consul配置中心

1 注册中心与配置中心的区别

1.1 配置中心

什么是配置中心?

配置中心管理的是系统启动需要的参数和依赖的各类外部信息，比如`laravel`里的配置项，一般情况都是。用`key-value`来存取，也可以做成是一个有层次结构的key-value。配置中心一般用来控制系统的启动，运行时的部分参数的调整，比如进程数、缓存类型等。这些数据的特点是，一般情况下变化不大，但是需要考虑不同环境(有时也叫组)的隔离性，同时需要考虑存在很多套不同的配置，不同环境的系统通过这样的一个配置中心来去拉取自己的参数。这一块基本上常见的apollo、zk、etcd都支持的很好。

配置中心的作用?

管理系统启动、运行期间需使用的配置信息，从而做到动态的配置管理。

怎么做配置中心?

我们采用Consul模板工具基于Consul KV数据集来渲染配置文件

1.3 什么是Consul模板

Consul模板命令启动后，它将读取一个到多个模板文件，并向Consul发出查询，加载它们所需的所有数据。通常，`consul-template`作为守护程序运行，一开始获取初始值，之后持续监控更新，在集群中发生任何相关更改时都会重新加载模板。更新过程完成后，模板还可以运行后置命令。例如，它可以在进行配置更改后将HUP信号发送到负载均衡器服务。
模板部署方式：根据使用情况，用户可能在少数几个主机上只有一个consul-template实例，或者也可能需要在每个主机上运行多个实例。每个Consul模板过程都可以管理多个不相关的文件，如果这些文件共享数据依赖项，则将根据需要对提取的内容进行重复数据删除，这样可以减少Consul服务器上可能共享的负载。

单次模板加载：用户也可以使用`-once`标志仅加载一次模板，这在测试或者由其他脚本触发时比较有用。

Conusl模板有啥用途?

1. `更新配置文件`： Consul模板工具可用于更新服务配置文件，一种常见的用法是管理负载均衡配置文件，这些文件需要在许多无法直接连接到Consul群集的计算机上定期动态更新；
2. `发现有关Consul群集和服务的数据`：可以收集有关Consul群集中服务的信息。例如，用户可以收集群集上运行的所有服务的列表，或者可以发现Redis所有服务的地址。注意，这个操作在生产环境有所限制。

1.3.1 安装Consul模板

在开发者模式下使用本地Consul代理，先执行`consul agent -dev`，Consul代理正常运行后才能进行以下其他步骤。 Consul模板工具本身不包含在Consul二进制文件中，需要单独安装。可以直接安装预编译的二进制文件，也可以下载源代码自行编译，我们将安装预编译的二进制文件。

首先，先下载`consul-template`二进制文件:

```
$ curl -O https://releases.hashicorp.com/consul-template/0.26.0/consul-template_0.26.0_linux_386.zip
```

```
# 1. 解压缩
$ unzip  consul-template_0.26.0_linux_386.zip
#将执行文件（解压缩之后的文件夹），复制到/usr/local/bin文件夹下
cp consul-template /usr/local/bin
# 2. 设置环境变量，输入命令：
vim /etc/profile
# 3. 将这里的代码添加到 profile 文件末尾（这里覆盖之前consul的配置）
# Consul
export CONSUL_TEMPLATE_HOME=/usr/local/bin/consul-template
export PATH=$PATH:$CONSUL_HOME:$CONSUL_TEMPLATE_HOME;
# 4. 执行命令使环境变量生效
source /etc/profile
# 5. 验证软件是否安装成功
consul-template -v
```

1.3.2 Consul-template 实战配置文件生成

实战步骤：

1. Consul-template 定义模板文件：`find_address.tpl`

```
#创建模板
touch find_address.tpl
#定义模板内容项
{{ key "/hashicorp/street_address" }}
```

2. consul-template读取KV数据，用于渲染生成配置文件

```
#运行consul-template命令，同时指定要使用的模板和需要更新的文件：
consul-template -template "find_address.tpl:hashicorp_address.conf"
```

注：`consul-template`命令会持续运行，可使用`CTRL+C`可以停止其运行。新开一个新终端，使用命令行指令将数据写入Consul

3. 根据`consul kv`之类 用来往consul的KV键值对中存入数据数据

```
$ consul kv put hashicorp/street_address "127.0.0.1:8080"
Success! Data written to: hashicorp/street_address
#查看hashicorp_address.conf 文件来确保有数据写入文件
cat  /consul/config/tpl/hashicorp_address.conf
127.0.0.1:8080
```

4. 想要动态更新模板的值

再次 采用kv指令，`consul kv put hashicorp/street_address "127.0.0.1:11000"`更新`hashicorp/street_address`值，可以看到该文件立即更新。有啥用？ 比如用户可以用相同的过程来更新HAProxy负载均衡器配置。

1.3.3 Consul-template 服务配置生成

1. 定义模板文件：`all-services.tpl`

```
{{range services}} # {{.Name}}{{range service .Name}}
{{.Address}}:{{.Port}} max_fails=3 fail_timeout=60 weight=1; {{end}}

{{end}}
```

2. 运行`consul-template`命令指定我们刚刚创建的模板，并使用`-once`标志，仅运行一次

```
$ consul-template -template="all-services.tpl:all-services.conf" -once
```

Consul Template 操作指令手册：https://codechina.csdn.net/mirrors/hashicorp/consul-template?utm_source=csdn_github_accelerator

1.4 Consul-KV

什么是consul的KV数据?

Consul 提供了一个简单的键值存储(KV store)。它可以用于保存动态配置、辅助服务间协作、建立领导选举以及其他开发者想要实现的功能。可以把`consul`理解为简单的`Redis`，根据`KV`来进行键值对的数据存储。

为什么使用Consul的KV数据?

consul的kv功可以用来做动态配置。kv的增加修改删除查询除了可以通过consul客户端，还可以通过**http api**和**consul的ui**。

怎么操作?

CLI的客户端方式:

获取kv结果值，目前KV存储没任何值

```
$ consul kv get redis/config/minconns
Error! No key exists at: redis/config/minconns
#用 -detailed 标识可以获取 Consul 保存的详细元数据
$ consul kv get -detailed redis/config/minconns
CreateIndex      207
Flags            0
Key              redis/config/minconns
LockIndex        0
ModifyIndex      207
Session          -
Value            1
# -recurse 选项可以递归查询子键，结果按字典序排列
$ consul kv get -recurse
redis/config/maxconns:25
redis/config/minconns:1
redis/config/users/admin:abcd1234
```

添加/修改kv的结果值

```
$ consul kv get redis/config/minconns
Error! No key exists at: redis/config/minconns
$ consul kv put redis/config/minconns 1
Success! Data written to: redis/config/minconns

$ consul kv put redis/config/maxconns 25
Success! Data written to: redis/config/maxconns

$ consul kv put -flags=42 redis/config/users/admin abcd1234
Success! Data written to: redis/config/users/admin
```

用 `delete` 可以删除一个键值对

```
$ consul kv delete redis/config/minconns
Success! Deleted key: redis/config/minconns
# -recurse 选项递归删除子键
$ consul kv delete -recurse redis
Success! Deleted keys with prefix: redis
```

HTTP-API的方式操作KV存存储

使用`HTTP GET`，URL 如下，IP为运行consul agent的机器ip，port默认为8500，key注意如果有path的话需要带上。

```
格式：http://ip:port/v1/kv/key
http://192.168.30.128:8530/v1/kv/config/db/data

#config/db/data 为key的路径值，获取内容如下
[
  {
    "LockIndex": 0,
    "Key": "config/db/data",
    "Flags": 0,
    "Value": "ewoic2lnbiI6Indpbm5lciIsCiAgImRyaXZlciI6InJlZGlzIgp9",
    "CreateIndex": 21275,
    "ModifyIndex": 21286
  }
]
```

> 注：value是经过base64编码的，程序里使用http api的话需要对结果中的value进行解码。

新增/修改数据

使用`HTTP PUT`，`URL`如下，body中携带value

```
#格式：http://ip:port/v1/kv/key
http://192.168.30.128:8530/v1/kv/config/db/connection
#CURL:
curl -X PUT  -d '{"ip":"127.0.0.1","port":"3306" }'
http://192.168.30.128:8530/v1/kv/config/db/connection
```

删除数据

使用`HTTP DELETE`，url如下

```
格式: http://ip:port/v1/kv/key
http://192.168.30.128:8530/v1/kv/config/db/connection
# CURL:
curl -X DELETE http://192.168.30.128:8530/v1/kv/config/db/connection
```

1.5 元数据中心

元数据中心是大家共享的数据模型实例，这些信息是系统运行起来后，能准确给予业务能力的一个关键信息抽象。比如在dubbo之类的框架，就是如何能定义和描述一个具体的业务服务。在MySQL里面的`information_schema`库中的`tables`里，就是具体的一些db实例里的`metadata`信息。这些信息一般情况也是准静态的。像zk，etcd之类的也都可以。

```
SELECT *  FROM `information_schema`.`TABLES` WHERE
`information_schema`.`TABLES`.`TABLE_SCHEMA` = 'mysql' AND
`information_schema`.`TABLES`.`TABLE_NAME` = 'user';
```

1.6 Consul的API操作:

本地请求：基于`Agent`

```
# 请求例子：curl http://localhost:8500/v1/agnect/services
/v1/agent/checks                                --获取本地agent注册的所有检查(包括配置
文件和http注册)
/v1/agent/services                              --获取本地agent注册的所有服务
/v1/agent/members                               --获取集群中的成员
/v1/agent/self                                  --获取本地agent的配置和成员信息
/v1/agent/join/<address>                        --触发本地agent加入node
/vq/agent/force-leave/<node>                    --强制删除node
/v1/agent/check/register                        --在本地agent增加一个检查项，使用PUT
方法传输一个json格式的数据
/v1/agent/check/deregister/<checkID>            --注销一个本地agent的检查项
/v1/agent/check/pass/<checkID>                  --设置一个本地检查项的状态为passing
/v1/agent/check/warn/<checkID>                  --设置一个本地检查项的状态为warning
/v1/agent/check/fail<checkID>                   --设置一个本地检查项的状态为critical
/v1/agent/service/register                      --在本地agent增加一个新的服务项，使用
PUT方法传输一个json格式的数据
/v1/agent/service/deregister/<serviceID>        --注销一个本地agent的服务项
```

数据中心请求：基于`catalog`类型

```
/v1/catalog/register                        --注册一个新的service、node、check
/v1/catalog/deregister                      --注销一个service、node、check
/v1/catalog/datacenters                     --列出知道的数据中心
/v1/catalog/nodes                           --在给定的数据中心列出node
/v1/catalog/services                        --在给定的数据中心列出service
/v1/catalog/service/<service>               --查看某个服务的信息
/v1/catalog/node/<node>
```

注册服务：

```
HTTP-API请求：
URL: http://192.168.30.128:8530/v1/agent/service/register
```

```
method: PUT
DATA:
{
      "id": "test222",
      "name": "test222name",
      "tags": [ "xdp-/core.product" ],
      "address": "192.168.30.128",
      "port": 18306,
      "checks": [
        {
          "name": "core.product.check",
          "http": "http://192.168.30.128:18306",
          "interval": "10s",
          "timeout": "5s"
        }
      ]
}
```

查询服务：

```
#本地请求(Agent)
HTTP-API请求：查询本地节点下所有的注册服务
URL：http://192.168.30.128:8530/v1/agent/services
method: GET
HTTP-API请求：查询指定本地节点下的服务
语法：/v1/catalog/service/{name}
URL：http://192.168.30.128:8530/v1/agent/service/test1name
method: GET

--数据中心(catalog)的查询方式：
/v1/catalog/services                              --在给定的数据中心列出service
/v1/catalog/service/<service>                     --查看某个服务的信息
HTTP-API：查询当前数据中心下的注册的服务
URL：http://192.168.30.128:8510/v1/catalog/services
    http://192.168.30.128:8510/v1/catalog/service/test1name
method:GET
```

删除服务：

```
语法：/v1/agent/service/deregister/<serviceID>
URL：http://192.168.30.128:8530/v1/agent/service/deregister/test1
method: PUT
```

更新服务：

```
语法：/v1/agent/service/register
URL：http://192.168.30.128:8530/v1/agent/service/register
method: PUT
DATA: JSON格式数据
  {
      "id": "test1",
      "name": "test1name",
      "tags": [ "xdp-/core.product" ],
      "address": "192.168.30.128",
      "port": 18306,
      "checks": [
```

```
        {
          "name": "core.product.check",
          "http": "http://192.168.30.128:18306",
          "interval": "10s",
          "timeout": "5s"
        }
      ]
    }
```

二 用户、订单跨服务调用

产生背景

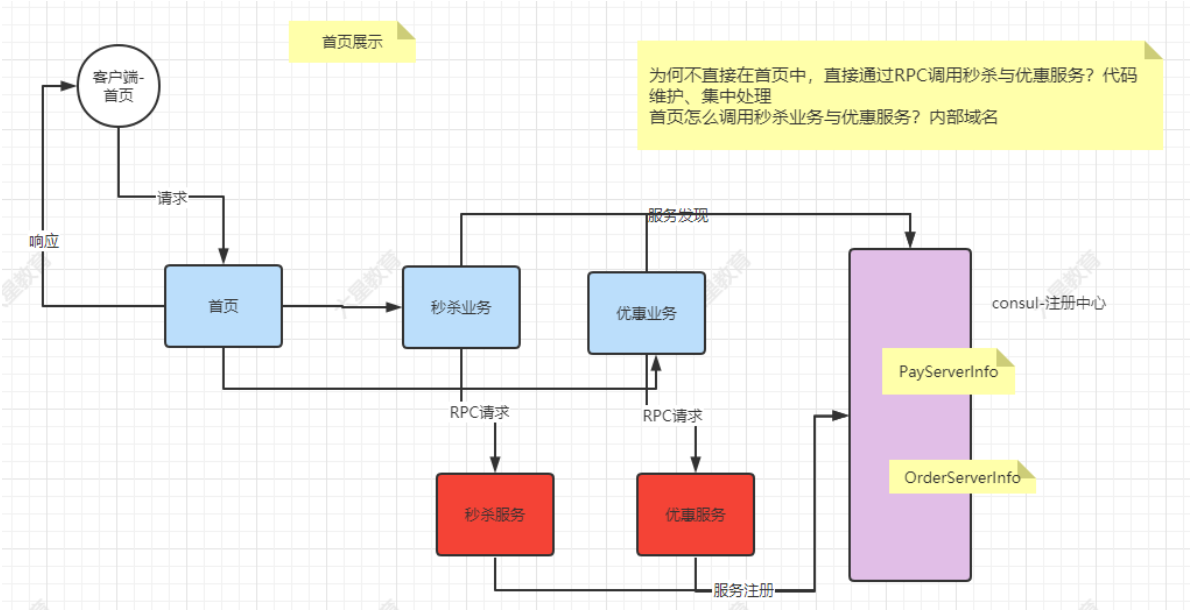
在分布式的部署模式下，各个系统都是独立部署，各司其职。但对一个业务请求来说，往往需要很多的业务系统参与其中才能完成相关业务操作。

例如：用户浏览商城首页，首页展示了推荐商品、新品热卖、秒杀、每日特价等等数据，这些数据都是来源于不同的业务系统。但用户看到的商城首页，所以当首页展示数据时，就需要调用其它的业务系统来提供数据。这就是跨服务调用的背景

什么是跨服务调用?

当A系统需要借助B、C、D系统才能完成当前业务时，就要采用跨服务调用不同的系统。

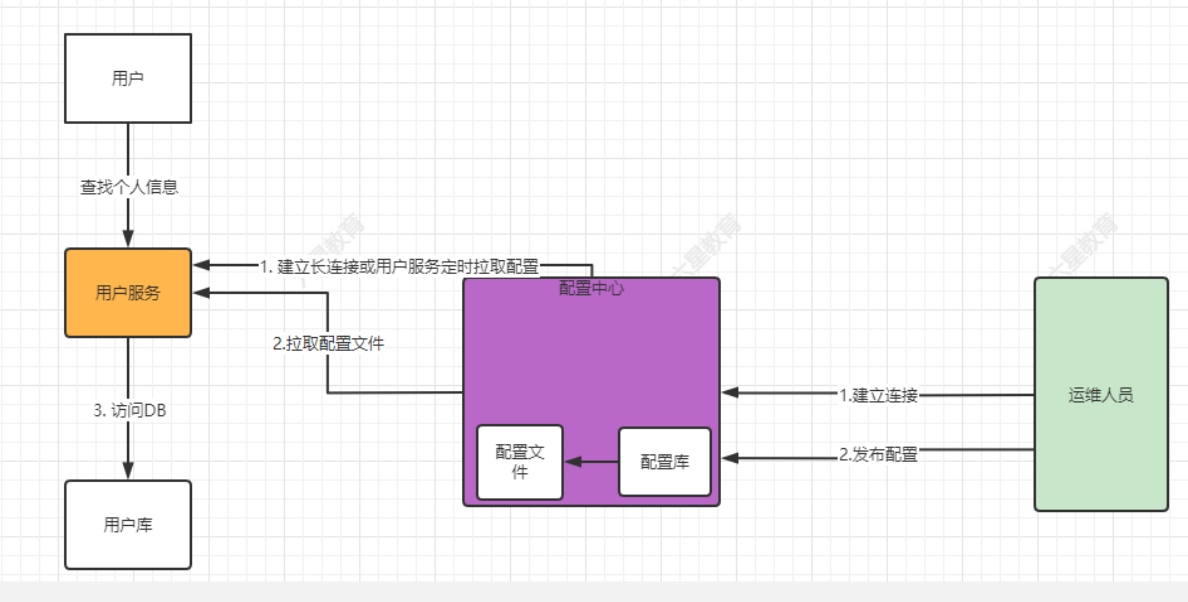
怎么做?



- 启动用户、订单服务时，把它们注册到consul 中
- 用户服务通过RPC接口调用订单服务
- 订单服务处理相关数据与逻辑操作
- 订单服务响应RPC请求到用户服务
- 用户服务获取数据后，做善后处理
- 最终由用户服务响应给客户端

三 手写配置中心原理实战

配置中心的原理



1. 启动配置中心服务，用于管理整个微服务用到的配置信息
2. 建立配置中心客户端连接，用户发布、更新、删除相关的配置项
3. 监听配置(变化事件)，一但配置中心有更新变化，监听的配置事件直接就把最新的配置信息更新到本地`config`文件中