

讲堂 > 从0开始学微服务 > 文章详情

# 22 | 如何管理服务配置？

2018-10-11 胡忠想



22 | 如何管理服务配置？  
朗读人：胡忠想 09'55" | 4.55M

在拆分为微服务架构前，曾经的单体应用只需要管理一套配置；而拆分为微服务后，每一个系统都有自己的配置，并且都各不相同，而且因为服务治理的需要，有些配置还需要能够动态改变，以达到动态降级、切流量、扩缩容等目的，这也是今天我要与你探讨的，在微服务架构下服务配置如何管理的问题。

## 本地配置

服务配置管理最简单的方案就是把配置当作代码同等看待，随着应用程序代码一起发布。比如下面这段代码用到了开源熔断框架 Hystrix，并且在代码里定义了几个配置，一个是线程的超时间时间是 3000ms，一个是熔断器触发的错误比率是 60%。

复制代码

```
1 @HystrixCommand(fallbackMethod = "getDefaultProductInventoryByCode",
2   commandProperties = {
3     @HystrixProperty(name = "execution.isolation.thread.timeoutInMilliseconds", value = "3000"),
4     @HystrixProperty(name = "circuitBreaker.errorThresholdPercentage", value="60")
5   }
6 }
```

```
6 )
7 public Optional<ProductInventoryResponse> getProductInventoryByCode(String productCode)
8 {
9     ....
10 }
```

还有一种方案就是把配置都抽离到单独的配置文件当中，使配置与代码分离，比如下面这段代码。

```
1 @HystrixCommand(commandKey = "inventory-by-productcode", fallbackMethod = "getDefaultProductInve
2 public Optional<ProductInventoryResponse> getProductInventoryByCode(String productCode)
3 {
4     ...
5 }
```

[复制代码](#)

相应的配置可以抽离到配置文件中，配置文件的内容如下：

```
1 hystrix.command.inventory-by-productcode.execution.isolation.thread.timeoutInMilliseconds=2000
2 hystrix.command.inventory-by-productcode.circuitBreaker.errorThresholdPercentage=60
```

[复制代码](#)

无论是把配置定义在代码里，还是把配置从代码中抽离出来，都相当于把配置存在了应用程序的本地。这样做的话，如果需要修改配置，就需要重新走一遍代码或者配置的发布流程，在实际的线上业务当中，这是一个很重的操作，往往相当于一次上线发布过程，甚至更繁琐，需要更谨慎。

这时你自然会想，如果能有一个集中管理配置的地方，如果需要修改配置，只需要在这个地方修改一下，线上服务就自动从这个地方同步过去，不需要走代码或者配置的发布流程，不就简单多了吗？没错，这就是下面要讲的配置中心。

## 配置中心

配置中心的思路就是把服务的各种配置，如代码里配置的各种参数、服务降级的开关甚至依赖的资源等都在一个地方统一进行管理。服务启动时，可以自动从配置中心中拉取所需的配置，并且如果有配置变更的情况，同样可以自动从配置中心拉取最新的配置信息，服务无须重新发布。

具体来讲，配置中心一般包含下面几个功能：

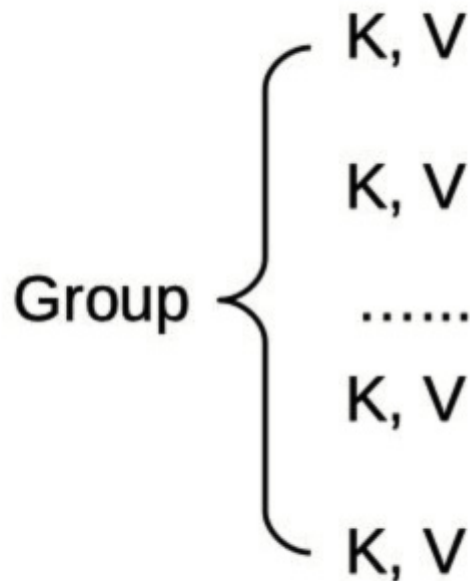
- 配置注册功能
- 配置反注册功能

- 配置查看功能
- 配置变更订阅功能

接下来我来给你详细讲解下配置中心的功能是如何实现的。

### 1. 配置存储结构

如下图所示，一般来讲，配置中心存储配置是按照 Group 来存储的，同一类配置放在一个 Group 下，以 K, V 键值对存储。



### 2. 配置注册

配置中心对外提供接口 `/config/service?action=register` 来完成配置注册功能，需要传递的参数包括配置对应的分组 Group，以及对应的 Key、Value 值。比如调用下面接口请求就会向配置项 `global.property` 中添加 Key 为 `reload.locations`、Value 为 `/data1/confs/system/reload.properties` 的配置。

```
1 curl "http://ip:port/config/service?action=register" -d "group=global.property&key=reload.locations" 📄 复制代码
```

### 3. 配置反注册


配置中心对外提供接口 `config/service?action=unregister` 来完成配置反注册功能，需要传递的参数包括配置对象的分组 Group，以及对应的 Key。比如调用下面的接口请求就会从配置项

global.property 中把 Key 为 reload.locations 的配置删除。

```
1 curl "http://ip:port/config/service?action=unregister"-d "group=global.property&key=reload.locations"  复制代码
```


#### 4. 配置查看

配置中心对外提供接口 config/service?action=lookup 来完成配置查看功能，需要传递的参数包括配置对象的分组 Group，以及对应的 Key。比如调用下面的接口请求就会返回配置项 global.property 中 Key 为 reload.locations 的配置值。

```
1 curl "http://ip:port/config/service?action=lookup&group=global.property&key=reload.locations"  复制代码
```

#### 5. 配置变更订阅

配置中心对外提供接口 config/service?action=getSign 来完成配置变更订阅接口，客户端本地会保存一个配置对象的分组 Group 的 sign 值，同时每隔一段时间去配置中心拉取该 Group 的 sign 值，与本地保存的 sign 值做对比。一旦配置中心中的 sign 值与本地的 sign 值不同，客户端就会从配置中心拉取最新的配置信息。比如调用下面的接口请求就会返回配置项 global.property 中 Key 为 reload.locations 的配置值。

```
1 curl "http://ip:port/config/service?action=getSign&group=global.property"  复制代码
```

讲到这里，你应该对配置中心的作用有所了解了，它可以便于我们管理服务的配置信息，并且如果要修改配置信息的话，只需要同配置中心交互就可以了，应用程序会通过订阅配置中心的配置，自动完成配置更新。那么实际业务中，有哪些场景应用配置中心比较合适呢？下面我就结合自己的经验，列举几个配置中心的典型应用场景，希望能给你一些启发。

- 资源服务化。对于大部分互联网业务来说，在应用规模不大的时候，所依赖的资源如 Memcached 缓存或者 MCQ 消息队列的数量也不多，因此对应的资源的 IP 可以直接写在配置里。但是当业务规模发展到一定程度后，所依赖的这些资源的数量也开始急剧膨胀。以微博的业务为例，核心缓存 Memcached 就有上千台机器，经常会遇到个别机器因为硬件故障而不可用，这个时候如果采用的是本地配置的话，就需要去更改本地配置，把不可用的 IP 改成可用的 IP，然后发布新的配置，这样的过程十分不便。但如果采用资源服务化的话，把对应的缓存统统归结为一类配置，然后如果有个别机器不可用的话，只需要在配置中心把对应的 IP 换成可用的 IP 即可，应用程序会自动同步到本机，也无须发布。
- 业务动态降级。微服务架构下，拆分的服务越多，出现故障的概率就越大，因此需要有对应的服务治理手段，比如要具备动态降级能力，在依赖的服务出现故障的情况下，可以快速降

级对这个服务的调用，从而保证不受影响。为此，服务消费者可以通过订阅依赖服务是否降级的配置，当依赖服务出现故障的时候，通过向配置中心下达指令，修改服务的配置为降级状态，这样服务消费者就可以订阅到配置的变更，从而降级对该服务的调用。

- 分组流量切换。前面我提到过，为了保证异地多活以及本地机房调用，一般服务提供者的部署会按照 IDC 维度进行部署，每个 IDC 划分为一个分组，这样的话，如果一个 IDC 出现故障，可以把故障 IDC 机房的调用切换到其他正常 IDC。为此，服务消费者可以通过订阅依赖服务的分组配置，当依赖服务的分组配置发生变更时，服务消费者就对应的把调用切换到新的分组，从而实现分组流量切换。

## 开源配置中心与选型

讲到这里，你可以根据我前面对配置中心的讲解自己去实现一个配置中心，但其实对于大部分中小团队来说，目前业界已经开源的配置中心实现可以说功能已经十分完善了，并且经过很多公司实际线上业务的充分论证，能满足大多数业务的需求，所以我建议是尽量选择成熟的开源配置中心实现，那么有哪些开源的配置中心可以使用呢？下面我就简单介绍下三个典型的开源实现：

- [Spring Cloud Config](#)。Spring Cloud 中使用的配置中心组件，只支持 Java 语言，配置存储在 git 中，变更配置也需要通过 git 操作，如果配置中心有配置变更，需要手动刷新。
- [Disconf](#)。百度开源的分布式配置管理平台，只支持 Java 语言，基于 Zookeeper 来实现配置变更实时推送给订阅的客户端，并且可以通过统一的管理界面来修改配置中心的配置。
- [Apollo](#)。携程开源的分布式配置中心，支持 Java 和 .Net 语言，客户端和配置中心通过 HTTP 长连接实现实时推送，并且有统一的管理界面来实现配置管理。

在实际选择的时候，Spring Cloud Config 作为配置中心的功能比较弱，只能通过 git 命令操作，而且变更配置的话还需要手动刷新，如果不是采用 Spring Cloud 框架的话不建议选择。而 Disconf 和 Apollo 的功能都比较强大，在国内许多互联网公司内部都有大量应用，其中 Apollo 对 Spring Boot 的支持比较好，如果应用本身采用的是 Spring Boot 开发的话，集成 Apollo 会更容易一些。

## 总结

今天我给你讲解了微服务架构下如何使用配置中心对服务的配置进行管理，以及实际业务中可能用到的场景，最后给出了一些开源配置中心的解决方案。关于业务中是否需要用到配置中心，以及选择哪种配置中心，要根据实际情况而定，如果业务比较简单，配置比较少并且不经常变更的话，采用本地配置是最简单的方案，这样的话不需要额外引入配置中心组件；相反，如果业务比较复杂，配置多而且有动态修改配置的需求的话，强烈建议引入配置中心来进行管理，而且最好做到配置变更实时推送给客户端，并且可以通过统一的管理界面来管理配置，这样的话能极大地降低运维的复杂度，减少人为介入，从而提高效率。



## 思考题

在前面我讲到 Zookeeper、Consul、etcd 作为服务的注册中心时，可以提供强一致性的服务发现功能，那么它们能够作为配置中心吗？为什么？

欢迎你在留言区写下自己的思考，与我一起讨论。



版权归极客邦科技所有，未经许可不得转载

写留言

通过留言可与作者互动