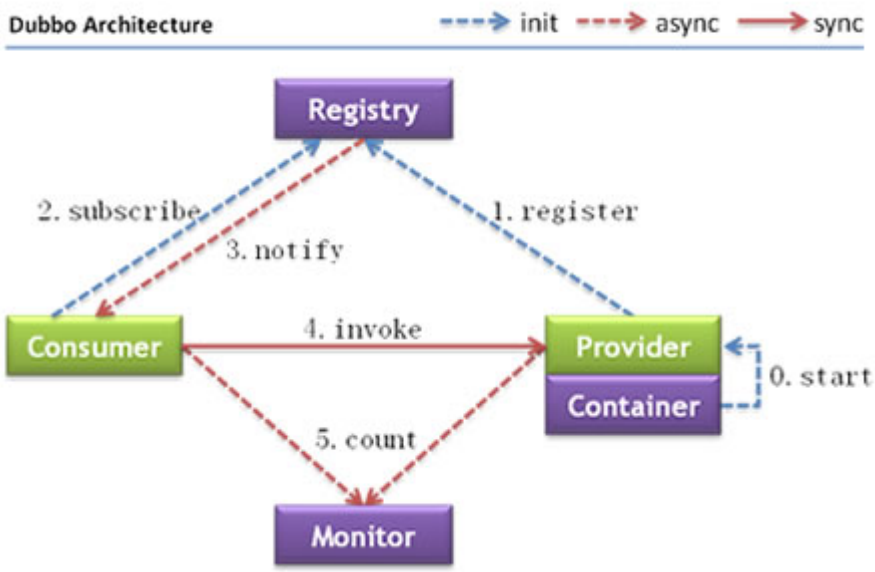
当服务越来越多时，服务 URL 配置管理变得非常困难，F5 硬件负载均衡器的单点压力也越来越大。 此时需要一个服务注册中心，动态地注册和发现服务，使服务的位置透明。并通过在消费方获取服务提供方地址列表，实现软负载均衡和 Failover，降低对 F5 硬件负载均衡器的依赖，也能减少部分成本。

当进一步发展，服务间依赖关系变得错踪复杂，甚至分不清哪个应用要在哪个应用之前启动，架构师都不能完整的描述应用的架构关系。 这时，需要自动画出应用间的依赖关系图，以帮助架构师理清理关系。

接着，服务的调用量越来越大，服务的容量问题就暴露出来，这个服务需要多少机器支撑？什么时候该加机器？ 为了解决这些问题，第一步，要将服务现在每天的调用量，响应时间，都统计出来，作为容量规划的参考指标。其次，要可以动态调整权重，在线上，将某台机器的权重一直加大，并在加大的过程中记录响应时间的变化，直到响应时间到达阈值，记录此时的访问量，再以此访问量乘以机器数反推总容量。

以上是 Dubbo 最基本的几个需求。

2 架构



节点角色说明

节点	角色说明
`Provider`	暴露服务的服务提供方
`Consumer`	调用远程服务的服务消费方
`Registry`	服务注册与发现的注册中心
`Monitor`	统计服务的调用次数和调用时间的监控中心
`Container`	服务运行容器

调用关系说明

1. 服务容器负责启动，加载，运行服务提供者。
2. 服务提供者在启动时，向注册中心注册自己提供的服务。
3. 服务消费者在启动时，向注册中心订阅自己所需的服务。
4. 注册中心返回服务提供者地址列表给消费者，如果有变更，注册中心将基于长连接推送变更数据给消费者。
5. 服务消费者，从提供者地址列表中，基于软负载均衡算法，选一台提供者进行调用，如果调用失败，再选另一台调用。
6. 服务消费者和提供者，在内存中累计调用次数和调用时间，定时每分钟发送一次统计数据到监控中心。

3 Dubbo管理控制台介绍

目前的管理控制台已经发布0.1版本，结构上采取了前后端分离的方式，前端使用Vue和Vuetify分别作为Javascript框架和UI框架，后端采用Spring Boot框架。既可以按照标准的Maven方式进行打包，部署，也可以采用前后端分离的部署方式，方便开发，功能上，目前具备了服务查询，服务治理(包括Dubbo2.7中新增的治理规则)以及服务测试三部分内容。

Maven方式部署

- 安装

```
git clone https://github.com/apache/incubator-dubbo-admin.git
cd incubator-dubbo-admin
mvn clean package
cd dubbo-distribution/target
java -jar dubbo-admin-0.1.jar
```

- 访问 http://localhost:8080

前后端分离部署

- 前端

```
cd dubbo-admin-ui
npm run install
npm run dev
```

- 后端

```
cd dubbo-admin-server
mvn clean package
cd target
java -jar dubbo-admin-server-0.1.jar
```

- 访问 http://localhost:8081
- 前后端分离模式下，前端的修改可以实时生效

配置: [1]

配置文件为：

```
dubbo-admin-server/src/main/resources/application.properties
```

主要的配置有：

```
admin.config-center=zookeeper://127.0.0.1:2181
admin.registry.address=zookeeper://127.0.0.1:2181
admin.metadata-report.address=zookeeper://127.0.0.1:2181
```

三个配置项分别指定了配置中心，注册中心和元数据中心的地址，关于这三个中心的详细说明，可以参考这里。 也可以和Dubbo2.7一样，在配置中心指定元数据和注册中心的地址，以zookeeper为例，配置的路径和内容如下:

```
# /dubbo/config/dubbo/dubbo.properties
dubbo.registry.address=zookeeper://127.0.0.1:2181
dubbo.metadata-report.address=zookeeper://127.0.0.1:2181
```

配置中心里的地址会覆盖掉本地`application.properties`的配置

其他配置请访问github中的文档:

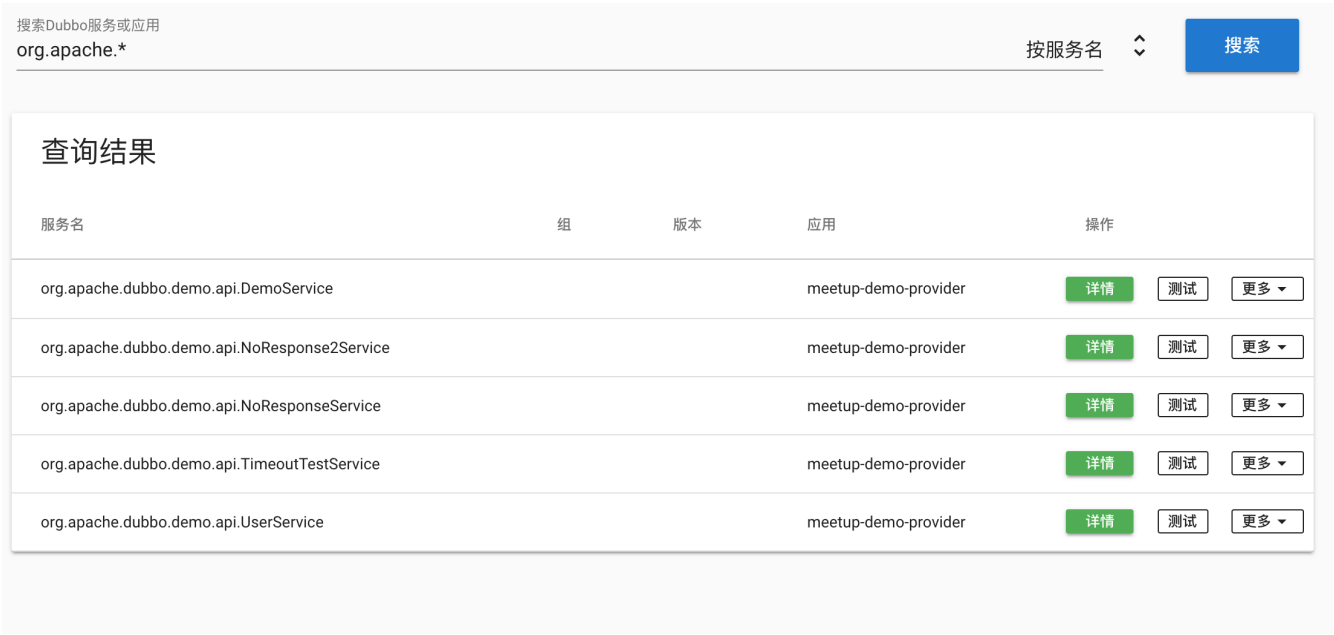
```
https://github.com/apache/incubator-dubbo-admin
```

---

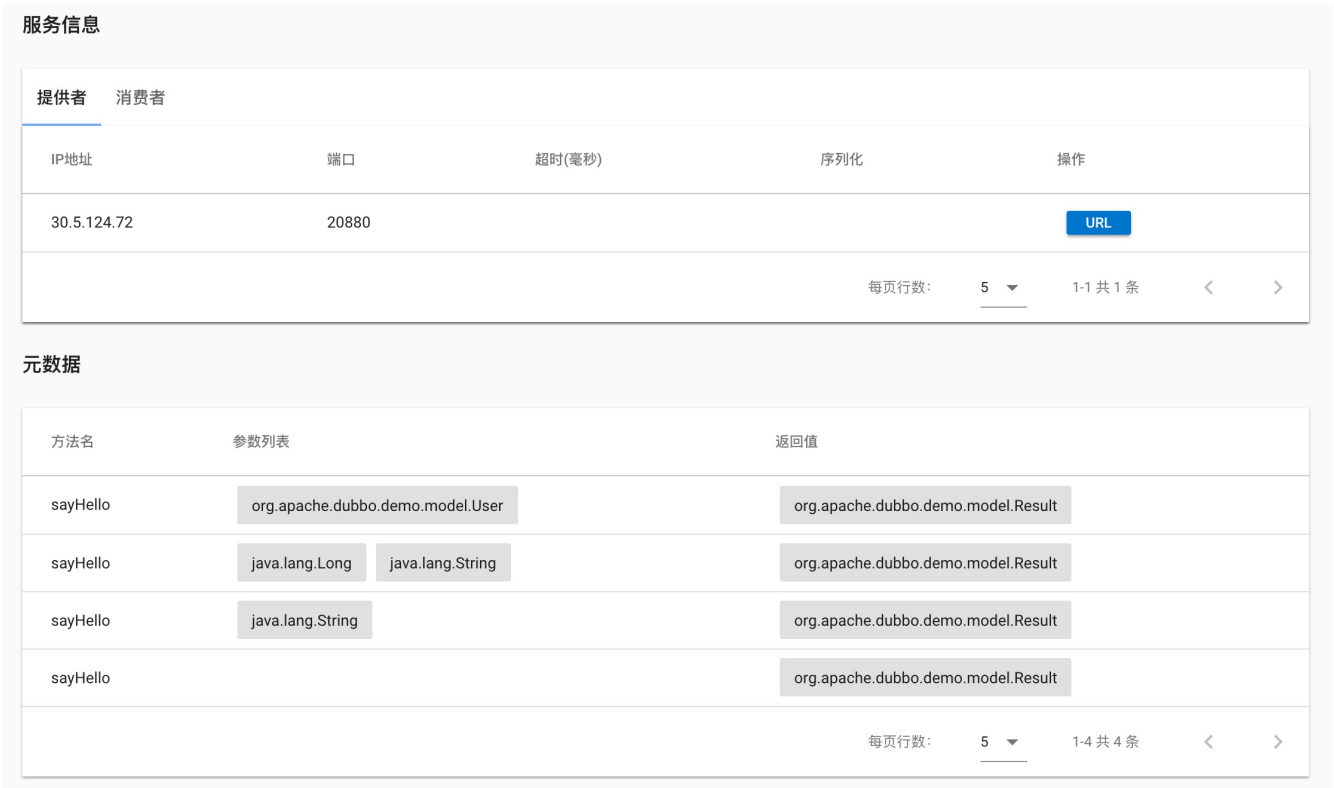
1. 当前版本中未实现登录功能，会在后续版本加上 ↩

4 服务查询和详情展示

服务查询是Dubbo OPS最基本的功能，目前支持服务，应用和IP三个维度的查询，并且服务和应用支持模糊查询和自动提示:



其中详情页展示了服务提供者，消费者等信息，元数据信息需要在Dubbo2.7及之后的版本才会展示:



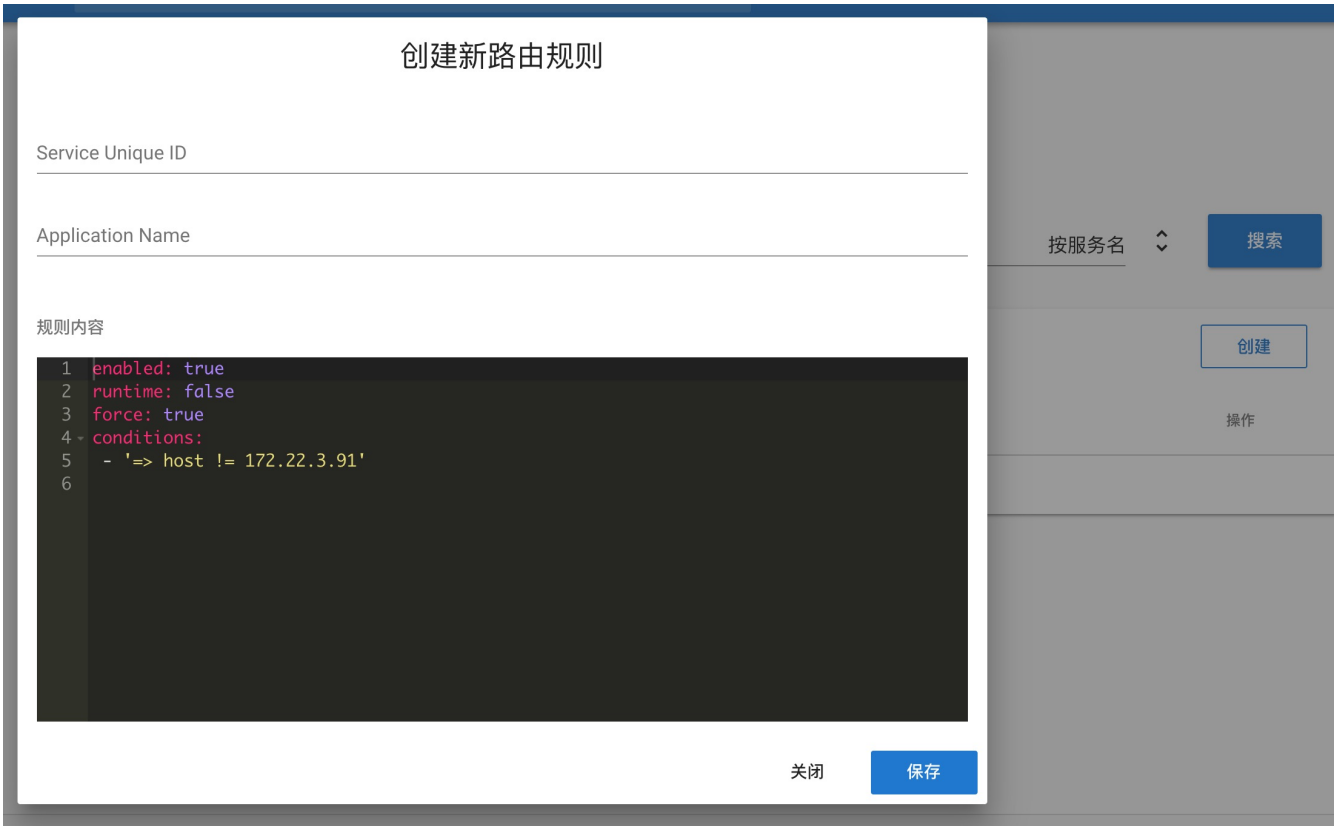
5 服务治理和配置管理

服务治理

服务治理主要作用是改变运行时服务的行为和选址逻辑，达到限流，权重配置等目的，主要有以下几个功能：

应用级别的服务治理

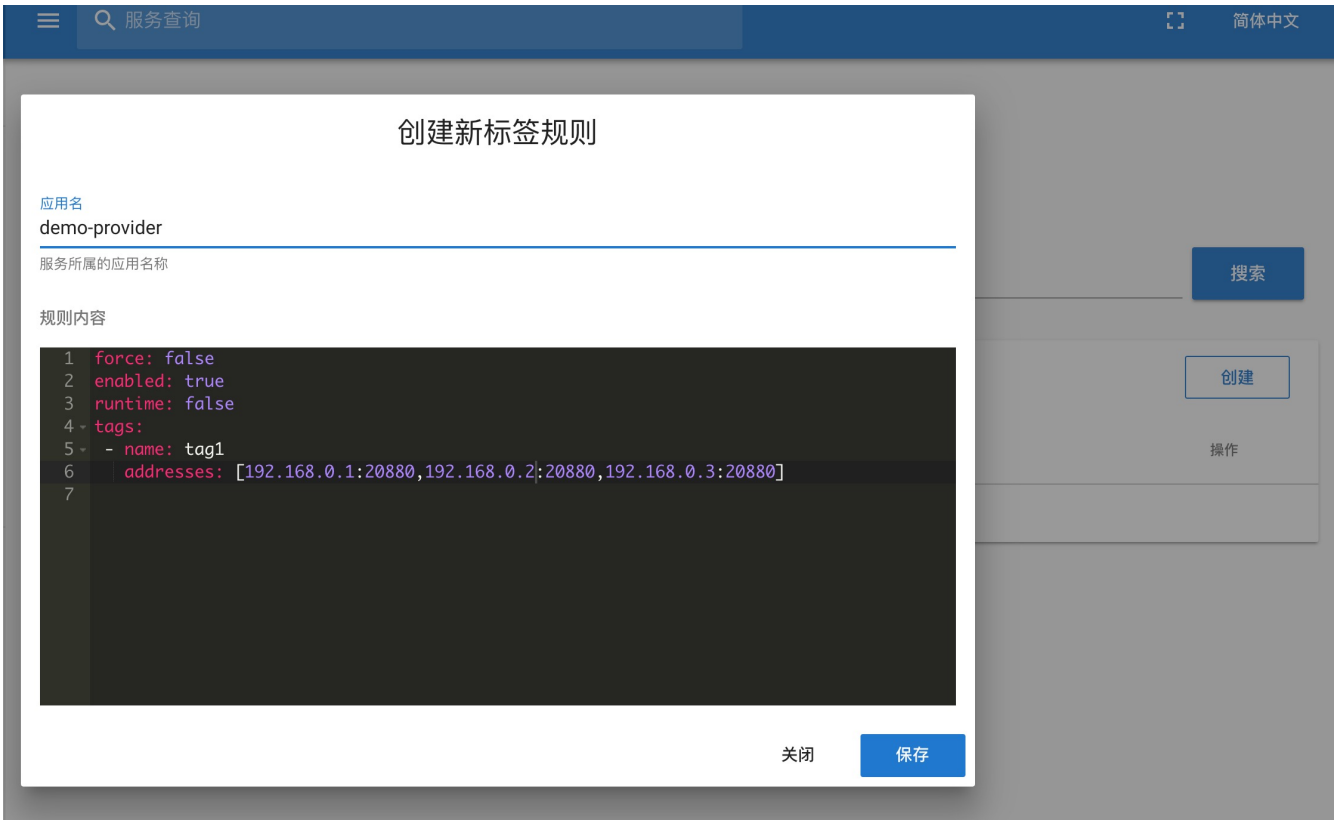
在Dubbo2.6及更早版本中，所有的服务治理规则都只针对服务粒度，如果要把某条规则作用到应用粒度上，需要为应用下的所有服务配合相同的规则，变更，删除的时候也需要对应的操作，这样的操作很不友好，因此Dubbo2.7版本中增加了应用粒度的服务治理操作，对于条件路由(包括黑白名单)，动态配置(包括权重，负载均衡)都可以做应用级别的配置：



上图是条件路由的配置，可以按照应用名，服务名两个维度来填写，也可以按照这两个维度来查询。

标签路由

标签路由是Dubbo2.7引入的新功能，配置以应用作为维度，给不同的服务器打上不同名字的标签，配置如下图所示：



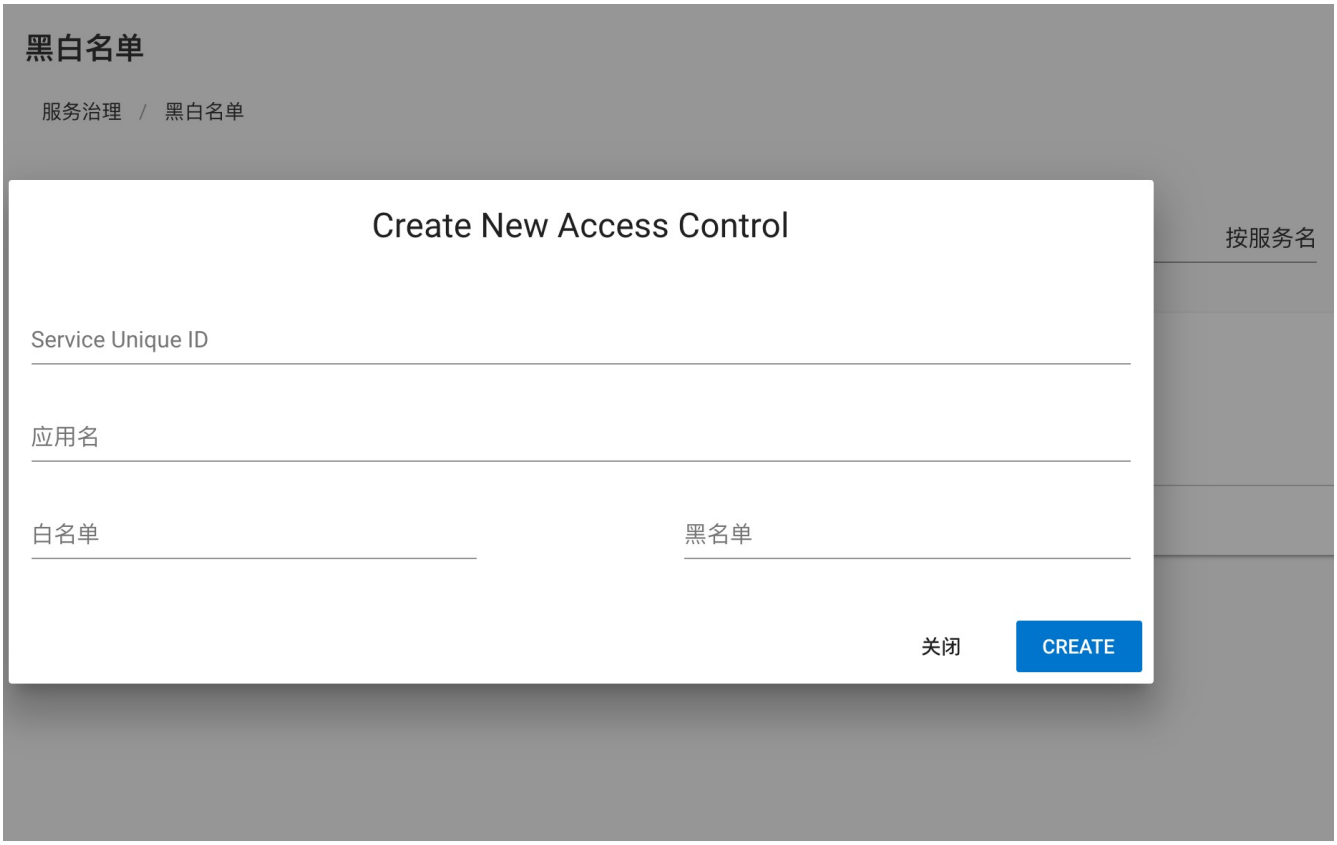
调用的时候，客户端可以通过`setAttachment`的方式，来设置不同的标签名称，比如本例中，`setAttachment(tag1)`，客户端的选址范围就在如图所示的三台机器中，可以通过这种方式来实现流量隔离，灰度发布等功能。

条件路由

条件路由是Dubbo一直以来就有的功能，目前可以配置服务和应用两个维度，条件路由为`yaml`格式，具体的规则体以及各种适用场景，请参考【6路由规则】

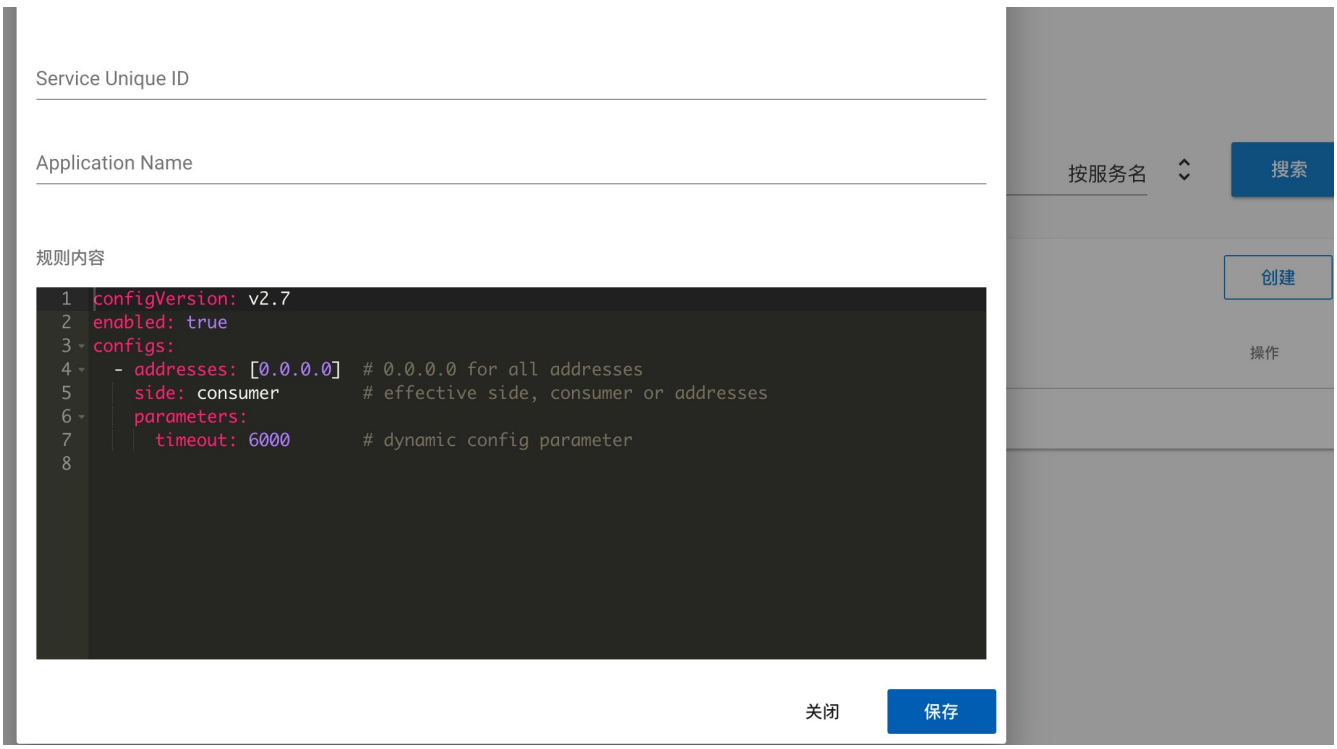
黑白名单

黑白名单是条件路由的一部分，规则存储和条件路由放在一起，为了方便配置所以单独拿出来，同样可以通过服务和应用两个维度，指定黑名单和白名单:



动态配置

动态配置是和路由规则平行的另一类服务治理治理功能，主要作用是在不重启服务的情况下，动态改变调用行为，从Dubbo2.7版本开始，支持服务和应用两个维度的配置，采用`yaml`格式，界面如下：



具体的规则体说明请参考【7配置规则】

权重调节

权重调节

权重调节是动态配置的子功能，主要作用是改变服务端的权重，更大的权重会有更大的几率被客户端选中作为服务提供者，从而达到流量分配的目的：

负载均衡

负载均衡也是动态配置的子功能，主要作用是调整客户端的选址逻辑，目前可选的负载均衡策略有随机，轮训和最小活跃，关于各个策略的解释请参考这里

配置管理

配置管理也是配合Dubbo2.7新增的功能，在Dubbo2.7中，增加了全局和应用维度的配置，分别在全局和应用范围内生效，其中应用配置也可以指定该应用中的服务级别的配置，可以在控制台中查看，修改配置规则，默认展示全局维度的配置。

- 全局配置：

全局配置里可以指定注册中心，元数据中心的地址，服务端和客户端的超时时间等，这些配置在全局内生效。除了配置写入，也可以用来查看。如果使用zookeeper作为注册中心和元数据中心，还可以看到配置文件所在位置的目录结构。

- 应用， 服务配置

应用级别的配置可以为应用或者应用内的服务指定配置，在服务维度上，需要区分提供者和消费者。`dubbo.reference.{serviceName}`表示作为该服务消费者的配置，`dubbo.provider.{servcieName}`表示作为该服务提供者的配置。其中注册中心和元数据中心的地址，只能在全局配置中指定，这也是Dubbo2.7中推荐的使用方式。

- 优先级： 服务配置 > 应用配置 > 全局配置

6 路由规则

在此查看老版本路由规则(2.6.x or before)

路由规则在发起一次RPC调用前起到过滤目标服务器地址的作用，过滤后的地址列表，将作为消费端最终发起RPC调用的备选地址。

- 条件路由。支持以服务或Consumer应用为粒度配置路由规则。
- 标签路由。以Provider应用为粒度配置路由规则。

后续我们计划在2.6.x版本的基础上继续增强脚本路由功能，老版本脚本路由规则配置方式请参见开篇链接。

条件路由

您可以随时在服务治理控制台Dubbo-Admin写入路由规则

简介

- 应用粒度

```
# app1的消费者只能消费所有端口为20880的服务实例
# app2的消费者只能消费所有端口为20881的服务实例
---
scope: application
force: true
runtime: true
enabled: true
key: governance-conditionrouter-consumer
conditions:
  - application=app1 => address=*:20880
  - application=app2 => address=*:20881
...
```

- 服务粒度

```
# DemoService的sayHello方法只能消费所有端口为20880的服务实例
# DemoService的sayHi方法只能消费所有端口为20881的服务实例
---
scope: service
force: true
runtime: true
enabled: true
key: org.apache.dubbo.samples.governance.api.DemoService
conditions:
  - method=sayHello => address=*:20880
  - method=sayHi => address=*:20881
...
```

规则详解

各字段含义

- `scope`

  表示路由规则的作用粒度，scope的取值会决定key的取值。

  必填

  。

  - service 务粒度
  - application 应用粒度

- `Key`

  明确规则体作用在哪个服务或应用。

  必填

  。
  - scope=service时，key取值为[{group}:]{service}[:{version}]的组合
  - scope=application时，key取值为application名称
- `enabled=true` 当前路由规则是否生效，可不填，缺省生效。
- `force=false` 当路由结果为空时，是否强制执行，如果不强制执行，路由结果为空的路由规则将自动失效，可不填，缺省为 `false`。
- `runtime=false` 是否在每次调用时执行路由规则，否则只在提供者地址列表变更时预先执行并缓存结果，调用时直接从缓存中获取路由结果。如果用了参数路由，必须设为 `true`，需要注意设置会影响调用的性能，可不填，缺省为 `false`。
- `priority=1` 路由规则的优先级，用于排序，优先级越大越靠前执行，可不填，缺省为 `0`。
- `conditions` 定义具体的路由规则内容。**必填。**

Conditions规则体

`conditions`部分是规则的主体，由1到任意多条规则组成，下面我们就每个规则的配置语法做详细说明：

1. **格式**

- `=>` 之前的为消费者匹配条件，所有参数和消费者的 URL 进行对比，当消费者满足匹配条件时，对该消费者执行后面的过滤规则。
- `=>` 之后为提供者地址列表的过滤条件，所有参数和提供者的 URL 进行对比，消费者最终只拿到过滤后的地址列表。
- 如果匹配条件为空，表示对所有消费方应用，如：`=> host != 10.20.153.11`
- 如果过滤条件为空，表示禁止访问，如：`host = 10.20.153.10 =>`

1. **表达式**

参数支持：

- 服务调用信息，如：method, argument 等，暂不支持参数路由
- URL 本身的字段，如：protocol, host, port 等
- 以及 URL 上的所有参数，如：application, organization 等

条件支持：

- 等号 `=` 表示"匹配"，如：`host = 10.20.153.10`
- 不等号 `!=` 表示"不匹配"，如：`host != 10.20.153.10`

值支持：

- 以逗号 `,` 分隔多个值，如：`host != 10.20.153.10,10.20.153.11`
- 以星号 `*` 结尾，表示通配，如：`host != 10.20.*`
- 以美元符 `$` 开头，表示引用消费者参数，如：`host = $host`

1. **Condition示例**

- 排除预发布机：

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

- 白名单 [1]：

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

- 黑名单：

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

- 服务寄宿在应用上，只暴露一部分的机器，防止整个集群挂掉：

```
=> host = 172.22.3.1*,172.22.3.2*
```

- 为重要应用提供额外的机器：

```
application != kylin => host != 172.22.3.95,172.22.3.96
```

- 读写分离：

```
method = find*,list*,get*,is* => host = 172.22.3.94,172.22.3.95,172.22.3.96
method != find*,list*,get*,is* => host = 172.22.3.97,172.22.3.98
```

- 前后台分离：

```
application = bops => host = 172.22.3.91,172.22.3.92,172.22.3.93
application != bops => host = 172.22.3.94,172.22.3.95,172.22.3.96
```

- 隔离不同机房网段：

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

- 提供者与消费者部署在同集群内，本机只访问本机的服务：

```
=> host = $host
```

标签路由规则

简介

标签路由通过将某一个或多个服务的提供者划分到同一个分组，约束流量只在指定分组中流转，从而实现流量隔离的目的，可以作为蓝绿发布、灰度发布等场景的能力基础。

Provider

标签主要是指对Provider端应用实例的分组，目前有两种方式可以完成实例分组，分别是`动态规则打标`和`静态规则打标`，其中动态规则相较于静态规则优先级更高，而当两种规则同时存在且出现冲突时，将以动态规则为准。

- 动态规则打标，可随时在服务治理控制台下发标签归组规则

```
# governance-tagrouter-provider应用增加了两个标签分组tag1和tag2
# tag1包含一个实例 127.0.0.1:20880
# tag2包含一个实例 127.0.0.1:20881
---
  force: false
  runtime: true
  enabled: true
  key: governance-tagrouter-provider
  tags:
    - name: tag1
      addresses: ["127.0.0.1:20880"]
    - name: tag2
      addresses: ["127.0.0.1:20881"]
 ...
```

- 静态打标

```
<dubbo:provider tag="tag1"/>
```

or

```
<dubbo:service tag="tag1"/>
```

or

```
java -jar xxx-provider.jar -Ddubbo.provider.tag={the tag you want, may come from OS ENV}
```

Consumer

```
RpcContext.getContext().setAttachment(Constants.REQUEST_TAG_KEY,"tag1");
```

请求标签的作用域为每一次 invocation，使用 attachment 来传递请求标签，注意保存在 attachment 中的值将会在一次完整的远程调用中持续传递，得益于这样的特性，我们只需要在起始调用时，通过一行代码的设置，达到标签的持续传递。

> 目前仅仅支持 hardcoding 的方式设置 requestTag。注意到 RpcContext 是线程绑定的，优雅的使用 TagRouter 特性，建议通过 servlet 过滤器(在 web 环境下)，或者定制的 SPI 过滤器设置 requestTag。

规则详解

格式

- `Key` 明确规则体作用到哪个应用。**必填**。
- `enabled=true` 当前路由规则是否生效，可不填，缺省生效。
- `force=false` 当路由结果为空时，是否强制执行，如果不强制执行，路由结果为空的路由规则将自动失效，可不填，缺省为 `false`。
- `runtime=false` 是否在每次调用时执行路由规则，否则只在提供者地址列表变更时预先执行并缓存结果，调用时直接从缓存中获取路由结果。如果用了参数路由，必须设为 `true`，需要注意设置会影响调用的性能，可不填，缺省为 `false`。
- `priority=1` 路由规则的优先级，用于排序，优先级越大越靠前执行，可不填，缺省为 `0`。
- `tags`

  定义具体的标签分组内容，可定义任意n（n>=1）个标签并为每个标签指定实例列表。

  必填

。
    - name， 标签名称
- addresses， 当前标签包含的实例列表

降级约定

1. `request.tag=tag1` 时优先选择 标记了 `tag=tag1` 的 provider。若集群中不存在与请求标记对应的服务，默认将降级请求 tag为空的provider；如果要该表这种默认行为，即找不到匹配tag1的provider返回异常，需设置 `request.tag.force=true`。
2. `request.tag` 未设置时，只会匹配tag为空的provider。即使集群中存在可用的服务，若 tag 不匹配也就无法调用，这与约定1不同，携带标签的请求可以降级访问到无标签的服务，但不携带标签/携带其他种类标签的请求永远无法访问到其他标签的服务。

---

1. 注意：一个服务只能有一条白名单规则，否则两条规则交叉，就都被筛选掉了 ↩

7 配置规则

查看老版本配置规则。

覆盖规则是Dubbo设计的在无需重启应用的情况下，动态调整RPC调用行为的一种能力。2.7.0版本开始，支持从**服务**和**应用**两个粒度来调整动态配置。

概览

请在服务治理控制台查看或修改覆盖规则。

- 应用粒度

```
# 将应用demo（key:demo）在20880端口上提供（side:provider）的所有服务
（scope:application）的权重修改为1000（weight:1000）。
---
scope: application
key: demo
enabled: true
configs:
- addresses: ["0.0.0.0:20880"]
  side: provider
  parameters:
    weight: 1000
...
```

- 服务粒度

```
# 所有消费（side:consumer）DemoService服务
（key:org.apache.dubbo.samples.governance.api.DemoService）的应用实例
（addresses:[0.0.0.0]），超时时间修改为6000ms
---
scope: service
key: org.apache.dubbo.samples.governance.api.DemoService
enabled: true
configs:
- addresses: [0.0.0.0]
  side: consumer
  parameters:
    timeout: 6000
...
```

规则详解

配置模板

```
---
scope: application/service
key: app-name/group+service+version
enabled: true
configs:
- addresses: ["0.0.0.0"]
  providerAddresses: ["1.1.1.1:20880", "2.2.2.2:20881"]
```

```
  side: consumer
  applications/services: []
  parameters:
    timeout: 1000
    cluster: failfase
    loadbalance: random
- addresses: ["0.0.0.0:20880"]
  side: provider
  applications/services: []
  parameters:
    threadpool: fixed
    threads: 200
    iothreads: 4
    dispatcher: all
    weight: 200
...
```

其中：

- `scope`表示配置作用范围，分别是应用（application）或服务（service）粒度。**必填**。
- `key`

  指定规则体作用在哪个服务或应用。

  必填

  。
  - scope=service时，key取值为[{group}:]{service}[:{version}]的组合
- scope=application时，key取值为application名称
- `enabled=true` 覆盖规则是否生效，可不填，缺省生效。
- `configs`

  定义具体的覆盖规则内容，可以指定n（n>=1）个规则体。

  必填

  。
  - side，
  - applications
  - services
  - parameters

- addresses
- providerAddresses

对于绝大多数配置场景，只需要理清楚以下问题基本就知道配置该怎么写了：

1. 要修改整个应用的配置还是某个服务的配置。
   - 应用：`scope: application, key: app-name`（还可使用`services`指定某几个服务）。
   - 服务：`scope: service, key:group+service+version`。
2. 修改是作用到消费者端还是提供者端。
   - 消费者：`side: consumer` ，作用到消费端时（你还可以进一步使用`providerAddress`, `applications`选定特定的提供者示例或应用）。
   - 提供者：`side: provider`。
3. 配置是否只对某几个特定实例生效。
   - 所有实例：`addresses: ["0.0.0.0"]`或`addresses: ["0.0.0.0:*"]`具体由side值决定。
   - 指定实例：`addersses[实例地址列表]`。
4. 要修改的属性是哪个。

示例

1. 禁用提供者：(通常用于临时踢除某台提供者机器，相似的，禁止消费者访问请使用路由规则)

```
---
scope: application
key: demo-provider
enabled: true
configs:
- addresses: ["10.20.153.10:20880"]
  side: provider
  parameters:
    disabled: true
...
```

2. 调整权重：(通常用于容量评估，缺省权重为 200)

```
---
scope: application
key: demo-provider
enabled: true
configs:
- addresses: ["10.20.153.10:20880"]
  side: provider
  parameters:
    weight: 200
...
```

3. 调整负载均衡策略：(缺省负载均衡策略为 random)

```
---
scope: application
key: demo-consumer
enabled: true
configs:
- side: consumer
  parameters:
    loadbalance: random
...
```

4. 服务降级：(通常用于临时屏蔽某个出错的非关键服务)

```
---
scope: service
key: org.apache.dubbo.samples.governance.api.DemoService
enabled: true
configs:
- side: consumer
 parameters:
   force: return null
...
```