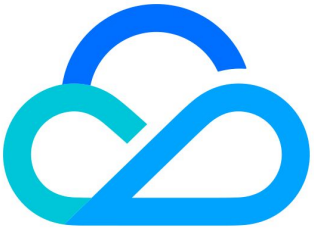
注册配置治理

TSF Consul（内测中）



腾讯云

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

文档目录

TSF Consul（内测中）
配置管理
配置管理概述

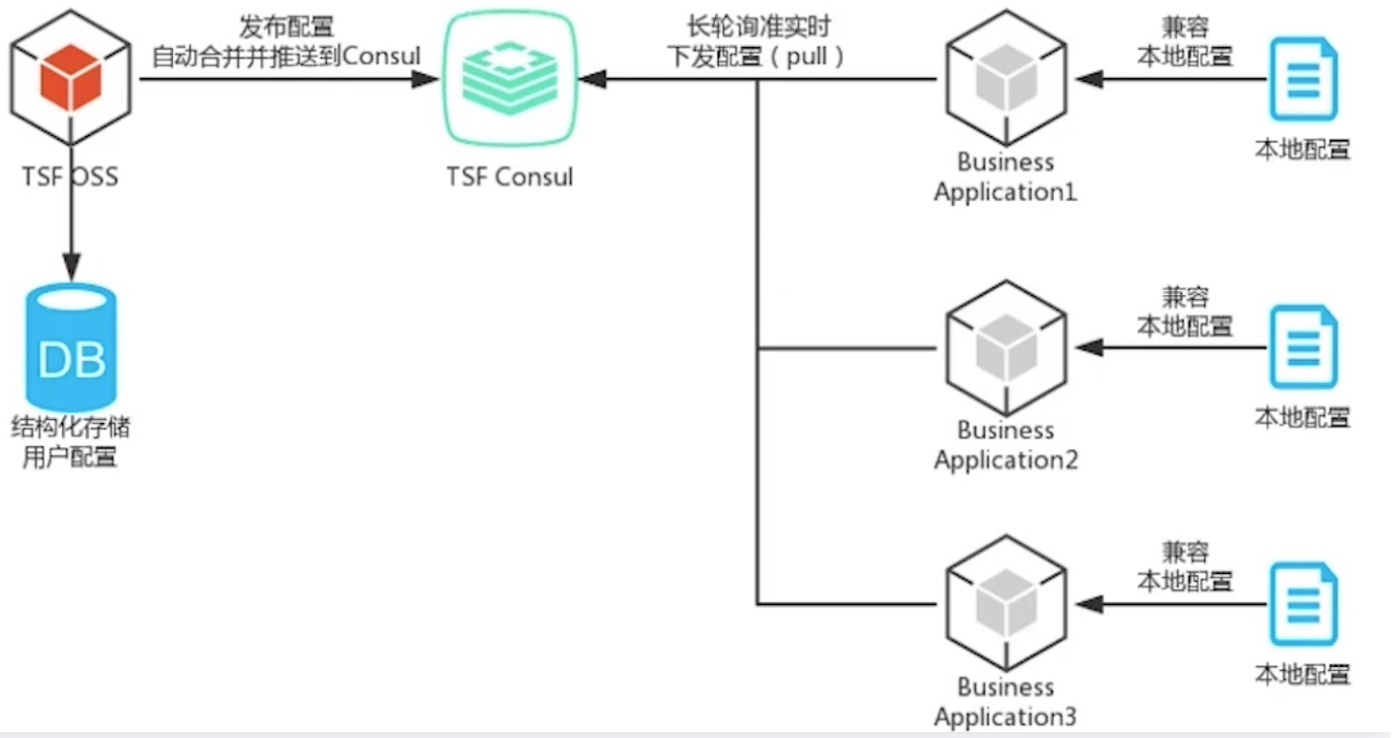
最近更新时间：2025-04-24 10:32:45

以下视频将为您介绍 TSF Consul 的配置管理功能：

观看视频

> **说明：**
> 当前 TSF-consul 已经转为内测，如果您有服务注册配置治理的需求，请您购买 Polaris（北极星）。

为了解决分布式环境下多台服务实例的配置统一管理问题，TSF consul 提供了配置管理功能。



配置类型

- **应用配置：**生效在单个应用上面，发布的范围是部署组维度，属于 TSF 平台上的配置。
- **全局配置：**生效在整个集群或者命名空间，发布的范围是命名空间维度，属于 TSF 平台上的配置。
- **本地配置：**是应用程序在代码工程中创建的配置（如 `application.yml` 和 `bootstrap.yml`）。
- **文件配置：**支持用户通过控制台将配置下发到服务器的指定目录。应用程序通过读取该目录下的配置文件实现特殊的业务逻辑。

配置类型	功能说明	适用应用类型	关联对象	发布对象
应用配置	动态更新 Spring Cloud，Dubbo 或者 Go-gRPC 应用内的配置	Spring Cloud，Dubbo 或 Go-gRPC 应用	应用	应用关联的部署组
全局配置	动态更新 Spring Cloud，Dubbo 或者 Go-gRPC 应用内的配置	Spring Cloud，Dubbo 或 Go-gRPC 应用	命名空间	命名空间
文件配置	将文件配置发布到实例指定路径，发布成功后触发回调	任何应用类型	应用	应用关联的部署组

说明：
原生应用与 Mesh 应用不支持应用配置和全局配置。

配置模板

为了方便用户保存常用的配置信息，TSF 提供了 Ribbon、Hystrix、Zuul 等 Spring Cloud 组件的配置模板。用户可以基于已有的配置模板进行修改，也可以自定义编写配置模板。
用户可以基于配置模板来创建应用配置或者全局配置。

配置导出

TSF 支持配置导出功能，帮助您更加地方便管理配置。

应用配置、全局配置、本地配置优先级

应用配置和全局配置属于 TSF 平台上的配置（下面称为**远程配置**），本地配置是应用程序在代码工程中创建的配置（例如 `application.yml` 和 `bootstrap.yml` ）。应用配置和全局配置的根本区别在于**配置发布的范围**，应用配置发布的范围是部署组维度，全局配置发布的范围是命名空间维度。
配置优先级：应用配置 > 全局配置 > 本地配置。
当用户通过 TSF 控制台发布**远程配置**，微服务应用会按照配置的 key 来进行合并操作。例如，微服务应用本地 `application.yml` 配置文件的内容中包括：

```
# application.yml
username: test_user1
feature.status: false
feature.color: red
```

TSF 平台上 `远程配置` 的内容如下：

```
# TSF 应用配置或者全局配置
```

```
username: test_user2
feature.status: true
```

当 远程配置 的发布范围包含了上面的服务实例，微服务应用会将远程配置和本地配置按照 key 进行合并，最终生成的配置如下：

```
# 远程配置与本地配置合并结果
username: test_user2
feature.status: true
feature.color: red
```

多份应用配置发布到同一个部署组

TSF 支持多份应用配置发布到同一个部署组，多份配置会根据发布时间的先后顺序以 key 进行合并。例如，应用 A 有两个应用配置项： config-1 、 config-2 。

- config-1 的配置内容：

```
# config-1
username: test_user1
feature.status: false
```

- config-2 的配置内容：

```
# config-2
username: test_user2
feature.color: red
```

- config-1 和 config-2 先后发布到部署组 group ，会按照 key 进行合并。

```
# config-1 与 config-2 合并结果
username: test_user2
feature.status: false
feature.color: red
```

应用配置

最近更新时间：2025-04-24 10:32:45

> **说明：**
> 当前 TSF-consul 已经转为内测，如果您有服务注册配置治理的需求，请您购买 Polaris（北极星）。

操作场景

应用配置生效在单个应用上面，发布的范围是部署组维度。应用配置功能**仅针对 Spring Cloud 应用，Dubbo 应用和 Go-gRPC 应用生效**，支持的功能如下：

- 创建配置项：一个配置项管理多个版本的配置。
- 生成新版本：基于历史版本生成新版本。
- 发布配置：支持发布配置到部署组。
- 查看发布情况：查看配置项发布到哪些部署组。
- 回滚：回滚到上一个版本的配置。

> **说明：**
> 原生应用与 Mesh 应用不支持分布式配置。

前提条件

在使用控制台的应用配置功能前，请确保已经按照 配置管理 配置了相关依赖项。

创建配置

应用配置功能有两个入口，一个入口是在单个应用的应用详情页内，另一个入口是在**配置管理**模块的**应用配置**。

1. 登录 TSF 控制台。
2. 在左侧导航栏选择 **TSF Consul** 后，单击**配置管理** > **应用配置**，进入应用配置页面。
3. 在页面顶部选择好地域和关联应用后，单击**新建配置**，填写配置信息。
   - 配置名称：填写配置名称，最长60个字符，只能包含字母、数字及分隔符（“-”、“_”），且不能以分隔符开头或结尾。
   - 配置内容：上传本地配置文件，使用 YAML 配置格式，YAML 格式规范参见 YAML 格式介绍，如果本地使用 Properties 配置格式，查看 将 Properties 转化为 YAML。

     > **注意：**
     > 单个应用配置版本的大小不能超过65535个字节，如果应用的配置超过了该上限值，可以分成多个应用配置项发布到同一个部署组，多个配置会合并成一份配置。

   - 版本号：填写初始版本号。

- 标签：用于分类管理资源，可留空。详情参见 标签 。
- 版本描述：填写应用配置初始版本的描述。
- 数据集：用于细粒度管理子账号权限，可留空。将应用配置添加到数据集中，数据集使用请参见 数据集管理 。

4. 单击**完成**。

发布配置

应用配置项创建完成后，用户需要将配置项发布到应用下的部署组上才能生效。

1. 在**应用配置**列表页，单击目标配置名称，进入详情页。
2. 在配置版本标签页，单击某个配置版本操作栏的**发布**，勾选配置发布的目标部署组，填写发布描述。
3. 单击**下一步**，对比当前配置版本和上一个版本的差异，确认无问题后单击**发布**。

生成新版本配置

1. 在**应用配置**列表页，单击目标配置名称，进入详情页。
2. 在配置版本标签页，单击某个配置版本操作栏的**生成新版本**，填写变更的新版本的配置内容和版本号。

> **注意：**
> 新版本配置的版本号不能与原版本相同。

3. 单击**完成**。

新版本配置生成后，您需要将新版本配置发布到绑定应用下的部署组上，即可生效。

配置合并逻辑说明

按照配置**下发时间**来排序执行合并（merge）。不同名的配置项中如果存在相同 key 会进行合并。合并规则：按照配置下发时间排序，离当前时间近的优先级较高。举例如下：

```
1. 创建配置项 config-abc，配置内容是 custom-key: value-1，发布时间 15:00:00
2. 创建配置项 config-bcd，配置内容是 custom-key: value-2， 发布时间15:00:01
```

最终在实例上生效的配置：`custom-key: value-2`

删除配置版本

> **注意：**
> 删除后所有数据将被清除且不可恢复，请提前备份数据。

1. 在**应用配置**列表页，单击目标配置名称，进入详情页。
2. 在配置版本标签页，单击某个配置版本操作栏的**删除**，确认后即可删除该配置版本。

查看部署组的配置发布历史

用户可以通过**查看发布信息**查看该配置相关部署组的配置发布记录。

1. 在**应用配置**列表页，单击操作列的**查看发布信息**，进入发布情况页面。
2. 展开部署组，查看该部署组的配置发布记录。
3. 单击每条发布记录， 可查看配置发布前后区别。

回滚配置

回滚配置会将部署组的配置回滚到**上一次**发布的版本。

1. 在**应用配置**列表页，单击操作列的**查看发布信息**，进入发布情况页面。
2. 找到目标部署组，单击操作栏的**回滚**，可查看回滚前后配置变化。
3. 单击**提交**，完成回滚。

全局配置

最近更新时间：2025-04-24 10:32:45

> **说明：**
> 当前 TSF-consul 已经转为内测，如果您有服务注册配置治理的需求，请您购买 Polaris（北极星）。

操作场景

全局配置生效在整个集群或者命名空间，发布的范围是命名空间维度。全局配置功能**仅针对 Spring Cloud 应用，Dubbo 应用和 Go-gRPC 应用生效**，支持的功能如下：

- 创建配置项：一个配置项管理多个版本的配置。
- 生成新版本：基于历史版本生成新版本。
- 发布配置：支持发布配置到命名空间。
- 查看发布情况：查看该配置相关命名空间的配置发布记录。
- 回滚：回滚到上一个版本的配置。

> **说明：**
> 原生应用与 Mesh 应用不支持全局配置。

前提条件

在使用控制台的全局配置功能前，请确保已经按照 配置管理 的操作步骤配置了相关依赖项。

创建配置

1. 登录 TSF 控制台。
2. 在左侧导航栏选择 TSF Consul 后，单击**配置管理 > 全局配置**，进入全局配置页面。
3. 在页面顶部选择好地域后，单击**新建配置**，填写配置信息。
   - 配置名称：填写配置名称，最长60个字符，只能包含字母、数字及分隔符（“-”、“_”），且不能以分隔符开头或结尾。
   - 配置内容：上传本地配置文件，使用 YAML 配置格式，YAML 格式规范参见 YAML 格式介绍，如果本地使用 Properties 配置格式，查看 将 Properties 转化为 YAML。

     > **注意：**
     > 单个全局配置版本的大小不能超过65535个字节，如果实际使用的配置超过了该上限值，可以分成多个全局配置项发布到同一个命名空间，多个配置会合并成一份配置。

   - 版本号：填写初始版本号。
   - 标签：用于分类管理资源，可留空。详情参见 标签。

- 版本描述：填写全局配置初始版本的描述。
- 数据集：用于细粒度管理子账号权限，可留空。将全局配置添加到数据集中，数据集使用请参见 数据集管理 。

4. 单击**完成**。

发布配置

全局配置项创建完成后，用户需要将配置项发布到命名空间上才能生效。

1. 在 全局配置 列表页，单击目标配置名称，进入详情页。
2. 在配置版本标签页，单击某个配置版本操作栏的**发布**，勾选配置发布的命名空间，填写发布描述。
3. 单击**下一步**，对比当前配置版本和上一个版本的差异，确认无问题后单击**发布**。

生成新版本配置

1. 在 全局配置 列表页，单击目标配置名称，进入详情页。
2. 在配置版本标签页，单击某个配置版本操作栏的**生成新版本**，填写变更的新版本的配置内容和版本号。

> **注意：**
> 新版本配置的版本号不能与原版本相同。

3. 单击**完成**。

> **说明：**
> 新版本配置生成后，您可以将新版本配置发布到绑定命名空间上，即可生效。

配置合并逻辑说明

按照配置**下发时间**排序执行合并（merge）。不同名的配置项中如果存在相同 key 会进行合并。合并规则：按照配置下发时间排序，离当前时间近的优先级较高。举例如下：

```
1. 创建配置项 config-abc，配置内容是 custom-key: value-1，发布时间 15:00:00
2. 创建配置项 config-bcd，配置内容是 custom-key: value-2， 发布时间15:00:01
```

最终在实例上生效的配置：`custom-key: value-2`

删除配置版本

> **注意：**
> 删除后所有数据将被清除且不可恢复，请提前备份数据。

1. 在 全局配置 列表页，单击目标配置名称，进入详情页。
2. 在配置版本标签页，单击某个配置版本操作栏的**删除**，确认后即可删除该配置版本。

查看命名空间的配置发布历史

用户可以通过**查看发布信息**查看该配置相关命名空间的配置发布记录。

1. 在 全局配置 列表页，单击操作列的**查看发布信息**，进入发布情况页面。
2. 展开命名空间，查看该命名空间的配置发布记录。
3. 单击每条发布记录， 可查看配置发布前后区别。

回滚配置

回滚配置会将部署组的配置回滚到**上一次**发布的版本。

1. 在 全局配置 列表页，单击操作列的**查看发布信息**，进入发布情况页面。
2. 找到目标命名空间，单击操作栏的**回滚**，可查看回滚前后配置变化。
3. 单击**提交**，完成回滚。

配置模板

最近更新时间：2025-04-24 10:32:45

> **说明：**
> 当前 TSF-consul 已经转为内测，如果您有服务注册配置治理的需求，请您购买 Polaris（北极星）。

操作场景

为了方便用户保存常用的配置信息，TSF 提供了 Ribbon、Hystrix、Zuul 等 Spring Cloud 组件的配置模板，用户可以基于已有的配置模板进行修改，也可以自定义编写配置模板。
用户可以基于配置模板来创建 应用配置 或者 全局配置 。

前提条件

在使用配置模板功能之前，请确保已经按照 配置管理 添加了代码注释。

新建配置模板

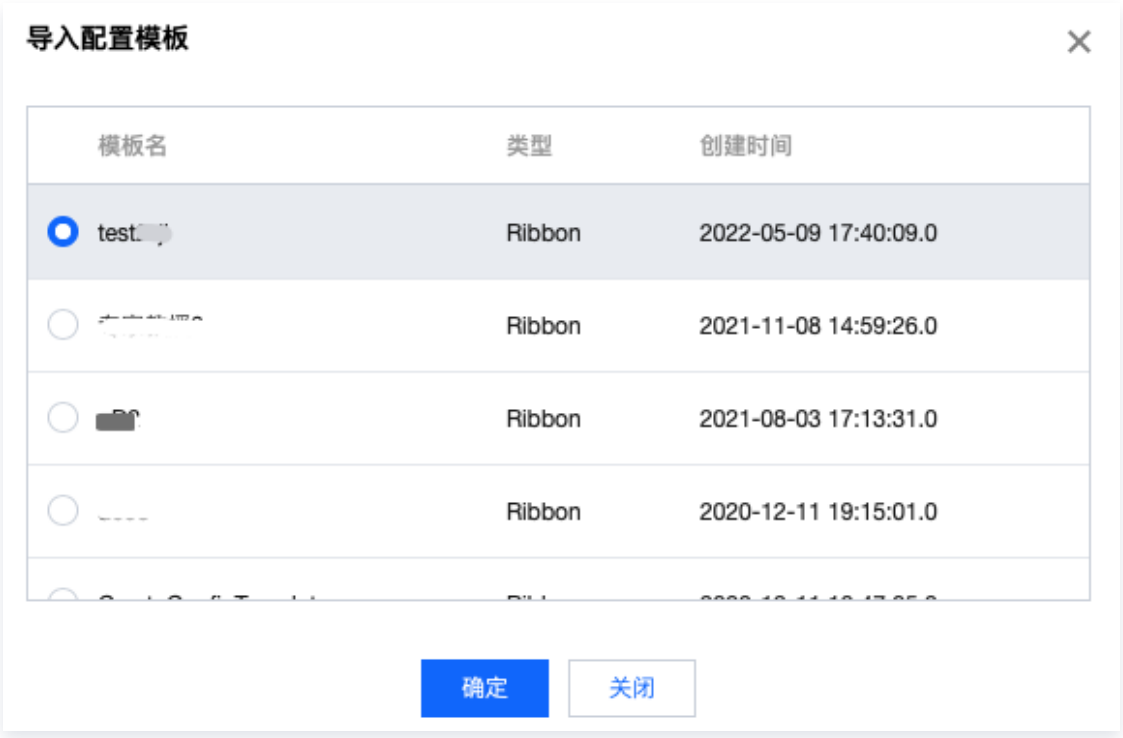
1. 登录 TSF 控制台 。
2. 在左侧导航栏选择 TSF Consul 后，单击**配置管理** > **配置模板**，进入配置模板页面。
3. 在页面顶部选择好地域后，单击**新建模板**，填写配置信息。
   - 模板名：填写模板名。
   - 类型：提供了 Ribbon、Hystrix 和 Zuul 组件的配置模板，您可以在此基础上修改，也可以自定义编写配置模板。
   - 配置内容：根据不同的类型，会自动生成对应的配置内容，用户可以进一步修改配置内容。
   - 描述：填写描述信息。
   - 数据集：用于细粒度管理子账号权限，可留空。将配置模板添加到数据集中，数据集使用请参见 数据集管理 。
4. 单击**提交**，完成新建。

使用配置模板

配置模板创建完成后，用户可以使用配置模板来创建应用配置或者全局配置，下面以全局配置举例。

1. 登录 TSF 控制台 。
2. 在左侧导航栏选择 TSF Consul，单击**配置管理** > **全局配置**，进入全局配置页面。

3. 在页面顶部选择好地域后，单击**导入配置模板**，勾选创建好的配置模板。



4. 单击**提交**，在新建配置页面中，补充全局配置的其他信息。

5. 单击**完成**，完成全局配置项创建。

文件配置

最近更新时间：2025-04-24 10:32:45

> **说明：**
> 当前 TSF-consul 已经转为内测，如果您有服务注册配置治理的需求，请您购买 Polaris（北极星）。

功能简介

文件配置功能支持用户通过 TSF 控制台将配置下发到服务器的指定目录，应用程序通过读取该目录下的配置文件实现特殊的业务逻辑。
文件配置支持如下功能：

- 创建文件配置项：一个文件配置项管理多个版本的配置。
- 生成新版本：基于历史版本生成新版本。
- 发布配置：支持发布配置到部署组。
- 发布情况：查看配置项的发布到哪些部署组。
- 回滚：回滚到上一个版本的配置。

应用场景

场景1：定时检查配置是否更新

- 应用程序中包含了读取指定目录配置文件的逻辑，例如定时去检查配置文件是否更新（通过文件 md5 是否变化等方式检查），如果更新了会执行特定逻辑。
- 在控制台上创建文件配置，下发到部署组。

场景2：动态替换 PHP 文件

通过控制台发布一个 PHP 文件到指定目录，来达到动态替换服务器上 PHP 文件的目的。

前提条件

能否使用文件配置功能，依赖于应用部署的环境是否满足以下条件：

- **对于使用虚拟机部署的应用**：只有2018年11月20号之后导入到集群的云主机上会具有满足应用配置功能的环境。
- **对于使用容器部署的应用**：该功能需要用户修改 Dockerfile。以下示例在 制作镜像 文档的基础上做修改：
  - 需要将 `tsf-consul-template-docker.tar.gz`（下载地址）添加到 `/root/` 目录下：

    ```
    ADD tsf-consul-template-docker.tar.gz /root/
    ```

  - 启动脚本中，需要执行 `/root/tsf-consul-template-docker/script` 目录下的 `start.sh` 脚本：

```
CMD ["sh", "-ec", "sh /root/tsf-consul-template-
docker/script/start.sh; exec java ${JAVA_OPTS} -jar ${jar} 2>&1"]
```

创建文件配置

1. 登录 TSF 控制台 。
2. 在左侧导航栏在左侧导航栏选择 **TSF Consul** 后，选择**配置管理** > **文件配置**，进入文件配置页面。
3. 在页面顶部选择好地域和关联的应用后，单击**新建配置**。
   - 配置名称：填写配置名称，最长60个字符，只能包含字母、数字及分隔符（ “-” 、 “_” ），且不能以分隔符开头或结尾。
   - 文件保存编码：支持 **utf-8** 和 **gbk**。
   - 配置内容：支持上传本地配置文件或者在控制台上直接编辑。
   - 配置文件名称：填写下发到服务器的配置文件的文件名称。
   - 版本号：填写文件配置初始版本的描述。
   - 版本描述：填写文件配置初始版本号。
   - 配置下发路径：配置下发到服务器的路径。
   - 后置脚本命令：配置下发到服务器后执行的命令（ **不需要** 包含 #! /bin/bash ）

   - 数据集：用于细粒度管理子账号权限，可不选。数据集使用，请参阅 数据集管理 。
   - 标签：用于分类管理资源，可不选。详情参见 标签 。
4. 单击**完成**。

发布配置

文件配置项创建完成后，用户需要将配置项发布到应用下的部署组上才能生效。

1. 在**文件配置**列表页，单击目标配置名称，进入详情页。
2. 在配置版本标签页，单击某个配置版本操作栏的**发布**，勾选配置发布的目标部署组，填写发布描述。

3. 单击**提交**，完成发布。

生成新版本配置

1. 在**文件配置**列表页，单击目标配置名称，进入详情页。
2. 在配置版本标签页，单击某个配置版本操作栏的**生成新版本**，填写变更的新版本的配置内容和版本号。

> **注意：**
> 新版本配置的版本号不能与原版本相同。

3. 单击**完成**。

> **说明：**
> 新版本配置生成后，您需要将新版本配置发布到绑定应用下的部署组上，即可生效。

删除配置版本

> **注意：**
> 删除后所有数据将被清除且不可恢复，请提前备份数据。

1. 在**文件配置**列表页，单击目标配置名称，进入详情页。
2. 在配置版本标签页，单击某个配置版本操作栏的**删除**，确认后即可删除该配置版本。

查看部署组的配置发布历史

用户可以通过**查看发布信息**查看该配置相关部署组的配置发布记录。

1. 在**文件配置**列表页，单击操作列的**查看发布信息**，进入发布情况页面。
2. 展开部署组，查看该部署组的配置发布记录。
3. 单击每条发布记录， 可查看配置发布前后区别。

回滚配置

回滚配置会将部署组的配置回滚到**上一次**发布的版本。

1. 在**文件配置**列表页，单击操作列的**查看发布信息**，进入发布情况页面。
2. 找到目标部署组，单击操作栏的**回滚**，可查看回滚前后配置变化。
3. 单击**提交**，完成回滚。

配置加密

最近更新时间：2025-06-23 10:45:41

> **说明：**
> 当前 TSF-consul 已经转为内测，如果您有服务注册配置治理的需求，请您购买 Polaris（北极星）。

场景说明

配置加密功能提供了对配置值加密的存储全套解决方案，通过增强源生 SDK 能力，同时兼容本地文件配置和分布式配置的配置值加密。

准备工作

1. 确保使用最新的 TSF SDK，参见 Spring Cloud 应用概述。
2. 按照 配置管理 添加了代码注释。
3. 下载 SDK 加密工具。
4. 准备需要加密的相关信息（此处为举例，用户使用时请调整）
   - 密码明文（plaintext）：TX_PwDemO_1hblsqT
   - 密钥（encypt password）：encyptPassword

SDK 加密工具

1. 找到加密工具包（spring-cloud-tsf-encrypt-1.1.1-RELEASE.jar）。
2. 执行以下命令对配置明文密码进行加密（**需升级到 Java8 161或以上版本，或使用 补丁 解决问题**）：

```
D:\repo\com\tencent\tsf\spring-cloud-tsf-encrypt\1.1.1-RELEASE>java -jar
spring-cloud-tsf-encrypt-1.1.1-RELEASE.jar encrypt TX_PwDemO_1hblsqT
encryptPassword
```

输出结果：

```
[encrypt] result:
3M7wGw2XtFc5Y+rxOgNBLrm2spUtgodjIxa+7F3XcAo=
```

用例：

```
D:\repo\com\tencent\tsf\spring-cloud-tsf-encrypt\1.1.1-RELEASE>java -jar
spring-cloud-tsf-encrypt-1.1.1-RELEASE.jar
At least 3 arguments required. Usage: [operation] [content] [password]
[operation]: Choose one from [encrypt | decrypt].
```

```
[content]: Plaintext when encrypt or ciphertext when decrypt.
[password]: Encrypt or decrypt password.
```

3. 执行以下命令对密文密码进行解密：

```
D:\repo\com\tencent\tsf\spring-cloud-tsf-encrypt\1.1.1-RELEASE>java -jar
spring-cloud-tsf-encrypt-1.1.1-RELEASE.jar decrypt
3M7wGw2XtFc5Y+rxOgNBLrm2spUtgodjIxa+7F3XcAo= encryptPassword
```

输出结果：

```
[decrypt] result:
TX_PwDemO_1hblsqT
```

用例：

```
D:\repo\com\tencent\tsf\spring-cloud-tsf-encrypt\1.1.1-RELEASE>java -jar
spring-cloud-tsf-encrypt-1.1.1-RELEASE.jar
At least 3 arguments required. Usage: [operation] [content] [password]
[operation]: Choose one from [encrypt | decrypt].
[content]: Plaintext when encrypt or ciphertext when decrypt.
[password]: Encrypt or decrypt password.
```

配置项填写方式

> **注意**
> 本地配置和线上配置同时支持（需要符合 spring-config 源生规范）。

本地 YAML

配置在 application.yml或application-*.yml：

```
tsf:
  inventory:
    password:
      encrypt1: ENC(3M7wGw2XtFc5Y+rxOgNBLrm2spUtgodjIxa+7F3XcAo=)
```

配置中心 YAML

配置在全局配置/应用配置，并发布：

```
tsf:
  inventory:
```

```
    password:
      encrypt2: ENC(3M7wGw2XtFc5Y+rxOgNBLrm2spUtgodjIxa+7F3XcAo=)
```

本地 Properties

配置在 application.properties或application-*.properties：

```
tsf.inventory.password.encrypt3=ENC(3M7wGw2XtFc5Y+rxOgNBLrm2spUtgodjIxa+7F3
XcAo=)
```

业务应用使用

环境变量（推荐）

在系统环境变量中配置密钥（password）：此时密钥泄露的风险最小。

```
tsf_config_encrypt_password=encryptPassword
```

JVM 参数（不推荐）

也可以在JVM参数中配置密钥（password）：

```
-Dtsf_config_encrypt_password=encryptPassword
```

启动参数（不推荐）

也可以在应用启动参数中配置密钥（password）：

```
--tsf_config_encrypt_password=encryptPassword
```

Java 测试代码

Java代码按照常规配置使用。

配置类：

```
@ConfigurationProperties("tsf.inventory.password")
@Component
@RefreshScope
public class PasswordConfiguration {

    private String encrypt1;
    private String encrypt2;
    private String encrypt3;

    @Value("${tsf.inventory.password.encrypt1}")
    private String encrypt4;
```

```
    @Value("${tsf.inventory.password.encrypt2}")
    private String encrypt5;
    @Value("${tsf.inventory.password.encrypt3}")
    private String encrypt6;

    // getters and setters
}
```

测试类：

```
@RestController
public class TestController {

    @Autowired
    private PasswordConfiguration pwConfig;
    /**
     * 显示明文密码
     *
     * @return 明文密码
     */
    @RequestMapping("/inventory/password")
    public String showPassword() {
        String content = "TX_PwDemO_1hblsqT";
        StringBuffer sb = new StringBuffer("Test Config
Encrypt/Decrypt:\n");
        // 内存读取
        sb.append(String.format("[%s]\t内存读取*.yml文件配置: %s\n",
                content.equals(

SpringCloudTsfApplication.ctx.getEnvironment().getProperty("tsf.inventory.p
assword.encrypt1")),

SpringCloudTsfApplication.ctx.getEnvironment().getProperty("tsf.inventory.p
assword.encrypt1")));
        sb.append(String.format("[%s]\t内存读取consul配置: %s\n",
                content.equals(

SpringCloudTsfApplication.ctx.getEnvironment().getProperty("tsf.inventory.p
assword.encrypt2")),

SpringCloudTsfApplication.ctx.getEnvironment().getProperty("tsf.inventory.p
assword.encrypt2")));
        sb.append(String.format("[%s]\t内存读取*.properties文件配置: %s\n",
```

```
                content.equals(
SpringCloudTsfApplication.ctx.getEnvironment().getProperty("tsf.inventory.p
assword.encrypt3")),
SpringCloudTsfApplication.ctx.getEnvironment().getProperty("tsf.inventory.p
assword.encrypt3")));
        // Bean读取
        sb.append(String.format("[%s]\tBean读取*.yml文件配置: %s\n",
content.equals(pwConfig.getEncrypt1()),
                pwConfig.getEncrypt1()));
        sb.append(String.format("[%s]\tBean读取consul配置: %s\n",
content.equals(pwConfig.getEncrypt2()),
                pwConfig.getEncrypt2()));
        sb.append(String.format("[%s]\tBean读取*.properties文件配置: %s\n",
content.equals(pwConfig.getEncrypt3()),
                pwConfig.getEncrypt3()));
        // @Value读取
        sb.append(String.format("[%s]\t@Value读取*.yml文件配置: %s\n",
content.equals(pwConfig.getEncrypt4()),
                pwConfig.getEncrypt4()));
        sb.append(String.format("[%s]\t@Value读取consul配置: %s\n",
content.equals(pwConfig.getEncrypt5()),
                pwConfig.getEncrypt5()));
        sb.append(String.format("[%s]\t@Value读取*.properties文件配置: %s\n",
content.equals(pwConfig.getEncrypt5()),
                pwConfig.getEncrypt5()));
        return sb.toString();
    }
}
```

输出结果如下：

```
Test Config Encrypt/Decrypt:
[true]  内存读取*.yml文件配置: TX_PwDemO_1hblsqT
[true]  内存读取consul配置: TX_PwDemO_1hblsqT
[true]  内存读取*.properties文件配置: TX_PwDemO_1hblsqT
[true]  Bean读取*.yml文件配置: TX_PwDemO_1hblsqT
[true]  Bean读取consul配置: TX_PwDemO_1hblsqT
[true]  Bean读取*.properties文件配置: TX_PwDemO_1hblsqT
[true]  @Value读取*.yml文件配置: TX_PwDemO_1hblsqT
[true]  @Value读取consul配置: TX_PwDemO_1hblsqT
[true]  @Value读取*.properties文件配置: TX_PwDemO_1hblsqT**
```

查看生效配置

最近更新时间：2025-04-24 10:32:45

> **说明：**
> 当前 TSF-consul 已经转为内测，如果您有服务注册配置治理的需求，请您购买 Polaris（北极星）。

操作场景

查看生效配置适用于查看已经下发到某个实例节点上的配置和已经生效的配置，已下发配置是由发布到部署组上的应用配置和全局配置合并之后的结果。
查看生效配置与关联应用/文件配置的区别是，关联应用/文件配置是指通过应用配置能力将某个配置下发到部署组的关联记录。

前提条件

- 仅 Spring Cloud 应用和微服务网关应用，且使用的 SDK 版本为1.29.x的 Finchley、Greenwich 或 Hoxton 的支持查看生效配置。
- 已经创建一个或者多个应用配置或者全局配置，并发布到目标部署组。具体操作步骤参见 应用配置 和 全局配置 。

操作步骤

1. 登录 TSF 控制台 。
2. 左侧导航栏选择应用管理后，在业务应用列表页选择目标应用。
3. 在左侧导航栏选择**应用部署>部署组**，单击目标部署组服务列表的“ID”，进入服务实例列表页面。

4. 选择要查看实例，单击操作列的**查看生效配置**。

> **注意**
> 由于生效配置是应用配置和全局配置合并之后的结果，因此不会保留原始配置的注释和结构。

查看已生效配置开关说明

支持本地对 TSF 控制台查看已生效配置功能进行关闭，下面为关闭配置，不配置时默认打开。

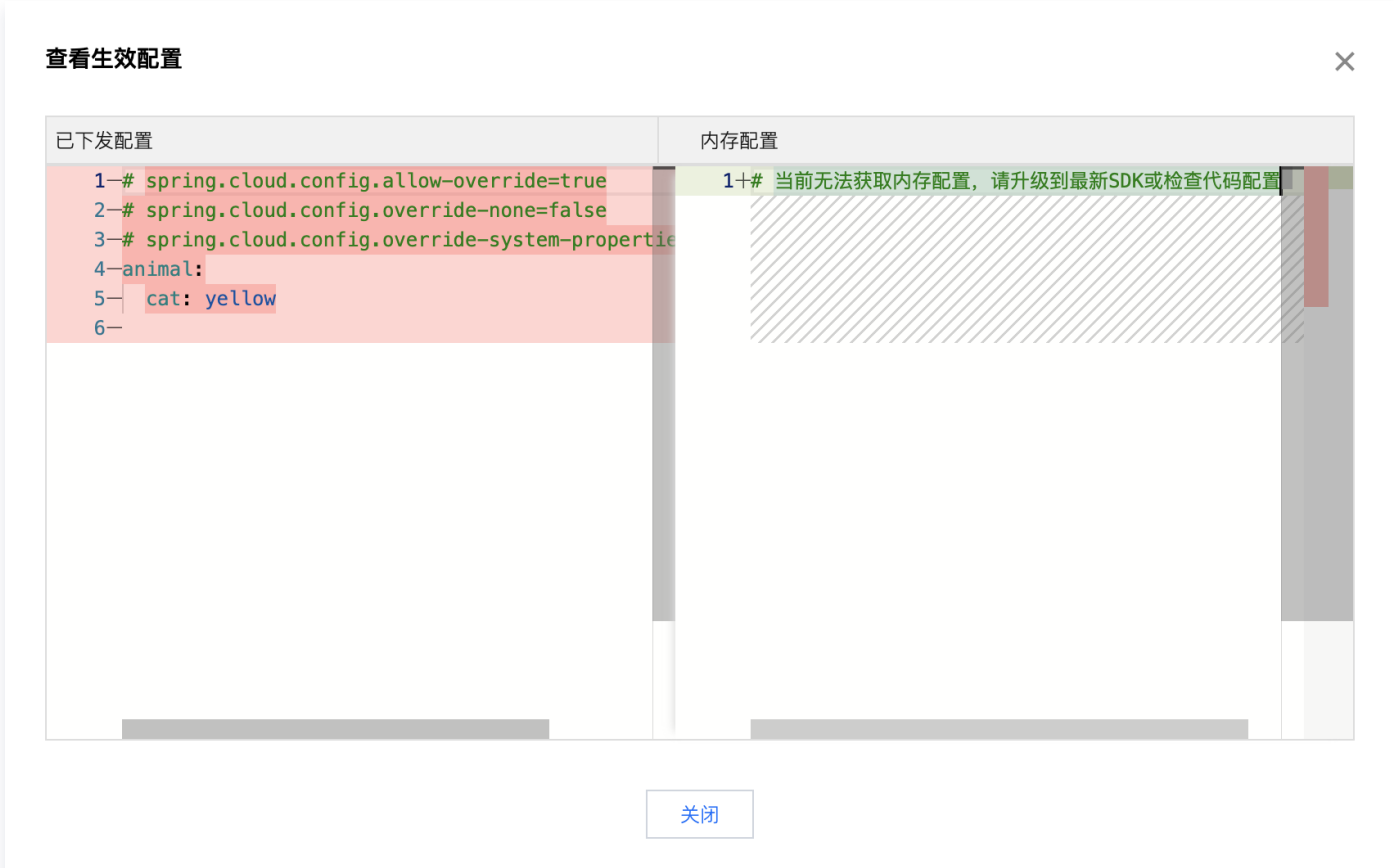
```
tsf:
  config:
    instance:
      released-config:
        lookup:
          enabled: false
```

关于无法查看配置的说明

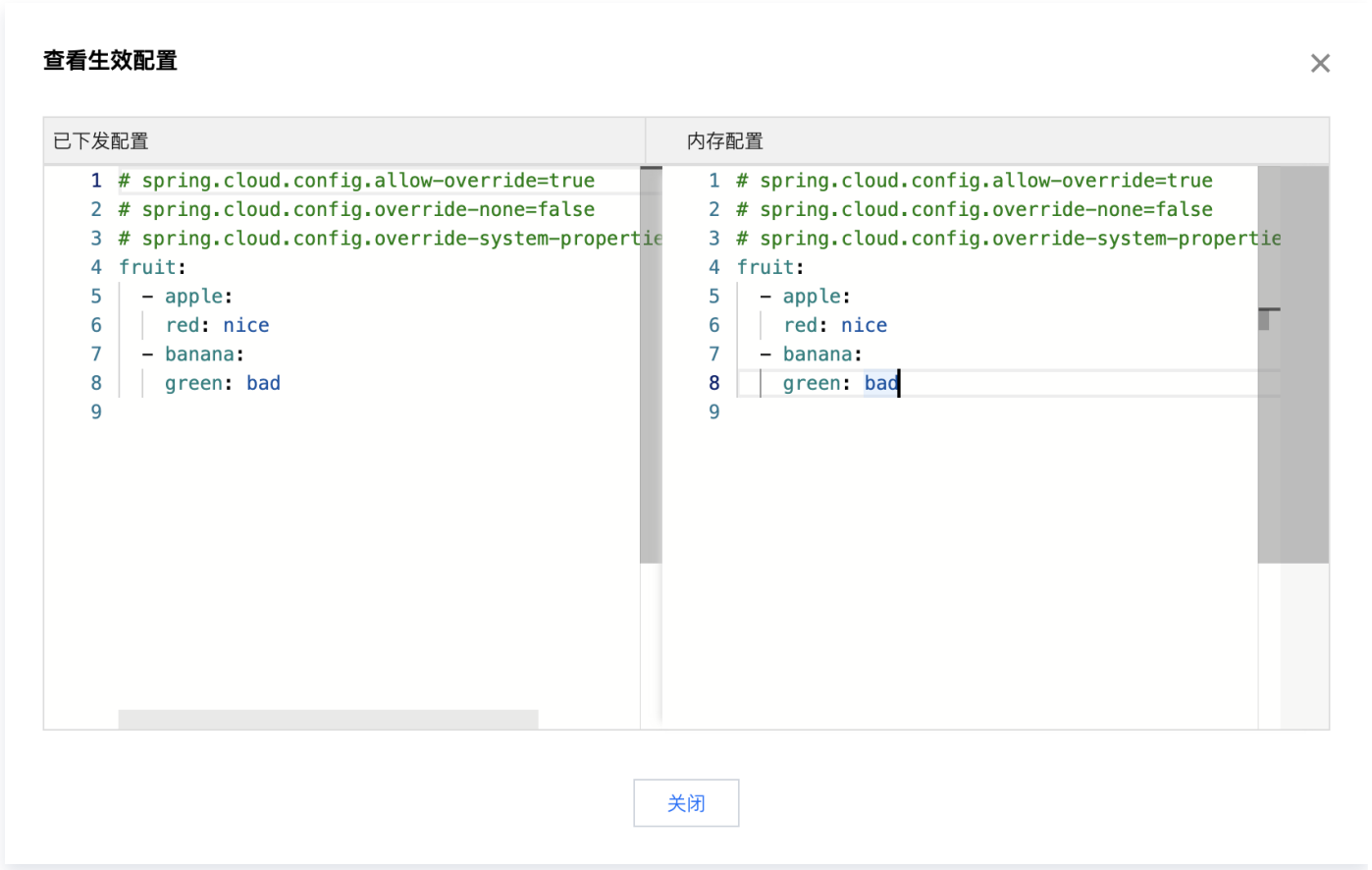
1. 使用旧版 SDK 或者非 Spring Cloud 应用。
   无配置：


查看生效配置
已下发配置
内存配置
1
1
关闭

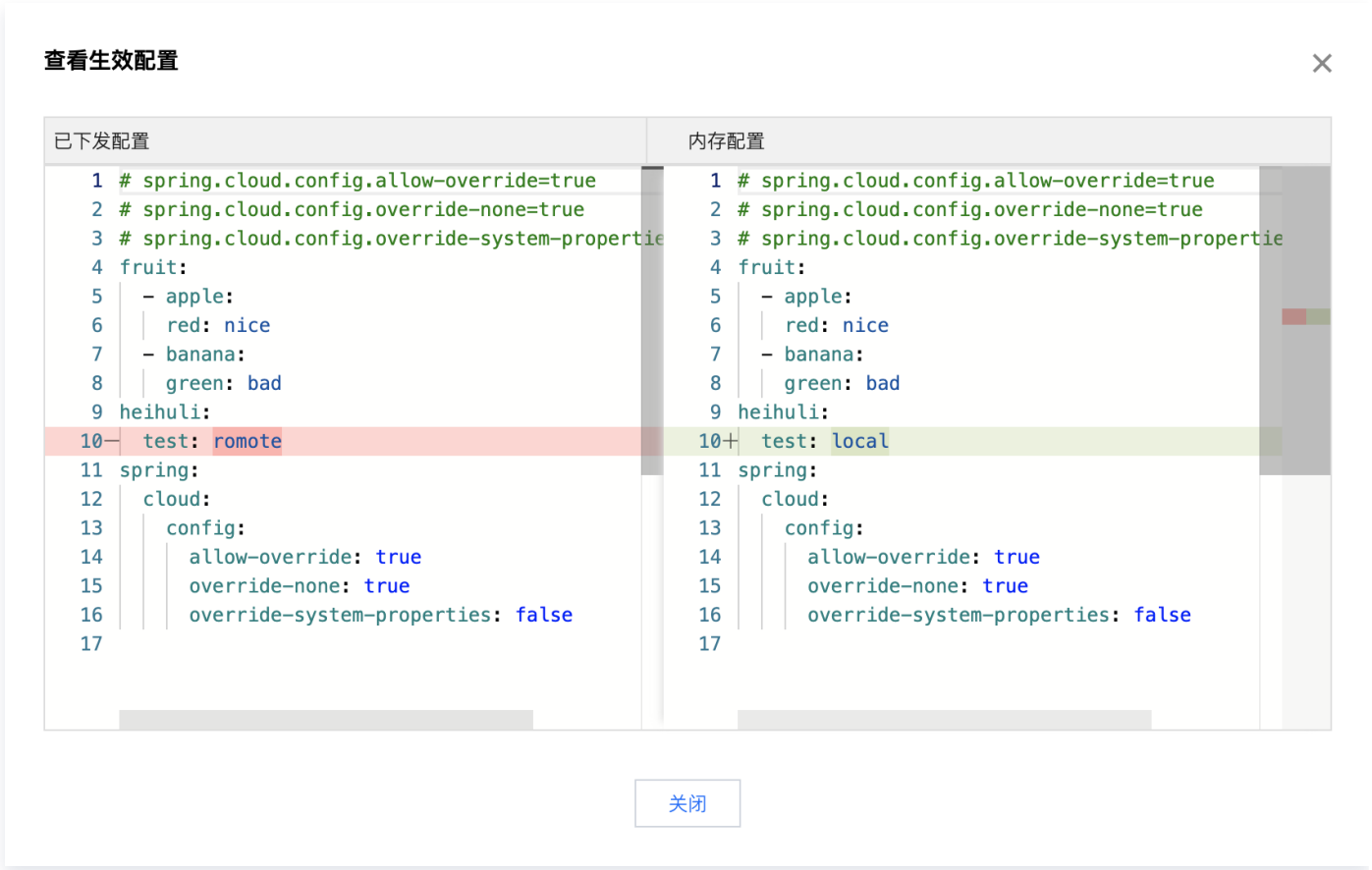
有配置：


查看生效配置
已下发配置
内存配置
1 # spring.cloud.config.allow-override=true
2 # spring.cloud.config.override-none=false
3 # spring.cloud.config.override-system-properti
4 animal:
5 cat: yellow
6
1 # 当前无法获取内存配置，请升级到最新SDK或检查代码配置
关闭

2. 使用新版本 SDK ，默认外部配置优先。



3. 使用新版本 SDK ，设置本地配置优先。



4. 在本地关闭查看已生效配置的开关。

服务治理
服务管理

最近更新时间：2025-04-24 10:32:45

> **说明：**
> 当前 TSF-consul 已经转为内测，如果您有服务注册配置治理的需求，请您购买 Polaris（北极星）。

场景说明

服务是微服务平台管理的基本单元，当微服务注册到注册中心时，服务会显示在服务治理列表中。您也可以提前手动创建服务，设置服务限流、路由等规则，当服务注册上来后规则会下发到匹配**服务名**的服务实例上。

创建服务

1. 登录 TSF 控制台。
2. 在左侧导航栏单击 **TSF Consul** 后，选择**服务治理**，选择好地域和所属命名空间后单击**新建服务**。
3. 填写服务的基本信息后，单击**提交**，完成创建。

编辑服务

在 服务治理 页面，单击目标服务操作栏的**编辑**，填写备注信息后，单击**提交**，完成修改。

删除服务

> **注意：**
> 只有当服务的状态为**离线**时，即服务运行实例数为0时，才能删除服务。

在 服务治理 页面，单击目标服务操作栏的**删除**，确认**删除**后即可删除服务。

服务监控

在 TSF 控制台服务治理页面可以看到线上服务的请求量、请求成功率、请求平均耗时等监控数据。数据统计周期都是24小时。

- **请求量**：对一个服务，统计其作为服务提供者，被所有消费他的服务消费者发起调用的24小时内总调用数。
- **请求成功率**：对于一个服务，统计24小时内其作为服务提供者，成功向消费他的所有服务消费者返回请求的总数比上服务请求总数。
- **请求平均耗时**：对于一个服务，统计24小时内其作为服务提供者，统计消费者从发起调用到调用返回到服务提供者的耗时平均值。

服务实例和手动下线

一个服务由多个服务实例构成，您可以在**服务详情页 > 服务实例列表**，查看服务下有多少实例。服务实例有**在线**和**离线**两种状态，离线的服务实例不会被其他服务发现，会在上次心跳时间24小时后自动清除。
当服务实例不可用且仍然注册到注册中心时，会导致请求发送到该问题实例上，此时可以开启**屏蔽实例**来手动下线该实例。服务被屏蔽后，该服务实例将不会被其他服务发现，流量不会分发到该实例上。

接口列表

最近更新时间：2025-04-24 10:32:45

> **说明：**
> 当前 TSF-consul 已经转为内测，如果您有服务注册配置治理的需求，请您购买 Polaris（北极星）。

操作场景

TSF 框架在微服务注册时，会自动收集并注册微服务提供的 API 接口，用户可通过 TSF 控制台实时掌握当前微服务提供的 API 情况。接口列表显示服务对外暴露的 API 列表，同时 API 调试提供用户在线调试 API 的能力。
您可以在 TSF 控制台中，通过接口列表查看 API 的详细信息，并进行 API 在线调试。

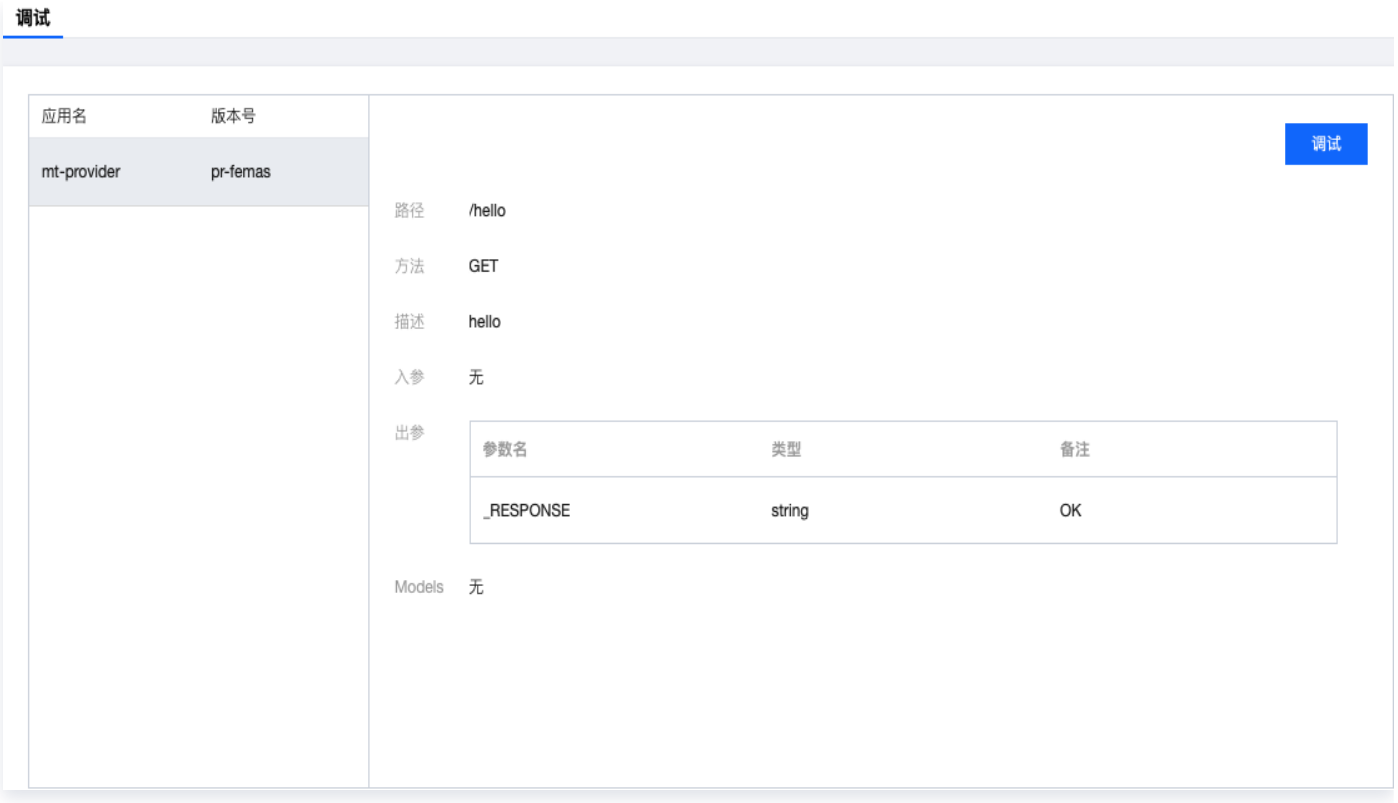
前提条件

要使用 API 列表和调试功能，需要先将服务的 API 注册到注册中心，具体请参见开发手册 API 注册。

查看 API 信息

1. 登录 TSF 控制台。
2. 在左侧导航栏单击 **TSF Consul** 后，选择**服务治理**，单击微服务的“ID”，进入服务概览页。
3. 在服务详情页，单击顶部的**接口列表**，会显示服务对外提供的 API 列表。

4. 单击 API 的“接口路径”，可以查看到 API 的详细信息。
   API 详情按照**应用名/版本号**划分显示了 API 的详细信息，包括：路径、方法、描述、入参、出参。其中 Models 表示参数中的复杂类型。



手动录入 API

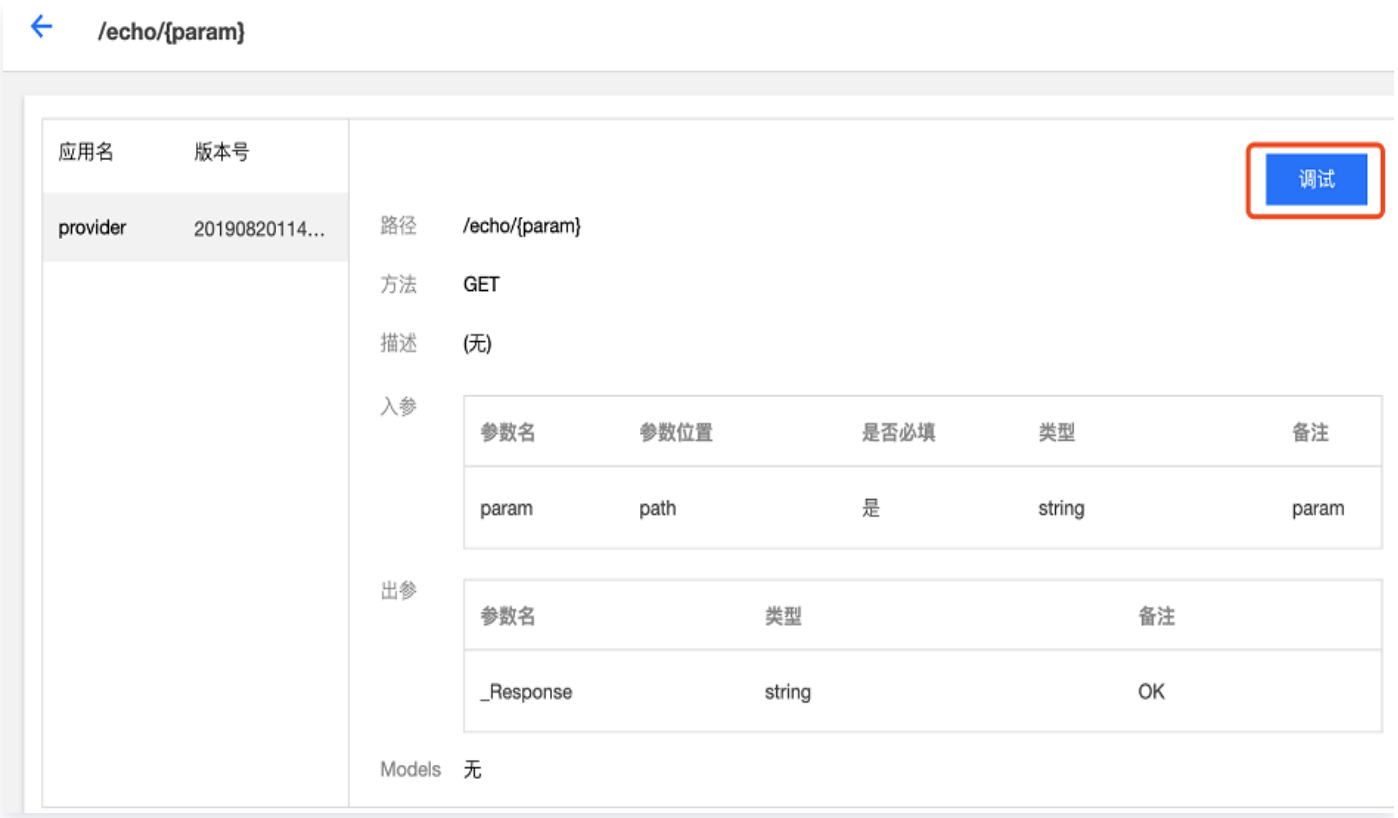
TSF 支持手动录入API，适用于当微服务尚未注册到注册中心，但是希望将 API 进行提前录入，方便配置一些服务治理规则的场景。

1. 在 服务治理 页面，单击微服务的“ID”，进入服务概览页。
2. 在服务详情页，选择**接口列表**页签，单击**手动录入API**。
3. 输入 API 路径（其中 path 参数可以使用 {param} 来进行描述），并选择请求方法。
4. 单击**完成**，完成录入。

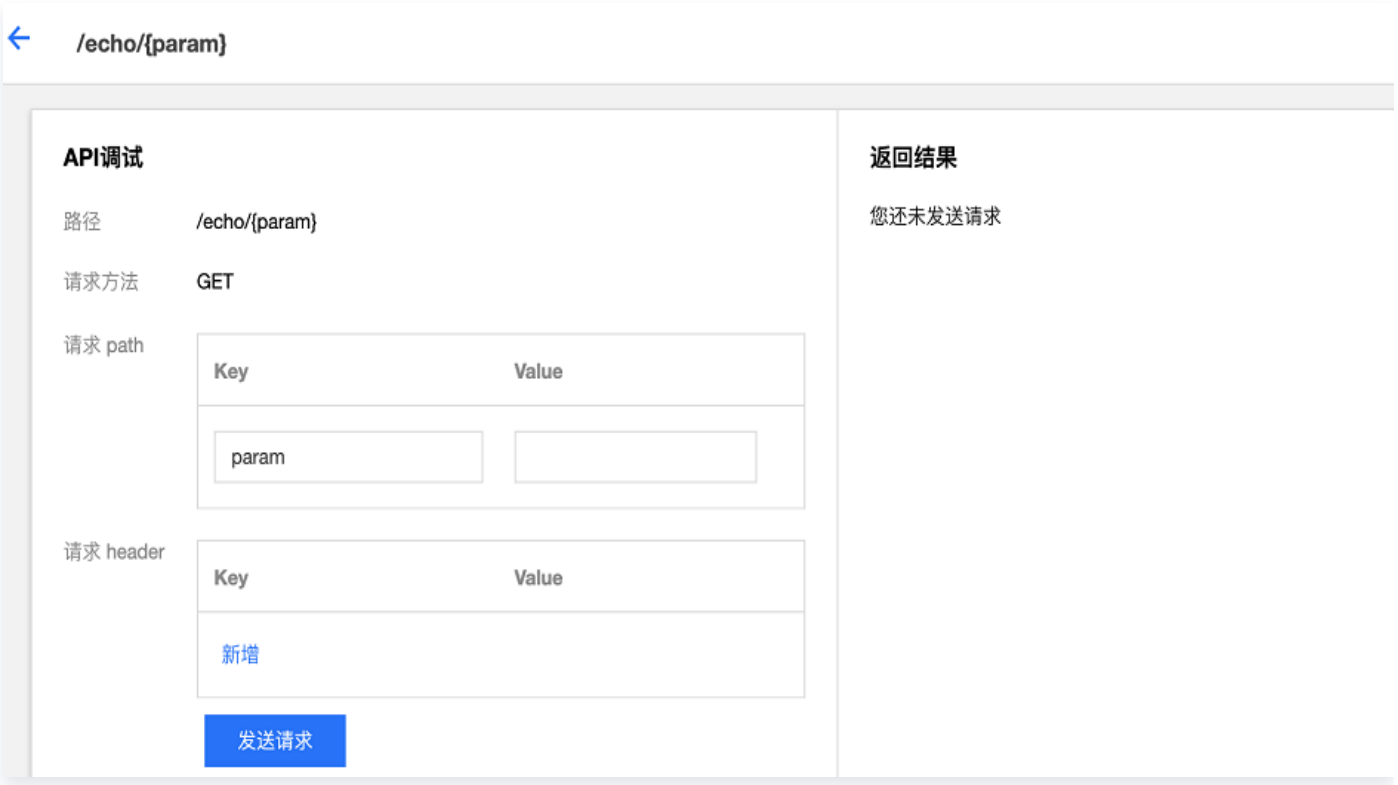
> **说明**
> - 同一个微服务下，通过请求路径和请求方法确认唯一的一个 API，不允许创建同请求方法和路径的 API。
> - 当注册中心获取到微服务注册的 API 与手动录入的 API 相同时，会认为是同一个 API。
> - 当注册中心判断某个 API 在当前任何一个部署组上都没有注册，会展示 API 状态为离线。
> - 仅当 API 状态为离线时，该 API 才可以被删除。

API 调试

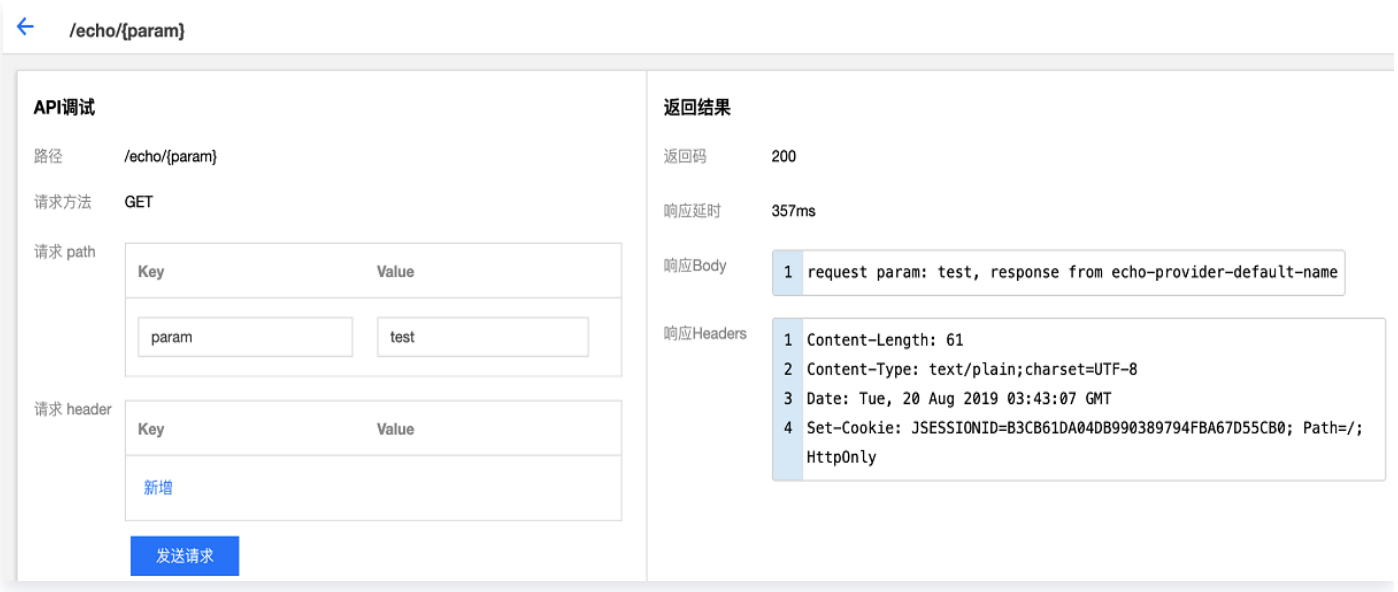
1. 在 服务治理 页面，单击微服务的“ID”，进入服务概览页。
2. 在服务详情页，选择**接口列表**页签，单击接口操作栏的**调试**，进入 API 调试页面。



3. 单击右上角的**调试**，填写调用 API 的默认参数，单击**发送请求**。



4. 右侧会展示调用 API 的返回结果。


/echo/{param}
API调试
路径
/echo/{param}
请求方法
GET
请求 path
Key
Value
param
test
请求 header
Key
Value
新增
发送请求
返回结果
返回码
200
响应延时
357ms
响应Body
1 request param: test, response from echo-provider-default-name
响应Headers
1 Content-Length: 61
2 Content-Type: text/plain;charset=UTF-8
3 Date: Tue, 20 Aug 2019 03:43:07 GMT
4 Set-Cookie: JSESSIONID=B3CB61DA04DB990389794FBA67D55CB0; Path=/;
HttpOnly

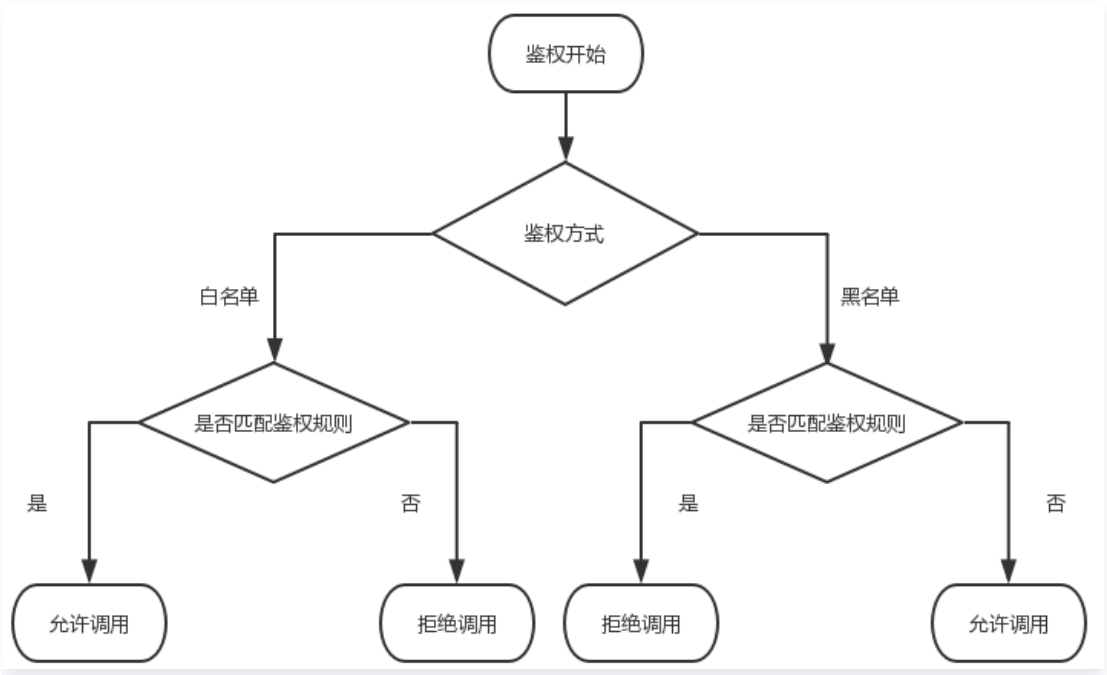
服务鉴权
服务鉴权原理

最近更新时间：2025-04-24 10:32:45

服务鉴权是处理微服务之间相互访问权限问题的解决方案。配置中心下发鉴权规则到服务，当请求到来时，服务根据鉴权规则判断鉴权结果，如果鉴权通过，则继续处理请求，否则返回鉴权失败的 HTTP 状态码403（Forbidden）。

鉴权原理

鉴权流程如下：



服务鉴权功能支持白名单和黑名单两种鉴权方式。

- **白名单**：当请求匹配**任意一条**鉴权规则时，允许调用；否则拒绝调用。
- **黑名单**：当请求匹配**任意一条**鉴权规则时，拒绝调用；否则允许调用。

以下视频将为您介绍 TSF 服务鉴权原理：
观看视频

多个鉴权规则

一个服务可能有多个鉴权规则，多个鉴权规则之间是逻辑或（OR）的关系，只要请求满足任意一条鉴权规则，就相当于匹配成功。

示例说明

示例1：

需求：服务 provider-demo 只允许来自 consumer-demo 服务**且**带有 user=foo 的自定义标签的请求调用。
解决方案：要满足上面的鉴权需求，用户可以在 provider-demo 的鉴权页面，设置鉴权方式为白名单，鉴权规则如下图（注意最后要将生效状态改为生效）：

结论：要满足 `逻辑与 AND` （既满足条件 A，又满足条件 B）时，需要使用标签表达式。

示例2：

需求：服务 provider-demo 只允许来自 consumer-demo 服务**或**带有 user=foo 的自定义标签的请求调用。
解决方案：要满足上面的鉴权需求，用户可以在 provider-demo 的鉴权页面，设置鉴权方式为白名单，创建2条鉴权规则，如下图：

结论：要满足 `逻辑与 OR` （满足条件 A 或 条件 B）时，需要使用多条鉴权规则。

说明

白名单鉴权方式示例：
鉴权规则内容是 username 等于 foo，当请求中带有 username=foo 的 tag 时，因为匹配规则，服务允许调用；当请求中带有 username=bar 的 tag 时，因为不匹配规则，服务拒绝调用。
黑名单鉴权方式示例：鉴权规则内容是 username 等于 foo，当请求中带有 username=foo 的 tag 时，因为匹配规则，服务拒绝调用；当请求中带有 username=bar 的 tag 时，因为不匹配规则，服务允许调用。

服务鉴权使用说明

最近更新时间：2025-04-24 10:32:45

> **说明：**
> 当前 TSF-consul 已经转为内测，如果您有服务注册配置治理的需求，请您购买 Polaris（北极星）。

使用鉴权功能时，用户需要先在客户端配置依赖项，然后在 TSF 控制台设置鉴权规则。

步骤1：配置依赖项

- 对于 Spring Cloud 应用，请参见开发指南中的 服务治理 。
- 对于 Mesh 应用，无须额外配置。

步骤2：设置鉴权规则

1. 登录 TSF 控制台 。
2. 在左侧导航栏单击 TSF Consul 后，选择**服务治理**，单击目标服务的“ID”，进入服务详情页。
3. 在页面上方选择**服务鉴权**页签，鉴权方式选择**不启用**，单击**新建鉴权规则**，填写鉴权规则和生效状态。
   - 规则名称：填写服务鉴权规则名称。
   - 类型：选择鉴权类型。
   - 鉴权标签：支持**系统标签**和**自定义标签**两种类型，详细说明请参见 系统与自定义标签 。
   - 生效状态：开启后，此条鉴权规则将开始生效。

4. 单击**完成**，返回服务鉴权规则列表页面，选择鉴权方式开启鉴权功能。
   - 不启用：关闭鉴权功能。

- 白名单：服务放行匹配鉴权规则的请求调用。
- 黑名单：服务拒绝匹配鉴权规则的请求调用。

步骤3：（可选）切换鉴权方式

用户可以通过控制台，从一种鉴权模式切换到另外一种鉴权模式。

- 不启用切换到白名单（或者黑名单）：开启鉴权功能，选择已生效鉴权规则对应的鉴权方式。

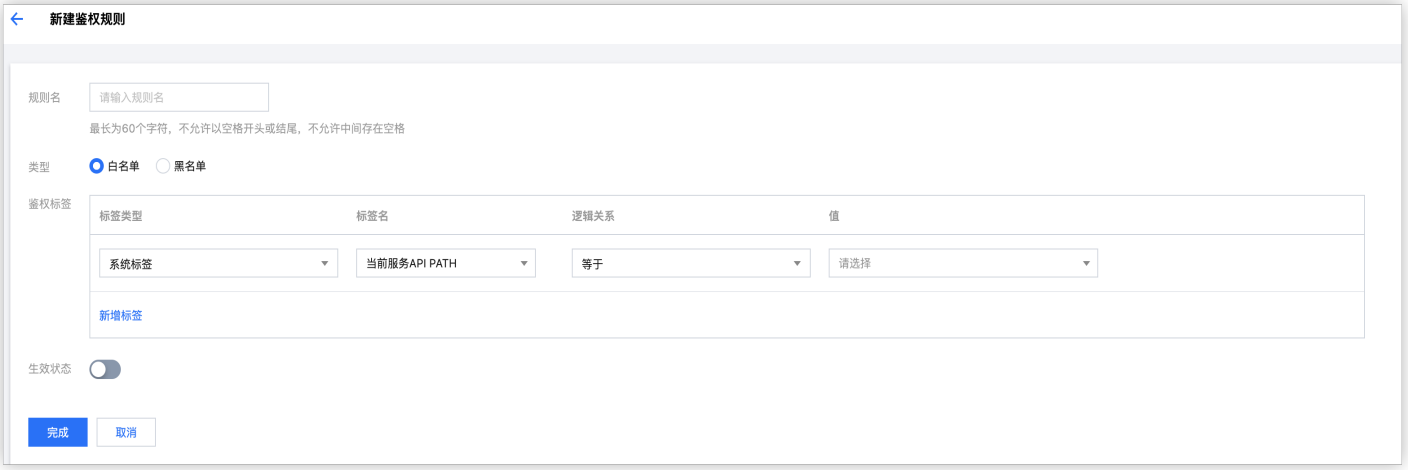
> **说明**
> 至少有一条规则确认生效后，才能开启对应的鉴权方式。例如只有一条黑名单规则生效，则只能切换到黑名单鉴权方式，不能切换到白名单鉴权方式。

- 白名单切换到黑名单（或黑名单切换到白名单）：不能直接切换，需要先切换到不启用，生效一条黑名单（白名单）规则后，才能切换到黑名单（白名单）。
- 白名单（或黑名单）切换到不启用：关闭鉴权功能。

步骤4：验证鉴权效果

以官网 Demo（包含一个 provider-demo 应用和一个 consumer-demo 应用）为例说明如何验证鉴权功能。consumer-demo 中已包含鉴权依赖 jar 包，因此这里只需要说明在控制台上创建鉴权规则用来限制特定 API 的调用。consumer-demo 中提供了三个 API `/echo-rest/{str}` 、`/echo-async-rest/{str}` 、`/echo-feign/{str}` 。

参见 部署 Spring Cloud TSF 应用 将官网 Demo 部署到 TSF 平台后，在控制台上新建鉴权规则，类型选择**白名单**，鉴权规则的标签表达式如下：



创建好规则后，登录云服务器，使用 curl 命令来验证鉴权是否生效。其中 <IP> 为云服务器的IP，在云主机列表页面获取；<PORT> 为主机端口，在部署组基本信息页面的**服务访问**模块获取。

命令	预期
`curl IP:PORT/echo-rest/hello?user=test`	正常返回
`curl IP:PORT/echo-async-rest/hello?user=test`	返回鉴权失败
`curl IP:PORT/echo-feign/hello?user=test`	返回鉴权失败

限制说明

等于、不等于、包含、不包含属于严格匹配，正则表达式属于模糊匹配。因此当系统标签是被调方 API PATH 时，目前仅支持使用**正则表达式**的逻辑关系来匹配带参数的 API 请求。
例如标签的逻辑关系是正则表达式，值填写 `/echo/.*`，可以匹配带参数的请求 /echo/test123（其中 test123 是参数）；当标签的逻辑关系是等于、不等于、包含、不包含关系，值是 `/echo/{param}` 时，不能匹配带参数的请求 /echo/test123（其中 test123 是参数）。

服务路由
服务路由基本原理

最近更新时间：2025-04-24 10:32:45

概述

服务路由功能是指用户根据符合自己特定要求的属性选择服务的提供者，对服务间流量的分配起到掌控的作用。为了满足客户的定制化需求，TSF 支持用户定制自己的路由标签，并支持选择不同的逻辑形式配置标签值，定向分配流量。 总而言之，服务路由功能的主要作用是将调用流量按照自己的需求进行分配。

应用场景

- **场景1：**用户在使用 TSF 运行自己的业务时，由于业务的复杂程度，经常需要部署数目庞大的服务运行在现网环境中。这些服务运行在属性不同的实例上、部署在不同的地域中，用户经常需要根据符合自己特定要求的属性选择服务的提供者，对服务间流量的分配起到掌控的作用。
- **场景2：**在微服务的场景下，用户研发新版本上线的迭代周期越来越快，稳定敏捷的上线新版本需要微服务框架能够支持灰度发布、金丝雀发布、滚动发布等发布方式。通过服务路由功能，用户可以配置流量分配权重，设置某些权重的流量被分配到某个版本号中，为灰度发布等上线模式提供了无需终止服务的底层能力支持。

以下视频将为您介绍 TSF 的服务路由功能：
观看视频

服务路由原理

要实现服务路由需要完成两部分操作：

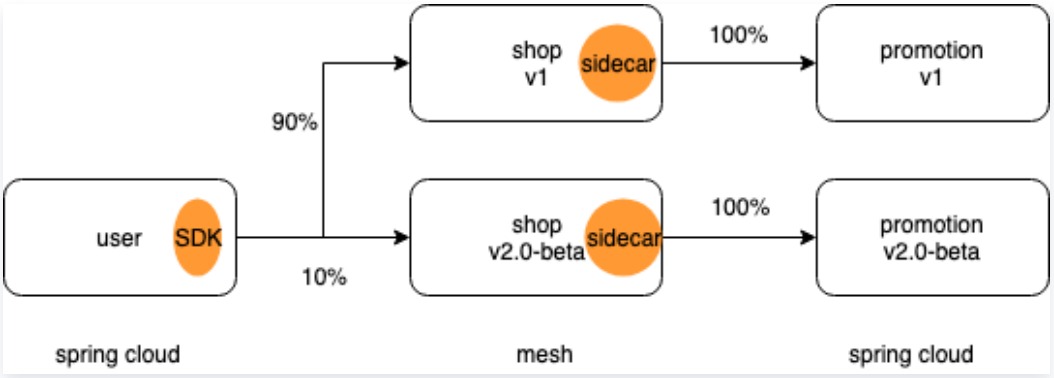
- 在控制台上，给**服务端（服务提供者）**设置路由规则。
- **客户端（服务消费者）**获取路由规则， 根据规则来分发请求。

以 `user -> shop -> promotion` 为例说明服务路由的原理，三个服务特点如下：

- user：Spring Cloud 应用，使用路由 SDK。
- shop：Mesh 应用，有两个版本 v1 和 v2.0-beta。
- promotion：Spring Cloud 应用，有两个版本 v1 和 v2.0-beta。

服务调用和路由情况如下图所示。用户需要在控制台创建如下路由规则：

- shop 服务详情页中配置路由规则：90%的流量分配到 v1 版本，10%的流量分配到 v2 版本。
- promotion 服务详情页中配置路由规则：服务名等于 shop 且版本号为 v1 的流量100%分配到 v1 版本，服务名等于 shop 且版本号为 v2 的流量100%分配到 v2 版本。


shop
v1
sidecar
100%
promotion
v1
90%
user
SDK
shop
v2.0-beta
sidecar
100%
promotion
v2.0-beta
10%
spring cloud
mesh
spring cloud

服务路由使用说明

最近更新时间：2025-04-24 10:32:45

> **说明：**
> 当前 TSF-consul 已经转为内测，如果您有服务注册配置治理的需求，请您购买 Polaris（北极星）。

场景说明

使用路由功能前，用户需要在客户端配置依赖项，然后在 TSF 控制台设置路由规则。

配置依赖项

- Spring Cloud 应用请参见开发手册中的 服务路由，添加依赖和注解。
- 对于 Mesh 应用，如果希望使用基于自定义标签的路由，需要在代码中设置标签，关于如何设置标签参见 Mesh 开发使用指引。

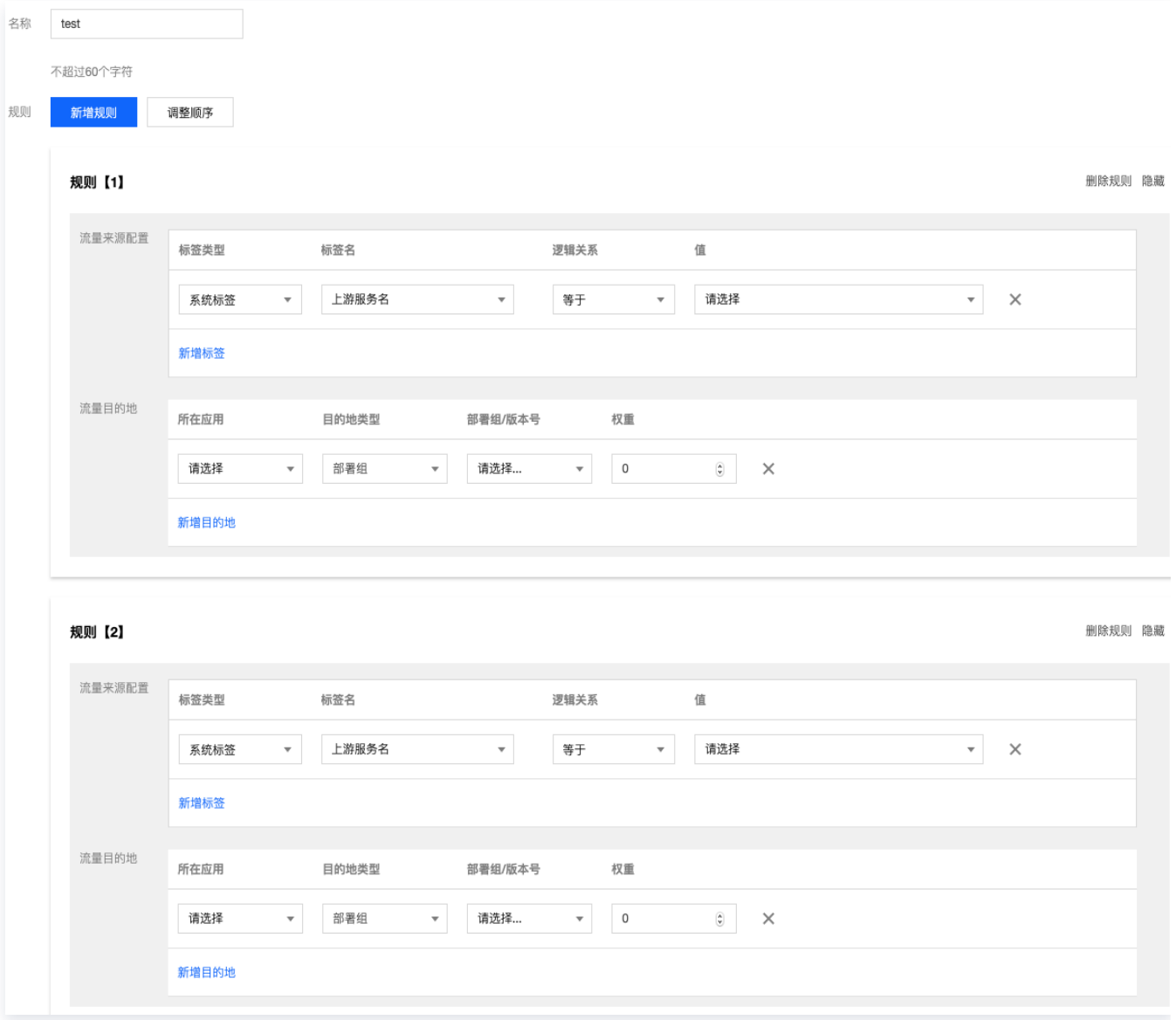
新建路由规则

1. 登录 TSF 控制台。
2. 在左侧导航栏单击 TSF Consul 后，选择**服务治理**，单击目标服务的“ID”，进入服务详情页。
3. 在页面上方选择**服务路由**页签，单击**新建路由规则**，填写规则后单击**提交**。

> **说明：**
> 一个微服务下最多50条路由规则。

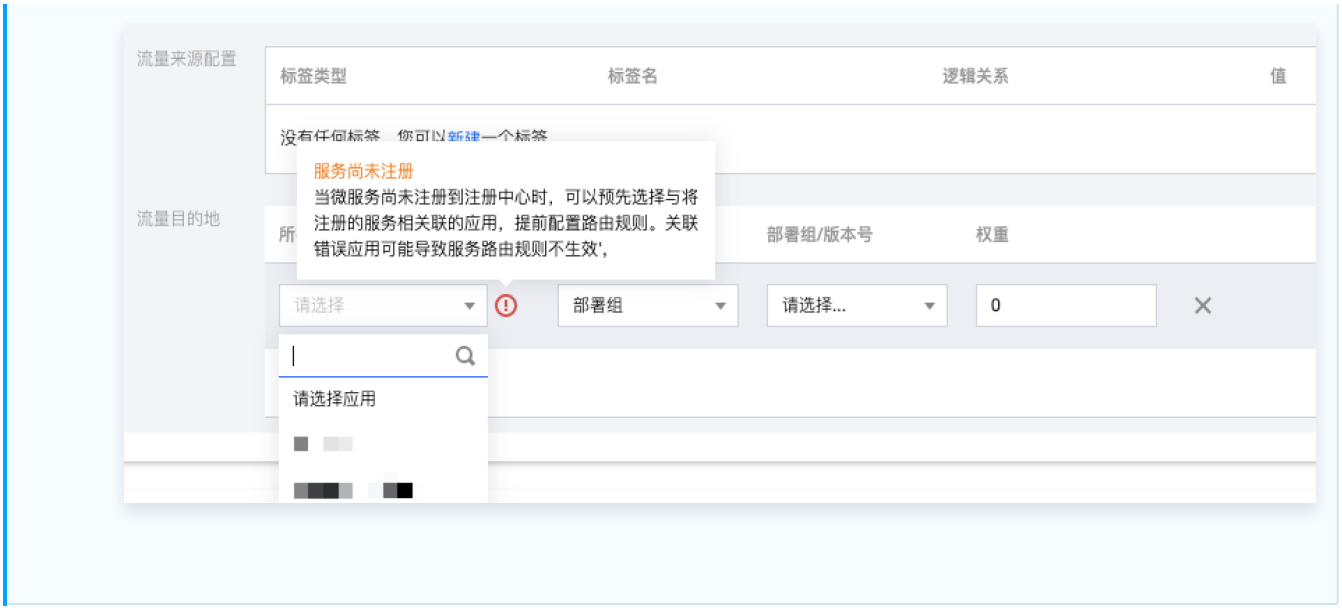
- 名称：填写路由规则名称，不超过60个字符。
- 流量来源配置：支持设置系统标签和自定义标签表达式，详细说明请参见 系统与自定义标签。

- 流量目的地：支持部署组和版本号两种目的地类型，确保权重加总为100。



说明：

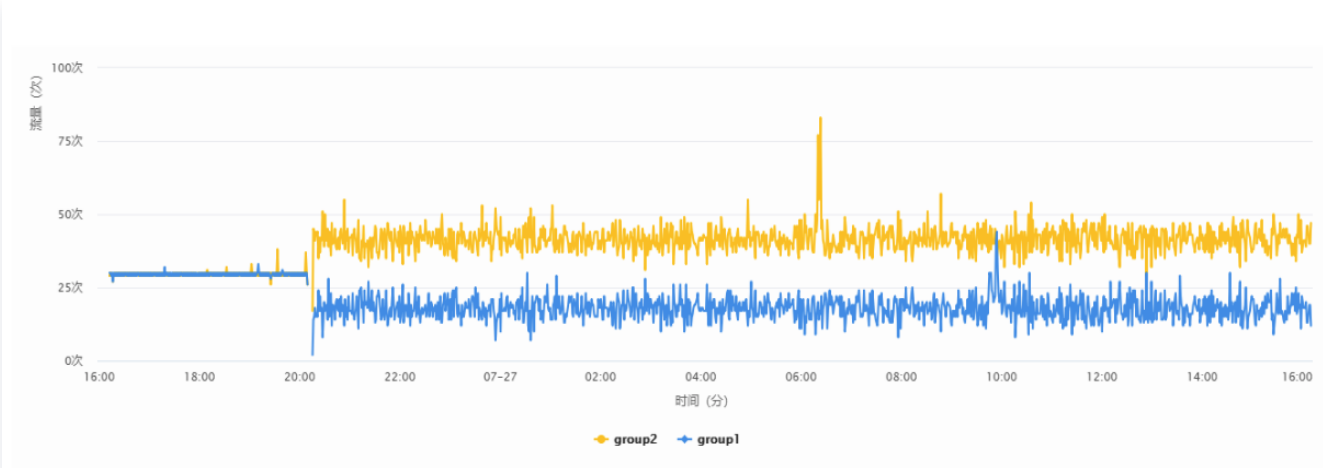
- 当服务在线时，您可以通过服务当前关联的应用来过滤部署组，配置流量规则指向哪一个部署组。
- 当服务尚未上线或已经离线时，系统无法判断该服务与哪一个部署组关联，您可以预先选择与将注册的服务相关联的应用，并选择对应的部署组或者版本号，提前配置路由规则。



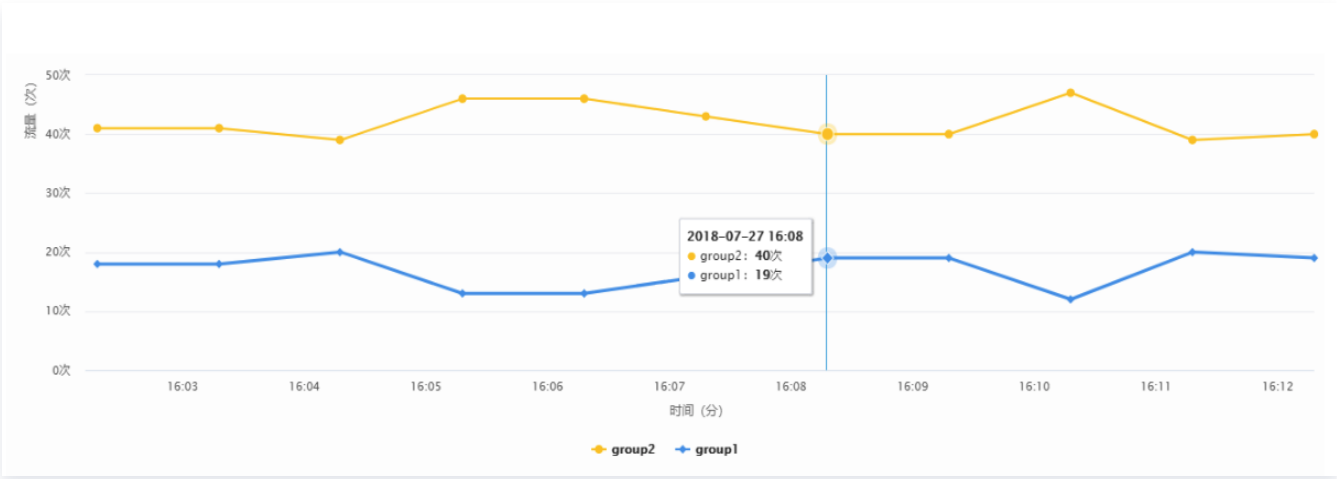
启用路由规则

1. 在服务列表页面，单击目标规则**生效状态**栏的切换按钮，当按钮为蓝色表明已经生效。

2. 配置生效后，可以在列表项的下面流量分配图中查看流量分配情况，用户可以选择时间段，查看部署组上流量分配情况。
   - 24小时内的流量分配情况如下：



- 应用路由规则后10分钟之内的流量分配曲线如下：



> **注意**
> 路由配置了流量的权重比例后，要使路由准确性达到预期，请求数至少要在1000以上；如果请求样本数不高的情况下偏差会比较大，样本数越高准确性就才越高。

容错保护

开启容错保护后，会实现兜底策略。例如服务设置了如下路由规则：

- 10%的流量分配到 v1 版本。
- 40%的流量分配到 v2 版本。
- 50%的流量请求分配到 v3 版本。

假设场景：v1 版本的实例全部不可用：

- 如果不开启容错保护，仍然会有10%的请求分发到 v1 版本的实例上，此时请求会失败。
- 如果开启容错保护，SDK 发现 v1 版本的实例不可用时，会采用 Round Robin 轮询算法将请求随机分发到所有可用实例上。

使用说明

- 填写路由规则需要在服务提供方进行配置，例如 A 服务调用 B 服务，需要在 B 服务上配置服务路由规则。

- 对于 Spring Cloud 服务，配置路由规则后，若配置的目标部署组无法运行，流量将按照原有默认的轮询方式分配到其他部署组上。
- 对于 Spring Cloud 服务，当服务提示未绑定应用时，需要在服务详情页单击编辑，绑定服务，才能开始配置路由规则。**服务绑定应用操作，一经绑定，不能修改**。
- Spring Cloud 服务调用其他服务的场景时，要使服务路由生效，需要确保 Spring Cloud 服务使用了 SDK 并添加开启路由注解，详情请参见开发手册中 服务路由 。
- 对于 Mesh 应用，配置路由规则后，若配置的目标部署组无法运行，则路由规则配置失败，请求无法发送。

编辑路由规则

> **说明：**
> 在生效状态的路由规则可以编辑，编辑之后立即生效。

1. 在服务列表页面，单击已经提交的规则操作栏的**编辑**。在编辑页面，仅支持编辑规则的详情，不支持编辑规则类型。
2. 单击**提交**，完成路由编辑。

删除路由规则

> **说明：**
> 在生效状态的服务路由规则不能被删除，只能先停用，再删除。

在服务列表页面，单击已经提交的规则操作栏的**删除**，**确认**后即可删除。

限制说明

等于、不等于、包含、不包含属于严格匹配，正则表达式属于模糊匹配。因此当系统标签是被调方 API PATH 时，目前仅支持使用**正则表达式**的逻辑关系来匹配带参数的 API 请求。

- 当标签的逻辑关系是正则表达式，值填写 `/echo/.*` 时，可以匹配带参数的请求 /echo/test123（其中 test123 是参数）。
- 当标签的逻辑关系是等于、不等于、包含、不包含关系，值是 `/echo/{param}` 时，不能匹配带参数的请求 /echo/test123（其中 test123 是参数）。

服务路由实践教程

最近更新时间：2025-04-24 10:32:45

灰度发布

- 使用目的：当用户需要上线新的功能时，希望使用灰度发布的手段在小范围内进行新版本发布测试。
- 使用方法：用户可以将新的程序包上传到原有的应用中。用户选择按照权重的方式配置路由规则，填写权重大小，并选择目标版本版本号，便可以实现使用部分流量进行灰度发布的能力。生效中的权重可以被编辑，实时生效，间接实现了滚动发布的功能。

同地机房优先

- 使用目的：当企业规模较大时，单个机房的容量已经不能满足业务需求，业务经常出现跨机房部署的情况。然而由于异地跨机房调用出现的网络延迟问题，需要能够保证服务消费方能优先调用本地的服务消费方，这就需要采用服务路由的方式。
- 使用方法：用户选择系统自带标签路由选项，配置系统自带标签为发起方 IP，在正则表达式中填写服务消费方的 IP 字段规则。对于服务提供方，用户可以将 IP 地址相近的实例归属在同一个部署组上，作为目标部署组，实现优先调用同地机房。

部分账号内测

- 使用目的：希望配置某些使用者使用的版本为新的内测版本。
- 使用方法：用户可以配置自定义标签为用户 ID，设置 ID 值的正则表达式计算方式，保证服务消费方发起的请求带有以上条件的流量分配到服务提供方的某个版本号上，实现账号内测功能。

其他实践

在实际的使用中，用户也可以通过服务路由功能，实现优先保护重要服务的运行质量、前后端分离、读写分离等功能。

服务限流

最近更新时间：2025-04-24 10:32:45

> **说明：**
> 当前 TSF-consul 已经转为内测，如果您有服务注册配置治理的需求，请您购买 Polaris（北极星）。

场景说明

服务限流主要是保护服务节点或者数据节点，防止瞬时流量过大造成服务和数据崩溃，导致服务不可用。当资源成为瓶颈时，服务框架需要对请求做限流，启动流控保护机制。

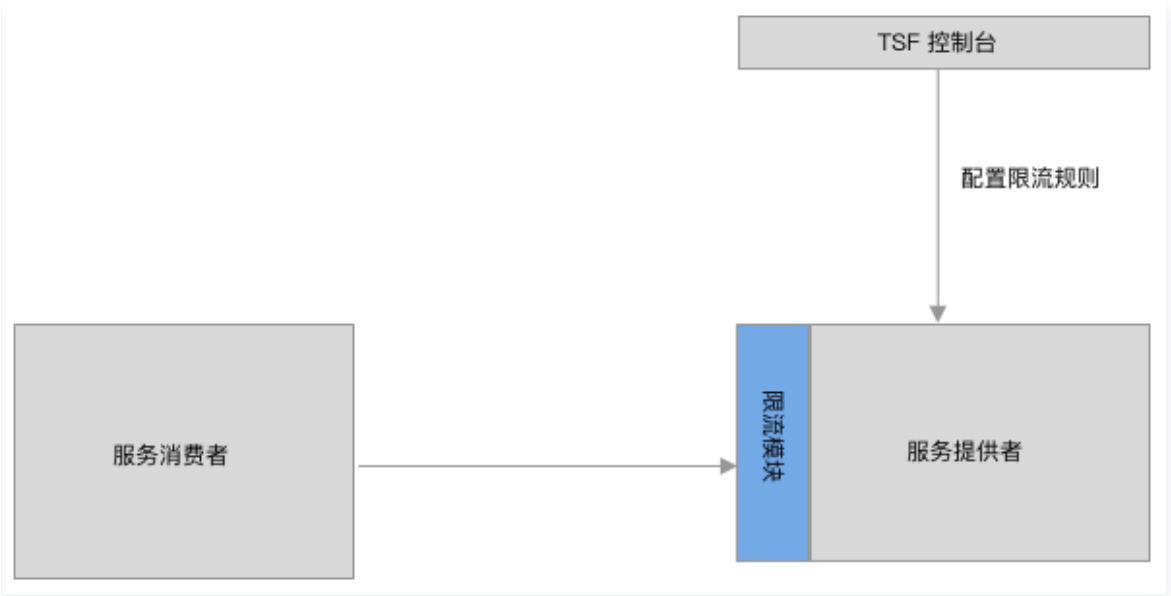
功能说明

限流的原理是监控服务流量的 QPS 指标，当达到指定的阈值时进行流量控制，避免被瞬时高峰流量冲垮，从而确保服务的高可用。
TSF 目前支持在被调服务上设置限流规则，限流模式有两种：

- 单机限流：适合用于给每个服务实例设置兜底保护。
- 集群限流：适合用于设置整个服务的流量上限。

服务的限流对象（下文中称为“限流资源”）可以通过标签表达式灵活配置，常见的限流对象如当前服务、当前服务的特定 API 等，并且可以通过标签表达式区分不同的调用来源，针对不同的调用关系进行限流。一条限流规则主要包括以下几个元素：

- 限流粒度：支持**服务整体限流**和**基于标签限流**，基于标签限流是指通过标签表达式表示**被调方的限流资源**和**调用来源**。
- 限流阈值：支持按照**请求数**（如果单位时间设置为1秒，则限流阈值为 QPS）或者**并发线程数**（仅Spring Cloud F G H 2020 1.40之后版本支持）限流。
- 生效状态：限流规则是否生效。



应用场景

场景1：根据调用方进行限流

调用关系中包括调用方和被调用方，一个被调服务可能同时被多个服务调用。在限流规则中，**限流粒度**字段可以用于根据调用来源进行流量控制，举例如下：

- **不区分调用者**：限流粒度选择**服务整体限流**时，来自任何调用者的请求都将进行限流统计。如果限流资源的调用总和超过了这条规则定义的阈值，则触发限流。
- **针对特定的调用者**：限流粒度选择**基于标签限流**，设置系统标签为**上游服务名**，逻辑关系为**等于**，值为特定的调用服务。
- **针对除特定调用者之外的调用方**：限流粒度选择**基于标签限流**，设置系统标签为**上游服务名**，逻辑关系为**不等于**，值为特定的调用服务。

区分调用方除了使用上游服务名等系统标签外，还可以使用自定义标签来区分带有不同业务信息的调用。例如针对特定用户 `foo` 的调用进行限流，可以在代码中设置**user**参数，然后在限流规则中配置业务标签为**user**，逻辑关系为**等于**，值为 `foo` 。
Spring Cloud 应用在代码中设置参数可参见 参数传递 ，Mesh 应用参见 设置自定义标签 。

场景2：针对不同的资源进行限流

一个服务包含一个或多个 API，TSF 支持针对服务或者 API 进行限流。在限流规则中，**限流粒度**字段可以用于区分不同的限流资源，举例如下：

- **针对当前服务**：无须额外设置。
- **针对特定的API**： 限流粒度选择**基于标签限流**，设置系统标签为**当前服务的 API Path**，逻辑关系为**等于**，值为特定的 API Path。如果需要指定 HTTP Method，则需要再增加一条 `HTTP Method` 的系统标签来约束。
- **针对特定API之外的API**：限流粒度选择**基于标签限流**，设置系统标签为**当前服务的 API Path**，逻辑关系为**不等于**，值为特定的 API Path。如果需要指定 HTTP Method，则需要再增加一条 `HTTP Method` 的系统标签来约束。

操作步骤

要使用限流功能，用户需要在客户端配置依赖项，然后在 TSF 控制台设置限流规则。

步骤1：配置依赖项

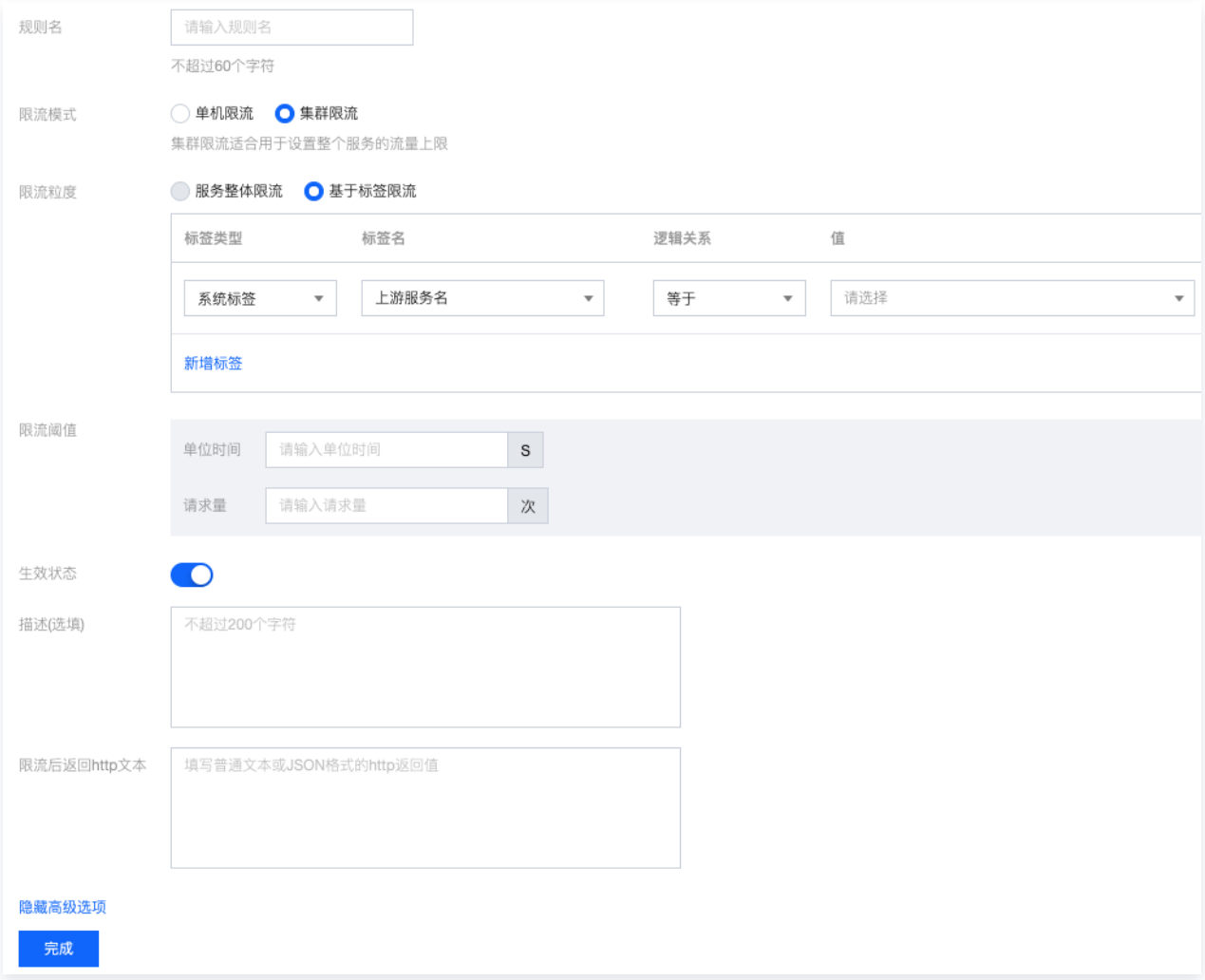
- 对于 Spring Cloud 应用，请参见开发指南中的 服务治理 。
- 对于 Mesh 应用，如果希望使用基于标签的限流，需要在代码中设置标签，参见 设置自定义标签 。

步骤2：新建限流规则

前提条件：服务列表上有“在线”状态的微服务。

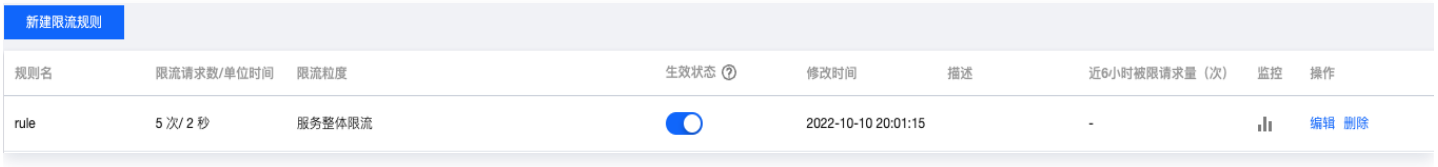
1. 登录 TSF 控制台 。
2. 在左侧导航栏单击 **TSF Consul** 后，选择**服务治理**，单击目标服务的“ID”，进入服务详情页。
3. 在页面上方选择**服务限流**页签，单击**新建限流规则**。

4. 填写限流规则信息。
   - 规则名：填写规则名称，不超过60个字符。
   - 限流模式：支持**单机限流**和**集群限流**两种模式
     - 单机限流：单机限流适合用于给每个服务实例设置兜底保护。
     - 集群限流：集群限流适合用于设置整个服务的流量上限。
   - 限流粒度：支持**全局限流**和**标签限流**两种方式。
     - 服务整体限流：不区分限流来源，统计所有请求。**单个服务仅支持1条服务整体限流规则。**
     - 基于标签限流：根据标签规则设置限流。支持**系统标签**和**自定义标签**两种类型，详细说明请参见 系统和自定义标签 。
   - 限流阈值：支持按照**请求量**或者**并发线程数**限流。
     - 请求量：填写单位时间和请求数，如果单位时间设置为1秒，则限流阈值为QPS。
     - 并发线程数：仅Spring Cloud F G H 2020 1.40之后版本支持。
   - 生效状态：是否立即启用限流规则。
   - 描述：填写描述信息。
   - 限流后返回 HTTP 文本：填写普通文本或者 JSON 格式的 HTTP 返回值。
5. 单击**完成**，完成限流规则创建，返回服务限流规则列表。



步骤3：启动限流规则

在限流规则列表中，可以修改规则的**生效状态**。多条限流规则都是生效状态时，只要服务接收到的请求满足**任意一条**限流规则，就会触发限流逻辑。

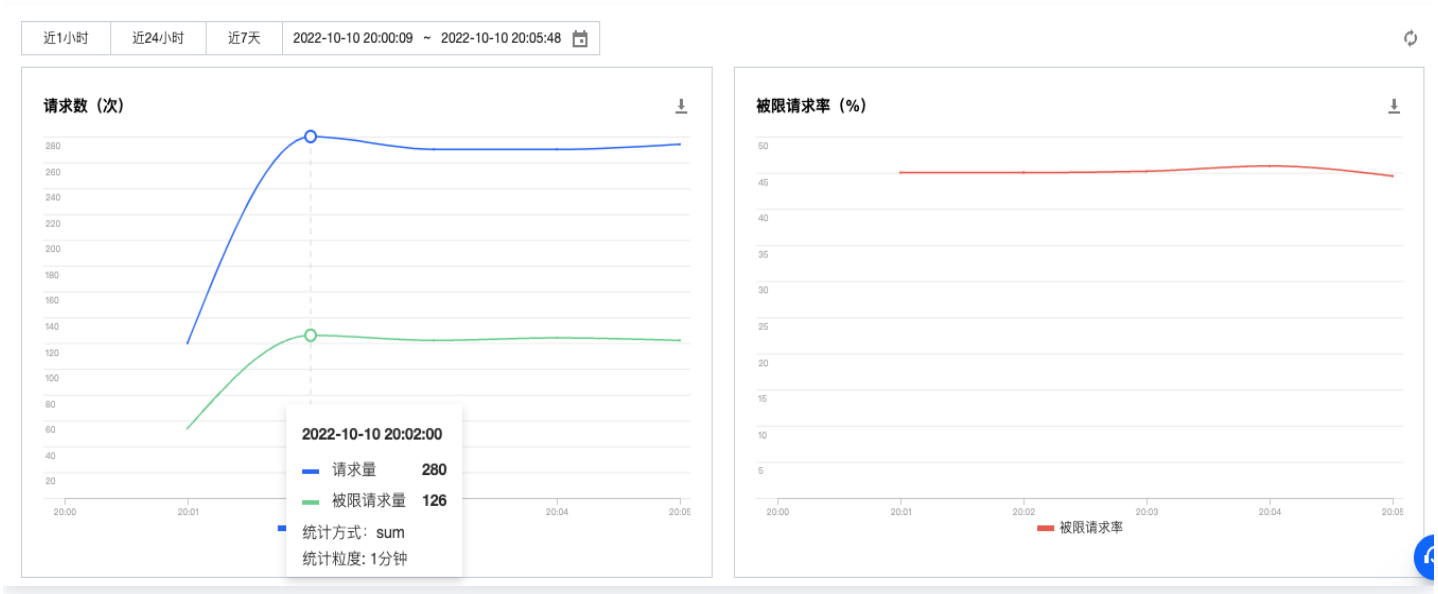


步骤4：触发限流

假设服务提供了 `/echo` API，可以通过不断执行 `curl /echo` 来模拟限流场景。
示例：在 官网 Demo 中 consumer-demo 服务提供了 `/echo-feign/{str}` API，那么针对 consumer-demo 服务新建限流规则，限流粒度为全局限流，单位时间 2 秒，请求数 5 次。启用限流规则。
下载脚本 tsf-ratelimit.sh ，登录可以访问到 consumer-demo 的机器（consumer-demo 所在云服务器也可以），执行 `./tsf-ratelimit.sh <IP>:<Port>` ，其中 IP 是 consumer-demo 所在云服务器 IP， Port 为服务监听端口 18083。脚本的作用是**每 2 秒触发 10 次**调用。由于调用的频率大于限流规则，正常情况下，会收到 HTTP 429 (Too Many Requests) 的状态码。

步骤5：查看限流效果

如果请求数达到了限流阈值，任何到达的请求都会限流模块处理。如果该服务上的配额已经消耗完，会对请求返回 HTTP 429（Too Many Requests）；否则会正常放行。用户可以在限流规则列表下方的**请求数-时间**图中查看到被限制的请求数或者**被限制请求率-时间**图中查看到被限制请求率（计算公式 `被限制请求率 = 被限制的请求数 / 请求数` ）随时间的变化。



限制说明

等于、不等于、包含、不包含属于严格匹配，正则表达式属于模糊匹配。因此当系统标签是被调方 API PATH 时，目前仅支持使用**正则表达式**的逻辑关系来匹配带参数的 API 请求。

- 当标签的逻辑关系是正则表达式，值填写 `/echo/.*` 时，可以匹配带参数的请求 /echo/test123（其中test123是参数）。
- 当标签的逻辑关系是等于、不等于、包含、不包含关系，值是 `/echo/{param}` 时，不能匹配带参数的请求 /echo/test123（其中 test123 是参数）。

服务熔断

最近更新时间：2025-04-24 10:32:45

> **说明：**
> 当前 TSF-consul 已经转为内测，如果您有服务注册配置治理的需求，请您购买 Polaris（北极星）。

场景说明

TSF 服务治理支持可视化熔断规则管理，支持设置服务、实例、API 三种隔离级别的熔断规则。

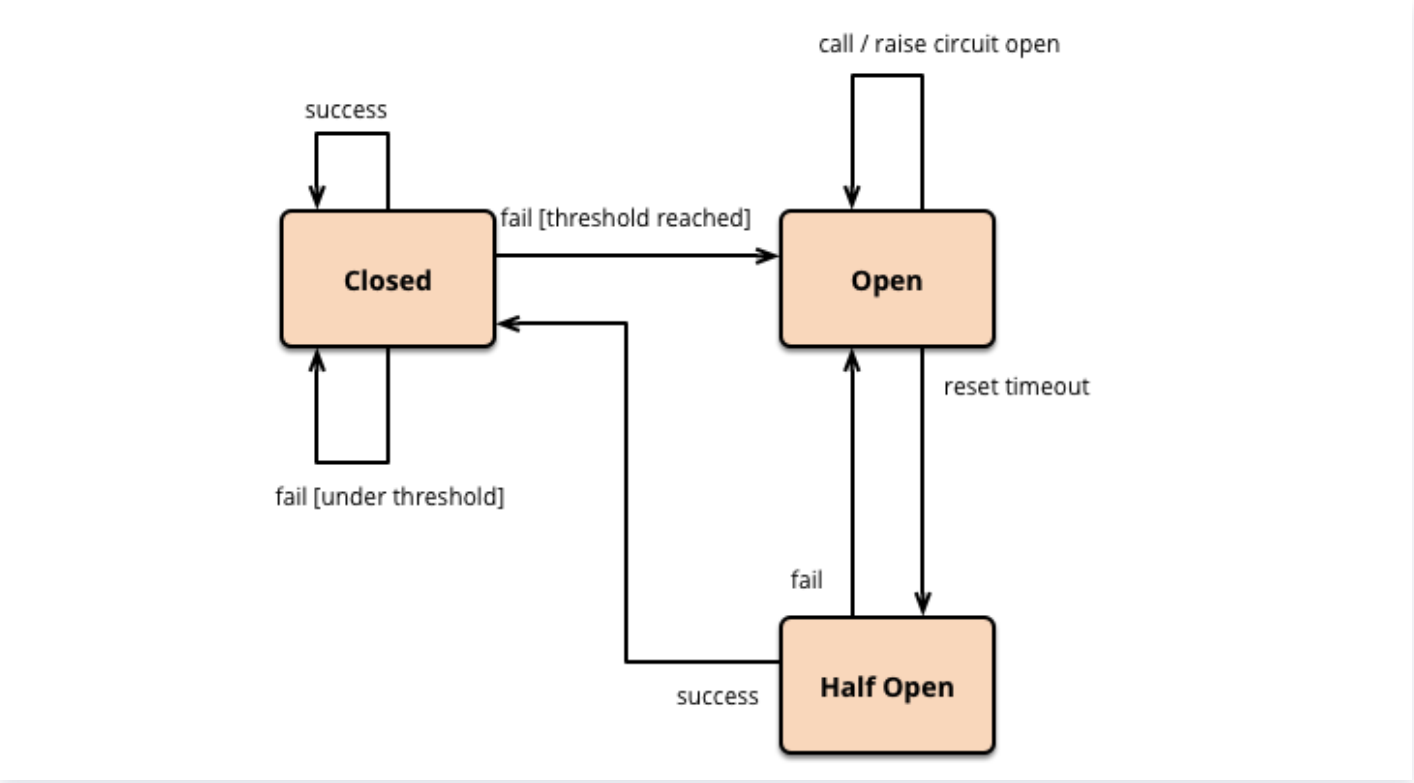
熔断原理

定义

服务熔断定义：当下游的服务因为某种原因导致服务不可用或响应过慢时，上游服务为了保证自己整体服务的可用性，不再继续调用目标服务，直接返回。当下游服务恢复后，上游服务会恢复调用。

熔断器状态

服务熔断中涉及到关键概念**熔断器**，熔断器的状态转化如下：



1. 最开始处于 `closed` 状态，一旦检测到错误（或慢响应）达到一定阈值，便转为 `open` 状态，此时不再调用下游目标服务。
2. 等待一段时间后，会转化为 `half open` 状态，尝试放行一部分请求到下游服务。
3. 一旦检测到响应成功，回归到 `closed` 状态，也即恢复服务；否则回到 `open` 状态。

其中熔断器从 `close` 变为 `open` 状态要**同时**满足以下2个条件：

- 前提条件：在滑动时间窗口内至少有一定数量的请求（即**最少请求数**）
- 指标达到阈值：在滑动时间窗口内统计的错误请求率或慢请求率达到一定阈值

隔离级别及场景

TSF 支持服务、实例、API 三种隔离级别的熔断规则：

隔离级别	请求统计范围	熔断对象	适用场景
服务	下游目标服务的所有实例的所有 API	服务	当下游服务属于不主要的业务，可以熔断所有实例
实例	下游目标服务的单个实例的所有 API	达到熔断触发条件的实例	当下游服务属于比较重要的业务，只对异常的实例进行熔断，避免所有实例被熔断后导致服务雪崩
API	下游目标服务的所有实例的指定 API	达到熔断触发条件的单个 API	下游服务不同 API 的重要程度不同，需要根据不同 API 设置不同的熔断策略

使用方法

不同于服务限流、路由和鉴权规则在被调服务上设置，服务熔断规则是在**主调方服务**上设置。

开发指南

目前 TSF 支持 Spring Cloud 应用及 TSF Mesh 应用两种微服务框架的服务熔断。
Spring Cloud 熔断开发指南参见 开发文档，注意 Spring Cloud 应用的服务熔断功能需要使用 1.19.0 版本及以上的 SDK，参见 SDK 版本更新日志。
TSF Mesh 仅需完成控制台熔断规则配置并启用即可实现 TSF Mesh 服务熔断能力，无侵入操作简单。TSF Mesh 迁移开发指引，参见 TSF Mesh 指南。

控制台基本操作

假设用户希望在 `consumer-demo` 上针对下游服务 `provider-demo` 设置一个熔断策略。

新建并启用熔断规则

> **注意**
> 一个服务的不同熔断规则的下游目标服务不能重复。

1. 登录 TSF 控制台，在左侧导航栏单击 **TSF Consul** 后，选择**服务治理**，单击 `consumer-demo` 服务进入服务详情页。
2. 切换至**服务熔断**标签页，单击**新建熔断规则**，在创建熔断规则对话框中填写熔断规则：
   - **下游服务**： 选择当前 `provider-demo` 所在命名空间和服务名。
   - **隔离级别**：根据业务场景需求设置隔离级别

- 服务：选择服务隔离级别后，设置熔断策略各参数。
- 实例：选择实例隔离级别后，设置熔断策略各参数和最大熔断实例比率。
- API：选择 API 隔离级别后，可选择不同的 API 设置熔断策略，熔断策略中各参数适用于选中的每个 API。

- **滑动时间窗口**：用于统计熔断器关闭时的请求结果。
- **最少请求次数**：配置熔断器可以计算错误率之前的最小请求数。
- **触发条件**（满足以下**任一条件**触发熔断）：
  - 失败请求率：在滑动时间窗口内统计的失败请求占所有请求比率（失败请求是指响应状态码为4XX和5XX，以及抛出异常的请求）。
  - 慢请求率：在滑动时间窗口内统计的慢响应的请求占所有请求比率，其中**慢响应**的时长支持配置。
- **开启到半开间隔**：熔断器从 `open` 状态等待一段时间后变为 `half-open` 状态，尝试放行一部分请求到下游服务。

3. 单击**完成**，跳转至熔断规则列表。
4. 在熔断规则列表上，单击熔断规则的**启用**，启用该规则。

系统和业务自定义标签

最近更新时间：2025-04-24 10:32:45

标签说明

TSF 引入**标签**概念以区分不同的请求来源，TSF 标签包括系统标签和业务自定义标签。

- **系统标签**
  每一个 TSF 上运行的服务都已经被预先设置好了某些标签，如发起请求的服务消费方所在的部署组、IP、服务发起方的版本号等。当前支持的系统标签有：
  - 上游服务名
  - 命名空间+上游服务名：该标签仅支持1.18.0（包含1.18.0）版本之后的 SDK，针对全局命名空间中的微服务，或微服务被全局命名空间中的网关访问的场景下生效。
  - 上游应用
  - 上游应用版本号
  - 上游部署组
  - 上游IP
  - 当前应用版本号
  - 当前部署组
  - 当前服务 API PATH
  - 请求 HTTP METHOD
- **业务自定义标签**
  在实际的使用中，如果系统自带标签不能保证用户使用的场景，用户可以自定义标签内容。对于 Spring Cloud 应用，TSF 提供了用户配置自定义标签的 SDK，参见开发手册 参数传递；对于 Mesh 应用，用户需要在 header 中设置标签，参见 Mesh 开发使用指引-设置自定义标签 。

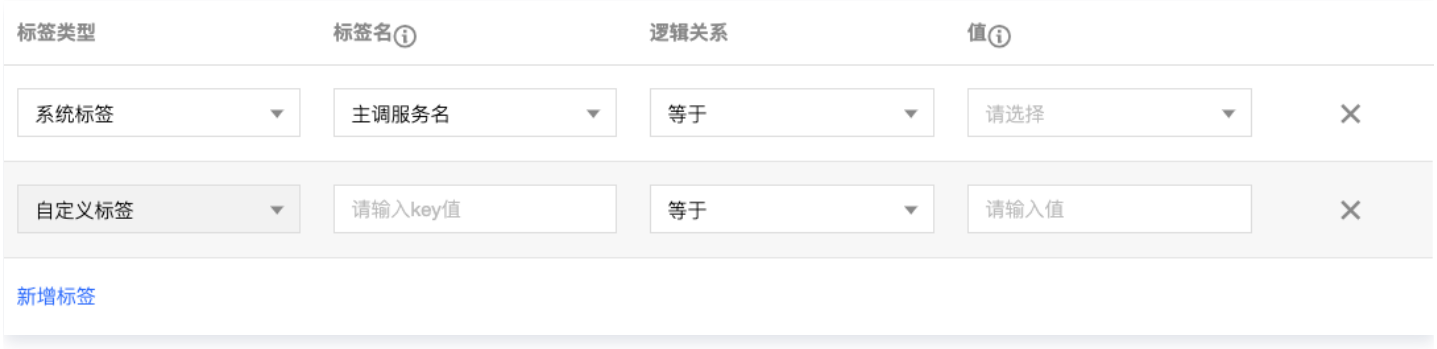
> **注意**
> 这里的标签和腾讯云的标签产品不是同一个概念。腾讯云的标签产品是一种划分资源的方式，而 TSF 服务治理中的标签是为了区分不同的请求来源。

标签表达式

用户在控制台创建服务治理规则时，可以选择通过设置**标签表达式**区分请求来源。多个标签表达式之间是**逻辑与（AND）**的关系。例如两条标签表达式分别是：

- 系统标签主调服务名等于 consumer-demo
- 自定义标签 userid 等于123456

只有当一条请求是 consumer-demo 发出，且带有 userid 是123456的自定义标签时才满足上面2个标签表达式。



一条标签表达式中，逻辑关系与值的个数对应如下：

逻辑关系	值个数
包含（IN）	多个
不包含（ NOT IN）	多个
等于（==）	一个
不等于（!=）	一个
正则表达式（regex）	一个