



21天微服务实战营-Day12

华为云DevCloud & ServiceStage服务联合出品



Day12 CSE实战之CSEGoSDK场景实战

大纲

- ❑ 熔断
- ❑ 限流
- ❑ 容错
- ❑ 灰度

熔断

熔断：该功能在服务运行时，能很好的隔离上游服务。在熔断的保护下，如果请求错误过多，服务将停止对该服务访问。

熔断支持：

- 全局配置。该配置对所有的服务的所有请求都起作用，只要服务任何一个接口达到熔断条件将导致该服务所有接口都被熔断隔离。
- 服务级别配置。此时熔断只对所配置的服务器作用，没有配置的服务将使用默认配置或全局。此时与全局类似，某个服务的一个接口导致熔断也会影响其他接口不同正常使用。
- API级别配置。API级别的配置仅对API生效，即使同一个服务API之间互相不影响。如/hello接口发生错误导致熔断，/hi接口依旧可以正常使用

熔断

配置示例

全局配置示例：

```
---
cse:
  isolation:
    Consumer:
      timeoutInMilliseconds: 10
      maxConcurrentRequests: 1000
  circuitBreaker:
    Consumer:
      enabled: true
      forceOpen: false
      forceClosed: false
      sleepWindowInMilliseconds: 10000
      requestVolumeThreshold: 10
      errorThresholdPercentage: 50
```

1，表示是否开启熔断功能

2，表示是否强制开启熔断，开启后，将无法正常进行访问，请求被直接隔离

3，表示是否强制关闭熔断，强制关闭后，无论发生什么错误都不会产生熔断

4，表示熔断后多少毫秒后重新尝试访问服务，如果失败继续熔断，如果成功则恢复原来的访问

5，表示多少请求后计算错误请求占比，此时需要1为true以及2,3为false才生效

6，表示请求错误数占比达到该值后，开启熔断。同样需要1为true以及2,3为false

熔断

服务级别配置示例：

```
---
cse:
  isolation:
    Consumer:
      timeoutInMilliseconds: 10
      maxConcurrentRequests: 1000
  circuitBreaker:
    Consumer:
      enabled: true
      forceOpen: false
      forceClosed: false
      sleepWindowInMilliseconds: 10000
      requestVolumeThreshold: 10
      errorThresholdPercentage: 50
      server_name:
        enabled: false
        forceOpen: false
        forceClosed: false
        sleepWindowInMilliseconds: 10000
        requestVolumeThreshold: 20
        errorThresholdPercentage: 50
```

server_name 为服务名，请根据实际配置的服务名进行填写

限流

➤ 介绍

用户可以通过配置限流策略限制Provider端或Consumer端的请求频率，使每秒请求数限制在最大请求量的大小。其中Provider端的配置可限制接收处理请求的频率，Consumer端的配置可限制发往指定微服务的请求的频率。使用是，需要添加以下 handler ： provider添加ratelimiter-provider，consumer添加ratelimiter-consumer。

➤ 配置

限流配置在rate_limiting.yaml中，同时需要在chassis.yaml的handler chain中添加ratelimiter-provider。其中qps.limit.[service] 是指限制从service 发来的请求的处理频率，若该项未配置则global.limit生效。Consumer端不支持global全局配置，其他配置项与Provider端一致， handler chain添加ratelimiter-consumer。

限流

➤ 配置演示

客户端配置，限流设置为10
那么访问任何服务时流量都是每
秒10次

consumer端：

```
---
cse:
  flowcontrol:
    Consumer:
      qps:
        enabled: true # enable rate limiting or not
        limit:
          server_name: 10 # rate limit for request to a provider
```

```
handler:
  chain:
    Consumer:
      default: bizkeeper-consumer, router, loadbalance, ratelimiter-consumer, transport
#ssl:
```

provider端

全局配置，限
流设置为100

```
---
cse:
  flowcontrol:
    Provider:
      qps:
        enabled: true # enable rate limiting or not
        global:
          limit: 100 # default limit of provider
        limit:
          sever_name: 1 # rate limit for request from a provider
```

```
20 handler:
21   chain:
22     Provider:
23       incoming: ratelimiter-provider
```

容错

在CSEGoSDK提供自动重试的容错能力，用户可配置retry及backOff策略自动启用重试功能

➤ 类型

默认情况下CSEGoSDK支持三种容错类型

- ❑ zero：固定重试时间为0的重试策略，即失败后立即重试不等待
- ❑ constant：固定时间为backoff.minMs的重试策略，即失败后等待backoff.minMs再重试。
- ❑ jittered：按指数增加重试时间的重试策略，初始重试时间为backoff.minMs，最大重试时间为backoff.MaxMs。推荐此方法

容错

配置实例

```
cse:
  loadbalance:
    retryEnabled: true
    retryOnNext: 0
    retryOnSame: 1
  backoff:
    kind: jittered # 重试策略: [jittered或constant或zero]。
    MinMs: 200
    MaxMs: 400
```

是否使用容错

请求失败后向其他实例重试的次数。

请求失败后向同一个实例重试的次数。

重试策略: [jittered或constant或zero]。

重试最小时间间隔, 单位ms。

重试最大时间间隔, 单位ms。

灰度发布

应用于AB测试场景和新版本的灰度升级等相关场景。CSEGoSDK通过对路由的管理实现以上场景，以下重点讲述CSEGoSDK如何通过router的管理实现灰度发布。

□ 路由管理

路由策略可应用于AB测试场景和新版本的灰度升级，主要通过路由规则来根据请求的来源、目标服务、Http Header及权重将服务访问请求分发到不同版本的微服务实例中。

灰度

- 配置路由管理：路由管理使用router.yaml 进行配置。同时也支持API方式进行设置，本文并不介绍，推荐使用router.yaml进行配置。
- 路由规则说明
 - 匹配特定请求由routeRule.{targetServiceName}.match配置，匹配条件是：source（源服务名）、source tags及headers，另外也可以使用refer字段来使用source模板进行匹配。
 - Match中的Source Tags用于和服务调用请求中的。sourceInfo中的tags进行逐一匹配。
 - Header中的字段的匹配支持正则、=、!=、>、<、>=、<=七种匹配方式。
 - 如果未定义match，则可匹配任何请求。
 - 转发权重定义在routeRule.{targetServiceName}.route下，由weight配置。
 - 服务分组定义在routeRule.{targetServiceName}.route下，由tags配置，配置内容有version和app。

灰度-配置示例

➤ 示例一：

每个路由规则的目标服务名称都由routeRule中的Key值指定。例如下表所示，所有以 “Carts”服务为目标服务的路由规则均被包含在以 “Carts”为Key值的列表中。

```
routeRule:
  Carts:
    - precedence: 1
      route:
        - weight: 100 #percent
          tags:
            version: 0.0.1
```

Key值（目标服务名称）应该满足是一个合法的域名称。例如，一个在服务中心中注册的服务名称。

灰度-配置示例

➤ 示例二：

每个在一个服务拥有多个不同版本的实例时，我们可以通过precedence进行设置优先级，默认为 0，该配置值越大，优先权越高。

```
routeRule:
  Carts:
    - precedence: 2
      match:
        headers:
          Foo:
            exact: bar
      route:
        - weight: 100
          tags:
            version: 2.0
    - precedence: 1
      route:
        - weight: 100
          tags:
            version: 1.0
```

图一

图一：在match到bar的请求，会优先将请求分流之version为2.0的实例

Demo演示

➤ Demo访问规则：

<http://127.0.0.1:8080/consumer/v0/delay?t=1s&delay=20&c=5>

t 表示对provider发起访问的时间

delay表示服务延时的时长，单位ms

c 表示开启的goroutine数

上面的URL表示开启5个goroutine进行访问1s，每次请求延时20ms

Demo演示

➤ 熔断：熔断配置

```
---
cse:
  isolation:
    Consumer:
      timeoutInMilliseconds: 1
      maxConcurrentRequests: 1000
  circuitBreaker:
    # scope: api
    Consumer:
      enabled: true
      forceOpen: false
      forceClosed: false
      sleepWindowInMilliseconds: 10000
      requestVolumeThreshold: 10 # 请求次数达到10次后, 若失败率高于50%服务开始熔断
      errorThresholdPercentage: 50
```

配置说明：该配置中，请求达到十次后将会开始计算错误率，如果错误率达到50%将会开启熔断。如下图，服务在连续访问十次都失败后，第十一开始熔断

访问URL: <http://127.0.0.1:8080/consumer/v0/delay?t=1s&delay=10&c=5>

```
36      "Error": "Get http://127.0.0.1:9092/provider/v0/delay/11: net/http: request canceled (Client.Timeout exceeded while awaiting headers)"
37    },
38    {
39      "Reply": "",
40      "Error": "Get http://127.0.0.1:9092/provider/v0/delay/11: net/http: request canceled (Client.Timeout exceeded while awaiting headers)"
41    },
42    {
43      "Reply": "",
44      "Error": "API provider:rest:/provider/v0/delay/11 is isolated because of error: circuit open"
45    }
46  ]
```

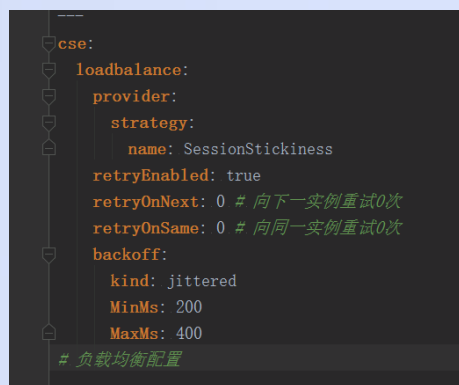
第十一次
出现熔断

第十次错误方
向

Demo演示

容错：配置设置为0，并且配置会话保持策略，然后像服务发起数次访问，请求。效果如下右下图。访问URL如下：

<http://127.0.0.1:8080/consumer/v0/delay?t=1s&delay=10&c=1>



有图中我们可以看出失败后并没有继续访问统一实例，而是访问下一个实例

```
2019-03-21 16:30:13.056 +08:00 INFO schema/consumer.go:69 launched one http benchmark thread
2019-03-21 16:30:13.058 +08:00 ERROR handler/transport_handler.go:48 Call got Error, err [Get http://127.0.0.1:9092/provider/v0/delay/1: net/http: request canceled (Client.Timeout exceeded while awaiting headers)]
2019-03-21 16:30:13.060 +08:00 ERROR handler/transport_handler.go:48 Call got Error, err [Get http://127.0.0.1:9092/provider/v0/delay/1: net/http: request canceled (Client.Timeout exceeded while awaiting headers)]
2019-03-21 16:30:13.062 +08:00 ERROR handler/transport_handler.go:48 Call got Error, err [Get http://127.0.0.1:9091/provider/v0/delay/1: net/http: request canceled (Client.Timeout exceeded while awaiting headers)]
2019-03-21 16:30:13.064 +08:00 ERROR handler/transport_handler.go:48 Call got Error, err [Get http://127.0.0.1:9091/provider/v0/delay/1: net/http: request canceled (Client.Timeout exceeded while awaiting headers)]
2019-03-21 16:30:13.066 +08:00 ERROR handler/transport_handler.go:48 Call got Error, err [Get http://127.0.0.1:9092/provider/v0/delay/1: net/http: request canceled (Client.Timeout exceeded while awaiting headers)]
2019-03-21 16:30:13.068 +08:00 ERROR handler/transport_handler.go:48 Call got Error, err [Get http://127.0.0.1:9092/provider/v0/delay/1: net/http: request canceled (Client.Timeout exceeded while awaiting headers)]
2019-03-21 16:30:13.070 +08:00 ERROR handler/transport_handler.go:48 Call got Error, err [Get http://127.0.0.1:9091/provider/v0/delay/1: net/http: request canceled (Client.Timeout exceeded while awaiting headers)]
2019-03-21 16:30:13.072 +08:00 ERROR handler/transport_handler.go:48 Call got Error, err [Get http://127.0.0.1:9091/provider/v0/delay/1: net/http: request canceled (Client.Timeout exceeded while awaiting headers)]
2019-03-21 16:30:13.074 +08:00 ERROR handler/transport_handler.go:48 Call got Error, err [Get http://127.0.0.1:9092/provider/v0/delay/1: net/http: request canceled (Client.Timeout exceeded while awaiting headers)]
2019-03-21 16:30:13.076 +08:00 ERROR handler/transport_handler.go:48 Call got Error, err [Get http://127.0.0.1:9092/provider/v0/delay/1: net/http: request canceled (Client.Timeout exceeded while awaiting headers)]
2019-03-21 16:30:13.078 +08:00 ERROR handler/transport_handler.go:48 Call got Error, err [Get http://127.0.0.1:9091/provider/v0/delay/1: net/http: request canceled (Client.Timeout exceeded while awaiting headers)]
2019-03-21 16:30:13.080 +08:00 ERROR handler/transport_handler.go:48 Call got Error, err [Get http://127.0.0.1:9091/provider/v0/delay/1: net/http: request canceled (Client.Timeout exceeded while awaiting headers)]
2019-03-21 16:30:13.082 +08:00 ERROR handler/transport_handler.go:48 Call got Error, err [Get http://127.0.0.1:9092/provider/v0/delay/1: net/http: request canceled (Client.Timeout exceeded while awaiting headers)]
2019-03-21 16:30:13.084 +08:00 ERROR handler/transport_handler.go:48 Call got Error, err [Get http://127.0.0.1:9092/provider/v0/delay/1: net/http: request canceled (Client.Timeout exceeded while awaiting headers)]
2019-03-21 16:30:13.086 +08:00 ERROR handler/transport_handler.go:48 Call got Error, err [Get http://127.0.0.1:9091/provider/v0/delay/1: net/http: request canceled (Client.Timeout exceeded while awaiting headers)]
2019-03-21 16:30:13.088 +08:00 ERROR handler/transport_handler.go:48 Call got Error, err [Get http://127.0.0.1:9091/provider/v0/delay/1: net/http: request canceled (Client.Timeout exceeded while awaiting headers)]
2019-03-21 16:30:13.090 +08:00 ERROR handler/transport_handler.go:48 Call got Error, err [Get http://127.0.0.1:9092/provider/v0/delay/1: net/http: request canceled (Client.Timeout exceeded while awaiting headers)]
2019-03-21 16:30:13.092 +08:00 ERROR handler/transport_handler.go:48 Call got Error, err [Get http://127.0.0.1:9092/provider/v0/delay/1: net/http: request canceled (Client.Timeout exceeded while awaiting headers)]
2019-03-21 16:30:13.093 +08:00 ERROR handler/transport_handler.go:48 Call got Error, err [Get http://127.0.0.1:9091/provider/v0/delay/1: net/http: request canceled (Client.Timeout exceeded while awaiting headers)]
```


Demo演示

限流：在任意一个provider端设置限流，限流为10，开启十个线程访问1s

```
1  ---
2  cse:
3    flowcontrol:
4      Provider:
5        qps:
6          enabled: true # enable rate limiting or not
7          global:
8            limit: 10
```

首次访问由于框架内有需要进行初始化操作，导致访问不准确，二次访问后，每次访问次数都固定为20.

Demo演示

➤ 灰度

完成目标：配置router.yaml文件。文件配置要求在header设施 key:XXX后，并且启动provider_v1和provider_v2。访问规则会优先访问高版本实例，配置后，需要达到请求100%请求version 1.0。配置如下：

```
1 routeRule:
2   provider:
3     - precedence: 2
4       match:
5         headers:
6           Foo:
7             exact: bar
8         route:
9           - weight: 100
10            tags:
11              version: 1.0
12     - precedence: 1
13       route:
14         - weight: 100
15         tags:
16           version: 2.0
```

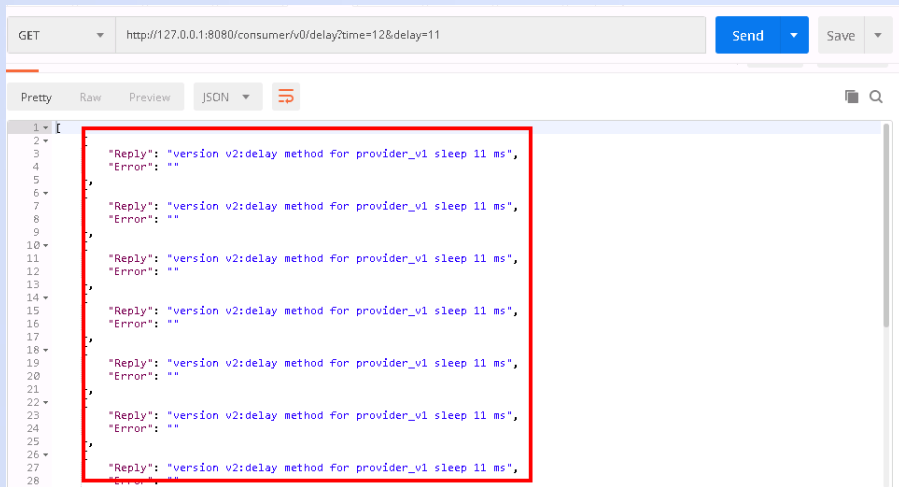
配置说明：左侧配置中我们在头部只要携带了 Foo:bar 该请求将全部访问 version 1.0的实例，头部信息设置如下图

```
req, err := rest.NewRequest(method, url, body: nil)
if err != nil {
    panic(err)
}
//req.Header.Set("Foo", "bar") ← 取消注释
resp, err := restInvoker.ContextDo(ctx, req)
if err != nil {
    // ...
}
```

Demo演示

➤ 灰度

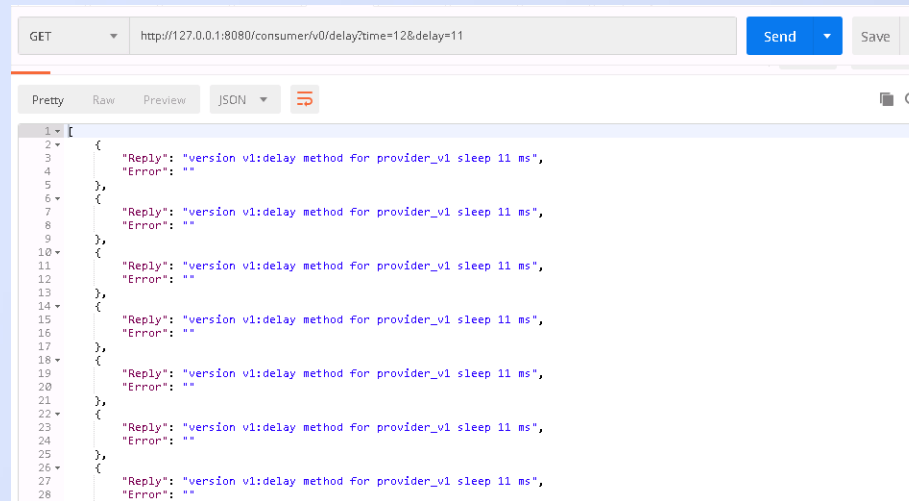
配置好所有的配置后，我们分别将配置注释和开启各访问一次，下方左侧图为没有使用router规则时访问，可以看到请求到达了version 2.0的实例。在使用router规则后，并且header包含Foo : bar时，请求将全部访问version 1.0。如下方右侧图



```
GET http://127.0.0.1:8080/consumer/v0/delay?time=12&delay=11
Send Save

Pretty Raw Preview JSON

1 [
2   {
3     "Reply": "version v2:delay method for provider_v1 sleep 11 ms",
4     "Error": ""
5   },
6   {
7     "Reply": "version v2:delay method for provider_v1 sleep 11 ms",
8     "Error": ""
9   },
10  {
11    "Reply": "version v2:delay method for provider_v1 sleep 11 ms",
12    "Error": ""
13  },
14  {
15    "Reply": "version v2:delay method for provider_v1 sleep 11 ms",
16    "Error": ""
17  },
18  {
19    "Reply": "version v2:delay method for provider_v1 sleep 11 ms",
20    "Error": ""
21  },
22  {
23    "Reply": "version v2:delay method for provider_v1 sleep 11 ms",
24    "Error": ""
25  },
26  {
27    "Reply": "version v2:delay method for provider_v1 sleep 11 ms",
28    "Error": ""
29  },
30  {
31    "Reply": "version v2:delay method for provider_v1 sleep 11 ms",
32    "Error": ""
33  },
34  {
35    "Reply": "version v2:delay method for provider_v1 sleep 11 ms",
36    "Error": ""
37  },
38  {
39    "Reply": "version v2:delay method for provider_v1 sleep 11 ms",
40    "Error": ""
41  }
42 ]
```



```
GET http://127.0.0.1:8080/consumer/v0/delay?time=12&delay=11
Send Save

Pretty Raw Preview JSON

1 [
2   {
3     "Reply": "version v1:delay method for provider_v1 sleep 11 ms",
4     "Error": ""
5   },
6   {
7     "Reply": "version v1:delay method for provider_v1 sleep 11 ms",
8     "Error": ""
9   },
10  {
11    "Reply": "version v1:delay method for provider_v1 sleep 11 ms",
12    "Error": ""
13  },
14  {
15    "Reply": "version v1:delay method for provider_v1 sleep 11 ms",
16    "Error": ""
17  },
18  {
19    "Reply": "version v1:delay method for provider_v1 sleep 11 ms",
20    "Error": ""
21  },
22  {
23    "Reply": "version v1:delay method for provider_v1 sleep 11 ms",
24    "Error": ""
25  },
26  {
27    "Reply": "version v1:delay method for provider_v1 sleep 11 ms",
28    "Error": ""
29  },
30  {
31    "Reply": "version v1:delay method for provider_v1 sleep 11 ms",
32    "Error": ""
33  },
34  {
35    "Reply": "version v1:delay method for provider_v1 sleep 11 ms",
36    "Error": ""
37  },
38  {
39    "Reply": "version v1:delay method for provider_v1 sleep 11 ms",
40    "Error": ""
41  }
42 ]
```

参考文献

配置参考

- ❑ [熔断配置](#)
- ❑ [限流配置](#)
- ❑ [容错配置](#)
- ❑ [灰度配置](#)

推荐文档:

- ❑ [微服务分布式系统熔断实战-为何我们需要API级别熔断？](#)
- ❑ [Go语言微服务开发框架实践-go chassis（上篇）](#)
- ❑ [Go语言微服务开发框架实践-go chassis（中篇）](#)
- ❑ [使用go chassis进行微服务路由管理](#)

Thank You

