

1.微服务架构体系总体概述

1.1 微服务架构演进过程

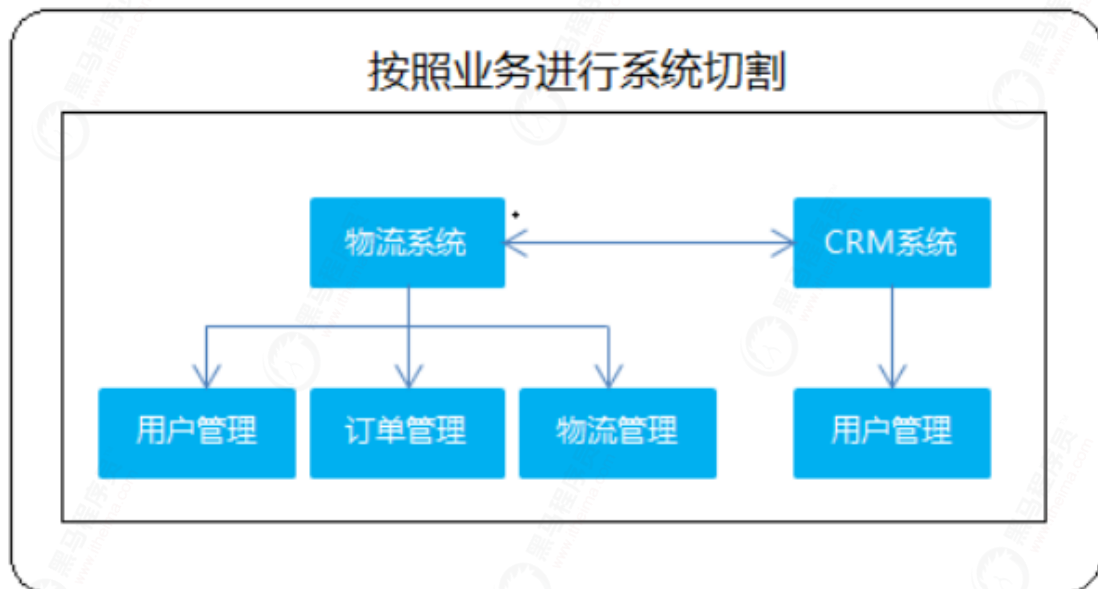
单体架构

一个归档包（例如war格式或者jar格式）包含了应用所有功能的应用程序，我们通常称之为单体应用。



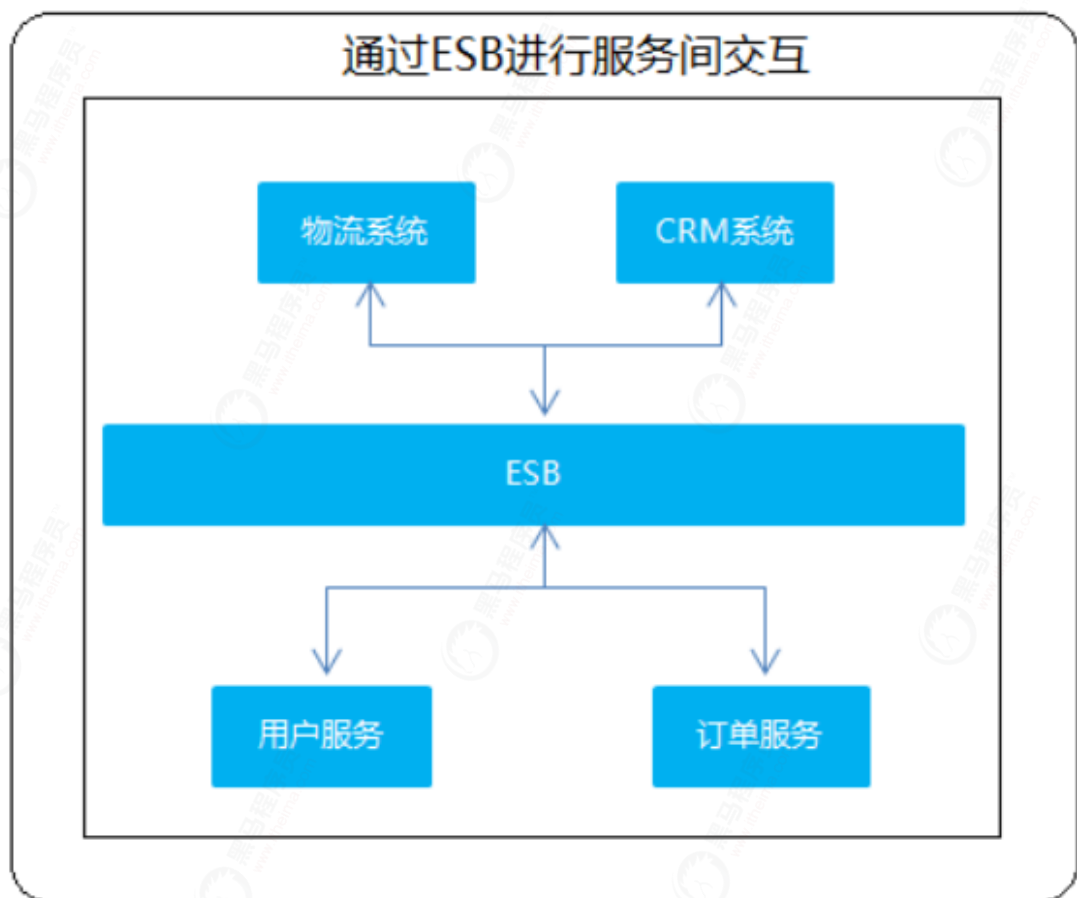
垂直架构

它把现有的系统根据业务的不同进行了拆分，按照业务进行切割，形成小的单体项目



SOA架构

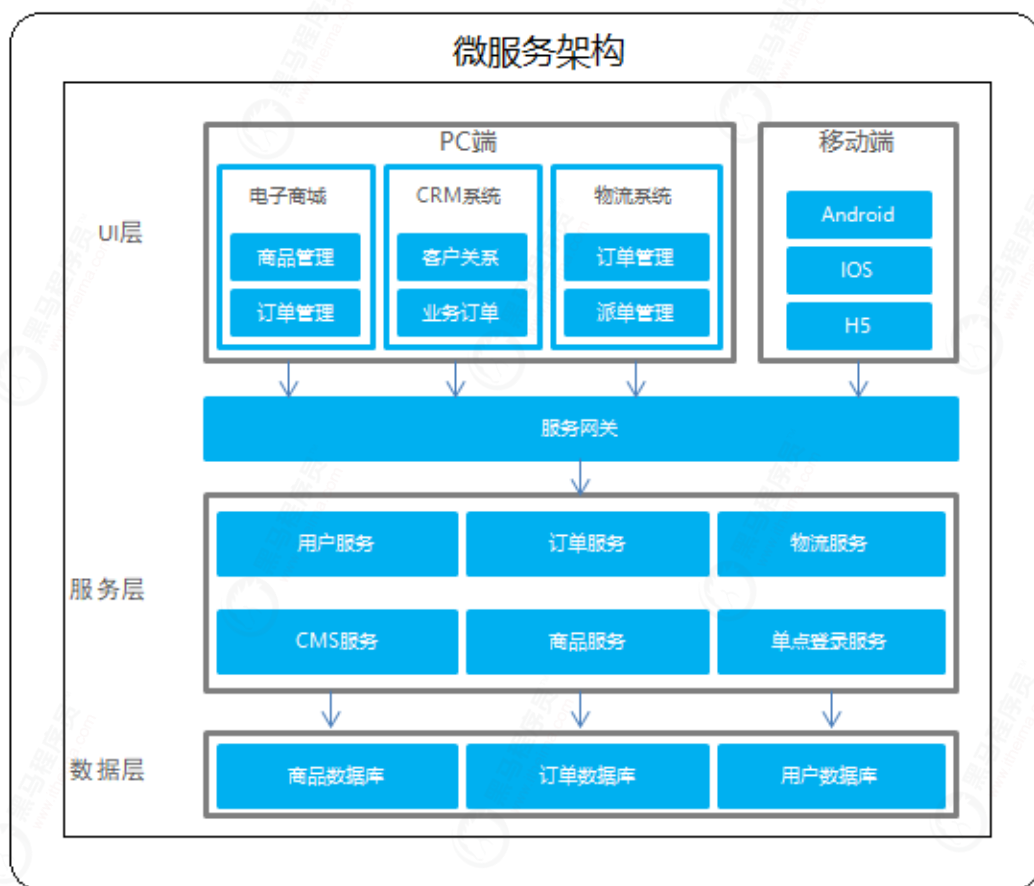
SOA (Service-Oriented Architecture, 面向服务的架构) 是一个组件模型，它将应用程序的不同功能单元（称为服务）进行拆分，并通过这些服务之间定义良好的接口和契约联系起来。



微服务架构

- 微服务简介

"微服务"一词源于 Martin Fowler 的名为 Microservices 的博文,可以在他的官方博客上找到<http://martinfowler.com/articles/microservices.html>。简单地说,微服务是系统架构上的一种设计风格,它的主旨是将一个原本独立的系统拆分成多个小型服务,这些小型服务都在各自独立的进程中运行,服务之间通过基于 HTTP 的 RESTful API 进行通信协作。



常见微服务框架

Spring的spring cloud、阿里dubbo、华为ServiceComb、腾讯Tars、[Facebook](#)的thrift、新浪微博 Motan

1.2 Spring Cloud介绍



Spring Cloud

<https://www.springcloud.cc/>



Spring Cloud Config

Spring

配置管理工具包，让你可以把配置放到远程服务器，集中化管理集群配置，目前支持本地存储、Git以及Subversion。



Spring Cloud Bus

Spring

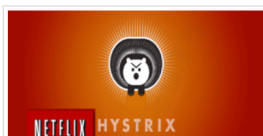
事件、消息总线，用于在集群（例如，配置变化事件）中传播状态变化，可与Spring Cloud Config联合实现热部署。



Eureka

Netflix

云服务发现，一个基于 REST 的服务，用于定位服务，以实现云端中间层服务发现和故障转移。



Hystrix

Netflix

熔断器，容错管理工具，旨在通过熔断机制控制服务和第三方库的节点，从而对延迟和故障提供更强大的容错能力。



Zuul

Netflix

Zuul 是在云平台上提供动态路由、监控、弹性安全等边缘服务的框架。Zuul 相当于是设备和 Netflix 流应用的 Web 网站后端所有请求的前门。



Archaius

Netflix

配置管理API，包含一系列配置管理API，提供动态类型化属性、线程安全配置操作、轮询框架、回调机制等功能。



Consul

HashiCorp

封装了Consul操作，consul是一个服务发现与配置工具，与Docker容器可以无缝集成。



Spring Cloud for Cloud Foundry

Pivotal

通过OAuth2协议绑定服务到CloudFoundry，CloudFoundry是VMware推出的开源PaaS云平台。



Spring Cloud Sleuth

Spring

日志收集工具包，封装了Dapper和log-based追踪以及Zipkin和HTrace操作，为SpringCloud应用实现了一种分布式追踪解决方案。



Spring Cloud Data Flow

Pivotal

大数据操作工具，作为Spring XD的替代产品，它是一个混合计算模型，结合了流数据与批量数据的处理方式。



Spring Cloud Security

Spring

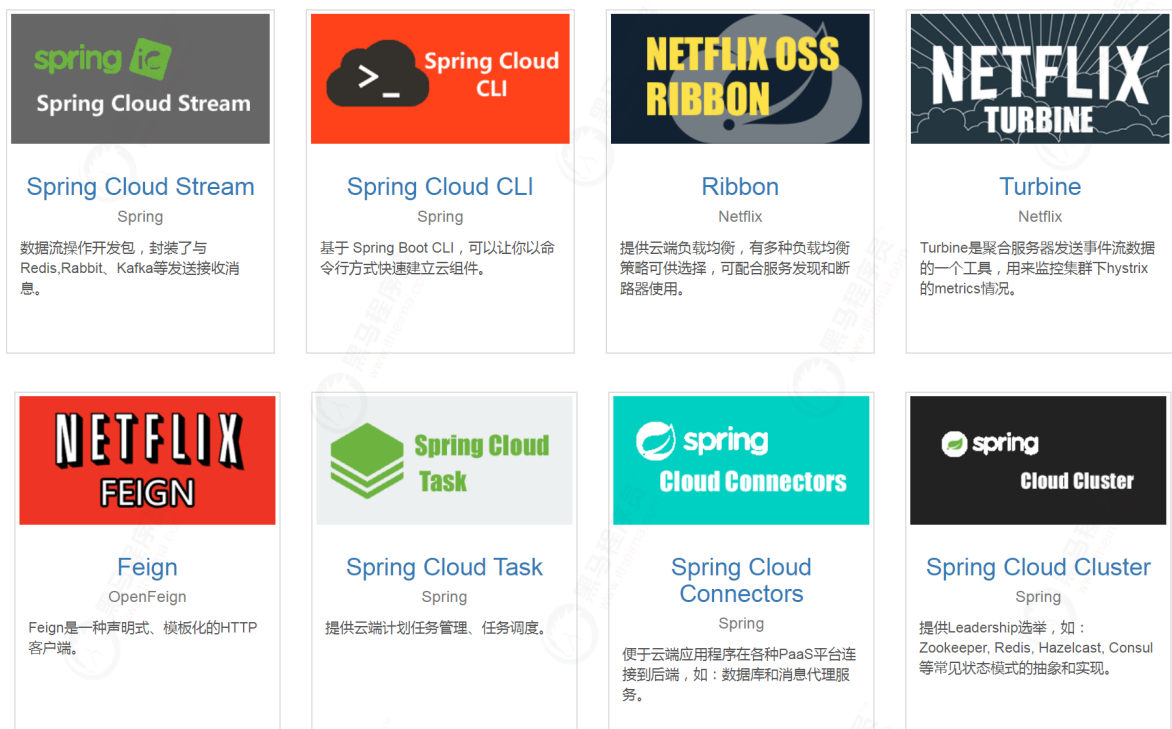
基于spring security的安全工具包，为你的应用程序添加安全控制。



Spring Cloud Zookeeper

Spring

操作Zookeeper的工具包，用于使用zookeeper方式的服务发现和配置管理。





Spring Cloud Starters

Spring Cloud Starters

Pivotal

Spring Boot式的启动项目，为Spring Cloud提供开箱即用的依赖管理。

1.3 Spring Cloud Alibaba介绍

Spring Cloud Alibaba 致力于提供微服务开发的一站式解决方案。此项目包含开发分布式应用微服务的必需组件，方便开发者通过 Spring Cloud 编程模型轻松使用这些组件来开发分布式应用服务

主要组件

- Sentinel：把流量作为切入点，从流量控制、熔断降级、系统负载保护等多个维度保护服务的稳定性。
- Nacos：一个更易于构建云原生应用的动态服务发现、配置管理和服务管理平台。
- RocketMQ：一款开源的分布式消息系统，基于高可用分布式集群技术，提供低延时的、高可靠的消息发布与订阅服务。
- Dubbo：Apache Dubbo™ 是一款高性能 Java RPC 框架。
- Seata：阿里巴巴开源产品，一个易于使用的高性能微服务分布式事务解决方案。

1.4 组件版本兼容介绍

Spring Cloud Alibaba版本

<https://spring.io/projects/spring-cloud-alibaba#learn>



spring cloud与spring cloud alibaba版本兼容关系

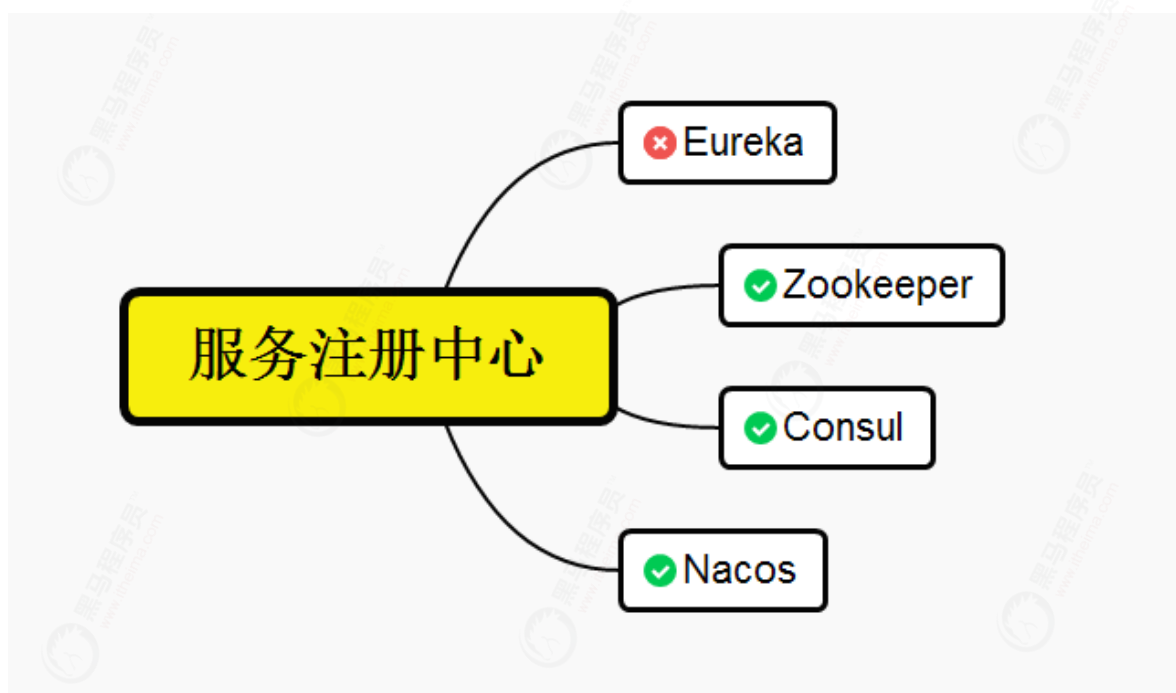
<https://github.com/alibaba/spring-cloud-alibaba/wiki/%E7%89%88%E6%9C%AC%E8%AF%B4%E6%98%8E>

毕业版本依赖关系(推荐使用)

Spring Cloud Version	Spring Cloud Alibaba Version	Spring Boot Version
Spring Cloud Hoxton.SR3	2.2.1.RELEASE	2.2.5.RELEASE
Spring Cloud Hoxton.RELEASE	2.2.0.RELEASE	2.2.X.RELEASE
Spring Cloud Greenwich	2.1.2.RELEASE	2.1.X.RELEASE
Spring Cloud Finchley	2.0.2.RELEASE	2.0.X.RELEASE
Spring Cloud Edgware	1.5.1.RELEASE	1.5.X.RELEASE

2.分布式服务治理解决方案与应用

2.1 关于服务治理组件的停更/替换介绍



Eureka: 官方宣布2.x不再开源（闭源），之前的版本已经停止更新；也就说Eureka将来更多的技术提升已经没有了。所以，如果希望注册中心有更多强大功能的话，还需要另辟蹊径。

Zookeeper: 在企业级Zookeeper注册中心与 Dubbo组合比较多一些，kafka使用的也是，随着Eureka的停更，我们可以通过spring-cloud-starter-zookeeper-discovery这个启动器，将Zookeeper做为springcloud的注册中心。

Consul: go语言开发的，也是一个优秀的服务注册框架，使用量也比较多。

Nacos: 来自于SpringCloudAlibaba，在企业中经过了百万级注册考验的，不但可以完美替换Eureka，还能做其他组件的替换，所以，Nacos也强烈建议使用

2.2 Consul服务注册与发现

2.2.1 Consul总体概述

1. Consul简介

Consul是HashiCorp（DevOps 服务商）公司推出的开源工具，它是分布式的、高可用的、可横向扩展的用于实现分布式系统的服务发现与配置组件。

Consul由Go语言开发，部署起来非常容易，只需要极少的可执行程序 and 配置文件，具有绿色、轻量级的特点。



2. Consul有哪些特点

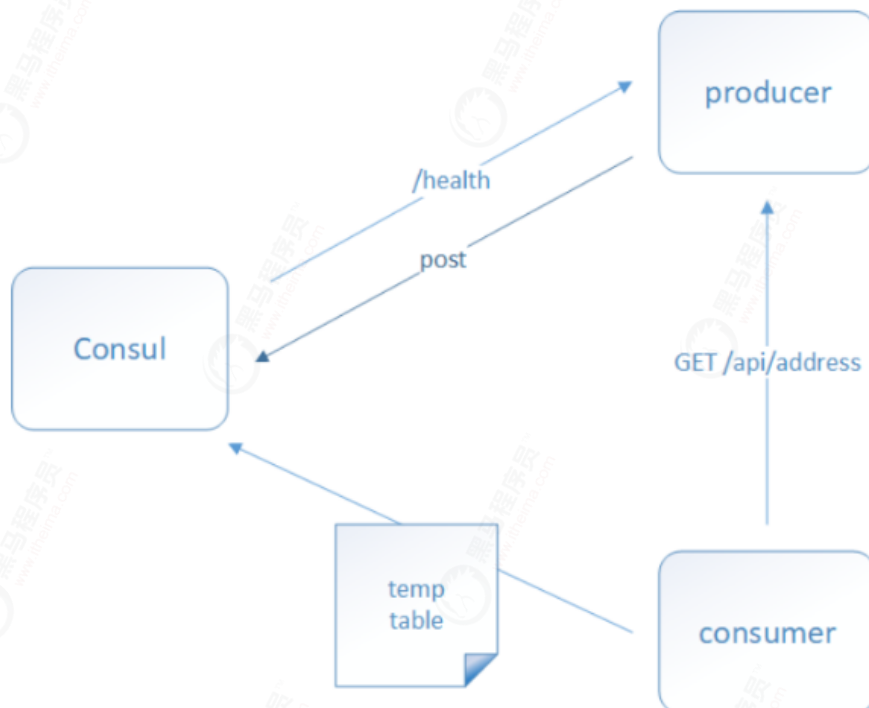
服务发现：能够注册一个服务，其他客户可以在Consul上查询一个指定服务的提供者。

健康检查：Consul可以灵活的使用脚本和http等来检测注册在其上的服务是否可用

key/value存储：这个功能和springcloud config、nacos有些类似，可以通过HTTP API方便地使用。

多数据中心支持：Consul支持开箱即用的多数据中心支持，这意味着用户不用建立额外的抽象层让业务扩展到各个区域。

3. Consul工作原理



2.2.2 Consul快速入门

本章节场景：

我们将建立两个Module，父项目是itheima-service-consul，里面没有任何代码，它用来管理我们的聚合工程。

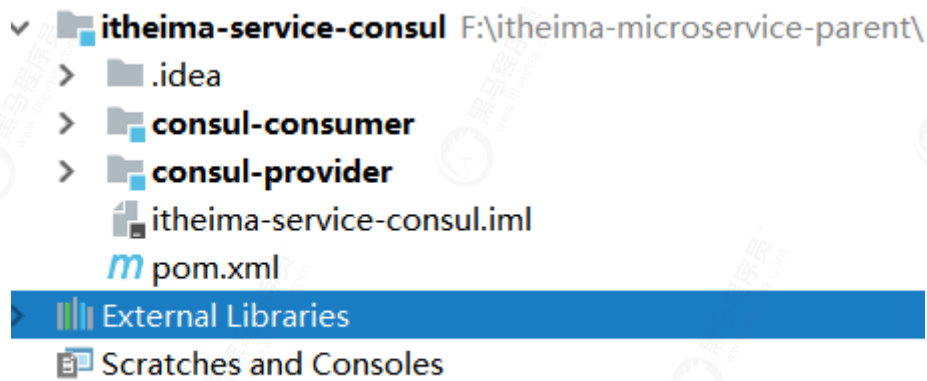
首先安装服务注册中心服务端Consul

然后将我们的consul-consumer、consul-provider注册到Consul服务端；

```
<module>consul-consumer</module>
<module>consul-provider</module>
```

然后consul-consumer通过Consul注册发现机制可以正常消费consul-provider提供的服务。

itheima-service-consul工程结构如下图：



1) Consul Server安装与启动

1) Windows下安装

1) 下载

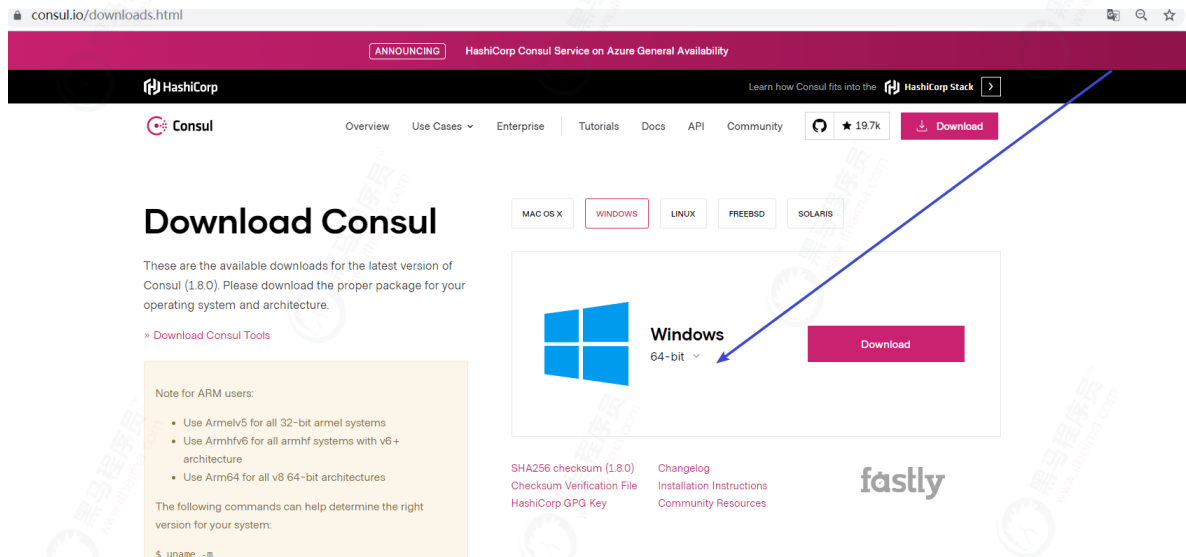
官网下载地址：

<https://www.consul.io/downloads.html>

进入首页，选择对应的版本，如下图

此处以windows系统为例

当前最新版本：1.8.0



2) 解压

将下载的安装包进行解压，根据自己实际情况选择路径

3) 安装/启动

cmd切换到当前目录下

```
consul.exe agent -dev
```

本地 8500访问

以dev模式启动consul

-dev 表示开发模式运行，默认客户端地址是在127.0.0.1 上

这个-dev（该节点的启动不能用于生产环境，因为该模式下不会持久化任何状态）启动模式仅仅是为了快速便捷的启动单节点consul

以server模式启动consul

```
consul agent -server -bind=192.168.0.109 -data-dir=/etc/consul.d
```

其实没有 -server，默认就是以client模式启动的

以client模式启动consul

```
consul agent -client=0.0.0.0 -bind=192.168.0.109 -data-dir=/etc/consul.d
```

Server 模式和 Client 模式的功能是一样的，唯一有区别的就是Server模式会把所有的信息持久化到本地，这样遇到故障，信息是可以被保留的

2) 容器化高可用集群搭建

最新版本1.8.0

```
docker pull consul:latest
```

启动节点1

```
docker run --name consul1 -d -p 8500:8500 -p 8300:8300 -p 8301:8301 -p 8302:8302 -p 8600:8600 consul agent -server -bootstrap-expect 2 -ui -bind=0.0.0.0 -client=0.0.0.0
```

端口	说明
TCP/8300	8300 端口用于服务器节点。客户端通过该端口 RPC 协议调用服务端节点。
TCP/UDP/8301	8301 端口用于单个数据中心所有节点之间的互相通信，即对 LAN 池信息的同步。它使得整个数据中心能够自动发现服务器地址，分布式检测节点故障，事件广播（如领导选举事件）。
TCP/UDP/8302	8302 端口用于单个或多个数据中心之间的服务器节点的信息同步，即对 WAN 池信息的同步。它针对互联网的高延迟进行了优化，能够实现跨数据中心请求。
8500	8500 端口基于 HTTP 协议，用于 API 接口或 WEB UI 访问。
8600	8600 端口作为 DNS 服务器，它使得我们可以通过节点名查询节点信息。

查看consul1的ip地址

```
docker inspect --format='{{.NetworkSettings.IPAddress}}' consul1
```

启动第二个consul服务(端口8501): consul2，并加入consul1（使用join命令）

```
docker run --name consul2 -d -p 8501:8500 consul agent -server -ui -bind=0.0.0.0 -client=0.0.0.0 -join 172.17.0.2
```


启动第三个consul服务(端口8502): consul3, 并加入consul1 (使用join命令)

```
docker run --name consul3 -d -p 8502:8500 consul agent -server -ui -bind=0.0.0.0 -client=0.0.0.0 -join 172.17.0.2
```

查看consul集群信息

```
docker exec -it consul1 consul members
```

可以看到集群里的三个节点

```
root@localhost ~]# docker exec -it consul1 consul members
node      Address        Status  Type   Build  Protocol  DC    Segment
91a990aee297 172.17.0.4:8301 alive   server  1.8.0   2      dc1    <all>
7b89dfadd1b9 172.17.0.2:8301 alive   server  1.8.0   2      dc1    <all>
ee3164623a7f 172.17.0.3:8301 alive   server  1.8.0   2      dc1    <all>
root@localhost ~]#
```

查看容器状态

```
docker ps
```

参数解释:

- agent: 表示启动 Agent 进程。
- -server: 表示启动 Consul Server 模式; -client: 表示启动 Consul Client 模式。
- bootstrap-expect 2: 集群至少两台服务器, 才能选举集群 leader, 数目一达到, 它就会被激活
- -ui: 表示启动 Web UI 管理器, 默认开放端口 8500, 所以上面使用 Docker 命令把 8500 端口对外开放。
- -node: 节点的名称, 集群中必须是唯一的, 默认是该节点的主机名。
- -client: consul服务侦听地址, 这个地址提供HTTP、DNS、RPC等服务, 默认是127.0.0.1所以不对外提供服务, 如果你要对外提供服务改成0.0.0.0
- bind: 监听网口, 0.0.0.0 表示所有网口, 如果不指定默认未 127.0.0.1, 则无法和容器通信

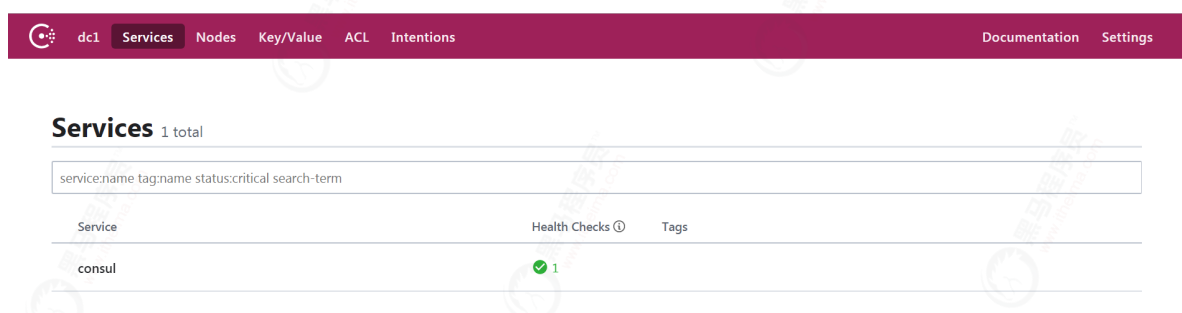
3) Consul访问

浏览器输入

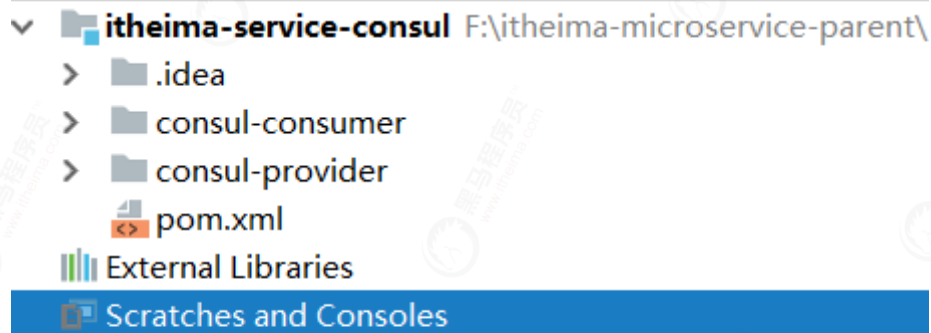
<http://localhost:8500/>

<http://192.168.23.139:8500/>

即可访问Consul, 如下图



2) Consul客户端搭建与应用



1) 创建服务提供者

1. 创建服务提供者consul-provider

2. 引入起步依赖

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-consul-discovery</artifactId>
</dependency>
<!--consul中健康检查需要用到actuator，不添加会check failing-->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

3. 修改bootstrap.yml文件

```
server:
  port: 8763
spring:
  application:
    #对应consul控制台的实例名称
    name: consul-provider
  cloud:
    consul:
      host: 192.168.23.139
      port: 8500
      discovery:
        #节点检查的时候默认使用hostname报错，建议true通过ip访问
        prefer-ip-address: true
        #对应consul控制台的service名称（restTemplate调用）
```

```
serviceName: consul-provider
```

坑:

必须设置为prefer-ip-address=true

否则在节点检查的时候报错

导致服务无法注册

4.启动入口

```
@SpringBootApplication
@EnableDiscoveryClient
public class ConsulProviderApplication {

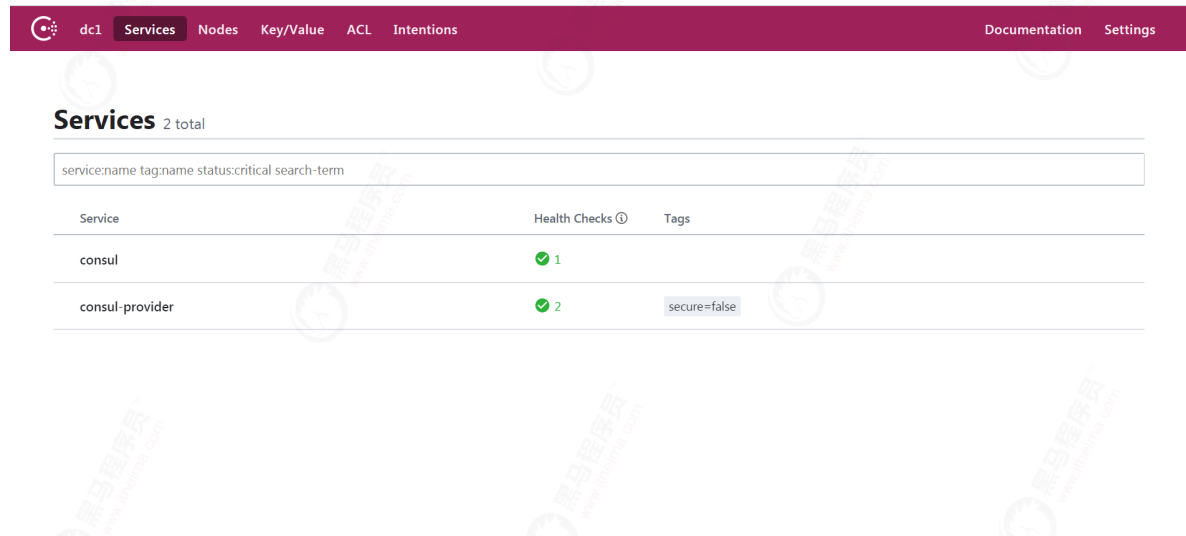
    public static void main(String[] args) {
        SpringApplication.run(ConsulProviderApplication.class, args);
    }

}
```

5. 访问

启动consul-provider, 打开浏览器

```
http://localhost:8500
```



Service	Health Checks ①	Tags
consul	✓ 1	
consul-provider	✓ 2	secure=false

查看 Consul web 界面可以看到consul-provider已经注册到Consul服务端

2) 创建服务消费者

创建consul-consumer模块

1. 引入起步依赖

忽略, 步骤同创建服务提供者。

2. 修改bootstrap.yml文件

```

server:
  port: 8766
spring:
  application:
    #对应consul控制台的实例名称
    name: consul-consumer
  cloud:
    consul:
      host: 192.168.23.139
      port: 8500
      discovery:
        #节点检查的时候默认使用hostname报错, 建议true通过ip访问
        prefer-ip-address: true
        #对应consul控制台的service名称 (restTemplate调用)
        serviceName: consul-consumer

```

3. 创建消费Controller

```

@RestController
public class ConsulConsumerController {
    @Autowired
    RestTemplate restTemplate;
    @GetMapping("/hello")
    public String hello(String name) {
        return restTemplate.getForObject("http://consul-provider/hello?name=" +
        name, String.class);
    }
}

```

在consul-consumer中通过RestTemplate去消费服务提供者consul-provider

4. 访问

启动consul-consumer, 打开浏览器<http://localhost:8500>, 如下:

Service	Health Checks ①	Tags
consul	✓ 1	
consul-consumer	✓ 2	secure=false
consul-provider	✓ 2	secure=false

这时, 服务提供者、服务消费者注册到Consul全部成功

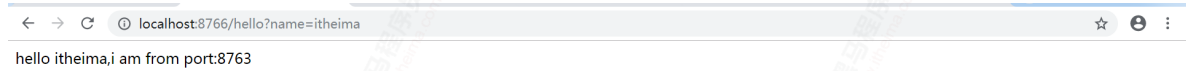
执行如下:

- 1、可以通过RestTemplate使用服务8766端口访问8763
- 2、直接访问8763都可以

```
http://localhost:8766/hello?name=itheima
```

或者

```
http://localhost:8763/hello?name=itheima
```



可以说明，consul-consumer通过Consul注册发现可以正常消费consul-provider提供的服务

节点健康检查

```
http://172.16.43.175:8763/actuator/health
```

删除无用的注册节点

执行PUT

```
http://192.168.23.139:8500/v1/agent/service/deregister/consul-provider-8763
```

192.168.23.139:8500consul的ip和端口

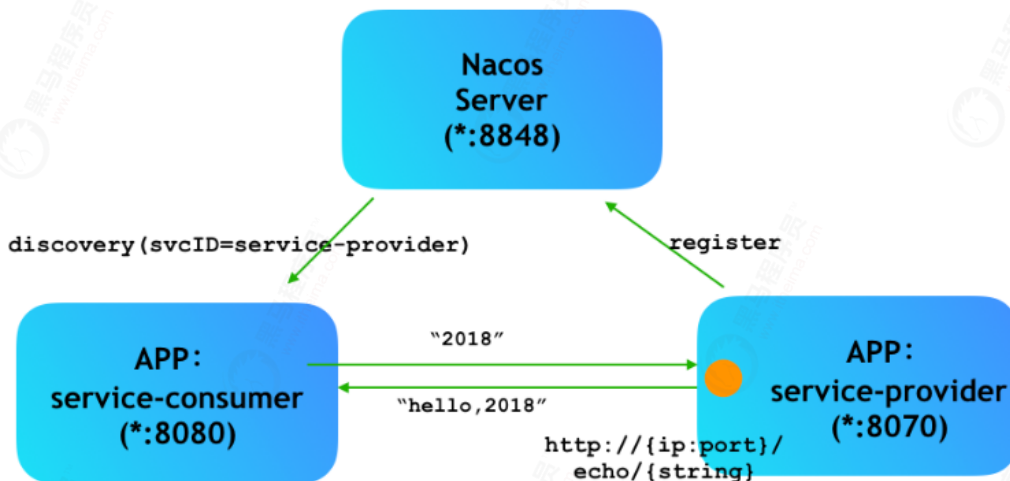
consul-provider-8733；对应控制台的CheckID

2.3 Nacos服务注册与发现

2.3.1 Nacos注册与发现概述

1. Nacos简介

Nacos (Dynamic Naming and Configuration Service) 是阿里巴巴2018年7月开源的项目，致力于发现、配置和管理微服务。



2. Nacos版本介绍

<https://github.com/alibaba/nacos/releases>

Nacos目前最新版本为：nacos-server-1.3.1（最新稳定版）

ReleasesTags

Latest release
1.3.1
5e53396
Verified
Compare

1.3.1 (July 10, 2020)

KomachiSion released this 15 days ago

This version mainly fixes some bugs and optimizes the code quality.

- [#2089] Fix config list return page bug.
- [#2237] Enhance HttpClient usage.
- [#2800] Update beatinfo when received instance modify info
- [#2810] Don't send beat when standalone model.
- [#2839] Fix Json conversion exception when loading Datum cache file.
- [#2844] Fix Time format error.
- [#2847] Fix config IT
- [#2982] Fix the error message could not be passed through at the RPC layer.
- [#2985] Fix In inline mode, the task execution condition determines the problem.
- [#2988][#2989] Fix can get configuration from standalone + embedded storage.
- [#2992] Use new code style.
- [#2994] Optimize the configuration post - release process.
- [#3060] Upgrade the database driver version of Mysql to 8
- [#3063] Fix parameter validation boundary problem.
- [#3110] Fix confused connection when one nacos nodes down.
- [#3160] Fix compatibility issue when reading cache files with suffix ".datum"
- [#3175] Fix probably the end value is negative number in ServiceController.java.

Assets 4

nacos-server-1.3.1.tar.gz	71.3 MB
nacos-server-1.3.1.zip	71.3 MB
Source code (zip)	
Source code (tar.gz)	

2.3.2 Nacos配置高可用构建

1) Nacos单点（Dev）

1、下载镜像


```
docker pull nacos/nacos-server:1.3.1
```

2、查看镜像

```
docker images
```

2、启动镜像

创建自定义网络

```
docker network create --subnet=172.188.0.0/16 czbkNetwork
```

启动

```
docker run --name nacos --env MODE=standalone --net czbkNetwork --ip 172.188.0.44 --privileged=true -p 8848:8848 --restart=always -d dc833dc45d8f
```

Docker容器的重启策略是面向生产环境的一个启动策略，在开发过程中可以忽略该策略

--restart常用参数有3个可选值：

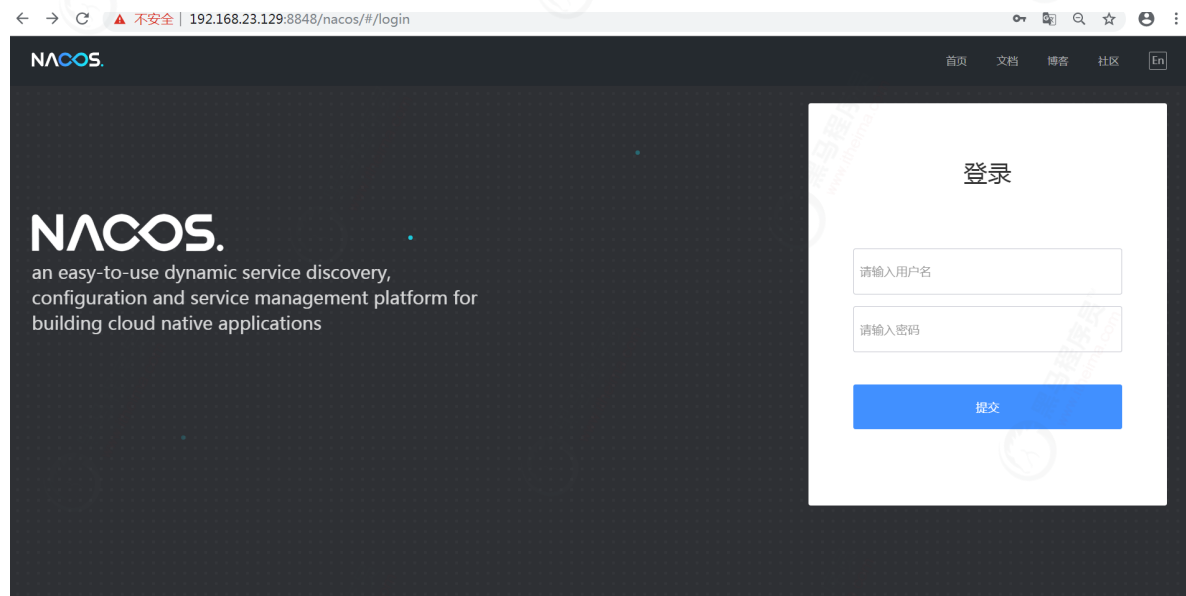
no：容器退出时不重启，默认

on-failure：容器故障退出（返回值非零）时重启

always：容器退出时总是重启

3、访问nacos

```
http://ip:8848/nacos
```



用户名nacos密码nacos

2) Nacos高可用集群

- 单机模式 - 用于测试和单机试用。
- 集群模式 - 用于生产环境，确保高可用。
- 多集群模式 - 用于多数据中心场景

集群安装约束：

1. 64 bit OS Linux/Unix/Mac，推荐使用Linux系统。
2. 集群需要依赖mysql，单机可不必
3. 3个或3个以上Nacos节点才能构成集群

搭建Nacos高可用集群步骤：

- 1、需要去Nacos官网clone Nacos集群项目nacos-docker
- 2、nacos-docker是使用的Docker Compose对容器进行编排，所以首先需要安装Docker Compose
详细信息可参照Nacos官网：<https://nacos.io/zh-cn/docs/quick-start-docker.html>

1) 安装Docker Compose

什么是Docker Compose

Compose项目是Docker官方的开源项目，负责实现对Docker容器集群的快速编排。

1、安装

```
#在Linux下下载（下载到/usr/local/bin）
curl -L "https://github.com/docker/compose/releases/download/1.25.0/docker-
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

# 设置文件可执行权限
chmod +x /usr/local/bin/docker-compose

# 查看版本信息
docker-compose --version
```

看到版本信息，说明安装成功。

2.卸载

```
# 二进制包方式安装的，删除二进制文件即可
rm /usr/local/bin/docker-compose
```

常见问题

如果安装完成后，查看版本频繁报错，如下

Cannot open self /usr/local/bin/docker-compose or archive /usr/local/bin/docker-
compose.pkg

可使用下面的解决方案

- 1、切换到/usr/local/bin，删除之前的下载/docker-compose
- 2、切换到/usr/local/bin执行下面的命令

wget https://github.com/docker/compose/releases/download/1.25.0-rc4/docker-compose-Linux-x86_64

3、下载完毕后重命名为docker-compose

mv docker-compose-Linux-x86_64 docker-compose

4、赋权限

5、查看版本成功

2) 克隆Nacos-docker项目

如果本机没有安装git

请使用 yum install git安装

```
#切换到自定义目录
cd /usr/local/nacos
#开始clone
git clone https://github.com/nacos-group/nacos-docker.git
```

3) 运行nacos-docker脚本

```
#执行编排命令
docker-compose -f /usr/local/nacos/nacos-docker/example/cluster-hostname.yaml up
```

上面的编排命令主要下载mysql镜像和nacos镜像，自动完成镜像下载/容器启动

Pulling from nacos/nacos-mysql, 版本是5.7（执行初始化脚本）

Pulling nacos3 (nacos/nacos-server:latest)最新版本

4) 停止、启动

```
#启动
docker-compose -f /usr/local/nacos/nacos-docker/example/cluster-hostname.yaml
start
```

```
[root@bj-yjy-hly-kg02 nacos-docker]# docker-compose -f /usr/local/nacos/nacos-docker/example/cluster-hostname.yaml start
Starting mysql ... done
Starting nacos3 ... done
Starting nacos2 ... done
Starting nacos1 ... done
[root@bj-yjy-hly-kg02 nacos-docker]#
```

Nacos集群启动成功

```
#停止
docker-compose -f /usr/local/nacos/nacos-docker/example/cluster-hostname.yaml
stop
```

```
[root@bj-yjy-hly-kg02 ~]# docker-compose -f /usr/local/nacos/nacos-docker/example/cluster-hostname.yaml stop
Stopping nacos2 ... done
Stopping nacos1 ... done
Stopping nacos3 ... done
Stopping mysql ... done
[root@bj-yjy-hly-kg02 ~]#
```

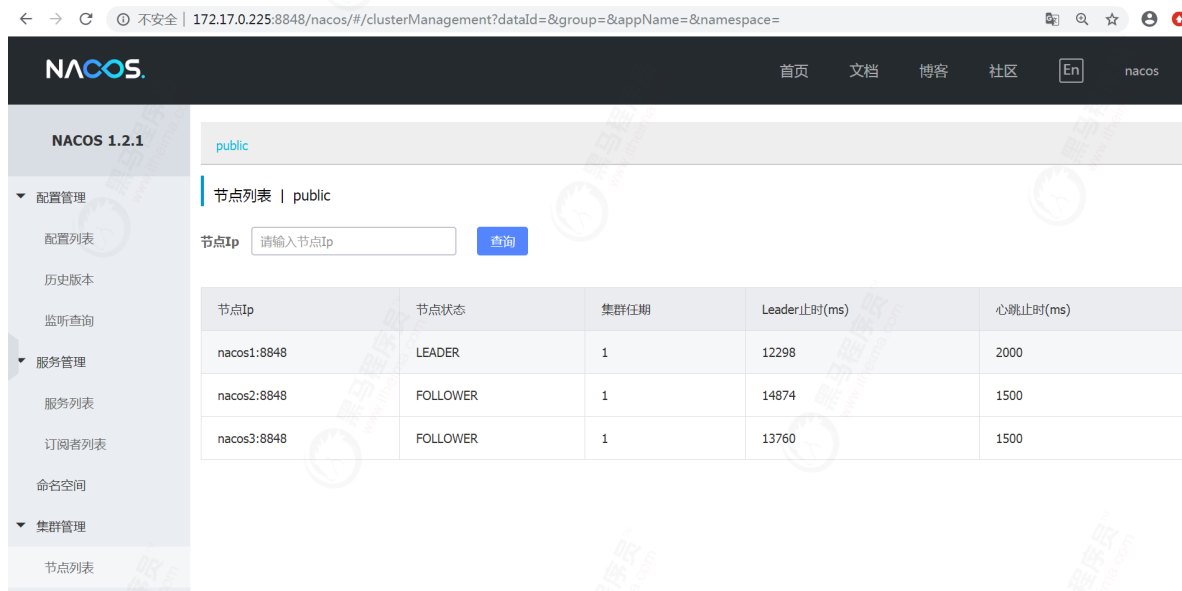
nacos所有节点、mysql成功终止（停止容器）

6) 访问Nacos

http://172.17.0.225:8848/nacos

http://172.17.0.225:8849/nacos

http://172.17.0.225:8850/nacos



- Leader：负责接收客户端的请求
- Candidate：用于选举Leader的一种角色
- Follower：负责响应来自Leader或者Candidate的请求

【集群管理】---【节点列表】可查看Nacos集群节点信息

2.3.3 Nacos快速入门

本章节场景：

我们将建立两个Module），父项目是itheima-service-nacos，里面没有任何代码，它用来管理我们的聚合工程。

首先我们下载安装Nacos Server，它用来作为服务注册中心

场景：

作为注册发现，将我们的nacos-spring-cloud-consumer-example、nacos-spring-cloud-discovery-example注册到Nacos上实现远程调用

章节将采取Nacos进行服务注册、服务发现、配置管理，章节代码：itheima-chapter-04，工程结构如下图：



1) Nacos Client服务注册发现

1) 创建服务提供者模块

与Spring Cloud集成，工程为nacos-spring-cloud-provider-example

2) 引入起步依赖

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>com.alibaba.cloud</groupId>
  <artifactId>spring-cloud-starter-alibaba-nacos-
discovery</artifactId>
</dependency>
```

3) 配置服务提供者

```
server.port=8070
spring.application.name=nacos-provider
spring.cloud.nacos.discovery.server-addr=192.168.23.139:8848
```

在 application.properties 中配置 Nacos server 的地址,,可以通过 Nacos 的服务注册发现功能将其服务注册到 Nacos server 上

4) 开启服务注册发现功能

```
@SpringBootApplication
@EnableDiscoveryClient
public class NacosProviderApplication {

    public static void main(String[] args) {
        SpringApplication.run(NacosProviderApplication.class, args);
    }

    @RestController
    class EchoController {
        @RequestMapping(value = "/echo/{string}", method = RequestMethod.GET)
        public String echo(@PathVariable String string) {
            return "Hello Nacos Discovery " + string;
        }
    }
}
```

```
}  
}
```

在 `application.properties` 中配置 Nacos server 的地址，从而服务消费者可以通过 Nacos 的服务注册发现功能从 Nacos server 上获取到它要调用的服务

5) 创建服务消费者模块

与Spring Cloud集成，工程为nacos-spring-cloud-consumer-example

7) 引入起步依赖

```
<dependency>  
    <groupId>com.alibaba.cloud</groupId>  
    <artifactId>spring-cloud-starter-alibaba-nacos-  
discovery</artifactId>  
</dependency>  
</dependencies>  
<dependencies>  
    <dependency>  
        <groupId>org.springframework.boot</groupId>  
        <artifactId>spring-boot-starter-web</artifactId>  
    </dependency>  
</dependencies>
```

8) 配置服务消费者

```
server.port=8080  
spring.application.name=nacos-consumer  
spring.cloud.nacos.discovery.server-addr=192.168.23.139:8848
```

9) 开启服务注册发现功能

```
@SpringBootApplication  
@EnabledDiscoveryClient  
public class NacosConsumerApplication {  
  
    public static void main(String[] args) {  
        SpringApplication.run(NacosConsumerApplication.class, args);  
    }  
  
    @LoadBalanced  
    @Bean  
    public RestTemplate restTemplate() {  
        return new RestTemplate();  
    }  
  
    @RestController  
    public class TestController {  
  
        private final RestTemplate restTemplate;  
  
        @Autowired  
        public TestController(RestTemplate restTemplate) {  
            this.restTemplate = restTemplate;  
        }  
    }  
}
```

```

    @RequestMapping(value = "/echo/{str}", method = RequestMethod.GET)
    public String echo(@PathVariable String str) {
        return restTemplate.getForObject("http://nacos-provider/echo/" +
            str, String.class);
    }
}

```

通过 Spring Cloud 原生注解 `@EnableDiscoveryClient` 开启服务注册发现功能。给 `RestTemplate` 实例添加 `@LoadBalanced` 注解，开启 `@LoadBalanced` 与 `Ribbon` 的集成

7) 启动

启动 `nacos-spring-cloud-discovery-example` 下的 `ProviderApplication` 和 `ConsumerApplication`

The screenshot shows the Nacos 1.3.1 web interface. On the left is a sidebar with navigation options like '配置管理', '服务管理', '服务列表', etc. The main area displays the 'public' namespace service list. A table contains the following data:

服务名	分组名称	集群数目	实例数	健康实例数	触发保护阈值	操作
nacos-consumer	DEFAULT_GROUP	1	1	1	false	详情 示例代码 删除
nacos-provider	DEFAULT_GROUP	1	1	1	false	详情 示例代码 删除

由上图可知，我们的生产者和消费者都注册到了Nacos Server上了

调用

```
http://localhost:8080/echo/itheima
```

返回内容为 `Hello Nacos Discovery itheima`,说明我们从Nacos成功的消费了，如下图：

The screenshot shows a web browser window with the address bar containing 'localhost:8080/echo/itheima'. The page content displays 'Hello Nacos Discovery itheima'.

2) 多服务注册中心切换

场景Consul升级到Nacos

1、修改POM引用

将


```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-consul-discovery</artifactId>
</dependency>
```

修改为

```
<dependency>
  <groupId>com.alibaba.cloud</groupId>
  <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
</dependency>
```

2、修改配置文件

将

```
spring.cloud.consul.host=192.168.23.139
spring.cloud.consul.port=8500
```

修改为

```
spring.cloud.nacos.discovery.server-addr=192.168.23.139:8848
```

3、启动类注解

最好使用通用的注解@EnableDiscoveryClient，可以保证只修改配置，不修改代码

3 分布式配置中心解决方案与应用

目前市面上用的比较多的配置中心有（时间顺序）

- **Disconf**: 2014年7月百度开源的配置管理中心，同样具备配置的管理能力，不过目前已经不维护了，最近的一次提交是4-5年前了。
- **Spring Cloud Config**: 2014年9月开源，Spring Cloud 生态组件，可以和Spring Cloud体系无缝整合。
- **Apollo**: 2016年5月，携程开源的配置管理中心，具备规范的权限、流程治理等特性。
- **Nacos**: 2018年6月，阿里开源的配置中心，也可以做DNS和RPC的服务发现

3.1 Spring Cloud Config原生配置

3.1.1 Spring Cloud Config概述

1. Config简介

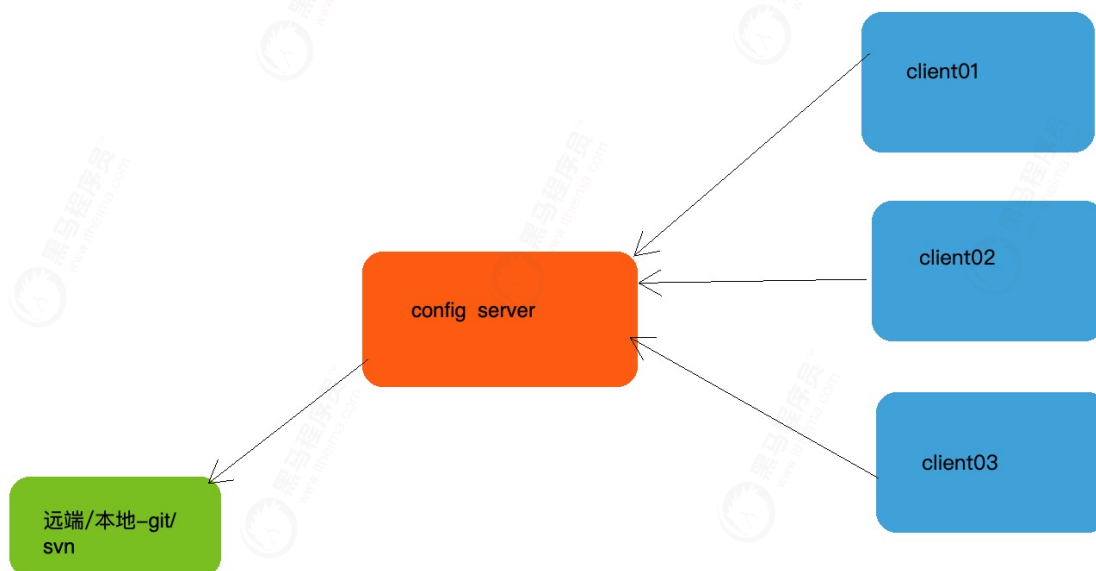
Spring Cloud Config主要分客户端、服务端。

服务端也称分布式配置中心，是一个独立的微服务应用，用来连接配置服务器并为客户端提供获取配置信息，加密/解密信息等访问接口。

客户端则是通过指定配置中心来管理应用资源，以及与业务相关的配置内容，并在启动的时候从配置中心获取和加载配置信息。

支持本地仓库和远程仓库读取

大体的工作原理如下



Config解决了哪些问题：

一句话总结，Config分布式配置中心解决了在分布式场景下多环境配置文件的管理和维护

3.1.2 Config快速入门

本章节场景：

我们将建立六个Module，父项目是itheima-service-config，里面没有任何代码，它用来管理我们的聚合工程。

```
<module>config-client</module>
<module>config-server</module>
<module>config-client-git</module>
<module>config-client-git-backup</module>
<module>config-server-git</module>
```

config-client配置中心客户端，用于本地读取

config-server配置中心服务端，用于服务端

config-client-git配置中心客户端。用于远程仓库码云读取

config-client-git-backup配置中心客户端。用于远程仓库码云读取

config-server-git配置中心服务端。用于配置远程仓库码云

具体场景：

1、本地文件读取

config-client、config-server启动注册到Nacos，然后读取本地文件内容

2、远程仓库码云读取

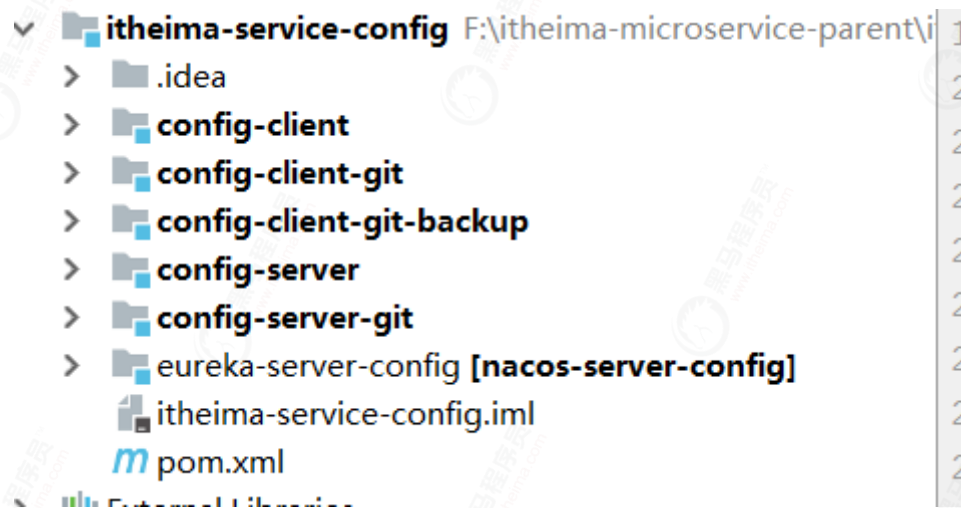
需要登录码云网站注册账号

将config-client-git-backup、config-server-git注册到服务注册发现中心

然后读取远程仓库码云上面的文件

3、远程仓库码云读取端点刷新

针对一个服务，修改了远程码云上的文件后，在不重启服务的情况下，通过暴露的refresh端点进行刷新注册到服务端



3.1.3 Config本地文件读取

1. 创建Config Server模块

创建config-server模块，当前模块为配置中心服务端，Config Server从本地读取配置文件

2. 引入起步依赖

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-config-server</artifactId>
</dependency>
```

3. 开启Config Server

```
package com.ithiema.config;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.config.server.EnableConfigServer;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication
@EnableConfigServer
@EnableDiscoveryClient
public class ConfigServerApplication {
```

```
public static void main(String[] args) {  
    SpringApplication.run(ConfigServerApplication.class, args);  
}  
}
```

在ConfigServerApplication类上新增@EnableConfigServer注解，开启config server功能。

4. 修改application.yml文件

```
spring:  
  cloud:  
    nacos:  
      discovery:  
        server-addr: 192.168.23.139:8848  
      config:  
        server:  
          native: #代表本地  
            search-locations: classpath:/shared  
    profiles:  
      active: native  
    application:  
      name: config-server  
  
server:  
  port: 8769
```

- #1. spring.profiles.active=native 用来配置Config Server从本地读取配置文件
- #2. spring.cloud.config.server.native.search-locations指定配置文件路径

5 新建目录

在resources/shared目录下新建config-client-dev.yml配置文件，配置数据如下：

```
server:  
  port: 8762  
  
itheima: itheima version 1
```

6. 创建Config Client模块

创建config-client模块， Config Client 通过调用Config Sever 的H即API 接口来读取配置文件

7. 引入起步依赖

```

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-config</artifactId>
</dependency>

```

8. 修改bootstrap.yml

```

spring:
  application:
    name: config-client
  cloud:
    config:
      uri: http://localhost:8769
      fail-fast: true
    profiles:
      active: dev
#配置说明:
# spring.cloud.config.uri 指定configServer的访问url
# spring.cloud.config.fail-fast 表示如果没有读取成功, 则执行快速失败
# spring.profiles.active表示读取dev环境的配置文件
# config-client就会去读config-server/resource/shared目录下面的 config-client-
dev.yml文件

```

9. 读取itheima

在ConfigClientApplication类中写一个API接口, 读取配置文件itheima变量

```

@SpringBootApplication
@RestController
public class ConfigClientApplication {

    @Value("${itheima}")
    String itheima;

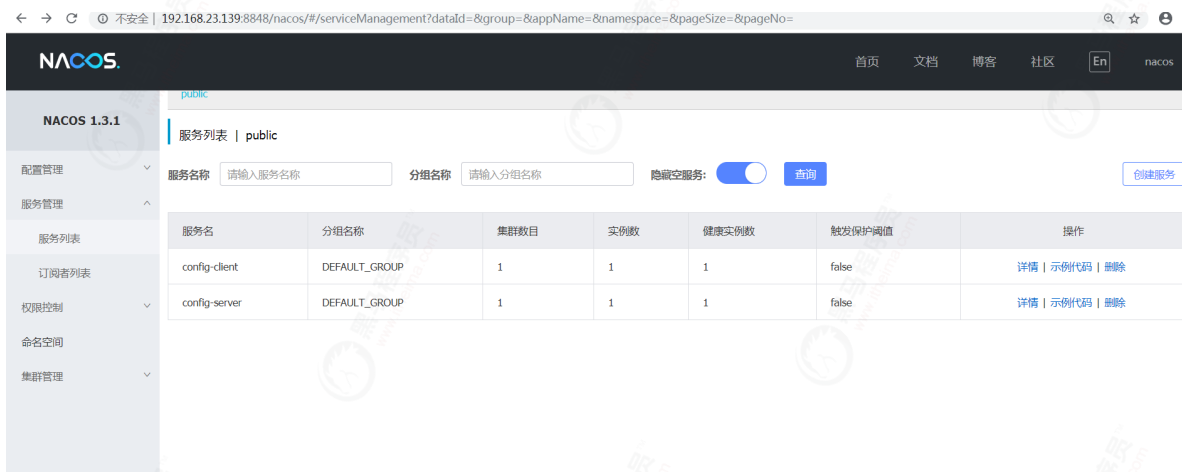
    public static void main(String[] args) {
        SpringApplication.run(ConfigClientApplication.class, args);
    }

    @RequestMapping(value = "/itheima")
    public String hi() {
        return itheima;
    }
}

```

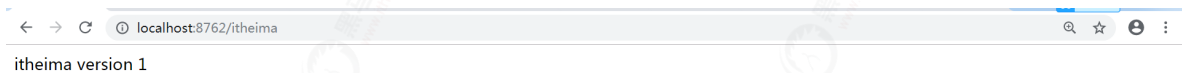
10. 启动

启动config-server工程、config-client工程, 如下图:



在浏览器输入

http://localhost:8762/itheima



启动config-client工程会在控制台的日志中发现config-client向url为<http://localhost:8769>的Config Server读取了配置文件

最终config-client程序启动的端口为8762，8762端口是在config-server/resource/shared目录中的config-client-dev.yml文件中配置的。

由此可见config-client向config-server中 成功读取配置文件

弊端

- 1、这里需要注意的是，server的配置（shared下的）如果发生变化，客户端服务端都要重启，而且是必须先启动server在启动客户端
- 2、没有界面化的设置，非常不灵活
- 3、不能及时通知到所有的微服务，需要重启获取，也就是说实时性丧失

3.1.4 Config远程git读取

1. 创建Config Server-git模块

创建config-server-git模块，当前模块为配置中心服务端，Config Server从远程git读取配置文件

2. 引入起步依赖

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-config-server</artifactId>
</dependency>
```

3. 开启Config Server

```
@SpringBootApplication
@EnableConfigServer
public class ConfigServerApplication {

    public static void main(String[] args) {
        SpringApplication.run(ConfigServerApplication.class, args);
    }
}
```

4. 修改application.yml

```
server:
  port: 8769

# 从git 仓库读取配置文件的配置项
# GIT目录已经上传config-client-dev.properties文件 该文件没有指定端口 只配置了foo变量
spring:
  cloud:
    nacos:
      discovery:
        server-addr: 192.168.23.139:8848
    config:
      server:
        git:
          #下面的git为码云地址
          uri: https://gitee.com/code80341157/itheima-repository.git
          searchPaths: itheima
          username:
          password:
          label: master
      application:
        name: config-server-git
#配置说明:
#1. spring.cloud.config.server.git.uri配置git仓库地址
#2. spring.cloud.config.server.git.searchPaths配置搜索远程仓库的文件夹地址
#3. spring.cloud.config.server.git.username配置git仓库的登录名
#4. spring.cloud.config.server.git.password配置git仓库的密码（公开的Git仓库不需要用户名、密码；私人Git仓库需要）
#5. spring.cloud.config.label为git仓库的分支名，本例从master读取。
```

5. 创建Config Client模块

创建config-client-git模块

6. 引入起步依赖


```

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-config</artifactId>
</dependency>
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
</dependency>

```

7. 修改bootstrap.yml

```

spring:
  application:
    name: config-client-git
  cloud:
    config:
      uri: http://localhost:8769
      fail-fast: true
    profiles:
      active: dev
  server:
    port: 8762
  eureka:
    client:
      service-url:
        defaultZone: http://localhost:8761/eureka/
    instance:
      prefer-ip-address: true
      instance-id: localhost:${server.port}

```

8. 读取

在ConfigClientApplication类中写一个API接口，读取配置文件itheima变量

```

@SpringBootApplication
@RestController
public class ConfigClientApplication {

    @Value("${itheima}")
    String itheima;

    public static void main(String[] args) {
        SpringApplication.run(ConfigClientApplication.class, args);
    }

    @RequestMapping(value = "/itheima")
    public String hi() {
        return itheima;
    }
}

```

9. 上传

将config-server-git下resource/shared目录中的config-client-git-dev.yml文件上传到码云（也可以是github）

浏览器输入

```
https://gitee.com/code80341157/itheima-repository/tree/master/itheima
```



点击上传文件



将config-client-git-dev.yml上传到码云就可以了

10. 启动

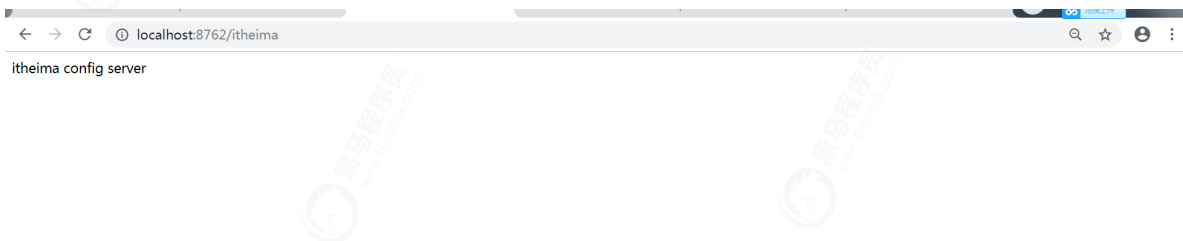
启动config-server-git、config-client-git工程，如下图：



11. 访问

`http://localhost:8762/itheima`

浏览器显示：



远程仓库（码云）config-client-git-dev.yml数据



评论 (0)

由此可见，在访问Config Client时，Config Client成功连接到了Config Server，而Config Server也成功的从码云远程仓库获取到了数据。

12. 存在刷新问题

此时，存在一个问题，如果我更改了远程仓库的数据

将之前的itheima: itheima config server 修改为itheima: itheima config server Version 1.0.0，如下图：



这个时候，我在不重启Config Servier的情况下，在重新获取，如下图：



我们发现，获取到的数据不是我们想要的。

这个问题如何解决？请继续往下看

13. 解决刷新问题

修改我们上面的工程config-client-git，config-server-git保持不变。

config-client-git引入刷新的起步依赖

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

新增@RefreshScope注解，只有加上了该注解，才会在不重启服务的情况下更新配置：

```
@RefreshScope
public class ConfigClientApplication {

    ....
}
```

config-client-git工程的bootstrap.yml文件暴露刷端点

```
spring:
  application:
    name: config-client-git
  cloud:
    nacos:
      discovery:
        server-addr: 192.168.23.139:8848
      config:
        uri: http://localhost:8769
        fail-fast: true
    profiles:
      active: dev
  server:
    port: 8762
  management:
```

```
endpoints:
  web:
    exposure:
      include: refresh
```

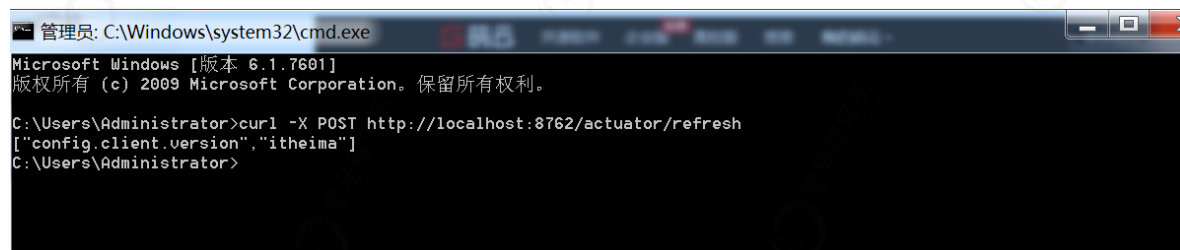
此时，重启config-client-git工程，其他先不要从重启，重启成功后，修改码云远程仓库 config-client-git-dev.yml的值: itheima: itheima config server Version 2.0.0，如下：



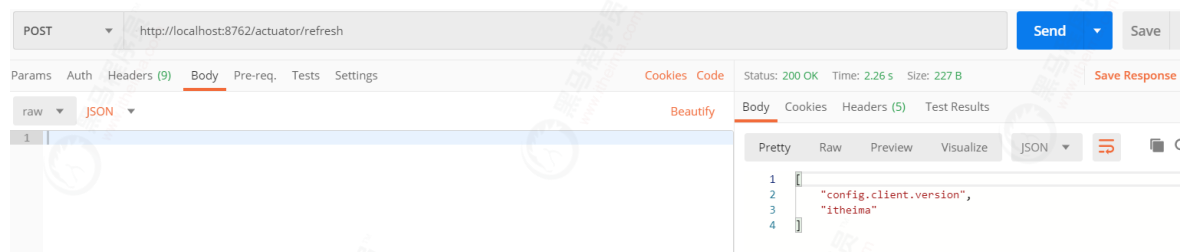
此时，是无法获取远程仓库最新的值，我们需要执行最重要的一步：

通过postman（或curl）工具发送post请求到

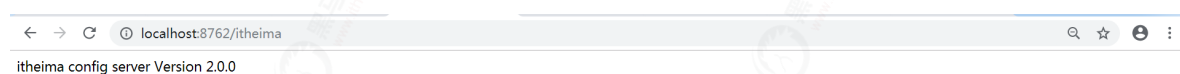
<http://localhost:8762/actuator/refresh>



或者



成功后，然后在去访问<http://localhost:8762/itheima>，如下图：



此时，我们获取到了远程仓库上最新的值。

3.2 Apollo分布式配置中

3.2.1 Apollo分布式配置中心介绍



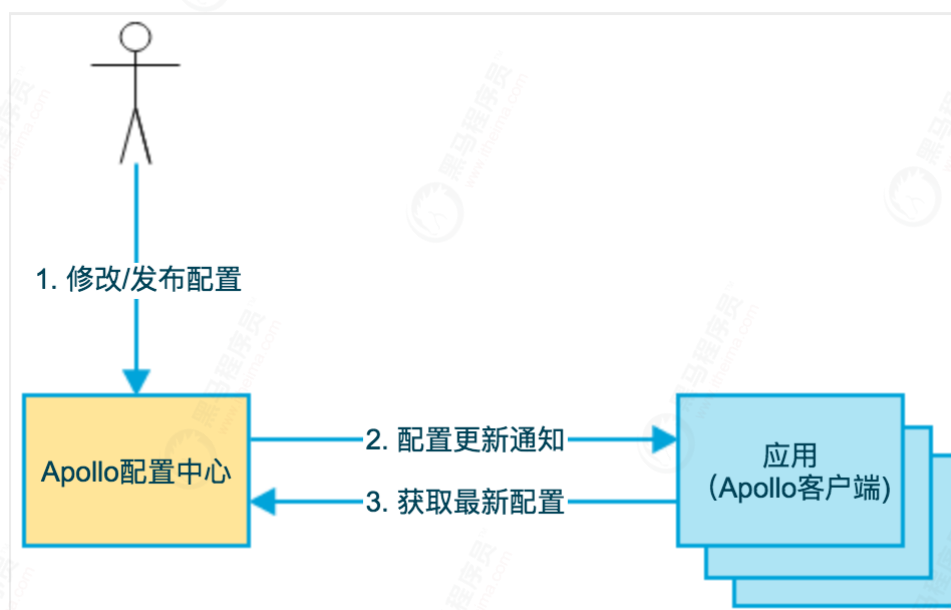
Apollo（阿波罗）是携程框架部门研发的分布式配置中心。

能够集中化管理应用不同环境、不同集群的配置，配置修改后能够实时推送到应用端

服务端基于Spring Boot和Spring Cloud（内置eureka）开发，打包后可以直接运行，不需要额外安装Tomcat等应用容器。

Java客户端不依赖任何框架，能够运行于所有Java运行时环境，同时对Spring/Spring Boot环境也有较好的支持。

3.2.2 Apollo工作流程介绍



1. 用户在配置中心对配置进行修改并发布
2. 配置中心通知Apollo客户端有配置更新
3. Apollo客户端从配置中心拉取最新的配置、更新本地配置并通知到应用

3.2.3 Apollo容器化部署

Apollo部署方式有三种

- 1、通过源码编译（麻烦，但是能学到中间的原理）
- 2、通过docker pull（方便，但是会忽略很多东西）
- 3、通过quickstart（快捷）

1) 基础环境搭建

docker-compose

```
#docker compose下载
curl -L "https://github.com/docker/compose/releases/download/1.25.0/docker-
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

#赋权
chmod +x /usr/local/bin/docker-compose
#版本查看
docker-compose --version
```

坑：maven需要3.2.5或以上

```
下载
wget https://zysd-shanghai.oss-cn-
shanghai.aliyuncs.com/software/linux/maven/apache-maven-3.6.1-bin.tar.gz
```

```
解压
tar -zxvf apache-maven-3.6.1-bin.tar.gz
```

1. 编辑环境变量

```
vim /etc/profile
```

2. 添加Maven的M2_HOME地址

```
export M2_HOME=/opt/apache-maven-3.6.1
```

```
export PATH=$PATH:$M2_HOME/bin
```


Apollo 最主要有三个工程，会用这三个工程进行部署

apollo-configservice:

提供获取配置的接口供客户端使用以及当有新配置更新时会通知客户端

客户端可以以一定周期定时从ConfigService获取最新配置，configservice和客户端维护一个长连接，当有配置更新的时候，会通过长连接通知客户端去取最新配置

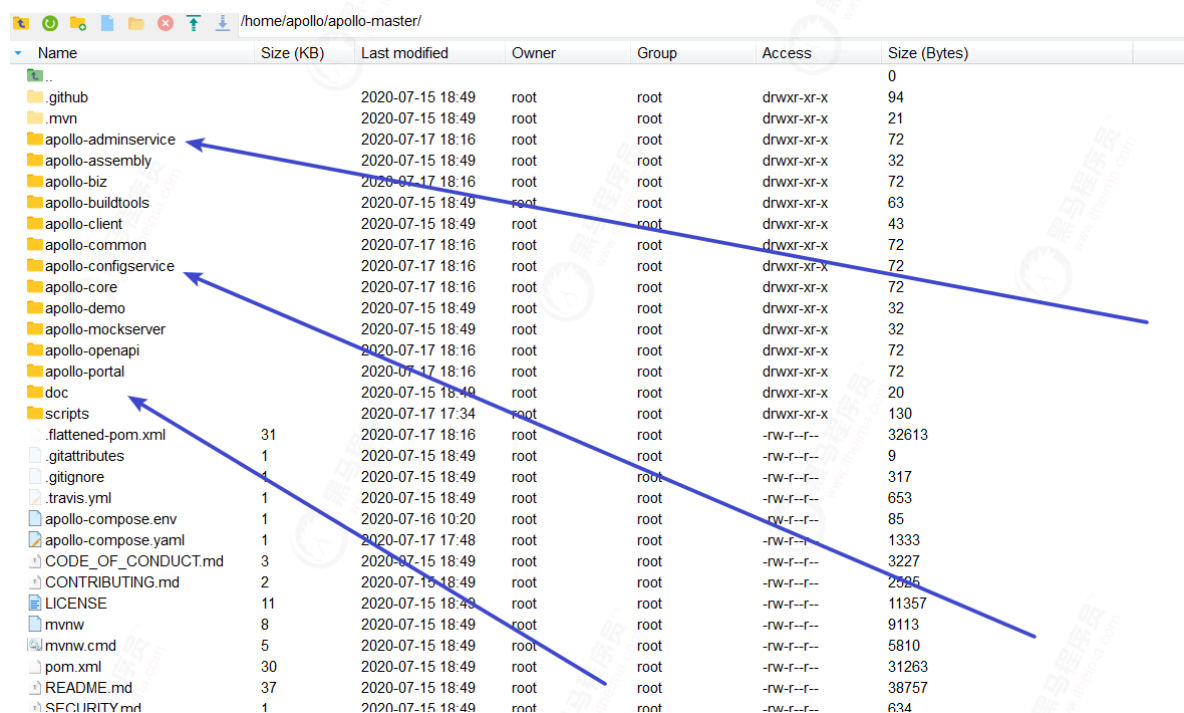
apollo-adminservice:

提供配置管理的接口供portal管理页面使用

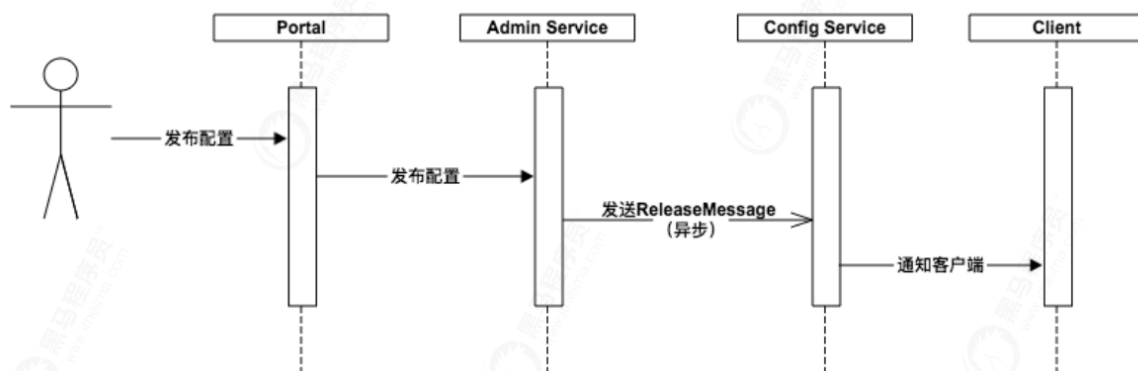
例如配置的修改和发布 AdminService和ConfigService共用一个数据库，adminService每次发布配置的时候都会表里面插入一条记录，而configService会扫描这张表看是否有新插入数据，如果有变化就会通知客户端

apollo-portal:

提供Web界面供用户管理配置，提供登陆和审计日志的功能，portal需要通过rpc调用AdminService提供的配置管理的接口，portal有自己的数据库



Name	Size (KB)	Last modified	Owner	Group	Access	Size (Bytes)
..						0
.github		2020-07-15 18:49	root	root	drwxr-xr-x	94
.mvn		2020-07-15 18:49	root	root	drwxr-xr-x	21
apollo-adminservice		2020-07-17 18:16	root	root	drwxr-xr-x	72
apollo-assembly		2020-07-15 18:49	root	root	drwxr-xr-x	32
apollo-biz		2020-07-17 18:16	root	root	drwxr-xr-x	72
apollo-buildtools		2020-07-15 18:49	root	root	drwxr-xr-x	63
apollo-client		2020-07-15 18:49	root	root	drwxr-xr-x	43
apollo-common		2020-07-17 18:16	root	root	drwxr-xr-x	72
apollo-configservice		2020-07-17 18:16	root	root	drwxr-xr-x	72
apollo-core		2020-07-17 18:16	root	root	drwxr-xr-x	72
apollo-demo		2020-07-15 18:49	root	root	drwxr-xr-x	32
apollo-mockserver		2020-07-15 18:49	root	root	drwxr-xr-x	32
apollo-openapi		2020-07-17 18:16	root	root	drwxr-xr-x	72
apollo-portal		2020-07-17 18:16	root	root	drwxr-xr-x	72
doc		2020-07-15 18:49	root	root	drwxr-xr-x	20
scripts		2020-07-17 17:34	root	root	drwxr-xr-x	130
flattened-pom.xml	31	2020-07-17 18:16	root	root	-rw-r--r--	32613
gitattributes	1	2020-07-15 18:49	root	root	-rw-r--r--	9
gitignore	4	2020-07-15 18:49	root	root	-rw-r--r--	317
.travis.yml	1	2020-07-15 18:49	root	root	-rw-r--r--	653
apollo-compose.env	1	2020-07-16 10:20	root	root	-rw-r--r--	85
apollo-compose.yaml	1	2020-07-17 17:48	root	root	-rw-r--r--	1333
CODE_OF_CONDUCT.md	3	2020-07-15 18:49	root	root	-rw-r--r--	3227
CONTRIBUTING.md	2	2020-07-15 18:49	root	root	-rw-r--r--	2025
LICENSE	11	2020-07-15 18:49	root	root	-rw-r--r--	11357
mvnw	8	2020-07-15 18:49	root	root	-rw-r--r--	9113
mvnw.cmd	5	2020-07-15 18:49	root	root	-rw-r--r--	5810
pom.xml	30	2020-07-15 18:49	root	root	-rw-r--r--	31263
README.md	37	2020-07-15 18:49	root	root	-rw-r--r--	38757
SECURITY.md	1	2020-07-15 18:49	root	root	-rw-r--r--	634



用户在Portal操作配置发布

Portal调用Admin Service的接口操作发布

Admin Service发配置后，发送消息给各个Config Service

Config Service收到发布消息后，通知对应的客户端

2、源码编译-下载

```
git克隆
git clone https://github.com/ctripcorp/apollo.git
```

or

```
直接下载
https://github.com/ctripcorp/apollo
```

```
https://github.com/ctripcorp/apollo/releases
```

新建目录，下载代码

```
/home/apollo/
```

3、修改build.sh文件的配置

```
/home/apollo/apollo-master/scripts/
```

修改build.sh（根据操作系统环境选择）

```
#!/bin/sh

# apollo config db info
apollo_config_db_url=jdbc:mysql://192.168.23.139:3306/ApolloConfigDB?
characterEncoding=utf8
apollo_config_db_username=root
apollo_config_db_password=root

# apollo portal db info
apollo_portal_db_url=jdbc:mysql://192.168.23.139:3306/ApolloPortalDB?
characterEncoding=utf8
apollo_portal_db_username=root
apollo_portal_db_password=root

# meta server url, different environments should have different meta server
addresses
dev_meta=http://192.168.23.139:8080
```

8080为 apollo-configservice的端口

修改/apollo-master/apollo-adminservice/src/main/resources/bootstrap.yml

增加ip-address: 192.168.23.139

```
eureka:
  instance:
    hostname: ${hostname:localhost}
    preferIpAddress: true
    status-page-url-path: /info
    health-check-url-path: /health
    ip-address: 192.168.23.139
  client:
    serviceUrl:
      # This setting will be overridden by eureka.service.url setting from
      # ApolloConfigDB.ServerConfig or System Property
      # see com.ctrip.framework.apollo.biz.eureka.ApolloEurekaClientConfig
      defaultZone: http://${eureka.instance.hostname}:8080/eureka/
    healthcheck:
      enabled: true
      eurekaServiceUrlPollIntervalSeconds: 60

  management:
    health:
      status:
        order: DOWN, OUT_OF_SERVICE, UNKNOWN, UP
```

apollo/apollo-master/apollo-configservice/src/main/resources//bootstrap.yml

增加ip-address: 192.168.23.139

```
eureka:
  instance:
    hostname: ${hostname:localhost}
    preferIpAddress: true
    status-page-url-path: /info
    health-check-url-path: /health
    ip-address: 192.168.23.139
  server:
    peerEurekaNodesUpdateIntervalMs: 60000
    enableSelfPreservation: false
  client:
    serviceUrl:
      # This setting will be overridden by eureka.service.url setting from
      # ApolloConfigDB.ServerConfig or System Property
      # see com.ctrip.framework.apollo.biz.eureka.ApolloEurekaClientConfig
      defaultZone: http://${eureka.instance.hostname}:8080/eureka/
    healthcheck:
      enabled: true
      eurekaServiceUrlPollIntervalSeconds: 60

  management:
    health:
      status:
        order: DOWN, OUT_OF_SERVICE, UNKNOWN, UP
```

修改apollo/apollo-master/apollo-portal/src/main/resources/apollo-env.proper

```
local.meta=http://192.168.23.139:8080
dev.meta=${dev_meta}
fat.meta=${fat_meta}
uat.meta=${uat_meta}
lpt.meta=${lpt_meta}
pro.meta=${pro_meta}
```

执行打包

```
./build.sh
```

```
[INFO] Building zip: /home/apollo/apollo-master/apollo-portal/target/apollo-portal-1.7.0-SNAPSHOT-github.zip
[INFO]
[INFO] Reactor Summary for Apollo 1.7.0-SNAPSHOT:
[INFO]
[INFO] Apollo ..... SUCCESS [ 3.237 s]
[INFO] Apollo Core ..... SUCCESS [ 6.331 s]
[INFO] Apollo Common ..... SUCCESS [ 2.751 s]
[INFO] Apollo Open Api ..... SUCCESS [ 1.190 s]
[INFO] Apollo Portal ..... SUCCESS [ 12.591 s]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 27.664 s
[INFO] Finished at: 2020-07-16T15:16:37+08:00
[INFO]
==== building portal finished ====
[root@localhost scripts]#
```

重要:

打包完毕后

随后把三个文件夹各自对应的路径/apollo-adminservice/target、/apollo-configservice/target、/apollo-portal/target中的zip包拷贝到对应目录分别为/apollo-adminservice/src/main/docker、/apollo-configservice/src/main/docker、/apollo-portal/src/main/docker中

开始编排

```
docker-compose -f /home/apollo/apollo-master/apollo-compose.yaml up -d
docker-compose -f /home/apollo/apollo-master/apollo-compose.yaml start
docker-compose -f /home/apollo/apollo-master/apollo-compose.yaml restart
docker-compose -f /home/apollo/apollo-master/apollo-compose.yaml stop
```

docker-compose文件

```
version: "3"
services:
  apollo-configservice:
    container_name: apollo-configservice
    build: apollo-configservice/src/main/docker/
    image: apollo-configservice:1.7.0

    ports:
      - "8080:8080"
    #volumes:
      #- /home/apollo/logs/config:/opt/logs/100003171
```

```

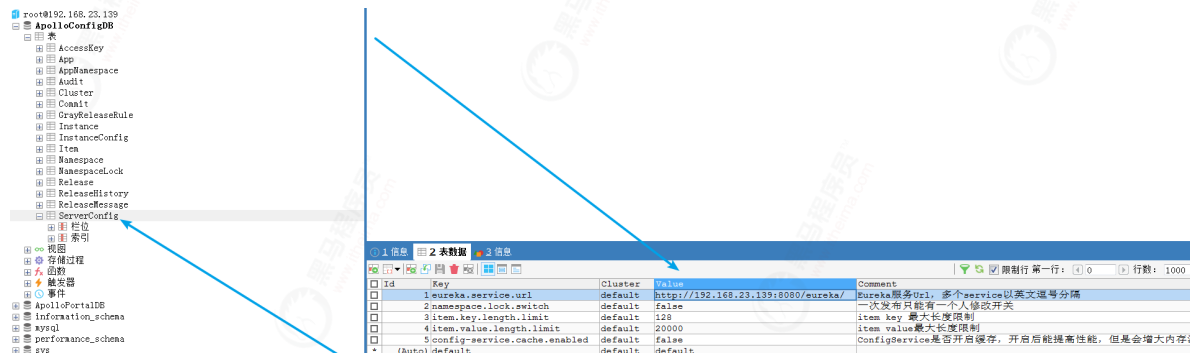
depends_on:
  - mysql-dev
apollo-adminservice:
  container_name: apollo-adminservice
  build: apollo-adminservice/src/main/docker/
  image: apollo-adminservice:1.7.0
  ports:
    - "8090:8090"
depends_on:
  - apollo-configservice
  - mysql-dev
#volumes:
  #- /home/apollo/logs/admin:/opt/logs/100003172
apollo-portal:
  container_name: apollo-portal
  build: apollo-portal/src/main/docker/
  image: apollo-portal:1.7.0
  ports:
    - "8070:8070"
depends_on:
  - apollo-adminservice
  - mysql-dev
#volumes:
  #- /home/apollo/logs/portal:/opt/logs/100003173
mysql-dev:
  container_name: mysql-dev
  image: mysql:5.7
  ports:
    - "3306:3306"
#env_file:
  #- /home/apollo/apollo-master/apollo-compose.env

```

执行sql

\apollo-master\scripts\sql

修改eureka注册地址



The screenshot shows the Apollo ConfigDB interface. On the left, the 'ServerConfig' section is selected. On the right, a table displays the configuration for the 'eureka.service.url'. The table has columns for Id, Key, Cluster, Value, and Comment. The 'eureka.service.url' row is highlighted, showing its value as 'http://192.168.23.139:8080/eureka/'.

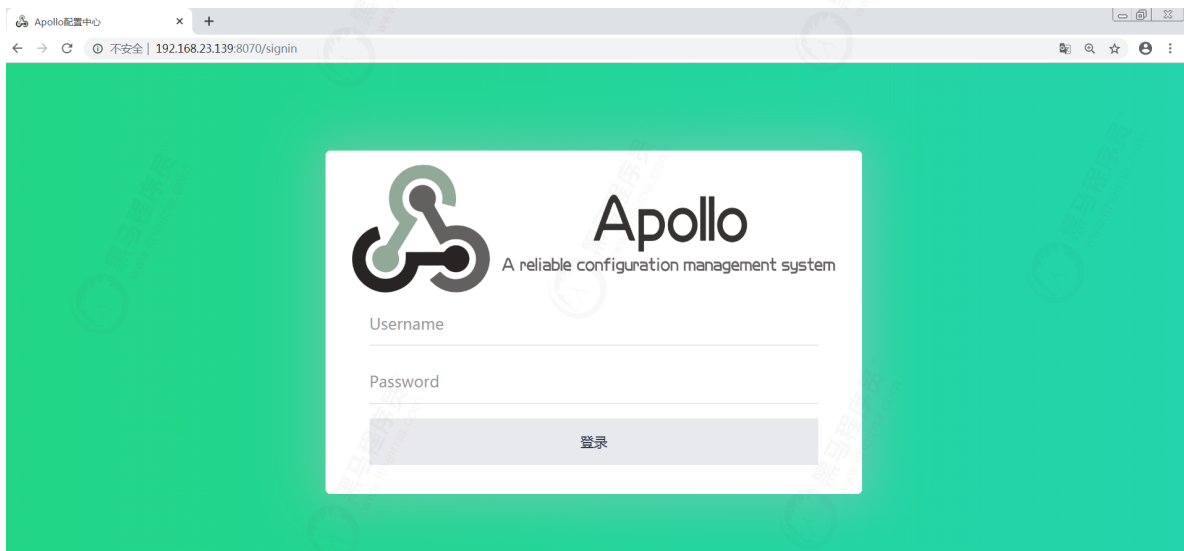
Id	Key	Cluster	Value	Comment
1	eureka.service.url	default	http://192.168.23.139:8080/eureka/	Eureka服务Url, 多个service以英文逗号分隔
2	namespace.lock.switch	default	false	一次发布只能有一个人修改开关
3	item.key.length.limit	default	128	item key 最大长度限制
4	item.value.length.limit	default	20000	item values最大长度限制
5	config-service.cache.enabled	default	false	ConfigService是否开启缓存, 开启后能提高性能, 但是会增大内存
*	(Auto) default	default	default	

访问

默认用户名 apollo

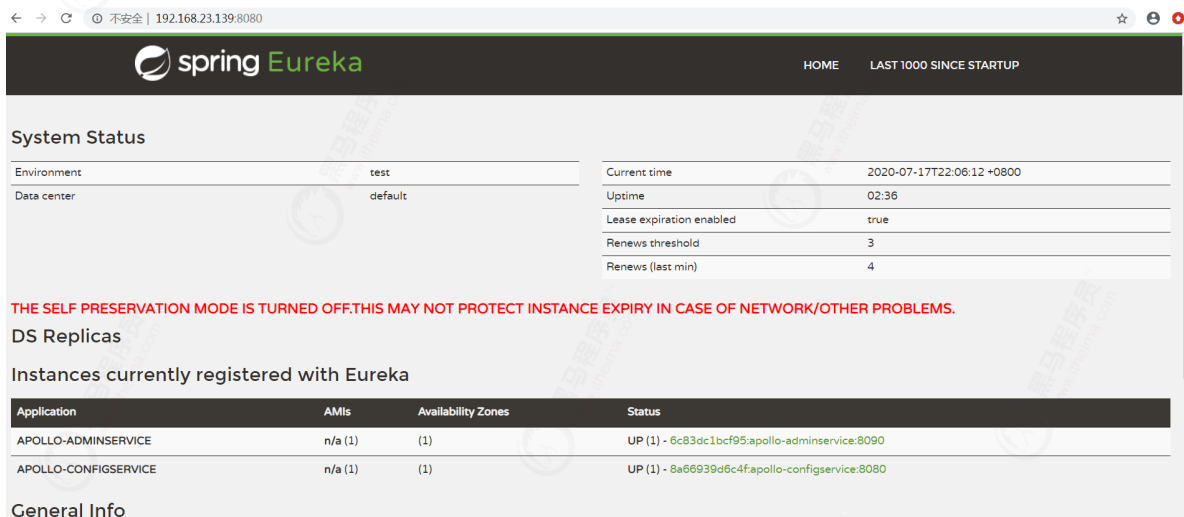
密码为 admin

http://192.168.23.139:8070/



http://192.168.23.139:8080

apollo-configservice的端口为8080



1、常见错误

```
[root@localhost apollo-master]# docker-compose -f apollo-compose.yaml up -d
Building apollo-configservice
Step 1/10 : FROM openjdk:8-jre-alpine
--> f7a292bbb70c
Step 2/10 : MAINTAINER ameizi <sxyx2008@163.com>
--> Using cache
--> dbba27e10913
Step 3/10 : ENV APOLLO_RUN_MODE "Docker"
--> Using cache
--> 6f1704c46fab
Step 4/10 : ENV VERSION 1.7.0-SNAPSHOT
--> Using cache
--> c1562780ad08
Step 5/10 : ENV SERVER_PORT 8080
--> Using cache
--> f56b1b02933e
Step 6/10 : RUN echo "http://mirrors.aliyun.com/alpine/v3.8/main" > /etc/apk/repositories && echo "http://mirrors.aliyun.com/alpine/v3.8/community" > /etc/apk/repositories && apk update upgrade && apk add --no-cache procs unzip curl bash tzdata && ln -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime && echo "Asia/Shanghai" > /etc/timezone
--> [Warning] IPv4 forwarding is disabled. Networking will not work.
--> Running in 2fb471c69d0b

fetch http://mirrors.aliyun.com/alpine/v3.8/main/x86_64/APKINDEX.tar.gz
fetch http://mirrors.aliyun.com/alpine/v3.8/community/x86_64/APKINDEX.tar.gz
ERROR: http://mirrors.aliyun.com/alpine/v3.8/main: temporary error (try again later)
WARNING: Ignoring APKINDEX.e3d33561.tar.gz: No such file or directory
2 errors; 53 distinct packages available
ERROR: http://mirrors.aliyun.com/alpine/v3.8/community: temporary error (try again later)
WARNING: Ignoring APKINDEX.39d9158a.tar.gz: No such file or directory
ERROR: Service 'apollo-configservice' failed to build: The command '/bin/sh -c echo "http://mirrors.aliyun.com/alpine/v3.8/main" > /etc/apk/repositories && echo "http://mirrors.aliyun.com/alpine/v3.8/community" > /etc/apk/repositories && apk update upgrade && apk add --no-cache procs unzip curl bash tzdata && ln -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime && echo "Asia/Shanghai" > /etc/timezone' returned a non-zero code: 2
```

解决方案

```
#vim /etc/default/docker

#DOCKER_OPTS="--dns 114.114.114.114"

#systemctl restart docker
```

3.2.4 SpringCloud集成Apollo

```

v itheima-service-apollo F:\itheima-microservice-parent\
> .idea
> .mvn
> src
> target
  .gitignore
  HELP.md
  itheima-service-apollo.iml
  mvnw
  mvnw.cmd
  pom.xml
> External Libraries

```

1、增加配置文件


```
server.port=8888
spring.application.name=itheima-apollo
### apollo server地址
apollo.meta=http://192.168.23.139:8080
### apollo server的应用id
app.id=itheima-apollo
```

2、增加起步依赖

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.2.5.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.itheima.apollo</groupId>
  <artifactId>itheima-apollo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>itheima-apollo</name>
  <description>Demo project for Spring Boot</description>

  <properties>
    <java.version>1.8</java.version>
    <spring-cloud.version>Hoxton.SR3</spring-cloud.version>

    <spring.cloud.alibaba.version>2.2.1.RELEASE</spring.cloud.alibaba.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
      <exclusions>
        <exclusion>
          <groupId>org.junit.vintage</groupId>
          <artifactId>junit-vintage-engine</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
    <!--
https://mvnrepository.com/artifact/com.ctrip.framework.apollo/apollo-client -->
    <dependency>
      <groupId>com.ctrip.framework.apollo</groupId>
      <artifactId>apollo-client</artifactId>
      <version>1.7.0</version>
```

```

</dependency>
    <!--nacos-->
    <dependency>
        <groupId>com.alibaba.cloud</groupId>
        <artifactId>spring-cloud-starter-alibaba-nacos-
discovery</artifactId>
    </dependency>
    </dependencies>
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring-cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>

        <dependency>
            <groupId>com.alibaba.cloud</groupId>
            <artifactId>spring-cloud-alibaba-dependencies</artifactId>
            <version>${spring.cloud.alibaba.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>

    </dependencies>
</dependencyManagement>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>

</project>

```

3、开启Apollo客户端配置

启动类增加@EnableApolloConfig

4、新增控制器测试

5、开始验证

1) 创建项目

2) 新增配置

添加配置项 (温馨提示: 可以通过文本模式批量添加配置)

* Key

czbk

Value

黑马程序员, 全国第一

Comment

* 选择集群

☐ 环境

集群

☒ DEV

default

取消

提交

3) 发布配置

环境列表

DEV

项目信息

AppId: itheima-apollo

应用名称: itheima-apollo

部门: 样例部门1(TES T1)

负责人: apollo

邮箱: apollo@acme.com

application 有修改

发布

回滚

发布历史

授权

灰度

设置

过滤配置

同步配置

撤销配置

比较配置

新增配置

发布状态	Key	Value	备注	最后修改人	最后修改时间	操作
未发布	czbk	黑马程序员, 全国第一		apollo	2020-07-20 17:53:48	编辑 删除

4) 访问验证

提取启动Nacos

http://localhost:8888/conf

localhost:8888/conf
hi,黑马程序员, 全国第一

再次修改配置 (本次不发布)

环境列表

DEV

项目信息

AppId: itheima-apollo

应用名称: itheima-apollo

部门: 样例部门1(TES11)

负责人: apollo

邮箱: apollo@acme.com

properties

application

发布

回滚

发布历史

授权

灰度

更多

表格

文本

更改历史

实例列表

过滤配置

同步配置

撤销配置

比较配置

新增配置

发布状态	Key	Value	备注	最后修改人	最后修改时间	操作
已发布	czbk	黑马程序员, 全国第一		apollo	2020-07-20 17:53:48	修改

修改配置项

Key

czbk

Value

黑马程序员, 全国第一, 欢迎大家

Comment

取消

提交

验证

发现数据没有变化

localhost:8888/conf

hi,黑马程序员, 全国第一

发布配置

发布 (只有发布过的配置才会被客户端获取到, 此次发布只会作用于当前环境:DEV)

Changes	Key	发布的值	未发布的值	修改人	修改时间
	czbk 改	黑马程序员, 全国第一	黑马程序员, 全国第一, 欢迎大家	apollo	2020-07-20 17:56:17

* Release Name

Comment

取消 发布

再次验证

localhost:8888/conf
hi,黑马程序员, 全国第一, 欢迎大家

3.3 Nacos分布式配置中心

▼ **itheima-service-nacos** F:\itheima-microservice-parent\i
 > .idea
 > **nacos-spring-cloud-config-example**
 > **nacos-spring-cloud-discovery-example**
 itheima-service-nacos.iml
 pom.xml
 > External Libraries
 Scratches and Consoles

3.3.1 SpringCloud集成配置中心

1) 创建nacos-spring-cloud-config-example模块

与Spring Cloud集成, 创建自动刷新模块nacos-spring-cloud-config-example

2) 引入起步依赖

```
<dependencies>  
  <dependency>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-web</artifactId>  
  </dependency>
```

```

<dependency>
  <groupId>com.alibaba.cloud</groupId>
  <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
</dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
  </dependency>
</dependencies>

```

3) 配置bootstrap.properties

```

server.port=9999
#配置中心
spring.cloud.nacos.config.server-addr=192.168.23.139:8848
#注册中心
spring.cloud.nacos.discovery.server-addr=192.168.23.139:8848
spring.application.name=nacos-conf
#配置中心文件扩展名称
spring.cloud.nacos.config.file-extension=properties

```

配置 Nacos server 的地址和应用名

说明：之所以需要配置 `spring.application.name`，是因为它是构成 Nacos 配置管理 `dataId` 字段的一部分。

在 Nacos S 中，`dataId` 的完整格式如下

```

${prefix}-${spring.profile.active}.${file-extension}

```

- `prefix` 默认为 `spring.application.name` 的值，也可以通过配置项 `spring.cloud.nacos.config.prefix` 来配置。
- `spring.profile.active` 即为当前环境对应的 profile，注意：当 `spring.profile.active` 为空时，对应的连接符 - 也将不存在，`dataId` 的拼接格式变成 `${prefix}.${file-extension}`
- `file-extension` 为配置内容的数据格式，可以通过配置项 `spring.cloud.nacos.config.file-extension` 来配置。目前只支持 `properties` 和 `yaml` 类型

4) 实现自动更新

```

@RestController
@RequestMapping("/config")
@RefreshScope
public class ConfigController {

    @Value("${useLocalCache:false}")
    private boolean useLocalCache;

    @RequestMapping("/get")
    public boolean get() {
        return useLocalCache;
    }
}

```


通过 Spring Cloud 原生注解 @RefreshScope 实现配置自动更新

3.3.2 配置中心发布与配置

1) 发布配置

```
curl -X POST "http://127.0.0.1:8848/nacos/v1/cs/configs?dataId=example.properties&group=DEFAULT_GROUP&content=useLocalCache=true"
```



2) 访问

运行 NacosConfigApplication，调用 `curl http://localhost:8080/config/get`，返回内容是 `true`

```
C:\Users\Administrator>curl http://localhost:8080/config/get
true
C:\Users\Administrator>
```

3) 再次发布配置

```
curl -X POST "http://127.0.0.1:8848/nacos/v1/cs/configs?dataId=example.properties&group=DEFAULT_GROUP&content=useLocalCache=false"
```

4) 再次访问

再次访问 `http://localhost:8080/config/get`，此时返回内容为 `false`，说明程序中的 `useLocalCache` 值已经被动态更新了



比较（Nacos与Apollo）：

- (1) Nacos部署简化，Nacos整合了注册中心、配置中心功能，且部署相比apollo简单，方便管理和监控。
- (2) apollo容器化较困难（1.7之前），Nacos有官网的镜像可以直接部署
- (3) 性能方面，Nacos读写tps比apollo强一些